Probabilistic Noninterference

Andrei Popescu Johannes Hölzl

February 6, 2026

Abstract

We formalize a probabilistic noninterference for a multi-threaded language with uniform scheduling, where probabilistic behaviour comes from both the scheduler and the individual threads. We define notions probabilistic noninterference in two variants: resumption-based and trace-based. For the resumption-based notions, we prove compositionality w.r.t. the language constructs and establish sound type-system-like syntactic criteria.

This is a formalization of the mathematical development presented in the papers [1, 2]. It is the probabilistic variant of the Possibilistic Noninterference AFP entry.

Contents

1 Simple While Language with probabilistic choice and parallel execution **2**
- 1.1 Preliminaries . . . 2
- 1.2 Syntax . . . 11
  - 1.2.1 Operational Small-Step Semantics . . . 15
  - 1.2.2 Operations on configurations . . . 35

2 Resumption-Based Noninterference **35**
- 2.1 Preliminaries . . . 35
- 2.2 Infrastructure for partitions . . . 38
- 2.3 Basic setting for bisimilarity . . . 50
- 2.4 Discreetness . . . 51
- 2.5 Self-isomorphism . . . 52
- 2.6 Notions of bisimilarity . . . 54
- 2.7 List versions of the bisimilarities . . . 66
- 2.8 Discreetness for configurations . . . 68

3 Trace-Based Noninterference **70**
- 3.1 Preliminaries . . . 70
- 3.2 Quasi strong termination traces . . . 75

3.3 Terminating configurations 77
3.4 Final Theorems 90

4 Compositionality of Resumption-Based Noninterference 92
4.1 Compatibility and discreetness of atoms, tests and choices . . 92
4.2 Compositionality of self-isomorphism 92
4.3 Discreetness versus language constructs: 98
4.4 Strong bisimilarity versus language constructs 99
4.5 01-bisimilarity versus language constructs 119

5 Syntactic Criteria 146

6 Concrete setting 149
6.1 The secure programs from the paper's Example 3 152

1 Simple While Language with probabilistic choice and parallel execution

theory *Language-Semantics*
imports *Interface*
begin

1.1 Preliminaries

Trivia

declare *zero-le-mult-iff*[*simp*]
declare *split-mult-pos-le*[*simp*]
declare *zero-le-divide-iff*[*simp*]

lemma *in-minus-Un*[*simp*]:
assumes $i \in I$
shows $I - \{i\}$ *Un* $\{i\} = I$ **and** $\{i\}$ *Un* $(I - \{i\}) = I$
apply(*metis Un-commute assms insert-Diff-single insert-absorb insert-is-Un*)
by (*metis assms insert-Diff-single insert-absorb insert-is-Un*)

lemma *less-plus-cases*[*case-names Left Right*]:
assumes
$*$: $(i::nat) < n1 \Longrightarrow phi$ **and**
$**$: $\bigwedge i2.\ i = n1 + i2 \Longrightarrow phi$
shows *phi*
proof(*cases* $i < n1$)
case *True*
thus *?thesis* **using** $*$ **by** *simp*
next
case *False* **hence** $n1 \leq i$ **by** *simp*
then obtain *i2* **where** $i = n1 + i2$ **by** (*metis le-iff-add*)
thus *?thesis* **using** $**$ **by** *blast*

qed

lemma *less-plus-elim*[*elim*!, *consumes 1*, *case-names Left Right*]:
assumes *i*: (*i*:: *nat*) < *n1* + *n2* **and**
∗: *i* < *n1* ⟹ *phi* **and**
∗∗: ⋀ *i2*. ⟦*i2* < *n2*; *i* = *n1* + *i2*⟧ ⟹ *phi*
shows *phi*
apply(*rule less-plus-cases*[*of i n1*])
using *assms* **by** *auto*

lemma *nth-append-singl*[*simp*]:
i < *length al* ⟹ (*al* @ [*a*]) ! *i* = *al* ! *i*
by (*auto simp add*: *nth-append*)

lemma *take-append-singl*[*simp*]:
assumes *n* < *length al* **shows** *take n* (*al* @ [*a*]) = *take n al*
using *assms* **by** (*induct al rule*: *rev-induct*) *auto*

lemma *length-unique-prefix*:
al1 ≤ *al* ⟹ *al2* ≤ *al* ⟹ *length al1* = *length al2* ⟹ *al1* = *al2*
by (*metis not-equal-is-parallel parallelE prefix-same-cases less-eq-list-def*)

take:

lemma *take-length*[*simp*]:
take (*length al*) *al* = *al*
using *take-all* **by** *auto*

lemma *take-le*:
assumes *n* < *length al*
shows *take n al* @ [*al* ! *n*] ≤ *al*
by (*metis assms take-Suc-conv-app-nth take-is-prefix less-eq-list-def*)

lemma *list-less-iff-prefix*: *a* < *b* ⟷ *strict-prefix a b*
by (*metis le-less less-eq-list-def less-irrefl prefix-order.le-less prefix-order.less-irrefl*)

lemma *take-lt*:
n < *length al* ⟹ *take n al* < *al*
unfolding *list-less-iff-prefix*
using *prefix-order.le-less*[*of take n al al*]
by (*simp add*: *take-is-prefix*)

lemma *le-take*:
assumes *n1* ≤ *n2*
shows *take n1 al* ≤ *take n2 al*
using *assms* **proof**(*induct al arbitrary*: *n1 n2*)
case (*Cons a al*)
thus *?case*
by (*cases n1 n2 rule*: *nat.exhaust*[*case-product nat.exhaust*]) *auto*
qed *auto*

lemma *inj-take*:
assumes *n1* ≤ *length al* **and** *n2* ≤ *length al*
shows *take n1 al* = *take n2 al* ⟷ *n1* = *n2*
proof−
{assume *take n1 al* = *take n2 al*
hence *n1* = *n2*
using *assms* **proof**(*induct al arbitrary*: *n1 n2*)
case (*Cons a al*)
thus *?case*
by (*cases n1 n2 rule*: *nat.exhaust*[*case-product nat.exhaust*]) *auto*
qed *auto*
}
thus *?thesis* **by** *auto*
qed

lemma *lt-take*: *n1* < *n2* ⟹ *n2* ≤ *length al* ⟹ *take n1 al* < *take n2 al*
by (*metis inj-take le-less-trans le-take not-less-iff-gr-or-eq order.not-eq-order-implies-strict order.strict-implies-order*)

lsum:

definition *lsum* :: ('*a* ⇒ *nat*) ⇒ '*a list* ⇒ *nat* **where**
lsum f al ≡ *sum-list* (*map f al*)

lemma *lsum-simps*[*simp*]:
lsum f [] = *0*
lsum f (*al* @ [*a*]) = *lsum f al* + *f a*
unfolding *lsum-def* **by** *auto*

lemma *lsum-append*:
lsum f (*al* @ *bl*) = *lsum f al* + *lsum f bl*
unfolding *lsum-def* **by** *auto*

lemma *lsum-cong*[*fundef-cong*]:
assumes ⋀ *a*. *a* ∈ *set al* ⟹ *f a* = *g a*
shows *lsum f al* = *lsum g al*
using *assms* **unfolding** *lsum-def* **by** (*metis map-eq-conv*)

lemma *lsum-gt-0*[*simp*]:
assumes *al* ≠ [] **and** ⋀ *a*. *a* ∈ *set al* ⟹ *0* < *f a*
shows *0* < *lsum f al*
using *assms* **by** (*induct rule*: *rev-induct*) *auto*

lemma *lsum-mono*[*simp*]:
assumes *al* ≤ *bl*
shows *lsum f al* ≤ *lsum f bl*
proof−
obtain *cl* **where** *bl*: *bl* = *al* @ *cl* **using** *assms* **unfolding** *prefix-def less-eq-list-def*
by *blast*

thus *?thesis* **unfolding** *bl lsum-append* **by** *simp*
qed

lemma *lsum-mono2*[*simp*]:
assumes *f*: $\bigwedge$ *b*. *b* ∈ *set bl* ⟹ *f b* > *0* **and** *le*: *al* < *bl*
shows *lsum f al* < *lsum f bl*
proof−
obtain *cl* **where** *bl*: *bl* = *al* @ *cl* **and** *cl*: *cl* ≠ []
using *assms* **unfolding** *list-less-iff-prefix prefix-def strict-prefix-def* **by** *blast*
have *lsum f al* < *lsum f al* + *lsum f cl*
using *f cl* **unfolding** *bl* **by** *simp*
also have ... = *lsum f bl* **unfolding** *bl lsum-append* **by** *simp*
finally show *?thesis* **.**
qed

lemma *lsum-take*[*simp*]:
lsum f (*take n al*) ≤ *lsum f al*
by (*metis lsum-mono take-is-prefix less-eq-list-def*)

lemma *less-lsum-nchotomy*:
assumes *f*: $\bigwedge$ *a*. *a* ∈ *set al* ⟹ *0* < *f a*
and *i*: (*i*::*nat*) < *lsum f al*
shows ∃ *n j*. *n* < *length al* ∧ *j* < *f* (*al* ! *n*) ∧ *i* = *lsum f* (*take n al*) + *j*
using *assms* **proof**(*induct rule*: *rev-induct*)
case (*snoc a al*)
hence *i*: *i* < *lsum f al* + *f a* **and**
pos: *0* < *f a* $\bigwedge$*a'*. *a'* ∈ *set al* ⟹ *0* < *f a'* **by** *auto*
from *i* **show** *?case*
proof(*cases rule*: *less-plus-elim*)
case *Left*
then obtain *n j* **where** *n* < *length al* ∧ *j* < *f* (*al* ! *n*) ∧ *i* = *lsum f* (*take n al*) + *j*
using *pos snoc* **by** *auto*
thus *?thesis*
apply − **apply**(*rule exI*[*of* - *n*]) **apply**(*rule exI*[*of* - *j*]) **by** *auto*
next
case (*Right j*)
thus *?thesis*
apply − **apply**(*rule exI*[*of* - *length al*]) **apply**(*rule exI*[*of* - *j*]) **by** *simp*
qed
qed *auto*

lemma *less-lsum-unique*:
assumes $\bigwedge$ *a*. *a* ∈ *set al* ⟹ (*0*::*nat*) < *f a*
and *n1* < *length al* ∧ *j1* < *f* (*al* ! *n1*)
and *n2* < *length al* ∧ *j2* < *f* (*al* ! *n2*)
and *lsum f* (*take n1 al*) + *j1* = *lsum f* (*take n2 al*) + *j2*
shows *n1* = *n2* ∧ *j1* = *j2*
using *assms* **proof**(*induct al arbitrary*: *n1 n2 j1 j2 rule*: *rev-induct*)

```
  case (snoc a al)
  hence pos: 0 < f a   ⋀ a'. a' ∈ set al ⟹ 0 < f a'
  and n1: n1 < length al + 1 and n2: n2 < length al + 1 by auto
  from n1 show ?case
  proof(cases rule: less-plus-elim)
    case Left note n1 = Left
    hence j1: j1 < f (al ! n1) using snoc by auto
    obtain al' where al: al = (take n1 al) @ ((al ! n1) # al')
    using n1 by (metis append-take-drop-id Cons-nth-drop-Suc)
    have j1 < lsum f ((al ! n1) # al') using pos j1 unfolding lsum-def by simp
    hence lsum f (take n1 al) + j1 < lsum f (take n1 al) + lsum f ((al ! n1) #
al') by simp
     also have ... = lsum f al unfolding lsum-append[THEN sym] using al by
simp
    finally have lsum1: lsum f (take n1 al) + j1 < lsum f al .
    from n2 show ?thesis
    proof(cases rule: less-plus-elim)
      case Left note n2 = Left
      hence j2: j2 < f (al ! n2) using snoc by auto
     show ?thesis apply(rule snoc(1)) using snoc using pos n1 j1 n2 j2 by auto
    next
      case Right
      hence n2: n2 = length al by simp
      hence j2: j2 < f a using snoc by simp
      have lsum f (take n1 al) + j1 = lsum f al + j2 using n1 n2 snoc by simp
      hence False using lsum1 by auto
      thus ?thesis by simp
    qed
  next
    case Right
    hence n1: n1 = length al by simp
    hence j1: j1 < f a using snoc by simp
    from n2 show ?thesis
    proof(cases rule: less-plus-elim)
      case Left note n2 = Left
      hence j2: j2 < f (al ! n2) using snoc by auto
      obtain al' where al: al = (take n2 al) @ ((al ! n2) # al')
      using n2 by (metis append-take-drop-id Cons-nth-drop-Suc)
     have j2 < lsum f ((al ! n2) # al') using pos j2 unfolding lsum-def by simp
      hence lsum f (take n2 al) + j2 < lsum f (take n2 al) + lsum f ((al ! n2) #
al') by simp
       also have ... = lsum f al unfolding lsum-append[THEN sym] using al by
simp
      finally have lsum2: lsum f (take n2 al) + j2 < lsum f al .
      have lsum f al + j1 = lsum f (take n2 al) + j2 using n1 n2 snoc by simp
      hence False using lsum2 by auto
      thus ?thesis by simp
    next
      case Right
```

hence *n2*: *n2* = *length al* **by** *simp*
have *j1* = *j2* **using** *n1 n2 snoc* **by** *simp*
thus *?thesis* **using** *n1 n2* **by** *simp*
qed
qed
qed *auto*

definition *locate-pred* **where**
locate-pred f al (*i*::*nat*) *n-j* ≡
fst n-j < *length al* ∧
snd n-j < *f* (*al* ! (*fst n-j*)) ∧
i = *lsum f* (*take* (*fst n-j*) *al*) + (*snd n-j*)

definition *locate* **where**
locate f al i ≡ *SOME n-j*. *locate-pred f al i n-j*

definition *locate1* **where** *locate1 f al i* ≡ *fst* (*locate f al i*)
definition *locate2* **where** *locate2 f al i* ≡ *snd* (*locate f al i*)

lemma *locate-pred-ex*:
assumes ⋀ *a*. *a* ∈ *set al* ⟹ *0* < *f a*
and *i* < *lsum f al*
shows ∃ *n-j*. *locate-pred f al i n-j*
using *assms less-lsum-nchotomy* **unfolding** *locate-pred-def* **by** *force*

lemma *locate-pred-unique*:
assumes ⋀ *a*. *a* ∈ *set al* ⟹ *0* < *f a*
and *locate-pred f al i n1-j1 locate-pred f al i n2-j2*
shows *n1-j1* = *n2-j2*
using *assms less-lsum-unique* **unfolding** *locate-pred-def*
apply(*cases n1-j1*, *cases n2-j2*) **apply** *simp* **by** *blast*

lemma *locate-locate-pred*:
assumes ⋀ *a*. *a* ∈ *set al* ⟹ (*0*::*nat*) < *f a*
and *i* < *lsum f al*
shows *locate-pred f al i* (*locate f al i*)
proof−
obtain *n-j* **where** *locate-pred f al i n-j*
using *assms locate-pred-ex*[*of al f i*] **by** *auto*
thus *?thesis* **unfolding** *locate-def* **apply**(*intro someI*[*of locate-pred f al i*]) **by** *blast*
qed

lemma *locate-locate-pred-unique*:
assumes ⋀ *a*. *a* ∈ *set al* ⟹ (*0*::*nat*) < *f a*
and *locate-pred f al i n-j*
shows *n-j* = *locate f al i*
unfolding *locate-def* **apply**(*rule sym*, *rule some-equality*)
using *assms locate-locate-pred* **apply** *force*

using *assms locate-pred-unique* **by** *blast*

lemma *locate*:
assumes $\bigwedge$ *a*. $a \in set\ al \Longrightarrow 0 < f\ a$
and $i < lsum\ f\ al$
shows *locate1 f al i* $<$ *length al* $\wedge$
$locate2\ f\ al\ i < f\ (al\ !\ (locate1\ f\ al\ i)) \wedge$
$i = lsum\ f\ (take\ (locate1\ f\ al\ i)\ al) + (locate2\ f\ al\ i)$
using *assms locate-locate-pred*
unfolding *locate1-def locate2-def locate-pred-def* **by** *auto*

lemma *locate-unique*:
assumes $\bigwedge$ *a*. $a \in set\ al \Longrightarrow 0 < f\ a$
and $n < length\ al$ **and** $j < f\ (al\ !\ n)$ **and** $i = lsum\ f\ (take\ n\ al) + j$
shows $n = locate1\ f\ al\ i \wedge j = locate2\ f\ al\ i$
proof−
define *n-j* **where** *n-j* = (*n*,*j*)
have *locate-pred f al i n-j* **using** *assms* **unfolding** *n-j-def locate-pred-def* **by** *auto*
hence *n-j* = *locate f al i* **using** *assms locate-locate-pred-unique* **by** *blast*
thus *?thesis* **unfolding** *n-j-def locate1-def locate2-def* **by** (*metis fst-conv n-j-def snd-conv*)
qed

sum:

lemma *sum-2*[*simp*]:
$sum\ f\ \{..<2\} = f\ 0 + f\ (Suc\ 0)$
proof−
have $\{..<2\} = \{0, Suc\ 0\}$ **by** *auto*
thus *?thesis* **by** *force*
qed

lemma *inj-Plus*[*simp*]:
inj ((+) (*a*::*nat*))
unfolding *inj-on-def* **by** *auto*

lemma *inj-on-Plus*[*simp*]:
inj-on ((+) (*a*::*nat*)) *A*
unfolding *inj-on-def* **by** *auto*

lemma *Plus-int*[*simp*]:
fixes *a* :: *nat*
shows $(+)\ b\ `\ \{..<a\} = \{b\ ..<\ b + a\}$
proof *safe*
fix *x*::*nat* **assume** $x \in \{b..<\ b + a\}$
hence $b \leq x$ **and** *x*: $x < a + b$ **by** *auto*
then obtain *y* **where** *xb*: $x = b + y$ **by** (*metis le-iff-add*)
hence $y < a$ **using** *x* **by** *simp*
thus $x \in (+)\ b\ `\ \{..<a\}$ **using** *xb* **by** *auto*
qed *auto*

lemma *sum-minus*[*simp*]:
fixes *a* :: *nat*
shows *sum f* {*a* ..< *a* + *b*} = *sum* (%*x*. *f* (*a* + *x*)) {..< *b*}
using *sum.reindex*[*of* (+) *a* {..< *b*} *f*] **by** *auto*

lemma *sum-Un-introL*:
assumes *A1* = *B1 Un C1* **and** *x* = *x1* + *x2*
finite A1 **and**
B1 Int C1 = {} **and**
sum f1 B1 = *x1* **and** *sum f1 C1* = *x2*
shows *sum f1 A1* = *x*
by (*metis assms finite-Un sum.union-disjoint*)

lemma *sum-Un-intro*:
assumes *A1* = *B1 Un C1* **and** *A2* = *B2 Un C2* **and**
finite A1 **and** *finite A2* **and**
B1 Int C1 = {} **and** *B2 Int C2* = {} **and**
sum f1 B1 = *sum f2 B2* **and** *sum f1 C1* = *sum f2 C2*
shows *sum f1 A1* = *sum f2 A2*
by (*metis assms finite-Un sum.union-disjoint*)

lemma *sum-UN-introL*:
assumes *A1*: *A1* = (*UN n* : *N*. *B1 n*) **and** *a2*: *a2* = *sum b2 N* **and**
fin: *finite N* ⋀ *n*. *n* ∈ *N* ⟹ *finite* (*B1 n*) **and**
int: ⋀ *m n*. {*m*, *n*} ⊆ *N* ∧ *m* ≠ *n* ⟹ *B1 m* ∩ *B1 n* = {} **and**
ss: ⋀ *n*. *n* ∈ *N* ⟹ *sum f1* (*B1 n*) = *b2 n*
shows *sum f1 A1* = *a2* (**is** *?L* = *a2*)
proof−
have *?L* = *sum* (%*n*. *sum f1* (*B1 n*)) *N*
unfolding *A1* **using** *sum.UNION-disjoint*[*of N B1 f1*] *fin int* **by** *simp*
also have ... = *sum b2 N* **using** *ss fin* **by** *auto*
also have ... = *a2* **using** *a2* **by** *simp*
finally show *?thesis* **.**
qed

lemma *sum-UN-intro*:
assumes *A1*: *A1* = (*UN n* : *N*. *B1 n*) **and** *A2*: *A2* = (*UN n* : *N*. *B2 n*) **and**
fin: *finite N* ⋀ *n*. *n* ∈ *N* ⟹ *finite* (*B1 n*) ∧ *finite* (*B2 n*) **and**
int: ⋀ *m n*. {*m*, *n*} ⊆ *N* ∧ *m* ≠ *n* ⟹ *B1 m* ∩ *B1 n* = {} ⋀ *m n*. {*m*, *n*} ⊆ *N*
⟹ *B2 m* ∩ *B2 n* = {} **and**
ss: ⋀ *n*. *n* ∈ *N* ⟹ *sum f1* (*B1 n*) = *sum f2* (*B2 n*)
shows *sum f1 A1* = *sum f2 A2* (**is** *?L* = *?R*)
proof−
have *?L* = *sum* (%*n*. *sum f1* (*B1 n*)) *N*
unfolding *A1* **using** *sum.UNION-disjoint*[*of N B1 f1*] *fin int* **by** *simp*
also have ... = *sum* (%*n*. *sum f2* (*B2 n*)) *N* **using** *ss fin sum.mono-neutral-left*
by *auto*
also have ... = *?R*

unfolding *A2* **using** *sum.UNION-disjoint*[*of N B2 f2*] *fin int* **by** *simp*
finally show *?thesis* **.**
qed

lemma *sum-Minus-intro*:
fixes *f1* :: *'a1* ⇒ *real* **and** *f2* :: *'a2* ⇒ *real*
assumes *B1* = *A1* − {*a1*} **and** *B2* = *A2* − {*a2*} **and**
a1 : *A1* **and** *a2* : *A2* **and** *finite A1* **and** *finite A2*
sum f1 A1 = *sum f2 A2* **and** *f1 a1* = *f2 a2*
shows *sum f1 B1* = *sum f2 B2*
proof−
have *1*: *A1* = *B1 Un* {*a1*} **and** *2*: *A2* = *B2 Un* {*a2*}
using *assms* **by** *blast*+
from *assms* **have** *a1* ∉ *B1* **by** *simp*
with *1* ‹*finite A1*› **have** *sum f1 A1* = *sum f1 B1* + *f1 a1*
by *simp*
hence *3*: *sum f1 B1* = *sum f1 A1* − *f1 a1* **by** *simp*
from *assms* **have** *a2* ∉ *B2* **by** *simp*
with *2* ‹*finite A2* ›**have** *sum f2 A2* = *sum f2 B2* + *f2 a2*
by *simp*
hence *sum f2 B2* = *sum f2 A2* − *f2 a2* **by** *simp*
thus *?thesis* **using** *3 assms* **by** *simp*
qed

lemma *sum-singl-intro*:
assumes *b* = *f a*
and *finite A* **and** *a* ∈ *A*
and ⋀ *a'*. ⟦*a'* ∈ *A*; *a'* ≠ *a*⟧ ⟹ *f a'* = *0*
shows *sum f A* = *b*
proof−
define *B* **where** *B* = *A* − {*a*}
have *A* = *B Un* {*a*} **unfolding** *B-def* **using** *assms* **by** *blast*
moreover have *B Int* {*a*} = {} **unfolding** *B-def* **by** *blast*
ultimately have *sum f A* = *sum f B* + *sum f* {*a*}
using *assms sum.union-disjoint* **by** *blast*
moreover have ∀ *b* ∈ *B*. *f b* = *0* **using** *assms* **unfolding** *B-def* **by** *auto*
ultimately show *?thesis* **using** *assms* **by** *auto*
qed

lemma *sum-all0-intro*:
assumes *b* = *0*
and ⋀ *a*. *a* ∈ *A* ⟹ *f a* = *0*
shows *sum f A* = *b*
using *assms* **by** *simp*

lemma *sum-1*:
assumes *I*: *finite I* **and** *ss*: *sum f I* = *1* **and** *i*: *i* ∈ *I* − *I1* **and** *I1*: *I1* ⊆ *I*
and *f*: ⋀*i*. *i* ∈ *I* ⟹ (*0*::*real*) ≤ *f i*
shows *f i* ≤ *1* − *sum f I1*

proof−
have *sum f I = sum f* ({*i*} *Un* (*I* − {*i*})) **using** *i*
by (*metis DiffE insert-Diff-single insert-absorb insert-is-Un*)
also have ... = *sum f* {*i*} + *sum f* (*I* − {*i*})
apply(*rule sum.union-disjoint*) **using** *I* **by** *auto*
finally have *1* = *f i* + *sum f* (*I* − {*i*}) **unfolding** *ss*[*THEN sym*] **by** *simp*
moreover have *sum f* (*I* − {*i*}) ≥ *sum f I1*
apply(*rule sum-mono2*) **using** *assms* **by** *auto*
ultimately have *1* ≥ *f i* + *sum f I1* **by** *auto*
thus *?thesis* **by** *auto*
qed

1.2 Syntax

datatype ('*test*, '*atom*, '*choice*) *cmd* =
Done
| *Atm* '*atom*
| *Seq* ('*test*, '*atom*, '*choice*) *cmd* ('*test*, '*atom*, '*choice*) *cmd* (‹- ;; -› [*60*, *61*] *60*)
| *While* '*test* ('*test*, '*atom*, '*choice*) *cmd*
| *Ch* '*choice* ('*test*, '*atom*, '*choice*) *cmd* ('*test*, '*atom*, '*choice*) *cmd*
| *Par* ('*test*, '*atom*, '*choice*) *cmd list*
| *ParT* ('*test*, '*atom*, '*choice*) *cmd list*

fun *noWhile* **where**
noWhile Done ⟷ *True*
| *noWhile* (*Atm atm*) ⟷ *True*
| *noWhile* (*c1* ;; *c2*) ⟷ *noWhile c1* ∧ *noWhile c2*
| *noWhile* (*While tst c*) ⟷ *False*
| *noWhile* (*Ch ch c1 c2*) ⟷ *noWhile c1* ∧ *noWhile c2*
| *noWhile* (*Par cl*) ⟷ (∀ *c* ∈ *set cl*. *noWhile c*)
| *noWhile* (*ParT cl*) ⟷ (∀ *c* ∈ *set cl*. *noWhile c*)

fun *finished* **where**
finished Done ⟷ *True*
| *finished* (*Atm atm*) ⟷ *False*
| *finished* (*c1* ;; *c2*) ⟷ *False*
| *finished* (*While tst c*) ⟷ *False*
| *finished* (*Ch ch c1 c2*) ⟷ *False*
| *finished* (*Par cl*) ⟷ (∀ *c* ∈ *set cl*. *finished c*)
| *finished* (*ParT cl*) ⟷ (∀ *c* ∈ *set cl*. *finished c*)

definition *noWhileL* **where**
noWhileL cl ≡ ∀ *c* ∈ *set cl*. *noWhile c*

lemma *fin-Par-noWhileL*[*simp*]:
noWhile (*Par cl*) ⟷ *noWhileL cl*
unfolding *noWhileL-def* **by** *simp*

lemma *fin-ParT-noWhileL*[*simp*]:
noWhile (*ParT cl*) ⟷ *noWhileL cl*
unfolding *noWhileL-def* **by** *simp*

declare *noWhile.simps*(6) [*simp del*]
declare *noWhile.simps*(7) [*simp del*]

lemma *noWhileL-intro*[*intro*]:
assumes ⋀ *c*. *c* ∈ *set cl* ⟹ *noWhile c*
shows *noWhileL cl*
using *assms* **unfolding** *noWhileL-def* **by** *auto*

lemma *noWhileL-fin*[*simp*]:
assumes *noWhileL cl* **and** *c* ∈ *set cl*
shows *noWhile c*
using *assms* **unfolding** *noWhileL-def* **by** *simp*

lemma *noWhileL-update*[*simp*]:
assumes *cl*: *noWhileL cl* **and** *c'*: *noWhile c'*
shows *noWhileL* (*cl*[*n* := *c'*])
proof(*cases n* < *length cl*)
case *True*
show *?thesis*
unfolding *noWhileL-def* **proof** *safe*
fix *c* **assume** *c* ∈ *set* (*cl*[*n* := *c'*])
hence *c* ∈ *insert c'* (*set cl*) **using** *set-update-subset-insert* **by** *fastforce*
thus *noWhile c* **using** *assms* **by** (*cases c* = *c'*) *auto*
qed
qed (*use cl* **in** ‹*auto simp*: *list-update-beyond*›)

definition *finishedL* **where**
finishedL cl ≡ ∀ *c* ∈ *set cl*. *finished c*

lemma *finished-Par-finishedL*[*simp*]:
finished (*Par cl*) ⟷ *finishedL cl*
unfolding *finishedL-def* **by** *simp*

lemma *finished-ParT-finishedL*[*simp*]:
finished (*ParT cl*) ⟷ *finishedL cl*
unfolding *finishedL-def* **by** *simp*

declare *finished.simps*(6) [*simp del*]
declare *finished.simps*(7) [*simp del*]

lemma *finishedL-intro*[*intro*]:
assumes ⋀ *c*. *c* ∈ *set cl* ⟹ *finished c*
shows *finishedL cl*
using *assms* **unfolding** *finishedL-def* **by** *auto*

lemma *finishedL-finished*[*simp*]:
assumes *finishedL cl* **and** $c \in set\ cl$
shows *finished c*
using *assms* **unfolding** *finishedL-def* **by** *simp*

lemma *finishedL-update*[*simp*]:
assumes *cl*: *finishedL cl* **and** c': *finished* c'
shows *finishedL* $(cl[n := c'])$
proof(*cases* $n < length\ cl$)
  case *True*
  show *?thesis*
  unfolding *finishedL-def* **proof** *safe*
    fix *c* **assume** $c \in set\ (cl[n := c'])$
    hence $c \in insert\ c'\ (set\ cl)$ **using** *set-update-subset-insert* **by** *fastforce*
    thus *finished c* **using** *assms* **by** (*cases* $c = c'$) *auto*
  qed
qed (*use cl* **in** ‹*auto simp*: *list-update-beyond*›)

lemma *finished-fin*[*simp*]:
  finished c $\Longrightarrow$ *noWhile c*
  by(*induct c*) *auto*

lemma *finishedL-noWhileL*[*simp*]:
  finishedL cl $\Longrightarrow$ *noWhileL cl*
  unfolding *finishedL-def noWhileL-def* **by** *auto*

locale *PL* =
  fixes
    aval :: *'atom* $\Rightarrow$ *'state* $\Rightarrow$ *'state* **and**
    tval :: *'test* $\Rightarrow$ *'state* $\Rightarrow$ *bool* **and**
    cval :: *'choice* $\Rightarrow$ *'state* $\Rightarrow$ *real*
  assumes
    properCh: $\bigwedge$ *ch s. 0* $\leq$ *cval ch s* $\wedge$ *cval ch s* $\leq$ *1*
begin

lemma [*simp*]: (*n*::*nat*) $<$ *N* $\Longrightarrow$ *0* $\leq$ *1 / N* **by** *auto*

lemma [*simp*]: (*n*::*nat*) $<$ *N* $\Longrightarrow$ *1 / N* $\leq$ *1* **by** *auto*

lemma [*simp*]: (*n*::*nat*) $<$ *N* $\Longrightarrow$ *0* $\leq$ *1 − 1 / N* **by** (*simp add*: *divide-simps*)

lemma *sum-equal*: *0* $<$ (*N*::*nat*) $\Longrightarrow$ *sum* $(\lambda\ n.\ 1/N)\ \{..< N\}$ = *1*
unfolding *sum-constant* **by** *auto*

fun *proper* **where**
  proper Done $\longleftrightarrow$ *True*
| *proper* (*Atm x*) $\longleftrightarrow$ *True*
| *proper* (*Seq c1 c2*) $\longleftrightarrow$ *proper c1* $\wedge$ *proper c2*

| *proper* (*While tst c*) ⟷ *proper c*
| *proper* (*Ch ch c1 c2*) ⟷ *proper c1* ∧ *proper c2*
| *proper* (*Par cl*) ⟷ *cl* ≠ [] ∧ (∀ *c* ∈ *set cl*. *proper c*)
| *proper* (*ParT cl*) ⟷ *cl* ≠ [] ∧ (∀ *c* ∈ *set cl*. *proper c*)

definition *properL* **where**
properL cl ≡ *cl* ≠ [] ∧ (∀ *c* ∈ *set cl*. *proper c*)

lemma *proper-Par-properL*[*simp*]:
proper (*Par cl*) ⟷ *properL cl*
unfolding *properL-def* **by** *simp*

lemma *proper-ParT-properL*[*simp*]:
proper (*ParT cl*) ⟷ *properL cl*
unfolding *properL-def* **by** *simp*

declare *proper.simps*(*6*) [*simp del*]
declare *proper.simps*(*7*) [*simp del*]

lemma *properL-intro*[*intro*]:
⟦*cl* ≠ []; ⋀ *c*. *c* ∈ *set cl* ⟹ *proper c*⟧ ⟹ *properL cl*
unfolding *properL-def* **by** *auto*

lemma *properL-notEmp*[*simp*]: *properL cl* ⟹ *cl* ≠ []
unfolding *properL-def* **by** *simp*

lemma *properL-proper*[*simp*]:
⟦*properL cl*; *c* ∈ *set cl*⟧ ⟹ *proper c*
unfolding *properL-def* **by** *simp*

lemma *properL-update*[*simp*]:
assumes *cl*: *properL cl* **and** *c'*: *proper c'*
shows *properL* (*cl*[*n* := *c'*])
proof(*cases n* < *length cl*)
  case *True*
  show *?thesis*
  unfolding *properL-def* **proof** *safe*
    fix *c* **assume** *c* ∈ *set* (*cl*[*n* := *c'*])
    hence *c* ∈ *insert c'* (*set cl*) **using** *set-update-subset-insert* **by** *fastforce*
    thus *proper c* **using** *assms* **by** (*cases c* = *c'*) *auto*
  qed (*insert cl*, *auto*)
qed (*use cl* **in** ‹*auto simp*: *list-update-beyond*›)

lemma *proper-induct*[*consumes 1*, *case-names Done Atm Seq While Ch Par ParT*]:
assumes ∗: *proper c*
and *Done*: *phi Done*
and *Atm*: ⋀ *atm*. *phi* (*Atm atm*)
and *Seq*: ⋀ *c1 c2*. ⟦*phi c1*; *phi c2*⟧ ⟹ *phi* (*c1* ;; *c2*)
and *While*: ⋀ *tst c*. *phi c* ⟹ *phi* (*While tst c*)

and *Ch*: $\bigwedge$ *ch c1 c2*. $[\![$*phi c1*; *phi c2*$]\!] \Longrightarrow$ *phi* (*Ch ch c1 c2*)
and *Par*: $\bigwedge$ *cl*. $[\![$*properL cl*; $\bigwedge$ *c*. *c* $\in$ *set cl* $\Longrightarrow$ *phi c*$]\!] \Longrightarrow$ *phi* (*Par cl*)
and *ParT*: $\bigwedge$ *cl*. $[\![$*properL cl*; $\bigwedge$ *c*. *c* $\in$ *set cl* $\Longrightarrow$ *phi c*$]\!] \Longrightarrow$ *phi* (*ParT cl*)
shows *phi c*
using $*$ **apply**(*induct c*)
using *assms* **unfolding** *properL-def* **by** *auto*

