

Category Theory for ZFC in HOL III
Universal Constructions for 1-Categories

Mihails Milehins

February 6, 2026

Abstract

This article provides a formalization of elements of the theory of universal constructions for 1-categories (such as limits, adjoints and Kan extensions) in the object logic *ZFC in HOL* ([13], also see [11]) of the formal proof assistant *Isabelle* [12].

Acknowledgements

The author would like to acknowledge the assistance that he received from the users of the mailing list of Isabelle in the form of answers given to his general queries. Special thanks go to Andreas Lochbihler for suggesting the use of the combination of unrestricted overloading and locales for structuring mathematical knowledge on the mailing list of Isabelle: the design pattern that is used in this study builds upon this idea. Special thanks also go to Thomas Sewell for suggesting a trick for rewriting expressions modulo associativity on the mailing list of Isabelle, which was used on numerous occasions throughout the development of this work. Furthermore, the author would like to mention that the tool “Sketch-and-Explore” [5] from the standard distribution of Isabelle was used extensively in the development of this work. Moreover, the author would like to acknowledge the positive role that numerous Q&A posted on the Stack Exchange network (especially Mathematics Stack Exchange, Stack Overflow and TeX Stack Exchange) played in the development of this work. Lastly, the author would like to express gratitude to all members of his family and friends for their continuous support.

Contents

1	Introduction	7
2	Universal arrow	8
2.1	Background	8
2.2	Universal map	8
2.3	Universal arrow: definition and elementary properties	12
2.4	Uniqueness	13
2.5	Universal natural transformation	18
3	Limits and colimits	29
3.1	Background	29
3.2	Limit and colimit	29
3.3	Small limit and small colimit	43
3.4	Finite limit and finite colimit	53
3.5	Creation of limits	55
3.6	Preservation of limits and colimits	56
3.7	Continuous and cocontinuous functor	60
3.8	Tiny-continuous and tiny-cocontinuous functor	65
4	Initial and terminal objects as limits and colimits	66
4.1	Initial and terminal objects as limits/colimits of an empty diagram	66
4.2	Initial cone and terminal cocone	69
4.3	Initial and terminal objects as limits/colimits of the identity functor	72
5	Products and coproducts as limits and colimits	78
5.1	Product and coproduct	78
5.2	Small product and small coproduct	81
5.3	Finite product and finite coproduct	83
5.4	Product and coproduct of two objects	85
5.5	Projection cone	89
6	Pullbacks and pushouts as limits and colimits	94
6.1	Pullback and pushout	94
7	Equalizers and coequalizers as limits and colimits	104
7.1	Equalizer and coequalizer	104
7.2	Equalizer and coequalizer for two arrows	116
7.3	Equalizer cone	127
8	Pointed arrows and natural transformations	130
8.1	Pointed arrow	130
8.2	Pointed natural transformation	132
8.3	Inverse pointed natural transformation	136
9	Representable and corepresentable functors	143
9.1	Representable and corepresentable functors	143
9.2	Limits and colimits as universal cones	149

10	Completeness and cocompleteness	155
10.1	Limits by products and equalizers	155
10.2	Small-complete and small-cocomplete category	167
10.3	Finite-complete and finite-cocomplete category	174
11	Comma categories and universal constructions	176
11.1	Relationship between the universal arrows, initial objects and terminal objects . .	176
11.2	A projection for a comma category constructed from a functor and an object creates small limits	180
12	Category <i>Set</i> and universal constructions	191
12.1	Discrete functor with tiny maps to the category <i>Set</i>	191
12.2	Product cone and coproduct cocone for the category <i>Set</i>	192
12.3	Equalizer for the category <i>Set</i>	199
12.4	The category <i>Set</i> is small-complete	206
13	Adjoints	207
13.1	Background	207
13.2	Definition and elementary properties	207
13.3	Opposite adjunction	208
13.4	Unit	213
13.5	Counit	218
13.6	Counit-unit equations	225
13.7	Construction of an adjunction from universal morphisms from objects to functors	228
13.8	Construction of an adjunction from a functor and universal morphisms from objects to functors	234
13.9	Construction of an adjunction from universal morphisms from functors to objects	241
13.10	Construction of an adjunction from a functor and universal morphisms from functors to objects	244
13.11	Construction of an adjunction from the counit-unit equations	249
13.12	Adjoints are unique up to isomorphism	254
13.13	Further properties of the adjoint functors	266
13.14	Adjoints on limits	269
14	Simple Kan extensions	274
14.1	Background	274
14.2	Kan extension	274
14.3	Opposite universal arrow for Kan extensions	283
14.4	The Kan extension	285
14.5	Preservation of Kan extensions	319
14.6	All concepts are Kan extensions	321
15	Pointwise Kan extensions	331
15.1	Pointwise Kan extensions	331
15.2	Lemma X.5: <i>L-10-5-N</i>	333
15.3	Lemma X.5: <i>L-10-5-v-arrow</i>	337
15.4	Lemma X.5: <i>L-10-5-τ</i>	342
15.5	Lemma X.5: <i>L-10-5-v</i>	346
15.6	Lemma X.5: <i>L-10-5-χ-arrow</i>	348
15.7	Lemma X.5: <i>L-10-5-χ'-arrow</i>	351
15.8	Lemma X.5: <i>L-10-5-χ</i>	358

15.9	The existence of a canonical limit or a canonical colimit for the pointwise Kan extensions	366
15.10	The limit and the colimit for the pointwise Kan extensions	386
16	Pointwise Kan extensions: application example	390
16.1	Background	390
16.2	$\mathfrak{K}23$	390
16.3	$LK23$: the functor associated with the left Kan extension along $\mathfrak{K}23$	394
16.4	$RK23$: the functor associated with the right Kan extension along $\mathfrak{K}23$	399
16.5	$RK\text{-}\sigma 23$: towards the universal property of the right Kan extension along $\mathfrak{K}23$	405
16.6	The right Kan extension along $\mathfrak{K}23$	408
16.7	$LK\text{-}\sigma 23$: towards the universal property of the left Kan extension along $\mathfrak{K}23$	410
16.8	The left Kan extension along $\mathfrak{K}23$	413
16.9	Pointwise Kan extensions along $\mathfrak{K}23$	415
References		431

1 Introduction

This article provides a formalization of further elements of the theory of 1-categories without an additional structure. More specifically, this article explores canonical universal constructions [1]¹ and their properties, building upon the formalization of the foundations of category theory in [10].

¹<https://ncatlab.org/nlab/show/universal+construction>

2 Universal arrow

2.1 Background

The following section is based, primarily, on the elements of the content of Chapter III-1 in [9].

named-theorems *ua-field-simps*

definition $UObj :: V$ **where** [*ua-field-simps*]: $UObj = 0$

definition $UArr :: V$ **where** [*ua-field-simps*]: $UArr = 1_{\mathbb{N}}$

lemma [*cat-cs-simps*]:

shows $UObj\text{-simp}$: $[a, b]_{\circ}(\downarrow UObj) = a$

and $UArr\text{-simp}$: $[a, b]_{\circ}(\downarrow UArr) = b$

unfolding *ua-field-simps* **by** (*simp-all add: nat-omega-simps*)

2.2 Universal map

The universal map is a convenience utility that allows treating a part of the definition of the universal arrow as an arrow in the category *Set*.

2.2.1 Definition and elementary properties

definition $umap\text{-of} :: V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$

where $umap\text{-of} \mathfrak{F} \ c \ r \ u \ d =$

$$\left[\begin{array}{l} (\lambda f' \in_{\circ} Hom \ (\mathfrak{F}(\downarrow HomDom)) \ r \ d. \ \mathfrak{F}(\downarrow ArrMap)(\downarrow f') \circ_A \mathfrak{F}(\downarrow HomCod) \ u), \\ Hom \ (\mathfrak{F}(\downarrow HomDom)) \ r \ d, \\ Hom \ (\mathfrak{F}(\downarrow HomCod)) \ c \ (\mathfrak{F}(\downarrow ObjMap)(\downarrow d)) \end{array} \right]_{\circ}$$

definition $umap\text{-fo} :: V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$

where $umap\text{-fo} \mathfrak{F} \ c \ r \ u \ d = umap\text{-of} \ (op\text{-cf} \ \mathfrak{F}) \ c \ r \ u \ d$

Components.

lemma (in *is-functor*) $umap\text{-of-components}$:

assumes $u : c \mapsto_{\mathfrak{B}} \mathfrak{F}(\downarrow ObjMap)(\downarrow r)$

shows $umap\text{-of} \ \mathfrak{F} \ c \ r \ u \ d(\downarrow ArrVal) = (\lambda f' \in_{\circ} Hom \ \mathfrak{A} \ r \ d. \ \mathfrak{F}(\downarrow ArrMap)(\downarrow f') \circ_{A\mathfrak{B}} \ u)$

and $umap\text{-of} \ \mathfrak{F} \ c \ r \ u \ d(\downarrow ArrDom) = Hom \ \mathfrak{A} \ r \ d$

and $umap\text{-of} \ \mathfrak{F} \ c \ r \ u \ d(\downarrow ArrCod) = Hom \ \mathfrak{B} \ c \ (\mathfrak{F}(\downarrow ObjMap)(\downarrow d))$

unfolding $umap\text{-of-def} \ arr\text{-field-simps}$

by (*simp-all add: cat-cs-simps nat-omega-simps*)

lemma (in *is-functor*) $umap\text{-fo-components}$:

assumes $u : \mathfrak{F}(\downarrow ObjMap)(\downarrow r) \mapsto_{\mathfrak{B}} c$

shows $umap\text{-fo} \ \mathfrak{F} \ c \ r \ u \ d(\downarrow ArrVal) = (\lambda f' \in_{\circ} Hom \ \mathfrak{A} \ d \ r. \ u \circ_{A\mathfrak{B}} \ \mathfrak{F}(\downarrow ArrMap)(\downarrow f'))$

and $umap\text{-fo} \ \mathfrak{F} \ c \ r \ u \ d(\downarrow ArrDom) = Hom \ \mathfrak{A} \ d \ r$

and $umap\text{-fo} \ \mathfrak{F} \ c \ r \ u \ d(\downarrow ArrCod) = Hom \ \mathfrak{B} \ (\mathfrak{F}(\downarrow ObjMap)(\downarrow d)) \ c$

unfolding

$umap\text{-fo-def}$

$is\text{-functor.}umap\text{-of-components}$

$OF \ is\text{-functor-op, unfolded cat-op-simps, OF assms}$

$\]$

proof(*rule vsv-eqI*)

fix f' **assume** $f' \in_{\circ} \mathcal{D}_{\circ} \ (\lambda f' \in_{\circ} Hom \ \mathfrak{A} \ d \ r. \ \mathfrak{F}(\downarrow ArrMap)(\downarrow f') \circ_{A \ op\text{-cat} \ \mathfrak{B}} \ u)$

then have $f' : f' : d \mapsto_{\mathfrak{A}} r$ **by** *simp*

then have $\mathfrak{F}f' : \mathfrak{F}(\downarrow ArrMap)(\downarrow f') : \mathfrak{F}(\downarrow ObjMap)(\downarrow d) \mapsto_{\mathfrak{B}} \mathfrak{F}(\downarrow ObjMap)(\downarrow r)$

by (*auto intro: cat-cs-intros*)

from f' **show**
 $(\lambda f' \in_{\circ} \text{Hom } \mathfrak{A} \ d \ r. \ \mathfrak{F}(\text{ArrMap})(f') \circ_{A \text{ op-cat } \mathfrak{B}} u)(f') =$
 $(\lambda f' \in_{\circ} \text{Hom } \mathfrak{A} \ d \ r. \ u \circ_{A \mathfrak{B}} \mathfrak{F}(\text{ArrMap})(f'))(f')$
by (*simp add: HomCod.op-cat-Comp[OF assms $\mathfrak{F}f'$]*)
qed *simp-all*

Universal maps for the opposite functor.

lemma (**in** *is-functor*) *op-umap-of[cat-op-simps]*: *umap-of (op-cf \mathfrak{F}) = umap-fo \mathfrak{F}*
unfolding *umap-fo-def* **by** *simp*

lemma (**in** *is-functor*) *op-umap-fo[cat-op-simps]*: *umap-fo (op-cf \mathfrak{F}) = umap-of \mathfrak{F}*
unfolding *umap-fo-def* **by** (*simp add: cat-op-simps*)

lemmas [*cat-op-simps*] =
is-functor.op-umap-of
is-functor.op-umap-fo

2.2.2 Arrow value

lemma *umap-of-ArrVal-vsuv[cat-cs-intros]*: *vsuv (umap-of \mathfrak{F} c r u d(ArrVal))*
unfolding *umap-of-def arr-field-simps* **by** (*simp add: nat-omega-simps*)

lemma *umap-fo-ArrVal-vsuv[cat-cs-intros]*: *vsuv (umap-fo \mathfrak{F} c r u d(ArrVal))*
unfolding *umap-fo-def* **by** (*rule umap-of-ArrVal-vsuv*)

lemma (**in** *is-functor*) *umap-of-ArrVal-vdomain*:
assumes $u : c \mapsto_{\mathfrak{B}} \mathfrak{F}(\text{ObjMap})(r)$
shows $\mathcal{D}_{\circ} (umap-of \ \mathfrak{F} \ c \ r \ u \ d(\text{ArrVal})) = \text{Hom } \mathfrak{A} \ r \ d$
unfolding *umap-of-components[OF assms]* **by** *simp*

lemmas [*cat-cs-simps*] = *is-functor.umap-of-ArrVal-vdomain*

lemma (**in** *is-functor*) *umap-fo-ArrVal-vdomain*:
assumes $u : \mathfrak{F}(\text{ObjMap})(r) \mapsto_{\mathfrak{B}} c$
shows $\mathcal{D}_{\circ} (umap-fo \ \mathfrak{F} \ c \ r \ u \ d(\text{ArrVal})) = \text{Hom } \mathfrak{A} \ d \ r$
unfolding *umap-fo-components[OF assms]* **by** *simp*

lemmas [*cat-cs-simps*] = *is-functor.umap-fo-ArrVal-vdomain*

lemma (**in** *is-functor*) *umap-of-ArrVal-app*:
assumes $f' : r \mapsto_{\mathfrak{A}} d$ **and** $u : c \mapsto_{\mathfrak{B}} \mathfrak{F}(\text{ObjMap})(r)$
shows *umap-of \mathfrak{F} c r u d(ArrVal)(f') = $\mathfrak{F}(\text{ArrMap})(f') \circ_{A \mathfrak{B}} u$*
using *assms(1)* **unfolding** *umap-of-components[OF assms(2)]* **by** *simp*

lemmas [*cat-cs-simps*] = *is-functor.umap-of-ArrVal-app*

lemma (**in** *is-functor*) *umap-fo-ArrVal-app*:
assumes $f' : d \mapsto_{\mathfrak{A}} r$ **and** $u : \mathfrak{F}(\text{ObjMap})(r) \mapsto_{\mathfrak{B}} c$
shows *umap-fo \mathfrak{F} c r u d(ArrVal)(f') = u \circ_{A \mathfrak{B}} \mathfrak{F}(\text{ArrMap})(f')*

proof–

from *assms* **have** $\mathfrak{F}(\text{ArrMap})(f') : \mathfrak{F}(\text{ObjMap})(d) \mapsto_{\mathfrak{B}} \mathfrak{F}(\text{ObjMap})(r)$
by (*auto intro: cat-cs-intros*)

from *this* *assms(2)* **have** $\mathfrak{F}f'$ [*simp*]:

$\mathfrak{F}(\text{ArrMap})(f') \circ_{A \text{ op-cat } \mathfrak{B}} u = u \circ_{A \mathfrak{B}} \mathfrak{F}(\text{ArrMap})(f')$

by (*simp add: cat-op-simps*)

from

is-functor-axioms

is-functor.umap-of-ArrVal-app

OF is-functor-op, unfolded cat-op-simps,
 OF assms
]

show ?thesis
by (simp add: cat-op-simps cat-cs-simps)

qed

lemmas [cat-cs-simps] = is-functor.umap-fo-ArrVal-app

lemma (in is-functor) umap-of-ArrVal-vrange:

assumes $u : c \mapsto_{\mathfrak{B}} \mathfrak{F}(\text{ObjMap})(r)$
shows $\mathcal{R}_o(\text{umap-of } \mathfrak{F} \ c \ r \ u \ d(\text{ArrVal})) \subseteq_o \text{Hom } \mathfrak{B} \ c \ (\mathfrak{F}(\text{ObjMap})(d))$

proof(intro vsubset-antisym vsubsetI)

interpret vsv $\langle \text{umap-of } \mathfrak{F} \ c \ r \ u \ d(\text{ArrVal}) \rangle$

unfolding umap-of-components[OF assms] **by** simp

fix g **assume** $g \in_o \mathcal{R}_o(\text{umap-of } \mathfrak{F} \ c \ r \ u \ d(\text{ArrVal}))$

then obtain f'

where g -def: $g = \text{umap-of } \mathfrak{F} \ c \ r \ u \ d(\text{ArrVal})(f')$
and f' : $f' \in_o \mathcal{D}_o(\text{umap-of } \mathfrak{F} \ c \ r \ u \ d(\text{ArrVal}))$

unfolding umap-of-components[OF assms] **by** auto

then have f' : $f' : r \mapsto_{\mathfrak{A}} d$

unfolding umap-of-ArrVal-vdomain[OF assms] **by** simp

then have $\mathfrak{F}f'$: $\mathfrak{F}(\text{ArrMap})(f') : \mathfrak{F}(\text{ObjMap})(r) \mapsto_{\mathfrak{B}} \mathfrak{F}(\text{ObjMap})(d)$
by (auto intro!: cat-cs-intros)

have g -def: $g = \mathfrak{F}(\text{ArrMap})(f') \circ_{\mathfrak{A}\mathfrak{B}} u$

unfolding g -def umap-of-ArrVal-app[OF f' assms]..

from $\mathfrak{F}f'$ assms **show** $g \in_o \text{Hom } \mathfrak{B} \ c \ (\mathfrak{F}(\text{ObjMap})(d))$

unfolding g -def **by** (auto intro: cat-cs-intros)

qed

lemma (in is-functor) umap-fo-ArrVal-vrange:

assumes $u : \mathfrak{F}(\text{ObjMap})(r) \mapsto_{\mathfrak{B}} c$
shows $\mathcal{R}_o(\text{umap-fo } \mathfrak{F} \ c \ r \ u \ d(\text{ArrVal})) \subseteq_o \text{Hom } \mathfrak{B} \ (\mathfrak{F}(\text{ObjMap})(d)) \ c$

by

(

rule is-functor.umap-of-ArrVal-vrange[
 OF is-functor-op, unfolded cat-op-simps, OF assms, folded umap-fo-def
]

)

2.2.3 Universal map is an arrow in the category Set

lemma (in is-functor) cf-arr-Set-umap-of:

assumes category $\alpha \mathfrak{A}$
and category $\alpha \mathfrak{B}$
and $r : r \in_o \mathfrak{A}(\text{Obj})$
and $d : d \in_o \mathfrak{A}(\text{Obj})$
and $w : u : c \mapsto_{\mathfrak{B}} \mathfrak{F}(\text{ObjMap})(r)$

shows arr-Set α (umap-of $\mathfrak{F} \ c \ r \ u \ d$)

proof(intro arr-SetI)

interpret HomDom: category $\alpha \mathfrak{A}$ **by** (rule assms(1))

interpret HomCod: category $\alpha \mathfrak{B}$ **by** (rule assms(2))

note umap-of-components = umap-of-components[OF u]

from $u \ d$ **have** $c : c \in_o \mathfrak{B}(\text{Obj})$ **and** $\mathfrak{F}d : (\mathfrak{F}(\text{ObjMap})(d)) \in_o \mathfrak{B}(\text{Obj})$
by (auto intro: cat-cs-intros)

show vfsequence (umap-of $\mathfrak{F} \ c \ r \ u \ d$) **unfolding** umap-of-def **by** simp

show vcard (umap-of $\mathfrak{F} \ c \ r \ u \ d$) = $3_{\mathbb{N}}$

unfolding umap-of-def **by** (simp add: nat-omega-simps)

from *umap-of-ArrVal-vrange*[*OF u*] **show**
 \mathcal{R}_\circ (*umap-of* \mathfrak{F} *c r u d*(*ArrVal*)) \subseteq_\circ *umap-of* \mathfrak{F} *c r u d*(*ArrCod*)
unfolding *umap-of-components* **by** *simp*
from *r d* **show** *umap-of* \mathfrak{F} *c r u d*(*ArrDom*) \in_\circ *Vset* α
unfolding *umap-of-components* **by** (*intro* *HomDom.cat-Hom-in-Vset*)
from *c* \mathfrak{F} *d* **show** *umap-of* \mathfrak{F} *c r u d*(*ArrCod*) \in_\circ *Vset* α
unfolding *umap-of-components* **by** (*intro* *HomCod.cat-Hom-in-Vset*)
qed (*auto simp: umap-of-components*[*OF u*])

lemma (*in is-functor*) *cf-arr-Set-umap-fo*:

assumes *category* α \mathfrak{A}
and *category* α \mathfrak{B}
and *r*: $r \in_\circ \mathfrak{A}(\text{Obj})$
and *d*: $d \in_\circ \mathfrak{A}(\text{Obj})$
and *w*: $u : \mathfrak{F}(\text{ObjMap})(\text{Obj}) \mapsto_{\mathfrak{B}} c$
shows *arr-Set* α (*umap-fo* \mathfrak{F} *c r u d*)

proof–

from *assms*(1) **have** \mathfrak{A} : *category* α (*op-cat* \mathfrak{A})
by (*auto intro: cat-cs-intros*)

from *assms*(2) **have** \mathfrak{B} : *category* α (*op-cat* \mathfrak{B})
by (*auto intro: cat-cs-intros*)

show *?thesis*

by

(

rule

is-functor.cf-arr-Set-umap-of[

OF is-functor-op, unfolded cat-op-simps, OF \mathfrak{A} \mathfrak{B} *r d u*

]

)

qed

lemma (*in is-functor*) *cf-umap-of-is-arr*:

assumes *category* α \mathfrak{A}
and *category* α \mathfrak{B}
and $r \in_\circ \mathfrak{A}(\text{Obj})$
and $d \in_\circ \mathfrak{A}(\text{Obj})$
and $u : c \mapsto_{\mathfrak{B}} \mathfrak{F}(\text{ObjMap})(\text{Obj})$
shows *umap-of* \mathfrak{F} *c r u d* : *Hom* \mathfrak{A} *r d* $\mapsto_{\text{cat-Set } \alpha}$ *Hom* \mathfrak{B} *c* ($\mathfrak{F}(\text{ObjMap})(\text{Obj})$)

proof(*intro cat-Set-is-arrI*)

show *arr-Set* α (*umap-of* \mathfrak{F} *c r u d*)

by (*rule cf-arr-Set-umap-of*[*OF assms*])

qed (*simp-all add: umap-of-components*[*OF assms*(5)])

lemma (*in is-functor*) *cf-umap-of-is-arr'*:

assumes *category* α \mathfrak{A}
and *category* α \mathfrak{B}
and $r \in_\circ \mathfrak{A}(\text{Obj})$
and $d \in_\circ \mathfrak{A}(\text{Obj})$
and $u : c \mapsto_{\mathfrak{B}} \mathfrak{F}(\text{ObjMap})(\text{Obj})$
and $A = \text{Hom } \mathfrak{A} \text{ } r \text{ } d$
and $B = \text{Hom } \mathfrak{B} \text{ } c \text{ } (\mathfrak{F}(\text{ObjMap})(\text{Obj}))$
and $\mathfrak{C} = \text{cat-Set } \alpha$
shows *umap-of* \mathfrak{F} *c r u d* : $A \mapsto_{\mathfrak{C}} B$
using *assms*(1–5) **unfolding** *assms*(6–8) **by** (*rule cf-umap-of-is-arr*)

lemmas [*cat-cs-intros*] = *is-functor.cf-umap-of-is-arr'*

lemma (*in is-functor*) *cf-umap-fo-is-arr*:

assumes *category* $\alpha \mathfrak{A}$
and *category* $\alpha \mathfrak{B}$
and $r \in_{\circ} \mathfrak{A}(\text{Obj})$
and $d \in_{\circ} \mathfrak{A}(\text{Obj})$
and $u : \mathfrak{F}(\text{ObjMap})(\downarrow r) \mapsto_{\mathfrak{B}} c$
shows *umap-fo* $\mathfrak{F} c r u d : \text{Hom } \mathfrak{A} d r \mapsto_{\text{cat-Set } \alpha} \text{Hom } \mathfrak{B} (\mathfrak{F}(\text{ObjMap})(\downarrow d)) c$
proof(*intro cat-Set-is-arrI*)
show *arr-Set* α (*umap-fo* $\mathfrak{F} c r u d$)
by (*rule cf-arr-Set-umap-fo[OF assms]*)
qed (*simp-all add: umap-fo-components[OF assms(5)]*)

lemma (*in is-functor*) *cf-umap-fo-is-arr'*:
assumes *category* $\alpha \mathfrak{A}$
and *category* $\alpha \mathfrak{B}$
and $r \in_{\circ} \mathfrak{A}(\text{Obj})$
and $d \in_{\circ} \mathfrak{A}(\text{Obj})$
and $u : \mathfrak{F}(\text{ObjMap})(\downarrow r) \mapsto_{\mathfrak{B}} c$
and $A = \text{Hom } \mathfrak{A} d r$
and $B = \text{Hom } \mathfrak{B} (\mathfrak{F}(\text{ObjMap})(\downarrow d)) c$
and $\mathfrak{C} = \text{cat-Set } \alpha$
shows *umap-fo* $\mathfrak{F} c r u d : A \mapsto_{\mathfrak{C}} B$
using *assms(1-5) unfolding assms(6-8) by (rule cf-umap-fo-is-arr)*

lemmas [*cat-cs-intros*] = *is-functor.cf-umap-fo-is-arr'*

2.3 Universal arrow: definition and elementary properties

See Chapter III-1 in [9].

definition *universal-arrow-of* $:: V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow \text{bool}$
where *universal-arrow-of* $\mathfrak{F} c r u \longleftrightarrow$
 $($
 $r \in_{\circ} \mathfrak{F}(\text{HomDom})(\downarrow \text{Obj}) \wedge$
 $u : c \mapsto_{\mathfrak{F}(\text{HomCod})} \mathfrak{F}(\text{ObjMap})(\downarrow r) \wedge$
 $($
 $\forall r' u'.$
 $r' \in_{\circ} \mathfrak{F}(\text{HomDom})(\downarrow \text{Obj}) \longrightarrow$
 $u' : c \mapsto_{\mathfrak{F}(\text{HomCod})} \mathfrak{F}(\text{ObjMap})(\downarrow r') \longrightarrow$
 $(\exists ! f'. f' : r \mapsto_{\mathfrak{F}(\text{HomDom})} r' \wedge u' = \text{umap-of } \mathfrak{F} c r u r'(\text{ArrVal})(\downarrow f'))$
 $)$
 $)$

definition *universal-arrow-fo* $:: V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow \text{bool}$
where *universal-arrow-fo* $\mathfrak{F} c r u \equiv \text{universal-arrow-of } (\text{op-cf } \mathfrak{F}) c r u$

Rules.

mk-ide (*in is-functor*) **rf**
universal-arrow-of-def[**where** $\mathfrak{F}=\mathfrak{F}$, *unfolded cf-HomDom cf-HomCod*]
[*intro universal-arrow-ofI*]
[*dest universal-arrow-ofD*[*dest*]]
[*elim universal-arrow-ofE*[*elim*]]

lemma (*in is-functor*) *universal-arrow-foI*:
assumes $r \in_{\circ} \mathfrak{A}(\text{Obj})$
and $u : \mathfrak{F}(\text{ObjMap})(\downarrow r) \mapsto_{\mathfrak{B}} c$
and $\wedge r' u'. [\![r' \in_{\circ} \mathfrak{A}(\text{Obj}); u' : \mathfrak{F}(\text{ObjMap})(\downarrow r') \mapsto_{\mathfrak{B}} c]\!] \implies$
 $\exists ! f'. f' : r' \mapsto_{\mathfrak{A}} r \wedge u' = \text{umap-fo } \mathfrak{F} c r u r'(\text{ArrVal})(\downarrow f')$
shows *universal-arrow-fo* $\mathfrak{F} c r u$

```

by
(
  simp add:
    is-functor.universal-arrow-ofI
    [
      OF is-functor-op,
      folded universal-arrow-fo-def,
      unfolded cat-op-simps,
      OF assms
    ]
)

```

```

lemma (in is-functor) universal-arrow-foD[dest]:
  assumes universal-arrow-fo  $\mathfrak{F}$  c r u
  shows  $r \in_{\circ} \mathfrak{A}(\text{Obj})$ 
  and  $u : \mathfrak{F}(\text{ObjMap})(r) \mapsto_{\mathfrak{B}} c$ 
  and  $\bigwedge r' u'. [\![ r' \in_{\circ} \mathfrak{A}(\text{Obj}); u' : \mathfrak{F}(\text{ObjMap})(r') \mapsto_{\mathfrak{B}} c ]\!] \implies$ 
     $\exists ! f'. f' : r' \mapsto_{\mathfrak{A}} r \wedge u' = \text{umap-fo } \mathfrak{F} c r u r'(\text{Arr Val})(f')$ 
  by
  (
    auto simp:
      is-functor.universal-arrow-ofD
      [
        OF is-functor-op,
        folded universal-arrow-fo-def,
        unfolded cat-op-simps,
        OF assms
      ]
  )

```

```

lemma (in is-functor) universal-arrow-foE[elim]:
  assumes universal-arrow-fo  $\mathfrak{F}$  c r u
  obtains  $r \in_{\circ} \mathfrak{A}(\text{Obj})$ 
  and  $u : \mathfrak{F}(\text{ObjMap})(r) \mapsto_{\mathfrak{B}} c$ 
  and  $\bigwedge r' u'. [\![ r' \in_{\circ} \mathfrak{A}(\text{Obj}); u' : \mathfrak{F}(\text{ObjMap})(r') \mapsto_{\mathfrak{B}} c ]\!] \implies$ 
     $\exists ! f'. f' : r' \mapsto_{\mathfrak{A}} r \wedge u' = \text{umap-fo } \mathfrak{F} c r u r'(\text{Arr Val})(f')$ 
  using assms by (auto simp: universal-arrow-foD)

```

Elementary properties.

```

lemma (in is-functor) op-cf-universal-arrow-of[cat-op-simps]:
  universal-arrow-of (op-cf  $\mathfrak{F}$ ) c r u  $\longleftrightarrow$  universal-arrow-fo  $\mathfrak{F}$  c r u
  unfolding universal-arrow-fo-def ..

```

```

lemma (in is-functor) op-cf-universal-arrow-fo[cat-op-simps]:
  universal-arrow-fo (op-cf  $\mathfrak{F}$ ) c r u  $\longleftrightarrow$  universal-arrow-of  $\mathfrak{F}$  c r u
  unfolding universal-arrow-fo-def cat-op-simps ..

```

```

lemmas (in is-functor) [cat-op-simps] =
  is-functor.op-cf-universal-arrow-of
  is-functor.op-cf-universal-arrow-fo

```

2.4 Uniqueness

The following properties are related to the uniqueness of the universal arrow. These properties can be inferred from the content of Chapter III-1 in [9].

```

lemma (in is-functor) cf-universal-arrow-of-ex-is-iso-arr:

```

— The proof is based on the ideas expressed in the proof of Theorem 5.2 in Chapter Introduction in [6].

assumes *universal-arrow-of* $\mathfrak{F} \ c \ r \ u$ **and** *universal-arrow-of* $\mathfrak{F} \ c \ r' \ u'$
obtains f **where** $f : r \mapsto_{iso\mathfrak{A}} r'$ **and** $u' = \text{umap-of } \mathfrak{F} \ c \ r \ u \ r'(\text{ArrVal})(f)$
proof–

note $ua1 = \text{universal-arrow-ofD}[OF \ \text{assms}(1)]$

note $ua2 = \text{universal-arrow-ofD}[OF \ \text{assms}(2)]$

from $ua1(1)$ **have** $\mathfrak{A}r : \mathfrak{A}(\text{CId})(r) : r \mapsto_{\mathfrak{A}} r$ **by** (*auto intro: cat-cs-intros*)

from $ua1(1)$ **have** $\mathfrak{F}(\text{ArrMap})(\mathfrak{A}(\text{CId})(r)) = \mathfrak{B}(\text{CId})(\mathfrak{F}(\text{ObjMap})(r))$

by (*auto intro: cat-cs-intros*)

with $ua1(1,2)$ **have** $u\text{-def}: u = \text{umap-of } \mathfrak{F} \ c \ r \ u \ r(\text{ArrVal})(\mathfrak{A}(\text{CId})(r))$

unfolding $\text{umap-of-ArrVal-app}[OF \ \mathfrak{A}r \ ua1(2)]$ **by** (*auto simp: cat-cs-simps*)

from $ua2(1)$ **have** $\mathfrak{A}r' : \mathfrak{A}(\text{CId})(r') : r' \mapsto_{\mathfrak{A}} r'$ **by** (*auto intro: cat-cs-intros*)

from $ua2(1)$ **have** $\mathfrak{F}(\text{ArrMap})(\mathfrak{A}(\text{CId})(r')) = \mathfrak{B}(\text{CId})(\mathfrak{F}(\text{ObjMap})(r'))$

by (*auto intro: cat-cs-intros*)

with $ua2(1,2)$ **have** $u'\text{-def}: u' = \text{umap-of } \mathfrak{F} \ c \ r' \ u' \ r'(\text{ArrVal})(\mathfrak{A}(\text{CId})(r'))$

unfolding $\text{umap-of-ArrVal-app}[OF \ \mathfrak{A}r' \ ua2(2)]$ **by** (*auto simp: cat-cs-simps*)

from $\mathfrak{A}r \ u\text{-def}$ *universal-arrow-ofD*(3)[$OF \ \text{assms}(1) \ ua1(1,2)$] **have** $eq\text{-CId-rI}$:

$\llbracket f' : r \mapsto_{\mathfrak{A}} r; u = \text{umap-of } \mathfrak{F} \ c \ r \ u \ r(\text{ArrVal})(f') \rrbracket \implies f' = \mathfrak{A}(\text{CId})(r)$

for f'

by *blast*

from $\mathfrak{A}r' \ u'\text{-def}$ *universal-arrow-ofD*(3)[$OF \ \text{assms}(2) \ ua2(1,2)$] **have** $eq\text{-CId-r'I}$:

$\llbracket f' : r' \mapsto_{\mathfrak{A}} r'; u' = \text{umap-of } \mathfrak{F} \ c \ r' \ u' \ r'(\text{ArrVal})(f') \rrbracket \implies$

$f' = \mathfrak{A}(\text{CId})(r')$

for f'

by *blast*

from $ua1(3)[OF \ ua2(1,2)]$ **obtain** f

where $f : r \mapsto_{\mathfrak{A}} r'$

and $u\text{-def}: u' = \text{umap-of } \mathfrak{F} \ c \ r \ u \ r'(\text{ArrVal})(f)$

and $g : r \mapsto_{\mathfrak{A}} r' \implies u' = \text{umap-of } \mathfrak{F} \ c \ r \ u \ r'(\text{ArrVal})(g) \implies f = g$

for g

by *metis*

from $ua2(3)[OF \ ua1(1,2)]$ **obtain** f'

where $f' : r' \mapsto_{\mathfrak{A}} r$

and $u\text{-def}: u = \text{umap-of } \mathfrak{F} \ c \ r' \ u' \ r(\text{ArrVal})(f')$

and $g : r' \mapsto_{\mathfrak{A}} r \implies u = \text{umap-of } \mathfrak{F} \ c \ r' \ u' \ r(\text{ArrVal})(g) \implies f' = g$

for g

by *metis*

have $f : r \mapsto_{iso\mathfrak{A}} r'$

proof(*intro is-iso-arrI is-inverseI*)

show $f : r \mapsto_{\mathfrak{A}} r'$ **by** (*rule f*)

show $f' : r' \mapsto_{\mathfrak{A}} r$ **by** (*rule f'*)

show $f : r \mapsto_{\mathfrak{A}} r'$ **by** (*rule f*)

from f' **have** $\mathfrak{F}f' : \mathfrak{F}(\text{ArrMap})(f') : \mathfrak{F}(\text{ObjMap})(r') \mapsto_{\mathfrak{B}} \mathfrak{F}(\text{ObjMap})(r)$

by (*auto intro: cat-cs-intros*)

from f **have** $\mathfrak{F}f : \mathfrak{F}(\text{ArrMap})(f) : \mathfrak{F}(\text{ObjMap})(r) \mapsto_{\mathfrak{B}} \mathfrak{F}(\text{ObjMap})(r')$

by (*auto intro: cat-cs-intros*)

note $u'\text{-def}' = u'\text{-def}[symmetric, \text{unfolded } \text{umap-of-ArrVal-app}[OF \ f \ ua1(2)]]$

and $u\text{-def}' = u\text{-def}[symmetric, \text{unfolded } \text{umap-of-ArrVal-app}[OF \ f' \ ua2(2)]]$

show $f' \circ_{A\mathfrak{A}} f = \mathfrak{A}(\text{CId})(r)$

proof(*rule eq-CId-rI*)

from $f \ f'$ **show** $f'f : f' \circ_{A\mathfrak{A}} f : r \mapsto_{\mathfrak{A}} r$

by (auto intro: cat-cs-intros)
 from ua1(2) $\mathfrak{F}f' \mathfrak{F}f$ show $u = \text{umap-of } \mathfrak{F} c r u r (\text{ArrVal}) (f' \circ_{A\mathfrak{Q}} f)$
 unfolding *umap-of-ArrVal-app*[OF f'f ua1(2)] *cf-ArrMap-Comp*[OF f'f]
 by (simp add: *HomCod.cat-Comp-assoc u'-def' u-def'*)
 qed
 show $f \circ_{A\mathfrak{Q}} f' = \mathfrak{Q}(\text{CIId})(r')$
 proof(rule *eq-CId-r'I*)
 from f f' show $\mathfrak{F}f': f \circ_{A\mathfrak{Q}} f' : r' \mapsto_{\mathfrak{Q}} r'$
 by (auto intro: cat-cs-intros)
 from ua2(2) $\mathfrak{F}f' \mathfrak{F}f$ show $u' = \text{umap-of } \mathfrak{F} c r' u' r' (\text{ArrVal}) (f \circ_{A\mathfrak{Q}} f')$
 unfolding *umap-of-ArrVal-app*[OF ff' ua2(2)] *cf-ArrMap-Comp*[OF f f']
 by (simp add: *HomCod.cat-Comp-assoc u'-def' u-def'*)
 qed
 qed

with u' -def that show ?thesis by auto

qed

lemma (in *is-functor*) *cf-universal-arrow-fo-ex-is-iso-arr*:
 assumes *universal-arrow-fo* $\mathfrak{F} c r u$
 and *universal-arrow-fo* $\mathfrak{F} c r' u'$
 obtains f where $f : r' \mapsto_{\text{iso}\mathfrak{Q}} r$ and $u' = \text{umap-fo } \mathfrak{F} c r u r' (\text{ArrVal}) (f)$
 by
 (
 elim
 is-functor.cf-universal-arrow-fo-ex-is-iso-arr[
 OF *is-functor-op*, *unfolded cat-op-simps*, OF *assms*
]
)

lemma (in *is-functor*) *cf-universal-arrow-fo-unique*:
 assumes *universal-arrow-fo* $\mathfrak{F} c r u$
 and *universal-arrow-fo* $\mathfrak{F} c r' u'$
 shows $\exists!f'. f' : r' \mapsto_{\mathfrak{Q}} r \wedge u' = \text{umap-of } \mathfrak{F} c r u r' (\text{ArrVal}) (f')$
 proof-
 note $ua1 = \text{universal-arrow-foD}[OF \text{assms}(1)]$
 note $ua2 = \text{universal-arrow-foD}[OF \text{assms}(2)]$
 from ua1(3)[OF ua2(1,2)] show ?thesis .
 qed

lemma (in *is-functor*) *cf-universal-arrow-fo-unique*:
 assumes *universal-arrow-fo* $\mathfrak{F} c r u$
 and *universal-arrow-fo* $\mathfrak{F} c r' u'$
 shows $\exists!f'. f' : r' \mapsto_{\mathfrak{Q}} r \wedge u' = \text{umap-fo } \mathfrak{F} c r u r' (\text{ArrVal}) (f')$
 proof-
 note $ua1 = \text{universal-arrow-foD}[OF \text{assms}(1)]$
 note $ua2 = \text{universal-arrow-foD}[OF \text{assms}(2)]$
 from ua1(3)[OF ua2(1,2)] show ?thesis .
 qed

lemma (in *is-functor*) *cf-universal-arrow-fo-is-iso-arr*:
 assumes *universal-arrow-fo* $\mathfrak{F} c r u$
 and *universal-arrow-fo* $\mathfrak{F} c r' u'$
 and $f : r \mapsto_{\mathfrak{Q}} r'$
 and $u' = \text{umap-of } \mathfrak{F} c r u r' (\text{ArrVal}) (f)$
 shows $f : r \mapsto_{\text{iso}\mathfrak{Q}} r'$
 proof-

from $assms(3,4)$ *cf-universal-arrow-of-unique*[OF $assms(1,2)$] **have** eq :
 $g : r \mapsto_{\mathfrak{A}} r' \implies u' = \text{umap-of } \mathfrak{F} \ c \ r \ u \ r' (\downarrow ArrVal) (\downarrow g) \implies f = g$ **for** g
by *blast*
from $assms(1,2)$ **obtain** f'
where $iso-f'$: $f' : r \mapsto_{iso\mathfrak{A}} r'$
and u' -def: $u' = \text{umap-of } \mathfrak{F} \ c \ r \ u \ r' (\downarrow ArrVal) (\downarrow f')$
by (*auto elim: cf-universal-arrow-of-ex-is-iso-arr*)
then have $f' : f' : r \mapsto_{\mathfrak{A}} r'$ **by** *auto*
from $iso-f'$ **show** *?thesis unfolding* $eq[OF \ f' \ u'$ -def, *symmetric*].
qed

lemma (**in** *is-functor*) *cf-universal-arrow-fo-is-iso-arr*:
assumes *universal-arrow-fo* $\mathfrak{F} \ c \ r \ u$
and *universal-arrow-fo* $\mathfrak{F} \ c \ r' \ u'$
and $f : r' \mapsto_{\mathfrak{A}} r$
and $u' = \text{umap-fo } \mathfrak{F} \ c \ r \ u \ r' (\downarrow ArrVal) (\downarrow f)$
shows $f : r' \mapsto_{iso\mathfrak{A}} r$
by
(*rule*
is-functor.cf-universal-arrow-of-is-iso-arr[
 OF *is-functor-op*, *unfolded cat-op-simps*, OF *assms*
])

lemma (**in** *is-functor*) *universal-arrow-of-if-universal-arrow-of*:
assumes *universal-arrow-of* $\mathfrak{F} \ c \ r \ u$
and $f : r \mapsto_{iso\mathfrak{A}} r'$
and $u' = \text{umap-of } \mathfrak{F} \ c \ r \ u \ r' (\downarrow ArrVal) (\downarrow f)$
shows *universal-arrow-of* $\mathfrak{F} \ c \ r' \ u'$
proof(*intro universal-arrow-ofI assms(2)*)

note $ua = \text{universal-arrow-ofD}[OF \ assms(1)]$
note $f = \text{is-iso-arrD}(1)[OF \ assms(2)]$
from $assms(3)$ $ua(1,2)$ f **have** u' -def: $u' = \mathfrak{F}(\downarrow ArrMap)(\downarrow f) \circ_{A\mathfrak{B}} u$
by (*cs-prems cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)
from $ua(2)$ f **show** $u' : u' : c \mapsto_{\mathfrak{B}} \mathfrak{F}(\downarrow ObjMap)(\downarrow r')$
unfolding u' -def **by** (*cs-concl cs-intro: cat-cs-intros*)

from $f(1)$ **show** $r' \in_0 \mathfrak{A}(\downarrow Obj)$ **by** *auto*

fix $r'' \ u''$ **assume** $prems : r'' \in_0 \mathfrak{A}(\downarrow Obj) \ u'' : c \mapsto_{\mathfrak{B}} \mathfrak{F}(\downarrow ObjMap)(\downarrow r'')$

from $ua(3)[OF \ prems]$ **obtain** f'
where f' : $f' : r \mapsto_{\mathfrak{A}} r''$
and u'' -def: $u'' = \text{umap-of } \mathfrak{F} \ c \ r \ u \ r'' (\downarrow ArrVal) (\downarrow f')$
and f' -unique: $\wedge f''$.
 $[[f'' : r \mapsto_{\mathfrak{A}} r''; u'' = \text{umap-of } \mathfrak{F} \ c \ r \ u \ r'' (\downarrow ArrVal) (\downarrow f'')]] \implies$
 $f'' = f'$
by *metis*

from u'' -def f' $ua(2)$ **have** [*cat-cs-simps*]: $\mathfrak{F}(\downarrow ArrMap)(\downarrow f') \circ_{A\mathfrak{B}} u = u''$
by (*cs-prems cs-simp: cat-cs-simps cs-intro: cat-cs-intros simp*)

show $\exists! f' . f' : r' \mapsto_{\mathfrak{A}} r'' \wedge u'' = \text{umap-of } \mathfrak{F} \ c \ r' \ u' \ r'' (\downarrow ArrVal) (\downarrow f')$

proof(*intro ex1I conjI; (elim conjE)?*)

from f' $assms(2)$ f **show** $f' \circ_{A\mathfrak{A}} f^{-1} \ C_{\mathfrak{A}} : r' \mapsto_{\mathfrak{A}} r''$
by (*cs-concl cs-intro: cat-cs-intros cat-arrow-cs-intros*)

have

$$\begin{aligned} & \mathfrak{F}(\downarrow \text{ArrMap})(f') \circ_{A\mathfrak{B}} (\mathfrak{F}(\downarrow \text{ArrMap})(f^{-1} C\mathfrak{A}) \circ_{A\mathfrak{B}} u') = \\ & \mathfrak{F}(\downarrow \text{ArrMap})(f') \circ_{A\mathfrak{B}} (\mathfrak{F}(\downarrow \text{ArrMap})(f^{-1} C\mathfrak{A}) \circ_{A\mathfrak{B}} (\mathfrak{F}(\downarrow \text{ArrMap})(f) \circ_{A\mathfrak{B}} u)) \end{aligned}$$

unfolding u' -def ..

also from f' *assms*(2) $u' f ua$ (2) have

$$\dots = \mathfrak{F}(\downarrow \text{ArrMap})(f') \circ_{A\mathfrak{B}} (\mathfrak{F}(\downarrow \text{ArrMap})(f^{-1} C\mathfrak{A} \circ_{A\mathfrak{A}} f)) \circ_{A\mathfrak{B}} u$$

by

$$\begin{aligned} & (\\ & \quad \text{cs-concl} \\ & \quad \text{cs-simp: cat-cs-simps cs-intro: cat-arrow-cs-intros cat-cs-intros} \\ &) \end{aligned}$$

also from f' *assms*(2) $f ua$ (2) have $\dots = u''$

by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)

finally have [cat-cs-simps]:

$$\mathfrak{F}(\downarrow \text{ArrMap})(f') \circ_{A\mathfrak{B}} (\mathfrak{F}(\downarrow \text{ArrMap})(f^{-1} C\mathfrak{A}) \circ_{A\mathfrak{B}} u') = u''.$$

from f' *assms*(2) $u' f$ show

$$u'' = \text{umap-of } \mathfrak{F} \text{ c } r' u' r''(\downarrow \text{ArrVal})(f' \circ_{A\mathfrak{A}} f^{-1} C\mathfrak{A})$$

by

$$\begin{aligned} & (\\ & \quad \text{cs-concl} \\ & \quad \text{cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-arrow-cs-intros} \\ &) \end{aligned}$$

fix g assume *prems*':

$$g : r' \mapsto_{\mathfrak{A}} r'' u'' = \text{umap-of } \mathfrak{F} \text{ c } r' u' r''(\downarrow \text{ArrVal})(g)$$

from *prems*'(1) f have $gf : g \circ_{A\mathfrak{A}} f : r \mapsto_{\mathfrak{A}} r''$

by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)

from *prems*'(2,1) *assms*(2) u' have $u'' = \mathfrak{F}(\downarrow \text{ArrMap})(g) \circ_{A\mathfrak{B}} u'$

by (cs-prems cs-simp: cat-cs-simps cs-intro: cat-cs-intros)

also from *prems*'(1) $f ua$ (2) have

$$\dots = \mathfrak{F}(\downarrow \text{ArrMap})(g) \circ_{A\mathfrak{B}} \mathfrak{F}(\downarrow \text{ArrMap})(f) \circ_{A\mathfrak{B}} u$$

by (cs-concl cs-simp: cat-cs-simps u' -def u'' -def cs-intro: cat-cs-intros)

also from *prems*'(1) $f ua$ (2) have

$$\dots = \text{umap-of } \mathfrak{F} \text{ c } r u r''(\downarrow \text{ArrVal})(g \circ_{A\mathfrak{A}} f)$$

by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)

finally have $u'' = \text{umap-of } \mathfrak{F} \text{ c } r u r''(\downarrow \text{ArrVal})(g \circ_{A\mathfrak{A}} f)$.

from f' -unique[OF gf this] have $g \circ_{A\mathfrak{A}} f = f'$.

then have $(g \circ_{A\mathfrak{A}} f) \circ_{A\mathfrak{A}} f^{-1} C\mathfrak{A} = f' \circ_{A\mathfrak{A}} f^{-1} C\mathfrak{A}$ by *simp*

from this *assms*(2) *prems*'(1) $u' f ua$ (2) show $g = f' \circ_{A\mathfrak{A}} f^{-1} C\mathfrak{A}$

by

$$\begin{aligned} & (\\ & \quad \text{cs-prems} \\ & \quad \text{cs-simp: cat-cs-simps cs-intro: cat-arrow-cs-intros cat-cs-intros} \\ &) \end{aligned}$$

qed

qed

lemma (in *is-functor*) *universal-arrow-fo-if-universal-arrow-fo*:

assumes *universal-arrow-fo* $\mathfrak{F} \text{ c } r u$

and $f : r' \mapsto_{\text{iso}\mathfrak{A}} r$

and $u' = \text{umap-fo } \mathfrak{F} \text{ c } r u r'(\downarrow \text{ArrVal})(f)$

shows *universal-arrow-fo* $\mathfrak{F} \text{ c } r' u'$

by

$$\begin{aligned} & (\\ & \quad \text{rule is-functor.universal-arrow-of-if-universal-arrow-of[} \\ & \quad \quad \text{OF is-functor-op, unfolded cat-op-simps, OF assms} \\ & \quad] \\ &) \end{aligned}$$

2.5 Universal natural transformation

2.5.1 Definition and elementary properties

The concept of the universal natural transformation is introduced for the statement of the elements of a variant of Proposition 1 in Chapter III-2 in [9].

definition $ntcf\text{-}ua\text{-}of :: V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$

where $ntcf\text{-}ua\text{-}of \alpha \mathfrak{F} c r u =$

[
 $(\lambda d \in_{\circ} \mathfrak{F}(\mathit{HomDom})(\mathit{Obj}). \mathit{umap}\text{-}of \mathfrak{F} c r u d),$
 $\mathit{Hom}_{O.C\alpha} \mathfrak{F}(\mathit{HomDom})(r, -),$
 $\mathit{Hom}_{O.C\alpha} \mathfrak{F}(\mathit{HomCod})(c, -) \circ_{CF} \mathfrak{F},$
 $\mathfrak{F}(\mathit{HomDom}),$
 $\mathit{cat}\text{-}Set \alpha$
]_o

definition $ntcf\text{-}ua\text{-}fo :: V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$

where $ntcf\text{-}ua\text{-}fo \alpha \mathfrak{F} c r u = ntcf\text{-}ua\text{-}of \alpha (op\text{-}cf \mathfrak{F}) c r u$

Components.

lemma $ntcf\text{-}ua\text{-}of\text{-}components:$

shows $ntcf\text{-}ua\text{-}of \alpha \mathfrak{F} c r u(\mathit{NTMap}) = (\lambda d \in_{\circ} \mathfrak{F}(\mathit{HomDom})(\mathit{Obj}). \mathit{umap}\text{-}of \mathfrak{F} c r u d)$

and $ntcf\text{-}ua\text{-}of \alpha \mathfrak{F} c r u(\mathit{NTDom}) = \mathit{Hom}_{O.C\alpha} \mathfrak{F}(\mathit{HomDom})(r, -)$

and $ntcf\text{-}ua\text{-}of \alpha \mathfrak{F} c r u(\mathit{NTCod}) = \mathit{Hom}_{O.C\alpha} \mathfrak{F}(\mathit{HomCod})(c, -) \circ_{CF} \mathfrak{F}$

and $ntcf\text{-}ua\text{-}of \alpha \mathfrak{F} c r u(\mathit{NTDGDom}) = \mathfrak{F}(\mathit{HomDom})$

and $ntcf\text{-}ua\text{-}of \alpha \mathfrak{F} c r u(\mathit{NTDGCod}) = \mathit{cat}\text{-}Set \alpha$

unfolding $ntcf\text{-}ua\text{-}of\text{-}def$ $nt\text{-}field\text{-}simps$ **by** $(simp\text{-}all \text{ add: } nat\text{-}\omega\text{-}simps)$

lemma $ntcf\text{-}ua\text{-}fo\text{-}components:$

shows $ntcf\text{-}ua\text{-}fo \alpha \mathfrak{F} c r u(\mathit{NTMap}) = (\lambda d \in_{\circ} \mathfrak{F}(\mathit{HomDom})(\mathit{Obj}). \mathit{umap}\text{-}fo \mathfrak{F} c r u d)$

and $ntcf\text{-}ua\text{-}fo \alpha \mathfrak{F} c r u(\mathit{NTDom}) = \mathit{Hom}_{O.C\alpha} op\text{-}cat (\mathfrak{F}(\mathit{HomDom}))(r, -)$

and $ntcf\text{-}ua\text{-}fo \alpha \mathfrak{F} c r u(\mathit{NTCod}) =$

$\mathit{Hom}_{O.C\alpha} op\text{-}cat (\mathfrak{F}(\mathit{HomCod}))(c, -) \circ_{CF} op\text{-}cf \mathfrak{F}$

and $ntcf\text{-}ua\text{-}fo \alpha \mathfrak{F} c r u(\mathit{NTDGDom}) = op\text{-}cat (\mathfrak{F}(\mathit{HomDom}))$

and $ntcf\text{-}ua\text{-}fo \alpha \mathfrak{F} c r u(\mathit{NTDGCod}) = \mathit{cat}\text{-}Set \alpha$

unfolding $ntcf\text{-}ua\text{-}fo\text{-}def$ $ntcf\text{-}ua\text{-}of\text{-}components$ $umap\text{-}fo\text{-}def$ $cat\text{-}op\text{-}simps$

by $simp\text{-}all$

context $is\text{-}functor$

begin

lemmas $ntcf\text{-}ua\text{-}of\text{-}components'$ =

$ntcf\text{-}ua\text{-}of\text{-}components$ [**where** $\alpha = \alpha$ **and** $\mathfrak{F} = \mathfrak{F}$, $unfolding$ $cat\text{-}cs\text{-}simps$]

lemmas [$cat\text{-}cs\text{-}simps$] = $ntcf\text{-}ua\text{-}of\text{-}components'$ (2-5)

lemma $ntcf\text{-}ua\text{-}fo\text{-}components':$

assumes $c \in_{\circ} \mathfrak{B}(\mathit{Obj})$ **and** $r \in_{\circ} \mathfrak{A}(\mathit{Obj})$

shows $ntcf\text{-}ua\text{-}fo \alpha \mathfrak{F} c r u(\mathit{NTMap}) = (\lambda d \in_{\circ} \mathfrak{A}(\mathit{Obj}). \mathit{umap}\text{-}fo \mathfrak{F} c r u d)$

and [$cat\text{-}cs\text{-}simps$]:

$ntcf\text{-}ua\text{-}fo \alpha \mathfrak{F} c r u(\mathit{NTDom}) = \mathit{Hom}_{O.C\alpha} \mathfrak{A}(-, r)$

and [$cat\text{-}cs\text{-}simps$]:

$ntcf\text{-}ua\text{-}fo \alpha \mathfrak{F} c r u(\mathit{NTCod}) = \mathit{Hom}_{O.C\alpha} \mathfrak{B}(-, c) \circ_{CF} op\text{-}cf \mathfrak{F}$

and [$cat\text{-}cs\text{-}simps$]: $ntcf\text{-}ua\text{-}fo \alpha \mathfrak{F} c r u(\mathit{NTDGDom}) = op\text{-}cat \mathfrak{A}$

and [$cat\text{-}cs\text{-}simps$]: $ntcf\text{-}ua\text{-}fo \alpha \mathfrak{F} c r u(\mathit{NTDGCod}) = \mathit{cat}\text{-}Set \alpha$

unfolding

$ntcf\text{-}ua\text{-}fo\text{-}components$ $cat\text{-}cs\text{-}simps$

$\mathit{HomDom}.cat\text{-}op\text{-}cat\text{-}cf\text{-}\mathit{Hom}\text{-}snd$ [OF $assms(2)$]

HomCod.cat-op-cat-cf-Hom-snd[*OF assms*(1)]
 by *simp-all*

end

lemmas [*cat-cs-simps*] =
is-functor.ntcf-ua-of-components'(2-5)
is-functor.ntcf-ua-fo-components'(2-5)

2.5.2 Natural transformation map

mk-VLambda (in *is-functor*)
ntcf-ua-of-components(1)[**where** $\alpha=\alpha$ **and** $\mathfrak{F}=\mathfrak{F}$, *unfolded cf-HomDom*]
 |*vsv ntcf-ua-of-NTMap-vsv*|
 |*vdomain ntcf-ua-of-NTMap-vdomain*|
 |*app ntcf-ua-of-NTMap-app*|

context *is-functor*
begin

context
fixes *c r*
assumes *r*: $r \in_{\circ} \mathfrak{A}(\text{Obj})$ **and** *c*: $c \in_{\circ} \mathfrak{B}(\text{Obj})$
begin

mk-VLambda *ntcf-ua-fo-components'*(1)[*OF c r*]
 |*vsv ntcf-ua-fo-NTMap-vsv*|
 |*vdomain ntcf-ua-fo-NTMap-vdomain*|
 |*app ntcf-ua-fo-NTMap-app*|

end

end

lemmas [*cat-cs-intros*] =
is-functor.ntcf-ua-fo-NTMap-vsv
is-functor.ntcf-ua-of-NTMap-vsv

lemmas [*cat-cs-simps*] =
is-functor.ntcf-ua-fo-NTMap-vdomain
is-functor.ntcf-ua-fo-NTMap-app
is-functor.ntcf-ua-of-NTMap-vdomain
is-functor.ntcf-ua-of-NTMap-app

lemma (in *is-functor*) *ntcf-ua-of-NTMap-vrange*:

assumes *category* $\alpha \mathfrak{A}$
and *category* $\alpha \mathfrak{B}$
and $r \in_{\circ} \mathfrak{A}(\text{Obj})$
and $u : c \mapsto_{\mathfrak{B}} \mathfrak{F}(\text{ObjMap})(r)$
shows $\mathcal{R}_{\circ}(\text{ntcf-ua-of } \alpha \mathfrak{F} \text{ } c \text{ } r \text{ } u(\text{NTMap})) \subseteq_{\circ} \text{cat-Set } \alpha(\text{Arr})$

proof(*rule vsv.vsv-vrange-vsubset, unfold ntcf-ua-of-NTMap-vdomain*)

show $vsv(\text{ntcf-ua-of } \alpha \mathfrak{F} \text{ } c \text{ } r \text{ } u(\text{NTMap}))$ **by** (*rule ntcf-ua-of-NTMap-vsv*)

fix *d* **assume** *prems*: $d \in_{\circ} \mathfrak{A}(\text{Obj})$

with *category-cat-Set is-functor-axioms assms* **show**

$\text{ntcf-ua-of } \alpha \mathfrak{F} \text{ } c \text{ } r \text{ } u(\text{NTMap})(d) \in_{\circ} \text{cat-Set } \alpha(\text{Arr})$

by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

qed

2.5.3 Commutativity of the universal maps and *hom*-functions

lemma (in *is-functor*) *cf-umap-of-cf-hom-commute*:

assumes *category* α \mathfrak{A}
and *category* α \mathfrak{B}
and $c \in_{\circ} \mathfrak{B}(\text{Obj})$
and $r \in_{\circ} \mathfrak{A}(\text{Obj})$
and $u : c \mapsto_{\mathfrak{B}} \mathfrak{F}(\text{ObjMap})(r)$
and $f : a \mapsto_{\mathfrak{A}} b$

shows

$umap\text{-of } \mathfrak{F} \ c \ r \ u \ b \circ_{A \text{ cat-Set } \alpha} cf\text{-hom } \mathfrak{A} \ [\mathfrak{A}(\text{CId})(r), f]_{\circ} =$
 $cf\text{-hom } \mathfrak{B} \ [\mathfrak{B}(\text{CId})(c), \mathfrak{F}(\text{ArrMap})(f)]_{\circ} \circ_{A \text{ cat-Set } \alpha} umap\text{-of } \mathfrak{F} \ c \ r \ u \ a$
(is $\langle ?uof\text{-}b \circ_{A \text{ cat-Set } \alpha} ?rf = ?cf \circ_{A \text{ cat-Set } \alpha} ?uof\text{-}a \rangle$ **)**

proof-

from *is-functor-axioms category-cat-Set assms(1,2,4-6)* **have** *b-rf*:

$?uof\text{-}b \circ_{A \text{ cat-Set } \alpha} ?rf : \text{Hom } \mathfrak{A} \ r \ a \mapsto_{\text{cat-Set } \alpha} \text{Hom } \mathfrak{B} \ c \ (\mathfrak{F}(\text{ObjMap})(b))$

by

(
cs-concl cs-shallow
cs-intro: *cat-cs-intros cat-op-intros cat-prod-cs-intros*
)

from *is-functor-axioms category-cat-Set assms(1,2,4-6)* **have** *cf-a*:

$?cf \circ_{A \text{ cat-Set } \alpha} ?uof\text{-}a : \text{Hom } \mathfrak{A} \ r \ a \mapsto_{\text{cat-Set } \alpha} \text{Hom } \mathfrak{B} \ c \ (\mathfrak{F}(\text{ObjMap})(b))$

by (*cs-concl cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros*)

show *?thesis*

proof(*rule arr-Set-eqI[of* α])

from *b-rf* **show** *arr-Set-b-rf: arr-Set* α ($?uof\text{-}b \circ_{A \text{ cat-Set } \alpha} ?rf$)

by (*auto dest: cat-Set-is-arrD(1)*)

from *b-rf* **have** *dom-lhs*:

$D_{\circ} ((?uof\text{-}b \circ_{A \text{ cat-Set } \alpha} ?rf)(\text{ArrVal})) = \text{Hom } \mathfrak{A} \ r \ a$

by (*cs-concl cs-shallow cs-simp: cat-cs-simps*)**+**

from *cf-a* **show** *arr-Set-cf-a: arr-Set* α ($?cf \circ_{A \text{ cat-Set } \alpha} ?uof\text{-}a$)

by (*auto dest: cat-Set-is-arrD(1)*)

from *cf-a* **have** *dom-rhs*:

$D_{\circ} ((?cf \circ_{A \text{ cat-Set } \alpha} ?uof\text{-}a)(\text{ArrVal})) = \text{Hom } \mathfrak{A} \ r \ a$

by (*cs-concl cs-shallow cs-simp: cat-cs-simps*)

show ($?uof\text{-}b \circ_{A \text{ cat-Set } \alpha} ?rf)(\text{ArrVal}) = (?cf \circ_{A \text{ cat-Set } \alpha} ?uof\text{-}a)(\text{ArrVal})$

proof(*rule vsu-eqI, unfold dom-lhs dom-rhs in-Hom-iff*)

fix q **assume** $q : r \mapsto_{\mathfrak{A}} a$

with *is-functor-axioms category-cat-Set assms* **show**

$(?uof\text{-}b \circ_{A \text{ cat-Set } \alpha} ?rf)(\text{ArrVal})(q) =$

$(?cf \circ_{A \text{ cat-Set } \alpha} ?uof\text{-}a)(\text{ArrVal})(q)$

by

(
cs-concl cs-shallow
cs-simp: *cat-cs-simps cat-op-simps*
cs-intro: *cat-cs-intros cat-op-intros cat-prod-cs-intros*
)

qed (*use arr-Set-b-rf arr-Set-cf-a in auto*)

qed (*use b-rf cf-a in* $\langle cs\text{-concl } cs\text{-shallow } cs\text{-simp: cat-cs-simps} \rangle$)**+**

qed

lemma *cf-umap-of-cf-hom-unit-commute*:

assumes *category* α \mathfrak{C}

and category $\alpha \mathcal{D}$
and $\mathfrak{F} : \mathcal{C} \mapsto \mapsto_{C\alpha} \mathcal{D}$
and $\mathfrak{G} : \mathcal{D} \mapsto \mapsto_{C\alpha} \mathcal{C}$
and $\eta : cf\text{-id } \mathcal{C} \mapsto_{CF} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathcal{C} \mapsto \mapsto_{C\alpha} \mathcal{C}$
and $g : c' \mapsto_{\mathcal{C}} c$
and $f : d \mapsto_{\mathcal{D}} d'$

shows

$umap\text{-of } \mathfrak{G} \ c' \ (\mathfrak{F}(\mathcal{O}bjMap)(\langle c' \rangle)) \ (\eta(\mathcal{N}TMap)(\langle c' \rangle)) \ d' \circ_{A\text{cat-Set } \alpha}$
 $cf\text{-hom } \mathcal{D} \ [\mathfrak{F}(\mathcal{A}rrMap)(\langle g \rangle), f]_{\circ} =$
 $cf\text{-hom } \mathcal{C} \ [g, \mathfrak{G}(\mathcal{A}rrMap)(\langle f \rangle)]_{\circ} \circ_{A\text{cat-Set } \alpha}$
 $umap\text{-of } \mathfrak{G} \ c \ (\mathfrak{F}(\mathcal{O}bjMap)(\langle c \rangle)) \ (\eta(\mathcal{N}TMap)(\langle c \rangle)) \ d$
(is $\langle ?uof\text{-}c'd' \circ_{A\text{cat-Set } \alpha} ?\mathfrak{F}gf = ?g\mathfrak{G}f \circ_{A\text{cat-Set } \alpha} ?uof\text{-}cd \rangle$)

proof-

interpret $\eta : is\text{-ntcf } \alpha \ \mathcal{C} \ \mathcal{C} \ \langle cf\text{-id } \mathcal{C} \rangle \ \langle \mathfrak{G} \circ_{CF} \mathfrak{F} \rangle \ \eta$ **by** (rule *assms(5)*)

from *assms* **have** $c'd'\text{-}\mathfrak{F}gf : ?uof\text{-}c'd' \circ_{A\text{cat-Set } \alpha} ?\mathfrak{F}gf :$
 $Hom \ \mathcal{D} \ (\mathfrak{F}(\mathcal{O}bjMap)(\langle c \rangle)) \ d \mapsto_{cat\text{-Set } \alpha} Hom \ \mathcal{C} \ c' \ (\mathfrak{G}(\mathcal{O}bjMap)(\langle d' \rangle))$
by
 (

 cs-concl
 cs-simp: *cat-cs-simps*
 cs-intro: *cat-cs-intros cat-op-intros cat-prod-cs-intros*
)

then have *dom-lhs:*

$\mathcal{D}_{\circ} \ ((?uof\text{-}c'd' \circ_{A\text{cat-Set } \alpha} ?\mathfrak{F}gf)(\mathcal{A}rrVal)) = Hom \ \mathcal{D} \ (\mathfrak{F}(\mathcal{O}bjMap)(\langle c \rangle)) \ d$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps*)

from *assms* **have** $g\mathfrak{G}f\text{-}cd : ?g\mathfrak{G}f \circ_{A\text{cat-Set } \alpha} ?uof\text{-}cd :$

$Hom \ \mathcal{D} \ (\mathfrak{F}(\mathcal{O}bjMap)(\langle c \rangle)) \ d \mapsto_{cat\text{-Set } \alpha} Hom \ \mathcal{C} \ c' \ (\mathfrak{G}(\mathcal{O}bjMap)(\langle d' \rangle))$
by

(

 cs-concl
 cs-simp: *cat-cs-simps*
 cs-intro: *cat-cs-intros cat-op-intros cat-prod-cs-intros*
)

then have *dom-rhs:*

$\mathcal{D}_{\circ} \ ((?g\mathfrak{G}f \circ_{A\text{cat-Set } \alpha} ?uof\text{-}cd)(\mathcal{A}rrVal)) = Hom \ \mathcal{D} \ (\mathfrak{F}(\mathcal{O}bjMap)(\langle c \rangle)) \ d$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps*)

show *?thesis*

proof(rule *arr-Set-eqI[of α]*)

from $c'd'\text{-}\mathfrak{F}gf$ **show** $arr\text{-Set-}c'd'\text{-}\mathfrak{F}gf :$

$arr\text{-Set } \alpha \ (\ ?uof\text{-}c'd' \circ_{A\text{cat-Set } \alpha} ?\mathfrak{F}gf)$
by (*auto dest: cat-Set-is-arrD(1)*)

from $g\mathfrak{G}f\text{-}cd$ **show** $arr\text{-Set-}g\mathfrak{G}f\text{-}cd :$

$arr\text{-Set } \alpha \ (\ ?g\mathfrak{G}f \circ_{A\text{cat-Set } \alpha} ?uof\text{-}cd)$
by (*auto dest: cat-Set-is-arrD(1)*)

show

$(?uof\text{-}c'd' \circ_{A\text{cat-Set } \alpha} ?\mathfrak{F}gf)(\mathcal{A}rrVal) =$
 $(?g\mathfrak{G}f \circ_{A\text{cat-Set } \alpha} ?uof\text{-}cd)(\mathcal{A}rrVal)$

proof(rule *vsv-eqI, unfold dom-lhs dom-rhs in-Hom-iff*)

fix h **assume** *prems*: $h : \mathfrak{F}(\mathcal{O}bjMap)(\langle c \rangle) \mapsto_{\mathcal{D}} d$

from $\eta.\text{ntcf-Comp-commute}[OF \text{ assms}(6)]$ *assms* **have** [*cat-cs-simps*]:

$\eta(\mathcal{N}TMap)(\langle c \rangle) \circ_{A\mathcal{C}} g = \mathfrak{G}(\mathcal{A}rrMap)(\langle \mathfrak{F}(\mathcal{A}rrMap)(\langle g \rangle) \rangle) \circ_{A\mathcal{C}} \eta(\mathcal{N}TMap)(\langle c' \rangle)$

by

(

 cs-prems cs-shallow
 cs-simp: *cat-cs-simps cat-op-simps cs-intro: cat-cs-intros*

)
from *assms prems show*
 $(?uof-c'd' \circ_{A \text{ cat-Set } \alpha} ?\mathfrak{F}gf)(\downarrow \text{ArrVal})(\downarrow h) =$
 $(?g\mathfrak{G}f \circ_{A \text{ cat-Set } \alpha} ?uof-cd)(\downarrow \text{ArrVal})(\downarrow h)$
by
 (
 cs-concl
 cs-intro: *cat-cs-intros cat-op-intros cat-prod-cs-intros*
 cs-simp: *cat-cs-simps*
)
qed (*use arr-Set-c'd'-\mathfrak{F}gf arr-Set-g\mathfrak{G}f-cd in auto*)
qed (*use c'd'-\mathfrak{F}gf g\mathfrak{G}f-cd in \langle cs-concl cs-shallow cs-simp: cat-cs-simps \rangle*)+
qed

2.5.4 Universal natural transformation is a natural transformation

lemma (*in is-functor*) *cf-ntcf-ua-of-is-ntcf:*
assumes $r \in_{\circ} \mathfrak{A}(\text{Obj})$
and $u : c \mapsto_{\mathfrak{B}} \mathfrak{F}(\text{ObjMap})(\downarrow r)$
shows *ntcf-ua-of* $\alpha \mathfrak{F} c r u :$
 $\text{Hom}_{O.C\alpha} \mathfrak{A}(r, -) \mapsto_{CF} \text{Hom}_{O.C\alpha} \mathfrak{B}(c, -) \circ_{CF} \mathfrak{F} : \mathfrak{A} \mapsto_{C\alpha} \text{cat-Set } \alpha$
proof(*intro is-ntcfI'*)
let $?ua = \langle \text{ntcf-ua-of } \alpha \mathfrak{F} c r u \rangle$
show *vfsequence* (*ntcf-ua-of* $\alpha \mathfrak{F} c r u$) **unfolding** *ntcf-ua-of-def* **by** *simp*
show *vcard* $?ua = 5_{\mathbb{N}}$ **unfolding** *ntcf-ua-of-def* **by** (*simp add: nat-omega-simps*)
from *assms(1)* **show** $\text{Hom}_{O.C\alpha} \mathfrak{A}(r, -) : \mathfrak{A} \mapsto_{C\alpha} \text{cat-Set } \alpha$
by (*cs-concl cs-shallow cs-intro: cat-cs-intros*)
from *is-functor-axioms assms(2)* **show**
 $\text{Hom}_{O.C\alpha} \mathfrak{B}(c, -) \circ_{CF} \mathfrak{F} : \mathfrak{A} \mapsto_{C\alpha} \text{cat-Set } \alpha$
by (*cs-concl cs-intro: cat-cs-intros*)
from *is-functor-axioms assms* **show** $\mathcal{D}_{\circ} (?ua(\downarrow \text{NTMap})) = \mathfrak{A}(\text{Obj})$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps*)
show $?ua(\downarrow \text{NTMap})(\downarrow a) :$
 $\text{Hom}_{O.C\alpha} \mathfrak{A}(r, -)(\downarrow \text{ObjMap})(\downarrow a) \mapsto_{\text{cat-Set } \alpha} (\text{Hom}_{O.C\alpha} \mathfrak{B}(c, -) \circ_{CF} \mathfrak{F})(\downarrow \text{ObjMap})(\downarrow a)$
if $a \in_{\circ} \mathfrak{A}(\text{Obj})$ **for** a
using *is-functor-axioms assms* **that**
by (*cs-concl cs-simp: cat-cs-simps cat-op-simps cs-intro: cat-cs-intros*)
show $?ua(\downarrow \text{NTMap})(\downarrow b) \circ_{A \text{ cat-Set } \alpha} \text{Hom}_{O.C\alpha} \mathfrak{A}(r, -)(\downarrow \text{ArrMap})(\downarrow f) =$
 $(\text{Hom}_{O.C\alpha} \mathfrak{B}(c, -) \circ_{CF} \mathfrak{F})(\downarrow \text{ArrMap})(\downarrow f) \circ_{A \text{ cat-Set } \alpha} ?ua(\downarrow \text{NTMap})(\downarrow a)$
if $f : a \mapsto_{\mathfrak{A}} b$ **for** $a b f$
using *is-functor-axioms assms* **that**
by
 (
 cs-concl
 cs-simp: *cf-umap-of-cf-hom-commute cat-cs-simps cat-op-simps*
 cs-intro: *cat-cs-intros cat-op-intros*
)
qed (*auto simp: ntcf-ua-of-components cat-cs-simps*)

lemma (*in is-functor*) *cf-ntcf-ua-fo-is-ntcf:*
assumes $r \in_{\circ} \mathfrak{A}(\text{Obj})$ **and** $u : \mathfrak{F}(\text{ObjMap})(\downarrow r) \mapsto_{\mathfrak{B}} c$
shows *ntcf-ua-fo* $\alpha \mathfrak{F} c r u :$
 $\text{Hom}_{O.C\alpha} \mathfrak{A}(-, r) \mapsto_{CF} \text{Hom}_{O.C\alpha} \mathfrak{B}(-, c) \circ_{CF} \text{op-cf } \mathfrak{F} :$
 $\text{op-cat } \mathfrak{A} \mapsto_{C\alpha} \text{cat-Set } \alpha$
proof-
from *assms(2)* **have** $c : c \in_{\circ} \mathfrak{B}(\text{Obj})$ **by** *auto*

```

show ?thesis
by
  (
    rule is-functor.cf-ntcf-ua-of-is-ntcf
    [
      OF is-functor-op,
      unfolded cat-op-simps,
      OF assms(1,2),
      unfolded
      HomDom.cat-op-cat-cf-Hom-snd[ OF assms(1)]
      HomCod.cat-op-cat-cf-Hom-snd[ OF c]
      ntcf-ua-fo-def[symmetric]
    ]
  )
qed

```

2.5.5 Universal natural transformation and universal arrow

The lemmas in this subsection correspond to variants of elements of Proposition 1 in Chapter III-2 in [9].

```

lemma (in is-functor) cf-ntcf-ua-of-is-iso-ntcf:
  assumes universal-arrow-of  $\mathfrak{F} \ c \ r \ u$ 
  shows ntcf-ua-of  $\alpha \ \mathfrak{F} \ c \ r \ u$  :
     $Hom_{O.C\alpha}\mathfrak{A}(r,-) \mapsto_{CF.iso} Hom_{O.C\alpha}\mathfrak{B}(c,-) \circ_{CF} \mathfrak{F} : \mathfrak{A} \mapsto_{C\alpha} cat-Set \ \alpha$ 
proof-

```

```

have  $r : r \in_o \mathfrak{A}(Obj)$ 
and  $u : u : c \mapsto_{\mathfrak{B}} \mathfrak{F}(ObjMap)(r)$ 
and  $bij : \wedge r' \ u'.$ 
  [
     $r' \in_o \mathfrak{A}(Obj);$ 
     $u' : c \mapsto_{\mathfrak{B}} \mathfrak{F}(ObjMap)(r')$ 
  ]  $\implies \exists ! f'. f' : r \mapsto_{\mathfrak{A}} r' \wedge u' = umap-of \ \mathfrak{F} \ c \ r \ u \ r' (ArrVal)(f')$ 
by (auto intro! : universal-arrow-ofD[ OF assms(1)])

```

```

show ?thesis
proof(intro is-iso-ntcfI)
  show ntcf-ua-of  $\alpha \ \mathfrak{F} \ c \ r \ u$  :
     $Hom_{O.C\alpha}\mathfrak{A}(r,-) \mapsto_{CF} Hom_{O.C\alpha}\mathfrak{B}(c,-) \circ_{CF} \mathfrak{F} : \mathfrak{A} \mapsto_{C\alpha} cat-Set \ \alpha$ 
    by (rule cf-ntcf-ua-of-is-ntcf[ OF r u])
  fix  $a$  assume prems:  $a \in_o \mathfrak{A}(Obj)$ 
  from is-functor-axioms prems r u have [simp]:
     $umap-of \ \mathfrak{F} \ c \ r \ u \ a : Hom \ \mathfrak{A} \ r \ a \mapsto_{cat-Set \ \alpha} Hom \ \mathfrak{B} \ c \ (\mathfrak{F}(ObjMap)(a))$ 
    by (cs-concl cs-shallow cs-intro: cat-cs-intros)
  then have dom:  $\mathcal{D}_o (umap-of \ \mathfrak{F} \ c \ r \ u \ a(ArrVal)) = Hom \ \mathfrak{A} \ r \ a$ 
    by (cs-concl cs-simp: cat-cs-simps)
  have  $umap-of \ \mathfrak{F} \ c \ r \ u \ a : Hom \ \mathfrak{A} \ r \ a \mapsto_{is \circ cat-Set \ \alpha} Hom \ \mathfrak{B} \ c \ (\mathfrak{F}(ObjMap)(a))$ 
proof(intro cat-Set-is-iso-arrI, unfold dom)

```

```

show umof-a: v11 ( $umap-of \ \mathfrak{F} \ c \ r \ u \ a(ArrVal)$ )
proof(intro vsv.vsv-valeq-v11I, unfold dom in-Hom-iff)
  fix  $g \ f$  assume prems':
     $g : r \mapsto_{\mathfrak{A}} a$ 
     $f : r \mapsto_{\mathfrak{A}} a$ 
     $umap-of \ \mathfrak{F} \ c \ r \ u \ a(ArrVal)(g) = umap-of \ \mathfrak{F} \ c \ r \ u \ a(ArrVal)(f)$ 
  from is-functor-axioms r u prems'(1) have  $\mathfrak{F}g$ :
     $\mathfrak{F}(ArrMap)(g) \circ_{A\mathfrak{B}} u : c \mapsto_{\mathfrak{B}} \mathfrak{F}(ObjMap)(a)$ 

```

by (*cs-concl cs-shallow cs-intro: cat-cs-intros*)
 from *bij*[*OF* *prems* $\mathfrak{F}g$] have *unique*:
 [[
 $f' : r \mapsto_{\mathfrak{A}} a$;
 $\mathfrak{F}(\text{ArrMap})(\downarrow g) \circ_{A\mathfrak{B}} u = \text{umap-of } \mathfrak{F} \ c \ r \ u \ a(\text{ArrVal})(\downarrow f')$
]] $\implies g = f'$
 for f' by (*metis* *prems'*(1) *u* *umap-of-ArrVal-app*)
 from *is-functor-axioms* *prems'*(1,2) *u* have $\mathfrak{F}g\text{-}u$:
 $\mathfrak{F}(\text{ArrMap})(\downarrow g) \circ_{A\mathfrak{B}} u = \text{umap-of } \mathfrak{F} \ c \ r \ u \ a(\text{ArrVal})(\downarrow f)$
 by (*cs-concl cs-simp: prems'*(3)[*symmetric*] *cat-cs-simps*)
 show $g = f$ by (*rule* *unique*[*OF* *prems'*(2) $\mathfrak{F}g\text{-}u$])
 qed (*auto simp: cat-cs-simps cat-cs-intros*)

interpret *umof-a: v11* $\langle \text{umap-of } \mathfrak{F} \ c \ r \ u \ a(\text{ArrVal}) \rangle$ by (*rule* *umof-a*)

show $\mathcal{R}_\circ (\text{umap-of } \mathfrak{F} \ c \ r \ u \ a(\text{ArrVal})) = \text{Hom } \mathfrak{B} \ c \ (\mathfrak{F}(\text{ObjMap})(\downarrow a))$
 proof(*intro vsubset-antisym*)
 from *u* show $\mathcal{R}_\circ (\text{umap-of } \mathfrak{F} \ c \ r \ u \ a(\text{ArrVal})) \subseteq_\circ \text{Hom } \mathfrak{B} \ c \ (\mathfrak{F}(\text{ObjMap})(\downarrow a))$
 by (*rule* *umap-of-ArrVal-vrange*)
 show $\text{Hom } \mathfrak{B} \ c \ (\mathfrak{F}(\text{ObjMap})(\downarrow a)) \subseteq_\circ \mathcal{R}_\circ (\text{umap-of } \mathfrak{F} \ c \ r \ u \ a(\text{ArrVal}))$
 proof(*rule vsubsetI, unfold in-Hom-iff*)
 fix f assume *prems'*: $f : c \mapsto_{\mathfrak{B}} \mathfrak{F}(\text{ObjMap})(\downarrow a)$
 from *bij*[*OF* *prems* *prems'*] obtain f'
 where $f' : f' : r \mapsto_{\mathfrak{A}} a$
 and *f-def*: $f = \text{umap-of } \mathfrak{F} \ c \ r \ u \ a(\text{ArrVal})(\downarrow f')$
 by *auto*
 from *is-functor-axioms* *prems* *prems'* *u* f' have
 $f' \in_\circ \mathcal{D}_\circ (\text{umap-of } \mathfrak{F} \ c \ r \ u \ a(\text{ArrVal}))$
 by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)
 from *this* show $f \in_\circ \mathcal{R}_\circ (\text{umap-of } \mathfrak{F} \ c \ r \ u \ a(\text{ArrVal}))$
 unfolding *f-def* by (*rule* *umof-a.vsv-vimageI2*)
 qed

qed

qed *simp-all*

from *is-functor-axioms* *prems* *r* *u* *this* show
 $\text{ntcf-ua-of } \alpha \ \mathfrak{F} \ c \ r \ u(\text{NTMap})(\downarrow a) :$
 $\text{Hom}_{O.C\alpha} \mathfrak{A}(r, -)(\text{ObjMap})(\downarrow a) \mapsto_{\text{iso } \text{cat-Set } \alpha}$
 $(\text{Hom}_{O.C\alpha} \mathfrak{B}(c, -) \circ_{CF} \mathfrak{F})(\text{ObjMap})(\downarrow a)$
 by
 (
 cs-concl
 cs-simp: cat-cs-simps cat-op-simps
 cs-intro: cat-cs-intros cat-op-intros
)
 qed

qed

lemmas [*cat-cs-intros*] = *is-functor.cf-ntcf-ua-of-is-iso-ntcf*

lemma (in *is-functor*) *cf-ntcf-ua-fo-is-iso-ntcf*:
 assumes *universal-arrow-fo* $\mathfrak{F} \ c \ r \ u$
 shows *ntcf-ua-fo* $\alpha \ \mathfrak{F} \ c \ r \ u :$
 $\text{Hom}_{O.C\alpha} \mathfrak{A}(-, r) \mapsto_{CF.\text{iso}} \text{Hom}_{O.C\alpha} \mathfrak{B}(-, c) \circ_{CF} \text{op-cf } \mathfrak{F} :$
 $\text{op-cat } \mathfrak{A} \mapsto_{C\alpha} \text{cat-Set } \alpha$

proof-

from *universal-arrow-foD*[*OF assms*] **have** $r: r \in_0 \mathfrak{A}(\text{Obj})$ **and** $c: c \in_0 \mathfrak{B}(\text{Obj})$

by *auto*

show *?thesis*

by

(
 (
rule is-functor.cf-ntcf-ua-of-is-iso-ntcf
 [
OF is-functor-op,
unfolded cat-op-simps,
OF assms,
unfolded
HomDom.cat-op-cat-cf-Hom-snd[*OF r*]
HomCod.cat-op-cat-cf-Hom-snd[*OF c*]
ntcf-ua-fo-def[*symmetric*]
]
)
)

qed

lemmas [*cat-cs-intros*] = *is-functor.cf-ntcf-ua-fo-is-iso-ntcf*

lemma (**in** *is-functor*) *cf-ua-of-if-ntcf-ua-of-is-iso-ntcf*:

assumes $r \in_0 \mathfrak{A}(\text{Obj})$

and $u: c \mapsto_{\mathfrak{B}} \mathfrak{F}(\text{ObjMap})(r)$

and *ntcf-ua-of* $\alpha \mathfrak{F} c r u$:

$\text{Hom}_{O.C\alpha} \mathfrak{A}(r, -) \mapsto_{CF.iso} \text{Hom}_{O.C\alpha} \mathfrak{B}(c, -) \circ_{CF} \mathfrak{F} : \mathfrak{A} \mapsto_{CF} C\alpha \text{ cat-Set } \alpha$

shows *universal-arrow-of* $\mathfrak{F} c r u$

proof(*rule universal-arrow-ofI*)

interpret *ua-of-u: is-iso-ntcf*

α

\mathfrak{A}

$\langle \text{cat-Set } \alpha \rangle$

$\langle \text{Hom}_{O.C\alpha} \mathfrak{A}(r, -) \rangle$

$\langle \text{Hom}_{O.C\alpha} \mathfrak{B}(c, -) \circ_{CF} \mathfrak{F} \rangle$

$\langle \text{ntcf-ua-of } \alpha \mathfrak{F} c r u \rangle$

by (*rule assms*(3))

fix $r' u'$ **assume** *prems*: $r' \in_0 \mathfrak{A}(\text{Obj})$ $u': c \mapsto_{\mathfrak{B}} \mathfrak{F}(\text{ObjMap})(r')$

have *ntcf-ua-of* $\alpha \mathfrak{F} c r u (\text{NTMap})(r')$:

$\text{Hom}_{O.C\alpha} \mathfrak{A}(r, -) (\text{ObjMap})(r') \mapsto_{iso} \text{cat-Set } \alpha$

$(\text{Hom}_{O.C\alpha} \mathfrak{B}(c, -) \circ_{CF} \mathfrak{F}) (\text{ObjMap})(r')$

by (*rule is-iso-ntcf.iso-ntcf-is-iso-arr*[*OF assms*(3) *prems*(1)])

from *this is-functor-axioms assms*(1-2) *prems* **have** *uof-r'*:

$\text{umap-of } \mathfrak{F} c r u r' : \text{Hom } \mathfrak{A} r r' \mapsto_{iso} \text{cat-Set } \alpha \text{ Hom } \mathfrak{B} c (\mathfrak{F}(\text{ObjMap})(r'))$

by (*cs-prems cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-op-intros*)

note $\text{uof-r}' = \text{cat-Set-is-iso-arrD}$ [*OF uof-r'*]

interpret *uof-r'*: $v11 \langle \text{umap-of } \mathfrak{F} c r u r' (\text{ArrVal}) \rangle$ **by** (*rule uof-r'*(2))

from

$\text{uof-r}'.v11-vrange-ex1-eq$ [

THEN iffD1, unfolded uof-r'(3,4) *in-Hom-iff, OF prems*(2)

]

show $\exists ! f'. f' : r \mapsto_{\mathfrak{A}} r' \wedge u' = \text{umap-of } \mathfrak{F} c r u r' (\text{ArrVal})(f')$

by *metis*

qed (*intro assms*)+

lemma (**in** *is-functor*) *cf-ua-fo-if-ntcf-ua-fo-is-iso-ntcf*:

assumes $r \in_0 \mathfrak{A}(\text{Obj})$

and $u: \mathfrak{F}(\text{ObjMap})(r) \mapsto_{\mathfrak{B}} c$

and *ntcf-ua-fo* $\alpha \mathfrak{F} c r u$:

$Hom_{O.C\alpha}\mathfrak{A}(-, r) \mapsto_{CF.iso} Hom_{O.C\alpha}\mathfrak{B}(-, c) \circ_{CF} op\text{-}cf \mathfrak{F} :$
 $op\text{-}cat \mathfrak{A} \mapsto_{C\alpha} cat\text{-}Set \alpha$

shows *universal-arrow-fo* $\mathfrak{F} \ c \ r \ u$

proof-

from *assms(2)* **have** $c : c \in_o \mathfrak{B}(\text{Obj})$ **by** *auto*

show *?thesis*

by

(
 rule *is-functor.cf-ua-of-if-ntcf-ua-of-is-iso-ntcf*
 [
OF is-functor-op,
unfolded cat-op-simps,
OF assms(1,2),
unfolded
HomDom.cat-op-cat-cf-Hom-snd[OF assms(1)]
HomCod.cat-op-cat-cf-Hom-snd[OF c]
ntcf-ua-fo-def[symmetric],
OF assms(3)
]
)

qed

lemma (in *is-functor*) *cf-universal-arrow-of-if-is-iso-ntcf*:

assumes $r \in_o \mathfrak{A}(\text{Obj})$

and $c \in_o \mathfrak{B}(\text{Obj})$

and $\varphi :$

$Hom_{O.C\alpha}\mathfrak{A}(r, -) \mapsto_{CF.iso} Hom_{O.C\alpha}\mathfrak{B}(c, -) \circ_{CF} \mathfrak{F} : \mathfrak{A} \mapsto_{C\alpha} cat\text{-}Set \alpha$

shows *universal-arrow-of* $\mathfrak{F} \ c \ r \ (\varphi(\text{NTMap})(\text{ArrVal})(\mathfrak{A}(\text{CIId})(r)))$

(is *universal-arrow-of* $\mathfrak{F} \ c \ r \ ?u$)

proof-

interpret $\varphi : is\text{-}iso\text{-}ntcf$

$\alpha \mathfrak{A} \langle cat\text{-}Set \alpha \rangle \langle Hom_{O.C\alpha}\mathfrak{A}(r, -) \rangle \langle Hom_{O.C\alpha}\mathfrak{B}(c, -) \circ_{CF} \mathfrak{F} \rangle \varphi$

by (rule *assms(3)*)

show *?thesis*

proof(*intro universal-arrow-ofI assms*)

from *assms(1,2)* **show** $u : c \mapsto_{\mathfrak{B}} \mathfrak{F}(\text{ObjMap})(r)$

by

(
cs-concl cs-shallow
cs-simp: cat-cs-simps cat-op-simps cs-intro: cat-cs-intros
)

fix $r' \ u'$ **assume** *prems*: $r' \in_o \mathfrak{A}(\text{Obj}) \ u' : c \mapsto_{\mathfrak{B}} \mathfrak{F}(\text{ObjMap})(r')$

have $\varphi r' \text{-ArrVal-app[symmetric, cat-cs-simps]}$:

$\varphi(\text{NTMap})(r')(\text{ArrVal})(f') =$
 $\mathfrak{F}(\text{ArrMap})(f') \circ_{A\mathfrak{B}} \varphi(\text{NTMap})(r)(\text{ArrVal})(\mathfrak{A}(\text{CIId})(r))$

if $f' : r \mapsto_{\mathfrak{A}} r'$ **for** f'

proof-

have $\varphi(\text{NTMap})(r') \circ_{A \text{ cat-Set } \alpha} Hom_{O.C\alpha}\mathfrak{A}(r, -)(\text{ArrMap})(f') =$
 $(Hom_{O.C\alpha}\mathfrak{B}(c, -) \circ_{CF} \mathfrak{F})(\text{ArrMap})(f') \circ_{A \text{ cat-Set } \alpha} \varphi(\text{NTMap})(r)$
using that by (*intro* $\varphi.ntcf\text{-Comp-commute}$)

then have

$\varphi(\text{NTMap})(r') \circ_{A \text{ cat-Set } \alpha} cf\text{-hom } \mathfrak{A} [\mathfrak{A}(\text{CIId})(r), f'] \circ =$
 $cf\text{-hom } \mathfrak{B} [\mathfrak{B}(\text{CIId})(c), \mathfrak{F}(\text{ArrMap})(f')] \circ_{A \text{ cat-Set } \alpha} \varphi(\text{NTMap})(r)$

using *assms(1,2)* **that prems**

by

(
cs-prems **cs-shallow**
cs-simp: *cat-cs-simps cat-op-simps* **cs-intro**: *cat-cs-intros*
)
then have
 $(\varphi(\downarrow NTMap)(\downarrow r')) \circ_{A \text{ cat-Set } \alpha}$
 $cf\text{-hom } \mathfrak{A} [\mathfrak{A}(\downarrow CId)(\downarrow r'), f'] \circ (\downarrow ArrVal)(\downarrow \mathfrak{A}(\downarrow CId)(\downarrow r)) =$
 $(cf\text{-hom } \mathfrak{B} [\mathfrak{B}(\downarrow CId)(\downarrow c), \mathfrak{F}(\downarrow ArrMap)(\downarrow f')]) \circ_{A \text{ cat-Set } \alpha}$
 $\varphi(\downarrow NTMap)(\downarrow r')(\downarrow ArrVal)(\downarrow \mathfrak{A}(\downarrow CId)(\downarrow r))$
by *simp*
from *this assms(1,2) u* that **show** *?thesis*
by
 (
cs-prems **cs-shallow**
cs-simp: *cat-cs-simps cat-op-simps*
cs-intro: *cat-cs-intros cat-op-intros cat-prod-cs-intros*
)
qed

show $\exists ! f'. f' : r \mapsto_{\mathfrak{A}} r' \wedge u' = \text{umap-of } \mathfrak{F} \text{ } c \text{ } r \text{ } ?u \text{ } r'(\downarrow ArrVal)(\downarrow f')$
proof(*intro exI conjI; (elim conjE)?*)
from *assms prems* **show**
 $(\varphi(\downarrow NTMap)(\downarrow r'))^{-1}_{C \text{ cat-Set } \alpha}(\downarrow ArrVal)(\downarrow u') : r \mapsto_{\mathfrak{A}} r'$
by
 (
cs-concl
cs-simp: *cat-cs-simps cat-op-simps*
cs-intro: *cat-cs-intros cat-arrow-cs-intros*
)
with *assms(1,2) prems* **show** $u' =$
 $\text{umap-of } \mathfrak{F} \text{ } c \text{ } r \text{ } ?u \text{ } r'(\downarrow ArrVal)((\varphi(\downarrow NTMap)(\downarrow r'))^{-1}_{C \text{ cat-Set } \alpha}(\downarrow ArrVal)(\downarrow u'))$
by
 (
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps cat-op-simps*
cs-intro: *cat-arrow-cs-intros cat-cs-intros cat-op-intros*
)
fix f' **assume** *prems'*:
 $f' : r \mapsto_{\mathfrak{A}} r'$
 $u' = \text{umap-of } \mathfrak{F} \text{ } c \text{ } r \text{ } (\varphi(\downarrow NTMap)(\downarrow r')(\downarrow ArrVal)(\downarrow \mathfrak{A}(\downarrow CId)(\downarrow r))) \text{ } r'(\downarrow ArrVal)(\downarrow f')$
from *prems'(2,1) assms(1,2)* **have** $u'\text{-def}$:
 $u' = \mathfrak{F}(\downarrow ArrMap)(\downarrow f') \circ_{A \mathfrak{B}} \varphi(\downarrow NTMap)(\downarrow r')(\downarrow ArrVal)(\downarrow \mathfrak{A}(\downarrow CId)(\downarrow r))$
by
 (
cs-prems **cs-shallow**
cs-simp: *cat-cs-simps cat-op-simps*
cs-intro: *cat-cs-intros cat-op-intros*
)
from *prems'* **show** $f' = (\varphi(\downarrow NTMap)(\downarrow r'))^{-1}_{C \text{ cat-Set } \alpha}(\downarrow ArrVal)(\downarrow u')$
unfolding $u'\text{-def } \varphi r'\text{-ArrVal-app}[OF \text{ } prems'(1)]$
by
 (
cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-arrow-cs-intros cat-cs-intros cat-op-intros*
)
qed

qed

qed

lemma (in *is-functor*) *cf-universal-arrow-fo-if-is-iso-ntcf*:

assumes $r \in_o \mathfrak{A}(\text{Obj})$

and $c \in_o \mathfrak{B}(\text{Obj})$

and $\varphi :$

$\text{Hom}_{O.C\alpha} \mathfrak{A}(-, r) \mapsto_{CF.iso} \text{Hom}_{O.C\alpha} \mathfrak{B}(-, c) \circ_{CF} \text{op-cf } \mathfrak{F} :$

$\text{op-cat } \mathfrak{A} \mapsto_{C\alpha} \text{cat-Set } \alpha$

shows *universal-arrow-fo* $\mathfrak{F} \ c \ r \ (\varphi(\text{NTMap})(r)(\text{ArrVal})(\mathfrak{A}(\text{CIId})(r)))$

by

(
 rule *is-functor.cf-universal-arrow-of-if-is-iso-ntcf*
 [
 OF is-functor-op,
 unfolded cat-op-simps,
 OF assms(1,2),
 unfolded
 HomDom.cat-op-cat-cf-Hom-snd[OF assms(1)]
 HomCod.cat-op-cat-cf-Hom-snd[OF assms(2)]
 ntcf-ua-fo-def[symmetric],
 OF assms(3)
]
)

3 Limits and colimits

3.1 Background

named-theorems *cat-lim-cs-simps*

named-theorems *cat-lim-cs-intros*

3.2 Limit and colimit

3.2.1 Definition and elementary properties

The concept of a limit is introduced in Chapter III-4 in [9]; the concept of a colimit is introduced in Chapter III-3 in [9].

locale *is-cat-limit* = *is-cat-cone* α r \mathfrak{J} \mathfrak{C} \mathfrak{F} u **for** α \mathfrak{J} \mathfrak{C} \mathfrak{F} r u +
assumes *cat-lim-ua-fo*: $\bigwedge u' r'. u' : r' <_{CF.cone} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C} \implies$
 $\exists ! f'. f' : r' \mapsto_{\mathfrak{C}} r \wedge u' = u \cdot_{NTCF} ntcf-const \mathfrak{J} \mathfrak{C} f'$

syntax *-is-cat-limit* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
 $(\langle \langle - : / - <_{CF.lim} - : / - \mapsto_{C1} - \rangle \rangle [51, 51, 51, 51, 51] 51)$

syntax-consts *-is-cat-limit* $\hat{=}$ *is-cat-limit*

translations $u : r <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C} \hat{=}$
 $CONST$ *is-cat-limit* α \mathfrak{J} \mathfrak{C} \mathfrak{F} r u

locale *is-cat-colimit* = *is-cat-cocone* α r \mathfrak{J} \mathfrak{C} \mathfrak{F} u **for** α \mathfrak{J} \mathfrak{C} \mathfrak{F} r u +
assumes *cat-colim-ua-of*: $\bigwedge u' r'. u' : \mathfrak{F} >_{CF.cocone} r' : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C} \implies$
 $\exists ! f'. f' : r \mapsto_{\mathfrak{C}} r' \wedge u' = ntcf-const \mathfrak{J} \mathfrak{C} f' \cdot_{NTCF} u$

syntax *-is-cat-colimit* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
 $(\langle \langle - : / - >_{CF.colim} - : / - \mapsto_{C1} - \rangle \rangle [51, 51, 51, 51, 51] 51)$

syntax-consts *-is-cat-colimit* $\hat{=}$ *is-cat-colimit*

translations $u : \mathfrak{F} >_{CF.colim} r : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C} \hat{=}$
 $CONST$ *is-cat-colimit* α \mathfrak{J} \mathfrak{C} \mathfrak{F} r u

Rules.

lemma (in *is-cat-limit*) *is-cat-limit-axioms*[*cat-lim-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $r' = r$ **and** $\mathfrak{J}' = \mathfrak{J}$ **and** $\mathfrak{C}' = \mathfrak{C}$ **and** $\mathfrak{F}' = \mathfrak{F}$
shows $u : r' <_{CF.lim} \mathfrak{F}' : \mathfrak{J}' \mapsto_{C\alpha'} \mathfrak{C}'$
unfolding *assms* **by** (rule *is-cat-limit-axioms*)

mk-ide rf *is-cat-limit-def*[*unfolded is-cat-limit-axioms-def*]

|*intro is-cat-limitI*]
|*dest is-cat-limitD*[*dest*]
|*elim is-cat-limitE*[*elim*]

lemmas [*cat-lim-cs-intros*] = *is-cat-limitD*(1)

lemma (in *is-cat-colimit*) *is-cat-colimit-axioms*[*cat-lim-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $r' = r$ **and** $\mathfrak{J}' = \mathfrak{J}$ **and** $\mathfrak{C}' = \mathfrak{C}$ **and** $\mathfrak{F}' = \mathfrak{F}$
shows $u : \mathfrak{F}' >_{CF.colim} r' : \mathfrak{J}' \mapsto_{C\alpha'} \mathfrak{C}'$
unfolding *assms* **by** (rule *is-cat-colimit-axioms*)

mk-ide rf *is-cat-colimit-def*[*unfolded is-cat-colimit-axioms-def*]

|*intro is-cat-colimitI*]
|*dest is-cat-colimitD*[*dest*]
|*elim is-cat-colimitE*[*elim*]

lemmas [*cat-lim-cs-intros*] = *is-cat-colimitD*(1)

Limits, colimits and universal arrows.

lemma (in *is-cat-limit*) *cat-lim-is-universal-arrow-fo*:

universal-arrow-fo ($\Delta_{CF} \alpha \mathfrak{J} \mathfrak{C}$) (*cf-map* \mathfrak{F}) *r* (*ntcf-arrow* *u*)

proof(*intro is-functor.universal-arrow-foI*)

define β **where** $\beta = \alpha + \omega$

have $\beta: \mathcal{Z} \beta$ **and** $\alpha\beta: \alpha \in_{\circ} \beta$

by (*simp-all add: β -def \mathcal{Z} -Limit- $\alpha\omega$ \mathcal{Z} - ω - $\alpha\omega$ \mathcal{Z} -def \mathcal{Z} - α - $\alpha\omega$*)

then interpret $\beta: \mathcal{Z} \beta$ **by** *simp*

show $\Delta_{CF} \alpha \mathfrak{J} \mathfrak{C} : \mathfrak{C} \mapsto_{CF} \text{cat-FUNCT } \alpha \mathfrak{J} \mathfrak{C}$

by

(
 intro
 $\beta \alpha\beta$
 cf-diagonal-is-functor
 NTDom.HomDom.category-axioms
 NTDom.HomCod.category-axioms
)

show $r \in_{\circ} \mathfrak{C}(\text{Obj})$ **by** (*intro cat-cone-obj*)

then show *ntcf-arrow* *u* : $\Delta_{CF} \alpha \mathfrak{J} \mathfrak{C}(\text{ObjMap})(\text{!}r) \mapsto_{\text{cat-FUNCT } \alpha \mathfrak{J} \mathfrak{C}} \text{cf-map } \mathfrak{F}$

by

(
 cs-concl cs-shallow
 cs-simp: *cat-cs-simps cs-intro: cat-cs-intros cat-FUNCT-cs-intros*
)

fix $r' u'$ **assume** *prems*:

$r' \in_{\circ} \mathfrak{C}(\text{Obj})$ $u' : \Delta_{CF} \alpha \mathfrak{J} \mathfrak{C}(\text{ObjMap})(\text{!}r') \mapsto_{\text{cat-FUNCT } \alpha \mathfrak{J} \mathfrak{C}} \text{cf-map } \mathfrak{F}$

from *prems(1)* **have** [*cat-cs-simps*]:

cf-of-cf-map $\mathfrak{J} \mathfrak{C} (\text{cf-map } \mathfrak{F}) = \mathfrak{F}$

cf-of-cf-map $\mathfrak{J} \mathfrak{C} (\text{cf-map } (\text{cf-const } \mathfrak{J} \mathfrak{C} r')) = \text{cf-const } \mathfrak{J} \mathfrak{C} r'$

by (*cs-concl cs-simp: cat-FUNCT-cs-simps cs-intro: cat-cs-intros*)+

from *prems(2,1)* **have**

$u' : \text{cf-map } (\text{cf-const } \mathfrak{J} \mathfrak{C} r') \mapsto_{\text{cat-FUNCT } \alpha \mathfrak{J} \mathfrak{C}} \text{cf-map } \mathfrak{F}$

by (*cs-prems cs-shallow cs-simp: cat-cs-simps*)

note $u'[\text{unfolded cat-cs-simps}] = \text{cat-FUNCT-is-arrD}[OF \text{ this}]$

from *cat-lim-ua-fo[OF is-cat-coneI[OF $u'(1)$ *prems(1)*]]* **obtain** *f*

where $f: r' \mapsto_{\mathfrak{C}} r$

and [*symmetric, cat-cs-simps*]:

ntcf-of-ntcf-arrow $\mathfrak{J} \mathfrak{C} u' = u \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{C} f$

and *f-unique*:

[[
 $f' : r' \mapsto_{\mathfrak{C}} r$;
 ntcf-of-ntcf-arrow $\mathfrak{J} \mathfrak{C} u' = u \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{C} f'$
]] $\implies f' = f$

for f'

by *metis*

show $\exists! f'$.

$f' : r' \mapsto_{\mathfrak{C}} r \wedge$

$u' = \text{umap-fo } (\Delta_{CF} \alpha \mathfrak{J} \mathfrak{C}) (\text{cf-map } \mathfrak{F}) r (\text{ntcf-arrow } u) r'(\text{!ArrVal})(\text{!}f')$

proof(*intro ex1I conjI; (elim conjE)?*)

show $f : r' \mapsto_{\mathfrak{C}} r$ **by** (*rule f*)

with $\alpha\beta$ *cat-cone-obj* **show** *u'-def*:

$u' = \text{umap-fo } (\Delta_{CF} \alpha \mathfrak{J} \mathfrak{C}) (\text{cf-map } \mathfrak{F}) r (\text{ntcf-arrow } u) r'(\text{!ArrVal})(\text{!}f)$

by
 (

 cs-concl

 cs-simp: $u'(2)[\textit{symmetric}] \textit{cat-cs-simps} \textit{cat-FUNCT-cs-simps}$

 cs-intro: $\textit{cat-cs-intros} \textit{cat-FUNCT-cs-intros}$

)

fix f' **assume** \textit{prems}' :

$f' : r' \mapsto_{\mathcal{C}} r$

 $u' = \textit{umap-fo} (\Delta_{CF} \alpha \mathfrak{J} \mathcal{C}) (\textit{cf-map} \mathfrak{F}) r (\textit{ntcf-arrow} u) r' (\textit{ArrVal}) (f')$

from $\textit{prems}'(2) \alpha\beta f \textit{prems}' \textit{cat-cone-obj}$ **have** $u'\textit{-def}'$:

$u' = \textit{ntcf-arrow} (u \cdot_{NTCF} \textit{ntcf-const} \mathfrak{J} \mathcal{C} f')$

by
 (

cs-prems

 cs-simp: $\textit{cat-cs-simps} \textit{cat-FUNCT-cs-simps}$

 cs-intro: $\textit{cat-cs-intros} \textit{cat-FUNCT-cs-intros}$

)

from $\textit{prems}'(1)$ **have** $\textit{ntcf-of-ntcf-arrow} \mathfrak{J} \mathcal{C} u' = u \cdot_{NTCF} \textit{ntcf-const} \mathfrak{J} \mathcal{C} f'$

by
 (

cs-concl

 cs-simp: $\textit{cat-FUNCT-cs-simps} u'\textit{-def}'$ **cs-intro:** $\textit{cat-cs-intros}$

)

from $f\textit{-unique}[OF \textit{prems}'(1) \textit{this}]$ **show** $f' = f$.

qed

qed

lemma (in $\textit{is-cat-cone}$) $\textit{cat-cone-is-cat-limit}$:

assumes $\textit{universal-arrow-fo} (\Delta_{CF} \alpha \mathfrak{J} \mathcal{C}) (\textit{cf-map} \mathfrak{F}) c (\textit{ntcf-arrow} \mathfrak{N})$

shows $\mathfrak{N} : c <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathcal{C}$

proof-

define β **where** $\beta = \alpha + \omega$

have $\beta : \mathcal{Z} \beta$ **and** $\alpha\beta : \alpha \in_{\circ} \beta$

by ($\textit{simp-all} \textit{add} : \beta\textit{-def} \mathcal{Z}\textit{-Limit-}\alpha\omega \mathcal{Z}\textit{-}\omega\textit{-}\alpha\omega \mathcal{Z}\textit{-def} \mathcal{Z}\textit{-}\alpha\textit{-}\alpha\omega$)

then interpret $\beta : \mathcal{Z} \beta$ **by** \textit{simp}

show $?thesis$

proof($\textit{intro} \textit{is-cat-limitI} \textit{is-cat-cone-axioms}$)

fix $u' c'$ **assume** $\textit{prems} : u' : c' <_{CF.cone} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathcal{C}$

interpret $u' : \textit{is-cat-cone} \alpha c' \mathfrak{J} \mathcal{C} \mathfrak{F} u'$ **by** ($\textit{rule} \textit{prems}$)

from $u'.\textit{cat-cone-obj}$ **have** $u'\textit{-is-arr}$:

$\textit{ntcf-arrow} u' : \Delta_{CF} \alpha \mathfrak{J} \mathcal{C} (\textit{ObjMap}) (c') \mapsto_{\textit{cat-FUNCT} \alpha \mathfrak{J} \mathcal{C}} \textit{cf-map} \mathfrak{F}$

by

(

cs-concl **cs-shallow**

cs-simp: $\textit{cat-cs-simps}$ **cs-intro:** $\textit{cat-cs-intros} \textit{cat-FUNCT-cs-intros}$

)

from $\textit{is-functor.universal-arrow-foD}(3)$

[

OF

$\textit{cf-diagonal-is-functor}$ [

$OF \beta \alpha\beta \textit{NTDom.HomDom.category-axioms} \textit{NTDom.HomCod.category-axioms}$

```

    ]
    assms
    u'.cat-cone-obj
    u'-is-arr
  ]
obtain  $f$  where  $f: c' \mapsto_{\mathfrak{C}} c$ 
and  $u'\text{-def}'$ :  $\text{ntcf-arrow } u' =$ 
   $\text{umap-fo } (\Delta_{CF} \alpha \mathfrak{J} \mathfrak{C}) \text{ (cf-map } \mathfrak{F}) \text{ } c \text{ (ntcf-arrow } \mathfrak{N}) \text{ } c' \text{ (ArrVal)} \text{ (} f \text{)}$ 
and  $f'\text{-unique}$ :
  [
     $f': c' \mapsto_{\mathfrak{C}} c$ ;
     $\text{ntcf-arrow } u' =$ 
     $\text{umap-fo } (\Delta_{CF} \alpha \mathfrak{J} \mathfrak{C}) \text{ (cf-map } \mathfrak{F}) \text{ } c \text{ (ntcf-arrow } \mathfrak{N}) \text{ } c' \text{ (ArrVal)} \text{ (} f' \text{)}$ 
  ]  $\implies f' = f$ 
for  $f'$ 
by metis

```

```

from  $u'\text{-def}' \alpha \beta \text{ } f \text{ cat-cone-obj}$  have  $u'\text{-def}$ :
   $u' = \mathfrak{N} \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{C} \text{ } f$ 
by
  (
    cs-prems
    cs-simp: cat-cs-simps cat-FUNCT-cs-simps
    cs-intro: cat-cs-intros cat-FUNCT-cs-intros
  )

```

```

show  $\exists ! f'. f': c' \mapsto_{\mathfrak{C}} c \wedge u' = \mathfrak{N} \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{C} \text{ } f'$ 
proof(intro exI conjI; (elim conjE)?, (rule f)?, (rule u'-def)?)
fix  $f''$  assume  $\text{prems}'$ :
   $f'': c' \mapsto_{\mathfrak{C}} c \wedge u' = \mathfrak{N} \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{C} \text{ } f''$ 
from  $\alpha \beta \text{ } \text{prems}'$  have
   $\text{ntcf-arrow } u' =$ 
   $\text{umap-fo } (\Delta_{CF} \alpha \mathfrak{J} \mathfrak{C}) \text{ (cf-map } \mathfrak{F}) \text{ } c \text{ (ntcf-arrow } \mathfrak{N}) \text{ } c' \text{ (ArrVal)} \text{ (} f'' \text{)}$ 
by
  (
    cs-concl
    cs-simp: cat-cs-simps cat-FUNCT-cs-simps
    cs-intro: cat-cs-intros cat-FUNCT-cs-intros
  )
from  $f'\text{-unique}$ [OF prems'(1) this] show  $f'' = f$ .
qed

```

qed

qed

```

lemma (in is-cat-colimit) cat-colim-is-universal-arrow-of:
  universal-arrow-of  $(\Delta_{CF} \alpha \mathfrak{J} \mathfrak{C}) \text{ (cf-map } \mathfrak{F}) \text{ } r \text{ (ntcf-arrow } u)$ 
proof(intro is-functor.universal-arrow-ofI)

```

```

define  $\beta$  where  $\beta = \alpha + \omega$ 
have  $\beta: \mathcal{Z} \beta$  and  $\alpha\beta: \alpha \in_{\circ} \beta$ 
by (simp-all add: beta-def Z-Limit-omega Z-omega-alpha Z-def Z-alpha-omega)
then interpret  $\beta: \mathcal{Z} \beta$  by simp

```

```

show  $\Delta_{CF} \alpha \mathfrak{J} \mathfrak{C} : \mathfrak{C} \mapsto \mapsto_{C\beta} \text{cat-FUNCT } \alpha \mathfrak{J} \mathfrak{C}$ 
by
  (

```

```

  intro
    β αβ
    cf-diagonal-is-functor
    NTDom.HomDom.category-axioms
    NTDom.HomCod.category-axioms
  )

```

show $r \in_{\circ} \mathfrak{C}(\text{Obj})$ **by** (*intro cat-cocone-obj*)

```

then show ntcf-arrow  $u : \text{cf-map } \mathfrak{F} \mapsto_{\text{cat-FUNCT}} \alpha \mathfrak{J} \mathfrak{C} \Delta_{CF} \alpha \mathfrak{J} \mathfrak{C}(\text{ObjMap})(r)$ 
by
  (
    cs-concl cs-shallow
    cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-FUNCT-cs-intros
  )

```

fix $r' u'$ **assume** *prems*:

```

   $r' \in_{\circ} \mathfrak{C}(\text{Obj})$   $u' : \text{cf-map } \mathfrak{F} \mapsto_{\text{cat-FUNCT}} \alpha \mathfrak{J} \mathfrak{C} \Delta_{CF} \alpha \mathfrak{J} \mathfrak{C}(\text{ObjMap})(r')$ 
from prems(1) have [cat-cs-simps]:
  cf-of-cf-map  $\mathfrak{J} \mathfrak{C} (\text{cf-map } \mathfrak{F}) = \mathfrak{F}$ 
  cf-of-cf-map  $\mathfrak{J} \mathfrak{C} (\text{cf-map } (\text{cf-const } \mathfrak{J} \mathfrak{C} r')) = \text{cf-const } \mathfrak{J} \mathfrak{C} r'$ 
by (cs-concl cs-simp: cat-FUNCT-cs-simps cs-intro: cat-cs-intros) +
from prems(2,1) have
   $u' : \text{cf-map } \mathfrak{F} \mapsto_{\text{cat-FUNCT}} \alpha \mathfrak{J} \mathfrak{C} \text{cf-map } (\text{cf-const } \mathfrak{J} \mathfrak{C} r')$ 
by (cs-prems cs-shallow cs-simp: cat-cs-simps)
note  $u'[\text{unfolded } \text{cat-cs-simps}] = \text{cat-FUNCT-is-arrD}[OF \text{ this}]$ 

```

from *cat-colim-ua-of*[*OF is-cat-coconeI*[*OF* $u'(1)$ *prems*(1)]] **obtain** f

```

where  $f : r \mapsto_{\mathfrak{C}} r'$ 
and [symmetric, cat-cs-simps]:
  ntcf-of-ntcf-arrow  $\mathfrak{J} \mathfrak{C} u' = \text{ntcf-const } \mathfrak{J} \mathfrak{C} f \cdot_{NTCF} u$ 
and f-unique:
  [[
     $f' : r \mapsto_{\mathfrak{C}} r'$ ;
    ntcf-of-ntcf-arrow  $\mathfrak{J} \mathfrak{C} u' = \text{ntcf-const } \mathfrak{J} \mathfrak{C} f' \cdot_{NTCF} u$ 
  ]]  $\implies f' = f$ 
for  $f'$ 
by metis

```

show $\exists ! f'$.

```

 $f' : r \mapsto_{\mathfrak{C}} r' \wedge$ 
 $u' = \text{umap-of } (\Delta_{CF} \alpha \mathfrak{J} \mathfrak{C}) (\text{cf-map } \mathfrak{F}) r (\text{ntcf-arrow } u) r'(\text{ArrVal})(f')$ 
proof(intro exI conjI; (elim conjE)?)

```

show $f : r \mapsto_{\mathfrak{C}} r'$ **by** (*rule f*)

with $\alpha\beta$ *cat-cocone-obj* **show** u' -def:

```

 $u' = \text{umap-of } (\Delta_{CF} \alpha \mathfrak{J} \mathfrak{C}) (\text{cf-map } \mathfrak{F}) r (\text{ntcf-arrow } u) r'(\text{ArrVal})(f)$ 
by
  (
    cs-concl
    cs-simp:  $u'(2)[\text{symmetric}]$  cat-cs-simps cat-FUNCT-cs-simps
    cs-intro: cat-cs-intros cat-FUNCT-cs-intros
  )

```

fix f' **assume** *prems'*:

```

 $f' : r \mapsto_{\mathfrak{C}} r'$ 
 $u' = \text{umap-of } (\Delta_{CF} \alpha \mathfrak{J} \mathfrak{C}) (\text{cf-map } \mathfrak{F}) r (\text{ntcf-arrow } u) r'(\text{ArrVal})(f')$ 
from prems'(2)  $\alpha\beta$   $f$  prems' cat-cocone-obj have  $u'$ -def':

```

$u' = \text{ntcf-arrow } (\text{ntcf-const } \mathfrak{J} \mathfrak{C} f' \cdot_{NTCF} u)$
by
 (

- cs-prems*
- cs-simp:** *cat-cs-simps cat-FUNCT-cs-simps*
- cs-intro:** *cat-cs-intros cat-FUNCT-cs-intros*

)

from *prems'(1)* **have** $\text{ntcf-of-ntcf-arrow } \mathfrak{J} \mathfrak{C} u' = \text{ntcf-const } \mathfrak{J} \mathfrak{C} f' \cdot_{NTCF} u$
by
 (

- cs-concl cs-shallow*
- cs-simp:** *cat-FUNCT-cs-simps u'-def'* **cs-intro:** *cat-cs-intros*

)

from *f-unique[OF prems'(1) this]* **show** $f' = f$.

qed

qed

lemma (in *is-cat-cocone*) *cat-cocone-is-cat-colimit:*
assumes *universal-arrow-of* $(\Delta_{CF} \alpha \mathfrak{J} \mathfrak{C})$ (*cf-map* \mathfrak{F}) *c* (*ntcf-arrow* \mathfrak{N})
shows $\mathfrak{N} : \mathfrak{F} >_{CF.colim} c : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$
proof-

define β **where** $\beta = \alpha + \omega$
have $\beta : \mathcal{Z} \beta$ **and** $\alpha\beta : \alpha \in_{\circ} \beta$
by (*simp-all add: beta-def Z-Limit- $\alpha\omega$ Z- ω - $\alpha\omega$ Z-def Z- α - $\alpha\omega$)*
then interpret $\beta : \mathcal{Z} \beta$ **by** *simp*

show *?thesis*

proof(*intro is-cat-colimitI is-cat-cocone-axioms*)

fix $u' c'$ **assume** *prems:* $u' : \mathfrak{F} >_{CF.cocone} c' : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$

interpret $u' : \text{is-cat-cocone } \alpha c' \mathfrak{J} \mathfrak{C} \mathfrak{F} u'$ **by** (*rule prems*)

from $u'.\text{cat-cocone-obj}$ **have** $u'.\text{is-arr}$:
 $\text{ntcf-arrow } u' : \text{cf-map } \mathfrak{F} \mapsto_{\text{cat-FUNCT}} \alpha \mathfrak{J} \mathfrak{C} \Delta_{CF} \alpha \mathfrak{J} \mathfrak{C} (\text{ObjMap})(\downarrow c')$
by
 (

- cs-concl cs-shallow*
- cs-simp:** *cat-cs-simps* **cs-intro:** *cat-cs-intros cat-FUNCT-cs-intros*

)

from *is-functor.universal-arrow-ofD(3)*

[

- OF*
- cf-diagonal-is-functor*[
 - OF* $\beta \alpha\beta$ *NTDom.HomDom.category-axioms* *NTDom.HomCod.category-axioms*
]
- assms*
- $u'.\text{cat-cocone-obj}$
- $u'.\text{is-arr}$

]

obtain f **where** $f : f : c \mapsto_{\mathfrak{C}} c'$
and $u'.\text{def}' : \text{ntcf-arrow } u' =$
 $\text{umap-of } (\Delta_{CF} \alpha \mathfrak{J} \mathfrak{C})$ (*cf-map* \mathfrak{F}) *c* (*ntcf-arrow* \mathfrak{N}) $c'(\text{ArrVal})(\downarrow f)$
and $f'.\text{unique}$:

$$\llbracket$$

$$f' : c \mapsto_{\mathfrak{C}} c';$$

$$ntcf\text{-arrow } u' =$$

$$umap\text{-of } (\Delta_{CF} \alpha \mathfrak{J} \mathfrak{C}) (cf\text{-map } \mathfrak{F}) c (ntcf\text{-arrow } \mathfrak{N}) c'(\text{ArrVal})(f')$$

$$\rrbracket \implies f' = f$$
for f'
by *metis*

from $u'\text{-def}' \alpha \beta f \text{ cat-cocone-obj}$ **have** $u'\text{-def}$:

$u' = ntcf\text{-const } \mathfrak{J} \mathfrak{C} f \cdot_{NTCF} \mathfrak{N}$

by

(

cs-prems

cs-simp: *cat-cs-simps cat-FUNCT-cs-simps*

cs-intro: *cat-cs-intros cat-FUNCT-cs-intros*

)

show $\exists! f'. f' : c \mapsto_{\mathfrak{C}} c' \wedge u' = ntcf\text{-const } \mathfrak{J} \mathfrak{C} f' \cdot_{NTCF} \mathfrak{N}$

proof(*intro exII conjI; (elim conjE)?, (rule f)?, (rule u'-def)?*)

fix f'' **assume** *prems'*:

$f'' : c \mapsto_{\mathfrak{C}} c' \wedge u' = ntcf\text{-const } \mathfrak{J} \mathfrak{C} f'' \cdot_{NTCF} \mathfrak{N}$

from $\alpha \beta$ *prems'* **have**

$ntcf\text{-arrow } u' =$

$umap\text{-of } (\Delta_{CF} \alpha \mathfrak{J} \mathfrak{C}) (cf\text{-map } \mathfrak{F}) c (ntcf\text{-arrow } \mathfrak{N}) c'(\text{ArrVal})(f'')$

by

(

cs-concl

cs-simp: *cat-cs-simps cat-FUNCT-cs-simps*

cs-intro: *cat-cs-intros cat-FUNCT-cs-intros*

)

from $f'\text{-unique}[OF \text{prems}'(1) \text{ this}]$ **show** $f'' = f$.

qed

qed

qed

Duality.

lemma (*in is-cat-limit*) *is-cat-colimit-op*:

$op\text{-ntcf } u : op\text{-cf } \mathfrak{F} >_{CF.colim} r : op\text{-cat } \mathfrak{J} \mapsto_{C\alpha} op\text{-cat } \mathfrak{C}$

proof(*intro is-cat-colimitI*)

show $op\text{-ntcf } u : op\text{-cf } \mathfrak{F} >_{CF.cocone} r : op\text{-cat } \mathfrak{J} \mapsto_{C\alpha} op\text{-cat } \mathfrak{C}$

by (*cs-concl cs-shallow cs-simp: cs-intro: cat-op-intros*)

fix $u' r'$ **assume** *prems*:

$u' : op\text{-cf } \mathfrak{F} >_{CF.cocone} r' : op\text{-cat } \mathfrak{J} \mapsto_{C\alpha} op\text{-cat } \mathfrak{C}$

interpret u' : *is-cat-cocone* α $r' \langle op\text{-cat } \mathfrak{J} \rangle \langle op\text{-cat } \mathfrak{C} \rangle \langle op\text{-cf } \mathfrak{F} \rangle u'$

by (*rule prems*)

from *cat-lim-ua-fo*[*OF u'.is-cat-cone-op[unfolding cat-op-simps]*] **obtain** f

where $f : r' \mapsto_{\mathfrak{C}} r$

and $op\text{-}u'\text{-def}$: $op\text{-ntcf } u' = u \cdot_{NTCF} ntcf\text{-const } \mathfrak{J} \mathfrak{C} f$

and $f\text{-unique}$:

$\llbracket f' : r' \mapsto_{\mathfrak{C}} r; op\text{-ntcf } u' = u \cdot_{NTCF} ntcf\text{-const } \mathfrak{J} \mathfrak{C} f' \rrbracket \implies$

$f' = f$

for f'

by *metis*

from $op\text{-}u'\text{-def}$ **have** $op\text{-ntcf } (op\text{-ntcf } u') = op\text{-ntcf } (u \cdot_{NTCF} ntcf\text{-const } \mathfrak{J} \mathfrak{C} f)$

by *simp*

from *this f* **have** $u'\text{-def}$:

$u' = \text{ntcf-const } (\text{op-cat } \mathfrak{J}) (\text{op-cat } \mathfrak{C}) f \cdot_{NTCF} \text{op-ntcf } u$
by (*cs-prems cs-simp: cat-op-simps cs-intro: cat-cs-intros*)
show $\exists ! f'$.
 $f' : r \mapsto_{\text{op-cat } \mathfrak{C}} r' \wedge$
 $u' = \text{ntcf-const } (\text{op-cat } \mathfrak{J}) (\text{op-cat } \mathfrak{C}) f' \cdot_{NTCF} \text{op-ntcf } u$
proof(*intro ex1I conjI; (elim conjE)?, (unfold cat-op-simps)?*)
fix f' **assume** *prems'*:
 $f' : r' \mapsto_{\mathfrak{C}} r$
 $u' = \text{ntcf-const } (\text{op-cat } \mathfrak{J}) (\text{op-cat } \mathfrak{C}) f' \cdot_{NTCF} \text{op-ntcf } u$
from *prems'*(2) **have**
 $\text{op-ntcf } u' = \text{op-ntcf } (\text{ntcf-const } (\text{op-cat } \mathfrak{J}) (\text{op-cat } \mathfrak{C}) f' \cdot_{NTCF} \text{op-ntcf } u)$
by *simp*
from *this prems'*(1) **have** $\text{op-ntcf } u' = u \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{C} f'$
by
(

cs-prems
cs-simp: *cat-cs-simps cat-op-simps*
cs-intro: *cat-cs-intros cat-op-intros*
)

from *f-unique*[*OF prems'*(1) *this*] **show** $f' = f$.
qed (*intro u'-def f*)
qed

lemma (**in** *is-cat-limit*) *is-cat-colimit-op'*[*cat-op-intros*]:
assumes $\mathfrak{F}' = \text{op-cf } \mathfrak{F}$ **and** $\mathfrak{J}' = \text{op-cat } \mathfrak{J}$ **and** $\mathfrak{C}' = \text{op-cat } \mathfrak{C}$
shows $\text{op-ntcf } u : \mathfrak{F}' >_{CF.colim} r : \mathfrak{J}' \mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-colimit-op*)

lemmas [*cat-op-intros*] = *is-cat-limit.is-cat-colimit-op'*

lemma (**in** *is-cat-colimit*) *is-cat-limit-op*:
 $\text{op-ntcf } u : r <_{CF.lim} \text{op-cf } \mathfrak{F} : \text{op-cat } \mathfrak{J} \mapsto_{C\alpha} \text{op-cat } \mathfrak{C}$
proof(*intro is-cat-limitI*)
show $\text{op-ntcf } u : r <_{CF.cone} \text{op-cf } \mathfrak{F} : \text{op-cat } \mathfrak{J} \mapsto_{C\alpha} \text{op-cat } \mathfrak{C}$
by (*cs-concl cs-shallow cs-simp: cs-intro: cat-op-intros*)
fix $u' r'$ **assume** *prems*:
 $u' : r' <_{CF.cone} \text{op-cf } \mathfrak{F} : \text{op-cat } \mathfrak{J} \mapsto_{C\alpha} \text{op-cat } \mathfrak{C}$
interpret u' : *is-cat-cone* α $r' \langle \text{op-cat } \mathfrak{J} \rangle \langle \text{op-cat } \mathfrak{C} \rangle \langle \text{op-cf } \mathfrak{F} \rangle u'$
by (*rule prems*)
from *cat-colim-ua-of*[*OF u'.is-cat-cocone-op*[*unfolded cat-op-simps*]] **obtain** f
where $f : r \mapsto_{\mathfrak{C}} r'$
and *op-u'-def*: $\text{op-ntcf } u' = \text{ntcf-const } \mathfrak{J} \mathfrak{C} f \cdot_{NTCF} u$
and *f-unique*:
 $\llbracket f' : r \mapsto_{\mathfrak{C}} r'; \text{op-ntcf } u' = \text{ntcf-const } \mathfrak{J} \mathfrak{C} f' \cdot_{NTCF} u \rrbracket \implies$
 $f' = f$
for f'
by *metis*
from *op-u'-def* **have** $\text{op-ntcf } (\text{op-ntcf } u') = \text{op-ntcf } (\text{ntcf-const } \mathfrak{J} \mathfrak{C} f \cdot_{NTCF} u)$
by *simp*
from *this f* **have** *u'-def*:
 $u' = \text{op-ntcf } u \cdot_{NTCF} \text{ntcf-const } (\text{op-cat } \mathfrak{J}) (\text{op-cat } \mathfrak{C}) f$
by (*cs-prems cs-simp: cat-op-simps cs-intro: cat-cs-intros*)
show $\exists ! f'$.
 $f' : r' \mapsto_{\text{op-cat } \mathfrak{C}} r \wedge$
 $u' = \text{op-ntcf } u \cdot_{NTCF} \text{ntcf-const } (\text{op-cat } \mathfrak{J}) (\text{op-cat } \mathfrak{C}) f'$
proof(*intro ex1I conjI; (elim conjE)?, (unfold cat-op-simps)?*)
fix f' **assume** *prems'*:
 $f' : r \mapsto_{\mathfrak{C}} r'$

$u' = \text{op-ntcf } u \cdot_{NTCF} \text{ntcf-const } (\text{op-cat } \mathfrak{J}) (\text{op-cat } \mathfrak{C}) f'$
from $\text{prems}'(2)$ **have**
 $\text{op-ntcf } u' = \text{op-ntcf } (\text{op-ntcf } u \cdot_{NTCF} \text{ntcf-const } (\text{op-cat } \mathfrak{J}) (\text{op-cat } \mathfrak{C}) f')$
by *simp*
from $\text{this } \text{prems}'(1)$ **have** $\text{op-ntcf } u' = \text{ntcf-const } \mathfrak{J} \mathfrak{C} f' \cdot_{NTCF} u$
by
(

cs-prems
cs-simp: *cat-cs-simps cat-op-simps*
cs-intro: *cat-cs-intros cat-op-intros*
)

from $f\text{-unique}[OF \text{prems}'(1) \text{this}]$ **show** $f' = f$.
qed (*intro u'-def f*)
qed

lemma (**in** *is-cat-colimit*) *is-cat-colimit-op'*[*cat-op-intros*]:
assumes $\mathfrak{F}' = \text{op-cf } \mathfrak{F}$ **and** $\mathfrak{J}' = \text{op-cat } \mathfrak{J}$ **and** $\mathfrak{C}' = \text{op-cat } \mathfrak{C}$
shows $\text{op-ntcf } u : r <_{CF.lim} \mathfrak{F}' : \mathfrak{J}' \mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-limit-op*)

lemmas [*cat-op-intros*] = *is-cat-colimit.is-cat-colimit-op'*

3.2.2 Universal property

lemma (**in** *is-cat-limit*) *cat-lim-unique-cone'*:
assumes $u' : r' <_{CF.cone} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$
shows
 $\exists ! f'. f' : r' \mapsto_{\mathfrak{C}} r \wedge (\forall j \in \circ \mathfrak{J}(\text{Obj}). u'(\text{NTMap})(j) = u(\text{NTMap})(j) \circ_{A\mathfrak{C}} f')$
by (*fold helper-cat-cone-Comp-ntcf-vcomp-iff[OF assms(1)]*)
(*intro cat-lim-ua-fo assms*)

lemma (**in** *is-cat-limit*) *cat-lim-unique*:
assumes $u' : r' <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$
shows $\exists ! f'. f' : r' \mapsto_{\mathfrak{C}} r \wedge u' = u \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{C} f'$
by (*intro cat-lim-ua-fo[OF is-cat-limitD(1)[OF assms]]*)

lemma (**in** *is-cat-limit*) *cat-lim-unique'*:
assumes $u' : r' <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$
shows
 $\exists ! f'. f' : r' \mapsto_{\mathfrak{C}} r \wedge (\forall j \in \circ \mathfrak{J}(\text{Obj}). u'(\text{NTMap})(j) = u(\text{NTMap})(j) \circ_{A\mathfrak{C}} f')$
by (*intro cat-lim-unique-cone'[OF is-cat-limitD(1)[OF assms]]*)

lemma (**in** *is-cat-colimit*) *cat-colim-unique-cocone*:
assumes $u' : \mathfrak{F} >_{CF.cocone} r' : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$
shows $\exists ! f'. f' : r \mapsto_{\mathfrak{C}} r' \wedge u' = \text{ntcf-const } \mathfrak{J} \mathfrak{C} f' \cdot_{NTCF} u$

proof-

interpret u' : *is-cat-cocone* α $r' \mathfrak{J} \mathfrak{C} \mathfrak{F} u'$ **by** (*rule assms(1)*)

from $u'.\text{cat-cocone-obj}$ **have** $\text{op-}r'$: $r' \in_{\circ} \text{op-cat } \mathfrak{C}(\text{Obj})$

unfolding *cat-op-simps* **by** *simp*

from

is-cat-limit.cat-lim-ua-fo[
 $OF \text{is-cat-limit-op } u'.\text{is-cat-cone-op, folded op-ntcf-ntcf-const}$
]

obtain f' **where** $f' : f' : r' \mapsto_{\text{op-cat } \mathfrak{C}} r$

and [*cat-cs-simps*]:

$\text{op-ntcf } u' = \text{op-ntcf } u \cdot_{NTCF} \text{op-ntcf } (\text{ntcf-const } \mathfrak{J} \mathfrak{C} f')$

and *unique*:

[[

$f'' : r' \mapsto_{op-cat} \mathfrak{C} r$;
 $op-ntcf\ u' = op-ntcf\ u \cdot_{NTCF} op-ntcf\ (ntcf-const\ \mathfrak{J}\ \mathfrak{C}\ f'')$
 $\Downarrow \implies f'' = f'$
for f''
by *metis*
show *?thesis*
proof(*intro ex1I conjI; (elim conjE)?*)
from f' **show** $f' : f' : r \mapsto_{\mathfrak{C}} r'$ **unfolding** *cat-op-simps by simp*
show $u' = ntcf-const\ \mathfrak{J}\ \mathfrak{C}\ f' \cdot_{NTCF} u$
by (*rule eq-op-ntcf-iff[THEN iffD1], insert f'*)
(cs-concl cs-intro: cat-cs-intros cs-simp: cat-cs-simps cat-op-simps)+
fix f'' **assume** *prems: f'' : r \mapsto_{\mathfrak{C}} r' u' = ntcf-const \mathfrak{J} \mathfrak{C} f'' \cdot_{NTCF} u*
from *prems(1)* **have** $f'' : r' \mapsto_{op-cat} \mathfrak{C} r$ **unfolding** *cat-op-simps by simp*
moreover from *prems(1)* **have**
 $op-ntcf\ u' = op-ntcf\ u \cdot_{NTCF} op-ntcf\ (ntcf-const\ \mathfrak{J}\ \mathfrak{C}\ f'')$
unfolding *prems(2)*
by (*cs-concl cs-intro: cat-cs-intros cs-simp: cat-cs-simps cat-op-simps*)
ultimately show $f'' = f'$ **by** (*rule unique*)
qed
qed

lemma (*in is-cat-colimit*) *cat-colim-unique-cocone'*:
assumes $u' : \mathfrak{F} >_{CF.cocone} r' : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$
shows
 $\exists !f'. f' : r \mapsto_{\mathfrak{C}} r' \wedge (\forall j \in \circ \mathfrak{J} (Obj). u' (NTMap) (j) = f' \circ_{A\mathfrak{C}} u (NTMap) (j))$
by (*fold helper-cat-cocone-Comp-ntcf-vcomp-iff[OF assms(1)]*)
(intro cat-colim-unique-cocone assms)

lemma (*in is-cat-colimit*) *cat-colim-unique*:
assumes $u' : \mathfrak{F} >_{CF.colim} r' : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$
shows $\exists !f'. f' : r \mapsto_{\mathfrak{C}} r' \wedge u' = ntcf-const\ \mathfrak{J}\ \mathfrak{C}\ f' \cdot_{NTCF} u$
by (*intro cat-colim-unique-cocone[OF is-cat-colimitD(1)[OF assms]]*)

lemma (*in is-cat-colimit*) *cat-colim-unique'*:
assumes $u' : \mathfrak{F} >_{CF.colim} r' : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$
shows
 $\exists !f'. f' : r \mapsto_{\mathfrak{C}} r' \wedge (\forall j \in \circ \mathfrak{J} (Obj). u' (NTMap) (j) = f' \circ_{A\mathfrak{C}} u (NTMap) (j))$

proof-
interpret $u' : is-cat-colimit\ \alpha\ \mathfrak{J}\ \mathfrak{C}\ \mathfrak{F}\ r'\ u'$ **by** (*rule assms(1)*)
show *?thesis*
by (*fold helper-cat-cocone-Comp-ntcf-vcomp-iff[OF u'.is-cat-cocone-axioms]*)
(intro cat-colim-unique assms)

qed

lemma *cat-lim-ex-is-iso-arr*:
assumes $u : r <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$ **and** $u' : r' <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$
obtains f **where** $f : r' \mapsto_{iso\mathfrak{C}} r$ **and** $u' = u \cdot_{NTCF} ntcf-const\ \mathfrak{J}\ \mathfrak{C}\ f$

proof-

interpret $u : is-cat-limit\ \alpha\ \mathfrak{J}\ \mathfrak{C}\ \mathfrak{F}\ r\ u$ **by** (*rule assms(1)*)
interpret $u' : is-cat-limit\ \alpha\ \mathfrak{J}\ \mathfrak{C}\ \mathfrak{F}\ r'\ u'$ **by** (*rule assms(2)*)
define β **where** $\beta = \alpha + \omega$
have $\beta : \mathcal{Z}\ \beta$ **and** $\alpha\beta : \alpha \in \circ \beta$
by (*simp-all add: \beta-def u.\mathcal{Z}-Limit-\alpha\omega u.\mathcal{Z}-\omega-\alpha\omega \mathcal{Z}-def u.\mathcal{Z}-\alpha-\alpha\omega*)
then interpret $\beta : \mathcal{Z}\ \beta$ **by** *simp*
have $\Delta : \Delta_{CF}\ \alpha\ \mathfrak{J}\ \mathfrak{C} : \mathfrak{C} \mapsto_{C\beta} cat-FUNCT\ \alpha\ \mathfrak{J}\ \mathfrak{C}$

by
 $($
intro

$\beta \alpha \beta$
cf-diagonal-is-functor
u.NTDom.HomDom.category-axioms
u.NTDom.HomCod.category-axioms
)

then interpret Δ : *is-functor* $\beta \mathfrak{C} \langle \text{cat-FUNCT } \alpha \mathfrak{J} \mathfrak{C} \rangle \langle \Delta_{CF} \alpha \mathfrak{J} \mathfrak{C} \rangle$ **by** *simp*
from *is-functor.cf-universal-arrow-fo-ex-is-iso-arr*[
OF Δ *u.cat-lim-is-universal-arrow-fo* *u'.cat-lim-is-universal-arrow-fo*
]
obtain f **where** $f: f: r' \mapsto_{\text{iso}\mathfrak{C}} r$
and u' : *ntcf-arrow* $u' =$
umap-fo ($\Delta_{CF} \alpha \mathfrak{J} \mathfrak{C}$) (*cf-map* \mathfrak{F}) r (*ntcf-arrow* u) $r'(\text{ArrVal})(f)$
by *auto*
from f **have** $f: r' \mapsto_{\mathfrak{C}} r$ **by** *auto*
from u' **this have** $u' = u \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{C} f$
by
(
cs-prems
cs-simp: *cat-cs-simps cat-FUNCT-cs-simps*
cs-intro: *cat-cs-intros cat-FUNCT-cs-intros*
)
with f **that show** *?thesis* **by** *simp*
qed

lemma *cat-lim-ex-is-iso-arr'*:
assumes $u: r <_{CF.lim} \mathfrak{F}: \mathfrak{J} \mapsto_{\mapsto} C\alpha \mathfrak{C}$ **and** $u': r' <_{CF.lim} \mathfrak{F}: \mathfrak{J} \mapsto_{\mapsto} C\alpha \mathfrak{C}$
obtains f **where** $f: r' \mapsto_{\text{iso}\mathfrak{C}} r$
and $\bigwedge j. j \in_{\circ} \mathfrak{J}(\text{Obj}) \implies u'(\text{NTMap})(j) = u(\text{NTMap})(j) \circ_{A\mathfrak{C}} f$
proof-
interpret u : *is-cat-limit* $\alpha \mathfrak{J} \mathfrak{C} \mathfrak{F} r u$ **by** (*rule assms(1)*)
interpret u' : *is-cat-limit* $\alpha \mathfrak{J} \mathfrak{C} \mathfrak{F} r' u'$ **by** (*rule assms(2)*)
from *assms* **obtain** f
where *iso-f*: $f: r' \mapsto_{\text{iso}\mathfrak{C}} r$ **and** *u'-def*: $u' = u \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{C} f$
by (*rule cat-lim-ex-is-iso-arr*)
then have $f: f: r' \mapsto_{\mathfrak{C}} r$ **by** *auto*
then have $u'(\text{NTMap})(j) = u(\text{NTMap})(j) \circ_{A\mathfrak{C}} f$ **if** $j \in_{\circ} \mathfrak{J}(\text{Obj})$ **for** j
by
(
intro *u.helper-cat-cone-ntcf-vcomp-Comp*[
OF *u'.is-cat-cone-axioms* *f u'-def* *that*
]
)
with *iso-f* **that show** *?thesis* **by** *simp*
qed

lemma *cat-colim-ex-is-iso-arr*:
assumes $u: \mathfrak{F} >_{CF.colim} r: \mathfrak{J} \mapsto_{\mapsto} C\alpha \mathfrak{C}$
and $u': \mathfrak{F} >_{CF.colim} r': \mathfrak{J} \mapsto_{\mapsto} C\alpha \mathfrak{C}$
obtains f **where** $f: r \mapsto_{\text{iso}\mathfrak{C}} r'$ **and** $u' = \text{ntcf-const } \mathfrak{J} \mathfrak{C} f \cdot_{NTCF} u$
proof-
interpret u : *is-cat-colimit* $\alpha \mathfrak{J} \mathfrak{C} \mathfrak{F} r u$ **by** (*rule assms(1)*)
interpret u' : *is-cat-colimit* $\alpha \mathfrak{J} \mathfrak{C} \mathfrak{F} r' u'$ **by** (*rule assms(2)*)
obtain f **where** $f: f: r' \mapsto_{\text{iso}\text{op-cat}} \mathfrak{C} r$
and [*cat-cs-simps*]:
op-ntcf $u' = \text{op-ntcf } u \cdot_{NTCF} \text{ntcf-const } (\text{op-cat } \mathfrak{J}) (\text{op-cat } \mathfrak{C}) f$
by
(
elim cat-lim-ex-is-iso-arr[

$OF\ u.is\text{-}cat\text{-}limit\text{-}op\ u'.is\text{-}cat\text{-}limit\text{-}op$
 $]$
 $)$
from f **have** $iso\text{-}f: f : r \mapsto_{iso} \mathfrak{C} r'$ **unfolding** $cat\text{-}op\text{-}simps$ **by** $simp$
then have $f: f : r \mapsto_{\mathfrak{C}} r'$ **by** $auto$
have $u' = ntcf\text{-}const\ \mathfrak{J}\ \mathfrak{C}\ f \cdot_{NTCF}\ u$
by $(rule\ eq\text{-}op\text{-}ntcf\text{-}iff[THEN\ iffD1],\ insert\ f)$
 $(cs\text{-}concl\ \mathbf{cs}\text{-}intro: cat\text{-}cs\text{-}intros\ \mathbf{cs}\text{-}simp: cat\text{-}cs\text{-}simps\ cat\text{-}op\text{-}simps)+$
from $iso\text{-}f$ **this that show** $?thesis$ **by** $simp$
qed

lemma $cat\text{-}colim\text{-}ex\text{-}is\text{-}iso\text{-}arr'$:
assumes $u : \mathfrak{F} >_{CF.colim} r : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$
and $u' : \mathfrak{F} >_{CF.colim} r' : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$
obtains f **where** $f : r \mapsto_{iso} \mathfrak{C} r'$
and $\bigwedge j. j \in_o \mathfrak{J}(Obj) \implies u'(NTMap)(j) = f \circ_{A\mathfrak{C}} u(NTMap)(j)$

proof-

interpret $u: is\text{-}cat\text{-}colimit\ \alpha\ \mathfrak{J}\ \mathfrak{C}\ \mathfrak{F}\ r\ u$ **by** $(rule\ assms(1))$
interpret $u': is\text{-}cat\text{-}colimit\ \alpha\ \mathfrak{J}\ \mathfrak{C}\ \mathfrak{F}\ r'\ u'$ **by** $(rule\ assms(2))$
from $assms$ **obtain** f
where $iso\text{-}f: f : r \mapsto_{iso} \mathfrak{C} r'$ **and** $u'\text{-}def: u' = ntcf\text{-}const\ \mathfrak{J}\ \mathfrak{C}\ f \cdot_{NTCF}\ u$
by $(rule\ cat\text{-}colim\text{-}ex\text{-}is\text{-}iso\text{-}arr)$
then have $f: f : r \mapsto_{\mathfrak{C}} r'$ **by** $auto$
then have $u'(NTMap)(j) = f \circ_{A\mathfrak{C}} u(NTMap)(j)$ **if** $j \in_o \mathfrak{J}(Obj)$ **for** j
by
 $($
 $intro\ u.helper\text{-}cat\text{-}cocone\text{-}ntcf\text{-}vcomp\text{-}Comp[$
 $OF\ u'.is\text{-}cat\text{-}cocone\text{-}axioms\ f\ u'\text{-}def\ that$
 $]$
 $)$
with $iso\text{-}f$ **that show** $?thesis$ **by** $simp$
qed

3.2.3 Further properties

lemma $(in\ is\text{-}cat\text{-}limit)\ cat\text{-}lim\text{-}is\text{-}cat\text{-}limit\text{-}if\text{-}is\text{-}iso\text{-}arr$:
assumes $f : r' \mapsto_{iso} \mathfrak{C} r$
shows $u \cdot_{NTCF}\ ntcf\text{-}const\ \mathfrak{J}\ \mathfrak{C}\ f : r' <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$
proof-
note $f = is\text{-}iso\text{-}arrD(1)[OF\ assms(1)]$
from $f(1)$ **interpret** $u': is\text{-}cat\text{-}cone\ \alpha\ r'\ \mathfrak{J}\ \mathfrak{C}\ \mathfrak{F}\ \langle u \cdot_{NTCF}\ ntcf\text{-}const\ \mathfrak{J}\ \mathfrak{C}\ f \rangle$
by $(cs\text{-}concl\ \mathbf{cs}\text{-}intro: cat\text{-}lim\text{-}cs\text{-}intros\ cat\text{-}cs\text{-}intros)$
define β **where** $\beta = \alpha + \omega$
have $\beta: \mathcal{Z}\ \beta$ **and** $\alpha\beta: \alpha \in_o \beta$
by $(simp\text{-}all\ add: \beta\text{-}def\ \mathcal{Z}\text{-}Limit\text{-}\alpha\omega\ \mathcal{Z}\text{-}\omega\text{-}\alpha\omega\ \mathcal{Z}\text{-}def\ \mathcal{Z}\text{-}\alpha\text{-}\alpha\omega)$
then interpret $\beta: \mathcal{Z}\ \beta$ **by** $simp$
show $?thesis$
proof
 $($
 $intro\ u'.cat\text{-}cone\text{-}is\text{-}cat\text{-}limit,$
 $rule\ is\text{-}functor.universal\text{-}arrow\text{-}fo\text{-}if\text{-}universal\text{-}arrow\text{-}fo,$
 $rule\ cf\text{-}diagonal\text{-}is\text{-}functor[OF\ \beta\ \alpha\beta],$
 $rule\ NTDom.HomDom.category\text{-}axioms,$
 $rule\ NTDom.HomCod.category\text{-}axioms,$
 $rule\ cat\text{-}lim\text{-}is\text{-}universal\text{-}arrow\text{-}fo$
 $)$
show $f : r' \mapsto_{iso} \mathfrak{C} r$ **by** $(rule\ assms(1))$
from $\alpha\beta$ f **show**

$ntcf\text{-arrow } (u \cdot_{NTCF} ntcf\text{-const } \mathfrak{J} \mathfrak{C} f) =$
 $umap\text{-fo } (\Delta_{CF} \alpha \mathfrak{J} \mathfrak{C}) (cf\text{-map } \mathfrak{F}) r (ntcf\text{-arrow } u) r' (ArrVal) (f)$
by
 (

- cs-concl*
- cs-simp:** *cat-cs-simps cat-FUNCT-cs-simps*
- cs-intro:** *cat-cs-intros cat-FUNCT-cs-intros*

)

qed
qed

lemma (in *is-cat-colimit*) *cat-colim-is-cat-colimit-if-is-iso-arr:*

assumes $f : r \mapsto_{iso} \mathfrak{C} r'$

shows $ntcf\text{-const } \mathfrak{J} \mathfrak{C} f \cdot_{NTCF} u : \mathfrak{F} >_{CF.colim} r' : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$

proof-

note $f = is\text{-iso-arr}D[OF\ assms(1)]$

from $f(1)$ **interpret** $u' : is\text{-cat-cocone } \alpha r' \mathfrak{J} \mathfrak{C} \mathfrak{F} \langle ntcf\text{-const } \mathfrak{J} \mathfrak{C} f \cdot_{NTCF} u \rangle$

by (*cs-concl cs-intro: cat-lim-cs-intros cat-cs-intros*)

from f **have** [*symmetric, cat-op-simps*]:

$op\text{-ntcf } (ntcf\text{-const } \mathfrak{J} \mathfrak{C} f \cdot_{NTCF} u) =$
 $op\text{-ntcf } u \cdot_{NTCF} ntcf\text{-const } (op\text{-cat } \mathfrak{J}) (op\text{-cat } \mathfrak{C}) f$

by

(

- cs-concl cs-shallow*
- cs-simp:** *cat-op-simps cs-intro: cat-cs-intros cat-op-intros*

)

show *?thesis*

by

(

- rule is-cat-limit.is-cat-colimit-op*
- [
 - $OF\ is\text{-cat-limit.cat-lim-is-cat-limit-if-is-iso-arr}[$
 - $OF\ is\text{-cat-limit-op, unfolded cat-op-simps, OF\ assms(1)$
 -],
 - unfolded cat-op-simps*
]
)

qed

lemma *ntcf-cf-comp-is-cat-limit-if-is-iso-functor:*

assumes $u : r <_{CF.lim} \mathfrak{F} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{G} : \mathfrak{A} \mapsto_{C.iso\alpha} \mathfrak{B}$

shows $u \circ_{NTCF-CF} \mathfrak{G} : r <_{CF.lim} \mathfrak{F} \circ_{CF} \mathfrak{G} : \mathfrak{A} \mapsto_{C\alpha} \mathfrak{C}$

proof(*intro is-cat-limitI*)

interpret $u : is\text{-cat-limit } \alpha \mathfrak{B} \mathfrak{C} \mathfrak{F} r u$ **by** (*rule assms(1)*)

interpret $\mathfrak{G} : is\text{-iso-functor } \alpha \mathfrak{A} \mathfrak{B} \mathfrak{G}$ **by** (*rule assms(2)*)

note [*cf-cs-simps*] = *is-iso-functor-is-iso-arr(2,3)*

show $u\mathfrak{G} : u \circ_{NTCF-CF} \mathfrak{G} : r <_{CF.cone} \mathfrak{F} \circ_{CF} \mathfrak{G} : \mathfrak{A} \mapsto_{C\alpha} \mathfrak{C}$

by (*intro is-cat-coneI*)

(*cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

fix $u' r'$ **assume** *prems*: $u' : r' <_{CF.cone} \mathfrak{F} \circ_{CF} \mathfrak{G} : \mathfrak{A} \mapsto_{C\alpha} \mathfrak{C}$

then interpret $u' : is\text{-cat-cone } \alpha r' \mathfrak{A} \mathfrak{C} \langle \mathfrak{F} \circ_{CF} \mathfrak{G} \rangle u'$ **by** *simp*

have $u' \circ_{NTCF-CF} inv\text{-cf } \mathfrak{G} : r' <_{CF.cone} \mathfrak{F} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{C}$

by (*intro is-cat-coneI*)

(

cs-concl
cs-simp: *cat-cs-simps cf-cs-simps*
cs-intro: *cat-cs-intros cf-cs-intros*

)

from *is-cat-limit.cat-lim-ua-fo*[*OF assms(1) this*] **obtain** f
where $f : r' \mapsto_{\mathcal{C}} r$
and $u'-\mathfrak{G} : u' \circ_{NTCF-CF} \text{inv-cf } \mathfrak{G} = u \cdot_{NTCF} \text{ntcf-const } \mathfrak{B} \mathcal{C} f$
and $f'f$:

$$\begin{aligned} & \llbracket \\ & \quad f' : r' \mapsto_{\mathcal{C}} r; \\ & \quad u' \circ_{NTCF-CF} \text{inv-cf } \mathfrak{G} = u \cdot_{NTCF} \text{ntcf-const } \mathfrak{B} \mathcal{C} f' \\ & \rrbracket \implies f' = f \end{aligned}$$

for f'
by *metis*
from $u'-\mathfrak{G}$ **have** $u'-\text{inv}\mathfrak{G}-\mathfrak{G}$:
 $(u' \circ_{NTCF-CF} \text{inv-cf } \mathfrak{G}) \circ_{NTCF-CF} \mathfrak{G} = (u \cdot_{NTCF} \text{ntcf-const } \mathfrak{B} \mathcal{C} f) \circ_{NTCF-CF} \mathfrak{G}$
by *simp*
show $\exists! f'. f' : r' \mapsto_{\mathcal{C}} r \wedge u' = u \circ_{NTCF-CF} \mathfrak{G} \cdot_{NTCF} \text{ntcf-const } \mathfrak{A} \mathcal{C} f'$
proof(*intro ex1I conjI; (elim conjE)?*)
show $f : r' \mapsto_{\mathcal{C}} r$ **by** (*rule f*)
from $u'-\text{inv}\mathfrak{G}-\mathfrak{G}$ f **show** $u' = u \circ_{NTCF-CF} \mathfrak{G} \cdot_{NTCF} \text{ntcf-const } \mathfrak{A} \mathcal{C} f'$
by
(

cs-prems
cs-simp:
cf-cs-simps cat-cs-simps
ntcf-cf-comp-ntcf-cf-comp-assoc
ntcf-vcomp-ntcf-cf-comp[symmetric]
cs-intro: *cat-cs-intros cf-cs-intros*
)

fix f' **assume** *prems*:
 $f' : r' \mapsto_{\mathcal{C}} r \wedge u' = u \circ_{NTCF-CF} \mathfrak{G} \cdot_{NTCF} \text{ntcf-const } \mathfrak{A} \mathcal{C} f'$
from *prems(2)* **have**
 $u' \circ_{NTCF-CF} \text{inv-cf } \mathfrak{G} =$
 $(u \circ_{NTCF-CF} \mathfrak{G} \cdot_{NTCF} \text{ntcf-const } \mathfrak{A} \mathcal{C} f') \circ_{NTCF-CF} \text{inv-cf } \mathfrak{G}$
by *simp*
from *this f prems(1)* **have** $u' \circ_{NTCF-CF} \text{inv-cf } \mathfrak{G} = u \cdot_{NTCF} \text{ntcf-const } \mathfrak{B} \mathcal{C} f'$
by
(

cs-prems
cs-simp:
cat-cs-simps cf-cs-simps
ntcf-vcomp-ntcf-cf-comp[symmetric]
ntcf-cf-comp-ntcf-cf-comp-assoc
cs-intro: *cf-cs-intros cat-cs-intros*
)

then show $f' = f$ **by** (*intro f'f prems(1)*)
qed
qed

lemma *ntcf-cf-comp-is-cat-limit-if-is-iso-functor'*[*cat-lim-cs-intros*]:
assumes $u : r <_{CF.lim} \mathfrak{F} : \mathfrak{B} \mapsto_{C\alpha} \mathcal{C}$
and $\mathfrak{G} : \mathfrak{A} \mapsto_{C.iso\alpha} \mathfrak{B}$
and $\mathfrak{A}' = \mathfrak{F} \circ_{CF} \mathfrak{G}$
shows $u \circ_{NTCF-CF} \mathfrak{G} : r <_{CF.lim} \mathfrak{A}' : \mathfrak{A} \mapsto_{C\alpha} \mathcal{C}$
using *assms(1,2)*
unfolding *assms(3)*
by (*rule ntcf-cf-comp-is-cat-limit-if-is-iso-functor*)

3.3 Small limit and small colimit

3.3.1 Definition and elementary properties

The concept of a limit is introduced in Chapter III-4 in [9]; the concept of a colimit is introduced in Chapter III-3 in [9]. The definitions of small limits were tailored for ZFC in HOL.

locale *is-tm-cat-limit* = *is-tm-cat-cone* α r \mathfrak{J} \mathfrak{C} \mathfrak{F} u **for** α \mathfrak{J} \mathfrak{C} \mathfrak{F} r u +
assumes *tm-cat-lim-ua-fo*:
 $\wedge u' r'. u' : r' <_{CF.cone} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C} \implies$
 $\exists ! f'. f' : r' \mapsto_{\mathfrak{C}} r \wedge u' = u \cdot_{NTCF} ntcf-const \mathfrak{J} \mathfrak{C} f'$

syntax *-is-tm-cat-limit* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
 $\langle \langle - : / - <_{CF.tm.lim} - : / - \mapsto_{C.tm1} - \rangle \rangle [51, 51, 51, 51, 51] 51$

syntax-consts *-is-tm-cat-limit* \equiv *is-tm-cat-limit*

translations $u : r <_{CF.tm.lim} \mathfrak{F} : \mathfrak{J} \mapsto_{C.tm\alpha} \mathfrak{C} \equiv$
 $CONST$ *is-tm-cat-limit* α \mathfrak{J} \mathfrak{C} \mathfrak{F} r u

locale *is-tm-cat-colimit* = *is-tm-cat-cocone* α r \mathfrak{J} \mathfrak{C} \mathfrak{F} u **for** α \mathfrak{J} \mathfrak{C} \mathfrak{F} r u +
assumes *tm-cat-colim-ua-of*:
 $\wedge u' r'. u' : \mathfrak{F} >_{CF.cocone} r' : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C} \implies$
 $\exists ! f'. f' : r \mapsto_{\mathfrak{C}} r' \wedge u' = ntcf-const \mathfrak{J} \mathfrak{C} f' \cdot_{NTCF} u$

syntax *-is-tm-cat-colimit* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
 $\langle \langle - : / - >_{CF.tm.colim} - : / - \mapsto_{C.tm1} - \rangle \rangle [51, 51, 51, 51, 51] 51$

syntax-consts *-is-tm-cat-colimit* \equiv *is-tm-cat-colimit*

translations $u : \mathfrak{F} >_{CF.tm.colim} r : \mathfrak{J} \mapsto_{C.tm\alpha} \mathfrak{C} \equiv$
 $CONST$ *is-tm-cat-colimit* α \mathfrak{J} \mathfrak{C} \mathfrak{F} r u

Rules.

lemma (in *is-tm-cat-limit*) *is-tm-cat-limit-axioms'*[*cat-lim-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $r' = r$ **and** $\mathfrak{J}' = \mathfrak{J}$ **and** $\mathfrak{C}' = \mathfrak{C}$ **and** $\mathfrak{F}' = \mathfrak{F}$
shows $u : r' <_{CF.tm.lim} \mathfrak{F}' : \mathfrak{J}' \mapsto_{C.tm\alpha'} \mathfrak{C}'$
unfolding *assms* **by** (rule *is-tm-cat-limit-axioms*)

mk-ide rf *is-tm-cat-limit-def*[*unfolded is-tm-cat-limit-axioms-def*]
 $|intro$ *is-tm-cat-limitI*
 $|dest$ *is-tm-cat-limitD*[*dest*]
 $|elim$ *is-tm-cat-limitE*[*elim*]

lemmas [*cat-lim-cs-intros*] = *is-tm-cat-limitD*(1)

lemma (in *is-tm-cat-colimit*) *is-tm-cat-colimit-axioms'*[*cat-lim-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $r' = r$ **and** $\mathfrak{J}' = \mathfrak{J}$ **and** $\mathfrak{C}' = \mathfrak{C}$ **and** $\mathfrak{F}' = \mathfrak{F}$
shows $u : \mathfrak{F}' >_{CF.tm.colim} r' : \mathfrak{J}' \mapsto_{C.tm\alpha'} \mathfrak{C}'$
unfolding *assms* **by** (rule *is-tm-cat-colimit-axioms*)

mk-ide rf *is-tm-cat-colimit-def*[*unfolded is-tm-cat-colimit-axioms-def*]
 $|intro$ *is-tm-cat-colimitI*
 $|dest$ *is-tm-cat-colimitD*[*dest*]
 $|elim$ *is-tm-cat-colimitE*[*elim*]

lemmas [*cat-lim-cs-intros*] = *is-tm-cat-colimitD*(1)

lemma *is-tm-cat-limitI'*:
assumes $u : r <_{CF.tm.cone} \mathfrak{F} : \mathfrak{J} \mapsto_{C.tm\alpha} \mathfrak{C}$
and $\wedge u' r'. u' : r' <_{CF.tm.cone} \mathfrak{F} : \mathfrak{J} \mapsto_{C.tm\alpha} \mathfrak{C} \implies$
 $\exists ! f'. f' : r' \mapsto_{\mathfrak{C}} r \wedge u' = u \cdot_{NTCF} ntcf-const \mathfrak{J} \mathfrak{C} f'$
shows $u : r <_{CF.tm.lim} \mathfrak{F} : \mathfrak{J} \mapsto_{C.tm\alpha} \mathfrak{C}$

proof(*rule is-tm-cat-limitI*, *rule assms(1)*)
interpret *is-tm-cat-cone* α r \mathfrak{J} \mathfrak{C} \mathfrak{F} u **by** (*rule assms(1)*)
fix r' u' **assume** *prems*: $u' : r' <_{CF.cone} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$
then interpret $u' : is-cat-cone$ α r' \mathfrak{J} \mathfrak{C} \mathfrak{F} u' .
have $u' : r' <_{CF.tm.cone} \mathfrak{F} : \mathfrak{J} \mapsto_{C.tm\alpha} \mathfrak{C}$
by
(
intro
is-tm-cat-coneI
NTCod.is-tm-functor-axioms
u'.cat-cone-obj
u'.is-ntcf-axioms
)
then show $\exists !f'. f' : r' \mapsto_{\mathfrak{C}} r \wedge u' = u \cdot_{NTCF} ntcf-const \mathfrak{J} \mathfrak{C} f'$
by (*rule assms(2)*)
qed

lemma *is-tm-cat-colimitI'*:
assumes $u : \mathfrak{F} >_{CF.tm.cocone} r : \mathfrak{J} \mapsto_{C.tm\alpha} \mathfrak{C}$
and $\bigwedge u' r'. u' : \mathfrak{F} >_{CF.tm.cocone} r' : \mathfrak{J} \mapsto_{C.tm\alpha} \mathfrak{C} \implies$
 $\exists !f'. f' : r \mapsto_{\mathfrak{C}} r' \wedge u' = ntcf-const \mathfrak{J} \mathfrak{C} f' \cdot_{NTCF} u$
shows $u : \mathfrak{F} >_{CF.tm.colim} r : \mathfrak{J} \mapsto_{C.tm\alpha} \mathfrak{C}$
proof(*intro is-tm-cat-colimitI*, *rule assms(1)*)
interpret *is-tm-cat-cocone* α r \mathfrak{J} \mathfrak{C} \mathfrak{F} u **by** (*rule assms(1)*)
fix r' u' **assume** *prems*: $u' : \mathfrak{F} >_{CF.cocone} r' : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$
then interpret $u' : is-cat-cocone$ α r' \mathfrak{J} \mathfrak{C} \mathfrak{F} u' .
have $u' : \mathfrak{F} >_{CF.tm.cocone} r' : \mathfrak{J} \mapsto_{C.tm\alpha} \mathfrak{C}$
by
(
intro
is-tm-cat-coconeI
NTDom.is-tm-functor-axioms
u'.cat-cocone-obj
u'.is-ntcf-axioms
)
then show $\exists !f'. f' : r \mapsto_{\mathfrak{C}} r' \wedge u' = ntcf-const \mathfrak{J} \mathfrak{C} f' \cdot_{NTCF} u$
by (*rule assms(2)*)
qed

Elementary properties.

sublocale *is-tm-cat-limit* \subseteq *is-cat-limit*
by (*intro is-cat-limitI*, *rule is-cat-cone-axioms*, *rule tm-cat-lim-ua-fo*)

sublocale *is-tm-cat-colimit* \subseteq *is-cat-colimit*
by (*intro is-cat-colimitI*, *rule is-cat-cocone-axioms*, *rule tm-cat-colim-ua-of*)

lemma (**in** *is-cat-limit*) *cat-lim-is-tm-cat-limit*:
assumes $\mathfrak{F} : \mathfrak{J} \mapsto_{C.tm\alpha} \mathfrak{C}$
shows $u : r <_{CF.tm.lim} \mathfrak{F} : \mathfrak{J} \mapsto_{C.tm\alpha} \mathfrak{C}$
proof(*intro is-tm-cat-limitI*)
interpret $\mathfrak{F} : is-tm-functor$ α \mathfrak{J} \mathfrak{C} \mathfrak{F} **by** (*rule assms*)
show $u : r <_{CF.tm.cone} \mathfrak{F} : \mathfrak{J} \mapsto_{C.tm\alpha} \mathfrak{C}$
by (*intro is-tm-cat-coneI* *assms is-ntcf-axioms cat-cone-obj*)
fix u' r' **assume** *prems*: $u' : r' <_{CF.cone} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$
show $\exists !f'. f' : r' \mapsto_{\mathfrak{C}} r \wedge u' = u \cdot_{NTCF} ntcf-const \mathfrak{J} \mathfrak{C} f'$
by (*rule cat-lim-ua-fo*[*OF prems*])
qed

lemma (in *is-cat-colimit*) *cat-colim-is-tm-cat-colimit*:
assumes $\mathfrak{F} : \mathfrak{J} \mapsto \mapsto_{C.tm\alpha} \mathfrak{C}$
shows $u : \mathfrak{F} >_{CF.tm.colim} r : \mathfrak{J} \mapsto \mapsto_{C.tm\alpha} \mathfrak{C}$
proof(intro *is-tm-cat-colimitI*)
interpret $\mathfrak{F} : is-tm-functor \alpha \mathfrak{J} \mathfrak{C} \mathfrak{F}$ **by** (*rule assms*)
show $u : \mathfrak{F} >_{CF.tm.cocone} r : \mathfrak{J} \mapsto \mapsto_{C.tm\alpha} \mathfrak{C}$
by (intro *is-tm-cat-coconeI assms is-ntcf-axioms cat-cocone-obj*)
fix $u' r'$ **assume** *prems*: $u' : \mathfrak{F} >_{CF.cocone} r' : \mathfrak{J} \mapsto \mapsto_{C\alpha} \mathfrak{C}$
show $\exists! f'. f' : r \mapsto_{\mathfrak{C}} r' \wedge u' = ntcf-const \mathfrak{J} \mathfrak{C} f' \cdot_{NTCF} u$
by (*rule cat-colim-ua-of[OF prems]*)
qed

Limits, colimits and universal arrows.

lemma (in *is-tm-cat-limit*) *tm-cat-lim-is-universal-arrow-fo*:
universal-arrow-fo ($\Delta_{CF.tm} \alpha \mathfrak{J} \mathfrak{C}$) (*cf-map* \mathfrak{F}) r (*ntcf-arrow* u)
proof(intro *is-functor.universal-arrow-foI*)

show $\Delta_{CF.tm} \alpha \mathfrak{J} \mathfrak{C} : \mathfrak{C} \mapsto \mapsto_{C\alpha} cat-Funct \alpha \mathfrak{J} \mathfrak{C}$
by
(
intro
tm-cf-diagonal-is-functor
NTCod.HomDom.tiny-category-axioms
NTDom.HomCod.category-axioms
)

show $r \in_{\circ} \mathfrak{C}(\mathit{Obj})$ **by** (*intro cat-cone-obj*)
then show *ntcf-arrow* $u : \Delta_{CF.tm} \alpha \mathfrak{J} \mathfrak{C}(\mathit{ObjMap})(\mathit{r}) \mapsto_{cat-Funct \alpha \mathfrak{J} \mathfrak{C}} \mathit{cf-map} \mathfrak{F}$
by
(
cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros*
)

fix $r' u'$ **assume** *prems*:
 $r' \in_{\circ} \mathfrak{C}(\mathit{Obj})$ $u' : \Delta_{CF.tm} \alpha \mathfrak{J} \mathfrak{C}(\mathit{ObjMap})(\mathit{r}') \mapsto_{cat-Funct \alpha \mathfrak{J} \mathfrak{C}} \mathit{cf-map} \mathfrak{F}$
from *prems*(1) **have** [*cat-cs-simps*]:
 $\mathit{cf-of-cf-map} \mathfrak{J} \mathfrak{C} (\mathit{cf-map} \mathfrak{F}) = \mathfrak{F}$
 $\mathit{cf-of-cf-map} \mathfrak{J} \mathfrak{C} (\mathit{cf-map} (\mathit{cf-const} \mathfrak{J} \mathfrak{C} r')) = \mathit{cf-const} \mathfrak{J} \mathfrak{C} r'$
by (*cs-concl cs-simp: cat-FUNCT-cs-simps cs-intro: cat-cs-intros*)+
from *prems*(2,1) **have**
 $u' : \mathit{cf-map} (\mathit{cf-const} \mathfrak{J} \mathfrak{C} r') \mapsto_{cat-Funct \alpha \mathfrak{J} \mathfrak{C}} \mathit{cf-map} \mathfrak{F}$
by (*cs-prems cs-shallow cs-simp: cat-cs-simps*)
note $u'[\mathit{unfolded cat-cs-simps}] = \mathit{cat-Funct-is-arrD}[OF \mathit{this}]$

from
tm-cat-lim-ua-fo[
OF is-cat-coneI[OF is-tm-ntcfD(1)[OF u'(1)] prems(1)]
]
obtain f
where $f : r' \mapsto_{\mathfrak{C}} r$
and [*symmetric, cat-cs-simps*]:
 $\mathit{ntcf-of-ntcf-arrow} \mathfrak{J} \mathfrak{C} u' = u \cdot_{NTCF} \mathit{ntcf-const} \mathfrak{J} \mathfrak{C} f$
and *f-unique*:
[[
 $f' : r' \mapsto_{\mathfrak{C}} r$;
 $\mathit{ntcf-of-ntcf-arrow} \mathfrak{J} \mathfrak{C} u' = u \cdot_{NTCF} \mathit{ntcf-const} \mathfrak{J} \mathfrak{C} f'$

$\mathbb{J} \implies f' = f$
for f'
by *metis*

show $\exists ! f'$.

$f' : r' \mapsto_{\mathfrak{C}} r \wedge$

$u' = \text{umap-fo } (\Delta_{CF.tm} \alpha \mathfrak{J} \mathfrak{C}) (cf\text{-map } \mathfrak{F}) r (ntcf\text{-arrow } u) r' (\text{ArrVal}) (f')$

proof(*intro ex1I conjI; (elim conjE)?*)

show $f : r' \mapsto_{\mathfrak{C}} r$ **by** (*rule f*)

with *cat-cone-obj* **show** u' -def:

$u' = \text{umap-fo } (\Delta_{CF.tm} \alpha \mathfrak{J} \mathfrak{C}) (cf\text{-map } \mathfrak{F}) r (ntcf\text{-arrow } u) r' (\text{ArrVal}) (f)$

by

(

cs-concl

cs-simp: $u'(2)[\text{symmetric}] \text{ cat-cs-simps cat-FUNCT-cs-simps}$

cs-intro: $\text{cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros}$

)

fix f' **assume** prems' :

$f' : r' \mapsto_{\mathfrak{C}} r$

$u' = \text{umap-fo } (\Delta_{CF.tm} \alpha \mathfrak{J} \mathfrak{C}) (cf\text{-map } \mathfrak{F}) r (ntcf\text{-arrow } u) r' (\text{ArrVal}) (f')$

from $\text{prems}'(2)$ f prems' *cat-cone-obj* **have** u' -def':

$u' = \text{ntcf-arrow } (u \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{C} f')$

by

(

cs-prems

cs-simp: $\text{cat-cs-simps cat-FUNCT-cs-simps}$

cs-intro: $\text{cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros}$

)

from $\text{prems}'(1)$ **have** $\text{ntcf-of-ntcf-arrow } \mathfrak{J} \mathfrak{C} u' = u \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{C} f'$

by (*cs-concl cs-simp: cat-FUNCT-cs-simps u'-def' cs-intro: cat-cs-intros*)

from f -unique[*OF prems'(1) this*] **show** $f' = f$.

qed

qed

lemma (*in is-tm-cat-cone*) *tm-cat-cone-is-tm-cat-limit:*

assumes $\text{universal-arrow-fo } (\Delta_{CF.tm} \alpha \mathfrak{J} \mathfrak{C}) (cf\text{-map } \mathfrak{F}) c (ntcf\text{-arrow } \mathfrak{N})$

shows $\mathfrak{N} : c <_{CF.tm.lim} \mathfrak{F} : \mathfrak{J} \mapsto_{C.tm} \alpha \mathfrak{C}$

proof(*intro is-tm-cat-limitI' is-tm-cat-cone-axioms*)

fix $u' c'$ **assume** $\text{prems} : u' : c' <_{CF.tm.cone} \mathfrak{F} : \mathfrak{J} \mapsto_{C.tm} \alpha \mathfrak{C}$

interpret $u' : \text{is-tm-cat-cone } \alpha c' \mathfrak{J} \mathfrak{C} \mathfrak{F} u'$ **by** (*rule prems*)

from $u'.tm\text{-cat-cone-obj}$ **have** u' -is-arr:

$\text{ntcf-arrow } u' : \Delta_{CF.tm} \alpha \mathfrak{J} \mathfrak{C} (\text{ObjMap}) (c') \mapsto_{\text{cat-Funct}} \alpha \mathfrak{J} \mathfrak{C} cf\text{-map } \mathfrak{F}$

by

(

cs-concl

cs-simp: cat-cs-simps

cs-intro: $\text{cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros}$

)

from $\text{is-functor.universal-arrow-foD}(3)$

[

OF

$\text{tm-cf-diagonal-is-functor}$ [

```

    OF NTCod.HomDom.tiny-category-axioms NTDom.HomCod.category-axioms
  ]
  assms
  u'.cat-cone-obj
  u'-is-arr
]
obtain f where f: f : c'  $\mapsto_{\mathfrak{C}}$  c
and u'-def': ntcf-arrow u' =
  umap-fo ( $\Delta_{CF.tm} \alpha \mathfrak{J} \mathfrak{C}$ ) (cf-map  $\mathfrak{F}$ ) c (ntcf-arrow  $\mathfrak{N}$ ) c' ( $\downarrow_{ArrVal}$ ) ( $\downarrow f$ )
and f'-unique:
  [[
    f' : c'  $\mapsto_{\mathfrak{C}}$  c;
    ntcf-arrow u' =
      umap-fo ( $\Delta_{CF.tm} \alpha \mathfrak{J} \mathfrak{C}$ ) (cf-map  $\mathfrak{F}$ ) c (ntcf-arrow  $\mathfrak{N}$ ) c' ( $\downarrow_{ArrVal}$ ) ( $\downarrow f'$ )
  ]]  $\implies$  f' = f
for f'
by metis

from u'-def' f cat-cone-obj have u'-def: u' =  $\mathfrak{N} \cdot_{NTCF}$  ntcf-const  $\mathfrak{J} \mathfrak{C} f$ 
by
  (
    cs-prems
    cs-simp: cat-cs-simps cat-FUNCT-cs-simps
    cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
  )

show  $\exists ! f'. f' : c' \mapsto_{\mathfrak{C}} c \wedge u' = \mathfrak{N} \cdot_{NTCF} ntcf-const \mathfrak{J} \mathfrak{C} f'$ 
proof(intro ex1I conjI; (elim conjE)?, (rule f)?, (rule u'-def)?)
fix f'' assume prems':
  f'' : c'  $\mapsto_{\mathfrak{C}}$  c u' =  $\mathfrak{N} \cdot_{NTCF} ntcf-const \mathfrak{J} \mathfrak{C} f''$ 
from prems' have
  ntcf-arrow u' =
    umap-fo ( $\Delta_{CF.tm} \alpha \mathfrak{J} \mathfrak{C}$ ) (cf-map  $\mathfrak{F}$ ) c (ntcf-arrow  $\mathfrak{N}$ ) c' ( $\downarrow_{ArrVal}$ ) ( $\downarrow f''$ )
by
  (
    cs-concl
    cs-simp: cat-cs-simps cat-FUNCT-cs-simps
    cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
  )
from f'-unique[OF prems'(1) this] show f'' = f.
qed

qed

lemma (in is-tm-cat-colimit) tm-cat-colim-is-universal-arrow-of:
  universal-arrow-of ( $\Delta_{CF.tm} \alpha \mathfrak{J} \mathfrak{C}$ ) (cf-map  $\mathfrak{F}$ ) r (ntcf-arrow u)
proof(intro is-functor.universal-arrow-ofI)

show  $\Delta_{CF.tm} \alpha \mathfrak{J} \mathfrak{C} : \mathfrak{C} \mapsto \mapsto_{C\alpha} cat-Funct \alpha \mathfrak{J} \mathfrak{C}$ 
by
  (
    intro
    tm-cf-diagonal-is-functor
    NTDom.HomDom.tiny-category-axioms
    NTDom.HomCod.category-axioms
  )

show r  $\in_{\circ} \mathfrak{C}(\downarrow Obj)$  by (intro cat-cocone-obj)

```

then show $ntcf\text{-arrow } u : cf\text{-map } \mathfrak{F} \mapsto_{cat\text{-Funct } \alpha \mathfrak{J} \mathfrak{C}} \Delta_{CF.tn} \alpha \mathfrak{J} \mathfrak{C} (ObjMap) (|r|)$
 by
 (
 cs-concl **cs-shallow**
 cs-simp: *cat-cs-simps*
 cs-intro: *cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros*
)

fix $r' u'$ assume *prems*:

$r' \in_{\circ} \mathfrak{C} (Obj) \quad u' : cf\text{-map } \mathfrak{F} \mapsto_{cat\text{-Funct } \alpha \mathfrak{J} \mathfrak{C}} \Delta_{CF.tn} \alpha \mathfrak{J} \mathfrak{C} (ObjMap) (|r'|)$

from *prems*(1) have [*cat-cs-simps*]:

$cf\text{-of-}cf\text{-map } \mathfrak{J} \mathfrak{C} (cf\text{-map } \mathfrak{F}) = \mathfrak{F}$

$cf\text{-of-}cf\text{-map } \mathfrak{J} \mathfrak{C} (cf\text{-map } (cf\text{-const } \mathfrak{J} \mathfrak{C} r')) = cf\text{-const } \mathfrak{J} \mathfrak{C} r'$

by (*cs-concl* **cs-simp**: *cat-FUNCT-cs-simps* **cs-intro**: *cat-cs-intros*) +

from *prems*(2,1) have

$u' : cf\text{-map } \mathfrak{F} \mapsto_{cat\text{-Funct } \alpha \mathfrak{J} \mathfrak{C}} cf\text{-map } (cf\text{-const } \mathfrak{J} \mathfrak{C} r')$

by (*cs-prems* **cs-shallow** **cs-simp**: *cat-cs-simps*)

note $u'[unfolded\ cat\text{-cs-simps}] = cat\text{-Funct-is-arrD}[OF\ this]$

from *cat-colim-ua-of*[*OF is-cat-coconeI*[*OF is-tm-ntcfD*(1)[*OF u'(1)*] *prems*(1)]]

obtain f

where $f : r \mapsto_{\mathfrak{C}} r'$

and [*symmetric, cat-cs-simps*]:

$ntcf\text{-of-}ntcf\text{-arrow } \mathfrak{J} \mathfrak{C} u' = ntcf\text{-const } \mathfrak{J} \mathfrak{C} f \cdot_{NTCF} u$

and *f-unique*:

$$\begin{aligned} & \llbracket \\ & \quad f' : r \mapsto_{\mathfrak{C}} r'; \\ & \quad ntcf\text{-of-}ntcf\text{-arrow } \mathfrak{J} \mathfrak{C} u' = ntcf\text{-const } \mathfrak{J} \mathfrak{C} f' \cdot_{NTCF} u \\ & \rrbracket \implies f' = f \end{aligned}$$

for f'

by *metis*

show $\exists! f'$.

$f' : r \mapsto_{\mathfrak{C}} r' \wedge$

$u' = umap\text{-of } (\Delta_{CF.tn} \alpha \mathfrak{J} \mathfrak{C}) (cf\text{-map } \mathfrak{F}) r (ntcf\text{-arrow } u) r' (|ArrVal|) (|f'|)$

proof(*intro exI conjI; (elim conjE)?*)

show $f : r \mapsto_{\mathfrak{C}} r'$ by (*rule f*)

with *cat-cocone-obj* show *u'-def*:

$u' = umap\text{-of } (\Delta_{CF.tn} \alpha \mathfrak{J} \mathfrak{C}) (cf\text{-map } \mathfrak{F}) r (ntcf\text{-arrow } u) r' (|ArrVal|) (|f|)$

by

(

cs-concl

cs-simp: $u'(2)[symmetric]$ *cat-cs-simps cat-FUNCT-cs-simps*

cs-intro: *cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros*

)

fix f' assume *prems'*:

$f' : r \mapsto_{\mathfrak{C}} r'$

$u' = umap\text{-of } (\Delta_{CF.tn} \alpha \mathfrak{J} \mathfrak{C}) (cf\text{-map } \mathfrak{F}) r (ntcf\text{-arrow } u) r' (|ArrVal|) (|f'|)$

from *prems'*(2) *f prems'* *cat-cocone-obj* have *u'-def'*:

$u' = ntcf\text{-arrow } (ntcf\text{-const } \mathfrak{J} \mathfrak{C} f' \cdot_{NTCF} u)$

by

(

cs-prems

cs-simp: *cat-cs-simps cat-FUNCT-cs-simps*

```

    cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
  )
from prems'(1) have ntcf-of-ntcf-arrow  $\mathfrak{J} \mathfrak{C} u' = \text{ntcf-const } \mathfrak{J} \mathfrak{C} f' \cdot_{NTCF} u$ 
by
  (
    cs-concl cs-shallow
    cs-simp: cat-FUNCT-cs-simps u'-def' cs-intro: cat-cs-intros
  )
from f-unique[OF prems'(1) this] show  $f' = f$  .

qed

qed

lemma (in is-tm-cat-cocone) tm-cat-cocone-is-tm-cat-colimit:
  assumes universal-arrow-of ( $\Delta_{CF.tm} \alpha \mathfrak{J} \mathfrak{C}$ ) (cf-map  $\mathfrak{F}$ ) c (ntcf-arrow  $\mathfrak{N}$ )
  shows  $\mathfrak{N} : \mathfrak{F} >_{CF.tm.colim} c : \mathfrak{J} \mapsto_{C.tm\alpha} \mathfrak{C}$ 
proof(intro is-tm-cat-colimitI' is-tm-cat-cocone-axioms)

  fix u' c' assume prems:  $u' : \mathfrak{F} >_{CF.tm.cocone} c' : \mathfrak{J} \mapsto_{C.tm\alpha} \mathfrak{C}$ 

  interpret u': is-tm-cat-cocone  $\alpha c' \mathfrak{J} \mathfrak{C} \mathfrak{F} u'$  by (rule prems)

from u'.tm-cat-cocone-obj have u'-is-arr:
  ntcf-arrow  $u' : \text{cf-map } \mathfrak{F} \mapsto_{\text{cat-Funct } \alpha \mathfrak{J} \mathfrak{C}} \Delta_{CF.tm} \alpha \mathfrak{J} \mathfrak{C} (\text{ObjMap})(c')$ 
by
  (
    cs-concl cs-shallow
    cs-simp: cat-cs-simps
    cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
  )

from is-functor.universal-arrow-ofD(3)
  [
    OF
    tm-cf-diagonal-is-functor[
      OF NTDom.HomDom.tiny-category-axioms NTDom.HomCod.category-axioms
    ]
    assms
    u'.cat-cocone-obj
    u'-is-arr
  ]
obtain f where  $f : c \mapsto_{\mathfrak{C}} c'$ 
and u'-def': ntcf-arrow  $u' =$ 
  umap-of ( $\Delta_{CF.tm} \alpha \mathfrak{J} \mathfrak{C}$ ) (cf-map  $\mathfrak{F}$ ) c (ntcf-arrow  $\mathfrak{N}$ )  $c'(\text{ArrVal})(f)$ 
and f'-unique:
  [[
     $f' : c \mapsto_{\mathfrak{C}} c'$ ;
    ntcf-arrow  $u' =$ 
    umap-of ( $\Delta_{CF.tm} \alpha \mathfrak{J} \mathfrak{C}$ ) (cf-map  $\mathfrak{F}$ ) c (ntcf-arrow  $\mathfrak{N}$ )  $c'(\text{ArrVal})(f')$ 
  ]]  $\implies f' = f$ 
for f'
by metis

from u'-def' f cat-cocone-obj have u'-def:  $u' = \text{ntcf-const } \mathfrak{J} \mathfrak{C} f \cdot_{NTCF} \mathfrak{N}$ 
by
  (
    cs-prems
  )

```

cs-simp: *cat-cs-simps cat-FUNCT-cs-simps*
cs-intro: *cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros*
)

show $\exists! f'. f' : c \mapsto_{\mathfrak{C}} c' \wedge u' = \text{ntcf-const } \mathfrak{J} \mathfrak{C} f' \cdot_{NTCF} \mathfrak{N}$
proof(*intro ex1I conjI; (elim conjE)?, (rule f)?, (rule u'-def)?*)
fix f'' **assume** *prems'*:
 $f'' : c \mapsto_{\mathfrak{C}} c' \wedge u' = \text{ntcf-const } \mathfrak{J} \mathfrak{C} f'' \cdot_{NTCF} \mathfrak{N}$
from *prems'* **have**
 $\text{ntcf-arrow } u' =$
 $\text{umap-of } (\Delta_{CF.tn} \alpha \mathfrak{J} \mathfrak{C}) (\text{cf-map } \mathfrak{F}) c (\text{ntcf-arrow } \mathfrak{N}) c' (\text{ArrVal}) (f'')$
by
(

cs-concl
cs-simp: *cat-cs-simps cat-FUNCT-cs-simps*
cs-intro: *cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros*
)

from f' -*unique*[*OF prems'(1) this*] **show** $f'' = f$.
qed

qed

Duality.

lemma (*in is-tm-cat-limit*) *is-tm-cat-colimit-op*:

$\text{op-ntcf } u : \text{op-cf } \mathfrak{F} >_{CF.tn.colim} r : \text{op-cat } \mathfrak{J} \mapsto_{\mapsto_{C.tn\alpha}} \text{op-cat } \mathfrak{C}$

proof(*intro is-tm-cat-colimitI'*)

show $\text{op-ntcf } u : \text{op-cf } \mathfrak{F} >_{CF.tn.cocone} r : \text{op-cat } \mathfrak{J} \mapsto_{\mapsto_{C.tn\alpha}} \text{op-cat } \mathfrak{C}$

by (*cs-concl cs-shallow cs-simp: cs-intro: cat-op-intros*)

fix $u' r'$ **assume** *prems*:

$u' : \text{op-cf } \mathfrak{F} >_{CF.tn.cocone} r' : \text{op-cat } \mathfrak{J} \mapsto_{\mapsto_{C.tn\alpha}} \text{op-cat } \mathfrak{C}$

interpret u' : *is-tm-cat-cocone* $\alpha r' \langle \text{op-cat } \mathfrak{J} \rangle \langle \text{op-cat } \mathfrak{C} \rangle \langle \text{op-cf } \mathfrak{F} \rangle u'$

by (*rule prems*)

from *tm-cat-lim-ua-fo*[*OF u'.is-cat-cone-op[unfolded cat-op-simps]*] **obtain** f

where $f : r' \mapsto_{\mathfrak{C}} r$

and *op-u'-def*: $\text{op-ntcf } u' = u \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{C} f$

and *f-unique*:

$\llbracket f' : r' \mapsto_{\mathfrak{C}} r; \text{op-ntcf } u' = u \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{C} f' \rrbracket \implies$
 $f' = f$

for f'

by *metis*

from *op-u'-def* **have** $\text{op-ntcf } (\text{op-ntcf } u') = \text{op-ntcf } (u \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{C} f)$

by *simp*

from *this f* **have** *u'-def*:

$u' = \text{ntcf-const } (\text{op-cat } \mathfrak{J}) (\text{op-cat } \mathfrak{C}) f \cdot_{NTCF} \text{op-ntcf } u$

by (*cs-prems cs-simp: cat-op-simps cs-intro: cat-cs-intros*)

show $\exists! f'$.