1.2.1 Operational Small-Step Semantics

definition *theFT cl* $\equiv$ {*n*. *n* $<$ *length cl* $\wedge$ *finished* (*cl*!*n*)}

definition *theNFT cl* $\equiv$ {*n*. *n* $<$ *length cl* $\wedge$ $\neg$ *finished* (*cl*!*n*)}

lemma *finite-theFT*[*simp*]: *finite* (*theFT cl*)
unfolding *theFT-def* **by** *simp*

lemma *theFT-length*[*simp*]: *n* $\in$ *theFT cl* $\Longrightarrow$ *n* $<$ *length cl*
unfolding *theFT-def* **by** *simp*

lemma *theFT-finished*[*simp*]: *n* $\in$ *theFT cl* $\Longrightarrow$ *finished* (*cl*!*n*)
unfolding *theFT-def* **by** *simp*

lemma *finite-theNFT*[*simp*]: *finite* (*theNFT cl*)
unfolding *theNFT-def* **by** *simp*

lemma *theNFT-length*[*simp*]: *n* $\in$ *theNFT cl* $\Longrightarrow$ *n* $<$ *length cl*
unfolding *theNFT-def* **by** *simp*

lemma *theNFT-notFinished*[*simp*]: *n* $\in$ *theNFT cl* $\Longrightarrow$ $\neg$ *finished* (*cl*!*n*)
unfolding *theNFT-def* **by** *simp*

lemma *theFT-Int-theNFT*[*simp*]:
theFT cl Int theNFT cl = {} **and** *theNFT cl Int theFT cl* = {}
unfolding *theFT-def theNFT-def* **by** *auto*

lemma *theFT-Un-theNFT*[*simp*]:
theFT cl Un theNFT cl = {..< *length cl*} **and**
theNFT cl Un theFT cl = {..< *length cl*}
unfolding *theFT-def theNFT-def* **by** *auto*

lemma *in-theFT-theNFT*[*simp*]:
assumes *n1* $\in$ *theFT cl* **and** *n2* $\in$ *theNFT cl*
shows *n1* $\neq$ *n2* **and** *n2* $\neq$ *n1*
using *assms theFT-Int-theNFT* **by** *blast*+

definition *WtFT cl* $\equiv$ *sum* (λ (*n*::*nat*). *1*/(*length cl*)) (*theFT cl*)

definition *WtNFT cl* ≡ *sum* (λ (*n*::*nat*). *1*/(*length cl*)) (*theNFT cl*)

lemma *WtFT-WtNFT*[*simp*]:
assumes *0* < *length cl*
shows *WtFT cl* + *WtNFT cl* = *1* (**is** *?A* = *1*)
proof−
let *?w* = λ *n*. *1* / *length cl* **let** *?L* = *theFT cl* **let** *?Lnot* = *theNFT cl*
have *1*: {..< *length cl*} = *?L Un ?Lnot* **by** *auto*
have *?A* = *sum ?w ?L* + *sum ?w ?Lnot* **unfolding** *WtFT-def WtNFT-def* **by** *simp*
also have ... = *sum ?w* {..< *length cl*} **unfolding** *1*
apply(*rule sum.union-disjoint*[*THEN sym*]) **by** *auto*
also have ... = *1* **unfolding** *sum-equal*[*OF assms*] **by** *auto*
finally show *?thesis* **.**
qed

lemma *WtNFT-1-WtFT*: *0* < *length cl* ⟹ *WtNFT cl* = *1* − *WtFT cl*
by (*simp add*: *algebra-simps*)

lemma *WtNFT-WtFT-1*[*simp*]:
assumes *0* < *length cl* **and** *WtFT cl* ≠ *1*
shows *WtNFT cl* / (*1* − *WtFT cl*) = *1* (**is** *?A* / *?B* = *1*)
proof−
have *A*: *?A* = *?B* **using** *assms WtNFT-1-WtFT* **by** *auto*
show *?thesis* **unfolding** *A* **using** *assms* **by** *auto*
qed

lemma *WtFT-ge-0*[*simp*]: *WtFT cl* ≥ *0*
unfolding *WtFT-def* **by** (*rule sum-nonneg*) *simp*

lemma *WtFT-le-1*[*simp*]: *WtFT cl* ≤ *1* (**is** *?L* ≤ *1*)
proof−
let *?N* = *length cl*
have *?L* ≤ *sum* (λ *n*::*nat*. *1*/*?N*) {..< *?N*}
unfolding *WtFT-def* **apply**(*rule sum-mono2*) **by** *auto*
also have ... ≤ *1*
by (*metis div-by-0 le-cases neq0-conv not-one-le-zero of-nat-0 sum.neutral sum-equal*)
finally show *?thesis* **.**
qed

lemma *le-1-WtFT*[*simp*]: *0* ≤ *1* − *WtFT cl* (**is** *0* ≤ *?R*)
proof−
have *a*: −*1* ≤ − *WtFT cl* **by** *simp*
have (*0* :: *real*) = *1* + (−*1*) **by** *simp*
also have *1* + (−*1*) ≤ *1* + (− *WtFT cl*) **using** *a* **by** *arith*
also have ... = *?R* **by** *simp*
finally show *?thesis* **.**
qed

lemma *WtFT-lt-1*[*simp*]: *WtFT cl* $\neq$ *1* $\Longrightarrow$ *WtFT cl* $<$ *1*
using *WtFT-le-1* **by** (*auto simp add*: *le-less*)

lemma *lt-1-WtFT*[*simp*]: *WtFT cl* $\neq$ *1* $\Longrightarrow$ *0* $<$ *1* $-$ *WtFT cl*
using *le-1-WtFT* **by** (*metis le-1-WtFT eq-iff-diff-eq-0 less-eq-real-def*)

lemma *notFinished-WtFT*[*simp*]:
assumes *n* $<$ *length cl* **and** $\neg$ *finished* (*cl* ! *n*)
shows *1* / *length cl* $\leq$ *1* $-$ *WtFT cl*
proof−
  have *0* $<$ *length cl* **using** *assms* **by** *auto*
  thus *?thesis* **unfolding** *WtFT-def* **apply**(*intro sum-1*[*of* {..< *length cl*}])
  using *assms* **by** *auto*
qed

fun *brn* :: (*'test*, *'atom*, *'choice*) *cmd* $\Rightarrow$ *nat* **where**
  brn Done = *1*
| *brn* (*Atm atm*) = *1*
| *brn* (*c1* ;; *c2*) = *brn c1*
| *brn* (*While tst c*) = *1*
| *brn* (*Ch ch c1 c2*) = *2*
| *brn* (*Par cl*) = *lsum brn cl*
| *brn* (*ParT cl*) = *lsum brn cl*

lemma *brn-gt-0*: *proper c* $\Longrightarrow$ *0* $<$ *brn c*
by (*induct rule*: *proper-induct*) *auto*

lemma *brn-gt-0-L*: ⟦*properL cl*; *c* $\in$ *set cl*⟧ $\Longrightarrow$ *0* $<$ *brn c*
by (*metis brn-gt-0 properL-def*)

definition *locateT* $\equiv$ *locate1 brn* **definition** *locateI* $\equiv$ *locate2 brn*

definition *brnL cl n* $\equiv$ *lsum brn* (*take n cl*)

lemma *brnL-lsum*: *brnL cl* (*length cl*) = *lsum brn cl*
unfolding *brnL-def* **by** *simp*

lemma *brnL-unique*:
assumes *properL cl* **and** *n1* $<$ *length cl* $\wedge$ *j1* $<$ *brn* (*cl* ! *n1*)
and *n2* $<$ *length cl* $\wedge$ *j2* $<$ *brn* (*cl* ! *n2*) **and** *brnL cl n1* + *j1* = *brnL cl n2* + *j2*
shows *n1* = *n2* $\wedge$ *j1* = *j2*
apply (*rule less-lsum-unique*) **using** *assms brn-gt-0* **unfolding** *brnL-def properL-def* **by** *auto*

lemma *brn-Par-simp*[*simp*]: *brn* (*Par cl*) = *brnL cl* (*length cl*)
unfolding *brnL-lsum* **by** *simp*

lemma *brn-ParT-simp*[*simp*]: *brn* (*ParT cl*) = *brnL cl* (*length cl*)
unfolding *brnL-lsum* **by** *simp*

declare *brn.simps*(*6*)[*simp del*] **declare** *brn.simps*(*7*)[*simp del*]

lemma *brnL-0*[*simp*]: *brnL cl 0* = *0*
unfolding *brnL-def* **by** *auto*

lemma *brnL-Suc*[*simp*]: *n* < *length cl* ⟹ *brnL cl* (*Suc n*) = *brnL cl n* + *brn* (*cl* ! *n*)
unfolding *brnL-def* **using** *take-Suc-conv-app-nth*[*of n cl*] **by** *simp*

lemma *brnL-mono*[*simp*]: *n1* ≤ *n2* ⟹ *brnL cl n1* ≤ *brnL cl n2*
using *le-take*[*of n1 n2 cl*] **unfolding** *brnL-def* **by** *simp*

lemma *brnL-mono2*[*simp*]:
assumes *p*: *properL cl* **and** *n*: *n1* < *n2* **and** *l*: *n2* ≤ *length cl*
shows *brnL cl n1* < *brnL cl n2* (**is** *?L* < *?R*)
proof−
  have *1*: ⋀*c*. *c* ∈ *set* (*take n2 cl*) ⟹ *0* < *brn c*
  using *p* **by** (*metis brn-gt-0 in-set-takeD properL-proper*)
  have *take n1 cl* < *take n2 cl* **using** *n l lt-take* **by** *auto*
  hence *lsum brn* (*take n1 cl*) < *lsum brn* (*take n2 cl*)
  using *lsum-mono2*[*of take n2 cl* %*c*. *brn c take n1 cl*] *1* **by** *simp*
  thus *?thesis* **unfolding** *brnL-def* **.**
qed

lemma *brn-index*[*simp*]:
assumes *n*: *n* < *length cl* **and** *i*: *i* < *brn* (*cl* ! *n*)
shows *brnL cl n* + *i* < *brnL cl* (*length cl*) (**is** *?L* < *?R*)
proof−
  have *?L* < *brnL cl* (*Suc n*) **using** *assms* **by** *simp*
  also have ... ≤ *?R*
  using *n brnL-mono*[*of Suc n length cl cl*] **by** *simp*
  finally show *?thesis* **.**
qed

lemma *brnL-gt-0*[*simp*]: ⟦*properL cl*; *0* < *n*⟧ ⟹ *0* < *brnL cl n*
by (*metis properL-def brnL-mono brnL-mono2 le-0-eq length-greater-0-conv nat-le-linear neq0-conv*)

lemma *locateTI*:
assumes *properL cl* **and** *ii* < *brn* (*Par cl*)
shows
locateT cl ii < *length cl* ∧
 locateI cl ii < *brn* (*cl* ! (*locateT cl ii*)) ∧
 ii = *brnL cl* (*locateT cl ii*) + *locateI cl ii*
using *assms locate*[*of cl brn ii*] *brn-gt-0*

unfolding *locateT-def locateI-def brnL-def*
unfolding *brnL-lsum*[*THEN sym*] **by** *auto*

lemma *locateTI-unique*:
assumes *properL cl* **and** $n <$ *length cl*
and $i <$ *brn* (*cl* ! *n*) **and** *ii* = *brnL cl n* + *i*
shows *n* = *locateT cl ii* ∧ *i* = *locateI cl ii*
using *assms locate-unique*[*of cl brn*] *brn-gt-0*
unfolding *locateT-def locateI-def brnL-def*
unfolding *brnL-lsum*[*THEN sym*] **by** *auto*

definition *pickFT-pred* **where** *pickFT-pred cl n* ≡ *n* < *length cl* ∧ *finished* (*cl*!*n*)
definition *pickFT* **where** *pickFT cl* ≡ *SOME n. pickFT-pred cl n*

lemma *pickFT-pred*:
assumes *WtFT cl* = *1* **shows** ∃ *n. pickFT-pred cl n*
proof(*rule ccontr*, *unfold not-ex*)
assume ∀ *n.* ¬ *pickFT-pred cl n*
hence ⋀ *n. n* < *length cl* ⟹ ¬ *finished* (*cl*!*n*)
unfolding *pickFT-pred-def* **by** *auto*
hence *theFT cl* = {} **unfolding** *theFT-def* **by** *auto*
hence *WtFT cl* = *0* **unfolding** *WtFT-def* **by** *simp*
thus *False* **using** *assms* **by** *simp*
qed

lemma *pickFT-pred-pickFT*: *WtFT cl* = *1* ⟹ *pickFT-pred cl* (*pickFT cl*)
unfolding *pickFT-def* **by**(*auto intro*: *someI-ex pickFT-pred*)

lemma *pickFT-length*[*simp*]: *WtFT cl* = *1* ⟹ *pickFT cl* < *length cl*
using *pickFT-pred-pickFT* **unfolding** *pickFT-pred-def* **by** *auto*

lemma *pickFT-finished*[*simp*]: *WtFT cl* = *1* ⟹ *finished* (*cl* ! (*pickFT cl*))
using *pickFT-pred-pickFT* **unfolding** *pickFT-pred-def* **by** *auto*

lemma *pickFT-theFT*[*simp*]: *WtFT cl* = *1* ⟹ *pickFT cl* ∈ *theFT cl*
unfolding *theFT-def* **by** *auto*

fun *wt-cont-eff* **where**
wt-cont-eff Done s i = (*1*, *Done*, *s*)
|
wt-cont-eff (*Atm atm*) *s i* = (*1*, *Done*, *aval atm s*)
|
wt-cont-eff (*c1* ;; *c2*) *s i* =
(*case wt-cont-eff c1 s i of*
(*x*, *c1*′, *s*′) ⇒
if finished c1′ *then* (*x*, *c2*, *s*′) *else* (*x*, *c1*′ ;; *c2*, *s*′))
|

```
wt-cont-eff (While tst c) s i =
(if tval tst s
   then (1, c ;; (While tst c), s)
   else (1, Done, s))
|
wt-cont-eff (Ch ch c1 c2) s i =
 (if i = 0 then cval ch s else 1 − cval ch s,
  if i = 0 then c1 else c2,
  s)
|
wt-cont-eff (Par cl) s ii =
 (if cl ! (locateT cl ii) ∈ set cl then
 (case wt-cont-eff
        (cl ! (locateT cl ii))
        s
        (locateI cl ii) of
    (w, c′, s′) ⇒
      ((1 / (length cl)) ∗ w,
       Par (cl [(locateT cl ii) := c′]),
       s′))
  else undefined)
|
wt-cont-eff (ParT cl) s ii =
 (if cl ! (locateT cl ii) ∈ set cl
   then
     (case wt-cont-eff
            (cl ! (locateT cl ii))
            s
            (locateI cl ii) of
        (w, c′, s′) ⇒
          (if WtFT cl = 1
           then (if locateT cl ii = pickFT cl ∧ locateI cl ii = 0
                   then 1
                   else 0)
           else if finished (cl ! (locateT cl ii))
             then 0
             else (1 / (length cl))
                  / (1 − WtFT cl)
                  ∗ w,
           ParT (cl [(locateT cl ii) := c′]),
           s′))
   else undefined)


definition wt where wt c s i = fst (wt-cont-eff c s i)
definition cont where cont c s i = fst (snd (wt-cont-eff c s i))
definition eff where eff c s i = snd (snd (wt-cont-eff c s i))
```

lemma *wt-Done*[*simp*]: *wt Done s i = 1*
unfolding *wt-def* **by** *simp*

lemma *wt-Atm*[*simp*]: *wt* (*Atm atm*) *s i = 1*
unfolding *wt-def* **by** *simp*

lemma *wt-Seq*[*simp*]:
wt (*c1* ;; *c2*) *s* = *wt c1 s*
proof−
{fix *i* **have** *wt* (*c1* ;; *c2*) *s i* = *wt c1 s i*
proof(*cases wt-cont-eff c1 s i*)
case (*fields - c1′* -)
thus *?thesis* **unfolding** *wt-def* **by**(*cases c1′*, *auto*)
qed
}
thus *?thesis* **by** *auto*
qed

lemma *wt-While*[*simp*]: *wt* (*While tst c*) *s i = 1*
unfolding *wt-def* **by** *simp*

lemma *wt-Ch-L*[*simp*]: *wt* (*Ch ch c1 c2*) *s 0 = cval ch s*
unfolding *wt-def* **by** *simp*

lemma *wt-Ch-R*[*simp*]: *wt* (*Ch ch c1 c2*) *s* (*Suc n*) = *1 − cval ch s*
unfolding *wt-def* **by** *simp*

lemma *cont-Done*[*simp*]: *cont Done s i = Done*
unfolding *cont-def* **by** *simp*

lemma *cont-Atm*[*simp*]: *cont* (*Atm atm*) *s i = Done*
unfolding *cont-def* **by** *simp*

lemma *cont-Seq-finished*[*simp*]: *finished* (*cont c1 s i*) ⟹ *cont* (*c1* ;; *c2*) *s i = c2*
unfolding *cont-def* **by**(*cases wt-cont-eff c1 s i*) *auto*

lemma *cont-Seq-notFinished*[*simp*]:
assumes ¬ *finished* (*cont c1 s i*)
shows *cont* (*c1* ;; *c2*) *s i* = (*cont c1 s i*) ;; *c2*
proof(*cases wt-cont-eff c1 s i*)
case (*fields - c1′* -)
thus *?thesis* **using** *assms* **unfolding** *cont-def* **by**(*cases c1′*) *auto*
qed

lemma *cont-Seq-not-eq-finished*[*simp*]: ¬ *finished c2* ⟹ ¬ *finished* (*cont* (*Seq c1 c2*) *s i*)
by (*cases finished* (*cont c1 s i*)) *auto*

lemma *cont-While-False*[*simp*]: *tval tst s* = *False* ⟹ *cont* (*While tst c*) *s i* = *Done*
unfolding *cont-def* **by** *simp*

lemma *cont-While-True*[*simp*]: *tval tst s* = *True* ⟹ *cont* (*While tst c*) *s i* = *c* ;; (*While tst c*)
unfolding *cont-def* **by** *simp*

lemma *cont-Ch-L*[*simp*]: *cont* (*Ch ch c1 c2*) *s 0* = *c1*
unfolding *cont-def* **by** *simp*

lemma *cont-Ch-R*[*simp*]: *cont* (*Ch ch c1 c2*) *s* (*Suc n*) = *c2*
unfolding *cont-def* **by** *simp*

lemma *eff-Done*[*simp*]: *eff Done s i* = *s*
unfolding *eff-def* **by** *simp*

lemma *eff-Atm*[*simp*]: *eff* (*Atm atm*) *s i* = *aval atm s*
unfolding *eff-def* **by** *simp*

lemma *eff-Seq*[*simp*]: *eff* (*c1* ;; *c2*) *s* = *eff c1 s*
proof−
 {fix *i* **have** *eff* (*c1* ;; *c2*) *s i* = *eff c1 s i*
  proof(*cases wt-cont-eff c1 s i*)
   case (*fields - c1′ -*)
   thus *?thesis*
   unfolding *eff-def* **by**(*cases c1′*) *auto*
  qed
 }
 thus *?thesis* **by** *auto*
qed

lemma *eff-While*[*simp*]: *eff* (*While tst c*) *s i* = *s*
unfolding *eff-def* **by** *simp*

lemma *eff-Ch*[*simp*]: *eff* (*Ch ch c1 c2*) *s i* = *s*
unfolding *eff-def* **by** *simp*

lemma *brnL-nchotomy*:
assumes *properL cl* **and** *ii* < *brnL cl* (*length cl*)
shows ∃ *n i*. *n* < *length cl* ∧ *i* < *brn* (*cl* ! *n*) ∧ *ii* = *brnL cl n* + *i*
unfolding *brnL-def* **apply**(*rule less-lsum-nchotomy*) **using** *assms brn-gt-0*
unfolding *brnL-lsum*[*THEN sym*] **by** *auto*

corollary *brnL-cases*[*consumes 2*, *case-names Local*, *elim*]:
assumes *properL cl* **and** *ii* < *brnL cl* (*length cl*) **and**

$\bigwedge n\ i.\ [\![n < length\ cl;\ i < brn\ (cl\ !\ n);\ ii = brnL\ cl\ n + i]\!] \Longrightarrow phi$
shows *phi*
using *assms brnL-nchotomy* **by** *blast*

lemma *wt-cont-eff-Par*[*simp*]:
assumes *p*: *properL cl*
and *n*: $n < length\ cl$ **and** *i*: $i < brn\ (cl\ !\ n)$
shows
$wt\ (Par\ cl)\ s\ (brnL\ cl\ n + i) =$
$1\ /\ (length\ cl) * wt\ (cl\ !\ n)\ s\ i$
(**is** *?wL* = *?wR*)

$cont\ (Par\ cl)\ s\ (brnL\ cl\ n + i) =$
$Par\ (cl\ [n := cont\ (cl\ !\ n)\ s\ i])$
(**is** *?mL* = *?mR*)

$eff\ (Par\ cl)\ s\ (brnL\ cl\ n + i) =$
$eff\ (cl\ !\ n)\ s\ i$
(**is** *?eL* = *?eR*)
proof−
define *ii* **where** $ii = brnL\ cl\ n + i$
define *n1* **where** $n1 = locateT\ cl\ ii$
define *i1* **where** $i1 = locateI\ cl\ ii$
have *n-i*: $n = n1\ i = i1$ **using** *p* **unfolding** *n1-def i1-def*
using *ii-def locateTI-unique n i* **by** *auto*
have *lsum1*: $ii < brnL\ cl\ (length\ cl)$ **unfolding** *ii-def* **using** *n i* **by** *simp*
hence $n1 < length\ cl$
unfolding *n1-def* **using** *i p locateTI*[*of cl ii*] **by** *simp*
hence *set*: $cl\ !\ n1 \in set\ cl$ **by** *simp*

have $?wL = 1\ /\ (length\ cl) * wt\ (cl\ !\ n1)\ s\ i1$
unfolding *ii-def*[*THEN sym*]
apply (*cases wt-cont-eff* $(cl\ !\ n1)\ s\ i1$)
using *set* **unfolding** *n1-def i1-def* **unfolding** *wt-def* **by** *simp*
also have ... = *?wR* **unfolding** *n-i* **by** *simp*
finally show *?wL* = *?wR* **.**

have $?mL = Par\ (cl\ [n1 := cont\ (cl\ !\ n1)\ s\ i1])$
unfolding *ii-def*[*THEN sym*]
apply (*cases wt-cont-eff* $(cl\ !\ n1)\ s\ i1$)
using *set* **unfolding** *n1-def i1-def* **unfolding** *cont-def* **by** *simp*
also have ... = *?mR* **unfolding** *n-i* **by** *simp*
finally show *?mL* = *?mR* **.**

have $?eL = eff\ (cl\ !\ n1)\ s\ i1$
unfolding *ii-def*[*THEN sym*]
apply (*cases wt-cont-eff* $(cl\ !\ n1)\ s\ i1$)
using *set* **unfolding** *n1-def i1-def* **unfolding** *eff-def* **by** *simp*
also have $eff\ (cl\ !\ n1)\ s\ i1$ = *?eR* **unfolding** *n-i* **by** *simp*

```
  finally show ?eL = ?eR .
qed

lemma cont-eff-ParT[simp]:
assumes p: properL cl
and n: n < length cl and i: i < brn (cl ! n)
shows
cont (ParT cl) s (brnL cl n + i) =
 ParT (cl [n := cont (cl ! n) s i])
(is ?mL = ?mR)

eff (ParT cl) s (brnL cl n + i) =
 eff (cl ! n) s i
(is ?eL = ?eR)
proof−
  define ii where ii = brnL cl n + i
  define n1 where n1 = locateT cl ii
  define i1 where i1 = locateI cl ii
  have n-i: n = n1 i = i1 using p unfolding n1-def i1-def
  using ii-def locateTI-unique n i by auto
  have lsum1: ii < brnL cl (length cl) unfolding ii-def using n i by simp
  hence n1 < length cl
  unfolding n1-def using i p locateTI[of cl ii] by simp
  hence set: cl ! n1 ∈ set cl by simp

  have ?mL = ParT (cl [n1 := cont (cl ! n1) s i1])
  unfolding ii-def[THEN sym]
  apply (cases wt-cont-eff (cl ! n1) s i1)
  using set unfolding n1-def i1-def unfolding cont-def by simp
  also have ... = ?mR unfolding n-i by simp
  finally show ?mL = ?mR .

  have ?eL = eff (cl ! n1) s i1
  unfolding ii-def[THEN sym]
  apply (cases wt-cont-eff (cl ! n1) s i1)
  using set unfolding n1-def i1-def unfolding eff-def by simp
  also have eff (cl ! n1) s i1 = ?eR unfolding n-i by simp
  finally show ?eL = ?eR .
qed

lemma wt-ParT-WtFT-pickFT-0[simp]:
assumes p: properL cl and WtFT: WtFT cl = 1
shows wt (ParT cl) s (brnL cl (pickFT cl)) = 1
(is ?wL = 1)
proof−
  define ii where ii = brnL cl (pickFT cl)
  define n1 where n1 = locateT cl ii
  define i1 where i1 = locateI cl ii
  have ni: pickFT cl < length cl 0 < brn (cl! (pickFT cl))
```

using *WtFT p brn-gt-0* **by** *auto*
hence *n-i*: *pickFT cl = n1 0 = i1* **using** *p* **unfolding** *n1-def i1-def*
using *ii-def locateTI-unique*[*of cl pickFT cl 0 ii*] **by** *auto*
have *lsum1*: *ii < brnL cl* (*length cl*) **unfolding** *ii-def* **using** *ni*
by (*metis add.comm-neutral brn-index*)
hence *n1 < length cl*
unfolding *n1-def* **using** *ni p locateTI*[*of cl ii*] **by** *simp*
hence *set*: *cl ! n1 ∈ set cl* **by** *simp*

have *n1i1*: *n1 = pickFT cl ∧ i1 = 0* **using** *WtFT ni* **unfolding** *n-i* **by** *auto*
show *?wL = 1*
unfolding *ii-def*[*THEN sym*]
apply (*cases wt-cont-eff* (*cl ! n1*) *s i1*)
using *WtFT n1i1 set* **unfolding** *n1-def i1-def* **unfolding** *wt-def* **by** *simp*
qed

lemma *wt-ParT-WtFT-notPickFT-0*[*simp*]:
assumes *p*: *properL cl* **and** *n*: *n < length cl* **and** *i*: *i < brn* (*cl ! n*)
and *WtFT*: *WtFT cl = 1* **and** *ni*: *n = pickFT cl ⟶ i ≠ 0*
shows *wt* (*ParT cl*) *s* (*brnL cl n + i*) *= 0* (**is** *?wL = 0*)
proof−
define *ii* **where** *ii = brnL cl n + i*
define *n1* **where** *n1 = locateT cl ii*
define *i1* **where** *i1 = locateI cl ii*
have *n-i*: *n = n1 i = i1* **using** *p* **unfolding** *n1-def i1-def*
using *ii-def locateTI-unique n i* **by** *auto*
have *lsum1*: *ii < brnL cl* (*length cl*) **unfolding** *ii-def* **using** *n i* **by** *simp*
hence *n1 < length cl*
unfolding *n1-def* **using** *i p locateTI*[*of cl ii*] **by** *simp*
hence *set*: *cl ! n1 ∈ set cl* **by** *simp*

have *n1i1*: *n1 ≠ pickFT cl ∨ i1 ≠ 0* **using** *WtFT ni* **unfolding** *n-i* **by** *auto*
show *?wL = 0*
unfolding *ii-def*[*THEN sym*]
apply (*cases wt-cont-eff* (*cl ! n1*) *s i1*)
using *WtFT n1i1 set* **unfolding** *n1-def i1-def* **unfolding** *wt-def* **by** *force*
qed

lemma *wt-ParT-notWtFT-finished*[*simp*]:
assumes *p*: *properL cl* **and** *n*: *n < length cl* **and** *i*: *i < brn* (*cl ! n*)
and *WtFT*: *WtFT cl ≠ 1* **and** *f*: *finished* (*cl ! n*)
shows *wt* (*ParT cl*) *s* (*brnL cl n + i*) *= 0* (**is** *?wL = 0*)
proof−
define *ii* **where** *ii = brnL cl n + i*
define *n1* **where** *n1 = locateT cl ii*
define *i1* **where** *i1 = locateI cl ii*
have *n-i*: *n = n1 i = i1* **using** *p* **unfolding** *n1-def i1-def*
using *ii-def locateTI-unique n i* **by** *auto*
have *lsum1*: *ii < brnL cl* (*length cl*) **unfolding** *ii-def* **using** *n i* **by** *simp*

```
  hence n1 < length cl
  unfolding n1-def using i p locateTI[of cl ii] by simp
  hence set: cl ! n1 ∈ set cl by simp

  have f: finished (cl ! n1) using f unfolding n-i by auto
  show ?wL = 0
  unfolding ii-def[THEN sym]
  apply (cases wt-cont-eff (cl ! n1) s i1)
  using WtFT f set unfolding n1-def i1-def unfolding wt-def by simp
qed

lemma wt-cont-eff-ParT-notWtFT-notFinished[simp]:
assumes p: properL cl and n: n < length cl and i: i < brn (cl ! n)
and WtFT: WtFT cl ≠ 1 and nf: ¬ finished (cl ! n)
shows wt (ParT cl) s (brnL cl n + i) =
      (1 / (length cl)) / (1 − WtFT cl) ∗ wt (cl ! n) s i (is ?wL = ?wR)
proof−
  define ii where ii = brnL cl n + i
  define n1 where n1 = locateT cl ii
  define i1 where i1 = locateI cl ii
  have n-i: n = n1 i = i1 using p unfolding n1-def i1-def
  using ii-def locateTI-unique n i by auto
  have lsum1: ii < brnL cl (length cl) unfolding ii-def using n i by simp
  hence n1 < length cl unfolding n1-def using i p locateTI[of cl ii] by simp
  hence set: cl ! n1 ∈ set cl by simp

  have nf: ¬ finished (cl ! n1) using nf unfolding n-i by auto
  have ?wL = (1 / (length cl)) / (1 − WtFT cl) ∗ wt (cl ! n1) s i1
  unfolding ii-def[THEN sym]
  apply (cases wt-cont-eff (cl ! n1) s i1)
  using WtFT nf set unfolding n1-def i1-def unfolding wt-def by simp
  also have ... = ?wR unfolding n-i by simp
  finally show ?wL = ?wR .
qed


lemma wt-ge-0[simp]:
assumes proper c and i < brn c
shows 0 ≤ wt c s i
using assms proof (induct c arbitrary: i s rule: proper-induct)
  case (Ch ch c1 c2)
  thus ?case
  using properCh  by (cases i) (auto simp: algebra-simps)
next
  case (Par cl ii)
  have properL cl and ii < brnL cl (length cl) using Par by auto
  thus ?case
  apply (cases rule: brnL-cases)
  using Par by simp
```

```
next
  case (ParT cl ii)
  have properL cl and ii < brnL cl (length cl) using ParT by auto
  thus ?case
  proof(cases rule: brnL-cases)
    case (Local n i)
    show ?thesis
    proof (cases WtFT cl = 1)
      case True
      thus ?thesis using Local ParT by (cases n = pickFT cl ∧ i = 0) auto
    next
      case False
      thus ?thesis using Local ParT by (cases finished (cl ! n)) auto
    qed
  qed
qed auto

lemma wt-le-1[simp]:
assumes proper c and i < brn c
shows wt c s i ≤ 1
using assms proof (induct c arbitrary: i s rule: proper-induct)
  case (Ch ch c1 c2)
  thus ?case
  using properCh by (cases i) auto
next
  case (Par cl ii)
  hence properL cl and ii < brnL cl (length cl) by auto
  thus ?case
  apply (cases rule: brnL-cases) apply simp
  using Par apply auto
 by (metis add-increasing2 diff-is-0-eq gr0-conv-Suc less-imp-diff-less less-or-eq-imp-le
nth-mem of-nat-0-le-iff of-nat-Suc)
next
  case (ParT cl ii)
  have properL cl and ii < brnL cl (length cl) using ParT by auto
  thus ?case
  proof(cases rule: brnL-cases)
    case (Local n i)
    show ?thesis
    proof (cases WtFT cl = 1)
      case True
      thus ?thesis using Local ParT by (cases n = pickFT cl ∧ i = 0, auto)
    next
      case False note sch = False
      thus ?thesis using Local ParT proof (cases finished (cl ! n))
        case False note cln = False
        let ?L1 = 1 / (length cl) let ?L2 = wt (cl ! n) s i
        let ?R = WtFT cl
        have 0 ≤ ?L1 and 0 ≤ ?L2 using ParT Local by auto
```

```
      moreover have ?L2 ≤ 1 using ParT Local by auto
      ultimately have ?L1 * ?L2 ≤ ?L1 by (metis mult-right-le-one-le)
      also have ?L1 ≤ 1 − ?R using ParT Local cln by auto
      finally have ?L1 * ?L2 ≤ 1 − ?R .
      thus ?thesis using Local ParT cln sch
        by (auto simp: pos-divide-le-eq mult.commute)
    qed (insert sch ParT Local, auto)
  qed
 qed
qed auto

abbreviation fromPlus (‹(1{-..<+-})›) where
{a ..<+ b} ≡ {a ..< a + b}

lemma brnL-UN:
assumes properL cl
shows {..< brnL cl (length cl)} = (⋃ n < length cl. {brnL cl n ..<+ brn (cl!n)})
(is ?L = (⋃ n < length cl. ?R n))
proof safe
 fix ii assume ii: ii < brnL cl (length cl)
 from assms ii show ii ∈ (⋃ n < length cl. ?R n)
 proof(cases rule: brnL-cases)
  case (Local n i)
  hence ii ∈ ?R n by simp
  thus ?thesis using Local by force
 qed
qed auto

lemma brnL-Int-lt:
assumes n12: n1 < n2 and n2: n2 < length cl
shows
{brnL cl n1 ..<+ brn (cl!n1)} ∩ {brnL cl n2 ..<+ brn (cl!n2)} = {}
proof−
 have Suc n1 ≤ n2 using assms by simp
 hence brnL cl (Suc n1) ≤ brnL cl n2 by simp
 thus ?thesis using assms by simp
qed

lemma brnL-Int:
assumes n1 ≠ n2 and n1 < length cl and n2 < length cl
shows {brnL cl n1 ..<+ brn (cl!n1)} ∩ {brnL cl n2 ..<+ brn (cl!n2)} = {}
proof(cases n1 < n2)
 case True thus ?thesis using assms brnL-Int-lt by auto
next
 case False
 hence n2 < n1 using assms by simp
 thus ?thesis using brnL-Int-lt assms by fastforce
qed
```

lemma *sum-wt-Par-sub*[*simp*]:
assumes *cl*: *properL cl* **and** *n*: $n < length\ cl$ **and** *I*: $I \subseteq \{..< brn\ (cl\ !\ n)\}$
shows $sum\ (wt\ (Par\ cl)\ s)\ ((+)\ (brnL\ cl\ n)\ `\ I) =$
$1\ /(length\ cl) * sum\ (wt\ (cl\ !\ n)\ s)\ I$ (**is** *?L* = *?wSch* ∗ *?R*)
proof−
have *?L* = $sum\ (\%i.\ ?wSch * wt\ (cl\ !\ n)\ s\ i)\ I$
apply(*rule sum.reindex-cong*[*of* $(+)\ (brnL\ cl\ n)$]) **using** *assms* **by** *auto*
also have ... = *?wSch* ∗ *?R*
unfolding *sum-distrib-left* **by** *simp*
finally show *?thesis* **.**
qed

lemma *sum-wt-Par*[*simp*]:
assumes *cl*: *properL cl* **and** *n*: $n < length\ cl$
shows $sum\ (wt\ (Par\ cl)\ s)\ \{brnL\ cl\ n\ ..<+\ brn\ (cl!n)\} =$
$1\ /(length\ cl) * sum\ (wt\ (cl\ !\ n)\ s)\ \{..< brn\ (cl\ !\ n)\}$ (**is** *?L* = *?W* ∗ *?R*)
using *assms* **by** (*simp add*: *sum-distrib-left*)

lemma *sum-wt-ParT-sub-WtFT-pickFT-0*[*simp*]:
assumes *cl*: *properL cl* **and** *nf*: *WtFT cl* = *1*
and *I*: $I \subseteq \{..< brn\ (cl\ !\ (pickFT\ cl))\}\ 0 \in I$
shows $sum\ (wt\ (ParT\ cl)\ s)\ ((+)\ (brnL\ cl\ (pickFT\ cl))\ `\ I) = 1$ (**is** *?L* = *1*)
proof−
let *?n* = *pickFT cl*
let *?w* = $\%i.\ if\ i = 0\ then\ (1::real)\ else\ 0$
have *n*: $?n < length\ cl$ **using** *nf* **by** *simp*
have *0*: $I = \{0\}\ Un\ (I - \{0\})$ **using** *I* **by** *auto*
have *finI*: *finite I* **using** *I* **by** (*metis finite-nat-iff-bounded*)
have *?L* = *sum ?w I*
proof (*rule sum.reindex-cong* [*of plus* (*brnL cl ?n*)])
fix *i* **assume** *i*: $i \in I$
have $i < brn\ (cl\ !\ ?n)$ **using** *i I* **by** *auto*
note $i = this\ i$
show $wt\ (ParT\ cl)\ s\ (brnL\ cl\ ?n + i) = ?w\ i$
using *nf n i cl* **by** (*cases i* = *0*) *auto*
qed (*insert assms*, *auto*)
also have ... = $sum\ ?w\ (\{0\}\ Un\ (I - \{0\}))$ **using** *0* **by** *auto*
also have ... = $sum\ ?w\ \{0::real\} + sum\ ?w\ (I - \{0\})$
using *sum.union-disjoint*[*of* $\{0\}\ I - \{0\}$ *?w*] *finI* **by** *auto*
also have ... = *1* **by** *simp*
finally show *?thesis* **.**
qed

lemma *sum-wt-ParT-sub-WtFT-pickFT-0-2*[*simp*]:
assumes *cl*: *properL cl* **and** *nf*: *WtFT cl* = *1*
and *II*: $II \subseteq \{..< brnL\ cl\ (length\ cl)\}\ brnL\ cl\ (pickFT\ cl) \in II$
shows $sum\ (wt\ (ParT\ cl)\ s)\ II = 1$ (**is** *?L* = *1*)
proof−
let *?n* = *pickFT cl*

```
  let ?w = %ii. if ii = brnL cl (pickFT cl) then (1::real) else 0
  have n: ?n < length cl using nf by simp
  have 0: II = {brnL cl ?n} Un (II − {brnL cl ?n}) using II by auto
  have finI: finite II using II by (metis finite-nat-iff-bounded)
  have ?L = sum ?w II
  proof(rule sum.cong)
    fix ii assume ii: ii ∈ II
    hence ii: ii < brnL cl (length cl) using II by auto
    from cl ii show wt (ParT cl) s ii = ?w ii
    proof(cases rule: brnL-cases)
      case (Local n i)
      show ?thesis
      proof(cases ii = brnL cl (pickFT cl))
        case True
        have n = pickFT cl ∧ i = 0
        apply(intro brnL-unique[of cl]) using Local cl nf brn-gt-0 unfolding True
by auto
        thus ?thesis using cl nf True by simp
      next
        case False
        hence n = pickFT cl ⟶ i ≠ 0 unfolding Local by auto
        thus ?thesis using Local ii nf cl False by auto
      qed
    qed
  qed simp
  also have ... = sum ?w ({brnL cl ?n} Un (II − {brnL cl ?n})) using 0 by simp
  also have ... = sum ?w {brnL cl ?n} + sum ?w (II − {brnL cl ?n})
  apply(rule sum.union-disjoint) using finI by auto
  also have ... = 1 by simp
  finally show ?thesis .
qed

lemma sum-wt-ParT-sub-WtFT-notPickFT-0[simp]:
assumes cl: properL cl and nf: WtFT cl = 1 and n: n < length cl
and I: I ⊆ {..< brn (cl ! n)} and nI: n = pickFT cl ⟶ 0 ∉ I
shows sum (wt (ParT cl) s) ((+) (brnL cl n) ` I) = 0 (is ?L = 0)
proof−
  have ?L = sum (%i. 0) I
  proof (rule sum.reindex-cong [of plus (brnL cl n)])
    fix i assume i: i ∈ I
    hence n = pickFT cl ⟶ i ≠ 0 using nI by metis
    moreover have i < brn (cl ! n) using i I by auto
    ultimately show wt (ParT cl) s (brnL cl n + i) = 0
    using nf n cl by simp
  qed (insert assms, auto)
  also have ... = 0 by simp
  finally show ?thesis .
qed
```

lemma *sum-wt-ParT-sub-notWtFT-finished*[*simp*]:
assumes *cl*: *properL cl* **and** *nf*: *WtFT cl* ≠ *1*
and *n*: *n* < *length cl* **and** *cln*: *finished* (*cl*!*n*) **and** *I*: *I* ⊆ {..< *brn* (*cl* ! *n*)}
shows *sum* (*wt* (*ParT cl*) *s*) ((+) (*brnL cl n*) ‘ *I*) = *0* (**is** *?L* = *0*)
proof−
have *?L* = *sum* (%*i*. *0*) *I*
apply(*rule sum.reindex-cong*[*of* (+) (*brnL cl n*)]) **using** *assms* **by** *auto*
also have ... = *0* **by** *simp*
finally show *?thesis* **.**
qed

lemma *sum-wt-ParT-sub-notWtFT-notFinished*[*simp*]:
assumes *cl*: *properL cl* **and** *nf*: *WtFT cl* ≠ *1* **and** *n*: *n* < *length cl*
and *cln*: ¬ *finished* (*cl*!*n*) **and** *I*: *I* ⊆ {..< *brn* (*cl* ! *n*)}
shows
sum (*wt* (*ParT cl*) *s*) ((+) (*brnL cl n*) ‘ *I*) =
(*1* / (*length cl*)) / (*1* − *WtFT cl*) ∗ *sum* (*wt* (*cl* ! *n*) *s*) *I*
(**is** *?L* = *?w* / (*1* − *?wF*) ∗ *?R*)
proof−
have *?L* = *sum* (%*i*. *?w* / (*1* − *?wF*) ∗ *wt* (*cl* ! *n*) *s i*) *I*
apply(*rule sum.reindex-cong*[*of* (+) (*brnL cl n*)])
using *assms* **by** *auto*
also have ... = *?w* / (*1* − *?wF*) ∗ *?R*
unfolding *sum-distrib-left* **by** *simp*
finally show *?thesis* **.**
qed

lemma *sum-wt-ParT-WtFT-pickFT-0*[*simp*]:
assumes *cl*: *properL cl* **and** *nf*: *WtFT cl* = *1*
shows *sum* (*wt* (*ParT cl*) *s*) {*brnL cl* (*pickFT cl*) ..<+ *brn* (*cl* ! (*pickFT cl*))} =
1
proof−
let *?n* = *pickFT cl*
have *1*: {*brnL cl ?n* ..<+ *brn* (*cl* ! *?n*)} =
(+) (*brnL cl ?n*) ‘ {..< *brn* (*cl* ! *?n*)} **by** *simp*
show *?thesis* **unfolding** *1*
apply(*rule sum-wt-ParT-sub-WtFT-pickFT-0*)
using *assms* **apply** *simp-all*
by (*metis brn-gt-0-L nth-mem pickFT-length*)
qed

lemma *sum-wt-ParT-WtFT-notPickFT-0*[*simp*]:
assumes *cl*: *properL cl* **and** *nf*: *WtFT cl* = *1* **and** *n*: *n* < *length cl n* ≠ *pickFT cl*
shows *sum* (*wt* (*ParT cl*) *s*) {*brnL cl n* ..<+ *brn* (*cl*!*n*)} = *0*
proof−
have *1*: {*brnL cl n* ..<+ *brn* (*cl*!*n*)} = (+) (*brnL cl n*) ‘ {..< *brn* (*cl*!*n*)} **by** *simp*
show *?thesis* **unfolding** *1* **apply**(*rule sum-wt-ParT-sub-WtFT-notPickFT-0*)

using *assms* **by** *auto*
qed

lemma *sum-wt-ParT-notWtFT-finished*[*simp*]:
assumes *cl*: *properL cl* **and** *WtFT cl* ≠ *1*
and *n*: *n* < *length cl* **and** *cln*: *finished* (*cl*!*n*)
shows *sum* (*wt* (*ParT cl*) *s*) {*brnL cl n* ..<+ *brn* (*cl*!*n*)} = *0*
proof−
have *1*: {*brnL cl n* ..<+ *brn* (*cl*!*n*)} = (+) (*brnL cl n*) ‘ {..< *brn* (*cl*!*n*)} **by** *simp*
show *?thesis* **unfolding** *1*
apply(*rule sum-wt-ParT-sub-notWtFT-finished*) **using** *assms* **by** *auto*
qed

lemma *sum-wt-ParT-notWtFT-notFinished*[*simp*]:
assumes *cl*: *properL cl* **and** *nf*: *WtFT cl* ≠ *1*
and *n*: *n* < *length cl* **and** *cln*: ¬ *finished* (*cl*!*n*)
shows
sum (*wt* (*ParT cl*) *s*) {*brnL cl n* ..<+ *brn* (*cl*!*n*)} =
(*1* / (*length cl*)) / (*1* − *WtFT cl*) ∗
sum (*wt* (*cl* ! *n*) *s*) {..< *brn* (*cl* ! *n*)}
proof−
have *1*: {*brnL cl n* ..<+ *brn* (*cl*!*n*)} = (+) (*brnL cl n*) ‘ {..< *brn* (*cl*!*n*)} **by** *simp*
show *?thesis* **unfolding** *1* **apply**(*rule sum-wt-ParT-sub-notWtFT-notFinished*)
using *assms* **by** *auto*
qed

lemma *sum-wt*[*simp*]:
assumes *proper c*
shows *sum* (*wt c s*) {..< *brn c*} = *1*
using *assms* **proof** (*induct c arbitrary*: *s rule*: *proper-induct*)
case (*Par cl*)
let *?w* = λ *n*. *1* / (*length cl*) ∗ *sum* (*wt* (*cl* ! *n*) *s*) {..< *brn* (*cl* ! *n*)}
show *?case*
proof (*rule sum-UN-introL* [*of* - %*n*. {*brnL cl n* ..<+ *brn* (*cl*!*n*)} {..< *length cl*} - *?w*])
have *1* = *sum* (λ *n*. *1* / (*length cl*)) {..< *length cl*}
using *Par* **by** *simp*
also have ... = *sum ?w* {..< *length cl*} **using** *Par* **by** *simp*
finally show *1* = *sum ?w* {..< *length cl*} **.**
next
fix *m n* **assume** {*m*, *n*} ⊆ {..<*length cl*} ∧ *m* ≠ *n*
thus
{*brnL cl m* ..<+ *brn* (*cl*!*m*)} ∩ {*brnL cl n* ..<+ *brn* (*cl*!*n*)} = {}
using *brnL-Int* **by** *auto*
qed(*insert Par brnL-UN sum-wt-Par*, *auto*)
next
case (*ParT cl*)

```
let ?v = 1/(length cl) let ?wtF = WtFT cl let ?wtNF = WtNFT cl
let ?w = λn.
if ?wtF = 1
  then
    (if n = pickFT cl then 1 else 0)
  else
    (if finished (cl!n)
       then 0
       else ?v / (1 − ?wtF) ∗
            sum (wt (cl ! n) s) {..< brn (cl ! n)})
define w where w = ?w
have w: ⋀ n. ?wtF ≠ 1 ∧ n < length cl ∧ ¬ finished (cl!n)
⟹ w n = ?v / (1 − ?wtF)
unfolding w-def using ParT by simp
show ?case
proof(cases WtFT cl = 1)
  case True
  with ParT show ?thesis by simp
next
  case False note nf = False
  show ?thesis
  proof (rule sum-UN-introL [of - %n. {brnL cl n ..<+ brn (cl!n)} {..< length
cl} - w])
    show 1 = sum w {..< length cl}
    proof(cases ?wtF = 1)
      case True note sch = True
      let ?n = pickFT cl
      let ?L = {?n} let ?Lnot = {..<length cl} − {?n}
      have ?n < length cl using ParT True by auto
      hence {..< length cl} = ?L Un ?Lnot by auto
      hence sum w {..< length cl} = sum w (?L Un ?Lnot) by simp
      also have ... = sum w ?L + sum w ?Lnot
      apply(rule sum.union-disjoint) by auto
      also have ... = 1 unfolding w-def using sch by simp
      finally show ?thesis by simp
    next
      case False note sch = False
      let ?L = theFT cl let ?Lnot = theNFT cl
      have 1: {..< length cl} = ?L Un ?Lnot by auto
      have sum w {..< length cl} = sum w ?L + sum w ?Lnot
      unfolding 1 apply(rule sum.union-disjoint) by auto
      also have ... = sum w ?Lnot unfolding w-def using sch by simp
      also have ... = sum (%n. ?v / (1 − ?wtF)) ?Lnot
      apply(rule sum.cong) using w sch by auto
      also have ... = sum (%n. ?v) ?Lnot / (1 − ?wtF)
      unfolding sum-divide-distrib by simp
      also have ... = ?wtNF / (1 − ?wtF) unfolding WtNFT-def by simp
      also have ... = 1 using nf ParT by simp
      finally show ?thesis by simp
```

```
    qed
  next
    fix n assume n: n ∈ {..<length cl}
    show sum (wt (ParT cl) s) {brnL cl n..<+brn (cl ! n)} = w n
    proof−
      have (∑ i<brn (cl ! n). ?v ∗ wt (cl ! n) s i / (1 − ?wtF)) =
      ?v ∗ (∑ i<brn (cl ! n). wt (cl ! n) s i) / (1 − ?wtF)
      unfolding sum-distrib-left sum-divide-distrib by simp
      also have ... = ?v / (1 − ?wtF) using ParT n by simp
      finally have (∑ i<brn (cl ! n). ?v ∗ wt (cl ! n) s i / (1 − ?wtF)) =
      ?v / (1 − ?wtF) .
      thus ?thesis unfolding w-def using n nf ParT by simp
    qed
  qed(insert ParT brnL-UN brnL-Int sum-wt-Par, auto)
 qed
qed auto

lemma proper-cont[simp]:
assumes proper c and i < brn c
shows proper (cont c s i)
using assms proof(induct c arbitrary: i s rule: cmd.induct)
 case (Ch ch c1 c2)
 thus ?case by (cases i) auto
next
 case (Seq c1 c2) thus ?case
 by (cases finished (cont c1 s i)) auto
next
 case (While tst c) thus ?case
 by (cases tval tst s) auto
next
 case (Par cl ii)
 hence properL cl and ii < brnL cl (length cl) by auto
 thus ?case
 using Par by (cases rule: brnL-cases) auto
next
 case (ParT cl ii)
 have properL cl and ii < brnL cl (length cl) using ParT by auto
 thus ?case apply (cases rule: brnL-cases) using ParT by auto
qed auto

lemma sum-subset-le-1[simp]:
assumes ∗: proper c and ∗∗: I ⊆ {..< brn c}
shows sum (wt c s) I ≤ 1
proof−
 define J where J = {..< brn c}
 have I ⊆ J and finite J using ∗∗ unfolding J-def by auto
 moreover have ∀ j ∈ J. wt c s j ≥ 0 unfolding J-def using ∗ by simp
 ultimately have sum (wt c s) I ≤ sum (wt c s) J
 using sum-mono2[of J I wt c s] by auto
```

also have ... = *1* **using** ∗ **unfolding** *J-def* **by** *simp*
finally show *sum* (*wt c s*) *I* ≤ *1* **unfolding** *J-def* **by** *simp*
qed

lemma *sum-le-1*[*simp*]:
assumes ∗: *proper c* **and** ∗∗: *i* < *brn c*
shows *sum* (*wt c s*) {..*i*} ≤ *1*
proof−
have {..*i*} ⊆ {..< *brn c*} **using** ∗∗ **by** *auto*
thus *?thesis* **using** *assms sum-subset-le-1*[*of c* {..*i*} *s*] **by** *blast*
qed

1.2.2 Operations on configurations

definition *cont-eff cf b* = *snd* (*wt-cont-eff* (*fst cf*) (*snd cf*) *b*)

lemma *cont-eff*: *cont-eff cf b* = (*cont* (*fst cf*) (*snd cf*) *b*, *eff* (*fst cf*) (*snd cf*) *b*)
unfolding *cont-eff-def cont-def eff-def* **by** *simp*

end

end

2 Resumption-Based Noninterference

theory *Resumption-Based*
imports *Language-Semantics*
begin

type-synonym *'a rel* = (*'a* ×*'a*) *set*

2.1 Preliminaries

lemma *int-emp*[*simp*]:
assumes *i* > *0*
shows {..<*i*} ≠ {}
by (*metis assms emptyE lessThan-iff*)

lemma *inj-on-inv-into*[*simp*]:
assumes *inj-on F P*
shows *inv-into P F* ‘ (*F* ‘ *P*) = *P*
using *assms* **by** *auto*

lemma *inj-on-inv-into2*[*simp*]:
inj-on (*inv-into P F*) (*F* ‘ *P*)
by (*metis Hilbert-Choice.inj-on-inv-into subset-refl*)

lemma *refl-gfp*:
assumes *1*: *mono Retr* **and** *2*: $\bigwedge$ *theta*. *refl theta* $\Longrightarrow$ *refl* (*Retr theta*)
shows *refl* (*gfp Retr*)
proof−
define *bis* **where** *bis* = *gfp Retr*
define *th* **where** *th* = *Id Un bis*
have *bis* ⊆ *Retr bis*
using *1* **unfolding** *bis-def* **by** (*metis gfp-unfold subset-refl*)
also have ... ⊆ *Retr th* **using** *1* **unfolding** *mono-def th-def* **by** *auto*
finally have *bis* ⊆ *Retr th* **.**
moreover
{have *refl th* **unfolding** *th-def* **by** (*metis Un-commute refl-reflcl*)
hence *refl* (*Retr th*) **using** *2* **by** *simp*
}
ultimately have *Id* ⊆ *Retr th* **unfolding** *th-def refl-on-def* **by** *auto*
hence *Id* ⊆ *bis* **using** *1 coinduct* **unfolding** *th-def bis-def* **by** *blast*
thus *?thesis* **unfolding** *bis-def refl-on-def* **by** *auto*
qed

lemma *sym-gfp*:
assumes *1*: *mono Retr* **and** *2*: $\bigwedge$ *theta*. *sym theta* $\Longrightarrow$ *sym* (*Retr theta*)
shows *sym* (*gfp Retr*)
proof−
define *bis* **where** *bis* = *gfp Retr*
define *th* **where** *th* = *bis* ^−*1 Un bis*
have *bis* ⊆ *Retr bis*
using *1* **unfolding** *bis-def* **by** (*metis gfp-unfold subset-refl*)
also have ... ⊆ *Retr th* **using** *1* **unfolding** *mono-def th-def* **by** *auto*
finally have *bis* ⊆ *Retr th* **.**
moreover
{have *sym th* **unfolding** *th-def* **by** (*metis Un-commute sym-Un-converse*)
hence *sym* (*Retr th*) **using** *2* **by** *simp*
}
ultimately have *bis* ^−*1* ⊆ *Retr th*
by (*metis Un-absorb2 Un-commute Un-upper1 converse-Un sym-conv-converse-eq*)
hence *bis* ^−*1* ⊆ *bis* **using** *1 coinduct*[*of Retr bis* ^−*1*] **unfolding** *th-def bis-def*
by *blast*
thus *?thesis* **unfolding** *bis-def sym-def* **by** *blast*
qed

lemma *trancl-trans*[*simp*]:
assumes *trans R*
shows *P* ^+ ⊆ *R* ⟷ *P* ⊆ *R*
proof−
{assume *P* ⊆ *R*
hence *P* ^+ ⊆ *R* ^+ **using** *trancl-mono* **by** *auto*
also have *R* ^+ = *R* **using** *assms trans-trancl* **by** *auto*
finally have *P* ^+ ⊆ *R* **.**

```
  }
  thus ?thesis by auto
qed

lemma trans-gfp:
assumes 1: mono Retr and 2: ⋀ theta. trans theta ⟹ trans (Retr theta)
shows trans (gfp Retr)
proof−
  define bis where bis = gfp Retr
  define th where th = bis ^+
  have bis ⊆ Retr bis
  using 1 unfolding bis-def by (metis gfp-unfold subset-refl)
  also have ... ⊆ Retr th using 1 unfolding mono-def th-def
    by (metis trancl-trans order-refl trans-trancl)
  finally have bis ⊆ Retr th .
  moreover
  {have trans th unfolding th-def by (metis th-def trans-trancl)
   hence trans (Retr th) using 2 by simp
  }
  ultimately have bis ^+ ⊆ Retr th by simp
  hence bis ^+ ⊆ bis using 1 coinduct unfolding th-def bis-def
  by (metis bis-def gfp-upperbound th-def)
  thus ?thesis unfolding bis-def trans-def by auto
qed

lemma O-subset-trans:
assumes r O r ⊆ r
shows trans r
using assms unfolding trans-def by blast

lemma trancl-imp-trans:
assumes r ^+ ⊆ r
shows trans r
by (metis Int-absorb1 Int-commute trancl-trans assms subset-refl trans-trancl)

lemma sym-trans-gfp:
assumes 1: mono Retr and 2: ⋀ theta. sym theta ∧ trans theta ⟹ sym (Retr
theta) ∧ trans (Retr theta)
shows sym (gfp Retr) ∧ trans (gfp Retr)
proof−
  define bis where bis = gfp Retr
  define th where th = (bis Un bis ^−1) ^+
  have bis ⊆ Retr bis
  using 1 unfolding bis-def by (metis gfp-unfold subset-refl)
  also have ... ⊆ Retr th using 1 unfolding mono-def th-def
    by (metis inf-sup-absorb le-iff-inf sup-aci(2) trancl-unfold)
  finally have bis ⊆ Retr th .
  hence (bis Un bis ^−1) ^+ ⊆ ((Retr th) Un (Retr th) ^−1) ^+ by auto
  moreover
```

{have *sym th* **unfolding** *th-def* **by** (*metis sym-Un-converse sym-trancl*)
moreover have *trans th* **unfolding** *th-def* **by** (*metis th-def trans-trancl*)
ultimately have *sym* (*Retr th*) ∧ *trans* (*Retr th*) **using** *2* **by** *simp*
hence ((*Retr th*) *Un* (*Retr th*) ^*−1*) ^+ ⊆ *Retr th*
by (*metis Un-absorb subset-refl sym-conv-converse-eq trancl-id*)
}
ultimately have (*bis Un bis* ^*−1*) ^+ ⊆ *Retr th* **by** *blast*
hence (*bis Un bis* ^*−1*) ^+ ⊆ *bis* **using** *1 coinduct* **unfolding** *th-def bis-def*
by (*metis bis-def gfp-upperbound th-def*)
hence *bis* ^*−1* ⊆ *bis* **and** *bis* ^+ ⊆ *bis*
apply(*metis equalityI gfp-upperbound le-supI1 subset-refl sym-Un-converse sym-conv-converse-eq th-def trancl-id trancl-imp-trans*)
by (*metis Un-absorb* ‹(*bis* ∪ *bis*$^{-1}$)$^+$ ⊆ *bis*› *less-supI1 psubset-eq sym-Un-converse sym-conv-converse-eq sym-trancl trancl-id trancl-imp-trans*)
thus *?thesis* **unfolding** *bis-def sym-def* **using** *trancl-imp-trans* **by** *auto*
qed

2.2 Infrastructure for partitions

definition *part* **where**
part J P ≡
Union P = *J* ∧
(∀ *J1 J2*. *J1* ∈ *P* ∧ *J2* ∈ *P* ∧ *J1* ≠ *J2* ⟶ *J1* ∩ *J2* = {})

inductive-set *gen*
for *P* :: *'a set set* **and** *I* :: *'a set* **where**
incl[*simp*]: *i* ∈ *I* ⟹ *i* ∈ *gen P I*
|*ext*[*simp*]: ⟦*J* ∈ *P*; *j0* ∈ *J*; *j0* ∈ *gen P I*; *j* ∈ *J*⟧ ⟹ *j* ∈ *gen P I*

definition *partGen* **where**
partGen P ≡ {*gen P I* | *I* . *I* ∈ *P*}

definition *finer* **where**
finer P Q ≡
(∀ *J* ∈ *Q*. *J* = *Union* {*I* ∈ *P* . *I* ⊆ *J*}) ∧
(*P* ≠ {} ⟶ *Q* ≠ {})

definition *partJoin* **where**
partJoin P Q ≡ *partGen* (*P* ∪ *Q*)

definition *compat* **where**
compat I theta f ≡ ∀ *i j*. {*i*, *j*} ⊆ *I* ∧ *i* ≠ *j* ⟶ (*f i*, *f j*) ∈ *theta*

definition *partCompat* **where**
partCompat P theta f ≡
∀ *I* ∈ *P*. *compat I theta f*

definition *lift* **where**
lift P F II ≡ *Union* {*F I* | *I* . *I* ∈ *P* ∧ *I* ⊆ *II*}

part:

lemma *part-emp*[*simp*]:
part J (*insert* {} *P*) = *part J P*
unfolding *part-def* **by** *auto*

lemma *finite-part*[*simp*]:
assumes *finite I* **and** *part I P*
shows *finite P*
using *assms finite-UnionD* **unfolding** *part-def* **by** *auto*

lemma *part-sum*:
assumes *P*: *part* {..<*n*::*nat*} *P*
shows (∑ *i*<*n*. *f i*) = (∑ *p*∈*P*. ∑ *i*∈*p*. *f i*)
proof (*subst sum.Union-disjoint* [*symmetric*, *simplified*])
show ∀ *p*∈*P*. *finite p*
proof
fix *p* **assume** *p* ∈ *P*
with *P* **have** *p* ⊆ {*0*..<*n*} **by** (*auto simp*: *part-def*)
then show *finite p* **by** (*rule finite-subset*) *simp*
qed
show ∀ *A*∈*P*. ∀ *B*∈*P*. *A* ≠ *B* ⟶ *A* ∩ *B* = {}
using *P* **by** (*auto simp*: *part-def*)
show *sum f* {..<*n*} = *sum f* (⋃ *P*)
using *P* **by** (*auto simp*: *part-def atLeast0LessThan*)
qed

lemma *part-Un*[*simp*]:
assumes *part I1 P1* **and** *part I2 P2* **and** *I1 Int I2* = {}
shows *part* (*I1 Un I2*) (*P1 Un P2*)
using *assms* **unfolding** *part-def*
by (*metis Union-Un-distrib Union-disjoint inf-aci*(*1*) *mem-simps*(*3*))

lemma *part-Un-singl*[*simp*]:
assumes *part K P* **and** ⋀ *I*. *I* ∈ *P* ⟹ *I0 Int I* = {}
shows *part* (*I0 Un K*) ({*I0*} *Un P*)
using *assms* **unfolding** *part-def*
by (*metis complete-lattice-class.Sup-insert Int-commute insert-iff insert-is-Un*)

lemma *part-Un-singl2*:
assumes *K01* = *I0 Un K1*
and *part K1 P* **and** ⋀ *I*. *I* ∈ *P* ⟹ *I0 Int I* = {}

shows *part K01* ($\{I0\}$ *Un P*)
using *assms part-Un-singl* **by** *blast*

lemma *part-UN*:
assumes $\bigwedge$ *n*. *n* $\in$ *N* $\Longrightarrow$ *part* (*I n*) (*P n*)
and $\bigwedge$ *n1 n2*. $\{n1,n2\} \subseteq N \wedge n1 \neq n2 \Longrightarrow I\ n1 \cap I\ n2 = \{\}$
shows *part* (*UN n* : *N*. *I n*) (*UN n* : *N*. *P n*)
using *assms* **unfolding** *part-def* **apply** *auto*
apply (*metis UnionE*)
apply (*metis Union-upper disjoint-iff-not-equal insert-absorb insert-subset*)
by (*metis UnionI disjoint-iff-not-equal*)

gen:

lemma *incl-gen*[*simp*]:
I $\subseteq$ *gen P I*
by *auto*

lemma *gen-incl-Un*:
gen P I $\subseteq$ *I* $\cup$ (*Union P*)
proof
  fix *j* **assume** *j* $\in$ *gen P I*
  thus *j* $\in$ *I* $\cup \bigcup$ *P* **apply** *induct* **by** *blast*+
qed

lemma *gen-incl*:
assumes *I* $\in$ *P*
shows *gen P I* $\subseteq$ *Union P*
using *assms gen-incl-Un*[*of P I*] **by** *blast*

lemma *finite-gen*:
assumes *finite P* **and** $\bigwedge$ *J*. *J* $\in$ *P* $\Longrightarrow$ *finite J* **and** *finite I*
shows *finite* (*gen P I*)
by (*metis assms finite-Union gen-incl-Un infinite-Un infinite-super*)

lemma *subset-gen*[*simp*]:
assumes *J* $\in$ *P* **and** *gen P I* $\cap$ *J* $\neq$ $\{\}$
shows *J* $\subseteq$ *gen P I*
using *assms gen.ext*[*of J P - I*] **by** *blast*

lemma *gen-subset-gen*[*simp*]:
assumes *J* $\in$ *P* **and** *gen P I* $\cap$ *J* $\neq$ $\{\}$
shows *gen P J* $\subseteq$ *gen P I*
proof−
  have *J*: *J* $\subseteq$ *gen P I* **using** *assms* **by** *auto*
  show *?thesis* **proof**
    fix *i* **assume** *i* $\in$ *gen P J*
    thus *i* $\in$ *gen P I*
    proof *induct*
      case (*ext J' j0 j*)

```
      thus ?case
      using gen.ext[of J' P j0 I j] by blast
    qed (insert J, auto)
  qed
qed

lemma gen-mono[simp]:
assumes I ⊆ J
shows gen P I ⊆ gen P J
proof
  fix i assume i ∈ gen P I thus i ∈ gen P J
  proof induct
    case (ext I' j0 j)
    thus ?case
    using gen.ext[of I' P j0 J j] by blast
  qed (insert assms, auto)
qed

lemma gen-idem[simp]:
gen P (gen P I) = gen P I
proof−
  define J where J = gen P I
  have I ⊆ J unfolding J-def by auto
  hence gen P I ⊆ gen P J by simp
  moreover have gen P J ⊆ gen P I
  proof
    fix i assume i ∈ gen P J
    thus i ∈ gen P I
    proof induct
      case (ext J' j0 j)
      thus ?case
        using gen.ext[of J' P j0 I j] by blast
    qed (unfold J-def, auto)
  qed
  ultimately show ?thesis unfolding J-def by auto
qed

lemma gen-nchotomy:
assumes J ∈ P
shows J ⊆ gen P I ∨ gen P I ∩ J = {}
using assms subset-gen[of J P I] by blast

lemma gen-Union:
assumes I ∈ P
shows gen P I = Union {J ∈ P . J ⊆ gen P I}
proof safe
  fix i assume i: i ∈ gen P I
  then obtain J where J: J ∈ P  i ∈ J using assms gen-incl by blast
  hence J ⊆ gen P I using assms i gen-nchotomy by auto
```

thus $i \in \bigcup \{J \in P.\ J \subseteq gen\ P\ I\}$ **using** J **by** *auto*
qed *auto*

lemma *subset-gen2*:
assumes $*$: $\{I,J\} \subseteq P$ **and** $**$: $gen\ P\ I \cap gen\ P\ J \neq \{\}$
shows $I \subseteq gen\ P\ J$
proof−
{fix *i0 i* **assume** *i0*: $i0 \in I \wedge i0 \notin gen\ P\ J$
assume $i \in gen\ P\ I$
hence $i \notin gen\ P\ J$
proof *induct*
case (*incl i*)
thus *?case* **using** *i0 gen-nchotomy*[*of I P J*] $*$ **by** *blast*
next
case (*ext I′ j0 j*)
thus *?case*
using *gen.ext*[*of I′ P j0 J j*] *gen-nchotomy*[*of I′ P J*] **by** *blast*
qed
}
thus *?thesis* **using** *assms* **by** *auto*
qed

lemma *gen-subset-gen2*[*simp*]:
assumes $\{I,J\} \subseteq P$ **and** $gen\ P\ I \cap gen\ P\ J \neq \{\}$
shows $gen\ P\ I \subseteq gen\ P\ J$
proof
fix i **assume** $i \in gen\ P\ I$
thus $i \in gen\ P\ J$
proof *induct*
case (*incl i*)
thus *?case*
using *assms subset-gen2* **by** *auto*
next
case (*ext I′ j0 j*)
thus *?case*
using *gen.ext*[*of I′ P j0 J j*] **by** *blast*
qed
qed

lemma *gen-eq-gen*:
assumes $\{I,J\} \subseteq P$ **and** $gen\ P\ I \cap gen\ P\ J \neq \{\}$
shows $gen\ P\ I = gen\ P\ J$
using *assms gen-subset-gen2*[*of I J P*] *gen-subset-gen2*[*of J I P*] **by** *blast*

lemma *gen-empty*[*simp*]:
$gen\ P\ \{\} = \{\}$
proof−
{fix j **assume** $j \in gen\ P\ \{\}$ **hence** *False*
apply *induct* **by** *auto*