$f' : r \mapsto_{\text{op-cat } \mathfrak{C}} r' \wedge$

$u' = \text{ntcf-const } (\text{op-cat } \mathfrak{J}) (\text{op-cat } \mathfrak{C}) f' \cdot_{NTCF} \text{op-ntcf } u$

proof(*intro ex1I conjI; (elim conjE)?, (unfold cat-op-simps)?*)

fix f' **assume** *prems'*:

$f' : r \mapsto_{\mathfrak{C}} r$

$u' = \text{ntcf-const } (\text{op-cat } \mathfrak{J}) (\text{op-cat } \mathfrak{C}) f' \cdot_{NTCF} \text{op-ntcf } u$

from *prems'(2)* **have** $\text{op-ntcf } u' =$

$\text{op-ntcf } (\text{ntcf-const } (\text{op-cat } \mathfrak{J}) (\text{op-cat } \mathfrak{C}) f' \cdot_{NTCF} \text{op-ntcf } u)$

by *simp*

from *this prems'(1)* **have** $\text{op-ntcf } u' = u \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{C} f'$

by

(

$cs\text{-prems}$
cs-simp: $cat\text{-}cs\text{-simps}$ $cat\text{-}op\text{-simps}$
cs-intro: $cat\text{-}cs\text{-intros}$ $cat\text{-}op\text{-intros}$
)

from $f\text{-unique}[OF\text{ }prems'(1)\text{ }this]$ **show** $f' = f$.
qed ($intro\ u'\text{-def}\ f$)
qed

lemma (**in** $is\text{-}tm\text{-}cat\text{-}limit$) $is\text{-}tm\text{-}cat\text{-}colimit\text{-}op'[cat\text{-}op\text{-}intros]$:
assumes $\mathfrak{F}' = op\text{-}cf\ \mathfrak{F}$ **and** $\mathfrak{J}' = op\text{-}cat\ \mathfrak{J}$ **and** $\mathfrak{C}' = op\text{-}cat\ \mathfrak{C}$
shows $op\text{-}ntcf\ u : \mathfrak{F}' >_{CF.tm.colim} r : \mathfrak{J}' \mapsto_{C.tm\alpha} \mathfrak{C}'$
unfolding $assms$ **by** ($rule\ is\text{-}tm\text{-}cat\text{-}colimit\text{-}op$)

lemmas [$cat\text{-}op\text{-}intros$] = $is\text{-}tm\text{-}cat\text{-}limit.is\text{-}tm\text{-}cat\text{-}colimit\text{-}op'$

lemma (**in** $is\text{-}tm\text{-}cat\text{-}colimit$) $is\text{-}tm\text{-}cat\text{-}limit\text{-}op$:
 $op\text{-}ntcf\ u : r <_{CF.tm.lim} op\text{-}cf\ \mathfrak{F} : op\text{-}cat\ \mathfrak{J} \mapsto_{C.tm\alpha} op\text{-}cat\ \mathfrak{C}$
proof($intro\ is\text{-}tm\text{-}cat\text{-}limitI'$)
show $op\text{-}ntcf\ u : r <_{CF.tm.cone} op\text{-}cf\ \mathfrak{F} : op\text{-}cat\ \mathfrak{J} \mapsto_{C.tm\alpha} op\text{-}cat\ \mathfrak{C}$
by ($cs\text{-}concl$ **cs-shallow** **cs-simp**: $cat\text{-}op\text{-}simps$ **cs-intro**: $cat\text{-}op\text{-}intros$)
fix $u'\ r'$ **assume** $prems$:
 $u' : r' <_{CF.tm.cone} op\text{-}cf\ \mathfrak{F} : op\text{-}cat\ \mathfrak{J} \mapsto_{C.tm\alpha} op\text{-}cat\ \mathfrak{C}$
interpret u' : $is\text{-}tm\text{-}cat\text{-}cone\ \alpha\ r' \langle op\text{-}cat\ \mathfrak{J} \rangle \langle op\text{-}cat\ \mathfrak{C} \rangle \langle op\text{-}cf\ \mathfrak{F} \rangle u'$
by ($rule\ prems$)
from $tm\text{-}cat\text{-}colim\text{-}ua\text{-}of[OF\ u'.is\text{-}cat\text{-}cocone\text{-}op[unfolding\ cat\text{-}op\text{-}simps]]$ **obtain** f
where $f : f : r \mapsto_{\mathfrak{C}} r'$
and $op\text{-}u'\text{-def}$: $op\text{-}ntcf\ u' = ntcf\text{-}const\ \mathfrak{J}\ \mathfrak{C}\ f \cdot_{NTCF}\ u$
and $f\text{-unique}$:
 $[[\ f' : r \mapsto_{\mathfrak{C}} r';\ op\text{-}ntcf\ u' = ntcf\text{-}const\ \mathfrak{J}\ \mathfrak{C}\ f' \cdot_{NTCF}\ u\]] \implies f' = f$
for f'
by $metis$
from $op\text{-}u'\text{-def}$ **have** $op\text{-}ntcf\ (op\text{-}ntcf\ u') = op\text{-}ntcf\ (ntcf\text{-}const\ \mathfrak{J}\ \mathfrak{C}\ f \cdot_{NTCF}\ u)$
by $simp$
from $this\ f$ **have** $u'\text{-def}$:
 $u' = op\text{-}ntcf\ u \cdot_{NTCF}\ ntcf\text{-}const\ (op\text{-}cat\ \mathfrak{J})\ (op\text{-}cat\ \mathfrak{C})\ f$
by ($cs\text{-}prems$ **cs-simp**: $cat\text{-}op\text{-}simps$ **cs-intro**: $cat\text{-}cs\text{-}intros$)
show $\exists!f'$.
 $f' : r' \mapsto_{op\text{-}cat\ \mathfrak{C}} r \wedge$
 $u' = op\text{-}ntcf\ u \cdot_{NTCF}\ ntcf\text{-}const\ (op\text{-}cat\ \mathfrak{J})\ (op\text{-}cat\ \mathfrak{C})\ f'$
proof($intro\ ex1I\ conjI$; ($elim\ conjE$)?, ($unfold\ cat\text{-}op\text{-}simps$)?)
fix f' **assume** $prems'$:
 $f' : r \mapsto_{\mathfrak{C}} r'$
 $u' = op\text{-}ntcf\ u \cdot_{NTCF}\ ntcf\text{-}const\ (op\text{-}cat\ \mathfrak{J})\ (op\text{-}cat\ \mathfrak{C})\ f'$
from $prems'(2)$ **have** $op\text{-}ntcf\ u' =$
 $op\text{-}ntcf\ (op\text{-}ntcf\ u \cdot_{NTCF}\ ntcf\text{-}const\ (op\text{-}cat\ \mathfrak{J})\ (op\text{-}cat\ \mathfrak{C})\ f')$
by $simp$
from $this\ prems'(1)$ **have** $op\text{-}ntcf\ u' = ntcf\text{-}const\ \mathfrak{J}\ \mathfrak{C}\ f' \cdot_{NTCF}\ u$
by
(
 $cs\text{-}prems$
cs-simp: $cat\text{-}cs\text{-simps}$ $cat\text{-}op\text{-}simps$
cs-intro: $cat\text{-}cs\text{-intros}$ $cat\text{-}op\text{-}intros$
)

from $f\text{-unique}[OF\ prems'(1)\text{ }this]$ **show** $f' = f$.
qed ($intro\ u'\text{-def}\ f$)
qed

lemma (**in** $is\text{-}tm\text{-}cat\text{-}colimit$) $is\text{-}tm\text{-}cat\text{-}colimit\text{-}op'[cat\text{-}op\text{-}intros]$:

assumes $\mathfrak{F}' = \text{op-cf } \mathfrak{F}$ and $\mathfrak{J}' = \text{op-cat } \mathfrak{J}$ and $\mathfrak{C}' = \text{op-cat } \mathfrak{C}$
shows $\text{op-ntcf } u : r <_{CF.tm.lim} \mathfrak{F}' : \mathfrak{J}' \mapsto_{C.tm\alpha} \mathfrak{C}'$
unfolding *assms* by (rule *is-tm-cat-limit-op*)

lemmas [cat-op-intros] = *is-tm-cat-colimit.is-tm-cat-colimit-op'*

3.3.2 Further properties

lemma (in *is-tm-cat-limit*) *tm-cat-lim-is-tm-cat-limit-if-iso-arr*:

assumes $f : r' \mapsto_{iso\mathfrak{C}} r$

shows $u \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{C} f : r' <_{CF.tm.lim} \mathfrak{F} : \mathfrak{J} \mapsto_{C.tm\alpha} \mathfrak{C}$

proof-

note $f = \text{is-iso-arrD}(1)[OF \text{ assms}]$

from $f(1)$ interpret $u' : \text{is-tm-cat-cone } \alpha r' \mathfrak{J} \mathfrak{C} \mathfrak{F} \langle u \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{C} f \rangle$

by (cs-concl **cs-intro**: *cat-small-cs-intros cat-cs-intros*)

show *?thesis*

proof

(
intro $u'.tm\text{-cat-cone-is-tm-cat-limit}$,
rule *is-functor.universal-arrow-fo-if-universal-arrow-fo*,
rule *tm-cf-diagonal-is-functor*,
rule *NTCod.HomDom.tiny-category-axioms*,
rule *NTDom.HomCod.category-axioms*,
rule *tm-cat-lim-is-universal-arrow-fo*
)

show $f : r' \mapsto_{iso\mathfrak{C}} r$ by (rule *assms*)

from f show *ntcf-arrow* ($u \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{C} f$) =
 $\text{umap-fo } (\Delta_{CF.tm} \alpha \mathfrak{J} \mathfrak{C}) (\text{cf-map } \mathfrak{F}) r (\text{ntcf-arrow } u) r' (\text{ArrVal}) (f)$

by

(
cs-concl
cs-simp: *cat-cs-simps cat-FUNCT-cs-simps*
cs-intro: *cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros*
)

qed

qed

lemma (in *is-tm-cat-colimit*) *tm-cat-colim-is-tm-cat-colimit-if-iso-arr*:

assumes $f : r \mapsto_{iso\mathfrak{C}} r'$

shows $\text{ntcf-const } \mathfrak{J} \mathfrak{C} f \cdot_{NTCF} u : \mathfrak{F} >_{CF.tm.colim} r' : \mathfrak{J} \mapsto_{C.tm\alpha} \mathfrak{C}$

proof-

note $f = \text{is-iso-arrD}(1)[OF \text{ assms}]$

from $f(1)$ interpret u' :

is-tm-cat-cocone $\alpha r' \mathfrak{J} \mathfrak{C} \mathfrak{F} \langle \text{ntcf-const } \mathfrak{J} \mathfrak{C} f \cdot_{NTCF} u \rangle$

by (cs-concl **cs-intro**: *cat-small-cs-intros cat-cs-intros*)

from f have [*symmetric, cat-op-simps*]:

$\text{op-ntcf } (\text{ntcf-const } \mathfrak{J} \mathfrak{C} f \cdot_{NTCF} u) =$
 $\text{op-ntcf } u \cdot_{NTCF} \text{ntcf-const } (\text{op-cat } \mathfrak{J}) (\text{op-cat } \mathfrak{C}) f$

by

(
cs-concl **cs-shallow**
cs-simp: *cat-op-simps* **cs-intro**: *cat-cs-intros cat-op-intros*
)

show *?thesis*

by

(
rule *is-tm-cat-limit.is-tm-cat-colimit-op*
[

```

    OF is-tm-cat-limit.tm-cat-lim-is-tm-cat-limit-if-iso-arr[
      OF is-tm-cat-limit-op, unfolded cat-op-simps, OF assms(1)
    ],
    unfolded cat-op-simps
  ]
)
qed

```

3.4 Finite limit and finite colimit

locale *is-cat-finite-limit* =
is-cat-limit α \mathfrak{J} \mathfrak{C} \mathfrak{F} r u + *NTDom.HomDom*: *finite-category* α \mathfrak{J}
for α \mathfrak{J} \mathfrak{C} \mathfrak{F} r u

syntax *-is-cat-finite-limit* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
 $\langle \langle - : / - <_{CF.lim.fin} - : / - \mapsto_{C1} - \rangle \rangle$ [51, 51, 51, 51, 51] 51)

syntax-consts *-is-cat-finite-limit* \Leftrightarrow *is-cat-finite-limit*

translations $u : r <_{CF.lim.fin} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C} \Leftrightarrow$
CONST is-cat-finite-limit α \mathfrak{J} \mathfrak{C} \mathfrak{F} r u

locale *is-cat-finite-colimit* =
is-cat-colimit α \mathfrak{J} \mathfrak{C} \mathfrak{F} r u + *NTDom.HomDom*: *finite-category* α \mathfrak{J}
for α \mathfrak{J} \mathfrak{C} \mathfrak{F} r u

syntax *-is-cat-finite-colimit* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
 $\langle \langle - : / - >_{CF.colim.fin} - : / - \mapsto_{C1} - \rangle \rangle$ [51, 51, 51, 51, 51] 51)

syntax-consts *-is-cat-finite-colimit* \Leftrightarrow *is-cat-finite-colimit*

translations $u : \mathfrak{F} >_{CF.colim.fin} r : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C} \Leftrightarrow$
CONST is-cat-finite-colimit α \mathfrak{J} \mathfrak{C} \mathfrak{F} r u

Rules.

lemma (**in** *is-cat-finite-limit*) *is-cat-finite-limit-axioms'*[*cat-lim-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $r' = r$ **and** $\mathfrak{J}' = \mathfrak{J}$ **and** $\mathfrak{C}' = \mathfrak{C}$ **and** $\mathfrak{F}' = \mathfrak{F}$
shows $u : r' <_{CF.lim.fin} \mathfrak{F}' : \mathfrak{J}' \mapsto_{C\alpha'} \mathfrak{C}'$
unfolding *assms* **by** (rule *is-cat-finite-limit-axioms*)

mk-ide rf *is-cat-finite-limit-def*

```

|intro is-cat-finite-limitI|
|dest is-cat-finite-limitD[dest]|
|elim is-cat-finite-limitE[elim]|

```

lemmas [*cat-lim-cs-intros*] = *is-cat-finite-limitD*

lemma (**in** *is-cat-finite-colimit*)
is-cat-finite-colimit-axioms'[*cat-lim-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $r' = r$ **and** $\mathfrak{J}' = \mathfrak{J}$ **and** $\mathfrak{C}' = \mathfrak{C}$ **and** $\mathfrak{F}' = \mathfrak{F}$
shows $u : \mathfrak{F}' >_{CF.colim.fin} r' : \mathfrak{J}' \mapsto_{C\alpha'} \mathfrak{C}'$
unfolding *assms* **by** (rule *is-cat-finite-colimit-axioms*)

mk-ide rf *is-cat-finite-colimit-def*[*unfolded is-cat-colimit-axioms-def*]

```

|intro is-cat-finite-colimitI|
|dest is-cat-finite-colimitD[dest]|
|elim is-cat-finite-colimitE[elim]|

```

lemmas [*cat-lim-cs-intros*] = *is-cat-finite-colimitD*

Duality.

lemma (**in** *is-cat-finite-limit*) *is-cat-finite-colimit-op*:

$op\text{-}ntcf\ u : op\text{-}cf\ \mathfrak{F} >_{CF.colim.fin}\ r : op\text{-}cat\ \mathfrak{J} \mapsto_{C\alpha} op\text{-}cat\ \mathfrak{C}$
by
 (

- cs-concl cs-shallow*
- cs-intro:** *is-cat-finite-colimitI cat-op-intros cat-small-cs-intros*

)

lemma (in *is-cat-finite-limit*) *is-cat-finite-colimit-op'*[*cat-op-intros*]:
assumes $\mathfrak{F}' = op\text{-}cf\ \mathfrak{F}$ **and** $\mathfrak{J}' = op\text{-}cat\ \mathfrak{J}$ **and** $\mathfrak{C}' = op\text{-}cat\ \mathfrak{C}$
shows $op\text{-}ntcf\ u : \mathfrak{F}' >_{CF.colim.fin}\ r : \mathfrak{J}' \mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (rule *is-cat-finite-colimit-op*)

lemmas [*cat-op-intros*] = *is-cat-finite-limit.is-cat-finite-colimit-op'*

lemma (in *is-cat-finite-colimit*) *is-cat-finite-limit-op*:
 $op\text{-}ntcf\ u : r <_{CF.lim.fin}\ op\text{-}cf\ \mathfrak{F} : op\text{-}cat\ \mathfrak{J} \mapsto_{C\alpha} op\text{-}cat\ \mathfrak{C}$
by
 (

- cs-concl cs-shallow*
- cs-intro:** *is-cat-finite-limitI cat-op-intros cat-small-cs-intros*

)

lemma (in *is-cat-finite-colimit*) *is-cat-finite-colimit-op'*[*cat-op-intros*]:
assumes $\mathfrak{F}' = op\text{-}cf\ \mathfrak{F}$ **and** $\mathfrak{J}' = op\text{-}cat\ \mathfrak{J}$ **and** $\mathfrak{C}' = op\text{-}cat\ \mathfrak{C}$
shows $op\text{-}ntcf\ u : r <_{CF.lim.fin}\ \mathfrak{F}' : \mathfrak{J}' \mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (rule *is-cat-finite-limit-op*)

lemmas [*cat-op-intros*] = *is-cat-finite-colimit.is-cat-finite-colimit-op'*

Elementary properties.

sublocale *is-cat-finite-limit* \subseteq *is-tm-cat-limit*

by
 (

- intro*
- is-tm-cat-limitI*
- is-tm-cat-coneI*
- is-ntcf-axioms*
- cat-lim-ua-fo*
- cat-cone-obj*
- NTCod.cf-is-tm-functor-if-HomDom-finite-category*[
OF NTDom.HomDom.finite-category-axioms
]

)

sublocale *is-cat-finite-colimit* \subseteq *is-tm-cat-colimit*

by
 (

- intro*
- is-tm-cat-colimitI*
- is-tm-cat-coconeI*
- is-ntcf-axioms*
- cat-colim-ua-of*
- cat-cocone-obj*
- NTDom.cf-is-tm-functor-if-HomDom-finite-category*[
OF NTDom.HomDom.finite-category-axioms
]

)

3.5 Creation of limits

See Chapter V-1 in [9].

definition *cf-creates-limits* :: $V \Rightarrow V \Rightarrow V \Rightarrow bool$
where *cf-creates-limits* α \mathfrak{G} \mathfrak{F} =
 (
 $\forall \tau b.$
 $\tau : b <_{CF.lim} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{F}(\mathit{HomDom}) \mapsto_{C\alpha} \mathfrak{G}(\mathit{HomCod}) \longrightarrow$
 (
 (
 $\exists ! \sigma a. \exists \sigma a. \sigma a = \langle \sigma, a \rangle \wedge$
 $\sigma : a <_{CF.cone} \mathfrak{F} : \mathfrak{F}(\mathit{HomDom}) \mapsto_{C\alpha} \mathfrak{F}(\mathit{HomCod}) \wedge$
 $\tau = \mathfrak{G} \circ_{CF-NTCF} \sigma \wedge$
 $b = \mathfrak{G}(\mathit{ObjMap})(\mathit{!}a)$
) \wedge
 (
 $\forall \sigma a.$
 $\sigma : a <_{CF.cone} \mathfrak{F} : \mathfrak{F}(\mathit{HomDom}) \mapsto_{C\alpha} \mathfrak{F}(\mathit{HomCod}) \longrightarrow$
 $\tau = \mathfrak{G} \circ_{CF-NTCF} \sigma \longrightarrow$
 $b = \mathfrak{G}(\mathit{ObjMap})(\mathit{!}a) \longrightarrow$
 $\sigma : a <_{CF.lim} \mathfrak{F} : \mathfrak{F}(\mathit{HomDom}) \mapsto_{C\alpha} \mathfrak{F}(\mathit{HomCod})$
)
)
)

Rules.

context

fixes α \mathfrak{J} \mathfrak{A} \mathfrak{B} \mathfrak{G} \mathfrak{F}
assumes $\mathfrak{F} : \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{A}$
and $\mathfrak{G} : \mathfrak{G} : \mathfrak{A} \mapsto_{C\alpha} \mathfrak{B}$

begin

interpretation \mathfrak{F} : *is-functor* α \mathfrak{J} \mathfrak{A} \mathfrak{F} **by** (rule \mathfrak{F})
interpretation \mathfrak{G} : *is-functor* α \mathfrak{A} \mathfrak{B} \mathfrak{G} **by** (rule \mathfrak{G})

mk-ide rf *cf-creates-limits-def*[
where $\alpha = \alpha$ **and** $\mathfrak{F} = \mathfrak{F}$ **and** $\mathfrak{G} = \mathfrak{G}$, *unfolded cat-cs-simps*
]
 |*intro cf-creates-limitsI*|
 |*dest cf-creates-limitsD'*|
 |*elim cf-creates-limitsE'*|

end

lemmas *cf-creates-limitsD*[*dest!*] = *cf-creates-limitsD'*[*rotated 2*]
and *cf-creates-limitsE*[*elim!*] = *cf-creates-limitsE'*[*rotated 2*]

lemma *cf-creates-limitsE''*:

assumes *cf-creates-limits* α \mathfrak{G} \mathfrak{F}
and $\tau : b <_{CF.lim} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{A}$
and $\mathfrak{G} : \mathfrak{A} \mapsto_{C\alpha} \mathfrak{B}$
obtains σ r **where** $\sigma : r <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{A}$
and $\tau = \mathfrak{G} \circ_{CF-NTCF} \sigma$
and $b = \mathfrak{G}(\mathit{ObjMap})(\mathit{!}r)$

proof-

note *cfID* = *cf-creates-limitsD*[*OF assms*]
from *conjunct1*[*OF cfID*] **obtain** σ r

where $\sigma : \sigma : r <_{CF.cone} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{A}$
and τ -def: $\tau = \mathfrak{G} \circ_{CF-NTCF} \sigma$
and b -def: $b = \mathfrak{G}(\text{ObjMap})(\tau)$
by *metis*
moreover have $\sigma : r <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{A}$
by (*rule conjunct2*[*OF cflD*, *rule-format*, *OF* σ τ -def b -def])
ultimately show *?thesis* **using** *that by auto*
qed

3.6 Preservation of limits and colimits

3.6.1 Definitions and elementary properties

See Chapter V-4 in [9].

definition *cf-preserves-limits* :: $V \Rightarrow V \Rightarrow V \Rightarrow \text{bool}$

where *cf-preserves-limits* α \mathfrak{G} $\mathfrak{F} =$

(
 $\forall \sigma a.$
 $\sigma : a <_{CF.lim} \mathfrak{F} : \mathfrak{F}(\text{HomDom}) \mapsto_{C\alpha} \mathfrak{F}(\text{HomCod}) \longrightarrow$
 $\mathfrak{G} \circ_{CF-NTCF} \sigma : \mathfrak{G}(\text{ObjMap})(a) <_{CF.lim} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{F}(\text{HomDom}) \mapsto_{C\alpha} \mathfrak{G}(\text{HomCod})$
)

definition *cf-preserves-colimits* :: $V \Rightarrow V \Rightarrow V \Rightarrow \text{bool}$

where *cf-preserves-colimits* α \mathfrak{G} $\mathfrak{F} =$

(
 $\forall \sigma a.$
 $\sigma : \mathfrak{F} >_{CF.colim} a : \mathfrak{F}(\text{HomDom}) \mapsto_{C\alpha} \mathfrak{F}(\text{HomCod}) \longrightarrow$
 $\mathfrak{G} \circ_{CF-NTCF} \sigma : \mathfrak{G} \circ_{CF} \mathfrak{F} >_{CF.colim} \mathfrak{G}(\text{ObjMap})(a) : \mathfrak{F}(\text{HomDom}) \mapsto_{C\alpha} \mathfrak{G}(\text{HomCod})$
)

Rules.

context

fixes α \mathfrak{J} \mathfrak{A} \mathfrak{B} \mathfrak{G} \mathfrak{F}

assumes $\mathfrak{F} : \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{A}$

and $\mathfrak{G} : \mathfrak{G} : \mathfrak{A} \mapsto_{C\alpha} \mathfrak{B}$

begin

interpretation \mathfrak{F} : *is-functor* α \mathfrak{J} \mathfrak{A} \mathfrak{F} **by** (*rule* \mathfrak{F})

interpretation \mathfrak{G} : *is-functor* α \mathfrak{A} \mathfrak{B} \mathfrak{G} **by** (*rule* \mathfrak{G})

mk-ide rf *cf-preserves-limits-def*[

where $\alpha=\alpha$ **and** $\mathfrak{F}=\mathfrak{F}$ **and** $\mathfrak{G}=\mathfrak{G}$, *unfolded cat-cs-simps*

]

|*intro cf-preserves-limitsI*|

|*dest cf-preserves-limitsD'*|

|*elim cf-preserves-limitsE'*|

mk-ide rf *cf-preserves-colimits-def*[

where $\alpha=\alpha$ **and** $\mathfrak{F}=\mathfrak{F}$ **and** $\mathfrak{G}=\mathfrak{G}$, *unfolded cat-cs-simps*

]

|*intro cf-preserves-colimitsI*|

|*dest cf-preserves-colimitsD'*|

|*elim cf-preserves-colimitsE'*|

end

lemmas *cf-preserves-limitsD*[*dest!*] = *cf-preserves-limitsD'*[*rotated 2*]

and *cf-preserves-limitsE*[*elim!*] = *cf-preserves-limitsE'*[*rotated 2*]

lemmas $cf\text{-preserves-colimits}D[dest!] = cf\text{-preserves-colimits}D'[rotated\ 2]$
and $cf\text{-preserves-colimits}E[elim!] = cf\text{-preserves-colimits}E'[rotated\ 2]$

Duality.

lemma $cf\text{-preserves-colimits-op}[cat\text{-op-simps}]$:

assumes $\mathfrak{F} : \mathfrak{J} \mapsto \mapsto_{C\alpha} \mathfrak{A}$ **and** $\mathfrak{G} : \mathfrak{A} \mapsto \mapsto_{C\alpha} \mathfrak{B}$

shows

$cf\text{-preserves-colimits}\ \alpha\ (op\text{-cf}\ \mathfrak{G})\ (op\text{-cf}\ \mathfrak{F}) \longleftrightarrow$
 $cf\text{-preserves-limits}\ \alpha\ \mathfrak{G}\ \mathfrak{F}$

proof

interpret \mathfrak{F} : *is-functor* $\alpha\ \mathfrak{J}\ \mathfrak{A}\ \mathfrak{F}$ **by** (*rule assms(1)*)

interpret \mathfrak{G} : *is-functor* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{G}$ **by** (*rule assms(2)*)

show $cf\text{-preserves-limits}\ \alpha\ \mathfrak{G}\ \mathfrak{F}$

if $cf\text{-preserves-colimits}\ \alpha\ (op\text{-cf}\ \mathfrak{G})\ (op\text{-cf}\ \mathfrak{F})$

proof(*rule cf-preserves-limitsI, rule assms(1), rule assms(2)*)

fix $\sigma\ r$ **assume** $\sigma : r <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto \mapsto_{C\alpha} \mathfrak{A}$

then interpret σ : *is-cat-limit* $\alpha\ \mathfrak{J}\ \mathfrak{A}\ \mathfrak{F}\ r\ \sigma$.

show $\mathfrak{G} \circ_{CF\text{-}NTCF} \sigma : \mathfrak{G}(\mathit{ObjMap})(\downarrow r) <_{CF.lim} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{J} \mapsto \mapsto_{C\alpha} \mathfrak{B}$

by

(
rule is-cat-colimit.is-cat-limit-op
 [
 OF $cf\text{-preserves-colimits}D$
 [
 OF *that* $\sigma.is\text{-cat-colimit-op}\ \mathfrak{F}.is\text{-functor-op}\ \mathfrak{G}.is\text{-functor-op}$,
folded $op\text{-cf-cf-comp}\ op\text{-ntcf-cf-ntcf-comp}$
],
unfolded cat-op-simps
]
)

qed

show $cf\text{-preserves-colimits}\ \alpha\ (op\text{-cf}\ \mathfrak{G})\ (op\text{-cf}\ \mathfrak{F})$

if $cf\text{-preserves-limits}\ \alpha\ \mathfrak{G}\ \mathfrak{F}$

proof

(
rule cf-preserves-colimitsI,
rule $\mathfrak{F}.is\text{-functor-op}$,
rule $\mathfrak{G}.is\text{-functor-op}$,
unfold cat-op-simps
)

fix $\sigma\ r$ **assume** $\sigma : op\text{-cf}\ \mathfrak{F} >_{CF.colim} r : op\text{-cat}\ \mathfrak{J} \mapsto \mapsto_{C\alpha} op\text{-cat}\ \mathfrak{A}$

then interpret σ : *is-cat-colimit* $\alpha\ \langle op\text{-cat}\ \mathfrak{J} \rangle\ \langle op\text{-cat}\ \mathfrak{A} \rangle\ \langle op\text{-cf}\ \mathfrak{F} \rangle\ r\ \sigma$.

show $op\text{-cf}\ \mathfrak{G} \circ_{CF\text{-}NTCF} \sigma :$

$op\text{-cf}\ \mathfrak{G} \circ_{CF} op\text{-cf}\ \mathfrak{F} >_{CF.colim} \mathfrak{G}(\mathit{ObjMap})(\downarrow r) : op\text{-cat}\ \mathfrak{J} \mapsto \mapsto_{C\alpha} op\text{-cat}\ \mathfrak{B}$

by

(
rule is-cat-limit.is-cat-colimit-op
 [
 OF $cf\text{-preserves-limits}D$ [
 OF *that* $\sigma.is\text{-cat-limit-op}[unfolded\ cat\text{-op-simps}]$ *assms(1,2)*
],
unfolded cat-op-simps
]
)

qed

qed

lemma *cf-preserves-limits-op[cat-op-simps]*:

assumes $\mathfrak{F} : \mathfrak{J} \mapsto \mapsto_{C\alpha} \mathfrak{A}$ **and** $\mathfrak{G} : \mathfrak{A} \mapsto \mapsto_{C\alpha} \mathfrak{B}$

shows

$cf\text{-preserves-limits } \alpha \ (op\text{-cf } \mathfrak{G}) \ (op\text{-cf } \mathfrak{F}) \longleftrightarrow$
 $cf\text{-preserves-colimits } \alpha \ \mathfrak{G} \ \mathfrak{F}$

proof

interpret \mathfrak{F} : *is-functor* $\alpha \ \mathfrak{J} \ \mathfrak{A} \ \mathfrak{F}$ **by** (*rule assms(1)*)

interpret \mathfrak{G} : *is-functor* $\alpha \ \mathfrak{A} \ \mathfrak{B} \ \mathfrak{G}$ **by** (*rule assms(2)*)

show *cf-preserves-colimits* $\alpha \ \mathfrak{G} \ \mathfrak{F}$

if *cf-preserves-limits* $\alpha \ (op\text{-cf } \mathfrak{G}) \ (op\text{-cf } \mathfrak{F})$

proof(*rule cf-preserves-colimitsI, rule assms(1), rule assms(2)*)

fix $\sigma \ r$ **assume** $\sigma : \mathfrak{F} >_{CF.colim} r : \mathfrak{J} \mapsto \mapsto_{C\alpha} \mathfrak{A}$

then interpret σ : *is-cat-colimit* $\alpha \ \mathfrak{J} \ \mathfrak{A} \ \mathfrak{F} \ r \ \sigma$.

show $\mathfrak{G} \circ_{CF-NTCF} \sigma : \mathfrak{G} \circ_{CF} \mathfrak{F} >_{CF.colim} \mathfrak{G}(\mathit{ObjMap})(\mathit{|r|}) : \mathfrak{J} \mapsto \mapsto_{C\alpha} \mathfrak{B}$

by

(
 rule is-cat-limit.is-cat-colimit-op
 [
 OF cf-preserves-limitsD
 [
 OF that $\sigma.is-cat-limit-op \ \mathfrak{F}.is-functor-op \ \mathfrak{G}.is-functor-op,$
 folded op-cf-cf-comp op-ntcf-cf-ntcf-comp
],
],
 unfolded cat-op-simps
]
)

qed

show *cf-preserves-limits* $\alpha \ (op\text{-cf } \mathfrak{G}) \ (op\text{-cf } \mathfrak{F})$

if *cf-preserves-colimits* $\alpha \ \mathfrak{G} \ \mathfrak{F}$

proof

(
 rule cf-preserves-limitsI,
 rule $\mathfrak{F}.is-functor-op,$
 rule $\mathfrak{G}.is-functor-op,$
 unfold cat-op-simps
)

fix $\sigma \ r$ **assume** $\sigma : r <_{CF.lim} op\text{-cf } \mathfrak{F} : op\text{-cat } \mathfrak{J} \mapsto \mapsto_{C\alpha} op\text{-cat } \mathfrak{A}$

then interpret σ : *is-cat-limit* $\alpha \ \langle op\text{-cat } \mathfrak{J} \rangle \ \langle op\text{-cat } \mathfrak{A} \rangle \ \langle op\text{-cf } \mathfrak{F} \rangle \ r \ \sigma$.

show *op-cf* $\mathfrak{G} \circ_{CF-NTCF} \sigma :$

$\mathfrak{G}(\mathit{ObjMap})(\mathit{|r|}) <_{CF.lim} op\text{-cf } \mathfrak{G} \circ_{CF} op\text{-cf } \mathfrak{F} : op\text{-cat } \mathfrak{J} \mapsto \mapsto_{C\alpha} op\text{-cat } \mathfrak{B}$

by

(
 rule is-cat-colimit.is-cat-limit-op
 [
 OF cf-preserves-colimitsD[
 OF that $\sigma.is-cat-colimit-op[unfolded cat-op-simps] \ assms(1,2)$
],
],
 unfolded cat-op-simps
]
)

qed

qed

3.6.2 Further properties

lemma *cf-preserves-limits-if-cf-creates-limits*:

— See Theorem 2 in Chapter V-4 in [9].

assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$

and $\mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{A}$

and $\psi : b <_{CF.lim} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{B}$

and *cf-creates-limits* α \mathfrak{G} \mathfrak{F}

shows *cf-preserves-limits* α \mathfrak{G} \mathfrak{F}

proof–

interpret \mathfrak{G} : *is-functor* α \mathfrak{A} \mathfrak{B} \mathfrak{G} **by** (*rule assms(1)*)

interpret \mathfrak{F} : *is-functor* α \mathfrak{J} \mathfrak{A} \mathfrak{F} **by** (*rule assms(2)*)

interpret ψ : *is-cat-limit* α \mathfrak{J} \mathfrak{B} $\langle \mathfrak{G} \circ_{CF} \mathfrak{F} \rangle$ b ψ
by (*intro is-cat-limit.cat-lim-is-tm-cat-limit assms(3,4)*)

show *?thesis*

proof

(
 intro cf-preserves-limitsI,
 rule \mathfrak{F}.is-functor-axioms,
 rule \mathfrak{G}.is-functor-axioms
)

fix σ **assume** *prems*: $\sigma : a <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{A}$

then interpret σ : *is-cat-limit* α \mathfrak{J} \mathfrak{A} \mathfrak{F} a σ .

obtain τ A

where $\tau : \tau : A <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{A}$

and ψ -*def*: $\psi = \mathfrak{G} \circ_{CF-NTCF} \tau$

and b -*def*: $b = \mathfrak{G}(\text{ObjMap})(A)$

by

(
 rule cf-creates-limitsE''
 [
 OF
 assms(4)
 \psi.is-cat-limit-axioms
 \mathfrak{F}.is-functor-axioms
 \mathfrak{G}.is-functor-axioms
]
)

from τ **interpret** τ : *is-cat-limit* α \mathfrak{J} \mathfrak{A} \mathfrak{F} A τ .

from *cat-lim-ex-is-iso-arr*[*OF* τ .*is-cat-limit-axioms prems*] **obtain** f

where $f : f : a \mapsto_{iso\mathfrak{A}} A$ **and** σ -*def*: $\sigma = \tau \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{A} f$

by *auto*

note $f = f$ *is-iso-arrD(1)*[*OF* f]

from $f(2)$ **have** $\mathfrak{G} \circ_{CF-NTCF} \sigma : \mathfrak{G}(\text{ObjMap})(a) <_{CF.cone} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{B}$

by (*intro is-cat-coneI*)

(*cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

from σ -*def* **have** $\mathfrak{G} \circ_{CF-NTCF} \sigma = \mathfrak{G} \circ_{CF-NTCF} (\tau \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{A} f)$

by *simp*
also from $f(2)$ **have** $\dots = \psi \cdot_{NTCF} ntcf\text{-const } \mathfrak{J} \mathfrak{B} (\mathfrak{G}(\text{ArrMap})(f))$
 by (*cs-concl-step cf-ntcf-comp-ntcf-vcomp*)
 (
 cs-concl
 cs-simp: *cat-cs-simps* $\psi\text{-def}[\textit{symmetric}]$ **cs-intro:** *cat-cs-intros*
)
finally have $\mathfrak{G}\sigma : \mathfrak{G} \circ_{CF-NTCF} \sigma = \psi \cdot_{NTCF} ntcf\text{-const } \mathfrak{J} \mathfrak{B} (\mathfrak{G}(\text{ArrMap})(f))$.

show $\mathfrak{G} \circ_{CF-NTCF} \sigma : \mathfrak{G}(\text{ObjMap})(a) <_{CF.lim} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{B}$
 by
 (
 rule $\psi.\textit{cat-lim-is-cat-limit-if-is-iso-arr}$
 [
 OF $\mathfrak{G}.\textit{cf-ArrMap-is-iso-arr}[\textit{OF } f(1), \textit{folded b-def}]$,
 folded $\mathfrak{G}\sigma$
]
)
 qed

qed

3.7 Continuous and cocontinuous functor

3.7.1 Definition and elementary properties

definition *is-cf-continuous* :: $V \Rightarrow V \Rightarrow \text{bool}$
where *is-cf-continuous* $\alpha \mathfrak{G} \leftrightarrow$
 $(\forall \mathfrak{F} \mathfrak{J}. \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{G}(\text{HomDom}) \longrightarrow \textit{cf-preserves-limits } \alpha \mathfrak{G} \mathfrak{F})$

definition *is-cf-cocontinuous* :: $V \Rightarrow V \Rightarrow \text{bool}$
where *is-cf-cocontinuous* $\alpha \mathfrak{G} \leftrightarrow$
 $(\forall \mathfrak{F} \mathfrak{J}. \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{G}(\text{HomDom}) \longrightarrow \textit{cf-preserves-colimits } \alpha \mathfrak{G} \mathfrak{F})$

Rules.

context

fixes $\alpha \mathfrak{J} \mathfrak{A} \mathfrak{B} \mathfrak{G} \mathfrak{F}$

assumes $\mathfrak{G} : \mathfrak{G} : \mathfrak{A} \mapsto_{C\alpha} \mathfrak{B}$

begin

interpretation \mathfrak{G} : *is-functor* $\alpha \mathfrak{A} \mathfrak{B} \mathfrak{G}$ by (*rule* \mathfrak{G})

mk-ide rf *is-cf-continuous-def*[**where** $\alpha=\alpha$ **and** $\mathfrak{G}=\mathfrak{G}$, *unfolded cat-cs-simps*]

$| \textit{intro is-cf-continuous} I |$
 $| \textit{dest is-cf-continuous} D' |$
 $| \textit{elim is-cf-continuous} E' |$

mk-ide rf *is-cf-cocontinuous-def*[**where** $\alpha=\alpha$ **and** $\mathfrak{G}=\mathfrak{G}$, *unfolded cat-cs-simps*]

$| \textit{intro is-cf-cocontinuous} I |$
 $| \textit{dest is-cf-cocontinuous} D' |$
 $| \textit{elim is-cf-cocontinuous} E' |$

end

lemmas *is-cf-continuous* $D[\textit{dest!}] = \textit{is-cf-continuous}D'[\textit{rotated}]$

and *is-cf-continuous* $E[\textit{elim!}] = \textit{is-cf-continuous}E'[\textit{rotated}]$

lemmas *is-cf-cocontinuousD*[*dest!*] = *is-cf-cocontinuousD'*[*rotated*]
and *is-cf-cocontinuousE*[*elim!*] = *is-cf-cocontinuousE'*[*rotated*]

Duality.

lemma *is-cf-continuous-op*[*cat-op-simps*]:

assumes $\mathfrak{G} : \mathfrak{A} \mapsto \rightarrow_{C\alpha} \mathfrak{B}$

shows *is-cf-continuous* α (*op-cf* \mathfrak{G}) \leftrightarrow *is-cf-cocontinuous* α \mathfrak{G}

proof

interpret \mathfrak{G} : *is-functor* α \mathfrak{A} \mathfrak{B} \mathfrak{G} **by** (*rule assms*(1))

show *is-cf-cocontinuous* α \mathfrak{G} **if** *is-cf-continuous* α (*op-cf* \mathfrak{G})

proof(*intro is-cf-cocontinuousI*, *rule assms*)

fix $\mathfrak{J} \mathfrak{J}$ **assume** *prems'*: $\mathfrak{J} : \mathfrak{J} \mapsto \rightarrow_{C\alpha} \mathfrak{A}$

then interpret \mathfrak{J} : *is-functor* α \mathfrak{J} \mathfrak{A} \mathfrak{J} .

show *cf-preserves-colimits* α \mathfrak{G} \mathfrak{J}

by

(
rule cf-preserves-limits-op
 [
THEN iffD1,
OF
prems'
assms(1)
is-cf-continuousD[*OF that* \mathfrak{J} .*is-functor-op* \mathfrak{G} .*is-functor-op*]
]
)

qed

show *is-cf-continuous* α (*op-cf* \mathfrak{G}) **if** *is-cf-cocontinuous* α \mathfrak{G}

proof(*intro is-cf-continuousI*, *rule* \mathfrak{G} .*is-functor-op*)

fix $\mathfrak{J} \mathfrak{J}$ **assume** *prems'*: $\mathfrak{J} : \mathfrak{J} \mapsto \rightarrow_{C\alpha} \text{op-cat } \mathfrak{A}$

then interpret \mathfrak{J} : *is-functor* α \mathfrak{J} $\langle \text{op-cat } \mathfrak{A} \rangle$ \mathfrak{J} .

from that assms have *op-op-bundle*:

is-cf-cocontinuous α (*op-cf* (*op-cf* \mathfrak{G}))

op-cf (*op-cf* \mathfrak{G}) : $\mathfrak{A} \mapsto \rightarrow_{C\alpha} \mathfrak{B}$

unfolding *cat-op-simps* .

show *cf-preserves-limits* α (*op-cf* \mathfrak{G}) \mathfrak{J}

by

(
rule cf-preserves-colimits-op
 [
THEN iffD1,
OF
 \mathfrak{J} .*is-functor-axioms*
 \mathfrak{G} .*is-functor-op*
is-cf-cocontinuousD
 [
OF
op-op-bundle(1)
 \mathfrak{J} .*is-functor-op*[*unfolded cat-op-simps*]
op-op-bundle(2)
]
]
)

qed

qed

lemma *is-cf-cocontinuous-op*[*cat-op-simps*]:

assumes $\mathfrak{G} : \mathfrak{A} \mapsto \rightarrow_{C\alpha} \mathfrak{B}$

shows *is-cf-cocontinuous* α (*op-cf* \mathfrak{G}) \leftrightarrow *is-cf-continuous* α \mathfrak{G}

proof

interpret \mathfrak{G} : *is-functor* α \mathfrak{A} \mathfrak{B} \mathfrak{G} **by** (*rule* *assms(1)*)

show *is-cf-continuous* α \mathfrak{G} **if** *is-cf-cocontinuous* α (*op-cf* \mathfrak{G})

proof(*intro* *is-cf-continuousI*, *rule* *assms*)

fix \mathfrak{F} \mathfrak{J} **assume** *prems'*: $\mathfrak{F} : \mathfrak{J} \mapsto \mapsto_{C\alpha} \mathfrak{A}$

then interpret \mathfrak{F} : *is-functor* α \mathfrak{J} \mathfrak{A} \mathfrak{F} .

show *cf-preserves-limits* α \mathfrak{G} \mathfrak{F}

by

(

rule *cf-preserves-colimits-op*

[

THEN *iffD1*,

OF

prems'

assms(1)

is-cf-cocontinuousD[*OF* *that* \mathfrak{F} .*is-functor-op* \mathfrak{G} .*is-functor-op*]

]

)

qed

show *is-cf-cocontinuous* α (*op-cf* \mathfrak{G}) **if** *is-cf-continuous* α \mathfrak{G}

proof(*intro* *is-cf-cocontinuousI*, *rule* \mathfrak{G} .*is-functor-op*)

fix \mathfrak{F} \mathfrak{J} **assume** *prems'*: $\mathfrak{F} : \mathfrak{J} \mapsto \mapsto_{C\alpha}$ *op-cat* \mathfrak{A}

then interpret \mathfrak{F} : *is-functor* α \mathfrak{J} \langle *op-cat* $\mathfrak{A}\rangle$ \mathfrak{F} .

from *that* *assms* **have** *op-op-bundle*:

is-cf-continuous α (*op-cf* (*op-cf* \mathfrak{G}))

op-cf (*op-cf* \mathfrak{G}) : $\mathfrak{A} \mapsto \mapsto_{C\alpha} \mathfrak{B}$

unfolding *cat-op-simps* .

show *cf-preserves-colimits* α (*op-cf* \mathfrak{G}) \mathfrak{F}

by

(

rule *cf-preserves-limits-op*

[

THEN *iffD1*,

OF

\mathfrak{F} .*is-functor-axioms*

\mathfrak{G} .*is-functor-op*

is-cf-continuousD

[

OF

op-op-bundle(1)

\mathfrak{F} .*is-functor-op*[*unfolded* *cat-op-simps*]

op-op-bundle(2)

]

]

)

qed

qed

3.7.2 Category isomorphisms are continuous and cocontinuous

lemma (*in* *is-iso-functor*) *iso-cf-is-cf-continuous*: *is-cf-continuous* α \mathfrak{F}

proof(*intro* *is-cf-continuousI*)

fix \mathfrak{J} \mathfrak{G} **assume** *prems*: $\mathfrak{G} : \mathfrak{J} \mapsto \mapsto_{C\alpha} \mathfrak{A}$

then interpret \mathfrak{G} : *is-functor* α \mathfrak{J} \mathfrak{A} \mathfrak{G} .

show *cf-preserves-limits* α \mathfrak{F} \mathfrak{G}

proof(*intro* *cf-preserves-limitsI*)

fix a σ **assume** $\sigma : a <_{CF.lim} \mathfrak{G} : \mathfrak{J} \mapsto \mapsto_{C\alpha} \mathfrak{A}$

then interpret σ : *is-cat-limit* α \mathfrak{J} \mathfrak{A} \mathfrak{G} a σ .

show $\mathfrak{F} \circ_{CF-NTCF} \sigma : \mathfrak{F}(\text{ObjMap})(\downarrow a) <_{CF.lim} \mathfrak{F} \circ_{CF} \mathfrak{G} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{B}$
proof(*intro is-cat-limitI*)
fix $r' \tau$ **assume** $\tau : r' <_{CF.cone} \mathfrak{F} \circ_{CF} \mathfrak{G} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{B}$
then interpret $\tau : is-cat-cone \alpha r' \mathfrak{J} \mathfrak{B} \langle \mathfrak{F} \circ_{CF} \mathfrak{G} \rangle \tau$.
note [*cat-cs-simps*] = *cf-comp-assoc-helper*[
 where $\mathfrak{H} = \langle inv-cf \mathfrak{F} \rangle$ **and** $\mathfrak{G} = \mathfrak{F}$ **and** $\mathfrak{F} = \mathfrak{G}$ **and** $\mathcal{Q} = \langle cf-id \mathfrak{A} \rangle$
])
have $inv-\tau : inv-cf \mathfrak{F} \circ_{CF-NTCF} \tau :$
 $inv-cf \mathfrak{F}(\text{ObjMap})(\downarrow r') <_{CF.cone} \mathfrak{G} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{A}$
by
(
 cs-concl
 cs-simp: *cat-cs-simps cf-cs-simps*
 cs-intro: *cat-cs-intros cf-cs-intros*
))
from *is-cat-limit.cat-lim-unique-cone'*[*OF* $\sigma.is-cat-limit-axioms inv-\tau$]
obtain f **where** $f : f : inv-cf \mathfrak{F}(\text{ObjMap})(\downarrow r') \mapsto_{\mathfrak{A}} a$
and $f-up : \bigwedge j. j \in_{\circ} \mathfrak{J}(\text{Obj}) \implies$
 $(inv-cf \mathfrak{F} \circ_{CF-NTCF} \tau)(\downarrow NTMap)(\downarrow j) = \sigma(\downarrow NTMap)(\downarrow j) \circ_{A\mathfrak{A}} f$
and $f-unique :$
[[
 $f' : inv-cf \mathfrak{F}(\text{ObjMap})(\downarrow r') \mapsto_{\mathfrak{A}} a ;$
 $\bigwedge j. j \in_{\circ} \mathfrak{J}(\text{Obj}) \implies$
 $(inv-cf \mathfrak{F} \circ_{CF-NTCF} \tau)(\downarrow NTMap)(\downarrow j) = \sigma(\downarrow NTMap)(\downarrow j) \circ_{A\mathfrak{A}} f'$
]] $\implies f' = f$
for f'
by *metis*
have [*cat-cs-simps*]: $\mathfrak{F}(\downarrow ArrMap)(\downarrow \sigma(\downarrow NTMap)(\downarrow j)) \circ_{A\mathfrak{B}} \mathfrak{F}(\downarrow ArrMap)(\downarrow f) = \tau(\downarrow NTMap)(\downarrow j)$
if $j \in_{\circ} \mathfrak{J}(\text{Obj})$ **for** j
proof-
from $f-up$ [*OF that*] **that have**
 $inv-cf \mathfrak{F}(\downarrow ArrMap)(\downarrow \tau(\downarrow NTMap)(\downarrow j)) = \sigma(\downarrow NTMap)(\downarrow j) \circ_{A\mathfrak{A}} f$
by (*cs-prems cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)
then have
 $\mathfrak{F}(\downarrow ArrMap)(\downarrow inv-cf \mathfrak{F}(\downarrow ArrMap)(\downarrow \tau(\downarrow NTMap)(\downarrow j))) =$
 $\mathfrak{F}(\downarrow ArrMap)(\downarrow \sigma(\downarrow NTMap)(\downarrow j) \circ_{A\mathfrak{A}} f)$
by *simp*
from *this that f show ?thesis*
by
(
 cs-prems cs-shallow
 cs-simp: *cat-cs-simps cf-cs-simps cs-intro: cat-cs-intros*
))
simp
qed
show $\exists ! f'$.
 $f' : r' \mapsto_{\mathfrak{B}} \mathfrak{F}(\text{ObjMap})(\downarrow a) \wedge \tau = \mathfrak{F} \circ_{CF-NTCF} \sigma \cdot_{NTCF} ntcf-const \mathfrak{J} \mathfrak{B} f'$
proof(*intro ex1I conjI; (elim conjE) ?*)
from f **have**
 $\mathfrak{F}(\downarrow ArrMap)(\downarrow f) : \mathfrak{F}(\text{ObjMap})(\downarrow inv-cf \mathfrak{F}(\text{ObjMap})(\downarrow r')) \mapsto_{\mathfrak{B}} \mathfrak{F}(\text{ObjMap})(\downarrow a)$
by (*cs-concl cs-shallow cs-intro: cat-cs-intros*)
then show $\mathfrak{F}(\downarrow ArrMap)(\downarrow f) : r' \mapsto_{\mathfrak{B}} \mathfrak{F}(\text{ObjMap})(\downarrow a)$
by (*cs-prems cs-shallow cs-simp: cf-cs-simps cs-intro: cat-cs-intros*)
show $\tau = \mathfrak{F} \circ_{CF-NTCF} \sigma \cdot_{NTCF} ntcf-const \mathfrak{J} \mathfrak{B} (\mathfrak{F}(\downarrow ArrMap)(\downarrow f))$
proof(*rule ntcf-eqI, rule \tau.is-ntcf-axioms*)
from f **show** $\mathfrak{F} \circ_{CF-NTCF} \sigma \cdot_{NTCF} ntcf-const \mathfrak{J} \mathfrak{B} (\mathfrak{F}(\downarrow ArrMap)(\downarrow f)) :$
 $cf-const \mathfrak{J} \mathfrak{B} r' \mapsto_{CF} \mathfrak{F} \circ_{CF} \mathfrak{G} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{B}$
by

(

 cs-concl

 cs-simp: *cat-cs-simps cf-cs-simps cs-intro:* *cat-cs-intros*

)

then have *dom-rhs*:

$$\mathcal{D}_\circ ((\mathfrak{F} \circ_{CF-NTCF} \sigma \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{B} (\mathfrak{F}(\text{ArrMap})(f)))(\text{NTMap})) = \mathfrak{J}(\text{Obj})$$

by (*cs-concl cs-simp:* *cat-cs-simps*)

show

 $\tau(\text{NTMap}) = (\mathfrak{F} \circ_{CF-NTCF} \sigma \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{B} (\mathfrak{F}(\text{ArrMap})(f)))(\text{NTMap})$

proof(*rule vsv-eqI, unfold* $\tau.\text{ntcf-NTMap-vdomain dom-rhs}$)

fix *j* **assume** $j \in_\circ \mathfrak{J}(\text{Obj})$

with *f* **show** $\tau(\text{NTMap})(j) =$

 $(\mathfrak{F} \circ_{CF-NTCF} \sigma \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{B} (\mathfrak{F}(\text{ArrMap})(f)))(\text{NTMap})(j)$

by (*cs-concl cs-simp:* *cat-cs-simps cs-intro:* *cat-cs-intros*)

qed (*cs-concl cs-intro:* *V-cs-intros cat-cs-intros*)+

qed *simp-all*

fix *f'* **assume** *prems'*:

$$f' : r' \mapsto_{\mathfrak{B}} \mathfrak{F}(\text{ObjMap})(a)$$

$$\tau = \mathfrak{F} \circ_{CF-NTCF} \sigma \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{B} f'$$

have $\tau j\text{-def}$: $\tau(\text{NTMap})(j) = \mathfrak{F}(\text{ArrMap})(\sigma(\text{NTMap})(j)) \circ_{A\mathfrak{B}} f'$

if $j \in_\circ \mathfrak{J}(\text{Obj})$ **for** *j*

proof-

from *prems'(2)* **have**

 $\tau(\text{NTMap})(j) = (\mathfrak{F} \circ_{CF-NTCF} \sigma \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{B} f')(\text{NTMap})(j)$

by *simp*

from *this prems'(1)* **that show** *?thesis*

by (*cs-prems cs-simp:* *cat-cs-simps cs-intro:* *cat-cs-intros*)

qed

have *inv-cf* $\mathfrak{F}(\text{ArrMap})(f') = f$

proof(*rule f-unique*)

from *prems'(1)* **show**

 $\text{inv-cf } \mathfrak{F}(\text{ArrMap})(f') : \text{inv-cf } \mathfrak{F}(\text{ObjMap})(r') \mapsto_{\mathfrak{A}} a$

by

 (

cs-concl

 cs-simp: *cf-cs-simps cs-intro:* *cat-cs-intros cf-cs-intros*

)

fix *j* **assume** $j \in_\circ \mathfrak{J}(\text{Obj})$

from *this prems'(1)* **show**

 $(\text{inv-cf } \mathfrak{F} \circ_{CF-NTCF} \tau)(\text{NTMap})(j) =$

 $\sigma(\text{NTMap})(j) \circ_{A\mathfrak{A}} \text{inv-cf } \mathfrak{F}(\text{ArrMap})(f')$

by

 (

cs-concl

 cs-simp: *cat-cs-simps cf-cs-simps* $\tau j\text{-def}$

 cs-intro: *cat-cs-intros cf-cs-intros*

)

qed

then have $\mathfrak{F}(\text{ArrMap})(\text{inv-cf } \mathfrak{F}(\text{ArrMap})(f')) = \mathfrak{F}(\text{ArrMap})(f)$ **by** *simp*

from *this prems'(1)* **show** $f' = \mathfrak{F}(\text{ArrMap})(f)$

by

 (

cs-prems cs-shallow

 cs-simp: *cat-cs-simps cf-cs-simps cs-intro:* *cat-cs-intros*

)

qed

qed (*cs-concl cs-intro:* *cat-cs-intros cat-lim-cs-intros*)

qed (intro prems is-functor-axioms)+
 qed (rule is-functor-axioms)

lemma (in is-iso-functor) iso-cf-is-cf-cocontinuous: is-cf-cocontinuous α \mathfrak{F}
 using is-iso-functor.iso-cf-is-cf-continuous[OF is-iso-functor-op]
 by (cs-prems cs-shallow cs-simp: cat-op-simps cs-intro: cat-cs-intros)

3.8 Tiny-continuous and tiny-cocontinuous functor

3.8.1 Definition and elementary properties

definition is-tm-cf-continuous :: $V \Rightarrow V \Rightarrow bool$
 where is-tm-cf-continuous α $\mathfrak{G} =$
 ($\forall \mathfrak{J} \mathfrak{J}'. \mathfrak{J} : \mathfrak{J} \mapsto \mapsto_{C.tm\alpha} \mathfrak{G}(HomDom) \longrightarrow cf-preserves-limits \alpha \mathfrak{G} \mathfrak{J}$)

Rules.

context

fixes $\alpha \mathfrak{J} \mathfrak{A} \mathfrak{B} \mathfrak{G} \mathfrak{F}$
 assumes $\mathfrak{G}: \mathfrak{G} : \mathfrak{A} \mapsto \mapsto_{C\alpha} \mathfrak{B}$

begin

interpretation $\mathfrak{G}: is-functor \alpha \mathfrak{A} \mathfrak{B} \mathfrak{G}$ by (rule \mathfrak{G})

mk-ide rf is-tm-cf-continuous-def[where $\alpha=\alpha$ and $\mathfrak{G}=\mathfrak{G}$, unfolded cat-cs-simps]
 |intro is-tm-cf-continuousI|
 |dest is-tm-cf-continuousD'|
 |elim is-tm-cf-continuousE'|

end

lemmas is-tm-cf-continuousD[dest!] = is-tm-cf-continuousD'[rotated]
 and is-tm-cf-continuousE[elim!] = is-tm-cf-continuousE'[rotated]

Elementary properties.

lemma (in is-functor) cf-continuous-is-tm-cf-continuous:

assumes is-cf-continuous α \mathfrak{F}
 shows is-tm-cf-continuous α \mathfrak{F}

proof(intro is-tm-cf-continuousI, rule is-functor-axioms)

fix $\mathfrak{J}' \mathfrak{J}$ assume $\mathfrak{J}' : \mathfrak{J} \mapsto \mapsto_{C.tm\alpha} \mathfrak{A}$
 then interpret $\mathfrak{J}': is-tm-functor \alpha \mathfrak{J} \mathfrak{A} \mathfrak{J}'$.
 show cf-preserves-limits α $\mathfrak{F} \mathfrak{J}'$

by

(
 intro is-cf-continuousD[OF assms(1) - is-functor-axioms],
 rule $\mathfrak{J}'.is-functor-axioms$
)

qed

4 Initial and terminal objects as limits and colimits

4.1 Initial and terminal objects as limits/colimits of an empty diagram

4.1.1 Definition and elementary properties

See [1]², [1]³ and Chapter X-1 in [9].

locale *is-cat-obj-empty-terminal* = *is-cat-limit* α *cat-0* \mathfrak{C} \langle cf-0 \mathfrak{C} \rangle z \mathfrak{J}
for α \mathfrak{C} z \mathfrak{J}

syntax *is-cat-obj-empty-terminal* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow \text{bool}$
 \langle \langle - :/ - \langle $C_{F.1}$ $0_{CF} :/ 0_C \mapsto \mapsto_{C^1}$ - \rangle [51, 51] 51 \rangle

syntax-consts *is-cat-obj-empty-terminal* \equiv *is-cat-obj-empty-terminal*

translations $\mathfrak{J} : z \langle$ $C_{F.1}$ $0_{CF} : 0_C \mapsto \mapsto_{C\alpha}$ $\mathfrak{C} \equiv$
CONST is-cat-obj-empty-terminal α \mathfrak{C} z \mathfrak{J}

locale *is-cat-obj-empty-initial* = *is-cat-colimit* α *cat-0* \mathfrak{C} \langle cf-0 \mathfrak{C} \rangle z \mathfrak{J}
for α \mathfrak{C} z \mathfrak{J}

syntax *is-cat-obj-empty-initial* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow \text{bool}$
 \langle \langle - :/ $0_{CF} >_{CF.0}$ - :/ $0_C \mapsto \mapsto_{C^1}$ - \rangle [51, 51] 51 \rangle

syntax-consts *is-cat-obj-empty-initial* \equiv *is-cat-obj-empty-initial*

translations $\mathfrak{J} : 0_{CF} >_{CF.0}$ $z : 0_C \mapsto \mapsto_{C\alpha}$ $\mathfrak{C} \equiv$
CONST is-cat-obj-empty-initial α \mathfrak{C} z \mathfrak{J}

Rules.

lemma (in *is-cat-obj-empty-terminal*)
is-cat-obj-empty-terminal-axioms'[*cat-lim-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $z' = z$ **and** $\mathfrak{C}' = \mathfrak{C}$
shows $\mathfrak{J} : z' \langle$ $C_{F.1}$ $0_{CF} : 0_C \mapsto \mapsto_{C\alpha'}$ \mathfrak{C}'
unfolding *assms* **by** (rule *is-cat-obj-empty-terminal-axioms*)

mk-ide rf *is-cat-obj-empty-terminal-def*
|intro *is-cat-obj-empty-terminalI*||
|dest *is-cat-obj-empty-terminalD*[*dest*]|
|elim *is-cat-obj-empty-terminalE*[*elim*]|

lemmas [*cat-lim-cs-intros*] = *is-cat-obj-empty-terminalD*

lemma (in *is-cat-obj-empty-initial*)
is-cat-obj-empty-initial-axioms'[*cat-lim-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $z' = z$ **and** $\mathfrak{C}' = \mathfrak{C}$
shows $\mathfrak{J} : 0_{CF} >_{CF.0}$ $z' : 0_C \mapsto \mapsto_{C\alpha'}$ \mathfrak{C}'
unfolding *assms* **by** (rule *is-cat-obj-empty-initial-axioms*)

mk-ide rf *is-cat-obj-empty-initial-def*
|intro *is-cat-obj-empty-initialI*||
|dest *is-cat-obj-empty-initialD*[*dest*]|
|elim *is-cat-obj-empty-initialE*[*elim*]|

lemmas [*cat-lim-cs-intros*] = *is-cat-obj-empty-initialD*

Duality.

lemma (in *is-cat-obj-empty-terminal*) *is-cat-obj-empty-initial-op*:
op-ntcf $\mathfrak{J} : 0_{CF} >_{CF.0}$ $z : 0_C \mapsto \mapsto_{C\alpha}$ *op-cat* \mathfrak{C}

²<https://ncatlab.org/nlab/show/initial+object>

³<https://ncatlab.org/nlab/show/terminal+object>

by (intro is-cat-obj-empty-initialI)
 (
 cs-concl **cs-shallow**
 cs-simp: cat-op-simps op-cf-cf-0 **cs-intro**: cat-cs-intros cat-op-intros
)

lemma (in is-cat-obj-empty-terminal) is-cat-obj-empty-initial-op'[cat-op-intros]:
 assumes $\mathfrak{C}' = \text{op-cat } \mathfrak{C}$
 shows op-ntcf $\mathfrak{J} : 0_{CF} >_{CF.0} z : 0_C \mapsto_{C\alpha} \mathfrak{C}'$
 unfolding assms by (rule is-cat-obj-empty-initial-op)

lemmas [cat-op-intros] = is-cat-obj-empty-terminal.is-cat-obj-empty-initial-op'

lemma (in is-cat-obj-empty-initial) is-cat-obj-empty-terminal-op:
 op-ntcf $\mathfrak{J} : z <_{CF.1} 0_{CF} : 0_C \mapsto_{C\alpha} \text{op-cat } \mathfrak{C}$
 by (intro is-cat-obj-empty-terminalI)
 (
 cs-concl **cs-shallow**
 cs-simp: cat-op-simps op-cf-cf-0 **cs-intro**: cat-cs-intros cat-op-intros
)

lemma (in is-cat-obj-empty-initial) is-cat-obj-empty-terminal-op'[cat-op-intros]:
 assumes $\mathfrak{C}' = \text{op-cat } \mathfrak{C}$
 shows op-ntcf $\mathfrak{J} : z <_{CF.1} 0_{CF} : 0_C \mapsto_{C\alpha} \mathfrak{C}'$
 unfolding assms by (rule is-cat-obj-empty-terminal-op)

lemmas [cat-op-intros] = is-cat-obj-empty-initial.is-cat-obj-empty-terminal-op'

Elementary properties.

lemma (in is-cat-obj-empty-terminal) cat-oet-ntcf-0: $\mathfrak{J} = \text{ntcf-0 } \mathfrak{C}$
 by (rule is-ntcf-is-ntcf-0-if-cat-0)
 (cs-concl **cs-shallow cs-simp**: cat-cs-simps **cs-intro**: cat-cs-intros)

lemma (in is-cat-obj-empty-initial) cat-oei-ntcf-0: $\mathfrak{J} = \text{ntcf-0 } \mathfrak{C}$
 by (rule is-ntcf-is-ntcf-0-if-cat-0)
 (cs-concl **cs-shallow cs-simp**: cat-cs-simps **cs-intro**: cat-cs-intros)

4.1.2 Initial and terminal objects as limits/colimits of an empty diagram are initial and terminal objects

lemma (in category) cat-obj-terminal-is-cat-obj-empty-terminal:
 assumes obj-terminal $\mathfrak{C} z$
 shows ntcf-0 $\mathfrak{C} : z <_{CF.1} 0_{CF} : 0_C \mapsto_{C\alpha} \mathfrak{C}$
 proof-

from assms have $z : z \in_{\circ} \mathfrak{C}(\text{Obj})$ by auto
 from z have [cat-cs-simps]: cf-const cat-0 $\mathfrak{C} z = \text{cf-0 } \mathfrak{C}$
 by (intro is-functor-is-cf-0-if-cat-0) (cs-concl **cs-intro**: cat-cs-intros)
 note obj-terminalD = obj-terminalD[OF assms]

show ?thesis

proof

(
 intro is-cat-obj-empty-terminalI is-cat-limitI is-cat-coneI,
 unfold cat-cs-simps
)

show $\exists ! f'. f' : r' \mapsto_{\mathfrak{C}} z \wedge u' = \text{ntcf-0 } \mathfrak{C} \cdot_{NTCF} \text{ntcf-const cat-0 } \mathfrak{C} f'$
 if $u' : r' <_{CF.cone} \text{cf-0 } \mathfrak{C} : \text{cat-0 } \mapsto_{C\alpha} \mathfrak{C}$ for $u' r'$

proof-
interpret u' : *is-cat-cone* α r' *cat-0* \mathfrak{C} \langle *cf-0* \mathfrak{C} \rangle u' **by** (*rule that*)
from z **have** [*cat-cs-simps*]: *cf-const* *cat-0* \mathfrak{C} $r' = \text{cf-0 } \mathfrak{C}$
by (*intro is-functor-is-cf-0-if-cat-0*)
(*cs-concl* **cs-shallow** **cs-intro**: *cat-cs-intros*)
have u' -*def*: $u' = \text{ntcf-0 } \mathfrak{C}$
by
(
rule is-ntcf-is-ntcf-0-if-cat-0
OF $u'.\text{is-ntcf-axioms}$, *unfolded cat-cs-simps*
)
from *obj-terminalD*(2)[*OF* $u'.\text{cat-cone-obj}$] **obtain** f'
where f' : $f' : r' \mapsto_{\mathfrak{C}} z$
and f' -*unique*: $f'' : r' \mapsto_{\mathfrak{C}} z \implies f'' = f'$
for f''
by *auto*
from f' **have** [*cat-cs-simps*]: *ntcf-const* *cat-0* \mathfrak{C} $f' = \text{ntcf-0 } \mathfrak{C}$
by (*intro is-ntcf-is-ntcf-0-if-cat-0*(1))
(*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
show *?thesis*
proof(*intro ex1I conjI*; (*elim conjE*)?)
show $u' = \text{ntcf-0 } \mathfrak{C} \cdot_{NTCF} \text{ntcf-const } \text{cat-0 } \mathfrak{C} f'$
by
(
cs-concl **cs-shallow**
cs-simp: u' -*def* *cat-cs-simps* **cs-intro**: *cat-cs-intros*
)
fix f'' **assume** *prems*:
 $f'' : r' \mapsto_{\mathfrak{C}} z$ $u' = \text{ntcf-0 } \mathfrak{C} \cdot_{NTCF} \text{ntcf-const } \text{cat-0 } \mathfrak{C} f''$
show $f'' = f'$ **by** (*rule f'-unique*[*OF* *prems*(1)])
qed (*rule f'*)
qed
qed (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: z *cat-cs-intros*)

qed

lemma (*in category*) *cat-obj-initial-is-cat-obj-empty-initial*:

assumes *obj-initial* \mathfrak{C} z

shows *ntcf-0* $\mathfrak{C} : 0_{CF} >_{CF.0} z : 0_C \mapsto_{C\alpha} \mathfrak{C}$

proof-

have z : *obj-terminal* (*op-cat* \mathfrak{C}) z **unfolding** *cat-op-simps* **by** (*rule assms*)

show *?thesis*

by

(
rule is-cat-obj-empty-terminal.is-cat-obj-empty-initial-op
[
OF *category.cat-obj-terminal-is-cat-obj-empty-terminal*[
OF *category-op* z , *folded op-ntcf-ntcf-0*
],
unfolded cat-op-simps op-ntcf-ntcf-0
]
)
qed

lemma (*in is-cat-obj-empty-terminal*) *cat-oet-obj-terminal*: *obj-terminal* \mathfrak{C} z

proof-

show *obj-terminal* \mathfrak{C} z

```

proof(rule obj-terminalI)
  fix a assume prems: a  $\in_0 \mathfrak{C}(\text{Obj})$ 
  have [cat-cs-simps]: cf-const cat-0  $\mathfrak{C}$  a = cf-0  $\mathfrak{C}$ 
    by (rule is-functor-is-cf-0-if-cat-0)
      (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros prems)
  from prems have ntcf-0  $\mathfrak{C}$  : a  $<_{CF.cone}$  cf-0  $\mathfrak{C}$  : cat-0  $\mapsto_{\mapsto} C\alpha$   $\mathfrak{C}$ 
    by (intro is-cat-coneI)
      (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  from cat-lim-ua-fo[OF this] obtain f'
    where f': f' : a  $\mapsto_{\mathfrak{C}}$  z
      and ntcf-0  $\mathfrak{C}$  =  $\exists \cdot_{NTCF}$  ntcf-const cat-0  $\mathfrak{C}$  f'
      and f'-unique:
        [[ f'' : a  $\mapsto_{\mathfrak{C}}$  z; ntcf-0  $\mathfrak{C}$  =  $\exists \cdot_{NTCF}$  ntcf-const cat-0  $\mathfrak{C}$  f'' ]]  $\implies$ 
          f'' = f'
    for f''
    by metis
  show  $\exists !f'. f' : a \mapsto_{\mathfrak{C}} z$ 
  proof(intro exI)
    fix f'' assume prems': f'' : a  $\mapsto_{\mathfrak{C}}$  z
    from prems' have ntcf-0  $\mathfrak{C}$  = ntcf-0  $\mathfrak{C}$   $\cdot_{NTCF}$  ntcf-const cat-0  $\mathfrak{C}$  f''
      by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    from f'-unique[OF prems', unfolded cat-oet-ntcf-0, OF this]
    show f'' = f'.
  qed (rule f')
  qed (rule cat-cone-obj)
qed

```

```

lemma (in is-cat-obj-empty-initial) cat-oei-obj-initial: obj-initial  $\mathfrak{C}$  z
  by
  (
    rule is-cat-obj-empty-terminal.cat-oet-obj-terminal[
      OF is-cat-obj-empty-initial.is-cat-obj-empty-terminal-op[
        OF is-cat-obj-empty-initial-axioms
      ],
      unfolded cat-op-simps
    ]
  )

```

```

lemma (in category) cat-is-cat-obj-empty-terminal-obj-terminal-iff:
  (ntcf-0  $\mathfrak{C}$  : z  $<_{CF.1}$  0CF : 0C  $\mapsto_{\mapsto} C\alpha$   $\mathfrak{C}$ )  $\longleftrightarrow$  obj-terminal  $\mathfrak{C}$  z
  using
    cat-obj-terminal-is-cat-obj-empty-terminal
    is-cat-obj-empty-terminal.cat-oet-obj-terminal
  by auto

```

```

lemma (in category) cat-is-cat-obj-empty-initial-obj-initial-iff:
  (ntcf-0  $\mathfrak{C}$  : 0CF  $>_{CF.0}$  z : 0C  $\mapsto_{\mapsto} C\alpha$   $\mathfrak{C}$ )  $\longleftrightarrow$  obj-initial  $\mathfrak{C}$  z
  using
    cat-obj-initial-is-cat-obj-empty-initial
    is-cat-obj-empty-initial.cat-oei-obj-initial
  by auto

```

4.2 Initial cone and terminal cocone

4.2.1 Definitions and elementary properties

```

definition ntcf-initial :: V  $\Rightarrow$  V  $\Rightarrow$  V
  where ntcf-initial  $\mathfrak{C}$  z =

```

[
 $(\lambda b \in_{\circ} \mathfrak{C}(\text{Obj}). \text{THE } f. f : z \mapsto_{\mathfrak{C}} b),$
 $cf\text{-const } \mathfrak{C} \ \mathfrak{C} \ z,$
 $cf\text{-id } \mathfrak{C},$
 $\mathfrak{C},$
 \mathfrak{C}
]_o

definition *ntcf-terminal* :: $V \Rightarrow V \Rightarrow V$
where *ntcf-terminal* $\mathfrak{C} \ z =$

[
 $(\lambda b \in_{\circ} \mathfrak{C}(\text{Obj}). \text{THE } f. f : b \mapsto_{\mathfrak{C}} z),$
 $cf\text{-id } \mathfrak{C},$
 $cf\text{-const } \mathfrak{C} \ \mathfrak{C} \ z,$
 $\mathfrak{C},$
 \mathfrak{C}
]_o

Components.

lemma *ntcf-initial-components*:

shows $ntcf\text{-initial } \mathfrak{C} \ z(\text{NTMap}) = (\lambda c \in_{\circ} \mathfrak{C}(\text{Obj}). \text{THE } f. f : z \mapsto_{\mathfrak{C}} c)$
and $ntcf\text{-initial } \mathfrak{C} \ z(\text{NTDom}) = cf\text{-const } \mathfrak{C} \ \mathfrak{C} \ z$
and $ntcf\text{-initial } \mathfrak{C} \ z(\text{NTCod}) = cf\text{-id } \mathfrak{C}$
and $ntcf\text{-initial } \mathfrak{C} \ z(\text{NTDGDom}) = \mathfrak{C}$
and $ntcf\text{-initial } \mathfrak{C} \ z(\text{NTDGCod}) = \mathfrak{C}$
unfolding *ntcf-initial-def nt-field-simps*
by (*simp-all add: nat-omega-simps*)

lemmas [*cat-lim-cs-simps*] = *ntcf-initial-components(2-5)*

lemma *ntcf-terminal-components*:

shows $ntcf\text{-terminal } \mathfrak{C} \ z(\text{NTMap}) = (\lambda c \in_{\circ} \mathfrak{C}(\text{Obj}). \text{THE } f. f : c \mapsto_{\mathfrak{C}} z)$
and $ntcf\text{-terminal } \mathfrak{C} \ z(\text{NTDom}) = cf\text{-id } \mathfrak{C}$
and $ntcf\text{-terminal } \mathfrak{C} \ z(\text{NTCod}) = cf\text{-const } \mathfrak{C} \ \mathfrak{C} \ z$
and $ntcf\text{-terminal } \mathfrak{C} \ z(\text{NTDGDom}) = \mathfrak{C}$
and $ntcf\text{-terminal } \mathfrak{C} \ z(\text{NTDGCod}) = \mathfrak{C}$
unfolding *ntcf-terminal-def nt-field-simps*
by (*simp-all add: nat-omega-simps*)

lemmas [*cat-lim-cs-simps*] = *ntcf-terminal-components(2-5)*

Duality.

lemma *ntcf-initial-op[cat-op-simps]*:

$op\text{-ntcf } (ntcf\text{-initial } \mathfrak{C} \ z) = ntcf\text{-terminal } (op\text{-cat } \mathfrak{C}) \ z$
unfolding
 $ntcf\text{-initial-def } ntcf\text{-terminal-def } op\text{-ntcf-def}$
 $nt\text{-field-simps } cat\text{-op-simps}$
by (*auto simp: nat-omega-simps cat-op-simps*)

lemma *ntcf-cone-terminal-op[cat-op-simps]*:

$op\text{-ntcf } (ntcf\text{-terminal } \mathfrak{C} \ z) = ntcf\text{-initial } (op\text{-cat } \mathfrak{C}) \ z$
unfolding
 $ntcf\text{-initial-def } ntcf\text{-terminal-def } op\text{-ntcf-def}$
 $nt\text{-field-simps } cat\text{-op-simps}$
by (*auto simp: nat-omega-simps cat-op-simps*)

4.2.2 Natural transformation map

mk-VLambda *ntcf-initial-components(1)*
 |*vsv ntcf-initial-vsv[cat-lim-cs-intros]*||
 |*vdomain ntcf-initial-vdomain[cat-lim-cs-simps]*||
 |*app ntcf-initial-app*|

mk-VLambda *ntcf-terminal-components(1)*
 |*vsv ntcf-terminal-vsv[cat-lim-cs-intros]*||
 |*vdomain ntcf-terminal-vdomain[cat-lim-cs-simps]*||
 |*app ntcf-terminal-app*|

lemma (in *category*)
 assumes *obj-initial* \mathfrak{C} *z* and $c \in_{\circ} \mathfrak{C}(\text{Obj})$
 shows *ntcf-initial-NTMap-app-is-arr*:
 ntcf-initial $\mathfrak{C} z(\text{NTMap})(c) : z \mapsto_{\mathfrak{C}} c$
 and *ntcf-initial-NTMap-app-unique*:
 $\wedge f'. f' : z \mapsto_{\mathfrak{C}} c \implies f' = \text{ntcf-initial } \mathfrak{C} z(\text{NTMap})(c)$

proof-
 from *obj-initialD(2)[OF assms(1,2)]* obtain *f*
 where $f : z \mapsto_{\mathfrak{C}} c$
 and *f-unique*: $f' : z \mapsto_{\mathfrak{C}} c \implies f' = f$
 for *f'*
 by *auto*
 show *is-arr*: *ntcf-initial* $\mathfrak{C} z(\text{NTMap})(c) : z \mapsto_{\mathfrak{C}} c$
proof(*cs-concl-step ntcf-initial-app, rule assms(2), rule theI*)
 fix *f'* assume $f' : z \mapsto_{\mathfrak{C}} c$
 from *f-unique[OF this]* show $f' = f$.
 qed (*rule f*)
 fix *f'* assume $f' : z \mapsto_{\mathfrak{C}} c$
 from *f-unique[OF this, folded f-unique[OF is-arr]]*
 show $f' = \text{ntcf-initial } \mathfrak{C} z(\text{NTMap})(c)$.
 qed

lemma (in *category*) *ntcf-initial-NTMap-app-is-arr'[cat-lim-cs-intros]*:
 assumes *obj-initial* \mathfrak{C} *z*
 and $c \in_{\circ} \mathfrak{C}(\text{Obj})$
 and $\mathfrak{C}' = \mathfrak{C}$
 and $z' = z$
 and $c' = c$
 shows *ntcf-initial* $\mathfrak{C} z(\text{NTMap})(c) : z' \mapsto_{\mathfrak{C}'} c'$
 using *assms(1,2)*
 unfolding *assms(3-5)*
 by (*rule ntcf-initial-NTMap-app-is-arr*)

lemma (in *category*)
 assumes *obj-terminal* \mathfrak{C} *z* and $c \in_{\circ} \mathfrak{C}(\text{Obj})$
 shows *ntcf-terminal-NTMap-app-is-arr*:
 ntcf-terminal $\mathfrak{C} z(\text{NTMap})(c) : c \mapsto_{\mathfrak{C}} z$
 and *ntcf-terminal-NTMap-app-unique*:
 $\wedge f'. f' : c \mapsto_{\mathfrak{C}} z \implies f' = \text{ntcf-terminal } \mathfrak{C} z(\text{NTMap})(c)$
proof-
 from *obj-terminalD(2)[OF assms(1,2)]* obtain *f*
 where $f : c \mapsto_{\mathfrak{C}} z$
 and *f-unique*: $f' : c \mapsto_{\mathfrak{C}} z \implies f' = f$
 for *f'*
 by *auto*
 show *is-arr*: *ntcf-terminal* $\mathfrak{C} z(\text{NTMap})(c) : c \mapsto_{\mathfrak{C}} z$

```

proof(cs-concl-step ntcf-terminal-app, rule assms(2), rule theI)
  fix  $f'$  assume  $f' : c \mapsto_{\mathfrak{C}} z$ 
  from  $f$ -unique[OF this] show  $f' = f$ .
qed (rule f)
fix  $f'$  assume  $f' : c \mapsto_{\mathfrak{C}} z$ 
from  $f$ -unique[OF this, folded f-unique[OF is-arr]]
show  $f' = \text{ntcf-terminal } \mathfrak{C} \ z \ (NTMap) \ (c)$ .
qed

```

```

lemma (in category) ntcf-terminal-NTMap-app-is-arr'[cat-lim-cs-intros]:
assumes obj-terminal  $\mathfrak{C} \ z$ 
  and  $c \in_{\circ} \mathfrak{C} \ (Obj)$ 
  and  $\mathfrak{C}' = \mathfrak{C}$ 
  and  $z' = z$ 
  and  $c' = c$ 
shows ntcf-terminal  $\mathfrak{C} \ z \ (NTMap) \ (c) : c' \mapsto_{\mathfrak{C}'} z'$ 
using assms(1,2)
unfolding assms(3-5)
by (rule ntcf-terminal-NTMap-app-is-arr)

```

4.3 Initial and terminal objects as limits/colimits of the identity functor

4.3.1 Definition and elementary properties

See [1]⁴, [1]⁵ and Chapter X-1 in [9].

locale *is-cat-obj-id-initial* = *is-cat-limit* $\alpha \ \mathfrak{C} \ \mathfrak{C} \ \langle \text{cf-id } \mathfrak{C} \rangle \ z \ \mathfrak{J}$ **for** $\alpha \ \mathfrak{C} \ z \ \mathfrak{J}$

```

syntax -is-cat-obj-id-initial ::  $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow \text{bool}$ 
  ( $\langle (-) \cdot / - \langle_{CF.0} \text{id}_C \cdot / \mapsto_{C1} - \rangle [51, 51, 51] \ 51 \rangle$ )
syntax-consts -is-cat-obj-id-initial  $\equiv$  is-cat-obj-id-initial
translations  $\mathfrak{J} : z \langle_{CF.0} \text{id}_C : \mapsto_{C\alpha} \mathfrak{C} \rangle \rightleftharpoons$ 
  CONST is-cat-obj-id-initial  $\alpha \ \mathfrak{C} \ z \ \mathfrak{J}$ 

```

locale *is-cat-obj-id-terminal* = *is-cat-colimit* $\alpha \ \mathfrak{C} \ \mathfrak{C} \ \langle \text{cf-id } \mathfrak{C} \rangle \ z \ \mathfrak{J}$ **for** $\alpha \ \mathfrak{C} \ z \ \mathfrak{J}$

```

syntax -is-cat-obj-id-terminal ::  $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow \text{bool}$ 
  ( $\langle (-) \cdot / \text{id}_C \rangle_{CF.1} \cdot / \mapsto_{C1} - \rangle [51, 51, 51] \ 51$ )
syntax-consts -is-cat-obj-id-terminal  $\equiv$  is-cat-obj-id-terminal
translations  $\mathfrak{J} : \text{id}_C \rangle_{CF.1} \ z : \mapsto_{C\alpha} \mathfrak{C} \rangle \rightleftharpoons$ 
  CONST is-cat-obj-id-terminal  $\alpha \ \mathfrak{C} \ z \ \mathfrak{J}$ 

```

Rules.

```

lemma (in is-cat-obj-id-initial)
  is-cat-obj-id-initial-axioms'[cat-lim-cs-intros]:
assumes  $\alpha' = \alpha$  and  $z' = z$  and  $\mathfrak{C}' = \mathfrak{C}$ 
shows  $\mathfrak{J} : z' \langle_{CF.0} \text{id}_C : \mapsto_{C\alpha} \mathfrak{C}' \rangle \rightleftharpoons$ 
unfolding assms by (rule is-cat-obj-id-initial-axioms)

```

```

mk-ide rf is-cat-obj-id-initial-def
  |intro is-cat-obj-id-initialI|
  |dest is-cat-obj-id-initialD[dest]|
  |elim is-cat-obj-id-initialE[elim]|

```

lemmas [*cat-lim-cs-intros*] = *is-cat-obj-id-initialD*

⁴<https://ncatlab.org/nlab/show/initial+object>

⁵<https://ncatlab.org/nlab/show/terminal+object>

lemma (in *is-cat-obj-id-terminal*)
is-cat-obj-id-terminal-axioms[*cat-lim-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $z' = z$ **and** $\mathfrak{C}' = \mathfrak{C}$
shows $\exists : id_C >_{CF.1} z' : \mapsto_{C\alpha'} \mathfrak{C}'$
unfolding *assms* **by** (rule *is-cat-obj-id-terminal-axioms*)

mk-ide rf *is-cat-obj-id-terminal-def*
|*intro is-cat-obj-id-terminalI*||
|*dest is-cat-obj-id-terminalD*[*dest*]|
|*elim is-cat-obj-id-terminalE*[*elim*]|

lemmas [*cat-lim-cs-intros*] = *is-cat-obj-id-terminalD*

Duality.

lemma (in *is-cat-obj-id-initial*) *is-cat-obj-id-terminal-op*:
op-ntcf $\exists : id_C >_{CF.1} z : \mapsto_{C\alpha} op\text{-cat } \mathfrak{C}$
by (*intro is-cat-obj-id-terminalI*)
(*cs-concl cs-shallow cs-simp: cat-op-simps cs-intro: cat-op-intros*)

lemma (in *is-cat-obj-id-initial*) *is-cat-obj-id-terminal-op'*[*cat-op-intros*]:
assumes $\mathfrak{C}' = op\text{-cat } \mathfrak{C}$
shows *op-ntcf* $\exists : id_C >_{CF.1} z : \mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (rule *is-cat-obj-id-terminal-op*)

lemmas [*cat-op-intros*] = *is-cat-obj-id-initial.is-cat-obj-id-terminal-op'*

lemma (in *is-cat-obj-id-terminal*) *is-cat-obj-id-initial-op*:
op-ntcf $\exists : z <_{CF.0} id_C : \mapsto_{C\alpha} op\text{-cat } \mathfrak{C}$
by (*intro is-cat-obj-id-initialI*)
(*cs-concl cs-shallow cs-simp: cat-op-simps cs-intro: cat-op-intros*)

lemma (in *is-cat-obj-id-terminal*) *is-cat-obj-id-initial-op'*[*cat-op-intros*]:
assumes $\mathfrak{C}' = op\text{-cat } \mathfrak{C}$
shows *op-ntcf* $\exists : z <_{CF.0} id_C : \mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (rule *is-cat-obj-id-initial-op*)

lemmas [*cat-op-intros*] = *is-cat-obj-id-terminal.is-cat-obj-id-initial-op'*

4.3.2 Initial and terminal objects as limits/colimits are initial and terminal objects

lemma (in *category*) *cat-obj-initial-is-cat-obj-id-initial*:
assumes *obj-initial* $\mathfrak{C} z$
shows *ntcf-initial* $\mathfrak{C} z : z <_{CF.0} id_C : \mapsto_{C\alpha} \mathfrak{C}$
proof(*intro is-cat-obj-id-initialI is-cat-limitI*)

from *assms* **have** $z : z \in_{\circ} \mathfrak{C}(\text{Obj})$ **by** *auto*
note *obj-initialD* = *obj-initialD*[*OF assms*]

show *ntcf-initial* $\mathfrak{C} z : z <_{CF.cone} cf\text{-id } \mathfrak{C} : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{C}$
proof(*intro is-cat-coneI is-ntcfI', unfold cat-lim-cs-simps*)
show *vfsequence* (*ntcf-initial* $\mathfrak{C} z$)
unfolding *ntcf-initial-def* **by** *auto*
show *vcard* (*ntcf-initial* $\mathfrak{C} z$) = $5_{\mathbb{N}}$
unfolding *ntcf-initial-def* **by** (*simp add: nat-omega-simps*)
show *ntcf-initial* $\mathfrak{C} z(\text{NTMap})(\downarrow a) :$
 $cf\text{-const } \mathfrak{C} z(\text{ObjMap})(\downarrow a) \mapsto_{\mathfrak{C}} cf\text{-id } \mathfrak{C}(\text{ObjMap})(\downarrow a)$
if $a \in_{\circ} \mathfrak{C}(\text{Obj})$ **for** a
using *that assms(1)*

by
 (

 cs-concl

 cs-simp: *cat-cs-simps* **cs-intro:** *cat-cs-intros cat-lim-cs-intros*

)

show

 $ntcf\text{-}initial \mathfrak{C} z \langle NTMap \rangle \langle b \rangle \circ_{A\mathfrak{C}} cf\text{-}const \mathfrak{C} \mathfrak{C} z \langle ArrMap \rangle \langle f \rangle =$

 $cf\text{-}id \mathfrak{C} \langle ArrMap \rangle \langle f \rangle \circ_{A\mathfrak{C}} ntcf\text{-}initial \mathfrak{C} z \langle NTMap \rangle \langle a \rangle$

if $f : a \mapsto_{\mathfrak{C}} b$ **for** $a \ b \ f$

proof-

from *that assms(1)* **have**

 $f \circ_{A\mathfrak{C}} ntcf\text{-}initial \mathfrak{C} z \langle NTMap \rangle \langle a \rangle : z \mapsto_{\mathfrak{C}} b$

by (*cs-concl* **cs-intro:** *cat-cs-intros cat-lim-cs-intros*)

note [*cat-cs-simps*] = *ntcf-initial-NTMap-app-unique*[

 $OF \ assms(1) \ cat\text{-}is\text{-}arrD(3)[OF \ that] \ this$

]

from *that assms(1)* **show** *?thesis*

by

 (

 cs-concl

 cs-simp: *cat-cs-simps* **cs-intro:** *cat-cs-intros cat-lim-cs-intros*

)

qed

qed (*use z in* $\langle cs\text{-}concl \ cs\text{-}intro: \ cat\text{-}cs\text{-}intros \ cat\text{-}lim\text{-}cs\text{-}intros \rangle$) +

then interpret *i:* *is-cat-cone* $\alpha \ z \ \mathfrak{C} \ \mathfrak{C} \ \langle cf\text{-}id \ \mathfrak{C} \rangle \ \langle ntcf\text{-}initial \ \mathfrak{C} \ z \rangle$.

fix $u \ r$ **assume** $u : r \langle_{CF.cone} cf\text{-}id \ \mathfrak{C} : \ \mathfrak{C} \mapsto_{C\alpha} \ \mathfrak{C}$

then interpret $u:$ *is-cat-cone* $\alpha \ r \ \mathfrak{C} \ \mathfrak{C} \ \langle cf\text{-}id \ \mathfrak{C} \rangle \ u$.

from *obj-initialD(2)[OF u.cat-cone-obj]* **obtain** f

where $f : f : z \mapsto_{\mathfrak{C}} r$ **and** $f\text{-}unique: f' : z \mapsto_{\mathfrak{C}} r \implies f' = f$ **for** f'

by *auto*

note *u.cat-cone-Comp-commute[cat-cs-simps del]*

from *u.ntcf-Comp-commute[OF f]* f **have** $u \langle NTMap \rangle \langle r \rangle = f \circ_{A\mathfrak{C}} u \langle NTMap \rangle \langle z \rangle$

by (*cs-prems* **cs-simp:** *cat-cs-simps* **cs-intro:** *cat-cs-intros*)

show $\exists ! f'$.

 $f' : r \mapsto_{\mathfrak{C}} z \wedge$

 $u = ntcf\text{-}initial \ \mathfrak{C} \ z \ \bullet_{NTCF} \ ntcf\text{-}const \ \mathfrak{C} \ \mathfrak{C} \ f'$

proof(*intro ex1I conjI; (elim conjE)?*)

from f **show** $u \langle NTMap \rangle \langle z \rangle : r \mapsto_{\mathfrak{C}} z$

by (*cs-concl* **cs-simp:** *cat-cs-simps* **cs-intro:** *cat-cs-intros*)

show $u = ntcf\text{-}initial \ \mathfrak{C} \ z \ \bullet_{NTCF} \ ntcf\text{-}const \ \mathfrak{C} \ \mathfrak{C} \ (u \langle NTMap \rangle \langle z \rangle)$

proof(*rule ntcf-eqI, rule u.is-ntcf-axioms*)

show $ntcf\text{-}initial \ \mathfrak{C} \ z \ \bullet_{NTCF} \ ntcf\text{-}const \ \mathfrak{C} \ \mathfrak{C} \ (u \langle NTMap \rangle \langle z \rangle) :$

 $cf\text{-}const \ \mathfrak{C} \ \mathfrak{C} \ r \mapsto_{CF} \ cf\text{-}id \ \mathfrak{C} : \ \mathfrak{C} \mapsto_{C\alpha} \ \mathfrak{C}$

by (*cs-concl* **cs-simp:** *cat-cs-simps* **cs-intro:** *cat-cs-intros*)

from z **have** *dom-rhs:*

 $D_{\circ} ((ntcf\text{-}initial \ \mathfrak{C} \ z \ \bullet_{NTCF} \ ntcf\text{-}const \ \mathfrak{C} \ \mathfrak{C} \ (u \langle NTMap \rangle \langle z \rangle)) \langle NTMap \rangle) =$

 $\mathfrak{C} \langle Obj \rangle$

by (*cs-concl* **cs-simp:** *cat-cs-simps* **cs-intro:** *cat-cs-intros*)

show $u \langle NTMap \rangle =$

 $(ntcf\text{-}initial \ \mathfrak{C} \ z \ \bullet_{NTCF} \ ntcf\text{-}const \ \mathfrak{C} \ \mathfrak{C} \ (u \langle NTMap \rangle \langle z \rangle)) \langle NTMap \rangle$

proof(*rule vsu-eqI, unfold dom-rhs u.ntcf-NTMap-vdomain*)

fix c **assume** *prems:* $c \in_{\circ} \mathfrak{C} \langle Obj \rangle$

then have $ic: ntcf\text{-}initial \ \mathfrak{C} z \langle NTMap \rangle \langle c \rangle : z \mapsto_{\mathfrak{C}} c$

by (*cs-concl* **cs-simp:** *cat-cs-simps* **cs-intro:** *cat-cs-intros*)

from *u.ntcf-Comp-commute[OF ic]* ic **have** [*cat-cs-simps*]:

$ntcf\text{-initial} \mathfrak{C} z \langle NTMap \rangle \langle c \rangle \circ_{A\mathfrak{C}} u \langle NTMap \rangle \langle z \rangle = u \langle NTMap \rangle \langle c \rangle$
by (*cs-prems* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*) *simp*
from *prems* z **show** $u \langle NTMap \rangle \langle c \rangle =$
 $(ntcf\text{-initial} \mathfrak{C} z \cdot_{NTCF} ntcf\text{-const} \mathfrak{C} \mathfrak{C} (u \langle NTMap \rangle \langle z \rangle)) \langle NTMap \rangle \langle c \rangle$
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed (*auto intro*: *cat-cs-intros*)
qed *simp-all*
fix f' **assume** *prems*:
 $f' : r \mapsto_{\mathfrak{C}} z$
 $u = ntcf\text{-initial} \mathfrak{C} z \cdot_{NTCF} ntcf\text{-const} \mathfrak{C} \mathfrak{C} f'$
from z **have** $ntcf\text{-initial} \mathfrak{C} z \langle NTMap \rangle \langle z \rangle : z \mapsto_{\mathfrak{C}} z$
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
note [*cat-cs-simps*] = *cat-obj-initial-CId*[*OF assms this, symmetric*]
from *prems*(2) **have**
 $u \langle NTMap \rangle \langle z \rangle = (ntcf\text{-initial} \mathfrak{C} z \cdot_{NTCF} ntcf\text{-const} \mathfrak{C} \mathfrak{C} f') \langle NTMap \rangle \langle z \rangle$
by *simp*
from *this prems*(1) **show** $f' = u \langle NTMap \rangle \langle z \rangle$
by (*cs-prems* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*) *simp*
qed
qed

lemma (*in category*) *cat-obj-terminal-is-cat-obj-id-terminal*:
assumes *obj-terminal* $\mathfrak{C} z$
shows *ntcf-terminal* $\mathfrak{C} z : id_C >_{CF.1} z : \mapsto_{C\alpha} \mathfrak{C}$
by
 (

 rule is-cat-obj-id-initial.is-cat-obj-id-terminal-op

 [

 OF category.cat-obj-initial-is-cat-obj-id-initial[

 OF category-op op-cat-obj-initial[*THEN iffD2, OF assms*(1)]

],

 unfolded cat-op-simps

]

)

lemma *cat-cone-CId-obj-initial*:
assumes $\exists : z <_{CF.cone} cf\text{-id} \mathfrak{C} : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{C}$ **and** $\exists \langle NTMap \rangle \langle z \rangle = \mathfrak{C} \langle CId \rangle \langle z \rangle$
shows *obj-initial* $\mathfrak{C} z$
proof(*intro obj-initialI*)
interpret \exists : *is-cat-cone* $\alpha z \mathfrak{C} \mathfrak{C} \langle cf\text{-id} \mathfrak{C} \rangle \exists$ **by** (*rule assms*(1))
show $z \in_{\circ} \mathfrak{C} \langle Obj \rangle$ **by** (*cs-concl* **cs-intro**: *cat-cs-intros*)
fix c **assume** *prems*: $c \in_{\circ} \mathfrak{C} \langle Obj \rangle$
show $\exists ! f. f : z \mapsto_{\mathfrak{C}} c$
proof(*intro ex1I*)
from *prems* **show** $\exists c : \exists \langle NTMap \rangle \langle c \rangle : z \mapsto_{\mathfrak{C}} c$
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
fix f **assume** *prems'*: $f : z \mapsto_{\mathfrak{C}} c$
from $\exists.ntcf\text{-Comp-commute}$ [*OF prems'*] *prems'* $\exists c$ **show** $f = \exists \langle NTMap \rangle \langle c \rangle$
by (*cs-prems* **cs-simp**: *cat-cs-simps* *assms*(2) **cs-intro**: *cat-cs-intros*) *simp*
qed
qed

lemma *cat-cocone-CId-obj-terminal*:
assumes $\exists : cf\text{-id} \mathfrak{C} >_{CF.cocone} z : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{C}$ **and** $\exists \langle NTMap \rangle \langle z \rangle = \mathfrak{C} \langle CId \rangle \langle z \rangle$
shows *obj-terminal* $\mathfrak{C} z$
proof-
interpret \exists : *is-cat-cocone* $\alpha z \mathfrak{C} \mathfrak{C} \langle cf\text{-id} \mathfrak{C} \rangle \exists$ **by** (*rule assms*(1))
show *?thesis*

by
 (
 rule *cat-cone-CId-obj-initial*
 [
 OF \exists .*is-cat-cone-op*[*unfolded cat-op-simps*],
 unfolded cat-op-simps,
 OF assms(2)
]
)
 qed

lemma (in *is-cat-obj-id-initial*) *cat-oi-obj-initial*: *obj-initial* \mathfrak{C} z
proof(rule *cat-cone-CId-obj-initial*, rule *is-cat-cone-axioms*)
from *cat-lim-unique-cone*'[*OF is-cat-cone-axioms*] **obtain** f
where $f: f : z \mapsto_{\mathfrak{C}} z$
and $\exists'j: \bigwedge j. j \in_{\circ} \mathfrak{C}(\text{Obj}) \implies \exists(\text{NTMap})(j) = \exists(\text{NTMap})(j) \circ_{A\mathfrak{C}} f$
and *f-unique*:
 [[
 $f' : z \mapsto_{\mathfrak{C}} z$;
 $\bigwedge j. j \in_{\circ} \mathfrak{C}(\text{Obj}) \implies \exists(\text{NTMap})(j) = \exists(\text{NTMap})(j) \circ_{A\mathfrak{C}} f'$
]] $\implies f' = f$
for f'
by *metis*
have *CId-z*: $\mathfrak{C}(\text{CId})(z) : z \mapsto_{\mathfrak{C}} z$
by (*cs-concl cs-intro*: *cat-cs-intros*)
have $\exists(\text{NTMap})(j) = \exists(\text{NTMap})(j) \circ_{A\mathfrak{C}} \mathfrak{C}(\text{CId})(z)$ **if** $j \in_{\circ} \mathfrak{C}(\text{Obj})$ **for** j
using *that* **by** (*cs-concl cs-simp*: *cat-cs-simps cs-intro*: *cat-cs-intros*)
from *f-unique*[*OF CId-z this*] **have** *CId-f*: $\mathfrak{C}(\text{CId})(z) = f$.
have $\exists z: \exists(\text{NTMap})(z) : z \mapsto_{\mathfrak{C}} z$
by (*cs-concl cs-simp*: *cat-cs-simps cs-intro*: *cat-cs-intros*)
have $\exists(\text{NTMap})(c) = \exists(\text{NTMap})(c) \circ_{A\mathfrak{C}} \exists(\text{NTMap})(z)$ **if** $c \in_{\circ} \mathfrak{C}(\text{Obj})$ **for** c
proof-
from *that* **have** $\exists c: \exists(\text{NTMap})(c) : z \mapsto_{\mathfrak{C}} c$
by (*cs-concl cs-simp*: *cat-cs-simps cs-intro*: *cat-cs-intros*)
note *cat-cone-Comp-commute*[*cat-cs-simps del*]
from *ntcf-Comp-commute*[*OF* $\exists c$] $\exists c$ **show**
 $\exists(\text{NTMap})(c) = \exists(\text{NTMap})(c) \circ_{A\mathfrak{C}} \exists(\text{NTMap})(z)$
by (*cs-prems cs-simp*: *cat-cs-simps cs-intro*: *cat-cs-intros*)
 qed
from *f-unique*[*OF* $\exists z$ *this*] **have** $\exists(\text{NTMap})(z) = f$.
with *CId-f* **show** $\exists(\text{NTMap})(z) = \mathfrak{C}(\text{CId})(z)$ **by** *simp*
 qed

lemma (in *is-cat-obj-id-terminal*) *cat-oi-obj-terminal*: *obj-terminal* \mathfrak{C} z
by
 (
 rule *is-cat-obj-id-initial.cat-oi-obj-initial*[
 OF is-cat-obj-id-initial-op, *unfolded cat-op-simps*
]
)

lemma (in *category*) *cat-is-cat-obj-id-initial-obj-initial-iff*:
 (*ntcf-initial* $\mathfrak{C} z : z <_{CF.0} id_C : \mapsto_{C\alpha} \mathfrak{C}$) \longleftrightarrow *obj-initial* \mathfrak{C} z
using
cat-obj-initial-is-cat-obj-id-initial
is-cat-obj-id-initial.cat-oi-obj-initial
by *auto*

lemma (in *category*) *cat-is-cat-obj-id-terminal-obj-terminal-iff*:
 (*ntcf-terminal* $\mathfrak{C} z : id_C >_{CF.1} z : \mapsto_{C\alpha} \mathfrak{C}$) \leftrightarrow *obj-terminal* $\mathfrak{C} z$
using
 cat-obj-terminal-is-cat-obj-id-terminal
 is-cat-obj-id-terminal.cat-oit-obj-terminal
by *auto*

5 Products and coproducts as limits and colimits

5.1 Product and coproduct

5.1.1 Definition and elementary properties

The definition of the product object is a specialization of the definition presented in Chapter III-4 in [9]. In the definition presented below, the discrete category that is used in the definition presented in [9] is parameterized by an index set and the functor from the discrete category is parameterized by a function from the index set to the set of the objects of the category.

locale *is-cat-obj-prod* =
is-cat-limit $\alpha \langle :_C I \rangle \mathfrak{C} \langle \cdot \rightarrow : I A \mathfrak{C} \rangle P \pi + \text{cf-discrete } \alpha I A \mathfrak{C}$
for $\alpha I A \mathfrak{C} P \pi$

syntax *is-cat-obj-prod* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow \text{bool}$
 $(\langle (- : / - <_{CF.\Pi} - : / - \mapsto_{C1} -) \rangle [51, 51, 51, 51, 51] 51)$
syntax-consts *is-cat-obj-prod* $\hat{=}$ *is-cat-obj-prod*
translations $\pi : P <_{CF.\Pi} A : I \mapsto_{C\alpha} \mathfrak{C} \hat{=}$
CONST is-cat-obj-prod $\alpha I A \mathfrak{C} P \pi$

locale *is-cat-obj-coproduct* =
is-cat-colimit $\alpha \langle :_C I \rangle \mathfrak{C} \langle \cdot \rightarrow : I A \mathfrak{C} \rangle U \pi + \text{cf-discrete } \alpha I A \mathfrak{C}$
for $\alpha I A \mathfrak{C} U \pi$

syntax *is-cat-obj-coproduct* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow \text{bool}$
 $(\langle (- : / - >_{CF.\Pi} - : / - \mapsto_{C1} -) \rangle [51, 51, 51, 51, 51] 51)$
syntax-consts *is-cat-obj-coproduct* $\hat{=}$ *is-cat-obj-coproduct*
translations $\pi : A >_{CF.\Pi} U : I \mapsto_{C\alpha} \mathfrak{C} \hat{=}$
CONST is-cat-obj-coproduct $\alpha I A \mathfrak{C} U \pi$

Rules.

lemma (**in** *is-cat-obj-prod*) *is-cat-obj-prod-axioms'*[*cat-lim-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $P' = P$ **and** $A' = A$ **and** $I' = I$ **and** $\mathfrak{C}' = \mathfrak{C}$
shows $\pi : P' <_{CF.\Pi} A' : I' \mapsto_{C\alpha'} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-obj-prod-axioms*)

mk-ide rf *is-cat-obj-prod-def*
 $| \text{intro } is\text{-cat-obj-prod}I |$
 $| \text{dest } is\text{-cat-obj-prod}D [\text{dest}] |$
 $| \text{elim } is\text{-cat-obj-prod}E [\text{elim}] |$

lemmas [*cat-lim-cs-intros*] = *is-cat-obj-prodD*

lemma (**in** *is-cat-obj-coproduct*) *is-cat-obj-coproduct-axioms'*[*cat-lim-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $U' = U$ **and** $A' = A$ **and** $I' = I$ **and** $\mathfrak{C}' = \mathfrak{C}$
shows $\pi : A' >_{CF.\Pi} U' : I' \mapsto_{C\alpha'} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-obj-coproduct-axioms*)

mk-ide rf *is-cat-obj-coproduct-def*
 $| \text{intro } is\text{-cat-obj-coproduct}I |$
 $| \text{dest } is\text{-cat-obj-coproduct}D [\text{dest}] |$
 $| \text{elim } is\text{-cat-obj-coproduct}E [\text{elim}] |$

lemmas [*cat-lim-cs-intros*] = *is-cat-obj-coproductD*

Duality.

lemma (**in** *is-cat-obj-prod*) *is-cat-obj-coproduct-op*:

$op\text{-}ntcf \pi : A >_{CF.\Pi} P : I \mapsto_{C\alpha} op\text{-}cat \mathfrak{C}$
using $cf\text{-discrete-vdomain-vsubset-Vset}$
by (*intro is-cat-obj-coproductI*)
(

 $cs\text{-concl}$ **cs-shallow**
 cs-simp: $cat\text{-op-simps}$ **cs-intro:** $cat\text{-cs-intros}$ $cat\text{-op-intros}$

)

lemma (*in is-cat-obj-prod*) $is\text{-cat-obj-coproduct-op}'[cat\text{-op-intros}]$:
assumes $\mathfrak{C}' = op\text{-}cat \mathfrak{C}$
shows $op\text{-}ntcf \pi : A >_{CF.\Pi} P : I \mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-obj-coproduct-op*)

lemmas $[cat\text{-op-intros}] = is\text{-cat-obj-prod.is-cat-obj-coproduct-op}'$

lemma (*in is-cat-obj-coproduct*) $is\text{-cat-obj-prod-op}$:
 $op\text{-}ntcf \pi : U <_{CF.\Pi} A : I \mapsto_{C\alpha} op\text{-}cat \mathfrak{C}$
using $cf\text{-discrete-vdomain-vsubset-Vset}$
by (*intro is-cat-obj-prodI*)
(

 $cs\text{-concl}$ **cs-shallow**
 cs-simp: $cat\text{-op-simps}$ **cs-intro:** $cat\text{-cs-intros}$ $cat\text{-op-intros}$

)

lemma (*in is-cat-obj-coproduct*) $is\text{-cat-obj-prod-op}'[cat\text{-op-intros}]$:
assumes $\mathfrak{C}' = op\text{-}cat \mathfrak{C}$
shows $op\text{-}ntcf \pi : U <_{CF.\Pi} A : I \mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-obj-prod-op*)

lemmas $[cat\text{-op-intros}] = is\text{-cat-obj-coproduct.is-cat-obj-prod-op}'$

5.1.2 Universal property

lemma (*in is-cat-obj-prod*) $cat\text{-obj-prod-unique-cone}'$:
assumes $\pi' : P' <_{CF.cone} \rightarrow : I A \mathfrak{C} : :_C I \mapsto_{C\alpha} \mathfrak{C}$
shows $\exists ! f'. f' : P' \rightarrow_{\mathfrak{C}} P \wedge (\forall j \in_o I. \pi'(\downarrow NTMap)(\downarrow j) = \pi(\downarrow NTMap)(\downarrow j) \circ_{A\mathfrak{C}} f')$
by
(

 $rule\ cat\text{-lim-unique-cone}'[$
 $OF\ assms, unfolded\ the\ cat\text{-discrete-components}(1)$
 $]$

)

lemma (*in is-cat-obj-prod*) $cat\text{-obj-prod-unique}$:
assumes $\pi' : P' <_{CF.\Pi} A : I \mapsto_{C\alpha} \mathfrak{C}$
shows $\exists ! f'. f' : P' \rightarrow_{\mathfrak{C}} P \wedge \pi' = \pi \cdot_{NTCF} ntcf\text{-const} (:_C I) \mathfrak{C} f'$
by (*intro cat-lim-unique[OF is-cat-obj-prodD(1)[OF assms]]*)

lemma (*in is-cat-obj-prod*) $cat\text{-obj-prod-unique}'$:
assumes $\pi' : P' <_{CF.\Pi} A : I \mapsto_{C\alpha} \mathfrak{C}$
shows $\exists ! f'. f' : P' \rightarrow_{\mathfrak{C}} P \wedge (\forall i \in_o I. \pi'(\downarrow NTMap)(\downarrow i) = \pi(\downarrow NTMap)(\downarrow i) \circ_{A\mathfrak{C}} f')$

proof-

interpret π' : $is\text{-cat-obj-prod} \alpha I A \mathfrak{C} P' \pi'$ **by** (*rule assms(1)*)
show *?thesis*
by
(

 $rule\ cat\text{-lim-unique}'[$
 $OF\ \pi'.is\text{-cat-limit-axioms, unfolded\ the\ cat\text{-discrete-components}(1)$
)

)
)
 qed

lemma (in *is-cat-obj-coprod*) *cat-obj-coprod-unique-cocone'*:
assumes $\pi' : \cdot \rightarrow I A \mathfrak{C} >_{CF.\Pi} U' : I \mapsto C\alpha \mathfrak{C}$
shows $\exists ! f'. f' : U \mapsto_{\mathfrak{C}} U' \wedge (\forall j \in_o I. \pi'(\backslash NTMap)(\backslash j) = f' \circ_A \mathfrak{C} \pi(\backslash NTMap)(\backslash j))$
by
 (
 rule cat-colim-unique-cocone'
 OF assms, unfolded the-cat-discrete-components(1)
)]
)

lemma (in *is-cat-obj-coprod*) *cat-obj-coprod-unique*:
assumes $\pi' : A >_{CF.\Pi} U' : I \mapsto C\alpha \mathfrak{C}$
shows $\exists ! f'. f' : U \mapsto_{\mathfrak{C}} U' \wedge \pi' = ntcf-const (:_C I) \mathfrak{C} f' \cdot_{NTCF} \pi$
by (*intro cat-colim-unique[OF is-cat-obj-coprodD(1)[OF assms]]*)

lemma (in *is-cat-obj-coprod*) *cat-obj-coprod-unique'*:
assumes $\pi' : A >_{CF.\Pi} U' : I \mapsto C\alpha \mathfrak{C}$
shows $\exists ! f'. f' : U \mapsto_{\mathfrak{C}} U' \wedge (\forall j \in_o I. \pi'(\backslash NTMap)(\backslash j) = f' \circ_A \mathfrak{C} \pi(\backslash NTMap)(\backslash j))$
by
 (
 rule cat-colim-unique'
 OF is-cat-obj-coprodD(1)[OF assms], unfolded the-cat-discrete-components
)]
)

lemma *cat-obj-prod-ex-is-iso-arr*:
assumes $\pi : P <_{CF.\Pi} A : I \mapsto C\alpha \mathfrak{C}$ **and** $\pi' : P' <_{CF.\Pi} A : I \mapsto C\alpha \mathfrak{C}$
obtains *f* **where** $f : P' \mapsto_{iso} P$ **and** $\pi' = \pi \cdot_{NTCF} ntcf-const (:_C I) \mathfrak{C} f$
proof-
interpret $\pi : is-cat-obj-prod \alpha I A \mathfrak{C} P \pi$ **by** (*rule assms(1)*)
interpret $\pi' : is-cat-obj-prod \alpha I A \mathfrak{C} P' \pi'$ **by** (*rule assms(2)*)
from that show *?thesis*
by
 (
 elim cat-lim-ex-is-iso-arr
 OF $\pi.is-cat-limit-axioms$ $\pi'.is-cat-limit-axioms$
)]
)
 qed

lemma *cat-obj-prod-ex-is-iso-arr'*:
assumes $\pi : P <_{CF.\Pi} A : I \mapsto C\alpha \mathfrak{C}$ **and** $\pi' : P' <_{CF.\Pi} A : I \mapsto C\alpha \mathfrak{C}$
obtains *f* **where** $f : P' \mapsto_{iso} P$
and $\wedge j. j \in_o I \implies \pi'(\backslash NTMap)(\backslash j) = \pi(\backslash NTMap)(\backslash j) \circ_A \mathfrak{C} f$
proof-
interpret $\pi : is-cat-obj-prod \alpha I A \mathfrak{C} P \pi$ **by** (*rule assms(1)*)
interpret $\pi' : is-cat-obj-prod \alpha I A \mathfrak{C} P' \pi'$ **by** (*rule assms(2)*)
from that show *?thesis*
by
 (
 elim cat-lim-ex-is-iso-arr'
 OF $\pi.is-cat-limit-axioms$ $\pi'.is-cat-limit-axioms$,
 unfolded the-cat-discrete-components(1)
)]

)
qed

lemma *cat-obj-coproduct-ex-is-iso-arr*:

assumes $\pi : A >_{CF.\Pi} U : I \mapsto_{\rightarrow C\alpha} \mathfrak{C}$ **and** $\pi' : A >_{CF.\Pi} U' : I \mapsto_{\rightarrow C\alpha} \mathfrak{C}$
obtains f **where** $f : U \mapsto_{iso\mathfrak{C}} U'$ **and** $\pi' = ntcf-const (\cdot_C I) \mathfrak{C} f \cdot_{NTCF} \pi$

proof-

interpret π : *is-cat-obj-coproduct* $\alpha I A \mathfrak{C} U \pi$ **by** (*rule assms(1)*)

interpret π' : *is-cat-obj-coproduct* $\alpha I A \mathfrak{C} U' \pi'$ **by** (*rule assms(2)*)

from that show *?thesis*

by

(
 elim cat-colim-ex-is-iso-arr [
 OF $\pi.is-cat-colimit-axioms$ $\pi'.is-cat-colimit-axioms$
]
)

)
qed

lemma *cat-obj-coproduct-ex-is-iso-arr'*:

assumes $\pi : A >_{CF.\Pi} U : I \mapsto_{\rightarrow C\alpha} \mathfrak{C}$ **and** $\pi' : A >_{CF.\Pi} U' : I \mapsto_{\rightarrow C\alpha} \mathfrak{C}$

obtains f **where** $f : U \mapsto_{iso\mathfrak{C}} U'$

and $\bigwedge j. j \in_o I \implies \pi'(NTMap)(j) = f \circ_A \mathfrak{C} \pi(NTMap)(j)$

proof-

interpret π : *is-cat-obj-coproduct* $\alpha I A \mathfrak{C} U \pi$ **by** (*rule assms(1)*)

interpret π' : *is-cat-obj-coproduct* $\alpha I A \mathfrak{C} U' \pi'$ **by** (*rule assms(2)*)

from that show *?thesis*

by

(
 elim cat-colim-ex-is-iso-arr' [
 OF $\pi.is-cat-colimit-axioms$ $\pi'.is-cat-colimit-axioms$,
 unfolded the-cat-discrete-components(1)
]
)

)
qed

5.2 Small product and small coproduct

5.2.1 Definition and elementary properties

locale *is-tm-cat-obj-prod* =

is-cat-limit $\alpha \langle \cdot_C I \rangle \mathfrak{C} \langle \cdot \rightarrow \cdot : I A \mathfrak{C} \rangle P \pi + tm-cf-discrete \alpha I A \mathfrak{C}$

for $\alpha I A \mathfrak{C} P \pi$

syntax *-is-tm-cat-obj-prod* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$

($\langle \langle \cdot \cdot / \cdot \cdot \rangle_{CF.tm.\Pi} \cdot \cdot / \cdot \cdot \mapsto_{\rightarrow C.tm1} \cdot \cdot \rangle [51, 51, 51, 51, 51] 51$)

syntax-consts *-is-tm-cat-obj-prod* $\hat{=} is-tm-cat-obj-prod$

translations $\pi : P <_{CF.tm.\Pi} A : I \mapsto_{\rightarrow C.tm\alpha} \mathfrak{C} \hat{=}$

CONST is-tm-cat-obj-prod $\alpha I A \mathfrak{C} P \pi$

locale *is-tm-cat-obj-coproduct* =

is-cat-colimit $\alpha \langle \cdot_C I \rangle \mathfrak{C} \langle \cdot \rightarrow \cdot : I A \mathfrak{C} \rangle U \pi + tm-cf-discrete \alpha I A \mathfrak{C}$

for $\alpha I A \mathfrak{C} U \pi$

syntax *-is-tm-cat-obj-coproduct* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$

($\langle \langle \cdot \cdot / \cdot \cdot \rangle_{CF.tm.\Pi} \cdot \cdot / \cdot \cdot \mapsto_{\rightarrow C.tm1} \cdot \cdot \rangle [51, 51, 51, 51, 51] 51$)

syntax-consts *-is-tm-cat-obj-coproduct* $\hat{=} is-tm-cat-obj-coproduct$

translations $\pi : A >_{CF.tm.\Pi} U : I \mapsto_{\rightarrow C.tm\alpha} \mathfrak{C} \hat{=}$

CONST is-tm-cat-obj-coproduct $\alpha I A \mathfrak{C} U \pi$

Rules.

lemma (in *is-tm-cat-obj-prod*) *is-tm-cat-obj-prod-axioms'*[*cat-lim-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $P' = P$ **and** $A' = A$ **and** $I' = I$ **and** $\mathfrak{C}' = \mathfrak{C}$
shows $\pi : P' <_{CF.tm.\Pi} A' : I' \mapsto \mapsto_{C.tm\alpha'} \mathfrak{C}'$
unfolding *assms* **by** (rule *is-tm-cat-obj-prod-axioms*)

mk-ide rf *is-tm-cat-obj-prod-def*
intro is-tm-cat-obj-prodI	
dest is-tm-cat-obj-prodD[dest]	
elim is-tm-cat-obj-prodE[elim]	

lemmas [*cat-lim-cs-intros*] = *is-tm-cat-obj-prodD*

lemma (in *is-tm-cat-obj-coproduct*)
is-tm-cat-obj-coproduct-axioms'[*cat-lim-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $U' = U$ **and** $A' = A$ **and** $I' = I$ **and** $\mathfrak{C}' = \mathfrak{C}$
shows $\pi : A' >_{CF.tm.\Pi} U' : I' \mapsto \mapsto_{C.tm\alpha'} \mathfrak{C}'$
unfolding *assms* **by** (rule *is-tm-cat-obj-coproduct-axioms*)

mk-ide rf *is-tm-cat-obj-coproduct-def*
intro is-tm-cat-obj-coproductI	
dest is-tm-cat-obj-coproductD[dest]	
elim is-tm-cat-obj-coproductE[elim]	

lemmas [*cat-lim-cs-intros*] = *is-tm-cat-obj-coproductD*

Elementary properties.

sublocale *is-tm-cat-obj-prod* \subseteq *is-cat-obj-prod*
by
(
intro is-cat-obj-prodI,
rule is-cat-limit-axioms,
rule cf-discrete-axioms
)

lemmas (in *is-tm-cat-obj-prod*) *tm-cat-obj-prod-is-cat-obj-prod* =
is-cat-obj-prod-axioms

sublocale *is-tm-cat-obj-coproduct* \subseteq *is-cat-obj-coproduct*
by
(
intro is-cat-obj-coproductI,
rule is-cat-colimit-axioms,
rule cf-discrete-axioms
)

lemmas (in *is-tm-cat-obj-coproduct*) *tm-cat-obj-coproduct-is-cat-obj-coproduct* =
is-cat-obj-coproduct-axioms

sublocale *is-tm-cat-obj-prod* \subseteq *is-tm-cat-limit* $\alpha \langle :_C I \rangle \mathfrak{C} \langle :- \rangle : I A \mathfrak{C} \rangle P \pi$
by
(
intro
is-tm-cat-limitI
is-tm-cat-coneI
is-ntcf-axioms
tm-cf-discrete-the-cf-discrete-is-tm-functor
cat-cone-obj
)

)
cat-lim-ua-fo

lemmas (in *is-tm-cat-obj-prod*) *tm-cat-obj-prod-is-tm-cat-limit* =
is-tm-cat-limit-axioms

sublocale *is-tm-cat-obj-coproduct* \subseteq *is-tm-cat-colimit* $\alpha \langle :_C I \rangle \mathfrak{C} \langle :=: I A \mathfrak{C} \rangle U \pi$
by
 (*intro*
is-tm-cat-colimitI
is-tm-cat-coconeI
is-ntcf-axioms
tm-cf-discrete-the-cf-discrete-is-tm-functor
cat-cocone-obj
cat-colim-ua-of
)

lemmas (in *is-tm-cat-obj-coproduct*) *tm-cat-obj-coproduct-is-tm-cat-colimit* =
is-tm-cat-colimit-axioms

Duality.

lemma (in *is-tm-cat-obj-prod*) *is-tm-cat-obj-coproduct-op*:
op-ntcf $\pi : A >_{CF.tm.\sqcup} P : I \mapsto \mapsto_{C.tm\alpha} op-cat \mathfrak{C}$
using *cf-discrete-vdomain-vsubset-Vset*
by (*intro is-tm-cat-obj-coproductI*)
 (*cs-concl cs-simp: cat-op-simps cs-intro: cat-op-intros*)

lemma (in *is-tm-cat-obj-prod*) *is-tm-cat-obj-coproduct-op'*[*cat-op-intros*]:
assumes $\mathfrak{C}' = op-cat \mathfrak{C}$
shows *op-ntcf* $\pi : A >_{CF.tm.\sqcup} P : I \mapsto \mapsto_{C.tm\alpha} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-tm-cat-obj-coproduct-op*)

lemmas [*cat-op-intros*] = *is-tm-cat-obj-prod.is-tm-cat-obj-coproduct-op'*

lemma (in *is-tm-cat-obj-coproduct*) *is-tm-cat-obj-coproduct-op*:
op-ntcf $\pi : U <_{CF.tm.\sqcap} A : I \mapsto \mapsto_{C.tm\alpha} op-cat \mathfrak{C}$
using *cf-discrete-vdomain-vsubset-Vset*
by (*intro is-tm-cat-obj-prodI*)
 (*cs-concl cs-simp: cat-op-simps cs-intro: cat-op-intros*)

lemma (in *is-tm-cat-obj-coproduct*) *is-tm-cat-obj-prod-op'*[*cat-op-intros*]:
assumes $\mathfrak{C}' = op-cat \mathfrak{C}$
shows *op-ntcf* $\pi : U <_{CF.tm.\sqcap} A : I \mapsto \mapsto_{C.tm\alpha} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-tm-cat-obj-coproduct-op*)

lemmas [*cat-op-intros*] = *is-tm-cat-obj-coproduct.is-tm-cat-obj-prod-op'*

5.3 Finite product and finite coproduct

locale *is-cat-finite-obj-prod* = *is-cat-obj-prod* $\alpha I A \mathfrak{C} P \pi$
for $\alpha I A \mathfrak{C} P \pi +$
assumes *cat-fin-obj-prod-index-in- ω* : $I \in_\omega \omega$

syntax *-is-cat-finite-obj-prod* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
 ($\langle (- : / - <_{CF.\sqcap}.fin - : / - \mapsto \mapsto_{C1} -) \rangle [51, 51, 51, 51, 51] 51$)
syntax-consts *-is-cat-finite-obj-prod* $\hat{=}$ *is-cat-finite-obj-prod*
translations $\pi : P <_{CF.\sqcap}.fin A : I \mapsto \mapsto_{C\alpha} \mathfrak{C} \hat{=}$

CONST is-cat-finite-obj-prod α *I A* \mathfrak{C} *P* π

locale *is-cat-finite-obj-coproduct* = *is-cat-obj-coproduct* α *I A* \mathfrak{C} *U* π
for α *I A* \mathfrak{C} *U* π +
assumes *cat-fin-obj-coproduct-index-in- ω* : *I* ϵ_ω ω

syntax *-is-cat-finite-obj-coproduct* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
($\langle (- : / - >_{CF.\Pi} .fin - : / - \mapsto_{C^1} -) \rangle$ [51, 51, 51, 51, 51] 51)
syntax-consts *-is-cat-finite-obj-coproduct* \equiv *is-cat-finite-obj-coproduct*
translations $\pi : A >_{CF.\Pi} .fin U : I \mapsto_{C\alpha} \mathfrak{C} \equiv$
CONST is-cat-finite-obj-coproduct α *I A* \mathfrak{C} *U* π

lemma (**in** *is-cat-finite-obj-prod*) *cat-fin-obj-prod-index-vfinite*: *vfinite I*
using *cat-fin-obj-prod-index-in- ω* **by** *auto*

sublocale *is-cat-finite-obj-prod* \subseteq *I*: *finite-category* α $\langle :_C I \rangle$
by (*intro finite-categoryI'*)
(
 auto
 simp: *NTDom.HomDom.category-axioms the-cat-discrete-components*
 intro!: *cat-fin-obj-prod-index-vfinite*
)

lemma (**in** *is-cat-finite-obj-coproduct*) *cat-fin-obj-coproduct-index-vfinite*:
vfinite I
using *cat-fin-obj-coproduct-index-in- ω* **by** *auto*

sublocale *is-cat-finite-obj-coproduct* \subseteq *I*: *finite-category* α $\langle :_C I \rangle$
by (*intro finite-categoryI'*)
(
 auto
 simp: *NTDom.HomDom.category-axioms the-cat-discrete-components*
 intro!: *cat-fin-obj-coproduct-index-vfinite*
)

Rules.

lemma (**in** *is-cat-finite-obj-prod*)
is-cat-finite-obj-prod-axioms'[*cat-lim-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $P' = P$ **and** $A' = A$ **and** $I' = I$ **and** $\mathfrak{C}' = \mathfrak{C}$
shows $\pi : P' <_{CF.\Pi} .fin A' : I' \mapsto_{C\alpha'} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-finite-obj-prod-axioms*)

mk-ide rf

is-cat-finite-obj-prod-def[*unfolded is-cat-finite-obj-prod-axioms-def*]
|*intro is-cat-finite-obj-prodI*||
|*dest is-cat-finite-obj-prodD*[*dest*]|
|*elim is-cat-finite-obj-prodE*[*elim*]|

lemmas [*cat-lim-cs-intros*] = *is-cat-finite-obj-prodD*

lemma (**in** *is-cat-finite-obj-coproduct*)
is-cat-finite-obj-coproduct-axioms'[*cat-lim-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $U' = U$ **and** $A' = A$ **and** $I' = I$ **and** $\mathfrak{C}' = \mathfrak{C}$
shows $\pi : A' >_{CF.\Pi} .fin U' : I' \mapsto_{C\alpha'} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-finite-obj-coproduct-axioms*)

mk-ide rf

is-cat-finite-obj-coproduct-def[*unfolded is-cat-finite-obj-coproduct-axioms-def*]

|intro is-cat-finite-obj-coproductI|
|dest is-cat-finite-obj-coproductD[dest]|
|elim is-cat-finite-obj-coproductE[elim]|

lemmas [cat-lim-cs-intros] = is-cat-finite-obj-coproductD

Duality.

lemma (in is-cat-finite-obj-prod) is-cat-finite-obj-coproduct-op:
op-ntcf $\pi : A >_{CF.\Pi.f\text{in}} P : I \mapsto_{C\alpha} \text{op-cat } \mathfrak{C}$
by (intro is-cat-finite-obj-coproductI)
(
cs-concl **cs-shallow**
cs-simp: cat-op-simps
cs-intro: cat-fin-obj-prod-index-in- ω cat-cs-intros cat-op-intros
)

lemma (in is-cat-finite-obj-prod) is-cat-finite-obj-coproduct-op'[cat-op-intros]:
assumes $\mathfrak{C}' = \text{op-cat } \mathfrak{C}$
shows op-ntcf $\pi : A >_{CF.\Pi.f\text{in}} P : I \mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (rule is-cat-finite-obj-coproduct-op)

lemmas [cat-op-intros] = is-cat-finite-obj-prod.is-cat-finite-obj-coproduct-op'

lemma (in is-cat-finite-obj-coproduct) is-cat-finite-obj-prod-op:
op-ntcf $\pi : U <_{CF.\Pi.f\text{in}} A : I \mapsto_{C\alpha} \text{op-cat } \mathfrak{C}$
by (intro is-cat-finite-obj-prodI)
(
cs-concl **cs-shallow**
cs-simp: cat-op-simps
cs-intro: cat-fin-obj-coproduct-index-in- ω cat-cs-intros cat-op-intros
)

lemma (in is-cat-finite-obj-coproduct) is-cat-finite-obj-prod-op'[cat-op-intros]:
assumes $\mathfrak{C}' = \text{op-cat } \mathfrak{C}$
shows op-ntcf $\pi : U <_{CF.\Pi.f\text{in}} A : I \mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (rule is-cat-finite-obj-prod-op)

lemmas [cat-op-intros] = is-cat-finite-obj-coproduct.is-cat-finite-obj-prod-op'

5.4 Product and coproduct of two objects

5.4.1 Definition and elementary properties

locale is-cat-obj-prod-2 = is-cat-obj-prod $\alpha \langle 2_{\mathbb{N}} \rangle \langle \text{if2 } a \ b \rangle \mathfrak{C} \ P \ \pi$
for $\alpha \ a \ b \ \mathfrak{C} \ P \ \pi$

syntax -is-cat-obj-prod-2 :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow \text{bool}$
($\langle \langle - \ / \ - <_{CF.\times} \{ -, - \} \ / \ 2_C \mapsto_{C1} - \rangle \ [51, 51, 51, 51, 51] \ 51 \rangle$)

syntax-consts -is-cat-obj-prod-2 \cong is-cat-obj-prod-2

translations $\pi : P <_{CF.\times} \{ a, b \} : 2_C \mapsto_{C\alpha} \mathfrak{C} \cong$
CONST is-cat-obj-prod-2 $\alpha \ a \ b \ \mathfrak{C} \ P \ \pi$

locale is-cat-obj-coproduct-2 = is-cat-obj-coproduct $\alpha \langle 2_{\mathbb{N}} \rangle \langle \text{if2 } a \ b \rangle \mathfrak{C} \ P \ \pi$
for $\alpha \ a \ b \ \mathfrak{C} \ P \ \pi$

syntax -is-cat-obj-coproduct-2 :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow \text{bool}$
($\langle \langle - \ / \ \{ -, - \} >_{CF.\cup} - \ / \ 2_C \mapsto_{C1} - \rangle \ [51, 51, 51, 51, 51] \ 51 \rangle$)

syntax-consts -is-cat-obj-coproduct-2 \cong is-cat-obj-coproduct-2

translations $\pi : \{a, b\} >_{CF, \cup} U : \mathcal{2}_C \mapsto_{C\alpha} \mathfrak{C} \Leftrightarrow$
 $CONST \text{ is-cat-obj-coprod-2 } \alpha \ a \ b \ \mathfrak{C} \ U \ \pi$

abbreviation *proj-fst* **where** *proj-fst* $\pi \equiv \text{vpfst } (\pi(\text{NTMap}))$

abbreviation *proj-snd* **where** *proj-snd* $\pi \equiv \text{vpsnd } (\pi(\text{NTMap}))$

Rules.

lemma (**in** *is-cat-obj-prod-2*) *is-cat-obj-prod-2-axioms'*[*cat-lim-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $P' = P$ **and** $a' = a$ **and** $b' = b$ **and** $\mathfrak{C}' = \mathfrak{C}$
shows $\pi : P' <_{CF, \times} \{a', b'\} : \mathcal{2}_C \mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-obj-prod-2-axioms*)

mk-ide **rf** *is-cat-obj-prod-2-def*
|*intro is-cat-obj-prod-2I*|
|*dest is-cat-obj-prod-2D[dest]*|
|*elim is-cat-obj-prod-2E[elim]*|

lemmas [*cat-lim-cs-intros*] = *is-cat-obj-prod-2D*

lemma (**in** *is-cat-obj-coprod-2*) *is-cat-obj-coprod-2-axioms'*[*cat-lim-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $P' = P$ **and** $a' = a$ **and** $b' = b$ **and** $\mathfrak{C}' = \mathfrak{C}$
shows $\pi : \{a', b'\} >_{CF, \cup} P' : \mathcal{2}_C \mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-obj-coprod-2-axioms*)

mk-ide **rf** *is-cat-obj-coprod-2-def*
|*intro is-cat-obj-coprod-2I*|
|*dest is-cat-obj-coprod-2D[dest]*|
|*elim is-cat-obj-coprod-2E[elim]*|

lemmas [*cat-lim-cs-intros*] = *is-cat-obj-coprod-2D*

Duality.

lemma (**in** *is-cat-obj-prod-2*) *is-cat-obj-coprod-2-op*:
op-ntcf $\pi : \{a, b\} >_{CF, \cup} P : \mathcal{2}_C \mapsto_{C\alpha} \text{op-cat } \mathfrak{C}$
by (*rule is-cat-obj-coprod-2I[OF is-cat-obj-coprod-op]*)

lemma (**in** *is-cat-obj-prod-2*) *is-cat-obj-coprod-2-op'*[*cat-op-intros*]:
assumes $\mathfrak{C}' = \text{op-cat } \mathfrak{C}$
shows *op-ntcf* $\pi : \{a, b\} >_{CF, \cup} P : \mathcal{2}_C \mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-obj-coprod-2-op*)

lemmas [*cat-op-intros*] = *is-cat-obj-prod-2.is-cat-obj-coprod-2-op'*

lemma (**in** *is-cat-obj-coprod-2*) *is-cat-obj-prod-2-op*:
op-ntcf $\pi : P <_{CF, \times} \{a, b\} : \mathcal{2}_C \mapsto_{C\alpha} \text{op-cat } \mathfrak{C}$
by (*rule is-cat-obj-prod-2I[OF is-cat-obj-prod-op]*)

lemma (**in** *is-cat-obj-coprod-2*) *is-cat-obj-prod-2-op'*[*cat-op-intros*]:
assumes $\mathfrak{C}' = \text{op-cat } \mathfrak{C}$
shows *op-ntcf* $\pi : P <_{CF, \times} \{a, b\} : \mathcal{2}_C \mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-obj-prod-2-op*)

lemmas [*cat-op-intros*] = *is-cat-obj-coprod-2.is-cat-obj-prod-2-op'*

Product/coproduct of two objects is a finite product/coproduct.

sublocale *is-cat-obj-prod-2* \subseteq *is-cat-finite-obj-prod* $\alpha \ \langle \mathcal{2}_N \rangle \ \langle \text{if2 } a \ b \rangle \ \mathfrak{C} \ P \ \pi$
proof(*intro is-cat-finite-obj-prodI*)

show $2_{\mathbf{N}} \in_{\circ} \omega$ by *simp*
qed (*cs-concl cs-shallow cs-simp: two[symmetric] cs-intro: cat-lim-cs-intros*)

sublocale *is-cat-obj-coproduct-2* \subseteq *is-cat-finite-obj-coproduct* $\alpha \langle 2_{\mathbf{N}} \rangle \langle \text{if2 } a \ b \rangle \mathfrak{C} \ P \ \pi$
proof(*intro is-cat-finite-obj-coproductI*)

show $2_{\mathbf{N}} \in_{\circ} \omega$ by *simp*
qed (*cs-concl cs-shallow cs-simp: two[symmetric] cs-intro: cat-lim-cs-intros*)

Elementary properties.

lemma (in *is-cat-obj-prod-2*) *cat-obj-prod-2-lr-in-Obj*:

shows *cat-obj-prod-2-left-in-Obj*[*cat-lim-cs-intros*]: $a \in_{\circ} \mathfrak{C}(\text{Obj})$
and *cat-obj-prod-2-right-in-Obj*[*cat-lim-cs-intros*]: $b \in_{\circ} \mathfrak{C}(\text{Obj})$

proof–

have $0: 0 \in_{\circ} 2_{\mathbf{N}}$ **and** $1: 1_{\mathbf{N}} \in_{\circ} 2_{\mathbf{N}}$ by *simp-all*

show $a \in_{\circ} \mathfrak{C}(\text{Obj})$ **and** $b \in_{\circ} \mathfrak{C}(\text{Obj})$

by

(
intro
cf-discrete-selector-vrange[OF 0, simplified]
cf-discrete-selector-vrange[OF 1, simplified]
)+

qed

lemmas [*cat-lim-cs-intros*] = *is-cat-obj-prod-2.cat-obj-prod-2-lr-in-Obj*

lemma (in *is-cat-obj-coproduct-2*) *cat-obj-coproduct-2-lr-in-Obj*:

shows *cat-obj-coproduct-2-left-in-Obj*[*cat-lim-cs-intros*]: $a \in_{\circ} \mathfrak{C}(\text{Obj})$
and *cat-obj-coproduct-2-right-in-Obj*[*cat-lim-cs-intros*]: $b \in_{\circ} \mathfrak{C}(\text{Obj})$

by

(
intro is-cat-obj-prod-2.cat-obj-prod-2-lr-in-Obj[
OF is-cat-obj-prod-2-op, unfolded cat-op-simps
]
)+

lemmas [*cat-lim-cs-intros*] = *is-cat-obj-coproduct-2.cat-obj-coproduct-2-lr-in-Obj*

Utilities/help lemmas.

lemma *helper-I2-proj-fst-proj-snd-iff*:

$(\forall j \in_{\circ} 2_{\mathbf{N}}. \pi'(\text{NTMap})(j) = \pi(\text{NTMap})(j) \circ_{A\mathfrak{C}} f') \leftrightarrow$
 $(\text{proj-fst } \pi' = \text{proj-fst } \pi \circ_{A\mathfrak{C}} f' \wedge \text{proj-snd } \pi' = \text{proj-snd } \pi \circ_{A\mathfrak{C}} f')$
unfolding two by auto

lemma *helper-I2-proj-fst-proj-snd-iff'*:

$(\forall j \in_{\circ} 2_{\mathbf{N}}. \pi'(\text{NTMap})(j) = f' \circ_{A\mathfrak{C}} \pi(\text{NTMap})(j)) \leftrightarrow$
 $(\text{proj-fst } \pi' = f' \circ_{A\mathfrak{C}} \text{proj-fst } \pi \wedge \text{proj-snd } \pi' = f' \circ_{A\mathfrak{C}} \text{proj-snd } \pi)$
unfolding two by auto

5.4.2 Universal property

lemma (in *is-cat-obj-prod-2*) *cat-obj-prod-2-unique-cone'*:

assumes $\pi' : P' <_{CF.cone} \rightarrow : (2_{\mathbf{N}}) (\text{if2 } a \ b) \mathfrak{C} : :_{\mathfrak{C}} (2_{\mathbf{N}}) \mapsto \mapsto_{\mathfrak{C}} \alpha \ \mathfrak{C}$

shows

$\exists ! f'. f' : P' \mapsto_{\mathfrak{C}} P \wedge$
 $\text{proj-fst } \pi' = \text{proj-fst } \pi \circ_{A\mathfrak{C}} f' \wedge$
 $\text{proj-snd } \pi' = \text{proj-snd } \pi \circ_{A\mathfrak{C}} f'$

by

(

$$\text{rule cat-obj-prod-unique-cone}'[$$

$$OF \text{ assms, unfolded helper-I2-proj-fst-proj-snd-iff}$$

$$]$$

$$)$$

lemma (in is-cat-obj-prod-2) cat-obj-prod-2-unique:
assumes $\pi' : P' <_{CF.\times} \{a,b\} : 2_C \mapsto C\alpha \mathfrak{C}$
shows $\exists! f'. f' : P' \mapsto_{\mathfrak{C}} P \wedge \pi' = \pi \cdot_{NTCF} ntcf\text{-const} (:_C (2_{\mathbb{N}})) \mathfrak{C} f'$
by (rule cat-obj-prod-unique[OF is-cat-obj-prod-2D[OF assms]])

lemma (in is-cat-obj-prod-2) cat-obj-prod-2-unique':
assumes $\pi' : P' <_{CF.\times} \{a,b\} : 2_C \mapsto C\alpha \mathfrak{C}$
shows
 $\exists! f'. f' : P' \mapsto_{\mathfrak{C}} P \wedge$
 $proj\text{-fst } \pi' = proj\text{-fst } \pi \circ_{A\mathfrak{C}} f' \wedge$
 $proj\text{-snd } \pi' = proj\text{-snd } \pi \circ_{A\mathfrak{C}} f'$
by
(
$$\text{rule cat-obj-prod-unique}'[$$

$$OF \text{ is-cat-obj-prod-2D}[OF \text{ assms}],$$

$$\text{unfolded helper-I2-proj-fst-proj-snd-iff}$$

$$]$$
)

lemma (in is-cat-obj-coprod-2) cat-obj-coprod-2-unique-cocone':
assumes $\pi' : \rightarrow : (2_{\mathbb{N}}) (if2 \ a \ b) \mathfrak{C} >_{CF.cocone} P' : :_C (2_{\mathbb{N}}) \mapsto C\alpha \mathfrak{C}$
shows
 $\exists! f'. f' : P \mapsto_{\mathfrak{C}} P' \wedge$
 $proj\text{-fst } \pi' = f' \circ_{A\mathfrak{C}} proj\text{-fst } \pi \wedge$
 $proj\text{-snd } \pi' = f' \circ_{A\mathfrak{C}} proj\text{-snd } \pi$
by
(
$$\text{rule cat-obj-coprod-unique-cocone}'[$$

$$OF \text{ assms, unfolded helper-I2-proj-fst-proj-snd-iff}'$$

$$]$$
)

lemma (in is-cat-obj-coprod-2) cat-obj-coprod-2-unique:
assumes $\pi' : \{a,b\} >_{CF.\sqcup} P' : 2_C \mapsto C\alpha \mathfrak{C}$
shows $\exists! f'. f' : P \mapsto_{\mathfrak{C}} P' \wedge \pi' = ntcf\text{-const} (:_C (2_{\mathbb{N}})) \mathfrak{C} f' \cdot_{NTCF} \pi$
by (rule cat-obj-coprod-unique[OF is-cat-obj-coprod-2D[OF assms]])

lemma (in is-cat-obj-coprod-2) cat-obj-coprod-2-unique':
assumes $\pi' : \{a,b\} >_{CF.\sqcup} P' : 2_C \mapsto C\alpha \mathfrak{C}$
shows
 $\exists! f'. f' : P \mapsto_{\mathfrak{C}} P' \wedge$
 $proj\text{-fst } \pi' = f' \circ_{A\mathfrak{C}} proj\text{-fst } \pi \wedge$
 $proj\text{-snd } \pi' = f' \circ_{A\mathfrak{C}} proj\text{-snd } \pi$
by
(
$$\text{rule cat-obj-coprod-unique}'[$$

$$OF \text{ is-cat-obj-coprod-2D}[OF \text{ assms}],$$

$$\text{unfolded helper-I2-proj-fst-proj-snd-iff}'$$

$$]$$
)

lemma cat-obj-prod-2-ex-is-iso-arr:
assumes $\pi : P <_{CF.\times} \{a,b\} : 2_C \mapsto C\alpha \mathfrak{C}$

and $\pi' : P' <_{CF.\times} \{a,b\} : \mathcal{2}_C \mapsto \mapsto C\alpha \mathfrak{C}$
obtains f **where** $f : P' \mapsto_{iso} \mathfrak{C} P$ **and** $\pi' = \pi \cdot_{NTCF} ntcf\text{-const} (:_C (\mathcal{2}_N)) \mathfrak{C} f$
proof-
interpret $\pi : is\text{-cat-obj-prod-2} \alpha a b \mathfrak{C} P \pi$ **by** (*rule assms(1)*)
interpret $\pi' : is\text{-cat-obj-prod-2} \alpha a b \mathfrak{C} P' \pi'$ **by** (*rule assms(2)*)
from that show *?thesis*
by
(
elim cat-obj-prod-ex-is-iso-arr [
OF $\pi.is\text{-cat-obj-prod-axioms} \pi'.is\text{-cat-obj-prod-axioms}$
]
)
qed

lemma *cat-obj-coprod-2-ex-is-iso-arr*:
assumes $\pi : \{a,b\} >_{CF.\cup} U : \mathcal{2}_C \mapsto \mapsto C\alpha \mathfrak{C}$
and $\pi' : \{a,b\} >_{CF.\cup} U' : \mathcal{2}_C \mapsto \mapsto C\alpha \mathfrak{C}$
obtains f **where** $f : U \mapsto_{iso} \mathfrak{C} U'$ **and** $\pi' = ntcf\text{-const} (:_C (\mathcal{2}_N)) \mathfrak{C} f \cdot_{NTCF} \pi$
proof-
interpret $\pi : is\text{-cat-obj-coprod-2} \alpha a b \mathfrak{C} U \pi$ **by** (*rule assms(1)*)
interpret $\pi' : is\text{-cat-obj-coprod-2} \alpha a b \mathfrak{C} U' \pi'$ **by** (*rule assms(2)*)
from that show *?thesis*
by
(
elim cat-obj-coprod-ex-is-iso-arr [
OF $\pi.is\text{-cat-obj-coprod-axioms} \pi'.is\text{-cat-obj-coprod-axioms}$
]
)
qed

5.5 Projection cone

5.5.1 Definition and elementary properties

definition *ntcf-obj-prod-base* $:: V \Rightarrow V \Rightarrow (V \Rightarrow V) \Rightarrow V \Rightarrow (V \Rightarrow V) \Rightarrow V$
where *ntcf-obj-prod-base* $\mathfrak{C} I F P f =$
 $[(\lambda j \in \circ :_C I(\text{Obj}). f j), cf\text{-const} (:_C I) \mathfrak{C} P, \text{:}\rightarrow : I F \mathfrak{C}, :_C I, \mathfrak{C}]$.

definition *ntcf-obj-coprod-base* $:: V \Rightarrow V \Rightarrow (V \Rightarrow V) \Rightarrow V \Rightarrow (V \Rightarrow V) \Rightarrow V$
where *ntcf-obj-coprod-base* $\mathfrak{C} I F P f =$
 $[(\lambda j \in \circ :_C I(\text{Obj}). f j), \text{:}\rightarrow : I F \mathfrak{C}, cf\text{-const} (:_C I) \mathfrak{C} P, :_C I, \mathfrak{C}]$.

Components.

lemma *ntcf-obj-prod-base-components*:
shows *ntcf-obj-prod-base* $\mathfrak{C} I F P f(\text{NTMap}) = (\lambda j \in \circ :_C I(\text{Obj}). f j)$
and *ntcf-obj-prod-base* $\mathfrak{C} I F P f(\text{NTDom}) = cf\text{-const} (:_C I) \mathfrak{C} P$
and *ntcf-obj-prod-base* $\mathfrak{C} I F P f(\text{NTCod}) = \text{:}\rightarrow : I F \mathfrak{C}$
and *ntcf-obj-prod-base* $\mathfrak{C} I F P f(\text{NTDGDom}) = :_C I$
and *ntcf-obj-prod-base* $\mathfrak{C} I F P f(\text{NTDGCod}) = \mathfrak{C}$
unfolding *ntcf-obj-prod-base-def nt-field-simps*
by (*simp-all add: nat-omega-simps*)

lemma *ntcf-obj-coprod-base-components*:
shows *ntcf-obj-coprod-base* $\mathfrak{C} I F P f(\text{NTMap}) = (\lambda j \in \circ :_C I(\text{Obj}). f j)$
and *ntcf-obj-coprod-base* $\mathfrak{C} I F P f(\text{NTDom}) = \text{:}\rightarrow : I F \mathfrak{C}$
and *ntcf-obj-coprod-base* $\mathfrak{C} I F P f(\text{NTCod}) = cf\text{-const} (:_C I) \mathfrak{C} P$
and *ntcf-obj-coprod-base* $\mathfrak{C} I F P f(\text{NTDGDom}) = :_C I$
and *ntcf-obj-coprod-base* $\mathfrak{C} I F P f(\text{NTDGCod}) = \mathfrak{C}$

unfolding *ntcf-obj-coproduct-base-def nt-field-simps*
by (*simp-all add: nat-omega-simps*)

Duality.

lemma (**in** *cf-discrete*) *op-ntcf-ntcf-obj-coproduct-base[cat-op-simps]*:

op-ntcf (ntcf-obj-coproduct-base \mathfrak{C} I F P f) =
ntcf-obj-product-base (op-cat \mathfrak{C}) I F P f

proof–

note [*cat-op-simps*] = *the-cat-discrete-op[OF cf-discrete-vdomain-vsubset-Vset]*

show *?thesis*

unfolding

ntcf-obj-product-base-def ntcf-obj-coproduct-base-def op-ntcf-def nt-field-simps

by (*simp add: nat-omega-simps cat-op-simps*)

qed

lemma (**in** *cf-discrete*) *op-ntcf-ntcf-obj-product-base[cat-op-simps]*:

op-ntcf (ntcf-obj-product-base \mathfrak{C} I F P f) =
ntcf-obj-coproduct-base (op-cat \mathfrak{C}) I F P f

proof–

note [*cat-op-simps*] = *the-cat-discrete-op[OF cf-discrete-vdomain-vsubset-Vset]*

show *?thesis*

unfolding

ntcf-obj-product-base-def ntcf-obj-coproduct-base-def op-ntcf-def nt-field-simps

by (*simp add: nat-omega-simps cat-op-simps*)

qed

5.5.2 Natural transformation map

mk-VLambda *ntcf-obj-product-base-components(1)*

|*vsu ntcf-obj-product-base-NTMap-vsuv[cat-cs-intros]*|

|*vdomain ntcf-obj-product-base-NTMap-vdomain[cat-cs-simps]*|

|*app ntcf-obj-product-base-NTMap-app[cat-cs-simps]*|

mk-VLambda *ntcf-obj-coproduct-base-components(1)*

|*vsu ntcf-obj-coproduct-base-NTMap-vsuv[cat-cs-intros]*|

|*vdomain ntcf-obj-coproduct-base-NTMap-vdomain[cat-cs-simps]*|

|*app ntcf-obj-coproduct-base-NTMap-app[cat-cs-simps]*|

5.5.3 Projection natural transformation is a cone

lemma (**in** *tm-cf-discrete*) *tm-cf-discrete-ntcf-obj-product-base-is-cat-cone*:

assumes $P \in_{\circ} \mathfrak{C}(\text{Obj})$ **and** $\bigwedge a. a \in_{\circ} I \implies f a : P \mapsto_{\mathfrak{C}} F a$

shows *ntcf-obj-product-base \mathfrak{C} I F P f : P <_{CF.cone} \multimap : I F \mathfrak{C} : :_{\mathfrak{C}} I \mapsto_{\mathfrak{C}\alpha} \mathfrak{C}*

proof(*intro is-cat-coneI is-tm-ntcfI' is-ntcfI'*)

from *assms(2)* **have** [*cat-cs-intros*]:

$\llbracket a \in_{\circ} I; P' = P; Fa = F a \rrbracket \implies f a : P' \mapsto_{\mathfrak{C}} Fa$ **for** $a P' Fa$

by *simp*

show *vfsequence (ntcf-obj-product-base \mathfrak{C} I F P f)*

unfolding *ntcf-obj-product-base-def* **by** *auto*

show *vcard (ntcf-obj-product-base \mathfrak{C} I F P f) = 5_N*

unfolding *ntcf-obj-product-base-def* **by** (*auto simp: nat-omega-simps*)

from *assms* **show** *cf-const (:_C I) \mathfrak{C} P : :_C I \mapsto_{\mathfrak{C}\alpha} \mathfrak{C}*

by

(

cs-concl

cs-intro:

cf-discrete-vdomain-vsubset-Vset

cat-discrete-cs-intros

cat-cs-intros

)

show $\vdash: I F \mathfrak{C} : :_C I \mapsto \mapsto_{C\alpha} \mathfrak{C}$

by (*cs-concl cs-shallow cs-intro: cat-discrete-cs-intros*)

show *ntcf-obj-prod-base* $\mathfrak{C} I F P f(\backslash NTMap)(\backslash a) :$

cf-const $(: _C I) \mathfrak{C} P(\backslash ObjMap)(\backslash a) \mapsto_{\mathfrak{C}} \vdash: I F \mathfrak{C}(\backslash ObjMap)(\backslash a)$

if $a \in_{\circ} :_C I(\backslash Obj)$ **for** a

proof-

from *that have* $a \in_{\circ} I$ **unfolding** *the-cat-discrete-components* **by** *simp*

from *that this show* *?thesis*

by

 (

cs-concl cs-shallow

cs-simp: *cat-cs-simps cat-discrete-cs-simps cs-intro: cat-cs-intros*

)

qed

show

ntcf-obj-prod-base $\mathfrak{C} I F P f(\backslash NTMap)(\backslash b) \circ_A \mathfrak{C}$

cf-const $(: _C I) \mathfrak{C} P(\backslash ArrMap)(\backslash g) =$

$\vdash: I F \mathfrak{C}(\backslash ArrMap)(\backslash g) \circ_A \mathfrak{C} \text{ } \textit{ntcf-obj-prod-base} \mathfrak{C} I F P f(\backslash NTMap)(\backslash a)$

if $g : a \mapsto :_C I b$ **for** $a b g$

proof-

note $g = \textit{the-cat-discrete-is-arrD}[OF \textit{that}]$

from *that* $g(4)[\textit{unfolded } g(7-9)] g(1)[\textit{unfolded } g(7-9)]$ **show** *?thesis*

unfolding $g(7-9)$

by

 (

cs-concl

cs-simp: *cat-cs-simps cat-discrete-cs-simps*

cs-intro:

cf-discrete-vdomain-vsubset-Vset

cat-cs-intros cat-discrete-cs-intros

)

qed

qed

(

auto simp:

assms

ntcf-obj-prod-base-components

tm-cf-discrete-the-cf-discrete-is-tm-functor

)

lemma (*in tm-cf-discrete*) *tm-cf-discrete-ntcf-obj-coprod-base-is-cat-cocone:*

assumes $P \in_{\circ} \mathfrak{C}(\backslash Obj)$ **and** $\bigwedge a. a \in_{\circ} I \implies f a : F a \mapsto_{\mathfrak{C}} P$

shows *ntcf-obj-coprod-base* $\mathfrak{C} I F P f :$

$\vdash: I F \mathfrak{C} >_{CF.cocone} P : :_C I \mapsto \mapsto_{C\alpha} \mathfrak{C}$

proof-

note [*cat-op-simps*] =

the-cat-discrete-op[*OF cf-discrete-vdomain-vsubset-Vset*]

cf-discrete.op-ntcf-ntcf-obj-prod-base[*OF cf-discrete-op*]

cf-discrete.cf-discrete-the-cf-discrete-op[*OF cf-discrete-op*]

have *op-ntcf* (*ntcf-obj-coprod-base* $\mathfrak{C} I F P f$) :

$P <_{CF.cone} \textit{op-cf} (\vdash: I F \mathfrak{C}) : \textit{op-cat} (:_C I) \mapsto \mapsto_{C\alpha} \textit{op-cat} \mathfrak{C}$

unfolding *cat-op-simps*

by

(

rule tm-cf-discrete.tm-cf-discrete-ntcf-obj-prod-base-is-cat-cone[

OF tm-cf-discrete-op, unfolded cat-op-simps, OF assms

)
)
from *is-cat-cone.is-cat-cocone-op*[*OF this, unfolded cat-op-simps*]
show *?thesis* .
qed

lemma (**in** *tm-cf-discrete*) *tm-cf-discrete-ntcf-obj-prod-base-is-cat-obj-prod*:

assumes $P \in_{\circ} \mathfrak{C}(\text{Obj})$
and $\bigwedge a. a \in_{\circ} I \implies f a : P \mapsto_{\mathfrak{C}} F a$
and $\bigwedge u' r'.$
 $\llbracket u' : r' <_{CF.cocone} \text{>}: I F \mathfrak{C} : :_C I \mapsto_{C\alpha} \mathfrak{C} \rrbracket \implies$
 $\exists ! f'.$
 $f' : r' \mapsto_{\mathfrak{C}} P \wedge$
 $u' = \text{ntcf-obj-prod-base } \mathfrak{C} I F P f \cdot_{NTCF} \text{ntcf-const } (:_C I) \mathfrak{C} f'$
shows *ntcf-obj-prod-base* $\mathfrak{C} I F P f : P <_{CF.\Pi} F : I \mapsto_{C\alpha} \mathfrak{C}$

proof

(
intro
is-cat-obj-prodI
is-cat-limitI
tm-cf-discrete-ntcf-obj-prod-base-is-cat-cone[*OF assms(1,2), simplified*]
assms(1,3)
)

show *cf-discrete* $\alpha I F \mathfrak{C}$
by (*cs-concl cs-shallow cs-intro: cat-small-discrete-cs-intros*)

qed

lemma (**in** *tm-cf-discrete*) *tm-cf-discrete-ntcf-obj-coprod-base-is-cat-obj-coprod*:

assumes $P \in_{\circ} \mathfrak{C}(\text{Obj})$
and $\bigwedge a. a \in_{\circ} I \implies f a : F a \mapsto_{\mathfrak{C}} P$
and $\bigwedge u' r'. \llbracket u' : \text{>}: I F \mathfrak{C} >_{CF.cocone} r' : :_C I \mapsto_{C\alpha} \mathfrak{C} \rrbracket \implies$
 $\exists ! f'.$
 $f' : P \mapsto_{\mathfrak{C}} r' \wedge$
 $u' = \text{ntcf-const } (:_C I) \mathfrak{C} f' \cdot_{NTCF} \text{ntcf-obj-coprod-base } \mathfrak{C} I F P f$
shows *ntcf-obj-coprod-base* $\mathfrak{C} I F P f : F >_{CF.\Pi} P : I \mapsto_{C\alpha} \mathfrak{C}$
(is <?nc : F >_{CF.\Pi} P : I \mapsto_{C\alpha} \mathfrak{C})

proof–

let *?np* = *ntcf-obj-prod-base (op-cat* \mathfrak{C}) *I F P f*
interpret *is-cat-cocone* $\alpha P \langle :_C I \rangle \mathfrak{C} \langle \text{>}: I F \mathfrak{C} \rangle ?nc$
by (*intro tm-cf-discrete-ntcf-obj-coprod-base-is-cat-cocone*[*OF assms(1,2)*])

note [*cat-op-simps*] =
the-cat-discrete-op[*OF cf-discrete-vdomain-vsubset-Vset*]
cf-discrete.op-ntcf-ntcf-obj-prod-base[*OF cf-discrete-op*]
cf-discrete.cf-discrete-the-cf-discrete-op[*OF cf-discrete-op*]

have $\exists ! f'.$
 $f' : P \mapsto_{\mathfrak{C}} r \wedge$
 $u = ?np \cdot_{NTCF} \text{ntcf-const } (:_C I) (\text{op-cat } \mathfrak{C}) f'$
if $u : r <_{CF.cocone} \text{>}: I F (\text{op-cat } \mathfrak{C}) : :_C I \mapsto_{C\alpha} \text{op-cat } \mathfrak{C}$ **for** $u r$

proof–

interpret $u : \text{is-cat-cone } \alpha r \langle :_C I \rangle \langle \text{op-cat } \mathfrak{C} \rangle \langle \text{>}: I F (\text{op-cat } \mathfrak{C}) \rangle u$
by (*rule that*)

from *assms(3)*[*OF u.is-cat-cocone-op*[*unfolded cat-op-simps*]] **obtain** g

where $g : P \mapsto_{\mathfrak{C}} r$
and *op-u: op-ntcf* $u = \text{ntcf-const } (:_C I) \mathfrak{C} g \cdot_{NTCF} ?nc$
and *g-unique*:
 $\llbracket g' : P \mapsto_{\mathfrak{C}} r; \text{op-ntcf } u = \text{ntcf-const } (:_C I) \mathfrak{C} g' \cdot_{NTCF} ?nc \rrbracket \implies$
 $g' = g$
for g'

```

by metis
show ?thesis
proof(intro ex1I conjI; (elim conjE)?)
  from op-u have
    op-ntcf (op-ntcf u) = op-ntcf (ntcf-const (:C I)  $\mathfrak{C}$  g  $\cdot_{NTCF}$  ?nc)
  by simp
  from this g show u = ?np  $\cdot_{NTCF}$  ntcf-const (:C I) (op-cat  $\mathfrak{C}$ ) g
  by (cs-prems cs-simp: cat-op-simps cs-intro: cat-cs-intros)
  fix g' assume prems:
    g' : P  $\mapsto_{\mathfrak{C}}$  r
    u = ?np  $\cdot_{NTCF}$  ntcf-const (:C I) (op-cat  $\mathfrak{C}$ ) g'
  from prems(2) have
    op-ntcf u = op-ntcf (?np  $\cdot_{NTCF}$  ntcf-const (:C I) (op-cat  $\mathfrak{C}$ ) g')
  by simp
  from this prems(1) g have op-ntcf u = ntcf-const (:C I)  $\mathfrak{C}$  g'  $\cdot_{NTCF}$  ?nc
  by
    (
      subst (asm)
      the-cat-discrete-op[OF cf-discrete-vdomain-vsubset-Vset, symmetric]
    )
    (
      cs-prems
      cs-simp:
        cat-op-simps
        op-ntcf-ntcf-vcomp[symmetric]
        is-ntcf.ntcf-op-ntcf-op-ntcf
        op-ntcf-ntcf-obj-coprod-base[symmetric]
        op-ntcf-ntcf-const[symmetric]
      cs-intro: cat-cs-intros cat-op-intros
    )
  from g-unique[OF prems(1) this] show g' = g .
qed (rule g)
qed
from is-cat-obj-prod.is-cat-obj-coprod-op
[
  OF tm-cf-discrete.tm-cf-discrete-ntcf-obj-prod-base-is-cat-obj-prod
  [
    OF tm-cf-discrete-op,
    unfolded cat-op-simps,
    OF assms(1,2) this,
    folded op-ntcf-ntcf-obj-coprod-base
  ],
  unfolded cat-op-simps
]
show ?nc : F  $\succ_{CF}$ .  $\sqcup$  P : I  $\mapsto_{\mapsto_{C\alpha}}$   $\mathfrak{C}$ .
qed

```

6 Pullbacks and pushouts as limits and colimits

6.1 Pullback and pushout

6.1.1 Definition and elementary properties

The definitions and the elementary properties of the pullbacks and the pushouts can be found, for example, in Chapter III-3 and Chapter III-4 in [9].

locale *is-cat-pullback* =
is-cat-limit $\alpha \langle \rightarrow \leftarrow_C \rangle \mathcal{C} \langle \mathbf{a} \rightarrow \mathbf{g} \rightarrow \mathbf{o} \leftarrow \mathbf{f} \leftarrow \mathbf{b} \rangle_{CF\mathcal{C}} X x +$
cf-scospans $\alpha \mathbf{a} \mathbf{g} \mathbf{o} \mathbf{f} \mathbf{b} \mathcal{C}$
for $\alpha \mathbf{a} \mathbf{g} \mathbf{o} \mathbf{f} \mathbf{b} \mathcal{C} X x$

syntax *-is-cat-pullback* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
 $(\langle \langle - \text{ :/ } - \langle_{CF.pb} \dashrightarrow \dashrightarrow \dashleftarrow \dashleftarrow \dashrightarrow_{C^1} - \rangle \rangle [51, 51, 51, 51, 51, 51, 51, 51] 51)$

syntax-consts *-is-cat-pullback* \equiv *is-cat-pullback*

translations $x : X \langle_{CF.pb} \mathbf{a} \rightarrow \mathbf{g} \rightarrow \mathbf{o} \leftarrow \mathbf{f} \leftarrow \mathbf{b} \rangle \mapsto_{C\alpha} \mathcal{C} \equiv$
CONST is-cat-pullback $\alpha \mathbf{a} \mathbf{g} \mathbf{o} \mathbf{f} \mathbf{b} \mathcal{C} X x$

locale *is-cat-pushout* =
is-cat-colimit $\alpha \langle \leftarrow \rightarrow_C \rangle \mathcal{C} \langle \mathbf{a} \leftarrow \mathbf{g} \leftarrow \mathbf{o} \rightarrow \mathbf{f} \rightarrow \mathbf{b} \rangle_{CF\mathcal{C}} X x +$
cf-sspan $\alpha \mathbf{a} \mathbf{g} \mathbf{o} \mathbf{f} \mathbf{b} \mathcal{C}$
for $\alpha \mathbf{a} \mathbf{g} \mathbf{o} \mathbf{f} \mathbf{b} \mathcal{C} X x$

syntax *-is-cat-pushout* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
 $(\langle \langle - \text{ :/ } \leftarrow \leftarrow \dashrightarrow \dashrightarrow \dashrightarrow_{CF.po} - \mapsto_{C^1} - \rangle \rangle [51, 51, 51, 51, 51, 51, 51, 51] 51)$

syntax-consts *-is-cat-pushout* \equiv *is-cat-pushout*

translations $x : \mathbf{a} \leftarrow \mathbf{g} \leftarrow \mathbf{o} \rightarrow \mathbf{f} \rightarrow \mathbf{b} \rangle_{CF.po} X \mapsto_{C\alpha} \mathcal{C} \equiv$
CONST is-cat-pushout $\alpha \mathbf{a} \mathbf{g} \mathbf{o} \mathbf{f} \mathbf{b} \mathcal{C} X x$

Rules.

lemma (in *is-cat-pullback*) *is-cat-pullback-axioms'*[*cat-lim-cs-intros*]:

assumes $\alpha' = \alpha$
and $\mathbf{a}' = \mathbf{a}$
and $\mathbf{g}' = \mathbf{g}$
and $\mathbf{o}' = \mathbf{o}$
and $\mathbf{f}' = \mathbf{f}$
and $\mathbf{b}' = \mathbf{b}$
and $\mathcal{C}' = \mathcal{C}$
and $X' = X$

shows $x : X' \langle_{CF.pb} \mathbf{a}' \rightarrow \mathbf{g}' \rightarrow \mathbf{o}' \leftarrow \mathbf{f}' \leftarrow \mathbf{b}' \rangle \mapsto_{C\alpha'} \mathcal{C}'$
unfolding *assms* **by** (rule *is-cat-pullback-axioms*)

mk-ide **rf** *is-cat-pullback-def*

|*intro is-cat-pullbackI*||
|*dest is-cat-pullbackD*[*dest*]|
|*elim is-cat-pullbackE*[*elim*]|

lemmas [*cat-lim-cs-intros*] = *is-cat-pullbackD*

lemma (in *is-cat-pushout*) *is-cat-pushout-axioms'*[*cat-lim-cs-intros*]:

assumes $\alpha' = \alpha$
and $\mathbf{a}' = \mathbf{a}$
and $\mathbf{g}' = \mathbf{g}$
and $\mathbf{o}' = \mathbf{o}$
and $\mathbf{f}' = \mathbf{f}$
and $\mathbf{b}' = \mathbf{b}$
and $\mathcal{C}' = \mathcal{C}$

and $X' = X$
shows $x : \mathbf{a}' \leftarrow \mathbf{g}' \leftarrow \mathbf{o}' \rightarrow \mathbf{f}' \rightarrow \mathbf{b}' >_{CF.po} X' \mapsto \mapsto_{C\alpha'} \mathfrak{C}'$
unfolding *assms* by (rule *is-cat-pushout-axioms*)

mk-ide rf *is-cat-pushout-def*
|*intro is-cat-pushoutI*]
|*dest is-cat-pushoutD[dest]*]
|*elim is-cat-pushoutE[elim]*]

lemmas [*cat-lim-cs-intros*] = *is-cat-pushoutD*

Duality.

lemma (in *is-cat-pullback*) *is-cat-pushout-op*:
op-ntcf $x : \mathbf{a} \leftarrow \mathbf{g} \leftarrow \mathbf{o} \rightarrow \mathbf{f} \rightarrow \mathbf{b} >_{CF.po} X \mapsto \mapsto_{C\alpha} \text{op-cat } \mathfrak{C}$
by (*intro is-cat-pushoutI*)
(*cs-concl cs-shallow cs-simp: cat-op-simps cs-intro: cat-op-intros*)+

lemma (in *is-cat-pullback*) *is-cat-pushout-op'*[*cat-op-intros*]:
assumes $\mathfrak{C}' = \text{op-cat } \mathfrak{C}$
shows *op-ntcf* $x : \mathbf{a} \leftarrow \mathbf{g} \leftarrow \mathbf{o} \rightarrow \mathbf{f} \rightarrow \mathbf{b} >_{CF.po} X \mapsto \mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* by (rule *is-cat-pushout-op*)

lemmas [*cat-op-intros*] = *is-cat-pullback.is-cat-pushout-op'*

lemma (in *is-cat-pushout*) *is-cat-pullback-op*:
op-ntcf $x : X <_{CF.pb} \mathbf{a} \rightarrow \mathbf{g} \rightarrow \mathbf{o} \leftarrow \mathbf{f} \leftarrow \mathbf{b} \mapsto \mapsto_{C\alpha} \text{op-cat } \mathfrak{C}$
by (*intro is-cat-pullbackI*)
(*cs-concl cs-shallow cs-simp: cat-op-simps cs-intro: cat-op-intros*)+

lemma (in *is-cat-pushout*) *is-cat-pullback-op'*[*cat-op-intros*]:
assumes $\mathfrak{C}' = \text{op-cat } \mathfrak{C}$
shows *op-ntcf* $x : X <_{CF.pb} \mathbf{a} \rightarrow \mathbf{g} \rightarrow \mathbf{o} \leftarrow \mathbf{f} \leftarrow \mathbf{b} \mapsto \mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* by (rule *is-cat-pullback-op*)

lemmas [*cat-op-intros*] = *is-cat-pushout.is-cat-pullback-op'*

Elementary properties.

lemma *cat-cone-cospan*:
assumes $x : X <_{CF.cone} \langle \mathbf{a} \rightarrow \mathbf{g} \rightarrow \mathbf{o} \leftarrow \mathbf{f} \leftarrow \mathbf{b} \rangle_{CF\mathfrak{C}} : \rightarrow \leftarrow_C \mapsto \mapsto_{C\alpha} \mathfrak{C}$
and *cf-scspan* $\alpha \mathbf{a} \mathbf{g} \mathbf{o} \mathbf{f} \mathbf{b} \mathfrak{C}$
shows $x(\text{NTMap})(\mathbf{o}_{SS}) = \mathbf{g} \circ_{A\mathfrak{C}} x(\text{NTMap})(\mathbf{a}_{SS})$
and $x(\text{NTMap})(\mathbf{o}_{SS}) = \mathbf{f} \circ_{A\mathfrak{C}} x(\text{NTMap})(\mathbf{b}_{SS})$
and $\mathbf{g} \circ_{A\mathfrak{C}} x(\text{NTMap})(\mathbf{a}_{SS}) = \mathbf{f} \circ_{A\mathfrak{C}} x(\text{NTMap})(\mathbf{b}_{SS})$

proof-

interpret $x : \text{is-cat-cone } \alpha X \langle \rightarrow \leftarrow_C \rangle \mathfrak{C} \langle \langle \mathbf{a} \rightarrow \mathbf{g} \rightarrow \mathbf{o} \leftarrow \mathbf{f} \leftarrow \mathbf{b} \rangle_{CF\mathfrak{C}} \rangle x$
by (rule *assms(1)*)

interpret *cospan: cf-scspan* $\alpha \mathbf{a} \mathbf{g} \mathbf{o} \mathbf{f} \mathbf{b} \mathfrak{C}$ **by** (rule *assms(2)*)

have $\mathbf{g}_{SS} : \mathbf{g}_{SS} : \mathbf{a}_{SS} \mapsto \rightarrow \leftarrow_C \mathbf{o}_{SS}$ **and** $\mathbf{f}_{SS} : \mathbf{f}_{SS} : \mathbf{b}_{SS} \mapsto \rightarrow \leftarrow_C \mathbf{o}_{SS}$
by (*cs-concl cs-intro: cat-ss-cs-intros*)+

note *x.cat-cone-Comp-commute[cat-cs-simps del]*

from *x.ntcf-Comp-commute[OF g_{SS}] g_{SS} f_{SS}* **show**
 $x(\text{NTMap})(\mathbf{o}_{SS}) = \mathbf{g} \circ_{A\mathfrak{C}} x(\text{NTMap})(\mathbf{a}_{SS})$

by

(
cs-prems cs-shallow
cs-simp: cat-ss-cs-simps cat-cs-simps cs-intro: cat-cs-intros
)

moreover from *x.ntcf-Comp-commute[OF f_{SS}] g_{SS} f_{SS}* **show**

$x(\mathit{NTMap})(\mathit{o}_{SS}) = f \circ_{A\mathfrak{C}} x(\mathit{NTMap})(\mathit{b}_{SS})$
by
 (
 cs-prems **cs-shallow**
 cs-simp: *cat-ss-cs-simps cat-cs-simps* **cs-intro**: *cat-cs-intros*
)
ultimately show $g \circ_{A\mathfrak{C}} x(\mathit{NTMap})(\mathit{a}_{SS}) = f \circ_{A\mathfrak{C}} x(\mathit{NTMap})(\mathit{b}_{SS})$ **by** *simp*
qed

lemma (in *is-cat-pullback*) *cat-pb-cone-cospan*:
shows $x(\mathit{NTMap})(\mathit{o}_{SS}) = g \circ_{A\mathfrak{C}} x(\mathit{NTMap})(\mathit{a}_{SS})$
and $x(\mathit{NTMap})(\mathit{o}_{SS}) = f \circ_{A\mathfrak{C}} x(\mathit{NTMap})(\mathit{b}_{SS})$
and $g \circ_{A\mathfrak{C}} x(\mathit{NTMap})(\mathit{a}_{SS}) = f \circ_{A\mathfrak{C}} x(\mathit{NTMap})(\mathit{b}_{SS})$
by (*all* \langle rule *cat-cone-cospan* [*OF is-cat-cone-axioms cf-scospan-axioms*] \rangle)

lemma *cat-cocone-span*:
assumes $x : \langle \mathit{a} \leftarrow \mathit{g} \leftarrow \mathit{o} \rightarrow \mathit{f} \rightarrow \mathit{b} \rangle_{CF\mathfrak{C}} >_{CF.cocone} X : \langle \leftarrow \rightarrow_C \mapsto \rightarrow_C \alpha \mathfrak{C} \rangle$
and *cf-sspan* $\alpha \mathit{a} \mathit{g} \mathit{o} \mathit{f} \mathit{b} \mathfrak{C}$
shows $x(\mathit{NTMap})(\mathit{o}_{SS}) = x(\mathit{NTMap})(\mathit{a}_{SS}) \circ_{A\mathfrak{C}} g$
and $x(\mathit{NTMap})(\mathit{o}_{SS}) = x(\mathit{NTMap})(\mathit{b}_{SS}) \circ_{A\mathfrak{C}} f$
and $x(\mathit{NTMap})(\mathit{a}_{SS}) \circ_{A\mathfrak{C}} g = x(\mathit{NTMap})(\mathit{b}_{SS}) \circ_{A\mathfrak{C}} f$

proof-

interpret $x : \mathit{is-cat-cocone} \alpha X \langle \leftarrow \rightarrow_C \rangle \mathfrak{C} \langle \mathit{a} \leftarrow \mathit{g} \leftarrow \mathit{o} \rightarrow \mathit{f} \rightarrow \mathit{b} \rangle_{CF\mathfrak{C}} x$
by (*rule assms*(1))

interpret *span*: *cf-sspan* $\alpha \mathit{a} \mathit{g} \mathit{o} \mathit{f} \mathit{b} \mathfrak{C}$ **by** (*rule assms*(2))

note *op* =

cat-cone-cospan
 [
 OF
 x.is-cat-cone-op[*unfolded cat-op-simps*]
 span.cf-scospan-op,
 unfolded cat-op-simps
]

from *op*(1) **show** $x(\mathit{NTMap})(\mathit{o}_{SS}) = x(\mathit{NTMap})(\mathit{a}_{SS}) \circ_{A\mathfrak{C}} g$

by

(
 cs-prems
 cs-simp: *cat-ss-cs-simps cat-op-simps*
 cs-intro: *cat-cs-intros cat-ss-cs-intros*
)

moreover from *op*(2) **show** $x(\mathit{NTMap})(\mathit{o}_{SS}) = x(\mathit{NTMap})(\mathit{b}_{SS}) \circ_{A\mathfrak{C}} f$

by

(
 cs-prems
 cs-simp: *cat-ss-cs-simps cat-op-simps*
 cs-intro: *cat-cs-intros cat-ss-cs-intros*
)

ultimately show $x(\mathit{NTMap})(\mathit{a}_{SS}) \circ_{A\mathfrak{C}} g = x(\mathit{NTMap})(\mathit{b}_{SS}) \circ_{A\mathfrak{C}} f$ **by** *auto*

qed

lemma (in *is-cat-pushout*) *cat-po-cocone-span*:
shows $x(\mathit{NTMap})(\mathit{o}_{SS}) = x(\mathit{NTMap})(\mathit{a}_{SS}) \circ_{A\mathfrak{C}} g$
and $x(\mathit{NTMap})(\mathit{o}_{SS}) = x(\mathit{NTMap})(\mathit{b}_{SS}) \circ_{A\mathfrak{C}} f$
and $x(\mathit{NTMap})(\mathit{a}_{SS}) \circ_{A\mathfrak{C}} g = x(\mathit{NTMap})(\mathit{b}_{SS}) \circ_{A\mathfrak{C}} f$
by (*all* \langle rule *cat-cocone-span* [*OF is-cat-cocone-axioms cf-sspan-axioms*] \rangle)

6.1.2 Universal property

lemma *is-cat-pullbackI'*:

assumes $x : X <_{CF.cone} \langle a \rightarrow g \rightarrow o \leftarrow f \leftarrow b \rangle_{CF\mathfrak{C}} : \rightarrow \leftarrow C \mapsto \mapsto C\alpha \mathfrak{C}$
and $cf\text{-scospan } \alpha \ a \ g \ o \ f \ b \ \mathfrak{C}$
and $\wedge x' X'. x' : X' <_{CF.cone} \langle a \rightarrow g \rightarrow o \leftarrow f \leftarrow b \rangle_{CF\mathfrak{C}} : \rightarrow \leftarrow C \mapsto \mapsto C\alpha \mathfrak{C} \implies$
 $\exists ! f'.$
 $f' : X' \mapsto_{\mathfrak{C}} X \wedge$
 $x'(\ulcorner NTMap \urcorner)(\ulcorner a_{SS} \urcorner) = x(\ulcorner NTMap \urcorner)(\ulcorner a_{SS} \urcorner) \circ_{A\mathfrak{C}} f' \wedge$
 $x'(\ulcorner NTMap \urcorner)(\ulcorner b_{SS} \urcorner) = x(\ulcorner NTMap \urcorner)(\ulcorner b_{SS} \urcorner) \circ_{A\mathfrak{C}} f'$
shows $x : X <_{CF.pb} a \rightarrow g \rightarrow o \leftarrow f \leftarrow b \mapsto \mapsto C\alpha \mathfrak{C}$
proof(*intro is-cat-pullbackI is-cat-limitI*)

show $x : X <_{CF.cone} \langle a \rightarrow g \rightarrow o \leftarrow f \leftarrow b \rangle_{CF\mathfrak{C}} : \rightarrow \leftarrow C \mapsto \mapsto C\alpha \mathfrak{C}$
by (*rule assms(1)*)
interpret $x : is\text{-cat-cone } \alpha \ X \ \langle \rightarrow \leftarrow C \rangle \ \mathfrak{C} \ \langle \langle a \rightarrow g \rightarrow o \leftarrow f \leftarrow b \rangle_{CF\mathfrak{C}} \rangle \ x$
by (*rule assms(1)*)
show $cf\text{-scospan } \alpha \ a \ g \ o \ f \ b \ \mathfrak{C}$ **by** (*rule assms(2)*)
interpret $cospan : cf\text{-scospan } \alpha \ a \ g \ o \ f \ b \ \mathfrak{C}$ **by** (*rule assms(2)*)

fix $u' r'$ **assume** *prems*:

$u' : r' <_{CF.cone} \langle a \rightarrow g \rightarrow o \leftarrow f \leftarrow b \rangle_{CF\mathfrak{C}} : \rightarrow \leftarrow C \mapsto \mapsto C\alpha \mathfrak{C}$

interpret $u' : is\text{-cat-cone } \alpha \ r' \ \langle \rightarrow \leftarrow C \rangle \ \mathfrak{C} \ \langle \langle a \rightarrow g \rightarrow o \leftarrow f \leftarrow b \rangle_{CF\mathfrak{C}} \rangle \ u'$
by (*rule prems*)

from *assms(3)[OF prems]* **obtain** f'

where $f' : r' \mapsto_{\mathfrak{C}} X$
and $u'\text{-}a_{SS} : u'(\ulcorner NTMap \urcorner)(\ulcorner a_{SS} \urcorner) = x(\ulcorner NTMap \urcorner)(\ulcorner a_{SS} \urcorner) \circ_{A\mathfrak{C}} f'$
and $u'\text{-}b_{SS} : u'(\ulcorner NTMap \urcorner)(\ulcorner b_{SS} \urcorner) = x(\ulcorner NTMap \urcorner)(\ulcorner b_{SS} \urcorner) \circ_{A\mathfrak{C}} f'$
and *unique-f'*: $\wedge f''.$

\llbracket
 $f'' : r' \mapsto_{\mathfrak{C}} X;$
 $u'(\ulcorner NTMap \urcorner)(\ulcorner a_{SS} \urcorner) = x(\ulcorner NTMap \urcorner)(\ulcorner a_{SS} \urcorner) \circ_{A\mathfrak{C}} f'';$
 $u'(\ulcorner NTMap \urcorner)(\ulcorner b_{SS} \urcorner) = x(\ulcorner NTMap \urcorner)(\ulcorner b_{SS} \urcorner) \circ_{A\mathfrak{C}} f''$
 $\rrbracket \implies f'' = f'$

by *metis*

show $\exists ! f'. f' : r' \mapsto_{\mathfrak{C}} X \wedge u' = x \cdot_{NTCF} \text{ntcf-const } \rightarrow \leftarrow C \ \mathfrak{C} \ f'$

proof(*intro exI conjI; (elim conjE)?*)

show $u' = x \cdot_{NTCF} \text{ntcf-const } \rightarrow \leftarrow C \ \mathfrak{C} \ f'$

proof(*rule ntcf-eqI*)

show $u' : cf\text{-const } \rightarrow \leftarrow C \ \mathfrak{C} \ r' \mapsto_{CF} \langle a \rightarrow g \rightarrow o \leftarrow f \leftarrow b \rangle_{CF\mathfrak{C}} : \rightarrow \leftarrow C \mapsto \mapsto C\alpha \mathfrak{C}$
by (*rule u'.is-ntcf-axioms*)

from f' **show**

$x \cdot_{NTCF} \text{ntcf-const } \rightarrow \leftarrow C \ \mathfrak{C} \ f' :$
 $cf\text{-const } \rightarrow \leftarrow C \ \mathfrak{C} \ r' \mapsto_{CF} \langle a \rightarrow g \rightarrow o \leftarrow f \leftarrow b \rangle_{CF\mathfrak{C}} :$
 $\rightarrow \leftarrow C \mapsto \mapsto C\alpha \mathfrak{C}$

by (*cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

from f' **have** *dom-rhs*:

$\mathcal{D}_o ((x \cdot_{NTCF} \text{ntcf-const } \rightarrow \leftarrow C \ \mathfrak{C} \ f')(\ulcorner NTMap \urcorner)) = \rightarrow \leftarrow C(\ulcorner Obj \urcorner)$

by (*cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

show $u'(\ulcorner NTMap \urcorner) = (x \cdot_{NTCF} \text{ntcf-const } \rightarrow \leftarrow C \ \mathfrak{C} \ f')(\ulcorner NTMap \urcorner)$

proof(*rule vsv-eqI, unfold cat-cs-simps dom-rhs*)

fix a **assume** *prems'*: $a \in_o \rightarrow \leftarrow C(\ulcorner Obj \urcorner)$

from *this* f' *x.is-ntcf-axioms* **show**

$u'(\ulcorner NTMap \urcorner)(\ulcorner a \urcorner) = (x \cdot_{NTCF} \text{ntcf-const } \rightarrow \leftarrow C \ \mathfrak{C} \ f')(\ulcorner NTMap \urcorner)(\ulcorner a \urcorner)$

by (elim the-cat-scospans-ObjE; simp only)
 (
 cs-concl
 cs-simp:
 cat-cs-simps cat-ss-cs-simps
 u'-b_{SS} u'-a_{SS}
 cat-cone-cospan(1)[OF assms(1,2)]
 cat-cone-cospan(1)[OF prems assms(2)]
 cs-intro: cat-cs-intros cat-ss-cs-intros
)+
 qed (cs-concl **cs-intro**: cat-cs-intros | auto)+
 qed simp-all

fix f'' assume prems:
 f'' : r' ↦_ℳ X u' = x ·_{NTCF} ntcf-const →_{←C} ℳ f''
 have a_{SS}: a_{SS} ε_o →_{←C} (Obj) and b_{SS}: b_{SS} ε_o →_{←C} (Obj)
 by (cs-concl **cs-intro**: cat-ss-cs-intros)+
 have u'⟨NTMap⟩(a) = x⟨NTMap⟩(a) ∘_{Aℳ} f'' if a ε_o →_{←C} (Obj) for a
 proof-
 from prems(2) have
 u'⟨NTMap⟩(a) = (x ·_{NTCF} ntcf-const →_{←C} ℳ f'')⟨NTMap⟩(a)
 by simp
 from this that prems(1) show u'⟨NTMap⟩(a) = x⟨NTMap⟩(a) ∘_{Aℳ} f''
 by (cs-prems **cs-simp**: cat-cs-simps **cs-intro**: cat-cs-intros)
 qed
 from unique-f'[OF prems(1) this[OF a_{SS}] this[OF b_{SS}]] show f'' = f'.

qed (intro f')

qed

lemma is-cat-pushoutI':

assumes x : ⟨a←g←o→f→b⟩_{CFℳ} >_{CF.cocone} X : ↔_C ℳ ↦_{→Cα} ℳ
 and cf-sspan α a g o f b ℳ
 and ∧x' X'. x' : ⟨a←g←o→f→b⟩_{CFℳ} >_{CF.cocone} X' : ↔_C ℳ ↦_{→Cα} ℳ ⇒
 ∃!f'.
 f' : X ↦_ℳ X' ∧
 x'⟨NTMap⟩(a_{SS}) = f' ∘_{Aℳ} x⟨NTMap⟩(a_{SS}) ∧
 x'⟨NTMap⟩(b_{SS}) = f' ∘_{Aℳ} x⟨NTMap⟩(b_{SS})
 shows x : a←g←o→f→b >_{CF.po} X ↦_{→Cα} ℳ

proof-
 interpret x : is-cat-cocone α X ⟨↔_C⟩ ℳ ⟨⟨a←g←o→f→b⟩_{CFℳ}⟩ x
 by (rule assms(1))
 interpret span: cf-sspan α a g o f b ℳ by (rule assms(2))
 have assms-3':
 ∃!f'.
 f' : X ↦_ℳ X' ∧
 x'⟨NTMap⟩(a_{SS}) = x⟨NTMap⟩(a_{SS}) ∘_{Aop-cat} ℳ f' ∧
 x'⟨NTMap⟩(b_{SS}) = x⟨NTMap⟩(b_{SS}) ∘_{Aop-cat} ℳ f'
 if x' : X' <_{CF.cone} ⟨a→g→o←f←b⟩_{CFop-cat} ℳ : →_{←C} ℳ ↦_{→Cα} op-cat ℳ
 for x' X'

proof-
 from that(1) have [cat-op-simps]:
 f' : X ↦_ℳ X' ∧
 x'⟨NTMap⟩(a_{SS}) = x⟨NTMap⟩(a_{SS}) ∘_{Aop-cat} ℳ f' ∧
 x'⟨NTMap⟩(b_{SS}) = x⟨NTMap⟩(b_{SS}) ∘_{Aop-cat} ℳ f' ↔
 f' : X ↦_ℳ X' ∧
 x'⟨NTMap⟩(a_{SS}) = f' ∘_{Aℳ} x⟨NTMap⟩(a_{SS}) ∧

$x'(\text{NTMap})(\mathfrak{b}_{SS}) = f' \circ_{A\mathfrak{C}} x(\text{NTMap})(\mathfrak{b}_{SS})$
for f'
by (*intro iffI conjI; (elim conjE)?*)
(

 cs-concl
 cs-simp: *category.op-cat-Comp[symmetric] cat-op-simps cat-cs-simps*
 cs-intro: *cat-cs-intros cat-ss-cs-intros*
) +
interpret x' :
is-cat-cone $\alpha X' \langle \rightarrow \leftarrow C \rangle \langle \text{op-cat } \mathfrak{C} \rangle \langle \langle \mathfrak{a} \rightarrow \mathfrak{g} \rightarrow \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \rangle_{CF \text{ op-cat } \mathfrak{C}} \rangle x'$
by (*rule that*)
show *?thesis*
 unfolding *cat-op-simps*
 by
 (

 rule assms(\exists)[
 OF $x'.\text{is-cat-cocone-op}[\text{unfolded cat-op-simps}]$,
 unfolded cat-op-simps
]

)

qed
interpret *op-x: is-cat-pullback* $\alpha \mathfrak{a} \mathfrak{g} \mathfrak{o} \mathfrak{f} \mathfrak{b} \langle \text{op-cat } \mathfrak{C} \rangle X \langle \text{op-ntcf } x \rangle$
using
 is-cat-pullbackI'
 [
 OF $x.\text{is-cat-cone-op}[\text{unfolded cat-op-simps}]$
 span.cf-scospan-op,
 unfolded cat-op-simps,
 OF assms-3'
]

 by *simp*
show $x : \mathfrak{a} \leftarrow \mathfrak{g} \leftarrow \mathfrak{o} \rightarrow \mathfrak{f} \rightarrow \mathfrak{b} \rangle_{CF.p\circ} X \mapsto \mapsto_{C\alpha} \mathfrak{C}$
 by (*rule op-x.is-cat-pushout-op[unfolded cat-op-simps]*)
qed

lemma (*in is-cat-pullback*) *cat-pb-unique-cone*:
assumes $x' : X' \langle \rightarrow \leftarrow C \rangle \langle \mathfrak{a} \rightarrow \mathfrak{g} \rightarrow \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \rangle_{CF\mathfrak{C}} : \rightarrow \leftarrow C \mapsto \mapsto_{C\alpha} \mathfrak{C}$
shows $\exists! f'$.
 $f' : X' \mapsto_{\mathfrak{C}} X \wedge$
 $x'(\text{NTMap})(\mathfrak{a}_{SS}) = x(\text{NTMap})(\mathfrak{a}_{SS}) \circ_{A\mathfrak{C}} f' \wedge$
 $x'(\text{NTMap})(\mathfrak{b}_{SS}) = x(\text{NTMap})(\mathfrak{b}_{SS}) \circ_{A\mathfrak{C}} f'$

proof-
interpret x' : *is-cat-cone* $\alpha X' \langle \rightarrow \leftarrow C \rangle \mathfrak{C} \langle \langle \mathfrak{a} \rightarrow \mathfrak{g} \rightarrow \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \rangle_{CF\mathfrak{C}} \rangle x'$
by (*rule assms*)
from *cat-lim-ua-fo[OF assms]* **obtain** f'
where $f' : X' \mapsto_{\mathfrak{C}} X$
 and $x'\text{-def}$: $x' = x \cdot_{NTCF} \text{ntcf-const} \rightarrow \leftarrow C \mathfrak{C} f'$
 and *unique-f'*: $\wedge f''$.
 [[$f'' : X' \mapsto_{\mathfrak{C}} X; x' = x \cdot_{NTCF} \text{ntcf-const} \rightarrow \leftarrow C \mathfrak{C} f''$]] \implies
 $f'' = f'$
by *auto*
have $\mathfrak{a}_{SS} \in_{\circ} \rightarrow \leftarrow C(\text{Obj})$ **and** $\mathfrak{b}_{SS} \in_{\circ} \rightarrow \leftarrow C(\text{Obj})$
by (*cs-concl cs-intro: cat-ss-cs-intros*) +
show *?thesis*
proof(*intro ex1I conjI; (elim conjE)?*)
 show $f' : X' \mapsto_{\mathfrak{C}} X$ **by** (*rule f'*)
 have $x'(\text{NTMap})(\mathfrak{a}) = x(\text{NTMap})(\mathfrak{a}) \circ_{A\mathfrak{C}} f'$ **if** $\mathfrak{a} \in_{\circ} \rightarrow \leftarrow C(\text{Obj})$ **for** a
proof-

from x' -def have

$$x'(\downarrow NTMap)(\downarrow a) = (x \cdot_{NTCF} ntcf\text{-const} \rightarrow_{\leftarrow C} \mathfrak{C} f')(\downarrow NTMap)(\downarrow a)$$

by *simp*

from *this* that f' show $x'(\downarrow NTMap)(\downarrow a) = x(\downarrow NTMap)(\downarrow a) \circ_A \mathfrak{C} f'$

by (*cs-prems cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

qed

from *this*[*OF* \mathfrak{a}_{SS}] *this*[*OF* \mathfrak{b}_{SS}] show

$$x'(\downarrow NTMap)(\downarrow \mathfrak{a}_{SS}) = x(\downarrow NTMap)(\downarrow \mathfrak{a}_{SS}) \circ_A \mathfrak{C} f'$$

$$x'(\downarrow NTMap)(\downarrow \mathfrak{b}_{SS}) = x(\downarrow NTMap)(\downarrow \mathfrak{b}_{SS}) \circ_A \mathfrak{C} f'$$

by *auto*

fix f'' assume *prems'*:

$$f'' : X' \mapsto_{\mathfrak{C}} X$$

$$x'(\downarrow NTMap)(\downarrow \mathfrak{a}_{SS}) = x(\downarrow NTMap)(\downarrow \mathfrak{a}_{SS}) \circ_A \mathfrak{C} f''$$

$$x'(\downarrow NTMap)(\downarrow \mathfrak{b}_{SS}) = x(\downarrow NTMap)(\downarrow \mathfrak{b}_{SS}) \circ_A \mathfrak{C} f''$$

have $x' = x \cdot_{NTCF} ntcf\text{-const} \rightarrow_{\leftarrow C} \mathfrak{C} f''$

proof(*rule ntcf-eqI*)

show $x' : cf\text{-const} \rightarrow_{\leftarrow C} \mathfrak{C} X' \mapsto_{CF} \langle \mathfrak{a} \rightarrow \mathfrak{g} \rightarrow \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \rangle_{CF} \mathfrak{C} : \rightarrow_{\leftarrow C} \mapsto_{C\alpha} \mathfrak{C}$

by (*rule x'.is-ntcf-axioms*)

from *prems'*(1) show

$$x \cdot_{NTCF} ntcf\text{-const} \rightarrow_{\leftarrow C} \mathfrak{C} f'' :$$

$$cf\text{-const} \rightarrow_{\leftarrow C} \mathfrak{C} X' \mapsto_{CF} \langle \mathfrak{a} \rightarrow \mathfrak{g} \rightarrow \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \rangle_{CF} \mathfrak{C} :$$

$$\rightarrow_{\leftarrow C} \mapsto_{C\alpha} \mathfrak{C}$$

by (*cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

have *dom-lhs*: $\mathcal{D}_o(x'(\downarrow NTMap)) = \rightarrow_{\leftarrow C}(\downarrow Obj)$

by (*cs-concl cs-shallow cs-simp: cat-cs-simps*)

from *prems'*(1) have *dom-rhs*:

$$\mathcal{D}_o((x \cdot_{NTCF} ntcf\text{-const} \rightarrow_{\leftarrow C} \mathfrak{C} f'')(\downarrow NTMap)) = \rightarrow_{\leftarrow C}(\downarrow Obj)$$

by (*cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

show $x'(\downarrow NTMap) = (x \cdot_{NTCF} ntcf\text{-const} \rightarrow_{\leftarrow C} \mathfrak{C} f'')(\downarrow NTMap)$

proof(*rule vsv-eqI, unfold dom-lhs dom-rhs*)

fix a assume *prems''*: $a \in_o \rightarrow_{\leftarrow C}(\downarrow Obj)$

from *this* *prems'*(1) show

$$x'(\downarrow NTMap)(\downarrow a) = (x \cdot_{NTCF} ntcf\text{-const} \rightarrow_{\leftarrow C} \mathfrak{C} f'')(\downarrow NTMap)(\downarrow a)$$

by (*elim the-cat-scospans-ObjE; simp only*):

(

cs-concl

cs-simp:

prems'(2,3)

cat-cone-cospan(1,2)[*OF* *assms cf-scospans-axioms*]

cat-pb-cone-cospan

cat-ss-cs-simps cat-cs-simps

cs-intro: *cat-ss-cs-intros cat-cs-intros*

)+

qed (*auto simp: cat-cs-intros*)

qed *simp-all*

from *unique-f'*[*OF* *prems'*(1) *this*] show $f'' = f'$.

qed

qed

lemma (in *is-cat-pullback*) *cat-pb-unique*:

assumes $x' : X' <_{CF.pb} \mathfrak{a} \rightarrow \mathfrak{g} \rightarrow \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \mapsto_{C\alpha} \mathfrak{C}$

shows $\exists! f' : X' \mapsto_{\mathfrak{C}} X \wedge x' = x \cdot_{NTCF} ntcf\text{-const} \rightarrow_{\leftarrow C} \mathfrak{C} f'$

by (*rule cat-lim-unique*[*OF* *is-cat-pullbackD*(1)[*OF* *assms*]])

lemma (in *is-cat-pullback*) *cat-pb-unique'*:

assumes $x' : X' <_{CF.pb} \mathfrak{a} \rightarrow \mathfrak{g} \rightarrow \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \mapsto_{C\alpha} \mathfrak{C}$

shows $\exists! f'$.

$f' : X' \mapsto_{\mathfrak{C}} X \wedge$

$$\begin{aligned} x'(\mathit{NTMap})(\mathbf{a}_{SS}) &= x(\mathit{NTMap})(\mathbf{a}_{SS}) \circ_{A\mathfrak{C}} f' \wedge \\ x'(\mathit{NTMap})(\mathbf{b}_{SS}) &= x(\mathit{NTMap})(\mathbf{b}_{SS}) \circ_{A\mathfrak{C}} f' \end{aligned}$$

proof-

interpret x' : *is-cat-pullback* α $\mathbf{a} \leftarrow \mathbf{g} \leftarrow \mathbf{o} \rightarrow \mathbf{f} \rightarrow \mathbf{b}$ \mathfrak{C} X' x' **by** (*rule assms(1)*)

show *?thesis* **by** (*rule cat-pb-unique-cone[OF x'.is-cat-cone-axioms]*)

qed

lemma (*in is-cat-pushout*) *cat-po-unique-cocone*:

assumes $x' : \langle \mathbf{a} \leftarrow \mathbf{g} \leftarrow \mathbf{o} \rightarrow \mathbf{f} \rightarrow \mathbf{b} \rangle_{CF\mathfrak{C}} >_{CF.cocone} X' : \leftarrow \rightarrow_C \mapsto \rightarrow_{C\alpha} \mathfrak{C}$

shows $\exists ! f'$.

$$f' : X \mapsto_{\mathfrak{C}} X' \wedge$$

$$x'(\mathit{NTMap})(\mathbf{a}_{SS}) = f' \circ_{A\mathfrak{C}} x(\mathit{NTMap})(\mathbf{a}_{SS}) \wedge$$

$$x'(\mathit{NTMap})(\mathbf{b}_{SS}) = f' \circ_{A\mathfrak{C}} x(\mathit{NTMap})(\mathbf{b}_{SS})$$

proof-

interpret x' : *is-cat-cocone* α $X' \langle \leftarrow \rightarrow_C \rangle \mathfrak{C} \langle \mathbf{a} \leftarrow \mathbf{g} \leftarrow \mathbf{o} \rightarrow \mathbf{f} \rightarrow \mathbf{b} \rangle_{CF\mathfrak{C}}$ x'

by (*rule assms(1)*)

have [*cat-op-simps*]:

$$f' : X \mapsto_{\mathfrak{C}} X' \wedge$$

$$x'(\mathit{NTMap})(\mathbf{a}_{SS}) = x(\mathit{NTMap})(\mathbf{a}_{SS}) \circ_{Aop-cat} \mathfrak{C} f' \wedge$$

$$x'(\mathit{NTMap})(\mathbf{b}_{SS}) = x(\mathit{NTMap})(\mathbf{b}_{SS}) \circ_{Aop-cat} \mathfrak{C} f' \leftarrow$$

$$f' : X \mapsto_{\mathfrak{C}} X' \wedge$$

$$x'(\mathit{NTMap})(\mathbf{a}_{SS}) = f' \circ_{A\mathfrak{C}} x(\mathit{NTMap})(\mathbf{a}_{SS}) \wedge$$

$$x'(\mathit{NTMap})(\mathbf{b}_{SS}) = f' \circ_{A\mathfrak{C}} x(\mathit{NTMap})(\mathbf{b}_{SS})$$

for f'

by (*intro iffI conjI; (elim conjE)?*)

(

cs-concl

cs-simp: *category.op-cat-Comp[symmetric] cat-op-simps cat-cs-simps*

cs-intro: *cat-cs-intros cat-ss-cs-intros*

)+

show *?thesis*

by

(

rule is-cat-pullback.cat-pb-unique-cone[

OF is-cat-pullback-op x'.is-cat-cone-op[unfolded cat-op-simps],

unfolded cat-op-simps

]

)

qed

lemma (*in is-cat-pushout*) *cat-po-unique*:

assumes $x' : \mathbf{a} \leftarrow \mathbf{g} \leftarrow \mathbf{o} \rightarrow \mathbf{f} \rightarrow \mathbf{b} >_{CF.po} X' \mapsto \rightarrow_{C\alpha} \mathfrak{C}$

shows $\exists ! f'. f' : X \mapsto_{\mathfrak{C}} X' \wedge x' = \mathit{ntcf-const} \leftarrow \rightarrow_C \mathfrak{C} f' \cdot_{NTCF} x$

by (*rule cat-colim-unique[OF is-cat-pushoutD(1)[OF assms]]*)

lemma (*in is-cat-pushout*) *cat-po-unique'*:

assumes $x' : \mathbf{a} \leftarrow \mathbf{g} \leftarrow \mathbf{o} \rightarrow \mathbf{f} \rightarrow \mathbf{b} >_{CF.po} X' \mapsto \rightarrow_{C\alpha} \mathfrak{C}$

shows $\exists ! f'$.

$$f' : X \mapsto_{\mathfrak{C}} X' \wedge$$

$$x'(\mathit{NTMap})(\mathbf{a}_{SS}) = f' \circ_{A\mathfrak{C}} x(\mathit{NTMap})(\mathbf{a}_{SS}) \wedge$$

$$x'(\mathit{NTMap})(\mathbf{b}_{SS}) = f' \circ_{A\mathfrak{C}} x(\mathit{NTMap})(\mathbf{b}_{SS})$$

proof-

interpret x' : *is-cat-pushout* α $\mathbf{a} \leftarrow \mathbf{g} \leftarrow \mathbf{o} \rightarrow \mathbf{f} \rightarrow \mathbf{b}$ \mathfrak{C} X' x' **by** (*rule assms(1)*)

show *?thesis* **by** (*rule cat-po-unique-cocone[OF x'.is-cat-cocone-axioms]*)

qed

lemma *cat-pullback-ex-is-iso-arr*:

assumes $x : X <_{CF.pb} \mathbf{a} \rightarrow \mathbf{g} \rightarrow \mathbf{o} \leftarrow \mathbf{f} \leftarrow \mathbf{b} \mapsto \rightarrow_{C\alpha} \mathfrak{C}$

and $x' : X' <_{CF.pb} \mathbf{a} \rightarrow \mathbf{g} \rightarrow \mathbf{o} \leftarrow \mathbf{f} \leftarrow \mathbf{b} \mapsto \mapsto_C \alpha \mathfrak{C}$
 obtains f where $f : X' \mapsto_{iso} \mathfrak{C} X$
 and $x' = x \cdot_{NTCF} ntcf-const \rightarrow \leftarrow_C \mathfrak{C} f$
proof-
 interpret x : *is-cat-pullback* $\alpha \mathbf{a} \mathbf{g} \mathbf{o} \mathbf{f} \mathbf{b} \mathfrak{C} X x$ by (rule *assms(1)*)
 interpret x' : *is-cat-pullback* $\alpha \mathbf{a} \mathbf{g} \mathbf{o} \mathbf{f} \mathbf{b} \mathfrak{C} X' x'$ by (rule *assms(2)*)
 from that show *?thesis*
 by
 (

$$\begin{array}{l} elim\ cat\text{-}lim\text{-}ex\text{-}is\text{-}iso\text{-}arr[\\ OF\ x.is\text{-}cat\text{-}limit\text{-}axioms\ x'.is\text{-}cat\text{-}limit\text{-}axioms \\] \end{array}$$
)
qed

lemma *cat-pullback-ex-is-iso-arr'*:
 assumes $x : X <_{CF.pb} \mathbf{a} \rightarrow \mathbf{g} \rightarrow \mathbf{o} \leftarrow \mathbf{f} \leftarrow \mathbf{b} \mapsto \mapsto_C \alpha \mathfrak{C}$
 and $x' : X' <_{CF.pb} \mathbf{a} \rightarrow \mathbf{g} \rightarrow \mathbf{o} \leftarrow \mathbf{f} \leftarrow \mathbf{b} \mapsto \mapsto_C \alpha \mathfrak{C}$
 obtains f where $f : X' \mapsto_{iso} \mathfrak{C} X$
 and $x'(\mathit{NTMap})(\mathbf{a}_{SS}) = x(\mathit{NTMap})(\mathbf{a}_{SS}) \circ_{A\mathfrak{C}} f$
 and $x'(\mathit{NTMap})(\mathbf{b}_{SS}) = x(\mathit{NTMap})(\mathbf{b}_{SS}) \circ_{A\mathfrak{C}} f$
proof-
 interpret x : *is-cat-pullback* $\alpha \mathbf{a} \mathbf{g} \mathbf{o} \mathbf{f} \mathbf{b} \mathfrak{C} X x$ by (rule *assms(1)*)
 interpret x' : *is-cat-pullback* $\alpha \mathbf{a} \mathbf{g} \mathbf{o} \mathbf{f} \mathbf{b} \mathfrak{C} X' x'$ by (rule *assms(2)*)
 obtain f where $f : X' \mapsto_{iso} \mathfrak{C} X$
 and $j \in \epsilon_o \rightarrow \leftarrow_C (\mathit{Obj}) \implies x'(\mathit{NTMap})(j) = x(\mathit{NTMap})(j) \circ_{A\mathfrak{C}} f$ for j
 by
 (

$$\begin{array}{l} elim\ cat\text{-}lim\text{-}ex\text{-}is\text{-}iso\text{-}arr'[\\ OF\ x.is\text{-}cat\text{-}limit\text{-}axioms\ x'.is\text{-}cat\text{-}limit\text{-}axioms \\] \end{array}$$
)
 then have
 $x'(\mathit{NTMap})(\mathbf{a}_{SS}) = x(\mathit{NTMap})(\mathbf{a}_{SS}) \circ_{A\mathfrak{C}} f$
 $x'(\mathit{NTMap})(\mathbf{b}_{SS}) = x(\mathit{NTMap})(\mathbf{b}_{SS}) \circ_{A\mathfrak{C}} f$
 by (auto simp: *cat-ss-cs-intros*)
 with f show *?thesis* using that by *simp*
qed

lemma *cat-pushout-ex-is-iso-arr*:
 assumes $x : \mathbf{a} \leftarrow \mathbf{g} \leftarrow \mathbf{o} \rightarrow \mathbf{f} \rightarrow \mathbf{b} >_{CF.po} X \mapsto \mapsto_C \alpha \mathfrak{C}$
 and $x' : \mathbf{a} \leftarrow \mathbf{g} \leftarrow \mathbf{o} \rightarrow \mathbf{f} \rightarrow \mathbf{b} >_{CF.po} X' \mapsto \mapsto_C \alpha \mathfrak{C}$
 obtains f where $f : X \mapsto_{iso} \mathfrak{C} X'$
 and $x' = ntcf-const \leftarrow \rightarrow_C \mathfrak{C} f \cdot_{NTCF} x$
proof-
 interpret x : *is-cat-pushout* $\alpha \mathbf{a} \mathbf{g} \mathbf{o} \mathbf{f} \mathbf{b} \mathfrak{C} X x$ by (rule *assms(1)*)
 interpret x' : *is-cat-pushout* $\alpha \mathbf{a} \mathbf{g} \mathbf{o} \mathbf{f} \mathbf{b} \mathfrak{C} X' x'$ by (rule *assms(2)*)
 from that show *?thesis*
 by
 (

$$\begin{array}{l} elim\ cat\text{-}colim\text{-}ex\text{-}is\text{-}iso\text{-}arr[\\ OF\ x.is\text{-}cat\text{-}colimit\text{-}axioms\ x'.is\text{-}cat\text{-}colimit\text{-}axioms \\] \end{array}$$
)
qed

lemma *cat-pushout-ex-is-iso-arr'*:
 assumes $x : \mathbf{a} \leftarrow \mathbf{g} \leftarrow \mathbf{o} \rightarrow \mathbf{f} \rightarrow \mathbf{b} >_{CF.po} X \mapsto \mapsto_C \alpha \mathfrak{C}$

and $x' : \mathbf{a} \leftarrow \mathbf{g} \leftarrow \mathbf{o} \rightarrow \mathbf{f} \rightarrow \mathbf{b} >_{CF.po} X' \mapsto \mapsto_C \alpha \mathfrak{C}$
obtains f **where** $f : X \mapsto_{iso} \mathfrak{C} X'$
and $x'(\mathit{NTMap})(\mathbf{a}_{SS}) = f \circ_{A\mathfrak{C}} x(\mathit{NTMap})(\mathbf{a}_{SS})$
and $x'(\mathit{NTMap})(\mathbf{b}_{SS}) = f \circ_{A\mathfrak{C}} x(\mathit{NTMap})(\mathbf{b}_{SS})$
proof-
interpret x : *is-cat-pushout* $\alpha \mathbf{a} \mathbf{g} \mathbf{o} \mathbf{f} \mathbf{b} \mathfrak{C} X x$ **by** (*rule assms(1)*)
interpret x' : *is-cat-pushout* $\alpha \mathbf{a} \mathbf{g} \mathbf{o} \mathbf{f} \mathbf{b} \mathfrak{C} X' x'$ **by** (*rule assms(2)*)
obtain f **where** $f: f : X \mapsto_{iso} \mathfrak{C} X'$
and $j \in_{\circ} \leftarrow \mapsto_C (\mathit{Obj}) \implies x'(\mathit{NTMap})(j) = f \circ_{A\mathfrak{C}} x(\mathit{NTMap})(j)$ **for** j
by
(
elim cat-colim-ex-is-iso-arr [
OF x.is-cat-colimit-axioms x'.is-cat-colimit-axioms
]
)
then have $x'(\mathit{NTMap})(\mathbf{a}_{SS}) = f \circ_{A\mathfrak{C}} x(\mathit{NTMap})(\mathbf{a}_{SS})$
and $x'(\mathit{NTMap})(\mathbf{b}_{SS}) = f \circ_{A\mathfrak{C}} x(\mathit{NTMap})(\mathbf{b}_{SS})$
by (*auto simp: cat-ss-cs-intros*)
with f **show** *?thesis* **using** *that* **by** *simp*
qed

7 Equalizers and coequalizers as limits and colimits

7.1 Equalizer and coequalizer

7.1.1 Definition and elementary properties

See [2]⁶.

locale *is-cat-equalizer* =
is-cat-limit $\alpha \langle \uparrow_C (\mathbf{a}_{PL} F) (\mathbf{b}_{PL} F) F \rangle \mathfrak{C} \langle \uparrow \rightarrow \uparrow_{CF} \mathfrak{C} (\mathbf{a}_{PL} F) (\mathbf{b}_{PL} F) F \mathbf{a} \mathbf{b} F' \rangle E \varepsilon +$
 $F': vsv F'$
for $\alpha \mathbf{a} \mathbf{b} F F' \mathfrak{C} E \varepsilon +$
assumes *cat-eq-F-in-Vset*[*cat-lim-cs-intros*]: $F \in_o Vset \alpha$
and *cat-eq-F-ne*[*cat-lim-cs-intros*]: $F \neq 0$
and *cat-eq-F'-vdomain*[*cat-lim-cs-simps*]: $\mathcal{D}_o F' = F$
and *cat-eq-F'-app-is-arr*[*cat-lim-cs-intros*]: $\mathfrak{f} \in_o F \Longrightarrow F'(\mathfrak{f}) : \mathbf{a} \mapsto_{\mathfrak{C}} \mathbf{b}$

syntax *-is-cat-equalizer* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
 $(\langle (- : / - <_{CF.eq} '(-,-,-)' : / \uparrow_C \mapsto_{C1} -) \rangle [51, 51, 51, 51, 51, 51] 51)$

syntax-consts *-is-cat-equalizer* \cong *is-cat-equalizer*

translations $\varepsilon : E <_{CF.eq} (\mathbf{a}, \mathbf{b}, F, F') : \uparrow_C \mapsto_{C\alpha} \mathfrak{C} \cong$
CONST is-cat-equalizer $\alpha \mathbf{a} \mathbf{b} F F' \mathfrak{C} E \varepsilon$

locale *is-cat-coequalizer* =
is-cat-colimit $\alpha \langle \uparrow_C (\mathbf{b}_{PL} F) (\mathbf{a}_{PL} F) F \rangle \mathfrak{C} \langle \uparrow \rightarrow \uparrow_{CF} \mathfrak{C} (\mathbf{b}_{PL} F) (\mathbf{a}_{PL} F) F \mathbf{b} \mathbf{a} F' \rangle E \varepsilon +$
 $F': vsv F'$
for $\alpha \mathbf{a} \mathbf{b} F F' \mathfrak{C} E \varepsilon +$
assumes *cat-coeq-F-in-Vset*[*cat-lim-cs-intros*]: $F \in_o Vset \alpha$
and *cat-coeq-F-ne*[*cat-lim-cs-intros*]: $F \neq 0$
and *cat-coeq-F'-vdomain*[*cat-lim-cs-simps*]: $\mathcal{D}_o F' = F$
and *cat-coeq-F'-app-is-arr*[*cat-lim-cs-intros*]: $\mathfrak{f} \in_o F \Longrightarrow F'(\mathfrak{f}) : \mathbf{b} \mapsto_{\mathfrak{C}} \mathbf{a}$

syntax *-is-cat-coequalizer* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
 $(\langle (- : / '(-,-,-)' >_{CF.coeq} - : / \uparrow_C \mapsto_{C1} -) \rangle [51, 51, 51, 51, 51, 51] 51)$

syntax-consts *-is-cat-coequalizer* \cong *is-cat-coequalizer*

translations $\varepsilon : (\mathbf{a}, \mathbf{b}, F, F') >_{CF.coeq} E : \uparrow_C \mapsto_{C\alpha} \mathfrak{C} \cong$
CONST is-cat-coequalizer $\alpha \mathbf{a} \mathbf{b} F F' \mathfrak{C} E \varepsilon$

Rules.

lemma (in *is-cat-equalizer*) *is-cat-equalizer-axioms'*[*cat-lim-cs-intros*]:
assumes $\alpha' = \alpha$
and $E' = E$
and $\mathbf{a}' = \mathbf{a}$
and $\mathbf{b}' = \mathbf{b}$
and $F'' = F$
and $F''' = F'$
and $\mathfrak{C}' = \mathfrak{C}$
shows $\varepsilon : E' <_{CF.eq} (\mathbf{a}', \mathbf{b}', F'', F''') : \uparrow_C \mapsto_{C\alpha'} \mathfrak{C}'$
unfolding *assms* by (rule *is-cat-equalizer-axioms*)

mk-ide rf *is-cat-equalizer-def*[*unfolded is-cat-equalizer-axioms-def*]
 $[intro\ is-cat-equalizerI]$
 $[dest\ is-cat-equalizerD[dest]]$
 $[elim\ is-cat-equalizerE[elim]]$

lemmas [*cat-lim-cs-intros*] = *is-cat-equalizerD(1)*

⁶[https://en.wikipedia.org/wiki/Equaliser_\(mathematics\)](https://en.wikipedia.org/wiki/Equaliser_(mathematics))

lemma (in *is-cat-coequalizer*) *is-cat-coequalizer-axioms'*[*cat-lim-cs-intros*]:

assumes $\alpha' = \alpha$

and $E' = E$

and $\mathfrak{a}' = \mathfrak{a}$

and $\mathfrak{b}' = \mathfrak{b}$

and $F'' = F$

and $F''' = F'$

and $\mathfrak{C}' = \mathfrak{C}$

shows $\varepsilon : (\mathfrak{a}', \mathfrak{b}', F'', F''') >_{CF.coeq} E' : \uparrow_C \mapsto \mapsto_{C\alpha'} \mathfrak{C}'$

unfolding *assms* by (rule *is-cat-coequalizer-axioms*)

mk-ide rf *is-cat-coequalizer-def*[*unfolded is-cat-coequalizer-axioms-def*]

|*intro is-cat-coequalizerI*|

|*dest is-cat-coequalizerD*[*dest*]|

|*elim is-cat-coequalizerE*[*elim*]|

lemmas [*cat-lim-cs-intros*] = *is-cat-coequalizerD*(1)

Elementary properties.

lemma (in *is-cat-equalizer*)

cat-eq-a[*cat-lim-cs-intros*]: $\mathfrak{a} \in_{\circ} \mathfrak{C}(\text{Obj})$

and *cat-eq-b*[*cat-lim-cs-intros*]: $\mathfrak{b} \in_{\circ} \mathfrak{C}(\text{Obj})$

proof–

from *cat-eq-F-ne* **obtain** \mathfrak{f} **where** $\mathfrak{f} : \mathfrak{f} \in_{\circ} F$ **by** *force*

have $F'(\mathfrak{f}) : \mathfrak{a} \mapsto_{\mathfrak{C}} \mathfrak{b}$ **by** (rule *cat-eq-F'-app-is-arr*[*OF f*])

then show $\mathfrak{a} \in_{\circ} \mathfrak{C}(\text{Obj})$ $\mathfrak{b} \in_{\circ} \mathfrak{C}(\text{Obj})$ **by** *auto*

qed

lemma (in *is-cat-coequalizer*)

cat-coeq-a[*cat-lim-cs-intros*]: $\mathfrak{a} \in_{\circ} \mathfrak{C}(\text{Obj})$

and *cat-coeq-b*[*cat-lim-cs-intros*]: $\mathfrak{b} \in_{\circ} \mathfrak{C}(\text{Obj})$

proof–

from *cat-coeq-F-ne* **obtain** \mathfrak{f} **where** $\mathfrak{f} : \mathfrak{f} \in_{\circ} F$ **by** *force*

have $F'(\mathfrak{f}) : \mathfrak{b} \mapsto_{\mathfrak{C}} \mathfrak{a}$ **by** (rule *cat-coeq-F'-app-is-arr*[*OF f*])

then show $\mathfrak{a} \in_{\circ} \mathfrak{C}(\text{Obj})$ $\mathfrak{b} \in_{\circ} \mathfrak{C}(\text{Obj})$ **by** *auto*

qed

sublocale *is-cat-equalizer* \subseteq *cf-parallel* $\alpha \langle \mathfrak{a}_{PL} F \rangle \langle \mathfrak{b}_{PL} F \rangle F \mathfrak{a} \mathfrak{b} F' \mathfrak{C}$

by (*intro cf-parallelI cat-parallelI*)

(

auto simp:

cat-lim-cs-simps cat-parallel-cs-intros cat-lim-cs-intros cat-cs-intros

)

sublocale *is-cat-coequalizer* \subseteq *cf-parallel* $\alpha \langle \mathfrak{b}_{PL} F \rangle \langle \mathfrak{a}_{PL} F \rangle F \mathfrak{b} \mathfrak{a} F' \mathfrak{C}$

by (*intro cf-parallelI cat-parallelI*)

(

auto simp:

cat-lim-cs-simps cat-parallel-cs-intros cat-lim-cs-intros cat-cs-intros

)

Duality.

lemma (in *is-cat-equalizer*) *is-cat-coequalizer-op*:

op-ntcf $\varepsilon : (\mathfrak{a}, \mathfrak{b}, F, F') >_{CF.coeq} E : \uparrow_C \mapsto \mapsto_{C\alpha} \text{op-cat } \mathfrak{C}$

by (*intro is-cat-coequalizerI*)

(

cs-concl

cs-simp: *cat-lim-cs-simps cat-op-simps*

cs-intro: $V\text{-cs-intros cat-op-intros cat-lim-cs-intros}$
 $)_+$

lemma (in is-cat-equalizer) is-cat-coequalizer-op' $[cat-op-intros]$:
assumes $\mathfrak{C}' = op\text{-cat } \mathfrak{C}$
shows $op\text{-ntcf } \varepsilon : (\mathbf{a}, \mathbf{b}, F, F') >_{CF.coeq} E : \uparrow_C \mapsto \mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (rule is-cat-coequalizer-op)

lemmas $[cat-op-intros] = is\text{-cat-equalizer.is-cat-coequalizer-op}'$

lemma (in is-cat-coequalizer) is-cat-equalizer-op:
 $op\text{-ntcf } \varepsilon : E <_{CF.eq} (\mathbf{a}, \mathbf{b}, F, F') : \uparrow_C \mapsto \mapsto_{C\alpha} op\text{-cat } \mathfrak{C}$
by (intro is-cat-equalizerI)
 (
 cs-concl
 cs-simp: $cat\text{-lim-cs-simps cat-op-simps}$
 cs-intro: $V\text{-cs-intros cat-op-intros cat-lim-cs-intros}$
)
 $)_+$

lemma (in is-cat-coequalizer) is-cat-equalizer-op' $[cat-op-intros]$:
assumes $\mathfrak{C}' = op\text{-cat } \mathfrak{C}$
shows $op\text{-ntcf } \varepsilon : E <_{CF.eq} (\mathbf{a}, \mathbf{b}, F, F') : \uparrow_C \mapsto \mapsto_{C\alpha} \mathfrak{C}'$
unfolding *assms* **by** (rule is-cat-equalizer-op)

lemmas $[cat-op-intros] = is\text{-cat-coequalizer.is-cat-equalizer-op}'$

Further properties.

lemma (in category) cat-cf-parallel-ab:
assumes $vsu F'$
and $F \in_o Vset \alpha$
and $\mathcal{D}_o F' = F$
and $\wedge f. f \in_o F \implies F'(\!|f) : \mathbf{a} \mapsto_{\mathfrak{C}} \mathbf{b}$
and $\mathbf{a} \in_o \mathfrak{C}(\!|Obj)$
and $\mathbf{b} \in_o \mathfrak{C}(\!|Obj)$
shows $cf\text{-parallel } \alpha (\mathbf{a}_{PL} F) (\mathbf{b}_{PL} F) F \mathbf{a} \mathbf{b} F' \mathfrak{C}$
proof-
have $\mathbf{a}_{PL} F \in_o Vset \alpha \mathbf{b}_{PL} F \in_o Vset \alpha$
by (*simp-all add: Axiom-of-Pairing b_{PL}-def a_{PL}-def assms(2)*)
then show *?thesis*
by (intro cf-parallelI cat-parallelI)
 (*simp-all add: assms cat-parallel-cs-intros cat-cs-intros*)
qed

lemma (in category) cat-cf-parallel-ba:
assumes $vsu F'$
and $F \in_o Vset \alpha$
and $\mathcal{D}_o F' = F$
and $\wedge f. f \in_o F \implies F'(\!|f) : \mathbf{b} \mapsto_{\mathfrak{C}} \mathbf{a}$
and $\mathbf{a} \in_o \mathfrak{C}(\!|Obj)$
and $\mathbf{b} \in_o \mathfrak{C}(\!|Obj)$
shows $cf\text{-parallel } \alpha (\mathbf{b}_{PL} F) (\mathbf{a}_{PL} F) F \mathbf{b} \mathbf{a} F' \mathfrak{C}$
proof-
have $\mathbf{a}_{PL} F \in_o Vset \alpha \mathbf{b}_{PL} F \in_o Vset \alpha$
by (*simp-all add: Axiom-of-Pairing b_{PL}-def a_{PL}-def assms(2)*)
then show *?thesis*
by (intro cf-parallelI cat-parallelI)
 (*simp-all add: assms cat-parallel-cs-intros cat-cs-intros*)
qed

lemma *cat-cone-cf-par-eps-NTMap-app*:

assumes ε :

$E <_{CF.cone} \uparrow \rightarrow \uparrow_{CF} \mathfrak{C} (\mathfrak{a}_{PL} F) (\mathfrak{b}_{PL} F) F \mathfrak{a} \mathfrak{b} F'$:

$\uparrow_C (\mathfrak{a}_{PL} F) (\mathfrak{b}_{PL} F) F \mapsto \rightarrow_{C\alpha} \mathfrak{C}$

and $vsv F'$

and $F \in_0 Vset \alpha$

and $\mathcal{D}_0 F' = F$

and $\wedge f. f \in_0 F \implies F'(\downarrow f) : \mathfrak{a} \mapsto_{\mathfrak{C}} \mathfrak{b}$

and $f \in_0 F$

shows $\varepsilon(\downarrow NTMap)(\downarrow \mathfrak{b}_{PL} F) = F'(\downarrow f) \circ_{A\mathfrak{C}} \varepsilon(\downarrow NTMap)(\downarrow \mathfrak{a}_{PL} F)$

proof-

let $?II = \langle \uparrow_C (\mathfrak{a}_{PL} F) (\mathfrak{b}_{PL} F) F \rangle$

and $?II-II = \langle \uparrow \rightarrow \uparrow_{CF} \mathfrak{C} (\mathfrak{a}_{PL} F) (\mathfrak{b}_{PL} F) F \mathfrak{a} \mathfrak{b} F' \rangle$

interpret ε : *is-cat-cone* $\alpha E ?II \mathfrak{C} ?II-II \varepsilon$ **by** (*rule assms(1)*)

from *assms(5,6)* **have** $\mathfrak{a} : \mathfrak{a} \in_0 \mathfrak{C}(\downarrow Obj)$ **and** $\mathfrak{b} : \mathfrak{b} \in_0 \mathfrak{C}(\downarrow Obj)$ **by** *auto*

interpret *par*: *cf-parallel* $\alpha \langle \mathfrak{a}_{PL} F \rangle \langle \mathfrak{b}_{PL} F \rangle F \mathfrak{a} \mathfrak{b} F' \mathfrak{C}$

by (*intro* $\varepsilon.NTDom.HomCod.cat-cf-parallel-ab$ *assms* $\mathfrak{a} \mathfrak{b}$)

from *assms(6)* **have** $\downarrow f : \mathfrak{a}_{PL} F \mapsto \uparrow_C (\mathfrak{a}_{PL} F) (\mathfrak{b}_{PL} F) F \mathfrak{b}_{PL} F$

by (*simp-all add: par.the-cat-parallel-is-arr-abF*)

note $\varepsilon.cat-cone-Comp-commute[cat-cs-simps del]$

from $\varepsilon.ntcf-Comp-commute[OF \downarrow f]$ *assms(6)* **show** *?thesis*

by

(

cs-prems

cs-simp: *cat-parallel-cs-simps cat-cs-simps*

cs-intro: *cat-cs-intros cat-parallel-cs-intros*

)

qed

lemma *cat-cocone-cf-par-eps-NTMap-app*:

assumes ε :

$\uparrow \rightarrow \uparrow_{CF} \mathfrak{C} (\mathfrak{b}_{PL} F) (\mathfrak{a}_{PL} F) F \mathfrak{b} \mathfrak{a} F' >_{CF.cocone} E$:

$\uparrow_C (\mathfrak{b}_{PL} F) (\mathfrak{a}_{PL} F) F \mapsto \rightarrow_{C\alpha} \mathfrak{C}$

and $vsv F'$

and $F \in_0 Vset \alpha$

and $\mathcal{D}_0 F' = F$

and $\wedge f. f \in_0 F \implies F'(\downarrow f) : \mathfrak{b} \mapsto_{\mathfrak{C}} \mathfrak{a}$

and $f \in_0 F$

shows $\varepsilon(\downarrow NTMap)(\downarrow \mathfrak{b}_{PL} F) = \varepsilon(\downarrow NTMap)(\downarrow \mathfrak{a}_{PL} F) \circ_{A\mathfrak{C}} F'(\downarrow f)$

proof-

let $?II = \langle \uparrow_C (\mathfrak{b}_{PL} F) (\mathfrak{a}_{PL} F) F \rangle$

and $?II-II = \langle \uparrow \rightarrow \uparrow_{CF} \mathfrak{C} (\mathfrak{b}_{PL} F) (\mathfrak{a}_{PL} F) F \mathfrak{b} \mathfrak{a} F' \rangle$

interpret ε : *is-cat-cocone* $\alpha E ?II \mathfrak{C} ?II-II \varepsilon$ **by** (*rule assms(1)*)

from *assms(5,6)*

have $\mathfrak{a} : \mathfrak{a} \in_0 \mathfrak{C}(\downarrow Obj)$ **and** $\mathfrak{b} : \mathfrak{b} \in_0 \mathfrak{C}(\downarrow Obj)$ **and** $F'\downarrow f : F'(\downarrow f) : \mathfrak{b} \mapsto_{\mathfrak{C}} \mathfrak{a}$

by *auto*

interpret *par*: *cf-parallel* $\alpha \langle \mathfrak{b}_{PL} F \rangle \langle \mathfrak{a}_{PL} F \rangle F \mathfrak{b} \mathfrak{a} F' \mathfrak{C}$

by (*intro* $\varepsilon.NTDom.HomCod.cat-cf-parallel-ba$ *assms* $\mathfrak{a} \mathfrak{b}$)

note $\varepsilon-NTMap-app =$

cat-cone-cf-par-eps-NTMap-app[

OF $\varepsilon.is-cat-cone-op[unfolded cat-op-simps]$,

unfolded cat-op-simps,

OF *assms(2-6)*,

simplified

]

from $\varepsilon-NTMap-app$ $F'\downarrow f$ **show** *?thesis*

by

(
cs-concl cs-shallow
cs-simp: *cat-parallel-cs-simps category.op-cat-Comp[symmetric]*
cs-intro: *cat-cs-intros cat-parallel-cs-intros*
)
qed

lemma (in *is-cat-equalizer*) *cat-eq-eps-NTMap-app*:
assumes $f \in_{\circ} F$
shows $\varepsilon(\text{NTMap})(\mathbf{b}_{PL} F) = F'(\mathbf{f}) \circ_{A\mathfrak{C}} \varepsilon(\text{NTMap})(\mathbf{a}_{PL} F)$
by
(
intro cat-cone-cf-par-eps-NTMap-app
OF
is-cat-cone-axioms
F'.vsv-axioms
cat-eq-F-in-Vset
cat-eq-F'-vdomain
cat-eq-F'-app-is-arr
assms
)
)+

lemma (in *is-cat-coequalizer*) *cat-coeq-eps-NTMap-app*:
assumes $f \in_{\circ} F$
shows $\varepsilon(\text{NTMap})(\mathbf{b}_{PL} F) = \varepsilon(\text{NTMap})(\mathbf{a}_{PL} F) \circ_{A\mathfrak{C}} F'(\mathbf{f})$
by
(
intro cat-cocone-cf-par-eps-NTMap-app
OF is-cat-cocone-axioms
F'.vsv-axioms
cat-coeq-F-in-Vset
cat-coeq-F'-vdomain
cat-coeq-F'-app-is-arr
assms
)
)+

lemma (in *is-cat-equalizer*) *cat-eq-Comp-eq*:
assumes $g \in_{\circ} F$ **and** $f \in_{\circ} F$
shows $F'(\mathbf{g}) \circ_{A\mathfrak{C}} \varepsilon(\text{NTMap})(\mathbf{a}_{PL} F) = F'(\mathbf{f}) \circ_{A\mathfrak{C}} \varepsilon(\text{NTMap})(\mathbf{a}_{PL} F)$
using *cat-eq-eps-NTMap-app*[*OF assms(1)*] *cat-eq-eps-NTMap-app*[*OF assms(2)*]
by *auto*

lemma (in *is-cat-coequalizer*) *cat-coeq-Comp-eq*:
assumes $g \in_{\circ} F$ **and** $f \in_{\circ} F$
shows $\varepsilon(\text{NTMap})(\mathbf{a}_{PL} F) \circ_{A\mathfrak{C}} F'(\mathbf{g}) = \varepsilon(\text{NTMap})(\mathbf{a}_{PL} F) \circ_{A\mathfrak{C}} F'(\mathbf{f})$
using *cat-coeq-eps-NTMap-app*[*OF assms(1)*] *cat-coeq-eps-NTMap-app*[*OF assms(2)*]
by *auto*

7.1.2 Universal property

lemma *is-cat-equalizerI'*:
assumes ε :
 $E <_{CF.cone} \uparrow \rightarrow \uparrow_{CF} \mathfrak{C} (\mathbf{a}_{PL} F) (\mathbf{b}_{PL} F) F \mathbf{a} \mathbf{b} F'$:
 $\uparrow_C (\mathbf{a}_{PL} F) (\mathbf{b}_{PL} F) F \mapsto \mapsto_{C\alpha} \mathfrak{C}$
and *vsv F'*
and $F \in_{\circ} Vset \alpha$

and $\mathcal{D}_\circ F' = F$
 and $\wedge f. f \in_\circ F \implies F'(\downarrow f) : \mathbf{a} \mapsto_{\mathfrak{C}} \mathbf{b}$
 and $f \in_\circ F$
 and $\wedge \varepsilon' E'. \varepsilon' :$
 $E' <_{CF.cone} \uparrow \rightarrow \uparrow_{CF} \mathfrak{C}(\mathbf{a}_{PL} F) (\mathbf{b}_{PL} F) F \mathbf{a} \mathbf{b} F' :$
 $\uparrow_C (\mathbf{a}_{PL} F) (\mathbf{b}_{PL} F) F \mapsto_{\rightarrow C\alpha} \mathfrak{C} \implies$
 $\exists ! f'. f' : E' \mapsto_{\mathfrak{C}} E \wedge \varepsilon' (\downarrow NTMap) (\downarrow \mathbf{a}_{PL} F) = \varepsilon (\downarrow NTMap) (\downarrow \mathbf{a}_{PL} F) \circ_A \mathfrak{C} f'$
 shows $\varepsilon : E <_{CF.eq} (\mathbf{a}, \mathbf{b}, F, F') : \uparrow_C \mapsto_{\rightarrow C\alpha} \mathfrak{C}$

proof-

let $?II = \langle \uparrow_C (\mathbf{a}_{PL} F) (\mathbf{b}_{PL} F) F \rangle$
 and $?II-II = \langle \uparrow \rightarrow \uparrow_{CF} \mathfrak{C} (\mathbf{a}_{PL} F) (\mathbf{b}_{PL} F) F \mathbf{a} \mathbf{b} F' \rangle$
 interpret $\varepsilon : is-cat-cone \alpha E ?II \mathfrak{C} ?II-II \varepsilon$ by (rule *assms(1)*)
 from *assms(5,6)* have $\mathbf{a} : \mathbf{a} \in_\circ \mathfrak{C}(\downarrow Obj)$ and $\mathbf{b} : \mathbf{b} \in_\circ \mathfrak{C}(\downarrow Obj)$ by *auto*
 interpret *par*: *cf-parallel* $\alpha \langle \mathbf{a}_{PL} F \rangle \langle \mathbf{b}_{PL} F \rangle F \mathbf{a} \mathbf{b} F' \mathfrak{C}$
 by (*intro* $\varepsilon.NTDom.HomCod.cat-cf-parallel-ab$ *assms* $\mathbf{a} \mathbf{b}$) *simp*

show *thesis*

proof(*intro is-cat-equalizerI is-cat-limitI assms(1-3)*)

fix $u' r'$ **assume** *prems*: $u' : r' <_{CF.cone} ?II-II : ?II \mapsto_{\rightarrow C\alpha} \mathfrak{C}$

interpret $u' : is-cat-cone \alpha r' ?II \mathfrak{C} ?II-II u'$ by (rule *prems*)

from *assms(7)[OF prems]* **obtain** f'

where $f' : f' : r' \mapsto_{\mathfrak{C}} E$

and $u'-NTMap-app-a : u'(\downarrow NTMap) (\downarrow \mathbf{a}_{PL} F) = \varepsilon (\downarrow NTMap) (\downarrow \mathbf{a}_{PL} F) \circ_A \mathfrak{C} f'$

and *unique-f'*:

$\wedge f''.$

\llbracket

$f'' : r' \mapsto_{\mathfrak{C}} E;$

$u'(\downarrow NTMap) (\downarrow \mathbf{a}_{PL} F) = \varepsilon (\downarrow NTMap) (\downarrow \mathbf{a}_{PL} F) \circ_A \mathfrak{C} f''$

$\rrbracket \implies f'' = f'$

by *metis*

show $\exists ! f'. f' : r' \mapsto_{\mathfrak{C}} E \wedge u' = \varepsilon \cdot_{NTCF} ntcf-const ?II \mathfrak{C} f'$

proof(*intro exII conjI; (elim conjE)?*)

show $u' = \varepsilon \cdot_{NTCF} ntcf-const ?II \mathfrak{C} f'$

proof(rule *ntcf-eqI*)

show $u' : cf-const ?II \mathfrak{C} r' \mapsto_{CF} ?II-II : ?II \mapsto_{\rightarrow C\alpha} \mathfrak{C}$

by (rule *u'.is-ntcf-axioms*)

from f' **show** $\varepsilon \cdot_{NTCF} ntcf-const ?II \mathfrak{C} f' :$

cf-const $?II \mathfrak{C} r' \mapsto_{CF} ?II-II : ?II \mapsto_{\rightarrow C\alpha} \mathfrak{C}$

by (*cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

have *dom-lhs*: $\mathcal{D}_\circ (u'(\downarrow NTMap)) = ?II(\downarrow Obj)$

unfolding *cat-cs-simps* **by** *simp*

from f' **have** *dom-rhs*:

$\mathcal{D}_\circ ((\varepsilon \cdot_{NTCF} ntcf-const ?II \mathfrak{C} f')(\downarrow NTMap)) = ?II(\downarrow Obj)$

by (*cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

show $u'(\downarrow NTMap) = (\varepsilon \cdot_{NTCF} ntcf-const ?II \mathfrak{C} f')(\downarrow NTMap)$

proof(rule *vsv-eqI, unfold dom-lhs dom-rhs*)

fix a **assume** *prems'*: $a \in_\circ ?II(\downarrow Obj)$

note [*cat-parallel-cs-simps*] =

cat-cone-cf-par-eps-NTMap-app[

OF u'.is-cat-cone-axioms assms(2-5), simplified

]

cat-cone-cf-par-eps-NTMap-app[*OF assms(1-5), simplified*]

u'-NTMap-app-a

from *prems' f' assms(6)* **show**

$u'(\downarrow NTMap) (\downarrow a) = (\varepsilon \cdot_{NTCF} ntcf-const ?II \mathfrak{C} f')(\downarrow NTMap) (\downarrow a)$

by (*elim the-cat-parallel-ObjE; simp only*):

(

$cs\text{-concl}$
cs-simp: $cat\text{-parallel-cs-simps}$ $cat\text{-cs-simps}$
cs-intro: $cat\text{-cs-intros}$ $cat\text{-parallel-cs-intros}$

)

qed ($cs\text{-concl}$ **cs-intro**: $V\text{-cs-intros}$ $cat\text{-cs-intros}$)+

qed $simp\text{-all}$

fix f'' **assume** $prems''$:

$f'' : r' \mapsto_{\mathfrak{C}} E$ $u' = \varepsilon \cdot_{NTCF} ntcf\text{-const } ?II \mathfrak{C} f''$

from $prems''(2)$ **have** $u'\text{-NTMap-a}$:

$u'(\text{NTMap})(a) = (\varepsilon \cdot_{NTCF} ntcf\text{-const } ?II \mathfrak{C} f'')(\text{NTMap})(a)$

for a

by $simp$

have $u'(\text{NTMap})(\mathfrak{a}_{PL} F) = \varepsilon(\text{NTMap})(\mathfrak{a}_{PL} F) \circ_{A\mathfrak{C}} f''$

using $u'\text{-NTMap-a}[of \langle \mathfrak{a}_{PL} F \rangle]$ $prems''(1)$

by

(

$cs\text{-prems}$

cs-simp: $cat\text{-parallel-cs-simps}$ $cat\text{-cs-simps}$

cs-intro: $cat\text{-parallel-cs-intros}$ $cat\text{-cs-intros}$

)

from $unique\text{-f}'[OF prems''(1) \text{ this}]$ **show** $f'' = f'$.

qed ($rule f'$)

qed ($use\ assms\ in\ fastforce$)+

qed

lemma $is\text{-cat-coequalizerI}'$:

assumes ε :

$\uparrow\uparrow_{CF} \mathfrak{C} (\mathfrak{b}_{PL} F) (\mathfrak{a}_{PL} F) F \mathfrak{b} \mathfrak{a} F' >_{CF.cocone} E$:

$\uparrow_C (\mathfrak{b}_{PL} F) (\mathfrak{a}_{PL} F) F \mapsto_{C\alpha} \mathfrak{C}$

and $vsv F'$

and $F \in_0 Vset \alpha$

and $\mathcal{D}_0 F' = F$

and $\wedge f. f \in_0 F \implies F'(\{f\}) : \mathfrak{b} \mapsto_{\mathfrak{C}} \mathfrak{a}$

and $f \in_0 F$

and $\wedge \varepsilon' E'. \varepsilon' :$

$\uparrow\uparrow_{CF} \mathfrak{C} (\mathfrak{b}_{PL} F) (\mathfrak{a}_{PL} F) F \mathfrak{b} \mathfrak{a} F' >_{CF.cocone} E' :$

$\uparrow_C (\mathfrak{b}_{PL} F) (\mathfrak{a}_{PL} F) F \mapsto_{C\alpha} \mathfrak{C} \implies$

$\exists ! f'. f' : E \mapsto_{\mathfrak{C}} E' \wedge \varepsilon'(\text{NTMap})(\mathfrak{a}_{PL} F) = f' \circ_{A\mathfrak{C}} \varepsilon(\text{NTMap})(\mathfrak{a}_{PL} F)$

shows $\varepsilon : (\mathfrak{a}, \mathfrak{b}, F, F') >_{CF.coeq} E : \uparrow_C \mapsto_{C\alpha} \mathfrak{C}$

proof-

let $?op\text{-II} = \langle \uparrow_C (\mathfrak{b}_{PL} F) (\mathfrak{a}_{PL} F) F \rangle$

and $?op\text{-II-II} = \langle \uparrow\uparrow_{CF} \mathfrak{C} (\mathfrak{b}_{PL} F) (\mathfrak{a}_{PL} F) F \mathfrak{b} \mathfrak{a} F' \rangle$

and $?II = \langle \uparrow_C (\mathfrak{a}_{PL} F) (\mathfrak{b}_{PL} F) F \rangle$

and $?II\text{-II} = \langle \uparrow\uparrow_{CF} (op\text{-cat } \mathfrak{C}) (\mathfrak{a}_{PL} F) (\mathfrak{b}_{PL} F) F \mathfrak{a} \mathfrak{b} F' \rangle$

interpret ε : $is\text{-cat-cocone } \alpha E ?op\text{-II } \mathfrak{C} ?op\text{-II-II } \varepsilon$ **by** ($rule\ assms(1)$)

from $assms(5,6)$ **have** $\mathfrak{a} : \mathfrak{a} \in_0 \mathfrak{C}(\text{Obj})$ **and** $\mathfrak{b} : \mathfrak{b} \in_0 \mathfrak{C}(\text{Obj})$ **by** $auto$

interpret par : $cf\text{-parallel } \alpha \langle \mathfrak{b}_{PL} F \rangle \langle \mathfrak{a}_{PL} F \rangle F \mathfrak{b} \mathfrak{a} F' \mathfrak{C}$

by ($intro \varepsilon.NTDom.HomCod.cat\text{-cf-parallel-ba } assms \mathfrak{a} \mathfrak{b}$) $simp$

interpret $op\text{-par}$: $cf\text{-parallel } \alpha \langle \mathfrak{a}_{PL} F \rangle \langle \mathfrak{b}_{PL} F \rangle F \mathfrak{a} \mathfrak{b} F' \langle op\text{-cat } \mathfrak{C} \rangle$

by ($rule\ par.cf\text{-parallel-op}$)

have $assms\text{-4}'$:

$\exists ! f'. f' : E \mapsto_{\mathfrak{C}} E' \wedge \varepsilon'(\text{NTMap})(\mathfrak{a}_{PL} F) = \varepsilon(\text{NTMap})(\mathfrak{a}_{PL} F) \circ_{A\text{op-cat}} \mathfrak{C} f'$

if $\varepsilon' : E' <_{CF.cocone} ?II\text{-II} : ?II \mapsto_{C\alpha} op\text{-cat } \mathfrak{C}$ **for** $\varepsilon' E'$

proof-

have [$cat\text{-op-simps}$]:

$f' : E \mapsto_{\mathfrak{C}} E' \wedge \varepsilon'(\ulcorner NTMap \urcorner)(\ulcorner \mathfrak{a}_{PL} F \urcorner) = \varepsilon(\ulcorner NTMap \urcorner)(\ulcorner \mathfrak{a}_{PL} F \urcorner) \circ_{A \text{ op-cat } \mathfrak{C}} f' \longleftrightarrow$
 $f' : E \mapsto_{\mathfrak{C}} E' \wedge \varepsilon'(\ulcorner NTMap \urcorner)(\ulcorner \mathfrak{a}_{PL} F \urcorner) = f' \circ_{A \mathfrak{C}} \varepsilon(\ulcorner NTMap \urcorner)(\ulcorner \mathfrak{a}_{PL} F \urcorner)$
for f'
by (*intro iffI conjI; (elim conjE)?*)
 (
 cs-concl cs-shallow
 cs-simp: *category.op-cat-Comp[symmetric] cat-op-simps cat-cs-simps*
 cs-intro: *cat-cs-intros cat-parallel-cs-intros*
)+
interpret ε' : *is-cat-cone* $\alpha E' ?II \langle \text{op-cat } \mathfrak{C} \rangle ?II-II \varepsilon'$ **by** (*rule that*)
show *?thesis*
 unfolding *cat-op-simps*
 by
 (
 rule assms(7)[
 OF \varepsilon'.is-cat-cocone-op[unfolded cat-op-simps],
 unfolded cat-op-simps
]
)
qed
interpret $op-\varepsilon$: *is-cat-equalizer* $\alpha \mathfrak{a} \mathfrak{b} F F' \langle \text{op-cat } \mathfrak{C} \rangle E \langle \text{op-ntcf } \varepsilon \rangle$
by
 (
 rule
 is-cat-equalizerI'
 [
 OF \varepsilon.is-cat-cone-op[unfolded cat-op-simps],
 unfolded cat-op-simps,
 OF assms(2-6) assms-4',
 simplified
]
)
show *?thesis* **by** (*rule op-\varepsilon.is-cat-coequalizer-op[unfolded cat-op-simps]*)

qed

lemma (*in is-cat-equalizer*) *cat-eq-unique-cone*:

assumes ε' :

$E' <_{CF.cone} \uparrow \uparrow_{CF} \mathfrak{C} (\ulcorner \mathfrak{a}_{PL} F \urcorner) (\ulcorner \mathfrak{b}_{PL} F \urcorner) F \mathfrak{a} \mathfrak{b} F' : \uparrow_C (\ulcorner \mathfrak{a}_{PL} F \urcorner) (\ulcorner \mathfrak{b}_{PL} F \urcorner) F \mapsto_{C \alpha} \mathfrak{C}$

(**is** $\langle \varepsilon' : E' <_{CF.cone} ?II-II : ?II \mapsto_{C \alpha} \mathfrak{C} \rangle$)

shows $\exists ! f'. f' : E' \mapsto_{\mathfrak{C}} E \wedge \varepsilon'(\ulcorner NTMap \urcorner)(\ulcorner \mathfrak{a}_{PL} F \urcorner) = \varepsilon(\ulcorner NTMap \urcorner)(\ulcorner \mathfrak{a}_{PL} F \urcorner) \circ_{A \mathfrak{C}} f'$

proof-

interpret ε' : *is-cat-cone* $\alpha E' ?II \mathfrak{C} ?II-II \varepsilon'$ **by** (*rule assms(1)*)

from *cat-lim-ua-fo[OF assms(1)]* **obtain** f' **where** $f' : E' \mapsto_{\mathfrak{C}} E$

and ε' -*def*: $\varepsilon' = \varepsilon \cdot_{NTCF} \text{ntcf-const } ?II \mathfrak{C} f'$

and *unique*:

$\llbracket f'' : E' \mapsto_{\mathfrak{C}} E; \varepsilon' = \varepsilon \cdot_{NTCF} \text{ntcf-const } ?II \mathfrak{C} f'' \rrbracket \implies f'' = f'$

for f''

by *auto*

from *cat-eq-F-ne* **obtain** f **where** $f : f \in_{\circ} F$ **by** *force*

show *?thesis*

proof(*intro ex1I conjI; (elim conjE)?*)

show $f' : E' \mapsto_{\mathfrak{C}} E$ **by** (*rule f'*)

from ε' -*def* **have** $\varepsilon'(\ulcorner NTMap \urcorner)(\ulcorner \mathfrak{a}_{PL} F \urcorner) = (\varepsilon \cdot_{NTCF} \text{ntcf-const } ?II \mathfrak{C} f')(\ulcorner NTMap \urcorner)(\ulcorner \mathfrak{a}_{PL} F \urcorner)$

by *simp*

from *this f'* **show** $\varepsilon' \text{-} NTMap \text{-} app \text{-} I : \varepsilon'(\ulcorner NTMap \urcorner)(\ulcorner \mathfrak{a}_{PL} F \urcorner) = \varepsilon(\ulcorner NTMap \urcorner)(\ulcorner \mathfrak{a}_{PL} F \urcorner) \circ_{A \mathfrak{C}} f'$

```

by
  (
    cs-prems
    cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-parallel-cs-intros
  )
fix f'' assume prems:
  f'' : E'  $\mapsto_{\mathfrak{C}}$  E  $\varepsilon'$ ( $\downarrow$ NTMap)( $\downarrow$ aPL F) =  $\varepsilon$ ( $\downarrow$ NTMap)( $\downarrow$ aPL F)  $\circ_{A\mathfrak{C}}$  f''
have  $\varepsilon' = \varepsilon \cdot_{NTCF} ntcf\text{-const } ?II \mathfrak{C} f''$ 
proof(rule ntcf-eqI[OF ])
  show  $\varepsilon' : cf\text{-const } ?II \mathfrak{C} E' \mapsto_{CF} ?II-II : ?II \mapsto \mapsto_{C\alpha} \mathfrak{C}$ 
    by (rule  $\varepsilon'$ .is-ntcf-axioms)
  from f' prems(1) show  $\varepsilon \cdot_{NTCF} ntcf\text{-const } ?II \mathfrak{C} f'' : cf\text{-const } ?II \mathfrak{C} E' \mapsto_{CF} ?II-II : ?II \mapsto \mapsto_{C\alpha} \mathfrak{C}$ 
    by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  show  $\varepsilon'(\downarrow$ NTMap) = ( $\varepsilon \cdot_{NTCF} ntcf\text{-const } ?II \mathfrak{C} f''$ )( $\downarrow$ NTMap)
  proof(rule vsv-eqI, unfold cat-cs-simps)
    show vsv (( $\varepsilon \cdot_{NTCF} ntcf\text{-const } ?II \mathfrak{C} f''$ )( $\downarrow$ NTMap))
      by (cs-concl cs-intro: cat-cs-intros)
    from prems(1) show  $?II(\downarrow$ Obj) =  $\mathcal{D}_\circ$  (( $\varepsilon \cdot_{NTCF} ntcf\text{-const } ?II \mathfrak{C} f''$ )( $\downarrow$ NTMap))
      by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    fix a assume prems': a  $\in_\circ ?II(\downarrow$ Obj)
    note [cat-cs-simps] =
      cat-eq-eps-NTMap-app[OF f]
      cat-cone-cf-par-eps-NTMap-app
    [
      OF
       $\varepsilon'$ .is-cat-cone-axioms
      F'.vsv-axioms
      cat-eq-F-in-Vset
      cat-eq-F'-vdomain
      cat-eq-F'-app-is-arr f,
      simplified
    ]
    from prems' prems(1) f have [cat-cs-simps]:
       $\varepsilon'(\downarrow$ NTMap)( $\downarrow$ a) =  $\varepsilon(\downarrow$ NTMap)( $\downarrow$ a)  $\circ_{A\mathfrak{C}}$  f''
      by (elim the-cat-parallel-ObjE; simp only:)
      (
        cs-concl
        cs-simp: cat-cs-simps cat-parallel-cs-simps prems(2)
        cs-intro: cat-cs-intros cat-parallel-cs-intros
      )+
    from prems' prems show
       $\varepsilon'(\downarrow$ NTMap)( $\downarrow$ a) = ( $\varepsilon \cdot_{NTCF} ntcf\text{-const } ?II \mathfrak{C} f''$ )( $\downarrow$ NTMap)( $\downarrow$ a)
      by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    qed auto
    qed simp-all
    from unique[OF prems(1) this] show f'' = f' .
  qed

```

qed

lemma (in is-cat-equalizer) cat-eq-unique:

assumes $\varepsilon' : E' <_{CF.eq} (a, b, F, F') : \uparrow_C \mapsto \mapsto_{C\alpha} \mathfrak{C}$

shows

$\exists !f'. f' : E' \mapsto_{\mathfrak{C}} E \wedge \varepsilon' = \varepsilon \cdot_{NTCF} ntcf\text{-const } (\uparrow_C (a_{PL} F) (b_{PL} F) F) \mathfrak{C} f'$

by (rule cat-lim-unique[OF is-cat-equalizerD(1)[OF assms]])

lemma (in is-cat-equalizer) cat-eq-unique':

assumes $\varepsilon' : E' <_{CF.eq} (a, b, F, F') : \uparrow_C \mapsto_{C\alpha} \mathfrak{C}$
shows $\exists ! f'. f' : E' \mapsto_{\mathfrak{C}} E \wedge \varepsilon'(\mathit{NTMap})(\mathit{a}_{PL} F) = \varepsilon(\mathit{NTMap})(\mathit{a}_{PL} F) \circ_{A\mathfrak{C}} f'$
proof-
interpret ε' : *is-cat-equalizer* α a b F F' \mathfrak{C} E' ε' **by** (*rule assms(1)*)
show *?thesis* **by** (*rule cat-eq-unique-cone[OF ε' .is-cat-cone-axioms]*)
qed

lemma (*in is-cat-coequalizer*) *cat-coeq-unique-cocone*:

assumes ε' :
 $\uparrow \rightarrow \uparrow_{CF} \mathfrak{C} (b_{PL} F) (a_{PL} F) F b a F' >_{CF.coccone} E'$
 $\uparrow_C (b_{PL} F) (a_{PL} F) F \mapsto_{C\alpha} \mathfrak{C}$
(is $\langle \varepsilon' : ?II-II >_{CF.coccone} E' : ?II \mapsto_{C\alpha} \mathfrak{C} \rangle$ **)**
shows $\exists ! f'. f' : E \mapsto_{\mathfrak{C}} E' \wedge \varepsilon'(\mathit{NTMap})(\mathit{a}_{PL} F) = f' \circ_{A\mathfrak{C}} \varepsilon(\mathit{NTMap})(\mathit{a}_{PL} F)$

proof-

interpret ε' : *is-cat-cocone* α E' $?II$ \mathfrak{C} $?II-II$ ε' **by** (*rule assms(1)*)

have [*cat-op-simps*]:

$f' : E \mapsto_{\mathfrak{C}} E' \wedge \varepsilon'(\mathit{NTMap})(\mathit{a}_{PL} F) = \varepsilon(\mathit{NTMap})(\mathit{a}_{PL} F) \circ_{A\mathfrak{C}} f' \longleftrightarrow$
 $f' : E \mapsto_{\mathfrak{C}} E' \wedge \varepsilon'(\mathit{NTMap})(\mathit{a}_{PL} F) = f' \circ_{A\mathfrak{C}} \varepsilon(\mathit{NTMap})(\mathit{a}_{PL} F)$

for f'

by (*intro iffI conjI; (elim conjE)?*)

(

cs-concl cs-shallow

cs-simp: *category.op-cat-Comp[symmetric] cat-op-simps cat-cs-simps*

cs-intro: *cat-cs-intros cat-parallel-cs-intros*

)+

show *?thesis*

by

(

rule is-cat-equalizer.cat-eq-unique-cone[
OF is-cat-equalizer-op ε' .is-cat-cone-op[unfolding cat-op-simps],
unfolding cat-op-simps

]

)

qed

lemma (*in is-cat-coequalizer*) *cat-coeq-unique*:

assumes $\varepsilon' : (a, b, F, F') >_{CF.coeq} E' : \uparrow_C \mapsto_{C\alpha} \mathfrak{C}$

shows $\exists ! f'$.