}
thus *?thesis* **by** *auto*
qed

lemma *gen-empty2*[*simp*]:
gen {} *I* = *I*
proof−
{fix *j* **assume** *j* ∈ *gen* {} *I* **hence** *j* ∈ *I*
apply *induct* **by** *auto*
}
thus *?thesis* **by** *auto*
qed

lemma *emp-gen*[*simp*]:
assumes *gen P I* = {}
shows *I* = {}
by (*metis all-not-in-conv assms gen.incl*)

partGen:

lemma *partGen-ex*:
assumes *I* ∈ *P*
shows ∃ *J* ∈ *partGen P*. *I* ⊆ *J*
using *assms* **unfolding** *partGen-def*
apply(*intro bexI*[*of - gen P I*]) **by** *auto*

lemma *ex-partGen*:
assumes *J* ∈ *partGen P* **and** *j*: *j* ∈ *J*
shows ∃ *I* ∈ *P*. *j* ∈ *I*
proof−
obtain *I0* **where** *I0*: *I0* ∈ *P* **and** *J*: *J* = *gen P I0*
using *assms* **unfolding** *partGen-def* **by** *auto*
thus *?thesis* **using** *j gen-incl*[*of I0 P*] **by** *auto*
qed

lemma *Union-partGen*: ⋃ (*partGen P*) = ⋃ *P*
using *ex-partGen*[*of - P*] *partGen-ex*[*of - P*] **by** *fastforce*

lemma *Int-partGen*:
assumes ∗: {*I*,*J*} ⊆ *partGen P* **and** ∗∗: *I* ∩ *J* ≠ {}
shows *I* = *J*
proof−
obtain *I0* **where** *I0*: *I0* ∈ *P* **and** *I*: *I* = *gen P I0*
using *assms* **unfolding** *partGen-def* **by** *auto*
obtain *J0* **where** *J0*: *J0* ∈ *P* **and** *J*: *J* = *gen P J0*
using *assms* **unfolding** *partGen-def* **by** *auto*
show *?thesis* **using** *gen-eq-gen*[*of I0 J0 P*] *I0 J0* ∗∗ **unfolding** *I J* **by** *blast*
qed

lemma *part-partGen*:

part (*Union P*) (*partGen P*)
unfolding *part-def* **apply**(*intro conjI allI impI*)
apply (*metis Union-partGen*)
using *Int-partGen* **by** *blast*

lemma *finite-partGen*[*simp*]:
assumes *finite P*
shows *finite* (*partGen P*)
using *assms* **unfolding** *partGen-def* **by** *auto*

lemma *emp-partGen*[*simp*]:
assumes $\{\} \notin P$
shows $\{\} \notin$ *partGen P*
using *assms* **unfolding** *partGen-def* **using** *emp-gen*[*of P*] **by** *blast*

finer:

lemma *finer-partGen*:
finer P (*partGen P*)
unfolding *finer-def partGen-def* **using** *gen-Union* **by** *auto*

lemma *finer-nchotomy*:
assumes *P*: *part I0 P* **and** *Q*: *part I0 Q* **and** *PQ*: *finer P Q*
and *I*: $I \in P$ **and** *II*: $II \in Q$
shows $I \subseteq II \vee (I \cap II = \{\})$
proof(*cases* $I \cap II = \{\}$)
  case *False*
  then obtain *i* **where** *i*: $i \in I \wedge i \in II$ **by** *auto*
  then obtain I' **where** $i \in I'$ **and** I': $I' \in P \wedge I' \subseteq II$
  using *PQ II* **unfolding** *finer-def* **by** *blast*
  hence *I Int* $I' \neq \{\}$ **using** *i* **by** *blast*
  hence $I = I'$ **using** *I* I' *P* **unfolding** *part-def* **by** *auto*
  hence $I \subseteq II$ **using** I' **by** *simp*
  thus *?thesis* **by** *auto*
qed *auto*

lemma *finer-ex*:
assumes *P*: *part I0 P* **and** *Q*: *part I0 Q* **and** *PQ*: *finer P Q*
and *I*: $I \in P$
shows $\exists\ II.\ II \in Q \wedge I \subseteq II$
proof(*cases* $I = \{\}$)
  case *True*
  have $Q \neq \{\}$ **using** *I PQ* **unfolding** *finer-def* **by** *auto*
  then obtain *JJ* **where** $JJ \in Q$ **by** *auto*
  with *True* **show** *?thesis* **by** *blast*
next
  case *False*
  then obtain *i* **where** *i*: $i \in I$ **by** *auto*
  hence $i \in I0$ **using** *I P* **unfolding** *part-def* **by** *auto*
  then obtain *II* **where** *II*: $II \in Q$ **and** $i \in II$ **using** *Q* **unfolding** *part-def* **by**

auto
hence *I Int II* $\neq \{\}$ **using** *i* **by** *auto*
thus *?thesis* **using** *assms I II finer-nchotomy*[*of I0 P Q I II*] **by** *auto*
qed

partJoin:

lemma *partJoin-commute*:
partJoin P Q = *partJoin Q P*
unfolding *partJoin-def partGen-def*
using *Un-commute* **by** *metis*

lemma *Union-partJoin-L*:
Union P $\subseteq$ *Union* (*partJoin P Q*)
unfolding *partJoin-def partGen-def* **by** *auto*

lemma *Union-partJoin-R*:
Union Q $\subseteq$ *Union* (*partJoin P Q*)
unfolding *partJoin-def partGen-def* **by** *auto*

lemma *part-partJoin*[*simp*]:
assumes *part I P* **and** *part I Q*
shows *part I* (*partJoin P Q*)
proof−
have *1*: *Union* (*P Un Q*) = *I*
using *assms* **unfolding** *part-def* **by** *auto*
show *?thesis* **using** *part-partGen*[*of P Un Q*]
unfolding *1 partJoin-def* **by** *auto*
qed

lemma *finer-partJoin-L*[*simp*]:
assumes ∗: *part I P* **and** ∗∗: *part I Q*
shows *finer P* (*partJoin P Q*)
proof−
have *1*: *part I* (*partJoin P Q*) **using** *assms* **by** *simp*
{fix *J j* **assume** *J*: *J* $\in$ *partJoin P Q* **and** *j*: *j* $\in$ *J*
hence *J* $\subseteq$ *I* **using** *1* **by** (*metis Union-upper part-def*)
with *j* **have** *j* $\in$ *I* **by** *auto*
then obtain *J′* **where** *jJ′*: *j* $\in$ *J′* **and** *J′*: *J′* $\in$ *P*
using ∗ **unfolding** *part-def* **by** *auto*
hence *J* $\cap$ *J′* $\neq \{\}$ **using** *j* **by** *auto*
moreover obtain *J0* **where** *J* = *gen* (*P Un Q*) *J0*
and *J0* $\in$ *P Un Q*
using *J* **unfolding** *partJoin-def partGen-def* **by** *blast*
ultimately have *J′* $\subseteq$ *J*
using *J′ gen-nchotomy*[*of J′ P Un Q J0*] **by** *blast*
hence *j* $\in \bigcup \{J' \in P.\ J' \subseteq J\}$ **using** *J′ jJ′* **by** *blast*
}
thus *?thesis* **unfolding** *finer-def*
unfolding *partJoin-def partGen-def* **by** *blast*

qed

lemma *finer-partJoin-R*[*simp*]:
assumes ∗: *part I P* **and** ∗∗: *part I Q*
shows *finer Q* (*partJoin P Q*)
using *assms finer-partJoin-L*[*of I Q P*] *partJoin-commute*[*of P Q*] **by** *auto*

lemma *finer-emp*[*simp*]:
assumes *finer* {} *Q*
shows *Q* ⊆ { {} }
using *assms* **unfolding** *finer-def* **by** *auto*

compat:

lemma *part-emp-R*[*simp*]:
part I {} ⟷ *I* = {}
unfolding *part-def* **by** *auto*

lemma *part-emp-L*[*simp*]:
part {} *P* ⟹ *P* ⊆ { {} }
unfolding *part-def* **by** *auto*

lemma *finite-partJoin*[*simp*]:
assumes *finite P* **and** *finite Q*
shows *finite* (*partJoin P Q*)
using *assms* **unfolding** *partJoin-def* **by** *auto*

lemma *emp-partJoin*[*simp*]:
assumes {} ∉ *P* **and** {} ∉ *Q*
shows {} ∉ *partJoin P Q*
using *assms* **unfolding** *partJoin-def* **by** *auto*

partCompat:

lemma *partCompat-Un*[*simp*]:
partCompat (*P Un Q*) *theta f* ⟷
partCompat P theta f ∧ *partCompat Q theta f*
unfolding *partCompat-def* **by** *auto*

lemma *partCompat-gen-aux*:
assumes *theta*: *sym theta trans theta*
and *fP*: *partCompat P theta f* **and** *I*: *I* ∈ *P*
and *i*: *i* ∈ *I* **and** *j*: *j* ∈ *gen P I* **and** *ij*: *i* ≠ *j*
shows (*f i*, *f j*) ∈ *theta*
using *j ij* **proof** *induct*
case (*incl j*)
thus *?case*
using *fP I i* **unfolding** *partCompat-def compat-def* **by** *blast*
next
case (*ext J j0 j*)
show *?case*

```
  proof(cases i = j0)
    case False note case-i = False
    hence 1: (f i, f j0) ∈ theta using ext by blast
    show ?thesis
    proof(cases j = j0)
      case True
      thus ?thesis using case-i 1 by simp
    next
      case False
      hence (f j, f j0) ∈ theta using ‹j0 ∈ J› ‹j ∈ J› ‹J ∈ P›
      using fP unfolding partCompat-def compat-def by auto
      hence (f j0, f j) ∈ theta using theta unfolding sym-def by simp
      thus ?thesis using 1 theta unfolding trans-def by blast
    qed
  next
    case True note case-i = True
    hence j0 ≠ j using ‹i ≠ j› by auto
    hence (f j0, f j) ∈ theta using ‹j0 ∈ J› ‹j ∈ J› ‹J ∈ P›
    using fP unfolding partCompat-def compat-def by auto
    thus ?thesis unfolding case-i .
  qed
qed

lemma partCompat-gen:
assumes theta: sym theta trans theta
and fP: partCompat P theta f and I: I ∈ P
shows compat (gen P I) theta f
unfolding compat-def proof clarify
  fix i j assume ij: {i, j} ⊆ gen P I  i ≠ j
  show (f i, f j) ∈ theta
  proof(cases i ∈ I)
    case True note i = True
    show ?thesis
    proof(cases j ∈ I)
      case True
      thus ?thesis using i ij I fP unfolding partCompat-def compat-def by blast
    next
      case False
      hence i ≠ j using i by auto
      thus ?thesis using assms partCompat-gen-aux i ij by auto
    qed
  next
    case False note i = False
    show ?thesis
    proof(cases j ∈ I)
      case True
      hence j ≠ i using i by auto
       hence (f j, f i) ∈ theta using assms partCompat-gen-aux[of theta P f I j i]
True ij by auto
```

thus *?thesis* **using** *theta* **unfolding** *sym-def* **by** *auto*
next
case *False* **note** $j = False$
show *?thesis*
proof(*cases* $I = \{\}$)
case *True*
hence *False* **using** *ij* **by** *simp*
thus *?thesis* **by** *simp*
next
case *False*
then obtain *i0* **where** *i0*: $i0 \in I$ **by** *auto*
hence *i0-not*: $i0 \notin \{i,j\}$ **using** *i j* **by** *auto*
have $(f\ i0, f\ i) \in theta$
using *assms i0 i0-not ij partCompat-gen-aux*[*of theta P f I i0 i*] **by** *blast*
hence $(f\ i, f\ i0) \in theta$ **using** *theta* **unfolding** *sym-def* **by** *auto*
moreover have $(f\ i0, f\ j) \in theta$
using *assms i0 i0-not ij partCompat-gen-aux*[*of theta P f I i0 j*] **by** *blast*
ultimately show *?thesis* **using** *theta* **unfolding** *trans-def* **by** *blast*
qed
qed
qed
qed

lemma *partCompat-partGen*:
assumes *sym theta* **and** *trans theta*
and *partCompat P theta f*
shows *partCompat* (*partGen P*) *theta f*
unfolding *partCompat-def partGen-def*
using *assms partCompat-gen*[*of theta P f*] **by** *auto*

lemma *partCompat-partJoin*[*simp*]:
assumes *sym theta* **and** *trans theta*
and *partCompat P theta f* **and** *partCompat Q theta f*
shows *partCompat* (*partJoin P Q*) *theta f*
by (*metis assms partCompat-Un partCompat-partGen partJoin-def*)

lift:

lemma *inj-on-lift*:
assumes *P*: *part I0 P* **and** *Q*: *part I0 Q* **and** *PQ*: *finer P Q*
and *F*: *inj-on F P* **and** *FP*: *part J0* (*F ' P*) **and** *emp*: $\{\} \notin F\ `\ P$
shows *inj-on* (*lift P F*) *Q*
unfolding *inj-on-def* **proof** *clarify*
fix *II II'* **assume** *II*: $II \in Q$ **and** *II'*: $II' \in Q$ **and** *eq*: *lift P F II* = *lift P F II'*
have *1*: *II* = *Union* $\{I \in P\ .\ I \subseteq II\}$ **using** *PQ II* **unfolding** *finer-def* **by** *auto*
have *2*: *II'* = *Union* $\{I \in P\ .\ I \subseteq II'\}$ **using** *PQ II'* **unfolding** *finer-def* **by** *auto*
{fix *I*
assume *I*: $I \in P$ $I \subseteq II$
hence *F I* $\subseteq$ *lift P F II* **unfolding** *lift-def*[*abs-def*] **by** *blast*

hence *3*: $F\ I \subseteq \mathit{lift}\ P\ F\ II'$ **unfolding** *eq* **.**
have $F\ I \neq \{\}$ **using** *emp I FP* **by** *auto*
then obtain j **where** j: $j \in F\ I$ **by** *auto*
with *3* **obtain** I' **where** I': $I' \in P \wedge I' \subseteq II'$ **and** $j \in F\ I'$ **unfolding** *lift-def* [*abs-def*] **by** *auto*
hence $F\ I\ \mathit{Int}\ F\ I' \neq \{\}$ **using** j **by** *auto*
hence $F\ I = F\ I'$ **using** *FP I I'* **unfolding** *part-def* **by** *auto*
hence $I = I'$ **using** $F\ I\ I'$ **unfolding** *inj-on-def* **by** *auto*
hence $I \subseteq II'$ **using** I' **by** *auto*
}
hence *a*: $II \subseteq II'$ **using** *1 2* **by** *blast*

{fix I
assume I: $I \in P\ \ I \subseteq II'$
hence $F\ I \subseteq \mathit{lift}\ P\ F\ II'$ **unfolding** *lift-def* [*abs-def*] **by** *blast*
hence *3*: $F\ I \subseteq \mathit{lift}\ P\ F\ II$ **unfolding** *eq* **.**
have $F\ I \neq \{\}$ **using** *emp I FP* **by** *auto*
then obtain j **where** j: $j \in F\ I$ **by** *auto*
with *3* **obtain** I' **where** I': $I' \in P \wedge I' \subseteq II$ **and** $j \in F\ I'$ **unfolding** *lift-def* [*abs-def*] **by** *auto*
hence $F\ I\ \mathit{Int}\ F\ I' \neq \{\}$ **using** j **by** *auto*
hence $F\ I = F\ I'$ **using** *FP I I'* **unfolding** *part-def* **by** *auto*
hence $I = I'$ **using** $F\ I\ I'$ **unfolding** *inj-on-def* **by** *auto*
hence $I \subseteq II$ **using** I' **by** *auto*
}
hence $II' \subseteq II$ **using** *1 2* **by** *blast*
with *a* **show** $II = II'$ **by** *auto*
qed

lemma *part-lift*:
assumes *P*: *part I0 P* **and** *Q*: *part I0 Q* **and** *PQ*: *finer P Q*
and *F*: *inj-on F P* **and** *FP*: *part J0* $(F\ `\ P)$ **and** *emp*: $\{\} \notin P\ \{\} \notin F\ `\ P$
shows *part J0* $(\mathit{lift}\ P\ F\ `\ Q)$
unfolding *part-def* **proof** (*intro conjI allI impI*)
show $\bigcup(\mathit{lift}\ P\ F\ `\ Q) = J0$
proof *safe*
fix *j II* **assume** $j \in \mathit{lift}\ P\ F\ II$ **and** *II*: $II \in Q$
then obtain I **where** $j \in F\ I$ **and** $I \in P$ **and** $I \subseteq II$
unfolding *lift-def* **by** *auto*
thus $j \in J0$ **using** *FP* **unfolding** *part-def* **by** *auto*
next
fix j **assume** $j \in J0$
then obtain J **where** J: $J \in F\ `\ P$ **and** j: $j \in J$ **using** *FP* **unfolding** *part-def* **by** *auto*
define I **where** $I = \mathit{inv\text{-}into}\ P\ F\ J$
have j: $j \in F\ I$ **unfolding** *I-def* **using** $j\ J\ F$ **by** *auto*
have I: $I \in P$ **unfolding** *I-def* **using** $F\ J$ **by** *auto*
obtain *II* **where** $I \subseteq II$ **and** $II \in Q$ **using** *P Q PQ I finer-ex*[*of I0 P Q I*] **by** *auto*

thus $j \in \bigcup$ (*lift P F ‘ Q*) **unfolding** *lift-def* [*abs-def*] **using** *j I* **by** *auto*
qed
next
fix *JJ1 JJ2* **assume** *JJ12*: *JJ1* $\in$ *lift P F ‘ Q* $\wedge$ *JJ2* $\in$ *lift P F ‘ Q* $\wedge$ *JJ1* $\neq$ *JJ2*
then obtain *II1 II2* **where** *II12*: {*II1*,*II2*} $\subseteq$ *Q* **and** *JJ1*: *JJ1* = *lift P F II1*
and *JJ2*: *JJ2* = *lift P F II2* **by** *auto*
have *II1* $\neq$ *II2* **using** *JJ12* **unfolding** *JJ1 JJ2* **using** *II12 assms*
using *inj-on-lift*[*of I0 P Q F*] **by** *auto*
hence *4*: *II1 Int II2* = {} **using** *II12 Q* **unfolding** *part-def* **by** *auto*
show *JJ1* $\cap$ *JJ2* = {}
proof(*rule ccontr*)
assume *JJ1* $\cap$ *JJ2* $\neq$ {}
then obtain *j* **where** *j*: *j* $\in$ *JJ1 j* $\in$ *JJ2* **by** *auto*
from *j* **obtain** *I1* **where** *j* $\in$ *F I1* **and** *I1*: *I1* $\in$ *P I1* $\subseteq$ *II1*
unfolding *JJ1 lift-def* [*abs-def*] **by** *auto*
moreover from *j* **obtain** *I2* **where** *j* $\in$ *F I2* **and** *I2*: *I2* $\in$ *P I2* $\subseteq$ *II2*
unfolding *JJ2 lift-def* [*abs-def*] **by** *auto*
ultimately have *F I1 Int F I2* $\neq$ {} **by** *blast*
hence *F I1* = *F I2* **using** *I1 I2 FP* **unfolding** *part-def* **by** *blast*
hence *I1* = *I2* **using** *I1 I2 F* **unfolding** *inj-on-def* **by** *auto*
moreover have *I1* $\neq$ {} **using** *I1 emp* **by** *auto*
ultimately have *II1 Int II2* $\neq$ {} **using** *I1 I2* **by** *auto*
thus *False* **using** *4* **by** *simp*
qed
qed

lemma *finer-lift*:
assumes *finer P Q*
shows *finer* (*F ‘ P*) (*lift P F ‘ Q*)
unfolding *finer-def* **proof** (*intro conjI ballI impI*)
fix *JJ* **assume** *JJ*: *JJ* $\in$ *lift P F ‘ Q*
show *JJ* = $\bigcup$ {*J* $\in$ *F ‘ P*. *J* $\subseteq$ *JJ*}
proof *safe*
fix *j* **assume** *j*: *j* $\in$ *JJ*
obtain *II* **where** *II*: *II* $\in$ *Q* **and** *JJ*: *JJ* = *lift P F II* **using** *JJ* **by** *auto*
then obtain *I* **where** *j*: *j* $\in$ *F I* **and** *I*: *I* $\in$ *P* $\wedge$ *I* $\subseteq$ *II* **and** *F I* $\subseteq$ *JJ*
using *j* **unfolding** *lift-def* [*abs-def*] **by** *auto*
thus *j* $\in \bigcup$ {*J* $\in$ *F ‘ P*. *J* $\subseteq$ *JJ*} **using** *I j* **by** *auto*
qed *auto*
next
assume *F ‘ P* $\neq$ {}
thus *lift P F ‘ Q* $\neq$ {}
using *assms* **unfolding** *lift-def* [*abs-def*] *finer-def* **by** *simp*
qed

2.3 Basic setting for bisimilarity

locale *PL-Indis* =
PL aval tval cval

for *aval* :: *'atom* ⇒ *'state* ⇒ *'state* **and**
tval :: *'test* ⇒ *'state* ⇒ *bool* **and**
cval :: *'choice* ⇒ *'state* ⇒ *real* +
fixes
indis :: *'state rel*
assumes
equiv-indis: *equiv UNIV indis*

context *PL-Indis*
begin

no-notation *eqpoll* (**infixl** ‹≈› *50*)

abbreviation *indisAbbrev* (**infix** ‹≈› *50*)
where *s1* ≈ *s2* ≡ (*s1*, *s2*) ∈ *indis*

lemma *refl-indis*: *refl indis*
and *trans-indis*: *trans indis*
and *sym-indis*: *sym indis*
using *equiv-indis* **unfolding** *equiv-def* **by** *auto*

lemma *indis-refl*[*intro*]: *s* ≈ *s*
using *refl-indis* **unfolding** *refl-on-def* **by** *simp*

lemma *indis-trans*[*trans*]: ⟦*s* ≈ *s'*; *s'* ≈ *s''*⟧ ⟹ *s* ≈ *s''*
using *trans-indis* **unfolding** *trans-def* **by** *blast*

lemma *indis-sym*[*sym*]: *s* ≈ *s'* ⟹ *s'* ≈ *s*
using *sym-indis* **unfolding** *sym-def* **by** *blast*

2.4 Discreetness

coinductive *discr* **where**
intro:
(⋀ *s i*. *i* < *brn c* ⟶ *s* ≈ *eff c s i* ∧ *discr* (*cont c s i*))
⟹ *discr c*

definition *discrL* **where**
discrL cl ≡ ∀ *c* ∈ *set cl*. *discr c*

lemma *discrL-intro*[*intro*]:
assumes ⋀ *c*. *c* ∈ *set cl* ⟹ *discr c*
shows *discrL cl*
using *assms* **unfolding** *discrL-def* **by** *auto*

lemma *discrL-discr*[*simp*]:
assumes *discrL cl* **and** *c* ∈ *set cl*

shows *discr c*
using *assms* **unfolding** *discrL-def* **by** *simp*

lemma *discrL-update*[*simp*]:
assumes *cl*: *discrL cl* **and** *c'*: *discr c'*
shows *discrL* (*cl*[*n* := *c'*])
proof(*cases n* < *length cl*)
  case *True*
  show *?thesis*
  unfolding *discrL-def* **proof** *safe*
    fix *c* **assume** *c* ∈ *set* (*cl*[*n* := *c'*])
    hence *c* ∈ *insert c'* (*set cl*) **using** *set-update-subset-insert* **by** *fastforce*
    thus *discr c* **using** *assms* **by** (*cases c* = *c'*) *auto*
  qed
qed (*use cl* **in** ‹*auto simp*: *list-update-beyond*›)

Coinduction for discreteness:

lemma *discr-coind*[*consumes 1*, *case-names Hyp*, *induct pred*: *discr*]:
assumes ∗: *phi c* **and**
∗∗: ⋀ *c s i*. ⟦ *phi c* ; *i* < *brn c*⟧
    ⟹ *s* ≈ *eff c s i* ∧ (*phi* (*cont c s i*) ∨ *discr* (*cont c s i*))
shows *discr c*
  by (*metis assms discr.coinduct*)

lemma *discr-raw-coind*[*consumes 1*, *case-names Hyp*]:
assumes ∗: *phi c* **and**
∗∗: ⋀ *c s i*. ⟦*i* < *brn c*; *phi c*⟧ ⟹ *s* ≈ *eff c s i* ∧ *phi* (*cont c s i*)
shows *discr c*
  by (*metis assms discr-coind*)

Discreetness versus transition:

lemma *discr-cont*[*simp*]:
assumes ∗: *discr c* **and** ∗∗: *i* < *brn c*
shows *discr* (*cont c s i*)
  by (*meson assms discr.cases*)

lemma *discr-eff-indis*[*simp*]:
assumes ∗: *discr c* **and** ∗∗: *i* < *brn c*
shows *s* ≈ *eff c s i*
  by (*meson assms discr.cases*)

2.5 Self-isomorphism

coinductive *siso* **where**
intro:
⟦⋀ *s i*. *i* < *brn c* ⟹ *siso* (*cont c s i*);
 ⋀ *s t i*.
   i < *brn c* ∧ *s* ≈ *t* ⟹
   eff c s i ≈ *eff c t i* ∧ *wt c s i* = *wt c t i* ∧ *cont c s i* = *cont c t i*⟧

$\Longrightarrow$ *siso c*

definition *sisoL* **where**
sisoL cl $\equiv \forall$ *c* $\in$ *set cl*. *siso c*

lemma *sisoL-intro*[*intro*]:
assumes $\bigwedge$ *c*. *c* $\in$ *set cl* $\Longrightarrow$ *siso c*
shows *sisoL cl*
using *assms* **unfolding** *sisoL-def* **by** *auto*

lemma *sisoL-siso*[*simp*]:
assumes *sisoL cl* **and** *c* $\in$ *set cl*
shows *siso c*
using *assms* **unfolding** *sisoL-def* **by** *simp*

lemma *sisoL-update*[*simp*]:
assumes *cl*: *sisoL cl* **and** *c'*: *siso c'*
shows *sisoL* (*cl*[*n* := *c'*])
proof(*cases n* < *length cl*)
case *True*
show *?thesis*
unfolding *sisoL-def* **proof** *safe*
fix *c* **assume** *c* $\in$ *set* (*cl*[*n* := *c'*])
hence *c* $\in$ *insert c'* (*set cl*) **using** *set-update-subset-insert* **by** *fastforce*
thus *siso c* **using** *assms* **by** (*cases c* = *c'*) *auto*
qed
qed (*use cl* **in** ‹*auto simp*: *list-update-beyond*›)

Coinduction for self-isomorphism:

lemma *siso-coind*[*consumes 1*, *case-names Obs Cont*, *induct pred*: *siso*]:
assumes $*$: *phi c* **and**
$**$: $\bigwedge$ *c s t i*. ⟦*i* < *brn c*; *phi c*; *s* $\approx$ *t*⟧ $\Longrightarrow$
eff c s i $\approx$ *eff c t i* $\wedge$ *wt c s i* = *wt c t i* $\wedge$ *cont c s i* = *cont c t i* **and**
$***$: $\bigwedge$ *c s i*. ⟦*i* < *brn c*; *phi c*⟧ $\Longrightarrow$ *phi* (*cont c s i*) $\vee$ *siso* (*cont c s i*)
shows *siso c*
using $*$ **apply**(*induct rule*: *siso.coinduct*) **using** $**$ $***$ **by** *auto*

lemma *siso-raw-coind*[*consumes 1*, *case-names Obs Cont*]:
assumes $*$: *phi c* **and**
$***$: $\bigwedge$ *c s t i*. ⟦*i* < *brn c*; *phi c*; *s* $\approx$ *t*⟧ $\Longrightarrow$
eff c s i $\approx$ *eff c t i* $\wedge$ *wt c s i* = *wt c t i* $\wedge$ *cont c s i* = *cont c t i* **and**
$**$: $\bigwedge$ *c s i*. ⟦*i* < *brn c*; *phi c*⟧ $\Longrightarrow$ *phi* (*cont c s i*)
shows *siso c*
using *assms siso-coind* **by** *auto*

Self-Isomorphism versus transition:

lemma *siso-cont*[*simp*]:
assumes $*$: *siso c* **and** $**$: *i* < *brn c*
shows *siso* (*cont c s i*)

by (*meson assms siso.cases*)

lemma *siso-cont-indis*[*simp*]:
assumes ∗: *siso c* **and** ∗∗: *s* ≈ *t i* < *brn c*
shows *eff c s i* ≈ *eff c t i* ∧ *wt c s i* = *wt c t i* ∧ *cont c s i* = *cont c t i*
by (*meson assms siso.cases*)

2.6 Notions of bisimilarity

Matchers

definition *mC-C-part* **where**
mC-C-part c d P F ≡
{} ∉ *P* ∧ {} ∉ *F* ‘ *P* ∧
part {..< *brn c*} *P* ∧ *part* {..< *brn d*} (*F* ‘ *P*)

definition *mC-C-wt* **where**
mC-C-wt c d s t P F ≡ ∀ *I* ∈ *P*. *sum* (*wt c s*) *I* = *sum* (*wt d t*) (*F I*)

definition *mC-C-eff-cont* **where**
mC-C-eff-cont theta c d s t P F ≡
∀ *I i j*.
I ∈ *P* ∧ *i* ∈ *I* ∧ *j* ∈ *F I* ⟶
eff c s i ≈ *eff d t j* ∧ (*cont c s i*, *cont d t j*) ∈ *theta*

definition *mC-C* **where**
mC-C theta c d s t P F ≡
mC-C-part c d P F ∧ *inj-on F P* ∧ *mC-C-wt c d s t P F* ∧ *mC-C-eff-cont theta c d s t P F*

definition *matchC-C* **where**
matchC-C theta c d ≡ ∀ *s t*. *s* ≈ *t* ⟶ (∃ *P F*. *mC-C theta c d s t P F*)

definition *mC-ZOC-part* **where**
mC-ZOC-part c d s t I0 P F ≡
{} ∉ *P* − {*I0*} ∧ {} ∉ *F* ‘ (*P* − {*I0*}) ∧ *I0* ∈ *P* ∧
part {..< *brn c*} *P* ∧ *part* {..< *brn d*} (*F* ‘ *P*)

definition *mC-ZOC-wt* **where**
mC-ZOC-wt c d s t I0 P F ≡
sum (*wt c s*) *I0* < *1* ∧ *sum* (*wt d t*) (*F I0*) < *1* ⟶
(∀ *I* ∈ *P* − {*I0*}.
sum (*wt c s*) *I* / (*1* − *sum* (*wt c s*) *I0*) =
sum (*wt d t*) (*F I*) / (*1* − *sum* (*wt d t*) (*F I0*)))

definition *mC-ZOC-eff-cont0* **where**
mC-ZOC-eff-cont0 theta c d s t I0 F ≡
(∀ *i* ∈ *I0*. *s* ≈ *eff c s i* ∧ (*cont c s i*, *d*) ∈ *theta*) ∧

$(\forall\ j \in F\ I0.\ t \approx \mathit{eff}\ d\ t\ j \wedge (c, \mathit{cont}\ d\ t\ j) \in \mathit{theta})$

definition *mC-ZOC-eff-cont* **where**
mC-ZOC-eff-cont theta c d s t I0 P F ≡
∀ *I i j*.
I ∈ *P* − {*I0*} ∧ *i* ∈ *I* ∧ *j* ∈ *F I* ⟶
eff c s i ≈ *eff d t j* ∧
(*cont c s i*, *cont d t j*) ∈ *theta*

definition *mC-ZOC* **where**
mC-ZOC theta c d s t I0 P F ≡
mC-ZOC-part c d s t I0 P F ∧
inj-on F P ∧
mC-ZOC-wt c d s t I0 P F ∧
mC-ZOC-eff-cont0 theta c d s t I0 F ∧
mC-ZOC-eff-cont theta c d s t I0 P F

definition *matchC-LC* **where**
matchC-LC theta c d ≡
∀ *s t*. *s* ≈ *t* ⟶ (∃ *I0 P F*. *mC-ZOC theta c d s t I0 P F*)

lemmas *m-defs* = *mC-C-def mC-ZOC-def*

lemmas *m-defsAll* =
mC-C-def mC-C-part-def mC-C-wt-def mC-C-eff-cont-def
mC-ZOC-def mC-ZOC-part-def mC-ZOC-wt-def mC-ZOC-eff-cont0-def mC-ZOC-eff-cont-def

lemmas *match-defs* =
matchC-C-def matchC-LC-def

lemma *mC-C-mono*:
assumes *mC-C theta c d s t P F* **and** *theta* ⊆ *theta′*
shows *mC-C theta′ c d s t P F*
using *assms* **unfolding** *m-defsAll* **by** *fastforce+*

lemma *matchC-C-mono*:
assumes *matchC-C theta c d* **and** *theta* ⊆ *theta′*
shows *matchC-C theta′ c d*
using *assms mC-C-mono* **unfolding** *matchC-C-def* **by** *blast*

lemma *mC-ZOC-mono*:
assumes *mC-ZOC theta c d s t I0 P F* **and** *theta* ⊆ *theta′*
shows *mC-ZOC theta′ c d s t I0 P F*
using *assms* **unfolding** *m-defsAll subset-eq* **by** *auto*

lemma *matchC-LC-mono*:
assumes *matchC-LC theta c d* **and** *theta* ⊆ *theta′*
shows *matchC-LC theta′ c d*
using *assms mC-ZOC-mono* **unfolding** *matchC-LC-def*

by *metis*

lemma *Int-not-in-eq-emp*:
$P \cap \{I.\ I \notin P\} = \{\}$
by *blast*

lemma *mC-C-mC-ZOC*:
assumes *mC-C theta c d s t P F*
shows *mC-ZOC theta c d s t* {} (*P Un* { {} }) (%*I. if I* ∈ *P then F I else* {})
(**is** *mC-ZOC theta c d s t ?I0 ?Q ?G*)
unfolding *mC-ZOC-def* **proof**(*intro conjI*)
show *mC-ZOC-part c d s t ?I0 ?Q ?G*
unfolding *mC-ZOC-part-def* **using** *assms* **unfolding** *mC-C-def mC-C-part-def*
by (*auto simp add*: *Int-not-in-eq-emp*)
next
show *inj-on ?G ?Q*
unfolding *inj-on-def* **proof** *clarify*
fix *I1 I2* **assume** *I12*: *I1* ∈ *?Q I2* ∈ *?Q*
and *G*: *?G I1* = *?G I2*
show *I1* = *I2*
proof(*cases I1* ∈ *P*)
case *True*
hence *I2*: *I2* ∈ *P* **apply**(*rule-tac ccontr*)
using *G assms* **unfolding** *mC-C-def mC-C-part-def* **by** *auto*
with *True G* **have** *F I1* = *F I2* **by** *simp*
thus *?thesis* **using** *True I2 I12 assms* **unfolding** *mC-C-def inj-on-def* **by**
blast
next
case *False* **note** *case1* = *False*
hence *I1*: *I1* = {} **using** *I12* **by** *blast*
show *?thesis*
proof(*cases I2* ∈ *P*)
case *False*
hence *I2* = {} **using** *I12* **by** *blast*
thus *?thesis* **using** *I1* **by** *blast*
next
case *True*
hence *I1* ∈ *P* **apply**(*rule-tac ccontr*)
using *G assms* **unfolding** *mC-C-def mC-C-part-def* **by** *auto*
thus *?thesis* **using** *case1* **by** *simp*
qed
qed
qed
qed(*insert assms*, *unfold m-defsAll*, *fastforce*+)

lemma *matchC-C-matchC-LC*:
assumes *matchC-C theta c d*
shows *matchC-LC theta c d*
using *assms mC-C-mC-ZOC* **unfolding** *match-defs* **by** *blast*

Retracts:

definition *Sretr* **where**
Sretr theta ≡
{(*c*, *d*). *matchC-C theta c d*}

definition *ZOretr* **where**
ZOretr theta ≡
{(*c*,*d*). *matchC-LC theta c d*}

lemmas *Retr-defs* =
Sretr-def
ZOretr-def

lemma *mono-Retr*:
mono Sretr
mono ZOretr
unfolding *mono-def Retr-defs*
by (*auto simp add*: *matchC-C-mono matchC-LC-mono*)

lemma *Retr-incl*:
Sretr theta ⊆ *ZOretr theta*
unfolding *Retr-defs*
using *matchC-C-matchC-LC* **by** *blast+*