$f' : E \mapsto_{\mathfrak{C}} E' \wedge \varepsilon' = \mathit{ntcf-const} (\uparrow_C (b_{PL} F) (a_{PL} F) F) \mathfrak{C} f' \cdot_{NTCF} \varepsilon$

by (*rule cat-colim-unique[OF is-cat-coequalizerD(1)[OF assms]]*)

lemma (*in is-cat-coequalizer*) *cat-coeq-unique'*:

assumes $\varepsilon' : (a, b, F, F') >_{CF.coeq} E' : \uparrow_C \mapsto_{C\alpha} \mathfrak{C}$

shows $\exists ! f'. f' : E \mapsto_{\mathfrak{C}} E' \wedge \varepsilon'(\mathit{NTMap})(\mathit{a}_{PL} F) = f' \circ_{A\mathfrak{C}} \varepsilon(\mathit{NTMap})(\mathit{a}_{PL} F)$

proof-

interpret ε' : *is-cat-coequalizer* α a b F F' \mathfrak{C} E' ε' **by** (*rule assms(1)*)

show *?thesis* **by** (*rule cat-coeq-unique-cocone[OF ε' .is-cat-cocone-axioms]*)

qed

lemma *cat-equalizer-ex-is-iso-arr*:

assumes $\varepsilon : E <_{CF.eq} (a, b, F, F') : \uparrow_C \mapsto_{C\alpha} \mathfrak{C}$

and $\varepsilon' : E' <_{CF.eq} (a, b, F, F') : \uparrow_C \mapsto_{C\alpha} \mathfrak{C}$

obtains f **where** $f : E' \mapsto_{iso\mathfrak{C}} E$

and $\varepsilon' = \varepsilon \cdot_{NTCF} \mathit{ntcf-const} (\uparrow_C (a_{PL} F) (b_{PL} F) F) \mathfrak{C} f$

proof-

interpret ε : *is-cat-equalizer* α a b F F' \mathfrak{C} E ε **by** (*rule assms(1)*)

interpret ε' : *is-cat-equalizer* α a b F F' \mathfrak{C} E' ε' **by** (*rule assms(2)*)

from that show *?thesis*

by
 (
 elim cat-lim-ex-is-iso-arr[
 OF ε .is-cat-limit-axioms ε' .is-cat-limit-axioms
]
)
 qed

lemma *cat-equalizer-ex-is-iso-arr'*:

assumes $\varepsilon : E <_{CF.eq} (a, b, F, F') : \uparrow_C \mapsto \rightarrow_{C\alpha} \mathfrak{C}$
 and $\varepsilon' : E' <_{CF.eq} (a, b, F, F') : \uparrow_C \mapsto \rightarrow_{C\alpha} \mathfrak{C}$
 obtains f where $f : E' \mapsto_{iso} \mathfrak{C} E$
 and $\varepsilon'(NTMap)(\mathfrak{a}_{PL} F) = \varepsilon(NTMap)(\mathfrak{a}_{PL} F) \circ_{A\mathfrak{C}} f$
 and $\varepsilon'(NTMap)(\mathfrak{b}_{PL} F) = \varepsilon(NTMap)(\mathfrak{b}_{PL} F) \circ_{A\mathfrak{C}} f$

proof–

interpret ε : is-cat-equalizer α a b F F' \mathfrak{C} E ε **by** (rule *assms(1)*)

interpret ε' : is-cat-equalizer α a b F F' \mathfrak{C} E' ε' **by** (rule *assms(2)*)

obtain f where $f : E' \mapsto_{iso} \mathfrak{C} E$

and $j \in \circ \uparrow_C (\mathfrak{a}_{PL} F) (\mathfrak{b}_{PL} F) F(Obj) \implies \varepsilon'(NTMap)(j) = \varepsilon(NTMap)(j) \circ_{A\mathfrak{C}} f$ **for** j

by

(
 elim cat-lim-ex-is-iso-arr'[
 OF ε .is-cat-limit-axioms ε' .is-cat-limit-axioms
]
)

then have

$\varepsilon'(NTMap)(\mathfrak{a}_{PL} F) = \varepsilon(NTMap)(\mathfrak{a}_{PL} F) \circ_{A\mathfrak{C}} f$

$\varepsilon'(NTMap)(\mathfrak{b}_{PL} F) = \varepsilon(NTMap)(\mathfrak{b}_{PL} F) \circ_{A\mathfrak{C}} f$

unfolding *the-cat-parallel-components* **by** *auto*

with f **show** *?thesis* **using** *that* **by** *simp*

qed

lemma *cat-coequalizer-ex-is-iso-arr*:

assumes $\varepsilon : (a, b, F, F') >_{CF.coeq} E : \uparrow_C \mapsto \rightarrow_{C\alpha} \mathfrak{C}$
 and $\varepsilon' : (a, b, F, F') >_{CF.coeq} E' : \uparrow_C \mapsto \rightarrow_{C\alpha} \mathfrak{C}$
 obtains f where $f : E \mapsto_{iso} \mathfrak{C} E'$
 and $\varepsilon' = ntcf-const (\uparrow_C (\mathfrak{b}_{PL} F) (\mathfrak{a}_{PL} F) F) \mathfrak{C} f \cdot_{NTCF} \varepsilon$

proof–

interpret ε : is-cat-coequalizer α a b F F' \mathfrak{C} E ε **by** (rule *assms(1)*)

interpret ε' : is-cat-coequalizer α a b F F' \mathfrak{C} E' ε' **by** (rule *assms(2)*)

from *that* **show** *?thesis*

by

(
 elim cat-colim-ex-is-iso-arr[
 OF ε .is-cat-colimit-axioms ε' .is-cat-colimit-axioms
]
)

qed

lemma *cat-coequalizer-ex-is-iso-arr'*:

assumes $\varepsilon : (a, b, F, F') >_{CF.coeq} E : \uparrow_C \mapsto \rightarrow_{C\alpha} \mathfrak{C}$
 and $\varepsilon' : (a, b, F, F') >_{CF.coeq} E' : \uparrow_C \mapsto \rightarrow_{C\alpha} \mathfrak{C}$
 obtains f where $f : E \mapsto_{iso} \mathfrak{C} E'$
 and $\varepsilon'(NTMap)(\mathfrak{a}_{PL} F) = f \circ_{A\mathfrak{C}} \varepsilon(NTMap)(\mathfrak{a}_{PL} F)$
 and $\varepsilon'(NTMap)(\mathfrak{b}_{PL} F) = f \circ_{A\mathfrak{C}} \varepsilon(NTMap)(\mathfrak{b}_{PL} F)$

proof–

interpret ε : is-cat-coequalizer α a b F F' \mathfrak{C} E ε **by** (rule *assms(1)*)

interpret ε' : is-cat-coequalizer α a b F F' \mathfrak{C} E' ε' **by** (rule *assms(2)*)

obtain f **where** $f: E \mapsto_{\text{iso}\mathfrak{C}} E'$
and $j \in_o \uparrow_C (\mathfrak{b}_{PL} F) (\mathfrak{a}_{PL} F) F(\text{Obj}) \implies \varepsilon'(\text{NTMap})(j) = f \circ_{A\mathfrak{C}} \varepsilon(\text{NTMap})(j)$ **for** j
by
(

elim cat-colim-ex-is-iso-arr[
OF *ε.is-cat-colimit-axioms* *ε'.is-cat-colimit-axioms*
]

)

then have
 $\varepsilon'(\text{NTMap})(\mathfrak{a}_{PL} F) = f \circ_{A\mathfrak{C}} \varepsilon(\text{NTMap})(\mathfrak{a}_{PL} F)$
 $\varepsilon'(\text{NTMap})(\mathfrak{b}_{PL} F) = f \circ_{A\mathfrak{C}} \varepsilon(\text{NTMap})(\mathfrak{b}_{PL} F)$
unfolding *the-cat-parallel-components* **by** *auto*
with f **show** *?thesis* **using** *that* **by** *simp*
qed

7.1.3 Further properties

lemma (in *is-cat-equalizer*) *cat-eq-is-monic-arr*:

— See subsection 3.3 in [3].

$\varepsilon(\text{NTMap})(\mathfrak{a}_{PL} F) : E \mapsto_{\text{mon}\mathfrak{C}} \mathfrak{a}$

proof(*intro is-monic-arrI*)

show $\varepsilon(\text{NTMap})(\mathfrak{a}_{PL} F) : E \mapsto_{\mathfrak{C}} \mathfrak{a}$

by

(

cs-concl
cs-simp: *cat-cs-simps* *cat-parallel-cs-simps*
cs-intro: *cat-cs-intros* *cat-parallel-cs-intros*
)

fix $f g a$

assume *prems*:

$f : a \mapsto_{\mathfrak{C}} E$

$g : a \mapsto_{\mathfrak{C}} E$

$\varepsilon(\text{NTMap})(\mathfrak{a}_{PL} F) \circ_{A\mathfrak{C}} f = \varepsilon(\text{NTMap})(\mathfrak{a}_{PL} F) \circ_{A\mathfrak{C}} g$

define ε' **where** $\varepsilon' = \varepsilon \cdot_{NTCF} \text{ntcf-const} (\uparrow_C (\mathfrak{a}_{PL} F) (\mathfrak{b}_{PL} F) F) \mathfrak{C} f$

from *prems(1)* **have** ε' :

$a <_{CF} \text{cone} \uparrow \uparrow_{CF} \mathfrak{C} (\mathfrak{a}_{PL} F) (\mathfrak{b}_{PL} F) F \mathfrak{a} \mathfrak{b} F'$:

$\uparrow_C (\mathfrak{a}_{PL} F) (\mathfrak{b}_{PL} F) F \mapsto_{C\alpha} \mathfrak{C}$

unfolding ε' -*def*

by (*cs-concl* **cs-shallow** **cs-intro**: *is-cat-coneI* *cat-cs-intros*)

from *cat-eq-unique-cone*[*OF this*] **obtain** f'

where $f' : f' : a \mapsto_{\mathfrak{C}} E$

and $\varepsilon' \cdot \mathfrak{a} : \varepsilon'(\text{NTMap})(\mathfrak{a}_{PL} F) = \varepsilon(\text{NTMap})(\mathfrak{a}_{PL} F) \circ_{A\mathfrak{C}} f'$

and *unique-f'*: $\wedge f''$.

$[[f'' : a \mapsto_{\mathfrak{C}} E; \varepsilon'(\text{NTMap})(\mathfrak{a}_{PL} F) = \varepsilon(\text{NTMap})(\mathfrak{a}_{PL} F) \circ_{A\mathfrak{C}} f'']] \implies f'' = f'$

by *meson*

from *prems(1)* **have** *unique-f*: $\varepsilon'(\text{NTMap})(\mathfrak{a}_{PL} F) = \varepsilon(\text{NTMap})(\mathfrak{a}_{PL} F) \circ_{A\mathfrak{C}} f$

unfolding ε' -*def*

by

(

cs-concl
cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros* *cat-parallel-cs-intros*
)

from *prems(1)* **have** *unique-g*: $\varepsilon'(\text{NTMap})(\mathfrak{a}_{PL} F) = \varepsilon(\text{NTMap})(\mathfrak{a}_{PL} F) \circ_{A\mathfrak{C}} g$

unfolding ε' -*def*

by

(

cs-concl

```

      cs-simp: prems(3) cat-cs-simps
      cs-intro: cat-cs-intros cat-parallel-cs-intros
    )
  show  $f = g$ 
  by
  (
    rule unique-f'
    [
      OF prems(1) unique-f,
      unfolded unique-f'[OF prems(2) unique-g, symmetric]
    ]
  )
qed

```

```

lemma (in is-cat-coequalizer) cat-coeq-is-epic-arr:
   $\varepsilon(\text{NTMap})(\text{a}_{PL} F) : \mathbf{a} \mapsto_{\text{epi}} \mathcal{C} E$ 
  by
  (
    rule is-cat-equalizer.cat-eq-is-monic-arr[
      OF is-cat-equalizer-op, unfolded cat-op-simps
    ]
  )

```

7.2 Equalizer and coequalizer for two arrows

7.2.1 Definition and elementary properties

See [2]⁷.

```

locale is-cat-equalizer-2 =
  is-cat-limit  $\alpha \langle \uparrow_C \mathbf{a}_{PL2} \mathbf{b}_{PL2} \mathbf{g}_{PL} \mathbf{f}_{PL} \rangle \mathcal{C} \langle \uparrow \rightarrow \uparrow_{CF} \mathcal{C} \mathbf{a}_{PL2} \mathbf{b}_{PL2} \mathbf{g}_{PL} \mathbf{f}_{PL} \mathbf{a} \mathbf{b} \mathbf{g} \mathbf{f} \rangle E \varepsilon$ 
  for  $\alpha \mathbf{a} \mathbf{b} \mathbf{g} \mathbf{f} \mathcal{C} E \varepsilon +$ 
  assumes cat-eq-g[cat-lim-cs-intros]:  $\mathbf{g} : \mathbf{a} \mapsto_{\mathcal{C}} \mathbf{b}$ 
  and cat-eq-f[cat-lim-cs-intros]:  $\mathbf{f} : \mathbf{a} \mapsto_{\mathcal{C}} \mathbf{b}$ 

```

```

syntax -is-cat-equalizer-2 ::  $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$ 
  ( $\langle (- \text{ :/ } - <_{CF.eq} '(-,-,-)' \text{ :/ } \uparrow_C \mapsto_{C1} -) \rangle$  [51, 51, 51, 51, 51, 51] 51)

```

```

syntax-consts -is-cat-equalizer-2  $\equiv$  is-cat-equalizer-2

```

```

translations  $\varepsilon : E <_{CF.eq} (\mathbf{a}, \mathbf{b}, \mathbf{g}, \mathbf{f}) : \uparrow_C \mapsto_{C\alpha} \mathcal{C} \equiv$ 
  CONST is-cat-equalizer-2  $\alpha \mathbf{a} \mathbf{b} \mathbf{g} \mathbf{f} \mathcal{C} E \varepsilon$ 

```

```

locale is-cat-coequalizer-2 =
  is-cat-colimit
   $\alpha \langle \uparrow_C \mathbf{b}_{PL2} \mathbf{a}_{PL2} \mathbf{f}_{PL} \mathbf{g}_{PL} \rangle \mathcal{C} \langle \uparrow \rightarrow \uparrow_{CF} \mathcal{C} \mathbf{b}_{PL2} \mathbf{a}_{PL2} \mathbf{f}_{PL} \mathbf{g}_{PL} \mathbf{b} \mathbf{a} \mathbf{f} \mathbf{g} \rangle E \varepsilon$ 
  for  $\alpha \mathbf{a} \mathbf{b} \mathbf{g} \mathbf{f} \mathcal{C} E \varepsilon +$ 
  assumes cat-coeq-g[cat-lim-cs-intros]:  $\mathbf{g} : \mathbf{b} \mapsto_{\mathcal{C}} \mathbf{a}$ 
  and cat-coeq-f[cat-lim-cs-intros]:  $\mathbf{f} : \mathbf{b} \mapsto_{\mathcal{C}} \mathbf{a}$ 

```

```

syntax -is-cat-coequalizer-2 ::  $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$ 
  ( $\langle (- \text{ :/ } '(-,-,-)' >_{CF.coeq} - \text{ :/ } \uparrow_C \mapsto_{C1} -) \rangle$  [51, 51, 51, 51, 51, 51] 51)

```

```

syntax-consts -is-cat-coequalizer-2  $\equiv$  is-cat-coequalizer-2

```

```

translations  $\varepsilon : (\mathbf{a}, \mathbf{b}, \mathbf{g}, \mathbf{f}) >_{CF.coeq} E : \uparrow_C \mapsto_{C\alpha} \mathcal{C} \equiv$ 
  CONST is-cat-coequalizer-2  $\alpha \mathbf{a} \mathbf{b} \mathbf{g} \mathbf{f} \mathcal{C} E \varepsilon$ 

```

Rules.

```

lemma (in is-cat-equalizer-2) is-cat-equalizer-2-axioms'[cat-lim-cs-intros]:
  assumes  $\alpha' = \alpha$ 

```

⁷[https://en.wikipedia.org/wiki/Equaliser_\(mathematics\)](https://en.wikipedia.org/wiki/Equaliser_(mathematics))

and $E' = E$
and $\mathbf{a}' = \mathbf{a}$
and $\mathbf{b}' = \mathbf{b}$
and $\mathbf{g}' = \mathbf{g}$
and $\mathbf{f}' = \mathbf{f}$
and $\mathfrak{C}' = \mathfrak{C}$
shows $\varepsilon : E' <_{CF.eq} (\mathbf{a}', \mathbf{b}', \mathbf{g}', \mathbf{f}') : \uparrow \uparrow_C \mapsto \mapsto_{C_{\alpha'}} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-equalizer-2-axioms*)

mk-ide rf *is-cat-equalizer-2-def*[*unfolded is-cat-equalizer-2-axioms-def*]
|*intro is-cat-equalizer-2I*]
|*dest is-cat-equalizer-2D*[*dest*]
|*elim is-cat-equalizer-2E*[*elim*]

lemmas [*cat-lim-cs-intros*] = *is-cat-equalizer-2D*(1)

lemma (**in** *is-cat-coequalizer-2*) *is-cat-coequalizer-2-axioms'*[*cat-lim-cs-intros*]:
assumes $\alpha' = \alpha$
and $E' = E$
and $\mathbf{a}' = \mathbf{a}$
and $\mathbf{b}' = \mathbf{b}$
and $\mathbf{g}' = \mathbf{g}$
and $\mathbf{f}' = \mathbf{f}$
and $\mathfrak{C}' = \mathfrak{C}$
shows $\varepsilon : (\mathbf{a}', \mathbf{b}', \mathbf{g}', \mathbf{f}') >_{CF.coeq} E' : \uparrow \uparrow_C \mapsto \mapsto_{C_{\alpha'}} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-coequalizer-2-axioms*)

mk-ide rf *is-cat-coequalizer-2-def*[*unfolded is-cat-coequalizer-2-axioms-def*]
|*intro is-cat-coequalizer-2I*]
|*dest is-cat-coequalizer-2D*[*dest*]
|*elim is-cat-coequalizer-2E*[*elim*]

lemmas [*cat-lim-cs-intros*] = *is-cat-coequalizer-2D*(1)

Helper lemmas.

lemma *cat-eq-F'-helper*:
 $(\lambda f \in_o \text{set } \{\mathbf{f}_{PL}, \mathbf{g}_{PL}\}. (f = \mathbf{g}_{PL} ? \mathbf{g} : \mathbf{f})) =$
 $(\lambda f \in_o \text{set } \{\mathbf{f}_{PL}, \mathbf{g}_{PL}\}. (f = \mathbf{f}_{PL} ? \mathbf{f} : \mathbf{g}))$
using *cat-PL2-gf* **by** (*simp add: VLambda-vdoubleton*)

Elementary properties.

sublocale *is-cat-equalizer-2* \subseteq *cf-parallel-2* α \mathbf{a}_{PL2} \mathbf{b}_{PL2} \mathbf{g}_{PL} \mathbf{f}_{PL} \mathbf{a} \mathbf{b} \mathbf{g} \mathbf{f} \mathfrak{C}
by (*intro cf-parallel-2I cat-parallel-2I*)
(*simp-all add: cat-parallel-cs-intros cat-lim-cs-intros cat-cs-intros*)

sublocale *is-cat-coequalizer-2* \subseteq *cf-parallel-2* α \mathbf{b}_{PL2} \mathbf{a}_{PL2} \mathbf{f}_{PL} \mathbf{g}_{PL} \mathbf{b} \mathbf{a} \mathbf{f} \mathbf{g} \mathfrak{C}
by (*intro cf-parallel-2I cat-parallel-2I*)
(
auto simp:
cat-parallel-cs-intros cat-lim-cs-intros cat-cs-intros
cat-PL2-ineq[symmetric]
)

lemma (**in** *is-cat-equalizer-2*) *cat-equalizer-2-is-cat-equalizer*:
 $\varepsilon :$
 $E <_{CF.eq} (\mathbf{a}, \mathbf{b}, \text{set } \{\mathbf{g}_{PL}, \mathbf{f}_{PL}\}, (\lambda f \in_o \text{set } \{\mathbf{g}_{PL}, \mathbf{f}_{PL}\}. (f = \mathbf{f}_{PL} ? \mathbf{f} : \mathbf{g}))) :$
 $\uparrow \uparrow_C \mapsto \mapsto_{C_{\alpha}} \mathfrak{C}$
by

```

(
  intro is-cat-equalizerI,
  rule is-cat-limit-axioms[
    unfolded the-cf-parallel-2-def the-cat-parallel-2-def aPL2-def bPL2-def
  ]
)
(auto simp: Limit-vdoubleton-in-VsetI cat-parallel-cs-intros)

```

lemma (in is-cat-coequalizer-2) cat-coequalizer-2-is-cat-coequalizer:

```

ε :
  (a,b,set {gPL, fPL},(λf∈oset {gPL, fPL}. (f = fPL ? f : g))) >CF.coeq E :
  ↑C ↦↦Cα C

```

proof

```

(
  intro is-cat-coequalizerI,
  fold the-cf-parallel-2-def the-cat-parallel-2-def aPL2-def bPL2-def
)

```

show ε :

```

↑↑→↑CF C bPL2 aPL2 gPL fPL b a g f >CF.colim E :
↑↑C bPL2 aPL2 gPL fPL ↦↦Cα C

```

by

```

(
  subst the-cat-parallel-2-commute,
  subst cf-parallel-2-the-cf-parallel-2-commute[symmetric]
)

```

(intro is-cat-colimit-axioms)

qed (auto simp: Limit-vdoubleton-in-VsetI cat-parallel-cs-intros)

lemma cat-equalizer-is-cat-equalizer-2:

assumes ε :

```

E <CF.eq (a,b,set {gPL, fPL},(λf∈oset {gPL, fPL}. (f = fPL ? f : g))) :
↑↑C ↦↦Cα C

```

shows ε : E <_{CF.eq} (a,b,g,f) : ↑↑_C ↦↦_{Cα} C

proof-

interpret ε: is-cat-equalizer

```

α a b <set {gPL, fPL} > (λf∈oset {gPL, fPL}. (f = fPL ? f : g)) > C E ε

```

by (rule assms)

have f_{PL}: f_{PL} ∈_o set {g_{PL}, f_{PL}} **and** g_{PL}: g_{PL} ∈_o set {g_{PL}, f_{PL}} **by** auto

show ?thesis

```

using ε.cat-eq-F'-app-is-arr[OF gPL] ε.cat-eq-F'-app-is-arr[OF fPL]

```

by

```

(
  intro
    is-cat-equalizer-2I
    ε.is-cat-limit-axioms
  [
    folded
      the-cf-parallel-2-def the-cat-parallel-2-def aPL2-def bPL2-def
  ]
)

```

(auto simp: cat-PL2-gf)

qed

lemma cat-coequalizer-is-cat-coequalizer-2:

assumes ε :

```

(a,b,set {gPL, fPL},(λf∈oset {gPL, fPL}. (f = fPL ? f : g))) >CF.coeq E :
↑↑C ↦↦Cα C

```

shows ε : (a,b,g,f) >_{CF.coeq} E : ↑↑_C ↦↦_{Cα} C

proof-

interpret *is-cat-coequalizer*

$\alpha \mathbf{a} \mathbf{b} \langle \text{set } \{\mathbf{g}_{PL}, \mathbf{f}_{PL}\} \rangle \langle (\lambda f \in_o \text{set } \{\mathbf{g}_{PL}, \mathbf{f}_{PL}\}. (f = \mathbf{f}_{PL} ? f : \mathbf{g})) \rangle \mathfrak{C} E \varepsilon$

by (*rule assms*)

interpret *cf-parallel-2* $\alpha \mathbf{b}_{PL2} \mathbf{a}_{PL2} \mathbf{g}_{PL} \mathbf{f}_{PL} \mathbf{b} \mathbf{a} \mathbf{g} \mathbf{f} \mathfrak{C}$

by

(
 rule cf-parallel-is-cf-parallel-2[
 OF cf-parallel-axioms cat-PL2-gf, folded a_{PL2}-def b_{PL2}-def
]
)

show $\varepsilon : (\mathbf{a}, \mathbf{b}, \mathbf{g}, \mathbf{f}) >_{CF.coeq} E : \uparrow \uparrow_C \mapsto \mapsto_{C\alpha} \mathfrak{C}$

by

(
 intro is-cat-coequalizer-2I,
 subst the-cat-parallel-2-commute,
 subst cf-parallel-2-the-cf-parallel-2-commute[symmetric],
 rule is-cat-colimit-axioms[
 folded a_{PL2}-def b_{PL2}-def the-cat-parallel-2-def the-cf-parallel-2-def
]
)

(*simp-all add: cf-parallel-f' cf-parallel-g'*)

qed

Duality.

lemma (*in is-cat-equalizer-2*) *is-cat-coequalizer-2-op:*

op-ntcf $\varepsilon : (\mathbf{a}, \mathbf{b}, \mathbf{g}, \mathbf{f}) >_{CF.coeq} E : \uparrow \uparrow_C \mapsto \mapsto_{C\alpha} \text{op-cat } \mathfrak{C}$

unfolding *is-cat-equalizer-def*

by

(
 rule cat-coequalizer-is-cat-coequalizer-2
 [
 OF is-cat-equalizer.is-cat-coequalizer-op[
 OF cat-equalizer-2-is-cat-equalizer
]
]
)

lemma (*in is-cat-equalizer-2*) *is-cat-coequalizer-2-op'*[*cat-op-intros*]:

assumes $\mathfrak{C}' = \text{op-cat } \mathfrak{C}$

shows *op-ntcf* $\varepsilon : (\mathbf{a}, \mathbf{b}, \mathbf{g}, \mathbf{f}) >_{CF.coeq} E : \uparrow \uparrow_C \mapsto \mapsto_{C\alpha} \mathfrak{C}'$

unfolding *assms by (rule is-cat-coequalizer-2-op)*

lemmas [*cat-op-intros*] = *is-cat-equalizer-2.is-cat-coequalizer-2-op'*

lemma (*in is-cat-coequalizer-2*) *is-cat-equalizer-2-op:*

op-ntcf $\varepsilon : E <_{CF.eq} (\mathbf{a}, \mathbf{b}, \mathbf{g}, \mathbf{f}) : \uparrow \uparrow_C \mapsto \mapsto_{C\alpha} \text{op-cat } \mathfrak{C}$

unfolding *is-cat-coequalizer-def*

by

(
 rule cat-equalizer-is-cat-equalizer-2
 [
 OF is-cat-coequalizer.is-cat-equalizer-op[
 OF cat-coequalizer-2-is-cat-coequalizer
]
]
)

qed

lemma *cat-cocone-cf-par-2-eps-NTMap-app*:

assumes ε :
 $\uparrow\uparrow\rightarrow\uparrow_{CF} \mathfrak{C} \mathfrak{b}_{PL2} \mathfrak{a}_{PL2} \mathfrak{f}_{PL} \mathfrak{g}_{PL} \mathfrak{b} \mathfrak{a} \mathfrak{f} \mathfrak{g} >_{CF.cocone} E$:
 $\uparrow\uparrow_C \mathfrak{b}_{PL2} \mathfrak{a}_{PL2} \mathfrak{f}_{PL} \mathfrak{g}_{PL} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{g} : \mathfrak{b} \mapsto_{\mathfrak{C}} \mathfrak{a}$
and $\mathfrak{f} : \mathfrak{b} \mapsto_{\mathfrak{C}} \mathfrak{a}$
shows
 $\varepsilon(\mathit{NTMap})(\mathfrak{b}_{PL2}) = \varepsilon(\mathit{NTMap})(\mathfrak{a}_{PL2}) \circ_{A\mathfrak{C}} \mathfrak{g}$
 $\varepsilon(\mathit{NTMap})(\mathfrak{b}_{PL2}) = \varepsilon(\mathit{NTMap})(\mathfrak{a}_{PL2}) \circ_{A\mathfrak{C}} \mathfrak{f}$

proof-

let $?II = \langle \uparrow\uparrow_C \mathfrak{b}_{PL2} \mathfrak{a}_{PL2} \mathfrak{f}_{PL} \mathfrak{g}_{PL} \rangle$
and $?II-II = \langle \uparrow\uparrow\rightarrow\uparrow_{CF} \mathfrak{C} \mathfrak{b}_{PL2} \mathfrak{a}_{PL2} \mathfrak{f}_{PL} \mathfrak{g}_{PL} \mathfrak{b} \mathfrak{a} \mathfrak{f} \mathfrak{g} \rangle$
and $?F = \langle \text{set } \{\mathfrak{g}_{PL}, \mathfrak{f}_{PL}\} \rangle$
have $\mathfrak{f}\mathfrak{g}\text{-}\mathfrak{g}\mathfrak{f} : \{\mathfrak{f}_{PL}, \mathfrak{g}_{PL}\} = \{\mathfrak{g}_{PL}, \mathfrak{f}_{PL}\}$ **by** *auto*
interpret $\varepsilon : \text{is-cat-cocone } \alpha \ E \ ?II \ \mathfrak{C} \ ?II-II \ \varepsilon$ **by** (*rule assms(1)*)
from $\varepsilon.\text{cat-PL2-f } \varepsilon.\text{cat-PL2-g}$ **have** $\mathfrak{g}\mathfrak{f} : ?F \in_0 \text{Vset } \alpha$
by (*intro Limit-vdoubleton-in-VsetI*) *auto*
from *assms(2,3)* **have**
 $(\wedge \mathfrak{f}'. \mathfrak{f}' \in_0 ?F \implies (\lambda \mathfrak{f} \in_0 ?F. (\mathfrak{f} = \mathfrak{g}_{PL} \ ? \ \mathfrak{g} : \mathfrak{f}))(\mathfrak{f}')) : \mathfrak{b} \mapsto_{\mathfrak{C}} \mathfrak{a}$
by *auto*

note *cat-cocone-cf-par-eps-NTMap-app = cat-cocone-cf-par-eps-NTMap-app*

[
OF assms(1)
 [
unfolded
the-cat-parallel-2-def
the-cf-parallel-2-def
a_{PL2}-def b_{PL2}-def
insert-commute,
unfolded fg-gf
],
folded a_{PL2}-def b_{PL2}-def,
OF - gf - this,
simplified
]

from

cat-cocone-cf-par-eps-NTMap-app[of g_{PL}, simplified]
cat-cocone-cf-par-eps-NTMap-app[of f_{PL}, simplified]
cat-PL2-gf

show

$\varepsilon(\mathit{NTMap})(\mathfrak{b}_{PL2}) = \varepsilon(\mathit{NTMap})(\mathfrak{a}_{PL2}) \circ_{A\mathfrak{C}} \mathfrak{g}$
 $\varepsilon(\mathit{NTMap})(\mathfrak{b}_{PL2}) = \varepsilon(\mathit{NTMap})(\mathfrak{a}_{PL2}) \circ_{A\mathfrak{C}} \mathfrak{f}$
by *fastforce+*

qed

lemma (**in** *is-cat-equalizer-2*) *cat-eq-2-eps-NTMap-app*:

$\varepsilon(\mathit{NTMap})(\mathfrak{b}_{PL2}) = \mathfrak{g} \circ_{A\mathfrak{C}} \varepsilon(\mathit{NTMap})(\mathfrak{a}_{PL2})$
 $\varepsilon(\mathit{NTMap})(\mathfrak{b}_{PL2}) = \mathfrak{f} \circ_{A\mathfrak{C}} \varepsilon(\mathit{NTMap})(\mathfrak{a}_{PL2})$

proof-

have $\mathfrak{g}_{PL} : \mathfrak{g}_{PL} \in_0 \text{set } \{\mathfrak{g}_{PL}, \mathfrak{f}_{PL}\}$ **and** $\mathfrak{f}_{PL} : \mathfrak{f}_{PL} \in_0 \text{set } \{\mathfrak{g}_{PL}, \mathfrak{f}_{PL}\}$ **by** *auto*

note *cat-eq-eps-NTMap-app = is-cat-equalizer.cat-eq-eps-NTMap-app*

[
OF cat-equalizer-2-is-cat-equalizer,
folded a_{PL2}-def b_{PL2}-def
]

from *cat-eq-eps-NTMap-app[OF g_{PL}]* *cat-eq-eps-NTMap-app[OF f_{PL}]* *cat-PL2-gf* **show**

$\varepsilon(\text{NTMap})(\mathbf{b}_{PL2}) = \mathbf{g} \circ_{A\mathfrak{C}} \varepsilon(\text{NTMap})(\mathbf{a}_{PL2})$
 $\varepsilon(\text{NTMap})(\mathbf{b}_{PL2}) = \mathbf{f} \circ_{A\mathfrak{C}} \varepsilon(\text{NTMap})(\mathbf{a}_{PL2})$
 by *auto*

qed

lemma (in *is-cat-coequalizer-2*) *cat-coeq-2-eps-NTMap-app*:

$\varepsilon(\text{NTMap})(\mathbf{b}_{PL2}) = \varepsilon(\text{NTMap})(\mathbf{a}_{PL2}) \circ_{A\mathfrak{C}} \mathbf{g}$
 $\varepsilon(\text{NTMap})(\mathbf{b}_{PL2}) = \varepsilon(\text{NTMap})(\mathbf{a}_{PL2}) \circ_{A\mathfrak{C}} \mathbf{f}$

proof–

have \mathbf{g}_{PL} : $\mathbf{g}_{PL} \in_0$ *set* $\{\mathbf{g}_{PL}, \mathbf{f}_{PL}\}$ **and** \mathbf{f}_{PL} : $\mathbf{f}_{PL} \in_0$ *set* $\{\mathbf{g}_{PL}, \mathbf{f}_{PL}\}$ **by** *auto*

note *cat-eq-eps-NTMap-app* = *is-cat-coequalizer.cat-coeq-eps-NTMap-app*

[
OF cat-coequalizer-2-is-cat-coequalizer,
folded a_{PL2}-def b_{PL2}-def
]

from *cat-eq-eps-NTMap-app*[*OF g_{PL}*] *cat-eq-eps-NTMap-app*[*OF f_{PL}*] *cat-PL2-gf* **show**

$\varepsilon(\text{NTMap})(\mathbf{b}_{PL2}) = \varepsilon(\text{NTMap})(\mathbf{a}_{PL2}) \circ_{A\mathfrak{C}} \mathbf{g}$
 $\varepsilon(\text{NTMap})(\mathbf{b}_{PL2}) = \varepsilon(\text{NTMap})(\mathbf{a}_{PL2}) \circ_{A\mathfrak{C}} \mathbf{f}$
 by *auto*

qed

lemma (in *is-cat-equalizer-2*) *cat-eq-2-Comp-eq*:

$\mathbf{g} \circ_{A\mathfrak{C}} \varepsilon(\text{NTMap})(\mathbf{a}_{PL2}) = \mathbf{f} \circ_{A\mathfrak{C}} \varepsilon(\text{NTMap})(\mathbf{a}_{PL2})$
 $\mathbf{f} \circ_{A\mathfrak{C}} \varepsilon(\text{NTMap})(\mathbf{a}_{PL2}) = \mathbf{g} \circ_{A\mathfrak{C}} \varepsilon(\text{NTMap})(\mathbf{a}_{PL2})$
unfolding *cat-eq-2-eps-NTMap-app[symmetric]* **by** *simp-all*

lemma (in *is-cat-coequalizer-2*) *cat-coeq-2-Comp-eq*:

$\varepsilon(\text{NTMap})(\mathbf{a}_{PL2}) \circ_{A\mathfrak{C}} \mathbf{g} = \varepsilon(\text{NTMap})(\mathbf{a}_{PL2}) \circ_{A\mathfrak{C}} \mathbf{f}$
 $\varepsilon(\text{NTMap})(\mathbf{a}_{PL2}) \circ_{A\mathfrak{C}} \mathbf{f} = \varepsilon(\text{NTMap})(\mathbf{a}_{PL2}) \circ_{A\mathfrak{C}} \mathbf{g}$
unfolding *cat-coeq-2-eps-NTMap-app[symmetric]* **by** *simp-all*

7.2.2 Universal property

lemma *is-cat-equalizer-2I'*:

assumes ε :

$E <_{CF.cone} \uparrow \rightarrow \uparrow \uparrow_{CF} \mathfrak{C} \mathbf{a}_{PL2} \mathbf{b}_{PL2} \mathbf{g}_{PL} \mathbf{f}_{PL} \mathbf{a} \mathbf{b} \mathbf{g} \mathbf{f} : \uparrow \uparrow_C \mathbf{a}_{PL2} \mathbf{b}_{PL2} \mathbf{g}_{PL} \mathbf{f}_{PL} \mapsto \rightarrow_{C\alpha} \mathfrak{C}$
and $\mathbf{g} : \mathbf{a} \mapsto_{\mathfrak{C}} \mathbf{b}$
and $\mathbf{f} : \mathbf{a} \mapsto_{\mathfrak{C}} \mathbf{b}$
and $\wedge \varepsilon' E'. \varepsilon' :$

$E' <_{CF.cone} \uparrow \rightarrow \uparrow \uparrow_{CF} \mathfrak{C} \mathbf{a}_{PL2} \mathbf{b}_{PL2} \mathbf{g}_{PL} \mathbf{f}_{PL} \mathbf{a} \mathbf{b} \mathbf{g} \mathbf{f} :$
 $\uparrow \uparrow_C \mathbf{a}_{PL2} \mathbf{b}_{PL2} \mathbf{g}_{PL} \mathbf{f}_{PL} \mapsto \rightarrow_{C\alpha} \mathfrak{C} \implies$
 $\exists ! f'. f' : E' \mapsto_{\mathfrak{C}} E \wedge \varepsilon'(\text{NTMap})(\mathbf{a}_{PL2}) = \varepsilon(\text{NTMap})(\mathbf{a}_{PL2}) \circ_{A\mathfrak{C}} f'$

shows $\varepsilon : E <_{CF.eq} (\mathbf{a}, \mathbf{b}, \mathbf{g}, \mathbf{f}) : \uparrow \uparrow_C \mapsto \rightarrow_{C\alpha} \mathfrak{C}$

proof–

let $?II = \langle \uparrow \uparrow_C \mathbf{a}_{PL2} \mathbf{b}_{PL2} \mathbf{g}_{PL} \mathbf{f}_{PL} \rangle$
and $?II-II = \langle \uparrow \rightarrow \uparrow \uparrow_{CF} \mathfrak{C} \mathbf{a}_{PL2} \mathbf{b}_{PL2} \mathbf{g}_{PL} \mathbf{f}_{PL} \mathbf{a} \mathbf{b} \mathbf{g} \mathbf{f} \rangle$
and $?F = \langle \text{set } \{\mathbf{g}_{PL}, \mathbf{f}_{PL}\} \rangle$

interpret ε : *is-cat-cone* α E $?II$ \mathfrak{C} $?II-II$ ε **by** (*rule assms(1)*)

from ε .*cat-PL2-f* ε .*cat-PL2-g* **have** \mathbf{gf} : $?F \in_0$ *Vset* α

by (*intro Limit-vdoubleton-in-VsetI*) *auto*

from *assms(2,3)* **have** $(\lambda f \in_0 ?F. (f = \mathbf{f}_{PL} ? f : \mathbf{g}))(\mathbf{f}') : \mathbf{a} \mapsto_{\mathfrak{C}} \mathbf{b}$

if $\mathbf{f}' \in_0 ?F$ **for** \mathbf{f}'

using *that* **by** *simp*

note *is-cat-equalizerI'* = *is-cat-equalizerI'*

[
OF
assms(1)[
unfolded
]

$\text{the-cat-parallel-2-def the-cf-parallel-2-def } \mathbf{a}_{PL2\text{-def}} \mathbf{b}_{PL2\text{-def}}$
 $\text{folded } \mathbf{a}_{PL2\text{-def}} \mathbf{b}_{PL2\text{-def}}$,
 OF
 $-$
 \mathbf{gf}
 $-$
 this
 $-$
 $\text{assms}(4)[\text{unfolded the-cf-parallel-2-def the-cat-parallel-2-def}]$,
 $\text{of } \mathbf{g}_{PL}$,
 simplified
 $]$
show $?thesis$ **by** $(\text{rule cat-equalizer-is-cat-equalizer-2}[OF \text{ is-cat-equalizerI}'])$
qed

lemma $\text{is-cat-coequalizer-2I}'$:

assumes ε :

$\uparrow\uparrow \rightarrow \uparrow\uparrow_{CF} \mathcal{C} \mathbf{b}_{PL2} \mathbf{a}_{PL2} \mathbf{f}_{PL} \mathbf{g}_{PL} \mathbf{b} \mathbf{a} \mathbf{f} \mathbf{g} >_{CF.cocone} E$:

$\uparrow\uparrow_C \mathbf{b}_{PL2} \mathbf{a}_{PL2} \mathbf{f}_{PL} \mathbf{g}_{PL} \mapsto \mapsto_{C\alpha} \mathcal{C}$

and $\mathbf{g} : \mathbf{b} \mapsto_{\mathcal{C}} \mathbf{a}$

and $\mathbf{f} : \mathbf{b} \mapsto_{\mathcal{C}} \mathbf{a}$

and $\wedge \varepsilon' E'. \varepsilon' :$

$\uparrow\uparrow \rightarrow \uparrow\uparrow_{CF} \mathcal{C} \mathbf{b}_{PL2} \mathbf{a}_{PL2} \mathbf{f}_{PL} \mathbf{g}_{PL} \mathbf{b} \mathbf{a} \mathbf{f} \mathbf{g} >_{CF.cocone} E'$:

$\uparrow\uparrow_C \mathbf{b}_{PL2} \mathbf{a}_{PL2} \mathbf{f}_{PL} \mathbf{g}_{PL} \mapsto \mapsto_{C\alpha} \mathcal{C} \implies$

$\exists ! f'. f' : E \mapsto_{\mathcal{C}} E' \wedge \varepsilon'(\text{NTMap})(\mathbf{a}_{PL2}) = f' \circ_A \varepsilon(\text{NTMap})(\mathbf{a}_{PL2})$

shows $\varepsilon : (\mathbf{a}, \mathbf{b}, \mathbf{g}, \mathbf{f}) >_{CF.coeq} E : \uparrow\uparrow_C \mapsto \mapsto_{C\alpha} \mathcal{C}$

proof-

let $?II = \langle \uparrow\uparrow_C \mathbf{b}_{PL2} \mathbf{a}_{PL2} \mathbf{f}_{PL} \mathbf{g}_{PL} \rangle$

and $?II-II = \langle \uparrow\uparrow \rightarrow \uparrow\uparrow_{CF} \mathcal{C} \mathbf{b}_{PL2} \mathbf{a}_{PL2} \mathbf{f}_{PL} \mathbf{g}_{PL} \mathbf{b} \mathbf{a} \mathbf{f} \mathbf{g} \rangle$

and $?F = \langle \text{set } \{\mathbf{g}_{PL}, \mathbf{f}_{PL}\} \rangle$

have $\mathbf{fg-gf} : \{\mathbf{f}_{PL}, \mathbf{g}_{PL}\} = \{\mathbf{g}_{PL}, \mathbf{f}_{PL}\}$ **by** auto

interpret $\varepsilon : \text{is-cat-cocone } \alpha E ?II \mathcal{C} ?II-II \varepsilon$ **by** $(\text{rule assms}(1))$

from $\varepsilon.\text{cat-PL2-f } \varepsilon.\text{cat-PL2-g}$ **have** $\mathbf{gf} : ?F \in_o Vset \alpha$

by $(\text{intro Limit-vdoubleton-in-VsetI}) \text{ auto}$

from $\text{assms}(2,3)$ **have** $(\lambda f \in_o \text{set } \{\mathbf{g}_{PL}, \mathbf{f}_{PL}\}. (f = \mathbf{g}_{PL} ? \mathbf{g} : \mathbf{f}))(\mathbf{f}')$: $\mathbf{b} \mapsto_{\mathcal{C}} \mathbf{a}$

if $\mathbf{f}' \in_o \text{set } \{\mathbf{g}_{PL}, \mathbf{f}_{PL}\}$ **for** \mathbf{f}'

using that **by** simp

note $\text{is-cat-coequalizerI}'$

[

$OF \text{ assms}(1)[$

unfolded

$\text{the-cat-parallel-2-def the-cf-parallel-2-def } \mathbf{a}_{PL2\text{-def}} \mathbf{b}_{PL2\text{-def}} \mathbf{fg-gf}$

$]$,

$\text{folded } \mathbf{a}_{PL2\text{-def}} \mathbf{b}_{PL2\text{-def}}$,

OF

$-$

\mathbf{gf}

$-$

this

$-$

$\text{assms}(4)[\text{unfolded the-cf-parallel-2-def the-cat-parallel-2-def } \mathbf{fg-gf}]$,

$\text{of } \mathbf{g}_{PL}$,

simplified

$]$

with cat-PL2-gf **have**

$\varepsilon : (\mathbf{a}, \mathbf{b}, ?F, (\lambda f \in_o ?F. (f = \mathbf{f}_{PL} ? \mathbf{f} : \mathbf{g}))) >_{CF.coeq} E : \uparrow\uparrow_C \mapsto \mapsto_{C\alpha} \mathcal{C}$

by $(\text{auto simp: VLambda-vdoubleton})$

from *cat-coequalizer-is-cat-coequalizer-2*[*OF this*] show *?thesis* by *simp*
qed

lemma (in *is-cat-equalizer-2*) *cat-eq-2-unique-cone*:

assumes ε' :

$E' <_{CF.cone} \uparrow \rightarrow \uparrow_{CF} \mathfrak{C} \mathfrak{a}_{PL2} \mathfrak{b}_{PL2} \mathfrak{g}_{PL} \mathfrak{f}_{PL} \mathfrak{a} \mathfrak{b} \mathfrak{g} \mathfrak{f}$:

$\uparrow \uparrow_C \mathfrak{a}_{PL2} \mathfrak{b}_{PL2} \mathfrak{g}_{PL} \mathfrak{f}_{PL} \mapsto \mapsto_{C\alpha} \mathfrak{C}$

shows $\exists ! f'. f' : E' \mapsto_{\mathfrak{C}} E \wedge \varepsilon'(\langle NTMap \rangle)(\mathfrak{a}_{PL2}) = \varepsilon(\langle NTMap \rangle)(\mathfrak{a}_{PL2}) \circ_A \mathfrak{C} f'$

by

(
rule is-cat-equalizer.cat-eq-unique-cone
 [
OF cat-equalizer-2-is-cat-equalizer,
folded \mathfrak{a}_{PL2} -def \mathfrak{b}_{PL2} -def,
OF assms[unfolded the-cf-parallel-2-def the-cat-parallel-2-def]
]
)

lemma (in *is-cat-equalizer-2*) *cat-eq-2-unique*:

assumes $\varepsilon' : E' <_{CF.eq} (\mathfrak{a}, \mathfrak{b}, \mathfrak{g}, \mathfrak{f}) : \uparrow \uparrow_C \mapsto \mapsto_{C\alpha} \mathfrak{C}$

shows

$\exists ! f'. f' : E' \mapsto_{\mathfrak{C}} E \wedge \varepsilon' = \varepsilon \cdot_{NTCF} ntcf-const (\uparrow \uparrow_C \mathfrak{a}_{PL2} \mathfrak{b}_{PL2} \mathfrak{g}_{PL} \mathfrak{f}_{PL}) \mathfrak{C} f'$

proof-

interpret ε' : *is-cat-equalizer-2* $\alpha \mathfrak{a} \mathfrak{b} \mathfrak{g} \mathfrak{f} \mathfrak{C} E' \varepsilon'$ by (*rule assms*)

show *?thesis*

by

(
rule is-cat-equalizer.cat-eq-unique
 [
OF cat-equalizer-2-is-cat-equalizer,
folded \mathfrak{a}_{PL2} -def \mathfrak{b}_{PL2} -def,
OF ε' .cat-equalizer-2-is-cat-equalizer,
folded the-cat-parallel-2-def
]
)

qed

lemma (in *is-cat-equalizer-2*) *cat-eq-2-unique'*:

assumes $\varepsilon' : E' <_{CF.eq} (\mathfrak{a}, \mathfrak{b}, \mathfrak{g}, \mathfrak{f}) : \uparrow \uparrow_C \mapsto \mapsto_{C\alpha} \mathfrak{C}$

shows $\exists ! f'. f' : E' \mapsto_{\mathfrak{C}} E \wedge \varepsilon'(\langle NTMap \rangle)(\mathfrak{a}_{PL2}) = \varepsilon(\langle NTMap \rangle)(\mathfrak{a}_{PL2}) \circ_A \mathfrak{C} f'$

proof-

interpret ε' : *is-cat-equalizer-2* $\alpha \mathfrak{a} \mathfrak{b} \mathfrak{g} \mathfrak{f} \mathfrak{C} E' \varepsilon'$ by (*rule assms*)

show *?thesis*

by

(
rule is-cat-equalizer.cat-eq-unique'
 [
OF cat-equalizer-2-is-cat-equalizer,
folded \mathfrak{a}_{PL2} -def \mathfrak{b}_{PL2} -def,
OF ε' .cat-equalizer-2-is-cat-equalizer,
folded the-cat-parallel-2-def
]
)

qed

lemma (in *is-cat-coequalizer-2*) *cat-coeq-2-unique-cocone*:

assumes ε' :

$\uparrow \uparrow \rightarrow \uparrow_{CF} \mathfrak{C} \mathfrak{b}_{PL2} \mathfrak{a}_{PL2} \mathfrak{f}_{PL} \mathfrak{g}_{PL} \mathfrak{b} \mathfrak{a} \mathfrak{f} \mathfrak{g} >_{CF.cocone} E'$:

$\uparrow\uparrow_C \mathbf{b}_{PL2} \mathbf{a}_{PL2} \mathbf{f}_{PL} \mathbf{g}_{PL} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\exists!f'. f' : E \mapsto_{\mathfrak{C}} E' \wedge \varepsilon'(\mathcal{N}TMap)(\mathbf{a}_{PL2}) = f' \circ_{A\mathfrak{C}} \varepsilon(\mathcal{N}TMap)(\mathbf{a}_{PL2})$
by
 (

- rule *is-cat-coequalizer.cat-coeq-unique-cocone*
- [
 - OF *cat-coequalizer-2-is-cat-coequalizer*,
 - folded *a_{PL2}-def b_{PL2}-def insert-commute*,
 - OF *assms*[
 - unfolded
 - the-cf-parallel-2-def the-cat-parallel-2-def cat-eq-F'-helper*

]
)

lemma (in *is-cat-coequalizer-2*) *cat-coeq-2-unique*:
assumes $\varepsilon' : (\mathbf{a}, \mathbf{b}, \mathbf{g}, \mathbf{f}) >_{CF.coeq} E' : \uparrow\uparrow_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\exists!f'$.
 $f' : E \mapsto_{\mathfrak{C}} E' \wedge$
 $\varepsilon' = ntcf-const (\uparrow\uparrow_C \mathbf{b}_{PL2} \mathbf{a}_{PL2} \mathbf{f}_{PL} \mathbf{g}_{PL}) \mathfrak{C} f' \cdot_{NTCF} \varepsilon$

proof-

interpret ε' : *is-cat-coequalizer-2* α \mathbf{a} \mathbf{b} \mathbf{g} \mathbf{f} \mathfrak{C} E' ε' **by** (*rule assms*)
show *?thesis*
by
 (

- rule *is-cat-coequalizer.cat-coeq-unique*
- [
 - OF *cat-coequalizer-2-is-cat-coequalizer*,
 - folded *a_{PL2}-def b_{PL2}-def*,
 - OF ε' .*cat-coequalizer-2-is-cat-coequalizer*,
 - folded *the-cat-parallel-2-def the-cat-parallel-2-commute*

]
)

qed

lemma (in *is-cat-coequalizer-2*) *cat-coeq-2-unique'*:
assumes $\varepsilon' : (\mathbf{a}, \mathbf{b}, \mathbf{g}, \mathbf{f}) >_{CF.coeq} E' : \uparrow\uparrow_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\exists!f'. f' : E \mapsto_{\mathfrak{C}} E' \wedge \varepsilon'(\mathcal{N}TMap)(\mathbf{a}_{PL2}) = f' \circ_{A\mathfrak{C}} \varepsilon(\mathcal{N}TMap)(\mathbf{a}_{PL2})$

proof-

interpret ε' : *is-cat-coequalizer-2* α \mathbf{a} \mathbf{b} \mathbf{g} \mathbf{f} \mathfrak{C} E' ε' **by** (*rule assms*)
show *?thesis*
by
 (

- rule *is-cat-coequalizer.cat-coeq-unique'*
- [
 - OF *cat-coequalizer-2-is-cat-coequalizer*,
 - folded *a_{PL2}-def b_{PL2}-def*,
 - OF ε' .*cat-coequalizer-2-is-cat-coequalizer*,
 - folded *the-cat-parallel-2-def*

]
)

qed

lemma *cat-equalizer-2-ex-is-iso-arr*:
assumes $\varepsilon : E <_{CF.eq} (\mathbf{a}, \mathbf{b}, \mathbf{g}, \mathbf{f}) : \uparrow\uparrow_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\varepsilon' : E' <_{CF.eq} (\mathbf{a}, \mathbf{b}, \mathbf{g}, \mathbf{f}) : \uparrow\uparrow_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$
obtains f **where** $f : E' \mapsto_{iso\mathfrak{C}} E$
and $\varepsilon' = \varepsilon \cdot_{NTCF} ntcf-const (\uparrow\uparrow_C \mathbf{a}_{PL2} \mathbf{b}_{PL2} \mathbf{g}_{PL} \mathbf{f}_{PL}) \mathfrak{C} f$

proof-

interpret ε : *is-cat-equalizer-2* α **a b g f** \mathfrak{C} E ε **by** (*rule assms(1)*)
interpret ε' : *is-cat-equalizer-2* α **a b g f** \mathfrak{C} E' ε' **by** (*rule assms(2)*)

show *?thesis*

using *that*

by

(
rule cat-equalizer-ex-is-iso-arr
 [
OF
 ε .cat-equalizer-2-is-cat-equalizer
 ε' .cat-equalizer-2-is-cat-equalizer,
folded \mathbf{a}_{PL2} -def \mathbf{b}_{PL2} -def the-cat-parallel-2-def
]
)

qed

lemma *cat-equalizer-2-ex-is-iso-arr'*:

assumes $\varepsilon : E <_{CF.eq} (\mathbf{a}, \mathbf{b}, \mathbf{g}, \mathbf{f}) : \uparrow\uparrow_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$

and $\varepsilon' : E' <_{CF.eq} (\mathbf{a}, \mathbf{b}, \mathbf{g}, \mathbf{f}) : \uparrow\uparrow_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$

obtains f **where** $f : E' \mapsto_{iso\mathfrak{C}} E$

and $\varepsilon'(\mathit{NTMap})(\mathbf{a}_{PL2}) = \varepsilon(\mathit{NTMap})(\mathbf{a}_{PL2}) \circ_{A\mathfrak{C}} f$

and $\varepsilon'(\mathit{NTMap})(\mathbf{b}_{PL2}) = \varepsilon(\mathit{NTMap})(\mathbf{b}_{PL2}) \circ_{A\mathfrak{C}} f$

proof-

interpret ε : *is-cat-equalizer-2* α **a b g f** \mathfrak{C} E ε **by** (*rule assms(1)*)

interpret ε' : *is-cat-equalizer-2* α **a b g f** \mathfrak{C} E' ε' **by** (*rule assms(2)*)

show *?thesis*

using *that*

by

(
rule cat-equalizer-ex-is-iso-arr'
 [
OF
 ε .cat-equalizer-2-is-cat-equalizer
 ε' .cat-equalizer-2-is-cat-equalizer,
folded \mathbf{a}_{PL2} -def \mathbf{b}_{PL2} -def the-cat-parallel-2-def
]
)

qed

lemma *cat-coequalizer-2-ex-is-iso-arr*:

assumes $\varepsilon : (\mathbf{a}, \mathbf{b}, \mathbf{g}, \mathbf{f}) >_{CF.coeq} E : \uparrow\uparrow_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$

and $\varepsilon' : (\mathbf{a}, \mathbf{b}, \mathbf{g}, \mathbf{f}) >_{CF.coeq} E' : \uparrow\uparrow_C \mapsto\mapsto_{C\alpha} \mathfrak{C}$

obtains f **where** $f : E \mapsto_{iso\mathfrak{C}} E'$

and $\varepsilon' = \mathit{ntcf-const} (\uparrow\uparrow_C \mathbf{b}_{PL2} \mathbf{a}_{PL2} \mathbf{f}_{PL} \mathbf{g}_{PL}) \mathfrak{C} f \cdot \mathit{NTCF} \varepsilon$

proof-

interpret ε : *is-cat-coequalizer-2* α **a b g f** \mathfrak{C} E ε **by** (*rule assms(1)*)

interpret ε' : *is-cat-coequalizer-2* α **a b g f** \mathfrak{C} E' ε' **by** (*rule assms(2)*)

show *?thesis*

using *that*

by

(
rule cat-coequalizer-ex-is-iso-arr
 [
OF
 ε .cat-coequalizer-2-is-cat-coequalizer
 ε' .cat-coequalizer-2-is-cat-coequalizer,
folded
]
)

$\mathbf{a}_{PL2}\text{-def } \mathbf{b}_{PL2}\text{-def the-cat-parallel-2-def the-cat-parallel-2-commute}$
]
)
qed

lemma *cat-coequalizer-2-ex-is-iso-arr'*:

assumes $\varepsilon : (\mathbf{a}, \mathbf{b}, \mathbf{g}, \mathbf{f}) >_{CF.coeq} E : \uparrow\uparrow_C \mapsto \mapsto_{C\alpha} \mathfrak{C}$
and $\varepsilon' : (\mathbf{a}, \mathbf{b}, \mathbf{g}, \mathbf{f}) >_{CF.coeq} E' : \uparrow\uparrow_C \mapsto \mapsto_{C\alpha} \mathfrak{C}$
obtains f **where** $f : E \mapsto_{iso} \mathfrak{C} E'$
and $\varepsilon'(\mathit{NTMap})(\mathbf{a}_{PL2}) = f \circ_A \mathfrak{C} \varepsilon(\mathit{NTMap})(\mathbf{a}_{PL2})$
and $\varepsilon'(\mathit{NTMap})(\mathbf{b}_{PL2}) = f \circ_A \mathfrak{C} \varepsilon(\mathit{NTMap})(\mathbf{b}_{PL2})$

proof-

interpret ε : *is-cat-coequalizer-2* α \mathbf{a} \mathbf{b} \mathbf{g} \mathbf{f} \mathfrak{C} E ε **by** (*rule assms(1)*)
interpret ε' : *is-cat-coequalizer-2* α \mathbf{a} \mathbf{b} \mathbf{g} \mathbf{f} \mathfrak{C} E' ε' **by** (*rule assms(2)*)
show *?thesis*

using *that*

by

(
 rule cat-coequalizer-ex-is-iso-arr'
 [
 OF
 ε.cat-coequalizer-2-is-cat-coequalizer
 ε'.cat-coequalizer-2-is-cat-coequalizer,
 folded
 α_{PL2}-def b_{PL2}-def the-cat-parallel-2-def the-cat-parallel-2-commute
]
)

qed

7.2.3 Further properties

lemma (*in is-cat-equalizer-2*) *cat-eq-2-is-monic-arr*:

$\varepsilon(\mathit{NTMap})(\mathbf{a}_{PL2}) : E \mapsto_{mon} \mathfrak{C} \mathbf{a}$

by

(
 rule is-cat-equalizer.cat-eq-is-monic-arr[
 OF cat-equalizer-2-is-cat-equalizer, folded α_{PL2}-def
]
)

lemma (*in is-cat-coequalizer-2*) *cat-coeq-2-is-epic-arr*:

$\varepsilon(\mathit{NTMap})(\mathbf{a}_{PL2}) : \mathbf{a} \mapsto_{epi} \mathfrak{C} E$

by

(
 rule is-cat-coequalizer.cat-coeq-is-epic-arr[
 OF cat-coequalizer-2-is-cat-coequalizer, folded α_{PL2}-def
]
)

7.3 Equalizer cone

7.3.1 Definition and elementary properties

definition *ntcf-equalizer-base* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow (V \Rightarrow V) \Rightarrow V$

where *ntcf-equalizer-base* $\mathfrak{C} \mathbf{a} \mathbf{b} \mathbf{g} \mathbf{f} E e =$

[
 $(\lambda x \in \circ \uparrow\uparrow_C \mathbf{a}_{PL2} \mathbf{b}_{PL2} \mathbf{g}_{PL} \mathbf{f}_{PL}(\mathit{Obj}). e x),$
 cf-const $(\uparrow\uparrow_C \mathbf{a}_{PL2} \mathbf{b}_{PL2} \mathbf{g}_{PL} \mathbf{f}_{PL}) \mathfrak{C} E,$
 $\uparrow\uparrow \rightarrow \uparrow\uparrow_{CF} \mathfrak{C} \mathbf{a}_{PL2} \mathbf{b}_{PL2} \mathbf{g}_{PL} \mathbf{f}_{PL} \mathbf{a} \mathbf{b} \mathbf{g} \mathbf{f},$
]

```

  ↑↑C aPL2 bPL2 gPL fPL,
  C
]₀

```

Components.

lemma *ntcf-equalizer-base-components*:

```

shows ntcf-equalizer-base C a b g f E e(NTMap) =
  (λxε₀.↑↑C aPL2 bPL2 gPL fPL(Obj). e x)
and [cat-lim-cs-simps]: ntcf-equalizer-base C a b g f E e(NTDom) =
  cf-const (↑↑C aPL2 bPL2 gPL fPL) C E
and [cat-lim-cs-simps]: ntcf-equalizer-base C a b g f E e(NTCod) =
  ↑↑→↑↑CF C aPL2 bPL2 gPL fPL a b g f
and [cat-lim-cs-simps]:
  ntcf-equalizer-base C a b g f E e(NTDGDom) = ↑↑C aPL2 bPL2 gPL fPL
and [cat-lim-cs-simps]:
  ntcf-equalizer-base C a b g f E e(NTDGCod) = C
unfolding ntcf-equalizer-base-def nt-field-simps
by (simp-all add: nat-omega-simps)

```

7.3.2 Natural transformation map

mk-VLambda *ntcf-equalizer-base-components(1)*

```

|vsu ntcf-equalizer-base-NTMap-vsuv[cat-lim-cs-intros]|
|vdomain ntcf-equalizer-base-NTMap-vdomain[cat-lim-cs-simps]|
|app ntcf-equalizer-base-NTMap-app[cat-lim-cs-simps]|

```

7.3.3 Equalizer cone is a cone

lemma (in *category*) *cat-ntcf-equalizer-base-is-cat-cone*:

```

assumes e aPL2 : E →C a
and e bPL2 : E →C b
and e bPL2 = g ∘AC e aPL2
and e bPL2 = f ∘AC e aPL2
and g : a →C b
and f : a →C b
shows ntcf-equalizer-base C a b g f E e :
  E <CF.cone ↑↑→↑↑CF C aPL2 bPL2 gPL fPL a b g f :
  ↑↑C aPL2 bPL2 gPL fPL →→Cα C

```

proof–

```

interpret par: cf-parallel-2 α aPL2 bPL2 gPL fPL a b g f C
by (intro cf-parallel-2I cat-parallel-2I assms(5,6))
  (simp-all add: cat-parallel-cs-intros cat-cs-intros)

```

show ?thesis

proof(intro *is-cat-coneI is-tm-ntcfI' is-ntcfI'*)

```

show vsequence (ntcf-equalizer-base C a b g f E e)
unfolding ntcf-equalizer-base-def by auto
show vcard (ntcf-equalizer-base C a b g f E e) = 5N
unfolding ntcf-equalizer-base-def by (simp add: nat-omega-simps)

```

from *assms(2)* **show**

```

  cf-const (↑↑C aPL2 bPL2 gPL fPL) C E : ↑↑C aPL2 bPL2 gPL fPL →→Cα C
by
  (
    cs-concl
    cs-simp: cat-cs-simps
    cs-intro: cat-small-cs-intros cat-parallel-cs-intros cat-cs-intros
  )

```

from *assms* **show**

```

  ↑↑→↑↑CF C aPL2 bPL2 gPL fPL a b g f : ↑↑C aPL2 bPL2 gPL fPL →→Cα C

```

```

  by (cs-concl cs-intro: cat-parallel-cs-intros cat-small-cs-intros)
show
  ntcf-equalizer-base  $\mathfrak{C} \mathfrak{a} \mathfrak{b} \mathfrak{g} \mathfrak{f} E e(\text{NTMap})(i) :$ 
    cf-const ( $\uparrow\uparrow_C \mathfrak{a}_{PL2} \mathfrak{b}_{PL2} \mathfrak{g}_{PL} \mathfrak{f}_{PL}$ )  $\mathfrak{C} E(\text{ObjMap})(i) \mapsto_{\mathfrak{C}}$ 
       $\uparrow\uparrow\rightarrow\uparrow\uparrow_{CF} \mathfrak{C} \mathfrak{a}_{PL2} \mathfrak{b}_{PL2} \mathfrak{g}_{PL} \mathfrak{f}_{PL} \mathfrak{a} \mathfrak{b} \mathfrak{g} \mathfrak{f}(\text{ObjMap})(i)$ 
  if  $i \in_{\circ} \uparrow\uparrow_C \mathfrak{a}_{PL2} \mathfrak{b}_{PL2} \mathfrak{g}_{PL} \mathfrak{f}_{PL}(\text{Obj})$  for  $i$ 
proof-
  from that assms(1,2,5,6) show ?thesis
  by (elim the-cat-parallel-2-ObjE; simp only:)
    (
      cs-concl
      cs-simp: cat-lim-cs-simps cat-cs-simps cat-parallel-cs-simps
      cs-intro: cat-cs-intros cat-parallel-cs-intros
    )
qed
show
  ntcf-equalizer-base  $\mathfrak{C} \mathfrak{a} \mathfrak{b} \mathfrak{g} \mathfrak{f} E e(\text{NTMap})(b') \circ_{A\mathfrak{C}}$ 
    cf-const ( $\uparrow\uparrow_C \mathfrak{a}_{PL2} \mathfrak{b}_{PL2} \mathfrak{g}_{PL} \mathfrak{f}_{PL}$ )  $\mathfrak{C} E(\text{ArrMap})(f') =$ 
       $\uparrow\uparrow\rightarrow\uparrow\uparrow_{CF} \mathfrak{C} \mathfrak{a}_{PL2} \mathfrak{b}_{PL2} \mathfrak{g}_{PL} \mathfrak{f}_{PL} \mathfrak{a} \mathfrak{b} \mathfrak{g} \mathfrak{f}(\text{ArrMap})(f') \circ_{A\mathfrak{C}}$ 
      ntcf-equalizer-base  $\mathfrak{C} \mathfrak{a} \mathfrak{b} \mathfrak{g} \mathfrak{f} E e(\text{NTMap})(a')$ 
  if  $f' : a' \mapsto \uparrow\uparrow_C \mathfrak{a}_{PL2} \mathfrak{b}_{PL2} \mathfrak{g}_{PL} \mathfrak{f}_{PL} b'$  for  $a' b' f'$ 
  using that assms(1,2,5,6)
  by (elim par.the-cat-parallel-2-is-arrE; simp only:)
    (
      cs-concl
      cs-simp:
        cat-cs-simps
        cat-lim-cs-simps
        cat-parallel-cs-simps
        assms(3,4)[symmetric]
      cs-intro: cat-parallel-cs-intros
    )+
qed
  (
    use assms(2) in
    <
      cs-concl
      cs-intro: cat-lim-cs-intros cat-cs-intros
      cs-simp: cat-lim-cs-simps
    >
  )+
qed

```

8 Pointed arrows and natural transformations

8.1 Pointed arrow

The terminology that is used in this section deviates from convention: a pointed arrow is merely an arrow in *Set* from a singleton set to another set.

8.1.1 Definition and elementary properties

See Chapter III-2 in [9].

definition $ntcf-paa :: V \Rightarrow V \Rightarrow V \Rightarrow V$
where $ntcf-paa \ a \ B \ b = [(\lambda a \in_o \text{set } \{a\}. b), \text{set } \{a\}, B]_o$

Components.

lemma $ntcf-paa-components$:
shows $ntcf-paa \ a \ B \ b(\text{ArrVal}) = (\lambda a \in_o \text{set } \{a\}. b)$
and $[cat-cs-simps]: ntcf-paa \ a \ B \ b(\text{ArrDom}) = \text{set } \{a\}$
and $[cat-cs-simps]: ntcf-paa \ a \ B \ b(\text{ArrCod}) = B$
unfolding $ntcf-paa-def \ arr-field-simps$ **by** $(simp-all \ add: \text{nat-omega-simps})$

8.1.2 Arrow value

mk-VLambda $ntcf-paa-components(1)$
 $[vsu \ ntcf-paa-ArrVal-vsv[cat-cs-intros]]$
 $[vdomain \ ntcf-paa-ArrVal-vdomain[cat-cs-simps]]$
 $[app \ ntcf-paa-ArrVal-app[unfolded \ vsingleton-iff, \ cat-cs-simps]]$

8.1.3 Pointed arrow is an arrow in *Set*

lemma $(in \ Z) \ ntcf-paa-is-arr$:
assumes $a \in_o \text{cat-Set } \alpha(\text{Obj})$ **and** $A \in_o \text{cat-Set } \alpha(\text{Obj})$ **and** $a \in_o A$
shows $ntcf-paa \ a \ A \ a : \text{set } \{a\} \mapsto_{\text{cat-Set } \alpha} A$
proof $(intro \ \text{cat-Set-is-arrI} \ \text{arr-SetI} \ \text{cat-cs-intros}, \ \text{unfold} \ \text{cat-cs-simps})$
show $vfsequence \ (ntcf-paa \ a \ A \ a)$ **unfolding** $ntcf-paa-def$ **by** $simp$
show $vcard \ (ntcf-paa \ a \ A \ a) = 3_{\mathbb{N}}$
unfolding $ntcf-paa-def$ **by** $(simp \ add: \ \text{nat-omega-simps})$
show $\mathcal{R}_o \ (ntcf-paa \ a \ A \ a(\text{ArrVal})) \subseteq_o A$
unfolding $ntcf-paa-components$ **by** $(intro \ \text{vrange-VLambda-vsubset} \ \text{assms})$
qed $(use \ \text{assms} \ \text{in} \ \langle \text{auto} \ \text{simp}: \ \text{cat-Set-components}(1) \ \text{Limit-vsingleton-in-VsetI} \rangle)$

lemma $(in \ Z) \ ntcf-paa-is-arr'[cat-cs-intros]$:
assumes $a \in_o \text{cat-Set } \alpha(\text{Obj})$
and $A \in_o \text{cat-Set } \alpha(\text{Obj})$
and $a \in_o A$
and $A' = \text{set } \{a\}$
and $B' = A$
and $\mathcal{C}' = \text{cat-Set } \alpha$
shows $ntcf-paa \ a \ A \ a : A' \mapsto_{\mathcal{C}'} B'$
using $\text{assms}(1-3)$ **unfolding** $\text{assms}(4-6)$ **by** $(rule \ \text{ntcf-paa-is-arr})$

lemmas $[cat-cs-intros] = Z.ntcf-paa-is-arr'$

8.1.4 Further properties

lemma $ntcf-paa-injective[cat-cs-simps]$:
 $ntcf-paa \ a \ A \ b = ntcf-paa \ a \ A \ c \longleftrightarrow b = c$
proof

assume $ntcf\text{-}paa\ a\ A\ b = ntcf\text{-}paa\ a\ A\ c$
then have $ntcf\text{-}paa\ a\ A\ b(\downarrow ArrVal)(\downarrow a) = ntcf\text{-}paa\ a\ A\ c(\downarrow ArrVal)(\downarrow a)$ **by** $simp$
then show $b = c$ **by** ($cs\text{-}prems$ **cs-simp**: $cat\text{-}cs\text{-}simps$)
qed $simp$

lemma (in \mathcal{Z}) $ntcf\text{-}paa\text{-}ArrVal$:

assumes $F : set\ \{a\} \mapsto_{cat\text{-}Set\ \alpha}\ X$
shows $ntcf\text{-}paa\ a\ X\ (F(\downarrow ArrVal)(\downarrow a)) = F$

proof–

interpret $F : arr\text{-}Set\ \alpha\ F$

rewrites [$cat\text{-}cs\text{-}simps$]: $F(\downarrow ArrDom) = set\ \{a\}$

and [$cat\text{-}cs\text{-}simps$]: $F(\downarrow ArrCod) = X$

by ($auto\ simp$: $cat\text{-}Set\text{-}is\text{-}arrD[OF\ assms]$)

from $F.arr\text{-}Par\text{-}ArrDom\text{-}in\text{-}Vset$ **have** $a : a \in_o\ Vset\ \alpha$ **by** $auto$

from $assms\ a\ F.arr\text{-}Par\text{-}ArrCod\text{-}in\text{-}Vset$ **have** $lhs\text{-}is\text{-}arr$:

$ntcf\text{-}paa\ a\ X\ (F(\downarrow ArrVal)(\downarrow a)) : set\ \{a\} \mapsto_{cat\text{-}Set\ \alpha}\ X$

by

(
 $cs\text{-}concl$ **cs-shallow**
cs-simp: $cat\text{-}Set\text{-}components(1)$
cs-intro: $V\text{-}cs\text{-}intros\ cat\text{-}Set\text{-}cs\text{-}intros\ cat\text{-}cs\text{-}intros$
)

then have $dom\text{-}lhs : \mathcal{D}_o\ (ntcf\text{-}paa\ a\ X\ (F(\downarrow ArrVal)(\downarrow a))(\downarrow ArrVal)) = set\ \{a\}$

by ($cs\text{-}concl$ **cs-shallow** **cs-simp**: $cat\text{-}cs\text{-}simps$)

from $assms$ **have** $dom\text{-}rhs : \mathcal{D}_o\ (F(\downarrow ArrVal)) = set\ \{a\}$

by ($cs\text{-}concl$ **cs-shallow** **cs-simp**: $cat\text{-}cs\text{-}simps$)

show $?thesis$

proof($rule\ arr\text{-}Set\text{-}eqI$)

from $lhs\text{-}is\text{-}arr\ assms$

show $arr\text{-}Set\text{-}lhs : arr\text{-}Set\ \alpha\ (ntcf\text{-}paa\ a\ X\ (F(\downarrow ArrVal)(\downarrow a)))$

and $arr\text{-}Set\text{-}rhs : arr\text{-}Set\ \alpha\ F$

by ($auto\ dest$: $cat\text{-}Set\text{-}is\text{-}arrD$)

show $ntcf\text{-}paa\ a\ X\ (F(\downarrow ArrVal)(\downarrow a))(\downarrow ArrVal) = F(\downarrow ArrVal)$

proof($rule\ vsu\text{-}eqI$, $unfold\ dom\text{-}lhs\ dom\text{-}rhs\ vsingleton\text{-}iff$; ($simp\ only$:)?)

show $ntcf\text{-}paa\ a\ X\ (F(\downarrow ArrVal)(\downarrow a))(\downarrow ArrVal)(\downarrow a) = F(\downarrow ArrVal)(\downarrow a)$

by ($cs\text{-}concl$ **cs-shallow** **cs-simp**: $cat\text{-}cs\text{-}simps$ **cs-intro**: $cat\text{-}cs\text{-}intros$)

qed ($use\ arr\text{-}Set\text{-}lhs\ arr\text{-}Set\text{-}rhs\ in\ auto$)

qed ($use\ assms\ in\ \langle cs\text{-}concl\ cs\text{-}shallow\ cs\text{-}simp : cat\text{-}cs\text{-}simps \rangle$)+

qed

lemma (in \mathcal{Z}) $ntcf\text{-}paa\text{-}ArrVal'$:

assumes $F : set\ \{a\} \mapsto_{cat\text{-}Set\ \alpha}\ X$ **and** $a = a$

shows $ntcf\text{-}paa\ a\ X\ (F(\downarrow ArrVal)(\downarrow a)) = F$

using $assms(1)$ **unfolding** $assms(2)$ **by** ($rule\ ntcf\text{-}paa\text{-}ArrVal$)

lemma (in \mathcal{Z}) $ntcf\text{-}paa\text{-}Comp\text{-}right[cat\text{-}cs\text{-}simps]$:

assumes $F : A \mapsto_{cat\text{-}Set\ \alpha}\ B$

and $a \in_o\ cat\text{-}Set\ \alpha(\downarrow Obj)$

and $a \in_o\ A$

shows $F \circ_A\ cat\text{-}Set\ \alpha\ ntcf\text{-}paa\ a\ A\ a = ntcf\text{-}paa\ a\ B\ (F(\downarrow ArrVal)(\downarrow a))$

proof–

from $assms$ **have** $F\text{-}paa$:

$F \circ_A\ cat\text{-}Set\ \alpha\ ntcf\text{-}paa\ a\ A\ a : set\ \{a\} \mapsto_{cat\text{-}Set\ \alpha}\ B$

by ($cs\text{-}concl$ **cs-intro**: $cat\text{-}cs\text{-}intros$)

then have $dom\text{-}lhs : \mathcal{D}_o\ ((F \circ_A\ cat\text{-}Set\ \alpha\ ntcf\text{-}paa\ a\ A\ a)(\downarrow ArrVal)) = set\ \{a\}$

by ($cs\text{-}concl$ **cs-shallow** **cs-simp**: $cat\text{-}cs\text{-}simps$)

from $assms$ **have** $paa : ntcf\text{-}paa\ a\ B\ (F(\downarrow ArrVal)(\downarrow a)) : set\ \{a\} \mapsto_{cat\text{-}Set\ \alpha}\ B$

by ($cs\text{-}concl$ **cs-shallow** **cs-intro**: $cat\text{-}Set\text{-}cs\text{-}intros\ cat\text{-}cs\text{-}intros$)

then have $dom\text{-}rhs: \mathcal{D}_o ((ntcf\text{-}paa \ \mathbf{a} \ B \ (F(\downarrow ArrVal)(\downarrow a)))(\downarrow ArrVal)) = set \ \{\mathbf{a}\}$
by $(cs\text{-}concl \ \mathbf{cs}\text{-}shallow \ \mathbf{cs}\text{-}simp: \ cat\text{-}cs\text{-}simps)$
show $?thesis$
proof $(rule \ arr\text{-}Set\text{-}eqI)$
from $F\text{-}paa \ paa \ assms$
show $arr\text{-}Set\text{-}lhs: arr\text{-}Set \ \alpha \ (F \circ_A \ cat\text{-}Set \ \alpha \ ntcf\text{-}paa \ \mathbf{a} \ A \ a)$
and $arr\text{-}Set\text{-}rhs: arr\text{-}Set \ \alpha \ (ntcf\text{-}paa \ \mathbf{a} \ B \ (F(\downarrow ArrVal)(\downarrow a)))$
by $(auto \ dest: \ cat\text{-}Set\text{-}is\text{-}arrD)$
show
 $(F \circ_A \ cat\text{-}Set \ \alpha \ ntcf\text{-}paa \ \mathbf{a} \ A \ a)(\downarrow ArrVal) =$
 $ntcf\text{-}paa \ \mathbf{a} \ B \ (F(\downarrow ArrVal)(\downarrow a))(\downarrow ArrVal)$
proof $(rule \ vsv\text{-}eqI, \ unfold \ dom\text{-}lhs \ dom\text{-}rhs \ vsingleton\text{-}iff; \ (simp \ only:)?)$
from $assms \ show$
 $(F \circ_A \ cat\text{-}Set \ \alpha \ ntcf\text{-}paa \ \mathbf{a} \ A \ a)(\downarrow ArrVal)(\downarrow a) =$
 $ntcf\text{-}paa \ \mathbf{a} \ B \ (F(\downarrow ArrVal)(\downarrow a))(\downarrow ArrVal)(\downarrow a)$
by $(cs\text{-}concl \ \mathbf{cs}\text{-}simp: \ cat\text{-}cs\text{-}simps \ \mathbf{cs}\text{-}intro: \ V\text{-}cs\text{-}intros \ cat\text{-}cs\text{-}intros)$
qed $(use \ arr\text{-}Set\text{-}lhs \ arr\text{-}Set\text{-}rhs \ in \ auto)$
qed $(use \ F\text{-}paa \ paa \ in \ \langle cs\text{-}concl \ cs\text{-}shallow \ cs\text{-}simp: \ cat\text{-}cs\text{-}simps \rangle) +$
qed

lemmas $[cat\text{-}cs\text{-}simps] = \mathcal{Z}.ntcf\text{-}paa\text{-}Comp\text{-}right$

8.2 Pointed natural transformation

8.2.1 Definition and elementary properties

See Chapter III-2 in [9].

definition $ntcf\text{-}pointed :: V \Rightarrow V \Rightarrow V$

where $ntcf\text{-}pointed \ \alpha \ \mathbf{a} =$

$[$
 $($
 $\lambda x \in_o \ cat\text{-}Set \ \alpha (\downarrow Obj).$
 $[$
 $(\lambda f \in_o \ Hom \ (cat\text{-}Set \ \alpha) \ (set \ \{\mathbf{a}\}) \ x. \ f(\downarrow ArrVal)(\downarrow a)),$
 $Hom \ (cat\text{-}Set \ \alpha) \ (set \ \{\mathbf{a}\}) \ x,$
 x
 $]$
 $]$
 $),$
 $Hom_{O.C} \ \alpha \ cat\text{-}Set \ \alpha (set \ \{\mathbf{a}\}, -),$
 $cf\text{-}id \ (cat\text{-}Set \ \alpha),$
 $cat\text{-}Set \ \alpha,$
 $cat\text{-}Set \ \alpha$
 $]$

Components.

lemma $ntcf\text{-}pointed\text{-}components:$

shows $ntcf\text{-}pointed \ \alpha \ \mathbf{a}(\downarrow NTMap) =$

$($
 $\lambda x \in_o \ cat\text{-}Set \ \alpha (\downarrow Obj).$
 $[$
 $(\lambda f \in_o \ Hom \ (cat\text{-}Set \ \alpha) \ (set \ \{\mathbf{a}\}) \ x. \ f(\downarrow ArrVal)(\downarrow a)),$
 $Hom \ (cat\text{-}Set \ \alpha) \ (set \ \{\mathbf{a}\}) \ x,$
 x
 $]$
 $]$
 $)$

and $[cat\text{-}cs\text{-}simps]: \ ntcf\text{-}pointed \ \alpha \ \mathbf{a}(\downarrow NTDom) = Hom_{O.C} \ \alpha \ cat\text{-}Set \ \alpha (set \ \{\mathbf{a}\}, -)$

and $[cat\text{-}cs\text{-}simps]: \ ntcf\text{-}pointed \ \alpha \ \mathbf{a}(\downarrow NTCod) = cf\text{-}id \ (cat\text{-}Set \ \alpha)$

and $[cat\text{-}cs\text{-}simps]: \ ntcf\text{-}pointed \ \alpha \ \mathbf{a}(\downarrow NTDGDom) = cat\text{-}Set \ \alpha$

and [cat-cs-simps]: ntcf-pointed α $\mathbf{a}(NTDGCod) = \text{cat-Set } \alpha$
unfolding ntcf-pointed-def nt-field-simps **by** (simp-all add: nat-omega-simps)

8.2.2 Natural transformation map

mk-VLambda ntcf-pointed-components(1)
 |vsu ntcf-pointed-NTMap-vsuv[cat-cs-intros]|
 |vdomain ntcf-pointed-NTMap-vdomain[cat-cs-simps]|
 |app ntcf-pointed-NTMap-app'|

lemma (in \mathcal{Z}) ntcf-pointed-NTMap-app-ArrVal-app[cat-cs-simps]:
assumes $X \in_{\circ} \text{cat-Set } \alpha(\text{Obj})$ **and** $F : \text{set } \{\mathbf{a}\} \mapsto_{\text{cat-Set } \alpha} X$
shows ntcf-pointed α $\mathbf{a}(\text{NTMap})(\downarrow X)(\downarrow \text{ArrVal})(\downarrow F) = F(\downarrow \text{ArrVal})(\downarrow \mathbf{a})$
by (simp add: assms(2) ntcf-pointed-NTMap-app'[OF assms(1)] arr-Rel-components)

lemma (in \mathcal{Z}) ntcf-pointed-NTMap-app-is-iso-arr:
assumes $\mathbf{a} \in_{\circ} \text{cat-Set } \alpha(\text{Obj})$ **and** $X \in_{\circ} \text{cat-Set } \alpha(\text{Obj})$
shows ntcf-pointed α $\mathbf{a}(\text{NTMap})(\downarrow X) :$
 $\text{Hom}(\text{cat-Set } \alpha)(\text{set } \{\mathbf{a}\}) X \mapsto_{\text{iso}} \text{cat-Set } \alpha X$

proof-

interpret Set: category α $\langle \text{cat-Set } \alpha \rangle$ **by** (rule category-cat-Set)

note app-X = ntcf-pointed-NTMap-app'[OF assms(2)]

show ?thesis

proof(intro cat-Set-is-iso-arrI cat-Set-is-arrI arr-SetI)

show ArrVal-vsuv: vsu (ntcf-pointed α $\mathbf{a}(\text{NTMap})(\downarrow X)(\downarrow \text{ArrVal})$)

unfolding app-X arr-Rel-components **by** simp

show vcard (ntcf-pointed α $\mathbf{a}(\text{NTMap})(\downarrow X)$) = $\mathfrak{3}_{\mathbb{N}}$

unfolding app-X arr-Rel-components **by** (simp add: nat-omega-simps)

show ArrVal-vdomain:

$\mathcal{D}_{\circ}(\text{ntcf-pointed } \alpha \mathbf{a}(\text{NTMap})(\downarrow X)(\downarrow \text{ArrVal})) = \text{Hom}(\text{cat-Set } \alpha)(\text{set } \{\mathbf{a}\}) X$

unfolding app-X arr-Rel-components **by** simp

show vrangle-left:

$\mathcal{R}_{\circ}(\text{ntcf-pointed } \alpha \mathbf{a}(\text{NTMap})(\downarrow X)(\downarrow \text{ArrVal})) \subseteq_{\circ}$

$\text{ntcf-pointed } \alpha \mathbf{a}(\text{NTMap})(\downarrow X)(\downarrow \text{ArrCod})$

unfolding app-X arr-Rel-components

by

(
 auto
 simp: in-Hom-iff
 intro: cat-Set-cs-intros
 intro!: vrangle-VLambda-vsubset
)

show $\mathcal{R}_{\circ}(\text{ntcf-pointed } \alpha \mathbf{a}(\text{NTMap})(\downarrow X)(\downarrow \text{ArrVal})) = X$

proof(intro vsubset-antisym)

show $X \subseteq_{\circ} \mathcal{R}_{\circ}(\text{ntcf-pointed } \alpha \mathbf{a}(\text{NTMap})(\downarrow X)(\downarrow \text{ArrVal}))$

proof(intro vsubsetI)

fix x **assume** prems: $x \in_{\circ} X$

from assms prems **have** F-in-vdomain:

$\text{ntcf-paa } \mathbf{a} X x \in_{\circ} \mathcal{D}_{\circ}((\text{ntcf-pointed } \alpha \mathbf{a}(\text{NTMap})(\downarrow X)(\downarrow \text{ArrVal})))$

unfolding app-X arr-Rel-components vdomain-VLambda in-Hom-iff

by (cs-concl cs-shallow cs-intro: cat-cs-intros)

from assms prems **have** x-def:

$x = \text{ntcf-pointed } \alpha \mathbf{a}(\text{NTMap})(\downarrow X)(\downarrow \text{ArrVal})(\text{ntcf-paa } \mathbf{a} X x)$

by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)

show $x \in_{\circ} \mathcal{R}_{\circ}(\text{ntcf-pointed } \alpha \mathbf{a}(\text{NTMap})(\downarrow X)(\downarrow \text{ArrVal}))$

by (subst x-def) (intro vsu.vsu-vimageI2 F-in-vdomain ArrVal-vsuv)

qed

qed (use vrangle-left in $\langle \text{simp add: app-X arr-Rel-components} \rangle$)

from *assms* **show** *ntcf-pointed* α $\mathfrak{a}(\text{NTMap})(\downarrow X)(\downarrow \text{ArrDom}) \in_{\circ} \text{Vset } \alpha$
unfolding *app-X arr-Rel-components cat-Set-components(1)*
by (*intro Set.cat-Hom-in-Vset[OF - assms(2)]*)
(auto simp: cat-Set-components(1))
show *v11 (ntcf-pointed* α $\mathfrak{a}(\text{NTMap})(\downarrow X)(\downarrow \text{ArrVal})$)
proof(*intro vsv.vsv-valeq-v11I ArrVal-vsv, unfold ArrVal-vdomain in-Hom-iff*)
fix $F G$ **assume** *prems*:
 $F : \text{set } \{\mathfrak{a}\} \mapsto \text{cat-Set } \alpha X$
 $G : \text{set } \{\mathfrak{a}\} \mapsto \text{cat-Set } \alpha X$
 $\text{ntcf-pointed } \alpha \mathfrak{a}(\text{NTMap})(\downarrow X)(\downarrow \text{ArrVal})(\downarrow F) =$
 $\text{ntcf-pointed } \alpha \mathfrak{a}(\text{NTMap})(\downarrow X)(\downarrow \text{ArrVal})(\downarrow G)$
note $F = \text{cat-Set-is-arrD}[OF \text{ prems}(1)]$ **and** $G = \text{cat-Set-is-arrD}[OF \text{ prems}(2)]$
from *prems(3,1,2) assms* **have** $F\text{-ArrVal-}G\text{-ArrVal}: F(\downarrow \text{ArrVal})(\downarrow \mathfrak{a}) = G(\downarrow \text{ArrVal})(\downarrow \mathfrak{a})$
by (*cs-prems cs-simp: cat-cs-simps*)
interpret $F: \text{arr-Set } \alpha F + G: \text{arr-Set } \alpha G$ **by** (*simp-all add: F G*)
show $F = G$
proof(*rule arr-Set-eqI*)
show $\text{arr-Set } \alpha F \text{ arr-Set } \alpha G$
by (*intro F.arr-Set-axioms G.arr-Set-axioms*)
show $F(\downarrow \text{ArrVal}) = G(\downarrow \text{ArrVal})$
by
(
rule vsv-eqI,
unfold F.arr-Set-ArrVal-vdomain G.arr-Set-ArrVal-vdomain F(2) G(2)
)
(auto simp: F-ArrVal-G-ArrVal)
qed (*simp-all add: F G*)
qed
qed (*use assms in (auto simp: app-X arr-Rel-components cat-Set-components(1))*)
qed

lemma (**in** \mathcal{Z}) *ntcf-pointed-NTMap-app-is-iso-arr'[cat-cs-intros]*:
assumes $\mathfrak{a} \in_{\circ} \text{cat-Set } \alpha(\downarrow \text{Obj})$
and $X \in_{\circ} \text{cat-Set } \alpha(\downarrow \text{Obj})$
and $A' = \text{Hom}(\text{cat-Set } \alpha)(\text{set } \{\mathfrak{a}\}) X$
and $B' = X$
and $\mathcal{C}' = \text{cat-Set } \alpha$
shows $\text{ntcf-pointed } \alpha \mathfrak{a}(\text{NTMap})(\downarrow X) : A' \mapsto_{\text{iso}} \mathcal{C}' B'$
using *assms(1,2)*
unfolding *assms(3-5)*
by (*rule ntcf-pointed-NTMap-app-is-iso-arr'*)

lemmas [*cat-cs-intros*] = $\mathcal{Z}.\text{ntcf-pointed-NTMap-app-is-iso-arr}'$

lemmas (**in** \mathcal{Z}) *ntcf-pointed-NTMap-app-is-arr'[cat-cs-intros]* =
is-iso-arrD(1)[OF $\mathcal{Z}.\text{ntcf-pointed-NTMap-app-is-iso-arr}'$]

lemmas [*cat-cs-intros*] = $\mathcal{Z}.\text{ntcf-pointed-NTMap-app-is-arr}'$

8.2.3 Pointed natural transformation is a natural isomorphism

lemma (**in** \mathcal{Z}) *ntcf-pointed-is-iso-ntcf*:
assumes $\mathfrak{a} \in_{\circ} \text{cat-Set } \alpha(\downarrow \text{Obj})$
shows $\text{ntcf-pointed } \alpha \mathfrak{a} :$
 $\text{Hom}_{O.C\alpha} \text{cat-Set } \alpha(\text{set } \{\mathfrak{a}\}, -) \mapsto_{C.F.\text{iso}} \text{cf-id}(\text{cat-Set } \alpha) :$
 $\text{cat-Set } \alpha \mapsto_{C\alpha} \text{cat-Set } \alpha$
proof(*intro is-iso-ntcfI is-ntcfI'*)

note $\mathbf{a} = \text{assms}[\text{unfolded cat-Set-components}(1)]$
from assms **have** $\text{set-}\mathbf{a}$: $\text{set } \{\mathbf{a}\} \in_{\circ} \text{cat-Set } \alpha(\text{Obj})$
unfolding $\text{cat-Set-components}$ **by** auto

show $\text{vfsequence}(\text{ntcf-pointed } \alpha \mathbf{a})$ **unfolding** ntcf-pointed-def **by** auto
show $\text{vcard}(\text{ntcf-pointed } \alpha \mathbf{a}) = 5_{\mathbb{N}}$
unfolding ntcf-pointed-def **by** $(\text{auto simp: nat-omega-simps})$

from assms $\text{set-}\mathbf{a}$ **show**
 $\text{Hom}_{\mathcal{O}.C\alpha} \text{cat-Set } \alpha(\text{set } \{\mathbf{a}\}, -) : \text{cat-Set } \alpha \mapsto_{C\alpha} \text{cat-Set } \alpha$
by $(\text{cs-concl } \mathbf{cs-shallow } \mathbf{cs-intro}: \text{cat-cs-intros})$

show $\text{ntcf-pointed } \alpha \mathbf{a}(\text{NTMap})(\text{!}a) :$
 $\text{Hom}_{\mathcal{O}.C\alpha} \text{cat-Set } \alpha(\text{set } \{\mathbf{a}\}, -)(\text{ObjMap})(\text{!}a) \mapsto_{\text{cat-Set } \alpha}$
 $\text{cf-id}(\text{cat-Set } \alpha)(\text{ObjMap})(\text{!}a)$
if $a \in_{\circ} \text{cat-Set } \alpha(\text{Obj})$ **for** a
using assms **that** $\text{set-}\mathbf{a}$
by
(

 $\text{cs-concl } \mathbf{cs-shallow}$
 $\mathbf{cs-simp}: \text{cat-cs-simps } \mathbf{cs-intro}: \text{cat-cs-intros } \text{cat-op-intros}$

)

show
 $\text{ntcf-pointed } \alpha \mathbf{a}(\text{NTMap})(\text{!}b) \circ_{A \text{cat-Set } \alpha}$
 $\text{Hom}_{\mathcal{O}.C\alpha} \text{cat-Set } \alpha(\text{set } \{\mathbf{a}\}, -)(\text{ArrMap})(\text{!}f) =$
 $\text{cf-id}(\text{cat-Set } \alpha)(\text{ArrMap})(\text{!}f) \circ_{A \text{cat-Set } \alpha} \text{ntcf-pointed } \alpha \mathbf{a}(\text{NTMap})(\text{!}a)$
if $f : a \mapsto_{\text{cat-Set } \alpha} b$ **for** $a b f$

proof-
let $?pb = \langle \text{ntcf-pointed } \alpha \mathbf{a}(\text{NTMap})(\text{!}b) \rangle$
and $?pa = \langle \text{ntcf-pointed } \alpha \mathbf{a}(\text{NTMap})(\text{!}a) \rangle$
and $?hom = \langle \text{cf-hom}(\text{cat-Set } \alpha) [\text{cat-Set } \alpha(\text{CIId})(\text{!set } \{\mathbf{a}\}), f]_{\circ} \rangle$
from assms $\text{set-}\mathbf{a}$ **that** **have** pb-hom :
 $?pb \circ_{A \text{cat-Set } \alpha} ?hom : \text{Hom}(\text{cat-Set } \alpha)(\text{set } \{\mathbf{a}\}) a \mapsto_{\text{cat-Set } \alpha} b$
by
(

 $\text{cs-concl } \mathbf{cs-shallow}$
 $\mathbf{cs-intro}: \text{cat-cs-intros } \text{cat-op-intros } \text{cat-prod-cs-intros}$

)

then **have** dom-lhs :
 $\mathcal{D}_{\circ}((?pb \circ_{A \text{cat-Set } \alpha} ?hom)(\text{!ArrVal})) = \text{Hom}(\text{cat-Set } \alpha)(\text{set } \{\mathbf{a}\}) a$
by $(\text{cs-concl } \mathbf{cs-shallow } \mathbf{cs-simp}: \text{cat-cs-simps})$

from assms $\text{set-}\mathbf{a}$ **that** **have** f-pa :
 $f \circ_{A \text{cat-Set } \alpha} ?pa : \text{Hom}(\text{cat-Set } \alpha)(\text{set } \{\mathbf{a}\}) a \mapsto_{\text{cat-Set } \alpha} b$
by $(\text{cs-concl } \mathbf{cs-intro}: \text{cat-cs-intros})$

then **have** dom-rhs :
 $\mathcal{D}_{\circ}((f \circ_{A \text{cat-Set } \alpha} ?pa)(\text{!ArrVal})) = \text{Hom}(\text{cat-Set } \alpha)(\text{set } \{\mathbf{a}\}) a$
by $(\text{cs-concl } \mathbf{cs-shallow } \mathbf{cs-simp}: \text{cat-cs-simps})$

have $[\text{cat-cs-simps}]$: $?pb \circ_{A \text{cat-Set } \alpha} ?hom = f \circ_{A \text{cat-Set } \alpha} ?pa$
proof $(\text{rule } \text{arr-Set-eqI})$

from pb-hom **show** $\text{arr-Set-pb-hom}: \text{arr-Set } \alpha (?pb \circ_{A \text{cat-Set } \alpha} ?hom)$
by $(\text{auto dest: cat-Set-is-arrD}(1))$

from f-pa **show** $\text{arr-Set-f-pa}: \text{arr-Set } \alpha (f \circ_{A \text{cat-Set } \alpha} ?pa)$
by $(\text{auto dest: cat-Set-is-arrD}(1))$

show $(?pb \circ_{A \text{cat-Set } \alpha} ?hom)(\text{!ArrVal}) = (f \circ_{A \text{cat-Set } \alpha} ?pa)(\text{!ArrVal})$
proof $(\text{rule } \text{vsu-eqI}, \text{unfold dom-lhs dom-rhs in-Hom-iff})$

fix g **assume** $g : \text{set } \{\mathbf{a}\} \mapsto_{\text{cat-Set } \alpha} a$
with assms \mathbf{a} $\text{set-}\mathbf{a}$ **that** **show**
 $(?pb \circ_{A \text{cat-Set } \alpha} ?hom)(\text{!ArrVal})(\text{!}g) = (f \circ_{A \text{cat-Set } \alpha} ?pa)(\text{!ArrVal})(\text{!}g)$
by
(

```

cs-concl cs-shallow
cs-simp:  $V$ -cs-simps cat-cs-simps
cs-intro:
   $V$ -cs-intros cat-cs-intros cat-op-intros cat-prod-cs-intros
)
qed (use arr-Set-pb-hom arr-Set-f-pa in auto)
qed (use pb-hom f-pa in (cs-concl cs-shallow cs-simp: cat-cs-simps))+
from assms that set-a show ?thesis
by
  (
    cs-concl cs-shallow
    cs-simp: cat-cs-simps cat-op-simps cs-intro: cat-cs-intros
  )
qed
show ntcf-pointed  $\alpha$  a (NTMap) (|a|) :
  Hom $_{O.C\alpha}$  cat-Set  $\alpha$  (set {a}, -) (ObjMap) (|a|)  $\mapsto_{iso}$  cat-Set  $\alpha$ 
  cf-id (cat-Set  $\alpha$ ) (ObjMap) (|a|)
if a  $\in_o$  cat-Set  $\alpha$  (Obj) for a
using assms a set-a that
by
  (
    cs-concl cs-shallow
    cs-simp: cat-cs-simps cat-op-simps cs-intro: cat-cs-intros
  )

```

qed (auto simp: ntcf-pointed-components intro: cat-cs-intros)

lemma (in \mathcal{Z}) ntcf-pointed-is-iso-ntcf' [cat-cs-intros]:
 assumes a \in_o cat-Set α (Obj)
 and $\mathfrak{F}' = \text{Hom}_{O.C\alpha} \text{cat-Set } \alpha (\text{set } \{a\}, -)$
 and $\mathfrak{G}' = \text{cf-id } (\text{cat-Set } \alpha)$
 and $\mathfrak{A}' = \text{cat-Set } \alpha$
 and $\mathfrak{B}' = \text{cat-Set } \alpha$
 and $\alpha' = \alpha$
 shows ntcf-pointed α a : $\mathfrak{F}' \mapsto_{CF.iso} \mathfrak{G}' : \mathfrak{A}' \mapsto_{C\alpha'} \mathfrak{B}'$
 using assms(1) unfolding assms(2-6) by (rule ntcf-pointed-is-iso-ntcf')

lemmas [cat-cs-intros] = $\mathcal{Z}.ntcf\text{-pointed-is-iso-ntcf}'$

8.3 Inverse pointed natural transformation

8.3.1 Definition and elementary properties

See Chapter III-2 in [9].

definition *ntcf-pointed-inv* :: $V \Rightarrow V \Rightarrow V$

where *ntcf-pointed-inv* α a =

```

[
  (
     $\lambda X \in_o \text{cat-Set } \alpha (\text{Obj}).$ 
    [( $\lambda x \in_o X. \text{ntcf-paa } a X x$ ),  $X$ , Hom (cat-Set  $\alpha$ ) (set {a}) X]
  ),
  cf-id (cat-Set  $\alpha$ ),
  Hom $_{O.C\alpha}$  cat-Set  $\alpha$  (set {a}, -),
  cat-Set  $\alpha$ ,
  cat-Set  $\alpha$ 
]

```

Components.

lemma *ntcf-pointed-inv-components*:
shows *ntcf-pointed-inv* α \mathbf{a} (*NTMap*) =
 ($\lambda X \in_{\circ} \text{cat-Set } \alpha$ (*Obj*).
 $[(\lambda x \in_{\circ} X. \text{ntcf-paa } \mathbf{a} X x), X, \text{Hom } (\text{cat-Set } \alpha) (\text{set } \{\mathbf{a}\}) X]$.
)
and [*cat-cs-simps*]: *ntcf-pointed-inv* α \mathbf{a} (*NTDom*) = *cf-id* (*cat-Set* α)
and [*cat-cs-simps*]:
 ntcf-pointed-inv α \mathbf{a} (*NTCod*) = *Hom*_{*O.C*} α *cat-Set* α (*set* $\{\mathbf{a}\}$, -)
and [*cat-cs-simps*]: *ntcf-pointed-inv* α \mathbf{a} (*NTDGDom*) = *cat-Set* α
and [*cat-cs-simps*]: *ntcf-pointed-inv* α \mathbf{a} (*NTDGCod*) = *cat-Set* α
unfolding *ntcf-pointed-inv-def nt-field-simps*
by (*simp-all add: nat-omega-simps*)

8.3.2 Natural transformation map

mk-VLambda *ntcf-pointed-inv-components(1)*
 |*vsu ntcf-pointed-inv-NTMap-vsuv[cat-cs-intros]*||
 |*vdomain ntcf-pointed-inv-NTMap-vdomain[cat-cs-simps]*||
 |*app ntcf-pointed-inv-NTMap-app*^|

lemma (in \mathcal{Z}) *ntcf-pointed-inv-NTMap-app-ArrVal-app[cat-cs-simps]*:
assumes $X \in_{\circ} \text{cat-Set } \alpha$ (*Obj*) **and** $x \in_{\circ} X$
shows *ntcf-pointed-inv* α \mathbf{a} (*NTMap*)(X)(*ArrVal*)(x) = *ntcf-paa* $\mathbf{a} X x$
by
 (*simp add:*
 assms(2) ntcf-pointed-inv-NTMap-app^|[*OF assms(1)*] *arr-Rel-components*
)

lemma (in \mathcal{Z}) *ntcf-pointed-inv-NTMap-app-is-arr*:
assumes $\mathbf{a} \in_{\circ} \text{cat-Set } \alpha$ (*Obj*) **and** $X \in_{\circ} \text{cat-Set } \alpha$ (*Obj*)
shows *ntcf-pointed-inv* α \mathbf{a} (*NTMap*)(X) :
 $X \mapsto_{\text{cat-Set } \alpha} \text{Hom } (\text{cat-Set } \alpha) (\text{set } \{\mathbf{a}\}) X$

proof-

interpret *Set*: *category* α $\langle \text{cat-Set } \alpha \rangle$ **by** (*rule category-cat-Set*)
note *app-X* = *ntcf-pointed-inv-NTMap-app*^|[*OF assms(2)*]
show *?thesis*
proof(*intro cat-Set-is-arrI arr-SetI*)
show *ArrVal-vsuv: vsu* (*ntcf-pointed-inv* α \mathbf{a} (*NTMap*)(X)(*ArrVal*))
unfolding *app-X arr-Rel-components* **by** *simp*
show *vcard* (*ntcf-pointed-inv* α \mathbf{a} (*NTMap*)(X)) = $\mathfrak{3}_{\mathbf{N}}$
unfolding *app-X arr-Rel-components* **by** (*simp add: nat-omega-simps*)
show *ArrVal-vdomain*:
 \mathcal{D}_{\circ} (*ntcf-pointed-inv* α \mathbf{a} (*NTMap*)(X)(*ArrVal*)) =
ntcf-pointed-inv α \mathbf{a} (*NTMap*)(X)(*ArrDom*)
unfolding *app-X arr-Rel-components* **by** *simp*
from *assms* **show** *vrange-left*:
 \mathcal{R}_{\circ} (*ntcf-pointed-inv* α \mathbf{a} (*NTMap*)(X)(*ArrVal*)) \subseteq_{\circ}
ntcf-pointed-inv α \mathbf{a} (*NTMap*)(X)(*ArrCod*)
unfolding *app-X arr-Rel-components* **by** (*auto intro: cat-cs-intros*)
from *assms* **show** *ntcf-pointed-inv* α \mathbf{a} (*NTMap*)(X)(*ArrCod*) $\in_{\circ} \text{Vset } \alpha$
unfolding *app-X arr-Rel-components cat-Set-components(1)*
by (*intro Set.cat-Hom-in-Vset[OF - assms(2)]*)
 (*auto simp: cat-Set-components(1)*)
qed (*use assms in* $\langle \text{auto simp: app-X arr-Rel-components cat-Set-components(1) \rangle$)
qed

lemma (in \mathcal{Z}) *ntcf-pointed-inv-NTMap-app-is-arr'* [cat-cs-intros]:
assumes $\mathbf{a} \in_{\circ} \text{cat-Set } \alpha(\text{Obj})$
and $X \in_{\circ} \text{cat-Set } \alpha(\text{Obj})$
and $A' = X$
and $B' = \text{Hom } (\text{cat-Set } \alpha) (\text{set } \{\mathbf{a}\}) X$
and $\mathcal{C}' = \text{cat-Set } \alpha$
shows *ntcf-pointed-inv* $\alpha \mathbf{a}(\text{NTMap})(X) : A' \mapsto_{\mathcal{C}'} B'$
using *assms*(1,2)
unfolding *assms*(3-5)
by (rule *ntcf-pointed-inv-NTMap-app-is-arr*)

lemmas [cat-cs-intros] = $\mathcal{Z}.\text{ntcf-pointed-inv-NTMap-app-is-arr}'$

lemma (in \mathcal{Z}) *is-inverse-ntcf-pointed-inv-NTMap-app*:
assumes $\mathbf{a} \in_{\circ} \text{cat-Set } \alpha(\text{Obj})$ **and** $X \in_{\circ} \text{cat-Set } \alpha(\text{Obj})$
shows
is-inverse
 $(\text{cat-Set } \alpha)$
 $(\text{ntcf-pointed-inv } \alpha \mathbf{a}(\text{NTMap})(X))$
 $(\text{ntcf-pointed } \alpha \mathbf{a}(\text{NTMap})(X))$
(is $\langle \text{is-inverse } (\text{cat-Set } \alpha) \text{ ?bwd ?fwd} \rangle$
proof(intro *is-inverseI*)

let $\text{?Hom} = \langle \text{Hom } (\text{cat-Set } \alpha) (\text{set } \{\mathbf{a}\}) X \rangle$

interpret *Set: category* $\alpha \langle \text{cat-Set } \alpha \rangle$ **by** (rule *category-cat-Set*)

from *assms*(1) **have** *set-a*: $\text{set } \{\mathbf{a}\} \in_{\circ} \text{cat-Set } \alpha(\text{Obj})$
unfolding *cat-Set-components*(1) **by** *blast*
have *Hom-aX*: $\text{?Hom} \in_{\circ} \text{cat-Set } \alpha(\text{Obj})$
by
 $($
auto
simp: *cat-Set-components*(1)
intro!: *Set.cat-Hom-in-Vset set-a assms*(2)
 $)$

from *assms* **show** $\text{?bwd} : X \mapsto_{\text{cat-Set } \alpha} \text{?Hom}$
by (*cs-concl cs-shallow cs-intro: cat-cs-intros*)
from *assms* **show** $\text{?fwd} : \text{?Hom} \mapsto_{\text{cat-Set } \alpha} X$
by (*cs-concl cs-shallow cs-intro: cat-cs-intros*)

from *assms* *set-a Hom-aX* **have** *lhs-is-arr*:
 $\text{?bwd} \circ_{A \text{cat-Set } \alpha} \text{?fwd} : \text{?Hom} \mapsto_{\text{cat-Set } \alpha} \text{?Hom}$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)
then have *dom-lhs*: $\mathcal{D}_{\circ} ((\text{?bwd} \circ_{A \text{cat-Set } \alpha} \text{?fwd})(\text{ArrVal})) = \text{?Hom}$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps*)

from *Hom-aX* **have** *rhs-is-arr*: $\text{cat-Set } \alpha(\text{CId})(\text{?Hom}) : \text{?Hom} \mapsto_{\text{cat-Set } \alpha} \text{?Hom}$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)
then have *dom-rhs*: $\mathcal{D}_{\circ} ((\text{cat-Set } \alpha(\text{CId})(\text{?Hom}))(\text{ArrVal})) = \text{?Hom}$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps*)

show $\text{?bwd} \circ_{A \text{cat-Set } \alpha} \text{?fwd} = \text{cat-Set } \alpha(\text{CId})(\text{?Hom})$
proof(rule *arr-Set-eqI*)
from *lhs-is-arr* **show** *arr-Set-lhs*: $\text{arr-Set } \alpha (\text{?bwd} \circ_{A \text{cat-Set } \alpha} \text{?fwd})$
by (*auto dest: cat-Set-is-arrD*)
from *rhs-is-arr* **show** *arr-Set-rhs*: $\text{arr-Set } \alpha (\text{cat-Set } \alpha(\text{CId})(\text{?Hom}))$

by (auto dest: cat-Set-is-arrD)
 show (?bwd \circ_A cat-Set α ?fwd)(ArrVal) = cat-Set α (CIId)(?Hom)(ArrVal)
 proof(rule vsv-eqI, unfold dom-lhs dom-rhs in-Hom-iff)
 fix F assume F : set {a} \mapsto cat-Set α X
 with assms Hom-aX show
 (?bwd \circ_A cat-Set α ?fwd)(ArrVal)(F) = cat-Set α (CIId)(?Hom)(ArrVal)(F)
 by
 (
 (
 cs-concl
 cs-simp: cat-cs-simps ntcf-paa-ArrVal
 cs-intro: V-cs-intros cat-Set-cs-intros cat-cs-intros
)
 qed (use arr-Set-lhs arr-Set-rhs in auto)
 qed (use lhs-is-arr rhs-is-arr in ‹cs-concl cs-shallow cs-simp: cat-cs-simps›)+

 from assms have lhs-is-arr: ?fwd \circ_A cat-Set α ?bwd : X \mapsto cat-Set α X
 by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
 then have dom-lhs: \mathcal{D}_\circ ((?fwd \circ_A cat-Set α ?bwd)(ArrVal)) = X
 by (cs-concl cs-shallow cs-simp: cat-cs-simps)

 from assms have rhs-is-arr: cat-Set α (CIId)(X) : X \mapsto cat-Set α X
 by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
 then have dom-rhs: \mathcal{D}_\circ ((cat-Set α (CIId)(X))(ArrVal)) = X
 by (cs-concl cs-shallow cs-simp: cat-cs-simps)

 show ?fwd \circ_A cat-Set α ?bwd = cat-Set α (CIId)(X)
 proof(rule arr-Set-eqI)
 from lhs-is-arr show arr-Set-lhs: arr-Set α (?fwd \circ_A cat-Set α ?bwd)
 by (auto dest: cat-Set-is-arrD)
 from rhs-is-arr show arr-Set-rhs: arr-Set α (cat-Set α (CIId)(X))
 by (auto dest: cat-Set-is-arrD)
 show (?fwd \circ_A cat-Set α ?bwd)(ArrVal) = cat-Set α (CIId)(X)(ArrVal)
 proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
 fix x assume x \in_\circ X
 with assms show
 (?fwd \circ_A cat-Set α ?bwd)(ArrVal)(x) = cat-Set α (CIId)(X)(ArrVal)(x)
 by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
 qed (use arr-Set-lhs arr-Set-rhs in auto)
 qed (use lhs-is-arr rhs-is-arr in ‹cs-concl cs-shallow cs-simp: cat-cs-simps›)+

 qed

qed

8.3.3 Inverse pointed natural transformation is a natural isomorphism

lemma (in Z) ntcf-pointed-inv-is-ntcf:
 assumes a \in_\circ cat-Set α (Obj)
 shows ntcf-pointed-inv α a :
 cf-id (cat-Set α) \mapsto_{CF} Hom $_{O.C\alpha}$ cat-Set α (set {a}, -) :
 cat-Set α $\mapsto_{C\alpha}$ cat-Set α
 proof(intro is-ntcfI')

interpret Set: category α ‹cat-Set α › by (rule category-cat-Set)

note a = assms[unfolded cat-Set-components(1)]
 from assms have set-a: set {a} \in_\circ cat-Set α (Obj)
 unfolding cat-Set-components by auto

show vsequence (ntcf-pointed-inv α a)

unfolding *ntcf-pointed-inv-def* **by** *simp*
show *vcard* (*ntcf-pointed-inv* α \mathbf{a}) = $5_{\mathbb{N}}$
unfolding *ntcf-pointed-inv-def* **by** (*simp add: nat-omega-simps*)
from *set-a* **show** $\text{Hom}_{O.C\alpha} \text{cat-Set } \alpha (\text{set } \{\mathbf{a}\}, -) : \text{cat-Set } \alpha \mapsto_{C\alpha} \text{cat-Set } \alpha$
by (*cs-concl cs-shallow cs-intro: cat-cs-intros*)

show *ntcf-pointed-inv* α $\mathbf{a}(\text{NTMap})(\downarrow a) :$
cf-id (*cat-Set* α)(*ObjMap*)($\downarrow a$) $\mapsto_{\text{cat-Set } \alpha}$
 $\text{Hom}_{O.C\alpha} \text{cat-Set } \alpha (\text{set } \{\mathbf{a}\}, -)(\text{ObjMap})(\downarrow a)$
if $a \in_{\circ} \text{cat-Set } \alpha(\text{Obj})$ **for** a
using *that assms set-a*
by
(

cs-concl cs-shallow
cs-simp: *cat-cs-simps cs-intro: cat-cs-intros cat-op-intros*
)

show
ntcf-pointed-inv α $\mathbf{a}(\text{NTMap})(\downarrow B) \circ_{A \text{cat-Set } \alpha}$ *cf-id* (*cat-Set* α)(*ArrMap*)($\downarrow F$) =
 $\text{Hom}_{O.C\alpha} \text{cat-Set } \alpha (\text{set } \{\mathbf{a}\}, -)(\text{ArrMap})(\downarrow F) \circ_{A \text{cat-Set } \alpha}$
ntcf-pointed-inv α $\mathbf{a}(\text{NTMap})(\downarrow A)$
if $F : A \mapsto_{\text{cat-Set } \alpha} B$ **for** $A B F$

proof-

let $?pb = \langle \text{ntcf-pointed-inv } \alpha \mathbf{a}(\text{NTMap})(\downarrow B) \rangle$
and $?pa = \langle \text{ntcf-pointed-inv } \alpha \mathbf{a}(\text{NTMap})(\downarrow A) \rangle$
and $?hom = \langle \text{cf-hom } (\text{cat-Set } \alpha) [\text{cat-Set } \alpha(\text{CId})(\downarrow \text{set } \{\mathbf{a}\}), F]_{\circ} \rangle$

from *assms set-a* **that have** $pb-F$:

$?pb \circ_{A \text{cat-Set } \alpha} F : A \mapsto_{\text{cat-Set } \alpha} \text{Hom } (\text{cat-Set } \alpha) (\text{set } \{\mathbf{a}\}) B$
by (*cs-concl cs-shallow cs-intro: cat-cs-intros cat-prod-cs-intros*)

then have *dom-lhs*: $\mathcal{D}_{\circ} ((?pb \circ_{A \text{cat-Set } \alpha} F)(\downarrow \text{ArrVal})) = A$

by (*cs-concl cs-shallow cs-simp: cat-cs-simps*)

from *that assms set-a* **that have** $hom-pa$:

$?hom \circ_{A \text{cat-Set } \alpha} ?pa : A \mapsto_{\text{cat-Set } \alpha} \text{Hom } (\text{cat-Set } \alpha) (\text{set } \{\mathbf{a}\}) B$
by (*cs-concl cs-intro: cat-cs-intros cat-prod-cs-intros cat-op-intros*)

then have *dom-rhs*: $\mathcal{D}_{\circ} ((?hom \circ_{A \text{cat-Set } \alpha} ?pa)(\downarrow \text{ArrVal})) = A$

by (*cs-concl cs-shallow cs-simp: cat-cs-simps*)

have [*cat-cs-simps*]: $?pb \circ_{A \text{cat-Set } \alpha} F = ?hom \circ_{A \text{cat-Set } \alpha} ?pa$

proof(*rule arr-Set-eqI*)

from $pb-F$ **show** *arr-Set-pb-F*: $\text{arr-Set } \alpha (?pb \circ_{A \text{cat-Set } \alpha} F)$

by (*auto dest: cat-Set-is-arrD(1)*)

from $hom-pa$ **show** *arr-Set-hom-pa*: $\text{arr-Set } \alpha (?hom \circ_{A \text{cat-Set } \alpha} ?pa)$

by (*auto dest: cat-Set-is-arrD(1)*)

show $(?pb \circ_{A \text{cat-Set } \alpha} F)(\downarrow \text{ArrVal}) = (?hom \circ_{A \text{cat-Set } \alpha} ?pa)(\downarrow \text{ArrVal})$

proof(*rule vsv-eqI, unfold dom-lhs dom-rhs*)

fix a **assume** $a \in_{\circ} A$

with *assms a set-a* **that show**

$(?pb \circ_{A \text{cat-Set } \alpha} F)(\downarrow \text{ArrVal})(\downarrow a) = (?hom \circ_{A \text{cat-Set } \alpha} ?pa)(\downarrow \text{ArrVal})(\downarrow a)$

by

(

cs-concl
cs-simp: *cat-cs-simps*
cs-intro:
cat-Set-cs-intros
cat-cs-intros
cat-prod-cs-intros
cat-op-intros
)

qed (*use arr-Set-pb-F arr-Set-hom-pa in auto*)

qed (use pb-F hom-pa in ⟨cs-concl cs-shallow cs-simp: cat-cs-simps⟩)+
 from *assms* that set-a show ?thesis
 by
 (
 cs-concl **cs-shallow**
cs-simp: cat-cs-simps cat-op-simps **cs-intro**: cat-cs-intros
)
 qed

qed (cs-concl **cs-simp**: cat-cs-simps **cs-intro**: cat-cs-intros)+

lemma (in \mathcal{Z}) *ntcf-pointed-inv-is-ntcf'*[cat-cs-intros]:
 assumes $\mathbf{a} \in_{\circ} \text{cat-Set } \alpha(\text{Obj})$
 and $\mathfrak{F}' = \text{cf-id } (\text{cat-Set } \alpha)$
 and $\mathfrak{G}' = \text{Hom}_{O.C\alpha} \text{cat-Set } \alpha(\text{set } \{\mathbf{a}\}, -)$
 and $\mathfrak{A}' = \text{cat-Set } \alpha$
 and $\mathfrak{B}' = \text{cat-Set } \alpha$
 and $\alpha' = \alpha$
 shows *ntcf-pointed-inv* α $\mathbf{a} : \mathfrak{F}' \mapsto_{CF} \mathfrak{G}' : \mathfrak{A}' \mapsto_{C\alpha'} \mathfrak{B}'$
 using *assms*(1) **unfolding** *assms*(2-6) **by** (rule *ntcf-pointed-inv-is-ntcf'*)

lemmas [cat-cs-intros] = $\mathcal{Z}.$ *ntcf-pointed-inv-is-ntcf'*

lemma (in \mathcal{Z}) *inv-ntcf-ntcf-pointed*[cat-cs-simps]:
 assumes $\mathbf{a} \in_{\circ} \text{cat-Set } \alpha(\text{Obj})$
 shows *inv-ntcf* (*ntcf-pointed* α \mathbf{a}) = *ntcf-pointed-inv* α \mathbf{a}
 by
 (
 rule *iso-ntcf-if-is-inverse*(3)[*symmetric*],
 rule *is-iso-ntcfD*(1)[*OF ntcf-pointed-is-iso-ntcf*[*OF assms*]],
 rule *ntcf-pointed-inv-is-ntcf*[*OF assms*],
 rule *is-inverse-ntcf-pointed-inv-NTMap-app*[*OF assms*]
)

lemma (in \mathcal{Z}) *inv-ntcf-ntcf-pointed-inv*[cat-cs-simps]:
 assumes $\mathbf{a} \in_{\circ} \text{cat-Set } \alpha(\text{Obj})$
 shows *inv-ntcf* (*ntcf-pointed-inv* α \mathbf{a}) = *ntcf-pointed* α \mathbf{a}
 by
 (
 rule *iso-ntcf-if-is-inverse*(4)[*symmetric*],
 rule *is-iso-ntcfD*(1)[*OF ntcf-pointed-is-iso-ntcf*[*OF assms*]],
 rule *ntcf-pointed-inv-is-ntcf*[*OF assms*],
 rule *is-inverse-ntcf-pointed-inv-NTMap-app*[*OF assms*]
)

lemma (in \mathcal{Z}) *ntcf-pointed-inv-is-iso-ntcf*:
 assumes $\mathbf{a} \in_{\circ} \text{cat-Set } \alpha(\text{Obj})$
 shows *ntcf-pointed-inv* α $\mathbf{a} :$
 $\text{cf-id } (\text{cat-Set } \alpha) \mapsto_{CF.iso} \text{Hom}_{O.C\alpha} \text{cat-Set } \alpha(\text{set } \{\mathbf{a}\}, -) :$
 $\text{cat-Set } \alpha \mapsto_{C\alpha} \text{cat-Set } \alpha$
 by
 (
 rule *iso-ntcf-if-is-inverse*(2),
 rule *is-iso-ntcfD*(1)[*OF ntcf-pointed-is-iso-ntcf*[*OF assms*]],
 rule *ntcf-pointed-inv-is-ntcf*[*OF assms*],
 rule *is-inverse-ntcf-pointed-inv-NTMap-app*[*OF assms*]
)

lemma (in \mathcal{Z}) *ntcf-pointed-inv-is-iso-ntcf'*[*cat-cs-intros*]:
assumes $\mathfrak{a} \in_{\circ} \text{cat-Set } \alpha(\text{Obj})$
and $\mathfrak{F}' = \text{cf-id } (\text{cat-Set } \alpha)$
and $\mathfrak{G}' = \text{Hom}_{O.C\alpha} \text{cat-Set } \alpha(\text{set } \{\mathfrak{a}\}, -)$
and $\mathfrak{A}' = \text{cat-Set } \alpha$
and $\mathfrak{B}' = \text{cat-Set } \alpha$
and $\alpha' = \alpha$
shows *ntcf-pointed-inv* $\alpha \mathfrak{a} : \mathfrak{F}' \mapsto_{CF.iso} \mathfrak{G}' : \mathfrak{A}' \mapsto_{C\alpha'} \mathfrak{B}'$
using *assms(1)* **unfolding** *assms(2-6)* **by** (rule *ntcf-pointed-inv-is-iso-ntcf'*)

lemmas [*cat-cs-intros*] = $\mathcal{Z}.$ *ntcf-pointed-inv-is-iso-ntcf'*

9 Representable and corepresentable functors

9.1 Representable and corepresentable functors

9.1.1 Definitions and elementary properties

See Chapter III-2 in [9] or Section 2.1 in [14].

definition *cat-representation* $:: V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$

where *cat-representation* $\alpha \mathfrak{F} c \psi \longleftrightarrow$

$c \in_{\circ} \mathfrak{F}(\mathit{HomDom})(\mathit{Obj}) \wedge$

$\psi : \mathit{Hom}_{O.C\alpha} \mathfrak{F}(\mathit{HomDom})(c, -) \mapsto_{CF.iso} \mathfrak{F} : \mathfrak{F}(\mathit{HomDom}) \mapsto_{C\alpha} \mathit{cat-Set} \alpha$

definition *cat-corepresentation* $:: V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$

where *cat-corepresentation* $\alpha \mathfrak{F} c \psi \longleftrightarrow$

$c \in_{\circ} \mathfrak{F}(\mathit{HomDom})(\mathit{Obj}) \wedge$

$\psi : \mathit{Hom}_{O.C\alpha} \mathit{op-cat}(\mathfrak{F}(\mathit{HomDom}))(-, c) \mapsto_{CF.iso} \mathfrak{F} : \mathfrak{F}(\mathit{HomDom}) \mapsto_{C\alpha} \mathit{cat-Set} \alpha$

Rules.

context

fixes $\alpha \mathfrak{C} \mathfrak{F}$

assumes $\mathfrak{F} : \mathfrak{C} \mapsto_{C\alpha} \mathit{cat-Set} \alpha$

begin

interpretation $\mathfrak{F} : \mathit{is-functor} \alpha \mathfrak{C} \langle \mathit{cat-Set} \alpha \rangle \mathfrak{F}$ by (rule \mathfrak{F})

mk-ide rf *cat-representation-def*[**where** $\alpha = \alpha$ **and** $\mathfrak{F} = \mathfrak{F}$, *unfolded cat-cs-simps*]

[intro cat-representationI

|dest cat-representationD'

|elim cat-representationE'

end

lemmas *cat-representationD*[*dest*] = *cat-representationD'*[*rotated*]

and *cat-representationE*[*elim*] = *cat-representationE'*[*rotated*]

lemma *cat-corepresentationI*:

assumes *category* $\alpha \mathfrak{C}$

and $\mathfrak{F} : \mathit{op-cat} \mathfrak{C} \mapsto_{C\alpha} \mathit{cat-Set} \alpha$

and $c \in_{\circ} \mathfrak{C}(\mathit{Obj})$

and $\psi : \mathit{Hom}_{O.C\alpha} \mathfrak{C}(-, c) \mapsto_{CF.iso} \mathfrak{F} : \mathit{op-cat} \mathfrak{C} \mapsto_{C\alpha} \mathit{cat-Set} \alpha$

shows *cat-corepresentation* $\alpha \mathfrak{F} c \psi$

proof-

interpret *category* $\alpha \mathfrak{C}$ by (rule *assms(1)*)

interpret $\mathfrak{F} : \mathit{is-functor} \alpha \langle \mathit{op-cat} \mathfrak{C} \rangle \langle \mathit{cat-Set} \alpha \rangle \mathfrak{F}$ by (rule *assms(2)*)

note [*cat-op-simps*] = $\mathfrak{F}.\mathit{HomDom}.\mathit{cat-op-cat-cf-Hom-snd}$ [

symmetric, unfolded cat-op-simps, OF assms(3)

]

show *?thesis*

unfolding *cat-corepresentation-def*

by (*intro conjI, unfold cat-cs-simps cat-op-simps; intro assms*)

qed

lemma *cat-corepresentationD*:

assumes *cat-corepresentation* $\alpha \mathfrak{F} c \psi$

and *category* $\alpha \mathfrak{C}$

and $\mathfrak{F} : \mathit{op-cat} \mathfrak{C} \mapsto_{C\alpha} \mathit{cat-Set} \alpha$

shows $c \in_{\circ} \mathfrak{C}(\mathit{Obj})$

and $\psi : \mathit{Hom}_{O.C\alpha} \mathfrak{C}(-, c) \mapsto_{CF.iso} \mathfrak{F} : \mathit{op-cat} \mathfrak{C} \mapsto_{C\alpha} \mathit{cat-Set} \alpha$

proof-

interpret *category* α \mathfrak{C} **by** (*rule* *assms*(2))
interpret \mathfrak{F} : *is-functor* α \langle *op-cat* \mathfrak{C} \rangle \langle *cat-Set* α \rangle \mathfrak{F} **by** (*rule* *assms*(3))
note $c\psi = \text{cat-corepresentation-def}$
THEN iffD1, OF assms(1), unfolded cat-cs-simps cat-op-simps
from $c\psi(1)$ **show** $c: c \in_{\circ} \mathfrak{C}(\text{Obj})$ **by** *auto*
note $[\text{cat-op-simps}] = \mathfrak{F}.\text{HomDom}.\text{cat-op-cat-cf-Hom-snd}$
symmetric, unfolded cat-op-simps, OF c
show $\psi: \text{Hom}_{O.C\alpha}\mathfrak{C}(-,c) \mapsto_{CF.iso} \mathfrak{F}: \text{op-cat } \mathfrak{C} \mapsto_{C\alpha} \text{cat-Set } \alpha$
by (*rule* *conjunct2*[*OF c* ψ , *unfolded cat-op-simps*])

qed

lemma *cat-corepresentationE*:

assumes *cat-corepresentation* α \mathfrak{F} c ψ
and *category* α \mathfrak{C}
and $\mathfrak{F}: \text{op-cat } \mathfrak{C} \mapsto_{C\alpha} \text{cat-Set } \alpha$
obtains $c \in_{\circ} \mathfrak{C}(\text{Obj})$
and $\psi: \text{Hom}_{O.C\alpha}\mathfrak{C}(-,c) \mapsto_{CF.iso} \mathfrak{F}: \text{op-cat } \mathfrak{C} \mapsto_{C\alpha} \text{cat-Set } \alpha$
by (*simp add: cat-corepresentationD*[*OF assms*])

9.1.2 Representable functors and universal arrows

lemma *universal-arrow-of-if-cat-representation*:

— See Proposition 2 in Chapter III-2 in [9].

assumes $\mathfrak{K}: \mathfrak{C} \mapsto_{C\alpha} \text{cat-Set } \alpha$
and *cat-representation* α \mathfrak{K} r ψ
and $\mathfrak{a} \in_{\circ} \text{cat-Set } \alpha(\text{Obj})$
shows *universal-arrow-of*
 $\mathfrak{K}(\text{set } \{\mathfrak{a}\})$ r (*ntcf-paa* \mathfrak{a} ($\mathfrak{K}(\text{ObjMap})(r)$) ($\psi(\text{NTMap})(r)(\text{ArrVal})(\mathfrak{C}(\text{CId})(r))$))

proof-

note $r\psi = \text{cat-representationD}$ [*OF assms*(2,1)]
interpret \mathfrak{K} : *is-functor* α \mathfrak{C} \langle *cat-Set* α \rangle \mathfrak{K} **by** (*rule* *assms*(1))
interpret ψ : *is-iso-ntcf* α \mathfrak{C} \langle *cat-Set* α \rangle \langle *Hom*_{*O.C* α} $\mathfrak{C}(r,-)$ \rangle \mathfrak{K} ψ
by (*rule* $r\psi(2)$)
from *assms*(3) **have** *set-a: set* $\{\mathfrak{a}\} \in_{\circ} \text{cat-Set } \alpha(\text{Obj})$
by (*simp add: Limit-vsingleton-in-VsetI cat-Set-components*(1))
from
ntcf-cf-comp-is-iso-ntcf
OF $\mathfrak{K}.\text{ntcf-pointed-inv-is-iso-ntcf}$ [*OF assms*(3)] *assms*(1)
have $\mathfrak{a}\mathfrak{K}$: *ntcf-pointed-inv* α \mathfrak{a} $\circ_{NTCF-CF} \mathfrak{K}$:
 $\mathfrak{K} \mapsto_{CF.iso} \text{Hom}_{O.C\alpha} \text{cat-Set } \alpha$ (*set* $\{\mathfrak{a}\}, -$) $\circ_{CF} \mathfrak{K}: \mathfrak{C} \mapsto_{C\alpha} \text{cat-Set } \alpha$
by (*cs-prems cs-simp: cat-cs-simps*)
from *iso-ntcf-is-iso-arr*(1)[*OF* $\mathfrak{a}\mathfrak{K}$] $r\psi$ *assms*(3) **have** [*cat-cs-simps*]:
 $((\text{ntcf-pointed-inv } \alpha \mathfrak{a} \circ_{NTCF-CF} \mathfrak{K} \cdot_{NTCF} \psi)(\text{NTMap})(r)(\text{ArrVal})(\mathfrak{C}(\text{CId})(r))) =$
 $\text{ntcf-paa } \mathfrak{a}$ ($\mathfrak{K}(\text{ObjMap})(r)$) ($\psi(\text{NTMap})(r)(\text{ArrVal})(\mathfrak{C}(\text{CId})(r))$)
by
 $($
cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-Set-cs-intros cat-cs-intros cat-op-intros*
 $)$
show *universal-arrow-of*
 $\mathfrak{K}(\text{set } \{\mathfrak{a}\})$ r (*ntcf-paa* \mathfrak{a} ($\mathfrak{K}(\text{ObjMap})(r)$) ($\psi(\text{NTMap})(r)(\text{ArrVal})(\mathfrak{C}(\text{CId})(r))$))
by
 $($

rule $\mathfrak{K}.cf\text{-universal-arrow-of-if-is-iso-ntcf}$
 $[$
 $OF\ r\psi(1)\ set\text{-}\mathfrak{a}\ ntcf\text{-vcomp-is-iso-ntcf}[OF\ \mathfrak{a}\mathfrak{K}\ r\psi(2)],$
unfolded cat-cs-simps
 $]$
 $)$
qed

lemma *universal-arrow-of-if-cat-corepresentation:*
— See Proposition 2 in Chapter III-2 in [9].
assumes *category* $\alpha\ \mathfrak{C}$
and $\mathfrak{K} : op\text{-cat}\ \mathfrak{C} \mapsto_{C\alpha} cat\text{-Set}\ \alpha$
and *cat-corepresentation* $\alpha\ \mathfrak{K}\ r\ \psi$
and $\mathfrak{a} \in_{\circ} cat\text{-Set}\ \alpha(\mathit{Obj})$
shows *universal-arrow-of*
 $\mathfrak{K}\ (set\ \{\mathfrak{a}\})\ r\ (ntcf\text{-paa}\ \mathfrak{a}\ (\mathfrak{K}(\mathit{ObjMap})(\mathit{r}))\ (\psi(\mathit{NTMap})(\mathit{r})(\mathit{ArrVal})(\mathfrak{C}(\mathit{CId})(\mathit{r}))))$

proof-
interpret \mathfrak{C} : *category* $\alpha\ \mathfrak{C}$ **by** (*rule* *assms*(1))
note $r\psi = cat\text{-corepresentation}D[OF\ assms(3,1,2)]$
note $[cat\text{-op-simps}] = \mathfrak{C}.cat\text{-op-cat-cf-Hom-snd}[OF\ r\psi(1)]$
have *rep*: *cat-representation* $\alpha\ \mathfrak{K}\ r\ \psi$
by (*intro* *cat-representationI*, *rule* *assms*(2), *unfold* *cat-op-simps*; *rule* $r\psi$)
show *?thesis*
by
 $($
rule *universal-arrow-of-if-cat-representation*
 $OF\ assms(2)\ rep\ assms(4),\ unfolded\ cat\text{-op-simps}$
 $]$
 $)$
qed

lemma *cat-representation-if-universal-arrow-of:*
— See Proposition 2 in Chapter III-2 in [9].
assumes $\mathfrak{K} : \mathfrak{C} \mapsto_{C\alpha} cat\text{-Set}\ \alpha$
and $\mathfrak{a} \in_{\circ} cat\text{-Set}\ \alpha(\mathit{Obj})$
and *universal-arrow-of* $\mathfrak{K}\ (set\ \{\mathfrak{a}\})\ r\ u$
shows *cat-representation* $\alpha\ \mathfrak{K}\ r\ (Yoneda\text{-arrow}\ \alpha\ \mathfrak{K}\ r\ (u(\mathit{ArrVal})(\mathfrak{a})))$

proof-

let $?Y = \langle Yoneda\text{-component}\ \mathfrak{K}\ r\ (u(\mathit{ArrVal})(\mathfrak{a})) \rangle$

interpret \mathfrak{K} : *is-functor* $\alpha\ \mathfrak{C}\ \langle cat\text{-Set}\ \alpha \rangle\ \mathfrak{K}$ **by** (*rule* *assms*(1))

note $ua = \mathfrak{K}.universal\text{-arrow-of}D[OF\ assms(3)]$

from $ua(2)$ **have** $ua : u(\mathit{ArrVal})(\mathfrak{a}) \in_{\circ} \mathfrak{K}(\mathit{ObjMap})(\mathit{r})$

by
 $($
cs-concl **cs-shallow**
cs-intro: *V-cs-intros* *cat-Set-cs-intros* *cat-cs-intros*
 $)$

have $[cat\text{-cs-simps}] : Yoneda\text{-arrow}\ \alpha\ \mathfrak{K}\ r\ (u(\mathit{ArrVal})(\mathfrak{a}))(\mathit{NTMap})(\mathit{c}) = ?Y\ c$

if $c \in_{\circ} \mathfrak{C}(\mathit{Obj})$ **for** c

using *that*

by (*cs-concl* **cs-shallow** **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)

from $ua(1)$ **have** $[cat\text{-cs-simps}] : Hom_{O.C\alpha}\mathfrak{C}(r,-)(\mathit{ObjMap})(\mathit{c}) = Hom\ \mathfrak{C}\ r\ c$

if $c \in_{\circ} \mathfrak{C}(\mathit{Obj})$ **for** c

using that
 by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-op-intros)

show ?thesis

proof

(
 intro cat-representationI is-iso-ntcfI,
 rule assms(1),
 rule ua(1),
 rule $\mathfrak{K}.HomDom.cat\text{-}Yoneda\text{-}arrow\text{-}is\text{-}ntcf[OF\ assms(1)\ ua(1)\ ua]$,
 rule cat-Set-is-iso-arrI,
 simp-all only: cat-cs-simps
)

fix c assume prems: $c \in_o \mathfrak{C}(Obj)$
 with ua(1,2) show $Yc: ?Y\ c : Hom\ \mathfrak{C}\ r\ c \mapsto_{cat\text{-}Set\ \alpha} \mathfrak{K}(ObjMap)(c)$
 by
 (
 cs-concl cs-shallow
 cs-intro: V-cs-intros cat-Set-cs-intros cat-cs-intros
)

note $YcD = cat\text{-}Set\text{-}is\text{-}arrD[OF\ Yc]$

interpret $Yc: arr\text{-}Set\ \alpha \langle ?Y\ c \rangle$ by (rule YcD(1))

show $dom\text{-}Yc: \mathcal{D}_o\ (?Y\ c(ArrVal)) = Hom\ \mathfrak{C}\ r\ c$
 by (simp add: $\mathfrak{K}.Yoneda\text{-}component\text{-}ArrVal\text{-}vdomain$)

show v11 ($?Y\ c(ArrVal)$)

proof(intro $Yc.ArrVal.vsv\text{-}valeq\text{-}v11I$, unfold dom-Yc in-Hom-iff)

fix g f assume prems':
 $g : r \mapsto_{\mathfrak{C}} c\ f : r \mapsto_{\mathfrak{C}} c\ ?Y\ c(ArrVal)(g) = ?Y\ c(ArrVal)(f)$

from prems have $c : c \in_o \mathfrak{C}(Obj)$ by auto

from prems'(3,1,2) have $\mathfrak{K}gua\text{-}\mathfrak{K}fua$:
 $\mathfrak{K}(ArrMap)(g)(ArrVal)(u(ArrVal)(a)) = \mathfrak{K}(ArrMap)(f)(ArrVal)(u(ArrVal)(a))$
 by (cs-prems cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)

from prems'(1,2) ua(1,2) have $\mathfrak{K}g\text{-}u$:
 $\mathfrak{K}(ArrMap)(g) \circ_{A\ cat\text{-}Set\ \alpha} u : set\ \{a\} \mapsto_{cat\text{-}Set\ \alpha} \mathfrak{K}(ObjMap)(c)$
 and $\mathfrak{K}f\text{-}u$:
 $\mathfrak{K}(ArrMap)(f) \circ_{A\ cat\text{-}Set\ \alpha} u : set\ \{a\} \mapsto_{cat\text{-}Set\ \alpha} \mathfrak{K}(ObjMap)(c)$
 by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)+
 then have dom-lhs: $\mathcal{D}_o\ ((\mathfrak{K}(ArrMap)(g) \circ_{A\ cat\text{-}Set\ \alpha} u)(ArrVal)) = set\ \{a\}$
 and dom-rhs: $\mathcal{D}_o\ ((\mathfrak{K}(ArrMap)(f) \circ_{A\ cat\text{-}Set\ \alpha} u)(ArrVal)) = set\ \{a\}$
 by (cs-concl cs-shallow cs-simp: cat-cs-simps)+

have $\mathfrak{K}g\text{-}\mathfrak{K}f$: $\mathfrak{K}(ArrMap)(g) \circ_{A\ cat\text{-}Set\ \alpha} u = \mathfrak{K}(ArrMap)(f) \circ_{A\ cat\text{-}Set\ \alpha} u$

proof(rule arr-Set-eqI)

from $\mathfrak{K}g\text{-}u$ show $arr\text{-}Set\text{-}\mathfrak{K}g\text{-}u: arr\text{-}Set\ \alpha\ (\mathfrak{K}(ArrMap)(g) \circ_{A\ cat\text{-}Set\ \alpha} u)$
 by (auto dest: cat-Set-is-arrD)

from $\mathfrak{K}f\text{-}u$ show $arr\text{-}Set\text{-}\mathfrak{K}f\text{-}u: arr\text{-}Set\ \alpha\ (\mathfrak{K}(ArrMap)(f) \circ_{A\ cat\text{-}Set\ \alpha} u)$
 by (auto dest: cat-Set-is-arrD)

show

$(\mathfrak{K}(ArrMap)(g) \circ_{A\ cat\text{-}Set\ \alpha} u)(ArrVal) =$

$(\mathfrak{K}(\text{ArrMap})(f) \circ_{A \text{ cat-Set } \alpha} u)(\text{ArrVal})$
proof (*rule vsv-eqI, unfold dom-lhs dom-rhs usingleton-iff; (simp only)?*)
from *prems'*(1,2) *ua*(2) **show**
 $(\mathfrak{K}(\text{ArrMap})(g) \circ_{A \text{ cat-Set } \alpha} u)(\text{ArrVal})(\mathbf{a}) =$
 $(\mathfrak{K}(\text{ArrMap})(f) \circ_{A \text{ cat-Set } \alpha} u)(\text{ArrVal})(\mathbf{a})$
by
(

cs-concl cs-shallow
cs-simp: *cat-cs-simps* $\mathfrak{K}g\mathfrak{u}\mathfrak{a}\mathfrak{K}f\mathfrak{u}\mathfrak{a}$
cs-intro: *V-cs-intros cat-cs-intros*
)

qed (*use arr-Set- $\mathfrak{K}g$ -u arr-Set- $\mathfrak{K}f$ -u in auto*)
qed (*use $\mathfrak{K}g$ -u $\mathfrak{K}f$ -u in \langle cs-concl cs-shallow cs-simp: cat-cs-simps \rangle*)
from *prems'*(1) *ua*(2) **have**
 $\mathfrak{K}(\text{ArrMap})(g) \circ_{A \text{ cat-Set } \alpha} u : \text{set } \{\mathbf{a}\} \mapsto_{\text{cat-Set } \alpha} \mathfrak{K}(\text{ObjMap})(c)$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)
from *ua*(3)[*OF c this*] **obtain** *h* **where** *h*: $h : r \mapsto_{\mathfrak{C}} c$
and *$\mathfrak{K}g$ -u-def*:
 $\mathfrak{K}(\text{ArrMap})(g) \circ_{A \text{ cat-Set } \alpha} u = \text{umap-of } \mathfrak{K}(\text{set } \{\mathbf{a}\}) r u c(\text{ArrVal})(h)$
and *h-unique*: $\wedge h'$.
[[
h' : $r \mapsto_{\mathfrak{C}} c$;
 $\mathfrak{K}(\text{ArrMap})(g) \circ_{A \text{ cat-Set } \alpha} u = \text{umap-of } \mathfrak{K}(\text{set } \{\mathbf{a}\}) r u c(\text{ArrVal})(h')$
]] $\implies h' = h$
by *metis*
from *prems'*(1,2) *ua*(2) **have**
 $\mathfrak{K}(\text{ArrMap})(g) \circ_{A \text{ cat-Set } \alpha} u = \text{umap-of } \mathfrak{K}(\text{set } \{\mathbf{a}\}) r u c(\text{ArrVal})(g)$
 $\mathfrak{K}(\text{ArrMap})(g) \circ_{A \text{ cat-Set } \alpha} u = \text{umap-of } \mathfrak{K}(\text{set } \{\mathbf{a}\}) r u c(\text{ArrVal})(f)$
by
(

cs-concl cs-shallow
cs-simp: *cat-cs-simps* $\mathfrak{K}g\mathfrak{K}f$ **cs-intro:** *cat-cs-intros*
)

from *h-unique*[*OF prems'*(1) *this*(1)] *h-unique*[*OF prems'*(2) *this*(2)] **show**
 $g = f$
by *simp*
qed

show $\mathcal{R}_o (?Y c(\text{ArrVal})) = \mathfrak{K}(\text{ObjMap})(c)$
proof
(

intro
vsubset-antisym Yc.arr-Par-ArrVal-vrange[unfolded YcD]
vsubsetI
)

fix *y* **assume** *prems'*: $y \in_o \mathfrak{K}(\text{ObjMap})(c)$
from *prems* **have** $\mathfrak{K}c : \mathfrak{K}(\text{ObjMap})(c) \in_o \text{cat-Set } \alpha(\text{Obj})$
by (*cs-concl cs-shallow cs-intro: cat-cs-intros*)
from *ua*(3)[*OF prems* $\mathfrak{K}.\text{ntcf-paa-is-arr}$ [*OF assms*(2) $\mathfrak{K}c$ *prems'*]] **obtain** *f*
where *f*: $f : r \mapsto_{\mathfrak{C}} c$
and *ntcf-paa-y*:
 $\text{ntcf-paa } \mathbf{a} (\mathfrak{K}(\text{ObjMap})(c)) y = \text{umap-of } \mathfrak{K}(\text{set } \{\mathbf{a}\}) r u c(\text{ArrVal})(f)$
and *f-unique*: $\wedge f'$.
[[
f' : $r \mapsto_{\mathfrak{C}} c$;
 $\text{ntcf-paa } \mathbf{a} (\mathfrak{K}(\text{ObjMap})(c)) y = \text{umap-of } \mathfrak{K}(\text{set } \{\mathbf{a}\}) r u c(\text{ArrVal})(f')$
]] $\implies f' = f$
by *metis*

```

from ntcf-paa-y f ua(2) have
  ntcf-paa a (K(ObjMap)(c)) y = K(ArrMap)(f) ∘A cat-Set α u
by (cs-prems cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
then have
  ntcf-paa a (K(ObjMap)(c)) y(ArrVal)(a) =
    (K(ArrMap)(f) ∘A cat-Set α u)(ArrVal)(a)
by simp
from this f ua(2) have [symmetric, cat-cs-simps]:
  y = K(ArrMap)(f)(ArrVal)(u(ArrVal)(a))
by
  (
    cs-prems cs-shallow
    cs-simp: cat-cs-simps cs-intro: V-cs-intros cat-cs-intros
  )
show y ∈o Ro (?Y c(ArrVal))
by (intro Yc.ArrVal.vsv-vimageI2')
  (
    use f in
    <
      cs-concl cs-shallow
      cs-simp: cat-cs-simps cs-intro: cat-cs-intros
    >
  )+
qed
qed

```

qed

lemma *cat-corepresentation-if-universal-arrow-of:*

— See Proposition 2 in Chapter III-2 in [9].

assumes category α C

and K : op-cat C ⇔_{C α} cat-Set α

and a ∈_o cat-Set α(Obj)

and universal-arrow-of K (set {a}) r u

shows cat-corepresentation α K r (Yoneda-arrow α K r (u(ArrVal)(a)))

proof—

interpret C: category α C **by** (rule assms(1))

interpret K: is-functor α <op-cat C> <cat-Set α> K **by** (rule assms(2))

note ua = K.universal-arrow-ofD[OF assms(4), unfolded cat-op-simps]

note [cat-op-simps] = C.cat-op-cat-cf-Hom-snd[OF ua(1)]

show ?thesis

by

```

  (
    intro cat-corepresentationI,
    rule assms(1),
    rule assms(2),
    rule ua(1),
    rule cat-representationD(2)
  [
    OF
    cat-representation-if-universal-arrow-of[OF assms(2,3,4)]
    assms(2),
    unfolded cat-op-simps
  ]
  )

```

qed

9.2 Limits and colimits as universal cones

lemma *is-tm-cat-limit-if-cat-corepresentation*:

— See Definition 3.1.5 in Section 3.1 in [14].

assumes $\mathfrak{F} : \mathfrak{J} \mapsto \mapsto C.tm\alpha \mathfrak{C}$

and *cat-corepresentation* α (*tm-cf-Cone* α \mathfrak{F}) $r \psi$

(**is** \langle *cat-corepresentation* α $?Cone$ $r \psi$ \rangle)

shows *ntcf-of-ntcf-arrow* $\mathfrak{J} \mathfrak{C}$ (ψ ($\backslash NTMap$) (r) ($\backslash ArrVal$) ($\backslash \mathfrak{C}$ ($\backslash CId$) (r))) :

$r <_{CF.tm.lim} \mathfrak{F} : \mathfrak{J} \mapsto \mapsto C.tm\alpha \mathfrak{C}$

(**is** \langle *ntcf-of-ntcf-arrow* $\mathfrak{J} \mathfrak{C}$ $? \psi r 1 r : r <_{CF.tm.lim} \mathfrak{F} : \mathfrak{J} \mapsto \mapsto C.tm\alpha \mathfrak{C}$ \rangle)

proof–

let $?P = \langle$ *ntcf-paa* 0 \rangle **and** $?Funct = \langle$ *cat-Funct* α $\mathfrak{J} \mathfrak{C}$ \rangle

interpret \mathfrak{F} : *is-tm-functor* α $\mathfrak{J} \mathfrak{C} \mathfrak{F}$ **by** (*rule* *assms*(1))

interpret *Funct*: *category* α $?Funct$

by

(
cs-concl **cs-shallow** **cs-intro**:
cat-small-cs-intros *cat-cs-intros* *cat-FUNCT-cs-intros*
)

note $r\psi =$ *cat-corepresentationD*[

OF *assms*(2) $\mathfrak{F}.HomCod.category-axioms$ $\mathfrak{F}.tm-cf-cf-Cone-is-functor$
]

interpret ψ : *is-iso-ntcf* α \langle *op-cat* \mathfrak{C} \rangle \langle *cat-Set* α \rangle \langle *Hom* $_{O.C\alpha} \mathfrak{C}(-, r)$ \rangle $?Cone$ ψ

by (*rule* $r\psi$ (2))

have $0 : 0 \in_0$ *cat-Set* α ($\backslash Obj$) **unfolding** *cat-Set-components* **by** *auto*

note $ua =$ *universal-arrow-of-if-cat-corepresentation*[

OF $\mathfrak{F}.HomCod.category-axioms$ $\mathfrak{F}.tm-cf-cf-Cone-is-functor$ *assms*(2) 0
]

show $?thesis$

proof(*rule* *is-tm-cat-limitI'*)

from $r\psi$ (1) **have** [*cat-FUNCT-cs-simps*]:

cf-of-cf-map $\mathfrak{J} \mathfrak{C}$ (*cf-map* (*cf-const* $\mathfrak{J} \mathfrak{C} r$)) = *cf-const* $\mathfrak{J} \mathfrak{C} r$

by

(
cs-concl
cs-simp: *cat-FUNCT-cs-simps*
cs-intro: *cat-cs-intros* *cat-FUNCT-cs-intros*
)

from $\psi.ntcf-NTMap-is-arr$ [*unfolded* *cat-op-simps*, *OF* $r\psi$ (1)] $r\psi$ (1) **have**

ψ ($\backslash NTMap$) (r) :

Hom $\mathfrak{C} r r \mapsto_{cat-Set \alpha} Hom ?Funct$ (*cf-map* (*cf-const* $\mathfrak{J} \mathfrak{C} r$)) (*cf-map* \mathfrak{F})

by

(
cs-prems
cs-simp: *cat-small-cs-simps* *cat-cs-simps* *cat-op-simps*
cs-intro: *cat-cs-intros*
)

with $r\psi$ (1) **have** $\psi r-r$:

$? \psi r 1 r :$ *cf-map* (*cf-const* $\mathfrak{J} \mathfrak{C} r$) $\mapsto_{?Funct}$ *cf-map* \mathfrak{F}

by

(
cs-concl **cs-shallow** **cs-intro**:
cat-Set-cs-intros *cat-cs-intros* *in-Hom-iff*[*symmetric*]
)

from $r\psi(1)$ *cat-Funct-is-arrD*(1)[*OF* ψr -*r*, *unfolded cat-FUNCT-cs-simps*]
show *ntcf-of-ntcf-arrow* $\mathfrak{J} \mathfrak{C} \ ?\psi r1r : r <_{CF.tm.cone} \mathfrak{F} : \mathfrak{J} \mapsto_{C.tm\alpha} \mathfrak{C}$
by (*intro is-tm-cat-coneI*)
(cs-concl cs-shallow cs-intro: cat-cs-intros cat-small-cs-intros)

fix $r' u'$ **assume** $u' : r' <_{CF.tm.cone} \mathfrak{F} : \mathfrak{J} \mapsto_{C.tm\alpha} \mathfrak{C}$
then interpret $u' : is-tm-cat-cone \alpha r' \mathfrak{J} \mathfrak{C} \mathfrak{F} u'$.

have *Cone-r'*: *tm-cf-Cone* $\alpha \mathfrak{F}(\text{ObjMap})(\text{!}r')$ \in_0 *cat-Set* $\alpha(\text{Obj})$
by (*cs-concl cs-intro: cat-lim-cs-intros cat-cs-intros cat-op-intros*)
from $r\psi(1)$ **have** *Cone-r*: *tm-cf-Cone* $\alpha \mathfrak{F}(\text{ObjMap})(\text{!}r)$ \in_0 *cat-Set* $\alpha(\text{Obj})$
by (*cs-concl cs-shallow cs-intro: cat-cs-intros cat-op-intros*)

from $r\psi(1)$ **have** $\psi r1r$:
 $\psi(\text{NTMap})(\text{!}r)(\text{!}ArrVal)(\text{!}\mathfrak{C}(\text{!}CIId)(\text{!}r)) \in_0$ *tm-cf-Cone* $\alpha \mathfrak{F}(\text{ObjMap})(\text{!}r)$
by
 (

cs-concl cs-shallow

cs-simp: *cat-small-cs-simps cat-cs-simps cat-op-simps*

cs-intro: *cat-cs-intros*

)

have $u' : ntcf-arrow u' \in_0 \ ?Cone(\text{ObjMap})(\text{!}r')$
by
 (

cs-concl

cs-simp: *cat-small-cs-simps*

cs-intro: *cat-small-cs-intros cat-FUNCT-cs-intros cat-cs-intros*

)

have [*cat-cs-simps*]:
cf-of-cf-map $\mathfrak{J} \mathfrak{C} (cf-map \mathfrak{F}) = \mathfrak{F}$
cf-of-cf-map $\mathfrak{J} \mathfrak{C} (cf-map (cf-const \mathfrak{J} \mathfrak{C} r)) = cf-const \mathfrak{J} \mathfrak{C} r$
by (*cs-concl cs-simp: cat-FUNCT-cs-simps*)+

from *Cone-r 0* $\psi r1r$ $r\psi(1)$ **have** $\psi r1r-is-arr$: $\psi(\text{NTMap})(\text{!}r)(\text{!}ArrVal)(\text{!}\mathfrak{C}(\text{!}CIId)(\text{!}r)) :$
cf-map $(cf-const \mathfrak{J} \mathfrak{C} r) \mapsto_{?Funct} cf-map \mathfrak{F}$
by
 (

cs-concl cs-shallow

cs-simp: *cat-cs-simps cat-small-cs-simps*

cs-intro: *cat-cs-intros cat-op-intros*

)

from $r\psi(1)$ **have** [*cat-cs-intros*]:
 $\text{Hom } ?Funct (cf-map (cf-const \mathfrak{J} \mathfrak{C} r)) (cf-map \mathfrak{F}) \in_0$ *cat-Set* $\alpha(\text{Obj})$
unfolding *cat-Set-components(1)*
by (*intro Funct.cat-Hom-in-Vset*)
 (

cs-concl

cs-simp: *cat-FUNCT-cs-simps*

cs-intro: *cat-small-cs-intros cat-FUNCT-cs-intros cat-cs-intros*

)+

note $\psi r1r-is-arrD = cat-Funct-is-arrD[OF \psi r1r-is-arr, unfolded cat-cs-simps]$

from *is-functor.universal-arrow-ofD(3)*
 [

OF $\mathfrak{F}.tm-cf-cf-Cone-is-functor ua,$

unfolded cat-op-simps,
OF u'.cat-cone-obj \mathfrak{F} .ntcf-paa-is-arr[OF 0 Cone-r' u']
]

obtain f **where** $f: f : r' \mapsto_{\mathfrak{C}} r$
and $Pf: ?P (?Cone(\text{ObjMap})(\downarrow r')) (ntcf\text{-arrow } u') =$
umap-of ?Cone (set {0}) r (?P (?Cone(\text{ObjMap})(\downarrow r)) ? $\psi r1r$) r'(\downarrow ArrVal)(\downarrow f)
and $f\text{-unique}: \wedge f'$.

[[
f' : r' \mapsto_{\mathfrak{C}} r;
?P (?Cone(\text{ObjMap})(\downarrow r')) (ntcf\text{-arrow } u') =
umap-of ?Cone (set {0}) r (?P (?Cone(\text{ObjMap})(\downarrow r)) ? $\psi r1r$) r'(\downarrow ArrVal)(\downarrow f')
]] $\implies f' = f$
by *metis*

show $\exists !f$.
f : r' \mapsto_{\mathfrak{C}} r \wedge
u' = ntcf-of-ntcf-arrow $\mathfrak{J} \mathfrak{C} ?\psi r1r \cdot_{NTCF} ntcf\text{-const } \mathfrak{J} \mathfrak{C} f$
proof(*intro exI conjI; (elim conjE)?*)
show $f : r' \mapsto_{\mathfrak{C}} r$ **by** (*rule f*)
from Pf *Cone-r 0 f $\psi r1r$ $\psi r1r\text{-is-arr}$ $\psi r1r\text{-is-arrD}(1)$* **show**
u' = ntcf-of-ntcf-arrow $\mathfrak{J} \mathfrak{C} ?\psi r1r \cdot_{NTCF} ntcf\text{-const } \mathfrak{J} \mathfrak{C} f$
by (*subst (asm) $\psi r1r\text{-is-arrD}(2)$*)
(

cs-prems
cs-simp: *cat-FUNCT-cs-simps cat-small-cs-simps cat-cs-simps*
cs-intro:
cat-small-cs-intros
cat-cs-intros
cat-FUNCT-cs-intros
cat-prod-cs-intros
cat-op-intros

)

fix f' **assume** *prems*:
f' : r' \mapsto_{\mathfrak{C}} r
u' = ntcf-of-ntcf-arrow $\mathfrak{J} \mathfrak{C} ?\psi r1r \cdot_{NTCF} ntcf\text{-const } \mathfrak{J} \mathfrak{C} f'$
from Pf *Cone-r 0 f $\psi r1r$ $\psi r1r\text{-is-arr}$ $\psi r1r\text{-is-arrD}(1)$* *prems(1)* **have**
?P (?Cone(\text{ObjMap})(\downarrow r')) (ntcf\text{-arrow } u') =
umap-of ?Cone (set {0}) r (?P (?Cone(\text{ObjMap})(\downarrow r)) ? $\psi r1r$) r'(\downarrow ArrVal)(\downarrow f')
by (*subst $\psi r1r\text{-is-arrD}(2)$*)
(

cs-concl
cs-simp: *cat-FUNCT-cs-simps cat-small-cs-simps cat-cs-simps prems(2)*
cs-intro:
cat-small-cs-intros
cat-FUNCT-cs-intros
cat-cs-intros
cat-prod-cs-intros
cat-op-intros

)

from $f\text{-unique}[OF$ *prems(1)* $this]$ **show** $f' = f$.
qed

qed

qed

lemma *cat-corepresentation-if-is-tm-cat-limit:*

— See Definition 3.1.5 in Section 3.1 in [14].

assumes $\psi : r <_{CF.tm.lim} \mathfrak{F} : \mathfrak{J} \mapsto_{C.tm\alpha} \mathfrak{C}$

shows *cat-corepresentation*

α (*tm-cf-Cone* α \mathfrak{F}) r (*Yoneda-arrow* α (*tm-cf-Cone* α \mathfrak{F}) r (*ntcf-arrow* ψ))
 (**is** *cat-corepresentation* α *?Cone* r *?Y ψ*)

proof-

let *?Funct* = *cat-Funct* α \mathfrak{J} \mathfrak{C}

and *?P- ψ* = *ntcf-paa* 0 (*?Cone*(*ObjMap*)(r)) (*ntcf-arrow* ψ)

and *?ntcf-of* = *ntcf-of-ntcf-arrow* \mathfrak{J} \mathfrak{C}

interpret ψ : *is-tm-cat-limit* α \mathfrak{J} \mathfrak{C} \mathfrak{F} r ψ **by** (*rule* *assms*(1))

interpret *Funct*: *category* α *?Funct*

by

(
cs-concl **cs-shallow** **cs-intro**:
cat-small-cs-intros *cat-cs-intros* *cat-FUNCT-cs-intros*
)

interpret *Cone*: *is-functor* α *op-cat* \mathfrak{C} *cat-Set* α *?Cone*

by (*rule* ψ .*NTCod.tm-cf-cf-Cone-is-functor*)

have 0: 0 \in_0 *cat-Set* α (*Obj*) **unfolding** *cat-Set-components* **by** *auto*

have *ntcf-arrow- ψ* :

ntcf-arrow ψ : *cf-map* (*cf-const* \mathfrak{J} \mathfrak{C} r) $\mapsto_{?Funct}$ *cf-map* \mathfrak{F}

by (*cs-concl* **cs-shallow** **cs-intro**: *cat-small-cs-intros* *cat-FUNCT-cs-intros*)

from ψ .*cat-cone-obj* 0 *ntcf-arrow- ψ* **have** *P- ψ* :

?P- ψ : *set* {0} $\mapsto_{cat-Set \alpha}$ *?Cone*(*ObjMap*)(r)

by

(
cs-concl **cs-shallow**
cs-intro: *cat-cs-intros* *cat-op-intros*
cs-simp: *cat-small-cs-simps* *cat-FUNCT-cs-simps*
)

have *universal-arrow-of* *?Cone* (*set* {0}) r *?P- ψ*

proof(*rule* *Cone.universal-arrow-ofI*, *unfold* *cat-op-simps*, *rule* ψ .*cat-cone-obj*)

from 0 ψ .*cat-cone-obj* **show** *?P- ψ* : *set* {0} $\mapsto_{cat-Set \alpha}$ *?Cone*(*ObjMap*)(r)

by

(
cs-concl
cs-intro:
cat-small-cs-intros
cat-cs-intros
cat-FUNCT-cs-intros
cat-op-intros
cs-simp: *cat-small-cs-simps*
)

fix r' u' **assume** *prems*:

$r' \in_0$ \mathfrak{C} (*Obj*) $u' : set \{0\} \mapsto_{cat-Set \alpha} ?Cone(ObjMap)(r')$

let *?const- r'* = *cf-map* (*cf-const* \mathfrak{J} \mathfrak{C} r')

let *?Hom- $r\mathfrak{F}$* = *Hom* *?Funct* *?const- r'* (*cf-map* \mathfrak{F})

from *prems*(2,1) **have** u' : $u' : set \{0\} \mapsto_{cat-Set \alpha} ?Hom-r\mathfrak{F}$

by

(

```

    cs-prems cs-shallow
      cs-simp: cat-small-cs-simps cat-cs-simps cs-intro: cat-cs-intros
    )
from prems(1) have [cat-FUNCT-cs-simps]:
  cf-of-cf-map  $\mathfrak{J} \mathfrak{C} ?const-r' = cf-const \mathfrak{J} \mathfrak{C} r'$ 
by
  (
    cs-concl
      cs-simp: cat-cs-simps cat-FUNCT-cs-simps cs-intro: cat-cs-intros
    )

from
  cat-Set-ArrVal-app-vrange[OF prems(2) vintersection-vsingleton]
  prems(1)
have  $u'(\downarrow ArrVal)(\downarrow 0) : ?const-r' \mapsto ?Funct\ cf-map \mathfrak{F}$ 
by (cs-prems cs-shallow cs-simp: cat-small-cs-simps cat-cs-simps)
note  $u'0 = cat-Funct-is-arrD[OF this, unfolded\ cat-FUNCT-cs-simps]$ 

interpret  $u'0: is-tm-cat-cone \alpha r' \mathfrak{J} \mathfrak{C} \mathfrak{F} \langle ?ntcf-of (u'(\downarrow ArrVal)(\downarrow 0)) \rangle$ 
by
  (
    rule is-tm-cat-coneI[
      OF is-tm-ntcfD(1)[OF  $u'0(1)$ ]  $\psi.NTCod.is-tm-functor-axioms\ prems(1)$ 
    ]
  )

from  $\psi.tm-cat-lim-ua-fo[OF\ u'0.is-cat-cone-axioms]$  obtain  $f$ 
where  $f: f : r' \mapsto_{\mathfrak{C}} r$ 
and  $u'0-def: ?ntcf-of (u'(\downarrow ArrVal)(\downarrow 0)) = \psi \cdot_{NTCF} ntcf-const \mathfrak{J} \mathfrak{C} f$ 
and  $f-unique: \wedge f'$ .
  [[
     $f' : r' \mapsto_{\mathfrak{C}} r;$ 
     $?ntcf-of (u'(\downarrow ArrVal)(\downarrow 0)) = \psi \cdot_{NTCF} ntcf-const \mathfrak{J} \mathfrak{C} f'$ 
  ]]  $\implies f' = f$ 
by metis

note [cat-FUNCT-cs-simps] =
   $\psi.ntcf-paa-ArrVal\ u'0(2)[symmetric]\ u'0-def[symmetric]$ 

show  $\exists !f'$ .
   $f' : r' \mapsto_{\mathfrak{C}} r \wedge u' = umap-of\ ?Cone (set\ \{0\})\ r\ ?P-\psi\ r'(\downarrow ArrVal)(\downarrow f')$ 
proof(intro exII conjI; (elim conjE)?; (rule f)?)

from  $f\ 0\ u'\ ntcf-arrow-\psi$  show
   $u' = umap-of\ ?Cone (set\ \{0\})\ r\ ?P-\psi\ r'(\downarrow ArrVal)(\downarrow f)$ 
by
  (
    cs-concl
      cs-simp: cat-cs-simps
      cs-intro:
        cat-small-cs-intros
        cat-FUNCT-cs-intros
        cat-prod-cs-intros
        cat-cs-intros
        cat-op-intros
      cs-simp: cat-FUNCT-cs-simps cat-small-cs-simps
    )

```

```

fix  $f'$  assume  $prems'$ :
   $f' : r' \mapsto_{\mathfrak{C}} r$ 
   $u' = \text{umap-of } ?Cone \text{ (set } \{0\}) \text{ } r \text{ } ?P\text{-}\psi \text{ } r' \text{ (} \downarrow \text{ArrVal)} \text{ (} \downarrow \text{f')} \text{)}$ 

let  $?f' = \langle \text{ntcf-const } \mathfrak{J} \text{ } \mathfrak{C} \text{ } f' \rangle$ 

from  $prems'(2,1) \text{ } 0 \text{ } \text{ntcf-arrow-}\psi \text{ } P\text{-}\psi$  have
   $u' = \text{ntcf-paa } 0 \text{ } ?Hom\text{-}r\mathfrak{F} \text{ (ntcf-arrow } (\psi \cdot_{NTCF} ?f'))$ 
  unfolding
     $Cone.\text{umap-of-ArrVal-app}[\text{unfolded cat-op-simps, OF } prems'(1) \text{ } P\text{-}\psi]$ 
  by
    (
       $cs\text{-prems}$ 
      cs-simp:  $cat\text{-FUNCT-}cs\text{-simps } cat\text{-small-}cs\text{-simps } cat\text{-cs-}simps$ 
      cs-intro:
         $cat\text{-small-}cs\text{-intros}$ 
         $cat\text{-FUNCT-}cs\text{-intros}$ 
         $cat\text{-prod-}cs\text{-intros}$ 
         $cat\text{-cs-}intros$ 
         $cat\text{-op-}intros$ 
      )
  then have
     $?ntcf\text{-of } (u' \text{ (} \downarrow \text{ArrVal)} \text{ (} \downarrow 0)) =$ 
     $?ntcf\text{-of } ((\text{ntcf-paa } 0 \text{ } ?Hom\text{-}r\mathfrak{F} \text{ (ntcf-arrow } (\psi \cdot_{NTCF} ?f')) \text{ (} \downarrow \text{ArrVal)} \text{ (} \downarrow 0))$ 
  by simp
from  $this \text{ } prems'(1)$  have  $?ntcf\text{-of } (u' \text{ (} \downarrow \text{ArrVal)} \text{ (} \downarrow 0)) = \psi \cdot_{NTCF} ?f'$ 
  by
    (
       $cs\text{-prems}$  cs-shallow
      cs-simp:  $cat\text{-cs-}simps \text{ } cat\text{-FUNCT-}cs\text{-simps}$  cs-intro:  $cat\text{-cs-}intros$ 
    )
from  $f\text{-unique}[\text{OF } prems'(1) \text{ } this]$  show  $f' = f$  .

qed

qed

from
   $cat\text{-corepresentation-if-universal-arrow-of}$ 
   $OF \text{ } \psi.NT\text{Dom.}Hom\text{Cod.}category\text{-axioms } Cone.is\text{-functor-axioms } 0 \text{ } this$ 
]
show  $cat\text{-corepresentation } \alpha \text{ } ?Cone \text{ } r \text{ } ?Y\psi$ 
by ( $cs\text{-prems}$  cs-shallow cs-simp:  $cat\text{-cs-}simps$ )

qed

```

10 Completeness and cocompleteness

10.1 Limits by products and equalizers

lemma *cat-limit-of-cat-prod-obj-and-cat-equalizer*:

— See Theorem 1 in Chapter V-2 in [9].

assumes $\mathfrak{F} : \mathfrak{J} \mapsto \mapsto_{C.tm\alpha} \mathfrak{C}$

and $\bigwedge a\ b\ g\ f. \llbracket f : a \mapsto_{\mathfrak{C}} b; g : a \mapsto_{\mathfrak{C}} b \rrbracket \implies$

$\exists E\ \varepsilon. \varepsilon : E <_{CF.eq} (a, b, g, f) : \uparrow_C \mapsto \mapsto_{C\alpha} \mathfrak{C}$

and $\bigwedge A. tm\text{-}cf\text{-discrete}\ \alpha\ (\mathfrak{J}(\mathcal{O}bj))\ A\ \mathfrak{C} \implies$

$\exists P\ \pi. \pi : P <_{CF.\Pi} A : \mathfrak{J}(\mathcal{O}bj) \mapsto \mapsto_{C\alpha} \mathfrak{C}$

and $\bigwedge A. tm\text{-}cf\text{-discrete}\ \alpha\ (\mathfrak{J}(\mathcal{A}rr))\ A\ \mathfrak{C} \implies$

$\exists P\ \pi. \pi : P <_{CF.\Pi} A : \mathfrak{J}(\mathcal{A}rr) \mapsto \mapsto_{C\alpha} \mathfrak{C}$

obtains $r\ u$ **where** $u : r <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto \mapsto_{C\alpha} \mathfrak{C}$

proof–

let $?L = \langle \lambda u. \mathfrak{F}(\mathcal{O}bjMap)(\mathfrak{J}(\mathcal{C}od)(u)) \rangle$ **and** $?R = \langle \lambda i. \mathfrak{F}(\mathcal{O}bjMap)(i) \rangle$

interpret \mathfrak{F} : *is-tm-functor* $\alpha\ \mathfrak{J}\ \mathfrak{C}\ \mathfrak{F}$ **by** (*rule assms(1)*)

have $?R\ j \in_{\circ} \mathfrak{C}(\mathcal{O}bj)$ **if** $j \in_{\circ} \mathfrak{J}(\mathcal{O}bj)$ **for** j

by (*cs-concl cs-shallow cs-intro: cat-cs-intros that*)

have *tm-cf-discrete* $\alpha\ (\mathfrak{J}(\mathcal{O}bj))\ ?R\ \mathfrak{C}$

proof(*intro tm-cf-discreteI*)

show $\mathfrak{F}(\mathcal{O}bjMap)(i) \in_{\circ} \mathfrak{C}(\mathcal{O}bj)$ **if** $i \in_{\circ} \mathfrak{J}(\mathcal{O}bj)$ **for** i

by (*cs-concl cs-shallow cs-intro: cat-cs-intros that*)

show $VLambda\ (\mathfrak{J}(\mathcal{O}bj))\ ?R \in_{\circ} Vset\ \alpha$

proof(*rule vrelation.vrelation-Limit-in-VsetI*)

show $\mathcal{R}_{\circ}\ (VLambda\ (\mathfrak{J}(\mathcal{O}bj))\ ?R) \in_{\circ} Vset\ \alpha$

proof–

have $\mathcal{R}_{\circ}\ (VLambda\ (\mathfrak{J}(\mathcal{O}bj))\ ?R) \subseteq_{\circ} \mathcal{R}_{\circ}\ (\mathfrak{F}(\mathcal{O}bjMap))$

by (*auto simp: \mathfrak{F}.cf-ObjMap-vdomain*)

moreover have $\mathcal{R}_{\circ}\ (\mathfrak{F}(\mathcal{O}bjMap)) \in_{\circ} Vset\ \alpha$

by (*force intro: vrange-in-VsetI \mathfrak{F}.tm-cf-ObjMap-in-Vset*)

ultimately show *?thesis* **by** *auto*

qed

qed (*auto simp: cat-small-cs-intros*)

show $(\lambda i \in_{\circ} \mathfrak{J}(\mathcal{O}bj). \mathfrak{C}(\mathcal{C}Id)(\mathfrak{F}(\mathcal{O}bjMap)(i))) \in_{\circ} Vset\ \alpha$

proof(*rule vrelation.vrelation-Limit-in-VsetI*)

show $\mathcal{R}_{\circ}\ (\lambda i \in_{\circ} \mathfrak{J}(\mathcal{O}bj). \mathfrak{C}(\mathcal{C}Id)(\mathfrak{F}(\mathcal{O}bjMap)(i))) \in_{\circ} Vset\ \alpha$

proof–

have $\mathcal{R}_{\circ}\ (\lambda i \in_{\circ} \mathfrak{J}(\mathcal{O}bj). \mathfrak{C}(\mathcal{C}Id)(\mathfrak{F}(\mathcal{O}bjMap)(i))) \subseteq_{\circ} \mathcal{R}_{\circ}\ (\mathfrak{F}(\mathcal{A}rrMap))$

proof(*rule vrange-VLambda-vsubset*)

fix x **assume** $x : x \in_{\circ} \mathfrak{J}(\mathcal{O}bj)$

then have $\mathfrak{J}(\mathcal{C}Id)(x) \in_{\circ} \mathcal{D}_{\circ}\ (\mathfrak{F}(\mathcal{A}rrMap))$

by (*auto intro: cat-cs-intros simp: cat-cs-simps*)

moreover from x **have** $\mathfrak{C}(\mathcal{C}Id)(\mathfrak{F}(\mathcal{O}bjMap)(x)) = \mathfrak{F}(\mathcal{A}rrMap)(\mathfrak{J}(\mathcal{C}Id)(x))$

by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

ultimately show $\mathfrak{C}(\mathcal{C}Id)(\mathfrak{F}(\mathcal{O}bjMap)(x)) \in_{\circ} \mathcal{R}_{\circ}\ (\mathfrak{F}(\mathcal{A}rrMap))$

by (*simp add: \mathfrak{F}.ArrMap.vsv-vimageI2*)

qed

moreover have $\mathcal{R}_{\circ}\ (\mathfrak{F}(\mathcal{A}rrMap)) \in_{\circ} Vset\ \alpha$

by (*force intro: vrange-in-VsetI \mathfrak{F}.tm-cf-ArrMap-in-Vset*)

ultimately show *?thesis* **by** *auto*

qed

qed (*auto simp: cat-small-cs-intros*)

qed (*auto intro: cat-cs-intros*)

from $assms(\beta)$ [where $A = \langle ?R \rangle$, OF this] obtain $P_O \pi_O$
 where $\pi_O : \pi_O : P_O <_{CF.\Pi} ?R : \mathfrak{J}(Obj) \mapsto_{C\alpha} \mathfrak{C}$
 by *clarsimp*

interpret π_O : *is-cat-obj-prod* $\alpha \langle \mathfrak{J}(Obj) \rangle ?R \mathfrak{C} P_O \pi_O$ by (rule π_O)

have $P_O : P_O \in_o \mathfrak{C}(Obj)$ by (*intro* π_O .*cat-cone-obj*)

have $?L u \in_o \mathfrak{C}(Obj)$ if $u \in_o \mathfrak{J}(Arr)$ for u

proof-

from *that* obtain $a b$ where $u : a \mapsto_{\mathfrak{J}} b$ by *auto*

then show *?thesis*

by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

qed

have *tm-cf-discrete: tm-cf-discrete* $\alpha (\mathfrak{J}(Arr)) ?L \mathfrak{C}$

proof(*intro tm-cf-discreteI*)

show $\mathfrak{F}(ObjMap)(\mathfrak{J}(Cod)(f)) \in_o \mathfrak{C}(Obj)$ if $f \in_o \mathfrak{J}(Arr)$ for f

proof-

from *that* obtain $a b$ where $f : a \mapsto_{\mathfrak{J}} b$ by *auto*

then show *?thesis*

by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

qed

show $(\lambda u \in_o \mathfrak{J}(Arr). \mathfrak{F}(ObjMap)(\mathfrak{J}(Cod)(u))) \in_o Vset \alpha$

proof(*rule vrelation.vrelation-Limit-in-VsetI*)

show $\mathcal{R}_o (\lambda u \in_o \mathfrak{J}(Arr). ?L u) \in_o Vset \alpha$

proof-

have $\mathcal{R}_o (\lambda u \in_o \mathfrak{J}(Arr). ?L u) \subseteq_o \mathcal{R}_o (\mathfrak{F}(ObjMap))$

proof(*rule vrange-VLambda-ussubset*)

fix f assume $f \in_o \mathfrak{J}(Arr)$

then obtain $a b$ where $f : a \mapsto_{\mathfrak{J}} b$ by *auto*

then show $\mathfrak{F}(ObjMap)(\mathfrak{J}(Cod)(f)) \in_o \mathcal{R}_o (\mathfrak{F}(ObjMap))$

by

(

cs-concl cs-shallow

cs-simp: cat-cs-simps cs-intro: V-cs-intros cat-cs-intros

)

qed

moreover have $\mathcal{R}_o (\mathfrak{F}(ObjMap)) \in_o Vset \alpha$

by (*auto intro: vrange-in-VsetI \mathfrak{F}.tm-cf-ObjMap-in-Vset*)

ultimately show *?thesis* by *auto*

qed

qed (*auto simp: cat-small-cs-intros*)

show $(\lambda i \in_o \mathfrak{J}(Arr). \mathfrak{C}(CIId)(\mathfrak{F}(ObjMap)(\mathfrak{J}(Cod)(i)))) \in_o Vset \alpha$

proof(*rule vrelation.vrelation-Limit-in-VsetI*)

show $\mathcal{R}_o (\lambda i \in_o \mathfrak{J}(Arr). \mathfrak{C}(CIId)(\mathfrak{F}(ObjMap)(\mathfrak{J}(Cod)(i)))) \in_o Vset \alpha$

proof-

have $\mathcal{R}_o (\lambda i \in_o \mathfrak{J}(Arr). \mathfrak{C}(CIId)(\mathfrak{F}(ObjMap)(\mathfrak{J}(Cod)(i)))) \subseteq_o \mathcal{R}_o (\mathfrak{F}(ArrMap))$

proof(*rule vrange-VLambda-ussubset*)

fix f assume $f \in_o \mathfrak{J}(Arr)$

then obtain $a b$ where $f : a \mapsto_{\mathfrak{J}} b$ by *auto*

then have $\mathfrak{J}(CIId)(b) \in_o \mathcal{D}_o (\mathfrak{F}(ArrMap))$

by (*auto intro: cat-cs-intros simp: cat-cs-simps*)

moreover from f have

$\mathfrak{C}(CIId)(\mathfrak{F}(ObjMap)(\mathfrak{J}(Cod)(f))) = \mathfrak{F}(ArrMap)(\mathfrak{J}(CIId)(b))$

by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)
 ultimately show $\mathfrak{C}(CId)(\mathfrak{F}(ObjMap)(\mathfrak{J}(Cod)(f))) \in_o \mathcal{R}_o(\mathfrak{F}(ArrMap))$
 by (*simp add: $\mathfrak{F}.ArrMap.vsv-vimageI2$*)
 qed
 moreover have $\mathcal{R}_o(\mathfrak{F}(ArrMap)) \in_o Vset \alpha$
 by (*force intro: vrange-in-VsetI $\mathfrak{F}.tm-cf-ArrMap-in-Vset$*)
 ultimately show *?thesis* by auto
 qed
 qed (*auto simp: cat-small-cs-intros*)
 qed (*auto intro: cat-cs-intros*)

from *assms(4)* [where $A = \langle ?L \rangle$, *OF this, simplified*] obtain $P_A \pi_A$
 where $\pi_A: \pi_A : P_A <_{CF} \Pi \quad ?L : \mathfrak{J}(Arr) \mapsto_{C\alpha} \mathfrak{C}$
 by *auto*

interpret π_A : *is-cat-obj-prod* $\alpha \langle \mathfrak{J}(Arr) \rangle ?L \mathfrak{C} P_A \pi_A$ by (*rule π_A*)

let $?F = \langle \lambda u. \mathfrak{F}(ObjMap)(\mathfrak{J}(Cod)(u)) \rangle$ and $?f = \langle \lambda u. \pi_O(NTMap)(\mathfrak{J}(Cod)(u)) \rangle$
 let $?P_O' = \langle ntcf-obj-prod-base \mathfrak{C} (:_C(\mathfrak{J}(Arr)))(Obj) \rangle ?F P_O ?f$

have $\pi_O': ?\pi_O' :$
 $P_O <_{CF} cone \mapsto: (\mathfrak{J}(Arr)) (\lambda u. \mathfrak{F}(ObjMap)(\mathfrak{J}(Cod)(u))) \mathfrak{C} :$
 $:_C(\mathfrak{J}(Arr)) \mapsto_{C\alpha} \mathfrak{C}$
 unfolding *the-cat-discrete-components(1)*

proof

(
 intro
 tm-cf-discrete.tm-cf-discrete-ntcf-obj-prod-base-is-cat-cone
 tm-cf-discrete
)

fix f assume $f \in_o \mathfrak{J}(Arr)$

then obtain $a b$ where $f : a \mapsto_{\mathfrak{J}} b$ by *auto*

then show $\pi_O(NTMap)(\mathfrak{J}(Cod)(f)) : P_O \mapsto_{\mathfrak{C}} \mathfrak{F}(ObjMap)(\mathfrak{J}(Cod)(f))$

by

(
 cs-concl cs-shallow
 cs-simp:
 the-cat-discrete-components(1) cat-discrete-cs-simps cat-cs-simps
 cs-intro: cat-cs-intros
)

qed (*intro P_O*)

from π_A .*cat-obj-prod-unique-cone* [*OF π_O'*] obtain f'

where $f': f' : P_O \mapsto_{\mathfrak{C}} P_A$

and π_O' -*NTMap-app*:

$\wedge j. j \in_o \mathfrak{J}(Arr) \implies ?\pi_O'(NTMap)(j) = \pi_A(NTMap)(j) \circ_A \mathfrak{C} f'$

and *unique-f'*:

\llbracket
 $f'' : P_O \mapsto_{\mathfrak{C}} P_A;$
 $\wedge j. j \in_o \mathfrak{J}(Arr) \implies ?\pi_O'(NTMap)(j) = \pi_A(NTMap)(j) \circ_A \mathfrak{C} f''$
 $\rrbracket \implies f'' = f'$

for f''

by *metis*

have π_O -*NTMap-app-Cod*:

$\pi_O(NTMap)(b) = \pi_A(NTMap)(f) \circ_A \mathfrak{C} f'$ if $f : a \mapsto_{\mathfrak{J}} b$ for $f a b$

proof-

from *that* have $f \in_o \mathfrak{J}(Arr)$ by *auto*

from π_O' -NTMap-app[OF this] **that show** ?thesis
by
 (

- cs-prems **cs-shallow**
- cs-simp:** cat-cs-simps the-cat-discrete-components(1)
- cs-intro:** cat-cs-intros

)

from this[symmetric] **have** π_A -NTMap-Comp-app:
 $\pi_A(\text{NTMap})(f) \circ_{A\mathfrak{C}} (f' \circ_{A\mathfrak{C}} q) = \pi_O(\text{NTMap})(b) \circ_{A\mathfrak{C}} q$
if $f : a \mapsto_{\mathfrak{J}} b$ **and** $q : c \mapsto_{\mathfrak{C}} P_O$ **for** $q f a b c$
using that f'
by (intro \mathfrak{F} .HomCod.cat-assoc-helper)
 (

- cs-concl **cs-shallow**
- cs-simp:**
 cat-cs-simps cat-discrete-cs-simps the-cat-discrete-components(1)
- cs-intro:** cat-cs-intros

)+

let $?g = \langle \lambda u. \mathfrak{F}(\text{ArrMap})(u) \circ_{A\mathfrak{C}} \pi_O(\text{NTMap})(\mathfrak{J}(\text{Dom})(u)) \rangle$
let $? \pi_O'' = \langle \text{ntcf-obj-prod-base } \mathfrak{C} (:_C (\mathfrak{J}(\text{Arr}))(\text{Obj})) ?F P_O ?g \rangle$

have $\pi_O'' : ? \pi_O'' : P_O <_{CF.cone} \text{--} : (\mathfrak{J}(\text{Arr})) ?L \mathfrak{C} : :_C (\mathfrak{J}(\text{Arr})) \mapsto_{C\alpha} \mathfrak{C}$
unfolding the-cat-discrete-components(1)

proof

(

- intro
- tm-cf-discrete.tm-cf-discrete-ntcf-obj-prod-base-is-cat-cone
- tm-cf-discrete

)

show $\mathfrak{F}(\text{ArrMap})(f) \circ_{A\mathfrak{C}} \pi_O(\text{NTMap})(\mathfrak{J}(\text{Dom})(f)) : P_O \mapsto_{\mathfrak{C}} \mathfrak{F}(\text{ObjMap})(\mathfrak{J}(\text{Cod})(f))$
if $f \in_{\circ} \mathfrak{J}(\text{Arr})$ **for** f

proof-

from that **obtain** $a b$ **where** $f : a \mapsto_{\mathfrak{J}} b$ **by** auto
then show ?thesis

by

(

- cs-concl
- cs-simp:**
 cat-cs-simps cat-discrete-cs-simps
 the-cat-discrete-components(1)
- cs-intro:** cat-cs-intros

)

qed

qed (intro P_O)

from π_A .cat-obj-prod-unique-cone'[OF π_O''] **obtain** g'

where $g' : g' : P_O \mapsto_{\mathfrak{C}} P_A$

and π_O'' -NTMap-app:

$\wedge j. j \in_{\circ} \mathfrak{J}(\text{Arr}) \implies ? \pi_O''(\text{NTMap})(j) = \pi_A(\text{NTMap})(j) \circ_{A\mathfrak{C}} g'$

and unique- g' :

\llbracket
 $g'' : P_O \mapsto_{\mathfrak{C}} P_A;$
 $\wedge j. j \in_{\circ} \mathfrak{J}(\text{Arr}) \implies ? \pi_O''(\text{NTMap})(j) = \pi_A(\text{NTMap})(j) \circ_{A\mathfrak{C}} g''$
 $\rrbracket \implies g'' = g'$

for g''

by (*metis (lifting)*)

have $\mathfrak{F}(\downarrow ArrMap)(\downarrow f) \circ_{A\mathfrak{C}} \pi_O(\downarrow NTMap)(\downarrow a) = \pi_A(\downarrow NTMap)(\downarrow f) \circ_{A\mathfrak{C}} g'$
 if $f : a \mapsto_{\mathfrak{J}} b$ for $f a b$

proof-

from that have $f \in_{\circ} \mathfrak{J}(\downarrow Arr)$ by *auto*

from π_O'' -*NTMap-app*[*OF this*] that show *?thesis*

by

(

cs-prems cs-shallow

cs-simp: *cat-cs-simps the-cat-discrete-components(1)*

cs-intro: *cat-cs-intros*

)

qed

then have π_O -*NTMap-app-Dom*:

$\mathfrak{F}(\downarrow ArrMap)(\downarrow f) \circ_{A\mathfrak{C}} (\pi_O(\downarrow NTMap)(\downarrow a) \circ_{A\mathfrak{C}} q) =$
 $(\pi_A(\downarrow NTMap)(\downarrow f) \circ_{A\mathfrak{C}} g') \circ_{A\mathfrak{C}} q$

if $f : a \mapsto_{\mathfrak{J}} b$ and $q : c \mapsto_{\mathfrak{C}} P_O$ for $q f a b c$

using that g'

by (*intro $\mathfrak{F}.HomCod.cat-assoc-helper$*)

(

cs-concl

cs-simp:

cat-cs-simps cat-discrete-cs-simps the-cat-discrete-components(1)

cs-intro: *cat-cs-intros*

)

from *assms(2)*[*OF f' g'*] obtain $E \varepsilon$ where ε :

$\varepsilon : E <_{CF.eq} (P_O, P_A, g', f') : \uparrow\uparrow_C \mapsto_{C\alpha} \mathfrak{C}$

by *clarsimp*

interpret ε : *is-cat-equalizer-2* $\alpha P_O P_A g' f' \mathfrak{C} E \varepsilon$ by (*rule ε*)

define μ where $\mu =$

$[(\lambda i \in_{\circ} \mathfrak{J}(\downarrow Obj)). \pi_O(\downarrow NTMap)(\downarrow i) \circ_{A\mathfrak{C}} \varepsilon(\downarrow NTMap)(\downarrow \mathbf{a}_{PL2})], cf-const \mathfrak{J} \mathfrak{C} E, \mathfrak{F}, \mathfrak{J}, \mathfrak{C}]$.

have μ -*components*:

$\mu(\downarrow NTMap) = (\lambda i \in_{\circ} \mathfrak{J}(\downarrow Obj)). \pi_O(\downarrow NTMap)(\downarrow i) \circ_{A\mathfrak{C}} \varepsilon(\downarrow NTMap)(\downarrow \mathbf{a}_{PL2})$

$\mu(\downarrow NTDom) = cf-const \mathfrak{J} \mathfrak{C} E$

$\mu(\downarrow NTCod) = \mathfrak{F}$

$\mu(\downarrow NTDGDom) = \mathfrak{J}$

$\mu(\downarrow NTDGCod) = \mathfrak{C}$

unfolding μ -*def nt-field-simps* by (*simp-all add: nat-omega-simps*)

have [*cat-cs-simps*]:

$\mu(\downarrow NTMap)(\downarrow i) = \pi_O(\downarrow NTMap)(\downarrow i) \circ_{A\mathfrak{C}} \varepsilon(\downarrow NTMap)(\downarrow \mathbf{a}_{PL2})$ if $i \in_{\circ} \mathfrak{J}(\downarrow Obj)$

for i

using that **unfolding** μ -*components* by *simp*

have $\mu : E <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$

proof(*intro is-cat-limitI*)

show $\mu : \mu : E <_{CF.cone} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$

proof(*intro is-cat-coneI is-tm-ntcfI' is-ntcfI'*)

show *vfsequence* μ **unfolding** μ -*def* by *simp*

show *vcard* $\mu = 5_N$ **unfolding** μ -*def* by (*simp add: nat-omega-simps*)

show *cf-const* $\mathfrak{J} \mathfrak{C} E : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$

by (*cs-concl cs-intro: cat-cs-intros cat-lim-cs-intros*)

```

show  $\mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$  by (cs-concl cs-shallow cs-intro: cat-cs-intros)
show  $\mu(\text{NTMap})(\downarrow a) : \text{cf-const } \mathfrak{J} \mathfrak{C} E(\text{ObjMap})(\downarrow a) \mapsto_{\mathfrak{C}} \mathfrak{F}(\text{ObjMap})(\downarrow a)$ 
  if  $a \in_{\circ} \mathfrak{J}(\text{Obj})$  for  $a$ 
  using that
  by
  (
    cs-concl
    cs-simp:
      cat-cs-simps
      cat-discrete-cs-simps
      cat-parallel-cs-simps
      the-cat-discrete-components(1)
    cs-intro: cat-cs-intros cat-lim-cs-intros cat-parallel-cs-intros
  )
show
 $\mu(\text{NTMap})(\downarrow b) \circ_{A\mathfrak{C}} \text{cf-const } \mathfrak{J} \mathfrak{C} E(\text{ArrMap})(\downarrow f) =$ 
 $\mathfrak{F}(\text{ArrMap})(\downarrow f) \circ_{A\mathfrak{C}} \mu(\text{NTMap})(\downarrow a)$ 
if  $f : a \mapsto_{\mathfrak{J}} b$  for  $a b f$ 
using that  $\varepsilon g' f'$ 
by
  (
    cs-concl
    cs-simp:
      cat-parallel-cs-simps
      cat-cs-simps
      the-cat-discrete-components(1)
       $\pi_{O\text{-NTMap-app-Cod}}$ 
       $\pi_{O\text{-NTMap-app-Dom}}$ 
       $\varepsilon.\text{cat-eq-2-Comp-eq}(1)$ 
    cs-intro: cat-lim-cs-intros cat-cs-intros cat-parallel-cs-intros
  )

qed (auto simp:  $\mu$ -components cat-cs-intros)

interpret  $\mu$ : is-cat-cone  $\alpha E \mathfrak{J} \mathfrak{C} \mathfrak{F} \mu$  by (rule  $\mu$ )

show  $\exists ! f'. f' : r' \mapsto_{\mathfrak{C}} E \wedge u' = \mu \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{C} f'$ 
if  $u' : r' <_{CF.\text{cone}} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$  for  $u' r'$ 
proof-

interpret  $u'$ : is-cat-cone  $\alpha r' \mathfrak{J} \mathfrak{C} \mathfrak{F} u'$  by (rule that)

let  $?u' = \langle \lambda j. u'(\text{NTMap})(\downarrow j) \rangle$ 
let  $?\pi' = \langle \text{ntcf-obj-prod-base } \mathfrak{C} (\mathfrak{J}(\text{Obj})) ?R r' ?u' \rangle$ 

have  $\pi'\text{-NTMap-app}$ :  $? \pi'(\text{NTMap})(\downarrow j) = u'(\text{NTMap})(\downarrow j)$  if  $j \in_{\circ} \mathfrak{J}(\text{Obj})$  for  $j$ 
using that
unfolding ntcf-obj-prod-base-components the-cat-discrete-components
by auto

have  $\pi'$ :  $? \pi' : r' <_{CF.\text{cone}} \text{--} : (\mathfrak{J}(\text{Obj})) ?R \mathfrak{C} \text{--} : :_C (\mathfrak{J}(\text{Obj})) \mapsto_{C\alpha} \mathfrak{C}$ 
unfolding the-cat-discrete-components(1)
proof(intro tm-cf-discrete.tm-cf-discrete-ntcf-obj-prod-base-is-cat-cone)
show tm-cf-discrete  $\alpha (\mathfrak{J}(\text{Obj})) ?R \mathfrak{C}$ 
proof(intro tm-cf-discreteI)
show  $\mathfrak{F}(\text{ObjMap})(\downarrow i) \in_{\circ} \mathfrak{C}(\text{Obj})$  if  $i \in_{\circ} \mathfrak{J}(\text{Obj})$  for  $i$ 
by (cs-concl cs-simp: cat-cs-simps cs-intro: that cat-cs-intros)
show category  $\alpha \mathfrak{C}$  by (auto intro: cat-cs-intros)

```

from $\mathfrak{F}.tm\text{-}cf\text{-}ObjMap\text{-}in\text{-}Vset$ **show** $(\lambda x \in \circ \mathfrak{J}(\mathcal{O}bj). \mathfrak{F}(\mathcal{O}bjMap)(x)) \in \circ Vset \alpha$
by $(auto\ simp: \mathfrak{F}.cf\text{-}ObjMap\text{-}vdomain)$
show $(\lambda i \in \circ \mathfrak{J}(\mathcal{O}bj). \mathfrak{C}(\mathcal{C}Id)(\mathfrak{F}(\mathcal{O}bjMap)(i))) \in \circ Vset \alpha$
proof $(rule\ vrelation.vrelation\text{-}Limit\text{-}in\text{-}VsetI)$
have $\mathcal{R}_\circ (\lambda i \in \circ \mathfrak{J}(\mathcal{O}bj). \mathfrak{C}(\mathcal{C}Id)(\mathfrak{F}(\mathcal{O}bjMap)(i))) \subseteq \circ \mathcal{R}_\circ (\mathfrak{F}(\mathcal{A}rrMap))$
proof $(intro\ vsubsetI)$
fix x **assume** $x \in \circ \mathcal{R}_\circ (\lambda i \in \circ \mathfrak{J}(\mathcal{O}bj). \mathfrak{C}(\mathcal{C}Id)(\mathfrak{F}(\mathcal{O}bjMap)(i)))$
then obtain i **where** $i \in \circ \mathfrak{J}(\mathcal{O}bj)$
and $x\text{-}def: x = \mathfrak{C}(\mathcal{C}Id)(\mathfrak{F}(\mathcal{O}bjMap)(i))$
by $auto$
from i **have** $x = \mathfrak{F}(\mathcal{A}rrMap)(\mathfrak{J}(\mathcal{C}Id)(i))$
by $(simp\ add: x\text{-}def\ \mathfrak{F}.cf\text{-}ObjMap\text{-}CId)$
moreover from i **have** $\mathfrak{J}(\mathcal{C}Id)(i) \in \circ \mathcal{D}_\circ (\mathfrak{F}(\mathcal{A}rrMap))$
by
(

 $cs\text{-}concl\ cs\text{-}shallow$
 cs-simp: $cat\text{-}cs\text{-}simps$ **cs-intro:** $cat\text{-}cs\text{-}intros$

)

ultimately show $x \in \circ \mathcal{R}_\circ (\mathfrak{F}(\mathcal{A}rrMap))$
by $(auto\ intro: \mathfrak{F}.\mathcal{A}rrMap.vsv\text{-}vimageI2)$
qed
then show $\mathcal{R}_\circ (\lambda i \in \circ \mathfrak{J}(\mathcal{O}bj). \mathfrak{C}(\mathcal{C}Id)(\mathfrak{F}(\mathcal{O}bjMap)(i))) \in \circ Vset \alpha$
by
(

 $auto\ simp:$
 $\mathfrak{F}.tm\text{-}cf\text{-}\mathcal{A}rrMap\text{-}in\text{-}Vset\ vrange\text{-}in\text{-}VsetI\ vsubset\text{-}in\text{-}VsetI$

)

qed $(auto\ intro: \mathfrak{F}.HomDom.tiny\text{-}cat\text{-}Obj\text{-}in\text{-}Vset)$
qed
show $u'(\mathcal{N}TMap)(j) : r' \mapsto_{\mathfrak{C}} \mathfrak{F}(\mathcal{O}bjMap)(j)$ **if** $j \in \circ \mathfrak{J}(\mathcal{O}bj)$ **for** j
using that
by $(cs\text{-}concl\ cs\text{-}shallow\ cs\text{-}simp: cat\text{-}cs\text{-}simps\ cs\text{-}intro: cat\text{-}cs\text{-}intros)$
qed $(auto\ simp: cat\text{-}cs\text{-}intros)$

from $\pi_{\mathcal{O}}.cat\text{-}obj\text{-}prod\text{-}unique\text{-}cone'[OF\ this]$ **obtain** h'
where $h': h' : r' \mapsto_{\mathfrak{C}} P_{\mathcal{O}}$
and $\pi'\text{-}\mathcal{N}TMap\text{-}app'$:
 $\wedge j. j \in \circ (\mathfrak{J}(\mathcal{O}bj)) \implies ?\pi'(\mathcal{N}TMap)(j) = \pi_{\mathcal{O}}(\mathcal{N}TMap)(j) \circ_{A\mathfrak{C}} h'$
and $unique\text{-}h': \wedge h''$.
[[

 $h'' : r' \mapsto_{\mathfrak{C}} P_{\mathcal{O}};$
 $\wedge j. j \in \circ (\mathfrak{J}(\mathcal{O}bj)) \implies ?\pi'(\mathcal{N}TMap)(j) = \pi_{\mathcal{O}}(\mathcal{N}TMap)(j) \circ_{A\mathfrak{C}} h''$

]] $\implies h'' = h'$
by $metis$

interpret π' :
 $is\text{-}cat\text{-}cone\ \alpha\ r' \langle :_C (\mathfrak{J}(\mathcal{O}bj)) \rangle \mathfrak{C} \langle \dashv \dashv : (\mathfrak{J}(\mathcal{O}bj)) (app (\mathfrak{F}(\mathcal{O}bjMap))) \rangle \mathfrak{C} \rangle ?\pi'$
by $(rule\ \pi')$

let $?u'' = \langle \lambda u. u'(\mathcal{N}TMap)(\mathfrak{J}(\mathcal{C}od)(u)) \rangle$
let $?\pi'' = \langle ntcf\text{-}obj\text{-}prod\text{-}base\ \mathfrak{C} (\mathfrak{J}(\mathcal{A}rr)) \ ?L\ r'\ ?u'' \rangle$

have $\pi''\text{-}\mathcal{N}TMap\text{-}app: ?\pi''(\mathcal{N}TMap)(f) = u'(\mathcal{N}TMap)(b)$
if $f : a \mapsto_{\mathfrak{J}} b$ **for** $f\ a\ b$
using that
unfolding $ntcf\text{-}obj\text{-}prod\text{-}base\text{-}components\ the\text{-}cat\text{-}discrete\text{-}components$
by
(

cs-concl cs-shallow
cs-simp: *V-cs-simps cat-cs-simps* **cs-intro:** *cat-cs-intros*
)

have $\pi'' : ?\pi'' : r' \langle_{CF.cone} \rightarrow : (\mathfrak{J}(\mathfrak{A}rr)) \ ?L \ \mathfrak{C} : :_C (\mathfrak{J}(\mathfrak{A}rr)) \mapsto_{C\alpha} \ \mathfrak{C}$
unfolding *the-cat-discrete-components(1)*
proof
(

 intro
 tm-cf-discrete.tm-cf-discrete-ntcf-obj-prod-base-is-cat-cone
 tm-cf-discrete
)

fix f **assume** $f \in_{\circ} \mathfrak{J}(\mathfrak{A}rr)$
then obtain $a \ b$ **where** $f : a \mapsto_{\mathfrak{J}} b$ **by** *auto*
then show $u'(\mathfrak{N}TMap)(\mathfrak{J}(\mathfrak{C}od)(f)) : r' \mapsto_{\mathfrak{C}} \mathfrak{F}(\mathfrak{O}bjMap)(\mathfrak{J}(\mathfrak{C}od)(f))$
by
(

 cs-concl cs-shallow
 cs-simp: *cat-cs-simps* **cs-intro:** *cat-cs-intros*
)

qed (*simp add: cat-cs-intros*)

from π_A .*cat-obj-prod-unique-cone'[OF this]* **obtain** h''
where $h'' : h'' : r' \mapsto_{\mathfrak{C}} P_A$
and π'' -*NTMap-app'*:
 $\wedge j. j \in_{\circ} \mathfrak{J}(\mathfrak{A}rr) \implies ?\pi''(\mathfrak{N}TMap)(j) = \pi_A(\mathfrak{N}TMap)(j) \circ_{A\mathfrak{C}} h''$
and *unique-h''*: $\wedge h'''.$
 \llbracket
 $h''' : r' \mapsto_{\mathfrak{C}} P_A;$
 $\wedge j. j \in_{\circ} \mathfrak{J}(\mathfrak{A}rr) \implies ?\pi''(\mathfrak{N}TMap)(j) = \pi_A(\mathfrak{N}TMap)(j) \circ_{A\mathfrak{C}} h'''$
 $\rrbracket \implies h''' = h''$
by *metis*

interpret π'' : *is-cat-cone* $\alpha \ r' \langle :_C (\mathfrak{J}(\mathfrak{A}rr)) \rangle \ \mathfrak{C} \langle : \rightarrow : (\mathfrak{J}(\mathfrak{A}rr)) \ ?L \ \mathfrak{C} \rangle \ ?\pi''$
by (*rule* π'')

have $g'h'-f'h' : g' \circ_{A\mathfrak{C}} h' = f' \circ_{A\mathfrak{C}} h'$
proof-

from $g' \ h'$ **have** $g'h' : g' \circ_{A\mathfrak{C}} h' : r' \mapsto_{\mathfrak{C}} P_A$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)
from $f' \ h'$ **have** $f'h' : f' \circ_{A\mathfrak{C}} h' : r' \mapsto_{\mathfrak{C}} P_A$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

have $?\pi''(\mathfrak{N}TMap)(f) = \pi_A(\mathfrak{N}TMap)(f) \circ_{A\mathfrak{C}} (g' \circ_{A\mathfrak{C}} h')$
if $f \in_{\circ} \mathfrak{J}(\mathfrak{A}rr)$ **for** f
proof-
from *that* **obtain** $a \ b$ **where** $f : a \mapsto_{\mathfrak{J}} b$ **by** *auto*
then have $?\pi''(\mathfrak{N}TMap)(f) = u'(\mathfrak{N}TMap)(b)$
by (*cs-concl cs-simp: π'' -NTMap-app cat-cs-simps*)
also from f **have** $\dots = \mathfrak{F}(\mathfrak{A}rrMap)(f) \circ_{A\mathfrak{C}} ?\pi''(\mathfrak{N}TMap)(a)$
by
(

 cs-concl
 cs-simp: π'' -*NTMap-app cat-cs-simps cat-lim-cs-simps*
 cs-intro: *cat-cs-intros*
)

also from $f \ g' \ h'$ **have** $\dots = \pi_A(\mathfrak{N}TMap)(f) \circ_{A\mathfrak{C}} (g' \circ_{A\mathfrak{C}} h')$

by
 (

 cs-concl

 cs-simp:

 cat-cs-simps

 cat-discrete-cs-simps

 the-cat-discrete-components(1)

 π' -*NTMap-app'*

 π_O -*NTMap-app-Dom*

 cs-intro: *cat-cs-intros*

)

 finally show *?thesis* by *simp*

 qed

from *unique-h''[OF g'h' this, simplified]* have $g'h'-h''$:

 $g' \circ_{A\mathfrak{C}} h' = h''$.

 have $?\pi''(\text{NTMap})(f) = \pi_A(\text{NTMap})(f) \circ_{A\mathfrak{C}} (f' \circ_{A\mathfrak{C}} h')$

 if $f \in_o \mathfrak{J}(\text{Arr})$ for f

 proof-

 from that obtain $a \ b$ where $f: f : a \mapsto_{\mathfrak{J}} b$ by *auto*

 then have $?\pi''(\text{NTMap})(f) = u'(\text{NTMap})(b)$

 by (*cs-concl cs-simp:* π'' -*NTMap-app cat-cs-simps*)

 also from f have $\dots = ?\pi'(\text{NTMap})(b)$

 by

 (

 cs-concl cs-shallow

 cs-simp: π' -*NTMap-app cs-intro:* *cat-cs-intros*

)

 also from f have $\dots = \pi_O(\text{NTMap})(b) \circ_{A\mathfrak{C}} h'$

 by

 (

 cs-concl cs-shallow

 cs-simp: π' -*NTMap-app' cs-intro:* *cat-cs-intros*

)

 also from $f \ g' \ h'$ have $\dots = (\pi_A(\text{NTMap})(f) \circ_{A\mathfrak{C}} f') \circ_{A\mathfrak{C}} h'$

 by

 (

 cs-concl cs-shallow

 cs-simp: π_O -*NTMap-app-Cod cs-intro:* *cat-cs-intros*

)

 also from that $f' \ h'$ have $\dots = \pi_A(\text{NTMap})(f) \circ_{A\mathfrak{C}} (f' \circ_{A\mathfrak{C}} h')$

 by

 (

 cs-concl cs-shallow

 cs-simp: *cat-cs-simps the-cat-discrete-components(1)*

 cs-intro: *cat-cs-intros*

)

 finally show *?thesis* by *simp*

 qed

from *unique-h''[OF f'h' this, simplified]* have $f'h'-h''$:

 $f' \circ_{A\mathfrak{C}} h' = h''$.

 from $g'h'-h'' \ f'h'-h''$ show *?thesis* by *simp*

 qed

let $?II = \langle \uparrow\uparrow_C \mathfrak{a}_{PL2} \ \mathfrak{b}_{PL2} \ \mathfrak{g}_{PL} \ \mathfrak{f}_{PL} \rangle$

 and $?II-II = \langle \uparrow\uparrow \rightarrow \uparrow\uparrow_{CF} \ \mathfrak{C} \ \mathfrak{a}_{PL2} \ \mathfrak{b}_{PL2} \ \mathfrak{g}_{PL} \ \mathfrak{f}_{PL} \ P_O \ P_A \ g' \ f' \rangle$

define ε' where $\varepsilon' =$

[
 $(\lambda f \in_{\circ} ?II(\text{Obj}). (f = \mathbf{a}_{PL2} \ ? h' : (f' \circ_{A\mathfrak{C}} h')))$,
cf-const $?II \ \mathfrak{C} \ r'$,
 $?II-II$,
 $?II$,
 \mathfrak{C}
]_o

have ε' -components:

$\varepsilon'(\text{NTMap}) = (\lambda f \in_{\circ} ?II(\text{Obj}). (f = \mathbf{a}_{PL2} \ ? h' : (f' \circ_{A\mathfrak{C}} h')))$
 $\varepsilon'(\text{NTDom}) = \text{cf-const } ?II \ \mathfrak{C} \ r'$
 $\varepsilon'(\text{NTCod}) = ?II-II$
 $\varepsilon'(\text{NTDGDom}) = ?II$
 $\varepsilon'(\text{NTDGCod}) = \mathfrak{C}$
unfolding ε' -def *nt-field-simps* **by** (*simp-all add: nat-omega-simps*)

have ε' -NTMap-app-I2: $\varepsilon'(\text{NTMap})(x) = h'$ if $x = \mathbf{a}_{PL2}$ for x

proof-

have $x \in_{\circ} ?II(\text{Obj})$
unfolding that by (*cs-concl cs-intro: cat-parallel-cs-intros*)
then show *?thesis* **unfolding** ε' -components **that by** *simp*
qed

have ε' -NTMap-app-sI2: $\varepsilon'(\text{NTMap})(x) = f' \circ_{A\mathfrak{C}} h'$ if $x = \mathbf{b}_{PL2}$ for x

proof-

have $x \in_{\circ} ?II(\text{Obj})$
unfolding that by (*cs-concl cs-shallow cs-intro: cat-parallel-cs-intros*)
with ε .*cat-parallel-ab* **show** *?thesis*
unfolding ε' -components **by** (*cs-concl cs-simp: V-cs-simps that*)
qed

interpret *par: cf-parallel-2* $\alpha_{PL2} \ \mathbf{b}_{PL2} \ \mathfrak{g}_{PL} \ \mathfrak{f}_{PL} \ P_O \ P_A \ g' \ f' \ \mathfrak{C}$

by (*intro cf-parallel-2I cat-parallel-2I*)
(simp-all add: cat-cs-intros cat-parallel-cs-intros)

have $\varepsilon' : r' <_{CF.cone} ?II-II : ?II \mapsto_{C\alpha} \mathfrak{C}$

proof(*intro is-cat-coneI is-tm-ntcfI' is-ntcfI'*)

show *ufsequence* ε' **unfolding** ε' -def **by** *auto*
show *vcard* $\varepsilon' = 5_{\mathbb{N}}$ **unfolding** ε' -def **by** (*simp add: nat-omega-simps*)
from h' **show** *cf-const* ($?II$) $\mathfrak{C} \ r' : ?II \mapsto_{C\alpha} \mathfrak{C}$
by (*cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)
show $?II-II : ?II \mapsto_{C\alpha} \mathfrak{C}$

by

(
cs-concl cs-shallow
cs-simp: *cat-parallel-cs-simps cs-intro: cat-cs-intros*
)

from h' **show** $\varepsilon'(\text{NTMap})(a) :$

cf-const $?II \ \mathfrak{C} \ r'(\text{ObjMap})(a) \mapsto_{\mathfrak{C}} ?II-II(\text{ObjMap})(a)$

if $a \in_{\circ} ?II(\text{Obj})$ **for** a

using that

by (*elim the-cat-parallel-2-ObjE; simp only:*)

(

cs-concl
cs-simp:
 ε' -NTMap-app-I2 ε' -NTMap-app-sI2
cat-cs-simps cat-parallel-cs-simps
cs-intro: *cat-cs-intros cat-parallel-cs-intros*

)

from $h' f' g' h' f' h'$ **show**

$$\varepsilon'(\text{NTMap})(b) \circ_{A\mathfrak{C}} \text{cf-const } ?II \mathfrak{C} r'(\text{ArrMap})(f) =$$

$$?II\text{-II}(\text{ArrMap})(f) \circ_{A\mathfrak{C}} \varepsilon'(\text{NTMap})(a)$$

if $f : a \mapsto_{?II} b$ **for** $a b f$

using *that*

by (*elim* ε .*the-cat-parallel-2-is-arrE*; *simp only*.)

(

cs-concl

cs-intro: *cat-cs-intros cat-parallel-cs-intros*

cs-simp:

cat-cs-simps

cat-parallel-cs-simps

ε' -*NTMap-app-I2*

ε' -*NTMap-app-sI2*

)+

qed

(

simp add: ε' -*components* |

cs-concl

cs-simp: *cat-cs-simps*

cs-intro: *cat-lim-cs-intros cat-cs-intros cat-small-cs-intros*

)+

from ε .*cat-eq-2-unique-cone*[*OF this*] **obtain** t'

where $t' : t' : r' \mapsto_{\mathfrak{C}} E$

and ε' -*NTMap-app:* $\varepsilon'(\text{NTMap})(\mathfrak{a}_{PL2}) = \varepsilon(\text{NTMap})(\mathfrak{a}_{PL2}) \circ_{A\mathfrak{C}} t'$

and *unique-t'*:

$$\llbracket t'' : r' \mapsto_{\mathfrak{C}} E; \varepsilon'(\text{NTMap})(\mathfrak{a}_{PL2}) = \varepsilon(\text{NTMap})(\mathfrak{a}_{PL2}) \circ_{A\mathfrak{C}} t'' \rrbracket \implies$$

$$t'' = t'$$

for t''

by *metis*

show $\exists ! f'. f' : r' \mapsto_{\mathfrak{C}} E \wedge u' = \mu \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{C} f'$

proof(*intro exII conjI*; (*elim conjE*)?, (*rule t'*)?)

show [*symmetric, cat-cs-simps*]: $u' = \mu \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{C} t'$

proof(*rule ntcf-eqI*[*OF u'.is-ntcf-axioms*])

from t' **show**

$\mu \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{C} t' : \text{cf-const } \mathfrak{J} \mathfrak{C} r' \mapsto_{CF} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$

by (*cs-concl* **cs-shallow** **cs-simp:** *cat-cs-simps* **cs-intro:** *cat-cs-intros*)

show $u'(\text{NTMap}) = (\mu \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{C} t')(\text{NTMap})$

proof(*rule vsv-eqI*)

show *vsv* ($(\mu \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{C} t')(\text{NTMap})$)

by (*cs-concl* **cs-simp:** *cat-cs-simps* **cs-intro:** *cat-cs-intros*)

from t' **show**

$\mathcal{D}_\circ (u'(\text{NTMap})) = \mathcal{D}_\circ ((\mu \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{C} t')(\text{NTMap}))$

by

(

cs-concl **cs-shallow**

cs-simp: *cat-cs-simps* **cs-intro:** *cat-cs-intros*

)

show $u'(\text{NTMap})(a) = (\mu \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{C} t')(\text{NTMap})(a)$

if $a \in_\circ \mathcal{D}_\circ (u'(\text{NTMap}))$ **for** a

proof-

from *that* **have** $a \in_\circ \mathfrak{J}(\text{Obj})$

by (*cs-prems* **cs-shallow** **cs-simp:** *cat-cs-simps*)

with t' **show** $u'(\text{NTMap})(a) = (\mu \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{C} t')(\text{NTMap})(a)$

by

(

```

    cs-concl
    cs-simp:
      cat-cs-simps
       $\pi'$ -NTMap-app
      cat-parallel-cs-simps
      the-cat-discrete-components(1)
       $\varepsilon'$ -NTMap-app[symmetric]
       $\varepsilon'$ -NTMap-app-I2
       $\pi'$ -NTMap-app'[symmetric]
    cs-intro: cat-cs-intros cat-parallel-cs-intros
  )
  qed
  qed auto
  qed simp-all

  fix  $t''$  assume  $\text{prems}' : t'' : r' \mapsto_{\mathfrak{C}} E \ u' = \mu \cdot_{NTCF} \text{ntcf-const} \ \mathfrak{J} \ \mathfrak{C} \ t''$ 
  then have  $u'$ -NTMap-app-x:
     $u'(\text{NTMap})(x) = (\mu \cdot_{NTCF} \text{ntcf-const} \ \mathfrak{J} \ \mathfrak{C} \ t'')(\text{NTMap})(x)$ 
  for  $x$ 
  by simp
  have  $?\pi'$ ( $\text{NTMap})(j) = \pi_O(\text{NTMap})(j) \circ_{A\mathfrak{C}} (\varepsilon(\text{NTMap})(\mathfrak{a}_{PL2}) \circ_{A\mathfrak{C}} t'')$ 
  if  $j \in_{\circ} \mathfrak{J}(\text{Obj})$  for  $j$ 
  using  $u'$ -NTMap-app-x[of  $j$ ]  $\text{prems}'(1)$  that
  by
  (
    cs-prems
    cs-simp:
      cat-cs-simps
      cat-discrete-cs-simps
      cat-parallel-cs-simps
      the-cat-discrete-components(1)
    cs-intro: cat-cs-intros cat-parallel-cs-intros
  )
  (simp add:  $\pi'$ -NTMap-app[OF that, symmetric])
  moreover from  $\text{prems}'(1)$  have  $\varepsilon(\text{NTMap})(\mathfrak{a}_{PL2}) \circ_{A\mathfrak{C}} t'' : r' \mapsto_{\mathfrak{C}} P_O$ 
  by
  (
    cs-concl
    cs-simp: cat-cs-simps cat-parallel-cs-simps
    cs-intro: cat-cs-intros cat-parallel-cs-intros
  )
  ultimately have [ $\text{cat-cs-simps}$ ]:  $\varepsilon(\text{NTMap})(\mathfrak{a}_{PL2}) \circ_{A\mathfrak{C}} t'' = h'$ 
  by (intro unique-h') simp
  show  $t'' = t'$ 
  by (rule unique-t', intro  $\text{prems}'(1)$ )
  (cs-concl cs-shallow cs-simp:  $\varepsilon'$ -NTMap-app-I2 cat-cs-simps)
  qed
  qed

```

qed

then show *?thesis using that by clarsimp*

qed

lemma *cat-colimit-of-cat-prod-obj-and-cat-coequalizer:*

— See Theorem 1 in Chapter V-2 in [9].

assumes $\mathfrak{F} : \mathfrak{J} \mapsto_{\mapsto} C.tmA \ \mathfrak{C}$

and $\wedge a \ b \ g \ f. \llbracket f : b \mapsto_{\mathcal{C}} a; g : b \mapsto_{\mathcal{C}} a \rrbracket \implies$
 $\exists E \ \varepsilon. \ \varepsilon : (a, b, g, f) >_{CF.coeq} E : \uparrow_C \mapsto_{C\alpha} \mathcal{C}$
and $\wedge A. \text{tm-cf-discrete } \alpha \ (\mathfrak{J}(\text{Obj})) \ A \ \mathcal{C} \implies$
 $\exists P \ \pi. \ \pi : A >_{CF.\Pi} P : \mathfrak{J}(\text{Obj}) \mapsto_{C\alpha} \mathcal{C}$
and $\wedge A. \text{tm-cf-discrete } \alpha \ (\mathfrak{J}(\text{Arr})) \ A \ \mathcal{C} \implies$
 $\exists P \ \pi. \ \pi : A >_{CF.\Pi} P : \mathfrak{J}(\text{Arr}) \mapsto_{C\alpha} \mathcal{C}$
obtains $r \ u$ **where** $u : \mathfrak{F} >_{CF.colim} r : \mathfrak{J} \mapsto_{C\alpha} \mathcal{C}$
proof-
interpret \mathfrak{F} : *is-tm-functor* $\alpha \ \mathfrak{J} \ \mathcal{C} \ \mathfrak{F}$ **by** (rule *assms(1)*)
have $\exists E \ \varepsilon. \ \varepsilon : E <_{CF.eq} (a, b, g, f) : \uparrow_C \mapsto_{C\alpha} \text{op-cat } \mathcal{C}$
if $f : b \mapsto_{\mathcal{C}} a \ g : b \mapsto_{\mathcal{C}} a$ **for** $a \ b \ g \ f$
proof-
from *assms(2)[OF that(1,2)]* **obtain** $E \ \varepsilon$
where $\varepsilon : \varepsilon : (a, b, g, f) >_{CF.coeq} E : \uparrow_C \mapsto_{C\alpha} \mathcal{C}$
by *clarsimp*
interpret ε : *is-cat-coequalizer-2* $\alpha \ a \ b \ g \ f \ \mathcal{C} \ E \ \varepsilon$ **by** (rule ε)
from ε .*is-cat-equalizer-2-op[unfolded cat-op-simps]* **show** *?thesis* **by** *auto*
qed
moreover **have** $\exists P \ \pi. \ \pi : P <_{CF.\Pi} A : \mathfrak{J}(\text{Obj}) \mapsto_{C\alpha} \text{op-cat } \mathcal{C}$
if *tm-cf-discrete* $\alpha \ (\mathfrak{J}(\text{Obj})) \ A$ (*op-cat* \mathcal{C}) **for** A
proof-
interpret *tm-cf-discrete* $\alpha \ (\mathfrak{J}(\text{Obj})) \ A$ (*op-cat* \mathcal{C}) **by** (rule *that*)
from *assms(3)[OF tm-cf-discrete-op[unfolded cat-op-simps]]* **obtain** $P \ \pi$
where $\pi : \pi : A >_{CF.\Pi} P : \mathfrak{J}(\text{Obj}) \mapsto_{C\alpha} \mathcal{C}$
by *clarsimp*
interpret π : *is-cat-obj-coproduct* $\alpha \ (\mathfrak{J}(\text{Obj})) \ A \ \mathcal{C} \ P \ \pi$ **by** (rule π)
from π .*is-cat-obj-prod-op* **show** *?thesis* **by** *auto*
qed
moreover **have** $\exists P \ \pi. \ \pi : P <_{CF.\Pi} A : \mathfrak{J}(\text{Arr}) \mapsto_{C\alpha} \text{op-cat } \mathcal{C}$
if *tm-cf-discrete* $\alpha \ (\mathfrak{J}(\text{Arr})) \ A$ (*op-cat* \mathcal{C}) **for** A
proof-
interpret *tm-cf-discrete* $\alpha \ (\mathfrak{J}(\text{Arr})) \ A$ (*op-cat* \mathcal{C}) **by** (rule *that*)
from *assms(4)[OF tm-cf-discrete-op[unfolded cat-op-simps]]* **obtain** $P \ \pi$
where $\pi : \pi : A >_{CF.\Pi} P : \mathfrak{J}(\text{Arr}) \mapsto_{C\alpha} \mathcal{C}$
by *clarsimp*
interpret π : *is-cat-obj-coproduct* $\alpha \ (\mathfrak{J}(\text{Arr})) \ A \ \mathcal{C} \ P \ \pi$ **by** (rule π)
from π .*is-cat-obj-prod-op* **show** *?thesis* **by** *auto*
qed
ultimately **obtain** $u \ r$ **where** u :
 $u : r <_{CF.lim} \text{op-cf } \mathfrak{F} : \text{op-cat } \mathfrak{J} \mapsto_{C\alpha} \text{op-cat } \mathcal{C}$
by
(
rule cat-limit-of-cat-prod-obj-and-cat-equalizer[
OF \mathfrak{F} .*is-tm-functor-op, unfolded cat-op-simps*
]
)
interpret u : *is-cat-limit* $\alpha \ (\text{op-cat } \mathfrak{J}) \ (\text{op-cat } \mathcal{C}) \ (\text{op-cf } \mathfrak{F}) \ r \ u$ **by** (rule u)
from u .*is-cat-colimit-op[unfolded cat-op-simps]* **that** **show** *?thesis* **by** *simp*
qed

10.2 Small-complete and small-cocomplete category

10.2.1 Definition and elementary properties

locale *cat-small-complete* = *category* $\alpha \ \mathcal{C}$ **for** $\alpha \ \mathcal{C} +$

assumes *cat-small-complete*:

$\wedge \mathfrak{F} \ \mathfrak{J}. \ \mathfrak{F} : \mathfrak{J} \mapsto_{C.tm\alpha} \mathcal{C} \implies \exists u \ r. \ u : r <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathcal{C}$

locale *cat-small-cocomplete* = *category* α \mathfrak{C} **for** α \mathfrak{C} +
assumes *cat-small-cocomplete*:
 $\wedge \mathfrak{J} \mathfrak{J}. \mathfrak{F} : \mathfrak{J} \mapsto_{C.tm\alpha} \mathfrak{C} \implies \exists u r. u : \mathfrak{F} >_{CF.colim} r : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$

Rules.

mk-ide rf *cat-small-complete-def*[*unfolded cat-small-complete-axioms-def*]
|*intro cat-small-completeI*]
|*dest cat-small-completeD*[*dest*]
|*elim cat-small-completeE*[*elim*]

lemma *cat-small-completeE'*[*elim*]:
assumes *cat-small-complete* α \mathfrak{C} **and** $\mathfrak{F} : \mathfrak{J} \mapsto_{C.tm\alpha} \mathfrak{C}$
obtains $u r$ **where** $u : r <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$
using *assms* **by** *auto*

mk-ide rf *cat-small-cocomplete-def*[*unfolded cat-small-cocomplete-axioms-def*]
|*intro cat-small-cocompleteI*]
|*dest cat-small-cocompleteD*[*dest*]
|*elim cat-small-cocompleteE*[*elim*]

lemma *cat-small-cocompleteE'*[*elim*]:
assumes *cat-small-cocomplete* α \mathfrak{C} **and** $\mathfrak{F} : \mathfrak{J} \mapsto_{C.tm\alpha} \mathfrak{C}$
obtains $u r$ **where** $u : \mathfrak{F} >_{CF.colim} r : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$
using *assms* **by** *auto*

10.2.2 Duality

lemma (**in** *cat-small-complete*) *cat-small-cocomplete-op*[*cat-op-intros*]:
cat-small-cocomplete α (*op-cat* \mathfrak{C})
proof(*intro cat-small-cocompleteI*)
fix $\mathfrak{J} \mathfrak{J}$ **assume** $\mathfrak{F} : \mathfrak{J} \mapsto_{C.tm\alpha} \text{op-cat } \mathfrak{C}$
then interpret \mathfrak{F} : *is-tm-functor* α \mathfrak{J} $\langle \text{op-cat } \mathfrak{C} \rangle$ \mathfrak{F} .
from *cat-small-complete*[*OF* \mathfrak{F} .*is-tm-functor-op*[*unfolded cat-op-simps*]]
obtain $u r$ **where** $u : r <_{CF.lim} \text{op-cf } \mathfrak{F} : \text{op-cat } \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$
by *auto*
then interpret u : *is-cat-limit* α $\langle \text{op-cat } \mathfrak{J} \rangle$ \mathfrak{C} $\langle \text{op-cf } \mathfrak{F} \rangle$ $r u$.
from u .*is-cat-colimit-op*[*unfolded cat-op-simps*] **show**
 $\exists u r. u : \mathfrak{F} >_{CF.colim} r : \mathfrak{J} \mapsto_{C\alpha} \text{op-cat } \mathfrak{C}$
by *auto*
qed (*auto intro: cat-cs-intros*)

lemmas [*cat-op-intros*] = *cat-small-complete.cat-small-cocomplete-op*

lemma (**in** *cat-small-cocomplete*) *cat-small-complete-op*[*cat-op-intros*]:
cat-small-complete α (*op-cat* \mathfrak{C})
proof(*intro cat-small-completeI*)
fix $\mathfrak{J} \mathfrak{J}$ **assume** *prems*: $\mathfrak{F} : \mathfrak{J} \mapsto_{C.tm\alpha} \text{op-cat } \mathfrak{C}$
then interpret \mathfrak{F} : *is-tm-functor* α \mathfrak{J} $\langle \text{op-cat } \mathfrak{C} \rangle$ \mathfrak{F} .
from *cat-small-cocomplete*[*OF* \mathfrak{F} .*is-tm-functor-op*[*unfolded cat-op-simps*]]
obtain $u r$ **where** $u : \text{op-cf } \mathfrak{F} >_{CF.colim} r : \text{op-cat } \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$
by *auto*
interpret u : *is-cat-colimit* α $\langle \text{op-cat } \mathfrak{J} \rangle$ \mathfrak{C} $\langle \text{op-cf } \mathfrak{F} \rangle$ $r u$ **by** (*rule u*)
from u .*is-cat-limit-op*[*unfolded cat-op-simps*] **show**
 $\exists u r. u : r <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \text{op-cat } \mathfrak{C}$
by *auto*
qed (*auto intro: cat-cs-intros*)

lemmas [*cat-op-intros*] = *cat-small-cocomplete.cat-small-complete-op*

10.2.3 A category with equalizers and small products is small-complete

lemma (in *category*) *cat-small-complete-if-eq-and-obj-prod*:

— See Corollary 2 in Chapter V-2 in [9]

assumes $\bigwedge a \ b \ g \ f. \llbracket f : a \mapsto_{\mathcal{C}} b; g : a \mapsto_{\mathcal{C}} b \rrbracket \implies$

$\exists E \ \varepsilon. \ \varepsilon : E <_{CF.eq} (a, b, g, f) : \uparrow\uparrow_C \mapsto_{C\alpha} \mathcal{C}$

and $\bigwedge A \ I. \ tm\text{-}cf\text{-discrete} \ \alpha \ I \ A \ \mathcal{C} \implies \exists P \ \pi. \ \pi : P <_{CF.\Pi} A : I \mapsto_{C\alpha} \mathcal{C}$

shows *cat-small-complete* $\alpha \ \mathcal{C}$

proof(*intro cat-small-completeI*)

fix $\mathfrak{F} \ \mathfrak{J}$ **assume** *prems*: $\mathfrak{F} : \mathfrak{J} \mapsto_{C.tm\alpha} \mathcal{C}$

then interpret \mathfrak{F} : *is-tm-functor* $\alpha \ \mathfrak{J} \ \mathcal{C} \ \mathfrak{F}$.

show $\exists u \ r. \ u : r <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathcal{C}$

by (*rule cat-limit-of-cat-prod-obj-and-cat-equalizer*[*OF prems assms(1)*])
(*auto intro: assms(2)*)

qed (*auto simp: cat-cs-intros*)

lemma (in *category*) *cat-small-cocomplete-if-eq-and-obj-prod*:

assumes $\bigwedge a \ b \ g \ f. \llbracket f : b \mapsto_{\mathcal{C}} a; g : b \mapsto_{\mathcal{C}} a \rrbracket \implies$

$\exists E \ \varepsilon. \ \varepsilon : (a, b, g, f) >_{CF.coeq} E : \uparrow\uparrow_C \mapsto_{C\alpha} \mathcal{C}$

and $\bigwedge A \ I. \ tm\text{-}cf\text{-discrete} \ \alpha \ I \ A \ \mathcal{C} \implies \exists P \ \pi. \ \pi : A >_{CF.\Pi} P : I \mapsto_{C\alpha} \mathcal{C}$

shows *cat-small-cocomplete* $\alpha \ \mathcal{C}$

proof–

have $\exists E \ \varepsilon. \ \varepsilon : E <_{CF.eq} (a, b, g, f) : \uparrow\uparrow_C \mapsto_{C\alpha} \text{op-cat} \ \mathcal{C}$

if $f : b \mapsto_{\mathcal{C}} a$ **and** $g : b \mapsto_{\mathcal{C}} a$ **for** $a \ b \ g \ f$

proof–

from *assms(1)*[*OF that*] **obtain** $\varepsilon \ E$ **where**

$\varepsilon : \varepsilon : (a, b, g, f) >_{CF.coeq} E : \uparrow\uparrow_C \mapsto_{C\alpha} \mathcal{C}$

by *clarsimp*

interpret ε : *is-cat-coequalizer-2* $\alpha \ a \ b \ g \ f \ \mathcal{C} \ E \ \varepsilon$ **by** (*rule* ε)

from ε .*is-cat-equalizer-2-op* **show** *?thesis* **by** *auto*

qed

moreover have $\exists P \ \pi. \ \pi : P <_{CF.\Pi} A : I \mapsto_{C\alpha} \text{op-cat} \ \mathcal{C}$

if *tm-cf-discrete* $\alpha \ I \ A$ (*op-cat* \mathcal{C}) **for** $A \ I$

proof–

interpret *tm-cf-discrete* $\alpha \ I \ A$ (*op-cat* \mathcal{C}) **by** (*rule that*)

from *assms(2)*[*OF tm-cf-discrete-op*[*unfolded cat-op-simps*]] **obtain** $P \ \pi$

where $\pi : \pi : A >_{CF.\Pi} P : I \mapsto_{C\alpha} \mathcal{C}$

by *auto*

interpret π : *is-cat-obj-coproduct* $\alpha \ I \ A \ \mathcal{C} \ P \ \pi$ **by** (*rule* π)

from π .*is-cat-obj-prod-op* **show** *?thesis* **by** *auto*

qed

ultimately interpret *cat-small-complete* α (*op-cat* \mathcal{C})

by

(
 rule category.cat-small-complete-if-eq-and-obj-prod[
 OF category-op, unfolded cat-op-simps
]
)

show *?thesis* **by** (*rule cat-small-cocomplete-op*[*unfolded cat-op-simps*])

qed

10.2.4 Existence of the initial and terminal objects in small-complete and small-cocomplete categories

lemma (in *cat-small-complete*) *cat-sc-ex-obj-initial*:

— See Theorem 1 in Chapter V-6 in [9].

assumes $A \sqsubseteq_{\circ} \mathcal{C}(\text{Obj})$

and $A \in_{\circ} \text{Vset} \ \alpha$

and $\bigwedge c. \ c \in_{\circ} \mathcal{C}(\text{Obj}) \implies \exists f \ a. \ a \in_{\circ} A \wedge f : a \mapsto_{\mathcal{C}} c$

obtains z where *obj-initial* \mathfrak{C} z
proof-

```

interpret tcd: tm-cf-discrete  $\alpha$  A id  $\mathfrak{C}$ 
proof(intro tm-cf-discreteI)
  show  $(\lambda i \in_{\circ} A. \mathfrak{C}(\backslash CId)(\backslash id\ i)) \in_{\circ} Vset\ \alpha$ 
    unfolding id-def
  proof(rule vbrelation.vbrelation-Limit-in-VsetI)
    from assms(1) have  $A \subseteq_{\circ} \mathcal{D}_{\circ}(\mathfrak{C}(\backslash CId))$  by (simp add: cat-CId-vdomain)
    then have  $\mathcal{R}_{\circ}(VLambda\ A\ (app\ (\mathfrak{C}(\backslash CId)))) = \mathfrak{C}(\backslash CId) \text{ ' } A$  by auto
    moreover have  $(\bigcup_{\circ} a \in_{\circ} A. \bigcup_{\circ} b \in_{\circ} A. Hom\ \mathfrak{C}\ a\ b) \in_{\circ} Vset\ \alpha$ 
      by (rule cat-Hom-vifunion-in-Vset[OF assms(1,1,2,2)])
    moreover have  $\mathfrak{C}(\backslash CId) \text{ ' } A \subseteq_{\circ} (\bigcup_{\circ} a \in_{\circ} A. \bigcup_{\circ} b \in_{\circ} A. Hom\ \mathfrak{C}\ a\ b)$ 
  proof(intro vsubsetI)
    fix f assume  $f \in_{\circ} \mathfrak{C}(\backslash CId) \text{ ' } A$ 
    then obtain a where  $a : a \in_{\circ} A$  and f-def:  $f = \mathfrak{C}(\backslash CId)(\backslash a)$  by auto
    from assms(1) show  $f \in_{\circ} (\bigcup_{\circ} a \in_{\circ} A. \bigcup_{\circ} b \in_{\circ} A. Hom\ \mathfrak{C}\ a\ b)$ 
      unfolding f-def by (intro vifunionI) (auto simp: cat-CId-is-arr)
    qed
    ultimately show  $\mathcal{R}_{\circ}(VLambda\ A\ (app\ (\mathfrak{C}(\backslash CId)))) \in_{\circ} Vset\ \alpha$  by auto
  qed (simp-all add: assms(2))
qed
(
  use assms in
   $\langle auto\ simp: assms(2)\ Limit-vid-on-in-Vset\ intro: cat-cs-intros \rangle$ 
)

```

```

have tcd:  $:\rightarrow: A\ id\ \mathfrak{C} : :_C\ A \mapsto_{\rightarrow C} tm\alpha\ \mathfrak{C}$ 
by
(
  cs-concl cs-shallow cs-intro:
  cat-small-cs-intros
  cat-cs-intros
  cat-small-discrete-cs-intros
  cat-discrete-cs-intros
)
from cat-small-complete[OF this] obtain  $\pi\ w$ 
where  $\pi : w <_{CF.lim} :\rightarrow: A\ id\ \mathfrak{C} : :_C\ A \mapsto_{\rightarrow C} \alpha\ \mathfrak{C}$ 
by auto
then interpret  $\pi$ : is-cat-obj-prod  $\alpha$  A id  $\mathfrak{C}$   $w\ \pi$ 
by (intro is-cat-obj-prodI tcd.cf-discrete-axioms)

```

let $?ww = \langle Hom\ \mathfrak{C}\ w\ w \rangle$

```

have CId-w:  $\mathfrak{C}(\backslash CId)(\backslash w) \in_{\circ} ?ww$ 
by (cs-concl cs-intro: cat-cs-intros cat-lim-cs-intros)
then have ww-neq-vempty:  $?ww \neq 0$  by force

```

have *wd*: $\exists h. h : w \mapsto_{\mathfrak{C}} d$ **if** $d \in_{\circ} \mathfrak{C}(\backslash Obj)$ **for** *d*

proof-

```

from assms(3)[OF that] obtain g a where  $a : a \in_{\circ} A$  and  $g : g : a \mapsto_{\mathfrak{C}} d$ 
by clarsimp
from  $\pi.ntcf-NTMap-is-arr$ [unfolded the-cat-discrete-components, OF a] have  $\pi(\backslash NTMap)(\backslash a) : w \mapsto_{\mathfrak{C}} a$ 
by
(
  cs-prems cs-shallow cs-simp:
  id-apply the-cat-discrete-components(1)
)

```

cat-discrete-cs-simps cat-cs-simps

)

with g **have** $g \circ_A \mathfrak{C} \pi(\downarrow NTMap)(\downarrow a) : w \mapsto_{\mathfrak{C}} d$
by (*cs-concl cs-shallow cs-intro: cat-cs-intros*)
then show *?thesis* **by** (*intro exI*)
qed

have *cf-parallel* α ($\mathfrak{a}_{PL} ?ww$) ($\mathfrak{b}_{PL} ?ww$) $?ww$ w w (*vid-on ?ww*) \mathfrak{C}
by (*intro cat-cf-parallel-ab* π .*cat-cone-obj cat-Hom-in-Vset simp-all*)
then have $\uparrow \rightarrow \uparrow_{CF} \mathfrak{C} (\mathfrak{a}_{PL} ?ww) (\mathfrak{b}_{PL} ?ww) ?ww$ w w (*vid-on ?ww*) :
 $\uparrow_C (\mathfrak{a}_{PL} ?ww) (\mathfrak{b}_{PL} ?ww) ?ww \mapsto_{C.tm\alpha} \mathfrak{C}$
by (*intro cf-parallel.cf-parallel-the-cf-parallel-is-tm-functor*)
from *cat-small-complete*[*OF this*] **obtain** ε v **where** $\varepsilon : \varepsilon :$
 $v <_{CF.lim} \uparrow \rightarrow \uparrow_{CF} \mathfrak{C} (\mathfrak{a}_{PL} ?ww) (\mathfrak{b}_{PL} ?ww) ?ww$ w w (*vid-on ?ww*) :
 $\uparrow_C (\mathfrak{a}_{PL} ?ww) (\mathfrak{b}_{PL} ?ww) ?ww \mapsto_{C\alpha} \mathfrak{C}$
by *clarsimp*
from *is-cat-equalizerI*[
 OF
this
-
cat-Hom-in-Vset[*OF* π .*cat-cone-obj* π .*cat-cone-obj*]
ww-neq-vempty
]
interpret $\varepsilon : is-cat-equalizer$ α w w $?ww$ $\langle vid-on ?ww \rangle \mathfrak{C} v$ ε **by** *auto*
note $\varepsilon-is-monic-arr =$
is-cat-equalizer.cat-eq-is-monic-arr[*OF* $\varepsilon.is-cat-equalizer-axioms$]
note $\varepsilon-is-monic-arrD = is-monic-arrD$ [*OF* $\varepsilon-is-monic-arr$]

show *?thesis*
proof(*rule, intro obj-initialI*)

show $v \in_0 \mathfrak{C}(\downarrow Obj)$ **by** (*rule* ε .*cat-cone-obj*)
then have *CId-v*: $\mathfrak{C}(\downarrow CId)(\downarrow v) : v \mapsto_{\mathfrak{C}} v$
by (*cs-concl cs-shallow cs-intro: cat-cs-intros*)

fix d **assume** *prems*: $d \in_0 \mathfrak{C}(\downarrow Obj)$
from *wd*[*OF prems*] **obtain** h **where** $h : h : w \mapsto_{\mathfrak{C}} d$ **by** *auto*

show $\exists ! f. f : v \mapsto_{\mathfrak{C}} d$
proof(*rule exII*)
define f **where** $f = h \circ_A \mathfrak{C} \varepsilon(\downarrow NTMap)(\downarrow \mathfrak{a}_{PL} ?ww)$
from $\varepsilon-is-monic-arrD(1)$ h **show** $f : f : v \mapsto_{\mathfrak{C}} d$
unfolding *f-def* **by** (*cs-concl cs-shallow cs-intro: cat-cs-intros*)
fix g **assume** *prems'*: $g : v \mapsto_{\mathfrak{C}} d$
have *cf-parallel-2* α \mathfrak{a}_{PL2} \mathfrak{b}_{PL2} \mathfrak{g}_{PL} \mathfrak{f}_{PL} v d g f \mathfrak{C}
by (*intro cat-cf-parallel-2-cat-equalizer prems' f*)
then have $\uparrow \rightarrow \uparrow_{CF} \mathfrak{C} \mathfrak{a}_{PL2} \mathfrak{b}_{PL2} \mathfrak{g}_{PL} \mathfrak{f}_{PL} v d g f :$
 $\uparrow_C \mathfrak{a}_{PL2} \mathfrak{b}_{PL2} \mathfrak{g}_{PL} \mathfrak{f}_{PL} \mapsto_{C.tm\alpha} \mathfrak{C}$
by (*intro cf-parallel-2.cf-parallel-2-the-cf-parallel-2-is-tm-functor*)
from *cat-small-complete*[*OF this*] **obtain** $\varepsilon' u$
where $\varepsilon' :$
 $u <_{CF.lim} \uparrow \rightarrow \uparrow_{CF} \mathfrak{C} \mathfrak{a}_{PL2} \mathfrak{b}_{PL2} \mathfrak{g}_{PL} \mathfrak{f}_{PL} v d g f :$
 $\uparrow_C \mathfrak{a}_{PL2} \mathfrak{b}_{PL2} \mathfrak{g}_{PL} \mathfrak{f}_{PL} \mapsto_{C\alpha} \mathfrak{C}$
by *clarsimp*
from *is-cat-equalizer-2I*[*OF this prems' f*] **interpret** $\varepsilon' :$
is-cat-equalizer-2 α v d g f $\mathfrak{C} u$ $\varepsilon'.$
note $\varepsilon'-is-monic-arr = is-cat-equalizer-2.cat-eq-2-is-monic-arr$ [
 OF $\varepsilon'.is-cat-equalizer-2-axioms$

$\quad]$
note ε' -is-*monic-arrD* = *is-monic-arrD*[*OF* ε' -is-*monic-arr*]
then have $u \in_{\circ} \mathfrak{C}(\text{Obj})$ **by** *auto*
from $wd[OF \text{ this}]$ **obtain** s **where** $s : s : w \mapsto_{\mathfrak{C}} u$ **by** *clarsimp*
from $s \varepsilon'$ -is-*monic-arrD*(1) ε -*is-monic-arrD*(1) **have** $\varepsilon\varepsilon's$:
 $\varepsilon(\text{NTMap})(\mathfrak{a}_{PL} ?ww) \circ_{A\mathfrak{C}} \varepsilon'(\text{NTMap})(\mathfrak{a}_{PL2}) \circ_{A\mathfrak{C}} s : w \mapsto_{\mathfrak{C}} v$
by (*cs-concl cs-shallow cs-intro: cat-cs-intros*)
from $s \varepsilon'$ -is-*monic-arrD*(1) ε -*is-monic-arrD*(1) **have** $\varepsilon's\varepsilon$:
 $\varepsilon'(\text{NTMap})(\mathfrak{a}_{PL2}) \circ_{A\mathfrak{C}} s \circ_{A\mathfrak{C}} \varepsilon(\text{NTMap})(\mathfrak{a}_{PL} ?ww) : v \mapsto_{\mathfrak{C}} v$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)
from ε -*is-monic-arrD*(1) ε' -*is-monic-arrD*(1) s **have**
 $\varepsilon(\text{NTMap})(\mathfrak{a}_{PL} ?ww) \circ_{A\mathfrak{C}} (\varepsilon'(\text{NTMap})(\mathfrak{a}_{PL2}) \circ_{A\mathfrak{C}} s \circ_{A\mathfrak{C}} \varepsilon(\text{NTMap})(\mathfrak{a}_{PL} ?ww)) =$
 $\varepsilon(\text{NTMap})(\mathfrak{a}_{PL} ?ww) \circ_{A\mathfrak{C}} \varepsilon'(\text{NTMap})(\mathfrak{a}_{PL2}) \circ_{A\mathfrak{C}} s \circ_{A\mathfrak{C}} \varepsilon(\text{NTMap})(\mathfrak{a}_{PL} ?ww)$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)
also from
 $\varepsilon.$ *cat-eq-Comp-eq*
 $\quad [$
 $\quad \quad \text{unfolded in-Hom-iff, } OF \text{ cat-CId-is-arr}[OF \pi.$ *cat-cone-obj* $] \varepsilon\varepsilon's,$
 $\quad \quad \text{symmetric}$
 $\quad]$
 $\varepsilon\varepsilon's \pi.$ *cat-cone-obj* ε -*is-monic-arr*(1)
have $\dots = \mathfrak{C}(\text{CId})(\downarrow w) \circ_{A\mathfrak{C}} \varepsilon(\text{NTMap})(\mathfrak{a}_{PL} ?ww)$
by (*cs-prems cs-shallow cs-simp: vid-on-atI cs-intro: cat-cs-intros*)
also from $\varepsilon.$ *cf-parallel-a'* ε -*is-monic-arrD*(1) **have**
 $\dots = \varepsilon(\text{NTMap})(\mathfrak{a}_{PL} ?ww) \circ_{A\mathfrak{C}} \mathfrak{C}(\text{CId})(\downarrow v)$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)
finally have
 $\varepsilon(\text{NTMap})(\mathfrak{a}_{PL} ?ww) \circ_{A\mathfrak{C}} (\varepsilon'(\text{NTMap})(\mathfrak{a}_{PL2}) \circ_{A\mathfrak{C}} s \circ_{A\mathfrak{C}} \varepsilon(\text{NTMap})(\mathfrak{a}_{PL} ?ww)) =$
 $\varepsilon(\text{NTMap})(\mathfrak{a}_{PL} ?ww) \circ_{A\mathfrak{C}} \mathfrak{C}(\text{CId})(\downarrow v).$
from
 $\text{is-monic-arrD}(2)[OF \varepsilon$ -*is-monic-arr* $\varepsilon's\varepsilon$ *CId-v this]
 ε' -*is-monic-arrD*(1) $s \varepsilon$ -*is-monic-arrD*(1)
have $\varepsilon's\varepsilon$ -*is-CId*:
 $\varepsilon'(\text{NTMap})(\mathfrak{a}_{PL2}) \circ_{A\mathfrak{C}} (s \circ_{A\mathfrak{C}} \varepsilon(\text{NTMap})(\mathfrak{a}_{PL} ?ww)) = \mathfrak{C}(\text{CId})(\downarrow v)$
by (*cs-prems cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)
have ε' -*is-iso-arr*: $\varepsilon'(\text{NTMap})(\mathfrak{a}_{PL2}) : u \mapsto_{is\circ\mathfrak{C}} v$
by
 $\quad ($
 $\quad \quad \text{intro}$
 $\quad \quad \text{cat-is-iso-arr-if-is-monic-arr-is-right-inverse}$
 $\quad \quad \varepsilon'$ -*is-monic-arr*,
 $\quad \quad \text{rule is-right-inverseI}[OF - \varepsilon'$ -*is-monic-arrD*(1) $\varepsilon's\varepsilon$ -*is-CId*]
 $\quad)$
 $\quad ($
 $\quad \quad \text{use } s \varepsilon$ -*is-monic-arrD*(1) **in**
 $\quad \quad \langle \text{cs-concl cs-shallow cs-intro: cat-cs-intros} \rangle$
 $\quad)$
from $\varepsilon'.$ *cat-eq-2-Comp-eq*(1) **have**
 $g \circ_{A\mathfrak{C}} \varepsilon'(\text{NTMap})(\mathfrak{a}_{PL2}) \circ_{A\mathfrak{C}} (\varepsilon'(\text{NTMap})(\mathfrak{a}_{PL2}))^{-1} C\mathfrak{C} =$
 $f \circ_{A\mathfrak{C}} \varepsilon'(\text{NTMap})(\mathfrak{a}_{PL2}) \circ_{A\mathfrak{C}} (\varepsilon'(\text{NTMap})(\mathfrak{a}_{PL2}))^{-1} C\mathfrak{C}$
by *simp*
from $\text{this } f \varepsilon'$ -*is-monic-arrD*(1) ε' -*is-iso-arr* prems' **show** $g = f$
by
 $\quad ($
 $\quad \quad \text{cs-prems cs-shallow}$
 $\quad \quad \text{cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-arrow-cs-intros}$
 $\quad)$
qed*

qed

qed

lemma (in *cat-small-cocomplete*) *cat-sc-ex-obj-terminal*:

— See Theorem 1 in Chapter V-6 in [9].

assumes $A \subseteq_{\circ} \mathfrak{C}(Obj)$

and $A \in_{\circ} Vset \alpha$

and $\bigwedge c. c \in_{\circ} \mathfrak{C}(Obj) \implies \exists f a. a \in_{\circ} A \wedge f : c \mapsto_{\mathfrak{C}} a$

obtains z **where** *obj-terminal* $\mathfrak{C} z$

using *that*

by

(
 rule cat-small-complete.cat-sc-ex-obj-initial[
 OF cat-small-complete-op, unfolded cat-op-simps, OF assms, simplified
]
)

10.2.5 Creation of limits, continuity and completeness

lemma

— See Theorem 2 in Chapter V-4 in [9].

assumes $\mathfrak{G} : \mathfrak{A} \mapsto_{\mapsto_{C\alpha}} \mathfrak{B}$

and *cat-small-complete* $\alpha \mathfrak{B}$

and $\bigwedge \mathfrak{J} \mathfrak{I}. \mathfrak{F} : \mathfrak{J} \mapsto_{\mapsto_{C.tm\alpha}} \mathfrak{A} \implies \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{J} \mapsto_{\mapsto_{C.tm\alpha}} \mathfrak{B}$

and $\bigwedge \mathfrak{J} \mathfrak{I}. \mathfrak{F} : \mathfrak{J} \mapsto_{\mapsto_{C.tm\alpha}} \mathfrak{A} \implies \text{cf-creates-limits } \alpha \mathfrak{G} \mathfrak{F}$

shows *is-tm-cf-continuous-if-cf-creates-limits*: *is-tm-cf-continuous* $\alpha \mathfrak{G}$

and *cat-small-complete-if-cf-creates-limits*: *cat-small-complete* $\alpha \mathfrak{A}$

proof–

interpret \mathfrak{G} : *is-functor* $\alpha \mathfrak{A} \mathfrak{B} \mathfrak{G}$ **by** (*rule assms(1)*)

interpret \mathfrak{B} : *cat-small-complete* $\alpha \mathfrak{B}$ **by** (*rule assms(2)*)

show *is-tm-cf-continuous* $\alpha \mathfrak{G}$

proof(*intro is-tm-cf-continuousI, rule assms*)

fix $\mathfrak{F} \mathfrak{J}$ **assume** *prems*: $\mathfrak{F} : \mathfrak{J} \mapsto_{\mapsto_{C.tm\alpha}} \mathfrak{A}$

then **interpret** \mathfrak{F} : *is-tm-functor* $\alpha \mathfrak{J} \mathfrak{A} \mathfrak{F}$.

from *cat-small-completeD(2)*[*OF assms(2) assms(3)*][*OF prems*]] **obtain** ψr

where $\psi: \psi : r <_{CF.lim} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{J} \mapsto_{\mapsto_{C\alpha}} \mathfrak{B}$

by *clarsimp*

show *cf-preserves-limits* $\alpha \mathfrak{G} \mathfrak{F}$

by

(
 rule cf-preserves-limits-if-cf-creates-limits,
 rule assms(1),
 rule \mathfrak{F}.is-functor-axioms,
 rule \psi,
 rule assms(4)[*OF prems*]
)

qed

show *cat-small-complete* $\alpha \mathfrak{A}$

proof(*intro cat-small-completeI \mathfrak{G}.HomDom.category-axioms*)

fix $\mathfrak{F} \mathfrak{J}$ **assume** *prems*: $\mathfrak{F} : \mathfrak{J} \mapsto_{\mapsto_{C.tm\alpha}} \mathfrak{A}$

then **interpret** \mathfrak{F} : *is-tm-functor* $\alpha \mathfrak{J} \mathfrak{A} \mathfrak{F}$.

from *cat-small-completeD(2)*[*OF assms(2) assms(3)*][*OF prems*]] **obtain** ψr

where $\psi: \psi : r <_{CF.lim} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{J} \mapsto_{\mapsto_{C\alpha}} \mathfrak{B}$

by *clarsimp*
 from *cf-creates-limitsE''*[
 OF assms(4)[*OF prems*] ψ \mathfrak{F} .*is-functor-axioms* *assms*(1)
]
 show $\exists u r. u : r <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{A}$
 by *metis*
 qed

qed

10.3 Finite-complete and finite-cocomplete category

locale *cat-finite-complete* = *category* α \mathfrak{C} **for** α \mathfrak{C} +
assumes *cat-finite-complete*:
 $\wedge \mathfrak{F} \mathfrak{J}. \llbracket \text{finite-category } \alpha \mathfrak{J}; \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C} \rrbracket \implies$
 $\exists u r. u : r <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$

locale *cat-finite-cocomplete* = *category* α \mathfrak{C} **for** α \mathfrak{C} +
assumes *cat-finite-cocomplete*:
 $\wedge \mathfrak{F} \mathfrak{J}. \llbracket \text{finite-category } \alpha \mathfrak{J}; \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C} \rrbracket \implies$
 $\exists u r. u : \mathfrak{F} >_{CF.colim} r : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$

Rules.

mk-ide rf *cat-finite-complete-def*[*unfolded cat-finite-complete-axioms-def*]
 $|intro \text{ cat-finite-complete}I|$
 $|dest \text{ cat-finite-complete}D[dest]|$
 $|elim \text{ cat-finite-complete}E[elim]|$

lemma *cat-finite-completeE'*[*elim*]:
assumes *cat-finite-complete* α \mathfrak{C}
and *finite-category* α \mathfrak{J}
and $\mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$
obtains $u r$ **where** $u : r <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$
using *assms* **by** *auto*

mk-ide rf *cat-finite-cocomplete-def*[*unfolded cat-finite-cocomplete-axioms-def*]
 $|intro \text{ cat-finite-cocomplete}I|$
 $|dest \text{ cat-finite-cocomplete}D[dest]|$
 $|elim \text{ cat-finite-cocomplete}E[elim]|$

lemma *cat-finite-cocompleteE'*[*elim*]:
assumes *cat-finite-cocomplete* α \mathfrak{C}
and *finite-category* α \mathfrak{J}
and $\mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$
obtains $u r$ **where** $u : \mathfrak{F} >_{CF.colim} r : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$
using *assms* **by** *auto*

Elementary properties.

sublocale *cat-small-complete* \subseteq *cat-finite-complete*

proof(*intro cat-finite-completeI*)

fix $\mathfrak{F} \mathfrak{J}$ **assume** *prems*: *finite-category* α \mathfrak{J} $\mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$
interpret \mathfrak{F} : *is-functor* α \mathfrak{J} \mathfrak{C} \mathfrak{F} **by** (*rule prems*(2))
from *cat-small-complete-axioms* **show** $\exists u r. u : r <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{C}$
by (*auto intro*: \mathfrak{F} .*cf-is-tm-functor-if-HomDom-finite-category*[*OF prems*(1)])
qed (*auto intro*: *cat-cs-intros*)

sublocale *cat-small-cocomplete* \subseteq *cat-finite-cocomplete*

proof(*intro cat-finite-cocompleteI*)

```

fix  $\mathfrak{F} \mathfrak{J}$  assume prems: finite-category  $\alpha \mathfrak{J} \mathfrak{F} : \mathfrak{J} \mapsto \mapsto C\alpha \mathfrak{C}$ 
interpret  $\mathfrak{F}$ : is-functor  $\alpha \mathfrak{J} \mathfrak{C} \mathfrak{F}$  by (rule prems(2))
from cat-small-cocomplete-axioms show  $\exists u r. u : \mathfrak{F} >_{CF.colim} r : \mathfrak{J} \mapsto \mapsto C\alpha \mathfrak{C}$ 
  by (auto intro:  $\mathfrak{F}$ .cf-is-tm-functor-if-HomDom-finite-category[OF prems(1)])
qed (auto intro: cat-cs-intros)

```

11 Comma categories and universal constructions

11.1 Relationship between the universal arrows, initial objects and terminal objects

lemma (in is-functor) universal-arrow-of-if-obj-initial:

— See Chapter III-1 in [9].

assumes $c \in_0 \mathfrak{B}(\text{Obj})$ **and** *obj-initial* $(c \downarrow_{CF} \mathfrak{F}) [0, r, u]$.

shows *universal-arrow-of* $\mathfrak{F} \ c \ r \ u$

proof(*intro universal-arrow-ofI*)

have $ru: [0, r, u]_0 \in_0 c \downarrow_{CF} \mathfrak{F}(\text{Obj})$

and *f-unique*: $C \in_0 c \downarrow_{CF} \mathfrak{F}(\text{Obj}) \implies \exists !f. f : [0, r, u]_0 \mapsto_c \downarrow_{CF} \mathfrak{F} \ C$

for C

by (*intro obj-initialD[OF assms(2)]+*)

show $r: r \in_0 \mathfrak{A}(\text{Obj})$ **and** $u: u : c \mapsto_{\mathfrak{B}} \mathfrak{F}(\text{ObjMap})(r)$

by (*intro cat-obj-cf-comma-ObjD[OF ru assms(1)]+*)

fix $r' \ u'$ **assume** *prems*: $r' \in_0 \mathfrak{A}(\text{Obj})$ $u' : c \mapsto_{\mathfrak{B}} \mathfrak{F}(\text{ObjMap})(r')$

from *assms(1) prems* **have** $r'u': [0, r', u']_0 \in_0 c \downarrow_{CF} \mathfrak{F}(\text{Obj})$

by (*cs-concl cs-shallow cs-intro: cat-comma-cs-intros*)

from *f-unique[OF r'u']* **obtain** F

where $F: F : [0, r, u]_0 \mapsto_c \downarrow_{CF} \mathfrak{F} [0, r', u']_0$

and *F-unique*: $F' : [0, r, u]_0 \mapsto_c \downarrow_{CF} \mathfrak{F} [0, r', u']_0 \implies F' = F$

for F'

by *metis*

from *cat-obj-cf-comma-is-arrE[OF F assms(1), simplified]* **obtain** t

where *F-def*: $F = [[0, r, u]_0, [0, r', u']_0, [0, t]_0]$

and $t: t : r \mapsto_{\mathfrak{A}} r'$

and [*cat-cs-simps*]: $\mathfrak{F}(\text{ArrMap})(t) \circ_{A\mathfrak{B}} u = u'$

by *metis*

show $\exists !f'. f' : r \mapsto_{\mathfrak{A}} r' \wedge u' = \text{umap-of } \mathfrak{F} \ c \ r \ u \ r'(\text{ArrVal})(f')$

proof(*intro exI conjI; (elim conjE)?; (rule t)?*)

from $t \ u$ **show** $u' = \text{umap-of } \mathfrak{F} \ c \ r \ u \ r'(\text{ArrVal})(t)$

by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

fix t' **assume** *prems'*: $t' : r \mapsto_{\mathfrak{A}} r' \wedge u' = \text{umap-of } \mathfrak{F} \ c \ r \ u \ r'(\text{ArrVal})(t')$

from *prems'(2,1) u* **have** [*symmetric, cat-cs-simps*]:

$u' = \mathfrak{F}(\text{ArrMap})(t') \circ_{A\mathfrak{B}} u$

by (*cs-prems cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

define F' **where** $F' = [[0, r, u]_0, [0, r', u']_0, [0, t']_0]$

from *assms(1) prems'(1) u prems(2)* **have** F' :

$F' : [0, r, u]_0 \mapsto_c \downarrow_{CF} \mathfrak{F} [0, r', u']_0$

unfolding *F'-def*

by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-comma-cs-intros*)

from *F-unique[OF this]* **show** $t' = t$ **unfolding** *F'-def F-def* **by** *simp*

qed

qed

lemma (in is-functor) obj-initial-if-universal-arrow-of:

— See Chapter III-1 in [9].

assumes *universal-arrow-of* $\mathfrak{F} \ c \ r \ u$

shows *obj-initial* $(c \downarrow_{CF} \mathfrak{F}) [0, r, u]$.

proof–

from *universal-arrow-ofD[OF assms]* **have** $r: r \in_0 \mathfrak{A}(\text{Obj})$

and $u: u : c \mapsto_{\mathfrak{B}} \mathfrak{F}(\text{ObjMap})(r)$

and *up*: $[[r' \in_0 \mathfrak{A}(\text{Obj}); u' : c \mapsto_{\mathfrak{B}} \mathfrak{F}(\text{ObjMap})(r')]] \implies$

$\exists !f'. f' : r \mapsto_{\mathfrak{A}} r' \wedge u' = \text{umap-of } \mathfrak{F} \ c \ r \ u \ r'(\text{ArrVal})(f')$

for $r' \ u'$

by *auto*

from u **have** $c: c \in_0 \mathfrak{B}(\text{Obj})$ **by** *auto*

show *?thesis*

proof(*intro obj-initialI*)

from $r\ u$ show $[0, r, u]_o \in_o c \downarrow_{CF} \mathfrak{F}(\text{Obj})$

by (*cs-concl cs-shallow cs-intro: cat-cs-intros cat-comma-cs-intros*)

fix B assume *prems*: $B \in_o c \downarrow_{CF} \mathfrak{F}(\text{Obj})$

from *cat-obj-cf-comma-ObjE*[*OF prems c*] obtain $r'\ u'$

where *B-def*: $B = [0, r', u']_o$

and $r': r' \in_o \mathfrak{A}(\text{Obj})$

and $u': u' : c \mapsto_{\mathfrak{B}} \mathfrak{F}(\text{ObjMap})(r')$

by *auto*

from *up*[*OF r' u'*] obtain f

where $f: f : r \mapsto_{\mathfrak{A}} r'$

and *u'-def*: $u' = \text{umap-of } \mathfrak{F} \ c \ r \ u \ r'(\text{ArrVal})(f)$

and *up'*: $[[f' : r \mapsto_{\mathfrak{A}} r'; u' = \text{umap-of } \mathfrak{F} \ c \ r \ u \ r'(\text{ArrVal})(f')]] \implies f' = f$

for f'

by *auto*

from *u'-def f u* have [*symmetric, cat-cs-simps*]: $u' = \mathfrak{F}(\text{ArrMap})(f) \circ_{A\mathfrak{B}} u$

by (*cs-prems cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

define F where $F = [[0, r, u]_o, [0, r', u']_o, [0, f]_o]$

show $\exists!f. f : [0, r, u]_o \mapsto_c \downarrow_{CF} \mathfrak{F} \ B$

unfolding *B-def*

proof(*rule ex1I*)

from $c\ u\ f\ u'$ show $F : [0, r, u]_o \mapsto_c \downarrow_{CF} \mathfrak{F} \ [0, r', u']_o$

unfolding *F-def*

by

(

cs-concl cs-shallow

cs-simp: cat-cs-simps cs-intro: cat-comma-cs-intros

)

fix F' assume *prems'*: $F' : [0, r, u]_o \mapsto_c \downarrow_{CF} \mathfrak{F} \ [0, r', u']_o$

from *cat-obj-cf-comma-is-arrE*[*OF prems' c, simplified*] obtain f'

where *F'-def*: $F' = [[0, r, u]_o, [0, r', u']_o, [0, f']_o]$

and $f': f' : r \mapsto_{\mathfrak{A}} r'$

and [*cat-cs-simps*]: $\mathfrak{F}(\text{ArrMap})(f') \circ_{A\mathfrak{B}} u = u'$

by *auto*

from $f'\ u$ have $u' = \text{umap-of } \mathfrak{F} \ c \ r \ u \ r'(\text{ArrVal})(f')$

by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

from *up'*[*OF f' this*] show $F' = F$ unfolding *F'-def F-def* by *simp*

qed

qed

qed

lemma (in *is-functor*) *universal-arrow-fo-if-obj-terminal*:

— See Chapter III-1 in [9].

assumes $c \in_o \mathfrak{B}(\text{Obj})$ and *obj-terminal* ($\mathfrak{F} \ c \downarrow \ c$) [$r, 0, u$].

shows *universal-arrow-fo* $\mathfrak{F} \ c \ r \ u$

proof–

let $?op\text{-}\mathfrak{F} \ c = \langle op\text{-}cat \ (\mathfrak{F} \ c \downarrow \ c) \rangle$

and $?c\text{-}op\text{-}\mathfrak{F} = \langle c \downarrow_{CF} (op\text{-}cf \ \mathfrak{F}) \rangle$

and $?iso = \langle op\text{-}cf\text{-}obj\text{-}comma \ \mathfrak{F} \ c \rangle$

from *cat-cf-obj-comma-ObjD*[*OF obj-terminalD(1)[OF assms(2)] assms(1)*]

have $r: r \in_o \mathfrak{A}(\text{Obj})$ and $u: u : \mathfrak{F}(\text{ObjMap})(r) \mapsto_{\mathfrak{B}} c$

by *simp-all*

interpret $\mathfrak{F} \ c$: *is-iso-functor* α $?op\text{-}\mathfrak{F} \ c$ $?c\text{-}op\text{-}\mathfrak{F}$ $?iso$

by (*rule op-cf-obj-comma-is-iso-functor*[*OF assms(1)*])

have *iso-cocontinuous*: *is-cf-cocontinuous* α $?iso$

by

```

(
  rule is-iso-functor.iso-cf-is-cf-cocontinuous[
    OF  $\mathfrak{F}c$ .is-iso-functor-axioms
  ]
)
have iso-preserves: cf-preserves-colimits  $\alpha$  ?iso (cf-0 ?op- $\mathfrak{F}c$ )
by
(
  rule is-cf-cocontinuousD
  [
    OF
      iso-cocontinuous
      cf-0-is-functor[OF  $\mathfrak{F}c$ .HomDom.category-axioms]
       $\mathfrak{F}c$ .is-functor-axioms
  ]
)
from category.cat-obj-initial-is-cat-obj-empty-initial[
  OF  $\mathfrak{F}c$ .HomDom.category-axioms op-cat-obj-initial[THEN iffD2, OF assms(2)]
]
interpret ntcf-0-op- $\mathfrak{F}c$ :
  is-cat-obj-empty-initial  $\alpha$  ?op- $\mathfrak{F}c$   $\langle [r, 0, u]_o \rangle$   $\langle$  ntcf-0 ?op- $\mathfrak{F}c$   $\rangle$ 
by simp
have cf-0 ?op- $\mathfrak{F}c$  : cat-0  $\mapsto$   $C_\alpha$  ?op- $\mathfrak{F}c$ 
by (cs-concl cs-shallow cs-intro: cat-cs-intros)
from
  cf-preserves-colimitsD
  [
    OF
      iso-preserves
      ntcf-0-op- $\mathfrak{F}c$ .is-cat-colimit-axioms
      this
       $\mathfrak{F}c$ .is-functor-axioms
  ]
  assms(1) r u
have ntcf-0 ?c-op- $\mathfrak{F}$  :
  cf-0 ?c-op- $\mathfrak{F}$   $>_{CF.colim} [0, r, u]_o$  : cat-0  $\mapsto$   $C_\alpha$  ?c-op- $\mathfrak{F}$ 
by
(
  cs-prems cs-shallow
  cs-simp: cat-cs-simps cat-comma-cs-simps
  cs-intro: cat-cs-intros cat-comma-cs-intros
)
then have obj-initial-ru: obj-initial ?c-op- $\mathfrak{F}$   $[0, r, u]_o$ 
by
(
  rule is-cat-obj-empty-initial.cat-oei-obj-initial[
    OF is-cat-obj-empty-initialI
  ]
)
from assms(1) have  $c \in_o$  op-cat  $\mathfrak{B}(\text{Obj})$ 
by (cs-concl cs-shallow cs-intro: cat-op-intros)
from
  is-functor.universal-arrow-of-if-obj-initial[
    OF is-functor-op this obj-initial-ru
  ]
have universal-arrow-of (op-cf  $\mathfrak{F}$ ) c r u
by simp
then show ?thesis unfolding cat-op-simps .

```

qed

lemma (in *is-functor*) *obj-terminal-if-universal-arrow-fo*:

— See Chapter III-1 in [9].

assumes *universal-arrow-fo* $\mathfrak{F} \ c \ r \ u$

shows *obj-terminal* ($\mathfrak{F} \ CF \downarrow \ c$) [r, θ, u].

proof–

let $?op\text{-}\mathfrak{F} \ c = \langle op\text{-}cat \ (\mathfrak{F} \ CF \downarrow \ c) \rangle$

and $?c\text{-}op\text{-}\mathfrak{F} = \langle c \downarrow_{CF} \ (op\text{-}cf \ \mathfrak{F}) \rangle$

and $?iso = \langle inv\text{-}cf \ (op\text{-}cf\text{-}obj\text{-}comma \ \mathfrak{F} \ c) \rangle$

from *universal-arrow-foD*[*OF assms*] **have** $r: r \in_0 \mathfrak{A}(\text{Obj})$

and $w: u: \mathfrak{F}(\text{ObjMap})(r) \mapsto_{\mathfrak{B}} c$

by *auto*

then **have** $c: c \in_0 \mathfrak{B}(\text{Obj})$ **by** *auto*

from u **have** $c\text{-}op\text{-}\mathfrak{F}$: *category* $\alpha \ ?c\text{-}op\text{-}\mathfrak{F}$

by

(
cs-concl cs-shallow cs-intro:
cat-cs-intros cat-comma-cs-intros cat-op-intros
)

interpret $\mathfrak{F} \ c$: *is-iso-functor* $\alpha \ ?op\text{-}\mathfrak{F} \ c \ ?c\text{-}op\text{-}\mathfrak{F} \ \langle op\text{-}cf\text{-}obj\text{-}comma \ \mathfrak{F} \ c \rangle$

by (*rule op-cf-obj-comma-is-iso-functor*[*OF c*])

interpret $inv\text{-}\mathfrak{F} \ c$: *is-iso-functor* $\alpha \ ?c\text{-}op\text{-}\mathfrak{F} \ ?op\text{-}\mathfrak{F} \ c \ ?iso$

by (*cs-concl cs-shallow cs-intro*: *cf-cs-intros*)

have *iso-cocontinuous*: *is-cf-cocontinuous* $\alpha \ ?iso$

by

(
rule is-iso-functor.iso-cf-is-cf-cocontinuous[
OF inv-FC.is-iso-functor-axioms
]
)

have *iso-preserves*: *cf-preserves-colimits* $\alpha \ ?iso \ (cf\text{-}0 \ ?c\text{-}op\text{-}\mathfrak{F})$

by

(
rule is-cf-cocontinuousD
 [
OF
iso-cocontinuous
cf-0-is-functor[*OF FC.HomCod.category-axioms*]
inv-FC.is-functor-axioms
]
)

from *assms* **have** *universal-arrow-of* ($op\text{-}cf \ \mathfrak{F}$) $c \ r \ u$ **unfolding** *cat-op-simps*.

from *is-cat-obj-empty-initialD*

[
OF category.cat-obj-initial-is-cat-obj-empty-initial
 [
OF c-op-F is-functor.obj-initial-if-universal-arrow-of[
OF is-functor-op this
]
]
]

have *ntcf-0-c-op-F*: *ntcf-0* $?c\text{-}op\text{-}\mathfrak{F}$:

$cf\text{-}0 \ ?c\text{-}op\text{-}\mathfrak{F} >_{CF.colim} [0, r, u]_0 : cat\text{-}0 \mapsto_{C\alpha} ?c\text{-}op\text{-}\mathfrak{F}$.

have $cf\text{-}0\text{-}c\text{-}op\text{-}\mathfrak{F}$: $cf\text{-}0 \ ?c\text{-}op\text{-}\mathfrak{F} : cat\text{-}0 \mapsto_{C\alpha} ?c\text{-}op\text{-}\mathfrak{F}$

by (*cs-concl cs-shallow cs-intro*: *cat-cs-intros*)

from

cf-preserves-colimitsD[

```

    OF iso-preserves ntcf-0-c-op- $\mathfrak{F}$  cf-0-c-op- $\mathfrak{F}$  inv- $\mathfrak{F}$ .is-functor-axioms
  ]
  r u
  have ntcf-0 ?op- $\mathfrak{F}$ c : cf-0 ?op- $\mathfrak{F}$ c >CF.colim [r, 0, u]_o : cat-0  $\mapsto$   $C\alpha$  ?op- $\mathfrak{F}$ c
  by
  (
    cs-prems cs-shallow
    cs-simp: cat-cs-simps cat-comma-cs-simps  $\mathfrak{F}$ c.inv-cf-ObjMap-app
    cs-intro: cat-cs-intros cat-comma-cs-intros cat-op-intros
  )
  from
  is-cat-obj-empty-initial.cat-oei-obj-initial[
    OF is-cat-obj-empty-initialI[OF this]
  ]
  show obj-terminal ( $\mathfrak{F}$   $\downarrow$ CF c) [r, 0, u]_o.
  unfolding op-cat-obj-initial[symmetric].
  qed

```

11.2 A projection for a comma category constructed from a functor and an object creates small limits

See Chapter V-6 in [9].

lemma *cf-obj-cf-comma-proj-creates-limits*:

```

assumes  $\mathfrak{G} : \mathfrak{A} \mapsto$   $C\alpha$   $\mathfrak{X}$ 
  and is-tm-cf-continuous  $\alpha$   $\mathfrak{G}$ 
  and  $x \in_o \mathfrak{X}(\text{Obj})$ 
  and  $\mathfrak{J} : \mathfrak{J} \mapsto$   $C.tm\alpha$   $x \downarrow$ CF  $\mathfrak{G}$ 
shows cf-creates-limits  $\alpha$  ( $x \circ$  $\square$ CF  $\mathfrak{G}$ )  $\mathfrak{F}$ 
proof(intro cf-creates-limitsI conjI allI impI)

```

```

interpret  $\mathfrak{G}$ : is-functor  $\alpha$   $\mathfrak{A}$   $\mathfrak{X}$   $\mathfrak{G}$  by (rule assms(1))
interpret  $\mathfrak{F}$ : is-tm-functor  $\alpha$   $\mathfrak{J}$   $\langle x \downarrow$ CF  $\mathfrak{G} \rangle$   $\mathfrak{F}$  by (rule assms(4))
interpret  $x\mathfrak{G}$ : is-functor  $\alpha$   $\langle x \downarrow$ CF  $\mathfrak{G} \rangle$   $\mathfrak{A}$   $\langle x \circ$  $\square$ CF  $\mathfrak{G} \rangle$ 
  by (rule  $\mathfrak{G}$ .cf-obj-cf-comma-proj-is-functor[OF assms(3)])

```

```

show  $\mathfrak{F} : \mathfrak{J} \mapsto$   $C\alpha$   $x \downarrow$ CF  $\mathfrak{G}$  by (rule  $\mathfrak{F}$ .is-functor-axioms)
show  $x \circ$  $\square$ CF  $\mathfrak{G} : x \downarrow$ CF  $\mathfrak{G} \mapsto$   $C\alpha$   $\mathfrak{A}$  by (rule  $x\mathfrak{G}$ .is-functor-axioms)

```

```

define  $\psi :: V$ 
where  $\psi =$ 
  [
    ( $\lambda j \in_o \mathfrak{J}(\text{Obj}). \mathfrak{F}(\text{ObjMap})(j)(2_{\mathbb{N}})$ ),
    cf-const  $\mathfrak{J}$   $\mathfrak{X}$   $x$ ,
     $\mathfrak{G} \circ$ CF ( $x \circ$  $\square$ CF  $\mathfrak{G} \circ$ CF  $\mathfrak{F}$ ),
     $\mathfrak{J}$ ,
     $\mathfrak{X}$ 
  ]_o

```

have *ψ -components*:

```

 $\psi(\text{NTMap}) = (\lambda j \in_o \mathfrak{J}(\text{Obj}). \mathfrak{F}(\text{ObjMap})(j)(2_{\mathbb{N}}))$ 
 $\psi(\text{NTDom}) = \text{cf-const } \mathfrak{J} \mathfrak{X} x$ 
 $\psi(\text{NTCod}) = \mathfrak{G} \circ$ CF ( $x \circ$  $\square$ CF  $\mathfrak{G} \circ$ CF  $\mathfrak{F}$ )
 $\psi(\text{NTDGDom}) = \mathfrak{J}$ 
 $\psi(\text{NTDGCod}) = \mathfrak{X}$ 
unfolding  $\psi$ -def nt-field-simps by (simp-all add: nat-omega-simps)

```

have ψ -*NTMap-app*: $\psi(\text{NTMap})(j) = f$

if $j \in \circ \mathfrak{J}(\text{Obj})$ and $\mathfrak{F}(\text{ObjMap})(\downarrow j) = [a, b, f]_{\circ}$ for $a b f j$
 using that unfolding ψ -components by (simp add: nat-omega-simps)

interpret ψ : is-cat-cone $\alpha x \mathfrak{J} \mathfrak{X} \langle \mathfrak{G} \circ_{CF} (x \text{ o}\square_{CF} \mathfrak{G} \circ_{CF} \mathfrak{F}) \rangle \psi$

proof(intro is-cat-coneI is-ntcfI')

show vsequence ψ unfolding ψ -def by clarsimp

show vcard $\psi = 5_{\mathbb{N}}$ unfolding ψ -def by (simp add: nat-omega-simps)

show $\psi(\text{NTMap})(\downarrow a)$:

cf-const $\mathfrak{J} \mathfrak{X} x(\text{ObjMap})(\downarrow a) \mapsto_{\mathfrak{X}} (\mathfrak{G} \circ_{CF} (x \text{ o}\square_{CF} \mathfrak{G} \circ_{CF} \mathfrak{F}))(\text{ObjMap})(\downarrow a)$

if $a \in \circ \mathfrak{J}(\text{Obj})$ for a

proof-

from that have $\mathfrak{F}(\text{ObjMap})(\downarrow a) \in \circ x \downarrow_{CF} \mathfrak{G}(\text{Obj})$

by (cs-concl cs-shallow cs-intro: cat-cs-intros)

from $\mathfrak{G}.\text{cat-obj-cf-comma-ObjE}[OF \text{ this assms}(\mathfrak{J})]$ obtain $c g$

where $\mathfrak{F}a\text{-def}$: $\mathfrak{F}(\text{ObjMap})(\downarrow a) = [0, c, g]_{\circ}$

and c : $c \in \circ \mathfrak{A}(\text{Obj})$

and g : $g : x \mapsto_{\mathfrak{X}} \mathfrak{G}(\text{ObjMap})(\downarrow c)$

by auto

from $c g$ show ?thesis

using that

by

(

cs-concl

cs-simp: cat-comma-cs-simps cat-cs-simps $\mathfrak{F}a\text{-def}$ ψ -NTMap-app

cs-intro: cat-cs-intros cat-comma-cs-intros

)

qed

show

$\psi(\text{NTMap})(\downarrow b) \circ_{A\mathfrak{X}} \text{cf-const } \mathfrak{J} \mathfrak{X} x(\text{ArrMap})(\downarrow f) =$

$(\mathfrak{G} \circ_{CF} (x \text{ o}\square_{CF} \mathfrak{G} \circ_{CF} \mathfrak{F}))(\text{ArrMap})(\downarrow f) \circ_{A\mathfrak{X}} \psi(\text{NTMap})(\downarrow a)$

if $f : a \mapsto_{\mathfrak{J}} b$ for $a b f$

proof-

from that have $\mathfrak{F}f$: $\mathfrak{F}(\text{ArrMap})(\downarrow f) : \mathfrak{F}(\text{ObjMap})(\downarrow a) \mapsto_x \downarrow_{CF} \mathfrak{G} \mathfrak{F}(\text{ObjMap})(\downarrow b)$

by (cs-concl cs-shallow cs-intro: cat-cs-intros)

from $\mathfrak{G}.\text{cat-obj-cf-comma-is-arrE}[OF \text{ this assms}(\mathfrak{J})]$ obtain $c h c' h' k$

where $\mathfrak{F}f\text{-def}$: $\mathfrak{F}(\text{ArrMap})(\downarrow f) = [[0, c, h]_{\circ}, [0, c', h']_{\circ}, [0, k]_{\circ}]_{\circ}$

and $\mathfrak{F}a\text{-def}$: $\mathfrak{F}(\text{ObjMap})(\downarrow a) = [0, c, h]_{\circ}$

and $\mathfrak{F}b\text{-def}$: $\mathfrak{F}(\text{ObjMap})(\downarrow b) = [0, c', h']_{\circ}$

and k : $k : c \mapsto_{\mathfrak{A}} c'$

and h : $h : x \mapsto_{\mathfrak{X}} \mathfrak{G}(\text{ObjMap})(\downarrow c)$

and h' : $h' : x \mapsto_{\mathfrak{X}} \mathfrak{G}(\text{ObjMap})(\downarrow c')$

and [cat-cs-simps]: $\mathfrak{G}(\text{ArrMap})(\downarrow k) \circ_{A\mathfrak{X}} h = h'$

by metis

from $\mathfrak{F}f k h h'$ that show ?thesis

unfolding $\mathfrak{F}f\text{-def}$ $\mathfrak{F}a\text{-def}$ $\mathfrak{F}b\text{-def}$

by

(

cs-concl

cs-simp:

cat-cs-simps cat-comma-cs-simps

$\mathfrak{F}f\text{-def}$ $\mathfrak{F}a\text{-def}$ $\mathfrak{F}b\text{-def}$ ψ -NTMap-app

cs-intro: cat-cs-intros

)

qed

qed (auto simp: assms(\mathfrak{J}) ψ -components intro: cat-cs-intros)

fix τb assume prems: $\tau : b <_{CF} \text{lim } x \text{ o}\square_{CF} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{A}$

interpret τ : is-cat-limit $\alpha \mathfrak{J} \mathfrak{A} \langle x \text{ o}\square_{CF} \mathfrak{G} \circ_{CF} \mathfrak{F} \rangle b \tau$ by (rule prems)

note $x\mathfrak{G}\text{-}\mathfrak{F} = \text{cf-comp-cf-obj-cf-comma-proj-is-tm-functor}[OF \text{ assms}(1,4,3)]$

have $\text{cf-preserves-limits } \alpha \mathfrak{G} (x \circ_{CF} \mathfrak{G} \circ_{CF} \mathfrak{F})$

by ($\text{rule is-tm-cf-continuousD } [OF \text{ assms}(2) \ x\mathfrak{G}\text{-}\mathfrak{F} \ \text{assms}(1)]$)

then have $\mathfrak{G}\tau: \mathfrak{G} \circ_{CF\text{-}NTCF} \tau :$

$\mathfrak{G}(\text{ObjMap})(b) <_{CF.lim} \mathfrak{G} \circ_{CF} (x \circ_{CF} \mathfrak{G} \circ_{CF} \mathfrak{F}) : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{X}$

by

(
 $\text{rule cf-preserves-limitsD}[$
 $\text{OF - prems}(1) \ \text{is-tm-functorD}(1)[OF \ x\mathfrak{G}\text{-}\mathfrak{F}] \ \text{assms}(1)$
 $]$
 $)$

from $\text{is-cat-limit.cat-lim-unique-cone}'[OF \ \mathfrak{G}\tau \ \psi.\text{is-cat-cone-axioms}]$ **obtain** f

where $f: f : x \mapsto_{\mathfrak{X}} \mathfrak{G}(\text{ObjMap})(b)$

and $\psi\text{-}f: \bigwedge j. j \in_{\circ} \mathfrak{J}(\text{Obj}) \implies$

$\psi(\text{NTMap})(j) = (\mathfrak{G} \circ_{CF\text{-}NTCF} \tau)(\text{NTMap})(j) \circ_{A\mathfrak{X}} f$

and $f\text{-unique}$:

\llbracket
 $f' : x \mapsto_{\mathfrak{X}} \mathfrak{G}(\text{ObjMap})(b);$
 $\bigwedge j. j \in_{\circ} \mathfrak{J}(\text{Obj}) \implies \psi(\text{NTMap})(j) = (\mathfrak{G} \circ_{CF\text{-}NTCF} \tau)(\text{NTMap})(j) \circ_{A\mathfrak{X}} f'$
 $\rrbracket \implies f' = f$

for f'

by metis

define $\sigma :: V$

where $\sigma =$

[
 $($
 $\lambda j \in_{\circ} \mathfrak{J}(\text{Obj}).$
 $[$
 $[0, b, f]_{\circ},$
 $[0, (x \circ_{CF} \mathfrak{G} \circ_{CF} \mathfrak{F})(\text{ObjMap})(j), \psi(\text{NTMap})(j)]_{\circ},$
 $[0, \tau(\text{NTMap})(j)]_{\circ}.$
 $]$
 $),$
 $\text{cf-const } \mathfrak{J} (x \downarrow_{CF} \mathfrak{G}) [0, b, f]_{\circ},$
 $\mathfrak{F},$
 $\mathfrak{J},$
 $x \downarrow_{CF} \mathfrak{G}$
 $]$
 $]$
 $]$

have $\sigma\text{-components: } \sigma(\text{NTMap}) =$

(
 $\lambda j \in_{\circ} \mathfrak{J}(\text{Obj}).$
 $[$
 $[0, b, f]_{\circ},$
 $[0, (x \circ_{CF} \mathfrak{G} \circ_{CF} \mathfrak{F})(\text{ObjMap})(j), \psi(\text{NTMap})(j)]_{\circ},$
 $[0, \tau(\text{NTMap})(j)]_{\circ}.$
 $]$
 $)$

and $[\text{cat-cs-simps}]: \sigma(\text{NTDom}) = \text{cf-const } \mathfrak{J} (x \downarrow_{CF} \mathfrak{G}) [0, b, f]_{\circ}$

and $[\text{cat-cs-simps}]: \sigma(\text{NTCod}) = \mathfrak{F}$

and $[\text{cat-cs-simps}]: \sigma(\text{NTDGDom}) = \mathfrak{J}$

and $[\text{cat-cs-simps}]: \sigma(\text{NTDGCod}) = x \downarrow_{CF} \mathfrak{G}$

unfolding $\sigma\text{-def nt-field-simps}$ **by** ($\text{simp-all add: nat-omega-simps}$)

have $\sigma\text{-NTMap-app: } \sigma(\text{NTMap})(j) =$

$[$
 $[0, b, f]_{\circ},$
 $[0, (x \circ \prod_{CF} \mathfrak{G} \circ_{CF} \mathfrak{F})(\text{ObjMap})(j), \psi(\text{NTMap})(j)]_{\circ},$
 $[0, \tau(\text{NTMap})(j)]_{\circ}$
 $]_{\circ}$
if $j \in_{\circ} \mathfrak{J}(\text{Obj})$ **for** j
using *that unfolding σ -components by simp*

interpret σ : *is-cat-cone* $\alpha \langle [0, b, f]_{\circ} \rangle \mathfrak{J} \langle x \downarrow_{CF} \mathfrak{G} \rangle \mathfrak{F} \sigma$
proof(*intro is-cat-coneI is-ntcfI'*)
show *vfsequence* σ **unfolding** σ -*def* **by** *auto*
show *vcard* $\sigma = \mathfrak{S}_N$ **unfolding** σ -*def* **by** (*simp add: nat-omega-simps*)
from f **show** *cf-const* $\mathfrak{J} (x \downarrow_{CF} \mathfrak{G}) [0, b, f]_{\circ} : \mathfrak{J} \mapsto_{C\alpha} x \downarrow_{CF} \mathfrak{G}$
by
 $($
cs-concl cs-intro:
cat-cs-intros cat-lim-cs-intros cat-comma-cs-intros
 $)$
show *vsu* $(\sigma(\text{NTMap}))$ **unfolding** σ -*components* **by** *auto*
show $\mathcal{D}_{\circ} (\sigma(\text{NTMap})) = \mathfrak{J}(\text{Obj})$ **unfolding** σ -*components* **by** *auto*
from *assms*(\mathfrak{B}) **show** $\sigma(\text{NTMap})(a) :$
cf-const $\mathfrak{J} (x \downarrow_{CF} \mathfrak{G}) [0, b, f]_{\circ}(\text{ObjMap})(a) \mapsto_{x \downarrow_{CF} \mathfrak{G}} \mathfrak{F}(\text{ObjMap})(a)$
if $a \in_{\circ} \mathfrak{J}(\text{Obj})$ **for** a
proof-
from *that* **have** $\mathfrak{F}a: \mathfrak{F}(\text{ObjMap})(a) \in_{\circ} x \downarrow_{CF} \mathfrak{G}(\text{Obj})$
by (*cs-concl cs-shallow cs-intro: cat-cs-intros*)
from $\mathfrak{G}.cat\text{-obj-cf-comma-ObjE}[OF \text{ this } assms(\mathfrak{B})]$ **obtain** $c \ g$
where $\mathfrak{F}a\text{-def}: \mathfrak{F}(\text{ObjMap})(a) = [0, c, g]_{\circ}$
and $c: c \in_{\circ} \mathfrak{A}(\text{Obj})$
and $g: g: x \mapsto_{\mathfrak{X}} \mathfrak{G}(\text{ObjMap})(c)$
by *auto*
from $\psi\text{-f}[OF \text{ that}]$ *that* $c \ f \ g \ \mathfrak{F}a$ **have** [*symmetric, cat-cs-simps*]:
 $g = \mathfrak{G}(\text{ArrMap})(\tau(\text{NTMap})(a)) \circ_{A\mathfrak{X}} f$
by
 $($
cs-prems cs-shallow
cs-simp: *cat-cs-simps ψ -NTMap-app $\mathfrak{F}a\text{-def}$ cs-intro: cat-cs-intros*
 $)$
from *that* $c \ f \ g \ \mathfrak{F}a$ **show** *?thesis*
unfolding $\mathfrak{F}a\text{-def}$
by
 $($
cs-concl
cs-simp:
cat-comma-cs-simps cat-cs-simps
 ψ -NTMap-app σ -NTMap-app $\mathfrak{F}a\text{-def}$
cs-intro: *cat-cs-intros cat-comma-cs-intros*
 $)$
qed
show
 $\sigma(\text{NTMap})(d) \circ_{Ax \downarrow_{CF} \mathfrak{G}} cf\text{-const} \ \mathfrak{J} (x \downarrow_{CF} \mathfrak{G}) [0, b, f]_{\circ}(\text{ArrMap})(g) =$
 $\mathfrak{F}(\text{ArrMap})(g) \circ_{Ax \downarrow_{CF} \mathfrak{G}} \sigma(\text{NTMap})(c)$
if $g: c \mapsto_{\mathfrak{J}} d$ **for** $c \ d \ g$
proof-
from *that* **have** $\mathfrak{F}g: \mathfrak{F}(\text{ArrMap})(g) : \mathfrak{F}(\text{ObjMap})(c) \mapsto_{x \downarrow_{CF} \mathfrak{G}} \mathfrak{F}(\text{ObjMap})(d)$
by (*cs-concl cs-shallow cs-intro: cat-cs-intros*)
from $\mathfrak{G}.cat\text{-obj-cf-comma-is-arrE}[OF \text{ this } assms(\mathfrak{B})]$ **obtain** $e \ h \ e' \ h' \ k$
where $\mathfrak{F}g\text{-def}: \mathfrak{F}(\text{ArrMap})(g) = [[0, e, h]_{\circ}, [0, e', h']_{\circ}, [0, k]_{\circ}]_{\circ}$

and $\mathfrak{F}c\text{-def}$: $\mathfrak{F}(\text{ObjMap})(c) = [0, e, h]_0$
and $\mathfrak{F}d\text{-def}$: $\mathfrak{F}(\text{ObjMap})(d) = [0, e', h']_0$
and k : $k : e \mapsto_{\mathfrak{A}} e'$
and h : $h : x \mapsto_{\mathfrak{X}} \mathfrak{G}(\text{ObjMap})(e)$
and h' : $h' : x \mapsto_{\mathfrak{X}} \mathfrak{G}(\text{ObjMap})(e')$
and $[\text{cat-cs-simps}]$: $\mathfrak{G}(\text{ArrMap})(k) \circ_{A\mathfrak{X}} h = h'$
by *metis*
from *that* **have** $c \in_0 \mathfrak{J}(\text{Obj})$ **by** *auto*
from $\psi\text{-f}[OF \text{ this}]$ *that* $k f h$ **have** $[\text{symmetric}, \text{cat-cs-simps}]$:
 $h = \mathfrak{G}(\text{ArrMap})(\tau(\text{NTMap})(c)) \circ_{A\mathfrak{X}} f$
by
(
cs-prems
cs-simp: *cat-cs-simps* $\psi\text{-NTMap-app}$ $\mathfrak{F}c\text{-def}$ **cs-intro**: *cat-cs-intros*
)
from *that* **have** $d \in_0 \mathfrak{J}(\text{Obj})$ **by** *auto*
from $\psi\text{-f}[OF \text{ this}]$ *that* $k f h'$ **have** $[\text{symmetric}, \text{cat-cs-simps}]$:
 $h' = \mathfrak{G}(\text{ArrMap})(\tau(\text{NTMap})(d)) \circ_{A\mathfrak{X}} f$
by
(
cs-prems **cs-shallow**
cs-simp: *cat-cs-simps* $\psi\text{-NTMap-app}$ $\mathfrak{F}d\text{-def}$ **cs-intro**: *cat-cs-intros*
)
note $\tau.\text{cat-cone-Comp-commute}[\text{cat-cs-simps del}]$
from $\tau.\text{ntcf-Comp-commute}[OF \text{ that}]$ *that* $\text{assms}(\beta) k h h'$
have $[\text{symmetric}, \text{cat-cs-simps}]$: $\tau(\text{NTMap})(d) = k \circ_{A\mathfrak{A}} \tau(\text{NTMap})(c)$
by
(
cs-prems
cs-simp: *cat-comma-cs-simps* *cat-cs-simps* $\mathfrak{F}g\text{-def}$ $\mathfrak{F}c\text{-def}$ $\mathfrak{F}d\text{-def}$
cs-intro: *cat-cs-intros* *cat-comma-cs-intros*
)
from *that* $f \mathfrak{F}g k h h'$ **show** *?thesis*
unfolding $\mathfrak{F}g\text{-def}$ $\mathfrak{F}c\text{-def}$ $\mathfrak{F}d\text{-def}$
by
(
cs-concl
cs-simp:
cat-comma-cs-simps *cat-cs-simps*
 $\psi\text{-NTMap-app}$ $\sigma\text{-NTMap-app}$ $\mathfrak{F}g\text{-def}$ $\mathfrak{F}c\text{-def}$ $\mathfrak{F}d\text{-def}$
cs-intro: *cat-cs-intros* *cat-comma-cs-intros*
)
qed
qed
(
use f **in**
 \langle
cs-concl *cs-shallow*
cs-intro: *cat-cs-intros* *cat-lim-cs-intros* *cat-comma-cs-intros*
cs-simp: *cat-cs-simps*
 \rangle
)
)
have $\tau\sigma$: $\tau = x \circ_{\square CF} \mathfrak{G} \circ_{CF\text{-NTCF}} \sigma$
proof(*rule ntcf-eqI*)
show $\tau : cf\text{-const} \mathfrak{J} \mathfrak{A} b \mapsto_{CF} x \circ_{\square CF} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{A}$
by (*rule* $\tau.\text{is-ntcf-axioms}$)
from f **show** $x \circ_{\square CF} \mathfrak{G} \circ_{CF\text{-NTCF}} \sigma :$

$cf\text{-const } \mathfrak{J} \mathfrak{A} b \mapsto_{CF} x \circ \sqcap_{CF} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{A}$
by
(

 cs-concl
 cs-simp: *cat-cs-simps cat-comma-cs-simps*
 cs-intro: *cat-lim-cs-intros cat-cs-intros cat-comma-cs-intros*
)

have *dom-lhs*: $\mathcal{D}_\circ (\tau(\downarrow NTMap)) = \mathfrak{J}(\downarrow Obj)$
 by (*cs-concl cs-shallow cs-simp:* *cat-cs-simps*)
have *dom-rhs*: $\mathcal{D}_\circ ((x \circ \sqcap_{CF} \mathfrak{G} \circ_{CF-NTCF} \sigma)(\downarrow NTMap)) = \mathfrak{J}(\downarrow Obj)$
 by (*cs-concl cs-shallow cs-simp:* *cat-cs-simps cs-intro:* *cat-cs-intros*)
show $\tau(\downarrow NTMap) = (x \circ \sqcap_{CF} \mathfrak{G} \circ_{CF-NTCF} \sigma)(\downarrow NTMap)$
proof(*rule vsv-eqI, unfold dom-lhs dom-rhs*)
 fix a **assume** *prems'*: $a \in_\circ \mathfrak{J}(\downarrow Obj)$
 then have $\mathfrak{F}(\downarrow ObjMap)(\downarrow a) \in_\circ x \downarrow_{CF} \mathfrak{G}(\downarrow Obj)$
 by (*cs-concl cs-shallow cs-intro:* *cat-cs-intros*)
 from $\mathfrak{G}.cat\text{-obj-cf-comma-ObjE}[OF \text{ this } assms(\mathfrak{J})]$ **obtain** $c \ g$
 where $\mathfrak{F}a\text{-def}$: $\mathfrak{F}(\downarrow ObjMap)(\downarrow a) = [0, c, g]_\circ$
 and c : $c \in_\circ \mathfrak{A}(\downarrow Obj)$
 and g : $g : x \mapsto_{\mathfrak{X}} \mathfrak{G}(\downarrow ObjMap)(\downarrow c)$
 by *auto*
 from $\psi\text{-f}[OF \text{ prems}']$ *prems'* $f \ g$ **have** [*symmetric, cat-cs-simps*]:
 $g = \mathfrak{G}(\downarrow ArrMap)(\tau(\downarrow NTMap)(\downarrow a)) \circ_{A\mathfrak{X}} f$
 by
 (

 cs-prems cs-shallow
 cs-simp: *cat-cs-simps $\psi\text{-NTMap-app}$ $\mathfrak{F}a\text{-def}$ cs-intro:* *cat-cs-intros*
)

 with *assms*(\mathfrak{J}) *prems'* $c \ g \ f$ **show**
 $\tau(\downarrow NTMap)(\downarrow a) = (x \circ \sqcap_{CF} \mathfrak{G} \circ_{CF-NTCF} \sigma)(\downarrow NTMap)(\downarrow a)$
 by
 (

 cs-concl cs-shallow
 cs-simp:
 cat-comma-cs-simps cat-cs-simps
 $\mathfrak{F}a\text{-def}$ $\psi\text{-NTMap-app}$ $\sigma\text{-NTMap-app}$
 cs-intro: *cat-cs-intros cat-comma-cs-intros*
)

 qed (*simp-all add: $\tau.ntcf\text{-NTMap-vsv}$ cat-cs-intros*)
qed *simp-all*

from f **have** $b\text{-def}$: $b = x \circ \sqcap_{CF} \mathfrak{G}(\downarrow ObjMap)(\downarrow 0, b, f)_\bullet$
 by (*cs-concl cs-simp:* *cat-comma-cs-simps cs-intro:* *cat-cs-intros*)

show $\sigma a\text{-unique}$: $\exists ! \sigma a. \exists \sigma a.$
 $\sigma a = \langle \sigma, a \rangle \wedge$
 $\sigma : a <_{CF.cone} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} x \downarrow_{CF} \mathfrak{G} \wedge$
 $\tau = x \circ \sqcap_{CF} \mathfrak{G} \circ_{CF-NTCF} \sigma \wedge b = x \circ \sqcap_{CF} \mathfrak{G}(\downarrow ObjMap)(\downarrow a)$
proof
(

 intro exII[where $a = \langle \sigma, [0, b, f]_\circ \rangle$] exI conjI, simp only;
 (*elim exE conjE*)?
)

show $\sigma : [0, b, f]_\circ <_{CF.cone} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} x \downarrow_{CF} \mathfrak{G}$
 by (*rule $\sigma.is\text{-cat-cone-axioms}$*)
show $\tau = x \circ \sqcap_{CF} \mathfrak{G} \circ_{CF-NTCF} \sigma$ **by** (*rule $\tau\sigma$*)
show $b = x \circ \sqcap_{CF} \mathfrak{G}(\downarrow ObjMap)(\downarrow 0, b, f)_\bullet$ **by** (*rule $b\text{-def}$*)

fix $\sigma a \sigma' a$ **assume** prems' :
 $\sigma a = \langle \sigma', a \rangle$
 $\sigma' : a \triangleleft_{CF.cone} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} x \downarrow_{CF} \mathfrak{G}$
 $\tau = x \circ_{\square_{CF}} \mathfrak{G} \circ_{CF-NTCF} \sigma'$
 $b = x \circ_{\square_{CF}} \mathfrak{G}(\text{ObjMap})(a)$
interpret σ' : *is-cat-cone* $\alpha a \mathfrak{J} \langle x \downarrow_{CF} \mathfrak{G} \rangle \mathfrak{F} \sigma'$ **by** (*rule* $\text{prems}'(2)$)

from $\mathfrak{G}.\text{cat-obj-cf-comma-ObjE}[OF \sigma'.\text{cat-cone-obj assms}(3)]$ **obtain** $c g$
where $a\text{-def}'' : a = [0, c, g]_{\circ}$
and $c' : c \in_{\circ} \mathfrak{A}(\text{Obj})$
and $g' : g : x \mapsto_{\mathfrak{X}} \mathfrak{G}(\text{ObjMap})(c)$
by *auto*
from $\text{prems}'(4)$ $c' g'$ *assms*(3) **have** $bc : b = c$
by
(

cs-prems **cs-shallow**
cs-simp: *cat-comma-cs-simps* $a\text{-def}''$ **cs-intro**: *cat-comma-cs-intros*
)

with $a\text{-def}'' c' g'$ **have** $a\text{-def}' : a = [0, b, g]_{\circ}$
and $b : b \in_{\circ} \mathfrak{A}(\text{Obj})$
and $g : g : x \mapsto_{\mathfrak{X}} \mathfrak{G}(\text{ObjMap})(b)$
by *auto*

from $\text{prems}'(3)$ **have** $\tau\text{-eq-x}\mathfrak{G}\text{-}\sigma'$:
 $\tau(\text{NTMap})(j) = (x \circ_{\square_{CF}} \mathfrak{G} \circ_{CF-NTCF} \sigma')(\text{NTMap})(j)$ **for** j
by *simp*

have $\psi(\text{NTMap})(j) = (\mathfrak{G} \circ_{CF-NTCF} \tau)(\text{NTMap})(j) \circ_{A\mathfrak{X}} g$
if $j \in_{\circ} \mathfrak{J}(\text{Obj})$ **for** j
proof–
from *that* **have** $\sigma'\text{-}j : \sigma'(\text{NTMap})(j) : [0, b, g]_{\circ} \mapsto_x \downarrow_{CF} \mathfrak{G} \mathfrak{F}(\text{ObjMap})(j)$
by
(

cs-concl **cs-shallow**
cs-simp: *cat-cs-simps* $a\text{-def}'[\text{symmetric}]$ **cs-intro**: *cat-cs-intros*
)

from $\mathfrak{G}.\text{cat-obj-cf-comma-is-arrE}[OF \text{this}]$ **obtain** $e h k$
where $\sigma'\text{-}j\text{-def} : \sigma'(\text{NTMap})(j) = [[0, b, g]_{\circ}, [0, e, h]_{\circ}, [0, k]_{\circ}]_{\circ}$
and $\mathfrak{F}j\text{-def} : \mathfrak{F}(\text{ObjMap})(j) = [0, e, h]_{\circ}$
and $k : k : b \mapsto_{\mathfrak{A}} e$
and $h : h : x \mapsto_{\mathfrak{X}} \mathfrak{G}(\text{ObjMap})(e)$
and $[\text{cat-cs-simps}] : \mathfrak{G}(\text{ArrMap})(k) \circ_{A\mathfrak{X}} g = h$
by (*metis* $\mathfrak{G}.\text{cat-obj-cf-comma-is-arrD}(4,7)$ $\sigma'\text{-}j$ *assms*(3))
from *that* $\sigma'\text{-}j$ **show** *?thesis*
unfolding $\sigma'\text{-}j\text{-def}$
by
(

cs-concl **cs-shallow**
cs-simp:
cat-cs-simps *cat-comma-cs-simps*
 $\sigma'\text{-}j\text{-def}$ $\psi\text{-NTMap-app}$ $\mathfrak{F}j\text{-def}$ $\text{prems}'(3)$
cs-intro: *cat-cs-intros*
)

qed
from $f\text{-unique}[OF g \text{this}]$ **have** $gf : g = f$.
with $a\text{-def}'$ **have** $a\text{-def} : a = [0, b, f]_{\circ}$ **by** *simp*

have $\sigma\sigma': \sigma = \sigma'$
proof(rule ntcf-eqI)
show $\sigma : cf\text{-const } \mathfrak{J} (x \downarrow_{CF} \mathfrak{G}) [0, b, f]_{\circ} \mapsto_{CF} \mathfrak{F} : \mathfrak{J} \mapsto_{\mapsto_{C\alpha}} x \downarrow_{CF} \mathfrak{G}$
by (cs-concl cs-shallow cs-intro: cat-cs-intros)
then have dom-lhs: $\mathcal{D}_{\circ} (\sigma(\downarrow_{NTMap})) = \mathfrak{J}(\downarrow_{Obj})$
by (cs-concl cs-shallow cs-simp: cat-cs-simps)
show $\sigma' : cf\text{-const } \mathfrak{J} (x \downarrow_{CF} \mathfrak{G}) [0, b, f]_{\circ} \mapsto_{CF} \mathfrak{F} : \mathfrak{J} \mapsto_{\mapsto_{C\alpha}} x \downarrow_{CF} \mathfrak{G}$
by (cs-concl cs-shallow cs-simp: a-def cs-intro: cat-cs-intros)
then have dom-rhs: $\mathcal{D}_{\circ} (\sigma'(\downarrow_{NTMap})) = \mathfrak{J}(\downarrow_{Obj})$
by (cs-concl cs-shallow cs-simp: cat-cs-simps)
show $\sigma(\downarrow_{NTMap}) = \sigma'(\downarrow_{NTMap})$
proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
fix j **assume** $prems'' : j \in_{\circ} \mathfrak{J}(\downarrow_{Obj})$
then have $\sigma'-j : \sigma'(\downarrow_{NTMap})(j) : [0, b, f]_{\circ} \mapsto_x \downarrow_{CF} \mathfrak{G} \mathfrak{F}(\downarrow_{ObjMap})(j)$
by
(

cs-concl cs-shallow
cs-simp: cat-cs-simps a-def[symmetric] gf[symmetric]
cs-intro: cat-cs-intros
)

from $\mathfrak{G}.cat\text{-obj-cf-comma-is-arrE}[OF \text{ this}]$ **obtain** $e h k$
where $\sigma'-j\text{-def} : \sigma'(\downarrow_{NTMap})(j) = [[0, b, f]_{\circ}, [0, e, h]_{\circ}, [0, k]_{\circ}]_{\circ}$
and $\mathfrak{F}j\text{-def} : \mathfrak{F}(\downarrow_{ObjMap})(j) = [0, e, h]_{\circ}$
and $k : k : b \mapsto_{\mathfrak{A}} e$
and $h : h : x \mapsto_{\mathfrak{X}} \mathfrak{G}(\downarrow_{ObjMap})(e)$
and $[cat\text{-cs-simps}] : \mathfrak{G}(\downarrow_{ArrMap})(k) \circ_{A\mathfrak{X}} f = h$
by (metis $\mathfrak{G}.cat\text{-obj-cf-comma-is-arrD}(4,7)$ $\sigma'-j$ assms(3))
from $prems'' k h$ assms(3) $f h$ **show** $\sigma(\downarrow_{NTMap})(j) = \sigma'(\downarrow_{NTMap})(j)$
by
(

cs-concl cs-shallow
cs-simp:
cat-cs-simps cat-comma-cs-simps
 $\tau\text{-eq-x}\mathfrak{G}\text{-}\sigma' \psi\text{-NTMap-app } \sigma\text{-NTMap-app } \mathfrak{F}j\text{-def } \sigma'-j\text{-def}$
cs-intro: cat-cs-intros cat-comma-cs-intros
)

qed (cs-concl cs-shallow cs-intro: V-cs-intros)
qed simp-all
show $\sigma a = \langle \sigma, [[\]_{\circ}, b, f]_{\circ} \rangle$ **unfolding** $prems''(1)$ $\sigma\sigma'$ a-def **by** simp
qed

show $\sigma' : a' <_{CF.lim} \mathfrak{F} : \mathfrak{J} \mapsto_{\mapsto_{C\alpha}} x \downarrow_{CF} \mathfrak{G}$
if $\sigma' : a' <_{CF.cone} \mathfrak{F} : \mathfrak{J} \mapsto_{\mapsto_{C\alpha}} x \downarrow_{CF} \mathfrak{G}$
and $\tau = x \circ_{\square_{CF}} \mathfrak{G} \circ_{CF\text{-NTCF}} \sigma'$
and $b = x \circ_{\square_{CF}} \mathfrak{G}(\downarrow_{ObjMap})(a')$
for $\sigma' a'$
proof(rule is-cat-limitI)

interpret $\sigma' : is\text{-cat-cone } \alpha a' \mathfrak{J} \langle x \downarrow_{CF} \mathfrak{G} \rangle \mathfrak{F} \sigma'$ **by** (rule that(1))

from $\sigma.is\text{-cat-cone-axioms}$ $\tau\sigma$ b-def that σa -unique **have**

$\langle \sigma', a' \rangle = \langle \sigma, [0, b, f]_{\circ} \rangle$

by metis

then have $\sigma'\text{-def} : \sigma' = \sigma$ **and** $a'\text{-def} : a' = [0, b, f]_{\circ}$ **by** simp-all

show $\sigma' : a' <_{CF.cone} \mathfrak{F} : \mathfrak{J} \mapsto_{\mapsto_{C\alpha}} x \downarrow_{CF} \mathfrak{G}$

by (rule $\sigma'.is\text{-cat-cone-axioms}$)

fix $\sigma'' a''$ **assume** *prems*: $\sigma'' : a'' <_{CF.cone} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} x \downarrow_{CF} \mathfrak{G}$
then interpret σ'' : *is-cat-cone* $\alpha a'' \mathfrak{J} \langle x \downarrow_{CF} \mathfrak{G} \rangle \mathfrak{F} \sigma''$.
from $\mathfrak{G}.cat-obj-cf-comma-ObjE[OF \sigma''.cat-cone-obj \text{assms}(\mathfrak{J})]$ **obtain** $b' f'$
where $a''\text{-def}$: $a'' = [0, b', f']_o$
and b' : $b' \in_o \mathfrak{A}(Obj)$
and f' : $f' : x \mapsto_{\mathfrak{X}} \mathfrak{G}(ObjMap)(b')$
by *auto*
from $b' f'$ **have** $x\mathfrak{G}\text{-}A'$: $x \circ_{\square_{CF}} \mathfrak{G}(ObjMap)(a'') = b'$
unfolding $a''\text{-def}$
by
(
cs-concl **cs-shallow**
cs-simp: *cat-comma-cs-simps* **cs-intro**: *cat-comma-cs-intros*
)

from $\tau.cat\text{-lim-unique-cone}'[$
OF cf-ntcf-comp-cf-cat-cone[*OF prems* $x\mathfrak{G}.is\text{-functor-axioms}$],
unfolded $x\mathfrak{G}\text{-}A'$
]
obtain h **where** $h : h : b' \mapsto_{\mathfrak{A}} b$
and $x\mathfrak{G}\text{-}\sigma''\text{-app}$: $\bigwedge j. j \in_o \mathfrak{J}(Obj) \implies$
 $(x \circ_{\square_{CF}} \mathfrak{G} \circ_{CF\text{-}NTCF} \sigma'')(NTMap)(j) = \tau(NTMap)(j) \circ_{A\mathfrak{A}} h$
and $h\text{-unique}$:
[[
 $h' : b' \mapsto_{\mathfrak{A}} b$;
 $\bigwedge j. j \in_o \mathfrak{J}(Obj) \implies$
 $(x \circ_{\square_{CF}} \mathfrak{G} \circ_{CF\text{-}NTCF} \sigma'')(NTMap)(j) = \tau(NTMap)(j) \circ_{A\mathfrak{A}} h'$
]] $\implies h' = h$
for h'
by *metis*

define F **where** $F = [a'', a', [0, h]_o]$.

show $\exists ! F'$.

$F' : a'' \mapsto_{x \downarrow_{CF} \mathfrak{G}} a' \wedge \sigma'' = \sigma' \cdot_{NTCF} ntcf\text{-const} \mathfrak{J} (x \downarrow_{CF} \mathfrak{G}) F'$
unfolding $a''\text{-def}$ $a'\text{-def}$ $\sigma'\text{-def}$
proof(*intro exI conjI*; (*elim conjE*)?)
from $f' h$ **have** $\mathfrak{G}h\text{-}f'$: $\mathfrak{G}(ArrMap)(h) \circ_{A\mathfrak{X}} f' : x \mapsto_{\mathfrak{X}} \mathfrak{G}(ObjMap)(b)$
by (*cs-concl* **cs-shallow** **cs-intro**: *cat-cs-intros*)
have $\psi(NTMap)(j) = (\mathfrak{G} \circ_{CF\text{-}NTCF} \tau)(NTMap)(j) \circ_{A\mathfrak{X}} (\mathfrak{G}(ArrMap)(h) \circ_{A\mathfrak{X}} f')$
if $j \in_o \mathfrak{J}(Obj)$ **for** j
proof-
from *that* **have** $\sigma''\text{-}j$:
 $\sigma''(NTMap)(j) : [0, b', f']_o \mapsto_{x \downarrow_{CF} \mathfrak{G}} \mathfrak{F}(ObjMap)(j)$
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps* $a''\text{-def}$ [*symmetric*]
cs-intro: *cat-cs-intros*
)
from $\mathfrak{G}.cat-obj-cf-comma-is-arrE[OF \text{this}]$ **obtain** $e h' k$
where $\sigma''\text{-}j\text{-def}$: $\sigma''(NTMap)(j) = [[0, b', f']_o, [0, e, h']_o, [0, k]_o]$.
and $\mathfrak{F}j\text{-def}$: $\mathfrak{F}(ObjMap)(j) = [0, e, h']_o$.
and k : $k : b' \mapsto_{\mathfrak{A}} e$
and [*cat-cs-simps*]: $\mathfrak{G}(ArrMap)(k) \circ_{A\mathfrak{X}} f' = h'$
by (*metis* $\mathfrak{G}.cat-obj-cf-comma-is-arrD(4,7)$ $\sigma''\text{-}j$ *assms*(\mathfrak{J}))
from *that* $\sigma''\text{-}j$ **have** $\psi\text{-}NTMap\text{-}j$: $\psi(NTMap)(j) =$
 $(\mathfrak{G} \circ_{CF\text{-}NTCF} (x \circ_{\square_{CF}} \mathfrak{G} \circ_{CF\text{-}NTCF} \sigma''))(NTMap)(j) \circ_{A\mathfrak{X}} f'$

unfolding σ'' - j -def $\mathfrak{J}j$ -def
by
(

 cs-concl **cs-shallow**
 cs-simp:
 cat-cs-simps cat-comma-cs-simps σ'' - j -def $\mathfrak{J}j$ -def ψ -NTMap-app
 cs-intro: *cat-cs-intros*
)+
from *that* h f' **show** *?thesis*
unfolding ψ -NTMap- j
by
(

 cs-concl **cs-shallow**
 cs-simp:
 cat-cs-simps
 is-ntcf.cf-ntcf-comp-NTMap-app
 $x\mathfrak{G}$ - σ'' -app[*OF that*]
 cs-intro: *cat-cs-intros*
)

qed

from *f-unique*[*OF* $\mathfrak{G}h$ - f' *this*] **have** [*cat-cs-simps*]:
 $\mathfrak{G}(\downarrow\text{ArrMap})(h) \circ_{A\mathfrak{X}} f' = f$.

from *assms*(\mathfrak{J}) h f' f **show** $F: F : [0, b', f]_{\circ} \mapsto_x \downarrow_{CF} \mathfrak{G} [0, b, f]_{\circ}$
unfolding F -def a'' -def a' -def
by
(

 cs-concl **cs-shallow**
 cs-simp: *cat-cs-simps* **cs-intro:** *cat-comma-cs-intros*
)

show $\sigma'' = \sigma \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} (x \downarrow_{CF} \mathfrak{G}) F$
proof(*rule ntcf-eqI*)
show $\sigma'' : \text{cf-const } \mathfrak{J} (x \downarrow_{CF} \mathfrak{G}) a'' \mapsto_{CF} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} x \downarrow_{CF} \mathfrak{G}$
by (*rule* σ'' .*is-ntcf-axioms*)
then have *dom-lhs*: $\mathcal{D}_{\circ} (\sigma''(\downarrow\text{NTMap})) = \mathfrak{J}(\downarrow\text{Obj})$
by (*cs-concl* **cs-shallow** **cs-simp:** *cat-cs-simps*)
from F **show** $\sigma \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} (x \downarrow_{CF} \mathfrak{G}) F :$
 $\text{cf-const } \mathfrak{J} (x \downarrow_{CF} \mathfrak{G}) a'' \mapsto_{CF} \mathfrak{F} : \mathfrak{J} \mapsto_{C\alpha} x \downarrow_{CF} \mathfrak{G}$
unfolding a'' -def **by** (*cs-concl* **cs-shallow** **cs-intro:** *cat-cs-intros*)
then have *dom-rhs*:
 $\mathcal{D}_{\circ} ((\sigma \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} (x \downarrow_{CF} \mathfrak{G}) F)(\downarrow\text{NTMap})) = \mathfrak{J}(\downarrow\text{Obj})$
by (*cs-concl* **cs-simp:** *cat-cs-simps*)
show $\sigma''(\downarrow\text{NTMap}) = (\sigma \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} (x \downarrow_{CF} \mathfrak{G}) F)(\downarrow\text{NTMap})$
proof(*rule vsv-eqI, unfold dom-lhs dom-rhs*)
fix j **assume** *prems'*: $j \in_{\circ} \mathfrak{J}(\downarrow\text{Obj})$
then have σ'' - j :
 $\sigma''(\downarrow\text{NTMap})(j) : [0, b', f]_{\circ} \mapsto_x \downarrow_{CF} \mathfrak{G} \mathfrak{F}(\downarrow\text{ObjMap})(j)$
by
(

 cs-concl **cs-shallow**
 cs-simp: *cat-cs-simps* a'' -def[*symmetric*]
 cs-intro: *cat-cs-intros*
)

from \mathfrak{G} .*cat-obj-cf-comma-is-arrE*[*OF this*] **obtain** e h' k
where σ'' - j -def: $\sigma''(\downarrow\text{NTMap})(j) = [[0, b', f]_{\circ}, [0, e, h']_{\circ}, [0, k]_{\circ}]_{\circ}$
and $\mathfrak{J}j$ -def: $\mathfrak{F}(\downarrow\text{ObjMap})(j) = [0, e, h']_{\circ}$

```

    and k: k : b' ↦ℳ e
    and h': h' : x ↦ℳ ℑ(ObjMap)(|e|)
    and [cat-cs-simps]: ℑ(ArrMap)(|k|) ∘Aℳ f' = h'
  by (metis ℑ.cat-obj-cf-comma-is-arrD(4,7) σ''-j-assms(3))
from assms(3) prems' F k h' h f f' show
σ''(NTMap)(|j|) = (σ ∙NTCF ntcf-const ℑ (x ↓CF ℑ) F)(NTMap)(|j|)
by
(
  cs-concl
  cs-simp:
    cat-cs-simps cat-comma-cs-simps
    σ''-j-def xℑ-σ''-app[OF prems', symmetric]
    σ-NTMap-app F-def ℑj-def a''-def a'-def
  cs-intro: cat-cs-intros cat-comma-cs-intros
  cs-simp: ψ-f ψ-NTMap-app
)
qed (cs-concl cs-intro: V-cs-intros cat-cs-intros)+
qed simp-all
fix F' assume prems':
  F' : [0, b', f']o ↦x ↓CF ℑ [0, b, f]o
  σ'' = σ ∙NTCF ntcf-const ℑ (x ↓CF ℑ) F'
from ℑ.cat-obj-cf-comma-is-arrE[OF prems'(1) assms(3), simplified]
obtain k
  where F'-def: F' = [[0, b', f']o, [0, b, f]o, [0, k]o]o
  and k: k : b' ↦ℳ b
  and [cat-cs-simps]: ℑ(ArrMap)(|k|) ∘Aℳ f' = f
  by metis
have k = h
proof(rule h-unique[OF k])
  fix j assume prems'': j ∈o ℑ(Obj)
  then have ℑ(ObjMap)(|j|) ∈o x ↓CF ℑ(Obj)
  by (cs-concl cs-shallow cs-intro: cat-cs-intros)
  from ℑ.cat-obj-cf-comma-ObjE[OF this assms(3)] obtain c g
  where ℑj-def: ℑ(ObjMap)(|j|) = [0, c, g]o
  and c: c ∈o ℳ(Obj)
  and g: g : x ↦ℳ ℑ(ObjMap)(|c|)
  by auto
  from prems'' prems'(1) assms(3) c g f show
  (x o∩CF ℑ ∘CF-NTCF σ'')(NTMap)(|j|) = τ(NTMap)(|j|) ∘Aℳ k
  unfolding prems'(2) τσ F'-def
  by
  (
    cs-concl
    cs-simp: cat-comma-cs-simps cat-cs-simps
    cs-intro: cat-cs-intros cat-comma-cs-intros
    cs-simp: ψ-f σ-NTMap-app ℑj-def
  )
qed
then show F' = F unfolding F'-def F-def a''-def a'-def by simp
qed
qed
qed

```

12 Category *Set* and universal constructions

12.1 Discrete functor with tiny maps to the category *Set*

```

lemma (in  $\mathcal{Z}$ ) tm-cf-discrete-cat-Set-if-VLambda-in-Vset:
  assumes  $VLambda\ I\ F\ \epsilon_o\ Vset\ \alpha$ 
  shows tm-cf-discrete  $\alpha\ I\ F$  (cat-Set  $\alpha$ )
proof (intro tm-cf-discreteI)
  from assms have vrange-F-in-Vset:  $\mathcal{R}_o\ (VLambda\ I\ F)\ \epsilon_o\ Vset\ \alpha$ 
  by (auto intro: vrange-in-VsetI)
  show  $(\lambda i \in_o I. \text{cat-Set } \alpha (|CID|) (|F\ i|)) \in_o Vset\ \alpha$ 
  proof (rule vrelation.vrelation-Limit-in-VsetI)
    from assms show  $\mathcal{D}_o\ (\lambda i \in_o I. \text{cat-Set } \alpha (|CID|) (|F\ i|)) \in_o Vset\ \alpha$ 
    by (metis vdomain-VLambda\ vdomain-in-VsetI)
  define  $Q$  where
     $Q\ i =$ 
    (
      if  $i = 0$ 
      then  $VPow\ ((\bigcup_o i \in_o I. F\ i) \times_o (\bigcup_o i \in_o I. F\ i))$ 
      else  $set\ (F\ 'elts\ I)$ 
    )
  for  $i :: V$ 
  have  $\mathcal{R}_o\ (\lambda i \in_o I. \text{cat-Set } \alpha (|CID|) (|F\ i|)) \subseteq_o (\prod_o i \in_o set\ \{0, 1_N, 2_N\}. Q\ i)$ 
  proof (intro vsubsetI, unfold cat-Set-components)
    fix  $y$  assume  $y \in_o \mathcal{R}_o\ (\lambda i \in_o I. VLambda\ (Vset\ \alpha)\ id-Set\ (F\ i))$ 
    then obtain  $i$  where  $i: i \in_o I$ 
      and  $y\text{-def}: y = VLambda\ (Vset\ \alpha)\ id-Set\ (F\ i)$ 
    by auto
    from  $i$  have  $F\ i \in_o \mathcal{R}_o\ (VLambda\ I\ F)$  by auto
    with vrange-F-in-Vset have  $F\ i \in_o Vset\ \alpha$  by auto
    then have  $y\text{-def}: y = id-Set\ (F\ i)$  unfolding  $y\text{-def}$  by auto
    show  $y \in_o (\prod_o i \in_o set\ \{0, 1_N, 2_N\}. Q\ i)$ 
      unfolding  $y\text{-def}$ 
  proof (intro vproductI, unfold Ball-def; (intro allI impI)?)
    show  $\mathcal{D}_o\ (id-Rel\ (F\ i)) = set\ \{0, 1_N, 2_N\}$ 
    by (simp add: id-Rel-def incl-Rel-def three nat-omega-simps)
    fix  $j$  assume  $j \in_o set\ \{0, 1_N, 2_N\}$ 
    then consider  $\langle j = 0 \rangle \mid \langle j = 1_N \rangle \mid \langle j = 2_N \rangle$  by auto
    then show  $id-Rel\ (F\ i) (|j|) \in_o Q\ j$ 
  proof cases
    case 1
      from  $i$  show ?thesis
      unfolding 1
      by
      (
        subst arr-field-simps(1)[symmetric],
        unfold id-Rel-components Q-def
      )
      force
    next
      case 2
      from  $i$  show ?thesis
      unfolding 2
      by
      (
        subst arr-field-simps(2)[symmetric],
        unfold id-Rel-components Q-def
      )
  end
end

```

```

      auto
    next
      case 3
      from i show ?thesis
      unfolding 3
      by
        (
          subst arr-field-simps(3)[symmetric],
          unfold id-Rel-components Q-def
        )
      auto
    qed
  qed (auto simp: id-Rel-def cat-Set-cs-intros)
qed
moreover have ( $\prod_{i \in \circ} \text{set } \{0, 1_{\mathbb{N}}, 2_{\mathbb{N}}\}. Q i) \in_{\circ} Vset \alpha$ 
proof(rule Limit-vproduct-in-VsetI)
  show  $\text{set } \{0, 1_{\mathbb{N}}, 2_{\mathbb{N}}\} \in_{\circ} Vset \alpha$  unfolding three[symmetric] by simp
  from assms have  $VPow ((\bigcup_{i \in \circ} I. F i) \times_{\circ} (\bigcup_{i \in \circ} I. F i)) \in_{\circ} Vset \alpha$ 
  by
    (
      intro
      Limit-VPow-in-VsetI
      Limit-vtimes-in-VsetI
      Limit-vifunion-in-Vset-if-VLambda-in-VsetI
    )
    auto
  then show  $Q i \in_{\circ} Vset \alpha$  if  $i \in_{\circ} \text{set } \{0, 1_{\mathbb{N}}, 2_{\mathbb{N}}\}$  for i
  using that vrange-VLambda
  by (auto intro!: vrange-F-in-Vset simp: Q-def nat-omega-simps)
qed auto
ultimately show  $\mathcal{R}_{\circ} (\lambda i \in_{\circ} I. \text{cat-Set } \alpha(\downarrow CI d)(\downarrow F i)) \in_{\circ} Vset \alpha$ 
  by (meson vsubset-in-VsetI)
qed auto
fix i assume prems:  $i \in_{\circ} I$ 
from assms have  $\mathcal{R}_{\circ} (VLambda I F) \in_{\circ} Vset \alpha$  by (auto simp: vrange-in-VsetI)
moreover from prems have  $F i \in_{\circ} \mathcal{R}_{\circ} (VLambda I F)$  by auto
ultimately show  $F i \in_{\circ} \text{cat-Set } \alpha(\downarrow Obj)$  unfolding cat-Set-components by auto
qed (cs-concl cs-shallow cs-intro: cat-cs-intros assms)+

```

12.2 Product cone and coproduct cocone for the category *Set*

12.2.1 Definition and elementary properties

definition *ntcf-Set-obj-prod* :: $V \Rightarrow V \Rightarrow (V \Rightarrow V) \Rightarrow V$
where *ntcf-Set-obj-prod* α $I F = \text{ntcf-obj-prod-base}$
 $(\text{cat-Set } \alpha) I F (\prod_{i \in \circ} I. F i) (\lambda i. \text{vprojection-arrow } I F i)$

definition *ntcf-Set-obj-coproduct* :: $V \Rightarrow V \Rightarrow (V \Rightarrow V) \Rightarrow V$
where *ntcf-Set-obj-coproduct* α $I F = \text{ntcf-obj-coproduct-base}$
 $(\text{cat-Set } \alpha) I F (\prod_{i \in \circ} I. F i) (\lambda i. \text{vcinjection-arrow } I F i)$

Components.

lemma *ntcf-Set-obj-prod-components*:
shows *ntcf-Set-obj-prod* α $I F(\downarrow NTMap) =$
 $(\lambda i \in_{\circ} :_C I(\downarrow Obj). \text{vprojection-arrow } I F i)$
and *ntcf-Set-obj-prod* α $I F(\downarrow NTDom) =$
 $\text{cf-const } (:_C I) (\text{cat-Set } \alpha) (\prod_{i \in \circ} I. F i)$
and *ntcf-Set-obj-prod* α $I F(\downarrow NTCod) = \text{:=: } I F (\text{cat-Set } \alpha)$

and $ntcf\text{-}Set\text{-}obj\text{-}prod \alpha I F(NTDGD\text{Dom}) = :_C I$
and $ntcf\text{-}Set\text{-}obj\text{-}prod \alpha I F(NTDGC\text{od}) = cat\text{-}Set \alpha$
unfolding $ntcf\text{-}Set\text{-}obj\text{-}prod\text{-}def$ $ntcf\text{-}obj\text{-}prod\text{-}base\text{-}components$ **by** $simp\text{-}all$

lemma $ntcf\text{-}Set\text{-}obj\text{-}coprod\text{-}components$:

shows $ntcf\text{-}Set\text{-}obj\text{-}coprod \alpha I F(NTMap) =$
 $(\lambda i \in \circ :_C I(\text{Obj}). \text{vcinjection}\text{-}arrow I F i)$
and $ntcf\text{-}Set\text{-}obj\text{-}coprod \alpha I F(NTD\text{Dom}) = \text{::}: I F (cat\text{-}Set \alpha)$
and $ntcf\text{-}Set\text{-}obj\text{-}coprod \alpha I F(NTC\text{od}) =$
 $cf\text{-}const (:_C I) (cat\text{-}Set \alpha) (\coprod \circ i \in \circ I. F i)$
and $ntcf\text{-}Set\text{-}obj\text{-}coprod \alpha I F(NTDGD\text{Dom}) = :_C I$
and $ntcf\text{-}Set\text{-}obj\text{-}coprod \alpha I F(NTDGC\text{od}) = cat\text{-}Set \alpha$
unfolding $ntcf\text{-}Set\text{-}obj\text{-}coprod\text{-}def$ $ntcf\text{-}obj\text{-}coprod\text{-}base\text{-}components$ **by** $simp\text{-}all$

12.2.2 Natural transformation map

mk-VLambda $ntcf\text{-}Set\text{-}obj\text{-}prod\text{-}components(1)$
 $|vsv\ ntcf\text{-}Set\text{-}obj\text{-}prod\text{-}NTMap\text{-}vsv[cat\text{-}cs\text{-}intros]]$
 $|vdomain\ ntcf\text{-}Set\text{-}obj\text{-}prod\text{-}NTMap\text{-}vdomain[cat\text{-}cs\text{-}simps]]$
 $|app\ ntcf\text{-}Set\text{-}obj\text{-}prod\text{-}NTMap\text{-}app[cat\text{-}cs\text{-}simps]]$

mk-VLambda $ntcf\text{-}Set\text{-}obj\text{-}coprod\text{-}components(1)$
 $|vsv\ ntcf\text{-}Set\text{-}obj\text{-}coprod\text{-}NTMap\text{-}vsv[cat\text{-}cs\text{-}intros]]$
 $|vdomain\ ntcf\text{-}Set\text{-}obj\text{-}coprod\text{-}NTMap\text{-}vdomain[cat\text{-}cs\text{-}simps]]$
 $|app\ ntcf\text{-}Set\text{-}obj\text{-}coprod\text{-}NTMap\text{-}app[cat\text{-}cs\text{-}simps]]$

12.2.3 Product cone for the category Set is a universal cone and product cocone for the category Set is a universal cocone

lemma (in \mathcal{Z}) $tm\text{-}cf\text{-}discrete\text{-}ntcf\text{-}obj\text{-}prod\text{-}base\text{-}is\text{-}cat\text{-}obj\text{-}prod$:

— See Theorem 5.2 in Chapter Introduction in [6].

assumes $VLambda I F \epsilon_\circ Vset \alpha$

shows $ntcf\text{-}Set\text{-}obj\text{-}prod \alpha I F :$

$(\prod \circ i \in \circ I. F i) <_{CF.\prod} F : I \mapsto \mapsto_{C\alpha} cat\text{-}Set \alpha$

proof($intro\ is\text{-}cat\text{-}obj\text{-}prodI\ is\text{-}cat\text{-}limitI$)

interpret $Set: tm\text{-}cf\text{-}discrete \alpha I F \langle cat\text{-}Set \alpha \rangle$

by ($rule\ tm\text{-}cf\text{-}discrete\text{-}cat\text{-}Set\text{-}if\text{-}VLambda\text{-}in\text{-}Vset[OF\ assms]$)

let $?F = \langle ntcf\text{-}Set\text{-}obj\text{-}prod \alpha I F \rangle$

show $cf\text{-}discrete \alpha I F (cat\text{-}Set \alpha)$

by ($auto\ simp: cat\text{-}small\text{-}discrete\text{-}cs\text{-}intros$)

show $F\text{-}is\text{-}cat\text{-}cone: ?F :$

$(\prod \circ i \in \circ I. F i) <_{CF.cone} \text{::}: I F (cat\text{-}Set \alpha) : :_C I \mapsto \mapsto_{C\alpha} cat\text{-}Set \alpha$

unfolding $ntcf\text{-}Set\text{-}obj\text{-}prod\text{-}def$

proof($rule\ Set.tm\text{-}cf\text{-}discrete\text{-}ntcf\text{-}obj\text{-}prod\text{-}base\text{-}is\text{-}cat\text{-}cone$)

show $(\prod \circ i \in \circ I. F i) \epsilon_\circ cat\text{-}Set \alpha(\text{Obj})$

unfolding $cat\text{-}Set\text{-}components$

by

(

$intro$

$Limit\text{-}vproduct\text{-}in\text{-}Vset\text{-}if\text{-}VLambda\text{-}in\text{-}VsetI$

$Set.tm\text{-}cf\text{-}discrete\text{-}ObjMap\text{-}in\text{-}Vset$

)

$auto$

qed ($intro\ vprojection\text{-}arrow\text{-}is\text{-}arr\ Set.tm\text{-}cf\text{-}discrete\text{-}ObjMap\text{-}in\text{-}Vset$)

interpret F : *is-cat-cone*

$\alpha \langle \prod_{\circ i \in_{\circ} I}. F i \rangle \langle :_C I \rangle \langle \text{cat-Set } \alpha \rangle \langle :- \rangle : I F (\text{cat-Set } \alpha) \langle ?F \rangle$
by (*rule F-is-cat-cone*)

fix $\pi' P'$ **assume** *prems*:

$\pi' : P' \langle :_{CF} \text{cone} \rangle :- \rangle : I F (\text{cat-Set } \alpha) : :_C I \mapsto \mapsto_{C\alpha} \text{cat-Set } \alpha$

let $?\pi' i = \langle \lambda i. \pi' (\text{NTMap}) (i) \rangle$

let $?up' = \langle \text{cat-Set-obj-prod-up } I F P' ?\pi' i \rangle$

interpret π' : *is-cat-cone* $\alpha P' \langle :_C I \rangle \langle \text{cat-Set } \alpha \rangle \langle :- \rangle : I F (\text{cat-Set } \alpha) \langle \pi' \rangle$

by (*rule prems(1)*)

show $\exists ! f'$.

$f' : P' \mapsto_{\text{cat-Set } \alpha} (\prod_{\circ i \in_{\circ} I}. F i) \wedge$
 $\pi' = ?F \cdot_{NTCF} \text{ntcf-const} (:_C I) (\text{cat-Set } \alpha) f'$

proof(*intro ex1I conjI; (elim conjE) ?*)

show $up' : ?up' : P' \mapsto_{\text{cat-Set } \alpha} (\prod_{\circ i \in_{\circ} I}. F i)$

proof(*rule cat-Set-obj-prod-up-cat-Set-is-arr*)

show $P' \in_{\circ} \text{cat-Set } \alpha (\text{Obj})$ **by** (*auto intro: cat-cs-intros cat-lim-cs-intros*)

fix i **assume** $i \in_{\circ} I$

then show $\pi' (\text{NTMap}) (i) : P' \mapsto_{\text{cat-Set } \alpha} F i$

by

(

cs-concl cs-shallow

cs-simp:

the-cat-discrete-components(1)

cat-cs-simps cat-discrete-cs-simps

cs-intro: *cat-cs-intros*

)

qed (*rule assms*)

then have $P' : P' \in_{\circ} \text{cat-Set } \alpha (\text{Obj})$

by (*auto intro: cat-cs-intros cat-lim-cs-intros*)

have $\pi' i \cdot i : ?\pi' i i : P' \mapsto_{\text{cat-Set } \alpha} F i$ **if** $i \in_{\circ} I$ **for** i

using

$\pi'.\text{ntcf-NTMap-is-arr}$ [*unfolded the-cat-discrete-components(1), OF that*]
that

by

(

cs-prems cs-shallow cs-simp:

cat-cs-simps cat-discrete-cs-simps the-cat-discrete-components(1)

)

from *cat-Set-obj-prod-up-cat-Set-is-arr*[*OF P' assms(1) $\pi' i \cdot i$*] **have** $\pi' i$:

$\text{cat-Set-obj-prod-up } I F P' ?\pi' i : P' \mapsto_{\text{cat-Set } \alpha} (\prod_{\circ i \in_{\circ} I}. F i)$.

show $\pi' = ?F \cdot_{NTCF} \text{ntcf-const} (:_C I) (\text{cat-Set } \alpha) ?up'$

proof(*rule ntcf-eqI, rule $\pi'.\text{is-ntcf-axioms}$*)

from *F-is-cat-cone $\pi' i$* **show**

$?F \cdot_{NTCF} \text{ntcf-const} (:_C I) (\text{cat-Set } \alpha) ?up' :$

$\text{cf-const} (:_C I) (\text{cat-Set } \alpha) P' \mapsto_{CF} :- \rangle : I F (\text{cat-Set } \alpha) :$

$:_C I \mapsto \mapsto_{C\alpha} \text{cat-Set } \alpha$

by (*cs-concl cs-shallow cs-intro: cat-cs-intros*)

have *dom-lhs*: $\mathcal{D}_{\circ} (\pi' (\text{NTMap})) = :_C I (\text{Obj})$

by (*cs-concl cs-shallow cs-simp: cat-cs-simps*)
 from *F-is-cat-cone* $\pi' i$ have *dom-rhs*:
 $\mathcal{D}_\circ ((?F \cdot_{NTCF} \text{ntcf-const } (:_C I) (\text{cat-Set } \alpha) ?up')(\text{NTMap})) = :_C I(\text{Obj})$
 by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

 show $\pi'(\text{NTMap}) = (?F \cdot_{NTCF} \text{ntcf-const } (:_C I) (\text{cat-Set } \alpha) ?up')(\text{NTMap})$
 proof(*rule vsv-eqI, unfold dom-lhs dom-rhs*)
 fix *i* assume *prems'*: $i \in_\circ :_C I(\text{Obj})$
 then have *i*: $i \in_\circ I$ **unfolding** *the-cat-discrete-components* **by** *simp*
 have [*cat-cs-simps*]:
 $v\text{projection-arrow } I F i \circ_{A \text{cat-Set } \alpha} ?up' = \pi'(\text{NTMap})(i)$
 by
 (
 (
 rule *cat-Set-cf-comp-proj-obj-prod-up*[
 $OF P' \text{ assms } \pi' i \text{ } i, \text{ symmetric}$
]
)
 auto
)
 from $\pi' i$ *prems'* show $\pi'(\text{NTMap})(i) =$
 $(?F \cdot_{NTCF} \text{ntcf-const } (:_C I) (\text{cat-Set } \alpha) ?up')(\text{NTMap})(i)$
 by
 (
 cs-concl cs-shallow
cs-simp: *cat-cs-simps cat-Rel-cs-simps cs-intro: cat-cs-intros*
)
 qed (*auto simp: cat-cs-intros*)

qed *simp-all*

fix *f'* assume *prems*:
 $f' : P' \mapsto_{\text{cat-Set } \alpha} (\prod_{i \in_\circ I} F i)$
 $\pi' = ?F \cdot_{NTCF} \text{ntcf-const } (:_C I) (\text{cat-Set } \alpha) f'$
 from *prems*(2) have $\pi' \text{-eq-} F \text{-} f' : \pi'(\text{NTMap})(i)(\text{Arr Val})(a) =$
 $(?F \cdot_{NTCF} \text{ntcf-const } (:_C I) (\text{cat-Set } \alpha) f')(\text{NTMap})(i)(\text{Arr Val})(a)$
 if $i \in_\circ I$ and $a \in_\circ P'$ for *i a*
 by *simp*
 have [*cat-Set-cs-simps*]: $\pi'(\text{NTMap})(i)(\text{Arr Val})(a) = f'(\text{Arr Val})(a)(i)$
 if $i \in_\circ I$ and $a \in_\circ P'$ for *i a*
 using
 $\pi' \text{-eq-} F \text{-} f'$ [*OF that*]
assms prems that
vprojection-arrow-is-arr [*OF that(1) assms*]
 by
 (
 (*cs-prems cs-shallow*
cs-simp:
cat-Set-cs-simps
cat-cs-simps
vprojection-arrow-Arr Val-app
the-cat-discrete-components(1)
cs-intro: *cat-Set-cs-intros cat-cs-intros*
)
)

note $f' = \text{cat-Set-is-arrD}[OF \text{prems}(1)]$

note $up' = \text{cat-Set-is-arrD}[OF up']$

interpret $f' : \text{arr-Set } \alpha f'$ **by** (*rule f'(1)*)

interpret $u' : \text{arr-Set } \alpha \langle (\text{cat-Set-obj-prod-up } I F P' (\text{app } (\pi'(\text{NTMap})))) \rangle$

by (rule up'(1))

show $f' = ?up'$

proof(rule arr-Set-eqI[of α])

have dom-lhs: $\mathcal{D}_\circ (f'(\downarrow ArrVal)) = P'$ by (simp add: cat-Set-cs-simps f')

have dom-rhs:

$\mathcal{D}_\circ (cat-Set-obj-prod-up\ I\ F\ P' (app (\pi'(\downarrow NTMap))))(\downarrow ArrVal) = P'$

by (simp add: cat-Set-cs-simps up')

show $f'(\downarrow ArrVal) = cat-Set-obj-prod-up\ I\ F\ P' (app (\pi'(\downarrow NTMap))))(\downarrow ArrVal)$

proof(rule vsv-eqI, unfold dom-lhs dom-rhs)

fix a assume prems': $a \in_\circ P'$

from prems(1) prems' have $f'(\downarrow ArrVal)(\downarrow a) \in_\circ (\prod_\circ i \in_\circ I. F\ i)$

by (cs-concl cs-shallow cs-intro: cat-Set-cs-intros)

note $f'a = vproductD[OF this]$

from prems' have dom-rhs:

$\mathcal{D}_\circ (cat-Set-obj-prod-up\ I\ F\ P' (app (\pi'(\downarrow NTMap))))(\downarrow ArrVal)(\downarrow a) = I$

by (cs-concl cs-shallow cs-simp: cat-Set-cs-simps)

show $f'(\downarrow ArrVal)(\downarrow a) =$

$cat-Set-obj-prod-up\ I\ F\ P' (app (\pi'(\downarrow NTMap))))(\downarrow ArrVal)(\downarrow a)$

proof(rule vsv-eqI, unfold f'a dom-rhs)

fix i assume $i \in_\circ I$

with prems' show $f'(\downarrow ArrVal)(\downarrow a)(\downarrow i) =$

$cat-Set-obj-prod-up\ I\ F\ P' (app (\pi'(\downarrow NTMap))))(\downarrow ArrVal)(\downarrow a)(\downarrow i)$

by (cs-concl cs-shallow cs-simp: cat-Set-cs-simps)

qed (simp-all add: prems' f'a(1) cat-Set-obj-prod-up-ArrVal-app)

qed auto

qed (simp-all add: cat-Set-obj-prod-up-components f' up'(1))

qed

qed

lemma (in \mathcal{Z}) tm-cf-discrete-ntcf-obj-prod-base-is-tm-cat-obj-prod:

— See Theorem 5.2 in Chapter Introduction in [6].

assumes $VLambda\ I\ F \in_\circ Vset\ \alpha$

shows $ntcf-Set-obj-prod\ \alpha\ I\ F :$

$(\prod_\circ i \in_\circ I. F\ i) <_{CF.tm.\Pi} F : I \mapsto_{C.tm\alpha} cat-Set\ \alpha$

proof(intro is-tm-cat-obj-prodI)

from assms show $tm-cf-discrete\ \alpha\ I\ F (cat-Set\ \alpha)$

by (rule tm-cf-discrete-cat-Set-if-VLambda-in-Vset)

show $ntcf-Set-obj-prod\ \alpha\ I\ F :$

$vproduct\ I\ F <_{CF.lim} \mapsto : I\ F (cat-Set\ \alpha) : :_C\ I \mapsto_{C\alpha} cat-Set\ \alpha$

by

(

rule is-cat-obj-prodD[

OF tm-cf-discrete-ntcf-obj-prod-base-is-cat-obj-prod[OF assms]

]

)

qed

lemma (in \mathcal{Z}) tm-cf-discrete-ntcf-obj-coproduct-base-is-cat-obj-coproduct:

— See Theorem 5.2 in Chapter Introduction in [6].

assumes $VLambda\ I\ F \in_\circ Vset\ \alpha$

shows $ntcf-Set-obj-coproduct\ \alpha\ I\ F :$

$F >_{CF.\Pi} (\prod_\circ i \in_\circ I. F\ i) : I \mapsto_{C\alpha} cat-Set\ \alpha$

proof(intro is-cat-obj-coproductI is-cat-colimitI)

interpret $Set: tm-cf-discrete\ \alpha\ I\ F \langle cat-Set\ \alpha \rangle$

```

by (rule tm-cf-discrete-cat-Set-if-VLambda-in-Vset[OF assms])

let ?F = ⟨ntcf-Set-obj-coproduct α I F⟩

show cf-discrete α I F (cat-Set α)
  by (auto simp: cat-small-discrete-cs-intros)
show F-is-cat-cocone: ?F :
  :=: I F (cat-Set α) >CF.cocone (∐o i ∈o I. F i) : :C I ↦→ Cα cat-Set α
  unfolding ntcf-Set-obj-coproduct-def
proof(rule Set.tm-cf-discrete-ntcf-obj-coproduct-base-is-cat-cocone)
  show (∐o i ∈o I. F i) ∈o cat-Set α(|Obj|)
    unfolding cat-Set-components
    by
      (
        intro
          Limit-union-in-Vset-if-VLambda-in-VsetI
          Set.tm-cf-discrete-ObjMap-in-Vset
      )
    auto
qed (intro vcinjection-arrow-is-arr Set.tm-cf-discrete-ObjMap-in-Vset)
then interpret F: is-cat-cocone
  α ⟨∐o i ∈o I. F i⟩ ⟨: :C I⟩ ⟨cat-Set α⟩ ⟨:=: I F (cat-Set α)⟩ ⟨?F⟩ .

fix π' P' assume prems:
  π' : :=: I F (cat-Set α) >CF.cocone P' : :C I ↦→ Cα cat-Set α

let ?π'i = ⟨λi. π'(|NTMap|)(|i|)⟩
let ?up' = ⟨cat-Set-obj-coproduct-up I F P' ?π'i⟩

interpret π': is-cat-cocone α P' ⟨: :C I⟩ ⟨cat-Set α⟩ ⟨:=: I F (cat-Set α)⟩ π'
  by (rule prems(1))

show ∃!f'.
  f' : VSigma I F ↦cat-Set α P' ∧
  π' = ntcf-const (:C I) (cat-Set α) f' •NTCF ntcf-Set-obj-coproduct α I F
proof(intro exI conjI; (elim conjE)?)
  show up': ?up' : (∐o i ∈o I. F i) ↦cat-Set α P'
  proof(rule cat-Set-obj-coproduct-up-cat-Set-is-arr)
    show P' ∈o cat-Set α(|Obj|)
      by (auto intro: cat-cs-intros cat-lim-cs-intros)
    fix i assume i ∈o I
    then show π'(|NTMap|)(|i|) : F i ↦cat-Set α P'
      by
        (
          cs-concl cs-shallow
          cs-simp:
            cat-cs-simps cat-discrete-cs-simps
            the-cat-discrete-components(1)
          cs-intro: cat-cs-intros
        )
  qed (rule assms)

then have P': P' ∈o cat-Set α(|Obj|)
  by (auto intro: cat-cs-intros cat-lim-cs-intros)

have π'i-i: ?π'i i : F i ↦cat-Set α P' if i ∈o I for i
  using
  π'.ntcf-NTMap-is-arr[unfolded the-cat-discrete-components(1), OF that]

```

that
 by
 (

 cs-prems **cs-shallow cs-simp**:
 cat-cs-simps cat-discrete-cs-simps the-cat-discrete-components(1)

)

from *cat-Set-obj-coprod-up-cat-Set-is-arr*[*OF P' assms*(1) $\pi' i$] **have** $\pi' i$:
 $?up' : (\coprod_{i \in_0 I} F i) \mapsto_{cat-Set \alpha} P'$

show $\pi' = ntcf-const (:_C I) (cat-Set \alpha) ?up' \cdot_{NTCF} ?F$
proof(*rule ntcf-eqI, rule $\pi'.is-ntcf-axioms$*)

from *F-is-cat-cocone $\pi' i$* **show**
 $ntcf-const (:_C I) (cat-Set \alpha) ?up' \cdot_{NTCF} ?F :$
 $\rightarrow : I F (cat-Set \alpha) \mapsto_{CF} cf-const (:_C I) (cat-Set \alpha) P' :$
 $:_C I \mapsto_{C\alpha} cat-Set \alpha$

by (*cs-concl cs-shallow cs-intro: cat-cs-intros*)

have *dom-lhs*: $\mathcal{D}_o (\pi'(\downarrow NTMap)) = :_C I(\downarrow Obj)$

by (*cs-concl cs-shallow cs-simp: cat-cs-simps*)

from *F-is-cat-cocone $\pi' i$* **have** *dom-rhs*:
 $\mathcal{D}_o ((?F \cdot_{NTCF} ntcf-const (:_C I) (cat-Set \alpha) ?up')(\downarrow NTMap)) = :_C I(\downarrow Obj)$

by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

show $\pi'(\downarrow NTMap) = (ntcf-const (:_C I) (cat-Set \alpha) ?up' \cdot_{NTCF} ?F)(\downarrow NTMap)$

proof(*rule vsv-eqI, unfold dom-lhs dom-rhs*)

fix i **assume** *prems'*: $i \in_0 :_C I(\downarrow Obj)$

then have $i : i \in_0 I$ **unfolding** *the-cat-discrete-components* **by** *simp*

have [*cat-cs-simps*]:
 $?up' \circ_A cat-Set \alpha \text{ vcinjection-arrow } I F i = \pi'(\downarrow NTMap)(\downarrow i)$

by
 (

 simp add: cat-Set-cf-comp-coprod-up-vcia[
 OF P' assms $\pi' i$ i, symmetric
]

)

from $\pi' i$ *prems'* **show** $\pi'(\downarrow NTMap)(\downarrow i) =$
 $(ntcf-const (:_C I) (cat-Set \alpha) ?up' \cdot_{NTCF} ?F)(\downarrow NTMap)(\downarrow i)$

by
 (

 cs-concl cs-shallow
 cs-simp: *cat-cs-simps cat-Rel-cs-simps cs-intro: cat-cs-intros*

)

qed (*cs-concl cs-simp: cat-cs-simps cs-intro: V-cs-intros cat-cs-intros*)
qed *simp-all*

fix f' **assume** *prems*:
 $f' : (\coprod_{i \in_0 I} F i) \mapsto_{cat-Set \alpha} P'$
 $\pi' = ntcf-const (:_C I) (cat-Set \alpha) f' \cdot_{NTCF} ?F$

from *prems*(2) **have** $\pi'-eq-F-f'$: $\pi'(\downarrow NTMap)(\downarrow i)(\downarrow ArrVal)(\downarrow a) =$
 $(ntcf-const (:_C I) (cat-Set \alpha) f' \cdot_{NTCF} ?F)(\downarrow NTMap)(\downarrow i)(\downarrow ArrVal)(\downarrow a)$

if $i \in_0 I$ **and** $a \in_0 P'$ **for** $i a$

by *simp*

note $f' = cat-Set-is-arrD[OF prems(1)]$

note $up' = cat-Set-is-arrD[OF up]$

interpret f' : *arr-Set α f'* **by** (*rule f'(1)*)

interpret u' : *arr-Set α $\langle (cat-Set-obj-coprod-up I F P' (app (\pi'(\downarrow NTMap)))) \rangle$*
by (*rule up'(1)*)

show $f' = ?up'$

proof(*rule arr-Set-eqI[of α]*)

have *dom-lhs*: $\mathcal{D}_o (f'(\downarrow ArrVal)) = (\coprod_{i \in_0 I} F i)$

```

  by (simp add: cat-Set-cs-simps f')
have dom-rhs:
   $\mathcal{D}_\circ (cat\text{-Set-obj-coproduct-up } I F P' (app (\pi'(\mathcal{N}TMap))))(ArrVal) =$ 
   $(\coprod_{\circ} i \in_\circ I. F i)$ 
  by (simp add: cat-Set-cs-simps up')
show f'(ArrVal) = cat-Set-obj-coproduct-up I F P' (app (\pi'(\mathcal{N}TMap)))(ArrVal)
proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
  fix ix assume prems': ix  $\in_\circ (\coprod_{\circ} i \in_\circ I. F i)$ 
  then obtain i x where ix-def: ix = ⟨i, x⟩
    and i: i  $\in_\circ I$ 
    and x: x  $\in_\circ F i$ 
  by auto
  from assms prems(1) prems' i x show f'(ArrVal)(ix) =
    cat-Set-obj-coproduct-up I F P' (app (\pi'(\mathcal{N}TMap)))(ArrVal)(ix)
  unfolding ix-def prems(2)
  by
  (
    cs-concl cs-shallow
    cs-simp:
      cat-Set-cs-simps cat-cs-simps the-cat-discrete-components(1)
    cs-intro: cat-cs-intros
  )
qed auto
qed (simp-all add: cat-Set-obj-coproduct-up-components f' up'(1))

```

qed

qed

lemma (in \mathcal{Z}) *ntcf-Set-obj-coproduct-is-tm-cat-obj-coproduct*:

— See Theorem 5.2 in Chapter Introduction in [6].

assumes $VLambda I F \in_\circ Vset \alpha$

shows *ntcf-Set-obj-coproduct* $\alpha I F$:

$F >_{CF.tm.\coprod} (\coprod_{\circ} i \in_\circ I. F i) : I \mapsto_{C.tm\alpha} cat\text{-Set } \alpha$

proof(intro *is-tm-cat-obj-coproductI*)

from assms **show** *tm-cf-discrete* $\alpha I F (cat\text{-Set } \alpha)$

by (rule *tm-cf-discrete-cat-Set-if-VLambda-in-Vset*)

show *ntcf-Set-obj-coproduct* $\alpha I F$:

$\mapsto: I F (cat\text{-Set } \alpha) >_{CF.colim} VSigma I F : :_C I \mapsto_{C\alpha} cat\text{-Set } \alpha$

by

```

  (
    rule is-cat-obj-coproductD[
      OF tm-cf-discrete-ntcf-obj-coproduct-base-is-cat-obj-coproduct[OF assms]
    ]
  )

```

qed

12.3 Equalizer for the category *Set*

12.3.1 Definition and elementary properties

abbreviation *ntcf-Set-equalizer-map* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$

where *ntcf-Set-equalizer-map* $\alpha a g f i \equiv$

```

  (
    i =  $\mathfrak{a}_{PL2}$  ?
    incl-Set (vequalizer a g f) a :
    g  $\circ_A cat\text{-Set } \alpha$  incl-Set (vequalizer a g f) a
  )

```

definition *ntcf-Set-equalizer* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
where *ntcf-Set-equalizer* α a b g f = *ntcf-equalizer-base*
(cat-Set α) a b g f (vequalizer a g f) (ntcf-Set-equalizer-map α a g f)

Components.

context

fixes a g f α :: V

begin

lemmas *ntcf-Set-equalizer-components* =
ntcf-equalizer-base-components[
where $\mathfrak{C} = \langle \text{cat-Set } \alpha \rangle$
and $e = \langle \text{ntcf-Set-equalizer-map } \alpha \text{ a g f} \rangle$
and $E = \langle \text{vequalizer a g f} \rangle$
and $\mathfrak{a} = a$ **and** $\mathfrak{g} = g$ **and** $\mathfrak{f} = f$,
folded ntcf-Set-equalizer-def
]

end

12.3.2 Natural transformation map

mk-VLambda *ntcf-Set-equalizer-components*(1)
|vsv ntcf-Set-equalizer-NTMap-vsv[cat-Set-cs-intros]|
|vdomain ntcf-Set-equalizer-NTMap-vdomain[cat-Set-cs-simps]|
|app ntcf-Set-equalizer-NTMap-app|

lemma *ntcf-Set-equalizer-2-NTMap-app-a[cat-Set-cs-simps]*:
assumes $x = \mathfrak{a}_{PL2}$
shows
ntcf-Set-equalizer α a b g f (NTMap)(x) =
incl-Set (vequalizer a g f) a
unfolding *assms the-cat-parallel-2-components(1) ntcf-Set-equalizer-components*
by *simp*

lemma *ntcf-Set-equalizer-2-NTMap-app-b[cat-Set-cs-simps]*:
assumes $x = \mathfrak{b}_{PL2}$
shows
ntcf-Set-equalizer α a b g f (NTMap)(x) =
g $\circ_{A \text{ cat-Set } \alpha}$ incl-Set (vequalizer a g f) a
unfolding *assms the-cat-parallel-2-components(1) ntcf-Set-equalizer-components*
using *cat-PL2-ineq*
by *auto*

12.3.3 Equalizer for the category *Set* is an equalizer

lemma (in \mathcal{Z}) *ntcf-Set-equalizer-2-is-cat-equalizer-2*:
assumes $\mathfrak{g} : \mathfrak{a} \mapsto_{\text{cat-Set } \alpha} \mathfrak{b}$ **and** $\mathfrak{f} : \mathfrak{a} \mapsto_{\text{cat-Set } \alpha} \mathfrak{b}$
shows *ntcf-Set-equalizer α a b g f* :
vequalizer a g f <_{CF.eq} (a,b,g,f) : $\uparrow_C \mapsto \mapsto_{C \alpha}$ cat-Set α
proof(*intro is-cat-equalizer-2I is-cat-equalizerI is-cat-limitI*)

let $?II-II = \langle \uparrow \mapsto \uparrow_{CF} (\text{cat-Set } \alpha) \mathfrak{a}_{PL2} \mathfrak{b}_{PL2} \mathfrak{g}_{PL} \mathfrak{f}_{PL} \mathfrak{a} \mathfrak{b} \mathfrak{g} \mathfrak{f} \rangle$
and $?II = \langle \uparrow_C \mathfrak{a}_{PL2} \mathfrak{b}_{PL2} \mathfrak{g}_{PL} \mathfrak{f}_{PL} \rangle$

note $\mathfrak{g} = \text{cat-Set-is-arrD}[OF \text{ assms}(1)]$

interpret \mathfrak{g} : *arr-Set α g*

```

rewrites g(⟦ArrDom⟧) = a and g(⟦ArrCod⟧) = b
by (rule g(1)) (simp-all add: g)
note f = cat-Set-is-arrD[OF assms(2)]
interpret f: arr-Set α f
rewrites f(⟦ArrDom⟧) = a and f(⟦ArrCod⟧) = b
by (rule f(1)) (simp-all add: f)

note [cat-Set-cs-intros] = g.arr-Set-ArrDom-in-Vset f.arr-Set-ArrCod-in-Vset

let ?incl = ⟨incl-Set (vequalizer a g f) a⟩

show abgf-is-cat-cone: ntcf-Set-equalizer α a b g f :
  vequalizer a g f <CF.cone ?II-II : ?II ↦↦Cα cat-Set α
  unfolding ntcf-Set-equalizer-def
proof
  (
    intro
    category.cat-ntcf-equalizer-base-is-cat-cone
    category.cat-cf-parallel-2-cat-equalizer
  )
  from assms show
    (bPL2 = aPL2 ? ?incl : g ∘A cat-Set α ?incl) :
    vequalizer a g f ↦cat-Set α b
  by
    (
      cs-concl
      cs-simp: V-cs-simps
      cs-intro:
        V-cs-intros cat-Set-cs-intros cat-cs-intros
        cat-PL2-ineq[symmetric]
    )
  show
    (bPL2 = aPL2 ? ?incl : g ∘A cat-Set α ?incl) =
    g ∘A cat-Set α (aPL2 = aPL2 ? ?incl : g ∘A cat-Set α ?incl)
  by
    (
      cs-concl
      cs-simp: V-cs-simps
      cs-intro:
        V-cs-intros cat-Set-cs-intros cat-cs-intros
        cat-PL2-ineq[symmetric]
    )
  from assms show
    (bPL2 = aPL2 ? ?incl : g ∘A cat-Set α ?incl) =
    f ∘A cat-Set α (aPL2 = aPL2 ? ?incl : g ∘A cat-Set α ?incl)
  by
    (
      cs-concl
      cs-simp: V-cs-simps cat-Set-incl-Set-commute
      cs-intro: V-cs-intros cat-PL2-ineq[symmetric]
    )
  qed
  (
    cs-concl
    cs-intro: cat-cs-intros V-cs-intros cat-Set-cs-intros assms
    cs-simp: V-cs-simps cat-cs-simps cat-Set-components(1)
  )+

```

interpret abgf : *is-cat-cone*
 $\alpha \langle \text{vequalizer } \mathbf{a} \ \mathbf{g} \ \mathbf{f} \rangle \ ?II \langle \text{cat-Set } \alpha \rangle \ ?II-II \langle \text{ntcf-Set-equalizer } \alpha \ \mathbf{a} \ \mathbf{b} \ \mathbf{g} \ \mathbf{f} \rangle$
by (*rule abgf-is-cat-cone*)

show $\exists ! f'$.

$f' : r' \mapsto_{\text{cat-Set } \alpha} \text{vequalizer } \mathbf{a} \ \mathbf{g} \ \mathbf{f} \wedge$
 $u' = \text{ntcf-Set-equalizer } \alpha \ \mathbf{a} \ \mathbf{b} \ \mathbf{g} \ \mathbf{f} \cdot_{NTCF} \text{ntcf-const } ?II \ (\text{cat-Set } \alpha) \ f'$
if $u' : r' \langle_{CF.cone} ?II-II : ?II \mapsto_{C\alpha} \text{cat-Set } \alpha \ \text{for } u' \ r'$

proof-

interpret u' : *is-cat-cone* $\alpha \ r' \ ?II \langle \text{cat-Set } \alpha \rangle \ ?II-II \ u'$ **by** (*rule that(1)*)

have $\mathbf{a}_{PL2} \in_o \ ?II(\text{Obj})$

unfolding *the-cat-parallel-2-components(1)* **by** *simp*

from

$u'.\text{ntcf-NTMap-is-arr}[OF \ \text{this}]$

$\text{abgf}.\text{NTDom.HomCod.cat-cf-parallel-2-cat-equalizer}[OF \ \text{assms}]$

have $u' \text{-}\mathbf{a}_{PL} \text{-is-arr}$: $u'(\text{NTMap})(\mathbf{a}_{PL2}) : r' \mapsto_{\text{cat-Set } \alpha} \mathbf{a}$

by (*cs-prems-atom-step cat-cs-simps*)

(

cs-prems

cs-simp: *cat-parallel-cs-simps*

cs-intro:

cat-parallel-cs-intros

cat-cs-intros

category.cat-cf-parallel-2-cat-equalizer

)

note $u' \text{-}\mathbf{a}_{PL} = \text{cat-Set-is-arrD}[OF \ u' \text{-}\mathbf{a}_{PL} \text{-is-arr}]$

interpret $u' \text{-}\mathbf{a}_{PL}$: *arr-Set* $\alpha \ \langle u'(\text{NTMap})(\mathbf{a}_{PL2}) \rangle$ **by** (*rule u'-a_{PL}(1)*)

have $\mathbf{b}_{PL2} \in_o \ ?II(\text{Obj})$

by (*cs-concl cs-shallow cs-intro: cat-parallel-cs-intros*)

from

$u'.\text{ntcf-NTMap-is-arr}[OF \ \text{this}]$

$\text{abgf}.\text{NTDom.HomCod.cat-cf-parallel-2-cat-equalizer}[OF \ \text{assms}]$

have $u'(\text{NTMap})(\mathbf{b}_{PL2}) : r' \mapsto_{\text{cat-Set } \alpha} \mathbf{b}$

by

(

cs-prems cs-shallow

cs-simp: *cat-cs-simps cat-parallel-cs-simps*

cs-intro: *cat-parallel-cs-intros*

)

note $u' \text{-}\mathbf{g} \ u' = \text{cat-cone-cf-par-2-eps-NTMap-app}(1)[OF \ \text{that}(1) \ \text{assms}]$

define q **where** $q = [u'(\text{NTMap})(\mathbf{a}_{PL2})(\text{ArrVal}), r', \text{vequalizer } \mathbf{a} \ \mathbf{g} \ \mathbf{f}]_o$

have q -*components*[*cat-Set-cs-simps*]:

$q(\text{ArrVal}) = u'(\text{NTMap})(\mathbf{a}_{PL2})(\text{ArrVal})$

$q(\text{ArrDom}) = r'$

$q(\text{ArrCod}) = \text{vequalizer } \mathbf{a} \ \mathbf{g} \ \mathbf{f}$

unfolding q -*def arr-field-simps* **by** (*simp-all add: nat-omega-simps*)

from *cat-cone-cf-par-2-eps-NTMap-app*[*OF that(1) assms*] **have** $\mathbf{g} \ u' \text{-eq-f} \ u'$:

$(\mathbf{g} \circ_{A \ \text{cat-Set } \alpha} u'(\text{NTMap})(\mathbf{a}_{PL2}))(\text{ArrVal})(x) =$

$(\mathbf{f} \circ_{A \ \text{cat-Set } \alpha} u'(\text{NTMap})(\mathbf{a}_{PL2}))(\text{ArrVal})(x)$

for x

by simp

show ?thesis

proof(intro ex1I conjI; (elim conjE)?)

have u' -NTMap-vrange: $\mathcal{R}_\circ (u'(\text{NTMap})(\mathbf{a}_{PL2})(\text{ArrVal})) \subseteq_\circ \text{vequalizer } \mathbf{a} \mathbf{g} \mathbf{f}$

proof(rule vsubsetI)

fix y assume prems: $y \in_\circ \mathcal{R}_\circ (u'(\text{NTMap})(\mathbf{a}_{PL2})(\text{ArrVal}))$

then obtain x where $x: x \in_\circ \mathcal{D}_\circ (u'(\text{NTMap})(\mathbf{a}_{PL2})(\text{ArrVal}))$

and y -def: $y = u'(\text{NTMap})(\mathbf{a}_{PL2})(\text{ArrVal})(x)$

by (blast dest: u' - \mathbf{a}_{PL} .ArrVal.vrange-atD)

have $x: x \in_\circ r'$

by (use x u' - \mathbf{a}_{PL} -is-arr in $\langle \text{cs-prems cs-shallow cs-simp: cat-cs-simps} \rangle$)

from $\mathbf{g}u'$ -eq-fu'[of x] assms x u' - \mathbf{a}_{PL} -is-arr have [simp]:

$\mathbf{g}(\text{ArrVal})(u'(\text{NTMap})(\mathbf{a}_{PL2})(\text{ArrVal})(x)) =$

$\mathbf{f}(\text{ArrVal})(u'(\text{NTMap})(\mathbf{a}_{PL2})(\text{ArrVal})(x))$

by (cs-prems **cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros**)

from prems u' - \mathbf{a}_{PL} .arr-Set-ArrVal-vrange[unfolded u' - \mathbf{a}_{PL}] show

$y \in_\circ \text{vequalizer } \mathbf{a} \mathbf{g} \mathbf{f}$

by (intro vequalizerI, unfold y -def) auto

qed

show q -is-arr: $q: r' \mapsto_{\text{cat-Set } \alpha} \text{vequalizer } \mathbf{a} \mathbf{g} \mathbf{f}$

proof(intro cat-Set-is-arrI arr-SetI)

show $q(\text{ArrCod}) \in_\circ \text{Vset } \alpha$

by (auto simp: q -components intro: cat-cs-intros cat-lim-cs-intros)

qed

(

auto

simp:

cat-Set-cs-simps nat-omega-simps

u' - \mathbf{a}_{PL}

q -def

u' -NTMap-vrange

\mathbf{abgf} .NTDom.HomCod.cat-in-Obj-in-Vset

intro: cat-cs-intros cat-lim-cs-intros

)

from q -is-arr have \mathbf{a} - q :

$\text{incl-Set } (\text{vequalizer } \mathbf{a} \mathbf{g} \mathbf{f}) \mathbf{a} \circ_A \text{cat-Set } \alpha \ q: r' \mapsto_{\text{cat-Set } \alpha} \mathbf{a}$

by

(

cs-concl

cs-simp: cat-cs-simps cat-Set-components(1)

cs-intro: V-cs-intros cat-cs-intros cat-Set-cs-intros

)

interpret arr-Set $\alpha \langle \text{incl-Set } (\text{vequalizer } \mathbf{a} \mathbf{g} \mathbf{f}) \mathbf{a} \circ_A \text{cat-Set } \alpha \ q \rangle$

using \mathbf{a} - q by (auto dest: cat-Set-is-arrD)

show $u' = \text{ntcf-Set-equalizer } \alpha \ \mathbf{a} \ \mathbf{b} \ \mathbf{g} \ \mathbf{f} \cdot_{NTCF} \text{ntcf-const } ?II \ (\text{cat-Set } \alpha) \ q$

proof(rule ntcf-eqI)

from q -is-arr show

$\text{ntcf-Set-equalizer } \alpha \ \mathbf{a} \ \mathbf{b} \ \mathbf{g} \ \mathbf{f} \cdot_{NTCF} \text{ntcf-const } ?II \ (\text{cat-Set } \alpha) \ q:$

$\text{cf-const } ?II \ (\text{cat-Set } \alpha) \ r' \mapsto_{CF}$

$?II-II: ?II \mapsto_{C\alpha} \text{cat-Set } \alpha$

by (cs-concl **cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros**)

have dom-lhs: $\mathcal{D}_\circ (u'(\text{NTMap})) = ?II(\text{Obj})$

by (cs-concl **cs-shallow cs-simp: cat-cs-simps**)

```

from  $q$ -is-arr have dom-rhs:
   $\mathcal{D}_\circ$ 
  (
    (ntcf-Set-equalizer  $\alpha$   $\mathbf{a}$   $\mathbf{b}$   $\mathbf{g}$   $\mathbf{f}$   $\cdot_{NTCF}$ 
      ntcf-const ?II (cat-Set  $\alpha$ )  $q$ 
    )(NTMap)) = ?II(Obj)
  by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
show  $u'$ (NTMap) =
  (
    ntcf-Set-equalizer  $\alpha$   $\mathbf{a}$   $\mathbf{b}$   $\mathbf{g}$   $\mathbf{f}$   $\cdot_{NTCF}$  ntcf-const ?II (cat-Set  $\alpha$ )  $q$ 
  )(NTMap)
proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
show vsv ((
  ntcf-Set-equalizer  $\alpha$   $\mathbf{a}$   $\mathbf{b}$   $\mathbf{g}$   $\mathbf{f}$   $\cdot_{NTCF}$  ntcf-const ?II (cat-Set  $\alpha$ )  $q$ 
  )(NTMap))
by (cs-concl cs-intro: cat-cs-intros)
fix  $a$  assume prems:  $a \in_\circ$  ?II(Obj)
have [symmetric, cat-Set-cs-simps]:
   $u'$ (NTMap)( $\mathbf{a}_{PL2}$ ) = incl-Set (vequalizer  $\mathbf{a}$   $\mathbf{g}$   $\mathbf{f}$ )  $\mathbf{a} \circ_{Acat-Set \alpha} q$ 
proof(rule arr-Set-eqI[of  $\alpha$ ])
from  $u'$ - $\mathbf{a}_{PL}$ -is-arr have dom-lhs:  $\mathcal{D}_\circ$  ( $u'$ (NTMap)( $\mathbf{a}_{PL2}$ )(ArrVal)) =  $r'$ 
by
  (
    cs-concl cs-shallow
    cs-simp: cat-cs-simps cs-intro: cat-cs-intros
  )
from  $\mathbf{a}$ - $q$  have dom-rhs:
   $\mathcal{D}_\circ$  ((incl-Set (vequalizer  $\mathbf{a}$   $\mathbf{g}$   $\mathbf{f}$ )  $\mathbf{a} \circ_{Acat-Set \alpha} q$ )(ArrVal)) =  $r'$ 
by
  (
    cs-concl cs-shallow
    cs-simp: cat-cs-simps cs-intro: cat-cs-intros
  )
show  $u'$ (NTMap)( $\mathbf{a}_{PL2}$ )(ArrVal) =
  (incl-Set (vequalizer  $\mathbf{a}$   $\mathbf{g}$   $\mathbf{f}$ )  $\mathbf{a} \circ_{Acat-Set \alpha} q$ )(ArrVal)
proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
fix  $a$  assume prems:  $a \in_\circ$   $r'$ 
with  $u'$ -NTMap-vrange dom-lhs  $u'$ - $\mathbf{a}_{PL}$ .ArrVal.vsv-vimageI2 have
   $u'$ (NTMap)( $\mathbf{a}_{PL2}$ )(ArrVal)( $a$ )  $\in_\circ$  vequalizer  $\mathbf{a}$   $\mathbf{g}$   $\mathbf{f}$ 
by blast
with prems  $q$ -is-arr  $u'$ - $\mathbf{a}_{PL}$ -is-arr show
   $u'$ (NTMap)( $\mathbf{a}_{PL2}$ )(ArrVal)( $a$ ) =
  (incl-Set (vequalizer  $\mathbf{a}$   $\mathbf{g}$   $\mathbf{f}$ )  $\mathbf{a} \circ_{Acat-Set \alpha} q$ )(ArrVal)( $a$ )
by
  (
    cs-concl cs-shallow
    cs-simp: cat-Set-cs-simps cat-cs-simps
    cs-intro: V-cs-intros cat-cs-intros cat-Set-cs-intros
  )
qed auto
qed
  (
    use  $u'$ - $\mathbf{a}_{PL}$   $\mathbf{a}$ - $q$  in  $\langle$ 
    cs-concl cs-shallow
    cs-intro: cat-Set-is-arrD(1) cs-simp: cat-cs-simps
     $\rangle$ 
  )+
from  $q$ -is-arr have  $u'$ -NTMap-app-I:  $u'$ (NTMap)( $\mathbf{a}_{PL2}$ ) =

```

```

(
  ntcf-Set-equalizer  $\alpha$   $\mathbf{a}$   $\mathbf{b}$   $\mathbf{g}$   $\mathbf{f} \cdot_{NTCF}$  ntcf-const ?II (cat-Set  $\alpha$ )  $q$ 
)(NTMap)( $\mathbf{a}_{PL2}$ )
by
  (
    cs-concl
    cs-intro: cat-cs-intros cat-parallel-cs-intros
    cs-simp: cat-Set-cs-simps cat-cs-simps V-cs-simps
  )
from q-is-arr assms have  $u'$ -NTMap-app-sI:  $u'$ (NTMap)( $\mathbf{b}_{PL2}$ ) =
  (
    ntcf-Set-equalizer  $\alpha$   $\mathbf{a}$   $\mathbf{b}$   $\mathbf{g}$   $\mathbf{f} \cdot_{NTCF}$  ntcf-const ?II (cat-Set  $\alpha$ )  $q$ 
)(NTMap)( $\mathbf{b}_{PL2}$ )
by
  (
    cs-concl
    cs-simp: cat-Set-cs-simps cat-cs-simps  $u'$ - $\mathbf{g}u'$ 
    cs-intro:
      V-cs-intros
      cat-cs-intros
      cat-Set-cs-intros
      cat-parallel-cs-intros
  )
from prems consider  $\langle a = \mathbf{a}_{PL2} \rangle \mid \langle a = \mathbf{b}_{PL2} \rangle$ 
  by (elim the-cat-parallel-2-ObjE)
then show
   $u'$ (NTMap)( $a$ ) =
  (
    ntcf-Set-equalizer  $\alpha$   $\mathbf{a}$   $\mathbf{b}$   $\mathbf{g}$   $\mathbf{f} \cdot_{NTCF}$ 
    ntcf-const ?II (cat-Set  $\alpha$ )  $q$ 
  )(NTMap)( $a$ )
  by cases (simp-all add:  $u'$ -NTMap-app-I  $u'$ -NTMap-app-sI)
qed auto
qed (simp-all add:  $u'$ .is-ntcf-axioms)

fix  $f'$  assume prems:
   $f' : r' \mapsto_{\text{cat-Set } \alpha}$  vequalizer  $\mathbf{a}$   $\mathbf{g}$   $\mathbf{f}$ 
   $u' = \text{ntcf-Set-equalizer } \alpha \mathbf{a} \mathbf{b} \mathbf{g} \mathbf{f} \cdot_{NTCF} \text{ntcf-const ?II (cat-Set } \alpha) f'$ 
from prems(2) have  $u'$ -NTMap-app:
   $u'$ (NTMap)( $x$ ) =
  (ntcf-Set-equalizer  $\alpha$   $\mathbf{a}$   $\mathbf{b}$   $\mathbf{g}$   $\mathbf{f} \cdot_{NTCF}$ 
  ntcf-const ?II (cat-Set  $\alpha$ )  $f'$ )(NTMap)( $x$ )
for  $x$ 
  by simp
have  $u'$ - $f'$ :
   $u'$ (NTMap)( $\mathbf{a}_{PL2}$ ) = incl-Set (vequalizer  $\mathbf{a}$   $\mathbf{g}$   $\mathbf{f}$ )  $\mathbf{a} \circ_{A \text{ cat-Set } \alpha} f'$ 
  using  $u'$ -NTMap-app[of  $\mathbf{a}_{PL2}$ ] prems(1)
by
  (
    cs-prems
    cs-simp: cat-cs-simps
    cs-intro: cat-cs-intros cat-parallel-cs-intros
  )
  (
    cs-prems cs-shallow
    cs-simp: cat-Set-cs-simps cs-intro: cat-parallel-cs-intros
  )

```

note $f' = \text{cat-Set-is-arrD}[OF \text{ prems}(1)]$

note $q = \text{cat-Set-is-arrD}[OF \text{ q-is-arr}]$

interpret $f': \text{arr-Set } \alpha \text{ } f'$ **using** $\text{prems}(1)$ **by** $(\text{auto dest: cat-Set-is-arrD})$

interpret $q: \text{arr-Set } \alpha \text{ } q$ **using** q **by** $(\text{auto dest: cat-Set-is-arrD})$

show $f' = q$

proof(*rule arr-Set-eqI[of α]*)

have $\text{dom-lhs: } \mathcal{D}_\circ (f'(\downarrow \text{ArrVal})) = r'$ **by** $(\text{simp add: cat-Set-cs-simps } f')$

from $q\text{-is-arr}$ **have** $\text{dom-rhs: } \mathcal{D}_\circ (q(\downarrow \text{ArrVal})) = r'$

by

(

cs-concl cs-shallow

cs-simp: *cat-cs-simps* **cs-intro:** *cat-Set-cs-intros*

)

show $f'(\downarrow \text{ArrVal}) = q(\downarrow \text{ArrVal})$

proof(*rule vsv-eqI, unfold dom-lhs dom-rhs*)

fix i **assume** $i \in_\circ r'$

with $\text{prems}(1)$ **show** $f'(\downarrow \text{ArrVal})(\downarrow i) = q(\downarrow \text{ArrVal})(\downarrow i)$

by

(

cs-concl

cs-simp:

cat-Set-cs-simps cat-cs-simps

q-components u'-f' cat-Set-components(1)

cs-intro: *V-cs-intros cat-cs-intros cat-Set-cs-intros*

)

qed *auto*

qed

(

use prems(1) q-is-arr in <

cs-concl cs-shallow

cs-simp: cat-cs-simps cs-intro: q cat-Set-is-arrD

)

)+

qed

qed

qed (*auto intro: assms*)

12.4 The category *Set* is small-complete

lemma (*in \mathcal{Z}*) *cat-small-complete-cat-Set: cat-small-complete α (cat-Set α)*

— This lemma appears as a remark on page 113 in [9].

proof(*rule category.cat-small-complete-if-eq-and-obj-prod*)

show $\exists E \varepsilon. \varepsilon : E <_{CF.eq} (\mathbf{a}, \mathbf{b}, \mathbf{g}, \mathbf{f}) : \uparrow \uparrow_C \mapsto_{C\alpha} \text{cat-Set } \alpha$

if $\mathbf{f} : \mathbf{a} \mapsto_{\text{cat-Set } \alpha} \mathbf{b}$ **and** $\mathbf{g} : \mathbf{a} \mapsto_{\text{cat-Set } \alpha} \mathbf{b}$ **for** $\mathbf{a} \ \mathbf{b} \ \mathbf{g} \ \mathbf{f}$

using *ntcf-Set-equalizer-2-is-cat-equalizer-2[OF that(2,1)]* **by** *auto*

show $\exists P \pi. \pi : P <_{CF.\Pi} A : I \mapsto_{C\alpha} \text{cat-Set } \alpha$

if *tm-cf-discrete α I A* (*cat-Set α*) **for** $A \ I$

proof(*intro exI, rule tm-cf-discrete-ntcf-obj-prod-base-is-cat-obj-prod*)

interpret *tm-cf-discrete α I A <cat-Set α >* **by** (*rule that*)

show $V\text{Lambda } I \ A \ \varepsilon_\circ \ V\text{set } \alpha$ **by** (*rule tm-cf-discrete-ObjMap-in-Vset*)

qed

qed (*rule category-cat-Set*)

13 Adjoints

13.1 Background

named-theorems *adj-cs-simps*

named-theorems *adj-cs-intros*

named-theorems *adj-field-simps*

definition *AdjLeft* :: *V* **where** [*adj-field-simps*]: *AdjLeft* = 0

definition *AdjRight* :: *V* **where** [*adj-field-simps*]: *AdjRight* = 1_N

definition *AdjNT* :: *V* **where** [*adj-field-simps*]: *AdjNT* = 2_N

13.2 Definition and elementary properties

See subsection 2.1 in [4] or Chapter IV-1 in [9].

locale *is-cf-adjunction* =

Z *α* +

vfsequence *Φ* +

L: *category* *α* *ℂ* +

R: *category* *α* *℔* +

LR: *is-functor* *α* *ℂ* *℔* *ℱ* +

RL: *is-functor* *α* *℔* *ℂ* *ℬ* +

NT: *is-iso-ntcf*

α

⟨*op-cat* *ℂ* ×_{*C*} *℔*⟩

⟨*cat-Set* *α*⟩

⟨*Hom*_{*O.Cα*} *℔*(*ℱ*-, -)⟩

⟨*Hom*_{*O.Cα*} *ℂ*(-, *ℬ*-)⟩

⟨*Φ*(*AdjNT*)⟩

for *α* *ℂ* *℔* *ℱ* *ℬ* *Φ* +

assumes *cf-adj-length*[*adj-cs-simps*]: *vcard* *Φ* = 3_N

and *cf-adj-AdjLeft*[*adj-cs-simps*]: *Φ*(*AdjLeft*) = *ℱ*

and *cf-adj-AdjRight*[*adj-cs-simps*]: *Φ*(*AdjRight*) = *ℬ*

syntax *-is-cf-adjunction* :: *V* ⇒ *V* ⇒ *V* ⇒ *V* ⇒ *V* ⇒ *V* ⇒ *bool*

(⟨(- : - ⇒_{*CF*} - : - ⇒_{*C1*} -)⟩ [51, 51, 51, 51, 51] 51)

syntax-consts *-is-cf-adjunction* ⇒ *is-cf-adjunction*

translations *Φ* : *ℱ* ⇒_{*CF*} *ℬ* : *ℂ* ⇒_{*Cα*} *℔* ⇒

CONST is-cf-adjunction *α* *ℂ* *℔* *ℱ* *ℬ* *Φ*

lemmas [*adj-cs-simps*] =

is-cf-adjunction.cf-adj-length

is-cf-adjunction.cf-adj-AdjLeft

is-cf-adjunction.cf-adj-AdjRight

Components.

lemma *cf-adjunction-components*[*adj-cs-simps*]:

[*ℱ*, *ℬ*, *φ*].(*AdjLeft*) = *ℱ*

[*ℱ*, *ℬ*, *φ*].(*AdjRight*) = *ℬ*

[*ℱ*, *ℬ*, *φ*].(*AdjNT*) = *φ*

unfolding *AdjLeft-def AdjRight-def AdjNT-def*

by (*simp-all add: nat-omega-simps*)

Rules.

lemma (**in** *is-cf-adjunction*) *is-cf-adjunction-axioms'*[*adj-cs-intros*]:

assumes *α'* = *α* **and** *ℂ'* = *ℂ* **and** *℔'* = *℔* **and** *ℱ'* = *ℱ* **and** *ℬ'* = *ℬ*

shows *Φ* : *ℱ'* ⇒_{*CF*} *ℬ'* : *ℂ'* ⇒_{*Cα'*} *℔'*

unfolding *assms* by (rule *is-cf-adjunction-axioms*)

lemmas (in *is-cf-adjunction*) [*adj-cs-intros*] = *is-cf-adjunction-axioms*

mk-ide rf *is-cf-adjunction-def*[*unfolded is-cf-adjunction-axioms-def*]
 |*intro is-cf-adjunctionI*||
 |*dest is-cf-adjunctionD*[*dest*]|
 |*elim is-cf-adjunctionE*[*elim*]|

lemmas [*adj-cs-intros*] = *is-cf-adjunctionD*(3-6)

lemma (in *is-cf-adjunction*) *cf-adj-is-iso-ntcf'*:
assumes $\mathfrak{F}' = \text{Hom}_{O.C\alpha}\mathfrak{D}(\mathfrak{F}, -)$
and $\mathfrak{G}' = \text{Hom}_{O.C\alpha}\mathfrak{C}(-, \mathfrak{G})$
and $\mathfrak{A}' = \text{op-cat } \mathfrak{C} \times_C \mathfrak{D}$
and $\mathfrak{B}' = \text{cat-Set } \alpha$
shows $\Phi(\text{AdjNT}) : \mathfrak{F}' \mapsto_{CF.iso} \mathfrak{G}' : \mathfrak{A}' \mapsto_{C\alpha} \mathfrak{B}'$
unfolding *assms* by (auto *intro: cat-cs-intros*)

lemmas [*adj-cs-intros*] = *is-cf-adjunction.cf-adj-is-iso-ntcf'*

lemma *cf-adj-eqI*:
assumes $\Phi : \mathfrak{F} \rightleftharpoons_{CF} \mathfrak{G} : \mathfrak{C} \rightleftharpoons_{C\alpha} \mathfrak{D}$
and $\Phi' : \mathfrak{F}' \rightleftharpoons_{CF} \mathfrak{G}' : \mathfrak{C}' \rightleftharpoons_{C\alpha} \mathfrak{D}'$
and $\mathfrak{C} = \mathfrak{C}'$
and $\mathfrak{D} = \mathfrak{D}'$
and $\mathfrak{F} = \mathfrak{F}'$
and $\mathfrak{G} = \mathfrak{G}'$
and $\Phi(\text{AdjNT}) = \Phi'(\text{AdjNT})$
shows $\Phi = \Phi'$

proof-

interpret Φ : *is-cf-adjunction* α \mathfrak{C} \mathfrak{D} \mathfrak{F} \mathfrak{G} Φ by (rule *assms(1)*)
interpret Φ' : *is-cf-adjunction* α \mathfrak{C}' \mathfrak{D}' \mathfrak{F}' \mathfrak{G}' Φ' by (rule *assms(2)*)
show *?thesis*

proof(rule *vsv-eqI*)

have *dom*: $\mathcal{D}_\circ \Phi = \mathcal{Z}_N$

by (*cs-concl cs-shallow cs-simp: V-cs-simps adj-cs-simps*)

show $\mathcal{D}_\circ \Phi = \mathcal{D}_\circ \Phi'$

by (*cs-concl cs-shallow cs-simp: V-cs-simps adj-cs-simps dom*)

from *assms(4-7)* **have** *sup*:

$\Phi(\text{AdjLeft}) = \Phi'(\text{AdjLeft})$

$\Phi(\text{AdjRight}) = \Phi'(\text{AdjRight})$

$\Phi(\text{AdjNT}) = \Phi'(\text{AdjNT})$

by (*simp-all add: adj-cs-simps*)

show $a \in_\circ \mathcal{D}_\circ \Phi \implies \Phi(|a|) = \Phi'(|a|)$ **for** a

by (*unfold dom, elim-in-numeral, insert sup*)

(*auto simp: adj-field-simps*)

qed (*auto simp: \Phi.L.vsv-axioms \Phi'.vsv-axioms*)

qed

13.3 Opposite adjunction

13.3.1 Definition and elementary properties

See [7] for further information.

abbreviation *op-cf-adj-nt* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$

where *op-cf-adj-nt* \mathfrak{C} \mathfrak{D} $\varphi \equiv \text{inv-ntcf}$ (*bnt-flip* (*op-cat* \mathfrak{C}) \mathfrak{D} φ)

definition *op-cf-adj* :: $V \Rightarrow V$

where *op-cf-adj* $\Phi =$

[
 $op-cf (\Phi(\!|AdjRight|))$,
 $op-cf (\Phi(\!|AdjLeft|))$,
 $op-cf-adj-nt (\Phi(\!|AdjLeft|)(\!|HomDom|)) (\Phi(\!|AdjLeft|)(\!|HomCod|)) (\Phi(\!|AdjNT|))$
]_o

lemma *op-cf-adj-components*:

shows *op-cf-adj* $\Phi(\!|AdjLeft|) = op-cf (\Phi(\!|AdjRight|))$

and *op-cf-adj* $\Phi(\!|AdjRight|) = op-cf (\Phi(\!|AdjLeft|))$

and *op-cf-adj* $\Phi(\!|AdjNT|) =$

$op-cf-adj-nt (\Phi(\!|AdjLeft|)(\!|HomDom|)) (\Phi(\!|AdjLeft|)(\!|HomCod|)) (\Phi(\!|AdjNT|))$

unfolding *op-cf-adj-def adj-field-simps* **by** (*simp-all add: nat-omega-simps*)

lemma (**in** *is-cf-adjunction*) *op-cf-adj-components*:

shows *op-cf-adj* $\Phi(\!|AdjLeft|) = op-cf \mathfrak{G}$

and *op-cf-adj* $\Phi(\!|AdjRight|) = op-cf \mathfrak{F}$

and *op-cf-adj* $\Phi(\!|AdjNT|) = inv-ntcf (bnt-flip (op-cat \mathfrak{C}) \mathfrak{D} (\Phi(\!|AdjNT|)))$

unfolding *op-cf-adj-components* **by** (*simp-all add: cat-cs-simps adj-cs-simps*)

lemmas [*cat-op-simps*] = *is-cf-adjunction.op-cf-adj-components*

The opposite adjunction is an adjunction.

lemma (**in** *is-cf-adjunction*) *is-cf-adjunction-op*:

— See comments in subsection 2.1 in [4].

op-cf-adj $\Phi : op-cf \mathfrak{G} \Rightarrow_{CF} op-cf \mathfrak{F} : op-cat \mathfrak{D} \Rightarrow_{C\alpha} op-cat \mathfrak{C}$

proof(*intro is-cf-adjunctionI, unfold cat-op-simps, unfold op-cf-adj-components*)

show *vfsequence* (*op-cf-adj* Φ) **unfolding** *op-cf-adj-def* **by** *simp*

show *vcard* (*op-cf-adj* Φ) = $3_{\mathbb{N}}$

unfolding *op-cf-adj-def* **by** (*simp add: nat-omega-simps*)

note *adj* = *is-cf-adjunctionD*[*OF is-cf-adjunction-axioms*]

from *adj* **have** *f-φ*: *bnt-flip* (*op-cat* \mathfrak{C}) $\mathfrak{D} (\Phi(\!|AdjNT|)) :$

$Hom_{O.C\alpha} op-cat \mathfrak{D}(-, op-cf \mathfrak{F}-) \mapsto_{CF.iso} Hom_{O.C\alpha} op-cat \mathfrak{C}(op-cf \mathfrak{G}-, -) :$

$\mathfrak{D} \times_C op-cat \mathfrak{C} \mapsto_{C\alpha} cat-Set \alpha$

by

(
cs-concl cs-shallow
cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-op-intros
)

show *op-cf-adj-nt* $\mathfrak{C} \mathfrak{D} (\Phi(\!|AdjNT|)) :$

$Hom_{O.C\alpha} op-cat \mathfrak{C}(op-cf \mathfrak{G}-, -) \mapsto_{CF.iso} Hom_{O.C\alpha} op-cat \mathfrak{D}(-, op-cf \mathfrak{F}-) :$

$\mathfrak{D} \times_C op-cat \mathfrak{C} \mapsto_{C\alpha} cat-Set \alpha$

by (*rule CZH-ECAT-NTCF.iso-ntcf-is-iso-arr(1)*[*OF f-φ*])

qed (*auto intro: cat-cs-intros cat-op-intros*)

lemmas *is-cf-adjunction-op* =

is-cf-adjunction.is-cf-adjunction-op

lemma (**in** *is-cf-adjunction*) *is-cf-adjunction-op'*[*cat-op-intros*]:

assumes $\mathfrak{G}' = op-cf \mathfrak{G}$

and $\mathfrak{F}' = op-cf \mathfrak{F}$

and $\mathfrak{D}' = op-cat \mathfrak{D}$

and $\mathfrak{C}' = op-cat \mathfrak{C}$

shows *op-cf-adj* $\Phi : \mathfrak{G}' \Rightarrow_{CF} \mathfrak{F}' : \mathfrak{D}' \Rightarrow_{C\alpha} \mathfrak{C}'$

unfolding *assms* **by** (*rule is-cf-adjunction-op*)

lemmas [*cat-op-intros*] = *is-cf-adjunction.is-cf-adjunction-op'*

The operation of taking the opposite adjunction is an involution.

```

lemma (in is-cf-adjunction) cf-adjunction-op-cf-adj-op-cf-adj[cat-op-simps]:
  op-cf-adj (op-cf-adj  $\Phi$ ) =  $\Phi$ 
proof(rule cf-adj-eqI)
  show  $\Phi'$ : op-cf-adj (op-cf-adj  $\Phi$ ) :  $\mathfrak{F} \rightleftharpoons_{CF} \mathfrak{G} : \mathfrak{C} \rightleftharpoons_{C\alpha} \mathfrak{D}$ 
  proof(intro is-cf-adjunctionI)
  show vfsequence (op-cf-adj (op-cf-adj  $\Phi$ )) unfolding op-cf-adj-def by simp
  from is-cf-adjunction-axioms show op-cf-adj (op-cf-adj  $\Phi$ )(AdjNT) :
    HomO.C $\alpha$  $\mathfrak{D}$ ( $\mathfrak{F}$ -, -)  $\mapsto_{CF.is\ o}$  HomO.C $\alpha$  $\mathfrak{C}$ (-,  $\mathfrak{G}$ -) :
    op-cat  $\mathfrak{C} \times_C \mathfrak{D} \mapsto\mapsto_{C\alpha}$  cat-Set  $\alpha$ 
  by
  (
    cs-concl cs-shallow
    cs-intro: cat-cs-intros cat-op-intros adj-cs-intros
    cs-simp: cat-cs-simps cat-op-simps
  )
  show vcard (op-cf-adj (op-cf-adj  $\Phi$ )) =  $\exists_{\mathbb{N}}$ 
  unfolding op-cf-adj-def by (simp add: nat-omega-simps)
  from is-cf-adjunction-axioms show op-cf-adj (op-cf-adj  $\Phi$ )(AdjLeft) =  $\mathfrak{F}$ 
  by (cs-concl cs-shallow cs-simp: cat-op-simps cs-intro: cat-op-intros)
  from is-cf-adjunction-axioms show op-cf-adj (op-cf-adj  $\Phi$ )(AdjRight) =  $\mathfrak{G}$ 
  by (cs-concl cs-shallow cs-simp: cat-op-simps cs-intro: cat-op-intros)
qed (auto intro: cat-cs-intros)
interpret  $\Phi'$ : is-cf-adjunction  $\alpha$   $\mathfrak{C}$   $\mathfrak{D}$   $\mathfrak{F}$   $\mathfrak{G}$   $\langle$ op-cf-adj (op-cf-adj  $\Phi$ ) $\rangle$ 
  by (rule  $\Phi'$ )
show op-cf-adj (op-cf-adj  $\Phi$ )(AdjNT) =  $\Phi$ (AdjNT)
proof(rule ntcf-eqI)
  show op-op- $\Phi$ :
    op-cf-adj (op-cf-adj  $\Phi$ )(AdjNT) :
      HomO.C $\alpha$  $\mathfrak{D}$ ( $\mathfrak{F}$ -, -)  $\mapsto_{CF}$  HomO.C $\alpha$  $\mathfrak{C}$ (-,  $\mathfrak{G}$ -) :
      op-cat  $\mathfrak{C} \times_C \mathfrak{D} \mapsto\mapsto_{C\alpha}$  cat-Set  $\alpha$ 
    by (rule  $\Phi'.NT.is-ntcf-axioms$ )
  show  $\Phi$ :  $\Phi$ (AdjNT) :
    HomO.C $\alpha$  $\mathfrak{D}$ ( $\mathfrak{F}$ -, -)  $\mapsto_{CF}$  HomO.C $\alpha$  $\mathfrak{C}$ (-,  $\mathfrak{G}$ -) :
    op-cat  $\mathfrak{C} \times_C \mathfrak{D} \mapsto\mapsto_{C\alpha}$  cat-Set  $\alpha$ 
    by (rule NT.is-ntcf-axioms)
  from op-op- $\Phi$  have dom-lhs:
     $\mathcal{D}_\circ$  (op-cf-adj (op-cf-adj  $\Phi$ )(AdjNT)(NTMap)) = (op-cat  $\mathfrak{C} \times_C \mathfrak{D}$ )(Obj)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  show op-cf-adj (op-cf-adj  $\Phi$ )(AdjNT)(NTMap) =  $\Phi$ (AdjNT)(NTMap)
  proof(rule vsv-eqI, unfold NT.ntcf-NTMap-vdomain dom-lhs)
  fix cd assume prems: cd  $\in_\circ$  (op-cat  $\mathfrak{C} \times_C \mathfrak{D}$ )(Obj)
  then obtain c d
    where cd-def: cd = [c, d] $\circ$ 
    and c: c  $\in_\circ$  op-cat  $\mathfrak{C}$ (Obj)
    and d: d  $\in_\circ$   $\mathfrak{D}$ (Obj)
    by (elim cat-prod-2-ObjE[OF L.category-op R.category-axioms prems])
  from is-cf-adjunction-axioms c d L.category-axioms R.category-axioms  $\Phi$ 
  show op-cf-adj (op-cf-adj  $\Phi$ )(AdjNT)(NTMap)(cd) =  $\Phi$ (AdjNT)(NTMap)(cd)
  unfolding cd-def cat-op-simps
  by
  (
    cs-concl
    cs-intro:
      cat-arrow-cs-intros
      ntcf-cs-intros
      adj-cs-intros
      cat-op-intros
  )

```

```

    cat-cs-intros
    cat-prod-cs-intros
  cs-simp: cat-cs-simps cat-op-simps
)
qed (auto intro: inv-ntcf-NTMap-vsν)
qed simp-all
qed (auto intro: adj-cs-intros)

```

lemmas [cat-op-simps] = is-cf-adjunction.cf-adjunction-op-cf-adj-op-cf-adj

13.3.2 Alternative form of the naturality condition

The lemmas in this subsection are based on the comments on page 81 in [9].

lemma (in is-cf-adjunction) cf-adj-Comp-commute-RL:

```

assumes x ∈o C(Obj)
and f : F(ObjMap)(x) ↦D a
and k : a ↦D a'

```

shows

$$\mathfrak{G}(\text{ArrMap})(k) \circ_{A\mathfrak{C}} (\Phi(\text{AdjNT})(\text{NTMap})(x, a)_{\bullet})(\text{ArrVal})(f) =$$

$$(\Phi(\text{AdjNT})(\text{NTMap})(x, a')_{\bullet})(\text{ArrVal})(k \circ_{A\mathfrak{D}} f)$$

proof-

from

```

  assms
  is-cf-adjunction-axioms
  L.category-axioms R.category-axioms
  L.category-op R.category-op

```

have $\varphi\text{-}x\text{-}a$: $\Phi(\text{AdjNT})(\text{NTMap})(x, a)_{\bullet} :$

$\text{Hom } \mathfrak{D} (\mathfrak{F}(\text{ObjMap})(x)) a \mapsto_{\text{cat-Set } \alpha} \text{Hom } \mathfrak{C} x (\mathfrak{G}(\text{ObjMap})(a))$

by

```

(
  cs-concl
  cs-simp: cat-cs-simps
  cs-intro: cat-cs-intros cat-op-intros adj-cs-intros cat-prod-cs-intros
)

```

note $\varphi\text{-}x\text{-}a\text{-}f =$

$\text{cat-Set-ArrVal-app-vrange}[OF \ \varphi\text{-}x\text{-}a, \text{unfolded in-Hom-iff}, OF \ \text{assms}(2)]$

from

```

  is-cf-adjunction-axioms assms
  L.category-axioms R.category-axioms
  L.category-op R.category-op

```

have $\varphi\text{-}x\text{-}a'$:

$\Phi(\text{AdjNT})(\text{NTMap})(x, a')_{\bullet} :$

$\text{Hom } \mathfrak{D} (\mathfrak{F}(\text{ObjMap})(x)) a' \mapsto_{\text{cat-Set } \alpha} \text{Hom } \mathfrak{C} x (\mathfrak{G}(\text{ObjMap})(a'))$

by

```

(
  cs-concl
  cs-simp: cat-cs-simps
  cs-intro: cat-cs-intros cat-op-intros adj-cs-intros cat-prod-cs-intros
)

```

from is-cf-adjunction-axioms this assms **have** $x\text{-}k$:

$[\mathfrak{C}(\text{CId})(x), k]_{\circ} : [x, a]_{\circ} \mapsto_{\text{op-cat } \mathfrak{C} \times_C \mathfrak{D}} [x, a']_{\circ}$

by

```

(
  cs-concl
  cs-simp: cat-cs-simps
  cs-intro: cat-cs-intros cat-op-intros adj-cs-intros cat-prod-cs-intros
)

```

from

NT.ntcf-Comp-commute[*OF this*] *is-cf-adjunction-axioms* *assms*
L.category-axioms *R.category-axioms*
L.category-op *R.category-op*

have

$\Phi(\text{AdjNT})(\text{NTMap})(x, a') \bullet \circ_{A \text{ cat-Set } \alpha} \text{cf-hom } \mathfrak{D} [\mathfrak{D}(\text{CIde})(\mathfrak{F}(\text{ObjMap})(x)), k] \circ =$
 $\text{cf-hom } \mathfrak{C} [\mathfrak{C}(\text{CIde})(x), \mathfrak{G}(\text{ArrMap})(k)] \circ_{A \text{ cat-Set } \alpha} \Phi(\text{AdjNT})(\text{NTMap})(x, a) \bullet$
(is < ?lhs = ?rhs >)

by

(
cs-prems **cs-ist-simple**
cs-simp: *cat-cs-simps*
cs-intro: *cat-cs-intros* *cat-op-intros* *adj-cs-intros* *cat-prod-cs-intros*
)

moreover from

is-cf-adjunction-axioms *assms* φ -*x-a'*
L.category-axioms *R.category-axioms*
L.category-op *R.category-op*

have $?\text{lhs}(\text{ArrVal})(f) = (\Phi(\text{AdjNT})(\text{NTMap})(x, a') \bullet)(\text{ArrVal})(k \circ_{A \mathfrak{D}} f)$

by

(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps*
cs-intro: *cat-cs-intros* *cat-op-intros* *adj-cs-intros* *cat-prod-cs-intros*
)

moreover from

is-cf-adjunction-axioms *assms* φ -*x-a-f*
L.category-axioms *R.category-axioms*
L.category-op *R.category-op*

have

$?\text{rhs}(\text{ArrVal})(f) = \mathfrak{G}(\text{ArrMap})(k) \circ_{A \mathfrak{C}} (\Phi(\text{AdjNT})(\text{NTMap})(x, a) \bullet)(\text{ArrVal})(f)$

by

(
cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-cs-intros* *cat-op-intros* *adj-cs-intros* *cat-prod-cs-intros*
)

ultimately show *?thesis* **by** *simp*

qed

lemma (in *is-cf-adjunction*) *cf-adj-Comp-commute-LR*:

assumes $x \in \circ \mathfrak{C}(\text{Obj})$

and $f : \mathfrak{F}(\text{ObjMap})(x) \mapsto_{\mathfrak{D}} a$

and $h : x' \mapsto_{\mathfrak{C}} x$

shows

$(\Phi(\text{AdjNT})(\text{NTMap})(x, a) \bullet)(\text{ArrVal})(f) \circ_{A \mathfrak{C}} h =$
 $(\Phi(\text{AdjNT})(\text{NTMap})(x', a) \bullet)(\text{ArrVal})(f \circ_{A \mathfrak{D}} \mathfrak{F}(\text{ArrMap})(h))$

proof-

from

is-cf-adjunction-axioms *assms*
L.category-axioms *R.category-axioms*
L.category-op *R.category-op*

have φ -*x-a*: $\Phi(\text{AdjNT})(\text{NTMap})(x, a) \bullet :$

$\text{Hom } \mathfrak{D} (\mathfrak{F}(\text{ObjMap})(x)) a \mapsto_{\text{cat-Set } \alpha} \text{Hom } \mathfrak{C} x (\mathfrak{G}(\text{ObjMap})(a))$

by

(
cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-cs-intros* *cat-op-intros* *adj-cs-intros* *cat-prod-cs-intros*
)

)

note $\varphi\text{-}x\text{-}a\text{-}f =$
cat-Set-ArrVal-app-vrange[*OF* $\varphi\text{-}x\text{-}a$, *unfolded in-Hom-iff*, *OF assms*(2)]

from *is-cf-adjunction-axioms assms* **have**
 $[h, \mathfrak{D}(\mathfrak{CId})(a)]_{\circ} : [x, a]_{\circ} \mapsto_{op\text{-}cat} \mathfrak{C} \times_C \mathfrak{D} [x', a]_{\circ}$
by
(
cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-cs-intros cat-op-intros adj-cs-intros cat-prod-cs-intros*
))

from
NT.ntcf-Comp-commute[*OF this*] *is-cf-adjunction-axioms assms*
L.category-axioms R.category-axioms
L.category-op R.category-op

have
 $\Phi(\mathfrak{AdjNT})(\mathfrak{NTMap})(x', a)_{\bullet} \circ_{A\text{cat-Set } \alpha} \text{cf-hom } \mathfrak{D} [\mathfrak{F}(\mathfrak{ArrMap})(h), \mathfrak{D}(\mathfrak{CId})(a)]_{\circ} =$
 $\text{cf-hom } \mathfrak{C} [h, \mathfrak{C}(\mathfrak{CId})(\mathfrak{G}(\mathfrak{ObjMap})(a))]_{\circ} \circ_{A\text{cat-Set } \alpha} \Phi(\mathfrak{AdjNT})(\mathfrak{NTMap})(x, a)_{\bullet}$
(**is** $\langle ?lhs = ?rhs \rangle$)
by
(
cs-prems
cs-simp: *cat-cs-simps*
cs-intro: *cat-cs-intros cat-op-intros adj-cs-intros cat-prod-cs-intros*
))

moreover from
is-cf-adjunction-axioms assms
L.category-axioms R.category-axioms
L.category-op R.category-op

have $?lhs(\mathfrak{ArrVal})(f) = (\Phi(\mathfrak{AdjNT})(\mathfrak{NTMap})(x', a)_{\bullet})(\mathfrak{ArrVal})(f \circ_A \mathfrak{D} \mathfrak{F}(\mathfrak{ArrMap})(h))$
by
(
cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-cs-intros cat-op-intros adj-cs-intros cat-prod-cs-intros*
))

moreover from
is-cf-adjunction-axioms assms $\varphi\text{-}x\text{-}a\text{-}f$
L.category-axioms R.category-axioms
L.category-op R.category-op

have $?rhs(\mathfrak{ArrVal})(f) = (\Phi(\mathfrak{AdjNT})(\mathfrak{NTMap})(x, a)_{\bullet})(\mathfrak{ArrVal})(f) \circ_A \mathfrak{C} h$
by
(
cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-cs-intros cat-op-intros adj-cs-intros cat-prod-cs-intros*
))

ultimately show *?thesis* **by** *simp*

qed

13.4 Unit

13.4.1 Definition and elementary properties

See Chapter IV-1 in [9].

definition *cf-adjunction-unit* :: $V \Rightarrow V$ ($\langle \eta_C \rangle$)

where $\eta_C \Phi =$

[

```

(
  λxεo.Φ(AdjLeft)(HomDom)(Obj).
  (Φ(AdjNT)(NTMap)(x, Φ(AdjLeft)(ObjMap)(x)))(ArrVal)(
    Φ(AdjLeft)(HomCod)(CId)(Φ(AdjLeft)(ObjMap)(x))
  )
),
cf-id (Φ(AdjLeft)(HomDom)),
(Φ(AdjRight)) ∘CF (Φ(AdjLeft)),
Φ(AdjLeft)(HomDom),
Φ(AdjLeft)(HomDom)
]₀.

```

Components.

lemma *cf-adjunction-unit-components*:

```

shows ηC Φ(NTMap) =
(
  λxεo.Φ(AdjLeft)(HomDom)(Obj).
  (Φ(AdjNT)(NTMap)(x, Φ(AdjLeft)(ObjMap)(x)))(ArrVal)(
    Φ(AdjLeft)(HomCod)(CId)(Φ(AdjLeft)(ObjMap)(x))
  )
)
and ηC Φ(NTDom) = cf-id (Φ(AdjLeft)(HomDom))
and ηC Φ(NTCod) = (Φ(AdjRight)) ∘CF (Φ(AdjLeft))
and ηC Φ(NTDGDom) = Φ(AdjLeft)(HomDom)
and ηC Φ(NTDGCod) = Φ(AdjLeft)(HomDom)
unfolding cf-adjunction-unit-def nt-field-simps
by (simp-all add: nat-omega-simps)

```

context *is-cf-adjunction*

begin

lemma *cf-adjunction-unit-components'*:

```

shows ηC Φ(NTMap) =
(λxεo.℄(Obj). (Φ(AdjNT)(NTMap)(x, ℄(ObjMap)(x)))(ArrVal)(℄(CId)(℄(ObjMap)(x))))
and ηC Φ(NTDom) = cf-id ℄
and ηC Φ(NTCod) = ℄ ∘CF ℄
and ηC Φ(NTDGDom) = ℄
and ηC Φ(NTDGCod) = ℄
unfolding cf-adjunction-unit-components
by (cs-concl cs-shallow cs-simp: cat-cs-simps adj-cs-simps)+

```

mk-VLambda *cf-adjunction-unit-components'(1)*

```

|vdomain cf-adjunction-unit-NTMap-vdomain[adj-cs-simps]|
|app cf-adjunction-unit-NTMap-app[adj-cs-simps]|

```

end

mk-VLambda *cf-adjunction-unit-components(1)*

```

|vsu cf-adjunction-unit-NTMap-vsuv[adj-cs-intros]|

```

lemmas [*adj-cs-simps*] =

```

is-cf-adjunction.cf-adjunction-unit-NTMap-vdomain
is-cf-adjunction.cf-adjunction-unit-NTMap-app

```

13.4.2 Natural transformation map

lemma (**in** *is-cf-adjunction*) *cf-adjunction-unit-NTMap-is-arr*:

```

assumes x εo ℄(Obj)

```

shows $\eta_C \Phi(\text{NTMap})(x) : x \mapsto_{\mathcal{C}} \mathfrak{G}(\text{ObjMap})(\mathfrak{F}(\text{ObjMap})(x))$

proof-

from

is-cf-adjunction-axioms *assms*

L.category-axioms *R.category-axioms*

L.category-op *R.category-op*

have $\varphi\text{-}\mathfrak{F}x$:

$\Phi(\text{AdjNT})(\text{NTMap})(x, \mathfrak{F}(\text{ObjMap})(x))_{\bullet} :$
 $\text{Hom } \mathfrak{D} (\mathfrak{F}(\text{ObjMap})(x)) (\mathfrak{F}(\text{ObjMap})(x)) \mapsto_{\text{cat-Set } \alpha}$
 $\text{Hom } \mathcal{C} x (\mathfrak{G}(\text{ObjMap})(\mathfrak{F}(\text{ObjMap})(x)))$

by

(
cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-cs-intros* *cat-op-intros* *adj-cs-intros* *cat-prod-cs-intros*
)

from *is-cf-adjunction-axioms* *assms* **have** *CId-}\mathfrak{F}x*:

$\mathfrak{D}(\text{CId})(\mathfrak{F}(\text{ObjMap})(x)) : \mathfrak{F}(\text{ObjMap})(x) \mapsto_{\mathfrak{D}} \mathfrak{F}(\text{ObjMap})(x)$

by (*cs-concl* **cs-intro:** *cat-cs-intros* *adj-cs-intros*)

from

is-cf-adjunction-axioms

assms

cat-Set-ArrVal-app-vrange[*OF* $\varphi\text{-}\mathfrak{F}x$, *unfolded in-Hom-iff*, *OF CId-}\mathfrak{F}x*]

show $\eta_C \Phi(\text{NTMap})(x) : x \mapsto_{\mathcal{C}} \mathfrak{G}(\text{ObjMap})(\mathfrak{F}(\text{ObjMap})(x))$

by (*cs-concl* **cs-shallow** **cs-simp:** *adj-cs-simps* **cs-intro:** *cat-cs-intros*)

qed

lemma (**in** *is-cf-adjunction*) *cf-adjunction-unit-NTMap-is-arr'*:

assumes $x \in_{\circ} \mathcal{C}(\text{Obj})$

and $a = x$

and $b = \mathfrak{G}(\text{ObjMap})(\mathfrak{F}(\text{ObjMap})(x))$

and $\mathcal{C}' = \mathcal{C}$

shows $\eta_C \Phi(\text{NTMap})(x) : x \mapsto_{\mathcal{C}'} b$

using *assms*(1) **unfolding** *assms*(2-4) **by** (*rule cf-adjunction-unit-NTMap-is-arr'*)

lemmas [*adj-cs-intros*] = *is-cf-adjunction.cf-adjunction-unit-NTMap-is-arr'*

lemma (**in** *is-cf-adjunction*) *cf-adjunction-unit-NTMap-vrange*:

$\mathcal{R}_{\circ}(\eta_C \Phi(\text{NTMap})) \subseteq_{\circ} \mathcal{C}(\text{Arr})$

proof(*rule vsv.vsv-vrange-vsubset*, *unfold cf-adjunction-unit-NTMap-vdomain*)

fix x **assume** *prems*: $x \in_{\circ} \mathcal{C}(\text{Obj})$

from *cf-adjunction-unit-NTMap-is-arr*[*OF prems*] **show** $\eta_C \Phi(\text{NTMap})(x) \in_{\circ} \mathcal{C}(\text{Arr})$

by *auto*

qed (*auto intro: adj-cs-intros*)

13.4.3 Unit is a natural transformation

lemma (**in** *is-cf-adjunction*) *cf-adjunction-unit-is-ntcf*:

$\eta_C \Phi : \text{cf-id } \mathcal{C} \mapsto_{CF} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathcal{C} \mapsto_{C\alpha} \mathcal{C}$

proof(*intro is-ntcfI'*)

show *vfsequence* ($\eta_C \Phi$) **unfolding** *cf-adjunction-unit-def* **by** *simp*

show *vcard* ($\eta_C \Phi$) = $5_{\mathbb{N}}$

unfolding *cf-adjunction-unit-def* **by** (*simp add: nat-omega-simps*)

from *is-cf-adjunction-axioms* **show** *cf-id* $\mathcal{C} : \mathcal{C} \mapsto_{C\alpha} \mathcal{C}$

by (*cs-concl* **cs-simp:** *cat-cs-simps* **cs-intro:** *cat-cs-intros* *adj-cs-intros*)

from *is-cf-adjunction-axioms* **show** $\mathfrak{G} \circ_{CF} \mathfrak{F} : \mathcal{C} \mapsto_{C\alpha} \mathcal{C}$

by (*cs-concl* **cs-simp:** *cat-cs-simps* **cs-intro:** *cat-cs-intros* *adj-cs-intros*)

from *is-cf-adjunction-axioms* **show** $\mathcal{D}_{\circ}(\eta_C \Phi(\text{NTMap})) = \mathcal{C}(\text{Obj})$

by (*cs-concl* **cs-shallow** **cs-simp**: *adj-cs-simps* **cs-intro**: *cat-cs-intros*)
show $\eta_C \Phi(\backslash NTMap)(\backslash a) : cf-id \mathfrak{C}(\backslash ObjMap)(\backslash a) \mapsto_{\mathfrak{C}} (\mathfrak{G} \circ_{CF} \mathfrak{F})(\backslash ObjMap)(\backslash a)$
if $a \in_{\circ} \mathfrak{C}(\backslash Obj)$ **for** a
using *is-cf-adjunction-axioms* **that**
 by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros* *adj-cs-intros*)
show
 $\eta_C \Phi(\backslash NTMap)(\backslash b) \circ_{A\mathfrak{C}} cf-id \mathfrak{C}(\backslash ArrMap)(\backslash f) =$
 $(\mathfrak{G} \circ_{CF} \mathfrak{F})(\backslash ArrMap)(\backslash f) \circ_{A\mathfrak{C}} \eta_C \Phi(\backslash NTMap)(\backslash a)$
if $f : a \mapsto_{\mathfrak{C}} b$ **for** $a b f$
using *is-cf-adjunction-axioms* **that**
by
 (

- cs-concl*
- cs-simp**:
 - cf-adj-Comp-commute-RL* *cf-adj-Comp-commute-LR*
 - cat-cs-simps*
 - adj-cs-simps*
- cs-intro**: *cat-cs-intros* *adj-cs-intros*

)

qed (*auto simp: cf-adjunction-unit-components'*)

lemma (**in** *is-cf-adjunction*) *cf-adjunction-unit-is-ntcf'*:
assumes $\mathfrak{G} = cf-id \mathfrak{C}$
and $\mathfrak{G}' = \mathfrak{G} \circ_{CF} \mathfrak{F}$
and $\mathfrak{A} = \mathfrak{C}$
and $\mathfrak{B} = \mathfrak{C}$
shows $\eta_C \Phi : \mathfrak{G} \mapsto_{CF} \mathfrak{G}' : \mathfrak{A} \mapsto_{C\alpha} \mathfrak{B}$
unfolding *assms* **by** (*rule cf-adjunction-unit-is-ntcf'*)

lemmas [*adj-cs-intros*] = *is-cf-adjunction.cf-adjunction-unit-is-ntcf'*

13.4.4 Every component of a unit is a universal arrow

The lemmas in this subsection are based on elements of the statement of Theorem 1 in Chapter IV-1 in [9].

lemma (**in** *is-cf-adjunction*) *cf-adj-umap-of-unit*:
assumes $x \in_{\circ} \mathfrak{C}(\backslash Obj)$ **and** $a \in_{\circ} \mathfrak{D}(\backslash Obj)$
shows $\Phi(\backslash AdjNT)(\backslash NTMap)(\backslash x, a)_{\bullet} = umap-of \mathfrak{G} x (\mathfrak{F}(\backslash ObjMap)(\backslash x)) (\eta_C \Phi(\backslash NTMap)(\backslash x)) a$
 (**is** $\langle \Phi(\backslash AdjNT)(\backslash NTMap)(\backslash x, a)_{\bullet} = ?uof-a \rangle$)

proof-

from

is-cf-adjunction-axioms *assms*
L.category-axioms *R.category-axioms*
L.category-op *R.category-op*

have $\varphi\text{-}xa: \Phi(\backslash AdjNT)(\backslash NTMap)(\backslash x, a)_{\bullet} :$
 $Hom \mathfrak{D} (\mathfrak{F}(\backslash ObjMap)(\backslash x)) a \mapsto_{cat\text{-}Set \alpha} Hom \mathfrak{C} x (\mathfrak{G}(\backslash ObjMap)(\backslash a))$

by

(

- cs-concl*
- cs-simp**: *cat-cs-simps*
- cs-intro**: *cat-cs-intros* *cat-op-intros* *adj-cs-intros* *cat-prod-cs-intros*

)

then have *dom-lhs*:

$\mathcal{D}_{\circ} ((\Phi(\backslash AdjNT)(\backslash NTMap)(\backslash x, a)_{\bullet})(\backslash ArrVal)) = Hom \mathfrak{D} (\mathfrak{F}(\backslash ObjMap)(\backslash x)) a$

by (*cs-concl* **cs-shallow** **cs-simp**: *cat-cs-simps*)

from *is-cf-adjunction-axioms* *assms* **have** *uof-a*:

$?uof-a : Hom \mathcal{D} (\mathfrak{F}(\mathcal{ObjMap})(\downarrow x)) a \mapsto_{cat-Set \alpha} Hom \mathfrak{C} x (\mathfrak{G}(\mathcal{ObjMap})(\downarrow a))$
by (*cs-concl cs-intro: cat-cs-intros adj-cs-intros*)
then have *dom-rhs*: $\mathcal{D}_o (?uof-a(\downarrow ArrVal)) = Hom \mathcal{D} (\mathfrak{F}(\mathcal{ObjMap})(\downarrow x)) a$
by (*cs-concl cs-simp: cat-cs-simps*)

show *?thesis*

proof(*rule arr-Set-eqI[of α]*)

from φ -*xa* **show** *arr-Set- φ -xa*: $arr-Set \alpha (\Phi(\downarrow AdjNT)(\downarrow NTMap)(\downarrow x, a)\bullet)$

by (*auto dest: cat-Set-is-arrD(1)*)

from *uof-a* **show** *arr-Set-uof-a*: $arr-Set \alpha ?uof-a$

by (*auto dest: cat-Set-is-arrD(1)*)

show $(\Phi(\downarrow AdjNT)(\downarrow NTMap)(\downarrow x, a)\bullet)(\downarrow ArrVal) = ?uof-a(\downarrow ArrVal)$

proof(*rule vsv-eqI, unfold dom-lhs dom-rhs in-Hom-iff*)

fix *g* **assume** *prems*: $g : \mathfrak{F}(\mathcal{ObjMap})(\downarrow x) \mapsto_{\mathcal{D}} a$

from *is-cf-adjunction-axioms* *assms* *prems* **show**

$(\Phi(\downarrow AdjNT)(\downarrow NTMap)(\downarrow x, a)\bullet)(\downarrow ArrVal)(\downarrow g) = ?uof-a(\downarrow ArrVal)(\downarrow g)$

by

(

cs-concl cs-shallow

cs-simp:

cf-adj-Comp-commute-RL

adj-cs-simps

cat-cs-simps

cat-op-simps

cat-prod-cs-simps

cs-intro:

adj-cs-intros

ntcf-cs-intros

cat-cs-intros

cat-op-intros

cat-prod-cs-intros

)

qed (*use arr-Set- φ -xa arr-Set-uof-a in auto*)

qed (*use φ -xa uof-a in $\langle cs-concl cs-shallow cs-simp: cat-cs-simps \rangle$*)⁺

qed

lemma (*in is-cf-adjunction*) *cf-adj-umap-of-unit'*:

assumes $x \in_o \mathfrak{C}(\mathcal{Obj})$

and $a \in_o \mathcal{D}(\mathcal{Obj})$

and $\eta = \eta_C \Phi(\downarrow NTMap)(\downarrow x)$

and $\mathfrak{F}x = \mathfrak{F}(\mathcal{ObjMap})(\downarrow x)$

shows $\Phi(\downarrow AdjNT)(\downarrow NTMap)(\downarrow x, a)\bullet = umap-of \mathfrak{G} x \mathfrak{F}x \eta a$

using *assms(1,2)* **unfolding** *assms(3,4)* **by** (*rule cf-adj-umap-of-unit*)

lemma (*in is-cf-adjunction*) *cf-adjunction-unit-component-is-ua-of*:

assumes $x \in_o \mathfrak{C}(\mathcal{Obj})$

shows *universal-arrow-of $\mathfrak{G} x (\mathfrak{F}(\mathcal{ObjMap})(\downarrow x)) (\eta_C \Phi(\downarrow NTMap)(\downarrow x))$*

(*is $\langle universal-arrow-of \mathfrak{G} x (\mathfrak{F}(\mathcal{ObjMap})(\downarrow x)) ?\eta x \rangle$*)

proof(*rule RL.cf-ua-of-if-ntcf-ua-of-is-iso-ntcf*)

from *is-cf-adjunction-axioms* *assms* **show** $\mathfrak{F}(\mathcal{ObjMap})(\downarrow x) \in_o \mathcal{D}(\mathcal{Obj})$

by (*cs-concl cs-shallow cs-intro: cat-cs-intros adj-cs-intros*)

from *is-cf-adjunction-axioms* *assms* **show**

$\eta_C \Phi(\downarrow NTMap)(\downarrow x) : x \mapsto_{\mathfrak{C}} \mathfrak{G}(\mathcal{ObjMap})(\downarrow \mathfrak{F}(\mathcal{ObjMap})(\downarrow x))$

by (*cs-concl cs-shallow cs-intro: cat-cs-intros adj-cs-intros*)

show

ntcf-ua-of $\alpha \mathfrak{G} x (\mathfrak{F}(\mathcal{ObjMap})(\downarrow x)) (\eta_C \Phi(\downarrow NTMap)(\downarrow x))$:

$Hom_{O.C\alpha} \mathfrak{D}(\mathfrak{F}(\text{ObjMap})(x), -) \mapsto_{CF.iso} Hom_{O.C\alpha} \mathfrak{C}(x, -) \circ_{CF} \mathfrak{G} :$
 $\mathfrak{D} \mapsto_{C\alpha} \text{cat-Set } \alpha$
(is $\langle ?ntcf-ua-of : ?H\mathfrak{F} \mapsto_{CF.iso} ?H\mathfrak{G} : \mathfrak{D} \mapsto_{C\alpha} \text{cat-Set } \alpha \rangle$)
proof(rule *is-iso-ntcfI*)
from *is-cf-adjunction-axioms* **assms** **show**
 $?ntcf-ua-of : ?H\mathfrak{F} \mapsto_{CF} ?H\mathfrak{G} : \mathfrak{D} \mapsto_{C\alpha} \text{cat-Set } \alpha$
by (intro *RL.cf-ntcf-ua-of-is-ntcf*)
(cs-concl **cs-shallow cs-intro**: *cat-cs-intros adj-cs-intros*)+
fix *a* **assume** *prems*: $a \in_o \mathfrak{D}(\text{Obj})$
from *assms prems* **have**
 $\Phi(\text{AdjNT})(\text{NTMap})(x, a) \bullet = \text{umap-of } \mathfrak{G} \ x \ (\mathfrak{F}(\text{ObjMap})(x)) \ ?\eta \ x \ a$
(is $\langle \Phi(\text{AdjNT})(\text{NTMap})(x, a) \bullet = ?uof-a \rangle$)
by (rule *cf-adj-umap-of-unit*)
from *assms prems L.category-axioms R.category-axioms* **have**
 $[x, a]_o \in_o (\text{op-cat } \mathfrak{C} \times_C \mathfrak{D})(\text{Obj})$
by (cs-concl **cs-shallow cs-intro**: *cat-op-intros cat-prod-cs-intros*)
from
 $NT.iso-ntcf-is-iso-arr[$
 $OF \text{ this, unfolded cf-adj-umap-of-unit}[OF \text{ assms prems}]$
 $]$
is-cf-adjunction-axioms *assms prems*
L.category-axioms R.category-axioms
have $?uof-a : Hom \ \mathfrak{D} \ (\mathfrak{F}(\text{ObjMap})(x)) \ a \mapsto_{iso \text{ cat-Set } \alpha} Hom \ \mathfrak{C} \ x \ (\mathfrak{G}(\text{ObjMap})(a))$
by
(
cs-prems
cs-simp: *cat-cs-simps*
cs-intro:
cat-cs-intros cat-op-intros adj-cs-intros cat-prod-cs-intros
)
with *is-cf-adjunction-axioms* *assms prems* **show**
 $?ntcf-ua-of(\text{NTMap})(a) : ?H\mathfrak{F}(\text{ObjMap})(a) \mapsto_{iso \text{ cat-Set } \alpha} ?H\mathfrak{G}(\text{ObjMap})(a)$
by
(
cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-cs-intros cat-op-intros adj-cs-intros*
)
qed
qed

13.5 Counit

13.5.1 Definition and elementary properties

definition *cf-adjunction-counit* :: $V \Rightarrow V \ (\langle \varepsilon_C \rangle)$

where $\varepsilon_C \ \Phi =$

[
(
 $\lambda x \in_o \Phi(\text{AdjLeft})(\text{HomCod})(\text{Obj}).$
 $(\Phi(\text{AdjNT})(\text{NTMap})(\Phi(\text{AdjRight})(\text{ObjMap})(x), x) \bullet)^{-1}_{Set}(\text{ArrVal})(\Phi(\text{AdjLeft})(\text{HomDom})(\text{CIId})(\Phi(\text{AdjRight})(\text{ObjMap})(x)))$
)
),
 $(\Phi(\text{AdjLeft})) \circ_{CF} (\Phi(\text{AdjRight})),$
cf-id $(\Phi(\text{AdjLeft})(\text{HomCod})),$
 $\Phi(\text{AdjLeft})(\text{HomCod}),$
 $\Phi(\text{AdjLeft})(\text{HomCod})$

]

Components.

lemma *cf-adjunction-counit-components*:

shows $\varepsilon_C \Phi(\text{NTMap}) =$
 ($\lambda x \in \circ \Phi(\text{AdjLeft})(\text{HomCod})(\text{Obj})$.
 ($\Phi(\text{AdjNT})(\text{NTMap})(\Phi(\text{AdjRight})(\text{ObjMap})(x), x) \bullet$)⁻¹ $\text{Set}(\text{ArrVal})($
 $\Phi(\text{AdjLeft})(\text{HomDom})(\text{CId})(\Phi(\text{AdjRight})(\text{ObjMap})(x))$
 $)$
)
and $\varepsilon_C \Phi(\text{NTDom}) = (\Phi(\text{AdjLeft})) \circ_{CF} (\Phi(\text{AdjRight}))$
and $\varepsilon_C \Phi(\text{NTCod}) = \text{cf-id } (\Phi(\text{AdjLeft})(\text{HomCod}))$
and $\varepsilon_C \Phi(\text{NTDGDom}) = \Phi(\text{AdjLeft})(\text{HomCod})$
and $\varepsilon_C \Phi(\text{NTDGCod}) = \Phi(\text{AdjLeft})(\text{HomCod})$
unfolding *cf-adjunction-counit-def nt-field-simps*
by (*simp-all add: nat-omega-simps*)

context *is-cf-adjunction*

begin

lemma *cf-adjunction-counit-components'*:

shows $\varepsilon_C \Phi(\text{NTMap}) =$
 ($\lambda x \in \circ \mathfrak{D}(\text{Obj})$.
 ($\Phi(\text{AdjNT})(\text{NTMap})(\mathfrak{C}(\text{ObjMap})(x), x) \bullet$)⁻¹ $\text{Set}(\text{ArrVal})(\mathfrak{C}(\text{CId})(\mathfrak{C}(\text{ObjMap})(x)))$
)
and $\varepsilon_C \Phi(\text{NTDom}) = \mathfrak{F} \circ_{CF} \mathfrak{C}$
and $\varepsilon_C \Phi(\text{NTCod}) = \text{cf-id } \mathfrak{D}$
and $\varepsilon_C \Phi(\text{NTDGDom}) = \mathfrak{D}$
and $\varepsilon_C \Phi(\text{NTDGCod}) = \mathfrak{D}$
unfolding *cf-adjunction-counit-components*
by (*cs-concl cs-shallow cs-simp: cat-cs-simps adj-cs-simps*)+

mk-VLambda *cf-adjunction-counit-components'(1)*

$|\text{vdomain } \text{cf-adjunction-counit-NTMap-vdomain}[\text{adj-cs-simps}]|$
 $|\text{app } \text{cf-adjunction-counit-NTMap-app}[\text{adj-cs-simps}]|$

end

mk-VLambda *cf-adjunction-counit-components(1)*

$|\text{vsu } \text{cf-adjunction-counit-NTMap-vsuv}[\text{adj-cs-intros}]|$

lemmas [*adj-cs-simps*] =

is-cf-adjunction.cf-adjunction-counit-NTMap-vdomain
is-cf-adjunction.cf-adjunction-counit-NTMap-app

13.5.2 Duality for the unit and counit

lemma (**in** *is-cf-adjunction*) *cf-adjunction-unit-NTMap-op*:

$\eta_C (\text{op-cf-adj } \Phi)(\text{NTMap}) = \varepsilon_C \Phi(\text{NTMap})$

proof–

interpret *op-Φ*:

is-cf-adjunction α $\langle \text{op-cat } \mathfrak{D} \rangle$ $\langle \text{op-cat } \mathfrak{C} \rangle$ $\langle \text{op-cf } \mathfrak{G} \rangle$ $\langle \text{op-cf } \mathfrak{F} \rangle$ $\langle \text{op-cf-adj } \Phi \rangle$

by (*rule is-cf-adjunction-op*)

show *?thesis*

proof

(

```

    rule vsv-eqI,
    unfold
      cf-adjunction-counit-NTMap-vdomain
      op-Φ.cf-adjunction-unit-NTMap-vdomain
  )
  fix a assume prems: a ∈o op-cat  $\mathfrak{D}$ (Obj)
  then have a: a ∈o  $\mathfrak{D}$ (Obj) unfolding cat-op-simps by simp
  from is-cf-adjunction-axioms a show
    ηC (op-cf-adj Φ)(NTMap)(a) = εC Φ(NTMap)(a)
  by
    (
      cs-concl cs-shallow
      cs-simp: cat-Set-cs-simps cat-cs-simps cat-op-simps adj-cs-simps
      cs-intro:
        cat-arrow-cs-intros cat-cs-intros cat-op-intros cat-prod-cs-intros
    )
  qed
  (
    simp-all add:
      cat-op-simps cf-adjunction-counit-NTMap-vsυ cf-adjunction-unit-NTMap-vsυ
  )
  qed

```

lemmas [cat-op-simps] = is-cf-adjunction.cf-adjunction-unit-NTMap-op

lemma (in is-cf-adjunction) cf-adjunction-counit-NTMap-op:

ε_C (op-cf-adj Φ)(NTMap) = η_C Φ(NTMap)

by

```

  (
    rule is-cf-adjunction.cf-adjunction-unit-NTMap-op[
      OF is-cf-adjunction-op,
      unfolded is-cf-adjunction.cf-adjunction-op-cf-adj-op-cf-adj[
        OF is-cf-adjunction-axioms
      ],
      unfolded cat-op-simps,
      symmetric
    ]
  )

```

lemmas [cat-op-simps] = is-cf-adjunction.cf-adjunction-counit-NTMap-op

lemma (in is-cf-adjunction) op-ntcf-cf-adjunction-counit:

op-ntcf (ε_C Φ) = η_C (op-cf-adj Φ)

(is ⟨?ε = ?η⟩)

proof(rule vsv-eqI)

interpret op-Φ:

is-cf-adjunction α ⟨op-cat \mathfrak{D} ⟩ ⟨op-cat \mathfrak{C} ⟩ ⟨op-cf \mathfrak{G} ⟩ ⟨op-cf \mathfrak{F} ⟩ ⟨op-cf-adj Φ⟩

by (rule is-cf-adjunction-op)

have dom-lhs: \mathcal{D}_o ?ε = 5_N **unfolding** op-ntcf-def **by** (simp add: nat-omega-simps)

have dom-rhs: \mathcal{D}_o ?η = 5_N

unfolding cf-adjunction-unit-def **by** (simp add: nat-omega-simps)

show \mathcal{D}_o ?ε = \mathcal{D}_o ?η **unfolding** dom-lhs dom-rhs **by** simp

show a ∈_o \mathcal{D}_o ?ε \implies ?ε(a) = ?η(a) **for** a

by

```

  (
    unfold dom-lhs,
    elim-in-numeral,
  )

```

```

    fold nt-field-simps,
    unfold cf-adjunction-unit-NTMap-op,
    unfold
      cf-adjunction-counit-components'
      cf-adjunction-unit-components'
      op-Φ.cf-adjunction-counit-components'
      op-Φ.cf-adjunction-unit-components'
      cat-op-simps
  )
  simp-all
qed (auto simp: op-ntcf-def cf-adjunction-unit-def)

lemmas [cat-op-simps] = is-cf-adjunction.op-ntcf-cf-adjunction-counit

lemma (in is-cf-adjunction) op-ntcf-cf-adjunction-unit:
  op-ntcf (ηC Φ) = εC (op-cf-adj Φ)
  (is ⟨?η = ?ε⟩)
proof(rule vsv-eqI)
  interpret op-Φ:
    is-cf-adjunction α ⟨op-cat D⟩ ⟨op-cat C⟩ ⟨op-cf G⟩ ⟨op-cf F⟩ ⟨op-cf-adj Φ⟩
  by (rule is-cf-adjunction-op)
  have dom-lhs: Do ?η = 5N
  unfolding op-ntcf-def by (simp add: nat-omega-simps)
  have dom-rhs: Do ?ε = 5N
  unfolding cf-adjunction-counit-def by (simp add: nat-omega-simps)
  show Do ?η = Do ?ε unfolding dom-lhs dom-rhs by simp
  show a ∈o Do ?η ⟹ ?η(|a|) = ?ε(|a|) for a
  by
    (
      unfold dom-lhs,
      elim-in-numeral,
      fold nt-field-simps,
      unfold cf-adjunction-counit-NTMap-op,
      unfold
        cf-adjunction-counit-components'
        cf-adjunction-unit-components'
        op-Φ.cf-adjunction-counit-components'
        op-Φ.cf-adjunction-unit-components'
        cat-op-simps
    )
  simp-all
qed (auto simp: op-ntcf-def cf-adjunction-counit-def)

```

lemmas [cat-op-simps] = is-cf-adjunction.op-ntcf-cf-adjunction-unit

13.5.3 Natural transformation map

```

lemma (in is-cf-adjunction) cf-adjunction-counit-NTMap-is-arr:
  assumes x ∈o D(|Obj|)
  shows εC Φ(|NTMap|)(|x|) : F(|ObjMap|)(|G(|ObjMap|)(|x|)|) ↦D x
proof-
  from assms have x: x ∈o op-cat D(|Obj|) unfolding cat-op-simps by simp
  show ?thesis
  by
    (
      rule is-cf-adjunction.cf-adjunction-unit-NTMap-is-arr[
        OF is-cf-adjunction-op x,
        unfolded cf-adjunction-unit-NTMap-op cat-op-simps
      ]
    )

```

)]
)
qed

lemma (in *is-cf-adjunction*) *cf-adjunction-counit-NTMap-is-arr'*:
assumes $x \in_{\circ} \mathfrak{D}(\mathfrak{Obj})$
and $a = \mathfrak{F}(\mathfrak{ObjMap})(\mathfrak{G}(\mathfrak{ObjMap})(x))$
and $b = x$
and $\mathfrak{D}' = \mathfrak{D}$
shows $\varepsilon_C \Phi(\mathfrak{NTMap})(x) : a \mapsto_{\mathfrak{D}'} b$
using *assms(1) unfolding assms(2-4)* **by** (rule *cf-adjunction-counit-NTMap-is-arr*)

lemmas [*adj-cs-intros*] = *is-cf-adjunction.cf-adjunction-counit-NTMap-is-arr'*

lemma (in *is-cf-adjunction*) *cf-adjunction-counit-NTMap-vrange*:
 $\mathcal{R}_{\circ}(\varepsilon_C \Phi(\mathfrak{NTMap})) \subseteq_{\circ} \mathfrak{D}(\mathfrak{Arr})$
by
 (
rule is-cf-adjunction.cf-adjunction-unit-NTMap-vrange[
OF is-cf-adjunction-op,
unfolded cf-adjunction-unit-NTMap-op cat-op-simps
]
)

13.5.4 Counit is a natural transformation

lemma (in *is-cf-adjunction*) *cf-adjunction-counit-is-ntcf*:
 $\varepsilon_C \Phi : \mathfrak{F} \circ_{CF} \mathfrak{G} \mapsto_{CF} \text{cf-id } \mathfrak{D} : \mathfrak{D} \mapsto_{C\alpha} \mathfrak{D}$
proof-
from *is-cf-adjunction.cf-adjunction-unit-is-ntcf*[*OF is-cf-adjunction-op*] **have**
 $\varepsilon_C \Phi :$
 $op\text{-cf } (op\text{-cf } \mathfrak{F} \circ_{CF} op\text{-cf } \mathfrak{G}) \mapsto_{CF} op\text{-cf } (cf\text{-id } (op\text{-cat } \mathfrak{D})) :$
 $op\text{-cat } (op\text{-cat } \mathfrak{D}) \mapsto_{C\alpha} op\text{-cat } (op\text{-cat } \mathfrak{D})$
unfolding
is-cf-adjunction.op-ntcf-cf-adjunction-unit[
OF is-cf-adjunction-op, unfolded cat-op-simps, symmetric
]
by (rule *is-ntcf.is-ntcf-op*)
then show *?thesis unfolding cat-op-simps* .
qed

lemma (in *is-cf-adjunction*) *cf-adjunction-counit-is-ntcf'*:
assumes $\mathfrak{G} = \mathfrak{F} \circ_{CF} \mathfrak{G}$
and $\mathfrak{G}' = \text{cf-id } \mathfrak{D}$
and $\mathfrak{A} = \mathfrak{D}$
and $\mathfrak{B} = \mathfrak{D}$
shows $\varepsilon_C \Phi : \mathfrak{G} \mapsto_{CF} \mathfrak{G}' : \mathfrak{A} \mapsto_{C\alpha} \mathfrak{B}$
unfolding *assms* **by** (rule *cf-adjunction-counit-is-ntcf*)

lemmas [*adj-cs-intros*] = *is-cf-adjunction.cf-adjunction-counit-is-ntcf'*

13.5.5 Every component of a counit is a universal arrow

The lemmas in this subsection are based on elements of the statement of Theorem 1 in Chapter IV-1 in [9].

lemma (in *is-cf-adjunction*) *cf-adj-umap-fo-counit*:
assumes $x \in_{\circ} \mathfrak{D}(\mathfrak{Obj})$ **and** $a \in_{\circ} \mathfrak{C}(\mathfrak{Obj})$
shows $op\text{-cf-adj } \Phi(\mathfrak{AdjNT})(\mathfrak{NTMap})(x, a)_{\bullet} =$

$umap\text{-}fo \ \mathfrak{F} \ x \ (\mathfrak{G}(\mathcal{O}bjMap)(x)) \ (\varepsilon_C \ \Phi(\mathcal{N}TMap)(x)) \ a$
by
 (

- rule *is-cf-adjunction.cf-adj-umap-of-unit*[
- OF is-cf-adjunction-op*,
- unfolded cat-op-simps*,
- OF assms*,
- unfolded cf-adjunction-unit-NTMap-op*

]
)

lemma (in *is-cf-adjunction*) *cf-adjunction-counit-component-is-ua-fo*:
assumes $x \in_o \ \mathfrak{D}(\mathcal{O}bj)$
shows $universal\text{-}arrow\text{-}fo \ \mathfrak{F} \ x \ (\mathfrak{G}(\mathcal{O}bjMap)(x)) \ (\varepsilon_C \ \Phi(\mathcal{N}TMap)(x))$
by
 (

- rule *is-cf-adjunction.cf-adjunction-unit-component-is-ua-of*[
- OF is-cf-adjunction-op*,
- unfolded cat-op-simps*,
- OF assms*,
- unfolded cf-adjunction-unit-NTMap-op*

]
)

13.5.6 Further properties

lemma (in *is-cf-adjunction*) *cf-adj-AdjNT-cf-adjunction-unit*:

— See Chapter IV-1 in [9].

assumes $x \in_o \ \mathfrak{C}(\mathcal{O}bj)$ **and** $f : \mathfrak{F}(\mathcal{O}bjMap)(x) \mapsto_{\mathfrak{D}} a$

shows

$$\mathfrak{G}(\mathcal{A}rrMap)(f) \circ_{A\mathfrak{C}} \eta_C \ \Phi(\mathcal{N}TMap)(x) = (\Phi(\mathcal{A}djNT)(\mathcal{N}TMap)(x, a) \bullet) (\mathcal{A}rrVal)(f)$$

proof—

from *assms(1)* **have** $\mathfrak{D}(\mathcal{C}Id)(\mathfrak{F}(\mathcal{O}bjMap)(x)) : \mathfrak{F}(\mathcal{O}bjMap)(x) \mapsto_{\mathfrak{D}} \mathfrak{F}(\mathcal{O}bjMap)(x)$

by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

from *cf-adj-Comp-commute-RL[OF assms(1) this assms(2)] assms* **show** *?thesis*

by

(

- cs-prems cs-shallow*
- cs-simp:**
- cat-cs-simps*
- is-cf-adjunction.cf-adjunction-unit-NTMap-app[symmetric]*
- cs-intro:** *adj-cs-intros*

)

qed

lemma (in *is-cf-adjunction*) *cf-adj-AdjNT-cf-adjunction-counit*:

— See Chapter IV-1 in [9].

assumes $x \in_o \ \mathfrak{D}(\mathcal{O}bj)$ **and** $g : a \mapsto_{\mathfrak{C}} \mathfrak{G}(\mathcal{O}bjMap)(x)$

shows

$$\varepsilon_C \ \Phi(\mathcal{N}TMap)(x) \circ_{A\mathfrak{D}} \mathfrak{F}(\mathcal{A}rrMap)(g) = (\Phi(\mathcal{A}djNT)(\mathcal{N}TMap)(a, x) \bullet)^{-1} \mathcal{C} \ cat\text{-}Set \ \alpha \ (\mathcal{A}rrVal)(g)$$

using

is-cf-adjunction.cf-adj-AdjNT-cf-adjunction-unit
 [

- OF is-cf-adjunction-op*,
- unfolded cat-op-simps cf-adjunction-unit-NTMap-op*,
- OF assms*

]

$\]$
assms
by
 $($
cs-prems
cs-simp: *cat-cs-simps cat-op-simps*
cs-intro:
cat-cs-intros
adj-cs-intros
cat-op-intros
cat-prod-cs-intros
 $)$

lemma (in *is-cf-adjunction*) *cf-adj-counit-unit-app*[*adj-cs-simps*]:

— See Chapter IV-1 in [9].

assumes $x \in_{\circ} \mathfrak{D}(\text{Obj})$ **and** $g : a \mapsto_{\mathfrak{C}} \mathfrak{G}(\text{ObjMap})(x)$

shows $\mathfrak{G}(\text{ArrMap})(\varepsilon_C \Phi(\text{NTMap})(x) \circ_{A\mathfrak{D}} \mathfrak{F}(\text{ArrMap})(g)) \circ_{A\mathfrak{C}} \eta_C \Phi(\text{NTMap})(a) = g$

proof—

from *assms*(2) **have** $a : a \in_{\circ} \mathfrak{C}(\text{Obj})$ **by** *auto*

from *assms* **have** *inv-Φ-g*:

$(\Phi(\text{AdjNT})(\text{NTMap})(a, x) \bullet)^{-1} \text{C}_{\text{cat-Set } \alpha}(\text{ArrVal})(g) : \mathfrak{F}(\text{ObjMap})(a) \mapsto_{\mathfrak{D}} x$

by

$($
cs-concl
cs-simp: *cat-cs-simps cat-op-simps*
cs-intro:
cat-arrow-cs-intros
cat-cs-intros
adj-cs-intros
cat-prod-cs-intros
cat-op-intros
 $)$

from *assms* **show** *?thesis*

unfolding

cf-adj-AdjNT-cf-adjunction-counit[*OF assms*]

cf-adj-AdjNT-cf-adjunction-unit[*OF a inv-Φ-g*]

by

$($
cs-concl
cs-simp: *cat-cs-simps cat-op-simps*
cs-intro:
cat-arrow-cs-intros
cat-cs-intros
adj-cs-intros
cat-prod-cs-intros
cat-op-intros
 $)$

qed

lemmas [*cat-cs-simps*] = *is-cf-adjunction.cf-adj-counit-unit-app*

lemma (in *is-cf-adjunction*) *cf-adj-unit-counit-app*[*adj-cs-simps*]:

— See Chapter IV-1 in [9].

assumes $x \in_{\circ} \mathfrak{C}(\text{Obj})$ **and** $f : \mathfrak{F}(\text{ObjMap})(x) \mapsto_{\mathfrak{D}} a$

shows $\varepsilon_C \Phi(\text{NTMap})(a) \circ_{A\mathfrak{D}} \mathfrak{F}(\text{ArrMap})(\mathfrak{G}(\text{ArrMap})(f)) \circ_{A\mathfrak{C}} \eta_C \Phi(\text{NTMap})(x) = f$

proof—

from *assms*(2) **have** $a : a \in_{\circ} \mathfrak{D}(\text{Obj})$ **by** *auto*

from *assms* **have** Φ -*f*:

```

( $\Phi$ (AdjNT)(NTMap)( $x, a$ ) $\bullet$ )(ArrVal)( $f$ ) :  $x \mapsto_{\mathcal{C}}$   $\mathfrak{G}$ (ObjMap)( $a$ )
by
(
  cs-concl
  cs-simp: cat-cs-simps cat-op-simps
  cs-intro:
    cat-arrow-cs-intros
    cat-cs-intros
    adj-cs-intros
    cat-prod-cs-intros
    cat-op-intros
)
from assms show ?thesis
unfolding
  cf-adj-AdjNT-cf-adjunction-unit[OF assms]
  cf-adj-AdjNT-cf-adjunction-counit[OF a  $\Phi$ -f]
by
(
  cs-concl
  cs-simp: cat-cs-simps cat-op-simps
  cs-intro:
    cat-arrow-cs-intros
    cat-cs-intros
    adj-cs-intros
    cat-prod-cs-intros
    cat-op-intros
)
qed

```

lemmas [*cat-cs-simps*] = *is-cf-adjunction.cf-adj-unit-counit-app*

13.6 Counit-unit equations

The following equations appear as part of the statement of Theorem 1 in Chapter IV-1 in [9]. These equations also appear in [2], where they are named *counit–unit equations*.

lemma (in *is-cf-adjunction*) *cf-adjunction-counit-unit*:

$$\begin{aligned}
& (\mathfrak{G} \circ_{CF-NTCF} \varepsilon_C \Phi) \cdot_{NTCF} (\eta_C \Phi \circ_{NTCF-CF} \mathfrak{G}) = \text{ntcf-id } \mathfrak{G} \\
& (\text{is } \langle (\mathfrak{G} \circ_{CF-NTCF} ?\varepsilon) \cdot_{NTCF} (? \eta \circ_{NTCF-CF} \mathfrak{G}) = \text{ntcf-id } \mathfrak{G} \rangle)
\end{aligned}$$

proof(*rule ntcf-eqI*)

from *is-cf-adjunction-axioms* **show**

$$\begin{aligned}
& (\mathfrak{G} \circ_{CF-NTCF} ?\varepsilon) \cdot_{NTCF} (? \eta \circ_{NTCF-CF} \mathfrak{G}) : \mathfrak{G} \mapsto_{CF} \mathfrak{G} : \mathfrak{D} \mapsto_{C\alpha} \mathfrak{C} \\
& \text{by } (cs-concl \text{ **cs-simp**: cat-cs-simps **cs-intro**: cat-cs-intros adj-cs-intros)
\end{aligned}$$

show *ntcf-id $\mathfrak{G} : \mathfrak{G} \mapsto_{CF} \mathfrak{G} : \mathfrak{D} \mapsto_{C\alpha} \mathfrak{C}$*

by (*rule is-functor.cf-ntcf-id-is-ntcf[*OF RL.is-functor-axioms*]*)

from *is-cf-adjunction-axioms* **have** *dom-lhs*:

$$\mathcal{D}_\circ (((\mathfrak{G} \circ_{CF-NTCF} ?\varepsilon) \cdot_{NTCF} (? \eta \circ_{NTCF-CF} \mathfrak{G}))(\text{NTMap})) = \mathfrak{D}(\text{Obj})$$

by

```

(
  cs-concl cs-shallow
  cs-simp: cat-cs-simps cs-intro: cat-cs-intros adj-cs-intros
)

```

from *is-cf-adjunction-axioms* **have** *dom-rhs*: $\mathcal{D}_\circ (\text{ntcf-id } \mathfrak{G}(\text{NTMap})) = \mathfrak{D}(\text{Obj})$

by (*cs-concl **cs-shallow **cs-simp**: cat-cs-simps **cs-intro**: adj-cs-intros***)

show $((\mathfrak{G} \circ_{CF-NTCF} ?\varepsilon) \cdot_{NTCF} (? \eta \circ_{NTCF-CF} \mathfrak{G}))(\text{NTMap}) = \text{ntcf-id } \mathfrak{G}(\text{NTMap})$

proof(*rule vsv-eqI, unfold dom-lhs dom-rhs*)

fix *a* **assume** *prems*: $a \in_\circ \mathfrak{D}(\text{Obj})$

let $? \varphi\text{-aa} = \langle \Phi(\text{AdjNT})(\text{NTMap})(\mathfrak{G}(\text{ObjMap})(a), a) \bullet \rangle$

```

have category  $\alpha$  (cat-Set  $\alpha$ )
  by (rule category-cat-Set)
from is-cf-adjunction-axioms prems
  L.category-axioms R.category-axioms
  L.category-op R.category-op
  LR.is-functor-axioms RL.is-functor-axioms
  category-cat-Set
have
   $?\varphi\text{-aa}(\downarrow \text{Arr Val})(\downarrow ?\varepsilon(\downarrow \text{NTMap})(\downarrow a)) =$ 
  ( $?\varphi\text{-aa} \circ_{A \text{ cat-Set } \alpha} ?\varphi\text{-aa}^{-1} C_{\text{cat-Set } \alpha}$ )( $\downarrow \text{Arr Val}$ )( $\downarrow \mathfrak{C}(\downarrow \text{CId})(\downarrow \mathfrak{G}(\downarrow \text{ObjMap})(\downarrow a))$ )
  by
  (
    cs-concl cs-shallow
    cs-simp:
    Z.cat-Set-Comp-Arr Val
    cat-Set-the-inverse[symmetric]
    cat-cs-simps adj-cs-simps cat-prod-cs-simps
    cs-intro:
    cat-arrow-cs-intros
    cat-cs-intros
    cat-op-intros
    adj-cs-intros
    cat-prod-cs-intros
  )
also from
  is-cf-adjunction-axioms prems
  L.category-axioms R.category-axioms
  L.category-op R.category-op
  LR.is-functor-axioms RL.is-functor-axioms
  category-cat-Set
have ... =  $\mathfrak{C}(\downarrow \text{CId})(\downarrow \mathfrak{G}(\downarrow \text{ObjMap})(\downarrow a))$ 
  by
  (
    cs-concl
    cs-simp:
    cat-cs-simps
    cat-Set-components(1)
    category.cat-the-inverse-Comp-CId
    cs-intro:
    cat-arrow-cs-intros
    cat-cs-intros
    cat-op-intros
    cat-prod-cs-intros
  )
finally have [cat-cs-simps]:
   $?\varphi\text{-aa}(\downarrow \text{Arr Val})(\downarrow ?\varepsilon(\downarrow \text{NTMap})(\downarrow a)) = \mathfrak{C}(\downarrow \text{CId})(\downarrow \mathfrak{G}(\downarrow \text{ObjMap})(\downarrow a))$ 
  by simp
from
  prems is-cf-adjunction-axioms
  L.category-axioms R.category-axioms
show (( $\mathfrak{G} \circ_{CF\text{-}NTCF} ?\varepsilon$ )  $\cdot_{NTCF}$  ( $? \eta \circ_{NTCF\text{-}CF} \mathfrak{G}$ ))( $\downarrow \text{NTMap}$ )( $\downarrow a$ ) = ntcf-id  $\mathfrak{G}(\downarrow \text{NTMap})(\downarrow a)$ 
  by
  (
    cs-concl
    cs-simp:
    cat-Set-the-inverse[symmetric]
    cf-adj-Comp-commute-RL
    cat-cs-simps
  )

```

```

    adj-cs-simps
    cat-prod-cs-simps
    cat-op-simps
cs-intro:
    cat-arrow-cs-intros
    cat-cs-intros
    adj-cs-intros
    cat-prod-cs-intros
    cat-op-intros
  )

qed (auto intro: cat-cs-intros)

qed simp-all

lemmas [adj-cs-simps] = is-cf-adjunction.cf-adjunction-counit-unit

lemma (in is-cf-adjunction) cf-adjunction-unit-counit:
  ( $\varepsilon_C \Phi \circ_{NTCF-CF} \mathfrak{F}$ )  $\cdot_{NTCF}$  ( $\mathfrak{F} \circ_{CF-NTCF} \eta_C \Phi$ ) = ntcf-id  $\mathfrak{F}$ 
  (is  $\langle$  ( $? \varepsilon \circ_{NTCF-CF} \mathfrak{F}$ )  $\cdot_{NTCF}$  ( $\mathfrak{F} \circ_{CF-NTCF} ? \eta$ ) = ntcf-id  $\mathfrak{F}$   $\rangle$ )
proof-
from is-cf-adjunction-axioms have  $\mathfrak{F} \eta$ :
   $\mathfrak{F} \circ_{CF-NTCF} ? \eta : \mathfrak{F} \mapsto_{CF} \mathfrak{F} \circ_{CF} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathcal{C} \mapsto_{C\alpha} \mathcal{D}$ 
by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros adj-cs-intros)
from is-cf-adjunction-axioms have  $\varepsilon \mathfrak{F}$ :
   $? \varepsilon \circ_{NTCF-CF} \mathfrak{F} : \mathfrak{F} \circ_{CF} \mathfrak{G} \circ_{CF} \mathfrak{F} \mapsto_{CF} \mathfrak{F} : \mathcal{C} \mapsto_{C\alpha} \mathcal{D}$ 
by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros adj-cs-intros)
from  $\mathfrak{F} \eta$   $\varepsilon \mathfrak{F}$  have  $\varepsilon \mathfrak{F} \mathfrak{F} \eta$ :
  ( $? \varepsilon \circ_{NTCF-CF} \mathfrak{F}$ )  $\cdot_{NTCF}$  ( $\mathfrak{F} \circ_{CF-NTCF} ? \eta$ ) :  $\mathfrak{F} \mapsto_{CF} \mathfrak{F} : \mathcal{C} \mapsto_{C\alpha} \mathcal{D}$ 
by (cs-concl cs-shallow cs-intro: cat-cs-intros)
from
  is-cf-adjunction.cf-adjunction-counit-unit[
    OF is-cf-adjunction-op,
    unfolded
    op-ntcf-cf-adjunction-unit[symmetric]
    op-ntcf-cf-adjunction-counit[symmetric]
    op-ntcf-cf-ntcf-comp[symmetric]
    op-ntcf-ntcf-cf-comp[symmetric]
    op-ntcf-ntcf-vcomp[symmetric]
    op-ntcf-ntcf-vcomp[symmetric, OF  $\varepsilon \mathfrak{F} \mathfrak{F} \eta$ ]
    LR.cf-ntcf-id-op-cf
  ]
have
  op-ntcf (op-ntcf (( $? \varepsilon \circ_{NTCF-CF} \mathfrak{F}$ )  $\cdot_{NTCF}$  ( $\mathfrak{F} \circ_{CF-NTCF} ? \eta$ ))) =
  op-ntcf (op-ntcf (ntcf-id  $\mathfrak{F}$ ))
by simp
from this is-cf-adjunction-axioms  $\varepsilon \mathfrak{F} \mathfrak{F} \eta$  show ?thesis
by
  (
    cs-prems cs-shallow
    cs-simp: cat-op-simps cs-intro: cat-cs-intros cat-prod-cs-intros
  )
qed

lemmas [adj-cs-simps] = is-cf-adjunction.cf-adjunction-unit-counit

```

13.7 Construction of an adjunction from universal morphisms from objects to functors

The subsection presents the construction of an adjunction given a structured collection of universal morphisms from objects to functors. The content of this subsection follows the statement and the proof of Theorem 2-i in Chapter IV-1 in [9].