The associated bisimilarity relations:

definition *Sbis* **where** *Sbis* ≡ *gfp Sretr*
definition *ZObis* **where** *ZObis* ≡ *gfp ZOretr*

abbreviation *Sbis-abbrev* (**infix** ‹≈*s*› *55*) **where** *c* ≈*s d* ≡ (*c*, *d*) : *Sbis*
abbreviation *ZObis-abbrev* (**infix** ‹≈*01*› *55*) **where** *c* ≈*01 d* ≡ (*c*, *d*) : *ZObis*

lemmas *bis-defs* = *Sbis-def ZObis-def*

lemma *bis-incl*:
Sbis ≤ *ZObis*
unfolding *bis-defs*
using *Retr-incl gfp-mono* **by** *blast+*

lemma *bis-imp*[*simp*]:
⋀ *c1 c2*. *c1* ≈*s c2* ⟹ *c1* ≈*01 c2*
using *bis-incl* **unfolding** *bis-defs* **by** *auto*

lemma *Sbis-prefix*:
Sbis ≤ *Sretr Sbis*
unfolding *Sbis-def*

using *def-gfp-unfold mono-Retr(1)* **by** *blast*

lemma *Sbis-fix*:
Sretr Sbis = Sbis
unfolding *Sbis-def*
by (*metis def-gfp-unfold mono-Retr(1)*)

lemma *Sbis-mC-C*:
assumes $c \approx s\ d$ **and** $s \approx t$
shows $\exists P\ F.$ *mC-C Sbis c d s t P F*
using *assms Sbis-prefix* **unfolding** *Sretr-def matchC-C-def* **by** *auto*

lemma *Sbis-coind*:
assumes *theta* $\leq$ *Sretr* (*theta Un Sbis*)
shows *theta* $\leq$ *Sbis*
using *assms* **unfolding** *Sbis-def*
by (*metis Sbis-def assms def-coinduct mono-Retr(1)*)

lemma *Sbis-raw-coind*:
assumes *theta* $\leq$ *Sretr theta*
shows *theta* $\leq$ *Sbis*
proof−
  have *Sretr theta* $\subseteq$ *Sretr* (*theta Un Sbis*)
  by (*metis Un-upper1 monoD mono-Retr(1)*)
  hence *theta* $\subseteq$ *Sretr* (*theta Un Sbis*) **using** *assms* **by** *blast*
  thus *?thesis* **using** *Sbis-coind* **by** *blast*
qed

lemma *mC-C-sym*:
assumes *mC-C theta c d s t P F*
shows *mC-C* (*theta* $\widehat{\ }-1$) *d c t s* (*F* ‘ *P*) (*inv-into P F*)
unfolding *mC-C-def* **proof** (*intro conjI*)
  show *mC-C-eff-cont* (*theta*$^{-1}$) *d c t s* (*F* ‘ *P*) (*inv-into P F*)
  unfolding *mC-C-eff-cont-def* **proof** *clarify*
    fix *i j I*
    assume *I*: $I \in P$ **and** *j*: $j \in F\ I$ **and** $i \in$ *inv-into P F* (*F I*)
    hence $i \in I$ **using** *assms* **unfolding** *mC-C-def* **by** *auto*
    hence *eff c s i* $\approx$ *eff d t j* $\wedge$ (*cont c s i*, *cont d t j*) $\in$ *theta*
    using *assms I j* **unfolding** *mC-C-def mC-C-eff-cont-def* **by** *auto*
    thus *eff d t j* $\approx$ *eff c s i* $\wedge$ (*cont d t j*, *cont c s i*) $\in$ *theta*$^{-1}$
    by (*metis converseI indis-sym*)
  qed
qed(*insert assms*, *unfold m-defsAll*, *auto*)

lemma *matchC-C-sym*:
assumes *matchC-C theta c d*
shows *matchC-C* (*theta* $\widehat{\ }-1$) *d c*

unfolding *matchC-C-def* **proof** *clarify*
fix *t s*
assume $t \approx s$ **hence** $s \approx t$ **by** (*metis indis-sym*)
then obtain *P F* **where** *mC-C theta c d s t P F*
using *assms* **unfolding** *matchC-C-def* **by** *blast*
hence *mC-C* ($theta^{-1}$) *d c t s* (*F ‘ P*) (*inv-into P F*)
using *mC-C-sym* **by** *auto*
thus $\exists$ *Q G*. *mC-C* ($theta^{-1}$) *d c t s Q G* **by** *blast*
qed

lemma *Sretr-sym*:
assumes *sym theta*
shows *sym* (*Sretr theta*)
unfolding *sym-def* **proof** *clarify*
fix *c d* **assume** (*c*, *d*) ∈ *Sretr theta*
hence (*d*, *c*) ∈ *Sretr* (*theta* ^−*1*)
unfolding *Sretr-def* **using** *matchC-C-sym* **by** *simp*
thus (*d*, *c*) ∈ *Sretr theta*
using *assms* **by** (*metis sym-conv-converse-eq*)
qed

lemma *sym-Sbis*: *sym Sbis*
by (*metis Sbis-def Sretr-sym mono-Retr*(*1*) *sym-gfp*)

lemma *Sbis-sym*: *c* ≈*s d* ⟹ *d* ≈*s c*
using *sym-Sbis* **unfolding** *sym-def* **by** *blast*

lemma *mC-C-trans*:
assumes ∗: *mC-C theta1 c d s t P F* **and** ∗∗: *mC-C theta2 d e t u* (*F ‘ P*) *G*
shows *mC-C* (*theta1 O theta2*) *c e s u P* (*G o F*)
unfolding *mC-C-def* **proof** (*intro conjI*)
show *mC-C-part c e P* (*G* ∘ *F*)
using *assms* **unfolding** *mC-C-def mC-C-part-def* **by** (*auto simp add*: *image-comp*)
next
show *inj-on* (*G* ∘ *F*) *P*
using *assms* **unfolding** *mC-C-def* **by** (*auto simp add*: *comp-inj-on*)
next
show *mC-C-eff-cont* (*theta1 O theta2*) *c e s u P* (*G* ∘ *F*)
unfolding *mC-C-eff-cont-def* **proof** *clarify*
fix *I i k*
assume *I*: *I* ∈ *P* **and** *i*: *i* ∈ *I* **and** *k*: *k* ∈ (*G* ∘ *F*) *I*
have *F I* ≠ {} **using** ∗ *I* **unfolding** *mC-C-def mC-C-part-def* **by** *auto*
then obtain *j* **where** *j*: *j* ∈ *F I* **by** *auto*
have *eff c s i* ≈ *eff d t j* ∧ (*cont c s i*, *cont d t j*) ∈ *theta1*
using *I i j* ∗ **unfolding** *mC-C-def mC-C-eff-cont-def* **by** *auto*
moreover

have *eff d t j* ≈ *eff e u k* ∧ (*cont d t j*, *cont e u k*) ∈ *theta2*
using *I j k* ∗∗ **unfolding** *mC-C-def mC-C-eff-cont-def* **by** *auto*
ultimately
show *eff c s i* ≈ *eff e u k* ∧ (*cont c s i*, *cont e u k*) ∈ *theta1 O theta2*
using *indis-trans* **by** *blast*
qed
qed(*insert assms*, *unfold m-defsAll*, *auto*)

lemma *mC-C-finer*:
assumes ∗: *mC-C theta c d s t P F*
and *theta*: *trans theta*
and *Q*: *finer P Q finite Q* {} ∉ *Q part* {..<*brn c*} *Q*
and *c*: *partCompat Q indis* (*eff c s*) *partCompat Q theta* (*cont c s*)
shows *mC-C theta c d s t Q* (*lift P F*)
unfolding *mC-C-def* **proof** (*intro conjI*)
show *mC-C-part c d Q* (*lift P F*)
unfolding *mC-C-part-def* **proof**(*intro conjI*)
show {} ∉ *lift P F* ‘ *Q* **unfolding** *lift-def* [*abs-def*] **proof** *clarify*
fix *II* **assume** *1*: {} = ⋃{*F I* |*I*. *I* ∈ *P* ∧ *I* ⊆ *II*} **and** *II*: *II* ∈ *Q*
hence *II* ≠ {} **using** *Q* **by** *auto*
then obtain *I* **where** *I*: *I* ∈ *P* **and** *I* ⊆ *II*
using *Q II* **unfolding** *finer-def* **by** *blast*
hence *F I* = {} **using** *1* **by** *auto*
thus *False* **using** ∗ *I* **unfolding** *mC-C-def mC-C-part-def* **by** *auto*
qed
next
show *part* {..<*brn d*} (*lift P F* ‘ *Q*)
using *Q* ∗ *part-lift*[*of* {..<*brn c*} *P Q F* {..<*brn d*}]
unfolding *mC-C-def mC-C-part-def* **by** *auto*
qed (*insert Q*, *auto*)
next
show *inj-on* (*lift P F*) *Q*
using *Q* ∗ *inj-on-lift*[*of* {..<*brn c*} *P Q F* {..<*brn d*}]
unfolding *mC-C-def mC-C-part-def* **by** *auto*
next
show *mC-C-wt c d s t Q* (*lift P F*)
unfolding *mC-C-wt-def* **proof** *clarify*
fix *II* **assume** *II*: *II* ∈ *Q*
define *S* **where** *S* = {*I* ∈ *P* . *I* ⊆ *II*}
have *II*: *II* = (*UN I* : *S* . *id I*) **using** *II Q* **unfolding** *finer-def S-def* **by** *auto*
have *S*: ⋀ *I J*. ⟦*I* : *S*; *J* : *S*; *I* ≠ *J*⟧ ⟹ *id I Int id J* = {} *finite S*
unfolding *S-def* **using** ∗ *finite-part*[*of* {..<*brn c*} *P*] **unfolding** *mC-C-def mC-C-part-def part-def* **by** *auto*
have *SS*: ∀ *I*∈*S*. *finite I* ∀ *i*∈*S*. *finite* (*F i*)
unfolding *S-def* **using** ∗ **unfolding** *mC-C-def mC-C-part-def part-def* **apply** *simp-all*
apply (*metis complete-lattice-class.Sup-upper finite-lessThan finite-subset*)
by (*metis finite-UN finite-UnionD finite-lessThan*)
have *SSS*: ∀ *i*∈*S*. ∀ *j*∈*S*. *i* ≠ *j* ⟶ *F i* ∩ *F j* = {}

```
    proof clarify
      fix I J assume I ∈ S and J ∈ S and diff: I ≠ J
      hence IJ: {I,J} ⊆ P unfolding S-def by simp
      hence F I ≠ F J using ∗ diff unfolding mC-C-def inj-on-def by auto
     thus F I ∩ F J = {} using ∗ IJ unfolding mC-C-def mC-C-part-def part-def
by auto
    qed
    have sum (wt c s) II = sum (sum (wt c s)) S
    unfolding II using S SS sum.UNION-disjoint[of S id wt c s] by simp
    also have ... = sum (% I. sum (wt d t) (F I)) S
    apply(rule sum.cong)
    using S apply force
     unfolding S-def using ∗ unfolding mC-C-def mC-C-part-def mC-C-wt-def
by auto
    also have ... = sum (wt d t) (UN I : S . F I)
     unfolding lift-def using S sum.UNION-disjoint[of S F wt d t] S SS SSS by
simp
    also have (UN I : S . F I) = lift P F II unfolding lift-def S-def by auto
    finally show sum (wt c s) II = sum (wt d t) (lift P F II) .
  qed
next
  show mC-C-eff-cont theta c d s t Q (lift P F)
  unfolding mC-C-eff-cont-def proof clarify
    fix II i j
    assume II: II ∈ Q and i: i ∈ II and j ∈ lift P F II
     then obtain I where j: j ∈ F I and I: I ∈ P ∧ I ⊆ II unfolding lift-def
by auto
    hence I ≠ {} using ∗ unfolding mC-C-def mC-C-part-def by auto
    then obtain i' where i': i' ∈ I by auto
    hence 1: eff c s i' ≈ eff d t j ∧ (cont c s i', cont d t j) ∈ theta
    using ∗ j I unfolding mC-C-def mC-C-eff-cont-def by auto
    show eff c s i ≈ eff d t j ∧ (cont c s i, cont d t j) ∈ theta
    proof(cases i' = i)
      case True show ?thesis using 1 unfolding True .
    next
      case False
      moreover have i' ∈ II using I i' by auto
      ultimately have eff c s i ≈ eff c s i' ∧ (cont c s i, cont c s i') ∈ theta
      using i II c unfolding partCompat-def compat-def by auto
      thus ?thesis using 1 indis-trans theta unfolding trans-def by blast
    qed
  qed
qed

lemma mC-C-partCompat-eff:
assumes ∗: mC-C theta c d s t P F
shows partCompat P indis (eff c s)
unfolding partCompat-def compat-def proof clarify
  fix I i i' assume I: I ∈ P and ii': {i, i'} ⊆ I i ≠ i'
```

hence $F\ I \neq \{\}$ **using** $*$ **unfolding** *m-defsAll* **by** *blast*
then obtain j **where** j: $j \in F\ I$ **by** *auto*
from $ii'\ j\ I$ **have** *1*: *eff c s i* $\approx$ *eff d t j*
using $*$ **unfolding** *m-defsAll* **by** *blast*
from $ii'\ j\ I$ **have** *2*: *eff c s i'* $\approx$ *eff d t j*
using $*$ **unfolding** *m-defsAll* **by** *blast*
from *1 2* **show** *eff c s i* $\approx$ *eff c s i'*
using *indis-trans indis-sym* **by** *blast*
qed

lemma *mC-C-partCompat-cont*:
assumes $*$: *mC-C theta c d s t P F*
and *theta*: *sym theta trans theta*
shows *partCompat P theta* (*cont c s*)
unfolding *partCompat-def compat-def* **proof** *clarify*
fix $I\ i\ i'$ **assume** I: $I \in P$ **and** ii': $\{i, i'\} \subseteq I\ i \neq i'$
hence $F\ I \neq \{\}$ **using** $*$ **unfolding** *m-defsAll* **by** *blast*
then obtain j **where** j: $j \in F\ I$ **by** *auto*
from $ii'\ j\ I$ **have** *1*: (*cont c s i*, *cont d t j*) $\in$ *theta*
using $*$ **unfolding** *m-defsAll* **by** *blast*
from $ii'\ j\ I$ **have** *2*: (*cont c s i'*, *cont d t j*) $\in$ *theta*
using $*$ **unfolding** *m-defsAll* **by** *blast*
from *1 2* **show** (*cont c s i*, *cont c s i'*) $\in$ *theta*
using *theta* **unfolding** *trans-def sym-def* **by** *blast*
qed

lemma *matchC-C-sym-trans*:
assumes $*$: *matchC-C theta c1 c* **and** $**$: *matchC-C theta c c2*
and *theta*: *sym theta trans theta*
shows *matchC-C theta c1 c2*
unfolding *matchC-C-def* **proof** *clarify*
fix *s1 s2* **assume** *s1s2*: *s1* $\approx$ *s2*
define s **where** $s = s1$
have s: $s \approx s1 \wedge s \approx s2$ **unfolding** *s-def* **using** *s1s2* **by** *auto*
have *th*: *theta* $^{-1}$ $=$ *theta theta O theta* $\subseteq$ *theta* **using** *theta*
by (*metis sym-conv-converse-eq trans-O-subset*)+
hence $*$: *matchC-C theta c c1* **by** (*metis* $*$ *matchC-C-sym*)
from s $*$ **obtain** *P1 F1* **where** *m1*: *mC-C theta c c1 s s1 P1 F1*
unfolding *matchC-C-def* **by** *blast*
from s $**$ **obtain** *P2 F2* **where** *m2*: *mC-C theta c c2 s s2 P2 F2*
unfolding *matchC-C-def* **by** *blast*
define P **where** P = *partJoin P1 P2*

have P:
finer P1 P $\wedge$ *finer P2 P* $\wedge$
finite P $\wedge$ $\{\} \notin P \wedge$ *part* $\{..< brn\ c\}\ P$
using *m1 m2 finer-partJoin-L finite-partJoin emp-partJoin part-partJoin finite-part*[*of* $\{..< brn\ c\}\ P$]
unfolding *P-def mC-C-def mC-C-part-def* **by** *force*

have *mC-C theta c c1 s s1 P* (*lift P1 F1*)
proof(*rule mC-C-finer*)
show *partCompat P indis* (*eff c s*)
unfolding *P-def* **apply**(*rule partCompat-partJoin*)
using *m1 m2 sym-indis trans-indis mC-C-partCompat-eff* **by** *auto*
next
show *partCompat P theta* (*cont c s*)
unfolding *P-def* **apply**(*rule partCompat-partJoin*)
using *m1 m2 theta mC-C-partCompat-cont* **by** *auto*
qed(*insert P m1 theta*, *auto*)
hence *A*: *mC-C theta c1 c s1 s* (*lift P1 F1 ‘ P*) (*inv-into P* (*lift P1 F1*))
using *mC-C-sym*[*of theta c c1 s s1 P lift P1 F1*] **unfolding** *th* **by** *blast*

have *B*: *mC-C theta c c2 s s2 P* (*lift P2 F2*)
proof(*rule mC-C-finer*)
show *partCompat P indis* (*eff c s*)
unfolding *P-def* **apply**(*rule partCompat-partJoin*)
using *m1 m2 sym-indis trans-indis mC-C-partCompat-eff* **by** *auto*
next
show *partCompat P theta* (*cont c s*)
unfolding *P-def* **apply**(*rule partCompat-partJoin*)
using *m1 m2 theta mC-C-partCompat-cont* **by** *auto*
qed(*insert P m2 theta*, *auto*)

have *inj-on* (*lift P1 F1*) *P*
apply(*rule inj-on-lift*) **using** *m1 m2 P* **unfolding** *m-defsAll* **by** *auto*
hence *inv-into P* (*lift P1 F1*) ‘ *lift P1 F1 ‘ P* = *P*
by (*metis inj-on-inv-into*)
hence *mC-C* (*theta O theta*) *c1 c2 s1 s2* (*lift P1 F1 ‘ P*) (*lift P2 F2 o* (*inv-into P* (*lift P1 F1*)))
apply − **apply**(*rule mC-C-trans*[*of theta c1 c s1 s - - theta c2 s2*])
using *A B* **by** *auto*
hence *mC-C theta c1 c2 s1 s2* (*lift P1 F1 ‘ P*) (*lift P2 F2 o* (*inv-into P* (*lift P1 F1*)))
using *th mC-C-mono* **by** *blast*
thus ∃ *R H*. *mC-C theta c1 c2 s1 s2 R H* **by** *blast*
qed

lemma *Sretr-sym-trans*:
assumes *sym theta* ∧ *trans theta*
shows *trans* (*Sretr theta*)
unfolding *trans-def* **proof** *clarify*
fix *c d e* **assume** (*c*, *d*) ∈ *Sretr theta* **and** (*d*, *e*) ∈ *Sretr theta*
thus (*c*, *e*) ∈ *Sretr theta*
unfolding *Sretr-def* **using** *assms matchC-C-sym-trans* **by** *blast*
qed

lemma *trans-Sbis*: *trans Sbis*

by (*metis Sbis-def Sretr-sym Sretr-sym-trans mono-Retr sym-trans-gfp*)

lemma *Sbis-trans*: $[\![c \approx s\ d;\ d \approx s\ e]\!] \Longrightarrow c \approx s\ e$
using *trans-Sbis* **unfolding** *trans-def* **by** *blast*

lemma *ZObis-prefix*:
$ZObis \leq ZOretr\ ZObis$
unfolding *ZObis-def*
using *def-gfp-unfold mono-Retr*(*2*) **by** *blast*

lemma *ZObis-fix*:
$ZOretr\ ZObis = ZObis$
unfolding *ZObis-def*
by (*metis def-gfp-unfold mono-Retr*(*2*))

lemma *ZObis-mC-ZOC*:
assumes $c \approx 01\ d$ **and** $s \approx t$
shows $\exists I0\ \ P\ F.\ mC\text{-}ZOC\ ZObis\ c\ d\ s\ t\ I0\ P\ F$
using *assms ZObis-prefix* **unfolding** *ZOretr-def matchC-LC-def* **by** *blast*

lemma *ZObis-coind*:
assumes $theta \leq ZOretr\ (theta\ Un\ ZObis)$
shows $theta \leq ZObis$
using *assms* **unfolding** *ZObis-def*
by (*metis ZObis-def assms def-coinduct mono-Retr*(*2*))

lemma *ZObis-raw-coind*:
assumes $theta \leq ZOretr\ theta$
shows $theta \leq ZObis$
proof−
  have $ZOretr\ theta \subseteq ZOretr\ (theta\ Un\ ZObis)$
  by (*metis Un-upper1 monoD mono-Retr*)
  hence $theta \subseteq ZOretr\ (theta\ Un\ ZObis)$ **using** *assms* **by** *blast*
  thus *?thesis* **using** *ZObis-coind* **by** *blast*
qed

lemma *mC-ZOC-sym*:
assumes *theta*: *sym theta* **and** ∗: *mC-ZOC theta c d s t I0 P F*
shows *mC-ZOC theta d c t s* (*F I0*) (*F* ‘ *P*) (*inv-into P F*)
unfolding *mC-ZOC-def* **proof** (*intro conjI*)
  show *mC-ZOC-part d c t s* (*F I0*) (*F* ‘ *P*) (*inv-into P F*)
  unfolding *mC-ZOC-part-def* **proof**(*intro conjI*)
    have *0*: *inj-on F P I0* ∈ *P* **using** ∗ **unfolding** *mC-ZOC-def mC-ZOC-part-def*
by *blast*+

```
    have inv-into P F ‘ (F ‘ P − {F I0}) = inv-into P F ‘ (F ‘ (P − {I0}))
    using 0 inj-on-image-set-diff[of F P P {I0}, OF - Set.Diff-subset] by simp
   also have ... = P − {I0} using 0 by (metis Diff-subset inv-into-image-cancel)
    finally have inv-into P F ‘ (F ‘ P − {F I0}) = P − {I0} .
    thus {} ∉ inv-into P F ‘ (F ‘ P − {F I0})
    using ∗ unfolding mC-ZOC-def mC-ZOC-part-def by simp

   have part {..<brn c} P using ∗ unfolding mC-ZOC-def mC-ZOC-part-def by
blast
    thus part {..<brn c} (inv-into P F ‘ F ‘ P) using 0 by auto
  qed(insert ∗, unfold mC-ZOC-def mC-ZOC-part-def, blast+)
next
  have 0: inj-on F P I0 ∈ P using ∗ unfolding mC-ZOC-def mC-ZOC-part-def
by blast+
  show mC-ZOC-wt d c t s (F I0) (F ‘ P) (inv-into P F)
  unfolding mC-ZOC-wt-def proof(intro conjI ballI impI)
    fix J assume J ∈ F ‘ P − {F I0} and le-1: sum (wt d t) (F I0) < 1 ∧ sum
(wt c s) (inv-into P F (F I0)) < 1
    then obtain I where I: I ∈ P − {I0} and J: J = F I
      by auto
    have 2: inv-into P F J = I unfolding J using 0 I by simp
    have 3: inv-into P F (F I0) = I0 using 0 by simp
    show
    sum (wt d t) J / (1 − sum (wt d t) (F I0)) =
     sum (wt c s) (inv-into P F J) / (1 − sum (wt c s) (inv-into P F (F I0)))
    unfolding 2 3 unfolding J
    using ∗ I le-1 unfolding mC-ZOC-def mC-ZOC-wt-def by (metis 3 J)
  qed
next
  show mC-ZOC-eff-cont theta d c t s (F I0) (F ‘ P) (inv-into P F)
  unfolding mC-ZOC-eff-cont-def proof(intro allI impI, elim conjE)
    fix i j J
    assume J ∈ F ‘ P − {F I0} and j: j ∈ J and i: i ∈ inv-into P F J
    then obtain I where J: J = F I and I: I ∈ P − {I0}
      by auto
    hence i ∈ I using assms i unfolding mC-ZOC-def by auto
    hence eff c s i ≈ eff d t j ∧ (cont c s i, cont d t j) ∈ theta
    using assms I j unfolding mC-ZOC-def mC-ZOC-eff-cont-def J by auto
    thus eff d t j ≈ eff c s i ∧ (cont d t j, cont c s i) ∈ theta
    by (metis assms indis-sym sym-def)
  qed
qed(insert assms, unfold sym-def m-defsAll, auto)

lemma matchC-LC-sym:
assumes ∗: sym theta and matchC-LC theta c d
shows matchC-LC theta d c
unfolding matchC-LC-def proof clarify
  fix t s
  assume t ≈ s  hence s ≈ t by (metis indis-sym)
```

then obtain *I0 P F* **where** *mC-ZOC theta c d s t I0 P F*
using *assms* **unfolding** *matchC-LC-def* **by** *blast*
hence *mC-ZOC theta d c t s* (*F I0*) (*F ‘ P*) (*inv-into P F*)
using *mC-ZOC-sym* ∗ **by** *auto*
thus ∃ *I0 Q G*. *mC-ZOC theta d c t s I0 Q G* **by** *blast*
qed

lemma *ZOretr-sym*:
assumes *sym theta*
shows *sym* (*ZOretr theta*)
unfolding *sym-def* **proof** *clarify*
fix *c d* **assume** (*c*, *d*) ∈ *ZOretr theta*
hence (*d*, *c*) ∈ *ZOretr theta*
unfolding *ZOretr-def* **using** *assms matchC-LC-sym* **by** *simp*
thus (*d*, *c*) ∈ *ZOretr theta*
using *assms* **by** (*metis sym-conv-converse-eq*)
qed

lemma *sym-ZObis*: *sym ZObis*
by (*metis ZObis-def ZOretr-sym mono-Retr sym-gfp*)

lemma *ZObis-sym*: *c* ≈*01 d* ⟹ *d* ≈*01 c*
using *sym-ZObis* **unfolding** *sym-def* **by** *blast*

2.7 List versions of the bisimilarities

definition *SbisL* **where**
SbisL cl dl ≡
length cl = *length dl* ∧ (∀ *n* < *length cl*. *cl* ! *n* ≈*s dl* ! *n*)

lemma *SbisL-intro*[*intro*]:
assumes *length cl* = *length dl* **and**
⋀ *n*. ⟦*n* < *length cl*; *n* < *length dl*⟧ ⟹ *cl* ! *n* ≈*s dl* ! *n*
shows *SbisL cl dl*
using *assms* **unfolding** *SbisL-def* **by** *auto*

lemma *SbisL-length*[*simp*]:
assumes *SbisL cl dl*
shows *length cl* = *length dl*
using *assms* **unfolding** *SbisL-def* **by** *simp*

lemma *SbisL-Sbis*[*simp*]:
assumes *SbisL cl dl* **and** *n* < *length cl* ∨ *n* < *length dl*
shows *cl* ! *n* ≈*s dl* ! *n*
using *assms* **unfolding** *SbisL-def* **by** *simp*

lemma *SbisL-update*[*simp*]:
assumes *cldl*: *SbisL cl dl* **and** *c′d′*: *c′* ≈*s d′*
shows *SbisL* (*cl* [*n* := *c′*]) (*dl* [*n* := *d′*]) (**is** *SbisL ?cl′ ?dl′*)

proof(*cases* $n < length\ cl$)
  case *True*
  show *?thesis* **proof**
    fix m **assume** m: $m < length\ ?cl'\ m < length\ ?dl'$
    thus $?cl' ! m \approx s\ ?dl' ! m$ **using** *assms* **by** (*cases* $m = n$) *auto*
  qed (*use cldl* **in** *simp*)
qed (*use cldl* **in** ‹*simp add*: *list-update-beyond*›)

lemma *SbisL-update-L*[*simp*]:
assumes *SbisL cl dl* **and** $c' \approx s\ dl!n$
shows *SbisL* ($cl[n := c']$) *dl*
  by (*metis SbisL-update assms list-update-id*)

lemma *SbisL-update-R*[*simp*]:
assumes *SbisL cl dl* **and** $cl!n \approx s\ d'$
shows *SbisL cl* ($dl[n := d']$)
  by (*metis SbisL-update assms list-update-id*)

definition *ZObisL* **where**
ZObisL cl dl ≡
 $length\ cl = length\ dl \wedge (\forall\ n < length\ cl.\ cl ! n \approx 01\ dl ! n)$

lemma *ZObisL-intro*[*intro*]:
assumes $length\ cl = length\ dl$ **and**
$\bigwedge n.\ [\![n < length\ cl;\ n < length\ dl]\!] \Longrightarrow cl ! n \approx 01\ dl ! n$
shows *ZObisL cl dl*
using *assms* **unfolding** *ZObisL-def* **by** *auto*

lemma *ZObisL-length*[*simp*]:
assumes *ZObisL cl dl*
shows $length\ cl = length\ dl$
using *assms* **unfolding** *ZObisL-def* **by** *simp*

lemma *ZObisL-ZObis*[*simp*]:
assumes *ZObisL cl dl* **and** $n < length\ cl \vee n < length\ dl$
shows $cl ! n \approx 01\ dl ! n$
using *assms* **unfolding** *ZObisL-def* **by** *simp*

lemma *ZObisL-update*[*simp*]:
assumes *cldl*: *ZObisL cl dl* **and** $c'd'$: $c' \approx 01\ d'$
shows *ZObisL* ($cl\ [n := c']$) ($dl\ [n := d']$) (**is** *ZObisL ?cl' ?dl'*)
proof(*cases* $n < length\ cl$)
  case *True*
  show *?thesis* **proof**
    fix m **assume** m: $m < length\ ?cl'\ m < length\ ?dl'$
    thus $?cl' ! m \approx 01\ ?dl' ! m$ **using** *assms* **by** (*cases* $m = n$) *auto*
  qed (*use cldl* **in** *simp*)

qed (*use cldl* **in** ‹*simp add*: *list-update-beyond*›)

lemma *ZObisL-update-L*[*simp*]:
assumes *ZObisL cl dl* **and** $c' \approx_{01} dl!n$
shows *ZObisL* (*cl*[*n* := *c'*]) *dl*
by (*metis ZObisL-update assms list-update-id*)

lemma *ZObisL-update-R*[*simp*]:
assumes *ZObisL cl dl* **and** $cl!n \approx_{01} d'$
shows *ZObisL cl* (*dl*[*n* := *d'*])
by (*metis ZObisL-update assms list-update-id*)

2.8 Discreetness for configurations

coinductive *discrCf* **where**
intro:
$(\bigwedge i.\ i < brn\ (fst\ cf) \longrightarrow$
$snd\ cf \approx snd\ (cont\text{-}eff\ cf\ i) \wedge discrCf\ (cont\text{-}eff\ cf\ i))$
$\Longrightarrow discrCf\ cf$

Coinduction for discrness:

lemma *discrCf-coind*[*consumes 1*, *case-names Hyp*, *induct pred*: *discr*]:
assumes ∗: *phi cf* **and**
∗∗: $\bigwedge cf\ i.$
$[\![i < brn\ (fst\ cf);\ phi\ cf]\!] \Longrightarrow$
$snd\ cf \approx snd\ (cont\text{-}eff\ cf\ i) \wedge (phi\ (cont\text{-}eff\ cf\ i) \vee discrCf\ (cont\text{-}eff\ cf\ i))$
shows *discrCf cf*
using ∗ **apply**(*induct rule*: *discrCf.coinduct*) **using** ∗∗ **by** *auto*

lemma *discrCf-raw-coind*[*consumes 1*, *case-names Hyp*]:
assumes ∗: *phi cf* **and**
∗∗: $\bigwedge cf\ i.\ [\![i < brn\ (fst\ cf);\ phi\ cf]\!] \Longrightarrow$
$snd\ cf \approx snd\ (cont\text{-}eff\ cf\ i) \wedge phi\ (cont\text{-}eff\ cf\ i)$
shows *discrCf cf*
using ∗ **apply**(*induct*) **using** ∗∗ **by** *blast*

Discreetness versus transition:

lemma *discrCf-cont*[*simp*]:
assumes ∗: *discrCf cf* **and** ∗∗: $i < brn\ (fst\ cf)$
shows *discrCf* (*cont-eff cf i*)
by (*meson assms discrCf.cases*)

lemma *discrCf-eff-indis*[*simp*]:
assumes ∗: *discrCf cf* **and** ∗∗: $i < brn\ (fst\ cf)$
shows $snd\ cf \approx snd\ (cont\text{-}eff\ cf\ i)$
by (*meson assms discrCf.cases*)

lemma *discr-discrCf*:
assumes *discr c*

shows *discrCf* (*c*, *s*)
proof−
{fix *cf* :: ('*test*, '*atom*, '*choice*) *cmd* × '*state*
assume *discr* (*fst cf*)
hence *discrCf cf*
apply (*induct*)
using *discr-eff-indis discr-cont* **unfolding** *eff-def cont-def cont-eff-def* **by** *auto*
}
thus *?thesis* **using** *assms* **by** *auto*
qed

lemma *ZObis-pres-discrCfL*:
assumes *fst cf* ≈*01 fst df* **and** *snd cf* ≈ *snd df* **and** *discrCf df*
shows *discrCf cf*
proof−
have ∃ *df*. *fst cf* ≈*01 fst df* ∧ *snd cf* ≈ *snd df* ∧ *discrCf df* **using** *assms* **by** *blast*
thus *?thesis*
proof (*induct rule*: *discrCf-raw-coind*)
case (*Hyp cf i*)
then obtain *df* **where** *i*: *i* < *brn* (*fst cf*) **and**
df: *discrCf df* **and** *cf-df*: *fst cf* ≈*01 fst df snd cf* ≈ *snd df* **by** *blast*
then obtain *I0 P F* **where**
match: *mC-ZOC ZObis* (*fst cf*) (*fst df*) (*snd cf*) (*snd df*) *I0 P F*
using *ZObis-mC-ZOC* **by** *blast*
hence ⋃ *P* = {..<*brn* (*fst cf*)}
using *i* **unfolding** *mC-ZOC-def mC-ZOC-part-def part-def* **by** *simp*
then obtain *I* **where** *I*: *I* ∈ *P* **and** *i*: *i* ∈ *I* **using** *i* **by** *auto*
show *?case*
proof(*cases I* = *I0*)
case *False*
then obtain *j* **where** *j*: *j* ∈ *F I* **using** *match I False* **unfolding** *m-defsAll* **by** *blast*
hence *j* < *brn* (*fst df*) **using** *I match*
unfolding *mC-ZOC-def mC-ZOC-part-def part-def* **apply** *simp* **by** *blast*
hence *md*: *discrCf* (*cont-eff df j*) **and** *s*: *snd df* ≈ *snd* (*cont-eff df j*)
using *df discrCf-cont*[*of df j*] *discrCf-eff-indis*[*of df j*] **by** *auto*
have *0*: *snd* (*cont-eff cf i*) ≈ *snd* (*cont-eff df j*) **and**
md2: *fst* (*cont-eff cf i*) ≈*01 fst* (*cont-eff df j*)
using *I i j match False* **unfolding** *mC-ZOC-def mC-ZOC-eff-cont-def*
unfolding *eff-def cont-def cont-eff-def* **by** *auto*
hence *snd cf* ≈ *snd* (*cont-eff cf i*) **using** *s indis-sym indis-trans cf-df* **by** *blast*
thus *?thesis* **using** *md md2 0* **by** *blast*
next
case *True*
hence *snd cf* ≈ *snd* (*cont-eff cf i*) ∧ *fst* (*cont-eff cf i*) ≈*01 fst df*
using *match i ZObis-sym* **unfolding** *mC-ZOC-def mC-ZOC-eff-cont0-def*
unfolding *cont-eff-def cont-def eff-def* **by** *blast*

moreover have *snd* (*cont-eff cf i*) ≈ *snd df*
using *match i indis-sym indis-trans cf-df* **unfolding** *mC-ZOC-def mC-ZOC-eff-cont0-def*
unfolding *cont-eff-def cont-def eff-def True* **by** *blast*
ultimately show *?thesis* **using** *df cf-df* **by** *blast*
qed
qed
qed

corollary *ZObis-pres-discrCfR*:
assumes *discrCf cf* **and** *fst cf* ≈*01 fst df* **and** *snd cf* ≈ *snd df*
shows *discrCf df*
using *assms ZObis-pres-discrCfL ZObis-sym indis-sym* **by** *blast*

end

end

3 Trace-Based Noninterference

theory *Trace-Based*
imports *Resumption-Based*
begin

3.1 Preliminaries

lemma *dist-sum*:
fixes $f :: 'a \Rightarrow real$ **and** $g :: 'a \Rightarrow real$
assumes $\bigwedge i.\ i \in I \Longrightarrow dist\ (f\ i)\ (g\ i) \leq e\ i$
shows $dist\ (\sum i{\in}I.\ f\ i)\ (\sum i{\in}I.\ g\ i) \leq (\sum i{\in}I.\ e\ i)$
proof *cases*
assume *finite I* **from** *this assms* **show** *?thesis*
proof *induct*
case (*insert i I*)
then have $dist\ (\sum i{\in}insert\ i\ I.\ f\ i)\ (\sum i{\in}insert\ i\ I.\ g\ i) \leq dist\ (f\ i)\ (g\ i) +$
$dist\ (\sum i{\in}I.\ f\ i)\ (\sum i{\in}I.\ g\ i)$
by (*simp add*: *dist-triangle-add*)
also have $\ldots \leq e\ i + (\sum i{\in}I.\ e\ i)$
using *insert* **by** (*intro add-mono*) *auto*
finally show *?case* **using** *insert* **by** *simp*
qed *simp*
qed *simp*

lemma *dist-mult*[*simp*]: $dist\ (x * y)\ (x * z) = |x| * dist\ y\ z$
unfolding *dist-real-def abs-mult*[*symmetric*] *right-diff-distrib* **..**