13.7.1 The natural transformation associated with the adjunction constructed from universal morphisms from objects to functors

definition *cf-adjunction-AdjNT-of-unit* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$

where *cf-adjunction-AdjNT-of-unit* α \mathfrak{F} \mathfrak{G} η =

[
 $(\lambda cd \in_0. (op-cat (\mathfrak{F}(HomDom)) \times_C \mathfrak{F}(HomCod)) (Obj)).$
 $umap-of \mathfrak{G} (cd(0)) (\mathfrak{F}(ObjMap)(cd(0))) (\eta(NTMap)(cd(0))) (cd(1_N)),$
 $Hom_{O.C\alpha} \mathfrak{F}(HomCod)(\mathfrak{F}-, -),$
 $Hom_{O.C\alpha} \mathfrak{F}(HomDom)(-, \mathfrak{G}-),$
 $op-cat (\mathfrak{F}(HomDom)) \times_C (\mathfrak{F}(HomCod)),$
 $cat-Set \alpha$
]

Components.

lemma *cf-adjunction-AdjNT-of-unit-components*:

shows *cf-adjunction-AdjNT-of-unit* α \mathfrak{F} \mathfrak{G} $\eta(NTMap)$ =

(
 $\lambda cd \in_0. (op-cat (\mathfrak{F}(HomDom)) \times_C \mathfrak{F}(HomCod)) (Obj).$
 $umap-of \mathfrak{G} (cd(0)) (\mathfrak{F}(ObjMap)(cd(0))) (\eta(NTMap)(cd(0))) (cd(1_N))$
)

and *cf-adjunction-AdjNT-of-unit* α \mathfrak{F} \mathfrak{G} $\eta(NTDom)$ = $Hom_{O.C\alpha} \mathfrak{F}(HomCod)(\mathfrak{F}-, -)$

and *cf-adjunction-AdjNT-of-unit* α \mathfrak{F} \mathfrak{G} $\eta(NTCod)$ = $Hom_{O.C\alpha} \mathfrak{F}(HomDom)(-, \mathfrak{G}-)$

and *cf-adjunction-AdjNT-of-unit* α \mathfrak{F} \mathfrak{G} $\eta(NTDGDom)$ =

$op-cat (\mathfrak{F}(HomDom)) \times_C (\mathfrak{F}(HomCod))$

and *cf-adjunction-AdjNT-of-unit* α \mathfrak{F} \mathfrak{G} $\eta(NTDGCod)$ = $cat-Set \alpha$

unfolding *cf-adjunction-AdjNT-of-unit-def nt-field-simps*

by (*simp-all add: nat-omega-simps*)

13.7.2 Natural transformation map

lemma *cf-adjunction-AdjNT-of-unit-NTMap-vsuv[adj-cs-intros]*:

vsuv (*cf-adjunction-AdjNT-of-unit* α \mathfrak{F} \mathfrak{G} $\eta(NTMap)$)

unfolding *cf-adjunction-AdjNT-of-unit-components* **by** *simp*

lemma *cf-adjunction-AdjNT-of-unit-NTMap-vdomain[adj-cs-simps]*:

assumes $\mathfrak{F} : \mathcal{C} \mapsto \mathcal{C} \alpha \mathcal{D}$

shows \mathcal{D}_o (*cf-adjunction-AdjNT-of-unit* α \mathfrak{F} \mathfrak{G} $\eta(NTMap)$) = $(op-cat \mathcal{C} \times_C \mathcal{D})(Obj)$

proof-

interpret *is-functor* $\alpha \mathcal{C} \mathcal{D} \mathfrak{F}$ **by** (*rule assms(1)*)

show *?thesis*

unfolding *cf-adjunction-AdjNT-of-unit-components*

by (*simp add: cat-cs-simps*)

qed

lemma *cf-adjunction-AdjNT-of-unit-NTMap-app[adj-cs-simps]*:

assumes $\mathfrak{F} : \mathcal{C} \mapsto \mathcal{C} \alpha \mathcal{D}$ **and** $c \in_0 \mathcal{C}(Obj)$ **and** $d \in_0 \mathcal{D}(Obj)$

shows

cf-adjunction-AdjNT-of-unit α \mathfrak{F} \mathfrak{G} $\eta(NTMap)(c, d)_\bullet$ =
 $umap-of \mathfrak{G} c (\mathfrak{F}(ObjMap)(c)) (\eta(NTMap)(c)) d$

proof-

interpret \mathfrak{F} : *is-functor* $\alpha \mathfrak{C} \mathfrak{D} \mathfrak{F}$ **by** (*rule assms(1)*)
from *assms* **have** $[c, d]_o \in_o (op-cat \mathfrak{C} \times_C \mathfrak{D})(Obj)$
by
 (
 cs-concl cs-shallow
 cs-simp: *cat-op-simps cs-intro:* *cat-cs-intros cat-prod-cs-intros*
)
then show *cf-adjunction-AdjNT-of-unit* $\alpha \mathfrak{F} \mathfrak{G} \eta(NTMap) ([c, d])_\bullet =$
umap-of $\mathfrak{G} c (\mathfrak{F}(ObjMap)([c])) (\eta(NTMap)([c])) d$
unfolding *cf-adjunction-AdjNT-of-unit-components*
by (*simp add: nat-omega-simps cat-cs-simps*)
qed

lemma *cf-adjunction-AdjNT-of-unit-NTMap-vrange:*

assumes *category* $\alpha \mathfrak{C}$
and *category* $\alpha \mathfrak{D}$
and $\mathfrak{F} : \mathfrak{C} \mapsto_C \alpha \mathfrak{D}$
and $\mathfrak{G} : \mathfrak{D} \mapsto_C \alpha \mathfrak{C}$
and $\eta : cf-id \mathfrak{C} \mapsto_{CF} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{C} \mapsto_C \alpha \mathfrak{C}$
shows $\mathcal{R}_o (cf-adjunction-AdjNT-of-unit \alpha \mathfrak{F} \mathfrak{G} \eta(NTMap)) \subseteq_o cat-Set \alpha(Arr)$

proof-

interpret \mathfrak{F} : *is-functor* $\alpha \mathfrak{C} \mathfrak{D} \mathfrak{F}$ **by** (*rule assms(3)*)

show *?thesis*

proof

(
 rule vsv.vsv-vrange-vsubset,
 unfold cf-adjunction-AdjNT-of-unit-NTMap-vdomain[OF assms(3)]
)
show *vsv (cf-adjunction-AdjNT-of-unit* $\alpha \mathfrak{F} \mathfrak{G} \eta(NTMap))$
by (*intro adj-cs-intros*)
fix *cd assume prems:* $cd \in_o (op-cat \mathfrak{C} \times_C \mathfrak{D})(Obj)$
then obtain *c d where cd-def:* $cd = [c, d]_o$
and $c : c \in_o \mathfrak{C}(Obj)$
and $d : d \in_o \mathfrak{D}(Obj)$
by
 (
 auto
 simp: cat-op-simps
 elim:
 cat-prod-2-ObjE[OF \mathfrak{F}.HomDom.category-op \mathfrak{F}.HomCod.category-axioms]
)

from *assms c d show*

cf-adjunction-AdjNT-of-unit $\alpha \mathfrak{F} \mathfrak{G} \eta(NTMap)([cd]) \in_o cat-Set \alpha(Arr)$

unfolding *cd-def*

by

(
 cs-concl
 cs-simp: *cat-cs-simps adj-cs-simps cs-intro:* *cat-cs-intros*
)

qed

qed

13.7.3 Adjunction constructed from universal morphisms from objects to functors is an adjunction

lemma *cf-adjunction-AdjNT-of-unit-is-ntcf:*

assumes *category* $\alpha \mathfrak{C}$

and *category* $\alpha \mathcal{D}$
and $\mathfrak{F} : \mathcal{C} \mapsto \mapsto_{C\alpha} \mathcal{D}$
and $\mathfrak{G} : \mathcal{D} \mapsto \mapsto_{C\alpha} \mathcal{C}$
and $\eta : \text{cf-id } \mathcal{C} \mapsto_{CF} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathcal{C} \mapsto \mapsto_{C\alpha} \mathcal{C}$
shows *cf-adjunction-AdjNT-of-unit* $\alpha \mathfrak{F} \mathfrak{G} \eta$:
 $Hom_{O.C\alpha} \mathcal{D}(\mathfrak{F}-, -) \mapsto_{CF} Hom_{O.C\alpha} \mathcal{C}(-, \mathfrak{G}-)$:
 $op\text{-cat } \mathcal{C} \times_C \mathcal{D} \mapsto \mapsto_{C\alpha} \text{cat-Set } \alpha$
proof-

interpret \mathcal{C} : *category* $\alpha \mathcal{C}$ **by** (*rule* *assms(1)*)
interpret \mathcal{D} : *category* $\alpha \mathcal{D}$ **by** (*rule* *assms(2)*)
interpret \mathfrak{F} : *is-functor* $\alpha \mathcal{C} \mathcal{D} \mathfrak{F}$ **by** (*rule* *assms(3)*)
interpret \mathfrak{G} : *is-functor* $\alpha \mathcal{D} \mathcal{C} \mathfrak{G}$ **by** (*rule* *assms(4)*)
interpret η : *is-ntcf* $\alpha \mathcal{C} \mathcal{C} \langle \text{cf-id } \mathcal{C} \rangle \langle \mathfrak{G} \circ_{CF} \mathfrak{F} \rangle \eta$ **by** (*rule* *assms(5)*)

show *?thesis*
proof(*intro is-ntcfI'*)

show *vfsequence* (*cf-adjunction-AdjNT-of-unit* $\alpha \mathfrak{F} \mathfrak{G} \eta$)
unfolding *cf-adjunction-AdjNT-of-unit-def* **by** *simp*
show *vcard* (*cf-adjunction-AdjNT-of-unit* $\alpha \mathfrak{F} \mathfrak{G} \eta$) = $5_{\mathbb{N}}$
unfolding *cf-adjunction-AdjNT-of-unit-def* **by** (*simp add: nat-omega-simps*)
from *assms(2,3)* **show**
 $Hom_{O.C\alpha} \mathcal{D}(\mathfrak{F}-, -) : op\text{-cat } \mathcal{C} \times_C \mathcal{D} \mapsto \mapsto_{C\alpha} \text{cat-Set } \alpha$
by (*cs-concl cs-shallow cs-intro: cat-cs-intros*)
from *assms* **show** $Hom_{O.C\alpha} \mathcal{C}(-, \mathfrak{G}-) : op\text{-cat } \mathcal{C} \times_C \mathcal{D} \mapsto \mapsto_{C\alpha} \text{cat-Set } \alpha$
by (*cs-concl cs-shallow cs-intro: cat-cs-intros*)
show *vsv* (*cf-adjunction-AdjNT-of-unit* $\alpha \mathfrak{F} \mathfrak{G} \eta$ (*NTMap*))
by (*intro adj-cs-intros*)
from *assms* **show**
 \mathcal{D}_o (*cf-adjunction-AdjNT-of-unit* $\alpha \mathfrak{F} \mathfrak{G} \eta$ (*NTMap*)) = ($op\text{-cat } \mathcal{C} \times_C \mathcal{D}$)(*Obj*)
by (*cs-concl cs-shallow cs-simp: cat-cs-simps adj-cs-simps*)

show *cf-adjunction-AdjNT-of-unit* $\alpha \mathfrak{F} \mathfrak{G} \eta$ (*NTMap*)(*cd*) :
 $Hom_{O.C\alpha} \mathcal{D}(\mathfrak{F}-, -)(\text{ObjMap})(\text{cd}) \mapsto_{\text{cat-Set } \alpha}$
 $Hom_{O.C\alpha} \mathcal{C}(-, \mathfrak{G}-)(\text{ObjMap})(\text{cd})$
if $cd \in_o (op\text{-cat } \mathcal{C} \times_C \mathcal{D})(\text{Obj})$ **for** cd
proof-

from *that* **obtain** $c \ d$
where *cd-def*: $cd = [c, d]_o$ **and** $c : c \in_o \mathcal{C}(\text{Obj})$ **and** $d : d \in_o \mathcal{D}(\text{Obj})$
by
(

auto
simp: cat-op-simps
elim: cat-prod-2-ObjE[OF \mathcal{C}.category-op \mathcal{D}.category-axioms]
)

from *assms* $c \ d$ **show** *?thesis*
unfolding *cd-def*
by
(

cs-concl cs-shallow
cs-simp: *adj-cs-simps cat-cs-simps cat-op-simps*
cs-intro: *cat-cs-intros cat-op-intros cat-prod-cs-intros*
)

qed

show
cf-adjunction-AdjNT-of-unit $\alpha \mathfrak{F} \mathfrak{G} \eta$ (*NTMap*)($c'd'$) $\circ_A \text{cat-Set } \alpha$

$Hom_{O.C\alpha}\mathfrak{D}(\mathfrak{F}-,-)(\mathit{ArrMap})(\mathit{gf}) =$
 $Hom_{O.C\alpha}\mathfrak{C}(-,\mathfrak{G}-)(\mathit{ArrMap})(\mathit{gf}) \circ_{A\mathit{cat-Set}} \alpha$
 $cf\text{-adjunction-AdjNT-of-unit } \alpha \ \mathfrak{F} \ \mathfrak{G} \ \eta(\mathit{NTMap})(\mathit{cd})$
if $gf : cd \mapsto_{op\text{-cat}} \mathfrak{C} \times_C \mathfrak{D} \ c'd'$ **for** $cd \ c'd' \ gf$
proof-
from *that* **obtain** $g \ f \ c \ c' \ d \ d'$
where $gf\text{-def}: gf = [g, f]_{\circ}$
and $cd\text{-def}: cd = [c, d]_{\circ}$
and $c'd'\text{-def}: c'd' = [c', d']_{\circ}$
and $g: g : c' \mapsto_{\mathfrak{C}} c$
and $f: f : d \mapsto_{\mathfrak{D}} d'$
by
(
auto
simp: cat-op-simps
elim: cat-prod-2-is-arrE[OF \mathfrak{C}.category-op \mathfrak{D}.category-axioms]
)
from *assms* $g \ f$ **that** **show** *?thesis*
unfolding $gf\text{-def} \ cd\text{-def} \ c'd'\text{-def}$
by
(
cs-concl
cs-simp: *cf-umap-of-cf-hom-unit-commute adj-cs-simps cat-cs-simps*
cs-intro: *cat-cs-intros cat-op-intros cat-prod-cs-intros*
)
qed

qed (*auto simp: cf-adjunction-AdjNT-of-unit-components cat-cs-simps*)

qed

lemma *cf-adjunction-AdjNT-of-unit-is-ntcf*'[*adj-cs-intros*]:

assumes *category* $\alpha \ \mathfrak{C}$
and *category* $\alpha \ \mathfrak{D}$
and $\mathfrak{F} : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{D}$
and $\mathfrak{G} : \mathfrak{D} \mapsto_{C\alpha} \mathfrak{C}$
and $\eta : cf\text{-id } \mathfrak{C} \mapsto_{CF} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{S} = Hom_{O.C\alpha}\mathfrak{D}(\mathfrak{F}-,-)$
and $\mathfrak{S}' = Hom_{O.C\alpha}\mathfrak{C}(-,\mathfrak{G}-)$
and $\mathfrak{A} = op\text{-cat } \mathfrak{C} \times_C \mathfrak{D}$
and $\mathfrak{B} = cat\text{-Set } \alpha$
shows *cf-adjunction-AdjNT-of-unit* $\alpha \ \mathfrak{F} \ \mathfrak{G} \ \eta : \mathfrak{S} \mapsto_{CF} \mathfrak{S}' : \mathfrak{A} \mapsto_{C\alpha} \mathfrak{B}$
using *assms(1-5) unfolding assms(6-9)*
by (*rule cf-adjunction-AdjNT-of-unit-is-ntcf*)

13.7.4 Adjunction constructed from universal morphisms from objects to functors

definition *cf-adjunction-of-unit* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$

where *cf-adjunction-of-unit* $\alpha \ \mathfrak{F} \ \mathfrak{G} \ \eta =$
 $[\mathfrak{F}, \mathfrak{G}, cf\text{-adjunction-AdjNT-of-unit } \alpha \ \mathfrak{F} \ \mathfrak{G} \ \eta]_{\circ}$

Components.

lemma *cf-adjunction-of-unit-components*:

shows [*adj-cs-simps*]: *cf-adjunction-of-unit* $\alpha \ \mathfrak{F} \ \mathfrak{G} \ \eta(\mathit{AdjLeft}) = \mathfrak{F}$
and [*adj-cs-simps*]: *cf-adjunction-of-unit* $\alpha \ \mathfrak{F} \ \mathfrak{G} \ \eta(\mathit{AdjRight}) = \mathfrak{G}$
and *cf-adjunction-of-unit* $\alpha \ \mathfrak{F} \ \mathfrak{G} \ \eta(\mathit{AdjNT}) =$
 $cf\text{-adjunction-AdjNT-of-unit } \alpha \ \mathfrak{F} \ \mathfrak{G} \ \eta$
unfolding *cf-adjunction-of-unit-def adj-field-simps*

by (simp-all add: nat-omega-simps)

Natural transformation map.

lemma *cf-adjunction-of-unit-AdjNT-NTMap-vdomain[adj-cs-simps]*:
assumes $\mathfrak{F} : \mathcal{C} \mapsto \mathcal{C}_\alpha \mathcal{D}$
shows \mathcal{D}_\circ (cf-adjunction-of-unit $\alpha \mathfrak{F} \mathfrak{G} \eta$ (AdjNT) (NTMap)) =
 (op-cat $\mathcal{C} \times_{\mathcal{C}} \mathcal{D}$) (Obj)
using *assms*
unfolding *cf-adjunction-of-unit-components(3)*
by (rule *cf-adjunction-AdjNT-of-unit-NTMap-vdomain*)

lemma *cf-adjunction-of-unit-AdjNT-NTMap-app[adj-cs-simps]*:
assumes $\mathfrak{F} : \mathcal{C} \mapsto \mathcal{C}_\alpha \mathcal{D}$ **and** $c \in_\circ \mathcal{C}(\text{Obj})$ **and** $d \in_\circ \mathcal{D}(\text{Obj})$
shows
 cf-adjunction-of-unit $\alpha \mathfrak{F} \mathfrak{G} \eta$ (AdjNT) (NTMap) (c, d) \bullet =
 umap-of $\mathfrak{G} c$ ($\mathfrak{F}(\text{ObjMap})(c)$) ($\eta(\text{NTMap})(c)$) d
using *assms*
unfolding *cf-adjunction-of-unit-components(3)*
by (rule *cf-adjunction-AdjNT-of-unit-NTMap-app*)

The adjunction constructed from universal morphisms from objects to functors is an adjunction.

lemma *cf-adjunction-of-unit-is-cf-adjunction*:
assumes *category* $\alpha \mathcal{C}$
and *category* $\alpha \mathcal{D}$
and $\mathfrak{F} : \mathcal{C} \mapsto \mathcal{C}_\alpha \mathcal{D}$
and $\mathfrak{G} : \mathcal{D} \mapsto \mathcal{C}_\alpha \mathcal{C}$
and $\eta : \text{cf-id } \mathcal{C} \mapsto_{CF} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathcal{C} \mapsto \mathcal{C}_\alpha \mathcal{C}$
and $\bigwedge x. x \in_\circ \mathcal{C}(\text{Obj}) \implies \text{universal-arrow-of } \mathfrak{G} x$ ($\mathfrak{F}(\text{ObjMap})(x)$) ($\eta(\text{NTMap})(x)$)
shows *cf-adjunction-of-unit* $\alpha \mathfrak{F} \mathfrak{G} \eta : \mathfrak{F} \rightleftharpoons_{CF} \mathfrak{G} : \mathcal{C} \rightleftharpoons_{\mathcal{C}_\alpha} \mathcal{D}$
and $\eta_{\mathcal{C}}$ (*cf-adjunction-of-unit* $\alpha \mathfrak{F} \mathfrak{G} \eta$) = η
proof-

interpret \mathcal{C} : *category* $\alpha \mathcal{C}$ **by** (rule *assms(1)*)
interpret \mathcal{D} : *category* $\alpha \mathcal{D}$ **by** (rule *assms(2)*)
interpret \mathfrak{F} : *is-functor* $\alpha \mathcal{C} \mathcal{D} \mathfrak{F}$ **by** (rule *assms(3)*)
interpret \mathfrak{G} : *is-functor* $\alpha \mathcal{D} \mathcal{C} \mathfrak{G}$ **by** (rule *assms(4)*)
interpret η : *is-ntcf* $\alpha \mathcal{C} \mathcal{C} \langle \text{cf-id } \mathcal{C} \rangle \langle \mathfrak{G} \circ_{CF} \mathfrak{F} \rangle \eta$ **by** (rule *assms(5)*)

show *caou- η* : *cf-adjunction-of-unit* $\alpha \mathfrak{F} \mathfrak{G} \eta : \mathfrak{F} \rightleftharpoons_{CF} \mathfrak{G} : \mathcal{C} \rightleftharpoons_{\mathcal{C}_\alpha} \mathcal{D}$

proof

(
intro
is-cf-adjunctionI[*OF* - - *assms(1-4)*]
is-iso-ntcf-if-bnt-proj-snd-is-iso-ntcf[
OF \mathcal{C} .*category-op* \mathcal{D} .*category-axioms*
],
unfold cat-op-simps cf-adjunction-of-unit-components
)

show *caou- η* : *cf-adjunction-AdjNT-of-unit* $\alpha \mathfrak{F} \mathfrak{G} \eta$:
 $\text{Hom}_{\mathcal{O}. \mathcal{C}_\alpha \mathcal{D}}(\mathfrak{F}-, -) \mapsto_{CF} \text{Hom}_{\mathcal{O}. \mathcal{C}_\alpha \mathcal{C}}(-, \mathfrak{G}-)$:
 $\text{op-cat } \mathcal{C} \times_{\mathcal{C}} \mathcal{D} \mapsto \mathcal{C}_\alpha \text{ cat-Set } \alpha$
unfolding *cf-adjunction-of-unit-components*
by (rule *cf-adjunction-AdjNT-of-unit-is-ntcf*[*OF assms(1-5)*])

fix a **assume** *prems*: $a \in_\circ \mathcal{C}(\text{Obj})$

have *ua-of- η* :

ntcf-ua-of $\alpha \mathfrak{G} a$ ($\mathfrak{F}(\text{ObjMap})(a)$) ($\eta(\text{NTMap})(a)$) :
 $\text{Hom}_{\mathcal{O}. \mathcal{C}_\alpha \mathcal{D}}(\mathfrak{F}(\text{ObjMap})(a), -) \mapsto_{CF. \text{iso}} \text{Hom}_{\mathcal{O}. \mathcal{C}_\alpha \mathcal{C}}(a, -) \circ_{CF} \mathfrak{G} :$
 $\mathcal{D} \mapsto \mathcal{C}_\alpha \text{ cat-Set } \alpha$

by
 (

$$\text{rule is-functor.cf-ntcf-ua-of-is-iso-ntcf}[$$

$$OF \text{ assms}(4) \text{ assms}(6)[OF \text{ prems}]$$

$$]$$
)

have [adj-cs-simps]:

$$\text{cf-adjunction-AdjNT-of-unit } \alpha \mathfrak{F} \mathfrak{G} \eta_{op-cat} \mathfrak{C}, \mathfrak{D}(a, -)_{NTCF} =$$

$$\text{ntcf-ua-of } \alpha \mathfrak{G} a (\mathfrak{F}(\downarrow \text{ObjMap})(\downarrow a)) (\eta(\downarrow \text{NTMap})(\downarrow a))$$

proof(rule ntcf-eqI)
from assms(1-5) caou- η prems **show** lhs:

$$\text{cf-adjunction-AdjNT-of-unit } \alpha \mathfrak{F} \mathfrak{G} \eta_{op-cat} \mathfrak{C}, \mathfrak{D}(a, -)_{NTCF} :$$

$$Hom_{O.C\alpha} \mathfrak{D}(\mathfrak{F}(\downarrow \text{ObjMap})(\downarrow a), -) \mapsto_{CF} Hom_{O.C\alpha} \mathfrak{C}(a, -) \circ_{CF} \mathfrak{G} :$$

$$\mathfrak{D} \mapsto_{C\alpha} \text{cat-Set } \alpha$$

by
 (

$$\text{cs-concl cs-shallow}$$

$$\text{cs-simp: cat-cs-simps cat-op-simps}$$

$$\text{cs-intro: cat-cs-intros cat-op-intros}$$
)

from ua-of- ηa **show** rhs:

$$\text{ntcf-ua-of } \alpha \mathfrak{G} a (\mathfrak{F}(\downarrow \text{ObjMap})(\downarrow a)) (\eta(\downarrow \text{NTMap})(\downarrow a)) :$$

$$Hom_{O.C\alpha} \mathfrak{D}(\mathfrak{F}(\downarrow \text{ObjMap})(\downarrow a), -) \mapsto_{CF} Hom_{O.C\alpha} \mathfrak{C}(a, -) \circ_{CF} \mathfrak{G} :$$

$$\mathfrak{D} \mapsto_{C\alpha} \text{cat-Set } \alpha$$

by (cs-concl cs-shallow cs-intro: ntcf-cs-intros)

from lhs **have** dom-lhs:

$$\mathfrak{D}_o ((\text{cf-adjunction-AdjNT-of-unit } \alpha \mathfrak{F} \mathfrak{G} \eta_{op-cat} \mathfrak{C}, \mathfrak{D}(a, -)_{NTCF})(\downarrow \text{NTMap})) =$$

$$\mathfrak{D}(\downarrow \text{Obj})$$

by (cs-concl cs-shallow cs-simp: cat-cs-simps)

from lhs assms(4) **have** dom-rhs:

$$\mathfrak{D}_o (\text{ntcf-ua-of } \alpha \mathfrak{G} a (\mathfrak{F}(\downarrow \text{ObjMap})(\downarrow a)) (\eta(\downarrow \text{NTMap})(\downarrow a))(\downarrow \text{NTMap})) = \mathfrak{D}(\downarrow \text{Obj})$$

by (cs-concl cs-shallow cs-simp: cat-cs-simps)

show

$$(\text{cf-adjunction-AdjNT-of-unit } \alpha \mathfrak{F} \mathfrak{G} \eta_{op-cat} \mathfrak{C}, \mathfrak{D}(a, -)_{NTCF})(\downarrow \text{NTMap}) =$$

$$\text{ntcf-ua-of } \alpha \mathfrak{G} a (\mathfrak{F}(\downarrow \text{ObjMap})(\downarrow a)) (\eta(\downarrow \text{NTMap})(\downarrow a))(\downarrow \text{NTMap})$$

proof(rule vsu-eqI, unfold dom-lhs dom-rhs)
fix d **assume** prems': $d \in_o \mathfrak{D}(\downarrow \text{Obj})$
from assms(3,4) prems prems' **show**

$$(\text{cf-adjunction-AdjNT-of-unit } \alpha \mathfrak{F} \mathfrak{G} \eta_{op-cat} \mathfrak{C}, \mathfrak{D}(a, -)_{NTCF})(\downarrow \text{NTMap})(\downarrow d) =$$

$$\text{ntcf-ua-of } \alpha \mathfrak{G} a (\mathfrak{F}(\downarrow \text{ObjMap})(\downarrow a)) (\eta(\downarrow \text{NTMap})(\downarrow a))(\downarrow \text{NTMap})(\downarrow d)$$

by (cs-concl cs-shallow cs-simp: adj-cs-simps cat-cs-simps)

qed (simp-all add: bnt-proj-snd-NTMap-vsuv $\mathfrak{G}.$ ntcf-ua-of-NTMap-vsuv)

qed simp-all

from assms(1-5) assms(6)[OF prems] prems **show**

$$\text{cf-adjunction-AdjNT-of-unit } \alpha \mathfrak{F} \mathfrak{G} \eta_{op-cat} \mathfrak{C}, \mathfrak{D}(a, -)_{NTCF} :$$

$$Hom_{O.C\alpha} \mathfrak{D}(\mathfrak{F} -, -)_{op-cat} \mathfrak{C}, \mathfrak{D}(a, -)_{CF} \mapsto_{CF} iso$$

$$Hom_{O.C\alpha} \mathfrak{C}(-, \mathfrak{G} -)_{op-cat} \mathfrak{C}, \mathfrak{D}(a, -)_{CF} :$$

$$\mathfrak{D} \mapsto_{C\alpha} \text{cat-Set } \alpha$$

by
 (

$$\text{cs-concl cs-shallow}$$

$$\text{cs-simp: adj-cs-simps cat-cs-simps cs-intro: cat-cs-intros}$$
)

qed (auto simp: cf-adjunction-of-unit-def nat-omega-simps)

show $\eta_C (\text{cf-adjunction-of-unit } \alpha \mathfrak{F} \mathfrak{G} \eta) = \eta$

proof(rule ntcf-eqI)

```

from caou-η show lhs:
   $\eta_C$  (cf-adjunction-of-unit  $\alpha$   $\mathfrak{F}$   $\mathfrak{G}$   $\eta$ ) :
    cf-id  $\mathfrak{C} \mapsto_{CF} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{C}$ 
  by (cs-concl cs-shallow cs-intro: adj-cs-intros)
show rhs:  $\eta : \text{cf-id } \mathfrak{C} \mapsto_{CF} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{C}$ 
  by (auto intro: cat-cs-intros)
from lhs have dom-lhs:
   $\mathcal{D}_\circ (\eta_C (\text{cf-adjunction-of-unit } \alpha \mathfrak{F} \mathfrak{G} \eta)(NTMap)) = \mathfrak{C}(\text{Obj})$ 
  by (cs-concl cs-shallow cs-simp: cat-cs-simps)
have dom-rhs:  $\mathcal{D}_\circ (\eta(NTMap)) = \mathfrak{C}(\text{Obj})$  by (auto simp: cat-cs-simps)
show  $\eta_C (\text{cf-adjunction-of-unit } \alpha \mathfrak{F} \mathfrak{G} \eta)(NTMap) = \eta(NTMap)$ 
proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
  fix a assume prems:  $a \in_\circ \mathfrak{C}(\text{Obj})$ 
  from assms(1-5) prems caou-η show
     $\eta_C (\text{cf-adjunction-of-unit } \alpha \mathfrak{F} \mathfrak{G} \eta)(NTMap)(a) = \eta(NTMap)(a)$ 
  by
    (
      cs-concl cs-shallow
      cs-simp:
        adj-cs-simps cat-cs-simps cf-adjunction-of-unit-components(3)
      cs-intro: cat-cs-intros
    )
  qed (auto intro: adj-cs-intros)
qed simp-all

```

qed

13.8 Construction of an adjunction from a functor and universal morphisms from objects to functors

The subsection presents the construction of an adjunction given a functor and a structured collection of universal morphisms from objects to functors. The content of this subsection follows the statement and the proof of Theorem 2-ii in Chapter IV-1 in [9].

13.8.1 Left adjoint

definition *cf-la-of-ra* :: $(V \Rightarrow V) \Rightarrow V \Rightarrow V \Rightarrow V$

where *cf-la-of-ra* $F \mathfrak{G} \eta =$

```

[
  ( $\lambda x \in_\circ \mathfrak{G}(\text{HomCod})(\text{Obj}). F x$ ),
  (
     $\lambda h \in_\circ \mathfrak{G}(\text{HomCod})(\text{Arr}). \text{THE } f'$ .
     $f' : F (\mathfrak{G}(\text{HomCod})(\text{Dom})(h)) \mapsto_{\mathfrak{G}(\text{HomDom})} F (\mathfrak{G}(\text{HomCod})(\text{Cod})(h)) \wedge$ 
     $\eta(\mathfrak{G}(\text{HomCod})(\text{Cod})(h)) \circ_{A\mathfrak{G}(\text{HomCod})} h =$ 
    (
      umap-of
       $\mathfrak{G}$ 
      ( $\mathfrak{G}(\text{HomCod})(\text{Dom})(h)$ )
      ( $F (\mathfrak{G}(\text{HomCod})(\text{Dom})(h))$ )
      ( $\eta(\mathfrak{G}(\text{HomCod})(\text{Dom})(h))$ )
      ( $F (\mathfrak{G}(\text{HomCod})(\text{Cod})(h))$ )
    )
    ( $\text{ArrVal})(f')$ 
  )
),
 $\mathfrak{G}(\text{HomCod})$ ,
 $\mathfrak{G}(\text{HomDom})$ 
]

```

Components.

lemma *cf-la-of-ra-components*:

shows *cf-la-of-ra* $F \mathfrak{G} \eta(\text{ObjMap}) = (\lambda x \in_{\circ} \mathfrak{G}(\text{HomCod})(\text{Obj}). F x)$

and *cf-la-of-ra* $F \mathfrak{G} \eta(\text{ArrMap}) =$

(
 $\lambda h \in_{\circ} \mathfrak{G}(\text{HomCod})(\text{Arr}). \text{THE } f'.$
 $f' : F (\mathfrak{G}(\text{HomCod})(\text{Dom})(h)) \mapsto_{\mathfrak{G}(\text{HomDom})} F (\mathfrak{G}(\text{HomCod})(\text{Cod})(h)) \wedge$
 $\eta(\mathfrak{G}(\text{HomCod})(\text{Cod})(h)) \circ_A \mathfrak{G}(\text{HomCod}) h =$
 (
 umap-of
 \mathfrak{G}
 $(\mathfrak{G}(\text{HomCod})(\text{Dom})(h))$
 $(F (\mathfrak{G}(\text{HomCod})(\text{Dom})(h)))$
 $(\eta(\mathfrak{G}(\text{HomCod})(\text{Dom})(h)))$
 $(F (\mathfrak{G}(\text{HomCod})(\text{Cod})(h)))$
 $)(\text{ArrVal})(f')$
)

and *cf-la-of-ra* $F \mathfrak{G} \eta(\text{HomDom}) = \mathfrak{G}(\text{HomCod})$

and *cf-la-of-ra* $F \mathfrak{G} \eta(\text{HomCod}) = \mathfrak{G}(\text{HomDom})$

unfolding *cf-la-of-ra-def dghm-field-simps* **by** (*simp-all add: nat-omega-simps*)

13.8.2 Object map

mk-VLambda *cf-la-of-ra-components(1)*

$|vsu \text{ cf-la-of-ra-ObjMap-vsuv}[adj\text{-cs-intros}]|$

mk-VLambda (**in** *is-functor*)

*cf-la-of-ra-components(1)[where ? $\mathfrak{G}=\mathfrak{F}$, unfolded *cf-HomCod*]*

$|vdomain \text{ cf-la-of-ra-ObjMap-vdomain}[adj\text{-cs-simps}]|$

$|app \text{ cf-la-of-ra-ObjMap-app}[adj\text{-cs-simps}]|$

lemmas $[adj\text{-cs-simps}] =$

is-functor.cf-la-of-ra-ObjMap-vdomain

is-functor.cf-la-of-ra-ObjMap-app

13.8.3 Arrow map

mk-VLambda *cf-la-of-ra-components(2)*

$|vsu \text{ cf-la-of-ra-ArrMap-vsuv}[adj\text{-cs-intros}]|$

mk-VLambda (**in** *is-functor*)

*cf-la-of-ra-components(2)[where ? $\mathfrak{G}=\mathfrak{F}$, unfolded *cf-HomCod cf-HomDom*]*

$|vdomain \text{ cf-la-of-ra-ArrMap-vdomain}[adj\text{-cs-simps}]|$

$|app \text{ cf-la-of-ra-ArrMap-app}|$

lemmas $[adj\text{-cs-simps}] = \text{is-functor.cf-la-of-ra-ArrMap-vdomain}$

lemma (**in** *is-functor*) *cf-la-of-ra-ArrMap-app'*:

assumes $h : a \mapsto_{\mathfrak{B}} b$

shows

cf-la-of-ra $F \mathfrak{F} \eta(\text{ArrMap})(h) =$

(
 $\text{THE } f'.$
 $f' : F a \mapsto_{\mathfrak{A}} F b \wedge$
 $\eta(b) \circ_{A\mathfrak{B}} h = \text{umap-of } \mathfrak{F} a (F a) (\eta(a)) (F b)(\text{ArrVal})(f')$
)

proof-

from *assms* **have** $h : h \in_{\circ} \mathfrak{B}(\text{Arr})$ **by** (*simp add: cat-cs-intros*)

from *assms* **have** $h\text{-Dom}: \mathfrak{B}(\text{Dom})(h) = a$ **and** $h\text{-Cod}: \mathfrak{B}(\text{Cod})(h) = b$
by (*simp-all add: cat-cs-simps*)
show *?thesis* **by** (*rule cf-la-of-ra-ArrMap-app[OF h, unfolded h-Dom h-Cod]*)
qed

lemma *cf-la-of-ra-ArrMap-app-unique*:

assumes $\mathfrak{G} : \mathfrak{D} \mapsto_{C\alpha} \mathfrak{C}$
and $f : a \mapsto_{\mathfrak{C}} b$
and *universal-arrow-of* $\mathfrak{G} a$ (*cf-la-of-ra* $F \mathfrak{G} \eta(\text{ObjMap})(a)$) ($\eta(a)$)
and *universal-arrow-of* $\mathfrak{G} b$ (*cf-la-of-ra* $F \mathfrak{G} \eta(\text{ObjMap})(b)$) ($\eta(b)$)
shows *cf-la-of-ra* $F \mathfrak{G} \eta(\text{ArrMap})(f) : F a \mapsto_{\mathfrak{D}} F b$
and $\eta(b) \circ_{A\mathfrak{C}} f =$
umap-of $\mathfrak{G} a (F a) (\eta(a)) (F b)(\text{ArrVal})(\text{cf-la-of-ra } F \mathfrak{G} \eta(\text{ArrMap})(f))$
and $\wedge f'$.
 \llbracket
 $f' : F a \mapsto_{\mathfrak{D}} F b;$
 $\eta(b) \circ_{A\mathfrak{C}} f = \text{umap-of } \mathfrak{G} a (F a) (\eta(a)) (F b)(\text{ArrVal})(f')$
 $\rrbracket \implies \text{cf-la-of-ra } F \mathfrak{G} \eta(\text{ArrMap})(f) = f'$

proof-

interpret \mathfrak{G} : *is-functor* $\alpha \mathfrak{D} \mathfrak{C} \mathfrak{G}$ **by** (*rule assms(1)*)

from *assms(2)* **have** $a : a \in_{\circ} \mathfrak{C}(\text{Obj})$ **and** $b : b \in_{\circ} \mathfrak{C}(\text{Obj})$

by (*simp-all add: cat-cs-intros*)

note $ua\text{-}\eta\text{-}a = \mathfrak{G}.\text{universal-arrow-of}D[\text{OF } \text{assms}(3)]$

note $ua\text{-}\eta\text{-}b = \mathfrak{G}.\text{universal-arrow-of}D[\text{OF } \text{assms}(4)]$

from $ua\text{-}\eta\text{-}b(2)$ **have** [*cat-cs-intros*]:

$\llbracket c = b; c' = \mathfrak{G}(\text{ObjMap})(\text{cf-la-of-ra } F \mathfrak{G} \eta(\text{ObjMap})(b)) \rrbracket \implies \eta(b) : c \mapsto_{\mathfrak{C}} c'$

for $c c'$

by *auto*

from *assms(1,2)* $ua\text{-}\eta\text{-}a(2)$ **have** $\eta a\text{-}f$:

$\eta(b) \circ_{A\mathfrak{C}} f : a \mapsto_{\mathfrak{C}} \mathfrak{G}(\text{ObjMap})(\text{cf-la-of-ra } F \mathfrak{G} \eta(\text{ObjMap})(b))$

by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

from *assms(1,2)* **have** $lara\text{-}a : \text{cf-la-of-ra } F \mathfrak{G} \eta(\text{ObjMap})(a) = F a$

and $lara\text{-}b : \text{cf-la-of-ra } F \mathfrak{G} \eta(\text{ObjMap})(b) = F b$

by (*cs-concl cs-simp: adj-cs-simps cs-intro: cat-cs-intros*)+

from *theD*

\llbracket
 OF
 $ua\text{-}\eta\text{-}a(3)[\text{OF } ua\text{-}\eta\text{-}b(1) \eta a\text{-}f, \text{ unfolded } lara\text{-}a \text{ lara}\text{-}b]$
 $\mathfrak{G}.\text{cf-la-of-ra-ArrMap-app}'[\text{OF } \text{assms}(2), \text{ of } F \eta]$
 \rrbracket

show *cf-la-of-ra* $F \mathfrak{G} \eta(\text{ArrMap})(f) : F a \mapsto_{\mathfrak{D}} F b$

and $\eta(b) \circ_{A\mathfrak{C}} f = \text{umap-of}$

$\mathfrak{G} a (F a) (\eta(a)) (F b)(\text{ArrVal})(\text{cf-la-of-ra } F \mathfrak{G} \eta(\text{ArrMap})(f))$

and $\wedge f'$.

\llbracket
 $f' : F a \mapsto_{\mathfrak{D}} F b;$
 $\eta(b) \circ_{A\mathfrak{C}} f = \text{umap-of } \mathfrak{G} a (F a) (\eta(a)) (F b)(\text{ArrVal})(f')$
 $\rrbracket \implies \text{cf-la-of-ra } F \mathfrak{G} \eta(\text{ArrMap})(f) = f'$

by *blast+*

qed

lemma *cf-la-of-ra-ArrMap-app-is-arr[adj-cs-intros]*:

assumes $\mathfrak{G} : \mathfrak{D} \mapsto_{C\alpha} \mathfrak{C}$

and $f : a \mapsto_{\mathfrak{C}} b$

and *universal-arrow-of* $\mathfrak{G} a$ (*cf-la-of-ra* $F \mathfrak{G} \eta(\text{ObjMap})(\downarrow a)$) ($\eta(\downarrow a)$)
and *universal-arrow-of* $\mathfrak{G} b$ (*cf-la-of-ra* $F \mathfrak{G} \eta(\text{ObjMap})(\downarrow b)$) ($\eta(\downarrow b)$)
and $F a = F a$
and $F b = F b$
shows *cf-la-of-ra* $F \mathfrak{G} \eta(\text{ArrMap})(\downarrow f) : Fa \mapsto_{\mathfrak{D}} Fb$
using *assms(1-4)* **unfolding** *assms(5,6)* **by** (*rule cf-la-of-ra-ArrMap-app-unique*)

13.8.4 An adjunction constructed from a functor and universal morphisms from objects to functors is an adjunction

lemma *cf-la-of-ra-is-functor*:

assumes $\mathfrak{G} : \mathfrak{D} \mapsto_{C\alpha} \mathfrak{C}$
and $\wedge c. c \in_{\circ} \mathfrak{C}(\text{Obj}) \implies F c \in_{\circ} \mathfrak{D}(\text{Obj})$
and $\wedge c. c \in_{\circ} \mathfrak{C}(\text{Obj}) \implies$
universal-arrow-of $\mathfrak{G} c$ (*cf-la-of-ra* $F \mathfrak{G} \eta(\text{ObjMap})(\downarrow c)$) ($\eta(\downarrow c)$)
and $\wedge c c' h. h : c \mapsto_{\mathfrak{C}} c' \implies$
 $\mathfrak{G}(\text{ArrMap})(\downarrow \text{cf-la-of-ra } F \mathfrak{G} \eta(\text{ArrMap})(\downarrow h)) \circ_{A\mathfrak{C}} \eta(\downarrow c) = \eta(\downarrow c') \circ_{A\mathfrak{C}} h$
shows *cf-la-of-ra* $F \mathfrak{G} \eta : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{D}$ (**is** $\langle ?\mathfrak{F} : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{D} \rangle$)

proof-

interpret \mathfrak{G} : *is-functor* $\alpha \mathfrak{D} \mathfrak{C} \mathfrak{G}$ **by** (*rule assms(1)*)

show *cf-la-of-ra* $F \mathfrak{G} \eta : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{D}$

proof(*rule is-functorI'*)

show *vfsequence* $?\mathfrak{F}$ **unfolding** *cf-la-of-ra-def* **by** *auto*

show *vcard* $?\mathfrak{F} = 4_{\mathbb{N}}$

unfolding *cf-la-of-ra-def* **by** (*simp add: nat-omega-simps*)

show $\mathcal{R}_{\circ} (?\mathfrak{F}(\text{ObjMap})) \subseteq_{\circ} \mathfrak{D}(\text{Obj})$

proof(*rule vsv.vsv-vrange-vsubset, unfold \mathfrak{G}.cf-la-of-ra-ObjMap-vdomain*)

fix x **assume** $x \in_{\circ} \mathfrak{C}(\text{Obj})$

with *assms(1)* **show** $?\mathfrak{F}(\text{ObjMap})(\downarrow x) \in_{\circ} \mathfrak{D}(\text{Obj})$

by (*cs-concl cs-shallow cs-simp: adj-cs-simps cs-intro: assms(2)*)

qed (*auto intro: adj-cs-intros*)

show $?\mathfrak{F}(\text{ArrMap})(\downarrow f) : ?\mathfrak{F}(\text{ObjMap})(\downarrow a) \mapsto_{\mathfrak{D}} ?\mathfrak{F}(\text{ObjMap})(\downarrow b)$

if $f : a \mapsto_{\mathfrak{C}} b$ **for** $a b f$

proof-

from *that* **have** $a : a \in_{\circ} \mathfrak{C}(\text{Obj})$ **and** $b : b \in_{\circ} \mathfrak{C}(\text{Obj})$

by (*simp-all add: cat-cs-intros*)

have *ua-η-a*: *universal-arrow-of* $\mathfrak{G} a$ ($?\mathfrak{F}(\text{ObjMap})(\downarrow a)$) ($\eta(\downarrow a)$)

and *ua-η-b*: *universal-arrow-of* $\mathfrak{G} b$ ($?\mathfrak{F}(\text{ObjMap})(\downarrow b)$) ($\eta(\downarrow b)$)

by (*intro assms(3)[OF a] assms(3)[OF b]*)+

from $a b$ *cf-la-of-ra-ArrMap-app-unique(1)[OF assms(1) that ua-η-a ua-η-b]*

show *thesis*

by (*cs-concl cs-shallow cs-simp: adj-cs-simps*)

qed

show $?\mathfrak{F}(\text{ArrMap})(\downarrow g \circ_{A\mathfrak{C}} f) = ?\mathfrak{F}(\text{ArrMap})(\downarrow g) \circ_{A\mathfrak{D}} ?\mathfrak{F}(\text{ArrMap})(\downarrow f)$

if $g : b \mapsto_{\mathfrak{C}} c$ **and** $f : a \mapsto_{\mathfrak{C}} b$ **for** $b c g a f$

proof-

from *that* **have** $a : a \in_{\circ} \mathfrak{C}(\text{Obj})$ **and** $b : b \in_{\circ} \mathfrak{C}(\text{Obj})$ **and** $c : c \in_{\circ} \mathfrak{C}(\text{Obj})$

by (*simp-all add: cat-cs-intros*)

from *assms(1)* **that** **have** $gf : g \circ_{A\mathfrak{C}} f : a \mapsto_{\mathfrak{C}} c$

by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

note *ua-η-a* = *assms(3)[OF a]*

and $ua-\eta-b = \text{assms}(3)[OF\ b]$
and $ua-\eta-c = \text{assms}(3)[OF\ c]$

note $\text{lara-f} =$
 $cf\text{-la-of-ra-}\text{ArrMap-app-unique}[OF\ \text{assms}(1)\ \text{that}(2)\ ua-\eta-a\ ua-\eta-b]$

note $\text{lara-g} =$
 $cf\text{-la-of-ra-}\text{ArrMap-app-unique}[OF\ \text{assms}(1)\ \text{that}(1)\ ua-\eta-b\ ua-\eta-c]$

note $\text{lara-gf} =$
 $cf\text{-la-of-ra-}\text{ArrMap-app-unique}[OF\ \text{assms}(1)\ \text{gf}\ ua-\eta-a\ ua-\eta-c]$

note $ua-\eta-a = \mathfrak{G}.\text{universal-arrow-ofD}[OF\ ua-\eta-a]$
and $ua-\eta-b = \mathfrak{G}.\text{universal-arrow-ofD}[OF\ ua-\eta-b]$
and $ua-\eta-c = \mathfrak{G}.\text{universal-arrow-ofD}[OF\ ua-\eta-c]$

from $ua-\eta-a(2)\ \text{assms}(1)\ \text{that have } \eta a:$

$\eta(|a|) : a \mapsto_{\mathfrak{C}} \mathfrak{G}(|\text{ObjMap}|)(F\ a)$

by (*cs-prems* **cs-simp**: *adj-cs-simps* **cs-intro**: *cat-cs-intros*)

from $ua-\eta-b(2)\ \text{assms}(1)\ \text{that have } \eta b:$

$\eta(|b|) : b \mapsto_{\mathfrak{C}} \mathfrak{G}(|\text{ObjMap}|)(F\ b)$

by (*cs-prems* **cs-shallow** **cs-simp**: *adj-cs-simps* **cs-intro**: *cat-cs-intros*)

from $ua-\eta-c(2)\ \text{assms}(1)\ \text{that have } \eta c:$

$\eta(|c|) : c \mapsto_{\mathfrak{C}} \mathfrak{G}(|\text{ObjMap}|)(F\ c)$

by (*cs-prems* **cs-shallow** **cs-simp**: *adj-cs-simps* **cs-intro**: *cat-cs-intros*)

from $\text{assms}(1)\ \text{that } \eta c\ \text{have}$

$\eta(|c|) \circ_{A\mathfrak{C}} (g \circ_{A\mathfrak{C}} f) = (\eta(|c|) \circ_{A\mathfrak{C}} g) \circ_{A\mathfrak{C}} f$

by (*cs-concl* **cs-shallow** **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)

also from $\text{assms}(1)\ \text{lara-g}(1)\ \text{that}(2)\ \eta b\ \text{have}$

$\dots = \mathfrak{G}(|\text{ArrMap}|)(\mathfrak{F}(|\text{ArrMap}|)(g)) \circ_{A\mathfrak{C}} (\eta(|b|) \circ_{A\mathfrak{C}} f)$

by

(
cs-concl
cs-simp: *lara-g(2) cat-cs-simps*
cs-intro: *cat-cs-intros cat-prod-cs-intros*
)

also from $\text{assms}(1)\ \text{lara-f}(1)\ \eta a\ \text{have } \dots =$

$\mathfrak{G}(|\text{ArrMap}|)(\mathfrak{F}(|\text{ArrMap}|)(g)) \circ_{A\mathfrak{C}}$

$(\mathfrak{G}(|\text{ArrMap}|)(\mathfrak{F}(|\text{ArrMap}|)(f)) \circ_{A\mathfrak{C}} \eta(|a|))$

by (*cs-concl* **cs-shallow** **cs-simp**: *lara-f(2) cat-cs-simps*)

finally have [*symmetric, cat-cs-simps*]:

$\eta(|c|) \circ_{A\mathfrak{C}} (g \circ_{A\mathfrak{C}} f) = \dots$

from $\text{assms}(1)\ \text{this } \eta a\ \eta b\ \eta c\ \text{lara-g}(1)\ \text{lara-f}(1)\ \text{have}$

$\eta(|c|) \circ_{A\mathfrak{C}} (g \circ_{A\mathfrak{C}} f) =$

$\text{umap-of } \mathfrak{G}\ a\ (F\ a)\ (\eta(|a|))\ (F\ c)\ (|\text{ArrVal}|)(\mathfrak{F}(|\text{ArrMap}|)(g)) \circ_{A\mathfrak{D}}$

$\mathfrak{F}(|\text{ArrMap}|)(f))$

by

(
cs-concl **cs-shallow**
cs-simp: *adj-cs-simps cat-cs-simps*
cs-intro: *adj-cs-intros cat-cs-intros*
)

moreover from $\text{assms}(1)\ \text{lara-g}(1)\ \text{lara-f}(1)\ \text{have}$

$\mathfrak{F}(|\text{ArrMap}|)(g) \circ_{A\mathfrak{D}} \mathfrak{F}(|\text{ArrMap}|)(f) : F\ a \mapsto_{\mathfrak{D}} F\ c$

by (*cs-concl* **cs-shallow** **cs-intro**: *adj-cs-intros cat-cs-intros*)

ultimately show *?thesis* **by** (*intro lara-gf(3)*)

qed

show $?\mathfrak{F}(\text{ArrMap})(\mathfrak{C}(\text{CId})(c)) = \mathfrak{D}(\text{CId})(?\mathfrak{F}(\text{ObjMap})(c))$ if $c \in \mathfrak{C}(\text{Obj})$ for c

proof-

note $\text{lara-c} = \text{cf-la-of-ra-ArrMap-app-unique}[$

OF

$\text{assms}(1)$

$\mathfrak{G}.\text{HomCod.cat-CId-is-arr}[OF \text{ that}]$

$\text{assms}(3)[OF \text{ that}]$

$\text{assms}(3)[OF \text{ that}]$

$]$

from $\text{assms}(1)$ that have $\mathfrak{D}c: \mathfrak{D}(\text{CId})(F c) : F c \mapsto_{\mathfrak{D}} F c$

by (cs-concl **cs-simp**: cat-cs-simps **cs-intro**: $\text{assms}(2)$ cat-cs-intros)

from $\mathfrak{G}.\text{universal-arrow-ofD}(2)[OF \text{ assms}(3)[OF \text{ that}]]$ $\text{assms}(1)$ that have $\eta c:$

$\eta(c) : c \mapsto_{\mathfrak{C}} \mathfrak{G}(\text{ObjMap})(F c)$

by (cs-prems **cs-shallow** **cs-simp**: adj-cs-simps **cs-intro**: cat-cs-intros)

from $\text{assms}(1)$ that ηc have

$\eta(c) \circ_{A\mathfrak{C}} \mathfrak{C}(\text{CId})(c) =$

$\text{umap-of } \mathfrak{G} c (F c) (\eta(c)) (F c)(\text{ArrVal})(\mathfrak{D}(\text{CId})(F c))$

by (cs-concl **cs-simp**: cat-cs-simps **cs-intro**: $\text{assms}(2)$ cat-cs-intros)

note $[\text{cat-cs-simps}] = \text{lara-c}(3)[OF \mathfrak{D}c \text{ this}]$

from $\text{assms}(1)$ that $\mathfrak{D}c$ show $?thesis$

by

(

cs-concl **cs-shallow**

cs-simp: adj-cs-simps cat-cs-simps **cs-intro**: cat-cs-intros

)

qed

qed ($\text{auto simp: cf-la-of-ra-components cat-cs-intros cat-cs-simps}$)

qed

lemma $\text{cf-la-of-ra-is-ntcf}$:

fixes $F \mathfrak{C} \mathfrak{F} \mathfrak{G} \eta_m \eta$

defines $\mathfrak{F} \equiv \text{cf-la-of-ra } F \mathfrak{G} \eta_m$

and $\eta \equiv [\eta_m, \text{cf-id } \mathfrak{C}, \mathfrak{G} \circ_{CF} \mathfrak{F}, \mathfrak{C}, \mathfrak{C}]_o$

assumes $\mathfrak{G} : \mathfrak{D} \mapsto_{C\alpha} \mathfrak{C}$

and $\wedge c. c \in \mathfrak{C}(\text{Obj}) \implies F c \in \mathfrak{D}(\text{Obj})$

and $\wedge c. c \in \mathfrak{C}(\text{Obj}) \implies \text{universal-arrow-of } \mathfrak{G} c (\mathfrak{F}(\text{ObjMap})(c)) (\eta(\text{NTMap})(c))$

and $\wedge c c' h. h : c \mapsto_{\mathfrak{C}} c' \implies$

$\mathfrak{G}(\text{ArrMap})(\mathfrak{F}(\text{ArrMap})(h)) \circ_{A\mathfrak{C}} (\eta(\text{NTMap})(c)) = (\eta(\text{NTMap})(c')) \circ_{A\mathfrak{C}} h$

and $\text{vsv } (\eta(\text{NTMap}))$

and $\mathfrak{D}_o (\eta(\text{NTMap})) = \mathfrak{C}(\text{Obj})$

shows $\eta : \text{cf-id } \mathfrak{C} \mapsto_{CF} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{C}$

proof-

interpret \mathfrak{G} : $\text{is-functor } \alpha \mathfrak{D} \mathfrak{C} \mathfrak{G}$ by ($\text{rule assms}(3)$)

have η -components:

$\eta(\text{NTMap}) = \eta_m$

$\eta(\text{NTDom}) = \text{cf-id } \mathfrak{C}$

$\eta(\text{NTCod}) = \mathfrak{G} \circ_{CF} \mathfrak{F}$

$\eta(\text{NTDGDom}) = \mathfrak{C}$

$\eta(\text{NTDGCod}) = \mathfrak{C}$

unfolding η -def nt-field-simps by ($\text{simp-all add: nat-omega-simps}$)

note $\mathfrak{F}\text{-def} = \mathfrak{F}\text{-def}[\text{folded } \eta\text{-components}(1)]$

have $\mathfrak{F}: \mathfrak{F} : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{D}$

unfolding $\mathfrak{F}\text{-def}$

by ($\text{auto intro: cf-la-of-ra-is-functor}[OF \text{ assms}(3-6)[\text{unfolded } \mathfrak{F}\text{-def}]]$)

show $\eta : \text{cf-id } \mathfrak{C} \mapsto_{CF} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{C}$

proof(rule is-ntcfI)

show $\text{vsequence } \eta$ unfolding η -def by simp

show $vcard \eta = 5_N$ **unfolding** η -def **by** (*simp-all add: nat-omega-simps*)
from *assms(2)* **show** $cf-id \mathfrak{C} : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{C}$
by (*cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)
from *assms(2)* $\mathfrak{F} \circ_{CF} \mathfrak{F} : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{C}$
by (*cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)
show $\eta(\mathcal{N}TMap)(\downarrow a) : cf-id \mathfrak{C}(\downarrow ObjMap)(\downarrow a) \mapsto_{\mathfrak{C}} (\mathfrak{G} \circ_{CF} \mathfrak{F})(\downarrow ObjMap)(\downarrow a)$
if $a \in_{\circ} \mathfrak{C}(\downarrow Obj)$ **for** a
using *assms(2)* \mathfrak{F} **that** $\mathfrak{G}.universal-arrow-ofD(2)[OF \text{ assms}(5)[OF \text{ that}]]$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)
show
 $\eta(\mathcal{N}TMap)(\downarrow b) \circ_{A\mathfrak{C}} cf-id \mathfrak{C}(\downarrow ArrMap)(\downarrow f) =$
 $(\mathfrak{G} \circ_{CF} \mathfrak{F})(\downarrow ArrMap)(\downarrow f) \circ_{A\mathfrak{C}} \eta(\mathcal{N}TMap)(\downarrow a)$
if $f : a \mapsto_{\mathfrak{C}} b$ **for** $a \ b \ f$
using *assms(3)* \mathfrak{F} **that**
by
(
cs-concl cs-shallow
cs-simp: assms(6) cat-cs-simps cs-intro: cat-cs-intros
)
qed (*auto simp: \eta-components(2-5) assms(7-8)*)
qed

lemma *cf-la-of-ra-is-unit*:

fixes $F \ \mathfrak{C} \ \mathfrak{F} \ \mathfrak{G} \ \eta_m \ \eta$

defines $\mathfrak{F} \equiv cf-la-of-ra \ F \ \mathfrak{G} \ \eta_m$

and $\eta \equiv [\eta_m, cf-id \ \mathfrak{C}, \ \mathfrak{G} \circ_{CF} \ \mathfrak{F}, \ \mathfrak{C}, \ \mathfrak{C}]_{\circ}$

assumes *category* $\alpha \ \mathfrak{C}$

and *category* $\alpha \ \mathfrak{D}$

and $\mathfrak{G} : \mathfrak{D} \mapsto_{C\alpha} \mathfrak{C}$

and $\wedge c. c \in_{\circ} \mathfrak{C}(\downarrow Obj) \implies F \ c \in_{\circ} \mathfrak{D}(\downarrow Obj)$

and $\wedge c. c \in_{\circ} \mathfrak{C}(\downarrow Obj) \implies$

universal-arrow-of $\mathfrak{G} \ c \ (\mathfrak{F}(\downarrow ObjMap)(\downarrow c)) \ (\eta(\mathcal{N}TMap)(\downarrow c))$

and $\wedge c \ c' \ h. h : c \mapsto_{\mathfrak{C}} c' \implies$

$\mathfrak{G}(\downarrow ArrMap)(\downarrow \mathfrak{F}(\downarrow ArrMap)(\downarrow h)) \circ_{A\mathfrak{C}} (\eta(\mathcal{N}TMap)(\downarrow c)) = (\eta(\mathcal{N}TMap)(\downarrow c')) \circ_{A\mathfrak{C}} h$

and *vsv* $(\eta(\mathcal{N}TMap))$

and $\mathfrak{D}_{\circ} (\eta(\mathcal{N}TMap)) = \mathfrak{C}(\downarrow Obj)$

shows *cf-adjunction-of-unit* $\alpha \ \mathfrak{F} \ \mathfrak{G} \ \eta : \mathfrak{F} \rightleftharpoons_{CF} \mathfrak{G} : \mathfrak{C} \rightleftharpoons_{C\alpha} \mathfrak{D}$

and $\eta_C \ (cf-adjunction-of-unit \ \alpha \ \mathfrak{F} \ \mathfrak{G} \ \eta) = \eta$

proof-

have *\eta-components*:

$\eta(\mathcal{N}TMap) = \eta_m$

$\eta(\mathcal{N}TDom) = cf-id \ \mathfrak{C}$

$\eta(\mathcal{N}TCod) = \mathfrak{G} \circ_{CF} \ \mathfrak{F}$

$\eta(\mathcal{N}TDGDom) = \mathfrak{C}$

$\eta(\mathcal{N}TDGCod) = \mathfrak{C}$

unfolding η -def *nt-field-simps* **by** (*simp-all add: nat-omega-simps*)

note \mathfrak{F} -def = \mathfrak{F} -def[*folded \eta-components(1)*]

note $\mathfrak{F} = cf-la-of-ra-is-functor$

[

where $F=F$ **and** $\mathfrak{C}=\mathfrak{C}$ **and** $\mathfrak{G}=\mathfrak{G}$ **and** $\eta=\eta_m$,

folded \mathfrak{F} -def[*unfolded \eta-components(1)*],

folded η -components(1),

OF assms(5-8),

simplified

]

note $\eta = cf-la-of-ra-is-ntcf$

[

where $F=F$ **and** $\mathfrak{C}=\mathfrak{C}$ **and** $\mathfrak{G}=\mathfrak{G}$ **and** $\eta_m=\eta_m$,

```

    folded  $\mathfrak{F}$ -def[unfolded  $\eta$ -components(1)],
    folded  $\eta$ -def,
    OF assms(5-10),
    simplified
  ]
show cf-adjunction-of-unit  $\alpha \mathfrak{F} \mathfrak{G} \eta : \mathfrak{F} \Rightarrow_{CF} \mathfrak{G} : \mathfrak{C} \Rightarrow_{C\alpha} \mathfrak{D}$ 
and  $\eta_C$  (cf-adjunction-of-unit  $\alpha \mathfrak{F} \mathfrak{G} \eta$ ) =  $\eta$ 
by
  (
    intro cf-adjunction-of-unit-is-cf-adjunction[
      OF assms(3,4)  $\mathfrak{F}$  assms(5)  $\eta$  assms(7), simplified, folded  $\mathfrak{F}$ -def
    ]
  )+
qed

```

13.9 Construction of an adjunction from universal morphisms from functors to objects

13.9.1 Definition and elementary properties

The subsection presents the construction of an adjunction given a structured collection of universal morphisms from functors to objects. The content of this subsection follows the statement and the proof of Theorem 2-iii in Chapter IV-1 in [9].

definition *cf-adjunction-of-counit* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
where *cf-adjunction-of-counit* $\alpha \mathfrak{F} \mathfrak{G} \varepsilon =$
 $op\text{-}cf\text{-}adj$ (*cf-adjunction-of-unit* α (*op-cf* \mathfrak{G}) (*op-cf* \mathfrak{F}) (*op-ntcf* ε))

Components.

lemma *cf-adjunction-of-counit-components*:
shows *cf-adjunction-of-counit* $\alpha \mathfrak{F} \mathfrak{G} \varepsilon$ (*AdjLeft*) = *op-cf* (*op-cf* \mathfrak{F})
and *cf-adjunction-of-counit* $\alpha \mathfrak{F} \mathfrak{G} \varepsilon$ (*AdjRight*) = *op-cf* (*op-cf* \mathfrak{G})
and *cf-adjunction-of-counit* $\alpha \mathfrak{F} \mathfrak{G} \varepsilon$ (*AdjNT*) = *op-cf-adj-nt*
 (*op-cf* \mathfrak{G} (*HomDom*))
 (*op-cf* \mathfrak{G} (*HomCod*))
 (*cf-adjunction-AdjNT-of-unit* α (*op-cf* \mathfrak{G}) (*op-cf* \mathfrak{F}) (*op-ntcf* ε))
unfolding
cf-adjunction-of-counit-def
op-cf-adj-components
cf-adjunction-of-unit-components
by (*simp-all add: cat-op-simps*)

13.9.2 Natural transformation map

lemma *cf-adjunction-of-counit-NTMap-vsν*:
 $vsν$ (*cf-adjunction-of-counit* $\alpha \mathfrak{F} \mathfrak{G} \varepsilon$ (*AdjNT*))(*NTMap*)
unfolding *cf-adjunction-of-counit-components* **by** (*rule inv-ntcf-NTMap-vsν*)

13.9.3 An adjunction constructed from universal morphisms from functors to objects is an adjunction

lemma *cf-adjunction-of-counit-is-cf-adjunction*:
assumes *category* $\alpha \mathfrak{C}$
and *category* $\alpha \mathfrak{D}$
and $\mathfrak{F} : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{D}$
and $\mathfrak{G} : \mathfrak{D} \mapsto_{C\alpha} \mathfrak{C}$
and $\varepsilon : \mathfrak{F} \circ_{CF} \mathfrak{G} \mapsto_{CF} cf\text{-}id \mathfrak{D} : \mathfrak{D} \mapsto_{C\alpha} \mathfrak{D}$
and $\wedge x. x \in_{\circ} \mathfrak{D}$ (*Obj*) \implies *universal-arrow-fo* $\mathfrak{F} x$ (\mathfrak{G} (*ObjMap*)(x)) (ε (*NTMap*)(x))

shows *cf-adjunction-of-counit* $\alpha \mathfrak{F} \mathfrak{G} \varepsilon : \mathfrak{F} \Rightarrow_{CF} \mathfrak{G} : \mathfrak{C} \Rightarrow_{C\alpha} \mathfrak{D}$
and ε_C (*cf-adjunction-of-counit* $\alpha \mathfrak{F} \mathfrak{G} \varepsilon$) = ε
and \mathcal{D}_o (*cf-adjunction-of-counit* $\alpha \mathfrak{F} \mathfrak{G} \varepsilon$) (*AdjNT*) (*NTMap*) =
(op-cat $\mathfrak{C} \times_C \mathfrak{D}$) (*Obj*)
and $\wedge c d$. $\llbracket c \varepsilon_o \mathfrak{C}(\text{Obj}); d \varepsilon_o \mathfrak{D}(\text{Obj}) \rrbracket \implies$
cf-adjunction-of-counit $\alpha \mathfrak{F} \mathfrak{G} \varepsilon$ (*AdjNT*) (*NTMap*) ($\langle c, d \rangle_\bullet$) =
(umap-fo $\mathfrak{F} d$ ($\mathfrak{G}(\text{ObjMap})(d)$) ($\varepsilon(\text{NTMap})(d)$) c)⁻¹ *Set*

proof-

interpret \mathfrak{C} : *category* $\alpha \mathfrak{C}$ **by** (*rule assms(1)*)
interpret \mathfrak{D} : *category* $\alpha \mathfrak{D}$ **by** (*rule assms(2)*)
interpret \mathfrak{F} : *is-functor* $\alpha \mathfrak{C} \mathfrak{D} \mathfrak{F}$ **by** (*rule assms(3)*)
interpret \mathfrak{G} : *is-functor* $\alpha \mathfrak{D} \mathfrak{C} \mathfrak{G}$ **by** (*rule assms(4)*)
interpret ε : *is-ntcf* $\alpha \mathfrak{D} \mathfrak{D} \langle \mathfrak{F} \circ_{CF} \mathfrak{G} \rangle \langle \text{cf-id } \mathfrak{D} \rangle \varepsilon$ **by** (*rule assms(5)*)

note *cf-adjunction-of-counit-def'* =
cf-adjunction-of-counit-def [**where** $\mathfrak{F} = \mathfrak{F}$, *unfolded* \mathfrak{F} .*cf-HomDom* \mathfrak{F} .*cf-HomCod*]

have *ua*:
universal-arrow-of (*op-cf* \mathfrak{F}) x (*op-cf* $\mathfrak{G}(\text{ObjMap})(x)$) (*op-ntcf* $\varepsilon(\text{NTMap})(x)$)
if $x \varepsilon_o$ *op-cat* $\mathfrak{D}(\text{Obj})$ **for** x
using that unfolding *cat-op-simps* **by** (*rule assms(6)*)

let $?aou = \langle \text{cf-adjunction-of-unit } \alpha$ (*op-cf* \mathfrak{G}) (*op-cf* \mathfrak{F}) (*op-ntcf* ε)
from

cf-adjunction-of-unit-is-cf-adjunction
 $\left[\begin{array}{l} OF \\ \mathfrak{D}.category-op \\ \mathfrak{C}.category-op \\ \mathfrak{G}.is-functor-op \\ \mathfrak{F}.is-functor-op \\ \varepsilon.is-ntcf-op[unfolding \text{ cat-op-simps}] \\ ua, \\ simplified \text{ cf-adjunction-of-counit-def}[symmetric] \end{array} \right]$

have *aou*: $?aou : \text{op-cf } \mathfrak{G} \Rightarrow_{CF} \text{op-cf } \mathfrak{F} : \text{op-cat } \mathfrak{D} \Rightarrow_{C\alpha} \text{op-cat } \mathfrak{C}$
and $\eta\text{-aou}$: $\eta_C ?aou = \text{op-ntcf } \varepsilon$
by *auto*

interpret *aou*:
is-cf-adjunction $\alpha \langle \text{op-cat } \mathfrak{D} \rangle \langle \text{op-cat } \mathfrak{C} \rangle \langle \text{op-cf } \mathfrak{G} \rangle \langle \text{op-cf } \mathfrak{F} \rangle ?aou$
by (*rule aou*)

from $\eta\text{-aou}$ **have**
 $\text{op-ntcf } (\eta_C ?aou) = \text{op-ntcf } (\text{op-ntcf } \varepsilon)$
by *simp*

then show ε_C (*cf-adjunction-of-counit* $\alpha \mathfrak{F} \mathfrak{G} \varepsilon$) = ε
unfolding

$\varepsilon.is-ntcf-op-ntcf-op-ntcf$
 $is-cf-adjunction.op-ntcf-cf-adjunction-unit[OF \text{ aou}]$
 $cf-adjunction-of-counit-def'[symmetric]$
by (*simp add: cat-op-simps*)

show *aoc-ε*: *cf-adjunction-of-counit* $\alpha \mathfrak{F} \mathfrak{G} \varepsilon : \mathfrak{F} \Rightarrow_{CF} \mathfrak{G} : \mathfrak{C} \Rightarrow_{C\alpha} \mathfrak{D}$
by

$\left(\begin{array}{l} rule \\ is-cf-adjunction-op[\\ OF \text{ aou}, folded \text{ cf-adjunction-of-counit-def}', unfolding \text{ cat-op-simps} \end{array} \right)$

)
interpret $aoc-\varepsilon$: $is-cf-adjunction \alpha \mathfrak{C} \mathfrak{D} \mathfrak{F} \mathfrak{G} \langle cf-adjunction-of-counit \alpha \mathfrak{F} \mathfrak{G} \varepsilon \rangle$
by (*rule aoc- ε*)

from $aoc-\varepsilon.NT.is-ntcf-axioms$ **show**

$\mathcal{D}_\circ (cf-adjunction-of-counit \alpha \mathfrak{F} \mathfrak{G} \varepsilon (AdjNT) (NTMap)) = (op-cat \mathfrak{C} \times_C \mathfrak{D}) (Obj)$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps*)

show $\wedge c d. [\![c \in_\circ \mathfrak{C} (Obj); d \in_\circ \mathfrak{D} (Obj)]\!] \implies$
 $cf-adjunction-of-counit \alpha \mathfrak{F} \mathfrak{G} \varepsilon (AdjNT) (NTMap) (c, d)_\bullet =$
 $(umap-fo \mathfrak{F} d (\mathfrak{G} (ObjMap) (d)) (\varepsilon (NTMap) (d)) c)^{-1}_{Set}$

proof-

fix $c d$ **assume** *prems*: $c \in_\circ \mathfrak{C} (Obj)$ $d \in_\circ \mathfrak{D} (Obj)$

from *assms(1-4)* *prems* **have** *aou-dc*:

cf-adjunction-AdjNT-of-unit

$\alpha (op-cf \mathfrak{G}) (op-cf \mathfrak{F}) (op-ntcf \varepsilon) (NTMap) (d, c)_\bullet =$
 $umap-fo \mathfrak{F} d (\mathfrak{G} (ObjMap) (d)) (\varepsilon (NTMap) (d)) c$

by

(
cs-concl cs-shallow
cs-simp: *cat-op-simps adj-cs-simps cs-intro: cat-op-intros*
)

from *assms(1-4)* *aou* *prems* **have** *ufo- ε -dc*:

$umap-fo \mathfrak{F} d (\mathfrak{G} (ObjMap) (d)) (\varepsilon (NTMap) (d)) c :$
 $Hom_{O.C} \alpha op-cat \mathfrak{C} (op-cf \mathfrak{G} -, -) (ObjMap) (d, c)_\bullet \mapsto_{isocat-Set} c$
 $Hom_{O.C} \alpha op-cat \mathfrak{D} (-, op-cf \mathfrak{F} -) (ObjMap) (d, c)_\bullet$

by

(
cs-concl cs-shallow
cs-simp:
aou-dc[symmetric] cf-adjunction-of-unit-components(3)[symmetric]
cs-intro:
is-iso-ntcf.is-ntcf-is-iso-arr'
adj-cs-intros
cat-cs-intros
cat-op-intros
cat-prod-cs-intros
)

from

assms(1-4)
 $aoc-\varepsilon[unfolding\ cf-adjunction-of-counit-def']$
aou
prems
ufo- ε -dc

show

$cf-adjunction-of-counit \alpha \mathfrak{F} \mathfrak{G} \varepsilon (AdjNT) (NTMap) (c, d)_\bullet =$
 $(umap-fo \mathfrak{F} d (\mathfrak{G} (ObjMap) (d)) (\varepsilon (NTMap) (d)) c)^{-1}_{Set}$

unfolding *cf-adjunction-of-counit-def'*

by

(
cs-concl
cs-simp: *cat-op-simps adj-cs-simps cat-cs-simps cat-Set-cs-simps*
cs-intro: *adj-cs-intros cat-cs-intros cat-prod-cs-intros*
)

qed

qed

13.10 Construction of an adjunction from a functor and universal morphisms from functors to objects

The subsection presents the construction of an adjunction given a functor and a structured collection of universal morphisms from functors to objects. The content of this subsection follows the statement and the proof of Theorem 2-iv in Chapter IV-1 in [9].

13.10.1 Definition and elementary properties

definition $cf\text{-}ra\text{-}of\text{-}la :: (V \Rightarrow V) \Rightarrow V \Rightarrow V \Rightarrow V$
where $cf\text{-}ra\text{-}of\text{-}la F \mathfrak{F} \varepsilon = op\text{-}cf (cf\text{-}la\text{-}of\text{-}ra F (op\text{-}cf \mathfrak{F}) \varepsilon)$

13.10.2 Object map

lemma $cf\text{-}ra\text{-}of\text{-}la\text{-}ObjMap\text{-}vsu[adj\text{-}cs\text{-}intros]$: $vsu (cf\text{-}ra\text{-}of\text{-}la F \mathfrak{F} \varepsilon (ObjMap))$
unfolding $cf\text{-}ra\text{-}of\text{-}la\text{-}def\ op\text{-}cf\text{-}components$ **by** $(auto\ intro: adj\text{-}cs\text{-}intros)$

lemma **(in** $is\text{-}functor$) $cf\text{-}ra\text{-}of\text{-}la\text{-}ObjMap\text{-}vdomain$:
 $\mathcal{D}_o (cf\text{-}ra\text{-}of\text{-}la F \mathfrak{F} \varepsilon (ObjMap)) = \mathfrak{B} (Obj)$
unfolding $cf\text{-}ra\text{-}of\text{-}la\text{-}def\ cf\text{-}la\text{-}of\text{-}ra\text{-}components\ cat\text{-}op\text{-}simps$
by $(simp\ add: cat\text{-}cs\text{-}simps)$

lemmas $[adj\text{-}cs\text{-}simps] = is\text{-}functor.cf\text{-}ra\text{-}of\text{-}la\text{-}ObjMap\text{-}vdomain$

lemma **(in** $is\text{-}functor$) $cf\text{-}ra\text{-}of\text{-}la\text{-}ObjMap\text{-}app$:
assumes $d \in_o \mathfrak{B} (Obj)$
shows $cf\text{-}ra\text{-}of\text{-}la F \mathfrak{F} \varepsilon (ObjMap) (d) = F d$
using $assms$
unfolding $cf\text{-}ra\text{-}of\text{-}la\text{-}def\ cf\text{-}la\text{-}of\text{-}ra\text{-}components\ cat\text{-}op\text{-}simps$
by $(simp\ add: cat\text{-}cs\text{-}simps)$

lemmas $[adj\text{-}cs\text{-}simps] = is\text{-}functor.cf\text{-}ra\text{-}of\text{-}la\text{-}ObjMap\text{-}app$

13.10.3 Arrow map

lemma $cf\text{-}ra\text{-}of\text{-}la\text{-}ArrMap\text{-}app\text{-}unique$:
assumes $\mathfrak{F} : \mathcal{C} \mapsto_{\mathcal{D}} \mathcal{C} \alpha \mathcal{D}$
and $f : a \mapsto_{\mathcal{D}} b$
and $universal\text{-}arrow\text{-}fo \mathfrak{F} a (cf\text{-}ra\text{-}of\text{-}la F \mathfrak{F} \varepsilon (ObjMap) (a)) (\varepsilon (a))$
and $universal\text{-}arrow\text{-}fo \mathfrak{F} b (cf\text{-}ra\text{-}of\text{-}la F \mathfrak{F} \varepsilon (ObjMap) (b)) (\varepsilon (b))$
shows $cf\text{-}ra\text{-}of\text{-}la F \mathfrak{F} \varepsilon (ArrMap) (f) : F a \mapsto_{\mathcal{C}} F b$
and $f \circ_{A \mathcal{D}} \varepsilon (a) =$
 $umap\text{-}fo \mathfrak{F} b (F b) (\varepsilon (b)) (F a) (ArrVal) (cf\text{-}ra\text{-}of\text{-}la F \mathfrak{F} \varepsilon (ArrMap) (f))$
and $\wedge f'$.
 \llbracket
 $f' : F a \mapsto_{\mathcal{C}} F b;$
 $f \circ_{A \mathcal{D}} \varepsilon (a) = umap\text{-}fo \mathfrak{F} b (F b) (\varepsilon (b)) (F a) (ArrVal) (f')$
 $\rrbracket \implies cf\text{-}ra\text{-}of\text{-}la F \mathfrak{F} \varepsilon (ArrMap) (f) = f'$

proof-

interpret $\mathfrak{F} : is\text{-}functor \alpha \mathcal{C} \mathcal{D} \mathfrak{F}$ **by** $(rule\ assms(1))$
from $assms(2)$ **have** $op\text{-}f : f : b \mapsto_{op\text{-}cat} \mathcal{D} a$ **unfolding** $cat\text{-}op\text{-}simps$ **by** $simp$
let $?lara = \langle cf\text{-}la\text{-}of\text{-}ra F (op\text{-}cf \mathfrak{F}) \varepsilon \rangle$
have $lara\text{-}ObjMap\text{-}eq\text{-}op : ?lara (ObjMap) = (op\text{-}cf ?lara (ObjMap))$
and $lara\text{-}ArrMap\text{-}eq\text{-}op : ?lara (ArrMap) = (op\text{-}cf ?lara (ArrMap))$
unfolding $cat\text{-}op\text{-}simps$ **by** $simp\text{-}all$
note $ua\text{-}\eta\text{-}a = \mathfrak{F}.universal\text{-}arrow\text{-}foD [OF\ assms(3)]$
and $ua\text{-}\eta\text{-}b = \mathfrak{F}.universal\text{-}arrow\text{-}foD [OF\ assms(4)]$
from $assms(1,2)$ $ua\text{-}\eta\text{-}a(2)$ **have** $[cat\text{-}op\text{-}simps]$:

$\varepsilon(a) \circ_{A \text{ op-cat } \mathcal{D}} f = f \circ_{A \mathcal{D}} \varepsilon(a)$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps cat-op-simps*)
show *cf-ra-of-la* $F \mathfrak{F} \varepsilon(\text{ArrMap})(f) : F a \mapsto_{\mathcal{C}} F b$
and $f \circ_{A \mathcal{D}} \varepsilon(a) =$
 $\text{umap-fo } \mathfrak{F} b (F b) (\varepsilon(b)) (F a)(\text{ArrVal})(\text{cf-ra-of-la } F \mathfrak{F} \varepsilon(\text{ArrMap})(f))$
and $\wedge f'$.
 \llbracket
 $f' : F a \mapsto_{\mathcal{C}} F b;$
 $f \circ_{A \mathcal{D}} \varepsilon(a) = \text{umap-fo } \mathfrak{F} b (F b) (\varepsilon(b)) (F a)(\text{ArrVal})(f')$
 $\rrbracket \implies \text{cf-ra-of-la } F \mathfrak{F} \varepsilon(\text{ArrMap})(f) = f'$
by
(
intro
cf-la-of-ra-ArrMap-app-unique
[
where $\eta = \varepsilon$ **and** $F = F$,
 $OF \mathfrak{F}.is\text{-functor-op } op\text{-f,}$
unfolded
 $\mathfrak{F}.op\text{-cf-universal-arrow-of}$
lara-ObjMap-eq-op
lara-ArrMap-eq-op,
folded cf-ra-of-la-def,
unfolded cat-op-simps, OF assms(4,3)
]
)+
qed

lemma *cf-ra-of-la-ArrMap-app-is-arr[adj-cs-intros]:*
assumes $\mathfrak{F} : \mathcal{C} \mapsto_{C\alpha} \mathcal{D}$
and $f : a \mapsto_{\mathcal{D}} b$
and *universal-arrow-fo* $\mathfrak{F} a (\text{cf-ra-of-la } F \mathfrak{F} \varepsilon(\text{ObjMap})(a)) (\varepsilon(a))$
and *universal-arrow-fo* $\mathfrak{F} b (\text{cf-ra-of-la } F \mathfrak{F} \varepsilon(\text{ObjMap})(b)) (\varepsilon(b))$
and $F a = F a$
and $F b = F b$
shows *cf-ra-of-la* $F \mathfrak{F} \varepsilon(\text{ArrMap})(f) : F a \mapsto_{\mathcal{C}} F b$
using *assms(1-4) unfolding assms(5,6) by (rule cf-ra-of-la-ArrMap-app-unique)*

13.10.4 An adjunction constructed from a functor and universal morphisms from functors to objects is an adjunction

lemma *op-cf-cf-la-of-ra-op[cat-op-simps]:*
 $op\text{-cf } (\text{cf-la-of-ra } F (op\text{-cf } \mathfrak{F}) \varepsilon) = \text{cf-ra-of-la } F \mathfrak{F} \varepsilon$
unfolding *cf-ra-of-la-def* **by** *simp*

lemma *cf-ra-of-la-commute-op:*
assumes $\mathfrak{F} : \mathcal{C} \mapsto_{C\alpha} \mathcal{D}$
and $\wedge d. d \in_{\circ} \mathcal{D}(\text{Obj}) \implies$
 $\text{universal-arrow-fo } \mathfrak{F} d (\text{cf-ra-of-la } F \mathfrak{F} \varepsilon(\text{ObjMap})(d)) (\varepsilon(d))$
and $\wedge d d' h. h : d \mapsto_{\mathcal{D}} d' \implies$
 $\varepsilon(d') \circ_{A \mathcal{D}} \mathfrak{F}(\text{ArrMap})(\text{cf-ra-of-la } F \mathfrak{F} \varepsilon(\text{ArrMap})(h)) =$
 $h \circ_{A \mathcal{D}} \varepsilon(d)$
and $h : c' \mapsto_{\mathcal{D}} c$
shows $\mathfrak{F}(\text{ArrMap})(\text{cf-ra-of-la } F \mathfrak{F} \varepsilon(\text{ArrMap})(h)) \circ_{A \text{ op-cat } \mathcal{D}} \varepsilon(c) =$
 $\varepsilon(c') \circ_{A \text{ op-cat } \mathcal{D}} h$

proof-
interpret $\mathfrak{F} : is\text{-functor } \alpha \mathcal{C} \mathcal{D} \mathfrak{F}$ **by** (*rule assms(1)*)
from *assms(4)* **have** $c' : c' \in_{\circ} \mathcal{D}(\text{Obj})$ **and** $c : c \in_{\circ} \mathcal{D}(\text{Obj})$ **by** *auto*
note $ua\text{-}\eta\text{-}c' = \mathfrak{F}.universal\text{-arrow-fo}D[OF \text{ assms}(2)[OF c']]$

and $ua-\eta-c = \mathfrak{F}.universal\text{-}arrow\text{-}foD[OF\ assms(2)[OF\ c]]$
note $rala-f = cf\text{-}ra\text{-}of\text{-}la\text{-}ArrMap\text{-}app\text{-}unique[$
 $OF\ assms(1)\ assms(4)\ assms(2)[OF\ c']\ assms(2)[OF\ c]$
 $]$
from $assms(1)\ assms(4)\ ua-\eta-c'(2)\ ua-\eta-c(2)\ rala-f(1)$ **show** *?thesis*
by
 $($
 $cs\text{-}concl\ \mathbf{cs}\text{-}shallow$
 $\mathbf{cs}\text{-}simp: assms(3)\ cat\text{-}op\text{-}simps\ adj\text{-}cs\text{-}simps\ cat\text{-}cs\text{-}simps$
 $\mathbf{cs}\text{-}intro: cat\text{-}cs\text{-}intros$
 $)$
qed

lemma

assumes $\mathfrak{F} : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{D}$
and $\bigwedge d. d \in_{\circ} \mathfrak{D}(\mathfrak{Obj}) \implies F\ d \in_{\circ} \mathfrak{C}(\mathfrak{Obj})$
and $\bigwedge d. d \in_{\circ} \mathfrak{D}(\mathfrak{Obj}) \implies$
 $universal\text{-}arrow\text{-}fo\ \mathfrak{F}\ d\ (cf\text{-}ra\text{-}of\text{-}la\ F\ \mathfrak{F}\ \varepsilon(\mathfrak{ObjMap})(d))\ (\varepsilon(d))$
and $\bigwedge d\ d'\ h. h : d \mapsto_{\mathfrak{D}} d' \implies$
 $\varepsilon(d') \circ_{A\mathfrak{D}} \mathfrak{F}(\mathfrak{ArrMap})(cf\text{-}ra\text{-}of\text{-}la\ F\ \mathfrak{F}\ \varepsilon(\mathfrak{ArrMap})(h)) =$
 $h \circ_{A\mathfrak{D}} \varepsilon(d)$
shows $cf\text{-}ra\text{-}of\text{-}la\text{-}is\text{-}functor: cf\text{-}ra\text{-}of\text{-}la\ F\ \mathfrak{F}\ \varepsilon : \mathfrak{D} \mapsto_{C\alpha} \mathfrak{C}$
and $cf\text{-}la\text{-}of\text{-}ra\text{-}op\text{-}is\text{-}functor:$
 $cf\text{-}la\text{-}of\text{-}ra\ F\ (op\text{-}cf\ \mathfrak{F})\ \varepsilon : op\text{-}cat\ \mathfrak{D} \mapsto_{C\alpha} op\text{-}cat\ \mathfrak{C}$

proof-

interpret $\mathfrak{F} : is\text{-}functor\ \alpha\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}$ **by** (rule $assms(1)$)
have $\mathfrak{F}h\text{-}\varepsilon c:$
 $\mathfrak{F}(\mathfrak{ArrMap})(cf\text{-}ra\text{-}of\text{-}la\ F\ \mathfrak{F}\ \varepsilon(\mathfrak{ArrMap})(h)) \circ_{A\ op\text{-}cat\ \mathfrak{D}} \varepsilon(c) =$
 $\varepsilon(c') \circ_{A\ op\text{-}cat\ \mathfrak{D}} h$
if $h : c' \mapsto_{\mathfrak{D}} c$ **for** $c\ c'\ h$

proof-

from *that* **have** $c' : c' \in_{\circ} \mathfrak{D}(\mathfrak{Obj})$ **and** $c : c \in_{\circ} \mathfrak{D}(\mathfrak{Obj})$ **by** *auto*
note $ua-\eta-c' = \mathfrak{F}.universal\text{-}arrow\text{-}foD[OF\ assms(3)[OF\ c']]$
and $ua-\eta-c = \mathfrak{F}.universal\text{-}arrow\text{-}foD[OF\ assms(3)[OF\ c]]$
note $rala-f = cf\text{-}ra\text{-}of\text{-}la\text{-}ArrMap\text{-}app\text{-}unique[$
 $OF\ assms(1)\ that\ assms(3)[OF\ c']\ assms(3)[OF\ c]$
 $]$
from $assms(1)\ that\ ua-\eta-c'(2)\ ua-\eta-c(2)\ rala-f(1)$ **show** *?thesis*
by
 $($
 $cs\text{-}concl\ \mathbf{cs}\text{-}shallow$
 $\mathbf{cs}\text{-}simp: assms(4)\ cat\text{-}op\text{-}simps\ adj\text{-}cs\text{-}simps\ cat\text{-}cs\text{-}simps$
 $\mathbf{cs}\text{-}intro: cat\text{-}cs\text{-}intros$
 $)$

qed

let $?lara = \langle cf\text{-}la\text{-}of\text{-}ra\ F\ (op\text{-}cf\ \mathfrak{F})\ \varepsilon \rangle$
have $lara\text{-}ObjMap\text{-}eq\text{-}op: ?lara(\mathfrak{ObjMap}) = (op\text{-}cf\ ?lara(\mathfrak{ObjMap}))$
and $lara\text{-}ArrMap\text{-}eq\text{-}op: ?lara(\mathfrak{ArrMap}) = (op\text{-}cf\ ?lara(\mathfrak{ArrMap}))$
by (*simp-all add: cat-op-simps del: op-cf-cf-la-of-ra-op*)
show $cf\text{-}la\text{-}of\text{-}ra\ F\ (op\text{-}cf\ \mathfrak{F})\ \varepsilon : op\text{-}cat\ \mathfrak{D} \mapsto_{C\alpha} op\text{-}cat\ \mathfrak{C}$
by
 $($
 $intro\ cf\text{-}la\text{-}of\text{-}ra\text{-}is\text{-}functor$
 $[$
 $\mathbf{where}\ F=F\ \mathbf{and}\ \eta=\varepsilon,$
 $OF\ \mathfrak{F}.is\text{-}functor\text{-}op,$
 $unfolded\ cat\text{-}op\text{-}simps,$
 $OF\ assms(2),$
 $]$
 $)$

simplified,
unfolded lara-ObjMap-eq-op lara-ArrMap-eq-op,
folded cf-ra-of-la-def,
OF assms(3) $\mathfrak{F}h\text{-}\varepsilon c$,
simplified
])
from
is-functor.is-functor-op[
OF this, unfolded cat-op-simps, folded cf-ra-of-la-def
]
show *cf-ra-of-la* $F \mathfrak{F} \varepsilon : \mathfrak{D} \mapsto_{C\alpha} \mathfrak{C}$.
qed

lemma *cf-ra-of-la-is-ntcf:*
fixes $F \mathfrak{D} \mathfrak{F} \mathfrak{G} \varepsilon_m \varepsilon$
defines $\mathfrak{G} \equiv \text{cf-ra-of-la } F \mathfrak{F} \varepsilon_m$
and $\varepsilon \equiv [\varepsilon_m, \mathfrak{F} \circ_{CF} \mathfrak{G}, \text{cf-id } \mathfrak{D}, \mathfrak{D}, \mathfrak{D}]_o$
assumes $\mathfrak{F} : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{D}$
and $\bigwedge d. d \in_o \mathfrak{D}(\text{Obj}) \implies F d \in_o \mathfrak{C}(\text{Obj})$
and $\bigwedge d. d \in_o \mathfrak{D}(\text{Obj}) \implies$
universal-arrow-fo $\mathfrak{F} d (\mathfrak{G}(\text{ObjMap})(d)) (\varepsilon(\text{NTMap})(d))$
and $\bigwedge d d' h. h : d \mapsto_{\mathfrak{D}} d' \implies$
 $\varepsilon(\text{NTMap})(d) \circ_{A\mathfrak{D}} \mathfrak{F}(\text{ArrMap})(\mathfrak{G}(\text{ArrMap})(h)) = h \circ_{A\mathfrak{D}} \varepsilon(\text{NTMap})(d)$
and *vsv* $(\varepsilon(\text{NTMap}))$
and $\mathfrak{D}_o (\varepsilon(\text{NTMap})) = \mathfrak{D}(\text{Obj})$
shows $\varepsilon : \mathfrak{F} \circ_{CF} \mathfrak{G} \mapsto_{CF} \text{cf-id } \mathfrak{D} : \mathfrak{D} \mapsto_{C\alpha} \mathfrak{D}$

proof-
interpret $\mathfrak{F} : \text{is-functor } \alpha \mathfrak{C} \mathfrak{D} \mathfrak{F}$ **by** (*rule assms(3)*)
have ε -*components:*
 $\varepsilon(\text{NTMap}) = \varepsilon_m$
 $\varepsilon(\text{NTDom}) = \mathfrak{F} \circ_{CF} \mathfrak{G}$
 $\varepsilon(\text{NTCod}) = \text{cf-id } \mathfrak{D}$
 $\varepsilon(\text{NTDGDom}) = \mathfrak{D}$
 $\varepsilon(\text{NTDGCod}) = \mathfrak{D}$
unfolding ε -*def nt-field-simps* **by** (*simp-all add: nat-omega-simps*)
note \mathfrak{G} -*def* = \mathfrak{G} -*def*[*folded* ε -*components(1)*]
interpret $\mathfrak{G} : \text{is-functor } \alpha \mathfrak{D} \mathfrak{C} \mathfrak{G}$
unfolding \mathfrak{G} -*def*
by (*auto intro: cf-ra-of-la-is-functor[OF assms(3-6)[unfolded \mathfrak{G} -def]]*)
interpret $\text{op-}\varepsilon : \text{is-functor}$
 $\alpha \langle \text{op-cat } \mathfrak{D} \rangle \langle \text{op-cat } \mathfrak{C} \rangle \langle \text{cf-la-of-ra } F (\text{op-cf } \mathfrak{F}) (\varepsilon(\text{NTMap})) \rangle$
by
(
intro cf-la-of-ra-op-is-functor[
where $F=F$ **and** $\varepsilon = \langle \varepsilon(\text{NTMap}) \rangle$, *OF assms(3-6)[unfolded \mathfrak{G} -def], simplified*
])
interpret $\varepsilon : \text{vfsequence } \varepsilon$ **unfolding** ε -*def* **by** *simp*
have [*cat-op-simps*]: *op-ntcf* (*op-ntcf* ε) = ε
proof(*rule vsv-eqI*)
have *dom-lhs*: $\mathfrak{D}_o (\text{op-ntcf } (\text{op-ntcf } \varepsilon)) = 5_{\mathfrak{N}}$
unfolding *op-ntcf-def* **by** (*simp add: nat-omega-simps*)
from *assms(7)* **show** $\mathfrak{D}_o (\text{op-ntcf } (\text{op-ntcf } \varepsilon)) = \mathfrak{D}_o \varepsilon$
unfolding *dom-lhs* **by** (*simp add: ε -def ε .vfsequence-vdomain nat-omega-simps*)
have *sup:*
 $\text{op-ntcf } (\text{op-ntcf } \varepsilon)(\text{NTDom}) = \varepsilon(\text{NTDom})$
 $\text{op-ntcf } (\text{op-ntcf } \varepsilon)(\text{NTCod}) = \varepsilon(\text{NTCod})$

```

op-ntcf (op-ntcf ε)(\NTDGD\om) = ε(\NTDGD\om)
op-ntcf (op-ntcf ε)(\NTDGC\od) = ε(\NTDGC\od)
unfolding op-ntcf-components cat-op-simps ε-components
by simp-all
show a ∈o Do (op-ntcf (op-ntcf ε)) ⇒ op-ntcf (op-ntcf ε)(\a) = ε(\a) for a
by (unfold dom-lhs, elim-in-numeral, fold nt-field-simps, unfold sup)
(simp-all add: cat-op-simps)
qed (auto simp: op-ntcf-def)
let ?lara = ⟨cf-la-of-ra F (op-cf ℱ) (ε(\NTMap))⟩
have lara-ObjMap-eq-op: ?lara(\ObjMap) = (op-cf ?lara(\ObjMap))
and lara-ArrMap-eq-op: ?lara(\ArrMap) = (op-cf ?lara(\ArrMap))
by (simp-all add: cat-op-simps del: op-cf-cf-la-of-ra-op)
have seq: vsequence (op-ntcf ε) unfolding op-ntcf-def by auto
have card: vcard (op-ntcf ε) = 5N
unfolding op-ntcf-def by (simp add: nat-omega-simps)
have op-cf-NTCod: op-cf (ε(\NTCod)) = cf-id (op-cat D)
unfolding ε-components cat-op-simps by simp
from assms(3) have op-cf-NTDom:
op-cf (ε(\NTDom)) = op-cf ℱ ∘CF cf-la-of-ra F (op-cf ℱ) (ε(\NTMap))
unfolding ε-components
by
(
simp
add: cat-op-simps ℑ-def cf-ra-of-la-def ε-components(1)[symmetric]
del: op-cf-cf-la-of-ra-op
)
note cf-la-of-ra-is-ntcf
[
where F=F and ηm=⟨ε(\NTMap)⟩,
OF is-functor.is-functor-op[OF assms(3)],
unfolded cat-op-simps,
OF assms(4),
unfolded ε-components(1),
folded op-cf-NTCod op-cf-NTDom[unfolded ε-components(1)] ε-components(1),
folded op-ntcf-def[of ε, unfolded ε-components(4,5)],
unfolded
cat-op-simps
lara-ObjMap-eq-op lara-ArrMap-eq-op cf-ra-of-la-def[symmetric],
folded ℑ-def,
simplified,
OF
assms(5)
cf-ra-of-la-commute-op[
OF assms(3,5,6)[unfolded ℑ-def], folded ℑ-def
]
assms(7,8),
unfolded ε-components,
simplified
]
from is-ntcf.is-ntcf-op[OF this, unfolded cat-op-simps ℑ-def[symmetric]] show
ε : ℱ ∘CF ℑ ↦CF cf-id D : D ↦↦Cα D.
qed

```

lemma cf-ra-of-la-is-counit:

```

fixes F D ℱ ℑ εm ε
defines ℑ ≡ cf-ra-of-la F ℱ εm
and ε ≡ [εm, ℱ ∘CF ℑ, cf-id D, D, D]o.
assumes category α C

```

```

and category  $\alpha \mathfrak{D}$ 
and  $\mathfrak{F} : \mathfrak{C} \mapsto \mapsto_{C\alpha} \mathfrak{D}$ 
and  $\wedge d. d \in_{\circ} \mathfrak{D}(\text{Obj}) \implies F d \in_{\circ} \mathfrak{C}(\text{Obj})$ 
and  $\wedge d. d \in_{\circ} \mathfrak{D}(\text{Obj}) \implies$ 
  universal-arrow-fo  $\mathfrak{F} d (\mathfrak{G}(\text{ObjMap})(d)) (\varepsilon(\text{NTMap})(d))$ 
and  $\wedge d d' h. h : d \mapsto_{\mathfrak{D}} d' \implies$ 
   $\varepsilon(\text{NTMap})(d') \circ_{A\mathfrak{D}} \mathfrak{F}(\text{ArrMap})(\mathfrak{G}(\text{ArrMap})(h)) = h \circ_{A\mathfrak{D}} \varepsilon(\text{NTMap})(d)$ 
and vfsequence  $\varepsilon$ 
and vsv  $(\varepsilon(\text{NTMap}))$ 
and  $\mathfrak{D}_{\circ} (\varepsilon(\text{NTMap})) = \mathfrak{D}(\text{Obj})$ 
shows cf-adjunction-of-counit  $\alpha \mathfrak{F} \mathfrak{G} \varepsilon : \mathfrak{F} \rightleftharpoons_{CF} \mathfrak{G} : \mathfrak{C} \rightleftharpoons_{C\alpha} \mathfrak{D}$ 
and  $\varepsilon_C (\text{cf-adjunction-of-counit } \alpha \mathfrak{F} \mathfrak{G} \varepsilon) = \varepsilon$ 
proof-
have  $\varepsilon$ -components:
   $\varepsilon(\text{NTMap}) = \varepsilon_m$ 
   $\varepsilon(\text{NTDom}) = \mathfrak{F} \circ_{CF} \mathfrak{G}$ 
   $\varepsilon(\text{NTCod}) = \text{cf-id } \mathfrak{D}$ 
   $\varepsilon(\text{NTDGDom}) = \mathfrak{D}$ 
   $\varepsilon(\text{NTDGCod}) = \mathfrak{D}$ 
unfolding  $\varepsilon$ -def nt-field-simps by (simp-all add: nat-omega-simps)
note  $\mathfrak{G}\text{-def} = \mathfrak{G}\text{-def}[\text{folded } \varepsilon\text{-components}(1)]$ 
note  $\mathfrak{F} = \text{cf-ra-of-la-is-functor}$ 
  where  $F=F$  and  $\varepsilon = \langle \varepsilon(\text{NTMap}) \rangle$ , OF assms(5-8)[unfolded  $\mathfrak{G}$ -def], simplified
note  $\varepsilon = \text{cf-ra-of-la-is-ntcf}$ 
  [
    where  $F=F$  and  $\varepsilon_m = \langle \varepsilon_m \rangle$  and  $\mathfrak{D}=\mathfrak{D}$  and  $\mathfrak{F}=\mathfrak{F}$ ,
    folded  $\mathfrak{G}$ -def[unfolded  $\varepsilon$ -components(1)],
    folded  $\varepsilon$ -def,
    OF assms(5-8) assms(10,11),
    simplified
  ]
show cf-adjunction-of-counit  $\alpha \mathfrak{F} \mathfrak{G} \varepsilon : \mathfrak{F} \rightleftharpoons_{CF} \mathfrak{G} : \mathfrak{C} \rightleftharpoons_{C\alpha} \mathfrak{D}$ 
and  $\varepsilon_C (\text{cf-adjunction-of-counit } \alpha \mathfrak{F} \mathfrak{G} \varepsilon) = \varepsilon$ 
by
  (
    intro cf-adjunction-of-counit-is-cf-adjunction
    [
      OF assms(3-5)  $\mathfrak{F}$ ,
      folded  $\mathfrak{G}$ -def,
      OF  $\varepsilon$  assms(7)[folded  $\mathfrak{G}$ -def],
      simplified
    ]
  )+
qed

```

13.11 Construction of an adjunction from the counit-unit equations

The subsection presents the construction of an adjunction given two natural transformations satisfying counit-unit equations. The content of this subsection follows the statement and the proof of Theorem 2-v in Chapter IV-1 in [9].

lemma *counit-unit-is-cf-adjunction:*

```

assumes category  $\alpha \mathfrak{C}$ 
and category  $\alpha \mathfrak{D}$ 
and  $\mathfrak{F} : \mathfrak{C} \mapsto \mapsto_{C\alpha} \mathfrak{D}$ 
and  $\mathfrak{G} : \mathfrak{D} \mapsto \mapsto_{C\alpha} \mathfrak{C}$ 
and  $\eta : \text{cf-id } \mathfrak{C} \mapsto_{CF} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{C} \mapsto \mapsto_{C\alpha} \mathfrak{C}$ 

```

and $\varepsilon : \mathfrak{F} \circ_{CF} \mathfrak{G} \mapsto_{CF} \text{cf-id } \mathfrak{D} : \mathfrak{D} \mapsto_{C\alpha} \mathfrak{D}$
 and $(\mathfrak{G} \circ_{CF-NTCF} \varepsilon) \cdot_{NTCF} (\eta \circ_{NTCF-CF} \mathfrak{G}) = \text{ntcf-id } \mathfrak{G}$
 and $(\varepsilon \circ_{NTCF-CF} \mathfrak{F}) \cdot_{NTCF} (\mathfrak{F} \circ_{CF-NTCF} \eta) = \text{ntcf-id } \mathfrak{F}$
 shows *cf-adjunction-of-unit* $\alpha \mathfrak{F} \mathfrak{G} \eta : \mathfrak{F} \rightleftharpoons_{CF} \mathfrak{G} : \mathfrak{C} \rightleftharpoons_{C\alpha} \mathfrak{D}$
 and η_C (*cf-adjunction-of-unit* $\alpha \mathfrak{F} \mathfrak{G} \eta$) = η
 and ε_C (*cf-adjunction-of-unit* $\alpha \mathfrak{F} \mathfrak{G} \eta$) = ε

proof-

interpret \mathfrak{C} : category $\alpha \mathfrak{C}$ by (rule *assms(1)*)
 interpret \mathfrak{D} : category $\alpha \mathfrak{D}$ by (rule *assms(2)*)
 interpret \mathfrak{F} : is-functor $\alpha \mathfrak{C} \mathfrak{D} \mathfrak{F}$ by (rule *assms(3)*)
 interpret \mathfrak{G} : is-functor $\alpha \mathfrak{D} \mathfrak{C} \mathfrak{G}$ by (rule *assms(4)*)
 interpret η : is-ntcf $\alpha \mathfrak{C} \mathfrak{C} \langle \text{cf-id } \mathfrak{C} \rangle \langle \mathfrak{G} \circ_{CF} \mathfrak{F} \rangle \eta$ by (rule *assms(5)*)
 interpret ε : is-ntcf $\alpha \mathfrak{D} \mathfrak{D} \langle \mathfrak{F} \circ_{CF} \mathfrak{G} \rangle \langle \text{cf-id } \mathfrak{D} \rangle \varepsilon$ by (rule *assms(6)*)

have $\mathfrak{G} \varepsilon x \eta \mathfrak{G} x$ [*cat-cs-simps*]:

$\mathfrak{G}(\downarrow \text{ArrMap})(\downarrow \varepsilon(\downarrow \text{NTMap})(\downarrow x)) \circ_{A\mathfrak{C}} \eta(\downarrow \text{NTMap})(\downarrow \mathfrak{G}(\downarrow \text{ObjMap})(\downarrow x)) = \mathfrak{C}(\downarrow \text{CId})(\downarrow \mathfrak{G}(\downarrow \text{ObjMap})(\downarrow x))$
 if $x \in_{\circ} \mathfrak{D}(\downarrow \text{Obj})$ for x

proof-

from *assms(7)* have

$((\mathfrak{G} \circ_{CF-NTCF} \varepsilon) \cdot_{NTCF} (\eta \circ_{NTCF-CF} \mathfrak{G}))(\downarrow \text{NTMap})(\downarrow x) = \text{ntcf-id } \mathfrak{G}(\downarrow \text{NTMap})(\downarrow x)$
 by *simp*

from *this assms(1-6)* that show

$\mathfrak{G}(\downarrow \text{ArrMap})(\downarrow \varepsilon(\downarrow \text{NTMap})(\downarrow x)) \circ_{A\mathfrak{C}} \eta(\downarrow \text{NTMap})(\downarrow \mathfrak{G}(\downarrow \text{ObjMap})(\downarrow x)) =$
 $\mathfrak{C}(\downarrow \text{CId})(\downarrow \mathfrak{G}(\downarrow \text{ObjMap})(\downarrow x))$

by (*cs-prems cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

qed

have [*cat-cs-simps*]:

$\mathfrak{G}(\downarrow \text{ArrMap})(\downarrow \varepsilon(\downarrow \text{NTMap})(\downarrow x)) \circ_{A\mathfrak{C}} (\eta(\downarrow \text{NTMap})(\downarrow \mathfrak{G}(\downarrow \text{ObjMap})(\downarrow x)) \circ_{A\mathfrak{C}} f) =$
 $\mathfrak{C}(\downarrow \text{CId})(\downarrow \mathfrak{G}(\downarrow \text{ObjMap})(\downarrow x)) \circ_{A\mathfrak{C}} f$

if $x \in_{\circ} \mathfrak{D}(\downarrow \text{Obj})$ and $f : a \mapsto_{\mathfrak{C}} \mathfrak{G}(\downarrow \text{ObjMap})(\downarrow x)$ for $x f a$

using *assms(1-6)* that

by (*intro C.cat-assoc-helper*)

(*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)+

have [*cat-cs-simps*]:

$\varepsilon(\downarrow \text{NTMap})(\downarrow \mathfrak{F}(\downarrow \text{ObjMap})(\downarrow x)) \circ_{A\mathfrak{D}} \mathfrak{F}(\downarrow \text{ArrMap})(\downarrow \eta(\downarrow \text{NTMap})(\downarrow x)) = \mathfrak{D}(\downarrow \text{CId})(\downarrow \mathfrak{F}(\downarrow \text{ObjMap})(\downarrow x))$
 if $x \in_{\circ} \mathfrak{C}(\downarrow \text{Obj})$ for x

proof-

from *assms(8)* have

$((\varepsilon \circ_{NTCF-CF} \mathfrak{F}) \cdot_{NTCF} (\mathfrak{F} \circ_{CF-NTCF} \eta))(\downarrow \text{NTMap})(\downarrow x) = \text{ntcf-id } \mathfrak{F}(\downarrow \text{NTMap})(\downarrow x)$
 by *simp*

from *this assms(1-6)* that show

$\varepsilon(\downarrow \text{NTMap})(\downarrow \mathfrak{F}(\downarrow \text{ObjMap})(\downarrow x)) \circ_{A\mathfrak{D}} \mathfrak{F}(\downarrow \text{ArrMap})(\downarrow \eta(\downarrow \text{NTMap})(\downarrow x)) = \mathfrak{D}(\downarrow \text{CId})(\downarrow \mathfrak{F}(\downarrow \text{ObjMap})(\downarrow x))$
 by (*cs-prems cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

qed

have *ua-Fx-ηx*: universal-arrow-of $\mathfrak{G} x$ ($\mathfrak{F}(\downarrow \text{ObjMap})(\downarrow x)$) ($\eta(\downarrow \text{NTMap})(\downarrow x)$)

if $x \in_{\circ} \mathfrak{C}(\downarrow \text{Obj})$ for x

proof(*intro is-functor.universal-arrow-ofI*)

from *assms(3)* that show $\mathfrak{F}(\downarrow \text{ObjMap})(\downarrow x) \in_{\circ} \mathfrak{D}(\downarrow \text{Obj})$

by (*cs-concl cs-shallow cs-intro: cat-cs-intros*)

from *assms(3-6)* that show $\eta(\downarrow \text{NTMap})(\downarrow x) : x \mapsto_{\mathfrak{C}} \mathfrak{G}(\downarrow \text{ObjMap})(\downarrow \mathfrak{F}(\downarrow \text{ObjMap})(\downarrow x))$

by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

fix $r' u'$ assume *prems'*: $r' \in_{\circ} \mathfrak{D}(\downarrow \text{Obj})$ $u' : x \mapsto_{\mathfrak{C}} \mathfrak{G}(\downarrow \text{ObjMap})(\downarrow r')$

show $\exists! f'$.

$f' : \mathfrak{F}(\downarrow \text{ObjMap})(\downarrow x) \mapsto_{\mathfrak{D}} r' \wedge$

$u' = \text{umap-of } \mathfrak{G} x$ ($\mathfrak{F}(\downarrow \text{ObjMap})(\downarrow x)$) ($\eta(\downarrow \text{NTMap})(\downarrow x)$) $r'(\downarrow \text{ArrVal})(\downarrow f')$

proof(*intro ex1I conjI; (elim conjE)?*)
from *assms(3-6)* **that** *prems'* **show**
 $\varepsilon(\backslash NTMap)(\backslash r')$ $\circ_{A\mathfrak{D}}$ $\mathfrak{F}(\backslash ArrMap)(\backslash u')$: $\mathfrak{F}(\backslash ObjMap)(\backslash x) \mapsto_{\mathfrak{D}} r'$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)
from *assms(3-6)* *prems'* **have** $\mathfrak{G}\mathfrak{F}u'$:
 $(\mathfrak{G} \circ_{CF} \mathfrak{F})(\backslash ArrMap)(\backslash u') = \mathfrak{G}(\backslash ArrMap)(\backslash \mathfrak{F}(\backslash ArrMap)(\backslash u'))$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)
note [*cat-cs-simps*] =
 $\eta.ntcf\text{-Comp-commute}[\textit{symmetric}, OF\ \textit{prems}'(2), \textit{unfolded } \mathfrak{G}\mathfrak{F}u']$
from *assms(3-6)* **that** *prems'* **show**
 $u' =$
 $umap\text{-of } \mathfrak{G} \ x \ (\mathfrak{F}(\backslash ObjMap)(\backslash x)) \ (\eta(\backslash NTMap)(\backslash x)) \ r'(\backslash ArrVal)(\backslash \varepsilon(\backslash NTMap)(\backslash r') \circ_{A\mathfrak{D}})$
 $\mathfrak{F}(\backslash ArrMap)(\backslash u'))$
by
(
cs-concl cs-shallow
cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-prod-cs-intros
)
fix f' **assume** *prems''*:
 $f' : \mathfrak{F}(\backslash ObjMap)(\backslash x) \mapsto_{\mathfrak{D}} r'$
 $u' = umap\text{-of } \mathfrak{G} \ x \ (\mathfrak{F}(\backslash ObjMap)(\backslash x)) \ (\eta(\backslash NTMap)(\backslash x)) \ r'(\backslash ArrVal)(\backslash f')$
from *prems''(2,1)* *assms(3-6)* **that** **have** u' -def:
 $u' = \mathfrak{G}(\backslash ArrMap)(\backslash f') \circ_{A\mathfrak{C}} \eta(\backslash NTMap)(\backslash x)$
by
(
cs-prems cs-shallow
cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-prod-cs-intros
)
from
 $\varepsilon.ntcf\text{-Comp-commute}[OF\ \textit{prems}''(1)]$
assms(3-6)
prems''(1)
have [*cat-cs-simps*]:
 $\varepsilon(\backslash NTMap)(\backslash r') \circ_{A\mathfrak{D}} \mathfrak{F}(\backslash ArrMap)(\backslash \mathfrak{G}(\backslash ArrMap)(\backslash f')) =$
 $f' \circ_{A\mathfrak{D}} \varepsilon(\backslash NTMap)(\backslash \mathfrak{F}(\backslash ObjMap)(\backslash x))$
by (*cs-prems cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)
have [*cat-cs-simps*]:
 $\varepsilon(\backslash NTMap)(\backslash r') \circ_{A\mathfrak{D}} (\mathfrak{F}(\backslash ArrMap)(\backslash \mathfrak{G}(\backslash ArrMap)(\backslash f')) \circ_{A\mathfrak{D}} f) =$
 $(f' \circ_{A\mathfrak{D}} \varepsilon(\backslash NTMap)(\backslash \mathfrak{F}(\backslash ObjMap)(\backslash x))) \circ_{A\mathfrak{D}} f$
if $f : a \mapsto_{\mathfrak{D}} \mathfrak{F}(\backslash ObjMap)(\backslash \mathfrak{G}(\backslash ObjMap)(\backslash \mathfrak{F}(\backslash ObjMap)(\backslash x)))$ **for** $f \ a$
using *assms(1-6)* *prems''(1)* *prems'* **that**
by (*intro D.cat-assoc-helper*)
(
cs-concl
cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-prod-cs-intros
)+
from *prems''(2,1)* *assms(3-6)* **that** **show**
 $f' = \varepsilon(\backslash NTMap)(\backslash r') \circ_{A\mathfrak{D}} \mathfrak{F}(\backslash ArrMap)(\backslash u')$
unfolding u' -def
by
(
cs-concl cs-shallow
cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-prod-cs-intros
)
qed
qed (*auto intro: cat-cs-intros*)

show *aou: cf-adjunction-of-unit* $\alpha \ \mathfrak{F} \ \mathfrak{G} \ \eta : \mathfrak{F} \rightleftharpoons_{CF} \mathfrak{G} : \mathfrak{C} \rightleftharpoons_{C\alpha} \mathfrak{D}$

by (intro cf-adjunction-of-unit-is-cf-adjunction ua- \mathfrak{F} x- η x assms(1-5))
 from \mathcal{C} .category-axioms \mathcal{D} .category-axioms show
 η_C (cf-adjunction-of-unit α \mathfrak{F} \mathfrak{G} η) = η
 by
 (

 cs-concl cs-shallow
 cs-intro: cf-adjunction-of-unit-is-cf-adjunction assms(1-5) ua- \mathfrak{F} x- η x
)

interpret aou: is-cf-adjunction α \mathcal{C} \mathcal{D} \mathfrak{F} \mathfrak{G} \langle cf-adjunction-of-unit α \mathfrak{F} \mathfrak{G} η
 by (rule aou)

show ε_C (cf-adjunction-of-unit α \mathfrak{F} \mathfrak{G} η) = ε
 proof(rule ntcf-eqI)
 show ε - η : ε_C (cf-adjunction-of-unit α \mathfrak{F} \mathfrak{G} η) :
 $\mathfrak{F} \circ_{CF} \mathfrak{G} \mapsto_{CF} \text{cf-id } \mathcal{D} : \mathcal{D} \mapsto_{C\alpha} \mathcal{D}$
 by (rule aou.cf-adjunction-counit-is-ntcf)
 from assms(1-6) ε - η have dom-lhs:
 \mathcal{D}_o . (ε_C (cf-adjunction-of-unit α \mathfrak{F} \mathfrak{G} η)($\downarrow NTMap$)) = \mathcal{D} ($\downarrow Obj$)
 by (cs-concl cs-shallow cs-simp: cat-cs-simps)
 from assms(1-6) ε - η have dom-rhs: \mathcal{D}_o . (ε ($\downarrow NTMap$)) = \mathcal{D} ($\downarrow Obj$)
 by (cs-concl cs-shallow cs-simp: cat-cs-simps)
 show ε_C (cf-adjunction-of-unit α \mathfrak{F} \mathfrak{G} η)($\downarrow NTMap$) = ε ($\downarrow NTMap$)
 proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
 fix a assume a \in_o \mathcal{D} ($\downarrow Obj$)
 with aou.is-cf-adjunction-axioms assms(1-6) show
 ε_C (cf-adjunction-of-unit α \mathfrak{F} \mathfrak{G} η)($\downarrow NTMap$)(a) = ε ($\downarrow NTMap$)(a)
 by
 (

 cs-concl
 cs-intro:
 cat-arrow-cs-intros
 cat-op-intros
 cat-cs-intros
 cat-prod-cs-intros
 cs-simp:
 aou.cf-adj-umap-of-unit'[symmetric]
 cat-Set-the-inverse[symmetric]
 adj-cs-simps cat-cs-simps cat-op-simps
)
 qed (auto simp: adj-cs-intros)
 qed (auto simp: assms)

qed

lemma counit-unit-cf-adjunction-of-counit-is-cf-adjunction:

assumes category α \mathcal{C}
 and category α \mathcal{D}
 and $\mathfrak{F} : \mathcal{C} \mapsto_{C\alpha} \mathcal{D}$
 and $\mathfrak{G} : \mathcal{D} \mapsto_{C\alpha} \mathcal{C}$
 and $\eta : \text{cf-id } \mathcal{C} \mapsto_{CF} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathcal{C} \mapsto_{C\alpha} \mathcal{C}$
 and $\varepsilon : \mathfrak{F} \circ_{CF} \mathfrak{G} \mapsto_{CF} \text{cf-id } \mathcal{D} : \mathcal{D} \mapsto_{C\alpha} \mathcal{D}$
 and $(\mathfrak{G} \circ_{CF-NTCF} \varepsilon) \cdot_{NTCF} (\eta \circ_{NTCF-CF} \mathfrak{G}) = \text{ntcf-id } \mathfrak{G}$
 and $(\varepsilon \circ_{NTCF-CF} \mathfrak{F}) \cdot_{NTCF} (\mathfrak{F} \circ_{CF-NTCF} \eta) = \text{ntcf-id } \mathfrak{F}$
 shows cf-adjunction-of-counit α \mathfrak{F} \mathfrak{G} $\varepsilon : \mathfrak{F} \rightleftharpoons_{CF} \mathfrak{G} : \mathcal{C} \rightleftharpoons_{C\alpha} \mathcal{D}$
 and η_C (cf-adjunction-of-counit α \mathfrak{F} \mathfrak{G} ε) = η
 and ε_C (cf-adjunction-of-counit α \mathfrak{F} \mathfrak{G} ε) = ε
 proof-

interpret \mathcal{C} : category α \mathcal{C} **by** (rule *assms(1)*)
interpret \mathcal{D} : category α \mathcal{D} **by** (rule *assms(2)*)
interpret \mathfrak{F} : is-functor α \mathcal{C} \mathcal{D} \mathfrak{F} **by** (rule *assms(3)*)
interpret \mathfrak{G} : is-functor α \mathcal{D} \mathcal{C} \mathfrak{G} **by** (rule *assms(4)*)
interpret η : is-ntcf α \mathcal{C} \mathcal{C} \langle cf-id \mathcal{C} \rangle \langle $\mathfrak{G} \circ_{CF} \mathfrak{F}$ \rangle η **by** (rule *assms(5)*)
interpret ε : is-ntcf α \mathcal{D} \mathcal{D} \langle $\mathfrak{F} \circ_{CF} \mathfrak{G}$ \rangle \langle cf-id \mathcal{D} \rangle ε **by** (rule *assms(6)*)

have *unit-op: cf-adjunction-of-unit* α (op-cf \mathfrak{G}) (op-cf \mathfrak{F}) (op-ntcf ε) :
op-cf $\mathfrak{G} \Rightarrow_{CF}$ op-cf \mathfrak{F} : op-cat $\mathcal{D} \Rightarrow_{C\alpha}$ op-cat \mathcal{C}
by (rule *counit-unit-is-cf-adjunction(1)* [**where** $\varepsilon = \langle$ op-ntcf η \rangle])

(

cs-concl

cs-simp:

cat-op-simps cat-cs-simps

 \mathfrak{G} .cf-ntcf-id-op-cf

 \mathfrak{F} .cf-ntcf-id-op-cf

op-ntcf-ntcf-vcomp[*symmetric*]

op-ntcf-ntcf-cf-comp[*symmetric*]

op-ntcf-cf-ntcf-comp[*symmetric*]

assms(7,8)

cs-intro: *cat-op-intros cat-cs-intros*

)+

then show *aou: cf-adjunction-of-counit* α \mathfrak{F} \mathfrak{G} ε : $\mathfrak{F} \Rightarrow_{CF}$ $\mathfrak{G} : \mathcal{C} \Rightarrow_{C\alpha}$ \mathcal{D}
unfolding *cf-adjunction-of-counit-def*

by

(

subst \mathfrak{F} .cf-op-cf-op-cf[*symmetric*],

subst \mathfrak{G} .cf-op-cf-op-cf[*symmetric*],

subst \mathcal{C} .cat-op-cat-op-cat[*symmetric*],

subst \mathcal{D} .cat-op-cat-op-cat[*symmetric*],

rule *is-cf-adjunction-op*

)

interpret *aou: is-cf-adjunction* α \mathcal{C} \mathcal{D} \mathfrak{F} \mathfrak{G} \langle cf-adjunction-of-counit α \mathfrak{F} \mathfrak{G} ε \rangle
by (rule *aou*)

show η_C (*cf-adjunction-of-counit* α \mathfrak{F} \mathfrak{G} ε) = η
unfolding *cf-adjunction-of-counit-def*

by

(

cs-concl-step is-cf-adjunction.op-ntcf-cf-adjunction-counit[*symmetric*],

rule *unit-op*,

cs-concl-step counit-unit-is-cf-adjunction(3) [**where** $\varepsilon = \langle$ op-ntcf η \rangle],

insert \mathcal{C} .category-op \mathcal{D} .category-op,

rule \mathcal{D} .category-op, rule \mathcal{C} .category-op

)

(

cs-concl

cs-simp:

cat-op-simps cat-cs-simps

 \mathfrak{G} .cf-ntcf-id-op-cf

 \mathfrak{F} .cf-ntcf-id-op-cf

op-ntcf-ntcf-vcomp[*symmetric*]

op-ntcf-ntcf-cf-comp[*symmetric*]

op-ntcf-cf-ntcf-comp[*symmetric*]

assms(7,8)

cs-intro: *cat-op-intros cat-cs-intros*

)+

show ε_C (*cf-adjunction-of-counit* α \mathfrak{F} \mathfrak{G} ε) = ε

unfolding *cf-adjunction-of-counit-def*

by

```
(
  cs-concl-step is-cf-adjunction.op-ntcf-cf-adjunction-unit[symmetric],
  rule unit-op,
  cs-concl-step counit-unit-is-cf-adjunction(2)[where  $\varepsilon = \langle \text{op-ntcf } \eta \rangle$ ],
  insert  $\mathfrak{C}$ .category-op  $\mathfrak{D}$ .category-op,
  rule  $\mathfrak{D}$ .category-op, rule  $\mathfrak{C}$ .category-op
)
(
  cs-concl
  cs-simp:
    cat-op-simps cat-cs-simps
     $\mathfrak{G}$ .cf-ntcf-id-op-cf
     $\mathfrak{F}$ .cf-ntcf-id-op-cf
    op-ntcf-ntcf-vcomp[symmetric]
    op-ntcf-ntcf-cf-comp[symmetric]
    op-ntcf-cf-ntcf-comp[symmetric]
    assms(7,8)
  cs-intro: cat-op-intros cat-cs-intros
)
)+
```

qed

13.12 Adjoints are unique up to isomorphism

The content of the following subsection is based predominantly on the statement and the proof of Corollary 1 in Chapter IV-1 in [9]. However, similar results can also be found in section 4 in [14] and in subsection 2.1 in [4].

13.12.1 Definitions and elementary properties

definition *cf-adj-LR-iso* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$

where *cf-adj-LR-iso* \mathfrak{C} \mathfrak{D} \mathfrak{G} \mathfrak{F} Φ \mathfrak{F}' $\Psi =$

```
[
  (
     $\lambda x \in_o \mathfrak{C}(\text{Obj}). \text{THE } f'$ .
    let
       $\eta = \eta_C \Phi$ ;
       $\eta' = \eta_C \Psi$ ;
       $\mathfrak{F}x = \mathfrak{F}(\text{ObjMap})(x)$ ;
       $\mathfrak{F}'x = \mathfrak{F}'(\text{ObjMap})(x)$ 
    in
       $f' : \mathfrak{F}x \mapsto_{\mathfrak{D}} \mathfrak{F}'x \wedge$ 
       $\eta'(\text{NTMap})(x) = \text{umap-of } \mathfrak{G} \ x \ (\mathfrak{F}x) \ (\eta(\text{NTMap})(x)) \ (\mathfrak{F}'x)(\text{ArrVal})(f')$ 
  ),
   $\mathfrak{F}$ ,
   $\mathfrak{F}'$ ,
   $\mathfrak{C}$ ,
   $\mathfrak{D}$ 
]o
```

definition *cf-adj-RL-iso* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$

where *cf-adj-RL-iso* \mathfrak{C} \mathfrak{D} \mathfrak{F} \mathfrak{G} Φ \mathfrak{G}' $\Psi =$

[
 (
 $\lambda x \in_o \mathcal{D}(\text{Obj}). \text{THE } f'$.
let
 $\varepsilon = \varepsilon_C \Phi$;
 $\varepsilon' = \varepsilon_C \Psi$;
 $\mathcal{G}x = \mathcal{G}(\text{ObjMap})(x)$;
 $\mathcal{G}'x = \mathcal{G}'(\text{ObjMap})(x)$
in
 $f' : \mathcal{G}'x \mapsto_{\mathcal{C}} \mathcal{G}x \wedge$
 $\varepsilon'(\text{NTMap})(x) = \text{umap-fo } \mathfrak{F} x \mathcal{G}x (\varepsilon(\text{NTMap})(x)) \mathcal{G}'x(\text{ArrVal})(f')$
),
 \mathcal{G}' ,
 \mathcal{G} ,
 \mathcal{D} ,
 \mathcal{C}
]_o.

Components.

lemma *cf-adj-LR-iso-components*:

shows *cf-adj-LR-iso* $\mathcal{C} \mathcal{D} \mathcal{G} \mathfrak{F} \Phi \mathfrak{F}' \Psi(\text{NTMap}) =$

(
 $\lambda x \in_o \mathcal{C}(\text{Obj}). \text{THE } f'$.
let
 $\eta = \eta_C \Phi$;
 $\eta' = \eta_C \Psi$;
 $\mathfrak{F}x = \mathfrak{F}(\text{ObjMap})(x)$;
 $\mathfrak{F}'x = \mathfrak{F}'(\text{ObjMap})(x)$
in
 $f' : \mathfrak{F}x \mapsto_{\mathcal{D}} \mathfrak{F}'x \wedge$
 $\eta'(\text{NTMap})(x) = \text{umap-of } \mathcal{G} x \mathfrak{F}x (\eta(\text{NTMap})(x)) \mathfrak{F}'x(\text{ArrVal})(f')$
)
and [*adj-cs-simps*]: *cf-adj-LR-iso* $\mathcal{C} \mathcal{D} \mathcal{G} \mathfrak{F} \Phi \mathfrak{F}' \Psi(\text{NTDom}) = \mathfrak{F}$
and [*adj-cs-simps*]: *cf-adj-LR-iso* $\mathcal{C} \mathcal{D} \mathcal{G} \mathfrak{F} \Phi \mathfrak{F}' \Psi(\text{NTCod}) = \mathfrak{F}'$
and [*adj-cs-simps*]: *cf-adj-LR-iso* $\mathcal{C} \mathcal{D} \mathcal{G} \mathfrak{F} \Phi \mathfrak{F}' \Psi(\text{NTDGDom}) = \mathcal{C}$
and [*adj-cs-simps*]: *cf-adj-LR-iso* $\mathcal{C} \mathcal{D} \mathcal{G} \mathfrak{F} \Phi \mathfrak{F}' \Psi(\text{NTDGCod}) = \mathcal{D}$
unfolding *cf-adj-LR-iso-def nt-field-simps*
by (*simp-all add: nat-omega-simps*)

lemma *cf-adj-RL-iso-components*:

shows *cf-adj-RL-iso* $\mathcal{C} \mathcal{D} \mathfrak{F} \mathcal{G} \Phi \mathcal{G}' \Psi(\text{NTMap}) =$

(
 $\lambda x \in_o \mathcal{D}(\text{Obj}). \text{THE } f'$.
let
 $\varepsilon = \varepsilon_C \Phi$;
 $\varepsilon' = \varepsilon_C \Psi$;
 $\mathcal{G}x = \mathcal{G}(\text{ObjMap})(x)$;
 $\mathcal{G}'x = \mathcal{G}'(\text{ObjMap})(x)$
in
 $f' : \mathcal{G}'x \mapsto_{\mathcal{C}} \mathcal{G}x \wedge$
 $\varepsilon'(\text{NTMap})(x) = \text{umap-fo } \mathfrak{F} x \mathcal{G}x (\varepsilon(\text{NTMap})(x)) \mathcal{G}'x(\text{ArrVal})(f')$
)
and [*adj-cs-simps*]: *cf-adj-RL-iso* $\mathcal{C} \mathcal{D} \mathfrak{F} \mathcal{G} \Phi \mathcal{G}' \Psi(\text{NTDom}) = \mathcal{G}'$
and [*adj-cs-simps*]: *cf-adj-RL-iso* $\mathcal{C} \mathcal{D} \mathfrak{F} \mathcal{G} \Phi \mathcal{G}' \Psi(\text{NTCod}) = \mathcal{G}$
and [*adj-cs-simps*]: *cf-adj-RL-iso* $\mathcal{C} \mathcal{D} \mathfrak{F} \mathcal{G} \Phi \mathcal{G}' \Psi(\text{NTDGDom}) = \mathcal{D}$
and [*adj-cs-simps*]: *cf-adj-RL-iso* $\mathcal{C} \mathcal{D} \mathfrak{F} \mathcal{G} \Phi \mathcal{G}' \Psi(\text{NTDGCod}) = \mathcal{C}$
unfolding *cf-adj-RL-iso-def nt-field-simps*
by (*simp-all add: nat-omega-simps*)

13.12.2 Natural transformation map

lemma *cf-adj-LR-iso-vsν[adj-cs-intros]*:

vsν (*cf-adj-LR-iso* \mathcal{C} \mathcal{D} \mathfrak{G} \mathfrak{F} Φ \mathfrak{F}' Ψ (*NTMap*))

unfolding *cf-adj-LR-iso-components* **by** *simp*

lemma *cf-adj-RL-iso-vsν[adj-cs-intros]*:

vsν (*cf-adj-RL-iso* \mathcal{C} \mathcal{D} \mathfrak{F} \mathfrak{G} Φ \mathfrak{G}' Ψ (*NTMap*))

unfolding *cf-adj-RL-iso-components* **by** *simp*

lemma *cf-adj-LR-iso-vdomain[adj-cs-simps]*:

\mathcal{D}_o (*cf-adj-LR-iso* \mathcal{C} \mathcal{D} \mathfrak{G} \mathfrak{F} Φ \mathfrak{F}' Ψ (*NTMap*)) = \mathcal{C} (*Obj*)

unfolding *cf-adj-LR-iso-components* **by** *simp*

lemma *cf-adj-RL-iso-vdomain[adj-cs-simps]*:

\mathcal{D}_o (*cf-adj-RL-iso* \mathcal{C} \mathcal{D} \mathfrak{F} \mathfrak{G} Φ \mathfrak{G}' Ψ (*NTMap*)) = \mathcal{D} (*Obj*)

unfolding *cf-adj-RL-iso-components* **by** *simp*

lemma *cf-adj-LR-iso-app*:

fixes \mathcal{C} \mathcal{D} \mathfrak{G} \mathfrak{F} Φ \mathfrak{F}' Ψ

assumes $x \in_o \mathcal{C}$ (*Obj*)

defines $\mathfrak{F}x \equiv \mathfrak{F}$ (*ObjMap*)(x)

and $\mathfrak{F}'x \equiv \mathfrak{F}'$ (*ObjMap*)(x)

and $\eta \equiv \eta_C \Phi$

and $\eta' \equiv \eta_C \Psi$

shows *cf-adj-LR-iso* \mathcal{C} \mathcal{D} \mathfrak{G} \mathfrak{F} Φ \mathfrak{F}' Ψ (*NTMap*)(x) =

(
THE f' .
 $f' : \mathfrak{F}x \mapsto_{\mathcal{D}} \mathfrak{F}'x \wedge$
 $\eta'(\text{NTMap})(x) = \text{umap-of } \mathfrak{G} \ x \ \mathfrak{F}x \ (\eta(\text{NTMap})(x)) \ \mathfrak{F}'x(\text{ArrVal})(f')$
)

using *assms(1)* **unfolding** *cf-adj-LR-iso-components* *assms(2-5)* **by** *simp meson*

lemma *cf-adj-RL-iso-app*:

fixes \mathcal{C} \mathcal{D} \mathfrak{F} \mathfrak{G} Φ \mathfrak{G}' Ψ

assumes $x \in_o \mathcal{D}$ (*Obj*)

defines $\mathfrak{G}x \equiv \mathfrak{G}$ (*ObjMap*)(x)

and $\mathfrak{G}'x \equiv \mathfrak{G}'$ (*ObjMap*)(x)

and $\varepsilon \equiv \varepsilon_C \Phi$

and $\varepsilon' \equiv \varepsilon_C \Psi$

shows *cf-adj-RL-iso* \mathcal{C} \mathcal{D} \mathfrak{F} \mathfrak{G} Φ \mathfrak{G}' Ψ (*NTMap*)(x) =

(
THE f' .
 $f' : \mathfrak{G}'x \mapsto_{\mathcal{C}} \mathfrak{G}x \wedge$
 $\varepsilon'(\text{NTMap})(x) = \text{umap-fo } \mathfrak{F} \ x \ \mathfrak{G}x \ (\varepsilon(\text{NTMap})(x)) \ \mathfrak{G}'x(\text{ArrVal})(f')$
)

using *assms(1)* **unfolding** *cf-adj-RL-iso-components* *assms(2-5)* *Let-def* **by** *simp*

lemma *cf-adj-LR-iso-app-unique*:

fixes \mathcal{C} \mathcal{D} \mathfrak{G} \mathfrak{F} Φ \mathfrak{F}' Ψ

assumes $\Phi : \mathfrak{F} \rightleftharpoons_{CF} \mathfrak{G} : \mathcal{C} \rightleftharpoons_{C\alpha} \mathcal{D}$

and $\Psi : \mathfrak{F}' \rightleftharpoons_{CF} \mathfrak{G} : \mathcal{C} \rightleftharpoons_{C\alpha} \mathcal{D}$

and $x \in_o \mathcal{C}$ (*Obj*)

defines $\mathfrak{F}x \equiv \mathfrak{F}$ (*ObjMap*)(x)

and $\mathfrak{F}'x \equiv \mathfrak{F}'$ (*ObjMap*)(x)

and $\eta \equiv \eta_C \Phi$

and $\eta' \equiv \eta_C \Psi$

and $f \equiv \text{cf-adj-LR-iso } \mathcal{C} \ \mathcal{D} \ \mathfrak{G} \ \mathfrak{F} \ \Phi \ \mathfrak{F}' \ \Psi(\text{NTMap})(x)$

shows

$\exists !f'$.
 $f' : \mathfrak{F}x \mapsto_{\mathfrak{D}} \mathfrak{F}'x \wedge$
 $\eta'(\downarrow NTMap)(\downarrow x) = \text{umap-of } \mathfrak{G} \ x \ \mathfrak{F}x \ (\eta(\downarrow NTMap)(\downarrow x)) \ \mathfrak{F}'x(\downarrow ArrVal)(\downarrow f')$
 $f : \mathfrak{F}x \mapsto_{\text{iso}\mathfrak{D}} \mathfrak{F}'x$
 $\eta'(\downarrow NTMap)(\downarrow x) = \text{umap-of } \mathfrak{G} \ x \ \mathfrak{F}x \ (\eta(\downarrow NTMap)(\downarrow x)) \ \mathfrak{F}'x(\downarrow ArrVal)(\downarrow f)$

proof-

interpret Φ : *is-cf-adjunction* $\alpha \ \mathfrak{C} \ \mathfrak{D} \ \mathfrak{F} \ \mathfrak{G} \ \Phi$ **by** (*rule assms(1)*)

interpret Ψ : *is-cf-adjunction* $\alpha \ \mathfrak{C} \ \mathfrak{D} \ \mathfrak{F}' \ \mathfrak{G} \ \Psi$ **by** (*rule assms(2)*)

note $\mathfrak{F}a\text{-}\eta =$

is-cf-adjunction.cf-adjunction-unit-component-is-ua-of
OF assms(1) assms(3), folded assms(4-8)
 $]$

note $\mathfrak{F}'a\text{-}\eta =$

is-cf-adjunction.cf-adjunction-unit-component-is-ua-of
OF assms(2) assms(3), folded assms(4-8)
 $]$

from

is-functor.cf-universal-arrow-of-unique
OF Φ .RL.is-functor-axioms $\mathfrak{F}a\text{-}\eta \ \mathfrak{F}'a\text{-}\eta$, folded assms(4-8)
 $]$

obtain f'

where $f' : f' : \mathfrak{F}x \mapsto_{\mathfrak{D}} \mathfrak{F}'x$

and η' -def:

$\eta'(\downarrow NTMap)(\downarrow x) = \text{umap-of } \mathfrak{G} \ x \ \mathfrak{F}x \ (\eta(\downarrow NTMap)(\downarrow x)) \ \mathfrak{F}'x(\downarrow ArrVal)(\downarrow f')$

and *unique-f'*:

$[[$
 $f'' : \mathfrak{F}x \mapsto_{\mathfrak{D}} \mathfrak{F}'x;$
 $\eta'(\downarrow NTMap)(\downarrow x) = \text{umap-of } \mathfrak{G} \ x \ \mathfrak{F}x \ (\eta(\downarrow NTMap)(\downarrow x)) \ \mathfrak{F}'x(\downarrow ArrVal)(\downarrow f'')$
 $]] \implies f'' = f'$

for f''

by *metis*

show *unique-f'*: $\exists !f'$.

$f' : \mathfrak{F}x \mapsto_{\mathfrak{D}} \mathfrak{F}'x \wedge$

$\eta'(\downarrow NTMap)(\downarrow x) = \text{umap-of } \mathfrak{G} \ x \ \mathfrak{F}x \ (\eta(\downarrow NTMap)(\downarrow x)) \ \mathfrak{F}'x(\downarrow ArrVal)(\downarrow f')$

by

(
rule is-functor.cf-universal-arrow-of-unique
OF Φ .RL.is-functor-axioms $\mathfrak{F}a\text{-}\eta \ \mathfrak{F}'a\text{-}\eta$, folded assms(4-8)
 $]$
 $)$

from

theD

$[$
OF unique-f' cf-adj-LR-iso-app
OF assms(3), of $\mathfrak{D} \ \mathfrak{G} \ \mathfrak{F} \ \Phi \ \mathfrak{F}' \ \Psi$, folded assms(4-8)
 $]$
 $]$

have $f : f : \mathfrak{F}x \mapsto_{\mathfrak{D}} \mathfrak{F}'x$

and $\eta' : \eta'(\downarrow NTMap)(\downarrow x) = \text{umap-of } \mathfrak{G} \ x \ \mathfrak{F}x \ (\eta(\downarrow NTMap)(\downarrow x)) \ \mathfrak{F}'x(\downarrow ArrVal)(\downarrow f)$

by *simp-all*

show $\eta'(\downarrow NTMap)(\downarrow x) = \text{umap-of } \mathfrak{G} \ x \ \mathfrak{F}x \ (\eta(\downarrow NTMap)(\downarrow x)) \ \mathfrak{F}'x(\downarrow ArrVal)(\downarrow f)$ **by** (*rule η'*)

show $f : \mathfrak{F}x \mapsto_{\text{iso}\mathfrak{D}} \mathfrak{F}'x$

by

(
rule
is-functor.cf-universal-arrow-of-is-iso-arr
OF Φ .RL.is-functor-axioms $\mathfrak{F}a\text{-}\eta \ \mathfrak{F}'a\text{-}\eta \ f \ \eta'$
 $)$

)
)
 qed

13.12.3 Main results

lemma *cf-adj-LR-iso-is-iso-functor*:

— See Corollary 1 in Chapter IV-1 in [9].

assumes $\Phi : \mathfrak{F} \rightleftharpoons_{CF} \mathfrak{G} : \mathcal{C} \rightleftharpoons_{C\alpha} \mathcal{D}$ **and** $\Psi : \mathfrak{F}' \rightleftharpoons_{CF} \mathfrak{G} : \mathcal{C} \rightleftharpoons_{C\alpha} \mathcal{D}$

shows $\exists ! \vartheta. \vartheta : \mathfrak{F} \mapsto_{CF} \mathfrak{F}' : \mathcal{C} \mapsto_{C\alpha} \mathcal{D} \wedge \eta_C \Psi = (\mathfrak{G} \circ_{CF-NTCF} \vartheta) \cdot_{NTCF} \eta_C \Phi$

and *cf-adj-LR-iso* $\mathcal{C} \mathcal{D} \mathfrak{G} \mathfrak{F} \Phi \mathfrak{F}' \Psi : \mathfrak{F} \mapsto_{CF.iso} \mathfrak{F}' : \mathcal{C} \mapsto_{C\alpha} \mathcal{D}$

and $\eta_C \Psi = (\mathfrak{G} \circ_{CF-NTCF} \text{cf-adj-LR-iso } \mathcal{C} \mathcal{D} \mathfrak{G} \mathfrak{F} \Phi \mathfrak{F}' \Psi) \cdot_{NTCF} \eta_C \Phi$

proof–

interpret Φ : *is-cf-adjunction* $\alpha \mathcal{C} \mathcal{D} \mathfrak{F} \mathfrak{G} \Phi$ **by** (*rule assms(1)*)

interpret Ψ : *is-cf-adjunction* $\alpha \mathcal{C} \mathcal{D} \mathfrak{F}' \mathfrak{G} \Psi$ **by** (*rule assms(2)*)

let $? \eta = \langle \eta_C \Phi \rangle$

let $? \eta' = \langle \eta_C \Psi \rangle$

let $? \Phi \Psi = \langle \text{cf-adj-LR-iso } \mathcal{C} \mathcal{D} \mathfrak{G} \mathfrak{F} \Phi \mathfrak{F}' \Psi \rangle$

show $\mathfrak{F}' \Psi : ? \Phi \Psi : \mathfrak{F} \mapsto_{CF.iso} \mathfrak{F}' : \mathcal{C} \mapsto_{C\alpha} \mathcal{D}$

proof(*intro is-iso-ntcfI is-ntcfI'*)

show *vfsequence* $? \Phi \Psi$ **unfolding** *cf-adj-LR-iso-def* **by** *auto*

show *vcard* $? \Phi \Psi = 5_N$

unfolding *cf-adj-LR-iso-def* **by** (*simp add: nat-omega-simps*)

show $? \Phi \Psi (\downarrow NTMap) (\downarrow a) : \mathfrak{F} (\downarrow ObjMap) (\downarrow a) \mapsto_{\mathcal{D}} \mathfrak{F}' (\downarrow ObjMap) (\downarrow a)$

if $a \in_{\circ} \mathcal{C} (\downarrow Obj)$ **for** a

using *cf-adj-LR-iso-app-unique(2)*[*OF assms that*] **by** *auto*

show $? \Phi \Psi (\downarrow NTMap) (\downarrow b) \circ_{A\mathcal{D}} \mathfrak{F} (\downarrow ArrMap) (\downarrow f) = \mathfrak{F}' (\downarrow ArrMap) (\downarrow f) \circ_{A\mathcal{D}} ? \Phi \Psi (\downarrow NTMap) (\downarrow a)$

if $f : a \mapsto_{\mathcal{C}} b$ **for** $a \ b \ f$

proof–

from *that* **have** $a : a \in_{\circ} \mathcal{C} (\downarrow Obj)$ **and** $b : b \in_{\circ} \mathcal{C} (\downarrow Obj)$ **by** *auto*

note *unique-a* = *cf-adj-LR-iso-app-unique*[*OF assms a*]

note *unique-b* = *cf-adj-LR-iso-app-unique*[*OF assms b*]

from *unique-a(2)* **have** *a-is-arr*:

$? \Phi \Psi (\downarrow NTMap) (\downarrow a) : \mathfrak{F} (\downarrow ObjMap) (\downarrow a) \mapsto_{\mathcal{D}} \mathfrak{F}' (\downarrow ObjMap) (\downarrow a)$

by *auto*

from *unique-b(2)* **have** *b-is-arr*:

$? \Phi \Psi (\downarrow NTMap) (\downarrow b) : \mathfrak{F} (\downarrow ObjMap) (\downarrow b) \mapsto_{\mathcal{D}} \mathfrak{F}' (\downarrow ObjMap) (\downarrow b)$

by *auto*

interpret η : *is-ntcf* $\alpha \mathcal{C} \mathcal{C} \langle \text{cf-id } \mathcal{C} \rangle \langle \mathfrak{G} \circ_{CF} \mathfrak{F} \rangle ? \eta$

by (*rule* Φ .*cf-adjunction-unit-is-ntcf*)

interpret η' : *is-ntcf* $\alpha \mathcal{C} \mathcal{C} \langle \text{cf-id } \mathcal{C} \rangle \langle \mathfrak{G} \circ_{CF} \mathfrak{F}' \rangle ? \eta'$

by (*rule* Ψ .*cf-adjunction-unit-is-ntcf*)

from *unique-a(3)* *a-is-arr* $a \ b$ **have** η' -*a-def*:

$? \eta' (\downarrow NTMap) (\downarrow a) = \mathfrak{G} (\downarrow ArrMap) (\downarrow ? \Phi \Psi (\downarrow NTMap) (\downarrow a)) \circ_{A\mathcal{C}} ? \eta (\downarrow NTMap) (\downarrow a)$

by

(

cs-prems **cs-shallow**

cs-simp: *cat-cs-simps* **cs-intro**: *adj-cs-intros cat-cs-intros*

)

from *unique-b*(β) *b-is-arr* a b **have** η' -*b-def*:
 $?\eta'(\text{NTMap})(b) = \mathfrak{G}(\text{ArrMap})(\text{?}\Phi\Psi(\text{NTMap})(b)) \circ_{A\mathfrak{C}} \text{?}\eta(\text{NTMap})(b)$
by
 (
 cs-prems **cs-shallow**
 cs-simp: *cat-cs-simps* **cs-intro**: *adj-cs-intros* *cat-cs-intros*
)

from *that a b a-is-arr* **have**
 $\mathfrak{G}(\text{ArrMap})(\text{?}\mathfrak{F}'(\text{ArrMap})(f)) \circ_{A\mathfrak{C}}$
 $(\mathfrak{G}(\text{ArrMap})(\text{?}\Phi\Psi(\text{NTMap})(a)) \circ_{A\mathfrak{C}} \text{?}\eta(\text{NTMap})(a)) =$
 $\mathfrak{G}(\text{ArrMap})(\text{?}\mathfrak{F}'(\text{ArrMap})(f)) \circ_{A\mathfrak{C}} \text{?}\eta'(\text{NTMap})(a)$
by
 (
 cs-concl **cs-shallow**
 cs-simp: *cat-cs-simps* η' -*a-def* **cs-intro**: *cat-cs-intros*
)

also from η' .*ntcf-Comp-commute*[*OF that, symmetric*] *that a b* **have**
 $\dots = \text{?}\eta'(\text{NTMap})(b) \circ_{A\mathfrak{C}} f$
by (*cs-prems* **cs-shallow** **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)

also from *that a b b-is-arr* **have**
 $\dots = \mathfrak{G}(\text{ArrMap})(\text{?}\Phi\Psi(\text{NTMap})(b)) \circ_{A\mathfrak{C}}$
 $(\text{?}\eta(\text{NTMap})(b) \circ_{A\mathfrak{C}} f)$
by
 (
 cs-concl **cs-shallow**
 cs-simp: *cat-cs-simps* η' -*b-def* **cs-intro**: *cat-cs-intros*
)

also from *that* **have**
 $\dots = \mathfrak{G}(\text{ArrMap})(\text{?}\Phi\Psi(\text{NTMap})(b)) \circ_{A\mathfrak{C}}$
 $((\mathfrak{G} \circ_{CF} \mathfrak{F})(\text{ArrMap})(f) \circ_{A\mathfrak{C}} \text{?}\eta(\text{NTMap})(a))$
unfolding η' .*ntcf-Comp-commute*[*OF that, symmetric*]
by
 (
 cs-concl **cs-shallow**
 cs-simp: *cat-cs-simps* η' -*b-def* **cs-intro**: *cat-cs-intros*
)

also from *that b-is-arr* **have**
 $\dots = \mathfrak{G}(\text{ArrMap})(\text{?}\Phi\Psi(\text{NTMap})(b)) \circ_{A\mathfrak{C}}$
 $(\mathfrak{G}(\text{ArrMap})(\text{?}\mathfrak{F}'(\text{ArrMap})(f)) \circ_{A\mathfrak{C}} \text{?}\eta(\text{NTMap})(a))$
by (*cs-concl* **cs-shallow** **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)

finally have [*cat-cs-simps*]:
 $\mathfrak{G}(\text{ArrMap})(\text{?}\mathfrak{F}'(\text{ArrMap})(f)) \circ_{A\mathfrak{C}} (\mathfrak{G}(\text{ArrMap})(\text{?}\Phi\Psi(\text{NTMap})(a)) \circ_{A\mathfrak{C}} \text{?}\eta(\text{NTMap})(a)) =$
 $\mathfrak{G}(\text{ArrMap})(\text{?}\Phi\Psi(\text{NTMap})(b)) \circ_{A\mathfrak{C}}$
 $(\mathfrak{G}(\text{ArrMap})(\text{?}\mathfrak{F}'(\text{ArrMap})(f)) \circ_{A\mathfrak{C}} \text{?}\eta(\text{NTMap})(a))$
by *simp*

note *unique-f-a = is-functor.universal-arrow-ofD*
 [
 OF
 Φ .*RL.is-functor-axioms*
 Φ .*cf-adjunction-unit-component-is-ua-of*[*OF a*]
]

from *that a b a-is-arr b-is-arr* **have** $\mathfrak{G}\mathfrak{F}'\eta$ a :
 $\mathfrak{G}(\text{ArrMap})(\text{?}\mathfrak{F}'(\text{ArrMap})(f)) \circ_{A\mathfrak{C}} \text{?}\eta'(\text{NTMap})(a) :$
 $a \mapsto_{\mathfrak{C}} \mathfrak{G}(\text{ObjMap})(\text{?}\mathfrak{F}'(\text{ObjMap})(b))$

by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

from b have $\mathfrak{F}'b: \mathfrak{F}'(\text{ObjMap})(b) \in_0 \mathfrak{D}(\text{Obj})$

by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

from *unique-f-a*(β)[*OF* $\mathfrak{F}'b \mathfrak{G}\mathfrak{F}'\eta a$] obtain f'

where $f': f' : \mathfrak{F}'(\text{ObjMap})(a) \mapsto_{\mathfrak{D}} \mathfrak{F}'(\text{ObjMap})(b)$

and $\eta a: \mathfrak{G}(\text{ArrMap})(\mathfrak{F}'(\text{ArrMap})(f)) \circ_{A\mathfrak{C}} ?\eta'(\text{NTMap})(a) =$

umap-of $\mathfrak{G} a (\mathfrak{F}'(\text{ObjMap})(a)) (?\eta'(\text{NTMap})(a)) (\mathfrak{F}'(\text{ObjMap})(b))(\text{ArrVal})(f')$

and *unique-f'*:

[[

$f'' : \mathfrak{F}'(\text{ObjMap})(a) \mapsto_{\mathfrak{D}} \mathfrak{F}'(\text{ObjMap})(b);$

$\mathfrak{G}(\text{ArrMap})(\mathfrak{F}'(\text{ArrMap})(f)) \circ_{A\mathfrak{C}} ?\eta'(\text{NTMap})(a) =$

umap-of $\mathfrak{G} a (\mathfrak{F}'(\text{ObjMap})(a)) (?\eta'(\text{NTMap})(a)) (\mathfrak{F}'(\text{ObjMap})(b))(\text{ArrVal})(f'')$

]] $\implies f'' = f'$

for f''

by *metis*

have $?\Phi\Psi(\text{NTMap})(b) \circ_{A\mathfrak{D}} \mathfrak{F}'(\text{ArrMap})(f) = f'$

by (*rule unique-f', insert a b a-is-arr b-is-arr that*)

(

cs-concl cs-shallow

cs-simp: η' -*a-def* *cat-cs-simps cs-intro: cat-cs-intros*

)

moreover have $\mathfrak{F}'(\text{ArrMap})(f) \circ_{A\mathfrak{D}} ?\Phi\Psi(\text{NTMap})(a) = f'$

by (*rule unique-f', insert a b a-is-arr b-is-arr that*)

(

cs-concl cs-shallow

cs-simp: η' -*a-def* *cat-cs-simps cs-intro: cat-cs-intros*

)

ultimately show *?thesis* by *simp*

qed

qed

(

auto

intro: cat-cs-intros adj-cs-intros

simp: adj-cs-simps cf-adj-LR-iso-app-unique(2)[OF assms]

)

interpret $\mathfrak{F}'\Psi: \text{is-iso-ntcf } \alpha \mathfrak{C} \mathfrak{D} \mathfrak{F} \mathfrak{F}' \langle ?\Phi\Psi \rangle$ by (*rule* $\mathfrak{F}'\Psi$)

show η' -*def*: $?\eta' = \mathfrak{G} \circ_{CF-NTCF} ?\Phi\Psi \cdot_{NTCF} \eta_C \Phi$

proof(*rule ntcf-eqI*)

have *dom-lhs*: $\mathcal{D}_0 (?\eta'(\text{NTMap})) = \mathfrak{C}(\text{Obj})$

by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: adj-cs-intros*)

have *dom-rhs*: $\mathcal{D}_0 ((\mathfrak{G} \circ_{CF-NTCF} ?\Phi\Psi \cdot_{NTCF} \eta_C \Phi)(\text{NTMap})) = \mathfrak{C}(\text{Obj})$

by

(

cs-concl cs-shallow

cs-simp: *cat-cs-simps cs-intro: adj-cs-intros cat-cs-intros*

)

show $?\eta'(\text{NTMap}) = (\mathfrak{G} \circ_{CF-NTCF} ?\Phi\Psi \cdot_{NTCF} \eta_C \Phi)(\text{NTMap})$

proof(*rule vsv-eqI, unfold dom-lhs dom-rhs*)

fix a **assume** *prems*: $a \in_0 \mathfrak{C}(\text{Obj})$

note *unique-a* = *cf-adj-LR-iso-app-unique*[*OF assms prems*]

from *unique-a*(β) **have** *a-is-arr*:

$?\Phi\Psi(\text{NTMap})(a) : \mathfrak{F}'(\text{ObjMap})(a) \mapsto_{\mathfrak{D}} \mathfrak{F}'(\text{ObjMap})(a)$

by *auto*

interpret η : *is-ntcf* α \mathfrak{C} \mathfrak{C} $\langle cf-id \mathfrak{C} \rangle$ $\langle \mathfrak{G} \circ_{CF} \mathfrak{F} \rangle$ $? \eta$
by (*rule* Φ .*cf-adjunction-unit-is-ntcf*)
from *unique-a*(\mathfrak{A}) *a-is-arr* *prems* **have** η' -*a-def*:
 $? \eta'(\downarrow NTMap)(\downarrow a) = \mathfrak{G}(\downarrow ArrMap)(\downarrow ? \Phi \Psi(\downarrow NTMap)(\downarrow a)) \circ_{A\mathfrak{C}} \eta_C \Phi(\downarrow NTMap)(\downarrow a)$
by
(

cs-prems **cs-shallow**
cs-simp: *cat-cs-simps* **cs-intro**: *adj-cs-intros* *cat-cs-intros*

)

from *prems* *a-is-arr* **show**
 $? \eta'(\downarrow NTMap)(\downarrow a) = (\mathfrak{G} \circ_{CF-NTCF} ? \Phi \Psi \cdot_{NTCF} ? \eta)(\downarrow NTMap)(\downarrow a)$
by
(

cs-concl **cs-shallow**
cs-simp: η' -*a-def* *cat-cs-simps* **cs-intro**: *cat-cs-intros*

)

qed (*auto intro*: *cat-cs-intros* *adj-cs-intros*)

qed
(

cs-concl **cs-shallow**
cs-simp: *cat-cs-simps* **cs-intro**: *adj-cs-intros* *cat-cs-intros*

)+

show $\exists ! \vartheta$. $\vartheta : \mathfrak{F} \mapsto_{CF} \mathfrak{F}' : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{D} \wedge ? \eta' = (\mathfrak{G} \circ_{CF-NTCF} \vartheta) \cdot_{NTCF} ? \eta$
proof(*intro ex1I conjI*; (*elim conjE*)?)
from $\mathfrak{F}' \Psi$ **show** $? \Phi \Psi : \mathfrak{F} \mapsto_{CF} \mathfrak{F}' : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{D}$ **by** *auto*
show $? \eta' = \mathfrak{G} \circ_{CF-NTCF} ? \Phi \Psi \cdot_{NTCF} \eta_C \Phi$ **by** (*rule* η' -*def*)
fix ϑ **assume** *prems*:
 $\vartheta : \mathfrak{F} \mapsto_{CF} \mathfrak{F}' : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{D}$
 $? \eta' = \mathfrak{G} \circ_{CF-NTCF} \vartheta \cdot_{NTCF} \eta_C \Phi$
interpret ϑ : *is-ntcf* α \mathfrak{C} \mathfrak{D} \mathfrak{F} \mathfrak{F}' ϑ **by** (*rule* *prems*(1))
from *prems* **have** η' -*a*:
 $? \eta'(\downarrow NTMap)(\downarrow a) = (\mathfrak{G} \circ_{CF-NTCF} \vartheta \cdot_{NTCF} \eta_C \Phi)(\downarrow NTMap)(\downarrow a)$
for a
by *simp*
have $\eta'_a : \eta_C \Psi(\downarrow NTMap)(\downarrow a) =$
 $\mathfrak{G}(\downarrow ArrMap)(\downarrow \vartheta(\downarrow NTMap)(\downarrow a)) \circ_{A\mathfrak{C}} \eta_C \Phi(\downarrow NTMap)(\downarrow a)$
if $a \in_o \mathfrak{C}(\downarrow Obj)$ **for** a
using η' -*a*[**where** $a=a$] **that**
by
(

cs-prems **cs-shallow**
cs-simp: *cat-cs-simps* **cs-intro**: *adj-cs-intros* *cat-cs-intros*

)

show $\vartheta = ? \Phi \Psi$
proof(*rule* *ntcf-eqI*)
have *dom-lhs*: $\mathcal{D}_o(\vartheta(\downarrow NTMap)) = \mathfrak{C}(\downarrow Obj)$
by (*cs-concl* **cs-shallow** **cs-simp**: *cat-cs-simps*)
have *dom-rhs*: $\mathcal{D}_o(? \Phi \Psi(\downarrow NTMap)) = \mathfrak{C}(\downarrow Obj)$
by (*cs-concl* **cs-shallow** **cs-simp**: *cat-cs-simps*)
show $\vartheta(\downarrow NTMap) = ? \Phi \Psi(\downarrow NTMap)$
proof(*rule* *vsu-eqI*, *unfold dom-lhs dom-rhs*)
fix a **assume** *prems'*: $a \in_o \mathfrak{C}(\downarrow Obj)$
let $?uof = \langle umap-of \mathfrak{G} a (\mathfrak{F}(\downarrow ObjMap)(\downarrow a)) (? \eta(\downarrow NTMap)(\downarrow a)) (\mathfrak{F}'(\downarrow ObjMap)(\downarrow a)) \rangle$
from *cf-adj-LR-iso-app-unique*[*OF assms prems'*] **obtain** f'
where $f' : f' : \mathfrak{F}(\downarrow ObjMap)(\downarrow a) \mapsto_{\mathfrak{D}} \mathfrak{F}'(\downarrow ObjMap)(\downarrow a)$
and η -*def*: $? \eta'(\downarrow NTMap)(\downarrow a) = ?uof(\downarrow ArrVal)(f')$
and *unique-f'*: $\wedge f''$.

$$\llbracket f'' : \mathfrak{F}(\text{ObjMap})(\text{!}a) \mapsto_{\mathfrak{D}} \mathfrak{F}'(\text{ObjMap})(\text{!}a);$$

$$\text{?}\eta'(\text{NTMap})(\text{!}a) = \text{?}uof(\text{ArrVal})(\text{!}f') \rrbracket$$

$$\implies f'' = f'$$

by *metis*
from *prems'* **have** $\vartheta a : \vartheta(\text{NTMap})(\text{!}a) : \mathfrak{F}(\text{ObjMap})(\text{!}a) \mapsto_{\mathfrak{D}} \mathfrak{F}'(\text{ObjMap})(\text{!}a)$
by (*cs-concl cs-shallow cs-intro: cat-cs-intros*)
from $\eta\text{-def } f'$ *prems'* **have**
 $\eta_C \Psi(\text{NTMap})(\text{!}a) = \mathfrak{G}(\text{ArrMap})(\text{!}f') \circ_{A\mathfrak{C}} \eta_C \Phi(\text{NTMap})(\text{!}a)$
by
(*cs-prems*
cs-simp: *cat-cs-simps cs-intro: adj-cs-intros cat-cs-intros*
))
from *prems'* **have** $\eta_C \Psi(\text{NTMap})(\text{!}a) = \text{?}uof(\text{ArrVal})(\vartheta(\text{NTMap})(\text{!}a))$
by
(*cs-concl*
cs-simp: *cat-cs-simps* $\eta'a[\text{OF } \text{prems}']$
cs-intro: *adj-cs-intros cat-cs-intros*
))
from *unique-f'* [*OF* ϑa *this*] **have** $\vartheta a : \vartheta(\text{NTMap})(\text{!}a) = f'$.
from *prems'* **have** $\Psi a :$
 $\text{?}\Phi\Psi(\text{NTMap})(\text{!}a) : \mathfrak{F}(\text{ObjMap})(\text{!}a) \mapsto_{\mathfrak{D}} \mathfrak{F}'(\text{ObjMap})(\text{!}a)$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)
from *prems'* **have** $\eta_C \Psi(\text{NTMap})(\text{!}a) = \text{?}uof(\text{ArrVal})(\text{?}\Phi\Psi(\text{NTMap})(\text{!}a))$
by
(*cs-concl*
cs-simp: *cf-adj-LR-iso-app-unique*(β)[*OF assms*] *cat-cs-simps*
cs-intro: *adj-cs-intros cat-cs-intros*
))
from *unique-f'* [*OF* Ψa *this*] **have** $\mathfrak{F}'\Psi\text{-def} : \text{?}\Phi\Psi(\text{NTMap})(\text{!}a) = f'$.
show $\vartheta(\text{NTMap})(\text{!}a) = \text{?}\Phi\Psi(\text{NTMap})(\text{!}a)$ **unfolding** ϑa $\mathfrak{F}'\Psi\text{-def}$..
qed auto
qed (*cs-concl cs-shallow cs-intro: cat-cs-intros*)+
qed

qed

lemma *op-ntcf-cf-adj-RL-iso*[*cat-op-simps*]:

assumes $\Phi : \mathfrak{F} \rightleftharpoons_{CF} \mathfrak{G} : \mathfrak{C} \rightleftharpoons_{C\alpha} \mathfrak{D}$

and $\Psi : \mathfrak{F} \rightleftharpoons_{CF} \mathfrak{G}' : \mathfrak{C} \rightleftharpoons_{C\alpha} \mathfrak{D}$

defines $op\text{-}\mathfrak{D} \equiv op\text{-}cat \ \mathfrak{D}$

and $op\text{-}\mathfrak{C} \equiv op\text{-}cat \ \mathfrak{C}$

and $op\text{-}\mathfrak{F} \equiv op\text{-}cf \ \mathfrak{F}$

and $op\text{-}\mathfrak{G} \equiv op\text{-}cf \ \mathfrak{G}$

and $op\text{-}\Phi \equiv op\text{-}cf\text{-}adj \ \Phi$

and $op\text{-}\mathfrak{G}' \equiv op\text{-}cf \ \mathfrak{G}'$

and $op\text{-}\Psi \equiv op\text{-}cf\text{-}adj \ \Psi$

shows

$op\text{-}ntcf \ (cf\text{-}adj\text{-}RL\text{-}iso \ \mathfrak{C} \ \mathfrak{D} \ \mathfrak{F} \ \mathfrak{G} \ \Phi \ \mathfrak{G}' \ \Psi) =$

$cf\text{-}adj\text{-}LR\text{-}iso \ op\text{-}\mathfrak{D} \ op\text{-}\mathfrak{C} \ op\text{-}\mathfrak{F} \ op\text{-}\mathfrak{G} \ op\text{-}\Phi \ op\text{-}\mathfrak{G}' \ op\text{-}\Psi$

proof-

interpret Φ : *is-cf-adjunction* $\alpha \ \mathfrak{C} \ \mathfrak{D} \ \mathfrak{F} \ \mathfrak{G} \ \Phi$ **by** (*rule assms*(1))

interpret Ψ : *is-cf-adjunction* $\alpha \ \mathfrak{C} \ \mathfrak{D} \ \mathfrak{F} \ \mathfrak{G}' \ \Psi$ **by** (*rule assms*(2))

interpret ε : *is-ntcf* $\alpha \ \mathfrak{D} \ \mathfrak{D} \ \langle \mathfrak{F} \circ_{CF} \mathfrak{G} \rangle \ \langle cf\text{-}id \ \mathfrak{D} \rangle \ \langle \varepsilon_C \ \Phi \rangle$

by (*rule* Φ .*cf-adjunction-counit-is-ntcf*)

have dom-lhs: $\mathcal{D}_o (op\text{-ntcf} (cf\text{-adj-RL-iso } \mathcal{C} \mathcal{D} \mathfrak{F} \mathfrak{G} \Phi \mathfrak{G}' \Psi)) = 5_N$
unfolding *op-ntcf-def* **by** (*simp add: nat-omega-simps*)
show *?thesis*
proof(*rule vsv-eqI, unfold dom-lhs*)
fix *a* **assume** *prems: a* $\in_o 5_N$
then have *a* $\in_o 5_N$ **unfolding** *dom-lhs* **by** *simp*
then show
 $op\text{-ntcf} (cf\text{-adj-RL-iso } \mathcal{C} \mathcal{D} \mathfrak{F} \mathfrak{G} \Phi \mathfrak{G}' \Psi)(|a|) =$
 $cf\text{-adj-LR-iso } op\text{-}\mathcal{D} \text{ } op\text{-}\mathcal{C} \text{ } op\text{-}\mathfrak{F} \text{ } op\text{-}\mathfrak{G} \text{ } op\text{-}\Phi \text{ } op\text{-}\mathfrak{G}' \text{ } op\text{-}\Psi(|a|)$
by
(

elim-in-numeral,
fold nt-field-simps,
unfold
cf-adj-LR-iso-components
op-ntcf-components
cf-adj-RL-iso-components
Let-def
 $\Phi.cf\text{-adjunction-unit-NTMap-op}$
 $\Psi.cf\text{-adjunction-unit-NTMap-op}$
assms(3-9)
cat-op-simps
)

simp-all
qed (*auto simp: op-ntcf-def cf-adj-LR-iso-def nat-omega-simps*)
qed

lemma *op-ntcf-cf-adj-LR-iso[cat-op-simps]*:
assumes $\Phi : \mathfrak{F} \rightleftharpoons_{CF} \mathfrak{G} : \mathcal{C} \rightleftharpoons_{C\alpha} \mathcal{D}$ **and** $\Psi : \mathfrak{F}' \rightleftharpoons_{CF} \mathfrak{G}' : \mathcal{C} \rightleftharpoons_{C\alpha} \mathcal{D}$
defines $op\text{-}\mathcal{D} \equiv op\text{-}cat \ \mathcal{D}$
and $op\text{-}\mathcal{C} \equiv op\text{-}cat \ \mathcal{C}$
and $op\text{-}\mathfrak{F} \equiv op\text{-}cf \ \mathfrak{F}$
and $op\text{-}\mathfrak{G} \equiv op\text{-}cf \ \mathfrak{G}$
and $op\text{-}\Phi \equiv op\text{-}cf\text{-adj} \ \Phi$
and $op\text{-}\mathfrak{F}' \equiv op\text{-}cf \ \mathfrak{F}'$
and $op\text{-}\Psi \equiv op\text{-}cf\text{-adj} \ \Psi$
shows
 $op\text{-ntcf} (cf\text{-adj-LR-iso } \mathcal{C} \mathcal{D} \mathfrak{G} \mathfrak{F} \Phi \mathfrak{F}' \Psi) =$
 $cf\text{-adj-LR-iso } op\text{-}\mathcal{D} \text{ } op\text{-}\mathcal{C} \text{ } op\text{-}\mathfrak{G} \text{ } op\text{-}\mathfrak{F} \text{ } op\text{-}\Phi \text{ } op\text{-}\mathfrak{F}' \text{ } op\text{-}\Psi$

proof-
interpret Φ : *is-cf-adjunction* $\alpha \ \mathcal{C} \ \mathcal{D} \ \mathfrak{F} \ \mathfrak{G} \ \Phi$ **by** (*rule assms(1)*)
interpret Ψ : *is-cf-adjunction* $\alpha \ \mathcal{C} \ \mathcal{D} \ \mathfrak{F}' \ \mathfrak{G}' \ \Psi$ **by** (*rule assms(2)*)
interpret ε : *is-ntcf* $\alpha \ \mathcal{D} \ \mathcal{D} \ \langle \mathfrak{F} \circ_{CF} \mathfrak{G} \rangle \ \langle cf\text{-id } \mathcal{D} \rangle \ \langle \varepsilon_C \ \Phi \rangle$
by (*rule* $\Phi.cf\text{-adjunction-counit-is-ntcf}$)
have dom-lhs: $\mathcal{D}_o (op\text{-ntcf} (cf\text{-adj-LR-iso } \mathcal{C} \mathcal{D} \mathfrak{G} \mathfrak{F} \Phi \mathfrak{F}' \Psi)) = 5_N$
unfolding *op-ntcf-def* **by** (*simp add: nat-omega-simps*)
show *?thesis*
proof(*rule vsv-eqI, unfold dom-lhs*)
fix *a* **assume** *prems: a* $\in_o 5_N$
then show
 $op\text{-ntcf} (cf\text{-adj-LR-iso } \mathcal{C} \mathcal{D} \mathfrak{G} \mathfrak{F} \Phi \mathfrak{F}' \Psi)(|a|) =$
 $cf\text{-adj-LR-iso } op\text{-}\mathcal{D} \text{ } op\text{-}\mathcal{C} \text{ } op\text{-}\mathfrak{G} \text{ } op\text{-}\mathfrak{F} \text{ } op\text{-}\Phi \text{ } op\text{-}\mathfrak{F}' \text{ } op\text{-}\Psi(|a|)$
by
(

elim-in-numeral,
use nothing in
<
fold nt-field-simps,

```

    unfold
      cf-adj-LR-iso-components
      op-ntcf-components
      cf-adj-RL-iso-components
      Let-def
       $\Phi.op-ntcf-cf-adjunction-unit[symmetric]$ 
       $\Psi.op-ntcf-cf-adjunction-unit[symmetric]$ 
       $assms(3-9)$ 
      cat-op-simps
  )
)
simp-all
qed (auto simp: op-ntcf-def cf-adj-RL-iso-def nat-omega-simps)
qed

lemma cf-adj-RL-iso-app-unique:
  fixes  $\mathcal{C} \mathcal{D} \mathfrak{F} \mathfrak{G} \Phi \mathfrak{G}' \Psi$ 
  assumes  $\Phi : \mathfrak{F} \rightleftharpoons_{CF} \mathfrak{G} : \mathcal{C} \rightleftharpoons_{C\alpha} \mathcal{D}$ 
    and  $\Psi : \mathfrak{F} \rightleftharpoons_{CF} \mathfrak{G}' : \mathcal{C} \rightleftharpoons_{C\alpha} \mathcal{D}$ 
    and  $x \in_o \mathcal{D} (Obj)$ 
  defines  $\mathfrak{G}x \equiv \mathfrak{G}(ObjMap)(x)$ 
    and  $\mathfrak{G}'x \equiv \mathfrak{G}'(ObjMap)(x)$ 
    and  $\varepsilon \equiv \varepsilon_C \Phi$ 
    and  $\varepsilon' \equiv \varepsilon_C \Psi$ 
    and  $f \equiv cf-adj-RL-iso \mathcal{C} \mathcal{D} \mathfrak{F} \mathfrak{G} \Phi \mathfrak{G}' \Psi (NTMap)(x)$ 
  shows
     $\exists ! f'$ .
     $f' : \mathfrak{G}'x \mapsto_{\mathcal{C}} \mathfrak{G}x \wedge$ 
     $\varepsilon'(NTMap)(x) = umap-fo \mathfrak{F} x \mathfrak{G}x (\varepsilon(NTMap)(x)) \mathfrak{G}'x (ArrVal)(f')$ 
     $f : \mathfrak{G}'x \mapsto_{iso_{\mathcal{C}}} \mathfrak{G}x$ 
     $\varepsilon'(NTMap)(x) = umap-fo \mathfrak{F} x \mathfrak{G}x (\varepsilon(NTMap)(x)) \mathfrak{G}'x (ArrVal)(f)$ 
  proof-
  interpret  $\Phi : is-cf-adjunction \alpha \mathcal{C} \mathcal{D} \mathfrak{F} \mathfrak{G} \Phi$  by (rule  $assms(1)$ )
  interpret  $\Psi : is-cf-adjunction \alpha \mathcal{C} \mathcal{D} \mathfrak{F} \mathfrak{G}' \Psi$  by (rule  $assms(2)$ )
  interpret  $\varepsilon : is-ntcf \alpha \mathcal{D} \mathcal{D} \langle \mathfrak{F} \circ_{CF} \mathfrak{G} \rangle \langle cf-id \mathcal{D} \rangle \langle \varepsilon_C \Phi \rangle$ 
    by (rule  $\Phi.cf-adjunction-counit-is-ntcf$ )
  show
     $\exists ! f'$ .
     $f' : \mathfrak{G}'x \mapsto_{\mathcal{C}} \mathfrak{G}x \wedge$ 
     $\varepsilon'(NTMap)(x) = umap-fo \mathfrak{F} x \mathfrak{G}x (\varepsilon(NTMap)(x)) \mathfrak{G}'x (ArrVal)(f')$ 
     $f : \mathfrak{G}'x \mapsto_{iso_{\mathcal{C}}} \mathfrak{G}x$ 
     $\varepsilon'(NTMap)(x) = umap-fo \mathfrak{F} x \mathfrak{G}x (\varepsilon(NTMap)(x)) \mathfrak{G}'x (ArrVal)(f)$ 
  by
    (
      intro cf-adj-LR-iso-app-unique
      [
        OF  $\Phi.is-cf-adjunction-op \Psi.is-cf-adjunction-op$ ,
        unfolded cat-op-simps,
        OF  $assms(3)$ ,
        unfolded  $\Psi.cf-adjunction-unit-NTMap-op$ ,
        folded  $\Phi.op-ntcf-cf-adjunction-counit$ ,
        folded  $op-ntcf-cf-adj-RL-iso[OF assms(1,2)]$ ,
        unfolded cat-op-simps,
        folded  $assms(4-8)$ 
      ]
    )+
  qed

```

lemma *cf-adj-RL-iso-is-iso-functor*:

— See Corollary 1 in Chapter IV-1 in [9].

assumes $\Phi : \mathfrak{F} \rightleftharpoons_{CF} \mathfrak{G} : \mathfrak{C} \rightleftharpoons_{C\alpha} \mathfrak{D}$ and $\Psi : \mathfrak{F} \rightleftharpoons_{CF} \mathfrak{G}' : \mathfrak{C} \rightleftharpoons_{C\alpha} \mathfrak{D}$

shows $\exists ! \vartheta$.

$\vartheta : \mathfrak{G}' \mapsto_{CF} \mathfrak{G} : \mathfrak{D} \mapsto_{C\alpha} \mathfrak{C} \wedge$

$\varepsilon_C \Psi = \varepsilon_C \Phi \cdot_{NTCF} (\mathfrak{F} \circ_{CF-NTCF} \vartheta)$

and *cf-adj-RL-iso* $\mathfrak{C} \mathfrak{D} \mathfrak{F} \mathfrak{G} \Phi \mathfrak{G}' \Psi : \mathfrak{G}' \mapsto_{CF.iso} \mathfrak{G} : \mathfrak{D} \mapsto_{C\alpha} \mathfrak{C}$

and $\varepsilon_C \Psi =$

$\varepsilon_C \Phi \cdot_{NTCF} (\mathfrak{F} \circ_{CF-NTCF} \text{cf-adj-RL-iso } \mathfrak{C} \mathfrak{D} \mathfrak{F} \mathfrak{G} \Phi \mathfrak{G}' \Psi)$

proof–

interpret Φ : *is-cf-adjunction* $\alpha \mathfrak{C} \mathfrak{D} \mathfrak{F} \mathfrak{G} \Phi$ **by** (*rule assms(1)*)

interpret Ψ : *is-cf-adjunction* $\alpha \mathfrak{C} \mathfrak{D} \mathfrak{F} \mathfrak{G}' \Psi$ **by** (*rule assms(2)*)

interpret ε : *is-ntcf* $\alpha \mathfrak{D} \mathfrak{D} \langle \mathfrak{F} \circ_{CF} \mathfrak{G} \rangle \langle \text{cf-id } \mathfrak{D} \rangle \langle \varepsilon_C \Phi \rangle$

by (*rule* Φ .*cf-adjunction-counit-is-ntcf*)

note *cf-adj-LR-iso-is-iso-functor-op* = *cf-adj-LR-iso-is-iso-functor*

[
OF Φ .*is-cf-adjunction-op* Ψ .*is-cf-adjunction-op*,
folded

Φ .*op-ntcf-cf-adjunction-counit*

Ψ .*op-ntcf-cf-adjunction-counit*

op-ntcf-cf-adj-RL-iso[*OF assms*]

]

from *cf-adj-LR-iso-is-iso-functor-op(1)* **obtain** ϑ

where $\vartheta : \mathfrak{D} : \text{op-cf } \mathfrak{G} \mapsto_{CF} \text{op-cf } \mathfrak{G}' : \text{op-cat } \mathfrak{D} \mapsto_{C\alpha} \text{op-cat } \mathfrak{C}$

and *op-ntcf- ε -def*: *op-ntcf* $(\varepsilon_C \Psi) =$

op-cf $\mathfrak{F} \circ_{CF-NTCF} \vartheta \cdot_{NTCF} \text{op-ntcf } (\varepsilon_C \Phi)$

and *unique- ϑ'* :

[[

$\vartheta' : \text{op-cf } \mathfrak{G} \mapsto_{CF} \text{op-cf } \mathfrak{G}' : \text{op-cat } \mathfrak{D} \mapsto_{C\alpha} \text{op-cat } \mathfrak{C};$

op-ntcf $(\varepsilon_C \Psi) = \text{op-cf } \mathfrak{F} \circ_{CF-NTCF} \vartheta' \cdot_{NTCF} \text{op-ntcf } (\varepsilon_C \Phi)$

]] $\implies \vartheta' = \vartheta$

for ϑ'

by *metis*

interpret ϑ : *is-ntcf* $\alpha \langle \text{op-cat } \mathfrak{D} \rangle \langle \text{op-cat } \mathfrak{C} \rangle \langle \text{op-cf } \mathfrak{G} \rangle \langle \text{op-cf } \mathfrak{G}' \rangle \vartheta$

by (*rule* ϑ)

show $\exists ! \vartheta$. $\vartheta : \mathfrak{G}' \mapsto_{CF} \mathfrak{G} : \mathfrak{D} \mapsto_{C\alpha} \mathfrak{C} \wedge \varepsilon_C \Psi = \varepsilon_C \Phi \cdot_{NTCF} (\mathfrak{F} \circ_{CF-NTCF} \vartheta)$

proof(*intro ex1I conjI*; (*elim conjE*)?)

show *op- ϑ* : *op-ntcf* $\vartheta : \mathfrak{G}' \mapsto_{CF} \mathfrak{G} : \mathfrak{D} \mapsto_{C\alpha} \mathfrak{C}$

by (*rule* ϑ .*is-ntcf-op*[*unfolded cat-op-simps*])

from *op-ntcf- ε -def* **have**

op-ntcf $(\text{op-ntcf } (\varepsilon_C \Psi)) =$

op-ntcf $(\text{op-cf } \mathfrak{F} \circ_{CF-NTCF} \vartheta \cdot_{NTCF} \text{op-ntcf } (\varepsilon_C \Phi))$

by *simp*

then show *ε -def*: $\varepsilon_C \Psi = \varepsilon_C \Phi \cdot_{NTCF} (\mathfrak{F} \circ_{CF-NTCF} \text{op-ntcf } \vartheta)$

by

(

cs-prems **cs-shallow**

cs-simp: *cat-op-simps*

cs-intro: *adj-cs-intros cat-cs-intros cat-op-intros*

)

fix ϑ' **assume** *prems*:

$\vartheta' : \mathfrak{G}' \mapsto_{CF} \mathfrak{G} : \mathfrak{D} \mapsto_{C\alpha} \mathfrak{C}$

$\varepsilon_C \Psi = \varepsilon_C \Phi \cdot_{NTCF} (\mathfrak{F} \circ_{CF-NTCF} \vartheta')$

interpret ϑ' : *is-ntcf* $\alpha \mathfrak{D} \mathfrak{C} \mathfrak{G}' \mathfrak{G} \vartheta'$ **by** (*rule* *prems(1)*)

have *op-ntcf* $(\varepsilon_C \Psi) = \text{op-cf } \mathfrak{F} \circ_{CF-NTCF} \text{op-ntcf } \vartheta' \cdot_{NTCF} \text{op-ntcf } (\varepsilon_C \Phi)$

by

(

cs-concl **cs-shallow**

cs-simp:
prems(2)
op-ntcf-cf-ntcf-comp[*symmetric*]
op-ntcf-ntcf-vcomp[*symmetric*]
cs-intro: *cat-cs-intros*
)

from *unique- ϑ'* [*OF ϑ' .is-ntcf-op this, symmetric*] **have**
op-ntcf $\vartheta = op-ntcf (op-ntcf \vartheta')$
by *simp*
then show *$\vartheta' = op-ntcf \vartheta$*
by (*cs-prems cs-shallow cs-simp: cat-cs-simps cat-op-simps*) *simp*
qed

from *is-iso-ntcf.is-iso-ntcf-op*[*OF cf-adj-LR-iso-is-iso-functor-op(2)*] **show**
cf-adj-RL-iso $\mathfrak{C} \mathfrak{D} \mathfrak{F} \mathfrak{G} \Phi \mathfrak{G}' \Psi : \mathfrak{G}' \mapsto_{CF.iso} \mathfrak{G} : \mathfrak{D} \mapsto_{C\alpha} \mathfrak{C}$
by
(

cs-prems cs-shallow
cs-simp: *cat-op-simps cs-intro: adj-cs-intros cat-op-intros*
)

from *cf-adj-LR-iso-is-iso-functor-op(3)* **have**
op-ntcf (op-ntcf ($\varepsilon_C \Psi$)) =
op-ntcf
(

op-cf $\mathfrak{F} \circ_{CF-NTCF} op-ntcf (cf-adj-RL-iso \mathfrak{C} \mathfrak{D} \mathfrak{F} \mathfrak{G} \Phi \mathfrak{G}' \Psi) \cdot_{NTCF}$
op-ntcf ($\varepsilon_C \Phi$)
)

by *simp*
from
this
cf-adj-LR-iso-is-iso-functor-op(2)[
unfolded op-ntcf-cf-adj-RL-iso[*OF assms*]
]

show *$\varepsilon_C \Psi = \varepsilon_C \Phi \cdot_{NTCF} (\mathfrak{F} \circ_{CF-NTCF} cf-adj-RL-iso \mathfrak{C} \mathfrak{D} \mathfrak{F} \mathfrak{G} \Phi \mathfrak{G}' \Psi)$*
by
(

cs-prems cs-shallow
cs-simp: *cat-op-simps cat-op-simps*
cs-intro: *ntcf-cs-intros adj-cs-intros cat-cs-intros cat-op-intros*
)

qed

13.13 Further properties of the adjoint functors

lemma (in *is-cf-adjunction*) *cf-adj-exp-cf-cat:*

— See Proposition 4.4.6 in [14].

assumes $Z \beta$ and $\alpha \in_o \beta$ and *category* $\alpha \mathfrak{J}$

shows

cf-adjunction-of-unit

β

(*exp-cf-cat* $\alpha \mathfrak{F} \mathfrak{J}$)

(*exp-cf-cat* $\alpha \mathfrak{G} \mathfrak{J}$)

(*exp-ntcf-cat* $\alpha (\eta_C \Phi) \mathfrak{J}$) :

exp-cf-cat $\alpha \mathfrak{F} \mathfrak{J} \rightleftharpoons_{CF} \text{exp-cf-cat } \alpha \mathfrak{G} \mathfrak{J}$:

cat-FUNCT $\alpha \mathfrak{J} \mathfrak{C} \rightleftharpoons_{C\beta} \text{cat-FUNCT } \alpha \mathfrak{J} \mathfrak{D}$

proof—

interpret β : $Z \beta$ **by** (*rule assms(1)*)

interpret \mathfrak{J} : *category* $\alpha \mathfrak{J}$ **by** (*rule assms(3)*)

show *?thesis*

proof

(
 rule counit-unit-is-cf-adjunction(1)[
 where $\varepsilon = \langle \text{exp-ntcf-cat } \alpha \ (\varepsilon_C \ \Phi) \ \mathfrak{J} \rangle$
]
)
from *assms* **show** $\text{exp-ntcf-cat } \alpha \ (\eta_C \ \Phi) \ \mathfrak{J} :$
 $\text{cf-id } (\text{cat-FUNCT } \alpha \ \mathfrak{J} \ \mathfrak{C}) \mapsto_{CF} \text{exp-cf-cat } \alpha \ \mathfrak{G} \ \mathfrak{J} \circ_{CF} \text{exp-cf-cat } \alpha \ \mathfrak{F} \ \mathfrak{J} :$
 $\text{cat-FUNCT } \alpha \ \mathfrak{J} \ \mathfrak{C} \mapsto_{C\beta} \text{cat-FUNCT } \alpha \ \mathfrak{J} \ \mathfrak{C}$
by
 (
 cs-concl
cs-simp:
 cat-cs-simps cat-FUNCT-cs-simps
 exp-cf-cat-cf-id-cat[symmetric] exp-cf-cat-cf-comp[symmetric]
cs-intro:
 cat-cs-intros cat-small-cs-intros cat-FUNCT-cs-intros adj-cs-intros
)
from *assms* **show**
 $\text{exp-ntcf-cat } \alpha \ (\varepsilon_C \ \Phi) \ \mathfrak{J} :$
 $\text{exp-cf-cat } \alpha \ \mathfrak{F} \ \mathfrak{J} \circ_{CF} \text{exp-cf-cat } \alpha \ \mathfrak{G} \ \mathfrak{J} \mapsto_{CF} \text{cf-id } (\text{cat-FUNCT } \alpha \ \mathfrak{J} \ \mathfrak{D}) :$
 $\text{cat-FUNCT } \alpha \ \mathfrak{J} \ \mathfrak{D} \mapsto_{C\beta} \text{cat-FUNCT } \alpha \ \mathfrak{J} \ \mathfrak{D}$
by
 (
 cs-concl
cs-simp:
 cat-cs-simps
 cat-FUNCT-cs-simps
 exp-cf-cat-cf-id-cat[symmetric]
 exp-cf-cat-cf-comp[symmetric]
cs-intro:
 cat-cs-intros cat-small-cs-intros cat-FUNCT-cs-intros adj-cs-intros
)
note [symmetric, cat-cs-simps] =
 ntcf-id-exp-cf-cat
 exp-ntcf-cat-ntcf-vcomp
 exp-ntcf-cat-ntcf-cf-comp
 exp-ntcf-cat-cf-ntcf-comp
from *assms* **show**
 $(\text{exp-cf-cat } \alpha \ \mathfrak{G} \ \mathfrak{J} \circ_{CF-NTCF} \text{exp-ntcf-cat } \alpha \ (\varepsilon_C \ \Phi) \ \mathfrak{J}) \cdot_{NTCF}$
 $(\text{exp-ntcf-cat } \alpha \ (\eta_C \ \Phi) \ \mathfrak{J} \circ_{NTCF-CF} \text{exp-cf-cat } \alpha \ \mathfrak{G} \ \mathfrak{J}) =$
 $\text{ntcf-id } (\text{exp-cf-cat } \alpha \ \mathfrak{G} \ \mathfrak{J})$
by
 (
 cs-concl **cs-shallow**
cs-simp: adj-cs-simps cat-cs-simps
cs-intro: adj-cs-intros cat-cs-intros
)
from *assms* **show**
 $\text{exp-ntcf-cat } \alpha \ (\varepsilon_C \ \Phi) \ \mathfrak{J} \circ_{NTCF-CF} \text{exp-cf-cat } \alpha \ \mathfrak{F} \ \mathfrak{J} \cdot_{NTCF}$
 $(\text{exp-cf-cat } \alpha \ \mathfrak{F} \ \mathfrak{J} \circ_{CF-NTCF} \text{exp-ntcf-cat } \alpha \ (\eta_C \ \Phi) \ \mathfrak{J}) =$
 $\text{ntcf-id } (\text{exp-cf-cat } \alpha \ \mathfrak{F} \ \mathfrak{J})$
by
 (
 cs-concl **cs-shallow**
cs-simp: adj-cs-simps cat-cs-simps
cs-intro: adj-cs-intros cat-cs-intros
)

qed
 (
 use *assms* in
 <
 cs-concl
 cs-intro: *cat-cs-intros* *cat-small-cs-intros* *cat-FUNCT-cs-intros*
 >
)+
 qed

lemma (in *is-cf-adjunction*) *cf-adj-exp-cf-cat-exp-cf-cat*:

— See Proposition 4.4.6 in [14].

assumes $Z \beta$ and $\alpha \in_o \beta$ and *category* $\alpha \mathfrak{A}$

shows

cf-adjunction-of-unit
 β
 (*exp-cat-cf* $\alpha \mathfrak{A} \mathfrak{G}$)
 (*exp-cat-cf* $\alpha \mathfrak{A} \mathfrak{F}$)
 (*exp-cat-ntcf* $\alpha \mathfrak{A} (\eta_C \Phi)$) :
exp-cat-cf $\alpha \mathfrak{A} \mathfrak{G} \Rightarrow_{CF} \text{exp-cat-cf } \alpha \mathfrak{A} \mathfrak{F}$:
cat-FUNCT $\alpha \mathfrak{C} \mathfrak{A} \Rightarrow_{C\beta} \text{cat-FUNCT } \alpha \mathfrak{D} \mathfrak{A}$

proof–

interpret β : $Z \beta$ **by** (*rule assms(1)*)

interpret \mathfrak{A} : *category* $\alpha \mathfrak{A}$ **by** (*rule assms(3)*)

show *?thesis*

proof

(
 rule *counit-unit-is-cf-adjunction(1)*[
 where $\varepsilon = \langle \text{exp-cat-ntcf } \alpha \mathfrak{A} (\varepsilon_C \Phi) \rangle$
]
)

from *assms is-cf-adjunction-axioms* **show**

exp-cat-ntcf $\alpha \mathfrak{A} (\eta_C \Phi)$:
cf-id (*cat-FUNCT* $\alpha \mathfrak{C} \mathfrak{A}$) $\mapsto_{CF} \text{exp-cat-cf } \alpha \mathfrak{A} \mathfrak{F} \circ_{CF} \text{exp-cat-cf } \alpha \mathfrak{A} \mathfrak{G}$:
cat-FUNCT $\alpha \mathfrak{C} \mathfrak{A} \mapsto_{C\beta} \text{cat-FUNCT } \alpha \mathfrak{C} \mathfrak{A}$

by

(
cs-concl
cs-simp:
 exp-cat-cf-cat-cf-id[symmetric] *exp-cat-cf-cf-comp[symmetric]*
cs-intro: *cat-small-cs-intros* *cat-FUNCT-cs-intros* *adj-cs-intros*
)

from *assms is-cf-adjunction-axioms* **show**

exp-cat-ntcf $\alpha \mathfrak{A} (\varepsilon_C \Phi)$:
exp-cat-cf $\alpha \mathfrak{A} \mathfrak{G} \circ_{CF} \text{exp-cat-cf } \alpha \mathfrak{A} \mathfrak{F} \mapsto_{CF} \text{cf-id } (\text{cat-FUNCT } \alpha \mathfrak{D} \mathfrak{A})$:
cat-FUNCT $\alpha \mathfrak{D} \mathfrak{A} \mapsto_{C\beta} \text{cat-FUNCT } \alpha \mathfrak{D} \mathfrak{A}$

by

(
cs-concl
cs-simp:
 exp-cat-cf-cat-cf-id[symmetric] *exp-cat-cf-cf-comp[symmetric]*
cs-intro: *cat-small-cs-intros* *cat-FUNCT-cs-intros* *adj-cs-intros*
)

note [*symmetric, cat-cs-simps*] =

ntcf-id-exp-cat-cf
exp-cat-ntcf-ntcf-vcomp

```

exp-cat-ntcf-ntcf-cf-comp
exp-cat-ntcf-cf-ntcf-comp
from assms show
  exp-cat-cf  $\alpha$   $\mathfrak{A}$   $\mathfrak{F}$   $\circ_{CF-NTCF}$  exp-cat-ntcf  $\alpha$   $\mathfrak{A}$   $(\varepsilon_C \Phi) \cdot_{NTCF}$ 
  (exp-cat-ntcf  $\alpha$   $\mathfrak{A}$   $(\eta_C \Phi) \circ_{NTCF-CF}$  exp-cat-cf  $\alpha$   $\mathfrak{A}$   $\mathfrak{F}$ ) =
  ntcf-id (exp-cat-cf  $\alpha$   $\mathfrak{A}$   $\mathfrak{F}$ )
by
  (
    cs-concl cs-shallow
    cs-simp: adj-cs-simps cat-cs-simps
    cs-intro: adj-cs-intros cat-cs-intros
  )
from assms show
  exp-cat-ntcf  $\alpha$   $\mathfrak{A}$   $(\varepsilon_C \Phi) \circ_{NTCF-CF}$  exp-cat-cf  $\alpha$   $\mathfrak{A}$   $\mathfrak{G}$   $\cdot_{NTCF}$ 
  (exp-cat-cf  $\alpha$   $\mathfrak{A}$   $\mathfrak{G}$   $\circ_{CF-NTCF}$  exp-cat-ntcf  $\alpha$   $\mathfrak{A}$   $(\eta_C \Phi)$ ) =
  ntcf-id (exp-cat-cf  $\alpha$   $\mathfrak{A}$   $\mathfrak{G}$ )
by
  (
    cs-concl cs-shallow
    cs-simp: adj-cs-simps cat-cs-simps
    cs-intro: adj-cs-intros cat-cs-intros
  )
qed
  (
    use assms in
    <
      cs-concl
      cs-intro: cat-cs-intros cat-small-cs-intros cat-FUNCT-cs-intros
    >
  )+

```

qed

13.14 Adjoints on limits

lemma *cf-AdjRight-preserves-limits:*

— See Chapter V-5 in [9].

assumes $\Phi : \mathfrak{F} \rightleftharpoons_{CF} \mathfrak{G} : \mathfrak{X} \rightleftharpoons_{C\alpha} \mathfrak{A}$

shows *is-cf-continuous* α \mathfrak{G}

proof(*intro is-cf-continuousI*)

interpret Φ : *is-cf-adjunction* α \mathfrak{X} \mathfrak{A} \mathfrak{F} \mathfrak{G} Φ **by** (*rule* *assms*(1))

show $\mathfrak{G} : \mathfrak{A} \mapsto_{C\alpha} \mathfrak{X}$ **by** (*rule* Φ .*RL.is-functor-axioms*)

fix \mathfrak{T} \mathfrak{J} **assume** *prems*: $\mathfrak{T} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{A}$

show *cf-preserves-limits* α \mathfrak{G} \mathfrak{T}

proof(*intro cf-preserves-limitsI*, *rule* *prems*, *rule* Φ .*RL.is-functor-axioms*)

fix τ a **assume** $\tau : a <_{CF.lim} \mathfrak{T} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{A}$

then interpret τ : *is-cat-limit* α \mathfrak{J} \mathfrak{A} \mathfrak{T} a τ .

show $\mathfrak{G} \circ_{CF-NTCF} \tau : \mathfrak{G}(\mathit{ObjMap})(\mathit{!}a) <_{CF.lim} \mathfrak{G} \circ_{CF} \mathfrak{T} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{X}$

proof(*intro is-cat-limitI*)

show $\mathfrak{G} \circ_{CF-NTCF} \tau : \mathfrak{G}(\mathit{ObjMap})(\mathit{!}a) <_{CF.cone} \mathfrak{G} \circ_{CF} \mathfrak{T} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{X}$

by

(
 intro cf-ntcf-comp-cf-cat-cone prems,
 rule τ .is-cat-cone-axioms,
 intro Φ .RL.is-functor-axioms
)

fix $\sigma' b'$ assume $\sigma' : b' <_{CF.cone} \mathfrak{G} \circ_{CF} \mathfrak{T} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{X}$
 then interpret $\sigma' : is-cat-cone \alpha b' \mathfrak{J} \mathfrak{X} \langle \mathfrak{G} \circ_{CF} \mathfrak{T} \rangle \sigma'$.

have $\varepsilon_C \Phi \circ_{NTCF-CF} \mathfrak{T} : \mathfrak{F} \circ_{CF} (\mathfrak{G} \circ_{CF} \mathfrak{T}) \mapsto_{CF} \mathfrak{T} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{A}$
 by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros adj-cs-intros)

moreover have $\mathfrak{F} \circ_{CF-NTCF} \sigma' :$
 $\mathfrak{F}(\text{ObjMap})(b') <_{CF.cone} \mathfrak{F} \circ_{CF} (\mathfrak{G} \circ_{CF} \mathfrak{T}) : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{A}$

by
 (
 intro cf-ntcf-comp-cf-cat-cone,
 rule σ' .is-cat-cone-axioms,
 rule Φ .LR.is-functor-axioms
)

ultimately have $(\varepsilon_C \Phi \circ_{NTCF-CF} \mathfrak{T}) \cdot_{NTCF} (\mathfrak{F} \circ_{CF-NTCF} \sigma') :$
 $\mathfrak{F}(\text{ObjMap})(b') <_{CF.cone} \mathfrak{T} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{A}$

by (rule ntcf-vcomp-is-cat-cone)

from τ .cat-lim-unique-cone'[OF this] obtain h

where $h : h : \mathfrak{F}(\text{ObjMap})(b') \mapsto_{\mathfrak{A}} a$

and $\varepsilon_{\mathfrak{T}} \mathfrak{F} \sigma' : \bigwedge j. j \in_{\circ} \mathfrak{J}(\text{Obj}) \implies$

$((\varepsilon_C \Phi \circ_{NTCF-CF} \mathfrak{T}) \cdot_{NTCF} (\mathfrak{F} \circ_{CF-NTCF} \sigma'))(\text{NTMap})(j) = \tau(\text{NTMap})(j) \circ_{A\mathfrak{A}} h$

and h -unique:

\llbracket
 $h' : \mathfrak{F}(\text{ObjMap})(b') \mapsto_{\mathfrak{A}} a;$
 $\bigwedge j. j \in_{\circ} \mathfrak{J}(\text{Obj}) \implies$
 $((\varepsilon_C \Phi \circ_{NTCF-CF} \mathfrak{T}) \cdot_{NTCF} (\mathfrak{F} \circ_{CF-NTCF} \sigma'))(\text{NTMap})(j) =$
 $\tau(\text{NTMap})(j) \circ_{A\mathfrak{A}} h'$
 $\rrbracket \implies h' = h$

for h'

by metis

have $\varepsilon_{\mathfrak{T}} \mathfrak{F} \sigma' :$

$\varepsilon_C \Phi(\text{NTMap})(\mathfrak{T}(\text{ObjMap})(j)) \circ_{A\mathfrak{A}} \mathfrak{F}(\text{ArrMap})(\sigma'(\text{NTMap})(j)) =$
 $\tau(\text{NTMap})(j) \circ_{A\mathfrak{A}} h$

if $j \in_{\circ} \mathfrak{J}(\text{Obj})$ for j

using $\varepsilon_{\mathfrak{T}} \mathfrak{F} \sigma'$ [OF that] that

by

(
 cs-prems cs-shallow
 cs-simp: cat-cs-simps cs-intro: adj-cs-intros cat-cs-intros
)

show $\exists! f'$.

$f' : b' \mapsto_{\mathfrak{X}} \mathfrak{G}(\text{ObjMap})(a) \wedge \sigma' = \mathfrak{G} \circ_{CF-NTCF} \tau \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{X} f'$

proof(intro ex1I conjI; (elim conjE)?)

let $?h' = \langle \mathfrak{G}(\text{ArrMap})(h) \circ_{A\mathfrak{X}} \eta_C \Phi(\text{NTMap})(b') \rangle$

from h show $?h' : b' \mapsto_{\mathfrak{X}} \mathfrak{G}(\text{ObjMap})(a)$

by

(
 cs-concl cs-shallow
 cs-intro: cat-cs-intros cat-lim-cs-intros adj-cs-intros
)

show $\sigma' = \mathfrak{G} \circ_{CF-NTCF} \tau \cdot_{NTCF} \text{ntcf-const } \mathfrak{J} \mathfrak{X} ?h'$

proof(rule ntcf-eqI)

show $\sigma' : cf\text{-const } \mathfrak{J} \mathfrak{X} b' \mapsto_{CF} \mathfrak{G} \circ_{CF} \mathfrak{T} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{X}$
by (*rule* $\sigma'.is\text{-ntcf-axioms}$)
then have *dom-lhs*: $\mathcal{D}_\circ (\sigma'(\mathcal{N}TMap)) = \mathfrak{J}(\mathcal{O}bj)$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps*)
from *h* **show**
 $\mathfrak{G} \circ_{CF-N\tau CF} \tau \cdot_{N\tau CF} n\tau cf\text{-const } \mathfrak{J} \mathfrak{X} ?h' :$
 $cf\text{-const } \mathfrak{J} \mathfrak{X} b' \mapsto_{CF} \mathfrak{G} \circ_{CF} \mathfrak{T} : \mathfrak{J} \mapsto_{C\alpha} \mathfrak{X}$
by
(

cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-lim-cs-intros adj-cs-intros cat-cs-intros*
)

then have *dom-rhs*:
 $\mathcal{D}_\circ ((\mathfrak{G} \circ_{CF-N\tau CF} \tau \cdot_{N\tau CF} n\tau cf\text{-const } \mathfrak{J} \mathfrak{X} ?h')(\mathcal{N}TMap)) = \mathfrak{J}(\mathcal{O}bj)$
by (*cs-concl cs-simp: cat-cs-simps*)
show $\sigma'(\mathcal{N}TMap) = (\mathfrak{G} \circ_{CF-N\tau CF} \tau \cdot_{N\tau CF} n\tau cf\text{-const } \mathfrak{J} \mathfrak{X} ?h')(\mathcal{N}TMap)$
proof(*rule vsv-eqI, unfold dom-lhs dom-rhs*)
fix *j* **assume** *prems'*: $j \in_\circ \mathfrak{J}(\mathcal{O}bj)$
note [*cat-cs-simps*] = $\Phi.L.cat\text{-assoc-helper}$
[

where $h = \langle \mathfrak{G}(\mathcal{A}rrMap)(\tau(\mathcal{N}TMap)(j)) \rangle$
and $g = \langle \mathfrak{G}(\mathcal{A}rrMap)(h) \rangle$
and $f = \langle \eta_C \Phi(\mathcal{N}TMap)(b') \rangle$
and $q = \langle \mathfrak{G}(\mathcal{A}rrMap)(\tau(\mathcal{N}TMap)(j) \circ_{A\mathfrak{X}} h) \rangle$
]

from *prems' h* **have** [*cat-cs-simps*]:
 $\mathfrak{G}(\mathcal{A}rrMap)(\tau(\mathcal{N}TMap)(j) \circ_{A\mathfrak{X}} h) \circ_{A\mathfrak{X}} \eta_C \Phi(\mathcal{N}TMap)(b') = \sigma'(\mathcal{N}TMap)(j)$
by
(

cs-concl cs-shallow
cs-simp: *cat-cs-simps* $\varepsilon\mathfrak{X}\text{-}\mathfrak{S}\sigma[OF\text{ }prems',\text{ }symmetric]$
cs-intro: *adj-cs-intros cat-cs-intros*
)

from *prems' h* **show**
 $\sigma'(\mathcal{N}TMap)(j) = (\mathfrak{G} \circ_{CF-N\tau CF} \tau \cdot_{N\tau CF} n\tau cf\text{-const } \mathfrak{J} \mathfrak{X} ?h')(\mathcal{N}TMap)(j)$
by
(

cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-lim-cs-intros adj-cs-intros cat-cs-intros*
)

qed (*cs-concl cs-intro: V-cs-intros cat-cs-intros*)+
qed *simp-all*

fix *f'* **assume** *prems'*:
 $f' : b' \mapsto_{\mathfrak{X}} \mathfrak{G}(\mathcal{O}bjMap)(a)$
 $\sigma' = \mathfrak{G} \circ_{CF-N\tau CF} \tau \cdot_{N\tau CF} n\tau cf\text{-const } \mathfrak{J} \mathfrak{X} f'$

from *prems'(2)* **have** $\sigma'\text{-}j\text{-def}'$:
 $\sigma'(\mathcal{N}TMap)(j) = (\mathfrak{G} \circ_{CF-N\tau CF} \tau \cdot_{N\tau CF} n\tau cf\text{-const } \mathfrak{J} \mathfrak{X} f')(\mathcal{N}TMap)(j)$
for *j*
by *simp*
have $\sigma'\text{-}j\text{-def}$: $\sigma'(\mathcal{N}TMap)(j) = \mathfrak{G}(\mathcal{A}rrMap)(\tau(\mathcal{N}TMap)(j)) \circ_{A\mathfrak{X}} f'$
if $j \in_\circ \mathfrak{J}(\mathcal{O}bj)$ **for** *j*
using $\sigma'\text{-}j\text{-def}'[of\ j]$ **that** *prems'(1)*
by
(

cs-prems
cs-simp: *cat-cs-simps* **cs-intro:** *cat-lim-cs-intros cat-cs-intros*
)

from *prems'(1)* **have** $\varepsilon a \cdot \mathfrak{F} f'$:
 $\varepsilon_C \Phi(\downarrow NTMap)(\downarrow a) \circ_{A\mathfrak{A}} \mathfrak{F}(\downarrow ArrMap)(\downarrow f')$: $\mathfrak{F}(\downarrow ObjMap)(\downarrow b')$ $\mapsto_{A\mathfrak{A}}$ a
by (*cs-concl* **cs-intro:** *cat-lim-cs-intros cat-cs-intros adj-cs-intros*)

interpret ε : *is-ntcf* $\alpha \mathfrak{A} \mathfrak{A} \langle \mathfrak{F} \circ_{CF} \mathfrak{G} \rangle \langle cf-id \mathfrak{A} \rangle \langle \varepsilon_C \Phi \rangle$
by (*rule* Φ .*cf-adjunction-counit-is-ntcf*)

have
 $(\varepsilon_C \Phi \circ_{NTCF-CF} \mathfrak{T} \cdot_{NTCF} (\mathfrak{F} \circ_{CF-NTCF} \sigma'))(\downarrow NTMap)(\downarrow j) =$
 $\tau(\downarrow NTMap)(\downarrow j) \circ_{A\mathfrak{A}} (\varepsilon_C \Phi(\downarrow NTMap)(\downarrow a) \circ_{A\mathfrak{A}} \mathfrak{F}(\downarrow ArrMap)(\downarrow f'))$
if $j \in_o \mathfrak{J}(\downarrow Obj)$ **for** j

proof-
from *that* **have** $\tau(\downarrow NTMap)(\downarrow j) : a \mapsto_{A\mathfrak{A}} \mathfrak{T}(\downarrow ObjMap)(\downarrow j)$
by
(

cs-concl **cs-shallow**
cs-simp: *cat-cs-simps* **cs-intro:** *cat-cs-intros*
)

from ε .*ntcf-Comp-commute*[*OF this*] **that** **have** [*cat-cs-simps*]:
 $\varepsilon_C \Phi(\downarrow NTMap)(\downarrow \mathfrak{T}(\downarrow ObjMap)(\downarrow j)) \circ_{A\mathfrak{A}} \mathfrak{F}(\downarrow ArrMap)(\downarrow \mathfrak{G}(\downarrow ArrMap)(\downarrow \tau(\downarrow NTMap)(\downarrow j))) =$
 $\tau(\downarrow NTMap)(\downarrow j) \circ_{A\mathfrak{A}} \varepsilon_C \Phi(\downarrow NTMap)(\downarrow a)$
by
(

cs-prems **cs-shallow**
cs-simp: *cat-cs-simps* **cs-intro:** *cat-cs-intros*
)

note [*cat-cs-simps*] = Φ .*R.cat-assoc-helper*
[

where $h = \langle \varepsilon_C \Phi(\downarrow NTMap)(\downarrow \mathfrak{T}(\downarrow ObjMap)(\downarrow j)) \rangle$
and $g = \langle \mathfrak{F}(\downarrow ArrMap)(\downarrow \mathfrak{G}(\downarrow ArrMap)(\downarrow \tau(\downarrow NTMap)(\downarrow j))) \rangle$
and $q = \langle \tau(\downarrow NTMap)(\downarrow j) \circ_{A\mathfrak{A}} \varepsilon_C \Phi(\downarrow NTMap)(\downarrow a) \rangle$
]

show *?thesis*
using *that* *prems'(1)*
by
(

cs-concl
cs-simp: *cat-cs-simps* σ' -*j-def*
cs-intro: *cat-lim-cs-intros cat-cs-intros adj-cs-intros*
)

qed
from *h-unique*[*OF* $\varepsilon a \cdot \mathfrak{F} f'$ *this*] **have**
 $\mathfrak{G}(\downarrow ArrMap)(\downarrow \varepsilon_C \Phi(\downarrow NTMap)(\downarrow a) \circ_{A\mathfrak{A}} \mathfrak{F}(\downarrow ArrMap)(\downarrow f')) \circ_{A\mathfrak{X}} \eta_C \Phi(\downarrow NTMap)(\downarrow b') = ?h'$
by *simp*
from *this* *prems'(1)* **show** $f' = \mathfrak{G}(\downarrow ArrMap)(\downarrow h) \circ_{A\mathfrak{X}} \eta_C \Phi(\downarrow NTMap)(\downarrow b')$
by
(

cs-prems
cs-simp: *cat-cs-simps* Φ .*cf-adj-counit-unit-app*
cs-intro: *cat-lim-cs-intros cat-cs-intros*
)

qed

qed

qed
qed

14 Simple Kan extensions

14.1 Background

named-theorems *cat-Kan-cs-simps*

named-theorems *cat-Kan-cs-intros*

14.2 Kan extension

14.2.1 Definition and elementary properties

See Chapter X-3 in [9].

locale *is-cat-rKe* =

AG: *is-functor* $\alpha \mathfrak{B} \mathfrak{C} \mathfrak{K} +$
Ran: *is-functor* $\alpha \mathfrak{C} \mathfrak{A} \mathfrak{G} +$
ntcf-rKe: *is-ntcf* $\alpha \mathfrak{B} \mathfrak{A} \langle \mathfrak{G} \circ_{CF} \mathfrak{K} \rangle \mathfrak{T} \varepsilon$
for $\alpha \mathfrak{B} \mathfrak{C} \mathfrak{A} \mathfrak{K} \mathfrak{T} \mathfrak{G} \varepsilon +$
assumes *cat-rKe-ua-fo*:
universal-arrow-fo
 (*exp-cat-cf* $\alpha \mathfrak{A} \mathfrak{K}$)
 (*cf-map* \mathfrak{T})
 (*cf-map* \mathfrak{G})
 (*ntcf-arrow* ε)

syntax *-is-cat-rKe* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$

($\langle (- \cdot / - \circ_{CF} - \mapsto_{CF.rKe1} - \cdot / - \mapsto_C - \mapsto_C -) \rangle$ [51, 51, 51, 51, 51, 51, 51] 51)

syntax-consts *-is-cat-rKe* \Leftarrow *is-cat-rKe*

translations $\varepsilon : \mathfrak{G} \circ_{CF} \mathfrak{K} \mapsto_{CF.rKe\alpha} \mathfrak{T} : \mathfrak{B} \mapsto_C \mathfrak{C} \mapsto_C \mathfrak{A} \Leftarrow$

CONST is-cat-rKe $\alpha \mathfrak{B} \mathfrak{C} \mathfrak{A} \mathfrak{K} \mathfrak{T} \mathfrak{G} \varepsilon$

locale *is-cat-lKe* =

AG: *is-functor* $\alpha \mathfrak{B} \mathfrak{C} \mathfrak{K} +$
Lan: *is-functor* $\alpha \mathfrak{C} \mathfrak{A} \mathfrak{F} +$
ntcf-lKe: *is-ntcf* $\alpha \mathfrak{B} \mathfrak{A} \mathfrak{T} \langle \mathfrak{F} \circ_{CF} \mathfrak{K} \rangle \eta$
for $\alpha \mathfrak{B} \mathfrak{C} \mathfrak{A} \mathfrak{K} \mathfrak{T} \mathfrak{F} \eta +$
assumes *cat-lKe-ua-fo*:
universal-arrow-fo
 (*exp-cat-cf* α (*op-cat* \mathfrak{A}) (*op-cf* \mathfrak{K}))
 (*cf-map* \mathfrak{T})
 (*cf-map* \mathfrak{F})
 (*ntcf-arrow* (*op-ntcf* η))

syntax *-is-cat-lKe* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$

($\langle (- \cdot / - \mapsto_{CF.lKe1} - \circ_{CF} - \cdot / - \mapsto_C - \mapsto_C -) \rangle$ [51, 51, 51, 51, 51, 51, 51] 51)

syntax-consts *-is-cat-lKe* \Leftarrow *is-cat-lKe*

translations $\eta : \mathfrak{T} \mapsto_{CF.lKe\alpha} \mathfrak{F} \circ_{CF} \mathfrak{K} : \mathfrak{B} \mapsto_C \mathfrak{C} \mapsto_C \mathfrak{A} \Leftarrow$

CONST is-cat-lKe $\alpha \mathfrak{B} \mathfrak{C} \mathfrak{A} \mathfrak{K} \mathfrak{T} \mathfrak{F} \eta$

Rules.

lemma (in *is-cat-rKe*) *is-cat-rKe-axioms'*[*cat-Kan-cs-intros*]:

assumes $\alpha' = \alpha$

and $\mathfrak{G}' = \mathfrak{G}$

and $\mathfrak{K}' = \mathfrak{K}$

and $\mathfrak{T}' = \mathfrak{T}$

and $\mathfrak{B}' = \mathfrak{B}$

and $\mathfrak{A}' = \mathfrak{A}$

and $\mathfrak{C}' = \mathfrak{C}$

shows $\varepsilon : \mathfrak{G}' \circ_{CF} \mathfrak{K}' \mapsto_{CF.rKe\alpha'} \mathfrak{T}' : \mathfrak{B}' \mapsto_C \mathfrak{C}' \mapsto_C \mathfrak{A}'$

unfolding *assms* by (rule *is-cat-rKe-axioms*)

mk-ide rf *is-cat-rKe-def*[*unfolded is-cat-rKe-axioms-def*]
 |*intro is-cat-rKeI*
 |*dest is-cat-rKeD*[*dest*]
 |*elim is-cat-rKeE*[*elim*]

lemmas [*cat-Kan-cs-intros*] = *is-cat-rKeD*(1-3)

lemma (in *is-cat-lKe*) *is-cat-lKe-axioms'*[*cat-Kan-cs-intros*]:

assumes $\alpha' = \alpha$
 and $\mathfrak{F}' = \mathfrak{F}$
 and $\mathfrak{K}' = \mathfrak{K}$
 and $\mathfrak{T}' = \mathfrak{T}$
 and $\mathfrak{B}' = \mathfrak{B}$
 and $\mathfrak{A}' = \mathfrak{A}$
 and $\mathfrak{C}' = \mathfrak{C}$
shows $\eta : \mathfrak{T}' \mapsto_{CF.lKe\alpha} \mathfrak{F}' \circ_{CF} \mathfrak{K}' : \mathfrak{B}' \mapsto_C \mathfrak{C}' \mapsto_C \mathfrak{A}'$
unfolding *assms* by (rule *is-cat-lKe-axioms*)

mk-ide rf *is-cat-lKe-def*[*unfolded is-cat-lKe-axioms-def*]
 |*intro is-cat-lKeI*
 |*dest is-cat-lKeD*[*dest*]
 |*elim is-cat-lKeE*[*elim*]

lemmas [*cat-Kan-cs-intros*] = *is-cat-lKeD*(1-3)

Duality.

lemma (in *is-cat-rKe*) *is-cat-lKe-op*:

op-ntcf ε :
 $op\text{-}cf \mathfrak{T} \mapsto_{CF.lKe\alpha} op\text{-}cf \mathfrak{G} \circ_{CF} op\text{-}cf \mathfrak{K} :$
 $op\text{-}cat \mathfrak{B} \mapsto_C op\text{-}cat \mathfrak{C} \mapsto_C op\text{-}cat \mathfrak{A}$
by (*intro is-cat-lKeI*, *unfold cat-op-simps*; (*intro cat-rKe-ua-fo*)?)
 (*cs-concl cs-shallow cs-simp: cat-op-simps cs-intro: cat-op-intros*)+

lemma (in *is-cat-rKe*) *is-cat-lKe-op'*[*cat-op-intros*]:

assumes $\mathfrak{T}' = op\text{-}cf \mathfrak{T}$
 and $\mathfrak{G}' = op\text{-}cf \mathfrak{G}$
 and $\mathfrak{K}' = op\text{-}cf \mathfrak{K}$
 and $\mathfrak{B}' = op\text{-}cat \mathfrak{B}$
 and $\mathfrak{A}' = op\text{-}cat \mathfrak{A}$
 and $\mathfrak{C}' = op\text{-}cat \mathfrak{C}$
shows $op\text{-}ntcf \varepsilon : \mathfrak{T}' \mapsto_{CF.lKe\alpha} \mathfrak{G}' \circ_{CF} \mathfrak{K}' : \mathfrak{B}' \mapsto_C \mathfrak{C}' \mapsto_C \mathfrak{A}'$
unfolding *assms* by (rule *is-cat-lKe-op*)

lemmas [*cat-op-intros*] = *is-cat-rKe.is-cat-lKe-op'*

lemma (in *is-cat-lKe*) *is-cat-rKe-op*:

op-ntcf η :
 $op\text{-}cf \mathfrak{F} \circ_{CF} op\text{-}cf \mathfrak{K} \mapsto_{CF.rKe\alpha} op\text{-}cf \mathfrak{T} :$
 $op\text{-}cat \mathfrak{B} \mapsto_C op\text{-}cat \mathfrak{C} \mapsto_C op\text{-}cat \mathfrak{A}$
by (*intro is-cat-rKeI*, *unfold cat-op-simps*; (*intro cat-lKe-ua-fo*)?)
 (*cs-concl cs-shallow cs-simp: cat-op-simps cs-intro: cat-op-intros*)+

lemma (in *is-cat-lKe*) *is-cat-lKe-op'*[*cat-op-intros*]:

assumes $\mathfrak{T}' = op\text{-}cf \mathfrak{T}$
 and $\mathfrak{F}' = op\text{-}cf \mathfrak{F}$
 and $\mathfrak{K}' = op\text{-}cf \mathfrak{K}$

and $\mathfrak{B}' = \text{op-cat } \mathfrak{B}$
 and $\mathfrak{A}' = \text{op-cat } \mathfrak{A}$
 and $\mathfrak{C}' = \text{op-cat } \mathfrak{C}$
 shows $\text{op-ntcf } \eta : \mathfrak{F}' \circ_{CF} \mathfrak{R}' \mapsto_{CF, rKe\alpha} \mathfrak{T}' : \mathfrak{B}' \mapsto_C \mathfrak{C}' \mapsto_C \mathfrak{A}'$
 unfolding *assms* by (rule *is-cat-rKe-op*)

lemmas [cat-op-intros] = *is-cat-lKe.is-cat-lKe-op'*

Elementary properties.

lemma (in *is-cat-rKe*) *cat-rKe-exp-cat-cf-cat-FUNCT-is-arr*:

assumes $\mathcal{Z} \beta$ and $\alpha \in_o \beta$

shows $\text{exp-cat-cf } \alpha \mathfrak{A} \mathfrak{R} : \text{cat-FUNCT } \alpha \mathfrak{C} \mathfrak{A} \mapsto_{C.tiny\beta} \text{cat-FUNCT } \alpha \mathfrak{B} \mathfrak{A}$

by

(
 rule *exp-cat-cf-is-tiny-functor*[
 OF assms Ran.HomCod.category-axioms AG.is-functor-axioms
]
)

lemma (in *is-cat-lKe*) *cat-lKe-exp-cat-cf-cat-FUNCT-is-arr*:

assumes $\mathcal{Z} \beta$ and $\alpha \in_o \beta$

shows $\text{exp-cat-cf } \alpha \mathfrak{A} \mathfrak{R} : \text{cat-FUNCT } \alpha \mathfrak{C} \mathfrak{A} \mapsto_{C.tiny\beta} \text{cat-FUNCT } \alpha \mathfrak{B} \mathfrak{A}$

by

(
 rule *exp-cat-cf-is-tiny-functor*[
 OF assms Lan.HomCod.category-axioms AG.is-functor-axioms
]
)

14.2.2 Universal property

See Chapter X-3 in [9] and [2]⁸.

lemma *is-cat-rKeI'*:

assumes $\mathfrak{R} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{C}$

and $\mathfrak{G} : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{A}$

and $\varepsilon : \mathfrak{G} \circ_{CF} \mathfrak{R} \mapsto_{CF} \mathfrak{T} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{A}$

and $\wedge \mathfrak{G}' \varepsilon'$.

$[[\mathfrak{G}' : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{A}; \varepsilon' : \mathfrak{G}' \circ_{CF} \mathfrak{R} \mapsto_{CF} \mathfrak{T} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{A}]] \implies$

$\exists ! \sigma. \sigma : \mathfrak{G}' \mapsto_{CF} \mathfrak{G} : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{A} \wedge \varepsilon' = \varepsilon \cdot_{NTCF} (\sigma \circ_{NTCF-CF} \mathfrak{R})$

shows $\varepsilon : \mathfrak{G} \circ_{CF} \mathfrak{R} \mapsto_{CF, rKe\alpha} \mathfrak{T} : \mathfrak{B} \mapsto_C \mathfrak{C} \mapsto_C \mathfrak{A}$

proof-

interpret $\mathfrak{R} : \text{is-functor } \alpha \mathfrak{B} \mathfrak{C} \mathfrak{R}$ by (rule *assms(1)*)

interpret $\mathfrak{G} : \text{is-functor } \alpha \mathfrak{C} \mathfrak{A} \mathfrak{G}$ by (rule *assms(2)*)

interpret $\varepsilon : \text{is-ntcf } \alpha \mathfrak{B} \mathfrak{A} \langle \mathfrak{G} \circ_{CF} \mathfrak{R} \rangle \mathfrak{T} \varepsilon$ by (rule *assms(3)*)

let $?\mathfrak{A}\mathfrak{R} = \langle \text{exp-cat-cf } \alpha \mathfrak{A} \mathfrak{R} \rangle$

and $? \mathfrak{T} = \langle \text{cf-map } \mathfrak{T} \rangle$

and $? \mathfrak{G} = \langle \text{cf-map } \mathfrak{G} \rangle$

show *?thesis*

proof(*intro is-cat-rKeI is-functor.universal-arrow-foI assms*)

define β where $\beta = \alpha + \omega$

have $\mathcal{Z} \beta$ and $\alpha\beta : \alpha \in_o \beta$

by (*simp-all add: \beta-def \mathfrak{R}.Z-Limit-\alpha\omega \mathfrak{R}.Z-\omega-\alpha\omega \mathcal{Z}-def \mathfrak{R}.Z-\alpha-\alpha\omega*)

then interpret $\beta : \mathcal{Z} \beta$ by *simp*

show $?\mathfrak{A}\mathfrak{R} : \text{cat-FUNCT } \alpha \mathfrak{C} \mathfrak{A} \mapsto_{C\beta} \text{cat-FUNCT } \alpha \mathfrak{B} \mathfrak{A}$

by

(

⁸https://en.wikipedia.org/wiki/Kan_extension

```

    cs-concl cs-shallow cs-intro:
      cat-small-cs-intros
      exp-cat-cf-is-tiny-functor[
        OF  $\beta$ .Z-axioms  $\alpha\beta$   $\mathfrak{G}$ .HomCod.category-axioms assms(1)
      ]
  )
from  $\alpha\beta$  assms(2) show cf-map  $\mathfrak{G} \in_{\circ}$  cat-FUNCT  $\alpha \mathfrak{C} \mathfrak{A}(\text{Obj})$ 
  unfolding cat-FUNCT-components
  by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-FUNCT-cs-intros)
from assms(1-3) show ntcf-arrow  $\varepsilon$  :
   $\mathfrak{A}\mathfrak{R}(\text{ObjMap})(\mathfrak{G}) \mapsto_{\text{cat-FUNCT } \alpha} \mathfrak{B} \mathfrak{A} \mathfrak{I}$ 
by
  (
    cs-concl cs-shallow
    cs-simp: cat-Kan-cs-simps cat-FUNCT-cs-simps cat-FUNCT-components(1)
    cs-intro: cat-FUNCT-cs-intros
  )
fix  $\mathfrak{F}' \varepsilon'$  assume prems:
   $\mathfrak{F}' \in_{\circ}$  cat-FUNCT  $\alpha \mathfrak{C} \mathfrak{A}(\text{Obj})$ 
   $\varepsilon'$  :  $\mathfrak{A}\mathfrak{R}(\text{ObjMap})(\mathfrak{F}')$   $\mapsto_{\text{cat-FUNCT } \alpha} \mathfrak{B} \mathfrak{A} \mathfrak{I}$ 
from prems(1) have  $\mathfrak{F}' \in_{\circ}$  cf-maps  $\alpha \mathfrak{C} \mathfrak{A}$ 
  unfolding cat-FUNCT-components(1) by simp
then obtain  $\mathfrak{F}$  where  $\mathfrak{F}'\text{-def}$ :  $\mathfrak{F}' = \text{cf-map } \mathfrak{F}$  and  $\mathfrak{F}$ :  $\mathfrak{F} : \mathfrak{C} \mapsto_{\text{C}\alpha} \mathfrak{A}$ 
by clarsimp
note  $\varepsilon' = \text{cat-FUNCT-is-arrD}[OF \text{prems}(2)]$ 
from  $\varepsilon'(1)$   $\mathfrak{F}$  have  $\varepsilon'\text{-is-ntcf}$ :
  ntcf-of-ntcf-arrow  $\mathfrak{B} \mathfrak{A} \varepsilon' : \mathfrak{F} \circ_{CF} \mathfrak{R} \mapsto_{CF} \mathfrak{I} : \mathfrak{B} \mapsto_{\text{C}\alpha} \mathfrak{A}$ 
by
  (
    cs-prems
    cs-simp:  $\mathfrak{F}'\text{-def}$  cat-Kan-cs-simps cat-FUNCT-cs-simps
    cs-intro: cat-cs-intros cat-FUNCT-cs-intros
  )
from assms(4)[OF  $\mathfrak{F} \varepsilon'\text{-is-ntcf}$ ] obtain  $\sigma$ 
where  $\sigma$ :  $\sigma : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{C} \mapsto_{\text{C}\alpha} \mathfrak{A}$ 
and  $\varepsilon'\text{-def}'$ : ntcf-of-ntcf-arrow  $\mathfrak{B} \mathfrak{A} \varepsilon' = \varepsilon \cdot_{NTCF} (\sigma \circ_{NTCF-CF} \mathfrak{R})$ 
and unique- $\sigma$ :  $\wedge \sigma'$ .
  [[
     $\sigma' : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{C} \mapsto_{\text{C}\alpha} \mathfrak{A}$ ;
    ntcf-of-ntcf-arrow  $\mathfrak{B} \mathfrak{A} \varepsilon' = \varepsilon \cdot_{NTCF} (\sigma' \circ_{NTCF-CF} \mathfrak{R})$ 
  ]]  $\implies \sigma' = \sigma$ 
by metis
show  $\exists ! f'$ .
   $f' : \mathfrak{F}' \mapsto_{\text{cat-FUNCT } \alpha} \mathfrak{C} \mathfrak{A} \mathfrak{I} \mathfrak{G} \wedge$ 
   $\varepsilon' = \text{umap-fo } \mathfrak{A}\mathfrak{R} \mathfrak{I} \mathfrak{I} \mathfrak{G} (\text{ntcf-arrow } \varepsilon) \mathfrak{F}'(\text{ArrVal})(f')$ 
proof(intro exI conjI; (elim conjE)?, unfold  $\mathfrak{F}'\text{-def}$ )
from  $\sigma$  show ntcf-arrow  $\sigma : \text{cf-map } \mathfrak{F} \mapsto_{\text{cat-FUNCT } \alpha} \mathfrak{C} \mathfrak{A} \mathfrak{I} \mathfrak{G}$ 
by (cs-concl cs-shallow cs-intro: cat-FUNCT-cs-intros)
from  $\alpha\beta$  assms(1-3)  $\sigma \varepsilon'(1)$  show
   $\varepsilon' = \text{umap-fo } \mathfrak{A}\mathfrak{R} \mathfrak{I} \mathfrak{I} \mathfrak{G} (\text{ntcf-arrow } \varepsilon) (\text{cf-map } \mathfrak{F})(\text{ArrVal})(\text{ntcf-arrow } \sigma)$ 
by (subst  $\varepsilon'$ )
  (
    cs-concl
    cs-simp:
       $\varepsilon'\text{-def}'$ [symmetric]
      cat-cs-simps
      cat-FUNCT-cs-simps
      cat-Kan-cs-simps
  )

```

cs-intro:
cat-small-cs-intros
cat-cs-intros
cat-Kan-cs-intros
cat-FUNCT-cs-intros
)

fix σ' **assume** *prems*:
 $\sigma' : cf\text{-map } \mathfrak{F} \mapsto_{cat\text{-FUNCT}} \alpha \mathfrak{C} \mathfrak{A} \ ?\mathfrak{G}$
 $\varepsilon' = umap\text{-fo } \mathfrak{A}\mathfrak{R} \ ?\mathfrak{T} \ ?\mathfrak{G} \ (ntcf\text{-arrow } \varepsilon) \ (cf\text{-map } \mathfrak{F}) \ (ArrVal) \ (\sigma')$
note $\sigma' = cat\text{-FUNCT-is-arrD}[OF \text{prems}(1)]$
from $\sigma'(1) \ \mathfrak{F}$ **have** *ntcf-of-ntcf-arrow* $\mathfrak{C} \ \mathfrak{A} \ \sigma' : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{A}$
by
(

cs-prems **cs-shallow**
cs-simp: *cat-FUNCT-cs-simps* **cs-intro:** *cat-cs-intros*
)

moreover from *prems(2)* *prems(1)* $\alpha\beta$ *assms(1-3)* **this** $\varepsilon'(1)$ **have**
ntcf-of-ntcf-arrow $\mathfrak{B} \ \mathfrak{A} \ \varepsilon' =$
 $\varepsilon \cdot_{NTCF} \ (ntcf\text{-of-ntcf-arrow } \mathfrak{C} \ \mathfrak{A} \ \sigma' \circ_{NTCF-CF} \mathfrak{R})$
by (*subst (asm) $\varepsilon'(2)$*)
(

cs-prems
cs-simp: *cat-Kan-cs-simps* *cat-FUNCT-cs-simps* *cat-cs-simps*
cs-intro:
cat-Kan-cs-intros
cat-small-cs-intros
cat-cs-intros
cat-FUNCT-cs-intros
)

ultimately have $\sigma\text{-def}$: $\sigma = ntcf\text{-of-ntcf-arrow } \mathfrak{C} \ \mathfrak{A} \ \sigma'$
by (*rule unique- σ [symmetric]*)
show $\sigma' = ntcf\text{-arrow } \sigma$
by (*subst $\sigma'(2)$, use nothing in $\langle subst \ \sigma\text{-def} \rangle$*)
(*cs-concl* **cs-shallow** **cs-simp:** *cat-cs-simps* **cs-intro:** *cat-cs-intros*)

qed
qed
qed

lemma *is-cat-lKeI'*:

assumes $\mathfrak{R} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{C}$

and $\mathfrak{F} : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{A}$

and $\eta : \mathfrak{T} \mapsto_{CF} \mathfrak{F} \circ_{CF} \mathfrak{R} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{A}$

and $\wedge \mathfrak{F}' \ \eta'$.

$[[\mathfrak{F}' : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{A}; \eta' : \mathfrak{T} \mapsto_{CF} \mathfrak{F}' \circ_{CF} \mathfrak{R} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{A}]] \implies$

$\exists ! \sigma. \sigma : \mathfrak{F} \mapsto_{CF} \mathfrak{F}' : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{A} \wedge \eta' = (\sigma \circ_{NTCF-CF} \mathfrak{R}) \cdot_{NTCF} \eta$

shows $\eta : \mathfrak{T} \mapsto_{CF.lKe\alpha} \mathfrak{F} \circ_{CF} \mathfrak{R} : \mathfrak{B} \mapsto_C \mathfrak{C} \mapsto_C \mathfrak{A}$

proof-

interpret \mathfrak{R} : *is-functor* $\alpha \ \mathfrak{B} \ \mathfrak{C} \ \mathfrak{R}$ **by** (*rule assms(1)*)

interpret \mathfrak{F} : *is-functor* $\alpha \ \mathfrak{C} \ \mathfrak{A} \ \mathfrak{F}$ **by** (*rule assms(2)*)

interpret η : *is-ntcf* $\alpha \ \mathfrak{B} \ \mathfrak{A} \ \mathfrak{T} \ \langle \mathfrak{F} \circ_{CF} \mathfrak{R} \rangle \ \eta$ **by** (*rule assms(3)*)

have

$\exists ! \sigma.$

$\sigma : \mathfrak{G}' \mapsto_{CF} op\text{-cf } \mathfrak{F} : op\text{-cat } \mathfrak{C} \mapsto_{C\alpha} op\text{-cat } \mathfrak{A} \wedge$

$\eta' = op\text{-ntcf } \eta \cdot_{NTCF} (\sigma \circ_{NTCF-CF} op\text{-cf } \mathfrak{R})$

if $\mathfrak{G}' : op\text{-cat } \mathfrak{C} \mapsto_{C\alpha} op\text{-cat } \mathfrak{A}$

and $\eta' : \mathfrak{G}' \circ_{CF} op\text{-cf } \mathfrak{R} \mapsto_{CF} op\text{-cf } \mathfrak{T} : op\text{-cat } \mathfrak{B} \mapsto_{C\alpha} op\text{-cat } \mathfrak{A}$

for $\mathfrak{G}' \ \eta'$

proof-

interpret \mathfrak{G}' : *is-functor* $\alpha \langle \text{op-cat } \mathfrak{C} \rangle \langle \text{op-cat } \mathfrak{A} \rangle \mathfrak{G}'$ **by** (rule *that(1)*)
interpret η' :
is-ntcf $\alpha \langle \text{op-cat } \mathfrak{B} \rangle \langle \text{op-cat } \mathfrak{A} \rangle \langle \mathfrak{G}' \circ_{CF} \text{op-cf } \mathfrak{K} \rangle \langle \text{op-cf } \mathfrak{T} \rangle \eta'$
by (rule *that(2)*)
from *assms(4)*[
OF is-functor.is-functor-op[OF that(1), unfolded cat-op-simps],
OF is-ntcf.is-ntcf-op[OF that(2), unfolded cat-op-simps]
]

obtain σ **where** $\sigma : \mathfrak{F} \mapsto_{CF} \text{op-cf } \mathfrak{G}' : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{A}$
and *op- η' -def*: $\text{op-ntcf } \eta' = \sigma \circ_{NTCF-CF} \mathfrak{K} \cdot_{NTCF} \eta$
and *unique- σ'* :
[[
 $\sigma' : \mathfrak{F} \mapsto_{CF} \text{op-cf } \mathfrak{G}' : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{A}$;
op-ntcf $\eta' = \sigma' \circ_{NTCF-CF} \mathfrak{K} \cdot_{NTCF} \eta$
]] $\implies \sigma' = \sigma$

for σ'
by *metis*

interpret σ : *is-ntcf* $\alpha \mathfrak{C} \mathfrak{A} \mathfrak{F} \langle \text{op-cf } \mathfrak{G}' \rangle \sigma$ **by** (rule σ)
show *?thesis*

proof(*intro exII conjI; (elim conjE)?*)
show *op-ntcf* $\sigma : \mathfrak{G}' \mapsto_{CF} \text{op-cf } \mathfrak{F} : \text{op-cat } \mathfrak{C} \mapsto_{C\alpha} \text{op-cat } \mathfrak{A}$
by (rule σ .*is-ntcf-op[unfolded cat-op-simps]*)
from *op- η' -def* **have** *op-ntcf* (*op-ntcf* η') = *op-ntcf* ($\sigma \circ_{NTCF-CF} \mathfrak{K} \cdot_{NTCF} \eta$)
by *simp*

from *this* σ *assms(1-3)* **show** *η' -def*:
 $\eta' = \text{op-ntcf } \eta \cdot_{NTCF} (\text{op-ntcf } \sigma \circ_{NTCF-CF} \text{op-cf } \mathfrak{K})$
by (*cs-prems* **cs-shallow** **cs-simp**: *cat-op-simps* **cs-intro**: *cat-cs-intros*)

fix σ' **assume** *prems*:
 $\sigma' : \mathfrak{G}' \mapsto_{CF} \text{op-cf } \mathfrak{F} : \text{op-cat } \mathfrak{C} \mapsto_{C\alpha} \text{op-cat } \mathfrak{A}$
 $\eta' = \text{op-ntcf } \eta \cdot_{NTCF} (\sigma' \circ_{NTCF-CF} \text{op-cf } \mathfrak{K})$

interpret σ' : *is-ntcf* $\alpha \langle \text{op-cat } \mathfrak{C} \rangle \langle \text{op-cat } \mathfrak{A} \rangle \langle \mathfrak{G}' \rangle \langle \text{op-cf } \mathfrak{F} \rangle \sigma'$
by (rule *prems(1)*)

from *prems(2)* **have**
op-ntcf $\eta' = \text{op-ntcf } (\text{op-ntcf } \eta \cdot_{NTCF} (\sigma' \circ_{NTCF-CF} \text{op-cf } \mathfrak{K}))$
by *simp*

also have $\dots = \text{op-ntcf } \sigma' \circ_{NTCF-CF} \mathfrak{K} \cdot_{NTCF} \eta$
by
(

cs-concl **cs-shallow**
cs-simp: *cat-cs-simps* *cat-op-simps*
cs-intro: *cat-cs-intros* *cat-op-intros*

)

finally have *op-ntcf* $\eta' = \text{op-ntcf } \sigma' \circ_{NTCF-CF} \mathfrak{K} \cdot_{NTCF} \eta$ **by** *simp*
from *unique- σ'* [*OF* σ' .*is-ntcf-op[unfolded cat-op-simps]* *this*] **show**
 $\sigma' = \text{op-ntcf } \sigma$
by (*auto simp: cat-op-simps*)

qed
qed
from
is-cat-rKeI'
[
OF \mathfrak{K} .*is-functor-op* \mathfrak{F} .*is-functor-op* η .*is-ntcf-op[unfolded cat-op-simps]*,
unfolded cat-op-simps,
OF this
]

interpret η : *is-cat-rKe*
 α
 $\langle \text{op-cat } \mathfrak{B} \rangle$

$\langle op\text{-}cat\ \mathfrak{C} \rangle$
 $\langle op\text{-}cat\ \mathfrak{A} \rangle$
 $\langle op\text{-}cf\ \mathfrak{R} \rangle$
 $\langle op\text{-}cf\ \mathfrak{T} \rangle$
 $\langle op\text{-}cf\ \mathfrak{F} \rangle$
 $\langle op\text{-}ntcf\ \eta \rangle$
by simp
show $\eta : \mathfrak{T} \mapsto_{CF.lKe\alpha} \mathfrak{F} \circ_{CF} \mathfrak{R} : \mathfrak{B} \mapsto_C \mathfrak{C} \mapsto_C \mathfrak{A}$
by (*rule* $\eta.is\text{-}cat\text{-}lKe\text{-}op[unfolding\ cat\text{-}op\text{-}simps]$)
qed

lemma (in *is-cat-rKe*) *cat-rKe-unique*:
assumes $\mathfrak{G}' : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{A}$ **and** $\varepsilon' : \mathfrak{G}' \circ_{CF} \mathfrak{R} \mapsto_{CF} \mathfrak{T} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{A}$
shows $\exists ! \sigma. \sigma : \mathfrak{G}' \mapsto_{CF} \mathfrak{G} : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{A} \wedge \varepsilon' = \varepsilon \cdot_{NTCF} (\sigma \circ_{NTCF-CF} \mathfrak{R})$

proof-

interpret \mathfrak{G}' : *is-functor* $\alpha\ \mathfrak{C}\ \mathfrak{A}\ \mathfrak{G}'$ **by** (*rule* *assms(1)*)
interpret ε' : *is-ntcf* $\alpha\ \mathfrak{B}\ \mathfrak{A}\ \langle \mathfrak{G}' \circ_{CF} \mathfrak{R} \rangle\ \mathfrak{T}\ \varepsilon'$ **by** (*rule* *assms(2)*)

let $? \mathfrak{T} = \langle cf\text{-}map\ \mathfrak{T} \rangle$
and $? \mathfrak{G} = \langle cf\text{-}map\ \mathfrak{G} \rangle$
and $? \mathfrak{G}' = \langle cf\text{-}map\ \mathfrak{G}' \rangle$
and $? \varepsilon = \langle ntcf\text{-}arrow\ \varepsilon \rangle$
and $? \varepsilon' = \langle ntcf\text{-}arrow\ \varepsilon' \rangle$

define β **where** $\beta = \alpha + \omega$
have $\mathcal{Z}\ \beta$ **and** $\alpha\beta : \alpha \in_{\circ} \beta$
by (*simp-all add: beta-def AG.Z-Limit- $\alpha\omega$ AG.Z- ω - $\alpha\omega$ Z-def AG.Z- α - $\alpha\omega$*)
then interpret $\beta : \mathcal{Z}\ \beta$ **by simp**

interpret $\mathfrak{A}\mathfrak{R}$: *is-tiny-functor*
 $\beta\ \langle cat\text{-}FUNCT\ \alpha\ \mathfrak{C}\ \mathfrak{A} \rangle\ \langle cat\text{-}FUNCT\ \alpha\ \mathfrak{B}\ \mathfrak{A} \rangle\ \langle exp\text{-}cat\text{-}cf\ \alpha\ \mathfrak{A}\ \mathfrak{R} \rangle$
by (*rule* *cat-rKe-exp-cat-cf-cat-FUNCT-is-arr[OF beta.Z-axioms $\alpha\beta$]*)

from *assms(1)* **have** $\mathfrak{G}' : ? \mathfrak{G}' \in_{\circ} cat\text{-}FUNCT\ \alpha\ \mathfrak{C}\ \mathfrak{A}(|Obj|)$
by
(

cs-concl **cs-shallow**
cs-simp: *cat-FUNCT-components(1)* **cs-intro**: *cat-FUNCT-cs-intros*

)

with *assms(2)* **have**
 $? \varepsilon' : exp\text{-}cat\text{-}cf\ \alpha\ \mathfrak{A}\ \mathfrak{R}(|ObjMap|)(? \mathfrak{G}') \mapsto_{cat\text{-}FUNCT\ \alpha\ \mathfrak{B}\ \mathfrak{A}} ? \mathfrak{T}$
by
(

cs-concl **cs-shallow**
cs-simp: *cat-Kan-cs-simps cat-FUNCT-cs-simps*
cs-intro: *cat-cs-intros cat-FUNCT-cs-intros*

)

from
is-functor.universal-arrow-foD(3)[
OF $\mathfrak{A}\mathfrak{R}.is\text{-}functor\text{-}axioms\ cat\text{-}rKe\text{-}ua\text{-}fo\ \mathfrak{G}'\ this$
]

obtain f' **where** $f' : f' : cf\text{-}map\ \mathfrak{G}' \mapsto_{cat\text{-}FUNCT\ \alpha\ \mathfrak{C}\ \mathfrak{A}} cf\text{-}map\ \mathfrak{G}$
and $\varepsilon'\text{-}def$: $? \varepsilon' = umap\text{-}fo\ (exp\text{-}cat\text{-}cf\ \alpha\ \mathfrak{A}\ \mathfrak{R})\ ? \mathfrak{T}\ ? \mathfrak{G}\ ? \varepsilon\ ? \mathfrak{G}'(|Arr\ Val|)(f')$
and $f'\text{-}unique$:
 \llbracket
 $f'' : ? \mathfrak{G}' \mapsto_{cat\text{-}FUNCT\ \alpha\ \mathfrak{C}\ \mathfrak{A}} ? \mathfrak{G};$

$ntcf\text{-arrow } \varepsilon' = \text{umap-fo } (\text{exp-cat-cf } \alpha \ \mathfrak{A} \ \mathfrak{K}) \ ?\mathfrak{T} \ ?\mathfrak{G} \ ?\varepsilon \ ?\mathfrak{G}'(\text{ArrVal})(f')$
 $\]] \implies f'' = f'$
for f''
by *metis*

show *?thesis*

proof(*intro ex1I conjI; (elim conjE)?*)

from ε' -def *cat-FUNCT-is-arrD(1)[OF f']* **show**

$\varepsilon' = \varepsilon \cdot_{NTCF} (\text{ntcf-of-ntcf-arrow } \mathfrak{C} \ \mathfrak{A} \ f' \circ_{NTCF-CF} \mathfrak{K})$

by (*subst (asm) cat-FUNCT-is-arrD(2)[OF f']*)

(

cs-prems **cs-shallow**

cs-simp: *cat-cs-simps cat-FUNCT-cs-simps cat-Kan-cs-simps*

cs-intro: *cat-cs-intros cat-FUNCT-cs-intros*

)

from *cat-FUNCT-is-arrD(1)[OF f']* **show** f' -is-arr:

ntcf-of-ntcf-arrow $\mathfrak{C} \ \mathfrak{A} \ f' : \mathfrak{G}' \mapsto_{CF} \mathfrak{G} : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{A}$

by

(

cs-prems **cs-shallow**

cs-simp: *cat-FUNCT-cs-simps* **cs-intro:** *cat-cs-intros*

)

fix σ **assume** *prems:*

$\sigma : \mathfrak{G}' \mapsto_{CF} \mathfrak{G} : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{A} \ \varepsilon' = \varepsilon \cdot_{NTCF} (\sigma \circ_{NTCF-CF} \mathfrak{K})$

interpret σ : *is-ntcf* $\alpha \ \mathfrak{C} \ \mathfrak{A} \ \mathfrak{G}' \ \mathfrak{G} \ \sigma$ **by** (*rule prems(1)*)

from *prems(1)* **have** σ :

ntcf-arrow $\sigma : \text{cf-map } \mathfrak{G}' \mapsto_{\text{cat-FUNCT}} \alpha \ \mathfrak{C} \ \mathfrak{A} \ \text{cf-map } \mathfrak{G}$

by (*cs-concl cs-shallow cs-intro: cat-FUNCT-cs-intros*)

from *prems* **have** ε' -def: *ntcf-arrow* $\varepsilon' =$

umap-fo (exp-cat-cf $\alpha \ \mathfrak{A} \ \mathfrak{K}) \ ?\mathfrak{T} \ ?\mathfrak{G} \ ?\varepsilon \ ?\mathfrak{G}'(\text{ArrVal})(\text{ntcf-arrow } \sigma)$

by

(

cs-concl **cs-shallow**

cs-simp: *prems(2) cat-Kan-cs-simps cat-cs-simps cat-FUNCT-cs-simps*

cs-intro: *cat-cs-intros cat-FUNCT-cs-intros*

)

show $\sigma = \text{ntcf-of-ntcf-arrow } \mathfrak{C} \ \mathfrak{A} \ f'$

unfolding *f'-unique[OF* $\sigma \ \varepsilon'$ -def, *symmetric]*

by

(

cs-concl **cs-shallow**

cs-simp: *cat-FUNCT-cs-simps* **cs-intro:** *cat-cs-intros*

)

qed

qed

lemma (*in is-cat-lKe*) *cat-lKe-unique:*

assumes $\mathfrak{F}' : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{A}$ **and** $\eta' : \mathfrak{T} \mapsto_{CF} \mathfrak{F}' \circ_{CF} \mathfrak{K} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{A}$

shows $\exists! \sigma. \sigma : \mathfrak{F} \mapsto_{CF} \mathfrak{F}' : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{A} \wedge \eta' = (\sigma \circ_{NTCF-CF} \mathfrak{K}) \cdot_{NTCF} \eta$

proof-

interpret \mathfrak{F}' : *is-functor* $\alpha \ \mathfrak{C} \ \mathfrak{A} \ \mathfrak{F}'$ **by** (*rule assms(1)*)

interpret η' : *is-ntcf* $\alpha \ \mathfrak{B} \ \mathfrak{A} \ \mathfrak{T} \ \langle \mathfrak{F}' \circ_{CF} \mathfrak{K} \rangle \ \eta'$ **by** (*rule assms(2)*)

interpret η : *is-cat-rKe*

$\alpha \ \langle \text{op-cat } \mathfrak{B} \rangle \ \langle \text{op-cat } \mathfrak{C} \rangle \ \langle \text{op-cat } \mathfrak{A} \rangle \ \langle \text{op-cf } \mathfrak{K} \rangle \ \langle \text{op-cf } \mathfrak{T} \rangle \ \langle \text{op-cf } \mathfrak{F} \rangle \ \langle \text{op-ntcf } \eta \rangle$

by (*rule is-cat-rKe-op*)

from η .*cat-rKe-unique*[*OF* \mathfrak{F}' .*is-functor-op* η' .*is-ntcf-op*[*unfolded cat-op-simps*]]
obtain σ **where** $\sigma : \sigma : \text{op-cf } \mathfrak{F}' \mapsto_{CF} \text{op-cf } \mathfrak{F} : \text{op-cat } \mathfrak{C} \mapsto_{C\alpha} \text{op-cat } \mathfrak{A}$
and η' -*def*: $\text{op-ntcf } \eta' = \text{op-ntcf } \eta \cdot_{NTCF} (\sigma \circ_{NTCF-CF} \text{op-cf } \mathfrak{K})$
and *unique- σ'* : $\wedge \sigma'$.
 \llbracket
 $\sigma' : \text{op-cf } \mathfrak{F}' \mapsto_{CF} \text{op-cf } \mathfrak{F} : \text{op-cat } \mathfrak{C} \mapsto_{C\alpha} \text{op-cat } \mathfrak{A};$
 $\text{op-ntcf } \eta' = \text{op-ntcf } \eta \cdot_{NTCF} (\sigma' \circ_{NTCF-CF} \text{op-cf } \mathfrak{K})$
 $\rrbracket \implies \sigma' = \sigma$
by *metis*

interpret σ : *is-ntcf* α $\langle \text{op-cat } \mathfrak{C} \rangle \langle \text{op-cat } \mathfrak{A} \rangle \langle \text{op-cf } \mathfrak{F}' \rangle \langle \text{op-cf } \mathfrak{F} \rangle \sigma$
by (*rule* σ)

show *?thesis*

proof(*intro exI conjI; (elim conjE)?*)

show *op-ntcf* $\sigma : \mathfrak{F} \mapsto_{CF} \mathfrak{F}' : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{A}$

by (*rule* σ .*is-ntcf-op*[*unfolded cat-op-simps*])

have $\eta' = \text{op-ntcf } (\text{op-ntcf } \eta')$

by (*cs-concl cs-shallow cs-simp: cat-op-simps*)

also from η' -*def* **have** $\dots = \text{op-ntcf } (\text{op-ntcf } \eta \cdot_{NTCF} (\sigma \circ_{NTCF-CF} \text{op-cf } \mathfrak{K}))$

by *simp*

also have $\dots = \text{op-ntcf } \sigma \circ_{NTCF-CF} \mathfrak{K} \cdot_{NTCF} \eta$

by (*cs-concl cs-shallow cs-simp: cat-op-simps cs-intro: cat-cs-intros*)

finally show $\eta' = \text{op-ntcf } \sigma \circ_{NTCF-CF} \mathfrak{K} \cdot_{NTCF} \eta$ **by** *simp*

fix σ' **assume** *prems*:

$\sigma' : \mathfrak{F} \mapsto_{CF} \mathfrak{F}' : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{A}$

$\eta' = \sigma' \circ_{NTCF-CF} \mathfrak{K} \cdot_{NTCF} \eta$

interpret σ' : *is-ntcf* α \mathfrak{C} \mathfrak{A} \mathfrak{F} \mathfrak{F}' σ' **by** (*rule* *prems*(1))

from *prems*(2) **have** $\text{op-ntcf } \eta' = \text{op-ntcf } (\sigma' \circ_{NTCF-CF} \mathfrak{K} \cdot_{NTCF} \eta)$

by *simp*

also have $\dots = \text{op-ntcf } \eta \cdot_{NTCF} (\text{op-ntcf } \sigma' \circ_{NTCF-CF} \text{op-cf } \mathfrak{K})$

by (*cs-concl cs-shallow cs-simp: cat-op-simps cs-intro: cat-cs-intros*)

finally have $\text{op-ntcf } \eta' = \text{op-ntcf } \eta \cdot_{NTCF} (\text{op-ntcf } \sigma' \circ_{NTCF-CF} \text{op-cf } \mathfrak{K})$

by *simp*

from *unique- σ'* [*OF* σ' .*is-ntcf-op this*] **show** $\sigma' = \text{op-ntcf } \sigma$

by (*auto simp: cat-op-simps*)

qed

qed

14.2.3 Further properties

lemma (in *is-cat-rKe*) *cat-rKe-ntcf-ua-fo-is-iso-ntcf-if-ge-Limit*:

assumes \mathcal{Z} β **and** $\alpha \in_o \beta$

shows

ntcf-ua-fo β (*exp-cat-cf* α \mathfrak{A} \mathfrak{K}) (*cf-map* \mathfrak{T}) (*cf-map* \mathfrak{G}) (*ntcf-arrow* ε) :

$\text{Hom}_{O.C\beta} \text{cat-FUNCT } \alpha \mathfrak{C} \mathfrak{A} (-, \text{cf-map } \mathfrak{G}) \mapsto_{CF} \text{iso}$

$\text{Hom}_{O.C\beta} \text{cat-FUNCT } \alpha \mathfrak{B} \mathfrak{A} (-, \text{cf-map } \mathfrak{T}) \circ_{CF} \text{op-cf } (\text{exp-cat-cf } \alpha \mathfrak{A} \mathfrak{K}) :$

$\text{op-cat } (\text{cat-FUNCT } \alpha \mathfrak{C} \mathfrak{A}) \mapsto_{C\beta} \text{cat-Set } \beta$

proof-

interpret \mathfrak{A} - \mathfrak{K} :

is-tiny-functor β $\langle \text{cat-FUNCT } \alpha \mathfrak{C} \mathfrak{A} \rangle \langle \text{cat-FUNCT } \alpha \mathfrak{B} \mathfrak{A} \rangle \langle \text{exp-cat-cf } \alpha \mathfrak{A} \mathfrak{K} \rangle$

by

(

rule *exp-cat-cf-is-tiny-functor*[

OF *assms Ran.HomCod.category-axioms AG.is-functor-axioms*

]

)

```

show ?thesis
  by
    (
      rule is-functor.cf-ntcf-ua-fo-is-iso-ntcf[
        OF  $\mathfrak{A}$ - $\mathfrak{K}$ .is-functor-axioms cat-rKe-ua-fo
      ]
    )
qed

lemma (in is-cat-lKe) cat-lKe-ntcf-ua-fo-is-iso-ntcf-if-ge-Limit:
  assumes  $\mathcal{Z}$   $\beta$  and  $\alpha \in_{\circ} \beta$ 
  defines  $\mathfrak{A}\mathfrak{K} \equiv \text{exp-cat-cf } \alpha \text{ (op-cat } \mathfrak{A}) \text{ (op-cf } \mathfrak{K})$ 
  and  $\mathfrak{A}\mathfrak{C} \equiv \text{cat-FUNCT } \alpha \text{ (op-cat } \mathfrak{C}) \text{ (op-cat } \mathfrak{A})$ 
  and  $\mathfrak{A}\mathfrak{B} \equiv \text{cat-FUNCT } \alpha \text{ (op-cat } \mathfrak{B}) \text{ (op-cat } \mathfrak{A})$ 
  shows
    ntcf-ua-fo  $\beta$   $\mathfrak{A}\mathfrak{K}$  (cf-map  $\mathfrak{T}$ ) (cf-map  $\mathfrak{F}$ ) (ntcf-arrow (op-ntcf  $\eta$ )) :
      Hom $_{O.C\beta}$   $\mathfrak{A}\mathfrak{C}(-, \text{cf-map } \mathfrak{F}) \mapsto_{CF.iso}$  Hom $_{O.C\beta}$   $\mathfrak{A}\mathfrak{B}(-, \text{cf-map } \mathfrak{T}) \circ_{CF}$  op-cf  $\mathfrak{A}\mathfrak{K}$  :
      op-cat  $\mathfrak{A}\mathfrak{C} \mapsto_{\circ} \text{cat-Set } \beta$ 
proof-
  note simps =  $\mathfrak{A}\mathfrak{C}$ -def  $\mathfrak{A}\mathfrak{B}$ -def  $\mathfrak{A}\mathfrak{K}$ -def
  interpret  $\mathfrak{A}$ - $\mathfrak{K}$ : is-tiny-functor  $\beta$   $\mathfrak{A}\mathfrak{C}$   $\mathfrak{A}\mathfrak{B}$   $\mathfrak{A}\mathfrak{K}$ 
  unfolding simps
  by
    (
      rule exp-cat-cf-is-tiny-functor[
        OF assms(1,2) Lan.HomCod.category-op AG.is-functor-op
      ]
    )
  show ?thesis
  unfolding simps
  by
    (
      rule is-functor.cf-ntcf-ua-fo-is-iso-ntcf[
        OF  $\mathfrak{A}$ - $\mathfrak{K}$ .is-functor-axioms[unfolded simps] cat-lKe-ua-fo
      ]
    )
qed

```

14.3 Opposite universal arrow for Kan extensions

14.3.1 Definition and elementary properties

The following definition is merely a convenience utility for the exposition of dual results associated with the formula for the right Kan extension and the pointwise right Kan extension.

definition $op\text{-}ua :: (V \Rightarrow V) \Rightarrow V \Rightarrow V \Rightarrow V$
where $op\text{-}ua \text{ lim-Obj } \mathfrak{K} \text{ } c =$
 $[$
 $\text{lim-Obj } c(\text{UObj}),$
 $op\text{-}ntcf (\text{lim-Obj } c(\text{UArr})) \circ_{NTCF-CF} \text{inv-cf } (op\text{-cf-obj-comma } \mathfrak{K} \text{ } c)$
 $]$.

Components.

lemma *op-ua-components*:
shows $[cat\text{-}op\text{-}simps]: op\text{-}ua \text{ lim-Obj } \mathfrak{K} \text{ } c(\text{UObj}) = \text{lim-Obj } c(\text{UObj})$
and $op\text{-}ua \text{ lim-Obj } \mathfrak{K} \text{ } c(\text{UArr}) =$
 $op\text{-}ntcf (\text{lim-Obj } c(\text{UArr})) \circ_{NTCF-CF} \text{inv-cf } (op\text{-cf-obj-comma } \mathfrak{K} \text{ } c)$
unfolding *op-ua-def ua-field-simps* **by** (*simp-all add: nat-omega-simps*)

14.3.2 Opposite universal arrow for Kan extensions is a limit

lemma *op-ua-UArr-is-cat-limit*:

assumes $\mathfrak{K} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{C}$

and $\mathfrak{T} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{A}$

and $c \in_{\circ} \mathfrak{C}(\text{Obj})$

and $u : \mathfrak{T} \circ_{CF} \mathfrak{K} \text{ }_{CF} \sqcap \text{ }_O c >_{CF} \text{colim } r : \mathfrak{K} \text{ }_{CF} \downarrow c \mapsto \mapsto_{C\alpha} \mathfrak{A}$

shows $op\text{-}ntcf\ u \circ_{NTCF-CF} inv\text{-}cf (op\text{-}cf\text{-}obj\text{-}comma\ \mathfrak{K}\ c) :$

$r <_{CF} \text{lim } op\text{-}cf\ \mathfrak{T} \circ_{CF} c \text{ }_O \sqcap \text{ }_{CF} (op\text{-}cf\ \mathfrak{K}) : c \downarrow_{CF} (op\text{-}cf\ \mathfrak{K}) \mapsto \mapsto_{C\alpha} op\text{-}cat\ \mathfrak{A}$

proof-

note $[cf\text{-}cs\text{-}simps] = is\text{-}iso\text{-}functor\text{-}is\text{-}iso\text{-}arr(2,3)$

let $?op\text{-}\mathfrak{K} = \langle \lambda c. op\text{-}cf\text{-}obj\text{-}comma\ \mathfrak{K}\ c \rangle$

let $?op\text{-}\mathfrak{K}c = \langle ?op\text{-}\mathfrak{K}\ c \rangle$

and $?op\text{-}ua\text{-}UArr = \langle op\text{-}ntcf\ u \circ_{NTCF-CF} inv\text{-}cf (op\text{-}cf\text{-}obj\text{-}comma\ \mathfrak{K}\ c) \rangle$

interpret \mathfrak{K} : *is-functor* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{K}$ **by** (*rule assms(1)*)

interpret \mathfrak{T} : *is-functor* $\alpha\ \mathfrak{B}\ \mathfrak{A}\ \mathfrak{T}$ **by** (*rule assms(2)*)

interpret u : *is-cat-colimit* $\alpha\ \langle \mathfrak{K} \text{ }_{CF} \downarrow c \rangle\ \mathfrak{A}\ \langle \mathfrak{T} \circ_{CF} \mathfrak{K} \text{ }_{CF} \sqcap \text{ }_O c \rangle\ r\ u$

by (*rule assms(4)*)

from $\mathfrak{K}.op\text{-}cf\text{-}cf\text{-}obj\text{-}comma\text{-}proj[OF\ assms(3)]$ **have**

$op\text{-}cf (\mathfrak{K} \text{ }_{CF} \sqcap \text{ }_O c) \circ_{CF} inv\text{-}cf (?op\text{-}\mathfrak{K}\ c) =$

$c \text{ }_O \sqcap \text{ }_{CF} (op\text{-}cf\ \mathfrak{K}) \circ_{CF} (?op\text{-}\mathfrak{K}\ c) \circ_{CF} inv\text{-}cf (?op\text{-}\mathfrak{K}\ c)$

by *simp*

from *this assms(3)* **have** [*cat-comma-cs-simps*]:

$op\text{-}cf (\mathfrak{K} \text{ }_{CF} \sqcap \text{ }_O c) \circ_{CF} inv\text{-}cf (?op\text{-}\mathfrak{K}\ c) = c \text{ }_O \sqcap \text{ }_{CF} (op\text{-}cf\ \mathfrak{K})$

by

(

cs-prems

cs-simp: *cat-cs-simps cat-comma-cs-simps cf-cs-simps cat-op-simps*

cs-intro: *cf-cs-intros cat-cs-intros cat-comma-cs-intros cat-op-intros*

)

from *assms(3)* **show** $?op\text{-}ua\text{-}UArr :$

$r <_{CF} \text{lim } op\text{-}cf\ \mathfrak{T} \circ_{CF} c \text{ }_O \sqcap \text{ }_{CF} (op\text{-}cf\ \mathfrak{K}) : c \downarrow_{CF} (op\text{-}cf\ \mathfrak{K}) \mapsto \mapsto_{C\alpha} op\text{-}cat\ \mathfrak{A}$

by

(

cs-concl

cs-simp:

cf-cs-simps cat-cs-simps cat-comma-cs-simps cat-op-simps

$\mathfrak{K}.op\text{-}cf\text{-}cf\text{-}obj\text{-}comma\text{-}proj[symmetric]$

cs-intro:

cat-cs-intros

cf-cs-intros

cat-lim-cs-intros

cat-comma-cs-intros

cat-op-intros

)

qed

context

fixes $lim\text{-}Obj :: V \Rightarrow V$ **and** $c :: V$

begin

lemmas $op\text{-}ua\text{-}UArr\text{-}is\text{-}cat\text{-}limit' = op\text{-}ua\text{-}UArr\text{-}is\text{-}cat\text{-}limit$

[

$unf\text{-}op\text{-}ua\text{-}components(2)[\textit{symmetric}],$
where $u = \langle \textit{lim-Obj } c(\textit{UArr}) \rangle$ **and** $r = \langle \textit{lim-Obj } c(\textit{UObj}) \rangle$ **and** $c = c,$
 $folded\ op\text{-}ua\text{-}components(2)[\textit{where } \textit{lim-Obj} = \textit{lim-Obj} \textit{ and } c = c]$
 $]$

end

14.4 The Kan extension

The following subsection is based on the statement and proof of Theorem 1 in Chapter X-3 in [9].

14.4.1 Definition and elementary properties

definition $the\text{-}cf\text{-}rKe :: V \Rightarrow V \Rightarrow V \Rightarrow (V \Rightarrow V) \Rightarrow V$

where $the\text{-}cf\text{-}rKe \alpha \mathfrak{I} \mathfrak{K} \textit{lim-Obj} =$

$[$
 $(\lambda c \in_o \mathfrak{K}(\textit{HomCod})(\textit{Obj}). \textit{lim-Obj } c(\textit{UObj})),$
 $($
 $\lambda g \in_o \mathfrak{K}(\textit{HomCod})(\textit{Arr}). \textit{THE } f.$
 $f :$
 $\textit{lim-Obj } (\mathfrak{K}(\textit{HomCod})(\textit{Dom})(g))(\textit{UObj}) \mapsto_{\mathfrak{I}(\textit{HomCod})}$
 $\textit{lim-Obj } (\mathfrak{K}(\textit{HomCod})(\textit{Cod})(g))(\textit{UObj}) \wedge$
 $\textit{lim-Obj } (\mathfrak{K}(\textit{HomCod})(\textit{Dom})(g))(\textit{UArr}) \circ_{NTCF-CF} g \downarrow_{CF} \mathfrak{K} =$
 $\textit{lim-Obj } (\mathfrak{K}(\textit{HomCod})(\textit{Cod})(g))(\textit{UArr}) \cdot_{NTCF}$
 $\textit{ntcf-const } ((\mathfrak{K}(\textit{HomCod})(\textit{Cod})(g)) \downarrow_{CF} \mathfrak{K}) (\mathfrak{I}(\textit{HomCod})) f$
 $),$
 $\mathfrak{K}(\textit{HomCod}),$
 $\mathfrak{I}(\textit{HomCod})$
 $].$

definition $the\text{-}ntcf\text{-}rKe :: V \Rightarrow V \Rightarrow V \Rightarrow (V \Rightarrow V) \Rightarrow V$

where $the\text{-}ntcf\text{-}rKe \alpha \mathfrak{I} \mathfrak{K} \textit{lim-Obj} =$

$[$
 $($
 $\lambda c \in_o \mathfrak{I}(\textit{HomDom})(\textit{Obj}).$
 $\textit{lim-Obj } (\mathfrak{K}(\textit{ObjMap})(c))(\textit{UArr})(\textit{NTMap})(\textit{Id}, c, \mathfrak{K}(\textit{HomCod})(\textit{CId})(\mathfrak{K}(\textit{ObjMap})(c)))$
 $),$
 $the\text{-}cf\text{-}rKe \alpha \mathfrak{I} \mathfrak{K} \textit{lim-Obj} \circ_{CF} \mathfrak{K},$
 $\mathfrak{I},$
 $\mathfrak{I}(\textit{HomDom}),$
 $\mathfrak{I}(\textit{HomCod})$
 $].$

definition $the\text{-}cf\text{-}lKe :: V \Rightarrow V \Rightarrow V \Rightarrow (V \Rightarrow V) \Rightarrow V$

where $the\text{-}cf\text{-}lKe \alpha \mathfrak{I} \mathfrak{K} \textit{lim-Obj} =$

$op\text{-}cf (the\text{-}cf\text{-}rKe \alpha (op\text{-}cf \mathfrak{I}) (op\text{-}cf \mathfrak{K}) (op\text{-}ua \textit{lim-Obj } \mathfrak{K}))$

definition $the\text{-}ntcf\text{-}lKe :: V \Rightarrow V \Rightarrow V \Rightarrow (V \Rightarrow V) \Rightarrow V$

where $the\text{-}ntcf\text{-}lKe \alpha \mathfrak{I} \mathfrak{K} \textit{lim-Obj} =$

$op\text{-}ntcf (the\text{-}ntcf\text{-}rKe \alpha (op\text{-}cf \mathfrak{I}) (op\text{-}cf \mathfrak{K}) (op\text{-}ua \textit{lim-Obj } \mathfrak{K}))$

Components.

lemma $the\text{-}cf\text{-}rKe\text{-}components:$

shows $the\text{-}cf\text{-}rKe \alpha \mathfrak{I} \mathfrak{K} \textit{lim-Obj}(\textit{ObjMap}) =$

$(\lambda c \in_o \mathfrak{K}(\textit{HomCod})(\textit{Obj}). \textit{lim-Obj } c(\textit{UObj}))$

and $the\text{-}cf\text{-}rKe \alpha \mathfrak{I} \mathfrak{K} \textit{lim-Obj}(\textit{ArrMap}) =$

(
 $\lambda g \in_{\circ} \mathfrak{K}(\text{HomCod})(\downarrow \text{Arr})$. THE f .
 f :
 $\text{lim-Obj}(\mathfrak{K}(\text{HomCod})(\downarrow \text{Dom})(\downarrow g))(\downarrow \text{UObj}) \mapsto_{\mathfrak{T}}(\text{HomCod})$
 $\text{lim-Obj}(\mathfrak{K}(\text{HomCod})(\downarrow \text{Cod})(\downarrow g))(\downarrow \text{UObj}) \wedge$
 $\text{lim-Obj}(\mathfrak{K}(\text{HomCod})(\downarrow \text{Dom})(\downarrow g))(\downarrow \text{UArr}) \circ_{NTCF-CF} g \downarrow_{CF} \mathfrak{K} =$
 $\text{lim-Obj}(\mathfrak{K}(\text{HomCod})(\downarrow \text{Cod})(\downarrow g))(\downarrow \text{UArr}) \cdot_{NTCF}$
 $\text{ntcf-const}((\mathfrak{K}(\text{HomCod})(\downarrow \text{Cod})(\downarrow g)) \downarrow_{CF} \mathfrak{K})(\mathfrak{T}(\text{HomCod})) f$
)
and $\text{the-cf-rKe} \alpha \mathfrak{T} \mathfrak{K} \text{lim-Obj}(\text{HomDom}) = \mathfrak{K}(\text{HomCod})$
and $\text{the-cf-rKe} \alpha \mathfrak{T} \mathfrak{K} \text{lim-Obj}(\text{HomCod}) = \mathfrak{T}(\text{HomCod})$
unfolding $\text{the-cf-rKe-def dghm-field-simps}$ **by** ($\text{simp-all add: nat-omega-simps}$)

lemma $\text{the-ntcf-rKe-components}$:

shows $\text{the-ntcf-rKe} \alpha \mathfrak{T} \mathfrak{K} \text{lim-Obj}(\text{NTMap}) =$

(
 $\lambda c \in_{\circ} \mathfrak{T}(\text{HomDom})(\downarrow \text{Obj})$.
 $\text{lim-Obj}(\mathfrak{K}(\text{ObjMap})(\downarrow c))(\downarrow \text{UArr})(\downarrow \text{NTMap})(\downarrow 0, c, \mathfrak{K}(\text{HomCod})(\downarrow \text{CId})(\downarrow \mathfrak{K}(\text{ObjMap})(\downarrow c)))$.
)
and $\text{the-ntcf-rKe} \alpha \mathfrak{T} \mathfrak{K} \text{lim-Obj}(\text{NTDom}) = \text{the-cf-rKe} \alpha \mathfrak{T} \mathfrak{K} \text{lim-Obj} \circ_{CF} \mathfrak{K}$
and $\text{the-ntcf-rKe} \alpha \mathfrak{T} \mathfrak{K} \text{lim-Obj}(\text{NTCod}) = \mathfrak{T}$
and $\text{the-ntcf-rKe} \alpha \mathfrak{T} \mathfrak{K} \text{lim-Obj}(\text{NTDGDom}) = \mathfrak{T}(\text{HomDom})$
and $\text{the-ntcf-rKe} \alpha \mathfrak{T} \mathfrak{K} \text{lim-Obj}(\text{NTDGCod}) = \mathfrak{T}(\text{HomCod})$
unfolding $\text{the-ntcf-rKe-def nt-field-simps}$ **by** ($\text{simp-all add: nat-omega-simps}$)

context

fixes $\alpha \mathfrak{A} \mathfrak{B} \mathfrak{C} \mathfrak{K} \mathfrak{T}$

assumes $\mathfrak{K}: \mathfrak{K} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{C}$

and $\mathfrak{T}: \mathfrak{T} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{A}$

begin

interpretation \mathfrak{K} : $\text{is-functor} \alpha \mathfrak{B} \mathfrak{C} \mathfrak{K}$ **by** ($\text{rule } \mathfrak{K}$)

interpretation \mathfrak{T} : $\text{is-functor} \alpha \mathfrak{B} \mathfrak{A} \mathfrak{T}$ **by** ($\text{rule } \mathfrak{T}$)

lemmas $\text{the-cf-rKe-components}' = \text{the-cf-rKe-components}$ [
where $\mathfrak{K}=\mathfrak{K}$ **and** $\mathfrak{T}=\mathfrak{T}$ **and** $\alpha=\alpha$, $\text{unfolded } \mathfrak{K}.cf\text{-HomCod } \mathfrak{T}.cf\text{-HomCod}$
]

lemmas $[\text{cat-Kan-cs-simps}] = \text{the-cf-rKe-components}'(3,4)$

lemmas $\text{the-ntcf-rKe-components}' = \text{the-ntcf-rKe-components}$ [
where $\mathfrak{K}=\mathfrak{K}$ **and** $\mathfrak{T}=\mathfrak{T}$ **and** $\alpha=\alpha$, $\text{unfolded } \mathfrak{K}.cf\text{-HomCod } \mathfrak{T}.cf\text{-HomCod } \mathfrak{T}.cf\text{-HomDom}$
]

lemmas $[\text{cat-Kan-cs-simps}] = \text{the-ntcf-rKe-components}'(2-5)$

end

14.4.2 Functor: object map

mk-VLambda $\text{the-cf-rKe-components}(1)$
 $[\text{vsu } \text{the-cf-rKe-ObjMap-vsuv}[\text{cat-Kan-cs-intros}]]$

context

fixes $\alpha \mathfrak{A} \mathfrak{B} \mathfrak{C} \mathfrak{K} \mathfrak{T}$

assumes $\mathfrak{K}: \mathfrak{K} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{C}$

and $\mathfrak{T}: \mathfrak{T} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{A}$

begin

interpretation \mathfrak{K} : *is-functor* $\alpha \mathfrak{B} \mathfrak{C} \mathfrak{K}$ by (rule \mathfrak{K})

mk-VLambda *the-cf-rKe-components'(1)*[*OF* \mathfrak{K} \mathfrak{T}]
 |*vdomain the-cf-rKe-ObjMap-vdomain*[*cat-Kan-cs-simps*]
 |*app the-cf-rKe-ObjMap-impl-app*[*cat-Kan-cs-simps*]

lemma *the-cf-rKe-ObjMap-vrange*:
assumes $\bigwedge c. c \in \mathfrak{C}(\text{Obj}) \implies \text{lim-Obj } c(\text{UObj}) \in \mathfrak{A}(\text{Obj})$
shows \mathcal{R}_\circ (*the-cf-rKe* α \mathfrak{T} \mathfrak{K} *lim-Obj*(*ObjMap*)) $\subseteq \mathfrak{A}(\text{Obj})$
unfolding *the-cf-rKe-components'*[*OF* \mathfrak{K} \mathfrak{T}]
by (*intro vrange-VLambda-vsubset assms*)

end

14.4.3 Functor: arrow map

mk-VLambda *the-cf-rKe-components(2)*
 |*vsu the-cf-rKe-ArrMap-vsu*[*cat-Kan-cs-intros*]

context

fixes $\alpha \mathfrak{B} \mathfrak{C} \mathfrak{K}$
assumes $\mathfrak{K}: \mathfrak{K} : \mathfrak{B} \mapsto \mathfrak{C}_\alpha \mathfrak{C}$

begin

interpretation \mathfrak{K} : *is-functor* $\alpha \mathfrak{B} \mathfrak{C} \mathfrak{K}$ by (rule \mathfrak{K})

mk-VLambda *the-cf-rKe-components(2)*[**where** $\alpha=\alpha$ **and** $\mathfrak{K}=\mathfrak{K}$, *unfolded* \mathfrak{K} .*cf-HomCod*]
 |*vdomain the-cf-rKe-ArrMap-vdomain*[*cat-Kan-cs-simps*]

context

fixes $\mathfrak{A} \mathfrak{T} c c' g$
assumes $\mathfrak{T}: \mathfrak{T} : \mathfrak{B} \mapsto \mathfrak{C}_\alpha \mathfrak{A}$
and $g: g : c \mapsto_{\mathfrak{C}} c'$

begin

interpretation \mathfrak{T} : *is-functor* $\alpha \mathfrak{B} \mathfrak{A} \mathfrak{T}$ by (rule \mathfrak{T})

lemma $g': g \in \mathfrak{C}(\text{Arr})$ **using** g **by** *auto*

mk-VLambda *the-cf-rKe-components(2)*[
where $\alpha=\alpha$ **and** $\mathfrak{K}=\mathfrak{K}$ **and** $\mathfrak{T}=\mathfrak{T}$, *unfolded* \mathfrak{K} .*cf-HomCod* \mathfrak{T} .*cf-HomCod*
]
 |*app the-cf-rKe-ArrMap-app-impl'*]

lemmas *the-cf-rKe-ArrMap-app' = the-cf-rKe-ArrMap-app-impl'*[
OF g' , *unfolded* \mathfrak{K} .*HomCod.cat-is-arrD*[*OF* g]
]

end

end

lemma *the-cf-rKe-ArrMap-app-impl*:

assumes $\mathfrak{K} : \mathfrak{B} \mapsto \mathfrak{C}_\alpha \mathfrak{C}$
and $\mathfrak{T} : \mathfrak{B} \mapsto \mathfrak{C}_\alpha \mathfrak{A}$
and $g : c \mapsto_{\mathfrak{C}} c'$
and $u : r <_{CF} \text{lim } \mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} : c \downarrow_{CF} \mathfrak{K} \mapsto \mathfrak{C}_\alpha \mathfrak{A}$

and $u' : r' <_{CF.lim} \mathfrak{T} \circ_{CF} c' \circ \square_{CF} \mathfrak{K} : c' \downarrow_{CF} \mathfrak{K} \mapsto_{C\alpha} \mathfrak{A}$
shows $\exists! f$.
 $f : r \mapsto_{\mathfrak{A}} r' \wedge$
 $u \circ_{NTCF-CF} g \downarrow_{CF} \mathfrak{K} = u' \cdot_{NTCF} ntcf-const (c' \downarrow_{CF} \mathfrak{K}) \mathfrak{A} f$
proof-

interpret \mathfrak{K} : *is-functor* $\alpha \mathfrak{B} \mathfrak{C} \mathfrak{K}$ **by** (*rule assms(1)*)
interpret \mathfrak{T} : *is-functor* $\alpha \mathfrak{B} \mathfrak{A} \mathfrak{T}$ **by** (*rule assms(2)*)
interpret u : *is-cat-limit* $\alpha \langle c \downarrow_{CF} \mathfrak{K} \rangle \mathfrak{A} \langle \mathfrak{T} \circ_{CF} c \circ \square_{CF} \mathfrak{K} \rangle r u$
by (*rule assms(4)*)
interpret u' : *is-cat-limit* $\alpha \langle c' \downarrow_{CF} \mathfrak{K} \rangle \mathfrak{A} \langle \mathfrak{T} \circ_{CF} c' \circ \square_{CF} \mathfrak{K} \rangle r' u'$
by (*rule assms(5)*)

have *const-r-def*:

$cf-const (c' \downarrow_{CF} \mathfrak{K}) \mathfrak{A} r = cf-const (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} r \circ_{CF} g \downarrow_{CF} \mathfrak{K}$

proof(*rule cf-eqI*)

show *const-r*: $cf-const (c' \downarrow_{CF} \mathfrak{K}) \mathfrak{A} r : c' \downarrow_{CF} \mathfrak{K} \mapsto_{C\alpha} \mathfrak{A}$
by (*cs-concl cs-intro: cat-cs-intros cat-lim-cs-intros*)

from *assms(3)* **show** *const-r-gK*:

$cf-const (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} r \circ_{CF} g \downarrow_{CF} \mathfrak{K} : c' \downarrow_{CF} \mathfrak{K} \mapsto_{C\alpha} \mathfrak{A}$
by (*cs-concl cs-intro: cat-cs-intros cat-comma-cs-intros*)

have *ObjMap-dom-lhs*: $\mathcal{D}_\circ (cf-const (c' \downarrow_{CF} \mathfrak{K}) \mathfrak{A} r(\downarrow_{ObjMap})) = c' \downarrow_{CF} \mathfrak{K}(\downarrow_{Obj})$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

from *assms(3)* **have** *ObjMap-dom-rhs*:

$\mathcal{D}_\circ ((cf-const (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} r \circ_{CF} g \downarrow_{CF} \mathfrak{K})(\downarrow_{ObjMap})) = c' \downarrow_{CF} \mathfrak{K}(\downarrow_{Obj})$
by

(
cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-lim-cs-intros cat-cs-intros cat-comma-cs-intros*
)

have *ArrMap-dom-lhs*: $\mathcal{D}_\circ (cf-const (c' \downarrow_{CF} \mathfrak{K}) \mathfrak{A} r(\downarrow_{ArrMap})) = c' \downarrow_{CF} \mathfrak{K}(\downarrow_{Arr})$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

from *assms(3)* **have** *ArrMap-dom-rhs*:

$\mathcal{D}_\circ ((cf-const (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} r \circ_{CF} g \downarrow_{CF} \mathfrak{K})(\downarrow_{ArrMap})) = c' \downarrow_{CF} \mathfrak{K}(\downarrow_{Arr})$
by

(
cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-lim-cs-intros cat-cs-intros cat-comma-cs-intros*
)

show

$cf-const (c' \downarrow_{CF} \mathfrak{K}) \mathfrak{A} r(\downarrow_{ObjMap}) =$
 $(cf-const (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} r \circ_{CF} g \downarrow_{CF} \mathfrak{K})(\downarrow_{ObjMap})$

proof(*rule vsu-eqI, unfold ObjMap-dom-lhs ObjMap-dom-rhs*)

fix A **assume** *prems*: $A \in_\circ c' \downarrow_{CF} \mathfrak{K}(\downarrow_{Obj})$

from *prems assms* **obtain** $b f$

where *A-def*: $A = [0, b, f]$.

and $b : b \in_\circ \mathfrak{B}(\downarrow_{Obj})$

and $f : f : c' \mapsto_{\mathfrak{C}} \mathfrak{K}(\downarrow_{ObjMap})(\downarrow b)$

by *auto*

from *assms(1,3)* *prems f b* **show**

$cf-const (c' \downarrow_{CF} \mathfrak{K}) \mathfrak{A} r(\downarrow_{ObjMap})(\downarrow A) =$
 $(cf-const (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} r \circ_{CF} g \downarrow_{CF} \mathfrak{K})(\downarrow_{ObjMap})(\downarrow A)$

unfolding *A-def*

by

(
cs-concl
)

cs-simp: *cat-cs-simps cat-comma-cs-simps*
cs-intro: *cat-lim-cs-intros cat-cs-intros cat-comma-cs-intros*

)

qed

(

 use *assms*(β) in

 ⟨*cs-concl cs-shallow cs-intro: cat-cs-intros cat-comma-cs-intros*⟩

)+

show

cf-const ($c' \downarrow_{CF} \mathfrak{K}$) $\mathfrak{A} r(\text{ArrMap}) =$

 (*cf-const* ($c \downarrow_{CF} \mathfrak{K}$) $\mathfrak{A} r \circ_{CF} g \downarrow_{CF} \mathfrak{K}$)(*ArrMap*)

proof(*rule vsu-eqI, unfold ArrMap-dom-lhs ArrMap-dom-rhs*)

 show *vsu* (*cf-const* ($c' \downarrow_{CF} \mathfrak{K}$) $\mathfrak{A} r(\text{ArrMap})$)

 by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

 from *assms*(β) show *vsu* ((*cf-const* ($c \downarrow_{CF} \mathfrak{K}$) $\mathfrak{A} r \circ_{CF} g \downarrow_{CF} \mathfrak{K}$)(*ArrMap*))

 by (*cs-concl cs-shallow cs-intro: cat-cs-intros cat-comma-cs-intros*)

 fix F assume *prems*: $F \in_{\circ} c' \downarrow_{CF} \mathfrak{K}(\text{Arr})$

 with *prems* obtain $A B$ where $F: F: A \mapsto_{c' \downarrow_{CF} \mathfrak{K}} B$

 by (*auto intro: is-arrI*)

 with *assms* obtain $b f b' f' h'$

 where $F\text{-def}$: $F = [[0, b, f]_{\circ}, [0, b', f']_{\circ}, [0, h']_{\circ}]_{\circ}$.

 and $A\text{-def}$: $A = [0, b, f]_{\circ}$

 and $B\text{-def}$: $B = [0, b', f']_{\circ}$

 and h' : $h': b \mapsto_{\mathfrak{B}} b'$

 and f : $f: c' \mapsto_{\mathfrak{C}} \mathfrak{K}(\text{ObjMap})(b)$

 and f' : $f': c' \mapsto_{\mathfrak{C}} \mathfrak{K}(\text{ObjMap})(b')$

 and $f'\text{-def}$: $\mathfrak{K}(\text{ArrMap})(h') \circ_{A\mathfrak{C}} f = f'$

 by *auto*

 from *prems assms*(β) $F g' h' f f'$ show

cf-const ($c' \downarrow_{CF} \mathfrak{K}$) $\mathfrak{A} r(\text{ArrMap})(F) =$

 (*cf-const* ($c \downarrow_{CF} \mathfrak{K}$) $\mathfrak{A} r \circ_{CF} g \downarrow_{CF} \mathfrak{K}$)(*ArrMap*)(F)

 unfolding $F\text{-def} A\text{-def} B\text{-def}$

 by

 (

cs-concl

cs-simp: *cat-comma-cs-simps cat-cs-simps f'-def[symmetric]*

cs-intro: *cat-lim-cs-intros cat-cs-intros cat-comma-cs-intros*

)

 qed *simp*

qed *simp-all*

have $\mathfrak{T}c'\mathfrak{K}: \mathfrak{T} \circ_{CF} c' \circ \sqcap_{CF} \mathfrak{K} = \mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} \circ_{CF} g \downarrow_{CF} \mathfrak{K}$

proof(*rule cf-eqI*)

 show $\mathfrak{T} \circ_{CF} c' \circ \sqcap_{CF} \mathfrak{K}: c' \downarrow_{CF} \mathfrak{K} \mapsto_{C\alpha} \mathfrak{A}$

 by (*cs-concl cs-shallow cs-intro: cat-cs-intros*)

 from *assms* show $\mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} \circ_{CF} g \downarrow_{CF} \mathfrak{K}: c' \downarrow_{CF} \mathfrak{K} \mapsto_{C\alpha} \mathfrak{A}$

 by

 (

cs-concl

cs-simp: *cat-cs-simps*

cs-intro: *cat-comma-cs-intros cat-cs-intros*

)

 have *ObjMap-dom-lhs*: $\mathcal{D}_{\circ} ((\mathfrak{T} \circ_{CF} c' \circ \sqcap_{CF} \mathfrak{K})(\text{ObjMap})) = c' \downarrow_{CF} \mathfrak{K}(\text{Obj})$

 by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

 from *assms* have *ObjMap-dom-rhs*:

$\mathcal{D}_{\circ} ((\mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} \circ_{CF} g \downarrow_{CF} \mathfrak{K})(\text{ObjMap})) = c' \downarrow_{CF} \mathfrak{K}(\text{Obj})$

 by

 (

```

    cs-concl
    cs-simp: cat-cs-simps
    cs-intro: cat-comma-cs-intros cat-cs-intros
  )
show  $(\mathfrak{T} \circ_{CF} c' \circ \sqcap_{CF} \mathfrak{K})(ObjMap) = (\mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} \circ_{CF} g \downarrow_{CF} \mathfrak{K})(ObjMap)$ 
proof(rule vsv-eqI, unfold ObjMap-dom-lhs ObjMap-dom-rhs)
  from assms show vsv  $((\mathfrak{T} \circ_{CF} c' \circ \sqcap_{CF} \mathfrak{K})(ObjMap))$ 
  by
  (
    cs-concl cs-shallow
    cs-simp: cat-comma-cs-simps
    cs-intro: cat-cs-intros cat-comma-cs-intros
  )
  from assms show vsv  $((\mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} \circ_{CF} g \downarrow_{CF} \mathfrak{K})(ObjMap))$ 
  by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-comma-cs-intros)
fix A assume prems:  $A \in_0 c' \downarrow_{CF} \mathfrak{K}(Obj)$ 
from assms (3) prems obtain b f
  where A-def:  $A = [0, b, f]$ .
  and b:  $b \in_0 \mathfrak{B}(Obj)$ 
  and f:  $f : c' \mapsto_{\mathfrak{C}} \mathfrak{K}(ObjMap)(b)$ 
  by auto
from prems assms b f show
 $(\mathfrak{T} \circ_{CF} c' \circ \sqcap_{CF} \mathfrak{K})(ObjMap)(A) =$ 
 $(\mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} \circ_{CF} g \downarrow_{CF} \mathfrak{K})(ObjMap)(A)$ 
  unfolding A-def
  by
  (
    cs-concl
    cs-simp: cat-cs-simps cat-comma-cs-simps
    cs-intro: cat-cs-intros cat-comma-cs-intros
  )
qed simp

have ArrMap-dom-lhs:  $\mathcal{D}_0((\mathfrak{T} \circ_{CF} c' \circ \sqcap_{CF} \mathfrak{K})(ArrMap)) = c' \downarrow_{CF} \mathfrak{K}(Arr)$ 
  by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
from assms have ArrMap-dom-rhs:
 $\mathcal{D}_0((\mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} \circ_{CF} g \downarrow_{CF} \mathfrak{K})(ArrMap)) = c' \downarrow_{CF} \mathfrak{K}(Arr)$ 
  by
  (
    cs-concl
    cs-simp: cat-cs-simps
    cs-intro: cat-comma-cs-intros cat-cs-intros
  )

show  $(\mathfrak{T} \circ_{CF} c' \circ \sqcap_{CF} \mathfrak{K})(ArrMap) = (\mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} \circ_{CF} g \downarrow_{CF} \mathfrak{K})(ArrMap)$ 
proof(rule vsv-eqI, unfold ArrMap-dom-lhs ArrMap-dom-rhs)
  from assms show vsv  $((\mathfrak{T} \circ_{CF} c' \circ \sqcap_{CF} \mathfrak{K})(ArrMap))$ 
  by
  (
    cs-concl cs-shallow
    cs-simp: cat-comma-cs-simps
    cs-intro: cat-cs-intros cat-comma-cs-intros
  )
  from assms show vsv  $((\mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} \circ_{CF} g \downarrow_{CF} \mathfrak{K})(ArrMap))$ 
  by
  (
    cs-concl cs-shallow
    cs-simp: cs-intro: cat-cs-intros cat-comma-cs-intros
  )

```

)

fix F **assume** $\text{prems}: F \in_0 c' \downarrow_{CF} \mathfrak{K}(\text{Arr})$
with prems **obtain** $A B$ **where** $F: F: A \mapsto_{c' \downarrow_{CF} \mathfrak{K}} B$
unfolding $\text{cat-comma-cs-simps}$ **by** ($\text{auto intro: is-arrI}$)
with $\text{assms}(\mathcal{B})$ **obtain** $b f b' f' h'$
where $F\text{-def}: F = [[0, b, f]_0, [0, b', f']_0, [0, h']_0]$
and $A\text{-def}: A = [0, b, f]_0$
and $B\text{-def}: B = [0, b', f']_0$
and $h': h': b \mapsto_{\mathfrak{B}} b'$
and $f: f: c' \mapsto_{\mathfrak{C}} \mathfrak{K}(\text{ObjMap})(b)$
and $f': f': c' \mapsto_{\mathfrak{C}} \mathfrak{K}(\text{ObjMap})(b')$
and $f'\text{-def}: \mathfrak{K}(\text{ArrMap})(h') \circ_{A\mathfrak{C}} f = f'$
by auto
from prems $\text{assms}(\mathcal{B})$ $F g' h' f f'$ **show**
 $(\mathfrak{T} \circ_{CF} c' \circ \square_{CF} \mathfrak{K})(\text{ArrMap})(F) =$
 $(\mathfrak{T} \circ_{CF} c \circ \square_{CF} \mathfrak{K} \circ_{CF} g \downarrow_{CF} \mathfrak{K})(\text{ArrMap})(F)$
unfolding $F\text{-def}$ $A\text{-def}$ $B\text{-def}$
by
 (
 cs-concl
 cs-simp: $\text{cat-comma-cs-simps}$ cat-cs-simps $f'\text{-def}$ [symmetric]
 cs-intro: cat-lim-cs-intros cat-cs-intros $\text{cat-comma-cs-intros}$
)

qed simp

qed simp-all

from $\text{assms}(1-3)$ **have**

$u \circ_{NTCF-CF} g \downarrow_{CF} \mathfrak{K}: r <_{CF.cone} \mathfrak{T} \circ_{CF} c' \circ \square_{CF} \mathfrak{K}: c' \downarrow_{CF} \mathfrak{K} \mapsto_{C\alpha} \mathfrak{A}$

by ($\text{intro is-cat-coneI}$)

(
 cs-concl
 cs-intro: cat-cs-intros $\text{cat-comma-cs-intros}$ cat-lim-cs-intros
 cs-simp: const-r-def $\mathfrak{T} c' \mathfrak{K}$
)+

with $u'.\text{cat-lim-ua-fo}$ **show**

$\exists! G.$

$G: r \mapsto_{\mathfrak{A}} r' \wedge$

$u \circ_{NTCF-CF} g \downarrow_{CF} \mathfrak{K} = u' \cdot_{NTCF} \text{ntcf-const}(c' \downarrow_{CF} \mathfrak{K}) \mathfrak{A} G$

by simp

qed

lemma $\text{the-cf-rKe-ArrMap-app}$:

assumes $\mathfrak{K}: \mathfrak{B} \mapsto_{C\alpha} \mathfrak{C}$

and $\mathfrak{T}: \mathfrak{B} \mapsto_{C\alpha} \mathfrak{A}$

and $g: c \mapsto_{\mathfrak{C}} c'$

and $\text{lim-Obj } c(\text{UArr}):$

$\text{lim-Obj } c(\text{UObj}) <_{CF.lim} \mathfrak{T} \circ_{CF} c \circ \square_{CF} \mathfrak{K}: c \downarrow_{CF} \mathfrak{K} \mapsto_{C\alpha} \mathfrak{A}$

and $\text{lim-Obj } c'(\text{UArr}):$

$\text{lim-Obj } c'(\text{UObj}) <_{CF.lim} \mathfrak{T} \circ_{CF} c' \circ \square_{CF} \mathfrak{K}: c' \downarrow_{CF} \mathfrak{K} \mapsto_{C\alpha} \mathfrak{A}$

shows $\text{the-cf-rKe } \alpha \mathfrak{T} \mathfrak{K} \text{ lim-Obj}(\text{ArrMap})(g):$

$\text{lim-Obj } c(\text{UObj}) \mapsto_{\mathfrak{A}} \text{lim-Obj } c'(\text{UObj})$

and

$\text{lim-Obj } c(\text{UArr}) \circ_{NTCF-CF} g \downarrow_{CF} \mathfrak{K} =$

$\text{lim-Obj } c'(\text{UArr}) \cdot_{NTCF}$

$\text{ntcf-const}(c' \downarrow_{CF} \mathfrak{K}) \mathfrak{A} (\text{the-cf-rKe } \alpha \mathfrak{T} \mathfrak{K} \text{ lim-Obj}(\text{ArrMap})(g))$

and

$$\begin{aligned} & \llbracket \\ & f : \text{lim-Obj } c(\text{UObj}) \mapsto_{\mathfrak{A}} \text{lim-Obj } c'(\text{UObj}); \\ & \text{lim-Obj } c(\text{UArr}) \circ_{NTCF-CF} g \downarrow_{CF} \mathfrak{K} = \\ & \quad \text{lim-Obj } c'(\text{UArr}) \cdot_{NTCF} \text{ntcf-const } (c' \downarrow_{CF} \mathfrak{K}) \mathfrak{A} f \\ & \rrbracket \implies f = \text{the-cf-rKe } \alpha \mathfrak{T} \mathfrak{K} \text{lim-Obj}(\text{ArrMap})(g) \end{aligned}$$

proof-

interpret \mathfrak{K} : *is-functor* $\alpha \mathfrak{B} \mathfrak{C} \mathfrak{K}$ **by** (rule *assms(1)*)

interpret \mathfrak{T} : *is-functor* $\alpha \mathfrak{B} \mathfrak{A} \mathfrak{T}$ **by** (rule *assms(2)*)

interpret u : *is-cat-limit*

$\alpha \langle c \downarrow_{CF} \mathfrak{K} \rangle \mathfrak{A} \langle \mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} \rangle \langle \text{lim-Obj } c(\text{UObj}) \rangle \langle \text{lim-Obj } c(\text{UArr}) \rangle$
by (rule *assms(4)*)

interpret u' : *is-cat-limit*

$\alpha \langle c' \downarrow_{CF} \mathfrak{K} \rangle \mathfrak{A} \langle \mathfrak{T} \circ_{CF} c' \circ \sqcap_{CF} \mathfrak{K} \rangle \langle \text{lim-Obj } c'(\text{UObj}) \rangle \langle \text{lim-Obj } c'(\text{UArr}) \rangle$
by (rule *assms(5)*)

from *assms(3)* **have** $c : c \in_{\circ} \mathfrak{C}(\text{Obj})$ **and** $c' : c' \in_{\circ} \mathfrak{C}(\text{Obj})$ **by** *auto*

note *the-cf-rKe-ArrMap-app-impl'* =

the-cf-rKe-ArrMap-app-impl'[OF assms]

note *the-f* = *theI'[OF the-cf-rKe-ArrMap-app-impl'[OF assms]]*

note *the-f-is-arr* = *the-f[THEN conjunct1]*

and *the-f-commutes* = *the-f[THEN conjunct2]*

from *assms(3)* *the-f-is-arr* **show**

the-cf-rKe $\alpha \mathfrak{T} \mathfrak{K} \text{lim-Obj}(\text{ArrMap})(g)$:

$\text{lim-Obj } c(\text{UObj}) \mapsto_{\mathfrak{A}} \text{lim-Obj } c'(\text{UObj})$

by

(
cs-concl **cs-shallow**
cs-simp: *the-cf-rKe-ArrMap-app'* **cs-intro**: *cat-cs-intros*
)

moreover from *assms(3)* *the-f-commutes* **show**

$\text{lim-Obj } c(\text{UArr}) \circ_{NTCF-CF} g \downarrow_{CF} \mathfrak{K} =$

$\text{lim-Obj } c'(\text{UArr}) \cdot_{NTCF}$

$\text{ntcf-const } (c' \downarrow_{CF} \mathfrak{K}) \mathfrak{A} (\text{the-cf-rKe } \alpha \mathfrak{T} \mathfrak{K} \text{lim-Obj}(\text{ArrMap})(g))$

by

(
cs-concl **cs-shallow**
cs-simp: *the-cf-rKe-ArrMap-app'* **cs-intro**: *cat-cs-intros*
)

ultimately show $f = \text{the-cf-rKe } \alpha \mathfrak{T} \mathfrak{K} \text{lim-Obj}(\text{ArrMap})(g)$

if $f : \text{lim-Obj } c(\text{UObj}) \mapsto_{\mathfrak{A}} \text{lim-Obj } c'(\text{UObj})$

and $\text{lim-Obj } c(\text{UArr}) \circ_{NTCF-CF} g \downarrow_{CF} \mathfrak{K} =$

$\text{lim-Obj } c'(\text{UArr}) \cdot_{NTCF} \text{ntcf-const } (c' \downarrow_{CF} \mathfrak{K}) \mathfrak{A} f$

by (*metis that the-cf-rKe-ArrMap-app-impl'*)

qed

lemma *the-cf-rKe-ArrMap-is-arr'[cat-Kan-cs-intros]*:

assumes $\mathfrak{K} : \mathfrak{B} \mapsto_{\mapsto} C_{\alpha} \mathfrak{C}$

and $\mathfrak{T} : \mathfrak{B} \mapsto_{\mapsto} C_{\alpha} \mathfrak{A}$

and $g : c \mapsto_{\mathfrak{C}} c'$

and $\text{lim-Obj } c(\text{UArr}) :$

$\text{lim-Obj } c(\text{UObj}) \langle_{CF.lim} \mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} : c \downarrow_{CF} \mathfrak{K} \mapsto_{\mapsto} C_{\alpha} \mathfrak{A}$

and $\text{lim-Obj } c'(\text{UArr}) :$

$\text{lim-Obj } c'(\text{UObj}) \langle_{CF.lim} \mathfrak{T} \circ_{CF} c' \circ \sqcap_{CF} \mathfrak{K} : c' \downarrow_{CF} \mathfrak{K} \mapsto_{\mapsto} C_{\alpha} \mathfrak{A}$

and $a = \text{lim-Obj } c(\text{UObj})$

and $b = \text{lim-Obj } c'(\text{UObj})$
shows *the-cf-rKe* $\alpha \mathfrak{T} \mathfrak{K} \text{lim-Obj}(\text{ArrMap})(\text{!}g) : a \mapsto_{\mathfrak{A}} b$
unfolding *assms(6,7)* **by** (*rule the-cf-rKe-ArrMap-app*[*OF assms(1-5)*])

lemma *lim-Obj-the-cf-rKe-commute*:

assumes $\mathfrak{K} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{T} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{A}$
and $\text{lim-Obj } a(\text{UArr}) :$
 $\text{lim-Obj } a(\text{UObj}) <_{CF.lim} \mathfrak{T} \circ_{CF} a \circ \sqcap_{CF} \mathfrak{K} : a \downarrow_{CF} \mathfrak{K} \mapsto_{C\alpha} \mathfrak{A}$
and $\text{lim-Obj } b(\text{UArr}) :$
 $\text{lim-Obj } b(\text{UObj}) <_{CF.lim} \mathfrak{T} \circ_{CF} b \circ \sqcap_{CF} \mathfrak{K} : b \downarrow_{CF} \mathfrak{K} \mapsto_{C\alpha} \mathfrak{A}$
and $f : a \mapsto_{\mathfrak{C}} b$
and $[a', b', f']_{\circ} \in_{\circ} b \downarrow_{CF} \mathfrak{K}(\text{Obj})$

shows

$\text{lim-Obj } a(\text{UArr})(\text{NTMap})(\text{!}a', b', f' \circ_{A\mathfrak{C}} f)_{\bullet} =$
 $\text{lim-Obj } b(\text{UArr})(\text{NTMap})(\text{!}a', b', f')_{\bullet} \circ_{A\mathfrak{A}}$
 $\text{the-cf-rKe } \alpha \mathfrak{T} \mathfrak{K} \text{lim-Obj}(\text{ArrMap})(\text{!}f)$

proof-

interpret \mathfrak{K} : *is-functor* $\alpha \mathfrak{B} \mathfrak{C} \mathfrak{K}$ **by** (*rule assms(1)*)

interpret \mathfrak{T} : *is-functor* $\alpha \mathfrak{B} \mathfrak{A} \mathfrak{T}$ **by** (*rule assms(2)*)

note $f = \mathfrak{K}.\text{HomCod.cat-is-arrD}$ [*OF assms(5)*]

interpret *lim-a*: *is-cat-limit*

$\alpha \langle a \downarrow_{CF} \mathfrak{K} \rangle \mathfrak{A} \langle \mathfrak{T} \circ_{CF} a \circ \sqcap_{CF} \mathfrak{K} \rangle \langle \text{lim-Obj } a(\text{UObj}) \rangle \langle \text{lim-Obj } a(\text{UArr}) \rangle$
by (*rule assms(3)*)

interpret *lim-b*: *is-cat-limit*

$\alpha \langle b \downarrow_{CF} \mathfrak{K} \rangle \mathfrak{A} \langle \mathfrak{T} \circ_{CF} b \circ \sqcap_{CF} \mathfrak{K} \rangle \langle \text{lim-Obj } b(\text{UObj}) \rangle \langle \text{lim-Obj } b(\text{UArr}) \rangle$
by (*rule assms(4)*)

note $f\text{-app} = \text{the-cf-rKe-ArrMap-app}$
where $\text{lim-Obj} = \text{lim-Obj}$, *OF assms(1,2,5,3,4)*
 $\]$

from $f\text{-app}(2)$ **have** *lim-a-fK-NTMap-app*:

$(\text{lim-Obj } a(\text{UArr}) \circ_{NTCF-CF} f \downarrow_{CF} \mathfrak{K})(\text{NTMap})(\text{!}A) =$
 $($
 $\text{lim-Obj } b(\text{UArr}) \cdot_{NTCF}$
 $\text{ntcf-const } (b \downarrow_{CF} \mathfrak{K}) \mathfrak{A} (\text{the-cf-rKe } \alpha \mathfrak{T} \mathfrak{K} \text{lim-Obj}(\text{ArrMap})(\text{!}f))$
 $\)(\text{NTMap})(\text{!}A)$

if $\langle A \in_{\circ} b \downarrow_{CF} \mathfrak{K}(\text{Obj}) \rangle$ **for** A

by *simp*

show

$\text{lim-Obj } a(\text{UArr})(\text{NTMap})(\text{!}a', b', f' \circ_{A\mathfrak{C}} f)_{\bullet} =$
 $\text{lim-Obj } b(\text{UArr})(\text{NTMap})(\text{!}a', b', f')_{\bullet} \circ_{A\mathfrak{A}}$
 $\text{the-cf-rKe } \alpha \mathfrak{T} \mathfrak{K} \text{lim-Obj}(\text{ArrMap})(\text{!}f)$

proof-

from *assms(5,6)* **have** $a'\text{-def}$: $a' = 0$

and b' : $b' \in_{\circ} \mathfrak{B}(\text{Obj})$

and f' : $f' : b \mapsto_{\mathfrak{C}} \mathfrak{K}(\text{ObjMap})(\text{!}b')$

by *auto*

show

$\text{lim-Obj } a(\text{UArr})(\text{NTMap})(\text{!}a', b', f' \circ_{A\mathfrak{C}} f)_{\bullet} =$
 $\text{lim-Obj } b(\text{UArr})(\text{NTMap})(\text{!}a', b', f')_{\bullet} \circ_{A\mathfrak{A}}$
 $\text{the-cf-rKe } \alpha \mathfrak{T} \mathfrak{K} \text{lim-Obj}(\text{ArrMap})(\text{!}f)$

using *lim-a-fK-NTMap-app*[*OF assms(6)*] f' *assms(3-6)*

unfolding $a'\text{-def}$

```

by
  (
    cs-prems
    cs-simp: cat-cs-simps cat-comma-cs-simps cat-Kan-cs-simps
    cs-intro: cat-cs-intros cat-comma-cs-intros cat-Kan-cs-intros
  )
qed

```

qed

14.4.4 Natural transformation: natural transformation map

```

mk-VLambda the-ntcf-rKe-components(1)
|vsu the-ntcf-rKe-NTMap-vsuv[cat-Kan-cs-intros]]

```

context

```

fixes  $\alpha$   $\mathfrak{A}$   $\mathfrak{B}$   $\mathfrak{C}$   $\mathfrak{R}$   $\mathfrak{T}$ 
assumes  $\mathfrak{R}: \mathfrak{R} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{C}$ 
and  $\mathfrak{T}: \mathfrak{T} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{A}$ 

```

begin

```

interpretation  $\mathfrak{R}$ : is-functor  $\alpha$   $\mathfrak{B}$   $\mathfrak{C}$   $\mathfrak{R}$  by (rule  $\mathfrak{R}$ )
interpretation  $\mathfrak{T}$ : is-functor  $\alpha$   $\mathfrak{B}$   $\mathfrak{A}$   $\mathfrak{T}$  by (rule  $\mathfrak{T}$ )

```

```

mk-VLambda the-ntcf-rKe-components'(1)[OF  $\mathfrak{R}$   $\mathfrak{T}$ ]
|vdomain the-ntcf-rKe-ObjMap-vdomain[cat-Kan-cs-simps]]
|app the-ntcf-rKe-ObjMap-impl-app[cat-Kan-cs-simps]]

```

end

14.4.5 The Kan extension is a Kan extension

lemma the-cf-rKe-is-functor:

```

assumes  $\mathfrak{R} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{C}$ 
and  $\mathfrak{T} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{A}$ 
and  $\bigwedge c. c \in_{\circ} \mathfrak{C}(\text{Obj}) \implies \text{lim-Obj } c(\text{UArr}) :$ 
 $\text{lim-Obj } c(\text{UObj}) <_{CF.\text{lim}} \mathfrak{T} \circ_{CF} c \circ \prod_{CF} \mathfrak{R} : c \downarrow_{CF} \mathfrak{R} \mapsto \mapsto_{C\alpha} \mathfrak{A}$ 
shows the-cf-rKe  $\alpha$   $\mathfrak{T}$   $\mathfrak{R}$   $\text{lim-Obj} : \mathfrak{C} \mapsto \mapsto_{C\alpha} \mathfrak{A}$ 

```

proof-

```

let ?UObj =  $\langle \lambda a. \text{lim-Obj } a(\text{UObj}) \rangle$ 
let ?UArr =  $\langle \lambda a. \text{lim-Obj } a(\text{UArr}) \rangle$ 
let ?const-comma =  $\langle \lambda a b. \text{cf-const } (a \downarrow_{CF} \mathfrak{R}) \mathfrak{A} (?UObj b) \rangle$ 
let ?the-cf-rKe =  $\langle \text{the-cf-rKe } \alpha \mathfrak{T} \mathfrak{R} \text{lim-Obj} \rangle$ 

```

```

interpret  $\mathfrak{R}$ : is-functor  $\alpha$   $\mathfrak{B}$   $\mathfrak{C}$   $\mathfrak{R}$  by (rule  $\text{assms}(1)$ )
interpret  $\mathfrak{T}$ : is-functor  $\alpha$   $\mathfrak{B}$   $\mathfrak{A}$   $\mathfrak{T}$  by (rule  $\text{assms}(2)$ )

```

note [cat-lim-cs-intros] = is-cat-cone.cat-cone-obj

show ?thesis

proof(intro is-functorI')

```

show vsequence ?the-cf-rKe unfolding the-cf-rKe-def by simp
show vcard ?the-cf-rKe =  $4_{\mathbb{N}}$ 
unfolding the-cf-rKe-def by (simp add: nat-omega-simps)
show vsu (?the-cf-rKe(ObjMap))
by (cs-concl cs-shallow cs-intro: cat-Kan-cs-intros)

```

moreover show $\mathcal{D}_\circ (?the\text{-}cf\text{-}rKe(\text{ObjMap})) = \mathfrak{C}(\text{Obj})$
by (*cs-concl cs-shallow cs-simp: cat-Kan-cs-simps cs-intro: cat-cs-intros*)
moreover show $\mathcal{R}_\circ (?the\text{-}cf\text{-}rKe(\text{ObjMap})) \sqsubseteq_\circ \mathfrak{A}(\text{Obj})$
proof
(
 intro the-cf-rKe-ObjMap-vrange;
 (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)?
)
fix c **assume** $c \in_\circ \mathfrak{C}(\text{Obj})$
with *assms(3)[OF this]* **show** $?UObj\ c \in_\circ \mathfrak{A}(\text{Obj})$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-lim-cs-intros*)
qed
ultimately have [*cat-Kan-cs-intros*]:
 $?the\text{-}cf\text{-}rKe(\text{ObjMap})(c) \in_\circ \mathfrak{A}(\text{Obj})$ **if** $\langle c \in_\circ \mathfrak{C}(\text{Obj}) \rangle$ **for** c
by (*metis that vsubsetE vsu.vsv-value*)

show $?the\text{-}cf\text{-}rKe(\text{ArrMap})(f) :$
 $?the\text{-}cf\text{-}rKe(\text{ObjMap})(a) \mapsto_{\mathfrak{A}} ?the\text{-}cf\text{-}rKe(\text{ObjMap})(b)$
if $f : a \mapsto_{\mathfrak{C}} b$ **for** $a\ b\ f$
using *assms(2)* **that**
by
(
 cs-concl
 cs-simp: *cat-Kan-cs-simps*
 cs-intro: *assms(3) cat-cs-intros cat-Kan-cs-intros*
)
then have [*cat-Kan-cs-intros*]: $?the\text{-}cf\text{-}rKe(\text{ArrMap})(f) : A \mapsto_{\mathfrak{A}} B$
if $A = ?the\text{-}cf\text{-}rKe(\text{ObjMap})(a)$
and $B = ?the\text{-}cf\text{-}rKe(\text{ObjMap})(b)$
and $f : a \mapsto_{\mathfrak{C}} b$
for $A\ B\ a\ b\ f$
by (*simp add: that*)

show
 $?the\text{-}cf\text{-}rKe(\text{ArrMap})(g \circ_{A\mathfrak{C}} f) =$
 $?the\text{-}cf\text{-}rKe(\text{ArrMap})(g) \circ_{A\mathfrak{A}} ?the\text{-}cf\text{-}rKe(\text{ArrMap})(f)$
(**is** $\langle ?the\text{-}cf\text{-}rKe(\text{ArrMap})(g \circ_{A\mathfrak{C}} f) = ?the\text{-}rKe\text{-}g \circ_{A\mathfrak{A}} ?the\text{-}rKe\text{-}f \rangle$)
if $g\text{-is-arr: } g : b \mapsto_{\mathfrak{C}} c$ **and** $f\text{-is-arr: } f : a \mapsto_{\mathfrak{C}} b$ **for** $b\ c\ g\ a\ f$
proof-

let $?ntcf\text{-}const\text{-}c = \langle \lambda f. ntcf\text{-}const\ (c \downarrow_{CF} \mathfrak{K})\ \mathfrak{A}\ f \rangle$

note $g = \mathfrak{K}.HomCod.cat\text{-}is\text{-}arrD[OF\ that(1)]$
and $f = \mathfrak{K}.HomCod.cat\text{-}is\text{-}arrD[OF\ that(2)]$
note $lim\text{-}a = assms(3)[OF\ f(2)]$
and $lim\text{-}b = assms(3)[OF\ g(2)]$
and $lim\text{-}c = assms(3)[OF\ g(3)]$
from that have $gf : g \circ_{A\mathfrak{C}} f : a \mapsto_{\mathfrak{C}} c$
by (*cs-concl cs-shallow cs-intro: cat-cs-intros*)

interpret $lim\text{-}a$: *is-cat-limit*
 $\alpha \langle a \downarrow_{CF} \mathfrak{K} \rangle \mathfrak{A} \langle \mathfrak{I} \circ_{CF} a \circ \square_{CF} \mathfrak{K} \rangle \langle ?UObj\ a \rangle \langle ?UArr\ a \rangle$
by (*rule lim-a*)
interpret $lim\text{-}c$: *is-cat-limit*
 $\alpha \langle c \downarrow_{CF} \mathfrak{K} \rangle \mathfrak{A} \langle \mathfrak{I} \circ_{CF} c \circ \square_{CF} \mathfrak{K} \rangle \langle ?UObj\ c \rangle \langle ?UArr\ c \rangle$
by (*rule lim-c*)

show *?thesis*

proof

(
rule sym,
rule the-cf-rKe-ArrMap-app(3)[OF assms(1,2) gf lim-a lim-c]
)

from *assms(1,2)* **that** *lim-a lim-b lim-c* **show**

?the-rKe-g $\circ_{A\mathfrak{A}}$ *?the-rKe-f* : *?UObj a* $\mapsto_{\mathfrak{A}}$ *?UObj c*

by

(
cs-concl
cs-simp: *cat-cs-simps* **cs-intro:** *cat-cs-intros cat-Kan-cs-intros*
)

show

?UArr a $\circ_{NTCF-CF}$ (*g* $\circ_{A\mathfrak{C}}$ *f*) $A\downarrow_{CF}$ \mathfrak{K} =
?UArr c \cdot_{NTCF} *?ntcf-const-c* (*?the-rKe-g* $\circ_{A\mathfrak{A}}$ *?the-rKe-f*)

(

is

<
?UArr a $\circ_{NTCF-CF}$ (*g* $\circ_{A\mathfrak{C}}$ *f*) $A\downarrow_{CF}$ \mathfrak{K} =
?UArr c \cdot_{NTCF} *?ntcf-const-c* *?the-rKe-gf*
 >

)

proof(*rule ntcf-eqI*)

from *that* **show**

?UArr a $\circ_{NTCF-CF}$ (*g* $\circ_{A\mathfrak{C}}$ *f*) $A\downarrow_{CF}$ \mathfrak{K} :
cf-const (*a* \downarrow_{CF} \mathfrak{K}) \mathfrak{A} (*?UObj a*) \circ_{CF} (*g* $\circ_{A\mathfrak{C}}$ *f*) $A\downarrow_{CF}$ \mathfrak{K} \mapsto_{CF}
 $\mathfrak{T} \circ_{CF}$ *a* $\circ_{\square_{CF}}$ $\mathfrak{K} \circ_{CF}$ ((*g* $\circ_{A\mathfrak{C}}$ *f*) $A\downarrow_{CF}$ \mathfrak{K}) :
c \downarrow_{CF} \mathfrak{K} $\mapsto_{C\alpha}$ \mathfrak{A}

by (*cs-concl* **cs-shallow** **cs-intro:** *cat-cs-intros cat-comma-cs-intros*)

have [*cat-comma-cs-simps*]:

?const-comma a a \circ_{CF} (*g* $\circ_{A\mathfrak{C}}$ *f*) $A\downarrow_{CF}$ \mathfrak{K} = *?const-comma c a*

proof(*rule cf-eqI*)

from *g-is-arr f-is-arr* **show**

?const-comma a a \circ_{CF} (*g* $\circ_{A\mathfrak{C}}$ *f*) $A\downarrow_{CF}$ \mathfrak{K} : *c* \downarrow_{CF} \mathfrak{K} $\mapsto_{C\alpha}$ \mathfrak{A}

by

(
cs-concl
cs-simp: *cat-comma-cs-simps cat-cs-simps*
cs-intro:
cat-cs-intros cat-lim-cs-intros cat-comma-cs-intros
)

from *g-is-arr f-is-arr* **show** *?const-comma c a* : *c* \downarrow_{CF} \mathfrak{K} $\mapsto_{C\alpha}$ \mathfrak{A}

by

(
cs-concl
cs-simp: *cat-comma-cs-simps cat-cs-simps*
cs-intro:
cat-cs-intros cat-lim-cs-intros cat-comma-cs-intros
)

from *g-is-arr f-is-arr* **have** *ObjMap-dom-lhs:*

\mathcal{D}_\circ ((*?const-comma a a* \circ_{CF} (*g* $\circ_{A\mathfrak{C}}$ *f*) $A\downarrow_{CF}$ \mathfrak{K})(*ObjMap*)) =
c \downarrow_{CF} \mathfrak{K} (*Obj*)

by

(
cs-concl
cs-simp: *cat-comma-cs-simps cat-cs-simps*
)

cs-intro:
cat-comma-cs-intros cat-lim-cs-intros cat-cs-intros
)

from *g-is-arr f-is-arr* **have** *ObjMap-dom-rhs:*
 $\mathcal{D}_\circ (?const-comma\ c\ a(\text{ObjMap})) = c \downarrow_{CF} \mathfrak{K}(\text{Obj})$
by (*cs-concl cs-shallow cs-simp: cat-comma-cs-simps cat-cs-simps*)

show
 $(?const-comma\ a\ a \circ_{CF} (g \circ_{A\mathfrak{C}} f) \downarrow_{CF} \mathfrak{K})(\text{ObjMap}) =$
 $?const-comma\ c\ a(\text{ObjMap})$

proof(*rule vsu-eqI, unfold ObjMap-dom-lhs ObjMap-dom-rhs*)
from *f-is-arr g-is-arr* **show**
 $vsu\ ((?const-comma\ a\ a \circ_{CF} (g \circ_{A\mathfrak{C}} f) \downarrow_{CF} \mathfrak{K})(\text{ObjMap}))$
by
(

cs-concl
cs-simp: *cat-comma-cs-simps cat-cs-simps*
cs-intro:
cat-cs-intros cat-lim-cs-intros cat-comma-cs-intros
)

fix *A* **assume** *prems: A ∈_o c ↓_{CF} ℔(Obj)*
with *g-is-arr* **obtain** *b' f'*
where *A-def: A = [0, b', f']_o*
and *b': b' ∈_o ℔(Obj)*
and *f': f' : c ↦_C ℔(ObjMap)(b')*
by *auto*

from *prems b' f' g-is-arr f-is-arr* **show**
 $(?const-comma\ a\ a \circ_{CF} (g \circ_{A\mathfrak{C}} f) \downarrow_{CF} \mathfrak{K})(\text{ObjMap})(A) =$
 $?const-comma\ c\ a(\text{ObjMap})(A)$
unfolding *A-def*
by
(

cs-concl
cs-simp: *cat-comma-cs-simps cat-cs-simps*
cs-intro:
cat-cs-intros cat-lim-cs-intros cat-comma-cs-intros
)

qed (*cs-concl cs-shallow cs-intro: cat-cs-intros*)

from *g-is-arr f-is-arr* **have** *ArrMap-dom-lhs:*
 $\mathcal{D}_\circ ((?const-comma\ a\ a \circ_{CF} (g \circ_{A\mathfrak{C}} f) \downarrow_{CF} \mathfrak{K})(\text{ArrMap})) =$
 $c \downarrow_{CF} \mathfrak{K}(\text{Arr})$
by
(

cs-concl
cs-simp: *cat-comma-cs-simps cat-cs-simps*
cs-intro:
cat-comma-cs-intros cat-lim-cs-intros cat-cs-intros
)

from *g-is-arr f-is-arr* **have** *ArrMap-dom-rhs:*
 $\mathcal{D}_\circ (?const-comma\ c\ a(\text{ArrMap})) = c \downarrow_{CF} \mathfrak{K}(\text{Arr})$
by (*cs-concl cs-shallow cs-simp: cat-comma-cs-simps cat-cs-simps*)

show
 $(?const-comma\ a\ a \circ_{CF} (g \circ_{A\mathfrak{C}} f) \downarrow_{CF} \mathfrak{K})(\text{ArrMap}) =$
 $?const-comma\ c\ a(\text{ArrMap})$

proof(*rule vsu-eqI, unfold ArrMap-dom-lhs ArrMap-dom-rhs*)
from *f-is-arr g-is-arr* **show**

$vsu ((?const-comma a a \circ_{CF} (g \circ_{A\mathfrak{C}} f) A \downarrow_{CF} \mathfrak{K})(\downarrow_{ArrMap}))$
by
(

 cs-concl
 cs-simp: *cat-comma-cs-simps cat-cs-simps*
 cs-intro:
 cat-cs-intros cat-lim-cs-intros cat-comma-cs-intros
)

fix F **assume** $F \in_{\circ} c \downarrow_{CF} \mathfrak{K}(\downarrow_{Arr})$
then obtain $A B$ **where** $F: F: A \mapsto_c \downarrow_{CF} \mathfrak{K} B$
 unfolding *cat-comma-cs-simps* **by** (*auto intro: is-arrI*)
with *g-is-arr* **obtain** $b' f' b'' f'' h'$
 where $F\text{-def: } F = [[0, b', f']_{\circ}, [0, b'', f'']_{\circ}, [0, h']_{\circ}]_{\circ}$
 and $A\text{-def: } A = [0, b', f']_{\circ}$
 and $B\text{-def: } B = [0, b'', f'']_{\circ}$
 and $h': h': b' \mapsto_{\mathfrak{B}} b''$
 and $f': f': c \mapsto_{\mathfrak{C}} \mathfrak{K}(\downarrow_{ObjMap})(\downarrow_{b'})$
 and $f'': f'': c \mapsto_{\mathfrak{C}} \mathfrak{K}(\downarrow_{ObjMap})(\downarrow_{b''})$
 and $f''\text{-def: } \mathfrak{K}(\downarrow_{ArrMap})(\downarrow_{h'}) \circ_{A\mathfrak{C}} f' = f''$
by *auto*
from F *f-is-arr g-is-arr g' h' f' f''* **show**
 $(?const-comma a a \circ_{CF} (g \circ_{A\mathfrak{C}} f) A \downarrow_{CF} \mathfrak{K})(\downarrow_{ArrMap})(\downarrow_F) =$
 $?const-comma c a(\downarrow_{ArrMap})(\downarrow_F)$
unfolding $F\text{-def } A\text{-def } B\text{-def}$
by
(

 cs-concl
 cs-intro:
 cat-lim-cs-intros cat-cs-intros cat-comma-cs-intros
 cs-simp:
 cat-cs-simps cat-comma-cs-simps f''-def[symmetric]
)

qed (*cs-concl cs-shallow cs-intro: cat-cs-intros*)
qed *simp-all*

from that show
 $?UArr c \cdot_{NTCF} ?ntcf\text{-const-}c \text{ ?the-}rKe\text{-}gf :$
 $cf\text{-const} (a \downarrow_{CF} \mathfrak{K}) \mathfrak{A} (?UObj a) \circ_{CF} (g \circ_{A\mathfrak{C}} f) A \downarrow_{CF} \mathfrak{K} \mapsto_{CF}$
 $\mathfrak{T} \circ_{CF} a \circ \sqcap_{CF} \mathfrak{K} \circ_{CF} ((g \circ_{A\mathfrak{C}} f) A \downarrow_{CF} \mathfrak{K}) :$
 $c \downarrow_{CF} \mathfrak{K} \mapsto \mapsto_{C\alpha} \mathfrak{A}$

by
(

 cs-concl
 cs-simp: *cat-Kan-cs-simps cat-comma-cs-simps cat-cs-simps*
 cs-intro:
 cat-lim-cs-intros
 cat-comma-cs-intros
 cat-Kan-cs-intros
 cat-cs-intros
)

from that have dom-lhs:
 $\mathcal{D}_{\circ} ((?UArr a \circ_{NTCF-CF} (g \circ_{A\mathfrak{C}} f) A \downarrow_{CF} \mathfrak{K})(\downarrow_{NTMap})) = c \downarrow_{CF} \mathfrak{K}(\downarrow_{Obj})$
by
(

 cs-concl cs-shallow
 cs-intro: *cat-cs-intros cat-comma-cs-intros*
 cs-simp: *cat-cs-simps cat-comma-cs-simps*
)

from that have dom-rhs:
 $\mathcal{D}_o ((?UArr\ c \cdot_{NTCF} ?ntcf\text{-const-}c\ ?the\text{-}rKe\text{-}gf)(\downarrow NTMap)) =$
 $c \downarrow_{CF} \mathfrak{K}(\downarrow Obj)$
by
(

cs-concl
cs-intro: *cat-cs-intros cat-Kan-cs-intros cat-comma-cs-intros*
cs-simp: *cat-Kan-cs-simps cat-cs-simps cat-comma-cs-simps*
)

show
 $(?UArr\ a \circ_{NTCF-CF} (g \circ_{A\mathfrak{C}} f) \downarrow_{CF} \mathfrak{K})(\downarrow NTMap) =$
 $(?UArr\ c \cdot_{NTCF} ?ntcf\text{-const-}c\ ?the\text{-}rKe\text{-}gf)(\downarrow NTMap)$
proof(*rule vsu-eqI, unfold dom-lhs dom-rhs*)
fix A **assume** *prems:* $A \in_o c \downarrow_{CF} \mathfrak{K}(\downarrow Obj)$
with *g-is-arr* **obtain** $b' f'$
where *A-def:* $A = [0, b', f']_o$
and $b': b' \in_o \mathfrak{B}(\downarrow Obj)$
and $f': f' : c \mapsto_{\mathfrak{C}} \mathfrak{K}(\downarrow ObjMap)(\downarrow b')$
by *auto*
note $\mathfrak{T}.HomCod.cat\text{-}Comp\text{-}assoc[cat\text{-}cs\text{-}simps\ del]$
and $\mathfrak{K}.HomCod.cat\text{-}Comp\text{-}assoc[cat\text{-}cs\text{-}simps\ del]$
and *category.cat-Comp-assoc[cat-cs-simps del]*
note [*symmetric, cat-cs-simps*] =
lim-Obj-the-cf-rKe-commute[where lim-Obj=lim-Obj]
 $\mathfrak{K}.HomCod.cat\text{-}Comp\text{-}assoc$
 $\mathfrak{T}.HomCod.cat\text{-}Comp\text{-}assoc$
from *assms(1,2)* **that** *prems* $lim\text{-}a\ lim\text{-}b\ lim\text{-}c\ b'\ f'$ **show**
 $(?UArr\ a \circ_{NTCF-CF} (g \circ_{A\mathfrak{C}} f) \downarrow_{CF} \mathfrak{K})(\downarrow NTMap)(\downarrow A) =$
 $(?UArr\ c \cdot_{NTCF} ?ntcf\text{-const-}c\ ?the\text{-}rKe\text{-}gf)(\downarrow NTMap)(\downarrow A)$
unfolding *A-def*
by
(

cs-concl
cs-simp:
cat-cs-simps cat-Kan-cs-simps cat-comma-cs-simps
cs-intro:
cat-cs-intros cat-Kan-cs-intros cat-comma-cs-intros
)+
qed (*cs-concl cs-simp: cs-intro: cat-cs-intros*)+
qed *simp-all*
qed
qed

show $?the\text{-}cf\text{-}rKe(\downarrow ArrMap)(\downarrow \mathfrak{C}(\downarrow CId)(\downarrow c)) = \mathfrak{A}(\downarrow CId)(\downarrow ?the\text{-}cf\text{-}rKe(\downarrow ObjMap)(\downarrow c))$
if $c \in_o \mathfrak{C}(\downarrow Obj)$ **for** c
proof-

let $?ntcf\text{-const-}c = \langle ntcf\text{-const}\ (c \downarrow_{CF} \mathfrak{K})\ \mathfrak{A}\ (\mathfrak{A}(\downarrow CId)(\downarrow ?UObj\ c)) \rangle$

note $lim\text{-}c = assms(3)[OF\ that]$

from that have $CId\text{-}c: \mathfrak{C}(\downarrow CId)(\downarrow c) : c \mapsto_{\mathfrak{C}} c$
by (*cs-concl cs-shallow cs-intro: cat-cs-intros*)

interpret $lim\text{-}c: is\text{-}cat\text{-}limit$
 $\alpha \langle c \downarrow_{CF} \mathfrak{K} \rangle \mathfrak{A} \langle \mathfrak{T} \circ_{CF} c \circ \square_{CF} \mathfrak{K} \rangle \langle ?UObj\ c \rangle \langle ?UArr\ c \rangle$
by (*rule lim-c*)

show *?thesis*
proof
(
rule *sym*,
rule *the-cf-rKe-ArrMap-app(3)*[
where *lim-Obj=lim-Obj*, *OF assms(1,2) CId-c lim-c lim-c*
]
)
from *that lim-c show*
 $\mathfrak{A}(\mathcal{C}Id)(\downarrow ?the-cf-rKe(\downarrow ObjMap)(c)) : ?UObj\ c \mapsto_{\mathfrak{A}} ?UObj\ c$
by
(
cs-concl cs-shallow
cs-simp: *cat-Kan-cs-simps*
cs-intro: *cat-cs-intros cat-lim-cs-intros*
)
have $?UArr\ c \circ_{NTCF-CF} (\mathcal{C}(\mathcal{C}Id)(c)) \downarrow_{CF} \mathfrak{K} = ?UArr\ c \cdot_{NTCF} ?ntcf-const-c$
proof(*rule ntcf-eqI*)
from *lim-c that show*
 $?UArr\ c \circ_{NTCF-CF} (\mathcal{C}(\mathcal{C}Id)(c)) \downarrow_{CF} \mathfrak{K} :$
 $cf-const\ (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A}\ (?UObj\ c) \circ_{CF} (\mathcal{C}(\mathcal{C}Id)(c)) \downarrow_{CF} \mathfrak{K} \mapsto_{CF}$
 $\mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} \circ_{CF} (\mathcal{C}(\mathcal{C}Id)(c)) \downarrow_{CF} \mathfrak{K} :$
 $c \downarrow_{CF} \mathfrak{K} \mapsto_{C\alpha} \mathfrak{A}$
by (*cs-concl cs-shallow cs-intro: cat-cs-intros cat-comma-cs-intros*)
from *lim-c that show*
 $?UArr\ c \cdot_{NTCF} ?ntcf-const-c :$
 $cf-const\ (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A}\ (?UObj\ c) \circ_{CF} (\mathcal{C}(\mathcal{C}Id)(c)) \downarrow_{CF} \mathfrak{K} \mapsto_{CF}$
 $\mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} \circ_{CF} (\mathcal{C}(\mathcal{C}Id)(c)) \downarrow_{CF} \mathfrak{K} :$
 $c \downarrow_{CF} \mathfrak{K} \mapsto_{C\alpha} \mathfrak{A}$
by
(
cs-concl
cs-intro: *cat-cs-intros*
cs-simp: $\mathfrak{K}.cf-arr-cf-comma-CId\ cat-cs-simps$
cs-intro: *cat-lim-cs-intros*
)
from *that have dom-lhs:*
 $\mathcal{D}_o\ ((?UArr\ c \circ_{NTCF-CF} (\mathcal{C}(\mathcal{C}Id)(c)) \downarrow_{CF} \mathfrak{K})(\downarrow NTMap)) = c \downarrow_{CF} \mathfrak{K}(\downarrow Obj)$
by
(
cs-concl cs-shallow
cs-simp: *cat-cs-simps*
cs-intro: *cat-cs-intros cat-comma-cs-intros*
)
from *that have dom-rhs:*
 $\mathcal{D}_o\ ((?UArr\ c \cdot_{NTCF} ?ntcf-const-c)(\downarrow NTMap)) = c \downarrow_{CF} \mathfrak{K}(\downarrow Obj)$
by
(
cs-concl
cs-intro: *cat-lim-cs-intros cat-cs-intros*
cs-simp: *cat-cs-simps*
)
show
 $(?UArr\ c \circ_{NTCF-CF} (\mathcal{C}(\mathcal{C}Id)(c)) \downarrow_{CF} \mathfrak{K})(\downarrow NTMap) =$
 $(?UArr\ c \cdot_{NTCF} ?ntcf-const-c)(\downarrow NTMap)$
proof(*rule vsu-eqI, unfold dom-lhs dom-rhs*)
fix *A assume prems: A* $\epsilon_o\ c \downarrow_{CF} \mathfrak{K}(\downarrow Obj)$

with that obtain $b f$
where $A\text{-def}: A = [0, b, f]_0$
and $b: b \in_0 \mathfrak{B}(\text{Obj})$
and $f: f : c \mapsto_{\mathfrak{C}} \mathfrak{K}(\text{ObjMap})(b)$
by auto
from that prems f **have**
 $?UArr\ c(\text{NTMap})(0, b, f)_\bullet : ?UObj\ c \mapsto_{\mathfrak{A}} \mathfrak{T}(\text{ObjMap})(b)$
unfolding $A\text{-def}$
by
(

 $cs\text{-concl}$
 cs-simp: $cat\text{-cs-simps}\ cat\text{-comma-cs-simps}$
 cs-intro: $cat\text{-comma-cs-intros}\ cat\text{-cs-intros}$

)

from that prems f **show**
 $(?UArr\ c \circ_{NTCF-CF} (\mathfrak{C}(\text{CId})(c))\ A \downarrow_{CF} \mathfrak{K})(\text{NTMap})(A) =$
 $(?UArr\ c \cdot_{NTCF} ?ntcf\text{-const-c})(\text{NTMap})(A)$
unfolding $A\text{-def}$
by
(

 $cs\text{-concl}$
 cs-simp: $cat\text{-cs-simps}\ cat\text{-comma-cs-simps}$
 cs-intro:
 $cat\text{-lim-cs-intros}\ cat\text{-comma-cs-intros}\ cat\text{-cs-intros}$

)

qed ($cs\text{-concl}\ \mathbf{cs-intro}: cat\text{-cs-intros}$)
qed $simp\text{-all}$

with that show
 $?UArr\ c \circ_{NTCF-CF} (\mathfrak{C}(\text{CId})(c))\ A \downarrow_{CF} \mathfrak{K} =$
 $?UArr\ c \cdot_{NTCF} ntcf\text{-const}\ (c \downarrow_{CF} \mathfrak{K})\ \mathfrak{A}\ (\mathfrak{A}(\text{CId})(?the\text{-cf-rKe}(\text{ObjMap})(c)))$
by
(

 $cs\text{-concl}\ \mathbf{cs-shallow}$
 cs-simp: $cat\text{-Kan-cs-simps}\ \mathbf{cs-intro}: cat\text{-cs-intros}$

)

qed

qed

qed

qed
(

 $cs\text{-concl}$
 cs-simp: $cat\text{-Kan-cs-simps}\ \mathbf{cs-intro}: cat\text{-cs-intros}\ cat\text{-Kan-cs-intros}$

)+

qed

lemma $the\text{-cf-lKe-is-functor}$:
assumes $\mathfrak{K} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{T} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{A}$
and $\bigwedge c. c \in_0 \mathfrak{C}(\text{Obj}) \implies \text{lim-Obj}\ c(\text{UArr}) :$
 $\mathfrak{T} \circ_{CF} \mathfrak{K}\ CF \sqcap_0 c >_{CF.colim} \text{lim-Obj}\ c(\text{UObj}) : \mathfrak{K}\ CF \downarrow c \mapsto_{C\alpha} \mathfrak{A}$
shows $the\text{-cf-lKe}\ \alpha\ \mathfrak{T}\ \mathfrak{K}\ \text{lim-Obj} : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{A}$

proof-
interpret \mathfrak{K} : $is\text{-functor}\ \alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{K}$ **by** ($rule\ assms(1)$)
interpret \mathfrak{T} : $is\text{-functor}\ \alpha\ \mathfrak{B}\ \mathfrak{A}\ \mathfrak{T}$ **by** ($rule\ assms(2)$)

```

{
  fix c assume prems: c ∈o C(Obj)
  from assms(3)[OF this] have lim-Obj-UArr: lim-Obj c(UArr) :
    ℑ ∘CF ℝCF ⊓O c >CF.colim lim-Obj c(UObj) : ℝCF ↓ c ↦CF Cα ℒ.
  then interpret lim-Obj-c: is-cat-colimit
    α ⟨ℝCF ↓ c⟩ ℒ ⟨ℑ ∘CF ℝCF ⊓O c⟩ ⟨lim-Obj c(UObj)⟩ ⟨lim-Obj c(UArr)⟩
    by simp
  note op-ua-UArr-is-cat-limit'[
    where lim-Obj=lim-Obj, OF assms(1,2) prems lim-Obj-UArr
  ]
}
note the-cf-rKe-is-functor = the-cf-rKe-is-functor
[
  OF ℝ.is-functor-op ℑ.is-functor-op,
  unfolded cat-op-simps,
  where lim-Obj=⟨op-ua lim-Obj ℝ⟩,
  unfolded cat-op-simps,
  OF this,
  simplified,
  folded the-cf-lKe-def
]
show the-cf-lKe α ℑ ℝ lim-Obj : C ↦CF Cα ℒ
by
(
  rule is-functor.is-functor-op
  [
    OF the-cf-rKe-is-functor,
    folded the-cf-lKe-def,
    unfolded cat-op-simps
  ]
)
)
qed

```

lemma *the-ntcf-rKe-is-ntcf*:

```

assumes ℝ : B ↦CF Cα C
and ℑ : B ↦CF Cα ℒ
and ∧c. c ∈o C(Obj) ⟹ lim-Obj c(UArr) :
  lim-Obj c(UObj) <CF.lim ℑ ∘CF c ⊓O CF ℝ : c ↓CF ℝ ↦CF Cα ℒ
shows the-ntcf-rKe α ℑ ℝ lim-Obj :
  the-cf-rKe α ℑ ℝ lim-Obj ∘CF ℝ ↦CF ℑ : B ↦CF Cα ℒ

```

proof-

```

let ?UObj = ⟨λa. lim-Obj a(UObj)⟩
let ?UArr = ⟨λa. lim-Obj a(UArr)⟩
let ?const-comma = ⟨λa b. cf-const (a ↓CF ℝ) ℒ (?UObj b)⟩
let ?the-cf-rKe = ⟨the-cf-rKe α ℑ ℝ lim-Obj⟩
let ?the-ntcf-rKe = ⟨the-ntcf-rKe α ℑ ℝ lim-Obj⟩

```

```

interpret ℝ: is-functor α B C ℝ by (rule assms(1))
interpret ℑ: is-functor α B ℒ ℑ by (rule assms(2))
interpret cf-rKe: is-functor α C ℒ ⟨?the-cf-rKe⟩
by (rule the-cf-rKe-is-functor[OF assms, simplified])

```

show *?thesis*

proof(*rule is-ntcfI'*)

show *vfsequence* *?the-ntcf-rKe* **unfolding** *the-ntcf-rKe-def* **by** *simp*

show *vcard* *?the-ntcf-rKe* = 5_N

unfolding *the-ntcf-rKe-def* **by** (*simp add: nat-omega-simps*)

show $?the\text{-}ntcf\text{-}rKe(\downarrow NTMap)(\downarrow b) :$
 $(?the\text{-}cf\text{-}rKe \circ_{CF} \mathfrak{K})(\downarrow ObjMap)(\downarrow b) \mapsto_{\mathfrak{A}} \mathfrak{T}(\downarrow ObjMap)(\downarrow b)$
if $b \in_{\circ} \mathfrak{B}(\downarrow Obj)$ **for** b
proof-
let $?Rb = \langle \mathfrak{K}(\downarrow ObjMap)(\downarrow b) \rangle$
from *that* **have** $Rb: \mathfrak{K}(\downarrow ObjMap)(\downarrow b) \in_{\circ} \mathfrak{C}(\downarrow Obj)$
by (*cs-concl cs-shallow cs-intro: cat-cs-intros*)
note $lim\text{-}Rb = assms(\mathfrak{Z})[OF\ Rb]$
interpret $lim\text{-}Rb: is\text{-}cat\text{-}limit$
 $\alpha \langle ?Rb \downarrow_{CF} \mathfrak{K} \rangle \mathfrak{A} \langle \mathfrak{T} \circ_{CF} ?Rb \circ \square_{CF} \mathfrak{K} \rangle \langle ?UObj\ ?Rb \rangle \langle ?UArr\ ?Rb \rangle$
by (*rule lim-Rb*)
from *that* $lim\text{-}Rb$ **show** *?thesis*
by
 $($
cs-concl
cs-simp: *cat-cs-simps cat-comma-cs-simps cat-Kan-cs-simps*
cs-intro: *cat-cs-intros cat-comma-cs-intros cat-Kan-cs-intros*
 $)+$

qed
show

$?the\text{-}ntcf\text{-}rKe(\downarrow NTMap)(\downarrow b) \circ_{A\mathfrak{A}} (?the\text{-}cf\text{-}rKe \circ_{CF} \mathfrak{K})(\downarrow ArrMap)(\downarrow f) =$
 $\mathfrak{T}(\downarrow ArrMap)(\downarrow f) \circ_{A\mathfrak{A}} ?the\text{-}ntcf\text{-}rKe(\downarrow NTMap)(\downarrow a)$
if $f : a \mapsto_{\mathfrak{B}} b$ **for** $a\ b\ f$

proof-
let $?Ra = \langle \mathfrak{K}(\downarrow ObjMap)(\downarrow a) \rangle$ **and** $?Rb = \langle \mathfrak{K}(\downarrow ObjMap)(\downarrow b) \rangle$ **and** $?Rf = \langle \mathfrak{K}(\downarrow ArrMap)(\downarrow f) \rangle$
from *that* **have** $Ra: ?Ra \in_{\circ} \mathfrak{C}(\downarrow Obj)$
and $Rb: ?Rb \in_{\circ} \mathfrak{C}(\downarrow Obj)$
and $Rf: ?Rf : ?Ra \mapsto_{\mathfrak{C}} ?Rb$
by (*cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)
note $lim\text{-}Ra = assms(\mathfrak{Z})[OF\ Ra]$
and $lim\text{-}Rb = assms(\mathfrak{Z})[OF\ Rb]$
from *that* **have** $z\text{-}b\text{-}Rb: [0, b, \mathfrak{C}(\downarrow CId)(\downarrow ?Rb)]_{\circ} \in_{\circ} ?Rb \downarrow_{CF} \mathfrak{K}(\downarrow Obj)$
by (*cs-concl cs-intro: cat-cs-intros cat-comma-cs-intros*)
from
 $lim\text{-}Obj\text{-}the\text{-}cf\text{-}rKe\text{-}commute[$
 $OF\ assms(1,2)\ lim\text{-}Ra\ lim\text{-}Rb\ Rf\ z\text{-}b\text{-}Rb, symmetric$
 $]$
that
have [*cat-Kan-cs-simps*]:
 $?UArr\ ?Rb(\downarrow NTMap)(\downarrow 0, b, \mathfrak{C}(\downarrow CId)(\downarrow ?Rb))_{\bullet} \circ_{A\mathfrak{A}} ?the\text{-}cf\text{-}rKe(\downarrow ArrMap)(\downarrow ?Rf) =$
 $?UArr\ ?Ra(\downarrow NTMap)(\downarrow 0, b, ?Rf)_{\bullet}$
by (*cs-prems cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)
interpret $lim\text{-}Ra: is\text{-}cat\text{-}limit$
 $\alpha \langle ?Ra \downarrow_{CF} \mathfrak{K} \rangle \mathfrak{A} \langle \mathfrak{T} \circ_{CF} ?Ra \circ \square_{CF} \mathfrak{K} \rangle \langle ?UObj\ ?Ra \rangle \langle ?UArr\ ?Ra \rangle$
by (*rule lim-Ra*)
interpret $lim\text{-}Rb: is\text{-}cat\text{-}limit$
 $\alpha \langle ?Rb \downarrow_{CF} \mathfrak{K} \rangle \mathfrak{A} \langle \mathfrak{T} \circ_{CF} ?Rb \circ \square_{CF} \mathfrak{K} \rangle \langle ?UObj\ ?Rb \rangle \langle ?UArr\ ?Rb \rangle$
by (*rule lim-Rb*)
note $lim\text{-}Ra.\text{cat}\text{-}cone\text{-}Comp\text{-}commute[cat\text{-}cs\text{-}simps\ del]$
note $lim\text{-}Rb.\text{cat}\text{-}cone\text{-}Comp\text{-}commute[cat\text{-}cs\text{-}simps\ del]$
from *that* **have**
 $[[0, a, \mathfrak{C}(\downarrow CId)(\downarrow ?Ra)]_{\circ}, [0, b, ?Rf]_{\circ}, [0, f]_{\circ}]_{\circ} :$
 $[0, a, \mathfrak{C}(\downarrow CId)(\downarrow ?Ra)]_{\circ} \mapsto_{(?Ra) \downarrow_{CF} \mathfrak{K}} [0, b, ?Rf]_{\circ}$
by
 $($
cs-concl
cs-simp: *cat-cs-simps cat-comma-cs-simps*
cs-intro: *cat-cs-intros cat-comma-cs-intros*
 $)$

)
from $\text{lim-}\mathfrak{K}a.\text{ntcf-Comp-commute}$ [OF this, symmetric] that
have [cat-Kan-cs-simps]:
 $\mathfrak{T}(\text{ArrMap})(f) \circ_{A\mathfrak{A}} ?UArr (? \mathfrak{K}a)(\text{NTMap})(0, a, \mathfrak{C}(\text{CId})(? \mathfrak{K}a)) \bullet =$
 $?UArr ? \mathfrak{K}a(\text{NTMap})(0, b, ? \mathfrak{K}f) \bullet$
by
 (
 cs-prems
 cs-simp: *cat-cs-simps cat-comma-cs-simps*
 cs-intro: *cat-cs-intros cat-comma-cs-intros cat-1-is-arrI*
)
from that **show** *?thesis*
by
 (
 cs-concl
 cs-simp: *cat-cs-simps cat-Kan-cs-simps* **cs-intro:** *cat-cs-intros*
)
qed
qed
 (
 cs-concl
 cs-simp: *cat-Kan-cs-simps* **cs-intro:** *cat-cs-intros cat-Kan-cs-intros*
)+

qed

lemma *the-ntcf-lKe-is-ntcf:*

assumes $\mathfrak{K} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{C}$

and $\mathfrak{T} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{A}$

and $\bigwedge c. c \in_0 \mathfrak{C}(\text{Obj}) \implies \text{lim-Obj } c(\text{UArr}) :$

$\mathfrak{T} \circ_{CF} \mathfrak{K} \text{ }_{CF} \sqcap_O c >_{CF.colim} \text{lim-Obj } c(\text{UObj}) : \mathfrak{K} \text{ }_{CF} \downarrow c \mapsto \mapsto_{C\alpha} \mathfrak{A}$

shows *the-ntcf-lKe* α $\mathfrak{T} \mathfrak{K} \text{lim-Obj} :$

$\mathfrak{T} \mapsto_{CF} \text{the-cf-lKe} \alpha \mathfrak{T} \mathfrak{K} \text{lim-Obj} \circ_{CF} \mathfrak{K} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{A}$

proof-

interpret $\mathfrak{K} : \text{is-functor} \alpha \mathfrak{B} \mathfrak{C} \mathfrak{K}$ **by** (rule *assms(1)*)

interpret $\mathfrak{T} : \text{is-functor} \alpha \mathfrak{B} \mathfrak{A} \mathfrak{T}$ **by** (rule *assms(2)*)

{

fix c **assume** *prems:* $c \in_0 \mathfrak{C}(\text{Obj})$

from *assms(3)* [OF this] **have** *lim-Obj-UArr:* $\text{lim-Obj } c(\text{UArr}) :$

$\mathfrak{T} \circ_{CF} \mathfrak{K} \text{ }_{CF} \sqcap_O c >_{CF.colim} \text{lim-Obj } c(\text{UObj}) : \mathfrak{K} \text{ }_{CF} \downarrow c \mapsto \mapsto_{C\alpha} \mathfrak{A}.$

then interpret *lim-Obj-c:* *is-cat-colimit*

$\alpha \langle \mathfrak{K} \text{ }_{CF} \downarrow c \rangle \mathfrak{A} \langle \mathfrak{T} \circ_{CF} \mathfrak{K} \text{ }_{CF} \sqcap_O c \rangle \langle \text{lim-Obj } c(\text{UObj}) \rangle \langle \text{lim-Obj } c(\text{UArr}) \rangle$

by *simp*

note *op-ua-UArr-is-cat-limit'* [

where $\text{lim-Obj} = \text{lim-Obj}$, OF *assms(1,2)* *prems* *lim-Obj-UArr*

]

}

note *the-ntcf-rKe-is-ntcf* = *the-ntcf-rKe-is-ntcf*

[

OF $\mathfrak{K}.\text{is-functor-op}$ $\mathfrak{T}.\text{is-functor-op}$,

unfolded cat-op-simps,

where $\text{lim-Obj} = \langle \text{op-ua } \text{lim-Obj } \mathfrak{K} \rangle$,

unfolded cat-op-simps,

OF this,

simplified

]

show *the-ntcf-lKe* α $\mathfrak{T} \mathfrak{K} \text{lim-Obj} :$

$\mathfrak{T} \mapsto_{CF} \text{the-cf-lKe} \alpha \mathfrak{T} \mathfrak{K} \text{lim-Obj} \circ_{CF} \mathfrak{K} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{A}$

by
 (
 rule *is-ntcf.is-ntcf-op*
 [
 OF the-ntcf-rKe-is-ntcf,
 unfolded cat-op-simps,
 folded the-cf-lKe-def the-ntcf-lKe-def
]
)
 qed

lemma *the-ntcf-rKe-is-cat-rKe*:

assumes $\mathfrak{K} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{T} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{A}$
and $\bigwedge c. c \in_{\circ} \mathfrak{C}(\text{Obj}) \implies \text{lim-Obj } c(\text{UArr}) :$
 $\text{lim-Obj } c(\text{UObj}) <_{CF.\text{lim}} \mathfrak{T} \circ_{CF} c \circ \bigcap_{CF} \mathfrak{K} : c \downarrow_{CF} \mathfrak{K} \mapsto \mapsto_{C\alpha} \mathfrak{A}$
shows *the-ntcf-rKe* $\alpha \mathfrak{T} \mathfrak{K} \text{ lim-Obj} :$
the-cf-rKe $\alpha \mathfrak{T} \mathfrak{K} \text{ lim-Obj} \circ_{CF} \mathfrak{K} \mapsto_{CF.rKe\alpha} \mathfrak{T} : \mathfrak{B} \mapsto_C \mathfrak{C} \mapsto_C \mathfrak{A}$

proof-

let $?UObj = \langle \lambda a. \text{lim-Obj } a(\text{UObj}) \rangle$
let $?UArr = \langle \lambda a. \text{lim-Obj } a(\text{UArr}) \rangle$
let $?the-cf-rKe = \langle \text{the-cf-rKe } \alpha \mathfrak{T} \mathfrak{K} \text{ lim-Obj} \rangle$
let $?the-ntcf-rKe = \langle \text{the-ntcf-rKe } \alpha \mathfrak{T} \mathfrak{K} \text{ lim-Obj} \rangle$

interpret $\mathfrak{K} : \text{is-functor } \alpha \mathfrak{B} \mathfrak{C} \mathfrak{K}$ **by** (*rule assms(1)*)
interpret $\mathfrak{T} : \text{is-functor } \alpha \mathfrak{B} \mathfrak{A} \mathfrak{T}$ **by** (*rule assms(2)*)
interpret *cf-rKe*: *is-functor* $\alpha \mathfrak{C} \mathfrak{A} ?the-cf-rKe$
by (*rule the-cf-rKe-is-functor[OF assms, simplified]*)
interpret *ntcf-rKe*: *is-ntcf* $\alpha \mathfrak{B} \mathfrak{A} \langle ?the-cf-rKe \circ_{CF} \mathfrak{K} \rangle \mathfrak{T} ?the-ntcf-rKe$
by (*intro the-ntcf-rKe-is-ntcf assms(3)*)
 (*cs-concl cs-shallow cs-intro: cat-cs-intros*)+

show *?thesis*

proof(*rule is-cat-rKeI'*)

fix $\mathfrak{G} \in$ **assume** *prems*:

$\mathfrak{G} : \mathfrak{C} \mapsto \mapsto_{C\alpha} \mathfrak{A} \quad \varepsilon : \mathfrak{G} \circ_{CF} \mathfrak{K} \mapsto_{CF} \mathfrak{T} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{A}$

interpret $\mathfrak{G} : \text{is-functor } \alpha \mathfrak{C} \mathfrak{A} \mathfrak{G}$ **by** (*rule prems(1)*)

interpret $\varepsilon : \text{is-ntcf } \alpha \mathfrak{B} \mathfrak{A} \langle \mathfrak{G} \circ_{CF} \mathfrak{K} \rangle \mathfrak{T} \varepsilon$ **by** (*rule prems(2)*)

define ε' **where** $\varepsilon' c =$

[
 ($\lambda A \in_{\circ} c \downarrow_{CF} \mathfrak{K}(\text{Obj}). \varepsilon(\text{NTMap})(\downarrow A(\downarrow 1_{\mathbb{N}})) \circ_{A\mathfrak{A}} \mathfrak{G}(\text{ArrMap})(\downarrow A(\downarrow 2_{\mathbb{N}}))$),
cf-const ($c \downarrow_{CF} \mathfrak{K}$) \mathfrak{A} ($\mathfrak{G}(\text{ObjMap})(\downarrow c)$),
 $\mathfrak{T} \circ_{CF} c \circ \bigcap_{CF} \mathfrak{K}$,
 $c \downarrow_{CF} \mathfrak{K}$,
 \mathfrak{A}
]_o

for c

have ε' -*components*:

$\varepsilon' c(\text{NTMap}) = (\lambda A \in_{\circ} c \downarrow_{CF} \mathfrak{K}(\text{Obj}). \varepsilon(\text{NTMap})(\downarrow A(\downarrow 1_{\mathbb{N}})) \circ_{A\mathfrak{A}} \mathfrak{G}(\text{ArrMap})(\downarrow A(\downarrow 2_{\mathbb{N}})))$

$\varepsilon' c(\text{NTDom}) = \text{cf-const } (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} (\mathfrak{G}(\text{ObjMap})(\downarrow c))$

$\varepsilon' c(\text{NTCod}) = \mathfrak{T} \circ_{CF} c \circ \bigcap_{CF} \mathfrak{K}$

$\varepsilon' c(\text{NTDGDom}) = c \downarrow_{CF} \mathfrak{K}$

$\varepsilon' c(\text{NTDGCod}) = \mathfrak{A}$

for c
unfolding ε' -def *nt-field-simps* **by** (*simp-all add: nat-omega-simps*)
note $[cat\text{-}Kan\text{-}cs\text{-}simps] = \varepsilon'$ -components(2-5)
have $[cat\text{-}Kan\text{-}cs\text{-}simps]: \varepsilon' c(\mathcal{N}TMap)(\downarrow A) = \varepsilon(\mathcal{N}TMap)(\downarrow b) \circ_{A\mathfrak{A}} \mathfrak{G}(\mathcal{A}rrMap)(\downarrow f)$
if $A = [a, b, f]_o$ **and** $[a, b, f]_o \in_o c \downarrow_{CF} \mathfrak{K}(\mathcal{O}bj)$ **for** $A a b c f$
using that unfolding ε' -components **by** (*auto simp: nat-omega-simps*)

have $\varepsilon': \varepsilon' c : \mathfrak{G}(\mathcal{O}bjMap)(\downarrow c) <_{CF.cone} \mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} : c \downarrow_{CF} \mathfrak{K} \mapsto_{C\alpha} \mathfrak{A}$
and ε' -unique: $\exists ! f'$.
 $f' : \mathfrak{G}(\mathcal{O}bjMap)(\downarrow c) \mapsto_{\mathfrak{A}} ?UObj c \wedge$
 $\varepsilon' c = ?UArr c \cdot_{NTCF} ntcf\text{-}const (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} f'$
if $c : c \in_o \mathfrak{C}(\mathcal{O}bj)$ **for** c

proof-
from that have $?the\text{-}cf\text{-}rKe(\mathcal{O}bjMap)(\downarrow c) = ?UObj c$
by
(

cs-concl cs-shallow
cs-simp: *cat-Kan-cs-simps* **cs-intro:** *cat-cs-intros*

)

interpret *lim-c: is-cat-limit*
 $\alpha \langle c \downarrow_{CF} \mathfrak{K} \rangle \mathfrak{A} \langle \mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} \rangle \langle ?UObj c \rangle \langle ?UArr c \rangle$
by (*rule assms(3)[OF that]*)

show $\varepsilon' c : \mathfrak{G}(\mathcal{O}bjMap)(\downarrow c) <_{CF.cone} \mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} : c \downarrow_{CF} \mathfrak{K} \mapsto_{C\alpha} \mathfrak{A}$
proof(*intro is-cat-coneI is-ntcfI'*)

show *vfsequence* ($\varepsilon' c$) **unfolding** ε' -def **by** *simp*
show *vcard* ($\varepsilon' c$) = 5_N **unfolding** ε' -def **by** (*simp add: nat-omega-simps*)
show *vsu* ($\varepsilon' c(\mathcal{N}TMap)$) **unfolding** ε' -components **by** *simp*
show \mathcal{D}_o ($\varepsilon' c(\mathcal{N}TMap)$) = $c \downarrow_{CF} \mathfrak{K}(\mathcal{O}bj)$ **unfolding** ε' -components **by** *simp*
show $\varepsilon' c(\mathcal{N}TMap)(\downarrow A) :$
 $cf\text{-}const (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} (\mathfrak{G}(\mathcal{O}bjMap)(\downarrow c))(\mathcal{O}bjMap)(\downarrow A) \mapsto_{\mathfrak{A}}$
 $(\mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K})(\mathcal{O}bjMap)(\downarrow A)$
if $A \in_o c \downarrow_{CF} \mathfrak{K}(\mathcal{O}bj)$ **for** A

proof-
from that prems c **obtain** $b f$
where $A\text{-def}: A = [0, b, f]_o$
and $b : b \in_o \mathfrak{B}(\mathcal{O}bj)$
and $f : f : c \mapsto_{\mathfrak{C}} \mathfrak{K}(\mathcal{O}bjMap)(\downarrow b)$
by *auto*

from that prems $f c$ **that** $b f$ **show** *thesis*
unfolding $A\text{-def}$
by
(

cs-concl cs-shallow
cs-simp: *cat-cs-simps cat-Kan-cs-simps cat-comma-cs-simps*
cs-intro: *cat-cs-intros cat-comma-cs-intros*

)

qed
show
 $\varepsilon' c(\mathcal{N}TMap)(\downarrow B) \circ_{A\mathfrak{A}} cf\text{-}const (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} (\mathfrak{G}(\mathcal{O}bjMap)(\downarrow c))(\mathcal{A}rrMap)(\downarrow F) =$
 $(\mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K})(\mathcal{A}rrMap)(\downarrow F) \circ_{A\mathfrak{A}} \varepsilon' c(\mathcal{N}TMap)(\downarrow A)$
if $F : A \mapsto_c \downarrow_{CF} \mathfrak{K} B$ **for** $A B F$

proof-
from that c
obtain $b f b' f' k$
where $F\text{-def}: F = [[0, b, f]_o, [0, b', f']_o, [0, k]_o]_o$
and $A\text{-def}: A = [0, b, f]_o$
and $B\text{-def}: B = [0, b', f']_o$
and $k : k : b \mapsto_{\mathfrak{B}} b'$

```

    and f: f : c ↦ℳ ℝ(ObjMap)(b)
    and f': f' : c ↦ℳ ℝ(ObjMap)(b')
    and f'-def: ℝ(ArrMap)(k) ∘A f = f'
  by auto
  from c that k f f' show ?thesis
  unfolding F-def A-def B-def
  by
  (
    cs-concl
    cs-simp:
      cat-cs-simps
      cat-comma-cs-simps
      cat-Kan-cs-simps
      ε.ntcf-Comp-commute''
      f'-def[symmetric]
    cs-intro: cat-cs-intros cat-comma-cs-intros
  )
  qed
  qed
  (
    use c that in
    ⟨cs-concl cs-simp: cat-Kan-cs-simps cs-intro: cat-cs-intros⟩
  )+
  from is-cat-limit.cat-lim-ua-fo[OF assms(β)[OF that] this] show
  ∃!f'.
  f' : ℳ(ObjMap)(c) ↦ℳ ?UObj c ∧
  ε' c = ?UArr c •NTCF ntcf-const (c ↓CF ℝ) ℳ f'
  by simp
  qed

  define σ :: V where
  σ =
  [
    (
      λc ∈o ℳ(Obj). THE f.
      f : ℳ(ObjMap)(c) ↦ℳ ?UObj c ∧
      ε' c = ?UArr c •NTCF ntcf-const (c ↓CF ℝ) ℳ f
    ),
    ℳ,
    ?the-cf-rKe,
    ℳ,
    ℳ
  ]o

  have σ-components:
  σ(NTMap) =
  (
    λc ∈o ℳ(Obj). THE f.
    f : ℳ(ObjMap)(c) ↦ℳ ?UObj c ∧
    ε' c = ?UArr c •NTCF ntcf-const (c ↓CF ℝ) ℳ f
  )
  σ(NTDom) = ℳ
  σ(NTCod) = ?the-cf-rKe
  σ(NTDGDom) = ℳ
  σ(NTDGCod) = ℳ
  unfolding σ-def nt-field-simps by (simp-all add: nat-omega-simps)

  note [cat-Kan-cs-simps] = σ-components(2-5)

```

have σ -*NTMap-app-def*: $\sigma(\text{NTMap})(\downarrow c) =$
 (*THE* f .
 $f : \mathfrak{G}(\text{ObjMap})(\downarrow c) \mapsto_{\mathfrak{A}} ?UObj\ c \wedge$
 $\varepsilon' c = ?UArr\ c \cdot_{NTCF}\ ntcf\text{-const}\ (c \downarrow_{CF}\ \mathfrak{K})\ \mathfrak{A}\ f$
)
if $c \in_{\circ} \mathfrak{C}(\text{Obj})$ **for** c
using *that unfolding σ -components by simp*

have σ -*NTMap-app-is-arr*: $\sigma(\text{NTMap})(\downarrow c) : \mathfrak{G}(\text{ObjMap})(\downarrow c) \mapsto_{\mathfrak{A}} ?UObj\ c$
and ε' - σ -*commute*:
 $\varepsilon' c = ?UArr\ c \cdot_{NTCF}\ ntcf\text{-const}\ (c \downarrow_{CF}\ \mathfrak{K})\ \mathfrak{A}\ (\sigma(\text{NTMap})(\downarrow c))$
and σ -*NTMap-app-unique*:
 \llbracket
 $f : \mathfrak{G}(\text{ObjMap})(\downarrow c) \mapsto_{\mathfrak{A}} ?UObj\ c;$
 $\varepsilon' c = ?UArr\ c \cdot_{NTCF}\ ntcf\text{-const}\ (c \downarrow_{CF}\ \mathfrak{K})\ \mathfrak{A}\ f$
 $\rrbracket \implies f = \sigma(\text{NTMap})(\downarrow c)$
if $c : c \in_{\circ} \mathfrak{C}(\text{Obj})$ **for** $c\ f$

proof-
have
 $\sigma(\text{NTMap})(\downarrow c) : \mathfrak{G}(\text{ObjMap})(\downarrow c) \mapsto_{\mathfrak{A}} ?UObj\ c \wedge$
 $\varepsilon' c = ?UArr\ c \cdot_{NTCF}\ ntcf\text{-const}\ (c \downarrow_{CF}\ \mathfrak{K})\ \mathfrak{A}\ (\sigma(\text{NTMap})(\downarrow c))$
by
 (*cs-concl cs-shallow*
cs-simp: cat-Kan-cs-simps σ -NTMap-app-def
cs-intro: theI' ε' -unique that
)
then show $\sigma(\text{NTMap})(\downarrow c) : \mathfrak{G}(\text{ObjMap})(\downarrow c) \mapsto_{\mathfrak{A}} ?UObj\ c$
and $\varepsilon' c = ?UArr\ c \cdot_{NTCF}\ ntcf\text{-const}\ (c \downarrow_{CF}\ \mathfrak{K})\ \mathfrak{A}\ (\sigma(\text{NTMap})(\downarrow c))$
by *simp-all*
with c ε' -*unique*[*OF* c] **show** $f = \sigma(\text{NTMap})(\downarrow c)$
if $f : \mathfrak{G}(\text{ObjMap})(\downarrow c) \mapsto_{\mathfrak{A}} ?UObj\ c$
and $\varepsilon' c = ?UArr\ c \cdot_{NTCF}\ ntcf\text{-const}\ (c \downarrow_{CF}\ \mathfrak{K})\ \mathfrak{A}\ f$
using *that by metis*

qed

have σ -*NTMap-app-is-arr'*[*cat-Kan-cs-intros*]: $\sigma(\text{NTMap})(\downarrow c) : a \mapsto_{\mathfrak{A}'} b$
if $c \in_{\circ} \mathfrak{C}(\text{Obj})$
and $a = \mathfrak{G}(\text{ObjMap})(\downarrow c)$
and $b = ?UObj\ c$
and $\mathfrak{A}' = \mathfrak{A}$
for $\mathfrak{A}'\ a\ b\ c$
by (*simp add: that σ -NTMap-app-is-arr*)

have ε' -*NTMap-app-def*:
 $\varepsilon' c(\text{NTMap})(\downarrow A) =$
 $(?UArr\ c \cdot_{NTCF}\ ntcf\text{-const}\ (c \downarrow_{CF}\ \mathfrak{K})\ \mathfrak{A}\ (\sigma(\text{NTMap})(\downarrow c)))(\text{NTMap})(\downarrow A)$
if $A \in_{\circ} c \downarrow_{CF}\ \mathfrak{K}(\text{Obj})$ **and** $c \in_{\circ} \mathfrak{C}(\text{Obj})$ **for** $A\ c$
using ε' - σ -*commute*[*OF that(2)*] **by** *simp*

have εb - $\mathfrak{G}f$:
 $\varepsilon(\text{NTMap})(\downarrow b) \circ_{A\mathfrak{A}} \mathfrak{G}(\text{ArrMap})(\downarrow f) =$
 $?UArr\ c(\text{NTMap})(\downarrow a, b, f) \bullet_{A\mathfrak{A}} \sigma(\text{NTMap})(\downarrow c)$
if $A = [a, b, f]_{\circ}$ **and** $A \in_{\circ} c \downarrow_{CF}\ \mathfrak{K}(\text{Obj})$ **and** $c \in_{\circ} \mathfrak{C}(\text{Obj})$
for $A\ a\ b\ c\ f$

proof-
interpret *lim-c: is-cat-limit*

$\alpha \langle c \downarrow_{CF} \mathfrak{K} \rangle \mathfrak{A} \langle \mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} \rangle \langle ?UObj \ c \rangle \langle ?UArr \ c \rangle$
by (*rule assms*(3)[*OF that*(3)])
from *that* **have** $b : b \in_0 \mathfrak{B}(\mathfrak{Obj})$ **and** $f : f : c \mapsto_{\mathfrak{C}} \mathfrak{K}(\mathfrak{ObjMap})(\downarrow b)$
by *blast+*
show
 $\varepsilon(\downarrow NTMap)(\downarrow b) \circ_{A\mathfrak{A}} \mathfrak{G}(\downarrow ArrMap)(\downarrow f) =$
 $?UArr \ c(\downarrow NTMap)(\downarrow a, b, f) \bullet \circ_{A\mathfrak{A}} \sigma(\downarrow NTMap)(\downarrow c)$
using ε' -*NTMap-app-def*[*OF that*(2,3)] *that*(2,3)
unfolding *that*(1)
by
(

cs-prems **cs-shallow**
cs-simp: *cat-cs-simps* *cat-Kan-cs-simps*
cs-intro: *cat-cs-intros* *cat-Kan-cs-intros*

)

qed

show $\exists ! \sigma$.
 $\sigma : \mathfrak{G} \mapsto_{CF} ?the\text{-}cf\text{-}rKe : \mathfrak{C} \mapsto \mapsto_{C\alpha} \mathfrak{A} \wedge$
 $\varepsilon = ?the\text{-}ntcf\text{-}rKe \cdot_{NTCF} (\sigma \circ_{NTCF-CF} \mathfrak{K})$
proof(*intro exII*[**where** $a = \sigma$] *conjI*; (*elim conjE*)?)

define τ **where** $\tau \ a \ b \ f =$
[

(

 $\lambda F \in_0 b \downarrow_{CF} \mathfrak{K}(\mathfrak{Obj}).$
 $?UArr \ b(\downarrow NTMap)(\downarrow F) \circ_{A\mathfrak{A}} \sigma(\downarrow NTMap)(\downarrow b) \circ_{A\mathfrak{A}} \mathfrak{G}(\downarrow ArrMap)(\downarrow f)$

),
cf-const ($b \downarrow_{CF} \mathfrak{K}$) $\mathfrak{A} (\mathfrak{G}(\mathfrak{ObjMap})(\downarrow a))$,
 $\mathfrak{T} \circ_{CF} b \circ \sqcap_{CF} \mathfrak{K}$,
 $b \downarrow_{CF} \mathfrak{K}$,
 \mathfrak{A}

]

for $a \ b \ f$

have τ -*components*:
 $\tau \ a \ b \ f(\downarrow NTMap) =$
(

 $\lambda F \in_0 b \downarrow_{CF} \mathfrak{K}(\mathfrak{Obj}).$
 $?UArr \ b(\downarrow NTMap)(\downarrow F) \circ_{A\mathfrak{A}} \sigma(\downarrow NTMap)(\downarrow b) \circ_{A\mathfrak{A}} \mathfrak{G}(\downarrow ArrMap)(\downarrow f)$

)

 $\tau \ a \ b \ f(\downarrow NTDom) = \text{cf-const} \ (b \downarrow_{CF} \mathfrak{K}) \ \mathfrak{A} \ (\mathfrak{G}(\mathfrak{ObjMap})(\downarrow a))$
 $\tau \ a \ b \ f(\downarrow NTCod) = \mathfrak{T} \circ_{CF} b \circ \sqcap_{CF} \mathfrak{K}$
 $\tau \ a \ b \ f(\downarrow NTDGDom) = b \downarrow_{CF} \mathfrak{K}$
 $\tau \ a \ b \ f(\downarrow NTDGCod) = \mathfrak{A}$
for $a \ b \ f$

unfolding τ -*def nt-field-simps* **by** (*simp-all add: nat-omega-simps*)

note [*cat-Kan-cs-simps*] = τ -*components*(2-5)

have τ -*NTMap-app*[*cat-Kan-cs-simps*]:

$\tau \ a \ b \ f(\downarrow NTMap)(\downarrow F) =$
 $?UArr \ b(\downarrow NTMap)(\downarrow F) \circ_{A\mathfrak{A}} \sigma(\downarrow NTMap)(\downarrow b) \circ_{A\mathfrak{A}} \mathfrak{G}(\downarrow ArrMap)(\downarrow f)$
if $F \in_0 b \downarrow_{CF} \mathfrak{K}(\mathfrak{Obj})$ **for** $a \ b \ f \ F$
using *that* **unfolding** τ -*components* **by** *auto*

have $\tau : \tau \ a \ b \ f :$
 $\mathfrak{G}(\mathfrak{ObjMap})(\downarrow a) \langle_{CF.cone} \mathfrak{T} \circ_{CF} b \circ \sqcap_{CF} \mathfrak{K} : b \downarrow_{CF} \mathfrak{K} \mapsto \mapsto_{C\alpha} \mathfrak{A}$
if f -*is-arr*: $f : a \mapsto_{\mathfrak{C}} b$ **for** $a \ b \ f$
proof-

note $f = \mathfrak{K}.HomCod.cat-is-arrD[OF\ that]$
note $lim-a = assms(3)[OF\ f(2)]$ **and** $lim-b = assms(3)[OF\ f(3)]$

interpret $lim-b: is-cat-limit$
 $\alpha \langle b \downarrow_{CF} \mathfrak{K} \rangle \mathfrak{A} \langle \mathfrak{T} \circ_{CF} b \circ \square_{CF} \mathfrak{K} \rangle \langle ?UObj\ b \rangle \langle ?UArr\ b \rangle$
by (rule $lim-b$)

note $lim-b.cat-cone-Comp-commute[cat-cs-simps\ del]$

from f **have** $a: a \in_{\circ} \mathfrak{C}(\downarrow Obj)$ **and** $b: b \in_{\circ} \mathfrak{C}(\downarrow Obj)$ **by** *auto*

show *?thesis*

proof(*intro is-cat-coneI is-ntcfI'*)

show *vfsequence* ($\tau\ a\ b\ f$) **unfolding** τ -*def* **by** *simp*

show *vcard* ($\tau\ a\ b\ f$) = $5_{\mathbb{N}}$

unfolding τ -*def* **by** (*simp add: nat-omega-simps*)

show *vsv* ($\tau\ a\ b\ f(\downarrow NTMap)$) **unfolding** τ -*components* **by** *auto*

show \mathcal{D}_{\circ} ($\tau\ a\ b\ f(\downarrow NTMap)$) = $b \downarrow_{CF} \mathfrak{K}(\downarrow Obj)$ **by** (*auto simp: τ -components*)

show $\tau\ a\ b\ f(\downarrow NTMap)(\downarrow A)$:

cf-const ($b \downarrow_{CF} \mathfrak{K}$) \mathfrak{A} ($\mathfrak{G}(\downarrow ObjMap)(\downarrow a)$)($\downarrow ObjMap)(\downarrow A) \mapsto_{\mathfrak{A}}$

($\mathfrak{T} \circ_{CF} b \circ \square_{CF} \mathfrak{K}$)($\downarrow ObjMap)(\downarrow A)$)

if $A \in_{\circ} b \downarrow_{CF} \mathfrak{K}(\downarrow Obj)$ **for** A

proof-

from *that f-is-arr* **obtain** $b' f'$

where A -*def*: $A = [0, b', f']_{\circ}$

and b' : $b' \in_{\circ} \mathfrak{B}(\downarrow Obj)$

and f' : $f' : b \mapsto_{\mathfrak{C}} \mathfrak{K}(\downarrow ObjMap)(\downarrow b')$

by *auto*

from *f-is-arr* **that** $b' f'$ **a** b **show** *?thesis*

unfolding A -*def*

by

(

cs-concl cs-shallow

cs-simp: *cat-cs-simps cat-comma-cs-simps cat-Kan-cs-simps*

cs-intro: *cat-cs-intros cat-comma-cs-intros cat-Kan-cs-intros*

)

qed

show

$\tau\ a\ b\ f(\downarrow NTMap)(\downarrow B) \circ_{A\mathfrak{A}}$

cf-const ($b \downarrow_{CF} \mathfrak{K}$) \mathfrak{A} ($\mathfrak{G}(\downarrow ObjMap)(\downarrow a)$)($\downarrow ArrMap)(\downarrow F) =$

($\mathfrak{T} \circ_{CF} b \circ \square_{CF} \mathfrak{K}$)($\downarrow ArrMap)(\downarrow F) \circ_{A\mathfrak{A}} \tau\ a\ b\ f(\downarrow NTMap)(\downarrow A)$)

if $F : A \mapsto_b \downarrow_{CF} \mathfrak{K}\ B$ **for** $A\ B\ F$

proof-

from *that* **have** $F: F : A \mapsto_b \downarrow_{CF} \mathfrak{K}\ B$

by (*auto intro: is-arrI*)

with *f-is-arr* **obtain** $b' f' b'' f'' h'$

where F -*def*: $F = [[0, b', f']_{\circ}, [0, b'', f'']_{\circ}, [0, h']_{\circ}]_{\circ}$

and A -*def*: $A = [0, b', f']_{\circ}$

and B -*def*: $B = [0, b'', f'']_{\circ}$

and h' : $h' : b' \mapsto_{\mathfrak{B}} b''$

and f' : $f' : b \mapsto_{\mathfrak{C}} \mathfrak{K}(\downarrow ObjMap)(\downarrow b')$

and f'' : $f'' : b \mapsto_{\mathfrak{C}} \mathfrak{K}(\downarrow ObjMap)(\downarrow b'')$

and f'' -*def*: $\mathfrak{K}(\downarrow ArrMap)(\downarrow h') \circ_{A\mathfrak{C}} f' = f''$

by *auto*

from

$lim-b.ntcf-Comp-commute[OF\ that]$

```

    that f-is-arr g' h' f' f''
  have [cat-Kan-cs-simps]:
    ?UArr b(NTMap)(0, b'',  $\mathfrak{R}$ (ArrMap)(h')  $\circ_{A\mathfrak{C}}$  f')• =
       $\mathfrak{T}$ (ArrMap)(h')  $\circ_{A\mathfrak{A}}$  ?UArr b(NTMap)(0, b', f')•
  unfolding F-def A-def B-def
  by
    (
      cs-prems
      cs-simp:
        cat-cs-simps cat-comma-cs-simps f''-def[symmetric]
      cs-intro: cat-cs-intros cat-comma-cs-intros
    )
  from f-is-arr that g' h' f' f'' show ?thesis
  unfolding F-def A-def B-def
  by
    (
      cs-concl
      cs-simp:
        cat-cs-simps
        cat-Kan-cs-simps
        cat-comma-cs-simps
        f''-def[symmetric]
      cs-intro:
        cat-cs-intros cat-Kan-cs-intros cat-comma-cs-intros
    )+
  qed

  qed
  (
    use that f-is-arr in
    <
      cs-concl
      cs-simp: cat-cs-simps cat-Kan-cs-simps cs-intro: cat-cs-intros
    >
  )+
  qed

  show  $\sigma : \sigma : \mathfrak{G} \mapsto_{CF} ?the-cf-rKe : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{A}$ 
  proof(rule is-ntcfI')

    show vsequence  $\sigma$  unfolding  $\sigma$ -def by simp
    show vcard  $\sigma = 5_{\mathbb{N}}$  unfolding  $\sigma$ -def by (simp add: nat-omega-simps)
    show vsu ( $\sigma$ (NTMap)) unfolding  $\sigma$ -components by auto
    show  $\mathcal{D}_\circ$  ( $\sigma$ (NTMap)) =  $\mathfrak{C}$ (Obj) unfolding  $\sigma$ -components by simp
    show  $\sigma$ (NTMap)(a) :  $\mathfrak{G}$ (ObjMap)(a)  $\mapsto_{\mathfrak{A}}$  ?the-cf-rKe(ObjMap)(a)
    if  $a \in_\circ \mathfrak{C}$ (Obj) for a
    using that
    by
      (
        cs-concl
        cs-simp: cat-cs-simps cat-Kan-cs-simps
        cs-intro: cat-cs-intros cat-Kan-cs-intros
      )

  then have [cat-Kan-cs-intros]:  $\sigma$ (NTMap)(a) :  $b \mapsto_{\mathfrak{A}} c$ 
  if  $a \in_\circ \mathfrak{C}$ (Obj)
  and  $b = \mathfrak{G}$ (ObjMap)(a)
  and  $c = ?the-cf-rKe$ (ObjMap)(a)

```

for $a b c$
 using *that(1) unfolding that(2,3) by simp*

show

$\sigma(\text{NTMap})(\downarrow b) \circ_{A\mathfrak{A}} \mathfrak{G}(\downarrow \text{ArrMap})(\downarrow f) =$
 $\text{?the-cf-rKe}(\downarrow \text{ArrMap})(\downarrow f) \circ_{A\mathfrak{A}} \sigma(\downarrow \text{NTMap})(\downarrow a)$
 if *f-is-arr: f : a $\mapsto_{\mathfrak{C}}$ b for a b f*

proof-

note $f = \mathfrak{K}.\text{HomCod.cat-is-arrD}[OF \text{ that}]$

note $\text{lim-a} = \text{assms}(3)[OF f(2)]$ and $\text{lim-b} = \text{assms}(3)[OF f(3)]$

interpret *lim-a: is-cat-limit*

$\alpha \langle a \downarrow_{CF} \mathfrak{K} \rangle \mathfrak{A} \langle \mathfrak{T} \circ_{CF} a \circ \sqcap_{CF} \mathfrak{K} \rangle \langle ?UObj a \rangle \langle ?UArr a \rangle$
 by (rule *lim-a*)

interpret *lim-b: is-cat-limit*

$\alpha \langle b \downarrow_{CF} \mathfrak{K} \rangle \mathfrak{A} \langle \mathfrak{T} \circ_{CF} b \circ \sqcap_{CF} \mathfrak{K} \rangle \langle ?UObj b \rangle \langle ?UArr b \rangle$
 by (rule *lim-b*)

from *f* have $a: a \in_{\circ} \mathfrak{C}(\downarrow Obj)$ and $b: b \in_{\circ} \mathfrak{C}(\downarrow Obj)$ by *auto*

from *lim-b.cat-lim-unique-cone'[OF $\tau[OF \text{ that}]$]* obtain g'

where $g': g' : \mathfrak{G}(\downarrow ObjMap)(\downarrow a) \mapsto_{\mathfrak{A}} ?UObj b$

and $\tau\text{-NTMap-app}: \wedge A. A \in_{\circ} (b \downarrow_{CF} \mathfrak{K}(\downarrow Obj)) \implies$

$\tau a b f(\downarrow \text{NTMap})(\downarrow A) = ?UArr b(\downarrow \text{NTMap})(\downarrow A) \circ_{A\mathfrak{A}} g'$

and $g'\text{-unique}: \wedge g''.$

\llbracket
 $g'' : \mathfrak{G}(\downarrow ObjMap)(\downarrow a) \mapsto_{\mathfrak{A}} ?UObj b;$
 $\wedge A. A \in_{\circ} b \downarrow_{CF} \mathfrak{K}(\downarrow Obj) \implies$
 $\tau a b f(\downarrow \text{NTMap})(\downarrow A) = ?UArr b(\downarrow \text{NTMap})(\downarrow A) \circ_{A\mathfrak{A}} g''$
 $\rrbracket \implies g'' = g'$

by *metis*

have *lim-Obj-a-f \mathfrak{K} [symmetric, cat-Kan-cs-simps]*:

$?UArr a(\downarrow \text{NTMap})(\downarrow a', b', f' \circ_{A\mathfrak{C}} f) \bullet =$

$?UArr b(\downarrow \text{NTMap})(\downarrow A) \circ_{A\mathfrak{A}} \text{?the-cf-rKe}(\downarrow \text{ArrMap})(\downarrow f)$

if $A = [a', b', f']_{\circ}$ and $A \in_{\circ} b \downarrow_{CF} \mathfrak{K}(\downarrow Obj)$ for $A a' b' f'$

proof-

from *that(2) f-is-arr* have $a'\text{-def}: a' = 0$

and $b': b' \in_{\circ} \mathfrak{B}(\downarrow Obj)$

and $f': f' : b \mapsto_{\mathfrak{C}} \mathfrak{K}(\downarrow ObjMap)(\downarrow b')$

unfolding *that(1)* by *auto*

show *?thesis*

unfolding *that(1)*

by

(

rule

lim-Obj-the-cf-rKe-commute

[

where *lim-Obj=lim-Obj,*

OF

assms(1,2)

lim-a

lim-b

f-is-arr

that(2)[unfolded that(1)]

]

)

```

qed
{
  fix a' b' f' A
  note  $\mathfrak{T}.HomCod.cat\text{-}assoc\text{-}helper$ [
    where  $h = \langle ?UArr\ b\ (NTMap)\ (a', b', f') \bullet \rangle$ 
    and  $g = \langle ?the\text{-}cf\text{-}rKe\ (ArrMap)\ (f) \rangle$ 
    and  $q = \langle ?UArr\ a\ (NTMap)\ (a', b', f' \circ_{A\mathfrak{C}} f) \bullet \rangle$ 
  ]
}
note [cat-Kan-cs-simps] = this

show ?thesis
proof(rule trans-sym[where s=g'])
  show  $\sigma(NTMap)\ (b) \circ_{A\mathfrak{A}} \mathfrak{G}(ArrMap)\ (f) = g'$ 
  proof(rule g'-unique)
    from that show
       $\sigma(NTMap)\ (b) \circ_{A\mathfrak{A}} \mathfrak{G}(ArrMap)\ (f) : \mathfrak{G}(ObjMap)\ (a) \mapsto_{A\mathfrak{A}} ?UObj\ b$ 
      by (cs-concl cs-intro: cat-cs-intros cat-Kan-cs-intros)
    fix A assume prems':  $A \in_0 b \downarrow_{CF} \mathfrak{R}(Obj)$ 
    with f-is-arr obtain b' f'
      where A-def:  $A = [0, b', f']_0$ 
      and b':  $b' \in_0 \mathfrak{B}(Obj)$ 
      and f':  $f' : b \mapsto_{\mathfrak{C}} \mathfrak{R}(ObjMap)\ (b')$ 
    by auto
    from f-is-arr prems' show
       $\tau\ a\ b\ f\ (NTMap)\ (A) =$ 
       $?UArr\ b\ (NTMap)\ (A) \circ_{A\mathfrak{A}} (\sigma(NTMap)\ (b) \circ_{A\mathfrak{A}} \mathfrak{G}(ArrMap)\ (f))$ 
    unfolding A-def
    by
      (
        cs-concl
        cs-simp: cat-cs-simps cat-Kan-cs-simps
        cs-intro: cat-cs-intros cat-Kan-cs-intros
      )
  qed
  show ?the-cf-rKe(ArrMap)(f)  $\circ_{A\mathfrak{A}}$   $\sigma(NTMap)\ (a) = g'$ 
  proof(rule g'-unique)
    fix A assume prems':  $A \in_0 b \downarrow_{CF} \mathfrak{R}(Obj)$ 
    with f-is-arr obtain b' f'
      where A-def:  $A = [0, b', f']_0$ 
      and b':  $b' \in_0 \mathfrak{B}(Obj)$ 
      and f':  $f' : b \mapsto_{\mathfrak{C}} \mathfrak{R}(ObjMap)\ (b')$ 
    by auto
    {
      fix a' b' f' A
      note  $\mathfrak{T}.HomCod.cat\text{-}assoc\text{-}helper$ 
      [
        where  $h = \langle ?UArr\ b\ (NTMap)\ (a', b', f') \bullet \rangle$ 
        and  $g = \langle \sigma(NTMap)\ (b) \rangle$ 
        and  $q = \langle \varepsilon(NTMap)\ (b') \circ_{A\mathfrak{A}} \mathfrak{G}(ArrMap)\ (f') \rangle$ 
      ]
    }
    note [cat-Kan-cs-simps] =
      this
       $\varepsilon b\text{-}\mathfrak{G}f[OF\ A\text{-}def\ prems'\ b,\ symmetric]$ 
       $\varepsilon b\text{-}\mathfrak{G}f[symmetric]$ 
    from f-is-arr prems' b' f' show
       $\tau\ a\ b\ f\ (NTMap)\ (A) =$ 

```

```

      ?UArr b(NTMap)(A) ∘A
      (?the-cf-rKe(ArrMap)(f) ∘A σ(NTMap)(a))
unfolding A-def
by
  (
    cs-concl
    cs-simp:
      cat-cs-simps
      cat-Kan-cs-simps
      cat-comma-cs-simps
      cat-op-simps
    cs-intro:
      cat-cs-intros
      cat-Kan-cs-intros
      cat-comma-cs-intros
      cat-op-intros
  )
qed
  (
    use that in
    <
      cs-concl
      cs-simp: cat-Kan-cs-simps
      cs-intro: cat-cs-intros cat-Kan-cs-intros
    >
  )
qed
qed
qed
  (
    cs-concl cs-shallow
    cs-simp: cat-cs-simps cat-Kan-cs-simps
    cs-intro: cat-cs-intros
  )+
then interpret σ: is-ntcf α C A G <?the-cf-rKe> σ by simp

show ε = ?the-ntcf-rKe •NTCF (σ ∘NTCF-CF R)
proof(rule ntcf-eqI)
  have dom-lhs: Do (ε(NTMap)) = B(Obj)
  by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  have dom-rhs: Do ((?the-ntcf-rKe •NTCF (σ ∘NTCF-CF R))(NTMap)) = B(Obj)
  by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  show ε(NTMap) = (?the-ntcf-rKe •NTCF (σ ∘NTCF-CF R))(NTMap)
  proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
  fix b assume prems': b ∈o B(Obj)
  note [cat-Kan-cs-simps] = εb-Gf[
    where f = <C(CId)(ObjMap)(b)>, and c = <R(ObjMap)(b)>, symmetric
  ]
  from prems' σ show
    ε(NTMap)(b) = (?the-ntcf-rKe •NTCF (σ ∘NTCF-CF R))(NTMap)(b)
  by
  (
    cs-concl
    cs-simp: cat-cs-simps cat-comma-cs-simps cat-Kan-cs-simps
    cs-intro: cat-cs-intros cat-comma-cs-intros cat-Kan-cs-intros
  )
  qed (cs-concl cs-intro: cat-cs-intros V-cs-intros)
qed (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)+

```

```

fix  $\sigma'$  assume prems':
   $\sigma' : \mathfrak{G} \mapsto_{CF} ?the\text{-}cf\text{-}rKe : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{A}$ 
   $\varepsilon = ?the\text{-}ntcf\text{-}rKe \cdot_{NTCF} (\sigma' \circ_{NTCF-CF} \mathfrak{K})$ 

interpret  $\sigma'$ : is-ntcf  $\alpha$   $\mathfrak{C}$   $\mathfrak{A}$   $\mathfrak{G}$   $\langle ?the\text{-}cf\text{-}rKe \rangle$   $\sigma'$  by (rule prems'(1))

have  $\varepsilon$ -NTMap-app[symmetric, cat-Kan-cs-simps]:
   $\varepsilon(\downarrow NTMap)(\downarrow b')$  =
     $?UArr (\mathfrak{K}(\downarrow ObjMap)(\downarrow b'))(\downarrow NTMap)(\downarrow a', b', \mathfrak{C}(\downarrow CId)(\downarrow \mathfrak{K}(\downarrow ObjMap)(\downarrow b')))\bullet \circ_{A\mathfrak{A}}$ 
     $\sigma'(\downarrow NTMap)(\downarrow \mathfrak{K}(\downarrow ObjMap)(\downarrow b'))$ 
  if  $b' \in_0 \mathfrak{B}(\downarrow Obj)$  and  $a' = 0$  for  $a' b'$ 
proof-
  from prems'(2) have  $\varepsilon$ -NTMap-app:
     $\varepsilon(\downarrow NTMap)(\downarrow b') = (?the\text{-}ntcf\text{-}rKe \cdot_{NTCF} (\sigma' \circ_{NTCF-CF} \mathfrak{K}))(\downarrow NTMap)(\downarrow b')$ 
  for  $b'$ 
  by simp
  show thesis
  using  $\varepsilon$ -NTMap-app[of b'] that(1)
  unfolding that(2)
  by
  (
    cs-prems cs-shallow
    cs-simp: cat-cs-simps cat-comma-cs-simps cat-Kan-cs-simps
    cs-intro: cat-cs-intros cat-comma-cs-intros
  )
qed
{
  fix  $a' b' f' A$ 
  note  $\mathfrak{T}.HomCod.cat\text{-}assoc\text{-}helper$ 
  [
    where  $h = \langle ?UArr (\mathfrak{K}(\downarrow ObjMap)(\downarrow b'))(\downarrow NTMap)(\downarrow a', b', \mathfrak{C}(\downarrow CId)(\downarrow \mathfrak{K}(\downarrow ObjMap)(\downarrow b')))\bullet \rangle$ 
    and  $g = \langle \sigma'(\downarrow NTMap)(\downarrow \mathfrak{K}(\downarrow ObjMap)(\downarrow b')) \rangle$ 
    and  $q = \langle \varepsilon(\downarrow NTMap)(\downarrow b') \rangle$ 
  ]
}
note [cat-Kan-cs-simps] = this  $\varepsilon b$ - $\mathfrak{G}f$ [symmetric]
{
  fix  $a' b' f' A$ 
  note  $\mathfrak{T}.HomCod.cat\text{-}assoc\text{-}helper$ 
  [
    where  $h =$ 
       $\langle ?UArr (\mathfrak{K}(\downarrow ObjMap)(\downarrow b'))(\downarrow NTMap)(\downarrow a', b', \mathfrak{C}(\downarrow CId)(\downarrow \mathfrak{K}(\downarrow ObjMap)(\downarrow b')))\bullet \rangle$ 
    and  $g = \langle \sigma(\downarrow NTMap)(\downarrow \mathfrak{K}(\downarrow ObjMap)(\downarrow b')) \rangle$ 
    and  $q = \langle \varepsilon(\downarrow NTMap)(\downarrow b') \rangle$ 
  ]
}
note [cat-Kan-cs-simps] = this

show  $\sigma' = \sigma$ 
proof(rule ntcf-eqI)

  show  $\sigma' : \mathfrak{G} \mapsto_{CF} ?the\text{-}cf\text{-}rKe : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{A}$  by (rule prems'(1))
  show  $\sigma : \mathfrak{G} \mapsto_{CF} ?the\text{-}cf\text{-}rKe : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{A}$  by (rule  $\sigma$ )

  have dom-lhs:  $\mathcal{D}_\circ (\sigma(\downarrow NTMap)) = \mathfrak{C}(\downarrow Obj)$ 
  by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  have dom-rhs:  $\mathcal{D}_\circ (\sigma'(\downarrow NTMap)) = \mathfrak{C}(\downarrow Obj)$ 

```

by (*cs-concl cs-shallow cs-simp: cat-cs-simps*)

show $\sigma'(\mathcal{N}TMap) = \sigma(\mathcal{N}TMap)$
proof(*rule vsv-eqI, unfold dom-lhs dom-rhs*)

fix c **assume** $prems'$: $c \in_{\circ} \mathfrak{C}(\mathcal{O}bj)$

note $lim-c = assms(\mathcal{B})[OF\ prems']$

interpret $lim-c$: *is-cat-limit*

$\alpha \langle c \downarrow_{CF} \mathfrak{R} \rangle \mathfrak{A} \langle \mathfrak{T} \circ_{CF} c \circ \square_{CF} \mathfrak{R} \rangle \langle ?UObj\ c \rangle \langle ?UArr\ c \rangle$

by (*rule lim-c*)

from $prems'$ **have** $CId-c$: $\mathfrak{C}(\mathcal{C}Id)(\downarrow c) : c \mapsto_{\mathfrak{C}} c$

by (*cs-concl cs-shallow cs-intro: cat-cs-intros*)

from $lim-c$.*cat-lim-unique-cone*'[*OF* τ [*OF* $CId-c$]] **obtain** f

where f : $f : \mathfrak{G}(\mathcal{O}bjMap)(\downarrow c) \mapsto_{\mathfrak{A}} ?UObj\ c$

and $\wedge A$. $A \in_{\circ} c \downarrow_{CF} \mathfrak{R}(\mathcal{O}bj) \implies$

$\tau\ c\ c\ (\mathfrak{C}(\mathcal{C}Id)(\downarrow c))(\mathcal{N}TMap)(\downarrow A) = ?UArr\ c(\mathcal{N}TMap)(\downarrow A) \circ_{A\mathfrak{A}} f$

and f -*unique*: $\wedge f'$.

[[

$f' : \mathfrak{G}(\mathcal{O}bjMap)(\downarrow c) \mapsto_{\mathfrak{A}} ?UObj\ c;$

$\wedge A$. $A \in_{\circ} c \downarrow_{CF} \mathfrak{R}(\mathcal{O}bj) \implies$

$\tau\ c\ c\ (\mathfrak{C}(\mathcal{C}Id)(\downarrow c))(\mathcal{N}TMap)(\downarrow A) = ?UArr\ c(\mathcal{N}TMap)(\downarrow A) \circ_{A\mathfrak{A}} f'$

]] $\implies f' = f$

by *metis*

note [*symmetric, cat-cs-simps*] =

σ .*ntcf-Comp-commute*

σ' .*ntcf-Comp-commute*

show $\sigma'(\mathcal{N}TMap)(\downarrow c) = \sigma(\mathcal{N}TMap)(\downarrow c)$

proof(*rule trans-sym[where s=f]*)

show $\sigma'(\mathcal{N}TMap)(\downarrow c) = f$

proof(*rule f-unique*)

fix A **assume** $prems''$: $A \in_{\circ} c \downarrow_{CF} \mathfrak{R}(\mathcal{O}bj)$

with $prems'$ **obtain** $b' f'$

where A -*def*: $A = [\downarrow, b', f']_{\circ}$

and b' : $b' \in_{\circ} \mathfrak{B}(\mathcal{O}bj)$

and f' : $f' : c \mapsto_{\mathfrak{C}} \mathfrak{R}(\mathcal{O}bjMap)(\downarrow b')$

by *auto*

let $?Rb' = \langle \mathfrak{R}(\mathcal{O}bjMap)(\downarrow b') \rangle$

from b' **have** Rb' : $?Rb' \in_{\circ} \mathfrak{C}(\mathcal{O}bj)$

by (*cs-concl cs-shallow cs-intro: cat-cs-intros*)

interpret $lim-Rb'$: *is-cat-limit*

$\alpha \langle ?Rb' \downarrow_{CF} \mathfrak{R} \rangle \mathfrak{A} \langle \mathfrak{T} \circ_{CF} ?Rb' \circ \square_{CF} \mathfrak{R} \rangle \langle ?UObj\ ?Rb' \rangle \langle ?UArr\ ?Rb' \rangle$

by (*rule assms(\mathcal{B})[OF Rb']*)

from Rb' **have** $CId-Rb'$: $\mathfrak{C}(\mathcal{C}Id)(\downarrow ?Rb') : ?Rb' \mapsto_{\mathfrak{C}} ?Rb'$

by (*cs-concl cs-intro: cat-cs-intros*)

from $CId-Rb'$ b' **have** $a'-b'-CId-Rb'$:

$[\downarrow, b', \mathfrak{C}(\mathcal{C}Id)(\downarrow ?Rb')]_{\circ} \in_{\circ} ?Rb' \downarrow_{CF} \mathfrak{R}(\mathcal{O}bj)$

by

```

(
  cs-concl cs-shallow
  cs-simp: cat-cs-simps cat-comma-cs-simps
  cs-intro: cat-cs-intros cat-comma-cs-intros
)
from
  lim-Obj-the-cf-rKe-commute[
    where lim-Obj=lim-Obj,
    OF assms(1,2) lim-c assms(3)[OF  $\mathfrak{R}b'$ ]  $f'$   $a'-b'-CId-\mathfrak{R}b'$ 
  ]
   $f'$ 
have [cat-Kan-cs-simps]:
  ?UArr c( $\downarrow NTMap$ )( $\downarrow 0, b', f'$ )• =
  ?UArr ? $\mathfrak{R}b'$ ( $\downarrow NTMap$ )( $\downarrow 0, b', \mathfrak{C}(\downarrow CId)(\downarrow ?\mathfrak{R}b')$ )•  $\circ_{A\mathfrak{A}}$ 
  ?the-cf-rKe( $\downarrow ArrMap$ )( $\downarrow f'$ )
by (cs-prems cs-shallow cs-simp: cat-cs-simps)

from  $prems'$   $prems''$   $b'$   $f'$  show
 $\tau$   $c$   $c$  ( $\mathfrak{C}(\downarrow CId)(\downarrow c)$ )( $\downarrow NTMap$ )( $\downarrow A$ ) = ?UArr c( $\downarrow NTMap$ )( $\downarrow A$ )  $\circ_{A\mathfrak{A}}$   $\sigma'(\downarrow NTMap)(\downarrow c)$ 
  unfolding A-def
  by
    (
      cs-concl
      cs-simp:
        cat-cs-simps cat-comma-cs-simps cat-Kan-cs-simps
      cs-intro:
        cat-lim-cs-intros
        cat-cs-intros
        cat-comma-cs-intros
        cat-Kan-cs-intros
    )
qed
(
  use  $prems'$  in
  <
    cs-concl cs-shallow
    cs-simp: cat-Kan-cs-simps cs-intro: cat-cs-intros
  >
)

show  $\sigma(\downarrow NTMap)(\downarrow c) = f$ 
proof(rule f-unique)
  fix A assume  $prems''$ :  $A \in_0 c \downarrow_{CF} \mathfrak{R}(\downarrow Obj)$ 
  from this  $prems'$  obtain  $b'$   $f'$ 
  where A-def:  $A = [\downarrow 0, b', f']_0$ 
  and  $b'$ :  $b' \in_0 \mathfrak{B}(\downarrow Obj)$ 
  and  $f'$ :  $f' : c \mapsto_{\mathfrak{C}} \mathfrak{R}(\downarrow ObjMap)(\downarrow b')$ 
  by auto
  let ? $\mathfrak{R}b'$  =  $\langle \mathfrak{R}(\downarrow ObjMap)(\downarrow b') \rangle$ 
  from  $b'$  have  $\mathfrak{R}b'$ : ? $\mathfrak{R}b' \in_0 \mathfrak{C}(\downarrow Obj)$ 
  by (cs-concl cs-shallow cs-intro: cat-cs-intros)
  interpret lim- $\mathfrak{R}b'$ : is-cat-limit
   $\alpha$   $\langle ?\mathfrak{R}b' \downarrow_{CF} \mathfrak{R} \rangle \mathfrak{A} \langle \mathfrak{T} \circ_{CF} ?\mathfrak{R}b' \circ \square_{CF} \mathfrak{R} \rangle \langle ?UObj ?\mathfrak{R}b' \rangle \langle ?UArr ?\mathfrak{R}b' \rangle$ 
  by (rule assms(3)[OF  $\mathfrak{R}b'$ ])
  from  $\mathfrak{R}b'$  have  $CId-\mathfrak{R}b'$ :  $\mathfrak{C}(\downarrow CId)(\downarrow ?\mathfrak{R}b') : ?\mathfrak{R}b' \mapsto_{\mathfrak{C}} ?\mathfrak{R}b'$ 
  by (cs-concl cs-intro: cat-cs-intros)
  from  $CId-\mathfrak{R}b'$   $b'$  have  $a'-b'-CId-\mathfrak{R}b'$ :
   $[\downarrow 0, b', \mathfrak{C}(\downarrow CId)(\downarrow ?\mathfrak{R}b')]_0 \in_0 ?\mathfrak{R}b' \downarrow_{CF} \mathfrak{R}(\downarrow Obj)$ 

```

```

by
  (
    cs-concl cs-shallow
    cs-simp: cat-cs-simps cat-comma-cs-simps
    cs-intro: cat-cs-intros cat-comma-cs-intros
  )

from
  lim-Obj-the-cf-rKe-commute
  [
    where lim-Obj=lim-Obj,
    OF assms(1,2) lim-c assms(3)[OF  $\mathfrak{R}b'$ ] f' a'-b'-CId- $\mathfrak{R}b'$ 
  ]
  f'
have [cat-Kan-cs-simps]:
  ?UArr c(NTMap)(0, b', f')• =
  ?UArr (? $\mathfrak{R}b'$ )(NTMap)(0, b',  $\mathfrak{C}(CId)(? $\mathfrak{R}b'$ )$ )•  $\circ_{A\mathfrak{A}}$ 
  ?the-cf-rKe(ArrMap)(f')
by (cs-prems cs-shallow cs-simp: cat-cs-simps)
from prems' prems'' b' f' show
 $\tau c c (\mathfrak{C}(CId)(c))(NTMap)(A) = ?UArr c(NTMap)(A) \circ_{A\mathfrak{A}} \sigma(NTMap)(c)$ 
  unfolding A-def
by
  (
    cs-concl
    cs-simp:
      cat-cs-simps cat-comma-cs-simps cat-Kan-cs-simps
    cs-intro:
      cat-lim-cs-intros
      cat-cs-intros
      cat-comma-cs-intros
      cat-Kan-cs-intros
  )
qed
  (
    use prems' in
    <
      cs-concl cs-shallow
      cs-simp: cat-Kan-cs-simps cs-intro: cat-cs-intros
    >
  )
qed

qed auto

qed simp-all

qed

qed (cs-concl cs-shallow cs-intro: cat-cs-intros)+

```

qed

lemma the-ntcf-lKe-is-cat-lKe:

assumes $\mathfrak{K} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{C}$

and $\mathfrak{T} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{A}$

and $\bigwedge c. c \in_0 \mathfrak{C}(Obj) \implies \text{lim-Obj } c(UArr) :$

$\mathfrak{T} \circ_{CF} \mathfrak{K} \text{ CF} \sqcap c >_{CF.colim} \text{lim-Obj } c(UObj) : \mathfrak{K} \text{ CF} \downarrow c \mapsto_{C\alpha} \mathfrak{A}$

shows *the-ntcf-lKe* α \mathfrak{T} \mathfrak{R} *lim-Obj* :
 $\mathfrak{T} \mapsto_{CF.lKe\alpha}$ *the-cf-lKe* α \mathfrak{T} \mathfrak{R} *lim-Obj* \circ_{CF} $\mathfrak{R} : \mathfrak{B} \mapsto_C \mathfrak{C} \mapsto_C \mathfrak{A}$

proof-

interpret \mathfrak{R} : *is-functor* α \mathfrak{B} \mathfrak{C} \mathfrak{R} **by** (*rule* *assms*(1))

interpret \mathfrak{T} : *is-functor* α \mathfrak{B} \mathfrak{A} \mathfrak{T} **by** (*rule* *assms*(2))

{

fix c **assume** *prems*: $c \in_o \mathfrak{C}(\text{Obj})$

from *assms*(3) [*OF this*] **have** *lim-Obj-UArr*: *lim-Obj* $c(\text{UArr})$:
 $\mathfrak{T} \circ_{CF} \mathfrak{R} \text{ CF} \sqcap_O c >_{CF.colim}$ *lim-Obj* $c(\text{UObj})$: $\mathfrak{R} \text{ CF} \downarrow c \mapsto_{C\alpha} \mathfrak{A}$.

then interpret *lim-Obj-c*: *is-cat-colimit*
 $\alpha \langle \mathfrak{R} \text{ CF} \downarrow c \rangle \mathfrak{A} \langle \mathfrak{T} \circ_{CF} \mathfrak{R} \text{ CF} \sqcap_O c \rangle \langle \text{lim-Obj } c(\text{UObj}) \rangle \langle \text{lim-Obj } c(\text{UArr}) \rangle$

by *simp*

note *op-ua-UArr-is-cat-limit'* [
where *lim-Obj=lim-Obj*, *OF* *assms*(1,2) *prems* *lim-Obj-UArr*
]

}

note *the-ntcf-rKe-is-cat-rKe* = *the-ntcf-rKe-is-cat-rKe*
 [
OF \mathfrak{R} .*is-functor-op* \mathfrak{T} .*is-functor-op*,
unfolded cat-op-simps,
where *lim-Obj=op-ua lim-Obj* \mathfrak{R} ,
unfolded cat-op-simps,
OF this,
simplified,
folded the-cf-lKe-def the-ntcf-lKe-def
]

show *?thesis*
by
 (
rule is-cat-rKe.is-cat-lKe-op
 [
OF the-ntcf-rKe-is-cat-rKe,
unfolded cat-op-simps,
folded the-cf-lKe-def the-ntcf-lKe-def
]
)

qed

14.5 Preservation of Kan extensions

The following definitions are similar to the definitions that can be found in [14] or [8].

locale *is-cat-rKe-preserves* =
is-cat-rKe α \mathfrak{B} \mathfrak{C} \mathfrak{A} \mathfrak{R} \mathfrak{T} \mathfrak{G} ε + *is-functor* α \mathfrak{A} \mathfrak{D} \mathfrak{H}
for α \mathfrak{B} \mathfrak{C} \mathfrak{A} \mathfrak{D} \mathfrak{R} \mathfrak{T} \mathfrak{G} \mathfrak{H} ε +
assumes *cat-rKe-preserves*:
 $\mathfrak{H} \circ_{CF-NTCF} \varepsilon : (\mathfrak{H} \circ_{CF} \mathfrak{G}) \circ_{CF} \mathfrak{R} \mapsto_{CF.rKe\alpha} \mathfrak{H} \circ_{CF} \mathfrak{T} : \mathfrak{B} \mapsto_C \mathfrak{C} \mapsto_C \mathfrak{D}$

syntax *-is-cat-rKe-preserves* ::
 $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
 (
 $\langle (- : / - \circ_{CF} - \mapsto_{CF.rKe1} - : / - \mapsto_C - \mapsto_C - : - \mapsto_{C} -) \rangle$
 $[51, 51, 51, 51, 51, 51, 51, 51, 51]$ 51
)

syntax-consts *-is-cat-rKe-preserves* \equiv *is-cat-rKe-preserves*
translations $\varepsilon : \mathfrak{G} \circ_{CF} \mathfrak{R} \mapsto_{CF.rKe\alpha} \mathfrak{T} : \mathfrak{B} \mapsto_C \mathfrak{C} \mapsto_C (\mathfrak{H} : \mathfrak{A} \mapsto_{C} \mathfrak{D}) \equiv$
 $CONST$ *is-cat-rKe-preserves* α \mathfrak{B} \mathfrak{C} \mathfrak{A} \mathfrak{D} \mathfrak{R} \mathfrak{T} \mathfrak{G} \mathfrak{H} ε

locale *is-cat-lKe-preserves* =
is-cat-lKe α \mathfrak{B} \mathfrak{C} \mathfrak{A} \mathfrak{K} \mathfrak{T} \mathfrak{F} η + *is-functor* α \mathfrak{A} \mathfrak{D} \mathfrak{H}
for α \mathfrak{B} \mathfrak{C} \mathfrak{A} \mathfrak{D} \mathfrak{K} \mathfrak{T} \mathfrak{F} \mathfrak{H} η +
assumes *cat-lKe-preserves*:
 $\mathfrak{H} \circ_{CF-NTCF} \eta : \mathfrak{H} \circ_{CF} \mathfrak{T} \mapsto_{CF.lKe\alpha} (\mathfrak{H} \circ_{CF} \mathfrak{F}) \circ_{CF} \mathfrak{K} : \mathfrak{B} \mapsto_C \mathfrak{C} \mapsto_C \mathfrak{D}$

syntax *-is-cat-lKe-preserves* ::
 $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
(
 $\langle (- :/ - \mapsto_{CF.lKe\alpha} - \circ_{CF} - :/ - \mapsto_C - \mapsto_C - : - \mapsto_C -) \rangle$
 $[51, 51, 51, 51, 51, 51, 51, 51, 51, 51] 51$
)

syntax-consts *-is-cat-lKe-preserves* \Leftarrow *is-cat-lKe-preserves*
translations $\eta : \mathfrak{T} \mapsto_{CF.lKe\alpha} \mathfrak{F} \circ_{CF} \mathfrak{K} : \mathfrak{B} \mapsto_C \mathfrak{C} \mapsto_C (\mathfrak{H} : \mathfrak{A} \mapsto_C \mathfrak{D}) \Leftarrow$
CONST is-cat-lKe-preserves α \mathfrak{B} \mathfrak{C} \mathfrak{A} \mathfrak{D} \mathfrak{K} \mathfrak{T} \mathfrak{F} \mathfrak{H} η

Rules.

lemma (in *is-cat-rKe-preserves*) *is-cat-rKe-preserves-axioms'*:
assumes $\alpha' = \alpha$
and $\mathfrak{G}' = \mathfrak{G}$
and $\mathfrak{K}' = \mathfrak{K}$
and $\mathfrak{T}' = \mathfrak{T}$
and $\mathfrak{H}' = \mathfrak{H}$
and $\mathfrak{B}' = \mathfrak{B}$
and $\mathfrak{A}' = \mathfrak{A}$
and $\mathfrak{C}' = \mathfrak{C}$
and $\mathfrak{D}' = \mathfrak{D}$
shows $\varepsilon : \mathfrak{G}' \circ_{CF} \mathfrak{K}' \mapsto_{CF.rKe\alpha'} \mathfrak{T}' : \mathfrak{B}' \mapsto_C \mathfrak{C}' \mapsto_C (\mathfrak{H}' : \mathfrak{A}' \mapsto_C \mathfrak{D}')$
unfolding *assms* **by** (rule *is-cat-rKe-preserves-axioms*)

mk-ide rf *is-cat-rKe-preserves-def*[*unfolded is-cat-rKe-preserves-axioms-def*]
 $|intro\ is-cat-rKe-preservesI|$
 $|dest\ is-cat-rKe-preservesD[dest]|$
 $|elim\ is-cat-rKe-preservesE[elim]|$

lemmas [*cat-Kan-cs-intros*] = *is-cat-rKeD(1-3)*

lemma (in *is-cat-lKe-preserves*) *is-cat-lKe-preserves-axioms'*:
assumes $\alpha' = \alpha$
and $\mathfrak{F}' = \mathfrak{F}$
and $\mathfrak{K}' = \mathfrak{K}$
and $\mathfrak{T}' = \mathfrak{T}$
and $\mathfrak{H}' = \mathfrak{H}$
and $\mathfrak{B}' = \mathfrak{B}$
and $\mathfrak{A}' = \mathfrak{A}$
and $\mathfrak{C}' = \mathfrak{C}$
and $\mathfrak{D}' = \mathfrak{D}$
shows $\eta : \mathfrak{T}' \mapsto_{CF.lKe\alpha} \mathfrak{F}' \circ_{CF} \mathfrak{K}' : \mathfrak{B}' \mapsto_C \mathfrak{C}' \mapsto_C (\mathfrak{H}' : \mathfrak{A}' \mapsto_C \mathfrak{D}')$
unfolding *assms* **by** (rule *is-cat-lKe-preserves-axioms*)

mk-ide rf *is-cat-lKe-preserves-def*[*unfolded is-cat-lKe-preserves-axioms-def*]
 $|intro\ is-cat-lKe-preservesI|$
 $|dest\ is-cat-lKe-preservesD[dest]|$
 $|elim\ is-cat-lKe-preservesE[elim]|$

lemmas [*cat-Kan-cs-intros*] = *is-cat-lKe-preservesD(1-3)*

Duality.

lemma (in *is-cat-rKe-preserves*) *is-cat-rKe-preserves-op*:
op-ntcf ε :
op-cf $\mathfrak{T} \mapsto_{CF.lKe\alpha}$ *op-cf* $\mathfrak{G} \circ_{CF}$ *op-cf* \mathfrak{K} :
op-cat $\mathfrak{B} \mapsto_C$ *op-cat* $\mathfrak{C} \mapsto_C$ (*op-cf* $\mathfrak{H} : \text{op-cat } \mathfrak{A} \mapsto_C \text{op-cat } \mathfrak{D}$)
proof(intro *is-cat-lKe-preservesI*)
from *cat-rKe-preserves* **show** *op-cf* $\mathfrak{H} \circ_{CF-NTCF}$ *op-ntcf* ε :
op-cf $\mathfrak{H} \circ_{CF}$ *op-cf* $\mathfrak{T} \mapsto_{CF.lKe\alpha}$ (*op-cf* $\mathfrak{H} \circ_{CF}$ *op-cf* \mathfrak{G}) \circ_{CF} *op-cf* \mathfrak{K} :
op-cat $\mathfrak{B} \mapsto_C$ *op-cat* $\mathfrak{C} \mapsto_C$ *op-cat* \mathfrak{D}
by (*cs-concl-step op-ntcf-cf-ntcf-comp[symmetric]*)
(*cs-concl cs-shallow cs-simp: cat-op-simps cs-intro: cat-op-intros*)
qed (*cs-concl cs-shallow cs-simp: cat-op-simps cs-intro: cat-op-intros*)+

lemma (in *is-cat-rKe-preserves*) *is-cat-lKe-preserves-op'*[*cat-op-intros*]:
assumes $\mathfrak{T}' = \text{op-cf } \mathfrak{T}$
and $\mathfrak{G}' = \text{op-cf } \mathfrak{G}$
and $\mathfrak{K}' = \text{op-cf } \mathfrak{K}$
and $\mathfrak{B}' = \text{op-cat } \mathfrak{B}$
and $\mathfrak{A}' = \text{op-cat } \mathfrak{A}$
and $\mathfrak{C}' = \text{op-cat } \mathfrak{C}$
and $\mathfrak{D}' = \text{op-cat } \mathfrak{D}$
and $\mathfrak{H}' = \text{op-cf } \mathfrak{H}$
shows *op-ntcf* ε :
 $\mathfrak{T}' \mapsto_{CF.lKe\alpha}$ $\mathfrak{G}' \circ_{CF}$ $\mathfrak{K}' : \mathfrak{B}' \mapsto_C$ $\mathfrak{C}' \mapsto_C$ ($\mathfrak{H}' : \mathfrak{A}' \mapsto_C \mathfrak{D}'$)
unfolding *assms* **by** (*rule is-cat-rKe-preserves-op*)

lemmas [*cat-op-intros*] = *is-cat-rKe-preserves.is-cat-lKe-preserves-op'*

lemma (in *is-cat-lKe-preserves*) *is-cat-rKe-preserves-op*:
op-ntcf η :
op-cf $\mathfrak{F} \circ_{CF}$ *op-cf* $\mathfrak{K} \mapsto_{CF.rKe\alpha}$ *op-cf* \mathfrak{T} :
op-cat $\mathfrak{B} \mapsto_C$ *op-cat* $\mathfrak{C} \mapsto_C$ (*op-cf* $\mathfrak{H} : \text{op-cat } \mathfrak{A} \mapsto_C \text{op-cat } \mathfrak{D}$)
proof(intro *is-cat-rKe-preservesI*)
from *cat-lKe-preserves* **show** *op-cf* $\mathfrak{H} \circ_{CF-NTCF}$ *op-ntcf* η :
(*op-cf* $\mathfrak{H} \circ_{CF}$ *op-cf* \mathfrak{F}) \circ_{CF} *op-cf* $\mathfrak{K} \mapsto_{CF.rKe\alpha}$ *op-cf* $\mathfrak{H} \circ_{CF}$ *op-cf* \mathfrak{T} :
op-cat $\mathfrak{B} \mapsto_C$ *op-cat* $\mathfrak{C} \mapsto_C$ *op-cat* \mathfrak{D}
by (*cs-concl-step op-ntcf-cf-ntcf-comp[symmetric]*)
(*cs-concl cs-shallow cs-simp: cat-op-simps cs-intro: cat-op-intros*)
qed (*cs-concl cs-shallow cs-simp: cat-op-simps cs-intro: cat-op-intros*)+

lemma (in *is-cat-lKe-preserves*) *is-cat-rKe-preserves-op'*[*cat-op-intros*]:
assumes $\mathfrak{T}' = \text{op-cf } \mathfrak{T}$
and $\mathfrak{F}' = \text{op-cf } \mathfrak{F}$
and $\mathfrak{K}' = \text{op-cf } \mathfrak{K}$
and $\mathfrak{H}' = \text{op-cf } \mathfrak{H}$
and $\mathfrak{B}' = \text{op-cat } \mathfrak{B}$
and $\mathfrak{A}' = \text{op-cat } \mathfrak{A}$
and $\mathfrak{C}' = \text{op-cat } \mathfrak{C}$
and $\mathfrak{D}' = \text{op-cat } \mathfrak{D}$
shows *op-ntcf* η :
 $\mathfrak{F}' \circ_{CF}$ $\mathfrak{K}' \mapsto_{CF.rKe\alpha}$ $\mathfrak{T}' : \mathfrak{B}' \mapsto_C$ $\mathfrak{C}' \mapsto_C$ ($\mathfrak{H}' : \mathfrak{A}' \mapsto_C \mathfrak{D}'$)
unfolding *assms* **by** (*rule is-cat-rKe-preserves-op*)

14.6 All concepts are Kan extensions

Background information for this subsection is provided in Chapter X-7 in [9] and subsection 6.5 in [14]. It should be noted that only the connections between the Kan extensions, limits and adjunctions are exposed (an alternative proof of the Yoneda lemma using Kan extensions is not

provided in the context of this work).