lemma *dist-divide*[*simp*]: $dist\ (y\ /\ r)\ (z\ /\ r) = dist\ y\ z\ /\ |r|$

unfolding *dist-real-def abs-divide*[*symmetric*] *diff-divide-distrib* **..**

lemma *dist-weighted-sum*:
fixes $f :: 'a \Rightarrow real$ **and** $g :: 'b \Rightarrow real$
assumes *eps*: $\bigwedge i\ j.\ i \in I \Longrightarrow j \in J \Longrightarrow w\ i \neq 0 \Longrightarrow v\ j \neq 0 \Longrightarrow dist\ (f\ i)\ (g\ j) \leq d\ i + e\ j$
and *pos*: $\bigwedge i.\ i{\in}I \Longrightarrow 0 \leq w\ i\ \bigwedge j.\ j{\in}J \Longrightarrow 0 \leq v\ j$
and *sum*: $(\sum i{\in}I.\ w\ i) = 1\ (\sum j{\in}J.\ v\ j) = 1$
shows *dist* $(\sum i{\in}I.\ w\ i * f\ i)\ (\sum j{\in}J.\ v\ j * g\ j) \leq (\sum i{\in}I.\ w\ i * d\ i) + (\sum j{\in}J.\ v\ j * e\ j)$
proof −
let *?W h* $= (\sum (i,j){\in}I{\times}J.\ (w\ i * v\ j) * h\ (i,j))$
{ fix $g :: 'b \Rightarrow real$
have $(\sum j{\in}J.\ v\ j * g\ j) = (\sum i{\in}I.\ w\ i) * (\sum j{\in}J.\ v\ j * g\ j)$
using *sum* **by** *simp*
also have $\ldots =$ *?W* $(g{\circ}snd)$
by (*simp add*: *ac-simps sum-product sum.cartesian-product*)
finally have $(\sum j{\in}J.\ v\ j * g\ j) =$ *?W* $(g{\circ}snd)$ **. }**
moreover
{ fix $f :: 'a \Rightarrow real$
have $(\sum i{\in}I.\ w\ i * f\ i) = (\sum i{\in}I.\ w\ i * f\ i) * (\sum j{\in}J.\ v\ j)$
using *sum* **by** *simp*
also have $\ldots =$ *?W* $(f{\circ}fst)$
unfolding *sum-product sum.cartesian-product* **by** (*simp add*: *ac-simps*)
finally have $(\sum i{\in}I.\ w\ i * f\ i) =$ *?W* $(f{\circ}fst)$ **. }**
moreover
{ have *dist* (*?W* $(f{\circ}fst)$) (*?W* $(g{\circ}snd)$) $\leq$ *?W* $(\lambda(i,j).\ d\ i + e\ j)$
using *eps pos*
by (*intro dist-sum*)
(*auto simp*: *mult-le-cancel-left zero-less-mult-iff mult-le-0-iff not-le*[*symmetric*])
also have $\ldots =$ *?W* $(d \circ fst) +$ *?W* $(e \circ snd)$
by (*auto simp add*: *sum.distrib*[*symmetric*] *field-simps intro*!: *sum.cong*)
finally have *dist* (*?W* $(f{\circ}fst)$) (*?W* $(g{\circ}snd)$) $\leq$ *?W* $(d \circ fst) +$ *?W* $(e \circ snd)$
by *simp* **}**
ultimately show *?thesis* **by** *simp*
qed

lemma *field-abs-le-zero-epsilon*:
fixes $x :: 'a{::}\{linordered\text{-}field\}$
assumes *epsilon*: $\bigwedge e.\ 0 < e \Longrightarrow |x| \leq e$
shows $|x| = 0$
proof (*rule antisym*)
show $|x| \leq 0$
proof (*rule field-le-epsilon*)
fix $e :: 'a$ **assume** $0 < e$ **then show** $|x| \leq 0 + e$ **by** *simp fact*
qed
qed *simp*

lemma *nat-nat-induct*[*case-names less*]:

fixes $P :: nat \Rightarrow nat \Rightarrow bool$
assumes *less*: $\bigwedge n\ m.\ (\bigwedge j\ k.\ j + k < n + m \Longrightarrow P\ j\ k) \Longrightarrow P\ n\ m$
shows $P\ n\ m$
proof −
define N **where** $N \equiv n + m$
then show *?thesis*
proof (*induct N arbitrary*: $n\ m$ *rule*: *nat-less-induct*)
case *1* **show** $P\ n\ m$
apply (*rule less*)
apply (*rule 1*[*rule-format*])
apply *auto*
done
qed
qed

lemma *part-insert*:
assumes *part* $A\ P$ **assumes** $X \cap A = \{\}$
shows *part* $(A \cup X)$ (*insert* $X\ P$)
using *assms* **by** (*auto simp*: *part-def*)

lemma *part-insert-subset*:
assumes X: *part* $(A - X)\ P\ X \subseteq A$
shows *part* A (*insert* $X\ P$)
using X *part-insert*[*of* $A - X\ P\ X$] **by** (*simp add*: *Un-absorb2*)

lemma *part-is-subset*:
part $S\ P \Longrightarrow p \in P \Longrightarrow p \subseteq S$
unfolding *part-def* **by** (*metis Union-upper*)

lemma *dist-nonneg-bounded*:
fixes $l\ u\ x\ y$ :: *real*
assumes $l \leq x\ x \leq u\ l \leq y\ y \leq u$
shows *dist* $x\ y \leq u - l$
using *assms* **unfolding** *dist-real-def* **by** *arith*

lemma *integrable-count-space-finite-support*:
fixes $f :: 'a \Rightarrow 'b$::{*banach*,*second-countable-topology*}
shows *finite* $\{x{\in}X.\ f\ x \neq 0\} \Longrightarrow$ *integrable* (*count-space* X) f
by (*auto simp*: *nn-integral-count-space integrable-iff-bounded*)

lemma *lebesgue-integral-point-measure*:
fixes $g :: - \Rightarrow real$
assumes f: *finite* $\{a{\in}A.\ 0 < f\ a \wedge g\ a \neq 0\}$
shows *integral*^{L} (*point-measure* $A\ f$) $g = (\sum a|a{\in}A \wedge 0 < f\ a \wedge g\ a \neq 0.\ f\ a *$
$g\ a$)

proof −
have *eq*: $\{a \in A.\ max\ 0\ (f\ a) \neq 0 \wedge g\ a \neq 0\} = \{a{\in}A.\ 0 < f\ a \wedge g\ a \neq 0\}$
by *auto*

```
  have ∗: ennreal (f x) = ennreal (max 0 (f x)) for x
    by (cases 0 ≤ f x) (auto simp: max.absorb1 ennreal-neg)

  show ?thesis
    unfolding point-measure-def ∗
    using f
    apply (subst integral-real-density)
   apply (auto simp add: integral-real-density lebesgue-integral-count-space-finite-support
eq
      intro!: sum.cong)
    done
qed

lemma (in finite-measure) finite-measure-dist:
  assumes AE: AE x in M. x ∉ C ⟶ (x ∈ A ⟷ x ∈ B)
  assumes sets: A ∈ sets M B ∈ sets M C ∈ sets M
  shows dist (measure M A) (measure M B) ≤ measure M C
proof −
  { have measure M A ≤ measure M (B ∪ C)
      using AE sets by (auto intro: finite-measure-mono-AE)
    also have ... ≤ measure M B + measure M C
      using sets by (simp add: measure-Un-le)
    finally have A: measure M A ≤ measure M B + measure M C . }
  moreover
  { have measure M B ≤ measure M (A ∪ C)
      using AE sets by (auto intro: finite-measure-mono-AE)
    also have ... ≤ measure M A + measure M C
      using sets by (simp add: measure-Un-le)
    finally have B: measure M B ≤ measure M A + measure M C . }
  ultimately show ?thesis
    by (simp add: dist-real-def)
qed

lemma (in prob-space) prob-dist:
  assumes AE: AE x in M. ¬ C x ⟶ (A x ⟷ B x)
  assumes sets: Measurable.pred M A Measurable.pred M B Measurable.pred M C
  shows dist P(x in M. A x) P(x in M. B x) ≤ P(x in M. C x)
  using assms by (intro finite-measure-dist) auto

lemma Least-eq-0-iff: (∃ i::nat. P i) ⟹ (LEAST i. P i) = 0 ⟷ P 0
  by (metis LeastI-ex neq0-conv not-less-Least)

lemma case-nat-comp-Suc[simp]: case-nat x f ∘ Suc = f
  by auto

lemma sum-eq-0-iff:
  fixes f :: - ⇒ 'a :: {comm-monoid-add,ordered-ab-group-add}
  shows finite A ⟹ (⋀i. i ∈ A ⟹ 0 ≤ f i) ⟹ (∑ i∈A. f i) = 0 ⟷ (∀ i∈A.
f i = 0)
```

apply *rule*
apply (*blast intro*: *sum-nonneg-0*)
apply *simp*
done

lemma *sum-less-0-iff*:
fixes $f :: - \Rightarrow 'a :: \{comm\text{-}monoid\text{-}add, ordered\text{-}ab\text{-}group\text{-}add\}$
shows $finite\ A \Longrightarrow (\bigwedge i.\ i \in A \Longrightarrow 0 \leq f\ i) \Longrightarrow 0 < (\sum i{\in}A.\ f\ i) \longleftrightarrow (\exists\, i{\in}A.\ 0 < f\ i)$
using *sum-nonneg*[*of A f*] *sum-eq-0-iff*[*of A f*] **by** (*simp add*: *less-le*)

context *PL-Indis*
begin

declare *emp-gen*[*simp del*]

interpretation *pmf-as-function* **.**

lift-definition *wt-pmf* :: ('*test*, '*atom*, '*choice*) *cmd* × '*state* ⇒ *nat pmf* **is**
$\lambda(c,\ s)\ i.\ if\ proper\ c\ then\ if\ i < brn\ c\ then\ wt\ c\ s\ i\ else\ 0\ else\ if\ i = 0\ then\ 1\ else\ 0$
proof
let *?f* = $\lambda(c,\ s)\ i.\ if\ proper\ c\ then\ if\ i < brn\ c\ then\ wt\ c\ s\ i\ else\ 0\ else\ if\ i = 0\ then\ 1\ else\ 0$
fix *cf* **show** $\forall\, i.\ 0 \leq$ *?f cf i*
by (*auto split*: *prod.split*)
show $(\int^+ i.$ *?f cf i* ∂*count-space UNIV*$) = 1$
by (*subst nn-integral-count-space'*[**where** *A*=*if proper* (*fst cf*) *then* {..<*brn* (*fst cf*)} *else* {*0*} **for** *n*])
(*auto split*: *prod.split*)
qed

definition *trans* :: ('*test*, '*atom*, '*choice*) *cmd* × '*state* ⇒ (('*test*, '*atom*, '*choice*) *cmd* × '*state*) *pmf* **where**
trans cf = *map-pmf* ($\lambda i.$ *if proper* (*fst cf*) *then cont-eff cf i else cf*) (*wt-pmf cf*)

sublocale *T?*: *MC-syntax trans* **.**

abbreviation *G cf* ≡ *set-pmf* (*trans cf*)

lemma *set-pmf-map*: *set-pmf* (*map-pmf f M*) = *f* ‘ *set-pmf M*
using *pmf-set-map*[*of f*] **by** (*simp add*: *comp-def fun-eq-iff*)

lemma *set-pmf-wt*:
set-pmf (*wt-pmf cf*) = (*if proper* (*fst cf*) *then* {*i*. *i* < *brn* (*fst cf*) ∧ *0* < *wt* (*fst cf*) (*snd cf*) *i*} *else* {*0*})
by (*auto simp*: *set-eq-iff set-pmf-iff wt-pmf.rep-eq split*: *prod.split*) (*metis less-le wt-ge-0*)

lemma *G-eq*:
G cf = (*if proper* (*fst cf*) *then* {*cont-eff cf i* | *i*. *i* < *brn* (*fst cf*) ∧ *0* < *wt* (*fst cf*) (*snd cf*) *i* } *else* {*cf*})
by (*auto simp add*: *trans-def set-pmf-map set-pmf-wt*)

lemma *discrCf-G*: *discrCf cf* ⟹ *cf'* ∈ *G cf* ⟹ *discrCf cf'*
using *discrCf-cont*[*of cf*] **by** (*auto simp*: *G-eq split*: *if-split-asm*)

lemma *measurable-discrCf*[*measurable*]: *Measurable.pred* (*count-space UNIV*) *discrCf*
by *auto*

lemma *measurable-indis*[*measurable*]: *Measurable.pred* (*count-space UNIV*) (λ*x*. *snd x* ≈ *c*)
by *auto*

lemma *integral-trans*:
proper (*fst cf*) ⟹
(∫ *x*. *f x* ∂*trans cf*) = (∑ *i*<*brn* (*fst cf*). *wt* (*fst cf*) (*snd cf*) *i* ∗ *f* (*cont-eff cf i*))
unfolding *trans-def map-pmf-rep-eq*
apply (*simp add*: *integral-distr*)
apply (*subst integral-measure-pmf*[*of* {..< *brn* (*fst cf*)}])
apply (*auto simp*: *set-pmf-wt mult-ac wt-pmf.rep-eq split*: *prod.split*)
done

3.2 Quasi strong termination traces

abbreviation *qsend* ≡ *sfirst* (*holds discrCf*)

lemma *qsend-eq-0-iff*: *qsend cfT* = *0* ⟷ *discrCf* (*shd cfT*)
by (*simp add*: *sfirst.simps*[*of - cfT*])

lemma *qsend-eq-0*[*simp*]: *discrCf cf* ⟹ *qsend* (*cf* ## ω) = *0*
by (*simp add*: *qsend-eq-0-iff*)

lemma *alw-discrCf*: *enabled cf* ω ⟹ *discrCf cf* ⟹ *alw* (*holds discrCf*) ω
by (*coinduction arbitrary*: *cf* ω)
(*auto simp add*: *HLD-iff elim*: *enabled.cases intro*: *discrCf-G simp del*: *split-paired-Ex*)

lemma *alw-discrCf-indis'*:
enabled cf ω ⟹ *discrCf cf* ⟹ *snd cf* ≈ *t* ⟹ *alw* (*holds* (λ*cf'*. *snd cf'* ≈ *t*)) ω
proof (*coinduction arbitrary*: *cf* ω)
case *alw* **with** *discrCf-eff-indis*[*of cf*] **show** *?case*
by (*auto simp add*: *HLD-iff enabled.simps*[*of - ω*] *G-eq*
simp del: *split-paired-Ex discrCf-eff-indis*
intro!: *exI*[*of - shd* ω]
split: *if-split-asm*)
(*blast intro*: *indis-trans indis-sym*)+

qed

lemma *alw-discrCf-indis*:
enabled cf ω ⟹ discrCf cf ⟹ alw (holds (λcf'. snd cf' ≈ snd cf)) (cf ## ω)
using *alw-discrCf-indis'[of cf ω, OF - - indis-refl]*
by (*simp add*: *alw.simps[of - cf ## ω] indis-refl*)

lemma *enabled-sdrop*: *enabled cf ω ⟹ enabled ((cf ## ω) !! n) (sdrop n ω)*
proof (*induction n arbitrary*: *cf ω*)
case (*Suc n*) **from** *Suc.IH[of shd ω stl ω] Suc.prems* **show** *?case*
unfolding *enabled.simps[of cf]* **by** *simp*
qed *simp*

lemma *sfirst-eq-eSuc*: *sfirst P ω = eSuc n ⟷ (¬ P ω) ∧ sfirst P (stl ω) = n*
by (*subst sfirst.simps*) *auto*

lemma *qsend-snth*: *qsend ω = enat n ⟹ discrCf (ω !! n)*
by (*induction n arbitrary*: *ω*)
(*simp-all add*: *eSuc-enat[symmetric] enat-0 sfirst-eq-0 sfirst-eq-eSuc*)

lemma *indis-iff*: *a ≈ d ⟹ b ≈ d ⟹ a ≈ c ⟷ b ≈ c*
by (*metis indis-trans indis-sym indis-refl*)

lemma *enabled-qsend-indis*:
assumes *enabled cf ω qsend (cf ## ω) ≤ n qsend (cf ## ω) ≤ m*
shows *snd ((cf ## ω) !! n) ≈ t ⟷ snd ((cf ## ω) !! m) ≈ t*
proof −
from *assms* **obtain** *N* :: *nat* **where** *N*: *qsend (cf ## ω) = N*
by (*cases qsend (cf ## ω)*) *auto*
note *discr-N = qsend-snth[OF this]* **and** *sd = enabled-sdrop[OF assms(1), of N]*
have ⋀*ω*. *ω !! N ## sdrop N (stl ω) = sdrop N ω*
by (*induct N*) *auto*
from *this[of cf ## ω]* **have** *eq*: *(cf ## ω) !! N ## sdrop N ω = sdrop N (cf ## ω)*
by *simp*

from *discr-N alw-discrCf-indis[OF sd -] assms(2,3)* **show** *?thesis*
by (*simp add*: *N alw-iff-sdrop le-iff-add*[**where** *'a=nat*] *eq*)
(*metis indis-iff*)
qed

lemma *enabled-imp-UNTIL-alw-discrCf*:
enabled (shd ω) (stl ω) ⟹ (not (holds discrCf) until (alw (holds discrCf))) ω
proof (*coinduction arbitrary*: *ω*)
case *UNTIL* **then show** *?case*
using *alw-discrCf[of shd ω stl ω]*
by (*cases discrCf (shd ω)*)
(*simp-all add*: *enabled.simps[of shd ω] alw.simps[of - ω]*)
qed

lemma *less-qsend-iff-not-discrCf*:
enabled cf bT $\Longrightarrow$ *n* < *qsend* (*cf ## bT*) $\longleftrightarrow$ ¬ *discrCf* ((*cf ## bT*) !! *n*)
using *enabled-imp-UNTIL-alw-discrCf*[*THEN less-sfirst-iff*, *of cf ## bT*]
by (*simp add*: *holds.simps*[*abs-def*])

3.3 Terminating configurations

definition *qstermCf cf* $\longleftrightarrow$ (∀ *cfT*. *enabled cf cfT* $\longrightarrow$ *qsend* (*cf ## cfT*) < ∞)

lemma *qstermCf-E*:
qstermCf cf $\Longrightarrow$ *cf'* ∈ *G cf* $\Longrightarrow$ *qstermCf cf'*
apply (*auto simp*: *qstermCf-def*)
apply (*erule-tac x=cf' ## cfT* **in** *allE*)
apply (*auto simp*: *sfirst-Stream intro*: *enat-0 discrCf-G split*: *if-split-asm intro*: *enabled.intros*)
done

abbreviation *eff-at cf bT n* ≡ *snd* ((*cf ## bT*) !! *n*)
abbreviation *cont-at cf bT n* ≡ *fst* ((*cf ## bT*) !! *n*)

definition *amSec c* $\longleftrightarrow$ (∀ *s1 s2 n t*. *s1* ≈ *s2* $\longrightarrow$
P(*bT in T.T* (*c*, *s1*). *eff-at* (*c*, *s1*) *bT n* ≈ *t*) =
P(*bT in T.T* (*c*, *s2*). *eff-at* (*c*, *s2*) *bT n* ≈ *t*))

definition *eSec c* $\longleftrightarrow$ (∀ *s1 s2 t*. *s1* ≈ *s2* $\longrightarrow$
P(*bT in T.T* (*c*, *s1*). ∃ *n*. *qsend* ((*c*, *s1*) *## bT*) = *n* ∧ *eff-at* (*c*, *s1*) *bT n* ≈ *t*) =
P(*bT in T.T* (*c*, *s2*). ∃ *n*. *qsend* ((*c*, *s2*) *## bT*) = *n* ∧ *eff-at* (*c*, *s2*) *bT n* ≈ *t*))

definition *aeT c* $\longleftrightarrow$ (∀ *s*. *AE bT in T.T* (*c*,*s*). *qsend* ((*c*,*s*) *## bT*) < ∞)

lemma *dist-Ps-upper-bound*:
fixes *cf1 cf2* :: ('*test*, '*atom*, '*choice*) *cmd* × '*state* **and** *s* :: '*state* **and** *S*
defines *S cf bT* ≡ ∃ *n*. *qsend* (*cf ## bT*) = *n* ∧ *eff-at cf bT n* ≈ *s*
defines *Ps cf* ≡ *P*(*bT in T.T cf*. *S cf bT*)
defines *N cf n bT* ≡ ¬ *discrCf* ((*cf ## bT*) !! *n*)
defines *Pn cf n* ≡ *P*(*bT in T.T cf*. *N cf n bT*)
assumes *bisim*: *proper* (*fst cf1*) *proper* (*fst cf2*) *fst cf1* ≈*01 fst cf2 snd cf1* ≈ *snd cf2*
shows *dist* (*Ps cf1*) (*Ps cf2*) ≤ *Pn cf1 n* + *Pn cf2 m*
using *bisim* **proof** (*induct n m arbitrary*: *cf1 cf2 rule*: *nat-nat-induct*)
case (*less n m*)
note ‹*proper* (*fst cf1*)›[*simp*, *intro*]
note ‹*proper* (*fst cf2*)›[*simp*, *intro*]

define *W* **where** *W c* = *sum* (*wt* (*fst c*) (*snd c*)) **for** *c*
from *ZObis-mC-ZOC*[*OF* ‹*fst cf1* ≈*01 fst cf2*› ‹*snd cf1* ≈ *snd cf2*›]

obtain *I0 P F* **where** *mC*: *mC-ZOC ZObis* (*fst cf1*) (*fst cf2*) (*snd cf1*) (*snd cf2*) *I0 P F* **by** *blast*
then have *P*: {} ∉ *P* − {*I0*} *part* {..<*brn* (*fst cf1*)} *P* **and** *I0* ∈ *P*
and *FP*: {} ∉ *F*‘(*P*−{*I0*}) *part* {..<*brn* (*fst cf2*)} (*F*‘*P*) *inj-on F P*
and *eff*: ⋀*q i j*. *q*∈*P* ⟹ *q*≠*I0* ⟹ *i*∈*q* ⟹ *j*∈*F q* ⟹ *eff* (*fst cf1*) (*snd cf1*) *i* ≈ *eff* (*fst cf2*) (*snd cf2*) *j*
and *cont*: ⋀*q i j*. *q*∈*P* ⟹ *q*≠*I0* ⟹ *i*∈*q* ⟹ *j*∈*F q* ⟹ *cont* (*fst cf1*) (*snd cf1*) *i* ≈*01 cont* (*fst cf2*) (*snd cf2*) *j*
and *wt*:
⋀*I*. *I* ∈ *P* − {*I0*} ⟹ *W cf1 I0* < *1* ⟹ *W cf2* (*F I0*) < *1* ⟹
W cf1 I / (*1* − *W cf1 I0*) = *W cf2* (*F I*) / (*1* − *W cf2* (*F I0*))
and *I0*:
⋀*i*. *i* ∈ *I0* ⟹ *snd cf1* ≈ *eff* (*fst cf1*) (*snd cf1*) *i*
⋀*i*. *i* ∈ *I0* ⟹ *cont* (*fst cf1*) (*snd cf1*) *i* ≈*01 fst cf2*
and *FI0*:
⋀*i*. *i* ∈ *F I0* ⟹ *snd cf2* ≈ *eff* (*fst cf2*) (*snd cf2*) *i*
⋀*i*. *i* ∈ *F I0* ⟹ *fst cf1* ≈*01 cont* (*fst cf2*) (*snd cf2*) *i*
unfolding *mC-ZOC-def mC-ZOC-part-def mC-ZOC-eff-cont-def mC-ZOC-eff-cont0-def mC-ZOC-wt-def W-def*
by *simp-all*

have *finite P inj-on F* (*P* − {*I0*}) **and** *FP'*: *finite* (*F*‘*P*) *F I0* ∈ *F*‘*P*
using *finite-part*[*OF - P*(*2*)] *finite-part*[*OF - FP*(*2*)] ‹*I0* ∈ *P*› ‹*inj-on F P*›
by (*auto intro*: *inj-on-diff*)

{ fix *I i* **assume** *I* ∈ *P i* ∈ *I*
with *P* **have** *0* ≤ *wt* (*fst cf1*) (*snd cf1*) *i*
by (*auto dest*: *part-is-subset intro*!: *wt-ge-0*) **}**
note *wt1-nneg*[*intro*] = *this*

{ fix *I i* **assume** *I* ∈ *P i* ∈ *F I*
with *FP* **have** *0* ≤ *wt* (*fst cf2*) (*snd cf2*) *i*
by (*auto dest*: *part-is-subset intro*!: *wt-ge-0*) **}**
note *wt2-nneg*[*intro*] = *this*

{ fix *I* **assume** *I* ∈ *P*
then have *0* ≤ *W cf1 I*
unfolding *W-def* **by** (*auto intro*!: *sum-nonneg*) **}**
note *W1-nneg*[*intro*] = *this*

{ fix *I* **assume** *I* ∈ *P*
then have *0* ≤ *W cf2* (*F I*)
unfolding *W-def* **by** (*auto intro*!: *sum-nonneg*) **}**
note *W2-nneg*[*intro*] = *this*

show *?case*
proof *cases*
{ fix *n cf1 cf2* **assume** ∗: *fst cf1* ≈*01 fst cf2 snd cf1* ≈ *snd cf2*
have *dist* (*Ps cf1*) (*Ps cf2*) ≤ *Pn cf1 0*

```
      proof cases
        assume cf1: discrCf cf1
        moreover
        note cf2 = ZObis-pres-discrCfR[OF cf1 *]
        from cf1 cf2 have S cf1 = (λbT. snd cf1 ≈ s) S cf2 = (λbT. snd cf2 ≈ s)
          unfolding S-def[abs-def] by (auto simp: enat-0[symmetric])
        moreover from ‹snd cf1 ≈ snd cf2› have snd cf1 ≈ s ⟷ snd cf2 ≈ s
          by (blast intro: indis-sym indis-trans)
        ultimately show ?thesis
          using T.prob-space by (cases snd cf2 ≈ s) (simp-all add: Ps-def Pn-def
measure-nonneg)
      next
        assume cf1: ¬ discrCf cf1
        then have Pn cf1 0 = 1
          using T.prob-space by (simp add: Pn-def N-def)
        moreover have dist (Ps cf1) (Ps cf2) ≤ 1
         using dist-nonneg-bounded[where u=1 and l=0 and x=Ps cf1 and y=Ps
cf2]
          by (auto simp add: dist-real-def Ps-def measure-nonneg split: abs-split)
        ultimately show ?thesis by simp
      qed }
    note base-case = this

    assume n = 0 ∨ m = 0
    then show ?thesis
    proof
      assume n = 0
      moreover
      with T.prob-space ‹fst cf1 ≈01 fst cf2› ‹snd cf1 ≈ snd cf2›
      have dist (Ps cf1) (Ps cf2) ≤ Pn cf1 0
        by (intro base-case) (auto simp: Ps-def Pn-def)
      moreover have 0 ≤ Pn cf2 m
        by (simp add: Pn-def measure-nonneg)
      ultimately show ?thesis
        by simp
    next
      assume m = 0
      moreover
      with T.prob-space ‹fst cf1 ≈01 fst cf2› ‹snd cf1 ≈ snd cf2›
      have dist (Ps cf2) (Ps cf1) ≤ Pn cf2 0
        by (intro base-case) (auto simp: Ps-def Pn-def intro: indis-sym ZObis-sym)
      moreover have 0 ≤ Pn cf1 n
        by (simp add: Pn-def measure-nonneg)
      ultimately show ?thesis
        by (simp add: dist-commute)
    qed
  next
    assume ¬ (n = 0 ∨ m = 0)
    then have n ≠ 0 m ≠ 0 by auto
```

then obtain n' m' **where** *nm*: $n = Suc\ n'$ $m = Suc\ m'$
by (*metis nat.exhaust*)

define *ps pn*
where $ps\ cf\ I = (\sum b{\in}I.\ wt\ (fst\ cf)\ (snd\ cf)\ b\ /\ W\ cf\ I * Ps\ (cont\text{-}eff\ cf\ b))$
and $pn\ cf\ I\ n = (\sum b{\in}I.\ wt\ (fst\ cf)\ (snd\ cf)\ b\ /\ W\ cf\ I * Pn\ (cont\text{-}eff\ cf\ b)\ n)$
for *cf I n*

{ **fix** *I* **assume** $I \in P\ I \neq I0$ **and** *W*: $W\ cf1\ I \neq 0\ W\ cf2\ (F\ I) \neq 0$
then have $dist\ (ps\ cf1\ I)\ (ps\ cf2\ (F\ I)) \leq pn\ cf1\ I\ n' + pn\ cf2\ (F\ I)\ m'$
unfolding *ps-def pn-def*
proof (*intro dist-weighted-sum*)
fix *i j* **assume** *ij*: $i \in I\ j \in F\ I$
assume $wt\ (fst\ cf1)\ (snd\ cf1)\ i\ /\ W\ cf1\ I \neq 0$
and $wt\ (fst\ cf2)\ (snd\ cf2)\ j\ /\ W\ cf2\ (F\ I) \neq 0$
from ‹$I \in P$› *ij P*(*2*) *FP*(*2*) **have** *br*: $i < brn\ (fst\ cf1)\ j < brn\ (fst\ cf2)$
by (*auto dest*: *part-is-subset*)
show $dist\ (Ps\ (cont\text{-}eff\ cf1\ i))\ (Ps\ (cont\text{-}eff\ cf2\ j)) \leq$
$Pn\ (cont\text{-}eff\ cf1\ i)\ n' + Pn\ (cont\text{-}eff\ cf2\ j)\ m'$
proof (*rule less.hyps*)
show $n' + m' < n + m$ **using** *nm* **by** *simp*
show $proper\ (fst\ (cont\text{-}eff\ cf1\ i))\ proper\ (fst\ (cont\text{-}eff\ cf2\ j))$
using *br less.prems* **by** (*auto simp*: *cont-eff*)
show $fst\ (cont\text{-}eff\ cf1\ i) \approx 01\ fst\ (cont\text{-}eff\ cf2\ j)$
$snd\ (cont\text{-}eff\ cf1\ i) \approx snd\ (cont\text{-}eff\ cf2\ j)$
using *cont*[*OF* ‹$I \in P$› ‹$I \neq I0$› *ij*] *eff*[*OF* ‹$I \in P$› ‹$I \neq I0$› *ij*] **by** (*auto simp*: *cont-eff*)
qed
next
show $(\sum b{\in}F\ I.\ wt\ (fst\ cf2)\ (snd\ cf2)\ b\ /\ W\ cf2\ (F\ I)) = 1$
$(\sum b{\in}I.\ wt\ (fst\ cf1)\ (snd\ cf1)\ b\ /\ W\ cf1\ I) = 1$
using *W* **by** (*auto simp*: *sum-divide-distrib*[*symmetric*] *sum-nonneg W-def*)
qed *auto* }
note *dist-n'-m'* = *this*

{ **fix** *I* **assume** $I \in P\ I \neq I0$ **and** *W*: $W\ cf1\ I = 0 \longleftrightarrow W\ cf2\ (F\ I) = 0$
have $dist\ (ps\ cf1\ I)\ (ps\ cf2\ (F\ I)) \leq pn\ cf1\ I\ n' + pn\ cf2\ (F\ I)\ m'$
proof *cases*
assume $W\ cf2\ (F\ I) = 0$
then have $W\ cf2\ (F\ I) = 0\ W\ cf1\ I = 0$ **by** (*auto simp*: *W*)
then show *?thesis* **by** (*simp add*: *ps-def pn-def*)
next
assume $W\ cf2\ (F\ I) \neq 0$
then have $W\ cf1\ I \neq 0\ W\ cf2\ (F\ I) \neq 0$ **by** (*auto simp*: *W*)
from *dist-n'-m'*[*OF* ‹$I \in P$› ‹$I \neq I0$› *this*] **show** *?thesis* .
qed }
note *dist-n'-m'-W-iff* = *this*

```
  { fix I j assume W: W cf2 (F I0) ≠ 0
    from ‹I0 ∈ P› have dist (∑ i∈{()}. 1 * Ps cf1) (ps cf2 (F I0)) ≤ (∑ i∈{()}.
1 * Pn cf1 n) + pn cf2 (F I0) m'
      unfolding ps-def pn-def
     proof (intro dist-weighted-sum)
      fix j assume j ∈ F I0
      with FP(2) ‹I0 ∈ P› have br: j < brn (fst cf2)
       by (auto dest: part-is-subset)
     show dist (Ps cf1) (Ps (cont-eff cf2 j)) ≤ Pn cf1 n + Pn (cont-eff cf2 j) m'
      proof (rule less.hyps)
       show n + m' < n + m using nm by simp
       show proper (fst cf1) proper (fst (cont-eff cf2 j))
        using br by (auto simp: cont-eff)
       show fst cf1 ≈01 fst (cont-eff cf2 j)
        snd cf1 ≈ snd (cont-eff cf2 j)
        using FI0[OF ‹j ∈ F I0›] ‹snd cf1 ≈ snd cf2›
        by (auto simp: cont-eff intro: indis-trans)
      qed
     next
      show (∑ b∈F I0. wt (fst cf2) (snd cf2) b / W cf2 (F I0)) = 1
      using W ‹I0 ∈ P› by (auto simp: sum-divide-distrib[symmetric] sum-nonneg
W-def)
     qed auto
     then have dist (Ps cf1) (ps cf2 (F I0)) ≤ Pn cf1 n + pn cf2 (F I0) m'
      by simp }
   note dist-n-m' = this

   { fix I j assume W: W cf1 I0 ≠ 0
     from ‹I0 ∈ P› have dist (ps cf1 I0) (∑ i∈{()}. 1 * Ps cf2) ≤ pn cf1 I0 n'
+ (∑ i∈{()}. 1 * Pn cf2 m)
      unfolding ps-def pn-def
     proof (intro dist-weighted-sum)
      fix i assume i ∈ I0
      with P(2) ‹I0 ∈ P› have br: i < brn (fst cf1)
       by (auto dest: part-is-subset)
     show dist (Ps (cont-eff cf1 i)) (Ps cf2) ≤ Pn (cont-eff cf1 i) n' + Pn cf2 m
      proof (rule less.hyps)
       show n' + m < n + m using nm by simp
       show proper (fst (cont-eff cf1 i)) proper (fst cf2)
        using br less.prems by (auto simp: cont-eff)
       show fst (cont-eff cf1 i) ≈01 fst cf2
        snd (cont-eff cf1 i) ≈ snd cf2
        using I0[OF ‹i ∈ I0›] ‹snd cf1 ≈ snd cf2›
        by (auto simp: cont-eff intro: indis-trans indis-sym)
      qed
     next
      show (∑ b∈I0. wt (fst cf1) (snd cf1) b / W cf1 I0) = 1
      using W ‹I0 ∈ P› by (auto simp: sum-divide-distrib[symmetric] sum-nonneg
W-def)
```

qed *auto*
then have *dist* (*ps cf1 I0*) (*Ps cf2*) ≤ *pn cf1 I0 n′* + *Pn cf2 m*
by *simp* **}**
note *dist-n′-m* = *this*

have *S-measurable*[*measurable*]: ⋀*cf*. *Measurable.pred T.S* (*S cf*)
unfolding *S-def*[*abs-def*]
by *measurable*