14.6.1 Limits and colimits

lemma *cat-rKe-is-cat-limit*:

— The statement of the theorem is similar to the statement of a part of Theorem 1 in Chapter X-7 in [9] or Proposition 6.5.1 in [14].

assumes $\varepsilon : \mathfrak{G} \circ_{CF} \mathfrak{K} \mapsto_{CF.rKe\alpha} \mathfrak{T} : \mathfrak{B} \mapsto_C \text{cat-1 } \mathfrak{a} \text{ f} \mapsto_C \mathfrak{A}$

and $\mathfrak{T} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{A}$

shows $\varepsilon : \mathfrak{G}(\text{ObjMap})(\mathfrak{a}) <_{CF.lim} \mathfrak{T} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{A}$

proof–

interpret ε : *is-cat-rKe* α $\mathfrak{B} \langle \text{cat-1 } \mathfrak{a} \text{ f} \rangle \mathfrak{A} \mathfrak{K} \mathfrak{T} \mathfrak{G} \varepsilon$ **by** (*rule assms(1)*)

interpret \mathfrak{T} : *is-functor* α $\mathfrak{B} \mathfrak{A} \mathfrak{T}$ **by** (*rule assms(2)*)

from *cat-1-components(1)* **have** $\mathfrak{a} : \mathfrak{a} \in_o Vset \alpha$

by (*auto simp: \varepsilon.AG.HomCod.cat-in-Obj-in-Vset*)

from *cat-1-components(2)* **have** $\text{f} : \text{f} \in_o Vset \alpha$

by (*auto simp: \varepsilon.AG.HomCod.cat-in-Arr-in-Vset*)

have \mathfrak{K} -*def*: $\mathfrak{K} = \text{cf-const } \mathfrak{B} (\text{cat-1 } \mathfrak{a} \text{ f}) \mathfrak{a}$

by (*rule cf-const-if-HomCod-is-cat-1*)

(*cs-concl cs-shallow cs-intro: cat-cs-intros*)

have \mathfrak{G} - \mathfrak{K} -*def*: $\mathfrak{G} \circ_{CF} \mathfrak{K} = \text{cf-const } \mathfrak{B} \mathfrak{A} (\mathfrak{G}(\text{ObjMap})(\mathfrak{a}))$

by

(

cs-concl cs-shallow

cs-simp: *cat-1-components(1) \mathfrak{K}-def cat-cs-simps*

cs-intro: *V-cs-intros cat-cs-intros*

)

interpret ε : *is-ntcf* α $\mathfrak{B} \mathfrak{A} \langle \mathfrak{G} \circ_{CF} \mathfrak{K} \rangle \mathfrak{T} \varepsilon$

by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

show $\varepsilon : \mathfrak{G}(\text{ObjMap})(\mathfrak{a}) <_{CF.lim} \mathfrak{T} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{A}$

proof(*intro is-cat-limitI is-cat-coneI*)

show $\varepsilon : \text{cf-const } \mathfrak{B} \mathfrak{A} (\mathfrak{G}(\text{ObjMap})(\mathfrak{a})) \mapsto_{CF} \mathfrak{T} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{A}$

by (*rule \varepsilon.ntcf-rKe.is-ntcf-axioms[unfolded \mathfrak{G}\mathfrak{K}-def]*)

fix $u' r'$ **assume** *prems*: $u' : r' <_{CF.cone} \mathfrak{T} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{A}$

interpret u' : *is-cat-cone* α $r' \mathfrak{B} \mathfrak{A} \mathfrak{T} u'$ **by** (*rule prems*)

have \mathfrak{G} -*def*: $\mathfrak{G} = \text{cf-const } (\text{cat-1 } \mathfrak{a} \text{ f}) \mathfrak{A} (\mathfrak{G}(\text{ObjMap})(\mathfrak{a}))$

by (*rule cf-const-if-HomDom-is-cat-1[OF \varepsilon.Ran.is-functor-axioms]*)

from *prems* **have** *const-r'*: $\text{cf-const } (\text{cat-1 } \mathfrak{a} \text{ f}) \mathfrak{A} r' : \text{cat-1 } \mathfrak{a} \text{ f} \mapsto_{C\alpha} \mathfrak{A}$

by

(

cs-concl

cs-simp: *cat-cs-simps cs-intro: cat-lim-cs-intros cat-cs-intros*

)

have *cf-comp-cf-const-r-\mathfrak{K}-def*:

cf-const } (\text{cat-1 } \mathfrak{a} \text{ f}) \mathfrak{A} r' \circ_{CF} \mathfrak{K} = \text{cf-const } \mathfrak{B} \mathfrak{A} r'

by

(

cs-concl
cs-simp: *cat-cs-simps* \mathfrak{K} -def
cs-intro: *cat-cs-intros* *cat-lim-cs-intros*
)

from ε .*cat-rKe-unique*[
OF const-r', *unfolded cf-comp-cf-const-r- \mathfrak{K} -def*, *OF u'.is-ntcf-axioms*
]
obtain σ
where σ : *cf-const* (*cat-1 a f*) \mathfrak{A} $r' \mapsto_{CF} \mathfrak{G}$: *cat-1 a f* $\mapsto_{C\alpha} \mathfrak{A}$
and *u'-def*: $u' = \varepsilon \cdot_{NTCF} (\sigma \circ_{NTCF-CF} \mathfrak{K})$
and *unique- σ* : $\wedge \sigma'$.
[[
 σ' : *cf-const* (*cat-1 a f*) \mathfrak{A} $r' \mapsto_{CF} \mathfrak{G}$: *cat-1 a f* $\mapsto_{C\alpha} \mathfrak{A}$;
 $u' = \varepsilon \cdot_{NTCF} (\sigma' \circ_{NTCF-CF} \mathfrak{K})$
]] $\implies \sigma' = \sigma$
by *auto*

interpret σ : *is-ntcf* α \langle *cat-1 a f* \rangle \mathfrak{A} \langle *cf-const* (*cat-1 a f*) \mathfrak{A} r' \rangle \mathfrak{G} σ
by (*rule* σ)

show $\exists ! f'. f' : r' \mapsto_{\mathfrak{A}} \mathfrak{G} (\text{ObjMap}) (\mathfrak{a}) \wedge u' = \varepsilon \cdot_{NTCF} \text{ntcf-const } \mathfrak{B} \mathfrak{A} f'$
proof(*intro exI conjI*; (*elim conjE*)?)
fix f' **assume** *prems'*:
 $f' : r' \mapsto_{\mathfrak{A}} \mathfrak{G} (\text{ObjMap}) (\mathfrak{a})$ $u' = \varepsilon \cdot_{NTCF} \text{ntcf-const } \mathfrak{B} \mathfrak{A} f'$
from *prems'*(1) **have** *ntcf-const* (*cat-1 a f*) \mathfrak{A} f' :
cf-const (*cat-1 a f*) \mathfrak{A} $r' \mapsto_{CF} \mathfrak{G}$: *cat-1 a f* $\mapsto_{C\alpha} \mathfrak{A}$
by (*subst* \mathfrak{G} -def)
(*cs-concl* **cs-shallow** **cs-simp:** *cat-cs-simps* **cs-intro:** *cat-cs-intros*)
moreover with *prems'*(1) **have** $u' = \varepsilon \cdot_{NTCF} (\text{ntcf-const} (\text{cat-1 a f}) \mathfrak{A} f' \circ_{NTCF-CF} \mathfrak{K})$
by
(
cs-concl
cs-simp: *cat-cs-simps* *prems'*(2) \mathfrak{K} -def **cs-intro:** *cat-cs-intros*
)
ultimately have σ -def: $\sigma = \text{ntcf-const} (\text{cat-1 a f}) \mathfrak{A} f'$
by (*auto simp: unique- σ [symmetric]*)
show $f' = \sigma (\text{NTMap}) (\mathfrak{a})$
by (*cs-concl* **cs-simp:** *cat-cs-simps* σ -def **cs-intro:** *cat-cs-intros*)
qed (*cs-concl* **cs-simp:** *cat-cs-simps* *u'-def* \mathfrak{K} -def **cs-intro:** *cat-cs-intros*)⁺

qed (*cs-concl* **cs-simp:** \mathfrak{K} -def **cs-intro:** *cat-cs-intros*)

qed

lemma *cat-lKe-is-cat-colimit*:

assumes $\eta : \mathfrak{T} \mapsto_{CF.lKe\alpha} \mathfrak{F} \circ_{CF} \mathfrak{K} : \mathfrak{B} \mapsto_C \text{cat-1 a f} \mapsto_C \mathfrak{A}$
and $\mathfrak{T} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{A}$
shows $\eta : \mathfrak{T} >_{CF.colim} \mathfrak{F} (\text{ObjMap}) (\mathfrak{a}) : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{A}$
proof-
interpret η : *is-cat-lKe* α \mathfrak{B} \langle *cat-1 a f* \rangle \mathfrak{A} \mathfrak{K} \mathfrak{T} \mathfrak{F} η **by** (*rule* *assms*(1))
from *cat-1-components*(1) **have** $\mathfrak{a} : \mathfrak{a} \in_{\circ} \text{Vset } \alpha$
by (*auto simp: η .AG.HomCod.cat-in-Obj-in-Vset*)
from *cat-1-components*(2) **have** $\mathfrak{f} : \mathfrak{f} \in_{\circ} \text{Vset } \alpha$
by (*auto simp: η .AG.HomCod.cat-in-Arr-in-Vset*)
show *?thesis*
by
(

```

rule is-cat-limit.is-cat-colimit-op
[
  OF cat-rKe-is-cat-limit[
    OF
      η.is-cat-rKe-op[unfolded η.AG.cat-1-op[OF a f]]
      η.ntcf-lKe.NTDom.is-functor-op
  ],
  unfolded cat-op-simps
]
)
qed

```

lemma *cat-limit-is-rKe*:

— The statement of the theorem is similar to the statement of a part of Theorem 1 in Chapter X-7 in [9] or Proposition 6.5.1 in [14].

```

assumes ε : ℱ(ObjMap)(|a|) <CF.lim ℑ : ℑ ↦→ Cα ℒ
and ℔ : ℑ ↦→ Cα cat-1 a f
and ℱ : cat-1 a f ↦→ Cα ℒ
shows ε : ℱ ∘CF ℔ ↦CF.rKeα ℑ : ℑ ↦→ C cat-1 a f ↦→ C ℒ
proof-

```

interpret ε : *is-cat-limit* α ℑ ℒ ℑ <ℱ(ObjMap)(|a|)> ε **by** (*rule assms*)

interpret ℔ : *is-functor* α ℑ <cat-1 a f> ℔ **by** (*rule assms(2)*)

interpret ℱ : *is-functor* α <cat-1 a f> ℒ ℱ **by** (*rule assms(3)*)

show ?thesis

proof(*rule is-cat-rKeI'*)

note ℔-def = *cf-const-if-HomCod-is-cat-1*[OF *assms(2)*]

note ℱ-def = *cf-const-if-HomDom-is-cat-1*[OF *assms(3)*]

have ℱ℔-def: ℱ ∘_{CF} ℔ = *cf-const* ℑ ℒ (ℱ(ObjMap)(|a|))
by (*subst ℔-def, use nothing in <subst ℱ-def>*)
(cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)

show ε : ℱ ∘_{CF} ℔ ↦_{CF} ℑ : ℑ ↦_→ Cα ℒ

by

```

(
  cs-concl cs-shallow
  cs-simp: cat-cs-simps ℱ℔-def cs-intro: cat-cs-intros
)

```

fix ℱ' ε' **assume** *prems*:

ℱ' : *cat-1 a f* ↦_→ Cα ℒ

ε' : ℱ' ∘_{CF} ℔ ↦_{CF} ℑ : ℑ ↦_→ Cα ℒ

interpret *is-functor* α <cat-1 a f> ℒ ℱ' **by** (*rule prems(1)*)

note ℱ'-def = *cf-const-if-HomDom-is-cat-1*[OF *prems(1)*]

from *prems(2)* **have** ε':

ε' : *cf-const* ℑ ℒ (ℱ'(ObjMap)(|a|)) ↦_{CF} ℑ : ℑ ↦_→ Cα ℒ

unfolding ℱ'-def

by (*subst (asm) ℱ'-def*)

(cs-prems cs-simp: cat-cs-simps cs-intro: cat-cs-intros)

from *prems(2)* **have** ε' : ℱ'(ObjMap)(|a|) <_{CF.cone} ℑ : ℑ ↦_→ Cα ℒ

by (*intro is-cat-coneI ε'*) *(cs-concl cs-intro: cat-cs-intros)+*

from $\varepsilon.\text{cat-lim-ua-fo}[OF\ \text{this}]$ **obtain** f'
where $f': f' : \mathfrak{G}'(\text{ObjMap})(\mathfrak{a}) \mapsto_{\mathfrak{A}} \mathfrak{G}(\text{ObjMap})(\mathfrak{a})$
and $\varepsilon\text{-def}: \varepsilon' = \varepsilon \cdot_{NTCF} \text{ntcf-const } \mathfrak{B} \ \mathfrak{A} \ f'$
and $\text{unique-}f'$:

$$\llbracket$$

$$f'' : \mathfrak{G}'(\text{ObjMap})(\mathfrak{a}) \mapsto_{\mathfrak{A}} \mathfrak{G}(\text{ObjMap})(\mathfrak{a});$$

$$\varepsilon' = \varepsilon \cdot_{NTCF} \text{ntcf-const } \mathfrak{B} \ \mathfrak{A} \ f''$$

$$\rrbracket \implies f'' = f'$$

for f''
by *metis*

show $\exists! \sigma$.
 $\sigma : \mathfrak{G}' \mapsto_{CF} \mathfrak{G} : \text{cat-1 } \mathfrak{a} \ \mathfrak{f} \mapsto \mapsto_{C\alpha} \ \mathfrak{A} \wedge \varepsilon' = \varepsilon \cdot_{NTCF} (\sigma \circ_{NTCF-CF} \mathfrak{K})$
proof(*intro exII conjI; (elim conjE)?*)
from f' **show**
 $\text{ntcf-const } (\text{cat-1 } \mathfrak{a} \ \mathfrak{f}) \ \mathfrak{A} \ f' : \mathfrak{G}' \mapsto_{CF} \mathfrak{G} : \text{cat-1 } \mathfrak{a} \ \mathfrak{f} \mapsto \mapsto_{C\alpha} \ \mathfrak{A}$
by (*subst } \mathfrak{G}'\text{-def, use nothing in } \langle \text{subst } \mathfrak{G}\text{-def} \rangle*)
(cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
with f' **show** $\varepsilon' = \varepsilon \cdot_{NTCF} (\text{ntcf-const } (\text{cat-1 } \mathfrak{a} \ \mathfrak{f}) \ \mathfrak{A} \ f' \circ_{NTCF-CF} \mathfrak{K})$
by (*cs-concl cs-simp: cat-cs-simps } \varepsilon\text{-def } \mathfrak{K}\text{-def cs-intro: cat-cs-intros}*)
fix σ **assume** *prems*:
 $\sigma : \mathfrak{G}' \mapsto_{CF} \mathfrak{G} : \text{cat-1 } \mathfrak{a} \ \mathfrak{f} \mapsto \mapsto_{C\alpha} \ \mathfrak{A}$
 $\varepsilon' = \varepsilon \cdot_{NTCF} (\sigma \circ_{NTCF-CF} \mathfrak{K})$
interpret σ : *is-ntcf } \alpha \langle \text{cat-1 } \mathfrak{a} \ \mathfrak{f} \rangle \ \mathfrak{A} \ \mathfrak{G}' \ \mathfrak{G} \ \sigma **by** (*rule prems(1)*)
have $\sigma(\text{NTMap})(\mathfrak{a}) : \mathfrak{G}'(\text{ObjMap})(\mathfrak{a}) \mapsto_{\mathfrak{A}} \mathfrak{G}(\text{ObjMap})(\mathfrak{a})$
by (*cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)
moreover **have** $\varepsilon' = \varepsilon \cdot_{NTCF} \text{ntcf-const } \mathfrak{B} \ \mathfrak{A} \ (\sigma(\text{NTMap})(\mathfrak{a}))$
by
 $($
 cs-concl
 $\text{cs-simp: cat-cs-simps prems(2) } \mathfrak{K}\text{-def cs-intro: cat-cs-intros}$
 $)$
ultimately **have** $\sigma\mathfrak{a}: \sigma(\text{NTMap})(\mathfrak{a}) = f'$ **by** (*rule unique-} f'*)
show $\sigma = \text{ntcf-const } (\text{cat-1 } \mathfrak{a} \ \mathfrak{f}) \ \mathfrak{A} \ f'$
proof(*rule ntcf-eqI*)
from f' **show**
 $\text{ntcf-const } (\text{cat-1 } \mathfrak{a} \ \mathfrak{f}) \ \mathfrak{A} \ f' : \mathfrak{G}' \mapsto_{CF} \mathfrak{G} : \text{cat-1 } \mathfrak{a} \ \mathfrak{f} \mapsto \mapsto_{C\alpha} \ \mathfrak{A}$
by (*subst } \mathfrak{G}'\text{-def, use nothing in } \langle \text{subst } \mathfrak{G}\text{-def} \rangle*)
(cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
have *dom-lhs*: $\mathcal{D}_\circ (\sigma(\text{NTMap})) = \text{cat-1 } \mathfrak{a} \ \mathfrak{f}(\text{Obj})$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)
have *dom-rhs*: $\mathcal{D}_\circ (\text{ntcf-const } (\text{cat-1 } \mathfrak{a} \ \mathfrak{f}) \ \mathfrak{A} \ f'(\text{NTMap})) = \text{cat-1 } \mathfrak{a} \ \mathfrak{f}(\text{Obj})$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)
show $\sigma(\text{NTMap}) = \text{ntcf-const } (\text{cat-1 } \mathfrak{a} \ \mathfrak{f}) \ \mathfrak{A} \ f'(\text{NTMap})$
proof(*rule vsv-eqI, unfold dom-lhs dom-rhs*)
fix a **assume** *prems*: $a \in_\circ \text{cat-1 } \mathfrak{a} \ \mathfrak{f}(\text{Obj})$
then **have** $a\text{-def}: a = \mathfrak{a}$ **unfolding** *cat-1-components by simp*
from f' **show** $\sigma(\text{NTMap})(\mathfrak{a}) = \text{ntcf-const } (\text{cat-1 } \mathfrak{a} \ \mathfrak{f}) \ \mathfrak{A} \ f'(\text{NTMap})(\mathfrak{a})$
unfolding $a\text{-def } \sigma\mathfrak{a}$
by (*cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)
qed (*auto intro: cat-cs-intros*)
qed (*simp-all add: prems*)
qed*

qed (*auto simp: asms*)

qed

lemma *cat-colimit-is-lKe*:

assumes $\eta : \mathfrak{T} >_{CF.colim} \mathfrak{F}(\text{ObjMap})(\mathfrak{a}) : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{A}$
and $\mathfrak{K} : \mathfrak{B} \mapsto_{C\alpha} \text{cat-1 } \mathfrak{a} \mathfrak{f}$
and $\mathfrak{F} : \text{cat-1 } \mathfrak{a} \mathfrak{f} \mapsto_{C\alpha} \mathfrak{A}$
shows $\eta : \mathfrak{T} \mapsto_{CF.lKe\alpha} \mathfrak{F} \circ_{CF} \mathfrak{K} : \mathfrak{B} \mapsto_C \text{cat-1 } \mathfrak{a} \mathfrak{f} \mapsto_C \mathfrak{A}$

proof-

interpret η : *is-cat-colimit* α \mathfrak{B} \mathfrak{A} \mathfrak{T} $\langle \mathfrak{F}(\text{ObjMap})(\mathfrak{a}) \rangle \eta$
by (*rule assms(1)*)

interpret \mathfrak{K} : *is-functor* α \mathfrak{B} $\langle \text{cat-1 } \mathfrak{a} \mathfrak{f} \rangle \mathfrak{K}$ **by** (*rule assms(2)*)

interpret \mathfrak{F} : *is-functor* α $\langle \text{cat-1 } \mathfrak{a} \mathfrak{f} \rangle \mathfrak{A}$ \mathfrak{F} **by** (*rule assms(3)*)

from *cat-1-components(1)* **have** $\mathfrak{a} : \mathfrak{a} \in_o \text{Vset } \alpha$

by (*auto simp: \mathfrak{K}.HomCod.cat-in-Obj-in-Vset*)

from *cat-1-components(2)* **have** $\mathfrak{f} : \mathfrak{f} \in_o \text{Vset } \alpha$

by (*auto simp: \mathfrak{K}.HomCod.cat-in-Arr-in-Vset*)

have $\mathfrak{F}\mathfrak{a} : \mathfrak{F}(\text{ObjMap})(\mathfrak{a}) = \text{op-cf } \mathfrak{F}(\text{ObjMap})(\mathfrak{a})$ **unfolding** *cat-op-simps* **by** *simp*

note *cat-1-op* = $\eta.\text{cat-1-op}[OF \ \mathfrak{a} \ \mathfrak{f}]$

show *?thesis*

by

(
rule is-cat-rKe.is-cat-lKe-op
 [
OF cat-limit-is-rKe
 [
OF
 $\eta.\text{is-cat-limit-op}[\text{unfolded } \mathfrak{F}\mathfrak{a}]$
 $\mathfrak{K}.\text{is-functor-op}[\text{unfolded cat-1-op}]$
 $\mathfrak{F}.\text{is-functor-op}[\text{unfolded cat-1-op}]$
],
unfolded cat-op-simps cat-1-op
]
)

qed

14.6.2 Adjoints

lemma (*in is-cf-adjunction*) *cf-adjunction-counit-is-rKe*:

— The statement of the theorem is similar to the statement of a part of Theorem 2 in Chapter X-7 in [9] or Proposition 6.5.2 in [14]. The proof follows (approximately) the proof in [14].

shows $\varepsilon_C \ \Phi : \mathfrak{F} \circ_{CF} \mathfrak{G} \mapsto_{CF.rKe\alpha} \text{cf-id } \mathfrak{D} : \mathfrak{D} \mapsto_C \mathfrak{C} \mapsto_C \mathfrak{D}$

proof-

define β **where** $\beta = \alpha + \omega$

have $\beta : \mathcal{Z} \ \beta$ **and** $\alpha\beta : \alpha \in_o \beta$

by (*simp-all add: \beta-def \mathcal{Z}-Limit-\alpha\omega \mathcal{Z}-\omega-\alpha\omega \mathcal{Z}-def \mathcal{Z}-\alpha-\alpha\omega*)

then interpret $\beta : \mathcal{Z} \ \beta$ **by** *simp*

note *exp-adj* = *cf-adj-exp-cf-cat-exp-cf-cat*[*OF* β $\alpha\beta$ *R.category-axioms*]

let $? \eta = \langle \eta_C \ \Phi \rangle$

let $? \varepsilon = \langle \varepsilon_C \ \Phi \rangle$

let $? \mathfrak{D}\eta = \langle \text{exp-cat-ntcf } \alpha \ \mathfrak{D} \ ? \eta \rangle$

let $? \mathfrak{D}\mathfrak{F} = \langle \text{exp-cat-cf } \alpha \ \mathfrak{D} \ \mathfrak{F} \rangle$

let $? \mathfrak{D}\mathfrak{G} = \langle \text{exp-cat-cf } \alpha \ \mathfrak{D} \ \mathfrak{G} \rangle$

let $? \mathfrak{D}\mathfrak{D} = \langle \text{cat-FUNCT } \alpha \ \mathfrak{D} \ \mathfrak{D} \rangle$

let $? \mathfrak{C}\mathfrak{D} = \langle \text{cat-FUNCT } \alpha \ \mathfrak{C} \ \mathfrak{D} \rangle$

let $? \text{adj-}\mathfrak{D}\eta = \langle \text{cf-adjunction-of-unit } \beta \ ? \mathfrak{D}\mathfrak{G} \ ? \mathfrak{D}\mathfrak{F} \ ? \mathfrak{D}\eta \rangle$

interpret $\mathfrak{D}\eta$: *is-cf-adjunction* β $? \mathfrak{C}\mathfrak{D}$ $? \mathfrak{D}\mathfrak{D}$ $? \mathfrak{D}\mathfrak{G}$ $? \mathfrak{D}\mathfrak{F}$ *adj-}\mathfrak{D}\eta **by** (*rule exp-adj*)*

show *?thesis*
proof(*intro is-cat-rKeI*)
have *id- \mathcal{D}* : *cf-map (cf-id \mathcal{D}) \in_0 cat-FUNCT α \mathcal{D} \mathcal{D} (*Obj*)*
by
(

cs-concl
cs-simp: *cat-FUNCT-components(1)*
cs-intro: *cat-cs-intros cat-FUNCT-cs-intros*
)

then have *exp-id- \mathcal{D}* :
*exp-cat-cf α \mathcal{D} \mathfrak{F} (*ObjMap*)(*cf-map (cf-id \mathcal{D})*) = *cf-map \mathfrak{F}**
by
(

cs-concl
cs-simp: *cat-cs-simps cat-FUNCT-cs-simps* **cs-intro**: *cat-cs-intros*
)

have *\mathfrak{F}* : *cf-map \mathfrak{F} \in_0 cat-FUNCT α \mathfrak{C} \mathcal{D} (*Obj*)*
by
(

cs-concl **cs-shallow**
cs-simp: *cat-FUNCT-components(1)*
cs-intro: *cat-cs-intros cat-FUNCT-cs-intros*
)

have *ε* : *ntcf-arrow (ε_C Φ) \in_0 ntcf-arrows α \mathcal{D} \mathcal{D}*
by (*cs-concl* **cs-intro**: *cat-FUNCT-cs-intros adj-cs-intros*)
have *$\mathcal{D}\mathcal{D}$* : *category β (cat-FUNCT α \mathcal{D} \mathcal{D})*
by (*cs-concl* **cs-shallow** **cs-intro**: *cat-cs-intros*)
have *$\mathfrak{C}\mathcal{D}$* : *category β (cat-FUNCT α \mathfrak{C} \mathcal{D})*
by (*cs-concl* **cs-shallow** **cs-intro**: *cat-cs-intros*)

from
 ε \mathfrak{F} $\alpha\beta$ id- \mathcal{D}
 $\mathcal{D}\mathcal{D}$ $\mathfrak{C}\mathcal{D}$ LR.is-functor-axioms RL.is-functor-axioms R.cat-cf-id-is-functor
NT.is-iso-ntcf-axioms
have *ε -id- \mathcal{D}* : *ε_C ?adj- $\mathcal{D}\eta$ (*NTMap*)(*cf-map (cf-id \mathcal{D})*) = *ntcf-arrow ? ε**
by
(

cs-concl
cs-simp:
cat-Set-the-inverse[symmetric]
cat-op-simps
cat-cs-simps
cat-FUNCT-cs-simps
adj-cs-simps
cs-intro:
 $\mathcal{D}\eta$.NT.iso-ntcf-is-iso-arr''
cat-op-intros
adj-cs-intros
cat-cs-intros
cat-FUNCT-cs-intros
cat-prod-cs-intros
)

show *universal-arrow-fo ? $\mathcal{D}\mathfrak{G}$ (cf-map (cf-id \mathcal{D})) (cf-map \mathfrak{F}) (ntcf-arrow ? ε)*
by
(

rule is-cf-adjunction.cf-adjunction-counit-component-is-ua-fo[
OF exp-adj id- \mathcal{D} , unfolded exp-id- \mathcal{D} ε -id- \mathcal{D}
)

$\quad]$
 $\quad)$
qed (*cs-concl cs-intro: cat-cs-intros adj-cs-intros*)+

qed

lemma (*in is-cf-adjunction*) *cf-adjunction-unit-is-lKe*:
shows $\eta_C \Phi : \text{cf-id } \mathcal{C} \mapsto_{CF.lKe\alpha} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathcal{C} \mapsto_C \mathcal{D} \mapsto_C \mathcal{C}$
by
 $($
rule is-cat-rKe.is-cat-lKe-op
 $[$
OF is-cf-adjunction.cf-adjunction-counit-is-rKe
 $[$
OF is-cf-adjunction-op,
folded op-ntcf-cf-adjunction-unit op-cf-cf-id
 $],$
unfolded
cat-op-simps ntcf-op-ntcf-op-ntcf[OF cf-adjunction-unit-is-ntcf]
 $]$
 $)$

lemma *cf-adjunction-if-lKe-preserves*:

— The statement of the theorem is similar to the statement of a part of Theorem 2 in Chapter X-7 in [9] or Proposition 6.5.2 in [14].

assumes $\eta : \text{cf-id } \mathcal{D} \mapsto_{CF.lKe\alpha} \mathfrak{F} \circ_{CF} \mathfrak{G} : \mathcal{D} \mapsto_C \mathcal{C} \mapsto_C (\mathfrak{G} : \mathcal{D} \mapsto_C \mathcal{C})$
shows *cf-adjunction-of-unit* $\alpha \mathfrak{G} \mathfrak{F} \eta : \mathfrak{G} \Rightarrow_{CF} \mathfrak{F} : \mathcal{D} \Rightarrow_{C\alpha} \mathcal{C}$

proof–

interpret $\eta : \text{is-cat-lKe-preserves } \alpha \mathcal{D} \mathcal{C} \mathcal{D} \mathcal{C} \mathfrak{G} \langle \text{cf-id } \mathcal{D} \rangle \mathfrak{F} \mathfrak{G} \eta$
by (*rule assms*)

from $\eta.\text{cat-lKe-preserves}$ **interpret** $\mathfrak{G}\eta$:
is-cat-lKe $\alpha \mathcal{D} \mathcal{C} \mathfrak{G} \mathfrak{G} \langle \mathfrak{G} \circ_{CF} \mathfrak{F} \rangle \langle \mathfrak{G} \circ_{CF-NTCF} \eta \rangle$
by (*cs-prems cs-shallow cs-simp: cat-cs-simps*)

from
 $\mathfrak{G}\eta.\text{cat-lKe-unique}$
 $[$
OF $\eta.AG.HomCod.\text{cat-cf-id-is-functor},$
unfolded $\eta.AG.\text{cf-cf-comp-cf-id-left},$
OF $\eta.AG.\text{cf-ntcf-id-is-ntcf}$
 $]$

obtain ε **where** $\varepsilon : \mathfrak{G} \circ_{CF} \mathfrak{F} \mapsto_{CF} \text{cf-id } \mathcal{C} : \mathcal{C} \mapsto_C \alpha \mathcal{C}$
and *ntcf-id-G-def*: $\text{ntcf-id } \mathfrak{G} = \varepsilon \circ_{NTCF-CF} \mathfrak{G} \cdot_{NTCF} (\mathfrak{G} \circ_{CF-NTCF} \eta)$
by *metis*

interpret $\varepsilon : \text{is-ntcf } \alpha \mathcal{C} \mathcal{C} \langle \mathfrak{G} \circ_{CF} \mathfrak{F} \rangle \langle \text{cf-id } \mathcal{C} \rangle \varepsilon$ **by** (*rule* ε)

show *?thesis*

proof(*rule counit-unit-is-cf-adjunction*)

show [*cat-cs-simps*]: $\varepsilon \circ_{NTCF-CF} \mathfrak{G} \cdot_{NTCF} (\mathfrak{G} \circ_{CF-NTCF} \eta) = \text{ntcf-id } \mathfrak{G}$
by (*rule ntcf-id-G-def[symmetric]*)

have $\eta\text{-def}$: $\eta = (\text{ntcf-id } \mathfrak{F} \circ_{NTCF-CF} \mathfrak{G}) \cdot_{NTCF} \eta$
by

$($
cs-concl cs-shallow

cs-simp: *cat-cs-simps ntcf-id-cf-comp[symmetric]*
cs-intro: *cat-cs-intros*
)

note [*cat-cs-simps*] = *this[symmetric]*

let $?F\varepsilon\mathfrak{G} = \langle F \circ_{CF-NTCF} \varepsilon \circ_{NTCF-CF} \mathfrak{G} \rangle$
let $?ηF\mathfrak{G} = \langle η \circ_{NTCF-CF} F \circ_{NTCF-CF} \mathfrak{G} \rangle$
let $?F\mathfrak{G}η = \langle F \circ_{CF} \mathfrak{G} \circ_{CF-NTCF} η \rangle$

have $(?F\varepsilon\mathfrak{G} \cdot_{NTCF} ?ηF\mathfrak{G}) \cdot_{NTCF} η = (?F\varepsilon\mathfrak{G} \cdot_{NTCF} ?F\mathfrak{G}η) \cdot_{NTCF} η$
proof(*rule ntcf-eqI*)

have *dom-lhs*: $\mathcal{D}_\circ (((?F\varepsilon\mathfrak{G} \cdot_{NTCF} ?ηF\mathfrak{G}) \cdot_{NTCF} η)(NTMap)) = \mathcal{D}(\mathcal{O}bj)$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)
have *dom-rhs*: $\mathcal{D}_\circ (((?F\varepsilon\mathfrak{G} \cdot_{NTCF} ?F\mathfrak{G}η) \cdot_{NTCF} η)(NTMap)) = \mathcal{D}(\mathcal{O}bj)$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)
note *is-ntcf.ntcf-Comp-commute[cat-cs-simps del]*
note *category.cat-Comp-assoc[cat-cs-simps del]*
show

$((?F\varepsilon\mathfrak{G} \cdot_{NTCF} ?ηF\mathfrak{G}) \cdot_{NTCF} η)(NTMap) =$
 $((?F\varepsilon\mathfrak{G} \cdot_{NTCF} ?F\mathfrak{G}η) \cdot_{NTCF} η)(NTMap)$

proof(*rule vsv-eqI, unfold dom-lhs dom-rhs*)
fix *a* **assume** $a \in_\circ \mathcal{D}(\mathcal{O}bj)$
then show

$((?F\varepsilon\mathfrak{G} \cdot_{NTCF} ?ηF\mathfrak{G}) \cdot_{NTCF} η)(NTMap)(a) =$
 $((?F\varepsilon\mathfrak{G} \cdot_{NTCF} ?F\mathfrak{G}η) \cdot_{NTCF} η)(NTMap)(a)$

by
(

cs-concl
cs-simp: *cat-cs-simps η.ntcf-lKe.ntcf-Comp-commute[symmetric]*
cs-intro: *cat-cs-intros*

)

qed (*cs-concl cs-intro: cat-cs-intros*)+

qed (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)+
also have $\dots = (ntcf-id F \circ_{NTCF-CF} \mathfrak{G}) \cdot_{NTCF} η$
by
(

cs-concl cs-shallow
cs-simp:
cat-cs-simps
cf-comp-cf-ntcf-comp-assoc
cf-ntcf-comp-ntcf-cf-comp-assoc
cf-ntcf-comp-ntcf-vcomp[symmetric]
cs-intro: *cat-cs-intros*

)

also have $\dots = η$ **by** (*cs-concl cs-simp: cat-cs-simps*)
finally have $(?F\varepsilon\mathfrak{G} \cdot_{NTCF} ?ηF\mathfrak{G}) \cdot_{NTCF} η = η$ **by** *simp*
then have $η-def': η = (F \circ_{CF-NTCF} \varepsilon \cdot_{NTCF} (η \circ_{NTCF-CF} F) \circ_{NTCF-CF} \mathfrak{G}) \cdot_{NTCF} η$
by
(

cs-concl cs-shallow
cs-simp: *cat-cs-simps ntcf-vcomp-ntcf-cf-comp[symmetric]*
cs-intro: *cat-cs-intros*

)+

have $F\varepsilonηF: F \circ_{CF-NTCF} \varepsilon \cdot_{NTCF} (η \circ_{NTCF-CF} F) : F \mapsto_{CF} F : \mathcal{C} \mapsto_{C\alpha} \mathcal{D}$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

from $η.cat-lKe-unique[OF η.Lan.is-functor-axioms η.ntcf-lKe.is-ntcf-axioms]$

obtain σ **where**

$\llbracket \sigma' : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{C} \mapsto_{C\alpha} \mathfrak{D}; \eta = \sigma' \circ_{NTCF-CF} \mathfrak{G} \cdot_{NTCF} \eta \rrbracket \implies \sigma' = \sigma$
for σ'
by *metis*

from *this*[*OF* η .*Lan.cf-ntcf-id-is-ntcf* η -*def*] *this*[*OF* $\mathfrak{F}\varepsilon\eta\mathfrak{F}$ η -*def*'] **show**

$\mathfrak{F} \circ_{CF-NTCF} \varepsilon \cdot_{NTCF} (\eta \circ_{NTCF-CF} \mathfrak{F}) = \text{ntcf-id } \mathfrak{F}$
by *simp*

qed (*cs-concl cs-intro: cat-cs-intros*)⁺

qed

lemma *cf-adjunction-if-rKe-preserves*:

assumes $\varepsilon : \mathfrak{F} \circ_{CF} \mathfrak{G} \mapsto_{CF.rKe\alpha} \text{cf-id } \mathfrak{D} : \mathfrak{D} \mapsto_C \mathfrak{C} \mapsto_C (\mathfrak{G} : \mathfrak{D} \mapsto_{C} \mathfrak{C})$

shows *cf-adjunction-of-counit* $\alpha \mathfrak{F} \mathfrak{G} \varepsilon : \mathfrak{F} \rightleftharpoons_{CF} \mathfrak{G} : \mathfrak{C} \rightleftharpoons_{C\alpha} \mathfrak{D}$

proof-

interpret ε : *is-cat-rKe-preserves* $\alpha \mathfrak{D} \mathfrak{C} \mathfrak{D} \mathfrak{C} \mathfrak{G} \langle \text{cf-id } \mathfrak{D} \rangle \mathfrak{F} \mathfrak{G} \varepsilon$

by (*rule assms*)

have *op-cf* (*cf-id* \mathfrak{D}) = *cf-id* (*op-cat* \mathfrak{D}) **unfolding** *cat-op-simps* **by** *simp*

show *?thesis*

by

(
 rule is-cf-adjunction.is-cf-adjunction-op
 [
 OF cf-adjunction-if-lKe-preserves[
 OF ε .*is-cat-rKe-preserves-op*[*unfolded op-cf-cf-id*]
],
 folded cf-adjunction-of-counit-def,
 unfolded cat-op-simps
]
)

qed

15 Pointwise Kan extensions

15.1 Pointwise Kan extensions

The following subsection is based on elements of the content of section 6.3 in [14] and Chapter X-5 in [9].

locale $is-cat-pw-rKe = is-cat-rKe \alpha \mathfrak{B} \mathfrak{C} \mathfrak{A} \mathfrak{R} \mathfrak{T} \mathfrak{G} \varepsilon$
for $\alpha \mathfrak{B} \mathfrak{C} \mathfrak{A} \mathfrak{R} \mathfrak{T} \mathfrak{G} \varepsilon +$
assumes $cat-pw-rKe\text{-preserved}: a \in_{\circ} \mathfrak{A}(\text{Obj}) \implies$
 $\varepsilon :$
 $\mathfrak{G} \circ_{CF} \mathfrak{R} \mapsto_{CF.rKe\alpha} \mathfrak{T} :$
 $\mathfrak{B} \mapsto_C \mathfrak{C} \mapsto_C (Hom_{O.C\alpha} \mathfrak{A}(a, -) : \mathfrak{A} \mapsto \mapsto_C cat\text{-Set } \alpha)$

syntax $-is-cat-pw-rKe :: V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
 $($
 $\langle (- : / - \circ_{CF} - \mapsto_{CF.rKe.pw1} - : / - \mapsto_C - \mapsto_C -) \rangle$
 $[51, 51, 51, 51, 51, 51, 51] 51$
 $)$

syntax-consts $-is-cat-pw-rKe \rightleftharpoons is-cat-pw-rKe$
translations $\varepsilon : \mathfrak{G} \circ_{CF} \mathfrak{R} \mapsto_{CF.rKe.pw\alpha} \mathfrak{T} : \mathfrak{B} \mapsto_C \mathfrak{C} \mapsto_C \mathfrak{A} \rightleftharpoons$
 $CONST is-cat-pw-rKe \alpha \mathfrak{B} \mathfrak{C} \mathfrak{A} \mathfrak{R} \mathfrak{T} \mathfrak{G} \varepsilon$

locale $is-cat-pw-lKe = is-cat-lKe \alpha \mathfrak{B} \mathfrak{C} \mathfrak{A} \mathfrak{R} \mathfrak{T} \mathfrak{F} \eta$
for $\alpha \mathfrak{B} \mathfrak{C} \mathfrak{A} \mathfrak{R} \mathfrak{T} \mathfrak{F} \eta +$
assumes $cat-pw-lKe\text{-preserved}: a \in_{\circ} op-cat \mathfrak{A}(\text{Obj}) \implies$
 $op-ntcf \eta :$
 $op-cf \mathfrak{F} \circ_{CF} op-cf \mathfrak{R} \mapsto_{CF.rKe\alpha} op-cf \mathfrak{T} :$
 $op-cat \mathfrak{B} \mapsto_C op-cat \mathfrak{C} \mapsto_C (Hom_{O.C\alpha} \mathfrak{A}(-, a) : op-cat \mathfrak{A} \mapsto \mapsto_C cat\text{-Set } \alpha)$

syntax $-is-cat-pw-lKe :: V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
 $($
 $\langle (- : / - \mapsto_{CF.lKe.pw1} - \circ_{CF} - : / - \mapsto_C - \mapsto_C -) \rangle$
 $[51, 51, 51, 51, 51, 51, 51] 51$
 $)$

syntax-consts $-is-cat-pw-lKe \rightleftharpoons is-cat-pw-lKe$
translations $\eta : \mathfrak{T} \mapsto_{CF.lKe.pw\alpha} \mathfrak{F} \circ_{CF} \mathfrak{R} : \mathfrak{B} \mapsto_C \mathfrak{C} \mapsto_C \mathfrak{A} \rightleftharpoons$
 $CONST is-cat-pw-lKe \alpha \mathfrak{B} \mathfrak{C} \mathfrak{A} \mathfrak{R} \mathfrak{T} \mathfrak{F} \eta$

lemma (in $is-cat-pw-rKe$) $cat-pw-rKe\text{-preserved}'[cat-Kan-cs-intros]$:
assumes $a \in_{\circ} \mathfrak{A}(\text{Obj})$
and $\mathfrak{A}' = \mathfrak{A}$
and $\mathfrak{H}' = Hom_{O.C\alpha} \mathfrak{A}(a, -)$
and $\mathfrak{C}' = cat\text{-Set } \alpha$
shows $\varepsilon : \mathfrak{G} \circ_{CF} \mathfrak{R} \mapsto_{CF.rKe\alpha} \mathfrak{T} : \mathfrak{B} \mapsto_C \mathfrak{C} \mapsto_C (\mathfrak{H}' : \mathfrak{A}' \mapsto \mapsto_C \mathfrak{C}')$
using $assms(1)$ **unfolding** $assms(2-4)$ **by** $(rule\ cat-pw-rKe\text{-preserved})$

lemmas $[cat-Kan-cs-intros] = is-cat-pw-rKe.cat-pw-rKe\text{-preserved}'$

lemma (in $is-cat-pw-lKe$) $cat-pw-lKe\text{-preserved}'[cat-Kan-cs-intros]$:
assumes $a \in_{\circ} op-cat \mathfrak{A}(\text{Obj})$
and $\mathfrak{F}' = op-cf \mathfrak{F}$
and $\mathfrak{R}' = op-cf \mathfrak{R}$
and $\mathfrak{T}' = op-cf \mathfrak{T}$
and $\mathfrak{B}' = op-cat \mathfrak{B}$
and $\mathfrak{C}' = op-cat \mathfrak{C}$
and $\mathfrak{A}' = op-cat \mathfrak{A}$
and $\mathfrak{H}' = Hom_{O.C\alpha} \mathfrak{A}(-, a)$
and $\mathfrak{C}' = cat\text{-Set } \alpha$

shows *op-ntcf* η :

$\mathfrak{F}' \circ_{CF} \mathfrak{R}' \mapsto_{CF.rKe\alpha} \mathfrak{T}' : \mathfrak{B}' \mapsto_C \mathfrak{C}' \mapsto_C (\mathfrak{H}' : \mathfrak{A}' \mapsto_C \mathfrak{E}')$

using *assms*(1) **unfolding** *assms* by (rule *cat-pw-lKe-preserved*)

lemmas [*cat-Kan-cs-intros*] = *is-cat-pw-lKe.cat-pw-lKe-preserved'*

Rules.

lemma (in *is-cat-pw-rKe*) *is-cat-pw-rKe-axioms'*[*cat-Kan-cs-intros*]:

assumes $\alpha' = \alpha$

and $\mathfrak{G}' = \mathfrak{G}$

and $\mathfrak{R}' = \mathfrak{R}$

and $\mathfrak{T}' = \mathfrak{T}$

and $\mathfrak{B}' = \mathfrak{B}$

and $\mathfrak{A}' = \mathfrak{A}$

and $\mathfrak{C}' = \mathfrak{C}$

shows $\varepsilon : \mathfrak{G}' \circ_{CF} \mathfrak{R}' \mapsto_{CF.rKe.pw\alpha'} \mathfrak{T}' : \mathfrak{B}' \mapsto_C \mathfrak{C}' \mapsto_C \mathfrak{A}'$

unfolding *assms* by (rule *is-cat-pw-rKe-axioms*)

mk-ide rf *is-cat-pw-rKe-def*[*unfolded is-cat-pw-rKe-axioms-def*]

|*intro is-cat-pw-rKeI*|

|*dest is-cat-pw-rKeD*[*dest*]|

|*elim is-cat-pw-rKeE*[*elim*]|

lemmas [*cat-Kan-cs-intros*] = *is-cat-pw-rKeD*(1)

lemma (in *is-cat-pw-lKe*) *is-cat-pw-lKe-axioms'*[*cat-Kan-cs-intros*]:

assumes $\alpha' = \alpha$

and $\mathfrak{F}' = \mathfrak{F}$

and $\mathfrak{R}' = \mathfrak{R}$

and $\mathfrak{T}' = \mathfrak{T}$

and $\mathfrak{B}' = \mathfrak{B}$

and $\mathfrak{A}' = \mathfrak{A}$

and $\mathfrak{C}' = \mathfrak{C}$

shows $\eta : \mathfrak{T}' \mapsto_{CF.lKe.pw\alpha'} \mathfrak{F}' \circ_{CF} \mathfrak{R}' : \mathfrak{B}' \mapsto_C \mathfrak{C}' \mapsto_C \mathfrak{A}'$

unfolding *assms* by (rule *is-cat-pw-lKe-axioms*)

mk-ide rf *is-cat-pw-lKe-def*[*unfolded is-cat-pw-lKe-axioms-def*]

|*intro is-cat-pw-lKeI*|

|*dest is-cat-pw-lKeD*[*dest*]|

|*elim is-cat-pw-lKeE*[*elim*]|

lemmas [*cat-Kan-cs-intros*] = *is-cat-pw-lKeD*(1)

Duality.

lemma (in *is-cat-pw-rKe*) *is-cat-pw-lKe-op*:

op-ntcf ε :

op-cf $\mathfrak{T} \mapsto_{CF.lKe.pw\alpha} \text{op-cf } \mathfrak{G} \circ_{CF} \text{op-cf } \mathfrak{R} :$

op-cat $\mathfrak{B} \mapsto_C \text{op-cat } \mathfrak{C} \mapsto_C \text{op-cat } \mathfrak{A}$

proof(*intro is-cat-pw-lKeI*, *unfold cat-op-simps*)

fix *a* **assume** *prems*: $a \in_o \mathfrak{A}(\text{Obj})$

from *cat-pw-rKe-preserved*[*OF prems*] *prems* **show**

ε :

$\mathfrak{G} \circ_{CF} \mathfrak{R} \mapsto_{CF.rKe\alpha} \mathfrak{T} :$

$\mathfrak{B} \mapsto_C \mathfrak{C} \mapsto_C (\text{Hom}_{O.C\alpha} \text{op-cat } \mathfrak{A}(-, a) : \mathfrak{A} \mapsto_C \text{cat-Set } \alpha)$

by (*cs-concl cs-shallow cs-simp*: *cat-op-simps* **cs-intro**: *cat-cs-intros*)

qed (*cs-concl cs-shallow cs-intro*: *cat-op-intros*)

lemma (in *is-cat-pw-rKe*) *is-cat-pw-lKe-op'*[*cat-op-intros*]:

assumes $\mathfrak{T}' = \text{op-cf } \mathfrak{T}$
and $\mathfrak{G}' = \text{op-cf } \mathfrak{G}$
and $\mathfrak{K}' = \text{op-cf } \mathfrak{K}$
and $\mathfrak{B}' = \text{op-cat } \mathfrak{B}$
and $\mathfrak{A}' = \text{op-cat } \mathfrak{A}$
and $\mathfrak{C}' = \text{op-cat } \mathfrak{C}$
shows $\text{op-ntcf } \varepsilon : \mathfrak{T}' \mapsto_{CF.lKe.pw\alpha} \mathfrak{G}' \circ_{CF} \mathfrak{K}' : \mathfrak{B}' \mapsto_C \mathfrak{C}' \mapsto_C \mathfrak{A}'$
unfolding *assms* **by** (*rule is-cat-pw-lKe-op*)

lemmas [*cat-op-intros*] = *is-cat-pw-rKe.is-cat-pw-lKe-op'*

lemma (**in** *is-cat-pw-lKe*) *is-cat-pw-rKe-op*:

$\text{op-ntcf } \eta :$
 $\text{op-cf } \mathfrak{F} \circ_{CF} \text{op-cf } \mathfrak{K} \mapsto_{CF.rKe.pw\alpha} \text{op-cf } \mathfrak{T} :$
 $\text{op-cat } \mathfrak{B} \mapsto_C \text{op-cat } \mathfrak{C} \mapsto_C \text{op-cat } \mathfrak{A}$

proof(*intro is-cat-pw-rKeI, unfold cat-op-simps*)

fix *a* **assume** *prems*: $a \in_{\circ} \mathfrak{A}(\text{Obj})$

from *cat-pw-lKe-preserved*[*unfolded cat-op-simps, OF prems*] *prems* **show**

$\text{op-ntcf } \eta :$
 $\text{op-cf } \mathfrak{F} \circ_{CF} \text{op-cf } \mathfrak{K} \mapsto_{CF.rKe\alpha} \text{op-cf } \mathfrak{T} :$
 $\text{op-cat } \mathfrak{B} \mapsto_C \text{op-cat } \mathfrak{C} \mapsto_C$
 $(\text{Hom}_{O.C\alpha} \text{op-cat } \mathfrak{A}(a, -) : \text{op-cat } \mathfrak{A} \mapsto_C \text{cat-Set } \alpha)$

by (*cs-concl cs-shallow cs-simp: cat-op-simps cs-intro: cat-cs-intros*)

qed (*cs-concl cs-shallow cs-intro: cat-op-intros*)

lemma (**in** *is-cat-pw-lKe*) *is-cat-pw-lKe-op'*[*cat-op-intros*]:

assumes $\mathfrak{T}' = \text{op-cf } \mathfrak{T}$
and $\mathfrak{F}' = \text{op-cf } \mathfrak{F}$
and $\mathfrak{K}' = \text{op-cf } \mathfrak{K}$
and $\mathfrak{B}' = \text{op-cat } \mathfrak{B}$
and $\mathfrak{A}' = \text{op-cat } \mathfrak{A}$
and $\mathfrak{C}' = \text{op-cat } \mathfrak{C}$
shows $\text{op-ntcf } \eta : \mathfrak{F}' \circ_{CF} \mathfrak{K}' \mapsto_{CF.rKe.pw\alpha} \mathfrak{T}' : \mathfrak{B}' \mapsto_C \mathfrak{C}' \mapsto_C \mathfrak{A}'$
unfolding *assms* **by** (*rule is-cat-pw-rKe-op*)

lemmas [*cat-op-intros*] = *is-cat-pw-lKe.is-cat-pw-lKe-op'*

15.2 Lemma X.5: L-10-5-N

This subsection and several further subsections (15.2-15.8) expose definitions that are used in the proof of the technical lemma that was used in the proof of Theorem 3 from Chapter X-5 in [9].

definition *L-10-5-N* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$

where *L-10-5-N* $\alpha \beta \mathfrak{T} \mathfrak{K} c =$

$[$
 $($
 $\lambda a \in_{\circ} \mathfrak{T}(\text{HomCod})(\text{Obj}).$
 $\text{cf-nt } \alpha \beta \mathfrak{K}(\text{ObjMap})(\text{cf-map } (\text{Hom}_{O.C\alpha} \mathfrak{T}(\text{HomCod})(a, -) \circ_{CF} \mathfrak{T}), c).$
 $),$
 $($
 $\lambda f \in_{\circ} \mathfrak{T}(\text{HomCod})(\text{Arr}).$
 $\text{cf-nt } \alpha \beta \mathfrak{K}(\text{ArrMap})($
 $\text{ntcf-arrow } (\text{Hom}_{A.C\alpha} \mathfrak{T}(\text{HomCod})(f, -) \circ_{NTCF-CF} \mathfrak{T}), \mathfrak{K}(\text{HomCod})(\text{CIId})(c)$
 $).$
 $),$
 $\text{op-cat } (\mathfrak{T}(\text{HomCod})),$
 $\text{cat-Set } \beta$

]

Components.

lemma *L-10-5-N-components*:

shows *L-10-5-N* α β \mathfrak{I} \mathfrak{K} $c(\text{ObjMap}) =$

(
 $\lambda a \in_{\circ} \mathfrak{I}(\text{HomCod})(\text{Obj})$.
 $cf\text{-nt } \alpha \beta \mathfrak{K}(\text{ObjMap})(cf\text{-map } (\text{Hom}_{O.C\alpha} \mathfrak{I}(\text{HomCod}))(a, -) \circ_{CF} \mathfrak{I}), c)$.
)

and *L-10-5-N* α β \mathfrak{I} \mathfrak{K} $c(\text{ArrMap}) =$

(
 $\lambda f \in_{\circ} \mathfrak{I}(\text{HomCod})(\text{Arr})$.
 $cf\text{-nt } \alpha \beta \mathfrak{K}(\text{ArrMap})($
 $ntcf\text{-arrow } (\text{Hom}_{A.C\alpha} \mathfrak{I}(\text{HomCod}))(f, -) \circ_{NTCF-CF} \mathfrak{I}, \mathfrak{K}(\text{HomCod})(\text{CIId})(c)$
 $)$.
)

and *L-10-5-N* α β \mathfrak{I} \mathfrak{K} $c(\text{HomDom}) = op\text{-cat } (\mathfrak{I}(\text{HomCod}))$

and *L-10-5-N* α β \mathfrak{I} \mathfrak{K} $c(\text{HomCod}) = cat\text{-Set } \beta$

unfolding *L-10-5-N-def dghm-field-simps* **by** (*simp-all add: nat-omega-simps*)

context

fixes α \mathfrak{B} \mathfrak{C} \mathfrak{A} \mathfrak{K} \mathfrak{I}

assumes $\mathfrak{K}: \mathfrak{K} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{C}$

and $\mathfrak{I}: \mathfrak{I} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{A}$

begin

interpretation \mathfrak{K} : *is-functor* α \mathfrak{B} \mathfrak{C} \mathfrak{K} **by** (*rule* \mathfrak{K})

interpretation \mathfrak{I} : *is-functor* α \mathfrak{B} \mathfrak{A} \mathfrak{I} **by** (*rule* \mathfrak{I})

lemmas *L-10-5-N-components'* = *L-10-5-N-components*[
 where $\mathfrak{I}=\mathfrak{I}$ **and** $\mathfrak{K}=\mathfrak{K}$, *unfolded cat-cs-simps*
]

lemmas [*cat-Kan-cs-simps*] = *L-10-5-N-components'*(3,4)

end

15.2.1 Object map

mk-VLambda *L-10-5-N-components*(1)

[*vsv L-10-5-N-ObjMap-vsuv[cat-Kan-cs-intros]*]

context

fixes α \mathfrak{B} \mathfrak{C} \mathfrak{A} \mathfrak{K} \mathfrak{I} c

assumes $\mathfrak{K}: \mathfrak{K} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{C}$

and $\mathfrak{I}: \mathfrak{I} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{A}$

begin

mk-VLambda *L-10-5-N-components'*(1)[*OF* \mathfrak{K} \mathfrak{I}]

[*vdomain L-10-5-N-ObjMap-vdomain[cat-Kan-cs-simps]*]

[*app L-10-5-N-ObjMap-app[cat-Kan-cs-simps]*]

end

15.2.2 Arrow map

mk-VLambda *L-10-5-N-components*(2)

[*vsv L-10-5-N-ArrMap-vsuv[cat-Kan-cs-intros]*]

```

context
  fixes  $\alpha \mathfrak{B} \mathfrak{C} \mathfrak{A} \mathfrak{R} \mathfrak{T} c$ 
  assumes  $\mathfrak{R}: \mathfrak{R} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{C}$ 
    and  $\mathfrak{T}: \mathfrak{T} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{A}$ 
begin

mk-VLambda L-10-5-N-components'(2)[OF  $\mathfrak{R} \mathfrak{T}$ ]
  |vdomain L-10-5-N-ArrMap-vdomain[cat-Kan-cs-simps]
  |app L-10-5-N-ArrMap-app[cat-Kan-cs-simps]

```

end

15.2.3 *L-10-5-N* is a functor

lemma *L-10-5-N-is-functor*:

```

assumes  $Z \beta$ 
  and  $\alpha \in_o \beta$ 
  and  $\mathfrak{R} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{C}$ 
  and  $\mathfrak{T} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{A}$ 
  and  $c \in_o \mathfrak{C}(\text{Obj})$ 
shows L-10-5-N  $\alpha \beta \mathfrak{T} \mathfrak{R} c : \text{op-cat } \mathfrak{A} \mapsto \mapsto_{C\beta} \text{cat-Set } \beta$ 
proof-

```

let $?FUNCT = \langle \lambda \mathfrak{A}. \text{cat-FUNCT } \alpha \mathfrak{A} (\text{cat-Set } \alpha) \rangle$

interpret $\beta: Z \beta$ by (rule *assms(1)*)

interpret $\mathfrak{R}: \text{is-functor } \alpha \mathfrak{B} \mathfrak{C} \mathfrak{R}$ by (rule *assms(3)*)

interpret $\mathfrak{T}: \text{is-functor } \alpha \mathfrak{B} \mathfrak{A} \mathfrak{T}$ by (rule *assms(4)*)

from *assms(2)* interpret $FUNCT\text{-}\mathfrak{B}: \text{tiny-category } \beta \langle ?FUNCT \mathfrak{B} \rangle$
 by (cs-concl **cs-intro**: *cat-cs-intros cat-FUNCT-cs-intros*)

interpret $\beta\mathfrak{R}: \text{is-tiny-functor } \beta \mathfrak{B} \mathfrak{C} \mathfrak{R}$
 by (rule *is-functor.cf-is-tiny-functor-if-ge-Limit*)
 (use *assms(2)* in $\langle \text{cs-concl cs-intro: cat-cs-intros} \rangle$)+

interpret $\beta\mathfrak{T}: \text{is-tiny-functor } \beta \mathfrak{B} \mathfrak{A} \mathfrak{T}$
 by (rule *is-functor.cf-is-tiny-functor-if-ge-Limit*)
 (use *assms(2)* in $\langle \text{cs-concl cs-intro: cat-cs-intros} \rangle$)+

from *assms(2)* interpret *cf-nt*:
is-functor $\beta \langle ?FUNCT \mathfrak{B} \times_C \mathfrak{C} \rangle \langle \text{cat-Set } \beta \rangle \langle \text{cf-nt } \alpha \beta \mathfrak{R} \rangle$
 by (cs-concl **cs-simp**: *cat-cs-simps cs-intro: cat-cs-intros*)

show *?thesis*

proof(*intro is-functorI'*)

show *vfsequence* (*L-10-5-N* $\alpha \beta \mathfrak{T} \mathfrak{R} c$) unfolding *L-10-5-N-def* by *simp*

show *vcard* (*L-10-5-N* $\alpha \beta \mathfrak{T} \mathfrak{R} c$) = $4_{\mathbb{N}}$

unfolding *L-10-5-N-def* by (*simp add: nat-omega-simps*)

show *vsv* (*L-10-5-N* $\alpha \beta \mathfrak{T} \mathfrak{R} c(\text{ObjMap})$)

by (cs-concl **cs-shallow cs-intro**: *cat-Kan-cs-intros*)

from *assms(3,4)* show *vsv* (*L-10-5-N* $\alpha \beta \mathfrak{T} \mathfrak{R} c(\text{ArrMap})$)

by (cs-concl **cs-shallow cs-intro**: *cat-Kan-cs-intros*)

from *assms* show \mathcal{D}_o (*L-10-5-N* $\alpha \beta \mathfrak{T} \mathfrak{R} c(\text{ObjMap})$) = *op-cat* $\mathfrak{A}(\text{Obj})$

by

(

cs-concl cs-shallow
cs-simp: *cat-Kan-cs-simps cat-op-simps cs-intro: cat-cs-intros*
)

show $\mathcal{R}_\circ (L-10-5-N \alpha \beta \mathfrak{T} \mathfrak{K} c(\text{ObjMap})) \sqsubseteq_\circ \text{cat-Set } \beta(\text{Obj})$
unfolding *L-10-5-N-components'[OF \mathfrak{K}.is-functor-axioms \mathfrak{T}.is-functor-axioms]*
proof(*rule vrange-VLambda-vsubset*)
fix *a* **assume** *prems: a* $\in_\circ \mathfrak{A}(\text{Obj})$
from *prems assms* **show**
cf-nt $\alpha \beta \mathfrak{K}(\text{ObjMap})(\text{cf-map } (\text{Hom}_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} \mathfrak{T}), c) \bullet \in_\circ$
cat-Set $\beta(\text{Obj})$
by
(

cs-concl
cs-simp: *cat-Set-components(1) cat-cs-simps cat-FUNCT-cs-simps*
cs-intro:
cat-cs-intros FUNCT-\mathfrak{B}.cat-Hom-in-Vset cat-FUNCT-cs-intros

)

qed

from *assms* **show** $\mathcal{D}_\circ (L-10-5-N \alpha \beta \mathfrak{T} \mathfrak{K} c(\text{ArrMap})) = \text{op-cat } \mathfrak{A}(\text{Arr})$
by
(

cs-concl cs-shallow
cs-simp: *cat-Kan-cs-simps cat-op-simps cs-intro: cat-cs-intros*
)

show $L-10-5-N \alpha \beta \mathfrak{T} \mathfrak{K} c(\text{ArrMap})(f) :$
 $L-10-5-N \alpha \beta \mathfrak{T} \mathfrak{K} c(\text{ObjMap})(a) \mapsto_{\text{cat-Set } \beta} L-10-5-N \alpha \beta \mathfrak{T} \mathfrak{K} c(\text{ObjMap})(b)$
if $f : a \mapsto_{\text{op-cat } \mathfrak{A}} b$ **for** $a b f$
using *that assms*
unfolding *cat-op-simps*
by
(

cs-concl
cs-simp: *L-10-5-N-ArrMap-app L-10-5-N-ObjMap-app*
cs-intro: *cat-cs-intros cat-prod-cs-intros cat-FUNCT-cs-intros*
)

show
 $L-10-5-N \alpha \beta \mathfrak{T} \mathfrak{K} c(\text{ArrMap})(g \circ_{A \text{ op-cat } \mathfrak{A}} f) =$
 $L-10-5-N \alpha \beta \mathfrak{T} \mathfrak{K} c(\text{ArrMap})(g) \circ_{A \text{ cat-Set } \beta} L-10-5-N \alpha \beta \mathfrak{T} \mathfrak{K} c(\text{ArrMap})(f)$
if $g : b' \mapsto_{\text{op-cat } \mathfrak{A}} c'$ **and** $f : a' \mapsto_{\text{op-cat } \mathfrak{A}} b'$ **for** $b' c' g a' f$
proof-
from *that assms(5)* **show** *?thesis*
unfolding *cat-op-simps*
by
(

cs-concl
cs-intro:
cat-cs-intros
cat-prod-cs-intros
cat-FUNCT-cs-intros
cat-op-intros
cs-simp:
cat-cs-simps
cat-Kan-cs-simps
cat-FUNCT-cs-simps
cat-prod-cs-simps

$cat\text{-}op\text{-}simps$
 $cf\text{-}nt.cf\text{-}ArrMap\text{-}Comp[symmetric]$
))
qed
show
 $L\text{-}10\text{-}5\text{-}N \alpha \beta \mathfrak{T} \mathfrak{R} c(ArrMap)(op\text{-}cat \mathfrak{A}(CId)(a)) =$
 $cat\text{-}Set \beta(CId)(L\text{-}10\text{-}5\text{-}N \alpha \beta \mathfrak{T} \mathfrak{R} c(ObjMap)(a))$
if $a \in_o op\text{-}cat \mathfrak{A}(Obj)$ **for** a
proof-
note $[cat\text{-}cs\text{-}simps] =$
 $ntcf\text{-}id\text{-}cf\text{-}comp[symmetric]$
 $ntcf\text{-}arrow\text{-}id\text{-}ntcf\text{-}id[symmetric]$
 $cat\text{-}FUNCT\text{-}CId\text{-}app[symmetric]$
from $that[unfolding\ cat\text{-}op\text{-}simps]$ **assms** **show** $?thesis$
by
((
 $cs\text{-}concl$
cs-intro:
 $cat\text{-}cs\text{-}intros$
 $cat\text{-}FUNCT\text{-}cs\text{-}intros$
 $cat\text{-}prod\text{-}cs\text{-}intros$
 $cat\text{-}op\text{-}intros$
cs-simp:
 $cat\text{-}FUNCT\text{-}cs\text{-}simps\ cat\text{-}cs\text{-}simps\ cat\text{-}Kan\text{-}cs\text{-}simps\ cat\text{-}op\text{-}simps$
))
qed

qed $(cs\text{-}concl\ \mathbf{cs}\text{-}\mathbf{simp}: cat\text{-}Kan\text{-}cs\text{-}simps\ \mathbf{cs}\text{-}\mathbf{intro}: cat\text{-}cs\text{-}intros)+$

qed

lemma $L\text{-}10\text{-}5\text{-}N\text{-}is\text{-}functor'[cat\text{-}Kan\text{-}cs\text{-}intros]:$

assumes $Z \beta$
and $\alpha \in_o \beta$
and $\mathfrak{R} : \mathfrak{B} \mapsto_C \alpha \mathfrak{C}$
and $\mathfrak{T} : \mathfrak{B} \mapsto_C \alpha \mathfrak{A}$
and $c \in_o \mathfrak{C}(Obj)$
and $\mathfrak{A}' = op\text{-}cat \mathfrak{A}$
and $\mathfrak{B}' = cat\text{-}Set \beta$
and $\beta' = \beta$
shows $L\text{-}10\text{-}5\text{-}N \alpha \beta \mathfrak{T} \mathfrak{R} c : \mathfrak{A}' \mapsto_C \beta' \mathfrak{B}'$
using $assms(1\text{-}5)$ **unfolding** $assms(6\text{-}8)$ **by** $(rule\ L\text{-}10\text{-}5\text{-}N\text{-}is\text{-}functor)$

15.3 Lemma X.5: $L\text{-}10\text{-}5\text{-}v\text{-}arrow$

15.3.1 Definition and elementary properties

definition $L\text{-}10\text{-}5\text{-}v\text{-}arrow :: V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$

where $L\text{-}10\text{-}5\text{-}v\text{-}arrow \mathfrak{T} \mathfrak{R} c \tau a b =$

[
 $(\lambda f \in_o Hom(\mathfrak{R}(HomCod))\ c(\mathfrak{R}(ObjMap)(b)). \tau(NTMap)(0, b, f)\bullet),$
 $Hom(\mathfrak{R}(HomCod))\ c(\mathfrak{R}(ObjMap)(b)),$
 $Hom(\mathfrak{T}(HomCod))\ a(\mathfrak{T}(ObjMap)(b))$
]
]o

Components.

lemma $L\text{-}10\text{-}5\text{-}v\text{-}arrow\text{-}components:$

shows $L-10-5-v-arrow \mathfrak{T} \mathfrak{K} c \tau a b(\mathit{ArrVal}) =$
 $(\lambda f \in_{\circ} \mathit{Hom} (\mathfrak{K}(\mathit{HomCod})) c (\mathfrak{K}(\mathit{ObjMap})(b)). \tau(\mathit{NTMap})(0, b, f)).$
and $L-10-5-v-arrow \mathfrak{T} \mathfrak{K} c \tau a b(\mathit{ArrDom}) = \mathit{Hom} (\mathfrak{K}(\mathit{HomCod})) c (\mathfrak{K}(\mathit{ObjMap})(b))$
and $L-10-5-v-arrow \mathfrak{T} \mathfrak{K} c \tau a b(\mathit{ArrCod}) = \mathit{Hom} (\mathfrak{T}(\mathit{HomCod})) a (\mathfrak{T}(\mathit{ObjMap})(b))$
unfolding $L-10-5-v-arrow-def \mathit{arr-field-simps}$
by ($\mathit{simp-all} \mathit{add: nat-omega-simps}$)

context

fixes $\alpha \mathfrak{B} \mathfrak{C} \mathfrak{A} \mathfrak{K} \mathfrak{T}$
assumes $\mathfrak{K}: \mathfrak{K} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{T}: \mathfrak{T} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{A}$

begin

interpretation \mathfrak{K} : *is-functor* $\alpha \mathfrak{B} \mathfrak{C} \mathfrak{K}$ **by** (*rule* \mathfrak{K})

interpretation \mathfrak{T} : *is-functor* $\alpha \mathfrak{B} \mathfrak{A} \mathfrak{T}$ **by** (*rule* \mathfrak{T})

lemmas $L-10-5-v-arrow-components' = L-10-5-v-arrow-components[$
where $\mathfrak{T}=\mathfrak{T}$ **and** $\mathfrak{K}=\mathfrak{K}$, *unfolded cat-cs-simps*
 $]$

lemmas [$\mathit{cat-Kan-cs-simps}$] = $L-10-5-v-arrow-components'(2,3)$

end

15.3.2 Arrow value

mk-VLambda $L-10-5-v-arrow-components(1)$
 $[\mathit{vsu} \ L-10-5-v-arrow-ArrVal-\mathit{vsu}[\mathit{cat-Kan-cs-intros}]]$

context

fixes $\alpha \mathfrak{B} \mathfrak{C} \mathfrak{A} \mathfrak{K} \mathfrak{T}$
assumes $\mathfrak{K}: \mathfrak{K} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{T}: \mathfrak{T} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{A}$

begin

mk-VLambda $L-10-5-v-arrow-components'(1)[\mathit{OF} \ \mathfrak{K} \ \mathfrak{T}]$
 $[\mathit{vdomain} \ L-10-5-v-arrow-ArrVal-\mathit{vdomain}[\mathit{cat-Kan-cs-simps}]]$
 $[\mathit{app} \ L-10-5-v-arrow-ArrVal-\mathit{app}[\mathit{unfolded \ in-Hom-iff}]]$

end

lemma $L-10-5-v-arrow-ArrVal-\mathit{app}'$:

assumes $\mathfrak{K} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{T} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{A}$
and $f : c \mapsto_{\mathfrak{C}} \mathfrak{K}(\mathit{ObjMap})(b)$
shows $L-10-5-v-arrow \ \mathfrak{T} \ \mathfrak{K} \ c \ \tau \ a \ b(\mathit{ArrVal})(f) = \tau(\mathit{NTMap})(0, b, f).$

proof-

interpret \mathfrak{K} : *is-functor* $\alpha \mathfrak{B} \mathfrak{C} \mathfrak{K}$ **by** (*rule* $\mathit{assms}(1)$)
interpret \mathfrak{T} : *is-functor* $\alpha \mathfrak{B} \mathfrak{A} \mathfrak{T}$ **by** (*rule* $\mathit{assms}(2)$)
from $\mathit{assms}(3)$ **have** $c : c \in_{\circ} \mathfrak{C}(\mathit{Obj})$ **by** *auto*
show $?thesis$ **by** (*rule* $L-10-5-v-arrow-ArrVal-\mathit{app}[\mathit{OF} \ \mathit{assms}(1,2,3)]$)

qed

15.3.3 $L-10-5-v-arrow$ is an arrow

lemma $L-10-5-v-arrow-ArrVal-\mathit{is-arr}$:

assumes $\mathfrak{K} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{T} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{A}$

and $\tau' = \text{ntcf-arrow } \tau$
and $\tau : a \langle_{CF.cone} \mathfrak{T} \circ_{CF} c \text{ o}\square_{CF} \mathfrak{K} : c \downarrow_{CF} \mathfrak{K} \mapsto_{C\alpha} \mathfrak{A}$
and $f : c \mapsto_{\mathfrak{C}} \mathfrak{K}(\text{ObjMap})(\downarrow b)$
and $b \in_{\circ} \mathfrak{B}(\text{Obj})$
shows $L-10-5-v\text{-arrow } \mathfrak{T} \mathfrak{K} c \tau' a b(\text{ArrVal})(\downarrow f) : a \mapsto_{\mathfrak{A}} \mathfrak{T}(\text{ObjMap})(\downarrow b)$
proof-
interpret $\mathfrak{K} : \text{is-functor } \alpha \mathfrak{B} \mathfrak{C} \mathfrak{K}$ **by** (rule *assms(1)*)
interpret $\mathfrak{T} : \text{is-functor } \alpha \mathfrak{B} \mathfrak{A} \mathfrak{T}$ **by** (rule *assms(2)*)
interpret $\tau : \text{is-cat-cone } \alpha a \langle_{CF} \downarrow_{CF} \mathfrak{K} \rangle \mathfrak{A} \langle \mathfrak{T} \circ_{CF} c \text{ o}\square_{CF} \mathfrak{K} \rangle \tau$ **by** (rule *assms(4)*)
from *assms(5,6)* **show** ?thesis
unfolding *assms(3)*
by
(

cs-concl
cs-simp:
cat-cs-simps
L-10-5-v-arrow-ArrVal-app
cat-comma-cs-simps
cat-FUNCT-cs-simps
cs-intro: *cat-cs-intros cat-comma-cs-intros*
)

qed

lemma $L-10-5-v\text{-arrow-ArrVal-is-arr}'[\text{cat-Kan-cs-intros}]$:

assumes $\mathfrak{K} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{T} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{A}$
and $\tau' = \text{ntcf-arrow } \tau$
and $a' = a$
and $b' = \mathfrak{T}(\text{ObjMap})(\downarrow b)$
and $\mathfrak{A}' = \mathfrak{A}$
and $\tau : a \langle_{CF.cone} \mathfrak{T} \circ_{CF} c \text{ o}\square_{CF} \mathfrak{K} : c \downarrow_{CF} \mathfrak{K} \mapsto_{C\alpha} \mathfrak{A}$
and $f : c \mapsto_{\mathfrak{C}} \mathfrak{K}(\text{ObjMap})(\downarrow b)$
and $b \in_{\circ} \mathfrak{B}(\text{Obj})$
shows $L-10-5-v\text{-arrow } \mathfrak{T} \mathfrak{K} c \tau' a b(\text{ArrVal})(\downarrow f) : a' \mapsto_{\mathfrak{A}'} b'$
using *assms(1-3, 7-9)*
unfolding *assms(3-6)*
by (rule $L-10-5-v\text{-arrow-ArrVal-is-arr}$)

15.3.4 Further properties

lemma $L-10-5-v\text{-arrow-is-arr}$:

assumes $\mathfrak{K} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{T} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{A}$
and $c \in_{\circ} \mathfrak{C}(\text{Obj})$
and $\tau' = \text{ntcf-arrow } \tau$
and $\tau : a \langle_{CF.cone} \mathfrak{T} \circ_{CF} c \text{ o}\square_{CF} \mathfrak{K} : c \downarrow_{CF} \mathfrak{K} \mapsto_{C\alpha} \mathfrak{A}$
and $b \in_{\circ} \mathfrak{B}(\text{Obj})$
shows $L-10-5-v\text{-arrow } \mathfrak{T} \mathfrak{K} c \tau' a b :$
 $\text{Hom } \mathfrak{C} c (\mathfrak{K}(\text{ObjMap})(\downarrow b)) \mapsto_{\text{cat-Set } \alpha} \text{Hom } \mathfrak{A} a (\mathfrak{T}(\text{ObjMap})(\downarrow b))$

proof-

note $L-10-5-v\text{-arrow-components} = L-10-5-v\text{-arrow-components}'[\text{OF } \text{assms}(1,2)]$
interpret $\mathfrak{K} : \text{is-functor } \alpha \mathfrak{B} \mathfrak{C} \mathfrak{K}$ **by** (rule *assms(1)*)
interpret $\mathfrak{T} : \text{is-functor } \alpha \mathfrak{B} \mathfrak{A} \mathfrak{T}$ **by** (rule *assms(2)*)
interpret $\tau : \text{is-cat-cone } \alpha a \langle_{CF} \downarrow_{CF} \mathfrak{K} \rangle \mathfrak{A} \langle \mathfrak{T} \circ_{CF} c \text{ o}\square_{CF} \mathfrak{K} \rangle \tau$ **by** (rule *assms(5)*)
show ?thesis
proof(intro cat-Set-is-arrI)
show $\text{arr-Set } \alpha (L-10-5-v\text{-arrow } \mathfrak{T} \mathfrak{K} c \tau' a b)$
proof(intro arr-SetI)

show *vfsequence* (*L-10-5-v-arrow* $\mathfrak{T} \mathfrak{K} c \tau' a b$)
unfolding *L-10-5-v-arrow-def* **by** *simp*
show *vcard* (*L-10-5-v-arrow* $\mathfrak{T} \mathfrak{K} c \tau' a b$) = $\mathfrak{3}_\mathbb{N}$
unfolding *L-10-5-v-arrow-def* **by** (*simp add: nat-omega-simps*)
show
 \mathcal{R}_\circ (*L-10-5-v-arrow* $\mathfrak{T} \mathfrak{K} c \tau' a b(\text{ArrVal})$) \subseteq_\circ
L-10-5-v-arrow $\mathfrak{T} \mathfrak{K} c \tau' a b(\text{ArrCod})$
unfolding *L-10-5-v-arrow-components*
proof(*intro vrange-VLambda-vsubset, unfold in-Hom-iff*)
fix *f* **assume** $f : c \mapsto_{\mathfrak{C}} \mathfrak{K}(\text{ObjMap})(\text{!}b)$
from *L-10-5-v-arrow-ArrVal-is-arr*[*OF assms(1,2,4,5) this assms(6)*] *this*
show $\tau'(\text{NTMap})(\text{!}0, b, f)_\bullet : a \mapsto_{\mathfrak{A}} \mathfrak{T}(\text{ObjMap})(\text{!}b)$
by
(

cs-prems **cs-shallow**
cs-simp: *L-10-5-v-arrow-ArrVal-app' cat-cs-simps*
cs-intro: *cat-cs-intros*
)

qed
from *assms(3,6)* **show** *L-10-5-v-arrow* $\mathfrak{T} \mathfrak{K} c \tau' a b(\text{ArrDom}) \in_\circ \text{Vset } \alpha$
by (*cs-concl* **cs-simp:** *cat-Kan-cs-simps* **cs-intro:** *cat-cs-intros*)
from *assms(1-3,6)* τ .*cat-cone-obj* **show**
L-10-5-v-arrow $\mathfrak{T} \mathfrak{K} c \tau' a b(\text{ArrCod}) \in_\circ \text{Vset } \alpha$
by (*cs-concl* **cs-simp:** *cat-Kan-cs-simps* **cs-intro:** *cat-cs-intros*)
qed (*auto simp: L-10-5-v-arrow-components*)
qed (*simp-all add: L-10-5-v-arrow-components*)
qed

lemma *L-10-5-v-arrow-is-arr'*[*cat-Kan-cs-intros*]:
assumes $\mathfrak{K} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{T} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{A}$
and $c \in_\circ \mathfrak{C}(\text{Obj})$
and $\tau' = \text{ntcf-arrow } \tau$
and $\tau : a <_{CF.cone} \mathfrak{T} \circ_{CF} c \text{ o } \sqcap_{CF} \mathfrak{K} : c \downarrow_{CF} \mathfrak{K} \mapsto_{C\alpha} \mathfrak{A}$
and $b \in_\circ \mathfrak{B}(\text{Obj})$
and $A = \text{Hom } \mathfrak{C} c (\mathfrak{K}(\text{ObjMap})(\text{!}b))$
and $B = \text{Hom } \mathfrak{A} a (\mathfrak{T}(\text{ObjMap})(\text{!}b))$
and $\mathfrak{C}' = \text{cat-Set } \alpha$
shows *L-10-5-v-arrow* $\mathfrak{T} \mathfrak{K} c \tau' a b : A \mapsto_{\mathfrak{C}'} B$
using *assms(1-6)* **unfolding** *assms(7-9)* **by** (*rule L-10-5-v-arrow-is-arr'*)

lemma *L-10-5-v-cf-hom*[*cat-Kan-cs-simps*]:
assumes $\mathfrak{K} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{T} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{A}$
and $c \in_\circ \mathfrak{C}(\text{Obj})$
and $\tau' = \text{ntcf-arrow } \tau$
and $\tau : a <_{CF.cone} \mathfrak{T} \circ_{CF} c \text{ o } \sqcap_{CF} \mathfrak{K} : c \downarrow_{CF} \mathfrak{K} \mapsto_{C\alpha} \mathfrak{A}$
and $a \in_\circ \mathfrak{A}(\text{Obj})$
and $f : a' \mapsto_{\mathfrak{B}} b'$
shows
L-10-5-v-arrow $\mathfrak{T} \mathfrak{K} c \tau' a b' \circ_{A \text{ cat-Set } \alpha}$
cf-hom $\mathfrak{C} [\mathfrak{C}(\text{CId})(\text{!}c), \mathfrak{K}(\text{ArrMap})(\text{!}f)]_\circ =$
cf-hom $\mathfrak{A} [\mathfrak{A}(\text{CId})(\text{!}a), \mathfrak{T}(\text{ArrMap})(\text{!}f)]_\circ \circ_{A \text{ cat-Set } \alpha}$
L-10-5-v-arrow $\mathfrak{T} \mathfrak{K} c \tau' a a'$
(is ?lhs = ?rhs)

proof-

interpret \mathfrak{K} : *is-functor* $\alpha \mathfrak{B} \mathfrak{C} \mathfrak{K}$ **by** (*rule assms(1)*)

interpret \mathfrak{T} : *is-functor* $\alpha \mathfrak{B} \mathfrak{A} \mathfrak{T}$ **by** (*rule assms(2)*)

interpret τ : *is-cat-cone* $\alpha a \downarrow_{CF} \mathfrak{K} \mathfrak{A} \langle \mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} \rangle \tau$ **by** (*rule assms(5)*)

have [*cat-Kan-cs-simps*]:

$\tau(\mathcal{N}TMap)(\langle a'', b'', \mathfrak{K}(\mathcal{A}rrMap)(\langle h' \rangle) \circ_{A\mathfrak{C}} f' \rangle) \bullet =$
 $\mathfrak{T}(\mathcal{A}rrMap)(\langle h' \rangle) \circ_{A\mathfrak{A}} \tau(\mathcal{N}TMap)(\langle a', b', f' \rangle) \bullet$
if *F-def*: $F = [[a', b', f']_{\circ}, [a'', b'', f'']_{\circ}, [g', h']_{\circ}]_{\circ}$
and *A-def*: $A = [a', b', f']_{\circ}$
and *B-def*: $B = [a'', b'', f'']_{\circ}$
and *F*: $F : A \mapsto_c \downarrow_{CF} \mathfrak{K} B$
for $F A B a' b' f' a'' b'' f'' g' h'$

proof-

from F [*unfolded F-def A-def B-def*] *assms(3)* **have** *a'-def*: $a' = 0$
and *a''-def*: $a'' = 0$
and *g'-def*: $g' = 0$
and *h'*: $h' : b' \mapsto_{\mathfrak{B}} b''$
and *f'*: $f' : c \mapsto_{\mathfrak{C}} \mathfrak{K}(\mathcal{O}bjMap)(\langle b' \rangle)$
and *f''*: $f'' : c \mapsto_{\mathfrak{C}} \mathfrak{K}(\mathcal{O}bjMap)(\langle b'' \rangle)$
and *f''-def*: $\mathfrak{K}(\mathcal{A}rrMap)(\langle h' \rangle) \circ_{A\mathfrak{C}} f' = f''$
by *auto*

note τ .*cat-cone-Comp-commute* [*cat-cs-simps del*]

from

τ .*ntcf-Comp-commute* [*OF F*]
that(2) $F g' h' f' f''$
 \mathfrak{K} .*is-functor-axioms*
 \mathfrak{T} .*is-functor-axioms*

show

$\tau(\mathcal{N}TMap)(\langle a'', b'', \mathfrak{K}(\mathcal{A}rrMap)(\langle h' \rangle) \circ_{A\mathfrak{C}} f' \rangle) \bullet =$
 $\mathfrak{T}(\mathcal{A}rrMap)(\langle h' \rangle) \circ_{A\mathfrak{A}} \tau(\mathcal{N}TMap)(\langle a', b', f' \rangle) \bullet$
unfolding *F-def A-def B-def a'-def a''-def g'-def*
by
 (
 cs-prems
 cs-simp: *cat-cs-simps cat-comma-cs-simps f''-def* [*symmetric*]
 cs-intro: *cat-cs-intros cat-comma-cs-intros*
)

qed

from *assms(3)* *assms(6,7)* \mathfrak{K} .*HomCod.category-axioms* **have** *lhs-is-arr*:

$?lhs : Hom \mathfrak{C} c (\mathfrak{K}(\mathcal{O}bjMap)(\langle a' \rangle)) \mapsto_{cat-Set} \alpha Hom \mathfrak{A} a (\mathfrak{T}(\mathcal{O}bjMap)(\langle b' \rangle))$

unfolding *assms(4)*

by

(
 cs-concl **cs-intro**:
 cat-lim-cs-intros
 cat-cs-intros
 cat-Kan-cs-intros
 cat-prod-cs-intros
 cat-op-intros
)

then have *dom-lhs*: $\mathcal{D}_{\circ} ((?lhs)(\mathcal{A}rrVal)) = Hom \mathfrak{C} c (\mathfrak{K}(\mathcal{O}bjMap)(\langle a' \rangle))$

by (*cs-concl* **cs-shallow** **cs-simp**: *cat-cs-simps*)

from *assms(3)* *assms(6,7)* \mathfrak{K} .*HomCod.category-axioms* \mathfrak{T} .*HomCod.category-axioms*

have *rhs-is-arr*:

$?rhs : Hom \mathfrak{C} c (\mathfrak{K}(\mathcal{O}bjMap)(\langle a' \rangle)) \mapsto_{cat-Set} \alpha Hom \mathfrak{A} a (\mathfrak{T}(\mathcal{O}bjMap)(\langle b' \rangle))$

unfolding *assms(4)*

by

(

```

    cs-concl cs-intro:
      cat-lim-cs-intros
      cat-cs-intros
      cat-Kan-cs-intros
      cat-prod-cs-intros
      cat-op-intros
  )
then have dom-rhs:  $\mathcal{D}_o ((?rhs)(\downarrow ArrVal)) = Hom \mathfrak{C} c (\mathfrak{K}(\downarrow ObjMap)(\downarrow a'))$ 
  by (cs-concl cs-shallow cs-simp: cat-cs-simps)
show ?thesis
proof(rule arr-Set-eqI)
  from lhs-is-arr show arr-Set-lhs: arr-Set  $\alpha$  ?lhs
    by (auto dest: cat-Set-is-arrD(1))
  from rhs-is-arr show arr-Set-rhs: arr-Set  $\alpha$  ?rhs
    by (auto dest: cat-Set-is-arrD(1))
  show ?lhs(\downarrow ArrVal) = ?rhs(\downarrow ArrVal)
proof(rule vsu-eqI, unfold dom-lhs dom-rhs in-Hom-iff)
  fix g assume prems:  $g : c \mapsto_{\mathfrak{C}} \mathfrak{K}(\downarrow ObjMap)(\downarrow a')$ 
  from prems assms(7) have  $\mathfrak{K}f$ :
     $\mathfrak{K}(\downarrow ArrMap)(\downarrow f) \circ_{A\mathfrak{C}} g : c \mapsto_{\mathfrak{C}} \mathfrak{K}(\downarrow ObjMap)(\downarrow b')$ 
  by (cs-concl cs-shallow cs-intro: cat-cs-intros)
  with  $assms(6,7)$  prems  $\mathfrak{K}.HomCod.category-axioms \mathfrak{T}.HomCod.category-axioms$ 
  show ?lhs(\downarrow ArrVal)(\downarrow g) = ?rhs(\downarrow ArrVal)(\downarrow g)
  by
    (
      cs-concl
      cs-intro:
        cat-lim-cs-intros
        cat-cs-intros
        cat-Kan-cs-intros
        cat-comma-cs-intros
        cat-prod-cs-intros
        cat-op-intros
        cat-1-is-arrI
      cs-simp:
        L-10-5-v-arrow-ArrVal-app'
        cat-cs-simps
        cat-Kan-cs-simps
        cat-op-simps
        cat-FUNCT-cs-simps
        cat-comma-cs-simps
        assms(4)
    )+
  qed (use arr-Set-lhs arr-Set-rhs in auto)
qed
  (
    use lhs-is-arr rhs-is-arr in
    <cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros>
  )+
qed

```

15.4 Lemma X.5: $L-10-5-\tau$

15.4.1 Definition and elementary properties

definition $L-10-5-\tau$ where $L-10-5-\tau \mathfrak{T} \mathfrak{K} c v a =$

$$[(\lambda bf \in_o c \downarrow_{CF} \mathfrak{K}(\downarrow Obj). v(\downarrow NTMap)(\downarrow bf(\downarrow 1_{\mathbb{N}}))(\downarrow ArrVal)(\downarrow bf(\downarrow 2_{\mathbb{N}}))),$$

$cf\text{-const } (c \downarrow_{CF} \mathfrak{K}) (\mathfrak{T}(\mathit{HomCod})) a,$
 $\mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K},$
 $c \downarrow_{CF} \mathfrak{K},$
 $(\mathfrak{T}(\mathit{HomCod}))$
 $]_o$

Components.

lemma *L-10-5- τ -components*:

shows *L-10-5- τ* $\mathfrak{T} \mathfrak{K} c v a(\mathit{NTMap}) =$
 $(\lambda bf \in_o c \downarrow_{CF} \mathfrak{K}(\mathit{Obj}). v(\mathit{NTMap})(\mathit{bf})(\mathit{I}_{\mathbb{N}})(\mathit{ArrVal})(\mathit{bf})(\mathit{2}_{\mathbb{N}}))$
and *L-10-5- τ* $\mathfrak{T} \mathfrak{K} c v a(\mathit{NTDom}) = cf\text{-const } (c \downarrow_{CF} \mathfrak{K}) (\mathfrak{T}(\mathit{HomCod})) a$
and *L-10-5- τ* $\mathfrak{T} \mathfrak{K} c v a(\mathit{NTCod}) = \mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K}$
and *L-10-5- τ* $\mathfrak{T} \mathfrak{K} c v a(\mathit{NTDGDom}) = c \downarrow_{CF} \mathfrak{K}$
and *L-10-5- τ* $\mathfrak{T} \mathfrak{K} c v a(\mathit{NTDGCod}) = (\mathfrak{T}(\mathit{HomCod}))$
unfolding *L-10-5- τ -def nt-field-simps* **by** (*simp-all add: nat-omega-simps*)

context

fixes $\alpha \mathfrak{B} \mathfrak{C} \mathfrak{A} \mathfrak{K} \mathfrak{T}$
assumes $\mathfrak{K}: \mathfrak{K} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{T}: \mathfrak{T} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{A}$

begin

interpretation \mathfrak{K} : *is-functor* $\alpha \mathfrak{B} \mathfrak{C} \mathfrak{K}$ **by** (*rule* \mathfrak{K})

interpretation \mathfrak{T} : *is-functor* $\alpha \mathfrak{B} \mathfrak{A} \mathfrak{T}$ **by** (*rule* \mathfrak{T})

lemmas *L-10-5- τ -components'* = *L-10-5- τ -components*[
where $\mathfrak{T}=\mathfrak{T}$ **and** $\mathfrak{K}=\mathfrak{K}$, *unfolded cat-cs-simps*
 $]_o$

lemmas [*cat-Kan-cs-simps*] = *L-10-5- τ -components'*(2-5)

end

15.4.2 Natural transformation map

mk-VLambda *L-10-5- τ -components(1)*

$[vsv \text{ L-10-5-}\tau\text{-NTMap-vsv}[cat\text{-Kan-cs-intros}]]$
 $[vdomain \text{ L-10-5-}\tau\text{-NTMap-vdomain}[cat\text{-Kan-cs-simps}]]$

lemma *L-10-5- τ -NTMap-app*[*cat-Kan-cs-simps*]:

assumes $bf = [0, b, f]_o$ **and** $bf \in_o c \downarrow_{CF} \mathfrak{K}(\mathit{Obj})$
shows *L-10-5- τ* $\mathfrak{T} \mathfrak{K} c v a(\mathit{NTMap})(\mathit{bf}) = v(\mathit{NTMap})(\mathit{b})(\mathit{ArrVal})(\mathit{f})$
using *assms unfolding L-10-5- τ -components* **by** (*simp add: nat-omega-simps*)

15.4.3 *L-10-5- τ* is a cone

lemma *L-10-5- τ -is-cat-cone*[*cat-cs-intros*]:

assumes $\mathfrak{K} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{T} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{A}$
and $c \in_o \mathfrak{C}(\mathit{Obj})$
and $v'\text{-def}: v' = ntcf\text{-arrow } v$
and $v : v :$
 $Hom_{O.C\alpha}\mathfrak{C}(c, -) \circ_{CF} \mathfrak{K} \mapsto_{CF} Hom_{O.C\alpha}\mathfrak{A}(a, -) \circ_{CF} \mathfrak{T} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \text{cat-Set } \alpha$
and $a : a \in_o \mathfrak{A}(\mathit{Obj})$

shows *L-10-5- τ* $\mathfrak{T} \mathfrak{K} c v' a : a <_{CF.cone} \mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} : c \downarrow_{CF} \mathfrak{K} \mapsto \mapsto_{C\alpha} \mathfrak{A}$

proof-

let $?H\text{-}\mathfrak{C} = \langle \lambda c. Hom_{O.C\alpha}\mathfrak{C}(c, -) \rangle$

let $?H\mathfrak{A} = \langle \lambda a. \text{Hom}_{O.C\alpha} \mathfrak{A}(a, -) \rangle$

interpret \mathfrak{K} : *is-functor* $\alpha \mathfrak{B} \mathfrak{C} \mathfrak{K}$ by (rule *assms(1)*)

interpret \mathfrak{T} : *is-functor* $\alpha \mathfrak{B} \mathfrak{A} \mathfrak{T}$ by (rule *assms(2)*)

from *assms(3)* interpret $c\mathfrak{K}$: *category* $\alpha \langle c \downarrow_{CF} \mathfrak{K} \rangle$

by (*cs-concl cs-shallow cs-intro: cat-comma-cs-intros*)

from *assms(3)* interpret Πc : *is-functor* $\alpha \langle c \downarrow_{CF} \mathfrak{K} \rangle \mathfrak{B} \langle c \circ \sqcap_{CF} \mathfrak{K} \rangle$

by

(
cs-concl cs-shallow
cs-simp: cat-comma-cs-simps
cs-intro: cat-cs-intros cat-comma-cs-intros
)

interpret v : *is-ntcf* $\alpha \mathfrak{B} \langle \text{cat-Set } \alpha \rangle \langle ?H\mathfrak{C} \ c \circ_{CF} \mathfrak{K} \rangle \langle ?H\mathfrak{A} \ a \circ_{CF} \mathfrak{T} \rangle v$

by (rule v)

show *?thesis*

proof(*intro is-cat-coneI is-ntcfI'*)

show *vfsequence* ($L-10-5-\tau \ \mathfrak{T} \ \mathfrak{K} \ c \ v' \ a$) **unfolding** $L-10-5-\tau$ -def by *simp*

show *vcard* ($L-10-5-\tau \ \mathfrak{T} \ \mathfrak{K} \ c \ v' \ a$) = $5_{\mathbb{N}}$

unfolding $L-10-5-\tau$ -def by (*simp add: nat-omega-simps*)

from a interpret *cf-const*:

is-functor $\alpha \langle c \downarrow_{CF} \mathfrak{K} \rangle \mathfrak{A} \langle \text{cf-const} \ (c \downarrow_{CF} \mathfrak{K}) \ \mathfrak{A} \ a \rangle$

by (*cs-concl cs-intro: cat-cs-intros*)

show $L-10-5-\tau \ \mathfrak{T} \ \mathfrak{K} \ c \ v' \ a \ (\text{NTMap}) \ (\text{!}bf)$:

cf-const $(c \downarrow_{CF} \mathfrak{K}) \ \mathfrak{A} \ a \ (\text{ObjMap}) \ (\text{!}bf) \mapsto_{\mathfrak{A}} (\mathfrak{T} \circ_{CF} \ c \ \circ \sqcap_{CF} \ \mathfrak{K}) \ (\text{ObjMap}) \ (\text{!}bf)$

if $bf \in_{\circ} c \downarrow_{CF} \mathfrak{K} \ (\text{Obj})$ for bf

proof-

from *that assms(3)* obtain $b \ f$

where *bf-def*: $bf = [0, b, f]_{\circ}$

and b : $b \in_{\circ} \mathfrak{B} \ (\text{Obj})$

and f : $f : c \mapsto_{\mathfrak{C}} \mathfrak{K} \ (\text{ObjMap}) \ (\text{!}b)$

by *auto*

from $v.\text{ntcf-NTMap-is-arr} \ [OF \ b] \ a \ b \ \text{assms}(3) \ f$ have $v \ (\text{NTMap}) \ (\text{!}b)$:

$\text{Hom} \ \mathfrak{C} \ c \ (\mathfrak{K} \ (\text{ObjMap}) \ (\text{!}b)) \mapsto_{\text{cat-Set } \alpha} \text{Hom} \ \mathfrak{A} \ a \ (\mathfrak{T} \ (\text{ObjMap}) \ (\text{!}b))$

by

(
cs-prems cs-shallow
cs-simp: cat-cs-simps cat-op-simps
cs-intro: cat-cs-intros cat-op-intros
)

with *that b f* show $L-10-5-\tau \ \mathfrak{T} \ \mathfrak{K} \ c \ v' \ a \ (\text{NTMap}) \ (\text{!}bf)$:

cf-const $(c \downarrow_{CF} \mathfrak{K}) \ \mathfrak{A} \ a \ (\text{ObjMap}) \ (\text{!}bf) \mapsto_{\mathfrak{A}} (\mathfrak{T} \circ_{CF} \ c \ \circ \sqcap_{CF} \ \mathfrak{K}) \ (\text{ObjMap}) \ (\text{!}bf)$

unfolding *bf-def v'-def*

by

(
cs-concl
cs-simp:
cat-cs-simps
cat-Kan-cs-simps
cat-comma-cs-simps
cat-FUNCT-cs-simps
cs-intro: cat-cs-intros cat-comma-cs-intros
)

qed

show

$L-10-5-\tau \ \mathfrak{T} \ \mathfrak{K} \ c \ v' \ a \ (\mathcal{NTMap}) \ (\downarrow B) \ \circ_{A\mathfrak{A}} \ cf-const \ (c \ \downarrow_{CF} \ \mathfrak{K}) \ \mathfrak{A} \ a \ (\mathcal{ArrMap}) \ (\downarrow F) =$
 $(\mathfrak{T} \ \circ_{CF} \ c \ \circ \ \downarrow_{CF} \ \mathfrak{K}) \ (\mathcal{ArrMap}) \ (\downarrow F) \ \circ_{A\mathfrak{A}} \ L-10-5-\tau \ \mathfrak{T} \ \mathfrak{K} \ c \ v' \ a \ (\mathcal{NTMap}) \ (\downarrow A)$
if $F : A \mapsto_c \downarrow_{CF} \ \mathfrak{K} \ B$ **for** $A \ B \ F$

proof-

from \mathfrak{K} .*is-functor-axioms* **that** *assms*(β) **obtain** $a' \ f \ a'' \ f' \ g$

where F -*def*: $F = [[0, a', f]_{\circ}, [0, a'', f']_{\circ}, [0, g]_{\circ}]_{\circ}$

and A -*def*: $A = [0, a', f]_{\circ}$

and B -*def*: $B = [0, a'', f']_{\circ}$

and g : $g : a' \mapsto_{\mathfrak{B}} a''$

and f : $f : c \mapsto_{\mathfrak{C}} \mathfrak{K}(\mathcal{ObjMap})(\downarrow a')$

and f' : $f' : c \mapsto_{\mathfrak{C}} \mathfrak{K}(\mathcal{ObjMap})(\downarrow a'')$

and f' -*def*: $\mathfrak{K}(\mathcal{ArrMap})(\downarrow g) \ \circ_{A\mathfrak{C}} \ f = f'$

by *auto*

from v .*ntcf-Comp-commute*[$OF \ g$] **have**

$(v(\mathcal{NTMap})(\downarrow a'')) \ \circ_{A \text{cat-Set} \ \alpha} \ (\ ?H\text{-}\mathfrak{C} \ c \ \circ_{CF} \ \mathfrak{K})(\mathcal{ArrMap})(\downarrow g)(\mathcal{ArrVal})(\downarrow f) =$

$((\ ?H\text{-}\mathfrak{A} \ a \ \circ_{CF} \ \mathfrak{T})(\mathcal{ArrMap})(\downarrow g) \ \circ_{A \text{cat-Set} \ \alpha} \ v(\mathcal{NTMap})(\downarrow a'))(\mathcal{ArrVal})(\downarrow f)$

by *simp*

from *this* $a \ g \ f \ f' \ \mathfrak{K}$.*HomCod.category-axioms* \mathfrak{T} .*HomCod.category-axioms*

have [*cat-cs-simps*]:

$v(\mathcal{NTMap})(\downarrow a'')(\mathcal{ArrVal})(\downarrow \mathfrak{K}(\mathcal{ArrMap})(\downarrow g) \ \circ_{A\mathfrak{C}} \ f) =$

$\mathfrak{T}(\mathcal{ArrMap})(\downarrow g) \ \circ_{A\mathfrak{A}} \ v(\mathcal{NTMap})(\downarrow a')(\mathcal{ArrVal})(\downarrow f)$

by

(

cs-prems

cs-simp: *cat-cs-simps cat-op-simps*

cs-intro: *cat-cs-intros cat-prod-cs-intros cat-op-intros*

)

from *that* $a \ g \ f \ f' \ \mathfrak{K}$.*HomCod.category-axioms* \mathfrak{T} .*HomCod.category-axioms*

show *?thesis*

unfolding F -*def* A -*def* B -*def* v' -*def*

by

(

cs-concl

cs-simp:

f' -*def*[*symmetric*]

cat-cs-simps

cat-Kan-cs-simps

cat-comma-cs-simps

cat-FUNCT-cs-simps

cat-op-simps

cs-intro: *cat-cs-intros cat-op-intros*

)

qed

qed

(

use assms in

<

cs-concl

cs-simp: *cat-cs-simps cat-Kan-cs-simps*

cs-intro: *cat-cs-intros cat-Kan-cs-intros a*

>

)+

qed

lemma $L-10-5-\tau$ -*is-cat-cone'*[*cat-Kan-cs-intros*]:

assumes $\mathfrak{K} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$

and $\mathfrak{T} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{A}$
 and $c \in_{\circ} \mathfrak{C}(\text{Obj})$
 and $v' = \text{ntcf-arrow } v$
 and $\mathfrak{F}' = \mathfrak{T} \circ_{CF} c \circ \prod_{CF} \mathfrak{K}$
 and $c\mathfrak{K} = c \downarrow_{CF} \mathfrak{K}$
 and $\mathfrak{A}' = \mathfrak{A}$
 and $\alpha' = \alpha$
 and $v :$
 $\text{Hom}_{O.C\alpha} \mathfrak{C}(c, -) \circ_{CF} \mathfrak{K} \mapsto_{CF} \text{Hom}_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} \mathfrak{T} :$
 $\mathfrak{B} \mapsto_{C\alpha} \text{cat-Set } \alpha$
 and $a \in_{\circ} \mathfrak{A}(\text{Obj})$
shows *L-10-5- τ* $\mathfrak{T} \mathfrak{K} c v' a : a <_{CF, \text{cone}} \mathfrak{F}' : c\mathfrak{K} \mapsto_{C\alpha'} \mathfrak{A}'$
using *assms(1-4, 9, 10)* **unfolding** *assms(5-8)* **by** (*rule L-10-5- τ -is-cat-cone*)

15.5 Lemma X.5: *L-10-5-v*

15.5.1 Definition and elementary properties

definition *L-10-5-v* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$

where *L-10-5-v* $\alpha \mathfrak{T} \mathfrak{K} c \tau a =$

$[$
 $(\lambda b \in_{\circ} \mathfrak{T}(\text{HomDom})(\text{Obj}). \text{L-10-5-v-arrow } \mathfrak{T} \mathfrak{K} c \tau a b),$
 $\text{Hom}_{O.C\alpha} \mathfrak{K}(\text{HomCod})(c, -) \circ_{CF} \mathfrak{K},$
 $\text{Hom}_{O.C\alpha} \mathfrak{T}(\text{HomCod})(a, -) \circ_{CF} \mathfrak{T},$
 $\mathfrak{T}(\text{HomDom}),$
 $\text{cat-Set } \alpha$
 $]$

Components.

lemma *L-10-5-v-components*:

shows *L-10-5-v* $\alpha \mathfrak{T} \mathfrak{K} c \tau a(\text{NTMap}) =$

$(\lambda b \in_{\circ} \mathfrak{T}(\text{HomDom})(\text{Obj}). \text{L-10-5-v-arrow } \mathfrak{T} \mathfrak{K} c \tau a b)$

and *L-10-5-v* $\alpha \mathfrak{T} \mathfrak{K} c \tau a(\text{NTDom}) = \text{Hom}_{O.C\alpha} \mathfrak{K}(\text{HomCod})(c, -) \circ_{CF} \mathfrak{K}$

and *L-10-5-v* $\alpha \mathfrak{T} \mathfrak{K} c \tau a(\text{NTCod}) = \text{Hom}_{O.C\alpha} \mathfrak{T}(\text{HomCod})(a, -) \circ_{CF} \mathfrak{T}$

and *L-10-5-v* $\alpha \mathfrak{T} \mathfrak{K} c \tau a(\text{NTDGDom}) = \mathfrak{T}(\text{HomDom})$

and *L-10-5-v* $\alpha \mathfrak{T} \mathfrak{K} c \tau a(\text{NTDGCod}) = \text{cat-Set } \alpha$

unfolding *L-10-5-v-def nt-field-simps* **by** (*simp-all add: nat-omega-simps*)

context

fixes $\alpha \mathfrak{B} \mathfrak{C} \mathfrak{A} \mathfrak{K} \mathfrak{T}$

assumes $\mathfrak{K} : \mathfrak{K} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{T} : \mathfrak{T} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{A}$

begin

interpretation \mathfrak{K} : *is-functor* $\alpha \mathfrak{B} \mathfrak{C} \mathfrak{K}$ **by** (*rule* \mathfrak{K})

interpretation \mathfrak{T} : *is-functor* $\alpha \mathfrak{B} \mathfrak{A} \mathfrak{T}$ **by** (*rule* \mathfrak{T})

lemmas *L-10-5-v-components'* = *L-10-5-v-components*[

where $\mathfrak{T}=\mathfrak{T}$ **and** $\mathfrak{K}=\mathfrak{K}$, *unfolded cat-cs-simps*

]

lemmas [*cat-Kan-cs-simps*] = *L-10-5-v-components'*(2-5)

end

15.5.2 Natural transformation map

mk-VLambda *L-10-5-v-components(1)*

[*vsu L-10-5-v-NTMap-vsuv[cat-Kan-cs-intros]*]

context

fixes $\alpha \mathfrak{B} \mathfrak{C} \mathfrak{A} \mathfrak{K} \mathfrak{T}$

assumes $\mathfrak{K}: \mathfrak{K} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{C}$

and $\mathfrak{T}: \mathfrak{T} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{A}$

begin

interpretation \mathfrak{K} : *is-functor* $\alpha \mathfrak{B} \mathfrak{C} \mathfrak{K}$ by (rule \mathfrak{K})

interpretation \mathfrak{T} : *is-functor* $\alpha \mathfrak{B} \mathfrak{A} \mathfrak{T}$ by (rule \mathfrak{T})

mk-VLambda *L-10-5-v-components'(1)*[*OF* $\mathfrak{K} \mathfrak{T}$]
|*vdomain L-10-5-v-NTMap-vdomain[cat-Kan-cs-simps]*||
|*app L-10-5-v-NTMap-app[cat-Kan-cs-simps]*||

end

15.5.3 *L-10-5-v* is a natural transformation

lemma *L-10-5-v-is-ntcf*:

assumes $\mathfrak{K} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{C}$

and $\mathfrak{T} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{A}$

and $c \in_{\circ} \mathfrak{C}(\text{Obj})$

and τ' -def: $\tau' = \text{ntcf-arrow } \tau$

and $\tau: \tau : a \langle c \downarrow_{CF} \text{cone } \mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} : c \downarrow_{CF} \mathfrak{K} \mapsto \mapsto_{C\alpha} \mathfrak{A}$

and $a: a \in_{\circ} \mathfrak{A}(\text{Obj})$

shows *L-10-5-v* $\alpha \mathfrak{T} \mathfrak{K} c \tau' a :$

$\text{Hom}_{O.C\alpha} \mathfrak{C}(c, -) \circ_{CF} \mathfrak{K} \mapsto_{CF} \text{Hom}_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} \mathfrak{T} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \text{cat-Set } \alpha$

(is $\langle ?L-10-5-v : ?H\text{-}\mathfrak{C} c \circ_{CF} \mathfrak{K} \mapsto_{CF} ?H\text{-}\mathfrak{A} a \circ_{CF} \mathfrak{T} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \text{cat-Set } \alpha \rangle$)

proof-

interpret \mathfrak{K} : *is-functor* $\alpha \mathfrak{B} \mathfrak{C} \mathfrak{K}$ by (rule *assms(1)*)

interpret \mathfrak{T} : *is-functor* $\alpha \mathfrak{B} \mathfrak{A} \mathfrak{T}$ by (rule *assms(2)*)

interpret τ : *is-cat-cone* $\alpha a \langle c \downarrow_{CF} \mathfrak{K} \rangle \mathfrak{A} \langle \mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} \rangle \tau$
by (rule *assms(5)*)

from *assms(3)* interpret $c\mathfrak{K}$: *category* $\alpha \langle c \downarrow_{CF} \mathfrak{K} \rangle$

by (*cs-concl cs-shallow cs-intro: cat-comma-cs-intros*)

from *assms(3)* interpret Πc : *is-functor* $\alpha \langle c \downarrow_{CF} \mathfrak{K} \rangle \mathfrak{B} \langle c \circ \sqcap_{CF} \mathfrak{K} \rangle$

by

(

cs-concl cs-shallow

cs-simp: cat-comma-cs-simps

cs-intro: cat-cs-intros cat-comma-cs-intros

)

show $?L-10-5-v : ?H\text{-}\mathfrak{C} c \circ_{CF} \mathfrak{K} \mapsto_{CF} ?H\text{-}\mathfrak{A} a \circ_{CF} \mathfrak{T} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \text{cat-Set } \alpha$

proof(*intro is-ntcfI'*)

show *vfsequence* $?L-10-5-v$ **unfolding** *L-10-5-v-def* **by** *auto*

show *vcard* $?L-10-5-v = 5_{\mathbb{N}}$

unfolding *L-10-5-v-def* **by** (*simp add: nat-omega-simps*)

show $?L-10-5-v(\text{NTMap})(\text{Obj}) : \text{cat-Set } \alpha$

($?H\text{-}\mathfrak{C} c \circ_{CF} \mathfrak{K})(\text{ObjMap})(\text{Obj}) \mapsto_{\text{cat-Set } \alpha} (?H\text{-}\mathfrak{A} a \circ_{CF} \mathfrak{T})(\text{ObjMap})(\text{Obj})$)

if $b \in_{\circ} \mathfrak{B}(\text{Obj})$ for b

proof-

from a that *assms(3)* **show** *?thesis*

unfolding τ' -def

by

(

cs-concl cs-shallow
cs-simp: *cat-cs-simps cat-Kan-cs-simps*
cs-intro:
cat-Kan-cs-intros
cat-lim-cs-intros
cat-cs-intros
cat-op-intros

)

qed

show

$?L-10-5-v(\downarrow NTMap)(\downarrow b') \circ_{A \text{ cat-Set } \alpha} (?H-\mathfrak{C} \ c \circ_{CF} \ \mathfrak{K})(\downarrow ArrMap)(\downarrow f) =$
 $(?H-\mathfrak{A} \ a \circ_{CF} \ \mathfrak{T})(\downarrow ArrMap)(\downarrow f) \circ_{A \text{ cat-Set } \alpha} ?L-10-5-v(\downarrow NTMap)(\downarrow a')$

if $f : a' \mapsto_{\mathfrak{B}} b'$ **for** $a' b' f$

proof-

from *that a assms(3)* **show** *?thesis*

by

(

cs-concl

cs-simp: *cat-cs-simps cat-Kan-cs-simps cat-op-simps τ' -def*

cs-intro: *cat-lim-cs-intros cat-cs-intros*

)

qed

qed

(

use assms(3,6) **in**

<

cs-concl

cs-simp: cat-cs-simps cat-Kan-cs-simps

cs-intro: cat-cs-intros cat-Kan-cs-intros

>

)+

qed

lemma *L-10-5-v-is-ntcf'[cat-Kan-cs-intros]:*

assumes $\mathfrak{K} : \mathfrak{B} \mapsto_{CF} \mathfrak{C}$
and $\mathfrak{T} : \mathfrak{B} \mapsto_{CF} \mathfrak{A}$
and $c \in_{\circ} \mathfrak{C}(\downarrow Obj)$
and $\tau' = \text{ntcf-arrow } \tau$
and $\mathfrak{F}' = \text{Hom}_{O.C\alpha} \mathfrak{C}(c, -) \circ_{CF} \mathfrak{K}$
and $\mathfrak{G}' = \text{Hom}_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} \mathfrak{T}$
and $\mathfrak{B}' = \mathfrak{B}$
and $\mathfrak{C}' = \text{cat-Set } \alpha$
and $\alpha' = \alpha$
and $\tau : a <_{CF.cone} \mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} : c \downarrow_{CF} \mathfrak{K} \mapsto_{CF} \mathfrak{C}' \ \mathfrak{A}$
and $a \in_{\circ} \mathfrak{A}(\downarrow Obj)$

shows *L-10-5-v* $\alpha \ \mathfrak{T} \ \mathfrak{K} \ c \ \tau' \ a : \mathfrak{F}' \mapsto_{CF} \mathfrak{G}' : \mathfrak{B}' \mapsto_{CF} \mathfrak{C}'$

using *assms(1-4,10,11)* **unfolding** *assms(5-9)* **by** *(rule L-10-5-v-is-ntcf)*

15.6 Lemma X.5: *L-10-5- χ -arrow*

15.6.1 Definition and elementary properties

definition *L-10-5- χ -arrow*

where *L-10-5- χ -arrow* $\alpha \ \beta \ \mathfrak{T} \ \mathfrak{K} \ c \ a =$

[

$(\lambda v \in_{\circ} L-10-5-N \ \alpha \ \beta \ \mathfrak{T} \ \mathfrak{K} \ c(\downarrow ObjMap)(\downarrow a). \text{ntcf-arrow } (L-10-5-\tau \ \mathfrak{T} \ \mathfrak{K} \ c \ v \ a)),$

$L-10-5-N \alpha \beta \mathfrak{T} \mathfrak{K} c(\mathit{ObjMap})(\mathit{!}a),$
 $cf\text{-Cone } \alpha \beta (\mathfrak{T} \circ_{CF} c \circ \mathit{!} \square_{CF} \mathfrak{K})(\mathit{ObjMap})(\mathit{!}a)$
 $\mathit{!} \circ$

Components.

lemma $L-10-5-\chi\text{-arrow-components}$:

shows $L-10-5-\chi\text{-arrow } \alpha \beta \mathfrak{T} \mathfrak{K} c a(\mathit{ArrVal}) =$
 $(\lambda v \in_0 L-10-5-N \alpha \beta \mathfrak{T} \mathfrak{K} c(\mathit{ObjMap})(\mathit{!}a). ntcf\text{-arrow } (L-10-5-\tau \mathfrak{T} \mathfrak{K} c v a))$
and $L-10-5-\chi\text{-arrow } \alpha \beta \mathfrak{T} \mathfrak{K} c a(\mathit{ArrDom}) = L-10-5-N \alpha \beta \mathfrak{T} \mathfrak{K} c(\mathit{ObjMap})(\mathit{!}a)$
and $L-10-5-\chi\text{-arrow } \alpha \beta \mathfrak{T} \mathfrak{K} c a(\mathit{ArrCod}) =$
 $cf\text{-Cone } \alpha \beta (\mathfrak{T} \circ_{CF} c \circ \mathit{!} \square_{CF} \mathfrak{K})(\mathit{ObjMap})(\mathit{!}a)$
unfolding $L-10-5-\chi\text{-arrow-def arr-field-simps}$
by ($simp\text{-all add: nat-omega-simps}$)

lemmas [$cat\text{-Kan-cs-simps}$] = $L-10-5-\chi\text{-arrow-components}(2,3)$

15.6.2 Arrow value

mk-VLambda $L-10-5-\chi\text{-arrow-components}(1)$

$|vsv L-10-5-\chi\text{-arrow-vs}[cat\text{-Kan-cs-intros}]|$
 $|vdomain L-10-5-\chi\text{-arrow-vdomain}|$
 $|app L-10-5-\chi\text{-arrow-app}|$

lemma $L-10-5-\chi\text{-arrow-vdomain}'[cat\text{-Kan-cs-simps}]$:

assumes $Z \beta$
and $\alpha \in_0 \beta$
and $\mathfrak{K} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{T} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{A}$
and $c \in_0 \mathfrak{C}(\mathit{Obj})$
and $a \in_0 \mathfrak{A}(\mathit{Obj})$
shows $\mathcal{D}_0 (L-10-5-\chi\text{-arrow } \alpha \beta \mathfrak{T} \mathfrak{K} c a(\mathit{ArrVal})) = Hom$
 $(cat\text{-FUNCT } \alpha \mathfrak{B} (cat\text{-Set } \alpha))$
 $(cf\text{-map } (Hom_{O.C\alpha} \mathfrak{C}(c, -) \circ_{CF} \mathfrak{K}))$
 $(cf\text{-map } (Hom_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} \mathfrak{T}))$

using $assms$

by

$($
 $cs\text{-concl}$
cs-simp: $cat\text{-cs-simps } cat\text{-Kan-cs-simps } L-10-5-\chi\text{-arrow-vdomain}$
cs-intro: $cat\text{-cs-intros}$
 $)$

lemma $L-10-5-\chi\text{-arrow-app}'[cat\text{-Kan-cs-simps}]$:

assumes $Z \beta$
and $\alpha \in_0 \beta$
and $\mathfrak{K} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{T} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{A}$
and $c \in_0 \mathfrak{C}(\mathit{Obj})$
and $v'\text{-def: } v' = ntcf\text{-arrow } v$
and $v : v :$
 $Hom_{O.C\alpha} \mathfrak{C}(c, -) \circ_{CF} \mathfrak{K} \mapsto_{CF} Hom_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} \mathfrak{T} : \mathfrak{B} \mapsto \mapsto_{C\alpha} cat\text{-Set } \alpha$
and $a : a \in_0 \mathfrak{A}(\mathit{Obj})$

shows

$L-10-5-\chi\text{-arrow } \alpha \beta \mathfrak{T} \mathfrak{K} c a(\mathit{ArrVal})(\mathit{!}v') =$
 $ntcf\text{-arrow } (L-10-5-\tau \mathfrak{T} \mathfrak{K} c v' a)$

using $assms$

by

$($

cs-concl **cs-shallow**
cs-simp: *cat-cs-simps cat-Kan-cs-simps L-10-5-χ-arrow-app*
cs-intro: *cat-cs-intros cat-FUNCT-cs-intros*
)

lemma *vτa-def*:

assumes $\mathfrak{K} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{T} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{A}$
and $c \in_{\circ} \mathfrak{C}(\text{Obj})$
and *vτa'-def*: $v\tau a' = \text{ntcf-arrow } v\tau a$
and *vτa*: $v\tau a :$
 $\text{Hom}_{O.C\alpha} \mathfrak{C}(c, -) \circ_{CF} \mathfrak{K} \mapsto_{CF} \text{Hom}_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} \mathfrak{T} :$
 $\mathfrak{B} \mapsto_{C\alpha} \text{cat-Set } \alpha$
and $a : a \in_{\circ} \mathfrak{A}(\text{Obj})$
shows $v\tau a = L-10-5-v \alpha \mathfrak{T} \mathfrak{K} c (\text{ntcf-arrow } (L-10-5-\tau \mathfrak{T} \mathfrak{K} c v\tau a' a)) a$
(is $\langle v\tau a = ?L-10-5-v (\text{ntcf-arrow } ?L-10-5-\tau) a \rangle$
)

proof-

interpret \mathfrak{K} : *is-functor* $\alpha \mathfrak{B} \mathfrak{C} \mathfrak{K}$ **by** (*rule assms(1)*)
interpret \mathfrak{T} : *is-functor* $\alpha \mathfrak{B} \mathfrak{A} \mathfrak{T}$ **by** (*rule assms(2)*)

interpret *vτa*: *is-ntcf*

$\alpha \mathfrak{B} \langle \text{cat-Set } \alpha \rangle \langle \text{Hom}_{O.C\alpha} \mathfrak{C}(c, -) \circ_{CF} \mathfrak{K} \rangle \langle \text{Hom}_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} \mathfrak{T} \rangle v\tau a$
by (*rule vτa*)

show *?thesis*

proof(*rule ntcf-eqI*)

show *vτa* :

$\text{Hom}_{O.C\alpha} \mathfrak{C}(c, -) \circ_{CF} \mathfrak{K} \mapsto_{CF} \text{Hom}_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} \mathfrak{T} : \mathfrak{B} \mapsto_{C\alpha} \text{cat-Set } \alpha$
by (*rule vτa*)

from *assms(1-3)* **a show**

$?L-10-5-v (\text{ntcf-arrow } ?L-10-5-\tau) a :$

$\text{Hom}_{O.C\alpha} \mathfrak{C}(c, -) \circ_{CF} \mathfrak{K} \mapsto_{CF} \text{Hom}_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} \mathfrak{T} : \mathfrak{B} \mapsto_{C\alpha} \text{cat-Set } \alpha$
by

(

cs-concl

cs-simp: *cat-Kan-cs-simps vτa'-def*

cs-intro: *cat-cs-intros cat-Kan-cs-intros*

)

have *dom-lhs*: $\mathcal{D}_{\circ} (v\tau a(\text{NTMap})) = \mathfrak{B}(\text{Obj})$

by (*cs-concl cs-shallow cs-simp*: *cat-cs-simps*)

have *dom-rhs*: $\mathcal{D}_{\circ} (?L-10-5-v (\text{ntcf-arrow } (?L-10-5-\tau)) a(\text{NTMap})) = \mathfrak{B}(\text{Obj})$

by (*cs-concl cs-shallow cs-simp*: *cat-Kan-cs-simps cs-intro*: *cat-cs-intros*)

show $v\tau a(\text{NTMap}) = ?L-10-5-v (\text{ntcf-arrow } ?L-10-5-\tau) a(\text{NTMap})$

proof(*rule vsv-eqI, unfold dom-lhs dom-rhs*)

fix b **assume** *prems*: $b \in_{\circ} \mathfrak{B}(\text{Obj})$

from *prems assms(3)* **a have** *lhs*: $v\tau a(\text{NTMap})(b) :$

$\text{Hom } \mathfrak{C} c (\mathfrak{K}(\text{ObjMap})(b)) \mapsto_{\text{cat-Set } \alpha} \text{Hom } \mathfrak{A} a (\mathfrak{T}(\text{ObjMap})(b))$

by

(

cs-concl cs-shallow

cs-simp: *cat-cs-simps cs-intro*: *cat-cs-intros cat-op-intros*

)

then have *dom-lhs*: $\mathcal{D}_{\circ} (v\tau a(\text{NTMap})(b)(\text{ArrVal})) = \text{Hom } \mathfrak{C} c (\mathfrak{K}(\text{ObjMap})(b))$

by (*cs-concl cs-shallow cs-simp*: *cat-cs-simps*)

from *prems assms(3)* **a have** *rhs*:

$L-10-5-v\text{-arrow } \mathfrak{T} \mathfrak{K} c (\text{ntcf-arrow } ?L-10-5-\tau) a b :$

$\text{Hom } \mathfrak{C} c (\mathfrak{K}(\text{ObjMap})(b)) \mapsto_{\text{cat-Set } \alpha} \text{Hom } \mathfrak{A} a (\mathfrak{T}(\text{ObjMap})(b))$

```

unfolding  $v\tau a'$ -def
by
  (
    cs-concl cs-shallow
    cs-simp: cat-Kan-cs-simps
    cs-intro: cat-Kan-cs-intros cat-cs-intros
  )

then have dom-rhs:
   $\mathcal{D}_o (L-10-5-v\text{-arrow } \mathfrak{T} \mathfrak{K} c (ntcf\text{-arrow } ?L-10-5\text{-}\tau) a b(\text{ArrVal})) =$ 
   $\text{Hom } \mathfrak{C} c (\mathfrak{K}(\text{ObjMap})(b))$ 
by (cs-concl cs-shallow cs-simp: cat-cs-simps)
have [cat-cs-simps]:
   $v\tau a(\text{NTMap})(b) = L-10-5-v\text{-arrow } \mathfrak{T} \mathfrak{K} c (ntcf\text{-arrow } ?L-10-5\text{-}\tau) a b$ 
proof(rule arr-Set-eqI)
from lhs show arr-Set-lhs: arr-Set  $\alpha (v\tau a(\text{NTMap})(b))$ 
by (auto dest: cat-Set-is-arrD(1))
from rhs show arr-Set-rhs:
   $\text{arr-Set } \alpha (L-10-5-v\text{-arrow } \mathfrak{T} \mathfrak{K} c (ntcf\text{-arrow } (?L-10-5\text{-}\tau)) a b)$ 
by (auto dest: cat-Set-is-arrD(1))
show  $v\tau a(\text{NTMap})(b)(\text{ArrVal}) =$ 
   $L-10-5-v\text{-arrow } \mathfrak{T} \mathfrak{K} c (ntcf\text{-arrow } ?L-10-5\text{-}\tau) a b(\text{ArrVal})$ 
proof(rule vsu-eqI, unfold dom-lhs dom-rhs in-Hom-iff)
fix  $f$  assume  $f : c \mapsto_{\mathfrak{C}} \mathfrak{K}(\text{ObjMap})(b)$ 
with assms prems show
   $v\tau a(\text{NTMap})(b)(\text{ArrVal})(f) =$ 
   $L-10-5-v\text{-arrow } \mathfrak{T} \mathfrak{K} c (ntcf\text{-arrow } ?L-10-5\text{-}\tau) a b(\text{ArrVal})(f)$ 
unfolding  $v\tau a'$ -def
by
  (
    cs-concl cs-shallow
    cs-simp:
      cat-Kan-cs-simps cat-FUNCT-cs-simps L-10-5-v-arrow-ArrVal-app
    cs-intro: cat-cs-intros cat-comma-cs-intros
  )
qed (use arr-Set-lhs arr-Set-rhs in auto)
qed (use lhs rhs in <cs-concl cs-shallow cs-simp: cat-cs-simps>+)

from prems show
   $v\tau a(\text{NTMap})(b) = L-10-5-v \alpha \mathfrak{T} \mathfrak{K} c (ntcf\text{-arrow } ?L-10-5\text{-}\tau) a(\text{NTMap})(b)$ 
by
  (
    cs-concl cs-shallow
    cs-simp: cat-cs-simps cat-Kan-cs-simps cs-intro: cat-cs-intros
  )

qed (cs-concl cs-intro: cat-cs-intros cat-Kan-cs-intros V-cs-intros)+

qed simp-all

qed

```

15.7 Lemma X.5: $L-10-5\text{-}\chi'$ -arrow

15.7.1 Definition and elementary properties

definition $L-10-5\text{-}\chi'$ -arrow :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
 where $L-10-5\text{-}\chi'$ -arrow $\alpha \beta \mathfrak{T} \mathfrak{K} c a =$

[
 (
 $\lambda\tau \in_{\circ} \text{cf-Cone } \alpha \beta (\mathfrak{T} \circ_{CF} c \text{ o}\sqcap_{CF} \mathfrak{K})(\text{ObjMap})(|a|)$.
 $\text{ntcf-arrow } (L-10-5-v \alpha \mathfrak{T} \mathfrak{K} c \tau a)$
),
 $\text{cf-Cone } \alpha \beta (\mathfrak{T} \circ_{CF} c \text{ o}\sqcap_{CF} \mathfrak{K})(\text{ObjMap})(|a|)$,
 $L-10-5-N \alpha \beta \mathfrak{T} \mathfrak{K} c(\text{ObjMap})(|a|)$
]_o

Components.

lemma $L-10-5-\chi'$ -arrow-components:

shows $L-10-5-\chi'$ -arrow $\alpha \beta \mathfrak{T} \mathfrak{K} c a(\text{ArrVal}) =$

(
 $\lambda\tau \in_{\circ} \text{cf-Cone } \alpha \beta (\mathfrak{T} \circ_{CF} c \text{ o}\sqcap_{CF} \mathfrak{K})(\text{ObjMap})(|a|)$.
 $\text{ntcf-arrow } (L-10-5-v \alpha \mathfrak{T} \mathfrak{K} c \tau a)$
)

and [cat-Kan-cs-simps]: $L-10-5-\chi'$ -arrow $\alpha \beta \mathfrak{T} \mathfrak{K} c a(\text{ArrDom}) =$
 $\text{cf-Cone } \alpha \beta (\mathfrak{T} \circ_{CF} c \text{ o}\sqcap_{CF} \mathfrak{K})(\text{ObjMap})(|a|)$

and [cat-Kan-cs-simps]: $L-10-5-\chi'$ -arrow $\alpha \beta \mathfrak{T} \mathfrak{K} c a(\text{ArrCod}) =$
 $L-10-5-N \alpha \beta \mathfrak{T} \mathfrak{K} c(\text{ObjMap})(|a|)$

unfolding $L-10-5-\chi'$ -arrow-def arr-field-simps **by** ($\text{simp-all add: nat-omega-simps}$)

15.7.2 Arrow value

mk-VLambda $L-10-5-\chi'$ -arrow-components(1)

$|\text{vsu } L-10-5-\chi'$ -arrow-ArrVal-vsuv[cat-Kan-cs-intros]|

$|\text{vdomain } L-10-5-\chi'$ -arrow-ArrVal-vdomain|

$|\text{app } L-10-5-\chi'$ -arrow-ArrVal-app|

lemma $L-10-5-\chi'$ -arrow-ArrVal-vdomain'[cat-Kan-cs-simps]:

assumes $Z \beta$

and $\alpha \in_{\circ} \beta$

and $\tau : \tau : a \langle_{CF} \text{cone } \mathfrak{T} \circ_{CF} c \text{ o}\sqcap_{CF} \mathfrak{K} : c \downarrow_{CF} \mathfrak{K} \mapsto \mapsto_{C\alpha} \mathfrak{A}$

and $a : a \in_{\circ} \mathfrak{A}(\text{Obj})$

shows $\mathcal{D}_o (L-10-5-\chi'$ -arrow $\alpha \beta \mathfrak{T} \mathfrak{K} c a(\text{ArrVal})) = \text{Hom}$

($\text{cat-FUNCT } \alpha (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A}$)

($\text{cf-map } (\text{cf-const } (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} a)$)

($\text{cf-map } (\mathfrak{T} \circ_{CF} c \text{ o}\sqcap_{CF} \mathfrak{K})$)

proof-

interpret $\beta : Z \beta$ **by** ($\text{rule assms}(1)$)

interpret $\tau : \text{is-cat-cone } \alpha a \langle_{CF} \mathfrak{K} \rangle \mathfrak{A} \langle \mathfrak{T} \circ_{CF} c \text{ o}\sqcap_{CF} \mathfrak{K} \rangle \tau$

by ($\text{rule assms}(3)$)

from $\text{assms}(1,2,4)$ **show** $?thesis$

by

(
 $\text{cs-concl } \text{cs-shallow}$
 $\text{cs-simp: } \text{cat-cs-simps } L-10-5-\chi'$ -arrow-ArrVal-vdomain
 $\text{cs-intro: } \text{cat-cs-intros}$
)

qed

lemma $L-10-5-\chi'$ -arrow-ArrVal-app'[cat-cs-simps]:

assumes $Z \beta$

and $\alpha \in_{\circ} \beta$

and $\tau' \text{-def: } \tau' = \text{ntcf-arrow } \tau$

and $\tau : \tau : a \langle_{CF} \text{cone } \mathfrak{T} \circ_{CF} c \text{ o}\sqcap_{CF} \mathfrak{K} : c \downarrow_{CF} \mathfrak{K} \mapsto \mapsto_{C\alpha} \mathfrak{A}$

and $a : a \in_{\circ} \mathfrak{A}(\text{Obj})$

shows $L-10-5-\chi'$ -arrow $\alpha \beta \mathfrak{T} \mathfrak{K} c a(\text{ArrVal})(\tau') =$

ntcf-arrow ($L-10-5-v \alpha \mathfrak{T} \mathfrak{K} c \tau' a$)

proof-

interpret $\beta: \mathcal{Z} \beta$ **by** (*rule assms(1)*)

interpret $\tau: is-cat-cone \alpha a \langle c \downarrow_{CF} \mathfrak{K} \rangle \mathfrak{A} \langle \mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} \rangle \tau$

by (*rule assms(4)*)

from *assms(2,5)* **have** $\tau' \in_o cf-Cone \alpha \beta (\mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K})(\mathit{ObjMap})(a)$

unfolding τ' -def

by

(
cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-FUNCT-cs-intros cat-cs-intros*
)

then show

$L-10-5-\chi'$ -arrow $\alpha \beta \mathfrak{T} \mathfrak{K} c a(\mathit{ArrVal})(\tau') =$

$ntcf-arrow (L-10-5-v \alpha \mathfrak{T} \mathfrak{K} c \tau' a)$

unfolding $L-10-5-\chi'$ -arrow-components **by** *auto*

qed

15.7.3 $L-10-5-\chi'$ -arrow is an isomorphism in the category *Set*

lemma $L-10-5-\chi'$ -arrow-is-iso-arr:

assumes $\mathcal{Z} \beta$

and $\alpha \in_o \beta$

and $\mathfrak{K} : \mathfrak{B} \mapsto_{CF} \mathcal{C}$

and $\mathfrak{T} : \mathfrak{B} \mapsto_{CF} \mathfrak{A}$

and $c \in_o \mathcal{C}(\mathit{Obj})$

and $a \in_o \mathfrak{A}(\mathit{Obj})$

shows $L-10-5-\chi'$ -arrow $\alpha \beta \mathfrak{T} \mathfrak{K} c a :$

$cf-Cone \alpha \beta (\mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K})(\mathit{ObjMap})(a) \mapsto_{isocat-Set} \beta$

$L-10-5-N \alpha \beta \mathfrak{T} \mathfrak{K} c(\mathit{ObjMap})(a)$

(

is

<

? $L-10-5-\chi'$ -arrow :

$cf-Cone \alpha \beta (\mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K})(\mathit{ObjMap})(a) \mapsto_{isocat-Set} \beta$

? $L-10-5-N(\mathit{ObjMap})(a)$

>

)

proof-

let ? $FUNCT = \langle \lambda \mathfrak{A}. cat-FUNCT \alpha \mathfrak{A} (cat-Set \alpha) \rangle$

let ? $c\mathfrak{K}\mathfrak{A} = \langle cat-FUNCT \alpha (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} \rangle$

let ? $H\mathcal{C} = \langle \lambda c. Hom_{O.C\alpha} \mathcal{C}(c, -) \rangle$

let ? $H\mathfrak{A} = \langle \lambda c. Hom_{O.C\alpha} \mathfrak{A}(a, -) \rangle$

from *assms(1,2)* **interpret** $\beta: \mathcal{Z} \beta$ **by** *simp*

interpret $\mathfrak{K}: is-functor \alpha \mathfrak{B} \mathcal{C} \mathfrak{K}$ **by** (*rule assms(3)*)

interpret $\mathfrak{T}: is-functor \alpha \mathfrak{B} \mathfrak{A} \mathfrak{T}$ **by** (*rule assms(4)*)

from $\mathfrak{K}.empty-is-zet$ *assms* **interpret** $c\mathfrak{K}: category \alpha \langle c \downarrow_{CF} \mathfrak{K} \rangle$

by (*cs-concl cs-shallow cs-intro: cat-comma-cs-intros*)

from *assms(2,6)* **interpret** $c\mathfrak{K}\mathfrak{A}: category \beta ?c\mathfrak{K}\mathfrak{A}$

by

(
cs-concl cs-intro:
cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
)

)

from $\mathfrak{K}.vempty\text{-is-zet}\text{ assms}$ **interpret** Πc :
is-functor $\alpha \langle c \downarrow_{CF} \mathfrak{K} \rangle \mathfrak{B} \langle c \circ \sqcap_{CF} \mathfrak{K} \rangle$
by (*cs-concl cs-shallow cs-intro: cat-comma-cs-intros*)

from *assms(2)* **interpret** *FUNCT- \mathfrak{A}* : *tiny-category* $\beta \langle ?FUNCT \mathfrak{A} \rangle$
by (*cs-concl cs-intro: cat-cs-intros cat-FUNCT-cs-intros*)

from *assms(2)* **interpret** *FUNCT- \mathfrak{B}* : *tiny-category* $\beta \langle ?FUNCT \mathfrak{B} \rangle$
by (*cs-concl cs-intro: cat-cs-intros cat-FUNCT-cs-intros*)

from *assms(2)* **interpret** *FUNCT- \mathfrak{C}* : *tiny-category* $\beta \langle ?FUNCT \mathfrak{C} \rangle$
by (*cs-concl cs-intro: cat-cs-intros cat-FUNCT-cs-intros*)

have $\mathfrak{T}\Pi: \mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} : c \downarrow_{CF} \mathfrak{K} \mapsto \rightarrow_{C\alpha} \mathfrak{A}$
by (*cs-concl cs-intro: cat-cs-intros*)

from *assms(5,6)* **have** [*cat-cs-simps*]:
cf-of-cf-map ($c \downarrow_{CF} \mathfrak{K}$) \mathfrak{A} (*cf-map* (*cf-const* ($c \downarrow_{CF} \mathfrak{K}$) \mathfrak{A} a)) =
cf-const ($c \downarrow_{CF} \mathfrak{K}$) \mathfrak{A} a
cf-of-cf-map ($c \downarrow_{CF} \mathfrak{K}$) \mathfrak{A} (*cf-map* ($\mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K}$)) = $\mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K}$
cf-of-cf-map \mathfrak{B} (*cat-Set* α) (*cf-map* (*Hom* _{$O.C\alpha$} $\mathfrak{C}(c, -) \circ_{CF} \mathfrak{K}$)) =
Hom _{$O.C\alpha$} $\mathfrak{C}(c, -) \circ_{CF} \mathfrak{K}$
cf-of-cf-map \mathfrak{B} (*cat-Set* α) (*cf-map* (*Hom* _{$O.C\alpha$} $\mathfrak{A}(a, -) \circ_{CF} \mathfrak{T}$)) =
Hom _{$O.C\alpha$} $\mathfrak{A}(a, -) \circ_{CF} \mathfrak{T}$
by (*cs-concl cs-simp: cat-FUNCT-cs-simps cs-intro: cat-cs-intros*)+

note *cf-Cone-ObjMap-app* = *is-functor.cf-Cone-ObjMap-app*[*OF* $\mathfrak{T}\Pi$ *assms(1,2,6)*]

show *?thesis*

proof

(
intro cat-Set-is-iso-arrI cat-Set-is-arrI arr-SetI,
unfold L-10-5- χ' -arrow-components(3) cf-Cone-ObjMap-app
)
show *vfsequence ?L-10-5- χ' -arrow*
unfolding *L-10-5- χ' -arrow-def* **by** *auto*

show *χ' -arrow-ArrVal-vsuv: vsuv (?L-10-5- χ' -arrow(|ArrVal|))*
unfolding *L-10-5- χ' -arrow-components* **by** *auto*

show *vcard ?L-10-5- χ' -arrow = 3 \mathbb{N}*
unfolding *L-10-5- χ' -arrow-def* **by** (*simp add: nat-omega-simps*)

show [*cat-cs-simps*]:
 \mathcal{D}_\circ (*?L-10-5- χ' -arrow(|ArrVal|)*) = *?L-10-5- χ' -arrow(|ArrDom|)*
unfolding *L-10-5- χ' -arrow-components* **by** *simp*

show *vrange- χ' -arrow-vsubset-N''*:
 \mathcal{R}_\circ (*?L-10-5- χ' -arrow(|ArrVal|)*) \subseteq_\circ *?L-10-5-N(|ObjMap|)(|a|)*
unfolding *L-10-5- χ' -arrow-components*

proof(*rule vrange-VLambda-vsubset*)
fix τ **assume** *prems: $\tau \in_\circ$ cf-Cone $\alpha \beta$ ($\mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K}$)(|ObjMap|)(|a|)*
from *this assms c \mathfrak{K} - \mathfrak{A} .category-axioms* **have** τ -*is-arr*:
 $\tau : \text{cf-map } (\text{cf-const } (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} a) \mapsto \rightarrow_{c\mathfrak{K}\text{-}\mathfrak{A}} \text{cf-map } (\mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K})$
by
(
cs-prems
cs-simp: *cat-cs-simps cat-Kan-cs-simps cat-FUNCT-components(1)*
cs-intro: *cat-cs-intros*
)
note $\tau = \text{cat-FUNCT-is-arrD}(1,2)$ [*OF* τ -*is-arr*, *unfolded cat-cs-simps*]
have *cf-of-cf-map* ($c \downarrow_{CF} \mathfrak{K}$) \mathfrak{A} (*cf-map* ($\mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K}$)) = $\mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K}$
by (*cs-concl cs-simp: cat-FUNCT-cs-simps cs-intro: cat-cs-intros*)

from *prems* *assms* $\tau(1)$ **show**
 $ntcf\text{-arrow } (L\text{-}10\text{-}5\text{-}v \ \alpha \ \mathfrak{T} \ \mathfrak{K} \ c \ \tau \ a) \in_{\circ} \ ?L\text{-}10\text{-}5\text{-}N(\text{ObjMap})(a)$
by (*subst* $\tau(2)$)
(*cs-concl*
cs-simp: *cat-cs-simps* *cat-Kan-cs-simps*
cs-intro:
is-cat-coneI *cat-cs-intros* *cat-Kan-cs-intros* *cat-FUNCT-cs-intros*
))
qed

show $\mathcal{R}_{\circ} (\ ?L\text{-}10\text{-}5\text{-}\chi'\text{-arrow}(\text{ArrVal})) = \ ?L\text{-}10\text{-}5\text{-}N(\text{ObjMap})(a)$
proof
(*intro* *vsubset-antisym*[*OF* *vrange- χ' -arrow-vsubset- N'*],
intro *vsubsetI*
))
fix $v\tau a$ **assume** $v\tau a \in_{\circ} \ ?L\text{-}10\text{-}5\text{-}N(\text{ObjMap})(a)$
from *this* *assms* **have** $v\tau a$:
 $v\tau a : cf\text{-map } (\ ?H\text{-}\mathfrak{C} \ c \ \circ_{CF} \ \mathfrak{K}) \mapsto_{?FUNCT} \mathfrak{B} \ cf\text{-map } (\ ?H\text{-}\mathfrak{A} \ a \ \circ_{CF} \ \mathfrak{T})$
by
(*cs-prems*
cs-simp: *cat-cs-simps* *cat-Kan-cs-simps* **cs-intro:** *cat-cs-intros*
))
note $v\tau a = \text{cat-FUNCT-is-arrD}[\text{OF } \text{this}, \text{ unfolded } \text{cat-cs-simps}]$
interpret τ :
is-cat-cone $\alpha \ a \ \langle c \ \downarrow_{CF} \ \mathfrak{K} \rangle \ \mathfrak{A} \ \langle \mathfrak{T} \ \circ_{CF} \ c \ \circ \square_{CF} \ \mathfrak{K} \rangle \ \langle L\text{-}10\text{-}5\text{-}\tau \ \mathfrak{T} \ \mathfrak{K} \ c \ v\tau a \ a \rangle$
by (*rule* *L-10-5- τ -is-cat-cone*[*OF* *assms*(3,4,5) $v\tau a$ (2,1) *assms*(6)])

show $v\tau a \in_{\circ} \ \mathcal{R}_{\circ} (\ ?L\text{-}10\text{-}5\text{-}\chi'\text{-arrow}(\text{ArrVal}))$
proof(*rule* *vsv.vsv-vimageI2'*)
show *vsv* ($\ ?L\text{-}10\text{-}5\text{-}\chi'\text{-arrow}(\text{ArrVal}))$ **by** (*rule* $\chi'\text{-arrow-ArrVal-vs}$)
from τ .*is-cat-cone-axioms* *assms* **show**
 $ntcf\text{-arrow } (L\text{-}10\text{-}5\text{-}\tau \ \mathfrak{T} \ \mathfrak{K} \ c \ v\tau a \ a) \in_{\circ} \ \mathcal{D}_{\circ} (\ ?L\text{-}10\text{-}5\text{-}\chi'\text{-arrow}(\text{ArrVal}))$
by
(*cs-concl*
cs-simp: *cat-Kan-cs-simps*
cs-intro: *cat-cs-intros* *cat-FUNCT-cs-intros*
))
from *assms* $v\tau a(1,2)$ **show**
 $v\tau a = \ ?L\text{-}10\text{-}5\text{-}\chi'\text{-arrow}(\text{ArrVal})(ntcf\text{-arrow } (L\text{-}10\text{-}5\text{-}\tau \ \mathfrak{T} \ \mathfrak{K} \ c \ v\tau a \ a))$
by
(*subst* $v\tau a(2)$,
cs-concl-step $v\tau a\text{-def}[\text{OF } \text{assms}(3,4,5) \ v\tau a(2,1) \ \text{assms}(6)]$
))
(*cs-concl* **cs-shallow** **cs-simp:** *cat-cs-simps* **cs-intro:** *cat-cs-intros*)
qed
qed

from *assms* **show** $\ ?L\text{-}10\text{-}5\text{-}\chi'\text{-arrow}(\text{ArrDom}) \in_{\circ} \ Vset \ \beta$
by
(*cs-concl*
cs-simp: *cat-Kan-cs-simps* *cat-FUNCT-components*(1) *cf-Cone-ObjMap-app*
)

cs-intro: *cat-cs-intros cat-FUNCT-cs-intros c \mathfrak{K} - \mathfrak{A} .cat-Hom-in-Vset*
)

with *assms(2)* **have** *?L-10-5- χ' -arrow(ArrDom) \in_0 Vset β*
by (*meson Vset-in-mono Vset-trans*)

from *assms* **show** *?L-10-5-N(ObjMap)(a) \in_0 Vset β*
by
(

cs-concl
cs-simp: *cat-cs-simps cat-Kan-cs-simps cat-FUNCT-cs-simps*
cs-intro: *cat-cs-intros FUNCT- \mathfrak{B} .cat-Hom-in-Vset cat-FUNCT-cs-intros*
)

show *dom- χ' -arrow: \mathcal{D}_0 (?L-10-5- χ' -arrow(ArrVal)) =*
Hom ?c \mathfrak{K} - \mathfrak{A} (cf-map (cf-const (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} a)) (cf-map ($\mathfrak{T} \circ_{CF}$ c $O\sqcap_{CF}$ \mathfrak{K}))
unfolding *L-10-5- χ' -arrow-components cf-Cone-ObjMap-app* **by** *simp*

show *?L-10-5- χ' -arrow(ArrDom) =*
Hom ?c \mathfrak{K} - \mathfrak{A} (cf-map (cf-const (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} a)) (cf-map ($\mathfrak{T} \circ_{CF}$ c $O\sqcap_{CF}$ \mathfrak{K}))
unfolding *L-10-5- χ' -arrow-components cf-Cone-ObjMap-app* **by** *simp*

show *v11 (?L-10-5- χ' -arrow(ArrVal))*
proof(*rule vsu.vsu-valeq-v11I, unfold dom- χ' -arrow in-Hom-iff*)

fix $\tau' \tau''$ **assume** *prems:*
 $\tau' : cf\text{-map} (cf\text{-const} (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} a) \mapsto_{?c\mathfrak{K}\text{-}\mathfrak{A}} cf\text{-map} (\mathfrak{T} \circ_{CF} c \ O\sqcap_{CF} \mathfrak{K})$
 $\tau'' : cf\text{-map} (cf\text{-const} (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} a) \mapsto_{?c\mathfrak{K}\text{-}\mathfrak{A}} cf\text{-map} (\mathfrak{T} \circ_{CF} c \ O\sqcap_{CF} \mathfrak{K})$
?L-10-5- χ' -arrow(ArrVal)(τ') = ?L-10-5- χ' -arrow(ArrVal)(τ'')

note $\tau' = cat\text{-FUNCT-is-arrD}[OF \text{prems}(1), \text{unfolded cat-cs-simps}]$
and $\tau'' = cat\text{-FUNCT-is-arrD}[OF \text{prems}(2), \text{unfolded cat-cs-simps}]$

interpret τ' : *is-cat-cone*
 $\alpha a \langle c \downarrow_{CF} \mathfrak{K} \rangle \mathfrak{A} \langle \mathfrak{T} \circ_{CF} c \ O\sqcap_{CF} \mathfrak{K} \rangle \langle ntcf\text{-of-ntcf-arrow} (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} \tau' \rangle$
by (*rule is-cat-coneI[OF $\tau'(1)$ assms(6)]*)

interpret τ'' : *is-cat-cone*
 $\alpha a \langle c \downarrow_{CF} \mathfrak{K} \rangle \mathfrak{A} \langle \mathfrak{T} \circ_{CF} c \ O\sqcap_{CF} \mathfrak{K} \rangle \langle ntcf\text{-of-ntcf-arrow} (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} \tau'' \rangle$
by (*rule is-cat-coneI[OF $\tau''(1)$ assms(6)]*)

have $\tau'\tau'$: *ntcf-arrow (ntcf-of-ntcf-arrow (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} τ') = τ'*
by (*subst (2) $\tau'(2)$ (cs-concl cs-shallow cs-simp: cat-FUNCT-cs-simps)*)

have $\tau''\tau''$: *ntcf-arrow (ntcf-of-ntcf-arrow (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} τ'') = τ''*
by (*subst (2) $\tau''(2)$ (cs-concl cs-shallow cs-simp: cat-FUNCT-cs-simps)*)

from *prems(3) $\tau'(1) \tau''(1)$ assms* **have**
L-10-5-v α $\mathfrak{T} \mathfrak{K} c \tau' a = L-10-5-v \alpha \mathfrak{T} \mathfrak{K} c \tau'' a$
by (*subst (asm) $\tau'(2)$, use nothing in $\langle subst (asm) \tau''(2) \rangle$*)
(

cs-prems cs-shallow
cs-simp: *$\tau'\tau' \tau''\tau''$ cat-cs-simps cat-FUNCT-cs-simps*
cs-intro: *cat-lim-cs-intros cat-Kan-cs-intros cat-cs-intros*
)

from this **have** *$v\tau'a-v\tau''a$:*
L-10-5-v α $\mathfrak{T} \mathfrak{K} c \tau' a (\NTMap)(b) (\ArrVal)(f) =$
L-10-5-v α $\mathfrak{T} \mathfrak{K} c \tau'' a (\NTMap)(b) (\ArrVal)(f)$
if $b \in_0 \mathfrak{B}(\text{Obj})$ **and** $f : c \mapsto_{\mathfrak{C}} (\mathfrak{K}(\text{ObjMap})(b))$ **for** $b f$
by *simp*

have [*cat-cs-simps*]: *$\tau'(\NTMap)(0, b, f)_{\bullet} = \tau''(\NTMap)(0, b, f)_{\bullet}$*
if $b \in_0 \mathfrak{B}(\text{Obj})$ **and** $f : c \mapsto_{\mathfrak{C}} (\mathfrak{K}(\text{ObjMap})(b))$ **for** $b f$
using *$v\tau'a-v\tau''a$ [OF that] that*
by
(

cs-prems cs-shallow
cs-simp: *cat-Kan-cs-simps L-10-5-v-arrow-ArrVal-app*
cs-intro: *cat-cs-intros*
)

have

$ntcf\text{-of-}ntcf\text{-arrow } (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} \tau' =$
 $ntcf\text{-of-}ntcf\text{-arrow } (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} \tau''$

proof(rule *ntcf-eqI*)

show $ntcf\text{-of-}ntcf\text{-arrow } (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} \tau'$:

$cf\text{-const } (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} a \mapsto_{CF} \mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} : c \downarrow_{CF} \mathfrak{K} \mapsto_{CF} C\alpha \mathfrak{A}$

by (rule τ' .*is-ntcf-axioms*)

then have *dom-lhs*:

$\mathcal{D}_\circ (ntcf\text{-of-}ntcf\text{-arrow } (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} \tau'(\mathcal{NTMap})) = c \downarrow_{CF} \mathfrak{K}(\mathcal{Obj})$

by (*cs-concl cs-shallow cs-simp: cat-cs-simps*)

show $ntcf\text{-of-}ntcf\text{-arrow } (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} \tau''$:

$cf\text{-const } (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} a \mapsto_{CF} \mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} : c \downarrow_{CF} \mathfrak{K} \mapsto_{CF} C\alpha \mathfrak{A}$

by (rule τ'' .*is-ntcf-axioms*)

then have *dom-rhs*:

$\mathcal{D}_\circ (ntcf\text{-of-}ntcf\text{-arrow } (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} \tau''(\mathcal{NTMap})) = c \downarrow_{CF} \mathfrak{K}(\mathcal{Obj})$

by (*cs-concl cs-shallow cs-simp: cat-cs-simps*)

show

$ntcf\text{-of-}ntcf\text{-arrow } (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} \tau'(\mathcal{NTMap}) =$
 $ntcf\text{-of-}ntcf\text{-arrow } (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} \tau''(\mathcal{NTMap})$

proof(rule *vsv-eqI*, *unfold dom-lhs dom-rhs*)

fix A **assume** $A \in_\circ c \downarrow_{CF} \mathfrak{K}(\mathcal{Obj})$

with *assms(5)* **obtain** $b f$

where $A\text{-def}$: $A = [\emptyset, b, f]_\circ$

and b : $b \in_\circ \mathfrak{B}(\mathcal{Obj})$

and f : $f : c \mapsto_{\mathfrak{C}} \mathfrak{K}(\mathcal{ObjMap})(b)$

by *auto*

from $b f$ **show**

$ntcf\text{-of-}ntcf\text{-arrow } (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} \tau'(\mathcal{NTMap})(A) =$
 $ntcf\text{-of-}ntcf\text{-arrow } (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} \tau''(\mathcal{NTMap})(A)$

unfolding $A\text{-def}$

by (*cs-concl cs-simp: cat-cs-simps cat-map-extra-cs-simps*)

qed (*cs-concl cs-shallow cs-intro: V-cs-intros*)⁺

qed *simp-all*

then show $\tau' = \tau''$

proof(rule *inj-onD*[*OF bij-betw-imp-inj-on* [*OF bij-betw-ntcf-of-ntcf-arrow*]])

show $\tau' \in_\circ ntcf\text{-arrows } \alpha (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A}$

by (*subst* $\tau'(2)$)

(

cs-concl cs-intro:

cat-lim-cs-intros cat-cs-intros cat-FUNCT-cs-intros

)

show $\tau'' \in_\circ ntcf\text{-arrows } \alpha (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A}$

by (*subst* $\tau''(2)$)

(

cs-concl cs-intro:

cat-lim-cs-intros cat-cs-intros cat-FUNCT-cs-intros

)

qed

qed (*cs-concl cs-shallow cs-intro: cat-Kan-cs-intros*)

qed *auto*

qed

lemma *L-10-5- χ' -arrow-is-iso-arr'* [*cat-Kan-cs-intros*]:

assumes $Z \beta$

and $\alpha \in_\circ \beta$

and $\mathfrak{K} : \mathfrak{B} \mapsto_{CF} C\alpha \mathfrak{C}$

and $\mathfrak{T} : \mathfrak{B} \mapsto_{CF} C\alpha \mathfrak{A}$

and $c \in_{\circ} \mathfrak{C}(\mathit{Obj})$
and $a \in_{\circ} \mathfrak{A}(\mathit{Obj})$
and $A = \mathit{cf-Cone} \alpha \beta (\mathfrak{T} \circ_{CF} c \circ \mathit{O}\mathfrak{P}_{CF} \mathfrak{K})(\mathit{ObjMap})(\mathit{a})$
and $B = \mathit{L-10-5-N} \alpha \beta \mathfrak{T} \mathfrak{K} c(\mathit{ObjMap})(\mathit{a})$
and $\mathfrak{C}' = \mathit{cat-Set} \beta$
shows $\mathit{L-10-5-}\chi'\text{-arrow} \alpha \beta \mathfrak{T} \mathfrak{K} c a : A \mapsto_{\mathit{iso}} \mathfrak{C}' B$
using $\mathit{assms}(1-6)$
unfolding $\mathit{assms}(7-9)$
by (rule $\mathit{L-10-5-}\chi'\text{-arrow-is-iso-arr}$)

lemma $\mathit{L-10-5-}\chi'\text{-arrow-is-arr}$:

assumes $\mathfrak{Z} \beta$
and $\alpha \in_{\circ} \beta$
and $\mathfrak{K} : \mathfrak{B} \mapsto \mathfrak{C} \alpha \mathfrak{C}$
and $\mathfrak{T} : \mathfrak{B} \mapsto \mathfrak{C} \alpha \mathfrak{A}$
and $c \in_{\circ} \mathfrak{C}(\mathit{Obj})$
and $a \in_{\circ} \mathfrak{A}(\mathit{Obj})$
shows $\mathit{L-10-5-}\chi'\text{-arrow} \alpha \beta \mathfrak{T} \mathfrak{K} c a :$
 $\mathit{cf-Cone} \alpha \beta (\mathfrak{T} \circ_{CF} c \circ \mathit{O}\mathfrak{P}_{CF} \mathfrak{K})(\mathit{ObjMap})(\mathit{a}) \mapsto_{\mathit{cat-Set}} \beta$
 $\mathit{L-10-5-N} \alpha \beta \mathfrak{T} \mathfrak{K} c(\mathit{ObjMap})(\mathit{a})$
by
(

rule $\mathit{cat-Set-is-iso-arrD}(1)[$
 $\mathit{OF} \mathit{L-10-5-}\chi'\text{-arrow-is-iso-arr}[\mathit{OF} \mathit{assms}(1-6)]$
]
)

lemma $\mathit{L-10-5-}\chi'\text{-arrow-is-arr}'[\mathit{cat-Kan-cs-intros}]$:

assumes $\mathfrak{Z} \beta$
and $\alpha \in_{\circ} \beta$
and $\mathfrak{K} : \mathfrak{B} \mapsto \mathfrak{C} \alpha \mathfrak{C}$
and $\mathfrak{T} : \mathfrak{B} \mapsto \mathfrak{C} \alpha \mathfrak{A}$
and $c \in_{\circ} \mathfrak{C}(\mathit{Obj})$
and $a \in_{\circ} \mathfrak{A}(\mathit{Obj})$
and $A = \mathit{cf-Cone} \alpha \beta (\mathfrak{T} \circ_{CF} c \circ \mathit{O}\mathfrak{P}_{CF} \mathfrak{K})(\mathit{ObjMap})(\mathit{a})$
and $B = \mathit{L-10-5-N} \alpha \beta \mathfrak{T} \mathfrak{K} c(\mathit{ObjMap})(\mathit{a})$
and $\mathfrak{C}' = \mathit{cat-Set} \beta$
shows $\mathit{L-10-5-}\chi'\text{-arrow} \alpha \beta \mathfrak{T} \mathfrak{K} c a : A \mapsto_{\mathfrak{C}'} B$
using $\mathit{assms}(1-6)$ **unfolding** $\mathit{assms}(7-9)$ **by** (rule $\mathit{L-10-5-}\chi'\text{-arrow-is-arr}$)

15.8 Lemma X.5: $\mathit{L-10-5-}\chi$

15.8.1 Definition and elementary properties

definition $\mathit{L-10-5-}\chi :: V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$

where $\mathit{L-10-5-}\chi \alpha \beta \mathfrak{T} \mathfrak{K} c =$
[
 $(\lambda a \in_{\circ} \mathfrak{T}(\mathit{HomCod})(\mathit{Obj}). \mathit{L-10-5-}\chi'\text{-arrow} \alpha \beta \mathfrak{T} \mathfrak{K} c a),$
 $\mathit{cf-Cone} \alpha \beta (\mathfrak{T} \circ_{CF} c \circ \mathit{O}\mathfrak{P}_{CF} \mathfrak{K}),$
 $\mathit{L-10-5-N} \alpha \beta \mathfrak{T} \mathfrak{K} c,$
 $\mathit{op-cat} (\mathfrak{T}(\mathit{HomCod})),$
 $\mathit{cat-Set} \beta$
]_o

Components.

lemma $\mathit{L-10-5-}\chi\text{-components}$:

shows $\mathit{L-10-5-}\chi \alpha \beta \mathfrak{T} \mathfrak{K} c(\mathit{NTMap}) =$
 $(\lambda a \in_{\circ} \mathfrak{T}(\mathit{HomCod})(\mathit{Obj}). \mathit{L-10-5-}\chi'\text{-arrow} \alpha \beta \mathfrak{T} \mathfrak{K} c a)$

and [*cat-Kan-cs-simps*]:
 $L-10-5-\chi \alpha \beta \mathfrak{T} \mathfrak{K} c(\mathit{NTDom}) = \mathit{cf-Cone} \alpha \beta (\mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K})$
and [*cat-Kan-cs-simps*]:
 $L-10-5-\chi \alpha \beta \mathfrak{T} \mathfrak{K} c(\mathit{NTCod}) = L-10-5-N \alpha \beta \mathfrak{T} \mathfrak{K} c$
and $L-10-5-\chi \alpha \beta \mathfrak{T} \mathfrak{K} c(\mathit{NTDGDom}) = \mathit{op-cat} (\mathfrak{T}(\mathit{HomCod}))$
and [*cat-Kan-cs-simps*]: $L-10-5-\chi \alpha \beta \mathfrak{T} \mathfrak{K} c(\mathit{NTDGCod}) = \mathit{cat-Set} \beta$
unfolding $L-10-5-\chi\text{-def}$ *nt-field-simps* **by** (*simp-all add: nat-omega-simps*)

context

fixes $\alpha \mathfrak{A} \mathfrak{B} \mathfrak{T}$
assumes $\mathfrak{T}: \mathfrak{T} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{A}$

begin

interpretation *is-functor* $\alpha \mathfrak{B} \mathfrak{A} \mathfrak{T}$ **by** (*rule* \mathfrak{T})

lemmas $L-10-5-\chi\text{-components}' =$
 $L-10-5-\chi\text{-components}[\mathbf{where} \ \mathfrak{T}=\mathfrak{T}, \text{ unfolded } \mathit{cat-cs-simps}]$

lemmas [*cat-Kan-cs-simps*] = $L-10-5-\chi\text{-components}'(4)$

end

15.8.2 Natural transformation map

mk-VLambda $L-10-5-\chi\text{-components}(1)$
 $[\mathit{vsu} \ L-10-5-\chi\text{-NTMap-vsuv}[\mathit{cat-Kan-cs-intros}]]$

context

fixes $\alpha \mathfrak{A} \mathfrak{B} \mathfrak{T}$
assumes $\mathfrak{T}: \mathfrak{T} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{A}$

begin

interpretation *is-functor* $\alpha \mathfrak{B} \mathfrak{A} \mathfrak{T}$ **by** (*rule* \mathfrak{T})

mk-VLambda $L-10-5-\chi\text{-components}(1)[\mathbf{where} \ \mathfrak{T}=\mathfrak{T}, \text{ unfolded } \mathit{cat-cs-simps}]$
 $[\mathit{vdomain} \ L-10-5-\chi\text{-NTMap-vdomain}[\mathit{cat-Kan-cs-simps}]]$
 $[\mathit{app} \ L-10-5-\chi\text{-NTMap-app}[\mathit{cat-Kan-cs-simps}]]$

end

15.8.3 $L-10-5-\chi$ is a natural isomorphism

lemma $L-10-5-\chi\text{-is-iso-ntcf}$:

— See lemma on page 245 in [9].

assumes $Z \beta$

and $\alpha \in_{\circ} \beta$

and $\mathfrak{K} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{C}$

and $\mathfrak{T} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{A}$

and $c \in_{\circ} \mathfrak{C}(\mathit{Obj})$

shows $L-10-5-\chi \alpha \beta \mathfrak{T} \mathfrak{K} c :$

$\mathit{cf-Cone} \alpha \beta (\mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K}) \mapsto_{CF.iso} L-10-5-N \alpha \beta \mathfrak{T} \mathfrak{K} c :$

$\mathit{op-cat} \mathfrak{A} \mapsto \mapsto_{C\beta} \mathit{cat-Set} \beta$

(**is** $\langle ?L-10-5-\chi : ?\mathit{cf-Cone} \mapsto_{CF.iso} ?L-10-5-N : \mathit{op-cat} \mathfrak{A} \mapsto \mapsto_{C\beta} \mathit{cat-Set} \beta \rangle$)

proof-

let $?FUNCT = \langle \lambda \mathfrak{A}. \mathit{cat-FUNCT} \alpha \mathfrak{A} (\mathit{cat-Set} \alpha) \rangle$

let $?c\mathfrak{K}\text{-}\mathfrak{A} = \langle \mathit{cat-FUNCT} \alpha (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} \rangle$

let $?ntcf\text{-}c\mathfrak{K}\text{-}\mathfrak{A} = \langle \mathit{ntcf-const} (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} \rangle$

let $?T-c\mathfrak{K} = \langle T \circ_{CF} c \circ \prod_{CF} \mathfrak{K} \rangle$
let $?H-\mathfrak{C} = \langle \lambda c. Hom_{O.C\alpha} \mathfrak{C}(c, -) \rangle$
let $?H-\mathfrak{A} = \langle \lambda a. Hom_{O.C\alpha} \mathfrak{A}(a, -) \rangle$
let $?L-10-5-\chi'-arrow = \langle L-10-5-\chi'-arrow \alpha \beta T \mathfrak{K} c \rangle$
let $?cf-c\mathfrak{K}-\mathfrak{A} = \langle cf-const (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} \rangle$
let $?L-10-5-v = \langle L-10-5-v \alpha T \mathfrak{K} c \rangle$
let $?L-10-5-v-arrow = \langle L-10-5-v-arrow T \mathfrak{K} c \rangle$

interpret $\beta: Z \beta$ **by** (rule *assms(1)*)

interpret $\mathfrak{K}: is-functor \alpha \mathfrak{B} \mathfrak{C} \mathfrak{K}$ **by** (rule *assms(3)*)
interpret $T: is-functor \alpha \mathfrak{B} \mathfrak{A} T$ **by** (rule *assms(4)*)

from $\mathfrak{K}.empty-is-zet$ *assms(5)* **interpret** $c\mathfrak{K}: category \alpha \langle c \downarrow_{CF} \mathfrak{K} \rangle$
by (*cs-concl cs-shallow cs-intro: cat-comma-cs-intros*)
from *assms(1,2,5)* **interpret** $c\mathfrak{K}-\mathfrak{A}: category \beta ?c\mathfrak{K}-\mathfrak{A}$
by
(*cs-concl cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros*)
interpret $\beta-c\mathfrak{K}-\mathfrak{A}: category \beta ?c\mathfrak{K}-\mathfrak{A}$
by (*cs-concl cs-shallow cs-intro: cat-cs-intros assms(2)*)+
from *assms(2,5)* **interpret** $\Delta: is-functor \beta \mathfrak{A} ?c\mathfrak{K}-\mathfrak{A} \langle \Delta_{CF} \alpha (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} \rangle$
by (*cs-concl cs-intro: cat-cs-intros cat-op-intros*)+
from $\mathfrak{K}.empty-is-zet$ *assms(5)* **interpret** $\Pi c:$
is-functor $\alpha \langle c \downarrow_{CF} \mathfrak{K} \rangle \mathfrak{B} \langle c \circ \prod_{CF} \mathfrak{K} \rangle$
by
(*cs-concl cs-shallow cs-simp: cat-comma-cs-simps cs-intro: cat-cs-intros cat-comma-cs-intros*)
interpret $\beta \Pi c: is-tiny-functor \beta \langle c \downarrow_{CF} \mathfrak{K} \rangle \mathfrak{B} \langle c \circ \prod_{CF} \mathfrak{K} \rangle$
by (rule $\Pi c.cf-is-tiny-functor-if-ge-Limit[OF \textit{assms}(1,2)]$)

interpret $E: is-functor \beta \langle ?FUNCT \mathfrak{C} \times_C \mathfrak{C} \rangle \langle cat-Set \beta \rangle \langle cf-eval \alpha \beta \mathfrak{C} \rangle$
by (rule $\mathfrak{K}.HomCod.cat-cf-eval-is-functor[OF \textit{assms}(1,2)]$)

from *assms(2)* **interpret** $FUNCT-\mathfrak{A}: tiny-category \beta \langle ?FUNCT \mathfrak{A} \rangle$
by (*cs-concl cs-intro: cat-cs-intros cat-FUNCT-cs-intros*)
from *assms(2)* **interpret** $FUNCT-\mathfrak{B}: tiny-category \beta \langle ?FUNCT \mathfrak{B} \rangle$
by (*cs-concl cs-intro: cat-cs-intros cat-FUNCT-cs-intros*)
from *assms(2)* **interpret** $FUNCT-\mathfrak{C}: tiny-category \beta \langle ?FUNCT \mathfrak{C} \rangle$
by (*cs-concl cs-intro: cat-cs-intros cat-FUNCT-cs-intros*)

interpret $\beta \mathfrak{A}: tiny-category \beta \mathfrak{A}$
by (rule *category.cat-tiny-category-if-ge-Limit*)
(*use assms(2) in cs-concl cs-intro: cat-cs-intros*)+
interpret $\beta \mathfrak{B}: tiny-category \beta \mathfrak{B}$
by (rule *category.cat-tiny-category-if-ge-Limit*)
(*use assms(2) in cs-concl cs-intro: cat-cs-intros*)+
interpret $\beta \mathfrak{C}: tiny-category \beta \mathfrak{C}$
by (rule *category.cat-tiny-category-if-ge-Limit*)
(*use assms(2) in cs-concl cs-intro: cat-cs-intros*)+

interpret $\beta \mathfrak{K}: is-tiny-functor \beta \mathfrak{B} \mathfrak{C} \mathfrak{K}$
by (rule *is-functor.cf-is-tiny-functor-if-ge-Limit*)

(use *assms(2)* in $\langle \text{cs-concl cs-intro: cat-cs-intros} \rangle$)+
interpret $\beta\mathfrak{T}$: *is-tiny-functor* $\beta \mathfrak{B} \mathfrak{A} \mathfrak{T}$
 by (rule *is-functor.cf-is-tiny-functor-if-ge-Limit*)
 (use *assms(2)* in $\langle \text{cs-concl cs-intro: cat-cs-intros} \rangle$)+

interpret *cat-Set- $\alpha\beta$* : *subcategory* $\beta \langle \text{cat-Set } \alpha \rangle \langle \text{cat-Set } \beta \rangle$
 by (rule $\mathfrak{K}.\text{subcategory-cat-Set-cat-Set}[OF \text{ assms}(1,2)]$)

show *?thesis*
proof(*intro is-iso-ntcfI is-ntcfI', unfold cat-op-simps*)

show *vfsequence* ($?L-10-5-\chi$) **unfolding** *L-10-5- χ -def* **by** *auto*
show *vcard* ($?L-10-5-\chi$) = $5_{\mathbb{N}}$
unfolding *L-10-5- χ -def* **by** (*simp add: nat-omega-simps*)
from *assms(2)* **show** *?cf-Cone* : *op-cat* $\mathfrak{A} \mapsto \mapsto_{C\beta}$ *cat-Set* β
by (*intro is-functor.tm-cf-cf-Cone-is-functor-if-ge-Limit*)
 (*cs-concl cs-intro: cat-cs-intros*)+

from *assms* **show** *?L-10-5-N* : *op-cat* $\mathfrak{A} \mapsto \mapsto_{C\beta}$ *cat-Set* β
by (*cs-concl cs-shallow cs-intro: cat-Kan-cs-intros*)

show *?L-10-5- χ (NTMap)(|a)* :
?cf-Cone(*ObjMap*)(|a) $\mapsto_{\text{iso cat-Set } \beta}$ *?L-10-5-N*(*ObjMap*)(|a)
if $a \in_{\circ} \mathfrak{A}(|\text{Obj})$ **for** a
using *assms(2,3,4,5)* **that**
by
 (
 cs-concl
 cs-simp: *L-10-5- χ -NTMap-app*
 cs-intro: *cat-cs-intros L-10-5- χ' -arrow-is-iso-arr*
)

from *cat-Set-is-iso-arrD*[*OF this*] **show**
?L-10-5- χ (NTMap)(|a) : *?cf-Cone*(*ObjMap*)(|a) $\mapsto_{\text{cat-Set } \beta}$ *?L-10-5-N*(*ObjMap*)(|a)
if $a \in_{\circ} \mathfrak{A}(|\text{Obj})$ **for** a
using *that* **by** *auto*

have [*cat-cs-simps*]:
?L-10-5- χ' -arrow $b \circ_{A \text{ cat-Set } \beta}$
cf-hom $?c\mathfrak{K}\text{-}\mathfrak{A}$ [*ntcf-arrow* (*?ntcf-c \mathfrak{K} - \mathfrak{A} f*), *ntcf-arrow* (*ntcf-id ? \mathfrak{T} -c \mathfrak{K}*)] $_{\circ}$ =
cf-hom (*?FUNCT* \mathfrak{B})
 [
 ntcf-arrow (*ntcf-id* (*?H- \mathfrak{C} c* \circ_{CF} \mathfrak{K})),
 ntcf-arrow (*Hom* $_{A.C\alpha}$ \mathfrak{A} (*f*, $-$) $\circ_{NTCF-CF}$ \mathfrak{T})
] $_{\circ}$ $\circ_{A \text{ cat-Set } \beta}$ *?L-10-5- χ' -arrow* a
 (
 is
 ?L-10-5- χ' -arrow $b \circ_{A \text{ cat-Set } \beta}$ *?cf-hom-lhs* =
 ?cf-hom-rhs $\circ_{A \text{ cat-Set } \beta}$ *?L-10-5- χ' -arrow* a
)
if $f : b \mapsto_{\mathfrak{A}} a$ **for** $a b f$

proof-
let *?H-f* = $\langle \text{Hom}_{A.C\alpha}\mathfrak{A}(f,-) \rangle$
from *that* *assms c \mathfrak{K} - \mathfrak{A} .category-axioms c \mathfrak{K} - \mathfrak{A} .category-axioms* **have** *lhs*:
?L-10-5- χ' -arrow $b \circ_{A \text{ cat-Set } \beta}$ *?cf-hom-lhs* :
Hom $?c\mathfrak{K}\text{-}\mathfrak{A}$ (*cf-map* (*?cf-c \mathfrak{K} - \mathfrak{A} a*) (*cf-map* *? \mathfrak{T} -c \mathfrak{K}*)) $\mapsto_{\text{cat-Set } \beta}$
?L-10-5-N(*ObjMap*)(|b)
by
 (

cs-concl
cs-simp:
cat-Kan-cs-simps
cat-cs-simps
cat-FUNCT-cs-simps
cat-FUNCT-components(1)
cat-op-simps
cs-intro:
cat-Kan-cs-intros
cat-FUNCT-cs-intros
cat-cs-intros
cat-prod-cs-intros
cat-op-intros
)

then have dom-lhs:
 $\mathcal{D}_\circ ((?L-10-5-\chi'-arrow\ b \circ_{A\ cat-Set\ \beta} ?cf-hom-lhs)(\downarrow ArrVal)) =$
 $Hom\ ?c\mathfrak{K}-\mathfrak{A}\ (cf-map\ (?cf-c\mathfrak{K}-\mathfrak{A}\ a))\ (cf-map\ ?\mathfrak{T}-c\mathfrak{K})$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps*)

from *that assms c \mathfrak{K} - \mathfrak{A} .category-axioms c \mathfrak{K} - \mathfrak{A} .category-axioms have rhs:*
 $?cf-hom-rhs \circ_{A\ cat-Set\ \beta} ?L-10-5-\chi'-arrow\ a :$
 $Hom\ ?c\mathfrak{K}-\mathfrak{A}\ (cf-map\ (?cf-c\mathfrak{K}-\mathfrak{A}\ a))\ (cf-map\ ?\mathfrak{T}-c\mathfrak{K}) \mapsto_{cat-Set\ \beta}$
 $?L-10-5-N(\downarrow ObjMap)(\downarrow b)$
by
(

cs-concl
cs-simp:
cat-Kan-cs-simps
cat-cs-simps
cat-FUNCT-components(1)
cat-op-simps
cs-intro:
cat-Kan-cs-intros
cat-cs-intros
cat-prod-cs-intros
cat-FUNCT-cs-intros
cat-op-intros
)

then have dom-rhs:
 $\mathcal{D}_\circ ((?cf-hom-rhs \circ_{A\ cat-Set\ \beta} ?L-10-5-\chi'-arrow\ a)(\downarrow ArrVal)) =$
 $Hom\ ?c\mathfrak{K}-\mathfrak{A}\ (cf-map\ (?cf-c\mathfrak{K}-\mathfrak{A}\ a))\ (cf-map\ ?\mathfrak{T}-c\mathfrak{K})$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps*)

show ?thesis
proof(*rule arr-Set-eqI*)
from lhs show arr-Set-lhs:
 $arr-Set\ \beta\ (?L-10-5-\chi'-arrow\ b \circ_{A\ cat-Set\ \beta} ?cf-hom-lhs)$
by (*auto dest: cat-Set-is-arrD(1)*)
from rhs show arr-Set-rhs:
 $arr-Set\ \beta\ (?cf-hom-rhs \circ_{A\ cat-Set\ \beta} ?L-10-5-\chi'-arrow\ a)$
by (*auto dest: cat-Set-is-arrD(1)*)
show
 $(?L-10-5-\chi'-arrow\ b \circ_{A\ cat-Set\ \beta} ?cf-hom-lhs)(\downarrow ArrVal) =$
 $(?cf-hom-rhs \circ_{A\ cat-Set\ \beta} ?L-10-5-\chi'-arrow\ a)(\downarrow ArrVal)$
proof(*rule vsu-eqI, unfold dom-lhs dom-rhs in-Hom-iff*)
fix F **assume** *prems:* $F : cf-map\ (?cf-c\mathfrak{K}-\mathfrak{A}\ a) \mapsto_{?c\mathfrak{K}-\mathfrak{A}} cf-map\ ?\mathfrak{T}-c\mathfrak{K}$
let $?F = \langle ntcf-of-ntcf-arrow\ (c \downarrow_{CF}\ \mathfrak{K})\ \mathfrak{A}\ F \rangle$
from that have [*cat-cs-simps*]:

$cf\text{-of-}cf\text{-map } (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} (cf\text{-map } (?cf\text{-}c\mathfrak{K}\text{-}\mathfrak{A} a)) = ?cf\text{-}c\mathfrak{K}\text{-}\mathfrak{A} a$
 $cf\text{-of-}cf\text{-map } (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} (cf\text{-map } (?f\text{-}c\mathfrak{K})) = ?f\text{-}c\mathfrak{K}$
by (*cs-concl cs-simp: cat-FUNCT-cs-simps cs-intro: cat-cs-intros*)
note $F = \text{cat-FUNCT-is-arrD}[OF \text{ prems, unfolded cat-cs-simps}]$
from that $F(1)$ **have** $F\text{-const-is-cat-cone}$:
 $?F \cdot_{NTCF} ?ntcf\text{-}c\mathfrak{K}\text{-}\mathfrak{A} f : b <_{CF.cone} ?f\text{-}c\mathfrak{K} : c \downarrow_{CF} \mathfrak{K} \mapsto_{C\alpha} \mathfrak{A}$
by
(*cs-concl*
cs-simp: cat-cs-simps cs-intro: is-cat-coneI cat-cs-intros
)

have [*cat-cs-simps*]:
 $?L\text{-}10\text{-}5\text{-}v (ntcf\text{-arrow } (?F \cdot_{NTCF} ?ntcf\text{-}c\mathfrak{K}\text{-}\mathfrak{A} f)) b =$
 $?H\text{-}f \circ_{NTCF\text{-}CF} \mathfrak{T} \cdot_{NTCF} ?L\text{-}10\text{-}5\text{-}v (ntcf\text{-arrow } ?F) a$
proof(*rule ntcf-eqI*)
from *assms* that $F(1)$ **show**
 $?L\text{-}10\text{-}5\text{-}v (ntcf\text{-arrow } (?F \cdot_{NTCF} ?ntcf\text{-}c\mathfrak{K}\text{-}\mathfrak{A} f)) b :$
 $?H\text{-}\mathfrak{C} c \circ_{CF} \mathfrak{K} \mapsto_{CF} ?H\text{-}\mathfrak{A} b \circ_{CF} \mathfrak{T} : \mathfrak{B} \mapsto_{C\alpha} \text{cat-Set } \alpha$
by
(*cs-concl cs-intro:*
cat-Kan-cs-intros cat-cs-intros is-cat-coneI
)

then have *dom-v*:
 $\mathcal{D}_\circ (?L\text{-}10\text{-}5\text{-}v (ntcf\text{-arrow } (?F \cdot_{NTCF} ?ntcf\text{-}c\mathfrak{K}\text{-}\mathfrak{A} f)) b)(NTMap) =$
 $\mathfrak{B}(\text{Obj})$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps*)
from *assms* that $F(1)$ **show**
 $?H\text{-}f \circ_{NTCF\text{-}CF} \mathfrak{T} \cdot_{NTCF} ?L\text{-}10\text{-}5\text{-}v (ntcf\text{-arrow } ?F) a :$
 $?H\text{-}\mathfrak{C} c \circ_{CF} \mathfrak{K} \mapsto_{CF} ?H\text{-}\mathfrak{A} b \circ_{CF} \mathfrak{T} : \mathfrak{B} \mapsto_{C\alpha} \text{cat-Set } \alpha$
by
(*cs-concl cs-intro:*
cat-Kan-cs-intros cat-cs-intros is-cat-coneI
)

then have *dom-fT v*:
 $\mathcal{D}_\circ ((?H\text{-}f \circ_{NTCF\text{-}CF} \mathfrak{T} \cdot_{NTCF} ?L\text{-}10\text{-}5\text{-}v (ntcf\text{-arrow } ?F) a)(NTMap)) =$
 $\mathfrak{B}(\text{Obj})$
by (*cs-concl cs-simp: cat-cs-simps*)
show
 $?L\text{-}10\text{-}5\text{-}v (ntcf\text{-arrow } (?F \cdot_{NTCF} ?ntcf\text{-}c\mathfrak{K}\text{-}\mathfrak{A} f)) b(NTMap) =$
 $(?H\text{-}f \circ_{NTCF\text{-}CF} \mathfrak{T} \cdot_{NTCF} ?L\text{-}10\text{-}5\text{-}v (ntcf\text{-arrow } ?F) a)(NTMap)$
proof(*rule vsv-eqI, unfold dom-v dom-fT v*)
fix b' **assume** *prems'*: $b' \in_\circ \mathfrak{B}(\text{Obj})$
let $?Y = \langle \text{Yoneda-component } (?H\text{-}\mathfrak{A} b) a f (\mathfrak{T}(\text{ObjMap})(b')) \rangle$
let $?Rb' = \langle R(\text{ObjMap})(b') \rangle$
let $?Tb' = \langle T(\text{ObjMap})(b') \rangle$
have [*cat-cs-simps*]:
 $?L\text{-}10\text{-}5\text{-}v\text{-arrow } (ntcf\text{-arrow } (?F \cdot_{NTCF} ?ntcf\text{-}c\mathfrak{K}\text{-}\mathfrak{A} f)) b b' =$
 $?Y \circ_{A \text{ cat-Set } \alpha} ?L\text{-}10\text{-}5\text{-}v\text{-arrow } (ntcf\text{-arrow } ?F) a b'$
(is $\langle ?v\text{-}Ffb' = ?Yv \rangle$)
proof-
from *assms prems'* $F\text{-const-is-cat-cone}$ **have** $v\text{-}Ffb'$:
 $?v\text{-}Ffb' : \text{Hom } \mathfrak{C} c ?Rb' \mapsto_{\text{cat-Set } \alpha} \text{Hom } \mathfrak{A} b ?Tb'$
by
(*cs-concl cs-shallow*
cs-intro: cat-cs-intros L-10-5-v-arrow-is-arr
)

```

)
then have dom-v-Ffbb':  $\mathcal{D}_o$  ( $?v\text{-Ffbb}'(\downarrow\text{ArrVal})$ ) =  $\text{Hom } \mathfrak{C} \ c$  ( $?Rb'$ )
  by (cs-concl cs-shallow cs-simp: cat-cs-simps)
from assms that  $\mathfrak{T}.\text{HomCod}.\text{category-axioms}$  prems'  $F(1)$  have  $Yv$ :
   $?Yv : \text{Hom } \mathfrak{C} \ c \ ?Rb' \mapsto_{\text{cat-Set } \alpha} \text{Hom } \mathfrak{A} \ b \ ?\mathfrak{T}b'$ 
  by
  (
    cs-concl
    cs-simp: cat-Kan-cs-simps cat-cs-simps cat-op-simps
    cs-intro: is-cat-coneI cat-Kan-cs-intros cat-cs-intros
  )
then have dom-Yv:  $\mathcal{D}_o$  ( $?Yv(\downarrow\text{ArrVal})$ ) =  $\text{Hom } \mathfrak{C} \ c$  ( $?Rb'$ )
  by (cs-concl cs-shallow cs-simp: cat-cs-simps)
show ?thesis
proof(rule arr-Set-eqI)
  from  $v\text{-Ffbb}'$  show arr-Set-v-Ffbb':  $\text{arr-Set } \alpha \ ?v\text{-Ffbb}'$ 
    by (auto dest: cat-Set-is-arrD(1))
  from  $Yv$  show arr-Set-Yv:  $\text{arr-Set } \alpha \ ?Yv$ 
    by (auto dest: cat-Set-is-arrD(1))
  show  $?v\text{-Ffbb}'(\downarrow\text{ArrVal}) = ?Yv(\downarrow\text{ArrVal})$ 
  proof(rule vsu-eqI, unfold dom-v-Ffbb' dom-Yv in-Hom-iff)
    fix  $g$  assume  $g : c \mapsto_{\mathfrak{C}} ?Rb'$ 
    with
      assms(2-)
       $\mathfrak{R}.\text{is-functor-axioms}$ 
       $\mathfrak{T}.\text{is-functor-axioms}$ 
       $\mathfrak{T}.\text{HomCod}.\text{category-axioms}$ 
       $\mathfrak{R}.\text{HomCod}.\text{category-axioms}$ 
      that prems' F(1)
    show  $?v\text{-Ffbb}'(\downarrow\text{ArrVal})(g) = ?Yv(\downarrow\text{ArrVal})(g)$ 
    by
    (
      cs-concl
      cs-simp:
        cat-Kan-cs-simps
        cat-cs-simps
        L-10-5-v-arrow-ArrVal-app
        cat-comma-cs-simps
        cat-op-simps
      cs-intro:
        cat-Kan-cs-intros
        is-cat-coneI
        cat-cs-intros
        cat-comma-cs-intros
        cat-op-intros
      cs-simp: cat-FUNCT-cs-simps
    )
  qed (use arr-Set-v-Ffbb' arr-Set-Yv in auto)
qed
  (
    use v-Ffbb' Yv in
     $\langle \text{cs-concl cs-shallow cs-simp: cat-cs-simps} \rangle$ 
  )+
qed

from assms prems' that F(1) show
   $?L\text{-}10\text{-}5\text{-}v$  ( $\text{ntcf-arrow } (?F \cdot_{\text{NTCF}} ?\text{ntcf-c}\mathfrak{R}\text{-}\mathfrak{A} \ f)$ )  $b(\downarrow\text{NTMap})(\downarrow b')$  =
  ( $?H\text{-}f \circ_{\text{NTCF-CF}} \mathfrak{T} \cdot_{\text{NTCF}} ?L\text{-}10\text{-}5\text{-}v$  ( $\text{ntcf-arrow } ?F$ )  $a$ )  $(\downarrow\text{NTMap})(\downarrow b')$ 

```

```

    by
      (
        cs-concl
        cs-simp: cat-Kan-cs-simps cat-cs-simps
        cs-intro: is-cat-coneI cat-Kan-cs-intros cat-cs-intros
      )

  qed (cs-concl cs-intro: cat-Kan-cs-intros cat-cs-intros)

  qed simp-all

  from that F(1) interpret F: is-cat-cone  $\alpha$  a  $\langle c \downarrow_{CF} \mathcal{R} \rangle \mathcal{A} \langle ?\mathcal{T}\text{-}c\mathcal{R} \rangle ?F$ 
    by (cs-concl cs-shallow cs-intro: is-cat-coneI cat-cs-intros)
  from
    assms(2-) prems F(1) that
     $\mathcal{T}.HomCod.cat\text{-}ntcf\text{-}Hom\text{-}snd\text{-}is\text{-}ntcf[OF\ that]$ 
    c $\mathcal{R}$ - $\mathcal{A}$ .category-axioms
  show
    (?L-10-5- $\chi'$ -arrow b  $\circ_{A\ cat\text{-}Set\ \beta}$  ?cf-hom-lhs) (ArrVal) (F) =
    (?cf-hom-rhs  $\circ_{A\ cat\text{-}Set\ \beta}$  ?L-10-5- $\chi'$ -arrow a) (ArrVal) (F)
  by (subst (1 2) F(2))
    (
      cs-concl
      cs-simp:
        cat-cs-simps
        cat-Kan-cs-simps
        cat-FUNCT-cs-simps
        cat-FUNCT-components(1)
        cat-op-simps
      cs-intro:
        is-cat-coneI
        cat-Kan-cs-intros
        cat-cs-intros
        cat-prod-cs-intros
        cat-FUNCT-cs-intros
        cat-op-intros
    )
  qed (use arr-Set-lhs arr-Set-rhs in auto)

  qed (use lhs rhs in  $\langle$ cs-concl cs-shallow cs-simp: cat-cs-simps $\rangle$ )

  qed

  show
    ?L-10-5- $\chi$ (NTMap)(b)  $\circ_{A\ cat\text{-}Set\ \beta}$  ?cf-Cone(ArrMap)(f) =
    ?L-10-5-N(ArrMap)(f)  $\circ_{A\ cat\text{-}Set\ \beta}$  ?L-10-5- $\chi$ (NTMap)(a)
  if f : b  $\mapsto_{\mathcal{A}}$  a for a b f
  using that assms
  by
    (
      cs-concl
      cs-simp:
        cat-cs-simps
        cat-Kan-cs-simps
        cat-FUNCT-components(1)
        cat-FUNCT-cs-simps
        cat-op-simps
      cs-intro:
    )

```

cat-Kan-cs-intros
cat-cs-intros
cat-FUNCT-cs-intros
cat-op-intros
)

qed

(

cs-concl

cs-simp: *cat-Kan-cs-simps* **cs-intro:** *cat-cs-intros cat-Kan-cs-intros*

)+

qed

15.9 The existence of a canonical limit or a canonical colimit for the pointwise Kan extensions

lemma (in *is-cat-pw-rKe*) *cat-pw-rKe-ex-cat-limit:*

— Based on the elements of Chapter X-5 in [9].

assumes $\mathfrak{K} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{C}$

and $\mathfrak{T} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{A}$

and $c \in_o \mathfrak{C}(\text{Obj})$

obtains UA

where $UA : \mathfrak{G}(\text{ObjMap})(\downarrow c) <_{CF.lim} \mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} : c \downarrow_{CF} \mathfrak{K} \mapsto \mapsto_{C\alpha} \mathfrak{A}$

proof-

define β **where** $\beta = \alpha + \omega$

have $\beta : \mathfrak{Z} \beta$ **and** $\alpha\beta : \alpha \in_o \beta$

by (*simp-all add: β -def AG.Z-Limit- $\alpha\omega$ AG.Z- ω - $\alpha\omega$ Z-def AG.Z- α - $\alpha\omega$*)

then interpret $\beta : \mathfrak{Z} \beta$ **by** *simp*

let $?FUNCT = \langle \lambda \mathfrak{A}. \text{cat-FUNCT } \alpha \mathfrak{A} (\text{cat-Set } \alpha) \rangle$
let $?H-A = \langle \lambda f. \text{Hom}_{A.C\alpha} \mathfrak{A}(f, -) \rangle$
let $?H-A\mathfrak{G} = \langle \lambda f. ?H-A f \circ_{NTCF-CF} \mathfrak{G} \rangle$
let $?H-\mathfrak{A} = \langle \lambda a. \text{Hom}_{O.C\alpha} \mathfrak{A}(a, -) \rangle$
let $?H-\mathfrak{A}\mathfrak{T} = \langle \lambda a. ?H-\mathfrak{A} a \circ_{CF} \mathfrak{T} \rangle$
let $?H-\mathfrak{A}\mathfrak{G} = \langle \lambda a. ?H-\mathfrak{A} a \circ_{CF} \mathfrak{G} \rangle$
let $?H-\mathfrak{C} = \langle \lambda c. \text{Hom}_{O.C\alpha} \mathfrak{C}(c, -) \rangle$
let $?H-\mathfrak{C}\mathfrak{K} = \langle \lambda c. ?H-\mathfrak{C} c \circ_{CF} \mathfrak{K} \rangle$
let $?H-\mathfrak{A}\varepsilon = \langle \lambda b. ?H-\mathfrak{A} b \circ_{CF-NTCF} \varepsilon \rangle$
let $?SET-\mathfrak{K} = \langle \text{exp-cat-cf } \alpha (\text{cat-Set } \alpha) \mathfrak{K} \rangle$
let $?H-FUNCT = \langle \lambda \mathfrak{C} \mathfrak{F}. \text{Hom}_{O.C\beta} ?FUNCT \mathfrak{C}(-, \text{cf-map } \mathfrak{F}) \rangle$
let $?ua-NTDGD\text{om} = \langle \text{op-cat } (?FUNCT \mathfrak{C}) \rangle$
let $?ua-NTD\text{om} = \langle \lambda a. ?H-FUNCT \mathfrak{C} (?H-\mathfrak{A}\mathfrak{G} a) \rangle$
let $?ua-NTC\text{od} = \langle \lambda a. ?H-FUNCT \mathfrak{B} (?H-\mathfrak{A}\mathfrak{T} a) \circ_{CF} \text{op-cf } ?SET-\mathfrak{K} \rangle$
let $?c\mathfrak{K}-\mathfrak{A} = \langle \text{cat-FUNCT } \alpha (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} \rangle$
let $?ua =$
 \langle
 $\lambda a. \text{ntcf-ua-fo}$
 β
 $?SET-\mathfrak{K}$
 $(\text{cf-map } (?H-\mathfrak{A}\mathfrak{T} a))$
 $(\text{cf-map } (?H-\mathfrak{A}\mathfrak{G} a))$
 $(\text{ntcf-arrow } (?H-\mathfrak{A}\varepsilon a))$
 \rangle
let $?cf-nt = \langle \text{cf-nt } \alpha \beta (\text{cf-id } \mathfrak{C}) \rangle$
let $?cf-eval = \langle \text{cf-eval } \alpha \beta \mathfrak{C} \rangle$
let $?T-c\mathfrak{K} = \langle \mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} \rangle$

```

let ?cf-c $\mathfrak{K}$ - $\mathfrak{A}$  =  $\langle$ cf-const (c  $\downarrow_{CF}$   $\mathfrak{K}$ )  $\mathfrak{A}$  $\rangle$ 
let ? $\mathfrak{G}$ c =  $\langle$  $\mathfrak{G}$ (ObjMap)(c) $\rangle$ 
let ? $\Delta$  =  $\langle$  $\Delta_{CF}$   $\alpha$  (c  $\downarrow_{CF}$   $\mathfrak{K}$ )  $\mathfrak{A}$  $\rangle$ 
let ?ntcf-ua-fo =
   $\langle$ 
     $\lambda$ a. ntcf-ua-fo
       $\beta$ 
      ?SET- $\mathfrak{K}$ 
      (cf-map (?H- $\mathfrak{A}$  $\mathfrak{T}$  a))
      (cf-map (?H- $\mathfrak{A}$  $\mathfrak{G}$  a))
      (ntcf-arrow (?H- $\mathfrak{A}$  $\varepsilon$  a))
     $\rangle$ 
let ?umap-fo =
   $\langle$ 
     $\lambda$ b. umap-fo
      ?SET- $\mathfrak{K}$ 
      (cf-map (?H- $\mathfrak{A}$  $\mathfrak{T}$  b))
      (cf-map (?H- $\mathfrak{A}$  $\mathfrak{G}$  b))
      (ntcf-arrow (?H- $\mathfrak{A}$  $\varepsilon$  b))
      (cf-map (?H- $\mathfrak{C}$  c))
     $\rangle$ 

interpret  $\mathfrak{K}$ : is-functor  $\alpha$   $\mathfrak{B}$   $\mathfrak{C}$   $\mathfrak{K}$  by (rule assms(1))
interpret  $\mathfrak{T}$ : is-functor  $\alpha$   $\mathfrak{B}$   $\mathfrak{A}$   $\mathfrak{T}$  by (rule assms(2))

from AG.vempty-is-zet assms(3) interpret c $\mathfrak{K}$ : category  $\alpha$   $\langle$ c  $\downarrow_{CF}$   $\mathfrak{K}$  $\rangle$ 
  by (cs-concl cs-shallow cs-intro: cat-comma-cs-intros)
from  $\alpha\beta$  assms(3) interpret c $\mathfrak{K}$ - $\mathfrak{A}$ : category  $\beta$  ?c $\mathfrak{K}$ - $\mathfrak{A}$ 
  by
  (
    cs-concl cs-intro:
      cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
  )
from  $\alpha\beta$  assms(3) interpret  $\Delta$ : is-functor  $\beta$   $\mathfrak{A}$  ?c $\mathfrak{K}$ - $\mathfrak{A}$  ? $\Delta$ 
  by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-op-intros)+
from AG.vempty-is-zet assms(3) interpret  $\Pi$ c:
  is-functor  $\alpha$   $\langle$ c  $\downarrow_{CF}$   $\mathfrak{K}$  $\rangle$   $\mathfrak{B}$   $\langle$ c  $\circ\sqcap_{CF}$   $\mathfrak{K}$  $\rangle$ 
  by
  (
    cs-concl cs-shallow
    cs-simp: cat-comma-cs-simps
    cs-intro: cat-cs-intros cat-comma-cs-intros
  )

interpret  $\beta\Pi$ c: is-tiny-functor  $\beta$   $\langle$ c  $\downarrow_{CF}$   $\mathfrak{K}$  $\rangle$   $\mathfrak{B}$   $\langle$ c  $\circ\sqcap_{CF}$   $\mathfrak{K}$  $\rangle$ 
  by (rule  $\Pi$ c.cf-is-tiny-functor-if-ge-Limit[OF  $\beta$   $\alpha\beta$ ])

interpret E: is-functor  $\beta$   $\langle$ ?FUNCT  $\mathfrak{C}$   $\times_{\mathfrak{C}}$   $\mathfrak{C}$  $\rangle$   $\langle$ cat-Set  $\beta$  $\rangle$  ?cf-eval
  by (rule AG.HomCod.cat-cf-eval-is-functor[OF  $\beta$   $\alpha\beta$ ])

from  $\alpha\beta$  interpret FUNCT- $\mathfrak{A}$ : tiny-category  $\beta$   $\langle$ ?FUNCT  $\mathfrak{A}$  $\rangle$ 
  by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-FUNCT-cs-intros)
from  $\alpha\beta$  interpret FUNCT- $\mathfrak{B}$ : tiny-category  $\beta$   $\langle$ ?FUNCT  $\mathfrak{B}$  $\rangle$ 
  by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-FUNCT-cs-intros)
from  $\alpha\beta$  interpret FUNCT- $\mathfrak{C}$ : tiny-category  $\beta$   $\langle$ ?FUNCT  $\mathfrak{C}$  $\rangle$ 
  by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-FUNCT-cs-intros)

interpret  $\beta\mathfrak{A}$ : tiny-category  $\beta$   $\mathfrak{A}$ 

```

by (rule category.cat-tiny-category-if-ge-Limit)
 (use $\alpha\beta$ in \langle cs-concl cs-intro: cat-cs-intros \rangle)+
interpret $\beta\mathfrak{B}$: tiny-category $\beta \mathfrak{B}$
by (rule category.cat-tiny-category-if-ge-Limit)
 (use $\alpha\beta$ in \langle cs-concl cs-intro: cat-cs-intros \rangle)+
interpret $\beta\mathfrak{C}$: tiny-category $\beta \mathfrak{C}$
by (rule category.cat-tiny-category-if-ge-Limit)
 (use $\alpha\beta$ in \langle cs-concl cs-intro: cat-cs-intros \rangle)+

interpret $\beta\mathfrak{R}$: is-tiny-functor $\beta \mathfrak{B} \mathfrak{C} \mathfrak{R}$
by (rule is-functor.cf-is-tiny-functor-if-ge-Limit)
 (use $\alpha\beta$ in \langle cs-concl cs-shallow cs-intro: cat-cs-intros \rangle)+
interpret $\beta\mathfrak{G}$: is-tiny-functor $\beta \mathfrak{C} \mathfrak{A} \mathfrak{G}$
by (rule is-functor.cf-is-tiny-functor-if-ge-Limit)
 (use $\alpha\beta$ in \langle cs-concl cs-shallow cs-intro: cat-cs-intros \rangle)+
interpret $\beta\mathfrak{T}$: is-tiny-functor $\beta \mathfrak{B} \mathfrak{A} \mathfrak{T}$
by (rule is-functor.cf-is-tiny-functor-if-ge-Limit)
 (use $\alpha\beta$ in \langle cs-concl cs-shallow cs-intro: cat-cs-intros \rangle)+

interpret cat-Set- $\alpha\beta$: subcategory $\beta \langle$ cat-Set $\alpha \rangle \langle$ cat-Set $\beta \rangle$
by (rule AG.subcategory-cat-Set-cat-Set[OF $\beta \alpha\beta$])

from *assms*(3) $\alpha\beta$ **interpret** Hom-c: is-functor $\alpha \mathfrak{C} \langle$ cat-Set $\alpha \rangle \langle ?H\text{-}\mathfrak{C} \ c \rangle$
by (cs-concl **cs-intro**: cat-cs-intros)

define $E' :: V$ **where** $E' =$
 [
 ($\lambda a \in \circ \mathfrak{A}(\text{Obj}). ?cf\text{-eval}(\text{ObjMap})(\text{cf-map } (?H\text{-}\mathfrak{A}\mathfrak{G} \ a), \ c)\bullet$),
 ($\lambda f \in \circ \mathfrak{A}(\text{Arr}). ?cf\text{-eval}(\text{ArrMap})(\text{ntcf-arrow } (?H\text{-}\mathfrak{A}\mathfrak{G} \ f), \ \mathfrak{C}(\text{CId})(\text{c}))\bullet$),
 op-cat \mathfrak{A} ,
 cat-Set β
]_o

have E' -components:
 $E'(\text{ObjMap}) = (\lambda a \in \circ \mathfrak{A}(\text{Obj}). ?cf\text{-eval}(\text{ObjMap})(\text{cf-map } (?H\text{-}\mathfrak{A}\mathfrak{G} \ a), \ c)\bullet)$
 $E'(\text{ArrMap}) =$
 ($\lambda f \in \circ \mathfrak{A}(\text{Arr}). ?cf\text{-eval}(\text{ArrMap})(\text{ntcf-arrow } (?H\text{-}\mathfrak{A}\mathfrak{G} \ f), \ \mathfrak{C}(\text{CId})(\text{c}))\bullet$)
 $E'(\text{HomDom}) = \text{op-cat } \mathfrak{A}$
 $E'(\text{HomCod}) = \text{cat-Set } \beta$
unfolding E' -def *dghm-field-simps* **by** (*simp-all add: nat-omega-simps*)

note [*cat-cs-simps*] = E' -components(3,4)

have E' -ObjMap-app[*cat-cs-simps*]:
 $E'(\text{ObjMap})(a) = ?cf\text{-eval}(\text{ObjMap})(\text{cf-map } (?H\text{-}\mathfrak{A}\mathfrak{G} \ a), \ c)\bullet$
if $a \in \circ \mathfrak{A}(\text{Obj})$ **for** a
using that unfolding E' -components **by** *simp*
have E' -ArrMap-app[*cat-cs-simps*]:
 $E'(\text{ArrMap})(f) = ?cf\text{-eval}(\text{ArrMap})(\text{ntcf-arrow } (?H\text{-}\mathfrak{A}\mathfrak{G} \ f), \ \mathfrak{C}(\text{CId})(\text{c}))\bullet$
if $f \in \circ \mathfrak{A}(\text{Arr})$ **for** f
using that unfolding E' -components **by** *simp*

have E' : $E' : \text{op-cat } \mathfrak{A} \mapsto \text{cat-Set } \beta$
proof(*intro is-functorI'*)

show *vfsequence* E' **unfolding** E' -def **by** *auto*

```

show vcard E' = 4N unfolding E'-def by (simp add: nat-omega-simps)
show vsv (E'(|ObjMap|)) unfolding E'-components by simp
show vsv (E'(|ArrMap|)) unfolding E'-components by simp
show Do (E'(|ObjMap|)) = op-cat  $\mathfrak{A}$ (|Obj|)
  unfolding E'-components by (simp add: cat-op-simps)
show Ro (E'(|ObjMap|))  $\subseteq_o$  cat-Set  $\beta$ (|Obj|)
  unfolding E'-components
proof(rule vrange-VLambda-vsubset)
  fix a assume prems: a  $\in_o$   $\mathfrak{A}$ (|Obj|)
  then have ?H- $\mathfrak{A}\mathfrak{G}$  a :  $\mathfrak{C} \mapsto_{C\alpha}$  cat-Set  $\alpha$ 
    by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  with assms(3) prems show
    ?cf-eval(|ObjMap|)(cf-map (?H- $\mathfrak{A}\mathfrak{G}$  a), c)  $\bullet \in_o$  cat-Set  $\beta$ (|Obj|)
  by
    (
      cs-concl
      cs-simp: cat-cs-simps cat-Set-components(1)
      cs-intro: cat-cs-intros cat-op-intros Ran.HomCod.cat-Hom-in-Vset
    )
qed
show Do (E'(|ArrMap|)) = op-cat  $\mathfrak{A}$ (|Arr|)
  unfolding E'-components by (simp add: cat-op-simps)
show E'(|ArrMap|)(f) : E'(|ObjMap|)(a)  $\mapsto_{cat-Set \beta}$  E'(|ObjMap|)(b)
  if f : a  $\mapsto_{op-cat \mathfrak{A}}$  b for a b f
proof-
  from that[unfolded cat-op-simps] assms(3) show ?thesis
  by (intro cat-Set- $\alpha\beta$ .subcat-is-arrD)
    (
      cs-concl
      cs-simp:
        category.cf-eval-ObjMap-app
        category.cf-eval-ArrMap-app
        E'-ObjMap-app
        E'-ArrMap-app
      cs-intro: cat-cs-intros
    )
qed
then have [cat-cs-intros]: E'(|ArrMap|)(f) : A  $\mapsto_{cat-Set \beta}$  B
  if A = E'(|ObjMap|)(a) and B = E'(|ObjMap|)(b) and f : b  $\mapsto_{\mathfrak{A}}$  a
  for a b f A B
  using that by (simp add: cat-op-simps)
show
  E'(|ArrMap|)(g  $\circ_A$  op-cat  $\mathfrak{A}$  f) = E'(|ArrMap|)(g)  $\circ_A$  cat-Set  $\beta$  E'(|ArrMap|)(f)
  if g : b  $\mapsto_{op-cat \mathfrak{A}}$  c and f : a  $\mapsto_{op-cat \mathfrak{A}}$  b for b c g a f
proof-
  note g = that(1)[unfolded cat-op-simps]
  and f = that(2)[unfolded cat-op-simps]
  from g f assms(3)  $\alpha\beta$  show ?thesis
  by
    (
      cs-concl
      cs-intro:
        cat-cs-intros
        cat-prod-cs-intros
        cat-FUNCT-cs-intros
        cat-op-intros
      cs-simp:
        cat-cs-simps
    )

```

```

    cat-FUNCT-cs-simps
    cat-prod-cs-simps
    cat-op-simps
    E.cf-ArrMap-Comp[symmetric]
  )+
qed

show E'(ArrMap)(op-cat  $\mathfrak{A}$ (CId)(a)) = cat-Set  $\beta$ (CId)(E'(ObjMap)(a))
  if a  $\in_{\circ}$  op-cat  $\mathfrak{A}$ (Obj) for a
proof(cs-concl-step cat-Set- $\alpha\beta$ .subcat-CId[symmetric])
  from that[unfolded cat-op-simps] assms(3) show
    E'(ObjMap)(a)  $\in_{\circ}$  cat-Set  $\alpha$ (Obj)
  by
    (
      cs-concl
      cs-simp: cat-Set-components(1) cat-cs-simps cat-op-simps
      cs-intro: cat-cs-intros
    )
  from that[unfolded cat-op-simps] assms(3) show
    E'(ArrMap)(op-cat  $\mathfrak{A}$ (CId)(a)) = cat-Set  $\alpha$ (CId)(E'(ObjMap)(a))
  by
    (
      cs-concl
      cs-intro: cat-cs-intros
      cs-simp:
        cat-Set-components(1)
        cat-cs-simps
        cat-op-simps
        ntcf-id-cf-comp[symmetric]
    )
  )
qed
qed (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)+
then interpret E': is-functor  $\beta$   $\langle$ op-cat  $\mathfrak{A}$  $\rangle$   $\langle$ cat-Set  $\beta$  $\rangle$  E' by simp

```

define $N' :: V$ where $N' =$

```

[
  ( $\lambda a \in_{\circ} \mathfrak{A}(Obj). ?cf-nt(ObjMap)(cf-map (?H-\mathfrak{A}\mathfrak{G} a), c)\bullet$ ),
  ( $\lambda f \in_{\circ} \mathfrak{A}(Arr). ?cf-nt(ArrMap)(ntcf-arrow (?H-A\mathfrak{G} f), \mathfrak{C}(CId)(c))\bullet$ ),
  op-cat  $\mathfrak{A}$ ,
  cat-Set  $\beta$ 
]

```

have N' -components:

```

N'(ObjMap) = ( $\lambda a \in_{\circ} \mathfrak{A}(Obj). ?cf-nt(ObjMap)(cf-map (?H-\mathfrak{A}\mathfrak{G} a), c)\bullet$ )
N'(ArrMap) =
  ( $\lambda f \in_{\circ} \mathfrak{A}(Arr). ?cf-nt(ArrMap)(ntcf-arrow (?H-A\mathfrak{G} f), \mathfrak{C}(CId)(c))\bullet$ )
N'(HomDom) = op-cat  $\mathfrak{A}$ 
N'(HomCod) = cat-Set  $\beta$ 
unfolding N'-def dghm-field-simps by (simp-all add: nat-omega-simps)

```

note [cat-cs-simps] = N' -components(3,4)

have N' -ObjMap-app[cat-cs-simps]:

```

N'(ObjMap)(a) = ?cf-nt(ObjMap)(cf-map (?H-\mathfrak{A}\mathfrak{G} a), c)\bullet
if a  $\in_{\circ}$   $\mathfrak{A}(Obj)$  for a

```

using that unfolding N' -components **by simp**
have N' -ArrMap-app[cat-cs-simps]:
 $N'(\text{ArrMap})(f) = ?cf\text{-nt}(\text{ArrMap})(\text{ntcf}\text{-arrow} (?H\text{-A}\mathfrak{G} f), \mathfrak{C}(\text{CId})(c))\bullet$
if $f \in_{\circ} \mathfrak{A}(\text{Arr})$ **for** f
using that unfolding N' -components **by simp**

from $\alpha\beta$ **interpret** $cf\text{-nt}\text{-}\mathfrak{C}$: $is\text{-functor} \beta \langle ?FUNCT \mathfrak{C} \times_C \mathfrak{C} \rangle \langle cat\text{-Set} \beta \rangle \langle ?cf\text{-nt} \rangle$
by ($cs\text{-concl}$ **cs-simp**: $cat\text{-cs-simps}$ **cs-intro**: $cat\text{-cs-intros}$)

have N' : $N' : op\text{-cat} \mathfrak{A} \mapsto_{C\beta} cat\text{-Set} \beta$
proof($intro$ $is\text{-functor} I'$)

show $vfsequence$ N' **unfolding** $N'\text{-def}$ **by simp**
show $vcard$ $N' = 4_{\mathbb{N}}$ **unfolding** $N'\text{-def}$ **by** ($simp$ add : $nat\text{-omega-simps}$)
show vsu ($N'(\text{ObjMap})$) **unfolding** $N'\text{-components}$ **by simp**
show vsu ($N'(\text{ArrMap})$) **unfolding** $N'\text{-components}$ **by simp**
show \mathcal{D}_{\circ} ($N'(\text{ObjMap})$) = $op\text{-cat} \mathfrak{A}(\text{Obj})$
unfolding $N'\text{-components}$ **by** ($simp$ add : $cat\text{-op-simps}$)
show \mathcal{R}_{\circ} ($N'(\text{ObjMap})$) $\subseteq_{\circ} cat\text{-Set} \beta(\text{Obj})$
unfolding $N'\text{-components}$

proof($rule$ $vrang\text{-VLambda}\text{-vsubset}$)
fix a **assume** $prems$: $a \in_{\circ} \mathfrak{A}(\text{Obj})$
with $assms(3)$ $\alpha\beta$ **show**
 $?cf\text{-nt}(\text{ObjMap})(\text{cf}\text{-map} (?H\text{-}\mathfrak{A}\mathfrak{G} a), c)\bullet \in_{\circ} cat\text{-Set} \beta(\text{Obj})$
by
(

 $cs\text{-concl}$
cs-simp: $cat\text{-Set-components}(1)$ $cat\text{-cs-simps}$ $cat\text{-FUNCT-cs-simps}$
cs-intro: $cat\text{-cs-intros}$ $FUNCT\text{-}\mathfrak{C}$. $cat\text{-Hom-in-Vset}$ $cat\text{-FUNCT-cs-intros}$

)

qed

show \mathcal{D}_{\circ} ($N'(\text{ArrMap})$) = $op\text{-cat} \mathfrak{A}(\text{Arr})$
unfolding $N'\text{-components}$ **by** ($simp$ add : $cat\text{-op-simps}$)
show $N'(\text{ArrMap})(f) : N'(\text{ObjMap})(a) \mapsto_{cat\text{-Set} \beta} N'(\text{ObjMap})(b)$
if $f : a \mapsto_{op\text{-cat} \mathfrak{A}} b$ **for** $a b f$
using that[$unfolded$ $cat\text{-op-simps}$] $assms(3)$
by
(

 $cs\text{-concl}$
cs-simp: $N'\text{-ObjMap-app}$ $N'\text{-ArrMap-app}$
cs-intro: $cat\text{-cs-intros}$ $cat\text{-prod-cs-intros}$ $cat\text{-FUNCT-cs-intros}$

)

show
 $N'(\text{ArrMap})(g \circ_{A op\text{-cat} \mathfrak{A}} f) = N'(\text{ArrMap})(g) \circ_{A cat\text{-Set} \beta} N'(\text{ArrMap})(f)$
if $g : b \mapsto_{op\text{-cat} \mathfrak{A}} c$ **and** $f : a \mapsto_{op\text{-cat} \mathfrak{A}} b$ **for** $b c g a f$

proof-
from $that$ $assms(3)$ $\alpha\beta$ **show** $?thesis$
unfolding $cat\text{-op-simps}$
by
(

 $cs\text{-concl}$
cs-intro:
 $cat\text{-cs-intros}$
 $cat\text{-prod-cs-intros}$
 $cat\text{-FUNCT-cs-intros}$
 $cat\text{-op-intros}$
cs-simp:
 $cat\text{-cs-simps}$
 $cat\text{-FUNCT-cs-simps}$

)

```

    cat-prod-cs-simps
    cat-op-simps
    cf-nt- $\mathfrak{C}$ .cf-ArrMap-Comp[symmetric]
  )
qed
show  $N'(\text{ArrMap})(\text{op-cat } \mathfrak{A}(\text{CId})(a)) = \text{cat-Set } \beta(\text{CId})(N'(\text{ObjMap})(a))$ 
  if  $a \in_{\circ} \text{op-cat } \mathfrak{A}(\text{Obj})$  for  $a$ 
proof-
  note [cat-cs-simps] =
    ntcf-id-cf-comp[symmetric]
    ntcf-arrow-id-ntcf-id[symmetric]
    cat-FUNCT-CId-app[symmetric]
  from that[unfolded cat-op-simps] assms(3)  $\alpha\beta$  show ?thesis
  by
    (
      cs-concl
      cs-intro:
        cat-cs-intros
        cat-FUNCT-cs-intros
        cat-prod-cs-intros
        cat-op-intros
      cs-simp: cat-FUNCT-cs-simps cat-cs-simps cat-op-simps
    )+
qed
qed (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)+
then interpret  $N'$ : is-functor  $\beta \langle \text{op-cat } \mathfrak{A} \rangle \langle \text{cat-Set } \beta \rangle N'$  by simp

```

```

define  $Y' :: V$  where  $Y' =$ 
  [
    ( $\lambda a \in_{\circ} \mathfrak{A}(\text{Obj}). \text{ntcf-Yoneda } \alpha \beta \mathfrak{C}(\text{NTMap})(\text{cf-map } (?H\text{-}\mathfrak{A}\mathfrak{G} \ a), c)\bullet$ ),
     $N'$ ,
     $E'$ ,
     $\text{op-cat } \mathfrak{A}$ ,
     $\text{cat-Set } \beta$ 
  ] $\circ$ 

```

```

have  $Y'$ -components:
   $Y'(\text{NTMap}) = (\lambda a \in_{\circ} \mathfrak{A}(\text{Obj}). \text{ntcf-Yoneda } \alpha \beta \mathfrak{C}(\text{NTMap})(\text{cf-map } (?H\text{-}\mathfrak{A}\mathfrak{G} \ a), c)\bullet)$ 
   $Y'(\text{NTDom}) = N'$ 
   $Y'(\text{NTCod}) = E'$ 
   $Y'(\text{NTDGDom}) = \text{op-cat } \mathfrak{A}$ 
   $Y'(\text{NTDGCod}) = \text{cat-Set } \beta$ 
  unfolding  $Y'$ -def nt-field-simps by (simp-all add: nat-omega-simps)

```

```

note [cat-cs-simps] =  $Y'$ -components(2-5)

```

```

have  $Y'$ -NTMap-app[cat-cs-simps]:
   $Y'(\text{NTMap})(a) = \text{ntcf-Yoneda } \alpha \beta \mathfrak{C}(\text{NTMap})(\text{cf-map } (?H\text{-}\mathfrak{A}\mathfrak{G} \ a), c)\bullet$ 
  if  $a \in_{\circ} \mathfrak{A}(\text{Obj})$  for  $a$ 
  using that unfolding  $Y'$ -components by simp

```

```

from  $\beta \alpha\beta$  interpret  $Y$ :
  is-iso-ntcf  $\beta \langle ?FUNCT \ \mathfrak{C} \times_{\mathfrak{C}} \ \mathfrak{C} \rangle \langle \text{cat-Set } \beta \rangle ?cf\text{-nt } ?cf\text{-eval } \langle \text{ntcf-Yoneda } \alpha \beta \mathfrak{C} \rangle$ 
  by (rule AG.HomCod.cat-ntcf-Yoneda-is-ntcf)

```

have $Y' : Y' : N' \mapsto_{CF.iso} E' : op-cat \mathfrak{A} \mapsto_{C\beta} cat-Set \beta$
proof(*intro is-iso-ntcfI is-ntcfI'*)

show *vfsequence* Y' **unfolding** Y' -*def* **by** *simp*
show *vcard* $Y' = 5_{\mathbb{N}}$
unfolding Y' -*def* **by** (*simp add: nat-omega-simps*)
show *vsv* ($Y'(\backslash NTMap)$) **unfolding** Y' -*components* **by** *auto*
show $\mathcal{D}_o (Y'(\backslash NTMap)) = op-cat \mathfrak{A}(\backslash Obj)$
unfolding Y' -*components* **by** (*simp add: cat-op-simps*)
show Y' -*NTMap-a*: $Y'(\backslash NTMap)(\backslash a) : N'(\backslash ObjMap)(\backslash a) \mapsto_{iso} cat-Set \beta E'(\backslash ObjMap)(\backslash a)$
if $a \in_o op-cat \mathfrak{A}(\backslash Obj)$ **for** a
using *that[unfolded cat-op-simps]* *assms*(\mathcal{B}) $\alpha\beta$
by
 (
 cs-concl
cs-simp: *cat-cs-simps cat-FUNCT-cs-simps cat-op-simps*
cs-intro:
cat-arrow-cs-intros
cat-cs-intros
cat-prod-cs-intros
cat-FUNCT-cs-intros
)

then show $Y'(\backslash NTMap)(\backslash a) : N'(\backslash ObjMap)(\backslash a) \mapsto_{cat-Set \beta} E'(\backslash ObjMap)(\backslash a)$
if $a \in_o op-cat \mathfrak{A}(\backslash Obj)$ **for** a
by (*intro cat-Set-is-iso-arrD[OF Y'-NTMap-a[OF that]]*)

show
 $Y'(\backslash NTMap)(\backslash b) \circ_{A} cat-Set \beta N'(\backslash ArrMap)(\backslash f) =$
 $E'(\backslash ArrMap)(\backslash f) \circ_{A} cat-Set \beta Y'(\backslash NTMap)(\backslash a)$
if $f : a \mapsto_{op-cat \mathfrak{A}} b$ **for** $a b f$

proof-
note $f =$ *that[unfolded cat-op-simps]*
from f *assms*(\mathcal{B}) **show** *?thesis*
by
 (
 cs-concl
cs-simp: *cat-cs-simps Y.ntcf-Comp-commute*
cs-intro: *cat-cs-intros cat-prod-cs-intros cat-FUNCT-cs-intros*
)+

qed
qed (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)+

have E' -*def*: $E' = Hom_{O.C\beta\mathfrak{A}}(-, ?\mathfrak{G}c)$
proof(*rule cf-eqI*)
show $E' : op-cat \mathfrak{A} \mapsto_{C\beta} cat-Set \beta$
by (*cs-concl cs-shallow cs-intro: cat-cs-intros*)
from *assms*(\mathcal{B}) **show**
 $Hom_{O.C\beta\mathfrak{A}}(-, ?\mathfrak{G}c) : op-cat \mathfrak{A} \mapsto_{C\beta} cat-Set \beta$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)
have *dom-lhs*: $\mathcal{D}_o (E'(\backslash ObjMap)) = \mathfrak{A}(\backslash Obj)$ **unfolding** E' -*components* **by** *simp*
from *assms*(\mathcal{B}) **have** *dom-rhs*:
 $\mathcal{D}_o (Hom_{O.C\beta\mathfrak{A}}(-, ?\mathfrak{G}c)(\backslash ObjMap)) = \mathfrak{A}(\backslash Obj)$
unfolding E' -*components*
by
 (
 cs-concl cs-shallow
cs-simp: *cat-cs-simps cat-op-simps cs-intro: cat-cs-intros*
)

show $E'(\text{ObjMap}) = \text{Hom}_{O.C\beta}\mathfrak{A}(-, ?\mathfrak{G}c)(\text{ObjMap})$
proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
fix a **assume** $a \in_o \mathfrak{A}(\text{Obj})$
with $\text{assms}(\beta)$ **show** $E'(\text{ObjMap})(a) = \text{Hom}_{O.C\beta}\mathfrak{A}(-, ?\mathfrak{G}c)(\text{ObjMap})(a)$
by
(

cs-concl
cs-simp: *cat-op-simps cat-cs-simps*
cs-intro: *cat-cs-intros cat-op-intros*
)

qed (*auto simp: E'-components cat-cs-intros assms(\beta)*)

have *dom-lhs*: $\mathcal{D}_o(E'(\text{ArrMap})) = \mathfrak{A}(\text{Arr})$ **unfolding** *E'-components by simp*
from $\text{assms}(\beta)$ **have** *dom-rhs*:
 $\mathcal{D}_o(\text{Hom}_{O.C\beta}\mathfrak{A}(-, ?\mathfrak{G}c)(\text{ArrMap})) = \mathfrak{A}(\text{Arr})$
unfolding *E'-components*
by
(

cs-concl cs-shallow
cs-simp: *cat-cs-simps cat-op-simps* **cs-intro**: *cat-cs-intros*
)

show $E'(\text{ArrMap}) = \text{Hom}_{O.C\beta}\mathfrak{A}(-, ?\mathfrak{G}c)(\text{ArrMap})$
proof(rule vsv-eqI, unfold dom-lhs dom-rhs)

fix f **assume** $\text{prems}: f \in_o \mathfrak{A}(\text{Arr})$
then obtain a b **where** $f: a \mapsto_{\mathfrak{A}} b$ **by** *auto*
have [*cat-cs-simps*]:
cf-eval-arrow \mathfrak{C} (*ntcf-arrow* ($?H\text{-A}\mathfrak{G} f$)) ($\mathfrak{C}(\text{CId})(c)$) =
cf-hom \mathfrak{A} [$f, \mathfrak{A}(\text{CId})(?\mathfrak{G}c)$].
(is $\langle ?\text{cf-eval-arrow} = ?\text{cf-hom-f}\mathfrak{G}c \rangle$)

proof-
have *cf-eval-arrow-f-CId-Gc*:
?cf-eval-arrow :
 $\text{Hom } \mathfrak{A} b ?\mathfrak{G}c \mapsto_{\text{cat-Set } \alpha} \text{Hom } \mathfrak{A} a ?\mathfrak{G}c$
proof(rule *cf-eval-arrow-is-arr'*)
from f **show** $?H\text{-A}\mathfrak{G} f : ?H\text{-A}\mathfrak{G} b \mapsto_{CF} ?H\text{-A}\mathfrak{G} a : \mathfrak{C} \mapsto_{C\alpha} \text{cat-Set } \alpha$
by (*cs-concl cs-intro: cat-cs-intros*)
qed
(

use f assms(\beta) in
 \langle
cs-concl
cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-op-intros
 \rangle
)

from f $\text{assms}(\beta)$ **have** *dom-lhs*:
 $\mathcal{D}_o(?cf\text{-eval-arrow}(\text{ArrVal})) = \text{Hom } \mathfrak{A} b ?\mathfrak{G}c$
by
(

cs-concl
cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-op-intros*
)

from $\text{assms}(\beta)$ f *Ran.HomCod.category-axioms* **have** *cf-hom-fGc*:
?cf-hom-fGc :
 $\text{Hom } \mathfrak{A} b ?\mathfrak{G}c \mapsto_{\text{cat-Set } \alpha} \text{Hom } \mathfrak{A} a ?\mathfrak{G}c$
by
(

```

    cs-concl cs-shallow cs-intro:
      cat-cs-intros cat-prod-cs-intros cat-op-intros
  )
from f assms( $\beta$ ) have dom-rhs:
   $\mathcal{D}_o$  (?cf-hom-f $\mathfrak{G}c$ (ArrVal)) = Hom  $\mathfrak{A}$  b ? $\mathfrak{G}c$ 
by
  (
    cs-concl cs-shallow
    cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-op-intros
  )

show ?thesis
proof(rule arr-Set-eqI)
from cf-eval-arrow-f-CId- $\mathfrak{G}c$  show arr-Set  $\alpha$  ?cf-eval-arrow
by (auto dest: cat-Set-is-arrD(1))
from cf-hom-f $\mathfrak{G}c$  show arr-Set  $\alpha$  ?cf-hom-f $\mathfrak{G}c$ 
by (auto dest: cat-Set-is-arrD(1))
show ?cf-eval-arrow(ArrVal) = ?cf-hom-f $\mathfrak{G}c$ (ArrVal)
proof(rule vsv-eqI, unfold dom-lhs dom-rhs, unfold in-Hom-iff)
from f assms( $\beta$ ) show vsv (?cf-eval-arrow(ArrVal))
by (cs-concl cs-intro: cat-cs-intros)
from f assms( $\beta$ ) show vsv (?cf-hom-f $\mathfrak{G}c$ (ArrVal))
by
  (
    cs-concl cs-shallow
    cs-simp: cat-cs-simps cat-op-simps
    cs-intro: cat-cs-intros cat-op-intros
  )
fix g assume g : b  $\mapsto_{\mathfrak{A}}$  ? $\mathfrak{G}c$ 
with f assms( $\beta$ ) show
  ?cf-eval-arrow(ArrVal)(g) = ?cf-hom-f $\mathfrak{G}c$ (ArrVal)(g)
by
  (
    cs-concl
    cs-simp: cat-cs-simps cat-op-simps
    cs-intro: cat-cs-intros cat-op-intros
  )
qed simp

qed
  (
    use cf-eval-arrow-f-CId- $\mathfrak{G}c$  cf-hom-f $\mathfrak{G}c$  in
     $\langle$ cs-concl cs-simp: cat-cs-simps $\rangle$ 
  )+

qed

from f prems assms( $\beta$ ) show  $E'$ (ArrMap)(f) = Hom $_{O.C\beta}$  $\mathfrak{A}(-, ?\mathfrak{G}c)$ (ArrMap)(f)
by
  (
    cs-concl
    cs-simp: cat-op-simps cat-cs-simps
    cs-intro: cat-cs-intros cat-op-intros
  )

qed (auto simp:  $E'$ -components cat-cs-intros assms( $\beta$ ))

qed simp-all

```

from Y' **have** $inv\text{-}Y'$: $inv\text{-}ntcf\ Y'$:
 $Hom_{O.C\beta}\mathfrak{A}(-, ?\mathfrak{G}c) \mapsto_{CF.iso} N' : op\text{-}cat\ \mathfrak{A} \mapsto_{C\beta} cat\text{-}Set\ \beta$
unfolding E' -def **by** (*auto intro: iso-ntcf-is-iso-arr*)

interpret N'' : $is\text{-}functor\ \beta \langle op\text{-}cat\ \mathfrak{A} \rangle \langle cat\text{-}Set\ \beta \rangle \langle L\text{-}10\text{-}5\text{-}N\ \alpha\ \beta\ \mathfrak{T}\ \mathfrak{R}\ c \rangle$
by (*rule L-10-5-N-is-functor[OF $\beta\ \alpha\beta$ assms]*)

define $\psi :: V$
where $\psi =$
 $[$
 $(\lambda a \in_o \mathfrak{A}(\text{Obj}). ?ntcf\text{-}ua\text{-}fo\ a(\text{NTMap})(cf\text{-}map\ (?H\text{-}\mathfrak{C}\ c))),$
 N' ,
 $L\text{-}10\text{-}5\text{-}N\ \alpha\ \beta\ \mathfrak{T}\ \mathfrak{R}\ c,$
 $op\text{-}cat\ \mathfrak{A},$
 $cat\text{-}Set\ \beta$
 $]$ o

have ψ -components:
 $\psi(\text{NTMap}) = (\lambda a \in_o \mathfrak{A}(\text{Obj}). ?ntcf\text{-}ua\text{-}fo\ a(\text{NTMap})(cf\text{-}map\ (?H\text{-}\mathfrak{C}\ c)))$
 $\psi(\text{NTDom}) = N'$
 $\psi(\text{NTCod}) = L\text{-}10\text{-}5\text{-}N\ \alpha\ \beta\ \mathfrak{T}\ \mathfrak{R}\ c$
 $\psi(\text{NTDGDom}) = op\text{-}cat\ \mathfrak{A}$
 $\psi(\text{NTDGCod}) = cat\text{-}Set\ \beta$
unfolding ψ -def *nt-field-simps* **by** (*simp-all add: nat-omega-simps*)

note [*cat-cs-simps*] = Y' -components(2-5)

have ψ -NTMap-app[*cat-cs-simps*]:
 $\psi(\text{NTMap})(a) = ?ntcf\text{-}ua\text{-}fo\ a(\text{NTMap})(cf\text{-}map\ (?H\text{-}\mathfrak{C}\ c))$
if $a \in_o \mathfrak{A}(\text{Obj})$ **for** a
using *that* **unfolding** ψ -components **by** *simp*

have $\psi : \psi : N' \mapsto_{CF.iso} L\text{-}10\text{-}5\text{-}N\ \alpha\ \beta\ \mathfrak{T}\ \mathfrak{R}\ c : op\text{-}cat\ \mathfrak{A} \mapsto_{C\beta} cat\text{-}Set\ \beta$
proof-

show *?thesis*
proof(*intro is-iso-ntcfI is-ntcfI'*)

show *vfsequence* ψ **unfolding** ψ -def **by** *auto*
show *vcard* $\psi = 5_N$ **unfolding** ψ -def **by** (*simp-all add: nat-omega-simps*)
show $N' : op\text{-}cat\ \mathfrak{A} \mapsto_{C\beta} cat\text{-}Set\ \beta$ **by** (*rule N'*)
show $L\text{-}10\text{-}5\text{-}N\ \alpha\ \beta\ \mathfrak{T}\ \mathfrak{R}\ c : op\text{-}cat\ \mathfrak{A} \mapsto_{C\beta} cat\text{-}Set\ \beta$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)
show $\psi(\text{NTDom}) = N'$ **unfolding** ψ -components **by** *simp*
show $\psi(\text{NTCod}) = L\text{-}10\text{-}5\text{-}N\ \alpha\ \beta\ \mathfrak{T}\ \mathfrak{R}\ c$ **unfolding** ψ -components **by** *simp*
show $\psi(\text{NTDGDom}) = op\text{-}cat\ \mathfrak{A}$ **unfolding** ψ -components **by** *simp*
show $\psi(\text{NTDGCod}) = cat\text{-}Set\ \beta$ **unfolding** ψ -components **by** *simp*
show *vsv* ($\psi(\text{NTMap})$) **unfolding** ψ -components **by** *simp*
show $\mathcal{D}_o(\psi(\text{NTMap})) = op\text{-}cat\ \mathfrak{A}(\text{Obj})$
unfolding ψ -components **by** (*simp add: cat-op-simps*)

show ψ -NTMap-is-iso-arr[*unfolded cat-op-simps*]:
 $\psi(\text{NTMap})(a) : N'(\text{ObjMap})(a) \mapsto_{iso} cat\text{-}Set\ \beta\ L\text{-}10\text{-}5\text{-}N\ \alpha\ \beta\ \mathfrak{T}\ \mathfrak{R}\ c(\text{ObjMap})(a)$
if $a \in_o op\text{-}cat\ \mathfrak{A}(\text{Obj})$ **for** a

proof-

note $a = \text{that}[\text{unfolded cat-op-simps}]$

interpret ε :

$\text{is-cat-rKe-preserves } \alpha \mathfrak{B} \mathfrak{C} \mathfrak{A} \langle \text{cat-Set } \alpha \rangle \mathfrak{K} \mathfrak{T} \mathfrak{G} \langle ?H\text{-}\mathfrak{A} \ a \rangle \varepsilon$

by ($\text{rule cat-pw-rKe-preserved}[OF \ a]$)

interpret $a\varepsilon$:

$\text{is-cat-rKe } \alpha \mathfrak{B} \mathfrak{C} \langle \text{cat-Set } \alpha \rangle \mathfrak{K} \langle ?H\text{-}\mathfrak{A}\mathfrak{T} \ a \rangle \langle ?H\text{-}\mathfrak{A}\mathfrak{G} \ a \rangle \langle ?H\text{-}\mathfrak{A}\varepsilon \ a \rangle$

by ($\text{rule } \varepsilon.\text{cat-rKe-preserves}$)

interpret is-iso-ntcf

β

$\langle \text{op-cat } (?FUNCT \ \mathfrak{C}) \rangle$

$\langle \text{cat-Set } \beta \rangle$

$\langle ?H\text{-}FUNCT \ \mathfrak{C} \ (?H\text{-}\mathfrak{A}\mathfrak{G} \ a) \rangle$

$\langle ?H\text{-}FUNCT \ \mathfrak{B} \ (?H\text{-}\mathfrak{A}\mathfrak{T} \ a) \circ_{CF} \text{op-cf } ?SET\text{-}\mathfrak{K} \rangle$

$\langle ?ntcf\text{-ua-fo } a \rangle$

by ($\text{rule } a\varepsilon.\text{cat-rKe-ntcf-ua-fo-is-iso-ntcf-if-ge-Limit}[OF \ \beta \ \alpha\beta]$)

have $\text{cf-map } (?H\text{-}\mathfrak{C} \ c) \in_{\circ} ?FUNCT \ \mathfrak{C}(\text{Obj})$

by

(

$\text{cs-concl cs-shallow}$

$\text{cs-simp: cat-cs-simps cat-FUNCT-cs-simps}$

$\text{cs-intro: cat-cs-intros cat-FUNCT-cs-intros}$

)

from

$\text{iso-ntcf-is-iso-arr}[\text{unfolded cat-op-simps}, OF \ \text{this}]$

$a \text{ assms } \alpha\beta$

show $?thesis$

by

(

cs-prems

cs-simp:

$\text{cat-cs-simps cat-Kan-cs-simps cat-FUNCT-cs-simps cat-op-simps}$

cs-intro:

$\text{cat-small-cs-intros}$

cat-Kan-cs-intros

cat-cs-intros

$\text{cat-FUNCT-cs-intros}$

cat-op-intros

)

qed

show $\psi\text{-NTMap-is-arr}[\text{unfolded cat-op-simps}]$:

$\psi(\text{NTMap})(\text{Obj}) : N'(\text{ObjMap})(\text{Obj}) \mapsto_{\text{cat-Set } \beta} L\text{-10-5-N } \alpha \beta \mathfrak{T} \mathfrak{K} \ c(\text{ObjMap})(\text{Obj})$

if $a \in_{\circ} \text{op-cat } \mathfrak{A}(\text{Obj})$ **for** a

by

(

$\text{rule cat-Set-is-iso-arrD}[$

$OF \ \psi\text{-NTMap-is-iso-arr}[OF \ \text{that}[\text{unfolded cat-op-simps}]]$

$]$

)

show

$\psi(\text{NTMap})(\text{Obj}) \circ_{A \text{ cat-Set } \beta} N'(\text{ArrMap})(f) =$

$L\text{-10-5-N } \alpha \beta \mathfrak{T} \mathfrak{K} \ c(\text{ArrMap})(f) \circ_{A \text{ cat-Set } \beta} \psi(\text{NTMap})(\text{Obj})$

if $f : a \mapsto_{\text{op-cat } \mathfrak{A}} b$ **for** $a \ b \ f$

proof-

note $f = \text{that}[\text{unfolded cat-op-simps}]$

from f **have** $a : a \in_{\circ} \mathfrak{A}(\text{Obj})$ **and** $b : b \in_{\circ} \mathfrak{A}(\text{Obj})$ **by** auto

interpret $p\text{-}a\text{-}\varepsilon$:
 $is\text{-}cat\text{-}rKe\text{-}preserves \alpha \mathfrak{B} \mathfrak{C} \mathfrak{A} \langle cat\text{-}Set \alpha \rangle \mathfrak{K} \mathfrak{T} \mathfrak{G} \langle ?H\text{-}\mathfrak{A} a \rangle \varepsilon$
by (rule $cat\text{-}pw\text{-}rKe\text{-}preserved[OF a]$)

interpret $a\text{-}\varepsilon$: $is\text{-}cat\text{-}rKe$
 $\alpha \mathfrak{B} \mathfrak{C} \langle cat\text{-}Set \alpha \rangle \mathfrak{K} \langle ?H\text{-}\mathfrak{A}\mathfrak{T} a \rangle \langle ?H\text{-}\mathfrak{A}\mathfrak{G} a \rangle \langle ?H\text{-}\mathfrak{A}\varepsilon a \rangle$
by (rule $p\text{-}a\text{-}\varepsilon.cat\text{-}rKe\text{-}preserves$)

interpret $ntcf\text{-}ua\text{-}fo\text{-}a\text{-}\varepsilon$: $is\text{-}iso\text{-}ntcf$
 $\beta \text{?}ua\text{-}NTDGDom \langle cat\text{-}Set \beta \rangle \langle \text{?}ua\text{-}NTDom a \rangle \langle \text{?}ua\text{-}NTCod a \rangle \langle \text{?}ua a \rangle$
by (rule $a\text{-}\varepsilon.cat\text{-}rKe\text{-}ntcf\text{-}ua\text{-}fo\text{-}is\text{-}iso\text{-}ntcf\text{-}if\text{-}ge\text{-}Limit[OF \beta \alpha\beta]$)

interpret $p\text{-}b\text{-}\varepsilon$:
 $is\text{-}cat\text{-}rKe\text{-}preserves \alpha \mathfrak{B} \mathfrak{C} \mathfrak{A} \langle cat\text{-}Set \alpha \rangle \mathfrak{K} \mathfrak{T} \mathfrak{G} \langle ?H\text{-}\mathfrak{A} b \rangle \varepsilon$
by (rule $cat\text{-}pw\text{-}rKe\text{-}preserved[OF b]$)

interpret $b\text{-}\varepsilon$: $is\text{-}cat\text{-}rKe$
 $\alpha \mathfrak{B} \mathfrak{C} \langle cat\text{-}Set \alpha \rangle \mathfrak{K} \langle ?H\text{-}\mathfrak{A}\mathfrak{T} b \rangle \langle ?H\text{-}\mathfrak{A}\mathfrak{G} b \rangle \langle ?H\text{-}\mathfrak{A}\varepsilon b \rangle$
by (rule $p\text{-}b\text{-}\varepsilon.cat\text{-}rKe\text{-}preserves$)

interpret $ntcf\text{-}ua\text{-}fo\text{-}b\text{-}\varepsilon$: $is\text{-}iso\text{-}ntcf$
 $\beta \text{?}ua\text{-}NTDGDom \langle cat\text{-}Set \beta \rangle \langle \text{?}ua\text{-}NTDom b \rangle \langle \text{?}ua\text{-}NTCod b \rangle \langle \text{?}ua b \rangle$
by (rule $b\text{-}\varepsilon.cat\text{-}rKe\text{-}ntcf\text{-}ua\text{-}fo\text{-}is\text{-}iso\text{-}ntcf\text{-}if\text{-}ge\text{-}Limit[OF \beta \alpha\beta]$)

interpret $\mathfrak{K}\text{-}SET$: $is\text{-}tiny\text{-}functor \beta \langle ?FUNCT \mathfrak{C} \rangle \langle ?FUNCT \mathfrak{B} \rangle \text{?}SET\text{-}\mathfrak{K}$
by
(
rule $exp\text{-}cat\text{-}cf\text{-}is\text{-}tiny\text{-}functor[$
 $OF \beta \alpha\beta AG.category\text{-}cat\text{-}Set AG.is\text{-}functor\text{-}axioms$
 $]$
)
)
from f **interpret** $Hom\text{-}f$:
 $is\text{-}ntcf \alpha \mathfrak{A} \langle cat\text{-}Set \alpha \rangle \langle ?H\text{-}\mathfrak{A} a \rangle \langle ?H\text{-}\mathfrak{A} b \rangle \langle ?H\text{-}A f \rangle$
by ($cs\text{-}concl$ **cs-intro**: $cat\text{-}cs\text{-}intros$)

let $\text{?}cf\text{-}hom\text{-}lhs =$
 \langle
 $cf\text{-}hom$
 $(?FUNCT \mathfrak{C})$
 $[ntcf\text{-}arrow (ntcf\text{-}id (?H\text{-}\mathfrak{C} c)), ntcf\text{-}arrow (?H\text{-}A\mathfrak{G} f)].$
 \rangle

let $\text{?}cf\text{-}hom\text{-}rhs =$
 \langle
 $cf\text{-}hom$
 $(?FUNCT \mathfrak{B})$
 $[$
 $ntcf\text{-}arrow (ntcf\text{-}id (?H\text{-}\mathfrak{C} c \circ_{CF} \mathfrak{K})),$
 $ntcf\text{-}arrow (?H\text{-}A f \circ_{NTCF\text{-}CF} \mathfrak{T})$
 $].$
 \rangle

let $\text{?}dom =$
 $\langle Hom (?FUNCT \mathfrak{C}) (cf\text{-}map (?H\text{-}\mathfrak{C} c)) (cf\text{-}map (?H\text{-}\mathfrak{A}\mathfrak{G} a)) \rangle$
let $\text{?}cod = \langle Hom (?FUNCT \mathfrak{B}) (cf\text{-}map (?H\text{-}\mathfrak{C}\mathfrak{K} c)) (cf\text{-}map (?H\text{-}\mathfrak{A}\mathfrak{T} b)) \rangle$
let $\text{?}cf\text{-}hom\text{-}lhs\text{-}umap\text{-}fo\text{-}inter =$
 $\langle Hom (?FUNCT \mathfrak{C}) (cf\text{-}map (?H\text{-}\mathfrak{C} c)) (cf\text{-}map (?H\text{-}\mathfrak{A}\mathfrak{G} b)) \rangle$
let $\text{?}umap\text{-}fo\text{-}cf\text{-}hom\text{-}rhs\text{-}inter =$
 $\langle Hom (?FUNCT \mathfrak{B}) (cf\text{-}map (?H\text{-}\mathfrak{C}\mathfrak{K} c)) (cf\text{-}map (?H\text{-}\mathfrak{A}\mathfrak{T} a)) \rangle$

have [$cat\text{-}cs\text{-}simps$]:
 $\text{?}umap\text{-}fo b \circ_{A\text{ }cat\text{-}Set} \beta \text{?}cf\text{-}hom\text{-}lhs =$
 $\text{?}cf\text{-}hom\text{-}rhs \circ_{A\text{ }cat\text{-}Set} \beta \text{?}umap\text{-}fo a$

proof-

```
from  $f$  assms( $\beta$ )  $\alpha\beta$  have cf-hom-lhs:  
  ?cf-hom-lhs : ?dom  $\mapsto$  cat-Set  $\beta$  ?cf-hom-lhs-umap-fo-inter  
by  
  (  
    cs-concl  
    cs-simp: cat-cs-simps cat-FUNCT-cs-simps  
    cs-intro:  
      cat-cs-intros  
      cat-FUNCT-cs-intros  
      cat-prod-cs-intros  
      cat-op-intros  
  )  
from  $f$  assms( $\beta$ )  $\alpha\beta$  have umap-fo-b:  
  ?umap-fo b : ?cf-hom-lhs-umap-fo-inter  $\mapsto$  cat-Set  $\beta$  ?cod  
by  
  (  
    cs-concl  
    cs-simp: cat-cs-simps cat-FUNCT-cs-simps  
    cs-intro:  
      cat-cs-intros  
      cat-FUNCT-cs-intros  
      cat-prod-cs-intros  
      cat-op-intros  
  )  
from cf-hom-lhs umap-fo-b have umap-fo-cf-hom-lhs:  
  ?umap-fo b  $\circ_A$  cat-Set  $\beta$  ?cf-hom-lhs : ?dom  $\mapsto$  cat-Set  $\beta$  ?cod  
by  
  (  
    cs-concl cs-shallow  
    cs-simp: cat-cs-simps cs-intro: cat-cs-intros  
  )  
then have dom-umap-fo-cf-hom-lhs:  
   $\mathcal{D}_\circ ((?umap-fo b \circ_A cat-Set \beta ?cf-hom-lhs)(\downarrow ArrVal)) = ?dom$   
by  
  (  
    cs-concl cs-shallow  
    cs-simp: cat-cs-simps cs-intro: cat-cs-intros  
  )  
  
from  $f$  assms( $\beta$ )  $\alpha\beta$  have cf-hom-rhs:  
  ?cf-hom-rhs : ?umap-fo-cf-hom-rhs-inter  $\mapsto$  cat-Set  $\beta$  ?cod  
by  
  (  
    cs-concl  
    cs-simp: cat-cs-simps cat-FUNCT-cs-simps  
    cs-intro:  
      cat-cs-intros  
      cat-FUNCT-cs-intros  
      cat-prod-cs-intros  
      cat-op-intros  
  )  
from  $f$  assms( $\beta$ )  $\alpha\beta$  have umap-fo-a:  
  ?umap-fo a : ?dom  $\mapsto$  cat-Set  $\beta$  ?umap-fo-cf-hom-rhs-inter  
by  
  (  
    cs-concl
```

```

cs-simp: cat-cs-simps cat-FUNCT-cs-simps
cs-intro:
  cat-cs-intros
  cat-FUNCT-cs-intros
  cat-prod-cs-intros
  cat-op-intros
)
from cf-hom-rhs umap-fo-a have cf-hom-rhs-umap-fo-a:
  ?cf-hom-rhs  $\circ_A$  cat-Set  $\beta$  ?umap-fo a : ?dom  $\mapsto$  cat-Set  $\beta$  ?cod
by
  (
    cs-concl cs-shallow
    cs-simp: cat-cs-simps cs-intro: cat-cs-intros
  )
then have dom-cf-hom-rhs-umap-fo-a:
   $\mathcal{D}_\circ ((?cf-hom-rhs \circ_A cat-Set \beta ?umap-fo a)(ArrVal)) = ?dom$ 
by
  (
    cs-concl cs-shallow
    cs-simp: cat-cs-simps cs-intro: cat-cs-intros
  )

```

show *?thesis*

proof(*rule arr-Set-eqI*)

from *umap-fo-cf-hom-lhs* **show** *arr-Set-umap-fo-cf-hom-lhs:*

arr-Set β (*?umap-fo b* \circ_A *cat-Set* β *?cf-hom-lhs*)

by (*auto dest: cat-Set-is-arrD(1)*)

from *cf-hom-rhs-umap-fo-a* **show** *arr-Set-cf-hom-rhs-umap-fo-a:*

arr-Set β (*?cf-hom-rhs* \circ_A *cat-Set* β *?umap-fo a*)

by (*auto dest: cat-Set-is-arrD(1)*)

show

(*?umap-fo b* \circ_A *cat-Set* β *?cf-hom-lhs*)(*ArrVal*) =

(*?cf-hom-rhs* \circ_A *cat-Set* β *?umap-fo a*)(*ArrVal*)

proof

(

rule vsv-eqI,

unfold

dom-umap-fo-cf-hom-lhs dom-cf-hom-rhs-umap-fo-a in-Hom-iff;

(*rule refl*)?