{ fix *cf′ cf* **assume** *cf*[*simp*]: *proper* (*fst cf*) **and** *cf′*: *cf′* ∈ *G cf*
let *?S* = λ*bT n*. *qsend bT* = *enat n* ∧ *snd* (*bT* !! *n*) ≈ *s*
{ fix *bT n* **assume** ∗: *?S* (*cf* ## *cf′* ## *bT*) *n* **have** *S cf′ bT*
proof (*cases n*)
case *0* **with** ∗ *cf′* **show** *?thesis*
by (*auto simp*: *S-def enat-0 sfirst-Stream G-eq split*: *if-split-asm intro*!: *exI*[*of - 0*])
(*blast intro*: *indis-trans indis-sym discrCf-eff-indis*)
next
case (*Suc n*) **with** ∗ *cf′* **show** *S cf′ bT*
by (*auto simp*: *eSuc-enat*[*symmetric*] *sfirst-Stream S-def split*: *if-split-asm*)
qed }
moreover
{ fix *bT n* **assume** *?S* (*cf′* ## *bT*) *n* **with** *cf′* **have** *?S* (*cf* ## *cf′* ## *bT*) (*if discrCf cf then 0 else Suc n*)
by (*auto simp*: *eSuc-enat*[*symmetric*] *enat-0*[*symmetric*] *sfirst-Stream G-eq split*: *if-split-asm*)
(*blast intro*: *indis-trans indis-sym discrCf-eff-indis*)
then have *S cf* (*cf′* ## *bT*)
by (*auto simp*: *S-def*) **}**
ultimately have *AE bT in T.T cf′*. *S cf* (*cf′* ## *bT*) = *S cf′ bT*
by (*auto simp*: *S-def*) **}**
note *S-sets* = *this*

{ fix *cf* :: (′*test*, ′*atom*, ′*choice*) *cmd* × ′*state* **and** *P I0 S n st*
assume *cf*[*simp*]: *proper* (*fst cf*) **and** *P*: *part* {..<*brn* (*fst cf*)} *P* **and** *finite P I0* ∈ *P*

{ fix *I i* **assume** *I* ∈ *P i* ∈ *I*
with *P* **have** *0* ≤ *wt* (*fst cf*) (*snd cf*) *i*
by (*auto dest*: *part-is-subset intro*!: *wt-ge-0*) **}**
note *wt-nneg*[*simp*] = *this*

assume *S-measurable*[*measurable*]: ⋀*cf n*. *Measurable.pred T.S* (λ*bT*. *S cf n bT*)
and *S-next*: ⋀*cf cf′ n*. *proper* (*fst cf*) ⟹ *cf′* ∈ *G cf* ⟹
AE bT in T.T cf′. *S cf* (*Suc n*) (*cf′* ## *bT*) = *S cf′ n bT*
have *finite-I*: ⋀*I*. *I* ∈ *P* ⟹ *finite I*
using *finite-subset*[*OF part-is-subset*[*OF P*]] **by** *blast*
let *?P* = λ*I*. ∑*i*∈*I*. *wt* (*fst cf*) (*snd cf*) *i* ∗ *P*(*bT in T.T* (*cont-eff cf i*). *S*

(*cont-eff cf i*) *n bT*)
let *?P′* = *λI*. *W cf I* ∗ (∑ *i*∈*I*. *wt* (*fst cf*) (*snd cf*) *i* / *W cf I* ∗ *P*(*bT in T.T* (*cont-eff cf i*). *S* (*cont-eff cf i*) *n bT*))
have *P*(*bT in T.T cf*. *S cf* (*Suc n*) *bT*) = (∫ *cf′*. *P*(*bT in T.T cf′*. *S cf′ n bT*) *∂trans cf*)
apply (*subst T.prob-T*[*OF S-measurable*])
apply (*intro integral-cong-AE*)
apply (*auto simp*: *AE-measure-pmf-iff intro*!: *T.prob-eq-AE S-next simp del*: *space-T*)
apply *auto*
done
also have ... = (∑ *I*∈*P*. *?P I*)
unfolding *integral-trans*[*OF cf*] **by** (*simp add*: *part-sum*[*OF P*])
also have ... = (∑ *I*∈*P*. *?P′ I*)
using *finite-I*
by (*auto intro*!: *sum.cong simp add*: *sum-distrib-left sum-nonneg-eq-0-iff W-def*)
also have ... = *?P′ I0* + (∑ *I*∈*P*−{*I0*}. *?P′ I*)
unfolding *sum.remove*[*OF* ‹*finite P*› ‹*I0* ∈ *P*›] **..**
finally have *P*(*bT in T.T cf*. *S cf* (*Suc n*) *bT*) = ... **. }**
note *P-split* = *this*

have *Ps1*: *Ps cf1* = *W cf1 I0* ∗ *ps cf1 I0* + (∑ *I*∈*P*−{*I0*}. *W cf1 I* ∗ *ps cf1 I*)
unfolding *Ps-def ps-def* **using** *P*(*2*) ‹*finite P*› ‹*I0* ∈ *P*› **by** (*intro P-split S-sets*) *simp-all*

have *Ps cf2* = *W cf2* (*F I0*) ∗ *ps cf2* (*F I0*) + (∑ *I*∈*F*‘*P*−{*F I0*}. *W cf2 I* ∗ *ps cf2 I*)
unfolding *Ps-def ps-def* **using** *FP*(*2*) ‹*finite P*› ‹*I0* ∈ *P*› **by** (*intro P-split S-sets*) *simp-all*
moreover have *F-diff*: *F* ‘ *P* − {*F I0*} = *F* ‘ (*P* − {*I0*})
by (*auto simp*: ‹*inj-on F P*›[*THEN inj-on-eq-iff*] ‹*I0* ∈ *P*›)
ultimately have *Ps2*: *Ps cf2* = *W cf2* (*F I0*) ∗ *ps cf2* (*F I0*) + (∑ *I*∈*P*−{*I0*}. *W cf2* (*F I*) ∗ *ps cf2* (*F I*))
by (*simp add*: *sum.reindex* ‹*inj-on F* (*P*−{*I0*})›)

have *Pn1*: *Pn cf1 n* = *W cf1 I0* ∗ *pn cf1 I0 n′* + (∑ *I*∈*P*−{*I0*}. *W cf1 I* ∗ *pn cf1 I n′*)
unfolding *Pn-def pn-def nm* **using** *P*(*2*) ‹*finite P*› ‹*I0* ∈ *P*› **by** (*intro P-split*) (*simp-all add*: *N-def*)

have *Pn cf2 m* = *W cf2* (*F I0*) ∗ *pn cf2* (*F I0*) *m′* + (∑ *I*∈*F*‘*P*−{*F I0*}. *W cf2 I* ∗ *pn cf2 I m′*)
unfolding *Pn-def pn-def nm* **using** *FP*(*2*) ‹*finite P*› ‹*I0* ∈ *P*› **by** (*intro P-split*) (*simp-all add*: *N-def*)
with *F-diff* **have** *Pn2*: *Pn cf2 m* = *W cf2* (*F I0*) ∗ *pn cf2* (*F I0*) *m′* + (∑ *I*∈*P*−{*I0*}. *W cf2* (*F I*) ∗ *pn cf2* (*F I*) *m′*)
by (*simp add*: *sum.reindex* ‹*inj-on F* (*P*−{*I0*})›)

show *?thesis*
proof *cases*
assume *W cf1 I0 = 1 ∨ W cf2 (F I0) = 1*
then show *?thesis*
proof
assume ∗: *W cf1 I0 = 1*
then have *W cf1 I0 = W cf1 {..<brn (fst cf1)}* **by** (*simp add*: *W-def*)
also have *... = W cf1 I0 + (∑ I∈P − {I0}. W cf1 I)*
unfolding ‹*part {..<brn (fst cf1)} P*›[*THEN part-sum*] *W-def*
unfolding *sum.remove*[*OF* ‹*finite P*› ‹*I0 ∈ P*›] **..**
finally have *(∑ I∈P − {I0}. W cf1 I) = 0* **by** *simp*
then have *∀ I∈P − {I0}. W cf1 I = 0*
using ‹*finite P*› **by** (*subst* (*asm*) *sum-nonneg-eq-0-iff*) *auto*
then have *Ps cf1 = ps cf1 I0 Pn cf1 n = pn cf1 I0 n′*
unfolding *Ps1 Pn1* ∗ **by** *simp-all*
moreover note *dist-n′-m* ∗
ultimately show *?thesis* **by** *simp*
next
assume ∗: *W cf2 (F I0) = 1*
then have *W cf2 (F I0) = W cf2 {..<brn (fst cf2)}* **by** (*simp add*: *W-def*)
also have *... = W cf2 (F I0) + (∑ I∈F ‘ P − {F I0}. W cf2 I)*
unfolding *FP*(*2*)[*THEN part-sum*] *W-def*
unfolding *sum.remove*[*OF FP′*] **..**
finally have *(∑ I∈F‘P − {F I0}. W cf2 I) = 0* **by** *simp*
then have *∀ I∈F‘P − {F I0}. W cf2 I = 0*
using ‹*finite P*› **by** (*subst* (*asm*) *sum-nonneg-eq-0-iff*) *auto*
then have *Ps cf2 = ps cf2 (F I0) Pn cf2 m = pn cf2 (F I0) m′*
unfolding *Ps2 Pn2* ∗ **by** (*simp-all add*: *F-diff*)
moreover note *dist-n-m′* ∗
ultimately show *?thesis* **by** *simp*
qed
next
assume *W-neq1*: ¬ (*W cf1 I0 = 1 ∨ W cf2 (F I0) = 1*)
moreover
{ fix *cf* :: (*′test*, *′atom*, *′choice*) *cmd* × *′state* **and** *I P*
assume *proper* (*fst cf*) *part {..<brn* (*fst* (*cf*))} *P I ∈ P*
then have *W cf I ≤ W cf {..<brn* (*fst* (*cf*))}
unfolding *W-def*
by (*intro sum-mono2 part-is-subset*) *auto*
then have *W cf I ≤ 1* **using** ‹*proper* (*fst cf*)› **by** (*simp add*: *W-def*) **}**
ultimately have *wt-less1*: *W cf1 I0 < 1 W cf2 (F I0) < 1*
using *FP*(*2*) *FP′*(*2*) *P*(*2*) ‹*I0 ∈ P*›
unfolding *le-less* **by** *blast*+

{ fix *I* **assume** ∗: *I ∈ P − {I0}*
have *W cf1 I = 0 ⟷ W cf2 (F I) = 0*
using *wt*[*OF* ∗ *wt-less1*] *wt-less1* **by** *auto*
with ∗ **have** *dist (ps cf1 I) (ps cf2 (F I)) ≤ pn cf1 I n′ + pn cf2 (F I) m′*

```
      by (intro dist-n'-m'-W-iff) auto }
    note dist-eps = this

    { fix a b c d :: real
      have dist a b = dist ((a − c) + c) ((b − d) + d) by simp
      also have ... ≤ dist (a − c) (b − d) + dist c d
        using dist-triangle-add .
      finally have dist a b ≤ dist (a − c) (b − d) + dist c d . }
    note dist-triangle-diff = this

    have dist-diff-diff: ⋀a b c d::real. dist (a − b) (c − d) ≤ dist a b + dist c d
      unfolding dist-real-def by auto

    let ?v0 = W cf1 I0 and ?w0 = W cf2 (F I0)
    let ?v1 = 1 − ?v0 and ?w1 = 1 − ?w0
    let ?wQ = (Ps cf2 − ?w0 * ps cf2 (F I0)) / ?w1
    let ?wP = (Ps cf1 − ?v0 * ps cf1 I0) / ?v1
    let ?D = (?w0 * ?v1 * Ps cf1 + ?w1 * ?v0 * ps cf1 I0)
    let ?E = (?v0 * ?w1 * Ps cf2 + ?v1 * ?w0 * ps cf2 (F I0))

    have w0v0-less1: ?w0 * ?v0 < 1 * 1
      using wt-less1 ‹I0 ∈ P› by (intro mult-strict-mono) auto
    then have neg-w0v0-nonneg: 0 ≤ 1 − ?w0 * ?v0 by simp

    let ?e1 = (∑ I∈P−{I0}. W cf1 I / ?v1 * pn cf1 I n') +
        (∑ I∈P−{I0}. W cf2 (F I) / ?w1 * pn cf2 (F I) m')
    have dist ((1 − ?w0 * ?v0) * Ps cf1) ((1 − ?w0 * ?v0) * Ps cf2) ≤
      dist ((1 − ?w0 * ?v0) * Ps cf1 − ?D) ((1 − ?w0 * ?v0) * Ps cf2 − ?E)
+ dist ?D ?E
      by (rule dist-triangle-diff)
    also have ... ≤ ?v1 * ?w1 * ?e1 + (?v1 * ?w0 * (Pn cf1 n + pn cf2 (F I0)
m') + ?w1 * ?v0 * (pn cf1 I0 n' + Pn cf2 m))
    proof (rule add-mono)
      { have ?wP = (∑ I∈P−{I0}. W cf1 I * ps cf1 I) / ?v1
          unfolding Ps1 by (simp add: field-simps)
        also have ... = (∑ I∈P−{I0}. W cf1 I / ?v1 * ps cf1 I)
          by (subst sum-divide-distrib) simp
        finally have ?wP = (∑ I∈P−{I0}. W cf1 I / ?v1 * ps cf1 I) . }
      moreover
      { have ?wQ = (∑ I∈P−{I0}. W cf2 (F I) * ps cf2 (F I)) / ?w1
          using Ps2 by (simp add: field-simps)
        also have ... = (∑ I∈P−{I0}. W cf2 (F I) / ?w1 * ps cf2 (F I))
          by (subst sum-divide-distrib) simp
        also have ... = (∑ I∈P−{I0}. W cf1 I / ?v1 * ps cf2 (F I))
          using wt[OF - wt-less1] by simp
        finally have ?wQ = (∑ I∈P−{I0}. W cf1 I / ?v1 * ps cf2 (F I)) . }
      ultimately
      have dist ?wP ?wQ ≤ (∑ I∈P−{I0}. W cf1 I / ?v1 * (pn cf1 I n' + pn
cf2 (F I) m'))
```

using *wt-less1 dist-eps*
by (*simp*, *intro dist-sum*)
(*simp add*: *sum-nonneg divide-le-cancel mult-le-cancel-left not-le*[*symmetric*] *W1-nneg*)
also have ... = *?e1*
unfolding *sum.distrib*[*symmetric*] **using** *wt*[*OF - wt-less1*]
by (*simp add*: *field-simps add-divide-distrib*)
finally have *dist* (*?v1* ∗ *?w1* ∗ *?wP*) (*?v1* ∗ *?w1* ∗ *?wQ*) ≤ *?v1* ∗ *?w1* ∗ *?e1*
using *wt-less1* **unfolding** *dist-mult* **by** *simp*
also {
have *?v1* ∗ *?w1* ∗ *?wP* = *?w1* ∗ (*?v0* ∗ *Ps cf1* + *?v1* ∗ *Ps cf1*) − *?w1* ∗ *?v0* ∗ *ps cf1 I0*
using *wt-less1* **unfolding** *divide-eq-eq* **by** (*simp add*: *field-simps*)
also have ... = (1 − *?w0* ∗ *?v0*) ∗ *Ps cf1* − *?D*
by (*simp add*: *field-simps*)
finally have *?v1* ∗ *?w1* ∗ *?wP* = (1 − *?w0* ∗ *?v0*) ∗ *Ps cf1* − *?D* **. }**
also {
have *?v1* ∗ *?w1* ∗ *?wQ* = *?v1* ∗ (*?w0* ∗ *Ps cf2* + *?w1* ∗ *Ps cf2*) − *?v1* ∗ *?w0* ∗ (*ps cf2* (*F I0*))
using *wt-less1* **unfolding** *divide-eq-eq* **by** (*simp add*: *field-simps*)
also have ... = (1 − *?w0* ∗ *?v0*) ∗ *Ps cf2* − *?E*
by (*simp add*: *field-simps*)
finally have *?v1* ∗ *?w1* ∗ *?wQ* = (1 − *?w0* ∗ *?v0*) ∗ *Ps cf2* − *?E* **. }**
finally show *dist* ((1 − *?w0* ∗ *?v0*) ∗ *Ps cf1* − *?D*) ((1 − *?w0* ∗ *?v0*) ∗ *Ps cf2* − *?E*) ≤ *?v1* ∗ *?w1* ∗ *?e1* **.**
next
have *dist ?D ?E* = *dist*
(*?v1* ∗ *?w0* ∗ *Ps cf1* − *?v1* ∗ *?w0* ∗ *ps cf2* (*F I0*))
(*?w1* ∗ *?v0* ∗ *Ps cf2* − *?w1* ∗ *?v0* ∗ *ps cf1 I0*)
unfolding *dist-real-def* **by** (*simp add*: *ac-simps*)
also have ... ≤ *dist* (*?v1* ∗ *?w0* ∗ *Ps cf1*) (*?v1* ∗ *?w0* ∗ *ps cf2* (*F I0*)) +
dist (*?w1* ∗ *?v0* ∗ *Ps cf2*) (*?w1* ∗ *?v0* ∗ *ps cf1 I0*)
using *dist-diff-diff* **.**
also have ... ≤ *?v1* ∗ *?w0* ∗ (*Pn cf1 n* + *pn cf2* (*F I0*) *m′*) + *?w1* ∗ *?v0* ∗ (*pn cf1 I0 n′* + *Pn cf2 m*)
proof (*rule add-mono*)
show *dist* (*?v1* ∗ *?w0* ∗ *Ps cf1*) (*?v1* ∗ *?w0* ∗ *ps cf2* (*F I0*)) ≤ *?v1* ∗ *?w0* ∗ (*Pn cf1 n* + *pn cf2* (*F I0*) *m′*)
using *wt-less1 dist-n-m′* ‹*I0* ∈ *P*›
by (*simp add*: *sum-nonneg mult-le-cancel-left not-le*[*symmetric*] *mult-le-0-iff W2-nneg*)
show *dist* (*?w1* ∗ *?v0* ∗ *Ps cf2*) (*?w1* ∗ *?v0* ∗ *ps cf1 I0*) ≤ *?w1* ∗ *?v0* ∗ (*pn cf1 I0 n′* + *Pn cf2 m*)
using *wt-less1 dist-n′-m* ‹*I0* ∈ *P*›
by (*subst dist-commute*)
(*simp add*: *sum-nonneg mult-le-cancel-left not-le*[*symmetric*] *mult-le-0-iff W1-nneg*)
qed

finally show *dist ?D ?E* $\leq$ *?v1* $*$ *?w0* $*$ (*Pn cf1 n* + *pn cf2* (*F I0*) *m'*) + *?w1* $*$ *?v0* $*$ (*pn cf1 I0 n'* + *Pn cf2 m*) **.**
qed
also have ... = *?w1* $*$ ($\sum$ *I*$\in$*P*$-${*I0*}. *W cf1 I* $*$ *pn cf1 I n'*) + *?v1* $*$ ($\sum$ *I*$\in$*P*$-${*I0*}. *W cf2* (*F I*) $*$ *pn cf2* (*F I*) *m'*) +
?v1 $*$ *?w0* $*$ (*Pn cf1 n* + *pn cf2* (*F I0*) *m'*) + *?w1* $*$ *?v0* $*$ (*pn cf1 I0 n'* + *Pn cf2 m*)
using *W-neq1* **by** (*simp add*: *sum-divide-distrib*[*symmetric*] *add-divide-eq-iff divide-add-eq-iff*)
also have ... = (*1* $-$ *?w0* $*$ *?v0*) $*$ (*Pn cf1 n* + *Pn cf2 m*)
unfolding *Pn1 Pn2* **by** (*simp add*: *field-simps*)
finally show *dist* (*Ps cf1*) (*Ps cf2*) $\leq$ *Pn cf1 n* + *Pn cf2 m*
using *neg-w0v0-nonneg w0v0-less1* **by** (*simp add*: *mult-le-cancel-left*)
qed
qed
qed

lemma *AE-T-max-qsend-time*:
fixes *cf* **and** *e* :: *real* **assumes** *AE*: *AE bT in T.T cf*. *qsend* (*cf ## bT*) $<$ ∞ *0* $<$ *e*
shows $\exists$ *N*. $\mathcal{P}$(*bT in T.T cf*. $\neg$ *discrCf* ((*cf ## bT*) !! *N*)) $<$ *e*
proof $-$
from *AE-T-max-sfirst*[*OF - AE*] **obtain** *N* :: *nat*
where $\mathcal{P}$(*bT in T.T cf*. *N* $<$ *qsend* (*cf ## bT*)) $<$ *e*
by *auto*
also have $\mathcal{P}$(*bT in T.T cf*. *N* $<$ *qsend* (*cf ## bT*)) = $\mathcal{P}$(*bT in T.T cf*. $\neg$ *discrCf* ((*cf ## bT*) !! *N*))
using *less-qsend-iff-not-discrCf*[*of cf*] *AE-T-enabled*[*of cf*]
by (*intro T.prob-eq-AE*) *auto*
finally show *?thesis* **..**
qed

lemma *Ps-eq*:
fixes *cf1 cf2 s* **and** *S*
defines *S cf bT* $\equiv$ $\exists$ *n*. *qsend* (*cf ## bT*) = *n* $\wedge$ *eff-at cf bT n* $\approx$ *s*
defines *Ps cf* $\equiv$ $\mathcal{P}$(*bT in T.T cf*. *S cf bT*)
assumes *qsterm1*: *AE bT in T.T cf1*. *qsend* (*cf1 ## bT*) $<$ ∞
assumes *qsterm2*: *AE bT in T.T cf2*. *qsend* (*cf2 ## bT*) $<$ ∞
and *bisim*: *proper* (*fst cf1*) *proper* (*fst cf2*) *fst cf1* $\approx$*01* *fst cf2 snd cf1* $\approx$ *snd cf2*
shows *Ps cf1* = *Ps cf2*
proof $-$
let *?nT* = λ*cf n bT*. $\neg$ *discrCf* ((*cf ## bT*) !! *n*)
let *?PnT* = λ*cf n*. $\mathcal{P}$(*bT in T.T cf*. *?nT cf n bT*)

have *dist* (*Ps cf1*) (*Ps cf2*) = *0*
unfolding *dist-real-def*
proof (*rule field-abs-le-zero-epsilon*)
fix *e* ::*real* **assume** *0* $<$ *e*
then have *0* $<$ *e* / *2* **by** *auto*

```
  from AE-T-max-qsend-time[OF qsterm1 this] AE-T-max-qsend-time[OF qsterm2
this]
    obtain n m where ?PnT cf1 n < e / 2 ?PnT cf2 m < e / 2 by auto
    moreover have dist (Ps cf1) (Ps cf2) ≤ ?PnT cf1 n + ?PnT cf2 m
      unfolding Ps-def S-def using bisim by (rule dist-Ps-upper-bound)
    ultimately show |Ps cf1 − Ps cf2| ≤ e
      unfolding dist-real-def by auto
  qed
  then show Ps cf1 = Ps cf2 by auto
qed

lemma siso-trace:
  assumes siso c s ≈ t enabled (c, t) cfT
  shows siso (cont-at (c, s) cfT n)
    and cont-at (c, s) cfT n = cont-at (c, t) cfT n
    and eff-at (c, s) cfT n ≈ eff-at (c, t) cfT n
  using assms
proof (induction n arbitrary: c s t cfT)
  case (Suc n) case 1
  with Suc(1)[of fst (shd cfT) snd (shd cfT) snd (shd cfT) stl cfT] show ?case
   by (auto simp add: enabled.simps[of - cfT] G-eq cont-eff indis-refl split: if-split-asm)
qed auto

lemma Sbis-trace:
  assumes proper (fst cf1) proper (fst cf2) fst cf1 ≈s fst cf2 snd cf1 ≈ snd cf2
  shows P(cfT in T.T cf1. eff-at cf1 cfT n ≈ s′) = P(cfT in T.T cf2. eff-at cf2
cfT n ≈ s′)
    (is ?P cf1 n = ?P cf2 n)
using assms proof (induct n arbitrary: cf1 cf2)
  case 0
  show ?case
  proof cases
    assume snd cf1 ≈ s′
    with ‹snd cf1 ≈ snd cf2› ‹fst cf1 ≈s fst cf2› have snd cf1 ≈ s′ snd cf2 ≈ s′
      by (metis indis-trans indis-sym)+
    then show ?case
      using T.prob-space by simp
  next
    assume ¬ snd cf1 ≈ s′
    with ‹snd cf1 ≈ snd cf2› ‹fst cf1 ≈s fst cf2› have ¬ snd cf1 ≈ s′ ∧ ¬ snd cf2
≈ s′
      by (metis indis-trans indis-sym)
    then show ?case
      by auto
  qed
next
  case (Suc n)
  note ‹proper (fst cf1)›[simp] ‹proper (fst cf2)›[simp]
```

from *Sbis-mC-C* ‹*fst cf1* ≈*s fst cf2*› ‹*snd cf1* ≈ *snd cf2*›
obtain *P F* **where** *mP*: *mC-C Sbis* (*fst cf1*) (*fst cf2*) (*snd cf1*) (*snd cf2*) *P F*
by *blast*
then have
P: *part* {..<*brn* (*fst cf1*)} *P* {} ∉ *P* **and**
FP: *part* {..<*brn* (*fst cf2*)} (*F*‘*P*) {} ∉ *F* ‘ *P inj-on F P* **and**
W: ⋀*I*. *I* ∈ *P* ⟹ *sum* (*wt* (*fst cf1*) (*snd cf1*)) *I* = *sum* (*wt* (*fst cf2*) (*snd cf2*)) (*F I*) **and**
eff: ⋀*I i j*. *I* ∈ *P* ⟹ *i* ∈ *I* ⟹ *j* ∈ *F I* ⟹
eff (*fst cf1*) (*snd cf1*) *i* ≈ *eff* (*fst cf2*) (*snd cf2*) *j* **and**
cont: ⋀*I i j*. *I* ∈ *P* ⟹ *i* ∈ *I* ⟹ *j* ∈ *F I* ⟹
cont (*fst cf1*) (*snd cf1*) *i* ≈*s cont* (*fst cf2*) (*snd cf2*) *j*
unfolding *mC-C-def mC-C-eff-cont-def mC-C-part-def mC-C-wt-def* **by** *metis*+
{ fix *cf1* :: (′*test*, ′*atom*, ′*choice*) *cmd* × ′*state* **and** *P* **assume** *cf*[*simp*]: *proper* (*fst cf1*) **and** *P*: *part* {..<*brn* (*fst cf1*)} *P*
have *?P cf1* (*Suc n*) = (∫ *cf*′. *?P cf*′ *n* ∂*trans cf1*)
by (*subst T*.*prob-T*) *auto*
also have ... = (∑ *b*<*brn* (*fst cf1*). *wt* (*fst cf1*) (*snd cf1*) *b* ∗ *?P* (*cont-eff cf1 b*) *n*)
unfolding *integral-trans*[*OF cf*] **..**
also have ... = (∑ *I*∈*P*. ∑ *b*∈*I*. *wt* (*fst cf1*) (*snd cf1*) *b* ∗ *?P* (*cont-eff cf1 b*) *n*)
unfolding *part-sum*[*OF P*] **..**
finally have *?P cf1* (*Suc n*) = ... **. }**
note *split* = *this*

{ fix *I i* **assume** *I* ∈ *P i* ∈ *I*
with ‹*proper* (*fst cf1*)› **have** *i* < *brn* (*fst cf1*)
using *part-is-subset*[*OF P*(*1*) ‹*I* ∈ *P*›] **by** *auto* **}**
note *brn-cf*[*simp*] = *this*

{ fix *I i* **assume** *I* ∈ *P i* ∈ *F I*
with ‹*proper* (*fst cf2*)› **have** *i* < *brn* (*fst cf2*)
using *part-is-subset*[*OF FP*(*1*), *of F I*] **by** *auto* **}**
note *brn-cf2*[*simp*] = *this*

{ fix *I* **assume** *I* ∈ *P*
with ‹{} ∉ *P*› **obtain** *i* **where** *i* ∈ *I* **by** (*metis all-not-in-conv*)
from ‹*I* ∈ *P*› *FP* **have** *F I* ≠ {} *F I* ⊆ {..<*brn* (*fst cf2*)}
by (*auto simp*: *part-is-subset*)
then obtain *j* **where** *j* < *brn* (*fst cf2*) *j* ∈ *F I* **by** *auto*
{ fix *b* **assume** *b* ∈ *F I*
then have *?P* (*cont-eff cf1 i*) *n* = *?P* (*cont-eff cf2 b*) *n*
using ‹*I* ∈ *P*› ‹*i* ∈ *I*› *cont eff*
by (*intro Suc*) (*auto simp add*: *cont-eff*) **}**
note *cont-d-const* = *this*[*symmetric*]
{ fix *a* **assume** *a* ∈ *I*
with ‹*I* ∈ *P*› ‹*i* ∈ *I*› ‹*j* ∈ *F I*› *cont eff*
have *?P* (*cont-eff cf1 i*) *n* = *?P* (*cont-eff cf2 j*) *n* ∧

```
      ?P (cont-eff cf1 a) n = ?P (cont-eff cf2 j) n
      by (intro conjI Suc) (auto simp add: cont-eff)
    then have ?P (cont-eff cf1 a) n = ?P (cont-eff cf1 i) n by simp }
  then have (∑ b∈I. wt (fst cf1) (snd cf1) b * ?P (cont-eff cf1 b) n) =
      (∑ b∈I. wt (fst cf1) (snd cf1) b) * ?P (cont-eff cf1 i) n
    by (simp add: sum-distrib-right)
  also have ... = (∑ b∈F I. wt (fst cf2) (snd cf2) b) * ?P (cont-eff cf1 i) n
    using W ‹I ∈ P› by auto
  also have ... = (∑ b∈F I. wt (fst cf2) (snd cf2) b * ?P (cont-eff cf2 b) n)
    using cont-d-const by (auto simp add: sum-distrib-right)
  finally have (∑ b∈I. wt (fst cf1) (snd cf1) b * ?P (cont-eff cf1 b) n) = ... . }
 note sum-eq = this

 have ?P cf1 (Suc n) = (∑ I∈P. ∑ b∈I. wt (fst cf1) (snd cf1) b * ?P (cont-eff
cf1 b) n)
   using ‹proper (fst cf1)› P(1) by (rule split)
 also have ... = (∑ I∈P. ∑ b∈F I. wt (fst cf2) (snd cf2) b * ?P (cont-eff cf2
b) n)
   using sum-eq by simp
 also have ... = (∑ I∈F`P. ∑ b∈I. wt (fst cf2) (snd cf2) b * ?P (cont-eff cf2
b) n)
   using ‹inj-on F P› by (simp add: sum.reindex)
 also have ... = ?P cf2 (Suc n)
   using ‹proper (fst cf2)› FP(1) by (rule split[symmetric])
 finally show ?case .
qed
```

3.4 Final Theorems

```
theorem ZObis-eSec: ⟦proper c; c ≈01 c; aeT c⟧ ⟹ eSec c
 by (auto simp: aeT-def eSec-def intro!: Ps-eq[simplified])

theorem Sbis-amSec: ⟦proper c; c ≈s c⟧ ⟹ amSec c
 by (auto simp: amSec-def intro!: Sbis-trace[simplified])

theorem amSec-eSec:
 assumes [simp]: proper c and aeT c amSec c shows eSec c
proof (unfold eSec-def, intro allI impI)
 fix s1 s2 t assume s1 ≈ s2
 let ?T = λs bT. ∃ n. qsend ((c, s) ## bT) = n ∧ eff-at (c,s) bT n ≈ t
 let ?P = λs. 𝒫(bT in T.T (c, s). ?T s bT)

 have dist (?P s1) (?P s2) = 0
   unfolding dist-real-def
 proof (rule field-abs-le-zero-epsilon)
   fix e :: real assume 0 < e
   then have 0 < e / 2 by simp
   let ?N = λs n bT. ¬ discrCf (((c,s) ## bT) !! n)
   from AE-T-max-qsend-time[OF - ‹0 < e / 2›, of (c,s1)]
```

```
    obtain N1 where N1: 𝒫(bT in T.T (c, s1). ?N s1 N1 bT) < e / 2
      using ‹aeT c› unfolding aeT-def by auto
    from AE-T-max-qsend-time[OF - ‹0 < e / 2›, of (c,s2)]
    obtain N2 where N2: 𝒫(bT in T.T (c, s2). ?N s2 N2 bT) < e / 2
      using ‹aeT c› unfolding aeT-def by auto
    define N where N = max N1 N2

    let ?Tn = λn s bT. eff-at (c,s) bT n ≈ t

   have dist 𝒫(bT in T.T (c, s1). ?T s1 bT) 𝒫(bT in T.T (c, s1). ?Tn N s1 bT)
≤
       𝒫(bT in T.T (c, s1). ?N s1 N1 bT)
      using ‹aeT c›[unfolded aeT-def, rule-format] AE-T-enabled AE-space
    proof (intro T.prob-dist, eventually-elim, intro impI)
      fix bT assume bT: enabled (c,s1) bT and ¬ ¬ discrCf (((c,s1) ## bT) !!
N1)
      with bT have qsend ((c,s1) ## bT) ≤ N1
        using less-qsend-iff-not-discrCf[of (c,s1) bT N1] by simp
      then show ?T s1 bT ⟷ ?Tn N s1 bT
        using bT
        by (cases qsend ((c, s1) ## bT))
           (auto intro!: enabled-qsend-indis del: iffI simp: N-def)
    qed measurable
    moreover
   have dist 𝒫(bT in T.T (c, s2). ?T s2 bT) 𝒫(bT in T.T (c, s2). ?Tn N s2 bT)
≤
       𝒫(bT in T.T (c, s2). ?N s2 N2 bT)
      using ‹aeT c›[unfolded aeT-def, rule-format] AE-T-enabled AE-space
    proof (intro T.prob-dist, eventually-elim, intro impI)
      fix bT assume bT: enabled (c,s2) bT ¬ ¬ discrCf (((c,s2) ## bT) !! N2)
      with bT have qsend ((c,s2) ## bT) ≤ N2
        using less-qsend-iff-not-discrCf[of (c,s2) bT N2] by simp
      then show ?T s2 bT ⟷ ?Tn N s2 bT
        using bT
        by (cases qsend ((c, s2) ## bT))
           (auto intro!: enabled-qsend-indis del: iffI simp: N-def)
    qed measurable
    ultimately have dist 𝒫(bT in T.T (c, s1). ?T s1 bT) 𝒫(bT in T.T (c, s1).
?Tn N s1 bT) +
     dist 𝒫(bT in T.T (c, s2). ?T s2 bT) 𝒫(bT in T.T (c, s1). ?Tn N s1 bT) ≤ e
      using ‹amSec c›[unfolded amSec-def, rule-format, OF ‹s1 ≈ s2›, of N t]
      using N1 N2 by simp
    from dist-triangle-le[OF this]
    show |?P s1 − ?P s2| ≤ e by (simp add: dist-real-def)
  qed
  then show ?P s1 = ?P s2
    by simp
qed
```

end

end

4 Compositionality of Resumption-Based Noninterference

theory *Compositionality*
imports *Resumption-Based*
begin

context *PL-Indis*
begin

4.1 Compatibility and discreetness of atoms, tests and choices

definition *compatAtm* **where**
compatAtm atm ≡
ALL s t. s ≈ *t* ⟶ *aval atm s* ≈ *aval atm t*

definition *presAtm* **where**
presAtm atm ≡
ALL s. s ≈ *aval atm s*

definition *compatTst* **where**
compatTst tst ≡
ALL s t. s ≈ *t* ⟶ *tval tst s* = *tval tst t*

lemma *discrAt-compatAt*[*simp*]:
assumes *presAtm atm*
shows *compatAtm atm*
using *assms* **unfolding** *compatAtm-def*
by (*metis presAtm-def indis-sym indis-trans*)

definition *compatCh* **where**
compatCh ch ≡ ∀ *s t. s* ≈ *t* ⟶ *cval ch s* = *cval ch t*

lemma *compatCh-cval*[*simp*]:
assumes *compatCh ch* **and** *s* ≈ *t*
shows *cval ch s* = *cval ch t*
using *assms* **unfolding** *compatCh-def* **by** *auto*