)

fix \mathfrak{H} **assume** *prems:*

$\mathfrak{H} : cf-map$ (*?H-C* *c*) \mapsto *?FUNCT* \mathfrak{C} *cf-map* (*?H-A* \mathfrak{G} *a*)

let \mathfrak{H} = $\langle ntcf-of-ntcf-arrow \mathfrak{C} (cat-Set \alpha) \mathfrak{H} \rangle$

let $\mathfrak{H}\varepsilon$ = $\langle ?H-A\varepsilon b \cdot_{NTCF} ((?H-A\mathfrak{G} f \cdot_{NTCF} ?\mathfrak{H}) \circ_{NTCF-CF} \mathfrak{R}) \rangle$

let $\mathfrak{H}\mathfrak{R}$ =

$\langle (?H-A f \circ_{NTCF-CF} \mathfrak{T} \cdot_{NTCF} ?H-A\varepsilon a \cdot_{NTCF} (?\mathfrak{H} \circ_{NTCF-CF} \mathfrak{R})) \rangle$

let $\mathfrak{H}\mathfrak{A}\varepsilon$ = $\langle \lambda b b'. cf-hom \mathfrak{A} [\mathfrak{A}(\mathcal{C}Id)(b), \varepsilon(NTMap)(b')] \rangle_\circ$

let $\mathfrak{H}\mathfrak{Y}_c$ = $\langle \lambda Q. Yoneda-component (?H-A b) a f Q \rangle$

let $\mathfrak{H}\mathfrak{R}$ = $\langle \lambda b b'. ?\mathfrak{H}(\mathcal{N}TMap)(\mathfrak{R}(\mathcal{O}bjMap)(b')) \rangle$

let $\mathfrak{H}\mathfrak{G}\mathfrak{R}$ = $\langle \lambda b b'. \mathfrak{G}(\mathcal{O}bjMap)(\mathfrak{R}(\mathcal{O}bjMap)(b')) \rangle$

have [*cat-cs-simps*]:

cf-of-cf-map $\mathfrak{C} (cat-Set \alpha) (cf-map (?H-C c)) = ?H-C c$

by

```

(
  cs-concl cs-shallow
  cs-simp: cat-FUNCT-cs-simps cs-intro: cat-cs-intros
)
have [cat-cs-simps]:
  cf-of-cf-map  $\mathfrak{C}$  (cat-Set  $\alpha$ ) (cf-map ( $?H\text{-}\mathfrak{A}\mathfrak{G}$   $a$ )) =  $?H\text{-}\mathfrak{A}\mathfrak{G}$   $a$ 
  by
  (
    cs-concl cs-shallow
    cs-simp: cat-FUNCT-cs-simps cs-intro: cat-cs-intros
  )
note  $\mathfrak{H} = \text{cat-FUNCT-is-arrD}[OF \text{ prems, unfolded cat-cs-simps}]$ 
have Hom-c:  $?H\text{-}\mathfrak{C}\mathfrak{R}$   $c : \mathfrak{B} \mapsto \mapsto_{C\alpha} \text{cat-Set } \alpha$ 
  by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)

have [cat-cs-simps]:  $?lhs = ?rhs$ 
proof(rule ntcf-eqI)
  from  $\mathfrak{H}(1)$  f show lhs:
     $?lhs : ?H\text{-}\mathfrak{C}\mathfrak{R}$   $c \mapsto_{CF} ?H\text{-}\mathfrak{A}\mathfrak{T}$   $b : \mathfrak{B} \mapsto \mapsto_{C\alpha} \text{cat-Set } \alpha$ 
    by (cs-concl cs-simp: cs-intro: cat-cs-intros)
  then have dom-lhs:  $\mathcal{D}_\circ (?lhs(\text{NTMap})) = \mathfrak{B}(\text{Obj})$ 
    by (cs-concl cs-simp: cat-cs-simps)+
  from  $\mathfrak{H}(1)$  f show rhs:
     $?rhs : ?H\text{-}\mathfrak{C}\mathfrak{R}$   $c \mapsto_{CF} ?H\text{-}\mathfrak{A}\mathfrak{T}$   $b : \mathfrak{B} \mapsto \mapsto_{C\alpha} \text{cat-Set } \alpha$ 
    by (cs-concl cs-intro: cat-cs-intros)
  then have dom-rhs:  $\mathcal{D}_\circ (?rhs(\text{NTMap})) = \mathfrak{B}(\text{Obj})$ 
    by (cs-concl cs-simp: cat-cs-simps)+
  have [cat-cs-simps]:
     $?cf\text{-hom-}\mathfrak{A}\varepsilon$   $b$   $b' \circ_{A \text{cat-Set } \alpha}$ 
    ( $?Yc$  ( $?G\mathfrak{R}$   $b'$ )  $\circ_{A \text{cat-Set } \alpha}$   $?H\mathfrak{R}$   $b'$ ) =
     $?Yc$  ( $\mathfrak{T}(\text{ObjMap})(\text{Obj})$ )  $\circ_{A \text{cat-Set } \alpha}$ 
    ( $?cf\text{-hom-}\mathfrak{A}\varepsilon$   $a$   $b' \circ_{A \text{cat-Set } \alpha}$   $?H\mathfrak{R}$   $b'$ )
    (is  $\langle ?lhs\text{-Set} = ?rhs\text{-Set} \rangle$ )
    if  $b' \in_\circ \mathfrak{B}(\text{Obj})$  for  $b'$ 
  proof-
    let  $?Rb' = \langle \mathfrak{R}(\text{ObjMap})(\text{Obj}) \rangle$ 
    from  $\mathfrak{H}(1)$  f that assms( $\mathfrak{B}$ ) Ran.HomCod.category-axioms
    have lhs-Set-is-arr:  $?lhs\text{-Set} :$ 
       $\text{Hom } \mathfrak{C} \ c \ (\?Rb') \mapsto_{\text{cat-Set } \alpha} \text{Hom } \mathfrak{A} \ b \ (\mathfrak{T}(\text{ObjMap})(\text{Obj}))$ 
      by
      (
        cs-concl
        cs-simp: cat-cs-simps cat-op-simps
        cs-intro:
          cat-cs-intros cat-prod-cs-intros cat-op-intros
      )
    then have dom-lhs-Set:  $\mathcal{D}_\circ (?lhs\text{-Set}(\text{ArrVal})) = \text{Hom } \mathfrak{C} \ c \ ?Rb'$ 
      by (cs-concl cs-shallow cs-simp: cat-cs-simps)
    from  $\mathfrak{H}(1)$  f that assms( $\mathfrak{B}$ ) Ran.HomCod.category-axioms
    have rhs-Set-is-arr:  $?rhs\text{-Set} :$ 
       $\text{Hom } \mathfrak{C} \ c \ (\?Rb') \mapsto_{\text{cat-Set } \alpha} \text{Hom } \mathfrak{A} \ b \ (\mathfrak{T}(\text{ObjMap})(\text{Obj}))$ 
      by
      (
        cs-concl
        cs-simp: cat-cs-simps cat-op-simps
        cs-intro:
          cat-cs-intros cat-prod-cs-intros cat-op-intros
      )
  )

```

then have $dom\text{-}rhs\text{-}Set: \mathcal{D}_o (?rhs\text{-}Set(\downarrow ArrVal)) = Hom \mathfrak{C} c ?\mathfrak{R}b'$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps*)
show *?thesis*
proof(*rule arr-Set-eqI*)
from *lhs-Set-is-arr show arr-Set-lhs-Set: arr-Set α ?lhs-Set*
by (*auto dest: cat-Set-is-arrD(1)*)
from *rhs-Set-is-arr show arr-Set-rhs-Set: arr-Set α ?rhs-Set*
by (*auto dest: cat-Set-is-arrD(1)*)
show $?lhs\text{-}Set(\downarrow ArrVal) = ?rhs\text{-}Set(\downarrow ArrVal)$
proof(*rule vsv-eqI, unfold dom-lhs-Set dom-rhs-Set in-Hom-iff*)
fix h **assume** $h : c \mapsto_{\mathfrak{C}} ?\mathfrak{R}b'$
with $\mathfrak{H}(1)$ *f that assms Ran.HomCod.category-axioms show*
 $?lhs\text{-}Set(\downarrow ArrVal)(h) = ?rhs\text{-}Set(\downarrow ArrVal)(h)$
by
(
cs-concl
cs-simp: *cat-cs-simps cat-op-simps*
cs-intro:
cat-cs-intros cat-prod-cs-intros cat-op-intros
)
qed (*use arr-Set-lhs-Set arr-Set-rhs-Set in auto*)
qed
(
use lhs-Set-is-arr rhs-Set-is-arr in
 $\langle i\text{-}cs\text{-}concl\ i\text{-}cs\text{-}shallow\ i\text{-}cs\text{-}simp: cat\text{-}cs\text{-}simps \rangle$
)+

qed

show $?lhs(\downarrow NTMap) = ?rhs(\downarrow NTMap)$
proof(*rule vsv-eqI, unfold dom-lhs dom-rhs*)
fix b' **assume** $b' \in_o \mathfrak{B}(\downarrow Obj)$
with $\mathfrak{H}(1)$ *f assms(3) show* $?lhs(\downarrow NTMap)(b') = ?rhs(\downarrow NTMap)(b')$
by
(
cs-concl
cs-simp: *cat-cs-simps cat-op-simps*
cs-intro: *cat-cs-intros*
)
qed (*cs-concl cs-intro: cat-cs-intros*)

qed *simp-all*

from

assms(3) f $\mathfrak{H}(1)$ prems $\alpha\beta$

Ran.HomCod.category-axioms

FUNCT- \mathfrak{C} .category-axioms

FUNCT- \mathfrak{B} .category-axioms

AG.is-functor-axioms

Ran.is-functor-axioms

Hom-f.is-ntcf-axioms

show

$(?umap\text{-}fo\ b \circ_A cat\text{-}Set\ \beta\ ?cf\text{-}hom\text{-}lhs)(\downarrow ArrVal)(\mathfrak{H}) =$

$(?cf\text{-}hom\text{-}rhs \circ_A cat\text{-}Set\ \beta\ ?umap\text{-}fo\ a)(\downarrow ArrVal)(\mathfrak{H})$

by (*subst (1 2) $\mathfrak{H}(2)$*)

(*cs-concl*)

```

cs-simp: cat-cs-simps cat-FUNCT-cs-simps cat-op-simps
cs-intro:
  cat-cs-intros
  cat-prod-cs-intros
  cat-FUNCT-cs-intros
  cat-op-intros
)
qed
(
  use arr-Set-umap-fo-cf-hom-lhs arr-Set-cf-hom-rhs-umap-fo-a in
  auto
)

qed
(
  use umap-fo-cf-hom-lhs cf-hom-rhs-umap-fo-a in
  ⟨cs-concl cs-shallow cs-simp: cat-cs-simps⟩
)+

qed

from f assms αβ show ?thesis
by
(
  cs-concl
  cs-simp: cat-cs-simps cat-Kan-cs-simps cat-FUNCT-cs-simps
  cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
)

qed

qed auto

qed

from L-10-5-χ-is-iso-ntcf[OF β αβ assms] have inv-χ:
inv-ntcf (L-10-5-χ α β ℑ ℝ c) :
L-10-5-N α β ℑ ℝ c ↦CF.iso cf-Cone α β ?ℑ-cℝ :
op-cat ℑ ↦Cβ cat-Set β
by (auto intro: iso-ntcf-is-iso-arr)

define φ where φ = inv-ntcf (L-10-5-χ α β ℑ ℝ c) •NTCF ψ •NTCF inv-ntcf Y'

from inv-Y' ψ inv-χ have φ: φ :
HomO.Cβℑ(-, ?ℑc) ↦CF.iso cf-Cone α β ?ℑ-cℝ :
op-cat ℑ ↦Cβ cat-Set β
unfolding φ-def by (cs-concl cs-shallow cs-intro: cat-cs-intros)

interpret φ: is-iso-ntcf
β ⟨op-cat ℑ⟩ ⟨cat-Set β⟩ ⟨HomO.Cβℑ(-, ?ℑc)⟩ ⟨cf-Cone α β ?ℑ-cℝ⟩ φ
by (rule φ)

let ?φ-ℑc-CId = ⟨φ(NTMap)(?ℑc)(Arr Val)(ℑ(CId)(?ℑc))⟩
let ?ntcf-φ-ℑc-CId = ⟨ntcf-of-ntcf-arrow (c ↓CF ℝ) ℑ ?φ-ℑc-CId⟩

```

from $AG.vempty-is-zet\ assms(\beta)$ **have** $\Delta: ?\Delta : \mathfrak{A} \mapsto \mapsto_{C\beta} ?c\mathfrak{K}\text{-}\mathfrak{A}$
by
(

cs-concl **cs-shallow**
cs-simp: *cat-comma-cs-simps*
cs-intro: *cat-cs-intros cat-comma-cs-intros*
)

from $assms(\beta)$ **have** $\mathfrak{G}c: ?\mathfrak{G}c \in_0 \mathfrak{A}(\text{Obj})$
by (*cs-concl* **cs-shallow** **cs-intro:** *cat-cs-intros*)
from $AG.vempty-is-zet$ **have** $\mathfrak{T}\text{-}c\mathfrak{K}: cf\text{-}map\ (\mathfrak{T}\text{-}c\mathfrak{K}) \in_0 ?c\mathfrak{K}\text{-}\mathfrak{A}(\text{Obj})$
by
(

cs-concl
cs-simp: *cat-FUNCT-components(1)*
cs-intro: *cat-cs-intros cat-FUNCT-cs-intros*
)

from
 $\varphi.ntcf\text{-}NTMap\text{-}is\text{-}arr[\text{unfolded}\ cat\text{-}op\text{-}simps, OF\ \mathfrak{G}c]$
 $assms(\beta)$
 $AG.vempty-is-zet$
 $\beta.vempty-is-zet$
 $\alpha\beta$
have $\varphi\text{-}\mathfrak{G}c: \varphi(\text{NTMap})(?\mathfrak{G}c) :$
 $Hom\ \mathfrak{A}\ ?\mathfrak{G}c?\mathfrak{G}c \mapsto_{cat\text{-}Set}\ \beta$
 $Hom\ ?c\mathfrak{K}\text{-}\mathfrak{A}\ (cf\text{-}map\ (?cf\text{-}c\mathfrak{K}\text{-}\mathfrak{A}\ ?\mathfrak{G}c))\ (cf\text{-}map\ ?\mathfrak{T}\text{-}c\mathfrak{K})$
by
(

cs-prems
cs-simp:
cat-cs-simps
cat-Kan-cs-simps
cat-comma-cs-simps
cat-op-simps
cat-FUNCT-components(1)
cs-intro:
cat-Kan-cs-intros
cat-comma-cs-intros
cat-cs-intros
cat-FUNCT-cs-intros
cat-op-intros
)

with $assms(\beta)$ **have** $\varphi\text{-}\mathfrak{G}c\text{-}CId:$
 $?\varphi\text{-}\mathfrak{G}c\text{-}CId : cf\text{-}map\ (?cf\text{-}c\mathfrak{K}\text{-}\mathfrak{A}\ ?\mathfrak{G}c) \mapsto_{?c\mathfrak{K}\text{-}\mathfrak{A}} cf\text{-}map\ ?\mathfrak{T}\text{-}c\mathfrak{K}$
by (*cs-concl* **cs-shallow** **cs-intro:** *cat-cs-intros*)
have $ntcf\text{-}arrow\text{-}\varphi\text{-}\mathfrak{G}c\text{-}CId: ntcf\text{-}arrow\ ?ntcf\text{-}\varphi\text{-}\mathfrak{G}c\text{-}CId = ?\varphi\text{-}\mathfrak{G}c\text{-}CId$
by (*rule* *cat-FUNCT-is-arrD(2)*[*OF* $\varphi\text{-}\mathfrak{G}c\text{-}CId$, *symmetric*])

have $ua: universal\text{-}arrow\text{-}fo\ ?\Delta\ (cf\text{-}map\ (?\mathfrak{T}\text{-}c\mathfrak{K}))\ ?\mathfrak{G}c\ ?\varphi\text{-}\mathfrak{G}c\text{-}CId$
by
(

rule *is-functor.cf-universal-arrow-fo-if-is-iso-ntcf*[
OF $\Delta\ \mathfrak{G}c\ \mathfrak{T}\text{-}c\mathfrak{K}\ \varphi[\text{unfolded}\ cf\text{-}Cone\text{-}def\ cat\text{-}cs\text{-}simps]$
]

)

moreover **have** $ntcf\text{-}\varphi\text{-}\mathfrak{G}c\text{-}CId:$
 $?ntcf\text{-}\varphi\text{-}\mathfrak{G}c\text{-}CId : ?\mathfrak{G}c <_{CF.cone}\ ?\mathfrak{T}\text{-}c\mathfrak{K} : c \downarrow_{CF}\ \mathfrak{K} \mapsto \mapsto_{C\alpha}\ \mathfrak{A}$

proof(*intro is-cat-coneI*)
from *cat-FUNCT-is-arrD(1)[OF φ - $\mathfrak{G}c$ -CId]* *assms(3)* *AG.vempty-is-zet* **show**
 $ntcf\text{-of-ntcf-arrow } (c \downarrow_{CF} \mathfrak{K}) \mathfrak{A} \ ?\varphi\text{-}\mathfrak{G}c\text{-CId} :$
 $\ ?cf\text{-}c\mathfrak{K}\text{-}\mathfrak{A} \ ?\mathfrak{G}c \mapsto_{CF} \ ?\mathfrak{T}\text{-}c\mathfrak{K} : c \downarrow_{CF} \mathfrak{K} \mapsto_{C\alpha} \mathfrak{A}$
by
 (
 cs-prems
 cs-simp: *cat-cs-simps cat-FUNCT-cs-simps*
 cs-intro: *cat-cs-intros cat-FUNCT-cs-intros*
)
qed (*rule $\mathfrak{G}c$*)
ultimately have $\ ?ntcf\text{-}\varphi\text{-}\mathfrak{G}c\text{-CId} : \ ?\mathfrak{G}c <_{CF.lim} \ ?\mathfrak{T}\text{-}c\mathfrak{K} : c \downarrow_{CF} \mathfrak{K} \mapsto_{C\alpha} \mathfrak{A}$
by (*intro is-cat-cone.cat-cone-is-cat-limit*)
 (*simp-all add: ntcf-arrow- φ - $\mathfrak{G}c$ -CId*)
then show *?thesis using that by auto*

qed

lemma (*in is-cat-pw-lKe*) *cat-pw-lKe-ex-cat-colimit:*

— Based on the elements of Chapter X-5 in [9].

assumes $\mathfrak{K} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{C}$

and $\mathfrak{T} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{A}$

and $c \in_{\circ} \mathfrak{C}(\text{Obj})$

obtains *UA*

where $UA : \mathfrak{T} \circ_{CF} \mathfrak{K} \text{ }_{CF} \sqcap \text{ }_O c >_{CF.colim} \mathfrak{F}(\text{ObjMap})(\downarrow c) : \mathfrak{K} \text{ }_{CF} \downarrow c \mapsto_{C\alpha} \mathfrak{A}$

proof–

from

is-cat-pw-rKe.cat-pw-rKe-ex-cat-limit

[
 OF *is-cat-pw-rKe-op AG.is-functor-op ntcf-lKe.NTDom.is-functor-op,*
unfolded cat-op-simps,
 OF *assms(3)*
]

obtain *UA* **where** $UA : UA :$

$\mathfrak{F}(\text{ObjMap})(\downarrow c) <_{CF.lim} op\text{-}cf \ \mathfrak{T} \circ_{CF} c \text{ }_O \sqcap \text{ }_{CF} (op\text{-}cf \ \mathfrak{K}) :$

$c \downarrow_{CF} (op\text{-}cf \ \mathfrak{K}) \mapsto_{C\alpha} op\text{-}cat \ \mathfrak{A}$

by *auto*

from *assms(3)* **have** [*cat-cs-simps*]:

$op\text{-}cf \ \mathfrak{T} \circ_{CF} c \text{ }_O \sqcap \text{ }_{CF} (op\text{-}cf \ \mathfrak{K}) \circ_{CF} op\text{-}cf\text{-}obj\text{-}comma \ \mathfrak{K} \ c =$

$op\text{-}cf \ \mathfrak{T} \circ_{CF} op\text{-}cf \ (\mathfrak{K} \text{ }_{CF} \sqcap \text{ }_O c)$

by

(
 cs-concl cs-shallow
 cs-simp: *cat-cs-simps AG.op-cf-cf-obj-comma-proj[OF assms(3)]*
 cs-intro: *cat-cs-intros cat-comma-cs-intros cat-op-intros*
)

from *assms(3)* **have** [*cat-op-simps*]:

$\mathfrak{T} \circ_{CF} op\text{-}cf \ (c \text{ }_O \sqcap \text{ }_{CF} (op\text{-}cf \ \mathfrak{K})) \circ_{CF} op\text{-}cf \ (op\text{-}cf\text{-}obj\text{-}comma \ \mathfrak{K} \ c) =$

$\mathfrak{T} \circ_{CF} \mathfrak{K} \text{ }_{CF} \sqcap \text{ }_O c$

by

(
 cs-concl cs-shallow
 cs-simp:
 cat-cs-simps cat-op-simps
 op-cf-cf-comp[symmetric] AG.op-cf-cf-obj-comma-proj[symmetric]
 cs-intro: *cat-cs-intros cat-comma-cs-intros cat-op-intros*
)

from *assms(3)* **have** [*cat-op-simps*]: $op\text{-}cat \ (op\text{-}cat \ (\mathfrak{K} \text{ }_{CF} \downarrow c)) = \mathfrak{K} \text{ }_{CF} \downarrow c$

by
 (

 cs-concl cs-shallow

 cs-simp: *cat-op-simps* **cs-intro:** *cat-cs-intros cat-comma-cs-intros*

)

note *ntcf-cf-comp-is-cat-limit-if-is-iso-functor* =
ntcf-cf-comp-is-cat-limit-if-is-iso-functor

 [

 OF UA AG.op-cf-obj-comma-is-iso-functor[*OF assms*(3)],

 unfolded cat-op-simps

]

have *op-ntcf UA* $\circ_{NTCF-CF}$ *op-cf* (*op-cf-obj-comma* \mathfrak{K} *c*) :
 $\mathfrak{T} \circ_{CF} \mathfrak{K} \sqcap_{O} c >_{CF.colim} \mathfrak{F}(\mathcal{O}bjMap)(c) : \mathfrak{K} \downarrow_{CF} c \mapsto_{C\alpha} \mathfrak{A}$

 by

 (

 rule is-cat-limit.is-cat-colimit-op

 [

 OF ntcf-cf-comp-is-cat-limit-if-is-iso-functor,

 unfolded cat-op-simps

]

)

then show *?thesis using that by auto*

qed

15.10 The limit and the colimit for the pointwise Kan extensions

15.10.1 Definition and elementary properties

See Theorem 3 in Chapter X-5 in [9].

definition *the-pw-cat-rKe-limit* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$

where *the-pw-cat-rKe-limit* α \mathfrak{K} \mathfrak{T} \mathfrak{G} *c* =

[

 $\mathfrak{G}(\mathcal{O}bjMap)(c)$,

 (

 SOME UA.

 UA : $\mathfrak{G}(\mathcal{O}bjMap)(c) <_{CF.lim} \mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} : c \downarrow_{CF} \mathfrak{K} \mapsto_{C\alpha} \mathfrak{T}(\mathcal{H}omCod)$

)

]_o

definition *the-pw-cat-lKe-colimit* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$

where *the-pw-cat-lKe-colimit* α \mathfrak{K} \mathfrak{T} \mathfrak{F} *c* =

[

 $\mathfrak{F}(\mathcal{O}bjMap)(c)$,

 op-ntcf

 (

 the-pw-cat-rKe-limit α (*op-cf* \mathfrak{K}) (*op-cf* \mathfrak{T}) (*op-cf* \mathfrak{F}) *c*(*UArr*) $\circ_{NTCF-CF}$

 op-cf-obj-comma \mathfrak{K} *c*

)

]_o

Components.

lemma *the-pw-cat-rKe-limit-components:*

shows *the-pw-cat-rKe-limit* α \mathfrak{K} \mathfrak{T} \mathfrak{G} *c*(*UObj*) = $\mathfrak{G}(\mathcal{O}bjMap)(c)$

and *the-pw-cat-rKe-limit* α \mathfrak{K} \mathfrak{T} \mathfrak{G} *c*(*UArr*) =

(

 SOME UA.

 UA : $\mathfrak{G}(\mathcal{O}bjMap)(c) <_{CF.lim} \mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} : c \downarrow_{CF} \mathfrak{K} \mapsto_{C\alpha} \mathfrak{T}(\mathcal{H}omCod)$

)

unfolding *the-pw-cat-rKe-limit-def ua-field-simps*
by (*simp-all add: nat-omega-simps*)

lemma *the-pw-cat-lKe-colimit-components*:

shows *the-pw-cat-lKe-colimit* $\alpha \mathfrak{K} \mathfrak{T} \mathfrak{F} c(\mathcal{U}Obj) = \mathfrak{F}(\mathcal{O}bjMap)(c)$

and *the-pw-cat-lKe-colimit* $\alpha \mathfrak{K} \mathfrak{T} \mathfrak{F} c(\mathcal{U}Arr) = op-ntcf$

(
the-pw-cat-rKe-limit $\alpha (op-cf \mathfrak{K}) (op-cf \mathfrak{T}) (op-cf \mathfrak{F}) c(\mathcal{U}Arr) \circ_{NTCF-CF}$
op-cf-obj-comma $\mathfrak{K} c$
)

unfolding *the-pw-cat-lKe-colimit-def ua-field-simps*

by (*simp-all add: nat-omega-simps*)

context *is-functor*

begin

lemmas *the-pw-cat-rKe-limit-components'* =

the-pw-cat-rKe-limit-components[**where** $\mathfrak{T}=\mathfrak{F}$, *unfolded cat-cs-simps*]

end

15.10.2 The limit for the pointwise right Kan extension is a limit, the colimit for the pointwise left Kan extension is a colimit

lemma (in *is-cat-pw-rKe*) *cat-pw-rKe-the-pw-cat-rKe-limit-is-cat-limit*:

assumes $\mathfrak{K} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{T} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{A}$ **and** $c \in_o \mathfrak{C}(\mathcal{O}bj)$

shows *the-pw-cat-rKe-limit* $\alpha \mathfrak{K} \mathfrak{T} \mathfrak{C} c(\mathcal{U}Arr) :$

the-pw-cat-rKe-limit $\alpha \mathfrak{K} \mathfrak{T} \mathfrak{C} c(\mathcal{U}Obj) <_{CF.lim} \mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} :$

$c \downarrow_{CF} \mathfrak{K} \mapsto_{C\alpha} \mathfrak{A}$

proof-

from *cat-pw-rKe-ex-cat-limit*[*OF assms*] **obtain** *UA*

where *UA*: $UA : \mathfrak{C}(\mathcal{O}bjMap)(c) <_{CF.lim} \mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} \mathfrak{K} : c \downarrow_{CF} \mathfrak{K} \mapsto_{C\alpha} \mathfrak{A}$

by *auto*

show *?thesis*

unfolding *the-pw-cat-rKe-limit-components*

by (*rule someI2, unfold cat-cs-simps, rule UA*)

qed

lemma (in *is-cat-pw-lKe*) *cat-pw-lKe-the-pw-cat-lKe-colimit-is-cat-colimit*:

assumes $\mathfrak{K} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{T} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{A}$ **and** $c \in_o \mathfrak{C}(\mathcal{O}bj)$

shows *the-pw-cat-lKe-colimit* $\alpha \mathfrak{K} \mathfrak{T} \mathfrak{F} c(\mathcal{U}Arr) :$

$\mathfrak{T} \circ_{CF} \mathfrak{K} \circ \sqcap_{CF} c >_{CF.colim} \mathfrak{T} \circ_{CF} \mathfrak{F} c(\mathcal{U}Obj) :$

$\mathfrak{K} \circ \sqcap_{CF} c \mapsto_{C\alpha} \mathfrak{A}$

proof-

interpret \mathfrak{K} : *is-functor* $\alpha \mathfrak{B} \mathfrak{C} \mathfrak{K}$ **by** (*rule assms(1)*)

interpret \mathfrak{T} : *is-functor* $\alpha \mathfrak{B} \mathfrak{A} \mathfrak{T}$ **by** (*rule assms(2)*)

note *cat-pw-rKe-the-pw-cat-rKe-limit-is-cat-limit* =

is-cat-pw-rKe.cat-pw-rKe-the-pw-cat-rKe-limit-is-cat-limit

[
OF is-cat-pw-rKe-op AG.is-functor-op ntcflKe.NTDom.is-functor-op,
unfolded cat-op-simps,
OF assms(3)
]

from *assms(3)* **have**

op-cf $\mathfrak{T} \circ_{CF} c \circ \sqcap_{CF} (op-cf \mathfrak{K}) \circ_{CF} op-cf-obj-comma \mathfrak{K} c =$

op-cf $\mathfrak{T} \circ_{CF} op-cf (\mathfrak{K} \circ \sqcap_{CF} c)$

by

(

cs-concl cs-shallow
cs-simp:
cat-cs-simps cat-comma-cs-simps cat-op-simps
AG.op-cf-cf-obj-comma-proj[OF assms(3)]
cs-intro: *cat-cs-intros cat-comma-cs-intros cat-op-intros*
)

note *ntcf-cf-comp-is-cat-limit-if-is-iso-functor* =
ntcf-cf-comp-is-cat-limit-if-is-iso-functor
[
OF
cat-pw-rKe-the-pw-cat-rKe-limit-is-cat-limit
AG.op-cf-obj-comma-is-iso-functor[OF assms(3)],
unfolded this, folded op-cf-cf-comp
]

from *assms(3)* **have** [*cat-op-simps*]: $op\text{-}cat (op\text{-}cat (\mathfrak{K}_{CF} \downarrow c)) = \mathfrak{K}_{CF} \downarrow c$
by
(
cs-concl cs-shallow
cs-simp: *cat-op-simps* **cs-intro:** *cat-cs-intros cat-comma-cs-intros*
)

from *assms(3)* **have** [*cat-op-simps*]: $op\text{-}cf (op\text{-}cf (\mathfrak{K}_{CF} \sqcap_O c)) = \mathfrak{K}_{CF} \sqcap_O c$
by
(
cs-concl cs-shallow
cs-simp: *cat-op-simps* **cs-intro:** *cat-cs-intros cat-comma-cs-intros*
)

have [*cat-op-simps*]:
the-pw-cat-rKe-limit $\alpha (op\text{-}cf \mathfrak{K}) (op\text{-}cf \mathfrak{T}) (op\text{-}cf \mathfrak{F}) c(\mathcal{U}Obj) =$
the-pw-cat-lKe-colimit $\alpha \mathfrak{K} \mathfrak{T} \mathfrak{F} c(\mathcal{U}Obj)$
unfolding
the-pw-cat-lKe-colimit-components
the-pw-cat-rKe-limit-components
cat-op-simps
by simp

show *?thesis*
by
(
rule is-cat-limit.is-cat-colimit-op
[
OF ntcf-cf-comp-is-cat-limit-if-is-iso-functor,
folded the-pw-cat-lKe-colimit-components,
unfolded cat-op-simps
]
)

qed

lemma (**in** *is-cat-pw-rKe*) *cat-pw-rKe-the-ntcf-rKe-is-cat-rKe*:
assumes $\mathfrak{K} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{T} : \mathfrak{B} \mapsto_{C\alpha} \mathfrak{A}$
shows *the-ntcf-rKe* $\alpha \mathfrak{T} \mathfrak{K}$ (*the-pw-cat-rKe-limit* $\alpha \mathfrak{K} \mathfrak{T} \mathfrak{G}$) :
the-cf-rKe $\alpha \mathfrak{T} \mathfrak{K}$ (*the-pw-cat-rKe-limit* $\alpha \mathfrak{K} \mathfrak{T} \mathfrak{G}$) $\circ_{CF} \mathfrak{K} \mapsto_{CF.\tau Ke\alpha} \mathfrak{T} :$
 $\mathfrak{B} \mapsto_C \mathfrak{C} \mapsto_C \mathfrak{A}$

proof-
interpret $\mathfrak{T} : is\text{-}functor \alpha \mathfrak{B} \mathfrak{A} \mathfrak{T}$ **by** (*rule assms(2)*)
show *the-ntcf-rKe* $\alpha \mathfrak{T} \mathfrak{K}$ (*the-pw-cat-rKe-limit* $\alpha \mathfrak{K} \mathfrak{T} \mathfrak{G}$) :
the-cf-rKe $\alpha \mathfrak{T} \mathfrak{K}$ (*the-pw-cat-rKe-limit* $\alpha \mathfrak{K} \mathfrak{T} \mathfrak{G}$) $\circ_{CF} \mathfrak{K} \mapsto_{CF.\tau Ke\alpha} \mathfrak{T} :$
 $\mathfrak{B} \mapsto_C \mathfrak{C} \mapsto_C \mathfrak{A}$
by
(

```

rule
  the-ntcf-rKe-is-cat-rKe
  [
    OF
    assms(1)
    ntcf-rKe.NTCod.is-functor-axioms
    cat-pw-rKe-the-pw-cat-rKe-limit-is-cat-limit[OF assms]
  ]
)
qed

lemma (in is-cat-pw-lKe) cat-pw-lKe-the-ntcf-lKe-is-cat-lKe:
  assumes  $\mathfrak{K} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{C}$  and  $\mathfrak{T} : \mathfrak{B} \mapsto \mapsto_{C\alpha} \mathfrak{A}$ 
  shows the-ntcf-lKe  $\alpha$   $\mathfrak{T}$   $\mathfrak{K}$  (the-pw-cat-lKe-colimit  $\alpha$   $\mathfrak{K}$   $\mathfrak{T}$   $\mathfrak{F}$ ) :
     $\mathfrak{T} \mapsto_{CF.lKe\alpha}$  the-cf-lKe  $\alpha$   $\mathfrak{T}$   $\mathfrak{K}$  (the-pw-cat-lKe-colimit  $\alpha$   $\mathfrak{K}$   $\mathfrak{T}$   $\mathfrak{F}$ )  $\circ_{CF}$   $\mathfrak{K} :$ 
     $\mathfrak{B} \mapsto_C \mathfrak{C} \mapsto_C \mathfrak{A}$ 
proof-
  interpret  $\mathfrak{T} : is-functor \alpha \mathfrak{B} \mathfrak{A} \mathfrak{T}$  by (rule assms(2))
  show ?thesis
  by
    (
      rule the-ntcf-lKe-is-cat-lKe
      [
        OF
        assms(1,2)
        cat-pw-lKe-the-pw-cat-lKe-colimit-is-cat-colimit[OF assms],
        simplified
      ]
    )
)
qed

```

16 Pointwise Kan extensions: application example

16.1 Background

The application example presented in this section is based on Exercise 6.1.ii in [14]. The primary purpose of this section is the instantiation of the locales associated with the pointwise Kan extensions.

lemma *cat-ordinal-2-is-arrE*:

assumes $f : a \mapsto_{\text{cat-ordinal } (2_{\mathbb{N}})} b$
obtains $f = [0, 0]_{\circ}$ **and** $a = 0$ **and** $b = 0$
 $| f = [0, 1_{\mathbb{N}}]_{\circ}$ **and** $a = 0$ **and** $b = 1_{\mathbb{N}}$
 $| f = [1_{\mathbb{N}}, 1_{\mathbb{N}}]_{\circ}$ **and** $a = 1_{\mathbb{N}}$ **and** $b = 1_{\mathbb{N}}$
using *cat-ordinal-is-arrD*[*OF assms*] **unfolding** *two* **by** *auto*

lemma *cat-ordinal-3-is-arrE*:

assumes $f : a \mapsto_{\text{cat-ordinal } (3_{\mathbb{N}})} b$
obtains $f = [0, 0]_{\circ}$ **and** $a = 0$ **and** $b = 0$
 $| f = [0, 1_{\mathbb{N}}]_{\circ}$ **and** $a = 0$ **and** $b = 1_{\mathbb{N}}$
 $| f = [0, 2_{\mathbb{N}}]_{\circ}$ **and** $a = 0$ **and** $b = 2_{\mathbb{N}}$
 $| f = [1_{\mathbb{N}}, 1_{\mathbb{N}}]_{\circ}$ **and** $a = 1_{\mathbb{N}}$ **and** $b = 1_{\mathbb{N}}$
 $| f = [1_{\mathbb{N}}, 2_{\mathbb{N}}]_{\circ}$ **and** $a = 1_{\mathbb{N}}$ **and** $b = 2_{\mathbb{N}}$
 $| f = [2_{\mathbb{N}}, 2_{\mathbb{N}}]_{\circ}$ **and** $a = 2_{\mathbb{N}}$ **and** $b = 2_{\mathbb{N}}$
using *cat-ordinal-is-arrD*[*OF assms*] **unfolding** *three* **by** *auto*

lemma *0123*: $0 \in_{\circ} 2_{\mathbb{N}}$ $1_{\mathbb{N}} \in_{\circ} 2_{\mathbb{N}}$ $0 \in_{\circ} 3_{\mathbb{N}}$ $1_{\mathbb{N}} \in_{\circ} 3_{\mathbb{N}}$ $2_{\mathbb{N}} \in_{\circ} 3_{\mathbb{N}}$ **by** *auto*

16.2 $\mathfrak{K}23$

16.2.1 Definition and elementary properties

definition $\mathfrak{K}23 :: V$

where $\mathfrak{K}23 =$
 $[$
 $(\lambda a \in_{\circ} \text{cat-ordinal } (2_{\mathbb{N}})(\text{Obj}). \text{ if } a = 0 \text{ then } 0 \text{ else } 2_{\mathbb{N}}),$
 $($
 $\lambda f \in_{\circ} \text{cat-ordinal } (2_{\mathbb{N}})(\text{Arr}).$
 $\text{ if } f = [0, 0]_{\circ} \Rightarrow [0, 0]_{\circ}$
 $\text{ | } f = [0, 1_{\mathbb{N}}]_{\circ} \Rightarrow [0, 2_{\mathbb{N}}]_{\circ}$
 $\text{ | } f = [1_{\mathbb{N}}, 1_{\mathbb{N}}]_{\circ} \Rightarrow [2_{\mathbb{N}}, 2_{\mathbb{N}}]_{\circ}$
 $\text{ | otherwise } \Rightarrow 0$
 $)$,
 $\text{cat-ordinal } (2_{\mathbb{N}}),$
 $\text{cat-ordinal } (3_{\mathbb{N}})$
 $]$.

Components.

lemma *$\mathfrak{K}23$ -components*:

shows $\mathfrak{K}23(\text{ObjMap}) = (\lambda a \in_{\circ} \text{cat-ordinal } (2_{\mathbb{N}})(\text{Obj}). \text{ if } a = 0 \text{ then } 0 \text{ else } 2_{\mathbb{N}})$
and $\mathfrak{K}23(\text{ArrMap}) =$
 $($
 $\lambda f \in_{\circ} \text{cat-ordinal } (2_{\mathbb{N}})(\text{Arr}).$
 $\text{ if } f = [0, 0]_{\circ} \Rightarrow [0, 0]_{\circ}$
 $\text{ | } f = [0, 1_{\mathbb{N}}]_{\circ} \Rightarrow [0, 2_{\mathbb{N}}]_{\circ}$
 $\text{ | } f = [1_{\mathbb{N}}, 1_{\mathbb{N}}]_{\circ} \Rightarrow [2_{\mathbb{N}}, 2_{\mathbb{N}}]_{\circ}$
 $\text{ | otherwise } \Rightarrow 0$
 $)$

and $[cat\text{-}Kan\text{-}cs\text{-}simps]: \mathfrak{K}23(\text{HomDom}) = \text{cat-ordinal } (2_{\mathbb{N}})$
and $[cat\text{-}Kan\text{-}cs\text{-}simps]: \mathfrak{K}23(\text{HomCod}) = \text{cat-ordinal } (3_{\mathbb{N}})$
unfolding $\mathfrak{K}23\text{-def } dghm\text{-field-simps}$ **by** $(\text{simp-all add: nat-omega-simps})$

16.2.2 Object map

mk-VLambda $\mathfrak{K}23\text{-components}(1)$
 $|vsv \ \mathfrak{K}23\text{-ObjMap-vsv}[cat\text{-}Kan\text{-}cs\text{-}intros]|$
 $|vdomain \ \mathfrak{K}23\text{-ObjMap-vdomain}[cat\text{-}Kan\text{-}cs\text{-}simps]|$
 $|app \ \mathfrak{K}23\text{-ObjMap-app}|$

lemma $\mathfrak{K}23\text{-ObjMap-app-0}[cat\text{-}Kan\text{-}cs\text{-}simps]:$
assumes $x = 0$
shows $\mathfrak{K}23(\text{ObjMap})(\downarrow x) = 0$
by
 $($
 cs-concl
cs-simp: $\mathfrak{K}23\text{-ObjMap-app cat-ordinal-cs-simps V-cs-simps assms}$
cs-intro: nat-omega-intros
 $)$

lemma $\mathfrak{K}23\text{-ObjMap-app-1}[cat\text{-}Kan\text{-}cs\text{-}simps]:$
assumes $x = 1_{\mathbb{N}}$
shows $\mathfrak{K}23(\text{ObjMap})(\downarrow x) = 2_{\mathbb{N}}$
by
 $($
 cs-concl
cs-simp:
 $\text{cat-ordinal-cs-simps V-cs-simps omega-of-set } \mathfrak{K}23\text{-ObjMap-app assms}$
cs-intro: $\text{nat-omega-intros V-cs-intros}$
 $)$

16.2.3 Arrow map

mk-VLambda $\mathfrak{K}23\text{-components}(2)$
 $|vsv \ \mathfrak{K}23\text{-ArrMap-vsv}[cat\text{-}Kan\text{-}cs\text{-}intros]|$
 $|vdomain \ \mathfrak{K}23\text{-ArrMap-vdomain}[cat\text{-}Kan\text{-}cs\text{-}simps]|$
 $|app \ \mathfrak{K}23\text{-ArrMap-app}|$

lemma $\mathfrak{K}23\text{-ArrMap-app-00}[cat\text{-}Kan\text{-}cs\text{-}simps]:$
assumes $f = [0, 0]_{\circ}$
shows $\mathfrak{K}23(\text{ArrMap})(\downarrow f) = [0, 0]_{\circ}$
unfolding $assms$
by
 $($
 cs-concl
cs-simp: $\mathfrak{K}23\text{-ArrMap-app cat-ordinal-cs-simps V-cs-simps}$
cs-intro: $\text{cat-ordinal-cs-intros nat-omega-intros}$
 $)$

lemma $\mathfrak{K}23\text{-ArrMap-app-01}[cat\text{-}Kan\text{-}cs\text{-}simps]:$
assumes $f = [0, 1_{\mathbb{N}}]_{\circ}$
shows $\mathfrak{K}23(\text{ArrMap})(\downarrow f) = [0, 2_{\mathbb{N}}]_{\circ}$
proof-
have $[0, 1_{\mathbb{N}}]_{\circ} \in_{\circ} \text{ordinal-arrs } (2_{\mathbb{N}})$
by
 $($
 cs-concl
 $)$

cs-simp: *omega-of-set*
cs-intro: *cat-ordinal-cs-intros V-cs-intros nat-omega-intros*
)
then show *?thesis*
unfolding *assms* **by** (*simp add: $\mathfrak{K}23$ -components cat-ordinal-components*)
qed

lemma *$\mathfrak{K}23$ -ArrMap-app-11[cat-Kan-cs-simps]*:

assumes $f = [1_{\mathbb{N}}, 1_{\mathbb{N}}]_{\circ}$
shows $\mathfrak{K}23(\text{ArrMap})(f) = [2_{\mathbb{N}}, 2_{\mathbb{N}}]_{\circ}$

proof-

have $[1_{\mathbb{N}}, 1_{\mathbb{N}}]_{\circ} \in_{\circ} \text{ordinal-arrs } (2_{\mathbb{N}})$
by
 (
cs-concl cs-shallow
cs-simp: *omega-of-set*
cs-intro: *cat-ordinal-cs-intros V-cs-intros nat-omega-intros*
)

then show *?thesis*
unfolding *assms* **by** (*simp add: $\mathfrak{K}23$ -components cat-ordinal-components*)
qed

16.2.4 $\mathfrak{K}23$ is a tiny functor

lemma (**in** \mathcal{Z}) *$\mathfrak{K}23$ -is-functor:* $\mathfrak{K}23 : \text{cat-ordinal } (2_{\mathbb{N}}) \mapsto_{C\alpha} \text{cat-ordinal } (3_{\mathbb{N}})$

proof-

from *ord-of-nat- ω* **interpret** *cat-ordinal-2: finite-category $\alpha \langle \text{cat-ordinal } (2_{\mathbb{N}}) \rangle$*
by (*cs-concl cs-shallow cs-intro: cat-ordinal-cs-intros*)
from *ord-of-nat- ω* **interpret** *cat-ordinal-3: finite-category $\alpha \langle \text{cat-ordinal } (3_{\mathbb{N}}) \rangle$*
by (*cs-concl cs-shallow cs-intro: cat-ordinal-cs-intros*)

show *?thesis*

proof(*intro is-tiny-functorI' is-functorI'*)

show *vfsequence $\mathfrak{K}23$ unfolding $\mathfrak{K}23$ -def by auto*
show *vcard $\mathfrak{K}23 = 4_{\mathbb{N}}$ unfolding $\mathfrak{K}23$ -def by (simp add: nat-omega-simps)*

show $\mathcal{R}_{\circ}(\mathfrak{K}23(\text{ObjMap})) \subseteq_{\circ} \text{cat-ordinal } (3_{\mathbb{N}})(\text{Obj})$

proof

(
rule vsv.vsv-vrange-vsubset,
unfold cat-Kan-cs-simps cat-ordinal-cs-simps,
intro cat-Kan-cs-intros
)
fix x **assume** $x \in_{\circ} 2_{\mathbb{N}}$
then consider $\langle x = 0 \rangle \mid \langle x = 1_{\mathbb{N}} \rangle$ **unfolding** *two by auto*
then show $\mathfrak{K}23(\text{ObjMap})(x) \in_{\circ} 3_{\mathbb{N}}$
by (*cases, use nothing in $\langle \text{simp-all only} \rangle$*)
 (
cs-concl
cs-simp: *cat-Kan-cs-simps omega-of-set cs-intro: nat-omega-intros*
)+

qed

show $\mathfrak{K}23(\text{ArrMap})(f) : \mathfrak{K}23(\text{ObjMap})(a) \mapsto_{\text{cat-ordinal } (3_{\mathbb{N}})} \mathfrak{K}23(\text{ObjMap})(b)$

if $f : a \mapsto_{\text{cat-ordinal } (2_{\mathbb{N}})} b$ **for** $a \ b \ f$

using *that*

by (elim cat-ordinal-2-is-arrE; simp only)
 (
 cs-concl
 cs-simp: omega-of-set cat-Kan-cs-simps
 cs-intro: nat-omega-intros V-cs-intros cat-ordinal-cs-intros
)

show
 $\mathfrak{K}23(\text{ArrMap})(\langle g \circ_A \text{cat-ordinal } (2_{\mathbb{N}}) f \rangle) =$
 $\mathfrak{K}23(\text{ArrMap})(\langle g \rangle \circ_A \text{cat-ordinal } (3_{\mathbb{N}}) \mathfrak{K}23(\text{ArrMap})(\langle f \rangle))$
 if $g : b \mapsto_{\text{cat-ordinal } (2_{\mathbb{N}})} c$ and $f : a \mapsto_{\text{cat-ordinal } (2_{\mathbb{N}})} b$
 for $b \ c \ g \ a \ f$

proof-
 have $0 \in_o 3_{\mathbb{N}} \ 1_{\mathbb{N}} \in_o 3_{\mathbb{N}} \ 2_{\mathbb{N}} \in_o 3_{\mathbb{N}}$ by auto
 then show ?thesis
 using that
 by (elim cat-ordinal-2-is-arrE; simp only)
 (
 cs-concl
 cs-simp: cat-ordinal-cs-simps cat-Kan-cs-simps
 cs-intro: V-cs-intros cat-ordinal-cs-intros
)+
 qed

show
 $\mathfrak{K}23(\text{ArrMap})(\langle \text{cat-ordinal } (2_{\mathbb{N}})(\langle \text{CIId} \rangle(\langle c \rangle)) \rangle) =$
 $\text{cat-ordinal } (3_{\mathbb{N}})(\langle \text{CIId} \rangle(\langle \mathfrak{K}23(\text{ObjMap})(\langle c \rangle) \rangle))$
 if $c \in_o \text{cat-ordinal } (2_{\mathbb{N}})(\langle \text{Obj} \rangle)$ for c
 proof-
 from that consider $\langle c = 0 \rangle \mid \langle c = 1_{\mathbb{N}} \rangle$
 unfolding cat-ordinal-components(1) two by auto
 then show ?thesis
 by (cases, use nothing in $\langle \text{simp-all only} \rangle$)
 (
 cs-concl
 cs-simp: omega-of-set cat-Kan-cs-simps cat-ordinal-cs-simps
 cs-intro: nat-omega-intros cat-ordinal-cs-intros
)
 qed

qed (auto intro!: cat-cs-intros simp: $\mathfrak{K}23$ -components)

qed

lemma (in \mathcal{Z}) $\mathfrak{K}23$ -is-functor' [cat-Kan-cs-intros]:
 assumes $\mathfrak{A}' = \text{cat-ordinal } (2_{\mathbb{N}})$
 and $\mathfrak{B}' = \text{cat-ordinal } (3_{\mathbb{N}})$
 shows $\mathfrak{K}23 : \mathfrak{A}' \mapsto_{C\alpha} \mathfrak{B}'$
 unfolding assms by (rule $\mathfrak{K}23$ -is-functor)

lemmas [cat-Kan-cs-intros] = $\mathcal{Z}.\mathfrak{K}23$ -is-functor'

lemma (in \mathcal{Z}) $\mathfrak{K}23$ -is-tiny-functor:
 $\mathfrak{K}23 : \text{cat-ordinal } (2_{\mathbb{N}}) \mapsto_{C.\text{tiny}\alpha} \text{cat-ordinal } (3_{\mathbb{N}})$

proof-
 from ord-of-nat- ω interpret cat-ordinal-2: finite-category $\alpha \langle \text{cat-ordinal } (2_{\mathbb{N}}) \rangle$
 by (cs-concl **cs-shallow** **cs-intro**: cat-ordinal-cs-intros)
 from ord-of-nat- ω interpret cat-ordinal-3: finite-category $\alpha \langle \text{cat-ordinal } (3_{\mathbb{N}}) \rangle$

by (*cs-concl cs-shallow cs-intro: cat-ordinal-cs-intros*)
 show *?thesis*
 by (*intro is-tiny-functorI' $\mathfrak{K}23$ -is-functor*)
 (*auto intro!: cat-small-cs-intros*)
 qed

lemma (in \mathcal{Z}) *$\mathfrak{K}23$ -is-tiny-functor'* [*cat-Kan-cs-intros*]:
 assumes $\mathfrak{A}' = \text{cat-ordinal } (2_{\mathbb{N}})$
 and $\mathfrak{B}' = \text{cat-ordinal } (3_{\mathbb{N}})$
 shows $\mathfrak{K}23 : \mathfrak{A}' \mapsto \mapsto_{C.tiny\alpha} \mathfrak{B}'$
 unfolding *assms* by (*rule $\mathfrak{K}23$ -is-tiny-functor*)

lemmas [*cat-Kan-cs-intros*] = $\mathcal{Z}.\mathfrak{K}23$ -is-tiny-functor'

16.3 $LK23$: the functor associated with the left Kan extension along $\mathfrak{K}23$

16.3.1 Definition and elementary properties

definition $LK23 :: V \Rightarrow V$

where $LK23 \mathfrak{F} =$

[
 (
 $\lambda a \in_{\circ} \text{cat-ordinal } (3_{\mathbb{N}}) (\text{Obj})$.
 if $a = 0 \Rightarrow \mathfrak{F} (\text{ObjMap}) (0)$
 | $a = 1_{\mathbb{N}} \Rightarrow \mathfrak{F} (\text{ObjMap}) (0)$
 | $a = 2_{\mathbb{N}} \Rightarrow \mathfrak{F} (\text{ObjMap}) (1_{\mathbb{N}})$
 | otherwise $\Rightarrow \mathfrak{F} (\text{HomCod}) (\text{Obj})$
)
 (
 $\lambda f \in_{\circ} \text{cat-ordinal } (3_{\mathbb{N}}) (\text{Arr})$.
 if $f = [0, 0]_{\circ} \Rightarrow \mathfrak{F} (\text{ArrMap}) (0, 0)$.
 | $f = [0, 1_{\mathbb{N}}]_{\circ} \Rightarrow \mathfrak{F} (\text{ArrMap}) (0, 0)$.
 | $f = [0, 2_{\mathbb{N}}]_{\circ} \Rightarrow \mathfrak{F} (\text{ArrMap}) (0, 1_{\mathbb{N}})$.
 | $f = [1_{\mathbb{N}}, 1_{\mathbb{N}}]_{\circ} \Rightarrow \mathfrak{F} (\text{ArrMap}) (0, 0)$.
 | $f = [1_{\mathbb{N}}, 2_{\mathbb{N}}]_{\circ} \Rightarrow \mathfrak{F} (\text{ArrMap}) (0, 1_{\mathbb{N}})$.
 | $f = [2_{\mathbb{N}}, 2_{\mathbb{N}}]_{\circ} \Rightarrow \mathfrak{F} (\text{ArrMap}) (1_{\mathbb{N}}, 1_{\mathbb{N}})$.
 | otherwise $\Rightarrow \mathfrak{F} (\text{HomCod}) (\text{Arr})$
)
 $\text{cat-ordinal } (3_{\mathbb{N}})$,
 $\mathfrak{F} (\text{HomCod})$
]_o

Components.

lemma $LK23$ -components:

shows $LK23 \mathfrak{F} (\text{ObjMap}) =$

(
 $\lambda a \in_{\circ} \text{cat-ordinal } (3_{\mathbb{N}}) (\text{Obj})$.
 if $a = 0 \Rightarrow \mathfrak{F} (\text{ObjMap}) (0)$
 | $a = 1_{\mathbb{N}} \Rightarrow \mathfrak{F} (\text{ObjMap}) (0)$
 | $a = 2_{\mathbb{N}} \Rightarrow \mathfrak{F} (\text{ObjMap}) (1_{\mathbb{N}})$
 | otherwise $\Rightarrow \mathfrak{F} (\text{HomCod}) (\text{Obj})$
)

and $LK23 \mathfrak{F} (\text{ArrMap}) =$

(
 $\lambda f \in_{\circ} \text{cat-ordinal } (3_{\mathbb{N}}) (\text{Arr})$.
 if $f = [0, 0]_{\circ} \Rightarrow \mathfrak{F} (\text{ArrMap}) (0, 0)$.
 | $f = [0, 1_{\mathbb{N}}]_{\circ} \Rightarrow \mathfrak{F} (\text{ArrMap}) (0, 0)$.
 | $f = [0, 2_{\mathbb{N}}]_{\circ} \Rightarrow \mathfrak{F} (\text{ArrMap}) (0, 1_{\mathbb{N}})$.
)

```

|  $f = [1_{\mathbb{N}}, 1_{\mathbb{N}}]_{\circ} \Rightarrow \mathfrak{F}(\text{ArrMap})(0, 0)$ .
|  $f = [1_{\mathbb{N}}, 2_{\mathbb{N}}]_{\circ} \Rightarrow \mathfrak{F}(\text{ArrMap})(0, 1_{\mathbb{N}})$ .
|  $f = [2_{\mathbb{N}}, 2_{\mathbb{N}}]_{\circ} \Rightarrow \mathfrak{F}(\text{ArrMap})(1_{\mathbb{N}}, 1_{\mathbb{N}})$ .
| otherwise  $\Rightarrow \mathfrak{F}(\text{HomCod})(\text{Arr})$ 
)
and LK23  $\mathfrak{F}(\text{HomDom}) = \text{cat-ordinal } (3_{\mathbb{N}})$ 
and LK23  $\mathfrak{F}(\text{HomCod}) = \mathfrak{F}(\text{HomCod})$ 
unfolding LK23-def dghm-field-simps by (simp-all add: nat-omega-simps)

```

```

context is-functor
begin

```

```

lemmas LK23-components' = LK23-components[where  $\mathfrak{F}=\mathfrak{F}$ , unfolded cat-cs-simps]

```

```

lemmas [cat-Kan-cs-simps] = LK23-components'(3,4)

```

```

end

```

```

lemmas [cat-Kan-cs-simps] = is-functor.LK23-components'(3,4)

```

16.3.2 Object map

```

mk-VLambda LK23-components(1)
|vsv LK23-ObjMap-vsuv[cat-Kan-cs-intros]
|vdomain LK23-ObjMap-vdomain[cat-Kan-cs-simps]
|app LK23-ObjMap-app

```

```

lemma LK23-ObjMap-app-0[cat-Kan-cs-simps]:
  assumes  $a = 0$ 
  shows LK23  $\mathfrak{F}(\text{ObjMap})(a) = \mathfrak{F}(\text{ObjMap})(0)$ 
  unfolding LK23-components assms cat-ordinal-components by simp

```

```

lemma LK23-ObjMap-app-1[cat-Kan-cs-simps]:
  assumes  $a = 1_{\mathbb{N}}$ 
  shows LK23  $\mathfrak{F}(\text{ObjMap})(a) = \mathfrak{F}(\text{ObjMap})(0)$ 
  unfolding LK23-components assms cat-ordinal-components by simp

```

```

lemma LK23-ObjMap-app-2[cat-Kan-cs-simps]:
  assumes  $a = 2_{\mathbb{N}}$ 
  shows LK23  $\mathfrak{F}(\text{ObjMap})(a) = \mathfrak{F}(\text{ObjMap})(1_{\mathbb{N}})$ 
  unfolding LK23-components assms cat-ordinal-components by simp

```

16.3.3 Arrow map

```

mk-VLambda LK23-components(2)
|vsv LK23-ArrMap-vsuv[cat-Kan-cs-intros]
|vdomain LK23-ArrMap-vdomain[cat-Kan-cs-simps]
|app LK23-ArrMap-app

```

```

lemma LK23-ArrMap-app-00[cat-Kan-cs-simps]:
  assumes  $f = [0, 0]_{\circ}$ 
  shows LK23  $\mathfrak{F}(\text{ArrMap})(f) = \mathfrak{F}(\text{ArrMap})(0, 0)$ .

```

```

proof-

```

```

  from 0123 have  $f: f \in_{\circ} \text{cat-ordinal } (3_{\mathbb{N}})(\text{Arr})$ 

```

```

  by

```

```

  (

```

```

    cs-concl cs-shallow

```

```

    cs-intro: V-cs-intros cat-ordinal-cs-intros cat-cs-intros assms

```

)
then show *?thesis unfolding LK23-components assms by auto*
qed

lemma *LK23-ArrMap-app-01[cat-Kan-cs-simps]*:
assumes $f = [0, 1_{\mathbb{N}}]_{\circ}$
shows $LK23 \ \mathfrak{F}(\text{ArrMap})(f) = \mathfrak{F}(\text{ArrMap})(0, 0)$.

proof-
from *0123* **have** $f: f \in_{\circ} \text{cat-ordinal } (3_{\mathbb{N}})(\text{Arr})$
by
 (
 cs-concl cs-shallow
 cs-intro: *V-cs-intros cat-ordinal-cs-intros cat-cs-intros assms*
)

then show *?thesis unfolding LK23-components assms by auto*
qed

lemma *LK23-ArrMap-app-02[cat-Kan-cs-simps]*:
assumes $f = [0, 2_{\mathbb{N}}]_{\circ}$
shows $LK23 \ \mathfrak{F}(\text{ArrMap})(f) = \mathfrak{F}(\text{ArrMap})(0, 1_{\mathbb{N}})$.

proof-
from *0123* **have** $f: f \in_{\circ} \text{cat-ordinal } (3_{\mathbb{N}})(\text{Arr})$
by
 (
 cs-concl cs-shallow
 cs-intro: *V-cs-intros cat-ordinal-cs-intros cat-cs-intros assms*
)

then show *?thesis unfolding LK23-components assms by auto*
qed

lemma *LK23-ArrMap-app-11[cat-Kan-cs-simps]*:
assumes $f = [1_{\mathbb{N}}, 1_{\mathbb{N}}]_{\circ}$
shows $LK23 \ \mathfrak{F}(\text{ArrMap})(f) = \mathfrak{F}(\text{ArrMap})(0, 0)$.

proof-
from *0123* **have** $f: f \in_{\circ} \text{cat-ordinal } (3_{\mathbb{N}})(\text{Arr})$
by
 (
 cs-concl cs-shallow
 cs-intro: *V-cs-intros cat-ordinal-cs-intros cat-cs-intros assms*
)

then show *?thesis unfolding LK23-components assms by auto*
qed

lemma *LK23-ArrMap-app-12[cat-Kan-cs-simps]*:
assumes $f = [1_{\mathbb{N}}, 2_{\mathbb{N}}]_{\circ}$
shows $LK23 \ \mathfrak{F}(\text{ArrMap})(f) = \mathfrak{F}(\text{ArrMap})(0, 1_{\mathbb{N}})$.

proof-
from *0123* **have** $f: f \in_{\circ} \text{cat-ordinal } (3_{\mathbb{N}})(\text{Arr})$
by
 (
 cs-concl
 cs-simp: *omega-of-set*
 cs-intro: *nat-omega-intros cat-ordinal-cs-intros cat-cs-intros assms*
)

then show *?thesis unfolding LK23-components assms by auto*
qed

lemma *LK23-ArrMap-app-22[cat-Kan-cs-simps]*:

assumes $f = [2_{\mathbb{N}}, 2_{\mathbb{N}}]$.
shows $LK23 \mathfrak{F}(\text{ArrMap})(f) = \mathfrak{F}(\text{ArrMap})(1_{\mathbb{N}}, 1_{\mathbb{N}})$.
proof-
from 0123 **have** $f: f \in_{\circ} \text{cat-ordinal } (3_{\mathbb{N}})(\text{Arr})$
by
(

cs-concl **cs-shallow**
cs-intro: *nat-omega-intros cat-ordinal-cs-intros cat-cs-intros assms*
)

then show *?thesis unfolding LK23-components assms by simp*
qed

16.3.4 LK23 is a functor

lemma *cat-LK23-is-functor:*

assumes $\mathfrak{F} : \text{cat-ordinal } (2_{\mathbb{N}}) \mapsto C_{\alpha} \mathfrak{C}$
shows $LK23 \mathfrak{F} : \text{cat-ordinal } (3_{\mathbb{N}}) \mapsto C_{\alpha} \mathfrak{C}$
proof-

interpret $\mathfrak{F} : \text{is-functor } \alpha \langle \text{cat-ordinal } (2_{\mathbb{N}}) \rangle \mathfrak{C} \mathfrak{F}$ **by** (*rule assms(1)*)

from *ord-of-nat- ω* **interpret** *cat-ordinal-2: finite-category* $\alpha \langle \text{cat-ordinal } (2_{\mathbb{N}}) \rangle$
by (*cs-concl cs-shallow cs-intro: cat-ordinal-cs-intros*)
from *ord-of-nat- ω* **interpret** *cat-ordinal-3: finite-category* $\alpha \langle \text{cat-ordinal } (3_{\mathbb{N}}) \rangle$
by (*cs-concl cs-shallow cs-intro: cat-ordinal-cs-intros*)

interpret $\mathfrak{F} : \text{is-functor } \alpha \langle \text{cat-ordinal } (2_{\mathbb{N}}) \rangle \mathfrak{C} \mathfrak{F}$ **by** (*rule assms*)

show *?thesis*

proof(*intro is-functorI'*)

show *vfsequence (LK23 \mathfrak{F}) unfolding LK23-def by auto*
show *vcard (LK23 \mathfrak{F}) = 4 $_{\mathbb{N}}$ unfolding LK23-def by (simp add: nat-omega-simps)*
show $\mathcal{R}_{\circ} (LK23 \mathfrak{F}(\text{ObjMap})) \subseteq_{\circ} \mathfrak{C}(\text{Obj})$
proof(*rule vsv.vsv-vrange-vsubset, unfold cat-Kan-cs-simps*)
fix x **assume** *prems: $x \in_{\circ} \text{cat-ordinal } (3_{\mathbb{N}})(\text{Obj})$*
then consider $\langle x = 0 \rangle \mid \langle x = 1_{\mathbb{N}} \rangle \mid \langle x = 2_{\mathbb{N}} \rangle$
unfolding *cat-ordinal-cs-simps three by auto*
then show $LK23 \mathfrak{F}(\text{ObjMap})(x) \in_{\circ} \mathfrak{C}(\text{Obj})$

by cases

(

cs-concl
cs-simp: *cat-Kan-cs-simps cat-ordinal-cs-simps omega-of-set*
cs-intro: *cat-cs-intros nat-omega-intros*
)

)+

qed (*cs-concl cs-shallow cs-intro: cat-Kan-cs-intros*)

show $LK23 \mathfrak{F}(\text{ArrMap})(f) : LK23 \mathfrak{F}(\text{ObjMap})(a) \mapsto_{\mathfrak{C}} LK23 \mathfrak{F}(\text{ObjMap})(b)$

if $f : a \mapsto_{\text{cat-ordinal } (3_{\mathbb{N}})} b$ **for** $a \ b \ f$

proof-

from 0123 **that show** *?thesis*

by (*elim cat-ordinal-3-is-arrE; simp only:*)

(

cs-concl
cs-simp: *cat-Kan-cs-simps*
cs-intro: *V-cs-intros cat-cs-intros cat-ordinal-cs-intros*
)

)+

qed

show

$LK23 \mathfrak{F}(\text{ArrMap})(g \circ_{A \text{cat-ordinal } (3_{\mathbb{N}})} f) =$

$LK23 \mathfrak{F}(\text{ArrMap})(g) \circ_{A\mathfrak{C}} LK23 \mathfrak{F}(\text{ArrMap})(f)$
if $g : b \mapsto_{\text{cat-ordinal } (3_N)} c$ **and** $f : a \mapsto_{\text{cat-ordinal } (3_N)} b$
for $b \ c \ g \ a \ f$
proof-
from *0123 that show ?thesis*
by (*elim cat-ordinal-3-is-arrE; simp only; (solves⟨simp⟩)?*)
(

cs-concl
cs-simp:
cat-ordinal-cs-simps
cat-Kan-cs-simps
 $\mathfrak{F}.cf\text{-ArrMap-Comp[symmetric]$
cs-intro: *V-cs-intros cat-cs-intros cat-ordinal-cs-intros*
)+
qed
show $LK23 \mathfrak{F}(\text{ArrMap})(\text{cat-ordinal } (3_N)(\text{CId})(c)) = \mathfrak{C}(\text{CId})(LK23 \mathfrak{F}(\text{ObjMap})(c))$
if $c \in_o \text{cat-ordinal } (3_N)(\text{Obj})$ **for** c
proof-
from *that consider ⟨c = 0⟩ | ⟨c = 1_N⟩ | ⟨c = 2_N⟩*
unfolding *cat-ordinal-components three by auto*
moreover have $0 \in_o 2_N$ $1_N \in_o 2_N$ $0 \in_o 3_N$ $1_N \in_o 3_N$ $2_N \in_o 3_N$ **by auto**
ultimately show *?thesis*
by (*cases, use nothing in ⟨simp-all only⟩*)
(

cs-concl
cs-simp:
cat-ordinal-cs-simps
cat-Kan-cs-simps
is-functor.cf-ObjMap-CId[symmetric]
cs-intro: *V-cs-intros cat-cs-intros cat-ordinal-cs-intros*
)+
qed
qed
(

cs-concl **cs-shallow**
cs-simp: *cat-Kan-cs-simps* **cs-intro:** *cat-cs-intros cat-Kan-cs-intros*
)+
qed

qed

lemma *cat-LK23-is-functor'[cat-Kan-cs-intros]:*

assumes $\mathfrak{F} : \text{cat-ordinal } (2_N) \mapsto_{C\alpha} \mathfrak{C}$

and $\mathfrak{A}' = \text{cat-ordinal } (3_N)$

shows $LK23 \mathfrak{F} : \mathfrak{A}' \mapsto_{C\alpha} \mathfrak{C}$

using *assms(1) unfolding assms(2) by (rule cat-LK23-is-functor)*

16.3.5 The fundamental property of $LK23$

lemma *cf-comp-LK23- $\mathfrak{R}23$ [cat-Kan-cs-simps]:*

assumes $\mathfrak{F} : \text{cat-ordinal } (2_N) \mapsto_{C\alpha} \mathfrak{C}$

shows $LK23 \mathfrak{F} \circ_{CF} \mathfrak{R}23 = \mathfrak{F}$

proof-

interpret \mathfrak{F} : *is-functor* $\alpha \langle \text{cat-ordinal } (2_N) \rangle \mathfrak{C} \mathfrak{F}$ **by** (*rule assms(1)*)

interpret $\mathfrak{R}23$: *is-functor* $\alpha \langle \text{cat-ordinal } (2_N) \rangle \langle \text{cat-ordinal } (3_N) \rangle \langle \mathfrak{R}23 \rangle$

by (*cs-concl cs-shallow cs-intro: cat-cs-intros cat-Kan-cs-intros*)

interpret $LK23$: *is-functor* $\alpha \langle \text{cat-ordinal } (3_N) \rangle \mathfrak{C} \langle LK23 \mathfrak{F} \rangle$

by (*cs-concl cs-intro: cat-Kan-cs-intros cat-cs-intros*)

```

show ?thesis
proof(rule cf-eqI)
  show  $\mathfrak{F} : \text{cat-ordinal } (2_{\mathbb{N}}) \mapsto_{CF} \mathfrak{C}$  by (rule assms)
  have ObjMap-dom-lhs:  $\mathcal{D}_o((LK23 \mathfrak{F} \circ_{CF} \mathfrak{R}23)(ObjMap)) = 2_{\mathbb{N}}$ 
    by
      (
        cs-concl cs-shallow
        cs-simp: cat-cs-simps cat-ordinal-cs-simps cs-intro: cat-cs-intros
      )
  have ObjMap-dom-rhs:  $\mathcal{D}_o(\mathfrak{F}(ObjMap)) = 2_{\mathbb{N}}$ 
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cat-ordinal-cs-simps)
  show  $(LK23 \mathfrak{F} \circ_{CF} \mathfrak{R}23)(ObjMap) = \mathfrak{F}(ObjMap)$ 
  proof(rule vsu-eqI, unfold ObjMap-dom-lhs ObjMap-dom-rhs)
    fix a assume prems:  $a \in_o 2_{\mathbb{N}}$ 
    then consider  $\langle a = 0 \rangle \mid \langle a = 1_{\mathbb{N}} \rangle$  by force
    then show  $(LK23 \mathfrak{F} \circ_{CF} \mathfrak{R}23)(ObjMap)(a) = \mathfrak{F}(ObjMap)(a)$ 
      by (cases, use nothing in  $\langle \text{simp-all only} \rangle$ )
      (
        cs-concl
        cs-simp:
          omega-of-set cat-cs-simps cat-ordinal-cs-simps cat-Kan-cs-simps
        cs-intro: cat-cs-intros nat-omega-intros
      )+
  qed (cs-concl cs-intro: cat-cs-intros V-cs-intros)+
  have ArrMap-dom-lhs:  $\mathcal{D}_o((LK23 \mathfrak{F} \circ_{CF} \mathfrak{R}23)(ArrMap)) = \text{cat-ordinal } (2_{\mathbb{N}})(Arr)$ 
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  have ArrMap-dom-rhs:  $\mathcal{D}_o(\mathfrak{F}(ArrMap)) = \text{cat-ordinal } (2_{\mathbb{N}})(Arr)$ 
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  show  $(LK23 \mathfrak{F} \circ_{CF} \mathfrak{R}23)(ArrMap) = \mathfrak{F}(ArrMap)$ 
  proof(rule vsu-eqI, unfold ArrMap-dom-lhs ArrMap-dom-rhs)
    fix f assume prems:  $f \in_o \text{cat-ordinal } (2_{\mathbb{N}})(Arr)$ 
    then obtain a b where  $f : a \mapsto_{\text{cat-ordinal } (2_{\mathbb{N}})} b$  by auto
    then show  $(LK23 \mathfrak{F} \circ_{CF} \mathfrak{R}23)(ArrMap)(f) = \mathfrak{F}(ArrMap)(f)$ 
      by (elim cat-ordinal-2-is-arrE; simp only:)
      (
        cs-concl
        cs-simp: cat-cs-simps cat-Kan-cs-simps cs-intro: cat-cs-intros
      )+
  qed (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros V-cs-intros)+
  qed (cs-concl cs-intro: cat-Kan-cs-intros cat-cs-intros)

```

qed

16.4 RK23: the functor associated with the right Kan extension along $\mathfrak{R}23$

16.4.1 Definition and elementary properties

definition $RK23 :: V \Rightarrow V$

where $RK23 \mathfrak{F} =$

```

[
  (
     $\lambda a \in_o \text{cat-ordinal } (3_{\mathbb{N}})(Obj).$ 
    if  $a = 0 \Rightarrow \mathfrak{F}(ObjMap)(0)$ 
    |  $a = 1_{\mathbb{N}} \Rightarrow \mathfrak{F}(ObjMap)(1_{\mathbb{N}})$ 
    |  $a = 2_{\mathbb{N}} \Rightarrow \mathfrak{F}(ObjMap)(1_{\mathbb{N}})$ 
    | otherwise  $\Rightarrow \mathfrak{F}(HomCod)(Obj)$ 
  ),
]

```

```

(
  λf ∈o cat-ordinal (3N) (Arr).
  if f = [0, 0]o ⇒ ℱ(ArrMap) (0, 0)•
  | f = [0, 1N]o ⇒ ℱ(ArrMap) (0, 1N)•
  | f = [0, 2N]o ⇒ ℱ(ArrMap) (0, 1N)•
  | f = [1N, 1N]o ⇒ ℱ(ArrMap) (1N, 1N)•
  | f = [1N, 2N]o ⇒ ℱ(ArrMap) (1N, 1N)•
  | f = [2N, 2N]o ⇒ ℱ(ArrMap) (1N, 1N)•
  | otherwise ⇒ ℱ(HomCod) (Arr)
),
cat-ordinal (3N),
ℱ(HomCod)
]o

```

Components.

lemma *RK23-components*:

shows *RK23* ℱ(ObjMap) =

```

(
  λa ∈o cat-ordinal (3N) (Obj).
  if a = 0 ⇒ ℱ(ObjMap) (0)
  | a = 1N ⇒ ℱ(ObjMap) (1N)
  | a = 2N ⇒ ℱ(ObjMap) (1N)
  | otherwise ⇒ ℱ(HomCod) (Obj)
)

```

and *RK23* ℱ(ArrMap) =

```

(
  λf ∈o cat-ordinal (3N) (Arr).
  if f = [0, 0]o ⇒ ℱ(ArrMap) (0, 0)•
  | f = [0, 1N]o ⇒ ℱ(ArrMap) (0, 1N)•
  | f = [0, 2N]o ⇒ ℱ(ArrMap) (0, 1N)•
  | f = [1N, 1N]o ⇒ ℱ(ArrMap) (1N, 1N)•
  | f = [1N, 2N]o ⇒ ℱ(ArrMap) (1N, 1N)•
  | f = [2N, 2N]o ⇒ ℱ(ArrMap) (1N, 1N)•
  | otherwise ⇒ ℱ(HomCod) (Arr)
)

```

and *RK23* ℱ(HomDom) = *cat-ordinal* (3_N)

and *RK23* ℱ(HomCod) = ℱ(HomCod)

unfolding *RK23-def dghm-field-simps* **by** (*simp-all add: nat-omega-simps*)

context *is-functor*

begin

lemmas *RK23-components'* = *RK23-components* [**where** ℱ=ℱ, *unfolded cat-cs-simps*]

lemmas [*cat-Kan-cs-simps*] = *RK23-components'* (3,4)

end

lemmas [*cat-Kan-cs-simps*] = *is-functor.RK23-components'* (3,4)

16.4.2 Object map

mk-VLambda *RK23-components* (1)

|*vsv* *RK23-ObjMap-vsv* [*cat-Kan-cs-intros*]

|*vdomain* *RK23-ObjMap-vdomain* [*cat-Kan-cs-simps*]

|*app* *RK23-ObjMap-app*

lemma *RK23-ObjMap-app-0* [*cat-Kan-cs-simps*]:

assumes $a = 0$
shows $RK23 \mathfrak{F}(\text{ObjMap})(\downarrow a) = \mathfrak{F}(\text{ObjMap})(\downarrow 0)$
unfolding $RK23\text{-components}$ *assms* $\text{cat-ordinal-components}$ **by** *simp*

lemma $RK23\text{-ObjMap-app-1}$ [cat-Kan-cs-simps]:
assumes $a = 1_{\mathbb{N}}$
shows $RK23 \mathfrak{F}(\text{ObjMap})(\downarrow a) = \mathfrak{F}(\text{ObjMap})(\downarrow 1_{\mathbb{N}})$
unfolding $RK23\text{-components}$ *assms* $\text{cat-ordinal-components}$ **by** *simp*

lemma $RK23\text{-ObjMap-app-2}$ [cat-Kan-cs-simps]:
assumes $a = 2_{\mathbb{N}}$
shows $RK23 \mathfrak{F}(\text{ObjMap})(\downarrow a) = \mathfrak{F}(\text{ObjMap})(\downarrow 1_{\mathbb{N}})$
unfolding $RK23\text{-components}$ *assms* $\text{cat-ordinal-components}$ **by** *simp*

16.4.3 Arrow map

mk-VLambda $RK23\text{-components}(2)$
 $|\text{vsu } RK23\text{-ArrMap-vsuv}[\text{cat-Kan-cs-intros}]|$
 $|\text{vdomain } RK23\text{-ArrMap-vdomain}[\text{cat-Kan-cs-simps}]|$
 $|\text{app } RK23\text{-ArrMap-app}|$

lemma $RK23\text{-ArrMap-app-00}$ [cat-Kan-cs-simps]:
assumes $f = [0, 0]_{\circ}$
shows $RK23 \mathfrak{F}(\text{ArrMap})(\downarrow f) = \mathfrak{F}(\text{ArrMap})(\downarrow 0, 0)$.
proof-
from 0123 **have** $f: f \in_{\circ} \text{cat-ordinal } (3_{\mathbb{N}})(\downarrow \text{Arr})$
by
(
 $\text{cs-concl } \mathbf{cs-shallow } \mathbf{cs-intro}$:
 $V\text{-cs-intros } \text{cat-ordinal-cs-intros } \text{cat-cs-intros } \text{assms}$
)
then show *?thesis* **unfolding** $RK23\text{-components}$ *assms* **by** *auto*
qed

lemma $RK23\text{-ArrMap-app-01}$ [cat-Kan-cs-simps]:
assumes $f = [0, 1_{\mathbb{N}}]_{\circ}$
shows $RK23 \mathfrak{F}(\text{ArrMap})(\downarrow f) = \mathfrak{F}(\text{ArrMap})(\downarrow 0, 1_{\mathbb{N}})$.
proof-
from 0123 **have** $f: f \in_{\circ} \text{cat-ordinal } (3_{\mathbb{N}})(\downarrow \text{Arr})$
by
(
 $\text{cs-concl } \mathbf{cs-shallow } \mathbf{cs-intro}$:
 $V\text{-cs-intros } \text{cat-ordinal-cs-intros } \text{cat-cs-intros } \text{assms}$
)
then show *?thesis* **unfolding** $RK23\text{-components}$ *assms* **by** *auto*
qed

lemma $RK23\text{-ArrMap-app-02}$ [cat-Kan-cs-simps]:
assumes $f = [0, 2_{\mathbb{N}}]_{\circ}$
shows $RK23 \mathfrak{F}(\text{ArrMap})(\downarrow f) = \mathfrak{F}(\text{ArrMap})(\downarrow 0, 1_{\mathbb{N}})$.
proof-
from 0123 **have** $f: f \in_{\circ} \text{cat-ordinal } (3_{\mathbb{N}})(\downarrow \text{Arr})$
by
(
 $\text{cs-concl } \mathbf{cs-shallow } \mathbf{cs-intro}$:
 $V\text{-cs-intros } \text{cat-ordinal-cs-intros } \text{cat-cs-intros } \text{assms}$
)
then show *?thesis* **unfolding** $RK23\text{-components}$ *assms* **by** *auto*

qed

lemma *RK23-ArrMap-app-11*[*cat-Kan-cs-simps*]:

assumes $f = [1_{\mathbb{N}}, 1_{\mathbb{N}}]_{\circ}$

shows $RK23 \mathfrak{F}(\downarrow ArrMap)(\downarrow f) = \mathfrak{F}(\downarrow ArrMap)(\downarrow 1_{\mathbb{N}}, 1_{\mathbb{N}})$.

proof-

from *0123* have $f: f \in_{\circ} \text{cat-ordinal } (\mathfrak{3}_{\mathbb{N}})(\downarrow Arr)$

by

(

cs-concl cs-shallow cs-intro:

V-cs-intros cat-ordinal-cs-intros cat-cs-intros assms

)

then show *?thesis unfolding RK23-components assms by auto*

qed

lemma *RK23-ArrMap-app-12*[*cat-Kan-cs-simps*]:

assumes $f = [1_{\mathbb{N}}, 2_{\mathbb{N}}]_{\circ}$

shows $RK23 \mathfrak{F}(\downarrow ArrMap)(\downarrow f) = \mathfrak{F}(\downarrow ArrMap)(\downarrow 1_{\mathbb{N}}, 1_{\mathbb{N}})$.

proof-

from *0123* have $f: f \in_{\circ} \text{cat-ordinal } (\mathfrak{3}_{\mathbb{N}})(\downarrow Arr)$

by

(

cs-concl

cs-simp: omega-of-set

cs-intro: nat-omega-intros cat-ordinal-cs-intros cat-cs-intros assms

)

then show *?thesis unfolding RK23-components assms by auto*

qed

lemma *RK23-ArrMap-app-22*[*cat-Kan-cs-simps*]:

assumes $f = [2_{\mathbb{N}}, 2_{\mathbb{N}}]_{\circ}$

shows $RK23 \mathfrak{F}(\downarrow ArrMap)(\downarrow f) = \mathfrak{F}(\downarrow ArrMap)(\downarrow 1_{\mathbb{N}}, 1_{\mathbb{N}})$.

proof-

from *0123* have $f: f \in_{\circ} \text{cat-ordinal } (\mathfrak{3}_{\mathbb{N}})(\downarrow Arr)$

by

(

cs-concl cs-shallow cs-intro:

nat-omega-intros cat-ordinal-cs-intros cat-cs-intros assms

)

then show *?thesis unfolding RK23-components assms by simp*

qed

16.4.4 *RK23* is a functor

lemma *cat-RK23-is-functor*:

assumes $\mathfrak{F} : \text{cat-ordinal } (2_{\mathbb{N}}) \mapsto \rightarrow_{C\alpha} \mathfrak{C}$

shows $RK23 \mathfrak{F} : \text{cat-ordinal } (3_{\mathbb{N}}) \mapsto \rightarrow_{C\alpha} \mathfrak{C}$

proof-

interpret \mathfrak{F} : *is-functor* $\alpha \langle \text{cat-ordinal } (2_{\mathbb{N}}) \rangle \mathfrak{C} \mathfrak{F}$ by (*rule assms(1)*)

from *ord-of-nat- ω* interpret *cat-ordinal-2: finite-category* $\alpha \langle \text{cat-ordinal } (2_{\mathbb{N}}) \rangle$

by (*cs-concl cs-shallow cs-intro: cat-ordinal-cs-intros*)

from *ord-of-nat- ω* interpret *cat-ordinal-3: finite-category* $\alpha \langle \text{cat-ordinal } (3_{\mathbb{N}}) \rangle$

by (*cs-concl cs-shallow cs-intro: cat-ordinal-cs-intros*)

interpret \mathfrak{F} : *is-functor* $\alpha \langle \text{cat-ordinal } (2_{\mathbb{N}}) \rangle \mathfrak{C} \mathfrak{F}$ by (*rule assms*)

show *?thesis*
proof(*intro is-functorI*)
show *vfsequence* ($RK23 \mathfrak{F}$) **unfolding** *RK23-def* **by** *auto*
show *vcard* ($RK23 \mathfrak{F}$) = $4_{\mathbb{N}}$ **unfolding** *RK23-def* **by** (*simp add: nat-omega-simps*)
show \mathcal{R}_o ($RK23 \mathfrak{F}(\text{ObjMap})$) $\subseteq_o \mathfrak{C}(\text{Obj})$
proof(*rule vsv.vsv-vrange-vsubset, unfold cat-Kan-cs-simps*)
fix x **assume** *prems*: $x \in_o \text{cat-ordinal } (3_{\mathbb{N}})(\text{Obj})$
then consider $\langle x = 0 \rangle \mid \langle x = 1_{\mathbb{N}} \rangle \mid \langle x = 2_{\mathbb{N}} \rangle$
unfolding *cat-ordinal-cs-simps three* **by** *auto*
then show $RK23 \mathfrak{F}(\text{ObjMap})(x) \in_o \mathfrak{C}(\text{Obj})$
by *cases*
(

cs-concl
cs-simp: *cat-Kan-cs-simps cat-ordinal-cs-simps omega-of-set*
cs-intro: *cat-cs-intros nat-omega-intros*
)+
qed (*cs-concl cs-shallow cs-intro: cat-Kan-cs-intros*)
show $RK23 \mathfrak{F}(\text{ArrMap})(f) : RK23 \mathfrak{F}(\text{ObjMap})(a) \mapsto_{\mathfrak{C}} RK23 \mathfrak{F}(\text{ObjMap})(b)$
if $f : a \mapsto_{\text{cat-ordinal } (3_{\mathbb{N}})} b$ **for** $a \ b \ f$
proof-
from *0123* **that** **show** *?thesis*
by (*elim cat-ordinal-3-is-arrE; simp only:*)
(

cs-concl
cs-simp: *cat-Kan-cs-simps*
cs-intro: *V-cs-intros cat-cs-intros cat-ordinal-cs-intros*
)+
qed
show
 $RK23 \mathfrak{F}(\text{ArrMap})(g \circ_A \text{cat-ordinal } (3_{\mathbb{N}}) f) =$
 $RK23 \mathfrak{F}(\text{ArrMap})(g) \circ_A \mathfrak{C} RK23 \mathfrak{F}(\text{ArrMap})(f)$
if $g : b \mapsto_{\text{cat-ordinal } (3_{\mathbb{N}})} c$ **and** $f : a \mapsto_{\text{cat-ordinal } (3_{\mathbb{N}})} b$
for $b \ c \ g \ a \ f$
using *0123* **that**
by (*elim cat-ordinal-3-is-arrE; simp only; (solves⟨simp⟩)?*)
(

cs-concl
cs-simp:
cat-ordinal-cs-simps
cat-Kan-cs-simps
 $\mathfrak{F}.cf\text{-ArrMap-Comp[symmetric]}$
cs-intro: *V-cs-intros cat-cs-intros cat-ordinal-cs-intros*
)+
show $RK23 \mathfrak{F}(\text{ArrMap})(\text{cat-ordinal } (3_{\mathbb{N}})(\text{CId})(c)) = \mathfrak{C}(\text{CId})(RK23 \mathfrak{F}(\text{ObjMap})(c))$
if $c \in_o \text{cat-ordinal } (3_{\mathbb{N}})(\text{Obj})$ **for** c
proof-
from *that* **consider** $\langle c = 0 \rangle \mid \langle c = 1_{\mathbb{N}} \rangle \mid \langle c = 2_{\mathbb{N}} \rangle$
unfolding *cat-ordinal-components three* **by** *auto*
then show *?thesis*
by (*cases, use 0123 in ⟨simp-all only:⟩*)
(

cs-concl
cs-simp:
cat-ordinal-cs-simps
cat-Kan-cs-simps
 $is\text{-functor}.cf\text{-ObjMap-CId[symmetric]}$
cs-intro: *V-cs-intros cat-cs-intros cat-ordinal-cs-intros*
)+

qed
 qed
 (
 cs-concl **cs-shallow**
 cs-simp: *cat-Kan-cs-simps* **cs-intro:** *cat-cs-intros cat-Kan-cs-intros*
)+

qed

lemma *cat-RK23-is-functor'*[*cat-Kan-cs-intros*]:
 assumes $\mathfrak{F} : \text{cat-ordinal } (2_{\mathbb{N}}) \mapsto \mapsto_{C\alpha} \mathfrak{C}$
 and $\mathfrak{A}' = \text{cat-ordinal } (3_{\mathbb{N}})$
 shows *RK23* $\mathfrak{F} : \mathfrak{A}' \mapsto \mapsto_{C\alpha} \mathfrak{C}$
 using *assms(1)* **unfolding** *assms(2)* **by** (*rule cat-RK23-is-functor*)

16.4.5 The fundamental property of *RK23*

lemma *cf-comp-RK23-℔23*[*cat-Kan-cs-simps*]:

assumes $\mathfrak{F} : \text{cat-ordinal } (2_{\mathbb{N}}) \mapsto \mapsto_{C\alpha} \mathfrak{C}$
 shows *RK23* $\mathfrak{F} \circ_{CF} \mathfrak{R}23 = \mathfrak{F}$

proof-

interpret \mathfrak{F} : *is-functor* $\alpha \langle \text{cat-ordinal } (2_{\mathbb{N}}) \rangle \mathfrak{C} \mathfrak{F}$ **by** (*rule assms(1)*)
interpret $\mathfrak{R}23$: *is-functor* $\alpha \langle \text{cat-ordinal } (2_{\mathbb{N}}) \rangle \langle \text{cat-ordinal } (3_{\mathbb{N}}) \rangle \langle \mathfrak{R}23 \rangle$
by (*cs-concl cs-shallow cs-intro: cat-cs-intros cat-Kan-cs-intros*)
interpret *RK23*: *is-functor* $\alpha \langle \text{cat-ordinal } (3_{\mathbb{N}}) \rangle \mathfrak{C} \langle \text{RK23 } \mathfrak{F} \rangle$
by (*cs-concl cs-intro: cat-Kan-cs-intros cat-cs-intros*)

show *?thesis*

proof(*rule cf-eqI*)

show $\mathfrak{F} : \text{cat-ordinal } (2_{\mathbb{N}}) \mapsto \mapsto_{C\alpha} \mathfrak{C}$ **by** (*rule assms*)
have *ObjMap-dom-lhs*: $\mathcal{D}_\circ ((\text{RK23 } \mathfrak{F} \circ_{CF} \mathfrak{R}23)(\text{ObjMap})) = 2_{\mathbb{N}}$
by

(
 cs-concl **cs-shallow**
 cs-simp: *cat-cs-simps cat-ordinal-cs-simps* **cs-intro:** *cat-cs-intros*
)

have *ObjMap-dom-rhs*: $\mathcal{D}_\circ (\mathfrak{F}(\text{ObjMap})) = 2_{\mathbb{N}}$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps cat-ordinal-cs-simps*)
show $(\text{RK23 } \mathfrak{F} \circ_{CF} \mathfrak{R}23)(\text{ObjMap}) = \mathfrak{F}(\text{ObjMap})$

proof(*rule vsu-eqI, unfold ObjMap-dom-lhs ObjMap-dom-rhs*)

fix *a* **assume** *prems*: $a \in_\circ 2_{\mathbb{N}}$
then consider $\langle a = 0 \rangle \mid \langle a = 1_{\mathbb{N}} \rangle$ **by force**
then show $(\text{RK23 } \mathfrak{F} \circ_{CF} \mathfrak{R}23)(\text{ObjMap})(\text{ObjMap})(a) = \mathfrak{F}(\text{ObjMap})(a)$
by (*cases, use nothing in <simp-all only>*)

(
 cs-concl
 cs-simp:
 omega-of-set cat-cs-simps cat-ordinal-cs-simps cat-Kan-cs-simps
 cs-intro: *cat-cs-intros nat-omega-intros*
)+

qed (*cs-concl cs-intro: cat-cs-intros V-cs-intros*)
have *ArrMap-dom-lhs*: $\mathcal{D}_\circ ((\text{RK23 } \mathfrak{F} \circ_{CF} \mathfrak{R}23)(\text{ArrMap})) = \text{cat-ordinal } (2_{\mathbb{N}})(\text{Arr})$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)
have *ArrMap-dom-rhs*: $\mathcal{D}_\circ (\mathfrak{F}(\text{ArrMap})) = \text{cat-ordinal } (2_{\mathbb{N}})(\text{Arr})$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)
show $(\text{RK23 } \mathfrak{F} \circ_{CF} \mathfrak{R}23)(\text{ArrMap}) = \mathfrak{F}(\text{ArrMap})$
proof(*rule vsu-eqI, unfold ArrMap-dom-lhs ArrMap-dom-rhs*)

fix f **assume** $\text{prems: } f \in_{\circ} \text{cat-ordinal } (2_{\mathbb{N}})(\text{Arr})$
then obtain a b **where** $f : a \mapsto_{\text{cat-ordinal } (2_{\mathbb{N}})} b$ **by** *auto*
then show $(RK23 \mathfrak{F} \circ_{CF} \mathfrak{K}23)(\text{ArrMap})(f) = \mathfrak{F}(\text{ArrMap})(f)$
by (*elim cat-ordinal-2-is-arrE; simp only:*)
(

 cs-concl
 cs-simp: *cat-cs-simps cat-Kan-cs-simps* **cs-intro:** *cat-cs-intros*

) +
qed (*cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros V-cs-intros*) +
qed (*cs-concl cs-intro: cat-Kan-cs-intros cat-cs-intros*)

qed

16.5 $RK\text{-}\sigma 23$: towards the universal property of the right Kan extension along $\mathfrak{K}23$

16.5.1 Definition and elementary properties

definition $RK\text{-}\sigma 23 :: V \Rightarrow V \Rightarrow V \Rightarrow V$

where $RK\text{-}\sigma 23 \mathfrak{T} \varepsilon' \mathfrak{F}' =$

[

(

 $\lambda a \in_{\circ} \text{cat-ordinal } (3_{\mathbb{N}})(\text{Obj})$.
 if $a = 0 \Rightarrow \varepsilon'(\text{NTMap})(0)$
 | $a = 1_{\mathbb{N}} \Rightarrow \varepsilon'(\text{NTMap})(1_{\mathbb{N}}) \circ_{A\mathfrak{T}}(\text{HomCod}) \mathfrak{F}'(\text{ArrMap})(1_{\mathbb{N}}, 2_{\mathbb{N}})$.
 | $a = 2_{\mathbb{N}} \Rightarrow \varepsilon'(\text{NTMap})(1_{\mathbb{N}})$
 | *otherwise* $\Rightarrow \mathfrak{T}(\text{HomCod})(\text{Arr})$

),
 \mathfrak{F}' ,
 $RK23 \mathfrak{T}$,
cat-ordinal $(3_{\mathbb{N}})$,
 $\mathfrak{F}'(\text{HomCod})$

]

Components.

lemma $RK\text{-}\sigma 23\text{-components}$:

shows $RK\text{-}\sigma 23 \mathfrak{T} \varepsilon' \mathfrak{F}'(\text{NTMap}) =$

(

 $\lambda a \in_{\circ} \text{cat-ordinal } (3_{\mathbb{N}})(\text{Obj})$.
 if $a = 0 \Rightarrow \varepsilon'(\text{NTMap})(0)$
 | $a = 1_{\mathbb{N}} \Rightarrow \varepsilon'(\text{NTMap})(1_{\mathbb{N}}) \circ_{A\mathfrak{T}}(\text{HomCod}) \mathfrak{F}'(\text{ArrMap})(1_{\mathbb{N}}, 2_{\mathbb{N}})$.
 | $a = 2_{\mathbb{N}} \Rightarrow \varepsilon'(\text{NTMap})(1_{\mathbb{N}})$
 | *otherwise* $\Rightarrow \mathfrak{T}(\text{HomCod})(\text{Arr})$

)

and $RK\text{-}\sigma 23 \mathfrak{T} \varepsilon' \mathfrak{F}'(\text{NTDom}) = \mathfrak{F}'$

and $RK\text{-}\sigma 23 \mathfrak{T} \varepsilon' \mathfrak{F}'(\text{NTCod}) = RK23 \mathfrak{T}$

and $RK\text{-}\sigma 23 \mathfrak{T} \varepsilon' \mathfrak{F}'(\text{NTDGDom}) = \text{cat-ordinal } (3_{\mathbb{N}})$

and $RK\text{-}\sigma 23 \mathfrak{T} \varepsilon' \mathfrak{F}'(\text{NTDGCod}) = \mathfrak{F}'(\text{HomCod})$

unfolding $RK\text{-}\sigma 23\text{-def nt-field-simps}$ **by** (*simp-all add: nat-omega-simps*)

context

fixes $\alpha \mathfrak{A} \mathfrak{F}' \mathfrak{T}$

assumes $\mathfrak{F}': \text{cat-ordinal } (3_{\mathbb{N}}) \mapsto_{\mapsto_{C\alpha}} \mathfrak{A}$

and $\mathfrak{T}: \text{cat-ordinal } (2_{\mathbb{N}}) \mapsto_{\mapsto_{C\alpha}} \mathfrak{A}$

begin

interpretation $\mathfrak{F}': \text{is-functor } \alpha \langle \text{cat-ordinal } (3_{\mathbb{N}}) \rangle \mathfrak{A} \mathfrak{F}'$ **by** (*rule* \mathfrak{F}')

interpretation $\mathfrak{T}: \text{is-functor } \alpha \langle \text{cat-ordinal } (2_{\mathbb{N}}) \rangle \mathfrak{A} \mathfrak{T}$ **by** (*rule* \mathfrak{T})

lemmas $RK\text{-}\sigma 23\text{-components}' =$
 $RK\text{-}\sigma 23\text{-components}[\text{where } \mathfrak{F}' = \mathfrak{F}' \text{ and } \mathfrak{T} = \mathfrak{T}, \text{ unfolded cat-cs-simps}]$

lemmas $[cat\text{-Kan-cs-simps}] = RK\text{-}\sigma 23\text{-components}'(2-5)$

end

16.5.2 Natural transformation map

mk-VLambda $RK\text{-}\sigma 23\text{-components}(1)$
 $|vsv\ RK\text{-}\sigma 23\text{-NTMap}\text{-vsv}[cat\text{-Kan-cs-intros}]$
 $|vdomain\ RK\text{-}\sigma 23\text{-NTMap}\text{-vdomain}[cat\text{-Kan-cs-simps}]$
 $|app\ RK\text{-}\sigma 23\text{-NTMap}\text{-app}$

lemma $RK\text{-}\sigma 23\text{-NTMap}\text{-app-0}[cat\text{-Kan-cs-simps}]$:
assumes $a = 0$
shows $RK\text{-}\sigma 23\ \mathfrak{T}\ \varepsilon'\ \mathfrak{F}'(\downarrow NTMap)(\downarrow a) = \varepsilon'(\downarrow NTMap)(\downarrow 0)$
using *assms unfolding $RK\text{-}\sigma 23\text{-components}$ cat-ordinal-cs-simps* **by** *simp*

lemma (in *is-functor*) $RK\text{-}\sigma 23\text{-NTMap}\text{-app-1}[cat\text{-Kan-cs-simps}]$:
assumes $a = 1_{\mathbb{N}}$
shows $RK\text{-}\sigma 23\ \mathfrak{F}\ \varepsilon'\ \mathfrak{F}'(\downarrow NTMap)(\downarrow a) = \varepsilon'(\downarrow NTMap)(\downarrow 1_{\mathbb{N}}) \circ_{A\mathfrak{B}} \mathfrak{F}'(\downarrow ArrMap)(\downarrow 1_{\mathbb{N}}, 2_{\mathbb{N}})$.
using *assms*
unfolding $RK\text{-}\sigma 23\text{-components}$ *cat-ordinal-cs-simps* *cat-cs-simps*
by *simp*

lemmas $[cat\text{-Kan-cs-simps}] = is\text{-functor}.RK\text{-}\sigma 23\text{-NTMap}\text{-app-1}$

lemma $RK\text{-}\sigma 23\text{-NTMap}\text{-app-2}[cat\text{-Kan-cs-simps}]$:
assumes $a = 2_{\mathbb{N}}$
shows $RK\text{-}\sigma 23\ \mathfrak{T}\ \varepsilon'\ \mathfrak{F}'(\downarrow NTMap)(\downarrow a) = \varepsilon'(\downarrow NTMap)(\downarrow 1_{\mathbb{N}})$
using *assms unfolding $RK\text{-}\sigma 23\text{-components}$ cat-ordinal-cs-simps* **by** *simp*

16.5.3 $RK\text{-}\sigma 23$ is a natural transformation

lemma $RK\text{-}\sigma 23\text{-is-ntcf}$:
assumes $\mathfrak{F}' : cat\text{-ordinal}\ (3_{\mathbb{N}}) \mapsto_{C\alpha} \mathfrak{A}$
and $\mathfrak{T} : cat\text{-ordinal}\ (2_{\mathbb{N}}) \mapsto_{C\alpha} \mathfrak{A}$
and $\varepsilon' : \mathfrak{F}' \circ_{CF} \mathfrak{R}23 \mapsto_{CF} \mathfrak{T} : cat\text{-ordinal}\ (2_{\mathbb{N}}) \mapsto_{C\alpha} \mathfrak{A}$
shows $RK\text{-}\sigma 23\ \mathfrak{T}\ \varepsilon'\ \mathfrak{F}' : \mathfrak{F}' \mapsto_{CF} RK23\ \mathfrak{T} : cat\text{-ordinal}\ (3_{\mathbb{N}}) \mapsto_{C\alpha} \mathfrak{A}$
proof-

interpret $\mathfrak{F}' : is\text{-functor}\ \alpha\ \langle cat\text{-ordinal}\ (3_{\mathbb{N}}) \rangle\ \mathfrak{A}\ \mathfrak{F}'$ **by** (*rule assms(1)*)
interpret $\mathfrak{T} : is\text{-functor}\ \alpha\ \langle cat\text{-ordinal}\ (2_{\mathbb{N}}) \rangle\ \mathfrak{A}\ \mathfrak{T}$ **by** (*rule assms(2)*)
interpret $\varepsilon' : is\text{-ntcf}\ \alpha\ \langle cat\text{-ordinal}\ (2_{\mathbb{N}}) \rangle\ \mathfrak{A}\ \langle \mathfrak{F}' \circ_{CF} \mathfrak{R}23 \rangle\ \mathfrak{T}\ \varepsilon'$
by (*rule assms(3)*)

interpret $\mathfrak{R}23 : is\text{-functor}\ \alpha\ \langle cat\text{-ordinal}\ (2_{\mathbb{N}}) \rangle\ \langle cat\text{-ordinal}\ (3_{\mathbb{N}}) \rangle\ \langle \mathfrak{R}23 \rangle$
by (*cs-concl cs-shallow cs-intro: cat-cs-intros cat-Kan-cs-intros*)
interpret $RK23 : is\text{-functor}\ \alpha\ \langle cat\text{-ordinal}\ (3_{\mathbb{N}}) \rangle\ \mathfrak{A}\ \langle RK23\ \mathfrak{T} \rangle$
by (*cs-concl cs-intro: cat-Kan-cs-intros cat-cs-intros*)

from *0123* **have** $[cat\text{-cs-simps}] : \mathfrak{T}(\downarrow ArrMap)(\downarrow 1_{\mathbb{N}}, 1_{\mathbb{N}})_{\bullet} = \mathfrak{A}(\downarrow CId)(\downarrow \mathfrak{T}(\downarrow ObjMap)(\downarrow 1_{\mathbb{N}}))$
by
 (
cs-concl cs-shallow
cs-simp: *cat-ordinal-cs-simps is-functor.cf-ObjMap-CId[symmetric]*)

cs-intro: *cat-cs-intros*
)

show *?thesis*
proof(*rule is-ntcfI'*)

show *vfsequence (RK-σ23 ℑ ε' ℑ')* **unfolding** *RK-σ23-def* **by** *simp*
show *vcard (RK-σ23 ℑ ε' ℑ') = 5_N*
unfolding *RK-σ23-def* **by** (*simp-all add: nat-omega-simps*)
show *RK-σ23 ℑ ε' ℑ' (NTMap) (|a|) : ℑ' (ObjMap) (|a|) ↦_ℑ RK23 ℑ (ObjMap) (|a|)*
if *a ∈_o cat-ordinal (3_N) (Obj)* **for** *a*
proof-

from *that consider* *⟨a = 0⟩ | ⟨a = 1_N⟩ | ⟨a = 2_N⟩*
unfolding *cat-ordinal-cs-simps three* **by** *auto*
from *this 0123* **show** *?thesis*
by (*cases, use nothing in* *⟨simp-all only:⟩*)

(

cs-concl
cs-simp: *cat-cs-simps cat-ordinal-cs-simps cat-Kan-cs-simps*
cs-intro:
cat-cs-intros
cat-ordinal-cs-intros
cat-Kan-cs-intros
nat-omega-intros

)+

qed
show

RK-σ23 ℑ ε' ℑ' (NTMap) (|b|) ∘_{Aℑ} ℑ' (ArrMap) (|f|) =
RK23 ℑ (ArrMap) (|f|) ∘_{Aℑ} RK-σ23 ℑ ε' ℑ' (NTMap) (|a|)
if *f : a ↦_{cat-ordinal (3_N)} b* **for** *a b f*
using *that 0123*
by (*elim cat-ordinal-3-is-arrE, use nothing in* *⟨simp-all only:⟩*)

(

cs-concl
cs-simp:
cat-cs-simps
cat-ordinal-cs-simps
ℑ'.cf-ArrMap-Comp[symmetric]
ℑ'.HomCod.cat-Comp-assoc
ε'.ntcf-Comp-commute[symmetric]
cat-Kan-cs-simps
cs-intro: *cat-cs-intros cat-ordinal-cs-intros nat-omega-intros*

)+

qed
(

cs-concl
cs-simp: *cat-Kan-cs-simps* **cs-intro:** *cat-cs-intros cat-Kan-cs-intros*

)+

qed

lemma *RK-σ23-is-ntcf'[cat-Kan-cs-intros]:*
assumes *ℑ' : cat-ordinal (3_N) ↦_{→Cα} ℑ*
and *ℑ : cat-ordinal (2_N) ↦_{→Cα} ℑ*
and *ε' : ℑ' ∘_{CF} ℑ23 ↦_{CF} ℑ : cat-ordinal (2_N) ↦_{→Cα} ℑ*
and *ℑ' = ℑ'*
and *ℑ' = RK23 ℑ*
and *ℑ' = cat-ordinal (3_N)*
shows *RK-σ23 ℑ ε' ℑ' : ℑ' ↦_{CF} ℑ' : ℑ' ↦_{→Cα} ℑ*

using *assms(1-3)* unfolding *assms(4-6)* by (rule *RK-σ23-is-ntcf*)

16.6 The right Kan extension along $\mathfrak{K}23$

lemma *ε23-is-cat-rKe*:

assumes $\mathfrak{T} : \text{cat-ordinal } (2_{\mathbb{N}}) \mapsto_{C\alpha} \mathfrak{A}$

shows *ntcf-id* \mathfrak{T} :

$RK23 \mathfrak{T} \circ_{CF} \mathfrak{K}23 \mapsto_{CF} \tau_{Ke\alpha} \mathfrak{T} : \text{cat-ordinal } (2_{\mathbb{N}}) \mapsto_C \text{cat-ordinal } (3_{\mathbb{N}}) \mapsto_C \mathfrak{A}$

proof-

interpret \mathfrak{T} : *is-functor* $\alpha \langle \text{cat-ordinal } (2_{\mathbb{N}}) \rangle \mathfrak{A} \mathfrak{T}$ by (rule *assms(1)*)

interpret $\mathfrak{K}23$: *is-functor* $\alpha \langle \text{cat-ordinal } (2_{\mathbb{N}}) \rangle \langle \text{cat-ordinal } (3_{\mathbb{N}}) \rangle \langle \mathfrak{K}23 \rangle$

by (*cs-concl cs-shallow cs-intro*: *cat-cs-intros cat-Kan-cs-intros*)

interpret $RK23$: *is-functor* $\alpha \langle \text{cat-ordinal } (3_{\mathbb{N}}) \rangle \mathfrak{A} \langle RK23 \mathfrak{T} \rangle$

by (*cs-concl cs-intro*: *cat-Kan-cs-intros cat-cs-intros*)

from *0123* have [*cat-cs-simps*]: $\mathfrak{T}(\text{ArrMap})(\langle 1_{\mathbb{N}}, 1_{\mathbb{N}} \rangle)_{\bullet} = \mathfrak{A}(\text{CId})(\langle \mathfrak{T}(\text{ObjMap})(\langle 1_{\mathbb{N}} \rangle) \rangle)$

by

(
cs-concl cs-shallow
cs-simp: *cat-ordinal-cs-simps is-functor.cf-ObjMap-CId[symmetric]*
cs-intro: *cat-cs-intros*
)

show *?thesis*

proof(*intro is-cat-rKeI*)

fix $\mathfrak{F}' \varepsilon'$ assume *prems*:

$\mathfrak{F}' : \text{cat-ordinal } (3_{\mathbb{N}}) \mapsto_{C\alpha} \mathfrak{A}$

$\varepsilon' : \mathfrak{F}' \circ_{CF} \mathfrak{K}23 \mapsto_{CF} \mathfrak{T} : \text{cat-ordinal } (2_{\mathbb{N}}) \mapsto_{C\alpha} \mathfrak{A}$

interpret \mathfrak{F}' : *is-functor* $\alpha \langle \text{cat-ordinal } (3_{\mathbb{N}}) \rangle \mathfrak{A} \mathfrak{F}'$ by (rule *prems(1)*)

interpret ε' : *is-ntcf* $\alpha \langle \text{cat-ordinal } (2_{\mathbb{N}}) \rangle \mathfrak{A} \langle \mathfrak{F}' \circ_{CF} \mathfrak{K}23 \rangle \mathfrak{T} \varepsilon'$

by (rule *prems(2)*)

interpret $RK\text{-}\sigma 23$: *is-ntcf* $\alpha \langle \text{cat-ordinal } (3_{\mathbb{N}}) \rangle \mathfrak{A} \mathfrak{F}' \langle RK23 \mathfrak{T} \rangle \langle RK\text{-}\sigma 23 \mathfrak{T} \varepsilon' \mathfrak{F}' \rangle$

by (*intro RK-σ23-is-ntcf prems assms*)

show $\exists! \sigma$.

$\sigma : \mathfrak{F}' \mapsto_{CF} RK23 \mathfrak{T} : \text{cat-ordinal } (3_{\mathbb{N}}) \mapsto_{C\alpha} \mathfrak{A} \wedge$

$\varepsilon' = \text{ntcf-id } \mathfrak{T} \cdot_{NTCF} (\sigma \circ_{NTCF-CF} \mathfrak{K}23)$

proof(*intro ex1I conjI; (elim conjE)?*)

show $RK\text{-}\sigma 23 \mathfrak{T} \varepsilon' \mathfrak{F}' : \mathfrak{F}' \mapsto_{CF} RK23 \mathfrak{T} : \text{cat-ordinal } (3_{\mathbb{N}}) \mapsto_{C\alpha} \mathfrak{A}$

by (*intro RK-σ23.is-ntcf-axioms*)

show $\varepsilon' = \text{ntcf-id } \mathfrak{T} \cdot_{NTCF} (RK\text{-}\sigma 23 \mathfrak{T} \varepsilon' \mathfrak{F}' \circ_{NTCF-CF} \mathfrak{K}23)$

proof(*rule ntcf-eqI*)

show $\varepsilon' : \mathfrak{F}' \circ_{CF} \mathfrak{K}23 \mapsto_{CF} \mathfrak{T} : \text{cat-ordinal } (2_{\mathbb{N}}) \mapsto_{C\alpha} \mathfrak{A}$

by (*intro prems*)

then have *dom-lhs*: $\mathcal{D}_o(\varepsilon'(\text{NTMap})) = 2_{\mathbb{N}}$

by (*cs-concl cs-shallow cs-simp*: *cat-ordinal-cs-simps cat-cs-simps*)

show *rhs*:

$\text{ntcf-id } \mathfrak{T} \cdot_{NTCF} (RK\text{-}\sigma 23 \mathfrak{T} \varepsilon' \mathfrak{F}' \circ_{NTCF-CF} \mathfrak{K}23) :$

$\mathfrak{F}' \circ_{CF} \mathfrak{K}23 \mapsto_{CF} \mathfrak{T} : \text{cat-ordinal } (2_{\mathbb{N}}) \mapsto_{C\alpha} \mathfrak{A}$

by

(
cs-concl cs-shallow
cs-simp: *cat-Kan-cs-simps cat-cs-simps*
cs-intro: *cat-Kan-cs-intros cat-cs-intros*
)

then have *dom-rhs*:
 $\mathcal{D}_\circ ((ntcf-id \mathfrak{T} \cdot_{NTCF} (RK-\sigma 23 \mathfrak{T} \varepsilon' \mathfrak{F}' \circ_{NTCF-CF} \mathfrak{R}23)) (NTMap)) = 2_{\mathbb{N}}$
by (*cs-concl cs-simp: cat-ordinal-cs-simps cat-cs-simps*)
show $\varepsilon' (NTMap) = (ntcf-id \mathfrak{T} \cdot_{NTCF} (RK-\sigma 23 \mathfrak{T} \varepsilon' \mathfrak{F}' \circ_{NTCF-CF} \mathfrak{R}23)) (NTMap)$
proof(*rule vsv-eqI, unfold dom-lhs dom-rhs*)
fix *a* **assume** *prems: a* $\in_\circ 2_{\mathbb{N}}$
then consider $\langle a = 0 \rangle \mid \langle a = 1_{\mathbb{N}} \rangle$ **unfolding** *two* **by** *auto*
then show
 $\varepsilon' (NTMap) (a) =$
 $(ntcf-id \mathfrak{T} \cdot_{NTCF} (RK-\sigma 23 \mathfrak{T} \varepsilon' \mathfrak{F}' \circ_{NTCF-CF} \mathfrak{R}23)) (NTMap) (a)$
by (*cases; use nothing in* $\langle simp-all \text{ only} \rangle$)
(

cs-concl
cs-simp:
omega-of-set
cat-Kan-cs-simps
cat-cs-simps
cat-ordinal-cs-simps
cs-intro: *cat-Kan-cs-intros cat-cs-intros nat-omega-intros*
)
qed
(

use rhs in
 $\langle cs-concl cs-shallow cs-intro: V-cs-intros cat-cs-intros \rangle$
)
qed *simp-all*

fix σ **assume** *prems'*:
 $\sigma : \mathfrak{F}' \mapsto_{CF} RK23 \mathfrak{T} : cat-ordinal (3_{\mathbb{N}}) \mapsto_{C\alpha} \mathfrak{A}$
 $\varepsilon' = ntcf-id \mathfrak{T} \cdot_{NTCF} (\sigma \circ_{NTCF-CF} \mathfrak{R}23)$

interpret σ : *is-ntcf* α $\langle cat-ordinal (3_{\mathbb{N}}) \rangle \mathfrak{A} \mathfrak{F}' \langle RK23 \mathfrak{T} \rangle \sigma$
by (*rule prems'(1)*)

from *prems'(2)* **have**
 $\varepsilon' (NTMap) (0) = (ntcf-id \mathfrak{T} \cdot_{NTCF} (\sigma \circ_{NTCF-CF} \mathfrak{R}23)) (NTMap) (0)$
by *auto*
then have [*cat-cs-simps*]: $\varepsilon' (NTMap) (0) = \sigma (NTMap) (0)$
by
(

cs-prems cs-shallow
cs-simp: *cat-Kan-cs-simps cat-cs-simps cat-ordinal-cs-simps*
cs-intro: *cat-cs-intros nat-omega-intros*
)
from *prems'(2)* **have**
 $\varepsilon' (NTMap) (1_{\mathbb{N}}) = (ntcf-id \mathfrak{T} \cdot_{NTCF} (\sigma \circ_{NTCF-CF} \mathfrak{R}23)) (NTMap) (1_{\mathbb{N}})$
by *auto*
then have [*cat-cs-simps*]: $\varepsilon' (NTMap) (1_{\mathbb{N}}) = \sigma (NTMap) (1_{\mathbb{N}})$
by
(

cs-prems cs-shallow
cs-simp:
omega-of-set cat-Kan-cs-simps cat-cs-simps cat-ordinal-cs-simps
cs-intro: *cat-cs-intros nat-omega-intros*
)
show $\sigma = RK-\sigma 23 \mathfrak{T} \varepsilon' \mathfrak{F}'$
proof(*rule ntcf-eqI*)

```

show  $\sigma : \mathfrak{F}' \mapsto_{CF} RK23 \mathfrak{T} : \text{cat-ordinal } (3_N) \mapsto_{C\alpha} \mathfrak{A}$ 
  by (rule prems'(1))
then have dom-lhs:  $\mathcal{D}_\circ (\sigma(\mathcal{NTMap})) = 3_N$ 
  by (cs-concl cs-shallow cs-simp: cat-cs-simps cat-ordinal-cs-simps)
show  $RK\text{-}\sigma 23 \mathfrak{T} \varepsilon' \mathfrak{F}' : \mathfrak{F}' \mapsto_{CF} RK23 \mathfrak{T} : \text{cat-ordinal } (3_N) \mapsto_{C\alpha} \mathfrak{A}$ 
  by (cs-concl cs-intro: cat-cs-intros cat-Kan-cs-intros)
then have dom-rhs:  $\mathcal{D}_\circ (RK\text{-}\sigma 23 \mathfrak{T} \varepsilon' \mathfrak{F}'(\mathcal{NTMap})) = 3_N$ 
  by (cs-concl cs-shallow cs-simp: cat-cs-simps cat-ordinal-cs-simps)
from 0123 have 013:  $[0, 1_N]_\circ : 0 \mapsto_{\text{cat-ordinal } (3_N)} 1_N$ 
  by (cs-concl cs-intro: cat-ordinal-cs-intros nat-omega-intros)
from 0123 have 123:  $[1_N, 2_N]_\circ : 1_N \mapsto_{\text{cat-ordinal } (3_N)} 2_N$ 
  by (cs-concl cs-intro: cat-ordinal-cs-intros nat-omega-intros)

from  $\sigma.\text{ntcf-Comp-commute}[OF 123] 013 0123$ 
have [symmetric, cat-Kan-cs-simps]:
 $\sigma(\mathcal{NTMap})([2_N]) \circ_{A\mathfrak{A}} \mathfrak{F}'(\mathcal{ArrMap})([1_N, 2_N])_\bullet = \sigma(\mathcal{NTMap})([1_N])$ 
  by
  (
    cs-prems
    cs-simp: cat-cs-simps cat-Kan-cs-simps RK23-ArrMap-app-12
    cs-intro: cat-cs-intros
  )
show  $\sigma(\mathcal{NTMap}) = RK\text{-}\sigma 23 \mathfrak{T} \varepsilon' \mathfrak{F}'(\mathcal{NTMap})$ 
proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
  fix a assume prems:  $a \in_\circ 3_N$ 
  then consider  $\langle a = 0 \rangle \mid \langle a = 1_N \rangle \mid \langle a = 2_N \rangle$  unfolding three by auto
  then show  $\sigma(\mathcal{NTMap})([a]) = RK\text{-}\sigma 23 \mathfrak{T} \varepsilon' \mathfrak{F}'(\mathcal{NTMap})([a])$ 
    by (cases; use nothing in ⟨simp-all only⟩)
    (cs-concl cs-simp: cat-cs-simps cat-Kan-cs-simps)+
  qed auto
qed simp-all

qed

qed (cs-concl cs-shallow cs-simp: cat-Kan-cs-simps cs-intro: cat-cs-intros)+

```

qed

16.7 $LK\text{-}\sigma 23$: towards the universal property of the left Kan extension along $\mathcal{R}23$

16.7.1 Definition and elementary properties

definition $LK\text{-}\sigma 23 :: V \Rightarrow V \Rightarrow V \Rightarrow V$

where $LK\text{-}\sigma 23 \mathfrak{T} \eta' \mathfrak{F}' =$

```

[
  (
     $\lambda a \in_\circ \text{cat-ordinal } (3_N)([Obj]).$ 
    if  $a = 0 \Rightarrow \eta'(\mathcal{NTMap})([0])$ 
    |  $a = 1_N \Rightarrow \mathfrak{F}'(\mathcal{ArrMap})([0, 1_N])_\bullet \circ_{A\mathfrak{T}}(\mathcal{HomCod}) \eta'(\mathcal{NTMap})([0])$ 
    |  $a = 2_N \Rightarrow \eta'(\mathcal{NTMap})([1_N])$ 
    | otherwise  $\Rightarrow \mathfrak{T}(\mathcal{HomCod})([Arr])$ 
  ),
   $LK23 \mathfrak{T},$ 
   $\mathfrak{F}'$ ,
   $\text{cat-ordinal } (3_N),$ 
   $\mathfrak{F}'(\mathcal{HomCod})$ 
]_o

```

Components.

lemma *LK- σ 23-components*:

shows *LK- σ 23* \mathfrak{T} η' $\mathfrak{F}'(\text{NTMap}) =$
 (
 $\lambda a \in_{\circ} \text{cat-ordinal } (\mathfrak{I}_{\mathbb{N}})(\text{Obj})$.
 if $a = 0 \Rightarrow \eta'(\text{NTMap})(\text{!}0)$
 | $a = 1_{\mathbb{N}} \Rightarrow \mathfrak{F}'(\text{ArrMap})(\text{!}0, 1_{\mathbb{N}}) \bullet \circ_A \mathfrak{T}(\text{HomCod}) \eta'(\text{NTMap})(\text{!}0)$
 | $a = 2_{\mathbb{N}} \Rightarrow \eta'(\text{NTMap})(\text{!}1_{\mathbb{N}})$
 | otherwise $\Rightarrow \mathfrak{T}(\text{HomCod})(\text{!}Arr)$
)
and *LK- σ 23* \mathfrak{T} η' $\mathfrak{F}'(\text{NTDom}) = \text{LK}23 \mathfrak{T}$
and *LK- σ 23* \mathfrak{T} η' $\mathfrak{F}'(\text{NTCod}) = \mathfrak{F}'$
and *LK- σ 23* \mathfrak{T} η' $\mathfrak{F}'(\text{NTDGDom}) = \text{cat-ordinal } (\mathfrak{I}_{\mathbb{N}})$
and *LK- σ 23* \mathfrak{T} η' $\mathfrak{F}'(\text{NTDGCod}) = \mathfrak{F}'(\text{HomCod})$
unfolding *LK- σ 23-def nt-field-simps* **by** (*simp-all add: nat-omega-simps*)

context

fixes $\alpha \mathfrak{A} \mathfrak{F}' \mathfrak{T}$
assumes $\mathfrak{F}': \mathfrak{F}' : \text{cat-ordinal } (\mathfrak{I}_{\mathbb{N}}) \mapsto \mapsto_{C\alpha} \mathfrak{A}$
and $\mathfrak{T}: \mathfrak{T} : \text{cat-ordinal } (\mathfrak{I}_{\mathbb{N}}) \mapsto \mapsto_{C\alpha} \mathfrak{A}$

begin

interpretation \mathfrak{F}' : *is-functor* $\alpha \langle \text{cat-ordinal } (\mathfrak{I}_{\mathbb{N}}) \rangle \mathfrak{A} \mathfrak{F}'$ **by** (*rule* \mathfrak{F}')

interpretation \mathfrak{T} : *is-functor* $\alpha \langle \text{cat-ordinal } (\mathfrak{I}_{\mathbb{N}}) \rangle \mathfrak{A} \mathfrak{T}$ **by** (*rule* \mathfrak{T})

lemmas *LK- σ 23-components'* =

LK- σ 23-components[**where** $\mathfrak{F}' = \mathfrak{F}'$ **and** $\mathfrak{T} = \mathfrak{T}$, *unfolded cat-cs-simps*]

lemmas [*cat-Kan-cs-simps*] = *LK- σ 23-components'*(2-5)

end

16.7.2 Natural transformation map

mk-VLambda *LK- σ 23-components(1)*

|*vsv LK- σ 23-NTMap-vsuv*[*cat-Kan-cs-intros*]
 |*vdomain LK- σ 23-NTMap-vdomain*[*cat-Kan-cs-simps*]
 |*app LK- σ 23-NTMap-app*]

lemma *LK- σ 23-NTMap-app-0*[*cat-Kan-cs-simps*]:

assumes $a = 0$
shows *LK- σ 23* \mathfrak{T} η' $\mathfrak{F}'(\text{NTMap})(\text{!}a) = \eta'(\text{NTMap})(\text{!}0)$
using *assms unfolding LK- σ 23-components cat-ordinal-cs-simps* **by** *simp*

lemma (**in** *is-functor*) *LK- σ 23-NTMap-app-1*[*cat-Kan-cs-simps*]:

assumes $a = 1_{\mathbb{N}}$
shows *LK- σ 23* \mathfrak{T} η' $\mathfrak{F}'(\text{NTMap})(\text{!}a) = \mathfrak{F}'(\text{ArrMap})(\text{!}0, 1_{\mathbb{N}}) \bullet \circ_A \mathfrak{T} \eta'(\text{NTMap})(\text{!}0)$
using *assms unfolding LK- σ 23-components cat-ordinal-cs-simps cat-cs-simps* **by** *simp*

lemmas [*cat-Kan-cs-simps*] = *is-functor.LK- σ 23-NTMap-app-1*

lemma *LK- σ 23-NTMap-app-2*[*cat-Kan-cs-simps*]:

assumes $a = 2_{\mathbb{N}}$
shows *LK- σ 23* \mathfrak{T} η' $\mathfrak{F}'(\text{NTMap})(\text{!}a) = \eta'(\text{NTMap})(\text{!}1_{\mathbb{N}})$
using *assms unfolding LK- σ 23-components cat-ordinal-cs-simps* **by** *simp*

16.7.3 $LK\text{-}\sigma 23$ is a natural transformation

lemma $LK\text{-}\sigma 23\text{-is-ntcf}$:

assumes $\mathfrak{F}' : \text{cat-ordinal } (3_{\mathbb{N}}) \mapsto_{C\alpha} \mathfrak{A}$
 and $\mathfrak{T} : \text{cat-ordinal } (2_{\mathbb{N}}) \mapsto_{C\alpha} \mathfrak{A}$
 and $\eta' : \mathfrak{T} \mapsto_{CF} \mathfrak{F}' \circ_{CF} \mathfrak{K}23 : \text{cat-ordinal } (2_{\mathbb{N}}) \mapsto_{C\alpha} \mathfrak{A}$
 shows $LK\text{-}\sigma 23 \mathfrak{T} \eta' \mathfrak{F}' : LK23 \mathfrak{T} \mapsto_{CF} \mathfrak{F}' : \text{cat-ordinal } (3_{\mathbb{N}}) \mapsto_{C\alpha} \mathfrak{A}$

proof-

interpret \mathfrak{F}' : is-functor $\alpha \langle \text{cat-ordinal } (3_{\mathbb{N}}) \rangle \mathfrak{A} \mathfrak{F}'$ by (rule *assms(1)*)
 interpret \mathfrak{T} : is-functor $\alpha \langle \text{cat-ordinal } (2_{\mathbb{N}}) \rangle \mathfrak{A} \mathfrak{T}$ by (rule *assms(2)*)
 interpret η' : is-ntcf $\alpha \langle \text{cat-ordinal } (2_{\mathbb{N}}) \rangle \mathfrak{A} \mathfrak{T} \langle \mathfrak{F}' \circ_{CF} \mathfrak{K}23 \rangle \eta'$
 by (rule *assms(3)*)

interpret $\mathfrak{K}23$: is-functor $\alpha \langle \text{cat-ordinal } (2_{\mathbb{N}}) \rangle \langle \text{cat-ordinal } (3_{\mathbb{N}}) \rangle \langle \mathfrak{K}23 \rangle$
 by (cs-concl **cs-shallow cs-intro**: *cat-cs-intros cat-Kan-cs-intros*)
 interpret $LK23$: is-functor $\alpha \langle \text{cat-ordinal } (3_{\mathbb{N}}) \rangle \mathfrak{A} \langle LK23 \mathfrak{T} \rangle$
 by (cs-concl **cs-intro**: *cat-Kan-cs-intros cat-cs-intros*)

show ?thesis

proof(rule *is-ntcfI'*)

show *ufsequence* ($LK\text{-}\sigma 23 \mathfrak{T} \eta' \mathfrak{F}'$) unfolding $LK\text{-}\sigma 23\text{-def}$ by *simp*

show *vcard* ($LK\text{-}\sigma 23 \mathfrak{T} \eta' \mathfrak{F}'$) = $5_{\mathbb{N}}$

unfolding $LK\text{-}\sigma 23\text{-def}$ by (*simp-all add: nat-omega-simps*)

show $LK\text{-}\sigma 23 \mathfrak{T} \eta' \mathfrak{F}'(\text{NTMap})(\!|a) : LK23 \mathfrak{T}(\text{ObjMap})(\!|a) \mapsto_{\mathfrak{A}} \mathfrak{F}'(\text{ObjMap})(\!|a)$

if $a \in_{\circ} \text{cat-ordinal } (3_{\mathbb{N}})(\!|Obj)$ for a

proof-

from that consider $\langle a = 0 \rangle \mid \langle a = 1_{\mathbb{N}} \rangle \mid \langle a = 2_{\mathbb{N}} \rangle$

unfolding *cat-ordinal-cs-simps three* by *auto*

from this *0123* show

$LK\text{-}\sigma 23 \mathfrak{T} \eta' \mathfrak{F}'(\text{NTMap})(\!|a) : LK23 \mathfrak{T}(\text{ObjMap})(\!|a) \mapsto_{\mathfrak{A}} \mathfrak{F}'(\text{ObjMap})(\!|a)$

by (*cases, use nothing in <simp-all only>*)

(

cs-concl

cs-simp: *cat-cs-simps cat-ordinal-cs-simps cat-Kan-cs-simps*

cs-intro:

cat-cs-intros

cat-ordinal-cs-intros

cat-Kan-cs-intros

nat-omega-intros

)+

qed

show

$LK\text{-}\sigma 23 \mathfrak{T} \eta' \mathfrak{F}'(\text{NTMap})(\!|b) \circ_{A\mathfrak{A}} LK23 \mathfrak{T}(\text{ArrMap})(\!|f) =$

$\mathfrak{F}'(\text{ArrMap})(\!|f) \circ_{A\mathfrak{A}} LK\text{-}\sigma 23 \mathfrak{T} \eta' \mathfrak{F}'(\text{NTMap})(\!|a)$

if $f : a \mapsto_{\text{cat-ordinal } (3_{\mathbb{N}})} b$ for $a b f$

using that *0123*

by (*elim cat-ordinal-3-is-arrE, use nothing in <simp-all only>*)

(

cs-concl

cs-simp:

cat-cs-simps

cat-ordinal-cs-simps

$\mathfrak{F}'.\text{cf-ArrMap-Comp}[\text{symmetric}]$

$\mathfrak{F}'.\text{HomCod.cat-Comp-assoc}[\text{symmetric}]$

$\eta'.\text{ntcf-Comp-commute}$

cat-Kan-cs-simps

cs-intro: *cat-cs-intros cat-ordinal-cs-intros nat-omega-intros*

)+
qed
 (
 cs-concl
 cs-simp: *cat-Kan-cs-simps* **cs-intro**: *cat-cs-intros cat-Kan-cs-intros*
)+

qed

lemma *LK-σ23-is-ntcf'*[*cat-Kan-cs-intros*]:

assumes $\mathfrak{F}' : \text{cat-ordinal } (3_{\mathbb{N}}) \mapsto \mapsto_{C\alpha} \mathfrak{A}$
and $\mathfrak{T} : \text{cat-ordinal } (2_{\mathbb{N}}) \mapsto \mapsto_{C\alpha} \mathfrak{A}$
and $\eta' : \mathfrak{T} \mapsto_{CF} \mathfrak{F}' \circ_{CF} \mathfrak{R}23 : \text{cat-ordinal } (2_{\mathbb{N}}) \mapsto \mapsto_{C\alpha} \mathfrak{A}$
and $\mathfrak{G}' = \text{LK}23 \mathfrak{T}$
and $\mathfrak{H}' = \mathfrak{F}'$
and $\mathfrak{C}' = \text{cat-ordinal } (3_{\mathbb{N}})$
shows *LK-σ23* $\mathfrak{T} \eta' \mathfrak{F}' : \mathfrak{G}' \mapsto_{CF} \mathfrak{H}' : \mathfrak{C}' \mapsto \mapsto_{C\alpha} \mathfrak{A}$
using *assms(1-3)* **unfolding** *assms(4-6)* **by** (*rule LK-σ23-is-ntcf*)

16.8 The left Kan extension along $\mathfrak{R}23$

lemma *η23-is-cat-rKe*:

assumes $\mathfrak{T} : \text{cat-ordinal } (2_{\mathbb{N}}) \mapsto \mapsto_{C\alpha} \mathfrak{A}$
shows *ntcf-id* \mathfrak{T} :
 $\mathfrak{T} \mapsto_{CF.lKe\alpha} \text{LK}23 \mathfrak{T} \circ_{CF} \mathfrak{R}23 : \text{cat-ordinal } (2_{\mathbb{N}}) \mapsto_C \text{cat-ordinal } (3_{\mathbb{N}}) \mapsto_C \mathfrak{A}$

proof-

interpret \mathfrak{T} : *is-functor* $\alpha \langle \text{cat-ordinal } (2_{\mathbb{N}}) \rangle \mathfrak{A} \mathfrak{T}$ **by** (*rule assms(1)*)
interpret $\mathfrak{R}23$: *is-functor* $\alpha \langle \text{cat-ordinal } (2_{\mathbb{N}}) \rangle \langle \text{cat-ordinal } (3_{\mathbb{N}}) \rangle \langle \mathfrak{R}23 \rangle$
by (*cs-concl cs-shallow cs-intro: cat-cs-intros cat-Kan-cs-intros*)
interpret *LK23*: *is-functor* $\alpha \langle \text{cat-ordinal } (3_{\mathbb{N}}) \rangle \mathfrak{A} \langle \text{LK}23 \mathfrak{T} \rangle$
by (*cs-concl cs-intro: cat-Kan-cs-intros cat-cs-intros*)

show *?thesis*

proof(*intro is-cat-lKeI'*)

fix $\mathfrak{F}' \eta'$ **assume** *prems*:

$\mathfrak{F}' : \text{cat-ordinal } (3_{\mathbb{N}}) \mapsto \mapsto_{C\alpha} \mathfrak{A}$
 $\eta' : \mathfrak{T} \mapsto_{CF} \mathfrak{F}' \circ_{CF} \mathfrak{R}23 : \text{cat-ordinal } (2_{\mathbb{N}}) \mapsto \mapsto_{C\alpha} \mathfrak{A}$

interpret \mathfrak{F}' : *is-functor* $\alpha \langle \text{cat-ordinal } (3_{\mathbb{N}}) \rangle \mathfrak{A} \mathfrak{F}'$ **by** (*rule prems(1)*)

interpret η' : *is-ntcf* $\alpha \langle \text{cat-ordinal } (2_{\mathbb{N}}) \rangle \mathfrak{A} \mathfrak{T} \langle \mathfrak{F}' \circ_{CF} \mathfrak{R}23 \rangle \eta'$
by (*rule prems(2)*)

interpret *LK-σ23*: *is-ntcf* $\alpha \langle \text{cat-ordinal } (3_{\mathbb{N}}) \rangle \mathfrak{A} \langle \text{LK}23 \mathfrak{T} \rangle \mathfrak{F}' \langle \text{LK-σ23 } \mathfrak{T} \eta' \mathfrak{F}' \rangle$
by (*intro LK-σ23-is-ntcf prems assms*)

show $\exists ! \sigma$.

$\sigma : \text{LK}23 \mathfrak{T} \mapsto_{CF} \mathfrak{F}' : \text{cat-ordinal } (3_{\mathbb{N}}) \mapsto \mapsto_{C\alpha} \mathfrak{A} \wedge$
 $\eta' = \sigma \circ_{NTCF-CF} \mathfrak{R}23 \cdot_{NTCF} \text{ntcf-id } \mathfrak{T}$

proof(*intro ex1I conjI; (elim conjE)?*)

show *LK-σ23* $\mathfrak{T} \eta' \mathfrak{F}' : \text{LK}23 \mathfrak{T} \mapsto_{CF} \mathfrak{F}' : \text{cat-ordinal } (3_{\mathbb{N}}) \mapsto \mapsto_{C\alpha} \mathfrak{A}$
by (*intro LK-σ23.is-ntcf-axioms*)

show $\eta' = \text{LK-σ23 } \mathfrak{T} \eta' \mathfrak{F}' \circ_{NTCF-CF} \mathfrak{R}23 \cdot_{NTCF} \text{ntcf-id } \mathfrak{T}$

proof(*rule ntcf-eqI*)

show $\eta' : \mathfrak{T} \mapsto_{CF} \mathfrak{F}' \circ_{CF} \mathfrak{R}23 : \text{cat-ordinal } (2_{\mathbb{N}}) \mapsto \mapsto_{C\alpha} \mathfrak{A}$
by (*intro prems*)

then have *dom-lhs*: $\mathcal{D}_\circ (\eta' \langle \text{NTMap} \rangle) = 2_{\mathbb{N}}$

by (*cs-concl cs-shallow cs-simp: cat-ordinal-cs-simps cat-cs-simps*)

show *rhs*:

$LK\text{-}\sigma 23 \mathfrak{T} \eta' \mathfrak{F}' \circ_{NTCF\text{-}CF} \mathfrak{R}23 \cdot_{NTCF} ntcf\text{-}id \mathfrak{T} :$
 $\mathfrak{T} \mapsto_{CF} \mathfrak{F}' \circ_{CF} \mathfrak{R}23 : cat\text{-}ordinal (2_{\mathbb{N}}) \mapsto\mapsto_{C\alpha} \mathfrak{A}$
by
(

cs-concl cs-shallow
cs-simp: *cat-Kan-cs-simps cat-cs-simps*
cs-intro: *cat-Kan-cs-intros cat-cs-intros*
)

then have *dom-rhs:*
 $\mathcal{D}_o ((LK\text{-}\sigma 23 \mathfrak{T} \eta' \mathfrak{F}' \circ_{NTCF\text{-}CF} \mathfrak{R}23 \cdot_{NTCF} ntcf\text{-}id \mathfrak{T})(NTMap)) = 2_{\mathbb{N}}$
by (*cs-concl cs-simp: cat-ordinal-cs-simps cat-cs-simps*)
show $\eta'(NTMap) = (LK\text{-}\sigma 23 \mathfrak{T} \eta' \mathfrak{F}' \circ_{NTCF\text{-}CF} \mathfrak{R}23 \cdot_{NTCF} ntcf\text{-}id \mathfrak{T})(NTMap)$
proof(*rule vsu-eqI, unfold dom-lhs dom-rhs*)
fix *a* **assume** *prems:* $a \in_o 2_{\mathbb{N}}$
then consider $\langle a = 0 \rangle \mid \langle a = 1_{\mathbb{N}} \rangle$ **unfolding two by auto**
then show
 $\eta'(NTMap)(a) =$
 $(LK\text{-}\sigma 23 \mathfrak{T} \eta' \mathfrak{F}' \circ_{NTCF\text{-}CF} \mathfrak{R}23 \cdot_{NTCF} ntcf\text{-}id \mathfrak{T})(NTMap)(a)$
by (*cases; use nothing in <simp-all only>*)
(

cs-concl
cs-simp:
omega-of-set
cat-Kan-cs-simps
cat-cs-simps
cat-ordinal-cs-simps
cs-intro: *cat-Kan-cs-intros cat-cs-intros nat-omega-intros*
)+

qed
(

use rhs in
<cs-concl cs-shallow cs-intro: V-cs-intros cat-cs-intros>
)+

qed *simp-all*

fix σ **assume** *prems':*
 $\sigma : LK23 \mathfrak{T} \mapsto_{CF} \mathfrak{F}' : cat\text{-}ordinal (3_{\mathbb{N}}) \mapsto\mapsto_{C\alpha} \mathfrak{A}$
 $\eta' = \sigma \circ_{NTCF\text{-}CF} \mathfrak{R}23 \cdot_{NTCF} ntcf\text{-}id \mathfrak{T}$

interpret σ : *is-ntcf* α $\langle cat\text{-}ordinal (3_{\mathbb{N}}) \rangle \mathfrak{A} \langle LK23 \mathfrak{T} \rangle \mathfrak{F}' \sigma$
by (*rule prems'(1)*)

from *prems'(2)* **have**
 $\eta'(NTMap)(0) = (\sigma \circ_{NTCF\text{-}CF} \mathfrak{R}23 \cdot_{NTCF} ntcf\text{-}id \mathfrak{T})(NTMap)(0)$
by *auto*
then have [*cat-cs-simps*]: $\eta'(NTMap)(0) = \sigma(NTMap)(0)$
by
(

cs-prems cs-shallow
cs-simp: *cat-Kan-cs-simps cat-cs-simps cat-ordinal-cs-simps*
cs-intro: *cat-cs-intros nat-omega-intros*
)

from *prems'(2)* **have**
 $\eta'(NTMap)(1_{\mathbb{N}}) = (\sigma \circ_{NTCF\text{-}CF} \mathfrak{R}23 \cdot_{NTCF} ntcf\text{-}id \mathfrak{T})(NTMap)(1_{\mathbb{N}})$
by *auto*
then have [*cat-cs-simps*]: $\eta'(NTMap)(1_{\mathbb{N}}) = \sigma(NTMap)(1_{\mathbb{N}})$
by
(

cs-prems cs-shallow
cs-simp:
omega-of-set cat-Kan-cs-simps cat-cs-simps cat-ordinal-cs-simps
cs-intro: *cat-cs-intros nat-omega-intros*
)

show $\sigma = LK\text{-}\sigma 23 \mathfrak{T} \eta' \mathfrak{F}'$
proof(*rule ntcf-eqI*)

show $\sigma : LK23 \mathfrak{T} \mapsto_{CF} \mathfrak{F}' : \text{cat-ordinal } (3_{\mathbb{N}}) \mapsto_{C\alpha} \mathfrak{A}$
by (*rule prems'(1)*)
then have *dom-lhs*: $\mathcal{D}_\circ(\sigma(\text{NTMap})) = 3_{\mathbb{N}}$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps cat-ordinal-cs-simps*)
show $LK\text{-}\sigma 23 \mathfrak{T} \eta' \mathfrak{F}' : LK23 \mathfrak{T} \mapsto_{CF} \mathfrak{F}' : \text{cat-ordinal } (3_{\mathbb{N}}) \mapsto_{C\alpha} \mathfrak{A}$
by (*cs-concl cs-intro: cat-cs-intros cat-Kan-cs-intros*)
then have *dom-rhs*: $\mathcal{D}_\circ(LK\text{-}\sigma 23 \mathfrak{T} \eta' \mathfrak{F}'(\text{NTMap})) = 3_{\mathbb{N}}$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps cat-ordinal-cs-simps*)
from *0123* **have** *012*: $[0, 1_{\mathbb{N}}]_\circ : 0 \mapsto_{\text{cat-ordinal } (2_{\mathbb{N}})} 1_{\mathbb{N}}$
by (*cs-concl cs-intro: cat-ordinal-cs-intros nat-omega-intros*)
from *0123* **have** *013*: $[0, 1_{\mathbb{N}}]_\circ : 0 \mapsto_{\text{cat-ordinal } (3_{\mathbb{N}})} 1_{\mathbb{N}}$
by (*cs-concl cs-intro: cat-ordinal-cs-intros nat-omega-intros*)
from *0123* **have** *00*: $[0, 0]_\circ = (\text{cat-ordinal } (2_{\mathbb{N}}))(\text{CIId})(\emptyset)$
by (*cs-concl cs-shallow cs-simp: cat-ordinal-cs-simps*)
from $\sigma.\text{ntcf-Comp-commute}[OF\ 013]\ 013\ 0123$
have [*symmetric, cat-Kan-cs-simps*]:
 $\sigma(\text{NTMap})(\emptyset 1_{\mathbb{N}}) = \mathfrak{F}'(\text{ArrMap})(\emptyset, 1_{\mathbb{N}}) \bullet_{A\mathfrak{A}} \sigma(\text{NTMap})(\emptyset)$
by
(

cs-prems
cs-simp: *cat-cs-simps cat-Kan-cs-simps 00 LK23-ArrMap-app-01*
cs-intro: *cat-cs-intros cat-ordinal-cs-intros nat-omega-intros*
)

show $\sigma(\text{NTMap}) = LK\text{-}\sigma 23 \mathfrak{T} \eta' \mathfrak{F}'(\text{NTMap})$
proof(*rule vsv-eqI, unfold dom-lhs dom-rhs*)
fix *a* **assume** *prems*: $a \in_\circ 3_{\mathbb{N}}$
then consider $\langle a = 0 \rangle \mid \langle a = 1_{\mathbb{N}} \rangle \mid \langle a = 2_{\mathbb{N}} \rangle$ **unfolding** *three* **by** *auto*
then show $\sigma(\text{NTMap})(\emptyset a) = LK\text{-}\sigma 23 \mathfrak{T} \eta' \mathfrak{F}'(\text{NTMap})(\emptyset a)$
by (*cases; use nothing in <simp-all only>*)
(

cs-concl
cs-simp: *cat-ordinal-cs-simps cat-cs-simps cat-Kan-cs-simps*
cs-intro: *cat-cs-intros*
)+

qed *auto*
qed *simp-all*

qed

qed (*cs-concl cs-shallow cs-simp: cat-Kan-cs-simps cs-intro: cat-cs-intros*)+

qed

16.9 Pointwise Kan extensions along $\mathfrak{K}23$

lemma $\varepsilon 23\text{-is-cat-pw-rKe}$:

assumes $\mathfrak{T} : \text{cat-ordinal } (2_{\mathbb{N}}) \mapsto_{C\alpha} \mathfrak{A}$
shows *ntcf-id* \mathfrak{T} :

$RK23 \mathfrak{T} \circ_{CF} \mathfrak{R}23 \mapsto_{CF.\tau Ke.pw\alpha} \mathfrak{T} :$
 $cat\text{-ordinal} (2_{\mathbb{N}}) \mapsto_C cat\text{-ordinal} (3_{\mathbb{N}}) \mapsto_C \mathfrak{A}$

proof-

interpret \mathfrak{T} : *is-functor* $\alpha \langle cat\text{-ordinal} (2_{\mathbb{N}}) \rangle \mathfrak{A} \mathfrak{T}$ **by** (*rule assms(1)*)

show *?thesis*

proof(*intro is-cat-pw-rKeI* $\varepsilon23$ -*is-cat-rKe*[*OF assms*])

fix a **assume** *prems*: $a \in_{\circ} \mathfrak{A}(\text{Obj})$

show

ntcf-id $\mathfrak{T} :$

$RK23 \mathfrak{T} \circ_{CF} \mathfrak{R}23 \mapsto_{CF.\tau Ke\alpha} \mathfrak{T} :$

$cat\text{-ordinal} (2_{\mathbb{N}}) \mapsto_C$

$cat\text{-ordinal} (3_{\mathbb{N}}) \mapsto_C$

$(Hom_{O.C\alpha}\mathfrak{A}(a,-) : \mathfrak{A} \mapsto_{\mapsto_C} cat\text{-Set } \alpha)$

proof(*intro is-cat-rKe-preservesI* $\varepsilon23$ -*is-cat-rKe*[*OF assms*])

from *prems* **show** $Hom_{O.C\alpha}\mathfrak{A}(a,-) : \mathfrak{A} \mapsto_{\mapsto_{C\alpha}} cat\text{-Set } \alpha$

by (*cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros*)

show $Hom_{O.C\alpha}\mathfrak{A}(a,-) \circ_{CF-NTCF} ntcf\text{-id } \mathfrak{T} :$

$(Hom_{O.C\alpha}\mathfrak{A}(a,-) \circ_{CF} RK23 \mathfrak{T}) \circ_{CF} \mathfrak{R}23 \mapsto_{CF.\tau Ke\alpha} Hom_{O.C\alpha}\mathfrak{A}(a,-) \circ_{CF} \mathfrak{T} :$

$cat\text{-ordinal} (2_{\mathbb{N}}) \mapsto_C cat\text{-ordinal} (3_{\mathbb{N}}) \mapsto_C cat\text{-Set } \alpha$

proof(*intro is-cat-rKeI'*)

show $\mathfrak{R}23 : cat\text{-ordinal} (2_{\mathbb{N}}) \mapsto_{\mapsto_{C\alpha}} cat\text{-ordinal} (3_{\mathbb{N}})$

by (*cs-concl cs-shallow cs-intro: cat-Kan-cs-intros*)

from *prems* **show**

$Hom_{O.C\alpha}\mathfrak{A}(a,-) \circ_{CF} RK23 \mathfrak{T} : cat\text{-ordinal} (3_{\mathbb{N}}) \mapsto_{\mapsto_{C\alpha}} cat\text{-Set } \alpha$

by (*cs-concl cs-intro: cat-cs-intros cat-Kan-cs-intros*)

from *prems* **show**

$Hom_{O.C\alpha}\mathfrak{A}(a,-) \circ_{CF-NTCF} ntcf\text{-id } \mathfrak{T} :$

$Hom_{O.C\alpha}\mathfrak{A}(a,-) \circ_{CF} RK23 \mathfrak{T} \circ_{CF} \mathfrak{R}23 \mapsto_{CF} Hom_{O.C\alpha}\mathfrak{A}(a,-) \circ_{CF} \mathfrak{T} :$

$cat\text{-ordinal} (2_{\mathbb{N}}) \mapsto_{\mapsto_{C\alpha}} cat\text{-Set } \alpha$

by

(

cs-concl

cs-simp: *cat-cs-simps cat-Kan-cs-simps*

cs-intro: *cat-cs-intros cat-Kan-cs-intros*

)

fix $\mathfrak{G}' \varepsilon'$ **assume** *prems'*:

$\mathfrak{G}' : cat\text{-ordinal} (3_{\mathbb{N}}) \mapsto_{\mapsto_{C\alpha}} cat\text{-Set } \alpha$

$\varepsilon' :$

$\mathfrak{G}' \circ_{CF} \mathfrak{R}23 \mapsto_{CF} Hom_{O.C\alpha}\mathfrak{A}(a,-) \circ_{CF} \mathfrak{T} :$

$cat\text{-ordinal} (2_{\mathbb{N}}) \mapsto_{\mapsto_{C\alpha}} cat\text{-Set } \alpha$

interpret \mathfrak{G}' : *is-functor* $\alpha \langle cat\text{-ordinal} (3_{\mathbb{N}}) \rangle \langle cat\text{-Set } \alpha \rangle \mathfrak{G}'$

by (*rule prems'(1)*)

interpret ε' : *is-ntcf*

α

$\langle cat\text{-ordinal} (2_{\mathbb{N}}) \rangle$

$\langle cat\text{-Set } \alpha \rangle$

$\langle \mathfrak{G}' \circ_{CF} \mathfrak{R}23 \rangle$

$\langle Hom_{O.C\alpha}\mathfrak{A}(a,-) \circ_{CF} \mathfrak{T} \rangle$

ε'

by (*rule prems'(2)*)

show $\exists! \sigma$.

$\sigma :$
 $\mathfrak{G}' \mapsto_{CF} Hom_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} RK23 \mathfrak{T} :$
cat-ordinal $(\mathfrak{3}_N) \mapsto_{CF} cat-Set \alpha \wedge$
 $\varepsilon' = Hom_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF-NTCF} ntcf-id \mathfrak{T} \cdot_{NTCF} (\sigma \circ_{NTCF-CF} \mathfrak{R}23)$
proof(*intro ex1I conjI*; (*elim conjE*)?)
have [*cat-Kan-cs-simps*]:
 $Hom_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} RK23 \mathfrak{T} = RK23 (Hom_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} \mathfrak{T})$
proof(*rule cf-eqI*)
from *prems* **show** *lhs*: $Hom_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} RK23 \mathfrak{T} :$
cat-ordinal $(\mathfrak{3}_N) \mapsto_{CF} cat-Set \alpha$
by
(

cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-cs-intros cat-Kan-cs-intros*
)

from *prems* **show** *rhs*: $RK23 (Hom_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} \mathfrak{T}) :$
cat-ordinal $(\mathfrak{3}_N) \mapsto_{CF} cat-Set \alpha$
by
(

cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-cs-intros cat-Kan-cs-intros*
)

from *lhs prems* **have** *ObjMap-dom-lhs*:
 $\mathcal{D}_o ((Hom_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} RK23 \mathfrak{T})(ObjMap)) = \mathfrak{3}_N$
by
(

cs-concl
cs-simp: *cat-ordinal-cs-simps cat-cs-simps*
cs-intro: *cat-Kan-cs-intros cat-cs-intros*
)

from *rhs prems* **have** *ObjMap-dom-rhs*:
 $\mathcal{D}_o ((RK23 (Hom_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} \mathfrak{T}))(ObjMap)) = \mathfrak{3}_N$
by
(

cs-concl cs-shallow
cs-simp: *cat-ordinal-cs-simps cat-cs-simps*
cs-intro: *cat-Kan-cs-intros*
)

show
 $(Hom_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} RK23 \mathfrak{T})(ObjMap) =$
 $RK23 (Hom_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} \mathfrak{T})(ObjMap)$
proof(*rule vsu-eqI, unfold ObjMap-dom-lhs ObjMap-dom-rhs*)
fix *c* **assume** *prems''*: $c \in_o \mathfrak{3}_N$
with *0123* **consider** $\langle c = 0 \rangle \mid \langle c = 1_N \rangle \mid \langle c = 2_N \rangle$ **by force**
from *this prems prems'' 0123* **show**
 $(Hom_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} RK23 \mathfrak{T})(ObjMap)(c) =$
 $RK23 (Hom_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} \mathfrak{T})(ObjMap)(c)$
by (*cases; use nothing in* $\langle simp-all \text{ only} \rangle$)
(

cs-concl
cs-simp:
cat-ordinal-cs-simps
cat-cs-simps
cat-op-simps
cat-Kan-cs-simps
cs-intro: *cat-Kan-cs-intros cat-cs-intros*
)

)+
qed
 $($
use prems in \langle
cs-concl cs-intro: cat-Kan-cs-intros cat-cs-intros
 \rangle
)+
from *lhs prems have ArrMap-dom-lhs:*
 $\mathcal{D}_\circ ((Hom_{O.C\alpha}\mathfrak{A}(a,-) \circ_{CF} RK23 \mathfrak{T})(\downarrow ArrMap)) =$
cat-ordinal $(\mathfrak{Z}_N)(\downarrow Arr)$
by
 $($
cs-concl
cs-simp: *cat-ordinal-cs-simps cat-cs-simps*
cs-intro: *cat-Kan-cs-intros cat-cs-intros*
)
from *rhs prems have ArrMap-dom-rhs:*
 $\mathcal{D}_\circ ((RK23 (Hom_{O.C\alpha}\mathfrak{A}(a,-) \circ_{CF} \mathfrak{T})(\downarrow ArrMap)) =$
cat-ordinal $(\mathfrak{Z}_N)(\downarrow Arr)$
by
 $($
cs-concl cs-shallow
cs-simp: *cat-ordinal-cs-simps cat-cs-simps*
cs-intro: *cat-Kan-cs-intros*
)
show
 $(Hom_{O.C\alpha}\mathfrak{A}(a,-) \circ_{CF} RK23 \mathfrak{T})(\downarrow ArrMap) =$
 $RK23 (Hom_{O.C\alpha}\mathfrak{A}(a,-) \circ_{CF} \mathfrak{T})(\downarrow ArrMap)$
proof(*rule vsv-eqI, unfold ArrMap-dom-lhs ArrMap-dom-rhs*)
fix *f assume prems'': f* \in_\circ *cat-ordinal* $(\mathfrak{Z}_N)(\downarrow Arr)$
then obtain *a' b' where f : a' \mapsto cat-ordinal* (\mathfrak{Z}_N) *b' by auto*
from this 0123 prems show
 $(Hom_{O.C\alpha}\mathfrak{A}(a,-) \circ_{CF} RK23 \mathfrak{T})(\downarrow ArrMap)(\downarrow f) =$
 $RK23 (Hom_{O.C\alpha}\mathfrak{A}(a,-) \circ_{CF} \mathfrak{T})(\downarrow ArrMap)(\downarrow f)$
by
 $($
elim cat-ordinal-3-is-arrE;
use nothing in \langle *simp-all only;* \rangle
)
 $($
cs-concl
cs-simp: *cat-cs-simps cat-Kan-cs-simps cat-op-simps*
cs-intro:
cat-ordinal-cs-intros
cat-Kan-cs-intros
cat-cs-intros
nat-omega-intros
)+
qed
 $($
use prems in
 \langle *cs-concl cs-intro: cat-Kan-cs-intros cat-cs-intros* \rangle
)+
qed *simp-all*

show $RK\text{-}\sigma 23 (Hom_{O.C\alpha}\mathfrak{A}(a,-) \circ_{CF} \mathfrak{T}) \varepsilon' \mathfrak{G}' :$
 $\mathfrak{G}' \mapsto_{CF} Hom_{O.C\alpha}\mathfrak{A}(a,-) \circ_{CF} RK23 \mathfrak{T} :$
cat-ordinal $(\mathfrak{Z}_N) \mapsto_{C\alpha}$ *cat-Set* α

by (intro RK- $\sigma 23$ -is-ntcf')
 (
 cs-concl **cs-shallow**
 cs-simp: cat-Kan-cs-simps **cs-intro:** cat-cs-intros
)+
 show $\varepsilon' =$
 $Hom_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF-NTCF}$
 $ntcf-id \mathfrak{T} \cdot_{NTCF}$
 $(RK-\sigma 23 (Hom_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} \mathfrak{T}) \varepsilon' \mathfrak{G}' \circ_{NTCF-CF} \mathfrak{R}23)$
 proof(rule ntcf-eqI)
 show ε' :
 $\mathfrak{G}' \circ_{CF} \mathfrak{R}23 \mapsto_{CF} Hom_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} \mathfrak{T}$:
 cat-ordinal ($2_{\mathbb{N}}$) $\mapsto_{C\alpha}$ cat-Set α
 by (intro prems')
 then have dom-lhs: $\mathcal{D}_o (\varepsilon'(\downarrow NTMap)) = 2_{\mathbb{N}}$
 by (cs-concl **cs-shallow cs-simp:** cat-ordinal-cs-simps cat-cs-simps)
 from prems show
 $Hom_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF-NTCF}$
 $ntcf-id \mathfrak{T} \cdot_{NTCF}$
 $(RK-\sigma 23 (Hom_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} \mathfrak{T}) \varepsilon' \mathfrak{G}' \circ_{NTCF-CF} \mathfrak{R}23)$:
 $\mathfrak{G}' \circ_{CF} \mathfrak{R}23 \mapsto_{CF} Hom_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} \mathfrak{T}$:
 cat-ordinal ($2_{\mathbb{N}}$) $\mapsto_{C\alpha}$ cat-Set α
 by
 (
 cs-concl
 cs-simp: cat-Kan-cs-simps
 cs-intro: cat-Kan-cs-intros cat-cs-intros
)
 then have dom-rhs:
 \mathcal{D}_o
 (
 $(Hom_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF-NTCF}$
 $ntcf-id \mathfrak{T} \cdot_{NTCF}$
 $(RK-\sigma 23 (Hom_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} \mathfrak{T}) \varepsilon' \mathfrak{G}' \circ_{NTCF-CF} \mathfrak{R}23)$
) $(\downarrow NTMap)$ = $2_{\mathbb{N}}$
 by (cs-concl **cs-simp:** cat-ordinal-cs-simps cat-cs-simps)
 show $\varepsilon'(\downarrow NTMap) =$
 (
 $Hom_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF-NTCF}$
 $ntcf-id \mathfrak{T} \cdot_{NTCF}$
 $(RK-\sigma 23 (Hom_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} \mathfrak{T}) \varepsilon' \mathfrak{G}' \circ_{NTCF-CF} \mathfrak{R}23)$
) $(\downarrow NTMap)$
 proof(rule vsu-eqI, unfold dom-lhs dom-rhs)
 fix c assume prems'': $c \in_o 2_{\mathbb{N}}$
 then consider $\langle c = 0 \rangle \mid \langle c = 1_{\mathbb{N}} \rangle$ unfolding two by auto
 from this prems 0123 show $\varepsilon'(\downarrow NTMap)(\downarrow c) =$
 (
 $Hom_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF-NTCF}$
 $ntcf-id \mathfrak{T} \cdot_{NTCF}$
 $(RK-\sigma 23 (Hom_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} \mathfrak{T}) \varepsilon' \mathfrak{G}' \circ_{NTCF-CF} \mathfrak{R}23)$
) $(\downarrow NTMap)(\downarrow c)$
 by (cases; use nothing in $\langle simp-all \ only \rangle$)
 (
 cs-concl
 cs-simp:
 cat-Kan-cs-simps
 cat-ordinal-cs-simps
 cat-cs-simps

```

    cat-op-simps
    RK-σ23-NTMap-app-0
    cat-Set-components(1)
  cs-intro:
    cat-Kan-cs-intros
    cat-cs-intros
    cat-prod-cs-intros
    ℑ.HomCod.cat-Hom-in-Vset
  )+
qed (cs-concl cs-intro: cat-cs-intros V-cs-intros)+

qed simp-all

fix σ assume prems'':
  σ :
    ℑ' ↦CF HomO.Cαℒ(a,-) ∘CF RK23 ℑ :
    cat-ordinal (3N) ↦CF cat-Set α
  ε' = HomO.Cαℒ(a,-) ∘CF-NTCF ntcf-id ℑ •NTCF (σ ∘NTCF-CF ℔23)

interpret σ: is-ntcf
  α ⟨cat-ordinal (3N)⟩ ⟨cat-Set α⟩ ℑ' ⟨HomO.Cαℒ(a,-) ∘CF RK23 ℑ⟩ σ
  by (rule prems''(1))

from prems''(2) have ε'(|NTMap|)(|0|) =
  (HomO.Cαℒ(a,-) ∘CF-NTCF ntcf-id ℑ •NTCF (σ ∘NTCF-CF ℔23))(|NTMap|)(|0|)
  by auto
from this prems 0123 have ε'-NTMap-app-0: ε'(|NTMap|)(|0|) = σ(|NTMap|)(|0|)
  by
  (
    cs-prems
    cs-simp:
      cat-ordinal-cs-simps
      cat-cs-simps
      cat-Kan-cs-simps
      cat-op-simps
      ℔23-ObjMap-app-0
      cat-Set-components(1)
    cs-intro:
      cat-Kan-cs-intros
      cat-cs-intros
      cat-prod-cs-intros
      ℑ.HomCod.cat-Hom-in-Vset
  )
from 0123 have 01: [0, 1N]o : 0 ↦cat-ordinal (2N) 1N
  by
  (
    cs-concl
    cs-simp: cat-cs-simps
    cs-intro: cat-ordinal-cs-intros nat-omega-intros
  )
from prems''(2) have
  ε'(|NTMap|)(|1N|) =
  (
    HomO.Cαℒ(a,-) ∘CF-NTCF ntcf-id ℑ •NTCF (σ ∘NTCF-CF ℔23)
  )(|NTMap|)(|1N|)
  by auto
from this prems 0123 have ε'-NTMap-app-1:
  ε'(|NTMap|)(|1N|) = σ(|NTMap|)(|2N|)

```

```

by
  (
    cs-prems
    cs-simp:
      cat-ordinal-cs-simps
      cat-cs-simps
      cat-Kan-cs-simps
      cat-op-simps
       $\mathfrak{K}23$ -ObjMap-app-1
      cat-Set-components(1)
    cs-intro:
      cat-Kan-cs-intros
      cat-cs-intros
      cat-prod-cs-intros
       $\mathfrak{T}.HomCod.cat-Hom-in-Vset$ 
  )

from 0123 have 012:  $[0, 1_{\mathbb{N}}]_{\circ} : 0 \mapsto_{cat-ordinal} (2_{\mathbb{N}}) 1_{\mathbb{N}}$ 
by
  (
    cs-concl cs-intro:
      cat-ordinal-cs-intros nat-omega-intros
  )
from 0123 have 013:  $[0, 1_{\mathbb{N}}]_{\circ} : 0 \mapsto_{cat-ordinal} (3_{\mathbb{N}}) 1_{\mathbb{N}}$ 
by
  (
    cs-concl cs-intro:
      cat-ordinal-cs-intros nat-omega-intros
  )
from 0123 have 123:  $[1_{\mathbb{N}}, 2_{\mathbb{N}}]_{\circ} : 1_{\mathbb{N}} \mapsto_{cat-ordinal} (3_{\mathbb{N}}) 2_{\mathbb{N}}$ 
by
  (
    cs-concl cs-intro:
      cat-ordinal-cs-intros nat-omega-intros
  )
from 0123 have 11:  $[1_{\mathbb{N}}, 1_{\mathbb{N}}]_{\circ} = (cat-ordinal (2_{\mathbb{N}}))(CIId)(1_{\mathbb{N}})$ 
by (cs-concl cs-shallow cs-simp: cat-ordinal-cs-simps)

from  $\sigma.ntcf-Comp-commute[OF 123]$  prems 012 013
have [cat-Kan-cs-simps]:
   $\varepsilon'(\mathcal{N}TMap)(1_{\mathbb{N}}) \circ_{Acat-Set} \alpha \mathfrak{G}'(ArrMap)(1_{\mathbb{N}}, 2_{\mathbb{N}})_{\bullet} = \sigma(\mathcal{N}TMap)(1_{\mathbb{N}})$ 
by
  (
    cs-prems
    cs-simp:
      cat-cs-simps
      cat-Kan-cs-simps
       $\varepsilon'$ -NTMap-app-1[symmetric]
      is-functor.cf-ObjMap-CId
      RK23-ArrMap-app-12
      11
    cs-intro: cat-cs-intros nat-omega-intros
  )

show  $\sigma = RK\text{-}\sigma 23 (Hom_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} \mathfrak{T}) \varepsilon' \mathfrak{G}'$ 
proof(rule ntcf-eq1)

show  $\sigma : \sigma :$ 

```

$\mathfrak{G}' \mapsto_{CF} \text{Hom}_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} RK23 \mathfrak{T} :$
cat-ordinal $(\mathfrak{3}_N) \mapsto_{C\alpha} \text{cat-Set } \alpha$
by (*rule prems''(1)*)
then have *dom-lhs*: $\mathcal{D}_o(\sigma(\text{NTMap})) = \mathfrak{3}_N$
by (*cs-concl cs-shallow cs-simp: cat-ordinal-cs-simps cat-cs-simps*)
show $RK\text{-}\sigma 23 (\text{Hom}_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} \mathfrak{T}) \varepsilon' \mathfrak{G}' :$
 $\mathfrak{G}' \mapsto_{CF} \text{Hom}_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} RK23 \mathfrak{T} :$
cat-ordinal $(\mathfrak{3}_N) \mapsto_{C\alpha} \text{cat-Set } \alpha$
by
(

cs-concl cs-shallow
cs-simp: *cat-Kan-cs-simps*
cs-intro: *cat-Kan-cs-intros cat-cs-intros*
)

then have *dom-rhs*:
 $\mathcal{D}_o(RK\text{-}\sigma 23 (\text{Hom}_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} \mathfrak{T}) \varepsilon' \mathfrak{G}'(\text{NTMap})) = \mathfrak{3}_N$
by (*cs-concl cs-shallow cs-simp: cat-ordinal-cs-simps cat-cs-simps*)
show $\sigma(\text{NTMap}) = RK\text{-}\sigma 23 (\text{Hom}_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} \mathfrak{T}) \varepsilon' \mathfrak{G}'(\text{NTMap})$
proof(*rule vsv-eqI, unfold dom-lhs dom-rhs*)
fix c **assume** $c \in_o \mathfrak{3}_N$
then consider $\langle c = 0 \rangle \mid \langle c = 1_N \rangle \mid \langle c = 2_N \rangle$
unfolding three by auto
from this 0123 show
 $\sigma(\text{NTMap})(c) = RK\text{-}\sigma 23 (\text{Hom}_{O.C\alpha} \mathfrak{A}(a, -) \circ_{CF} \mathfrak{T}) \varepsilon' \mathfrak{G}'(\text{NTMap})(c)$
by (*cases; use nothing in <simp-all only>*)
(

cs-concl cs-simp:
cat-Kan-cs-simps $\varepsilon'\text{-NTMap-app-1}$ $\varepsilon'\text{-NTMap-app-0}$
)+

qed (*cs-concl cs-intro: cat-Kan-cs-intros V-cs-intros*)+

qed *simp-all*

qed

qed

qed

qed

qed

lemma $\eta 23\text{-is-cat-pw-lKe}$:

assumes $\mathfrak{T} : \text{cat-ordinal } (\mathfrak{2}_N) \mapsto_{C\alpha} \mathfrak{A}$

shows *ntcf-id* $\mathfrak{T} :$

$\mathfrak{T} \mapsto_{CF.lKe.pw\alpha} LK23 \mathfrak{T} \circ_{CF} \mathfrak{K}23 :$

cat-ordinal $(\mathfrak{2}_N) \mapsto_C \text{cat-ordinal } (\mathfrak{3}_N) \mapsto_C \mathfrak{A}$

proof-

interpret $\mathfrak{T} : \text{is-functor } \alpha \langle \text{cat-ordinal } (\mathfrak{2}_N) \rangle \mathfrak{A} \mathfrak{T}$ **by** (*rule assms(1)*)

from *ord-of-nat- ω* **interpret** *cat-ordinal-3: finite-category* $\alpha \langle \text{cat-ordinal } (\mathfrak{3}_N) \rangle$

by (*cs-concl cs-shallow cs-intro: cat-ordinal-cs-intros*)

from *0123* **have** *002*: $[0, 0]_o : 0 \mapsto_{\text{cat-ordinal } (\mathfrak{2}_N)} 0$

by
(

cs-concl cs-shallow
cs-simp: *cat-ordinal-cs-simps* **cs-intro:** *cat-cs-intros*
)

show *?thesis*

proof(*intro is-cat-pw-lKeI η23-is-cat-rKe assms, unfold cat-op-simps*)

fix *a* **assume** *prems: a ∈_o A(Obj)*

show

op-ntcf (ntcf-id T) :

op-cf (LK23 T) ∘_{CF} op-cf R23 ↦_{CF.rKeα} op-cf T :

op-cat (cat-ordinal (2_N)) ↦_C op-cat (cat-ordinal (3_N)) ↦_C

(Hom_{O.Cα}A(-,a) : op-cat A ↦_C cat-Set α)

proof(*intro is-cat-rKe-preservesI*)

show

op-ntcf (ntcf-id T) :

op-cf (LK23 T) ∘_{CF} op-cf R23 ↦_{CF.rKeα} op-cf T :

op-cat (cat-ordinal (2_N)) ↦_C op-cat (cat-ordinal (3_N)) ↦_C op-cat A

proof(*cs-intro-step cat-op-intros*)

show *ntcf-id T :*

T ↦_{CF.lKeα} LK23 T ∘_{CF} R23 :

cat-ordinal (2_N) ↦_C cat-ordinal (3_N) ↦_C A

by (*intro η23-is-cat-rKe assms*)

qed *simp-all*

from *prems* **show** *Hom_{O.Cα}A(-,a) : op-cat A ↦_{Cα} cat-Set α*

by (*cs-concl cs-shallow cs-intro: cat-cs-intros*)

have

op-cf Hom_{O.Cα}A(-,a) ∘_{CF-NTCF} ntcf-id T :

op-cf Hom_{O.Cα}A(-,a) ∘_{CF} T ↦_{CF.lKeα}

(op-cf Hom_{O.Cα}A(-,a) ∘_{CF} LK23 T) ∘_{CF} R23 :

cat-ordinal (2_N) ↦_C cat-ordinal (3_N) ↦_C op-cat (cat-Set α)

proof(*intro is-cat-lKeI'*)

show *R23 : cat-ordinal (2_N) ↦_{Cα} cat-ordinal (3_N)*

by (*cs-concl cs-shallow cs-intro: cat-Kan-cs-intros*)

from *prems* **show** *op-cf Hom_{O.Cα}A(-,a) ∘_{CF} LK23 T :*

cat-ordinal (3_N) ↦_{Cα} op-cat (cat-Set α)

by

(

cs-concl

cs-simp: *cat-cs-simps cat-op-simps*

cs-intro: *cat-Kan-cs-intros cat-cs-intros cat-op-intros*

)

from *prems* **show**

op-cf Hom_{O.Cα}A(-,a) ∘_{CF-NTCF} ntcf-id T :

op-cf Hom_{O.Cα}A(-,a) ∘_{CF} T ↦_{CF}

op-cf Hom_{O.Cα}A(-,a) ∘_{CF} LK23 T ∘_{CF} R23 :

cat-ordinal (2_N) ↦_{Cα} op-cat (cat-Set α)

by

(

cs-concl

cs-simp: *cat-cs-simps cat-Kan-cs-simps cat-op-simps*

cs-intro: *cat-Kan-cs-intros cat-cs-intros cat-op-intros*

)

fix *ℱ' η'* **assume** *prems'*:

ℱ' : cat-ordinal (3_N) ↦_{Cα} op-cat (cat-Set α)

η' :

$op\text{-}cf\ Hom_{O.C\alpha}\mathfrak{A}(-,a) \circ_{CF} \mathfrak{T} \mapsto_{CF} \mathfrak{F}' \circ_{CF} \mathfrak{K}23 :$
 $cat\text{-}ordinal\ (2_{\mathbb{N}}) \mapsto_{C\alpha} op\text{-}cat\ (cat\text{-}Set\ \alpha)$

interpret \mathfrak{F}' : *is-functor* α $\langle cat\text{-}ordinal\ (3_{\mathbb{N}}) \rangle$ $\langle op\text{-}cat\ (cat\text{-}Set\ \alpha) \rangle$ \mathfrak{F}'
by (*rule prems'(1)*)

interpret η' : *is-ntcf*

α
 $\langle cat\text{-}ordinal\ (2_{\mathbb{N}}) \rangle$
 $\langle op\text{-}cat\ (cat\text{-}Set\ \alpha) \rangle$
 $\langle op\text{-}cf\ Hom_{O.C\alpha}\mathfrak{A}(-,a) \circ_{CF} \mathfrak{T} \rangle$
 $\langle \mathfrak{F}' \circ_{CF} \mathfrak{K}23 \rangle$

η'
by (*rule prems'(2)*)

note [*unfolded cat-op-simps, cat-cs-intros*] =

$\eta'.ntcf\text{-}NTMap\text{-}is\text{-}arr'$

$\mathfrak{F}'.cf\text{-}ArrMap\text{-}is\text{-}arr'$

show

$\exists ! \sigma.$

$\sigma :$

$op\text{-}cf\ Hom_{O.C\alpha}\mathfrak{A}(-,a) \circ_{CF} LK23\ \mathfrak{T} \mapsto_{CF} \mathfrak{F}' :$
 $cat\text{-}ordinal\ (3_{\mathbb{N}}) \mapsto_{C\alpha} op\text{-}cat\ (cat\text{-}Set\ \alpha) \wedge$
 $\eta' = \sigma \circ_{NTCF-CF} \mathfrak{K}23 \cdot_{NTCF} (op\text{-}cf\ Hom_{O.C\alpha}\mathfrak{A}(-,a) \circ_{CF-NTCF} ntcf\text{-}id\ \mathfrak{T})$

proof(*intro ex1I conjI; (elim conjE) ?*)

have [*cat-Kan-cs-simps*]:

$op\text{-}cf\ Hom_{O.C\alpha}\mathfrak{A}(-,a) \circ_{CF} LK23\ \mathfrak{T} =$
 $LK23\ (op\text{-}cf\ Hom_{O.C\alpha}\mathfrak{A}(-,a) \circ_{CF} \mathfrak{T})$

proof(*rule cf-eqI*)

from *prems* **show** *lhs*: $op\text{-}cf\ Hom_{O.C\alpha}\mathfrak{A}(-,a) \circ_{CF} LK23\ \mathfrak{T} :$
 $cat\text{-}ordinal\ (3_{\mathbb{N}}) \mapsto_{C\alpha} op\text{-}cat\ (cat\text{-}Set\ \alpha)$

by

(
cs-concl
cs-simp: *cat-op-simps*
cs-intro: *cat-Kan-cs-intros cat-cs-intros cat-op-intros*
)

from *prems* **show** *rhs*: $LK23\ (op\text{-}cf\ Hom_{O.C\alpha}\mathfrak{A}(-,a) \circ_{CF} \mathfrak{T}) :$
 $cat\text{-}ordinal\ (3_{\mathbb{N}}) \mapsto_{C\alpha} op\text{-}cat\ (cat\text{-}Set\ \alpha)$

by (*cs-concl cs-intro: cat-Kan-cs-intros cat-cs-intros*)

from *lhs prems* **have** *ObjMap-dom-lhs*:

$\mathcal{D}_o\ ((op\text{-}cf\ Hom_{O.C\alpha}\mathfrak{A}(-,a) \circ_{CF} LK23\ \mathfrak{T})(\text{ObjMap})) = 3_{\mathbb{N}}$

by

(
cs-concl
cs-simp: *cat-ordinal-cs-simps cat-cs-simps cat-op-simps*
cs-intro: *cat-Kan-cs-intros cat-cs-intros*
)

from *rhs prems* **have** *ObjMap-dom-rhs*:

$\mathcal{D}_o\ (LK23\ (op\text{-}cf\ Hom_{O.C\alpha}\mathfrak{A}(-,a) \circ_{CF} \mathfrak{T})(\text{ObjMap})) = 3_{\mathbb{N}}$

by

(
cs-concl cs-shallow
cs-simp: *cat-ordinal-cs-simps cat-cs-simps*
)

show

$(op\text{-}cf\ Hom_{O.C\alpha}\mathfrak{A}(-,a) \circ_{CF} LK23\ \mathfrak{T})(\text{ObjMap}) =$
 $LK23\ (op\text{-}cf\ Hom_{O.C\alpha}\mathfrak{A}(-,a) \circ_{CF} \mathfrak{T})(\text{ObjMap})$

proof(*rule vsu-eqI, unfold ObjMap-dom-lhs ObjMap-dom-rhs*)

fix *c* **assume** *prems''*: $c \in_o 3_{\mathbb{N}}$

then consider $\langle c = 0 \rangle \mid \langle c = 1_{\mathbb{N}} \rangle \mid \langle c = 2_{\mathbb{N}} \rangle$
unfolding three by auto
from this prems 0123 show
 $(op\text{-}cf\ Hom_{O.C\alpha}\mathfrak{A}(-, a) \circ_{CF} LK23\ \mathfrak{T})(\downarrow ObjMap)(\downarrow c) =$
 $LK23\ (op\text{-}cf\ Hom_{O.C\alpha}\mathfrak{A}(-, a) \circ_{CF} \mathfrak{T})(\downarrow ObjMap)(\downarrow c)$
by (cases; use nothing in $\langle simp\text{-}all\ only \rangle$)
(

 cs-concl
 cs-simp:
 cat-ordinal-cs-simps
 cat-Kan-cs-simps
 cat-cs-simps
 cat-op-simps
 cs-intro: *cat-Kan-cs-intros cat-cs-intros cat-op-intros*
)+
qed
(

 use prems in
 \langle
 cs-concl
 cs-simp: cat-op-simps
 cs-intro: cat-Kan-cs-intros cat-cs-intros cat-op-intros
 \rangle
)+

from lhs prems have ArrMap-dom-lhs:
 $\mathcal{D}_{\circ} ((op\text{-}cf\ Hom_{O.C\alpha}\mathfrak{A}(-, a) \circ_{CF} LK23\ \mathfrak{T})(\downarrow ArrMap)) =$
cat-ordinal $(3_{\mathbb{N}})(\downarrow Arr)$
by
(

 cs-concl
 cs-simp: *cat-ordinal-cs-simps cat-cs-simps cat-op-simps*
 cs-intro: *cat-Kan-cs-intros cat-cs-intros*
)

from rhs prems have ArrMap-dom-rhs:
 $\mathcal{D}_{\circ} (LK23\ (op\text{-}cf\ Hom_{O.C\alpha}\mathfrak{A}(-, a) \circ_{CF} \mathfrak{T})(\downarrow ArrMap)) =$
cat-ordinal $(3_{\mathbb{N}})(\downarrow Arr)$
by (*cs-concl cs-shallow cs-simp: cat-cs-simps*)

show
 $(op\text{-}cf\ Hom_{O.C\alpha}\mathfrak{A}(-, a) \circ_{CF} LK23\ \mathfrak{T})(\downarrow ArrMap) =$
 $LK23\ (op\text{-}cf\ Hom_{O.C\alpha}\mathfrak{A}(-, a) \circ_{CF} \mathfrak{T})(\downarrow ArrMap)$
proof(*rule vsu-eqI, unfold ArrMap-dom-lhs ArrMap-dom-rhs*)
fix f **assume** $f \in_{\circ} \text{cat-ordinal } (3_{\mathbb{N}})(\downarrow Arr)$
then obtain $a' b'$ **where** $f: f : a' \mapsto \text{cat-ordinal } (3_{\mathbb{N}}) b'$
by auto

from f prems 0123 002 show
 $(op\text{-}cf\ Hom_{O.C\alpha}\mathfrak{A}(-, a) \circ_{CF} LK23\ \mathfrak{T})(\downarrow ArrMap)(\downarrow f) =$
 $LK23\ (op\text{-}cf\ Hom_{O.C\alpha}\mathfrak{A}(-, a) \circ_{CF} \mathfrak{T})(\downarrow ArrMap)(\downarrow f)$
by (*elim cat-ordinal-3-is-arrE, (simp-all only)?*)
(

 cs-concl
 cs-simp: *cat-cs-simps cat-Kan-cs-simps cat-op-simps*
 cs-intro:
 cat-ordinal-cs-intros
 cat-Kan-cs-intros
 cat-cs-intros
 cat-op-intros

```

      nat-omega-intros
    )+
  qed
  (
    use prems in
    <
      cs-concl
      cs-simp: cat-op-simps
      cs-intro: cat-Kan-cs-intros cat-cs-intros cat-op-intros>
    )+

  qed simp-all

  show LK-σ23 (op-cf HomO.Cαℳ(-, a) ∘CF ℑ) η' ℑ' :
    op-cf HomO.Cαℳ(-, a) ∘CF LK23 ℑ ↦CF ℑ' :
    cat-ordinal (ℑN) ↦Cα op-cat (cat-Set α)
  by
    (
      cs-concl cs-shallow
      cs-simp: cat-cs-simps cat-Kan-cs-simps cat-op-simps
      cs-intro: cat-Kan-cs-intros cat-cs-intros cat-op-intros
    )

  show η' =
    LK-σ23
    (
      op-cf HomO.Cαℳ(-, a) ∘CF ℑ) η' ℑ' ∘NTCF-CF
      ℑ23 ∙NTCF
      (op-cf HomO.Cαℳ(-, a) ∘CF-NTCF ntcf-id ℑ)
    )
  proof(rule ntcf-eqI)
  show lhs: η' :
    op-cf HomO.Cαℳ(-, a) ∘CF ℑ ↦CF ℑ' ∘CF ℑ23 :
    cat-ordinal (ℑN) ↦Cα op-cat (cat-Set α)
  by (rule prems'(2))
  from lhs have Do (η'(|NTMap|)) = cat-ordinal (ℑN)(|Obj|)
  by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  from prems show rhs:
    LK-σ23
    (
      op-cf HomO.Cαℳ(-, a) ∘CF ℑ) η' ℑ' ∘NTCF-CF
      ℑ23 ∙NTCF
      (op-cf HomO.Cαℳ(-, a) ∘CF-NTCF ntcf-id ℑ)
    ) :
    op-cf HomO.Cαℳ(-, a) ∘CF ℑ ↦CF ℑ' ∘CF ℑ23 :
    cat-ordinal (ℑN) ↦Cα op-cat (cat-Set α)
  by
    (
      cs-concl
      cs-simp: cat-Kan-cs-simps cat-op-simps
      cs-intro: cat-Kan-cs-intros cat-cs-intros cat-op-intros
    )
  from lhs have dom-lhs: Do (η'(|NTMap|)) = ℑN
  by
    (
      cs-concl cs-shallow
      cs-simp: cat-ordinal-cs-simps cat-cs-simps
    )

```


\mathfrak{F}'
 σ
by (*rule prems''(1)*)

note [*cat-Kan-cs-intros*] = $\sigma.npcf\text{-}NTMap\text{-}is\text{-}arr'[unfolding\ cat\text{-}op\text{-}simps]$

from *prems''(2)* **have**

$\eta'(\downarrow NTMap)(\downarrow 0) =$
 (
 $\sigma \circ_{NTCF\text{-}CF}$
 $\mathfrak{R}23 \cdot_{NTCF}$
 $(op\text{-}cf\ Hom_{O.C\alpha}\mathfrak{A}(-, a) \circ_{CF\text{-}NTCF}\ npcf\text{-}id\ \mathfrak{T})$
)($\downarrow NTMap$)($\downarrow 0$)

by *simp*

from *this prems 0123* **have** $\eta'\text{-}NTMap\text{-}app\text{-}0$: $\eta'(\downarrow NTMap)(\downarrow 0) = \sigma(\downarrow NTMap)(\downarrow 0)$

by

(
 cs-prems
 cs-simp:
 cat-ordinal-cs-simps
 cat-Kan-cs-simps
 cat-cs-simps
 cat-op-simps
 cat-Set-components(1)
 cs-intro:
 cat-Kan-cs-intros
 cat-cs-intros
 cat-prod-cs-intros
 cat-op-intros
 $\mathfrak{T}.HomCod.cat\text{-}Hom\text{-}in\text{-}Vset$
)

from *prems''(2)* **have**

$\eta'(\downarrow NTMap)(\downarrow 1_{\mathbb{N}}) =$
 (
 $\sigma \circ_{NTCF\text{-}CF}$
 $\mathfrak{R}23 \cdot_{NTCF}$
 $(op\text{-}cf\ Hom_{O.C\alpha}\mathfrak{A}(-, a) \circ_{CF\text{-}NTCF}\ npcf\text{-}id\ \mathfrak{T})$
)($\downarrow NTMap$)($\downarrow 1_{\mathbb{N}}$)

by *simp*

from *this prems 0123* **have** $\eta'\text{-}NTMap\text{-}app\text{-}1$: $\eta'(\downarrow NTMap)(\downarrow 1_{\mathbb{N}}) = \sigma(\downarrow NTMap)(\downarrow 2_{\mathbb{N}})$

by

(
 cs-prems
 cs-simp:
 cat-ordinal-cs-simps
 cat-Kan-cs-simps
 cat-cs-simps
 cat-op-simps
 cat-Set-components(1)
 cs-intro:
 cat-Kan-cs-intros
 cat-cs-intros
 cat-prod-cs-intros
 cat-op-intros
 $\mathfrak{T}.HomCod.cat\text{-}Hom\text{-}in\text{-}Vset$
)+

from 0123 **have** 013: $[0, 1_{\mathbb{N}}]_{\circ} : 0 \mapsto_{\text{cat-ordinal}} (3_{\mathbb{N}}) 1_{\mathbb{N}}$
by (cs-concl **cs-intro**: cat-ordinal-cs-intros nat-omega-intros)
from 0123 **have** 00: $[0, 0]_{\circ} = (\text{cat-ordinal } (2_{\mathbb{N}}))(\text{CId})(0)$
by (cs-concl **cs-shallow cs-simp**: cat-ordinal-cs-simps)

from σ .ntcf-Comp-commute[OF 013] *prems* 0123 013

have [cat-Kan-cs-simps]:

$\sigma(\text{NTMap})(1_{\mathbb{N}}) = \eta'(\text{NTMap})(0) \circ_{A \text{ cat-Set } \alpha} \mathfrak{F}'(\text{ArrMap})(0, 1_{\mathbb{N}})$.

by

(
cs-prems
cs-simp:
 cat-ordinal-cs-simps
 cat-Kan-cs-simps
 cat-cs-simps
 cat-op-simps
 LK23-ArrMap-app-01
cs-intro:
 cat-ordinal-cs-intros
 cat-Kan-cs-intros
 cat-cs-intros
 cat-prod-cs-intros
 cat-op-intros
 nat-omega-intros
cs-simp: 00 η' -NTMap-app-0[symmetric]
)

show $\sigma = \text{LK-}\sigma 23 (\text{op-cf Hom}_{O.C\alpha} \mathfrak{A}(-, a) \circ_{CF} \mathfrak{T}) \eta' \mathfrak{F}'$

proof(rule ntcf-eqI)

show lhs: σ :

$\text{op-cf Hom}_{O.C\alpha} \mathfrak{A}(-, a) \circ_{CF} \text{LK23 } \mathfrak{T} \mapsto_{CF} \mathfrak{F}'$:
 cat-ordinal $(3_{\mathbb{N}}) \mapsto_{C\alpha} \text{op-cat } (\text{cat-Set } \alpha)$

by (rule *prems''*(1))

show rhs: $\text{LK-}\sigma 23 (\text{op-cf Hom}_{O.C\alpha} \mathfrak{A}(-, a) \circ_{CF} \mathfrak{T}) \eta' \mathfrak{F}'$:

$\text{op-cf Hom}_{O.C\alpha} \mathfrak{A}(-, a) \circ_{CF} \text{LK23 } \mathfrak{T} \mapsto_{CF} \mathfrak{F}'$:
 cat-ordinal $(3_{\mathbb{N}}) \mapsto_{C\alpha} \text{op-cat } (\text{cat-Set } \alpha)$

by

(
 cs-concl **cs-shallow**
cs-simp: cat-Kan-cs-simps
cs-intro: cat-Kan-cs-intros cat-cs-intros
)

from lhs **have** dom-lhs: $\mathcal{D}_{\circ} (\sigma(\text{NTMap})) = 3_{\mathbb{N}}$

by (cs-concl **cs-shallow cs-simp**: cat-ordinal-cs-simps cat-cs-simps)

from rhs **have** dom-rhs:

$\mathcal{D}_{\circ} (\text{LK-}\sigma 23 (\text{op-cf Hom}_{O.C\alpha} \mathfrak{A}(-, a) \circ_{CF} \mathfrak{T}) \eta' \mathfrak{F}'(\text{NTMap})) = 3_{\mathbb{N}}$

by (cs-concl **cs-shallow cs-simp**: cat-ordinal-cs-simps cat-cs-simps)

show $\sigma(\text{NTMap}) = \text{LK-}\sigma 23 (\text{op-cf Hom}_{O.C\alpha} \mathfrak{A}(-, a) \circ_{CF} \mathfrak{T}) \eta' \mathfrak{F}'(\text{NTMap})$

proof(rule vsu-eqI, unfold dom-lhs dom-rhs)

fix c **assume** $c \in_{\circ} 3_{\mathbb{N}}$

then consider $\langle c = 0 \rangle \mid \langle c = 1_{\mathbb{N}} \rangle \mid \langle c = 2_{\mathbb{N}} \rangle$

unfolding three **by** auto

from this 0123 **show**

$\sigma(\text{NTMap})(c) =$

$\text{LK-}\sigma 23 (\text{op-cf Hom}_{O.C\alpha} \mathfrak{A}(-, a) \circ_{CF} \mathfrak{T}) \eta' \mathfrak{F}'(\text{NTMap})(c)$

by (cases, use nothing in $\langle \text{simp-all only} \rangle$)

(

cs-concl
cs-simp:
cat-ordinal-cs-simps
cat-cs-simps
cat-Kan-cs-simps
cat-op-simps
η'-NTMap-app-0
LK-σ23-NTMap-app-0
η'-NTMap-app-1
cs-intro:
cat-ordinal-cs-intros
cat-Kan-cs-intros
cat-cs-intros
cat-op-intros
nat-omega-intros

)+

qed (*cs-concl cs-intro: cat-Kan-cs-intros V-cs-intros*)+

qed *simp-all*

qed

qed

then have

op-ntcf (Hom_{O.Cα} ℳ(-, a) ∘_{CF-NTCF} op-ntcf (ntcf-id ℑ)) :
op-cf (Hom_{O.Cα} ℳ(-, a) ∘_{CF} op-cf ℑ) ↪_{CF.lKeα}
op-cf ((Hom_{O.Cα} ℳ(-, a) ∘_{CF} op-cf (LK23 ℑ))) ∘_{CF} op-cf (op-cf ℔23) :
op-cat (op-cat (cat-ordinal (2_N))) ↪_C
op-cat (op-cat (cat-ordinal (3_N))) ↪_C
op-cat (cat-Set α)

by

(

cs-concl
cs-simp: *cat-op-simps*
cs-intro: *cat-cs-intros cat-Kan-cs-intros cat-op-intros*

)

from *is-cat-lKe.is-cat-rKe-op[OF this] prems show*

Hom_{O.Cα} ℳ(-, a) ∘_{CF-NTCF} op-ntcf (ntcf-id ℑ) :
(Hom_{O.Cα} ℳ(-, a) ∘_{CF} op-cf (LK23 ℑ)) ∘_{CF} op-cf ℔23 ↪_{CF.rKeα}
Hom_{O.Cα} ℳ(-, a) ∘_{CF} op-cf ℑ :
op-cat (cat-ordinal (2_N)) ↪_C
op-cat (cat-ordinal (3_N)) ↪_C
cat-Set α

by

(

cs-prems
cs-simp: *cat-op-simps*
cs-intro: *cat-Kan-cs-intros cat-cs-intros cat-op-intros*

)

qed

qed

qed

References

- [1] nLab. URL <https://ncatlab.org/nlab/show/HomePage>.
- [2] Wikipedia, 2001. URL <https://www.wikipedia.org/>.
- [3] S. Awodey. *Category Theory*. Oxford University Press, Oxford, UK, 2010. ISBN 978-0-19-958736-0.
- [4] P. Bodo. *Categories and Functors*. Academic Press, New York, 1970.
- [5] F. Haftmann. Sketch-and-Explore, 2021. URL https://isabelle.in.tum.de/library/HOL/HOL-ex/Sketch_and_Explore.html.
- [6] T. W. Hungerford. *Algebra*. Springer, New York, 2003. ISBN 978-0-387-90518-1.
- [7] D. M. Kan. Adjoint Functors. *Transactions of the American Mathematical Society*, 87(2): 294–329, 1958.
- [8] M. C. Lehner. “All Concepts are Kan Extensions”: Kan Extensions as the Most Universal of the Universal Constructions. Bachelor of Arts Thesis, Harvard College, Cambridge, MA, 2014.
- [9] S. Mac Lane. *Categories for the Working Mathematician*. Number 5 in Graduate Texts in Mathematics. Springer, New York, 2 edition, 2010. ISBN 978-1-4419-3123-8.
- [10] M. Milehins. Category Theory for ZFC in HOL II: Elementary Theory of 1-Categories. *Archive of Formal Proofs*, 2021.
- [11] S. Obua. Partizan Games in Isabelle/HOLZF. In K. Barkaoui, A. Cavalcanti, and A. Cerone, editors, *ICTAC 2006*, volume 4281, pages 272–286. Springer, Berlin, 2006. ISBN 978-3-540-48815-6.
- [12] L. C. Paulson. Natural Deduction as Higher-Order Resolution. *The Journal of Logic Programming*, 3(3):237–258, 1986. doi: 10.1016/0743-1066(86)90015-4.
- [13] L. C. Paulson. Zermelo Fraenkel Set Theory in Higher-Order Logic. *Archive of Formal Proofs*, 2019.
- [14] E. Riehl. *Category Theory in Context*. Emily Riehl, 2016.