4.2 Compositionality of self-isomorphism

Self-Isomorphism versus language constructs:

lemma *siso-Done*[*simp*]:

siso Done
proof−
{fix *c* :: (*'test*, *'atom*, *'choice*) *cmd*
assume *c* = *Done* **hence** *siso c*
apply *induct* **by** *auto*
}
thus *?thesis* **by** *blast*
qed

lemma *siso-Atm*[*simp*]:
siso (*Atm atm*) = *compatAtm atm*
proof−
{fix *c* :: (*'test*, *'atom*, *'choice*) *cmd*
assume ∃ *atm*. *c* = *Atm atm* ∧ *compatAtm atm*
hence *siso c*
apply *induct*
apply (*metis compatAtm-def eff-Atm cont-Atm wt-Atm*)
by (*metis cont-Atm siso-Done*)
}
moreover have *siso* (*Atm atm*) ⟹ *compatAtm atm* **unfolding** *compatAtm-def*
by (*metis brn.simps eff-Atm less-Suc-eq mult-less-cancel1 nat-mult-1 siso-cont-indis*)
ultimately show *?thesis* **by** *blast*
qed

lemma *siso-Seq*[*simp*]:
assumes ∗: *siso c1* **and** ∗∗: *siso c2*
shows *siso* (*c1* ;; *c2*)
proof−
{fix *c* :: (*'test*, *'atom*, *'choice*) *cmd*
assume ∃ *c1 c2*. *c* = *c1* ;; *c2* ∧ *siso c1* ∧ *siso c2*
hence *siso c*
proof *induct*
case (*Obs c s t i*)
then obtain *c1 c2* **where** *i* < *brn c1*
and *c* = *c1* ;; *c2* **and** *siso c1* ∧ *siso c2*
and *s* ≈ *t* **by** *auto*
thus *?case* **by** (*cases finished* (*cont c1 s i*)) *auto*
next
case (*Cont c s i*)
then obtain *c1 c2* **where** *i* < *brn c1* **and**
c = *c1* ;; *c2* ∧ *siso c1* ∧ *siso c2* **by** *fastforce*
thus *?case* **by**(*cases finished* (*cont c1 s i*), *auto*)
qed
}
thus *?thesis* **using** *assms* **by** *blast*
qed

lemma *siso-While*[*simp*]:
assumes *compatTst tst* **and** *siso c*

shows *siso* (*While tst c*)
proof−
{fix *c* :: (*'test*, *'atom*, *'choice*) *cmd*
assume
(∃ *tst d*. *compatTst tst* ∧ *c* = *While tst d* ∧ *siso d*) ∨
(∃ *tst d1 d*. *compatTst tst* ∧ *c* = *d1* ;; (*While tst d*) ∧ *siso d1* ∧ *siso d*)
hence *siso c*
proof *induct*
case (*Obs c s t i*)
hence *i*: *i* < *brn c* **and** *st*: *s* ≈ *t* **by** *auto*
from *Obs* **show** *?case*
proof(*elim disjE exE conjE*)
fix *tst d*
assume *compatTst tst* **and** *c* = *While tst d* **and** *siso d*
thus *?thesis* **using** *i st* **unfolding** *compatTst-def*
by (*cases tval tst s*, *simp-all*)
next
fix *tst d1 d*
assume *compatTst tst* **and** *c* = *d1* ;; *While tst d*
and *siso d1* **and** *siso d*
thus *?thesis*
using *i st* **unfolding** *compatTst-def*
apply(*cases tval tst s*, *simp-all*)
by (*cases finished* (*cont d1 s i*), *simp-all*)+
qed
next
case (*Cont c s i*)
hence *i*: *i* < *brn c* **by** *simp*
from *Cont* **show** *?case*
proof(*elim disjE exE conjE*)
fix *tst d*
assume *compatTst tst* **and** *c* = *While tst d* **and** *siso d*
thus *?thesis* **by** (*cases tval tst s*, *simp-all*)
next
fix *tst d1 d*
assume *compatTst tst* **and** *c* = *d1* ;; *While tst d* **and** *siso d1* **and** *siso d*
thus *?thesis* **using** *i* **unfolding** *compatTst-def*
apply (*cases tval tst s*, *simp-all*)
by (*cases finished* (*cont d1 s i*), *simp-all*)+
qed
qed
}
thus *?thesis* **using** *assms* **by** *blast*
qed

lemma *siso-Ch*[*simp*]:
assumes *compatCh ch*
and ∗: *siso c1* **and** ∗∗: *siso c2*
shows *siso* (*Ch ch c1 c2*)

```
proof−
  {fix c :: ('test, 'atom, 'choice) cmd
   assume ∃ ch c1 c2. compatCh ch ∧ c = Ch ch c1 c2 ∧ siso c1 ∧ siso c2
   hence siso c
   proof induct
     case (Obs c s t i)
     then obtain ch c1 c2 where i < 2
     and compatCh ch and c = Ch ch c1 c2 and siso c1 ∧ siso c2
     and s ≈ t by fastforce
     thus ?case by (cases i) auto
   next
     case (Cont c s i)
     then obtain ch c1 c2 where i < 2 and
     compatCh ch ∧ c = Ch ch c1 c2 ∧ siso c1 ∧ siso c2 by fastforce
     thus ?case by (cases i) auto
   qed
  }
  thus ?thesis using assms by blast
qed

lemma siso-Par[simp]:
assumes properL cl and sisoL cl
shows siso (Par cl)
proof−
  {fix c :: ('test, 'atom, 'choice) cmd
   assume ∃ cl. c = Par cl ∧ properL cl ∧ sisoL cl
   hence siso c
   proof induct
     case (Obs c s t ii)
     then obtain cl where ii: ii < brnL cl (length cl)
     and cl: properL cl
     and c: c = Par cl and siso: sisoL cl
     and st: s ≈ t by auto
     let ?N = length cl
     from cl ii show ?case
     apply(cases rule: brnL-cases)
     using siso st cl unfolding c by fastforce
   next
     case (Cont c s ii)
     then obtain cl where ii: ii < brnL cl (length cl)
     and cl: properL cl
     and c: c = Par cl and sisoL: sisoL cl
     by auto
     from cl ii show ?case
     apply (cases rule: brnL-cases)
     using cl sisoL unfolding c by auto
   qed
  }
  thus ?thesis using assms by blast
```

```
qed

lemma siso-ParT[simp]:
assumes properL cl and sisoL cl
shows siso (ParT cl)
proof−
  {fix c :: ('test, 'atom, 'choice) cmd
   assume ∃ cl. c = ParT cl ∧ properL cl ∧ sisoL cl
   hence siso c
   proof induct
     case (Obs c s t ii)
     then obtain cl where ii: ii < brnL cl (length cl)
     and cl: properL cl
     and c: c = ParT cl and siso: sisoL cl
     and st: s ≈ t by auto
     let ?N = length cl
     from cl ii show ?case proof (cases rule: brnL-cases)
       case (Local n i)
       show ?thesis (is ?eff ∧ ?wt ∧ ?mv)
       proof−
         have eff-mv: ?eff ∧ ?mv using Local siso cl st unfolding c by force
         have wt: ?wt
         proof(cases WtFT cl = 1)
           case True
           thus ?thesis unfolding c using Local cl st siso True
           by (cases n = pickFT cl ∧ i = 0) auto
         next
           case False
           thus ?thesis unfolding c using Local cl st siso False
           by (cases finished (cl!n)) auto
         qed
         from eff-mv wt show ?thesis by simp
       qed
     qed
   next
     case (Cont c s ii)
     then obtain cl where ii: ii < brnL cl (length cl)
     and cl: properL cl
     and c: c = ParT cl and siso: sisoL cl
     by auto
     from cl ii show ?case apply (cases rule: brnL-cases)
     using siso cl unfolding c by force
   qed
  }
  thus ?thesis using assms by blast
qed
```

Self-isomorphism implies strong bisimilarity:

```
lemma bij-betw-emp[simp]:
```

bij-betw f {} {}
unfolding *bij-betw-def* **by** *auto*

lemma *part-full*[*simp*]:
part I {I}
unfolding *part-def* **by** *auto*

definition *singlPart* **where**
singlPart I ≡ {{i} | i . i ∈ I}

lemma *part-singlPart*[*simp*]:
part I (singlPart I)
unfolding *part-def singlPart-def* **by** *auto*

lemma *singlPart-inj-on*[*simp*]:
inj-on (image f) (singlPart I) = inj-on f I
unfolding *inj-on-def singlPart-def*
apply *auto*
by (*metis image-insert insertI1 insert-absorb insert-code singleton-inject*)

lemma *singlPart-surj*[*simp*]:
(image f) ‘ (singlPart I) = (singlPart J) ⟷ f ‘ I = J
unfolding *inj-on-def singlPart-def* **apply** *auto* **by** *blast*

lemma *singlPart-bij-betw*[*simp*]:
bij-betw (image f) (singlPart I) (singlPart J) = bij-betw f I J
unfolding *bij-betw-def* **by** *auto*

lemma *singlPart-finite1*:
assumes *finite (singlPart I)*
shows *finite (I::'a set)*
proof−
define *u* **where** *u i = {i}* **for** *i :: 'a*
have *u ‘ I ⊆ singlPart I* **unfolding** *u-def singlPart-def* **by** *auto*
moreover have *inj-on u I* **unfolding** *u-def inj-on-def* **by** *auto*
ultimately show *?thesis* **using** *assms*
by (*metis ‹inj-on u I› finite-imageD infinite-super*)
qed

lemma *singlPart-finite*[*simp*]:
finite (singlPart I) = finite I
using *singlPart-finite1*[*of I*] **unfolding** *singlPart-def* **by** *auto*

lemma *emp-notIn-singlPart*[*simp*]:
{} ∉ singlPart I
unfolding *singlPart-def* **by** *auto*

lemma *Sbis-coinduct*[*consumes 1*, *case-names step*, *coinduct set*]:
R c d ⟹

```
  (⋀c d s t. R c d ⟹ s ≈ t ⟹
    ∃P F. mC-C-part c d P F ∧ inj-on F P ∧ mC-C-wt c d s t P F ∧
      (∀I∈P. ∀i∈I. ∀j∈F I.
            eff c s i ≈ eff d t j ∧ (R (cont c s i) (cont d t j) ∨ (cont c s i, cont d
t j) ∈ Sbis)))
  ⟹ (c, d) ∈ Sbis
  using Sbis-coind[of {(x, y). R x y}]
  unfolding Sretr-def matchC-C-def mC-C-def mC-C-eff-cont-def
  apply (simp add: subset-eq Ball-def )
  apply metis
  done

lemma siso-Sbis[simp]: siso c ⟹ c ≈s c
proof (coinduction arbitrary: c)
  case (step s t c) with siso-cont-indis[of c s t] part-singlPart[of {..<brn c}] show
?case
    by (intro exI[of - singlPart {..< brn c}] exI[of - id])
      (auto simp add: mC-C-part-def mC-C-wt-def singlPart-def)
qed
```

4.3 Discreetness versus language constructs:

```
lemma discr-Done[simp]: discr Done
  by coinduction auto

lemma discr-Atm-presAtm[simp]: discr (Atm atm) = presAtm atm
proof−
  have presAtm atm ⟹ discr (Atm atm)
    by (coinduction arbitrary: atm) (auto simp: presAtm-def)
  moreover have discr (Atm atm) ⟹ presAtm atm
    unfolding presAtm-def
    by (metis One-nat-def brn.simps(2) discr.simps eff-Atm lessI)
  ultimately show ?thesis by blast
qed

lemma discr-Seq[simp]:
  discr c1 ⟹ discr c2 ⟹ discr (c1 ;; c2)
  by (coinduction arbitrary: c1 c2)
    (simp, metis cont-Seq-finished discr-cont cont-Seq-notFinished)

lemma discr-While[simp]: assumes discr c shows discr (While tst c)
proof−
  {fix c :: ('test, 'atom, 'choice) cmd
   assume
   (∃ tst d. c = While tst d ∧ discr d) ∨
    (∃ tst d1 d. c = d1 ;; (While tst d) ∧ discr d1 ∧ discr d)
   hence discr c
   apply induct apply safe
   apply (metis eff-While indis-refl)
```

```
   apply (metis cont-While-False discr-Done cont-While-True)
   apply (metis eff-Seq discr.simps brn.simps)
   by (metis cont-Seq-finished discr.simps cont-Seq-notFinished brn.simps)
  }
  thus ?thesis using assms by blast
qed

lemma discr-Ch[simp]: discr c1 ⟹ discr c2 ⟹ discr (Ch ch c1 c2)
  by coinduction (simp, metis indis-refl less-2-cases cont-Ch-L cont-Ch-R)

lemma discr-Par[simp]: properL cl ⟹ discrL cl ⟹ discr (Par cl)
proof (coinduction arbitrary: cl, clarsimp)
  fix cl ii s
  assume ∗: properL cl ii < brnL cl (length cl) and discrL cl
  from ∗ show s ≈ eff (Par cl) s ii ∧
    ((∃ cl'. cont (Par cl) s ii = Par cl' ∧ properL cl' ∧ discrL cl') ∨ discr (cont
(Par cl) s ii))
  proof (cases rule: brnL-cases)
    case (Local n i)
    with ‹discrL cl› have s ≈ eff (cl ! n) s i by simp
    thus ?thesis
      using Local ‹discrL cl› ‹properL cl› by auto
  qed
qed

lemma discr-ParT[simp]: properL cl ⟹ discrL cl ⟹ discr (ParT cl)
proof (coinduction arbitrary: cl, clarsimp)
  fix p cl s ii assume properL cl ii < brnL cl (length cl) discrL cl
  then show s ≈ eff (ParT cl) s ii ∧
    ((∃ cl'. cont (ParT cl) s ii = ParT cl' ∧ properL cl' ∧ discrL cl') ∨ discr (cont
(ParT cl) s ii))
  proof (cases rule: brnL-cases)
    case (Local n i)
    have s ≈ eff (cl ! n) s i using Local ‹discrL cl› by simp
    thus ?thesis using Local ‹discrL cl› ‹properL cl› by simp
  qed
qed

lemma discr-finished[simp]: proper c ⟹ finished c ⟹ discr c
  by (induct c rule: proper-induct) (auto simp: discrL-intro)
```

4.4 Strong bisimilarity versus language constructs

```
lemma Sbis-pres-discr-L:
  c ≈s d ⟹ discr d ⟹ discr c
proof (coinduction arbitrary: d c, clarsimp)
  fix c d s i
  assume d: c ≈s d discr d and i: i < brn c
  then obtain P F where
```

match: *mC-C Sbis c d s s P F*
using *Sbis-mC-C*[*of c d s s*] **by** *blast*
hence $\bigcup P = \{..<brn\ c\}$
using *i* **unfolding** *mC-C-def mC-C-part-def part-def* **by** *simp*
then obtain *I* **where** *I*: $I \in P$ **and** *i*: $i \in I$ **using** *i* **by** *auto*
obtain *j* **where** *j*: $j \in F\ I$
using *match I* **unfolding** *mC-C-def mC-C-part-def* **by** *blast*
hence $j < brn\ d$ **using** *I match*
unfolding *mC-C-def mC-C-part-def part-def* **apply** *simp* **by** *blast*
hence *md*: *discr* (*cont d s j*) **and** *s*: $s \approx$ *eff d s j*
using *d discr-cont*[*of d j s*] *discr-eff-indis*[*of d j s*] **by** *auto*
have *eff c s i* $\approx$ *eff d s j* **and** *md2*: *cont c s i* $\approx s$ *cont d s j*
using *I i j match* **unfolding** *mC-C-def mC-C-eff-cont-def* **by** *auto*
hence $s \approx$ *eff c s i* **using** *s indis-sym indis-trans* **by** *blast*
thus $s \approx$ *eff c s i* $\wedge$ $((\exists\, d.$ *cont c s i* $\approx s$ *d* $\wedge$ *discr d*$) \vee$ *discr* (*cont c s i*)$)$
using *md md2* **by** *blast*
qed

lemma *Sbis-pres-discr-R*:
assumes *discr c* **and** $c \approx s$ *d*
shows *discr d*
using *assms Sbis-pres-discr-L Sbis-sym* **by** *blast*

lemma *Sbis-finished-discr-L*:
assumes $c \approx s$ *d* **and** *proper d* **and** *finished d*
shows *discr c*
using *assms Sbis-pres-discr-L* **by** *auto*

lemma *Sbis-finished-discr-R*:
assumes *proper c* **and** *finished c* **and** $c \approx s$ *d*
shows *discr d*
using *assms Sbis-pres-discr-R*[*of c d*] **by** *auto*

definition *thetaSD* **where**
thetaSD $\equiv \{(c,\ d) \mid c\ d\ .$ *proper c* $\wedge$ *proper d* $\wedge$ *discr c* $\wedge$ *discr d*$\}$

lemma *thetaSD-Sretr*:
thetaSD $\subseteq$ *Sretr thetaSD*
unfolding *Sretr-def matchC-C-def* **proof** *safe*
fix *c d s t*
assume *c-d*: $(c,\ d) \in$ *thetaSD* **and** *st*: $s \approx t$
hence *p*: *proper c* $\wedge$ *proper d* **unfolding** *thetaSD-def* **by** *auto*
let *?P* = $\{\{..<brn\ c\}\}$
let *?F* = $\%\ I.\ \{..<\ brn\ d\}$
have *0*: $\{..<brn\ c\} \neq \{\}$ $\{..<brn\ d\} \neq \{\}$
using *p int-emp brn-gt-0* **by** *blast+*
show $\exists\, P\ F.$ *mC-C thetaSD c d s t P F*

apply −
apply(*rule exI*[*of* - *?P*]) **apply**(*rule exI*[*of* - *?F*])
unfolding *mC-C-def* **proof**(*intro conjI*)
show *mC-C-part c d ?P ?F*
unfolding *mC-C-part-def* **using** *0* **unfolding** *part-def* **by** *auto*
next
show *inj-on ?F ?P* **unfolding** *inj-on-def* **by** *simp*
next
show *mC-C-wt c d s t ?P ?F* **unfolding** *mC-C-wt-def* **using** *p* **by** *auto*
next
show *mC-C-eff-cont thetaSD c d s t ?P ?F*
unfolding *mC-C-eff-cont-def* **proof** *clarify*
fix *I i j*
assume *i*: $i < brn\ c$ **and** *j*: $j < brn\ d$
hence $s \approx eff\ c\ s\ i$ **using** *c-d* **unfolding** *thetaSD-def* **by** *simp*
moreover have $t \approx eff\ d\ t\ j$ **using** *i j c-d* **unfolding** *thetaSD-def* **by** *simp*
ultimately have $eff\ c\ s\ i \approx eff\ d\ t\ j$ **using** *st indis-sym indis-trans* **by** *blast*
thus $eff\ c\ s\ i \approx eff\ d\ t\ j \wedge (cont\ c\ s\ i,\ cont\ d\ t\ j) \in thetaSD$
using *c-d i j* **unfolding** *thetaSD-def* **by** *auto*
qed
qed
qed

lemma *thetaSD-Sbis*:
$thetaSD \subseteq Sbis$
using *Sbis-raw-coind thetaSD-Sretr* **by** *blast*

theorem *discr-Sbis*[*simp*]:
assumes *proper c* **and** *proper d* **and** *discr c* **and** *discr d*
shows $c \approx s\ d$
using *assms thetaSD-Sbis* **unfolding** *thetaSD-def* **by** *auto*

definition *thetaSDone* **where**
$thetaSDone \equiv \{(Done,\ Done)\}$

lemma *thetaSDone-Sretr*:
$thetaSDone \subseteq Sretr\ thetaSDone$
unfolding *Sretr-def matchC-C-def thetaSDone-def* **proof** *safe*
fix *s t* **assume** *st*: $s \approx t$
let $?P = \{\{0\}\}$ **let** $?F = id$
show $\exists P\ F.\ mC\text{-}C\ \{(Done,\ Done)\}\ Done\ Done\ s\ t\ P\ F$
apply(*intro exI*[*of* - *?P*]) **apply**(*intro exI*[*of* - *?F*])
unfolding *m-defsAll part-def* **using** *st* **by** *auto*
qed

lemma *thetaSDone-Sbis*:
$thetaSDone \subseteq Sbis$

using *Sbis-raw-coind thetaSDone-Sretr* **by** *blast*

theorem *Done-Sbis*[*simp*]:
Done ≈*s Done*
using *thetaSDone-Sbis* **unfolding** *thetaSDone-def* **by** *auto*

definition *thetaSAtm* **where**
thetaSAtm atm ≡
{(*Atm atm*, *Atm atm*), (*Done*, *Done*)}

lemma *thetaSAtm-Sretr*:
assumes *compatAtm atm*
shows *thetaSAtm atm* ⊆ *Sretr* (*thetaSAtm atm*)
unfolding *Sretr-def matchC-C-def thetaSAtm-def* **proof** *safe*
fix *s t* **assume** *st*: *s* ≈ *t*
let *?P* = {{*0*}} **let** *?F* = *id*
show ∃ *P F*. *mC-C* {(*Atm atm*, *Atm atm*), (*Done*, *Done*)} *Done Done s t P F*
apply(*intro exI*[*of - ?P*]) **apply**(*intro exI*[*of - ?F*])
unfolding *m-defsAll part-def* **using** *st* **by** *auto*
next
fix *s t* **assume** *st*: *s* ≈ *t*
let *?P* = {{*0*}} **let** *?F* = *id*
show ∃ *P F*. *mC-C* {(*Atm atm*, *Atm atm*), (*Done*, *Done*)} (*Atm atm*) (*Atm atm*) *s t P F*
apply(*intro exI*[*of - ?P*]) **apply**(*intro exI*[*of - ?F*])
unfolding *m-defsAll part-def* **using** *st assms* **unfolding** *compatAtm-def* **by** *auto*
qed

lemma *thetaSAtm-Sbis*:
assumes *compatAtm atm*
shows *thetaSAtm atm* ⊆ *Sbis*
using *assms Sbis-raw-coind thetaSAtm-Sretr* **by** *blast*

theorem *Atm-Sbis*[*simp*]:
assumes *compatAtm atm*
shows *Atm atm* ≈*s Atm atm*
using *assms thetaSAtm-Sbis* **unfolding** *thetaSAtm-def* **by** *auto*

definition *thetaSSeqI* **where**
thetaSSeqI ≡
{(*e* ;; *c*, *e* ;; *d*) | *e c d* . *siso e* ∧ *c* ≈*s d*}

lemma *thetaSSeqI-Sretr*:
thetaSSeqI ⊆ *Sretr* (*thetaSSeqI Un Sbis*)
unfolding *Sretr-def matchC-C-def* **proof** *safe*
fix *c d s t*
assume *c-d*: (*c*, *d*) ∈ *thetaSSeqI* **and** *st*: *s* ≈ *t*

```
  then obtain e c1 d1 where e: siso e and c1d1: c1 ≈s d1
  and c: c = e ;; c1 and d: d = e ;; d1
  unfolding thetaSSeqI-def by auto
  let ?P = {{i} | i . i < brn e}
  let ?F = %I. I
  show ∃ P F. mC-C (thetaSSeqI Un Sbis) c d s t P F
  apply(rule exI[of - ?P]) apply(rule exI[of - ?F])
  unfolding mC-C-def proof (intro conjI)
    show mC-C-part c d ?P ?F
    unfolding mC-C-part-def proof (intro conjI)
      show part {..<brn c} ?P
      unfolding part-def proof safe
        fix i assume i < brn c
        thus i ∈ ⋃ ?P using c e st siso-cont-indis[of e s t] by auto
      qed (unfold c, simp)

      thus part {..<brn d} (?F ‘ ?P) unfolding c d by auto
    qed auto
  next
    show mC-C-eff-cont (thetaSSeqI Un Sbis) c d s t ?P ?F
    unfolding mC-C-eff-cont-def proof (intro impI allI, elim conjE)
      fix I i j
      assume I: I ∈ ?P and i: i ∈ I and j: j ∈ I
      show eff c s i ≈ eff d t j ∧ (cont c s i, cont d t j) ∈ thetaSSeqI ∪ Sbis
      proof(cases I = {})
        case True thus ?thesis using i by simp
      next
        case False
        then obtain i' where j ∈ ?F {i'} and i' < brn e
        using I j by auto
        thus ?thesis
        using st c-d e i j I unfolding c d thetaSSeqI-def
        by (cases finished (cont e s i')) auto
      qed
    qed
  qed (insert st c-d c, unfold m-defsAll thetaSSeqI-def part-def, auto)
qed

lemma thetaSSeqI-Sbis:
thetaSSeqI ⊆ Sbis
using Sbis-coind thetaSSeqI-Sretr by blast

theorem Seq-siso-Sbis[simp]:
assumes siso e and c2 ≈s d2
shows e ;; c2 ≈s e ;; d2
using assms thetaSSeqI-Sbis unfolding thetaSSeqI-def by auto


definition thetaSSeqD where
```

```
thetaSSeqD ≡
 {(c1 ;; c2, d1 ;; d2) |
    c1 c2 d1 d2.
      proper c1 ∧ proper d1 ∧ proper c2 ∧ proper d2 ∧
      discr c2 ∧ discr d2 ∧
      c1 ≈s d1}

lemma thetaSSeqD-Sretr:
thetaSSeqD ⊆ Sretr (thetaSSeqD Un Sbis)
unfolding Sretr-def matchC-C-def proof safe
  fix c d s t
  assume c-d: (c, d) ∈ thetaSSeqD and st: s ≈ t
  then obtain c1 c2 d1 d2 where
  c1d1: proper c1 proper d1 c1 ≈s d1 and
  c2d2: proper c2 proper d2 discr c2 discr d2
  and c: c = c1 ;; c2 and d: d = d1 ;; d2
  unfolding thetaSSeqD-def by auto
  from c1d1 st obtain P F
  where match: mC-C Sbis c1 d1 s t P F
  using Sbis-mC-C by blast
  have P: ⋃ P = {..<brn c1} and FP: ⋃ (F ‘ P) = {..<brn d1}
  using match unfolding mC-C-def mC-C-part-def part-def by metis+
  show ∃ P F. mC-C (thetaSSeqD Un Sbis) c d s t P F
  apply(rule exI[of - P]) apply(rule exI[of - F])
  unfolding mC-C-def proof (intro conjI)
    show mC-C-eff-cont (thetaSSeqD ∪ Sbis) c d s t P F
    unfolding mC-C-eff-cont-def proof(intro allI impI, elim conjE)
      fix i j I assume I : I ∈ P and i: i ∈ I and j: j ∈ F I
      let ?c1' = cont c1 s i  let ?d1' = cont d1 t j
      let ?s' = eff c1 s i  let ?t' = eff d1 t j
      have i < brn c1 using i I P by blast note i = this i
      have j < brn d1 using j I FP by blast note j = this j
      have c1'd1': ?c1' ≈s ?d1'
      proper ?c1' proper ?d1'
      using c1d1 i j I match unfolding c mC-C-def mC-C-eff-cont-def by auto
      show eff c s i ≈ eff d t j ∧ (cont c s i, cont d t j) ∈ thetaSSeqD ∪ Sbis
      (is ?eff ∧ ?cont) proof
        show ?eff using match I i j unfolding c d m-defsAll by simp
      next
        show ?cont
        proof(cases finished ?c1')
          case True note c1' = True
         hence csi: cont c s i = c2 using i match unfolding c m-defsAll by simp
          show ?thesis
          proof(cases finished ?d1')
            case True
            hence cont d t j = d2 using j match unfolding d m-defsAll by simp
            thus ?thesis using csi c2d2 by simp
          next
```

```
          case False
          hence dtj: cont d t j = ?d1' ;; d2
          using j match unfolding d m-defsAll by simp
          have discr ?d1' using c1'd1' c1' Sbis-finished-discr-R by blast
          thus ?thesis using c1'd1' c2d2 unfolding csi dtj by simp
        qed
      next
        case False note Done-c = False
        hence csi: cont c s i = ?c1' ;; c2
        using i match unfolding c m-defsAll by simp
        show ?thesis
        proof(cases finished (cont d1 t j))
          case True note d1' = True
           hence dtj: cont d t j = d2 using j match unfolding d m-defsAll by
simp
          have discr ?c1' using c1'd1' d1' Sbis-finished-discr-L by blast
          thus ?thesis using c1'd1' c2d2 unfolding csi dtj by simp
        next
          case False
         hence dtj: cont d t j = ?d1' ;; d2 using j match unfolding d m-defsAll
by simp
          thus ?thesis unfolding csi dtj thetaSSeqD-def
          using c1'd1' c2d2 by blast
        qed
      qed
    qed
  qed
qed

lemma thetaSSeqD-Sbis:
thetaSSeqD ⊆ Sbis
using Sbis-coind thetaSSeqD-Sretr by blast

theorem Seq-Sbis[simp]:
assumes proper c1 and proper d1 and proper c2 and proper d2
and c1 ≈s d1 and discr c2 and discr d2
shows c1 ;; c2 ≈s d1 ;; d2
using assms thetaSSeqD-Sbis unfolding thetaSSeqD-def by auto




definition thetaSCh where
thetaSCh ch c1 c2 d1 d2 ≡ {(Ch ch c1 c2, Ch ch d1 d2)}

lemma thetaSCh-Sretr:
assumes compatCh ch and c1 ≈s d1 and c2 ≈s d2
shows thetaSCh ch c1 c2 d1 d2 ⊆
```

Sretr (*thetaSCh ch c1 c2 d1 d2* ∪ *Sbis*)
(**is** *?th* ⊆ *Sretr* (*?th* ∪ *Sbis*))
unfolding *Sretr-def matchC-C-def* **proof** *safe*
fix *c d s t*
assume *c-d*: (*c*, *d*) ∈ *?th* **and** *st*: *s* ≈ *t*
hence *c*: *c* = *Ch ch c1 c2 brn c* = *2*
and *d*: *d* = *Ch ch d1 d2 brn d* = *2*
unfolding *thetaSCh-def* **by** *auto*
let *?P* = {{*0*}, {*1*}}
let *?F* = %*I*. *I*
show ∃ *P F*. *mC-C* (*?th Un Sbis*) *c d s t P F*
apply(*rule exI*[*of* - *?P*]) **apply**(*rule exI*[*of* - *?F*])
using *assms st c-d c* **unfolding** *m-defsAll thetaSCh-def part-def* **by** *auto*
qed

lemma *thetaSCh-Sbis*:
assumes *compatCh ch* **and** *c1* ≈*s d1* **and** *c2* ≈*s d2*
shows *thetaSCh ch c1 c2 d1 d2* ⊆ *Sbis*
using *Sbis-coind thetaSCh-Sretr*[*OF assms*] **by** *blast*

theorem *Ch-siso-Sbis*[*simp*]:
assumes *compatCh ch* **and** *c1* ≈*s d1* **and** *c2* ≈*s d2*
shows *Ch ch c1 c2* ≈*s Ch ch d1 d2*
using *thetaSCh-Sbis*[*OF assms*] **unfolding** *thetaSCh-def* **by** *auto*

definition *shift* **where**
shift cl n ≡ *image* (%*i*. *brnL cl n* + *i*)

definition *back* **where**
back cl n ≡ *image* (% *ii*. *ii* − *brnL cl n*)

lemma *emp-shift*[*simp*]:
shift cl n I = {} ⟷ *I* = {}
unfolding *shift-def* **by** *auto*

lemma *emp-shift-rev*[*simp*]:
{} = *shift cl n I* ⟷ *I* = {}
unfolding *shift-def* **by** *auto*

lemma *emp-back*[*simp*]:
back cl n II = {} ⟷ *II* = {}
unfolding *back-def* **by** *force*

lemma *emp-back-rev*[*simp*]:
{} = *back cl n II* ⟷ *II* = {}
unfolding *back-def* **by** *force*

lemma *in-shift*[*simp*]:
brnL cl n + *i* ∈ *shift cl n I* ⟷ *i* ∈ *I*
unfolding *shift-def* **by** *auto*

lemma *in-back*[*simp*]:
ii ∈ *II* ⟹ *ii* − *brnL cl n* ∈ *back cl n II*
unfolding *back-def* **by** *auto*

lemma *in-back2*[*simp*]:
assumes *ii* > *brnL cl n* **and** *II* ⊆ {*brnL cl n* ..<+ *brn* (*cl*!*n*)}
shows *ii* − *brnL cl n* ∈ *back cl n II* ⟷ *ii* ∈ *II* (**is** *?L* ⟷ *?R*)
using *assms* **unfolding** *back-def* **by** *force*

lemma *shift*[*simp*]:
assumes *I* ⊆ {..< *brn* (*cl*!*n*)}
shows *shift cl n I* ⊆ {*brnL cl n* ..<+ *brn* (*cl*!*n*)}
using *assms* **unfolding** *shift-def* **by** *auto*

lemma *shift2*[*simp*]:
assumes *I* ⊆ {..< *brn* (*cl*!*n*)}
and *ii* ∈ *shift cl n I*
shows *brnL cl n* ≤ *ii* ∧ *ii* < *brnL cl n* + *brn* (*cl*!*n*)
using *assms* **unfolding** *shift-def* **by** *auto*

lemma *shift3*[*simp*]:
assumes *n*: *n* < *length cl* **and** *I*: *I* ⊆ {..< *brn* (*cl*!*n*)}
and *ii*: *ii* ∈ *shift cl n I*
shows *ii* < *brnL cl* (*length cl*)
proof−
  have *ii* < *brnL cl n* + *brn* (*cl*!*n*) **using** *I ii* **by** *simp*
  also have ... ≤ *brnL cl* (*length cl*) **using** *n*
  by (*metis brnL-Suc brnL-mono Suc-leI*)
  finally show *?thesis* **.**
qed

lemma *back*[*simp*]:
assumes *II* ⊆ {*brnL cl n* ..<+ *brn* (*cl*!*n*)}
shows *back cl n II* ⊆ {..< *brn* (*cl*!*n*)}
using *assms* **unfolding** *back-def* **by** *force*

lemma *back2*[*simp*]:
assumes *II* ⊆ {*brnL cl n* ..<+ *brn* (*cl*!*n*)}
and *i* ∈ *back cl n II*
shows *i* < *brn* (*cl*!*n*)
using *assms* **unfolding** *back-def* **by** *force*

lemma *shift-inj*[*simp*]:
shift cl n I1 = *shift cl n I2* ⟷ *I1* = *I2*
unfolding *shift-def* **by** *force*

lemma *shift-mono*[*simp*]:
shift cl n I1 $\subseteq$ *shift cl n I2* $\longleftrightarrow$ *I1* $\subseteq$ *I2*
unfolding *shift-def* **by** *auto*

lemma *shift-Int*[*simp*]:
shift cl n I1 $\cap$ *shift cl n I2* $= \{\} \longleftrightarrow$ *I1* $\cap$ *I2* $= \{\}$
unfolding *shift-def* **by** *force*

lemma *inj-shift*: *inj* (*shift cl n*)
unfolding *inj-on-def* **by** *simp*

lemma *inj-on-shift*: *inj-on* (*shift cl n*) *K*
using *inj-shift* **unfolding** *inj-on-def* **by** *simp*

lemma *back-shift*[*simp*]:
back cl n (*shift cl n I*) $=$ *I*
unfolding *back-def shift-def* **by** *force*

lemma *shift-back*[*simp*]:
assumes *II* $\subseteq \{$*brnL cl n* ..<+ *brn* (*cl*!*n*)$\}$
shows *shift cl n* (*back cl n II*) $=$ *II*
using *assms* **unfolding** *shift-def back-def atLeastLessThan-def* **by** *force*

lemma *back-inj*[*simp*]:
assumes *II1*: *II1* $\subseteq \{$*brnL cl n* ..<+ *brn* (*cl*!*n*)$\}$
and *II2*: *II2* $\subseteq \{$*brnL cl n* ..<+ *brn* (*cl*!*n*)$\}$
shows *back cl n II1* $=$ *back cl n II2* $\longleftrightarrow$ *II1* $=$ *II2* (**is** *?L* $=$ *?R* $\longleftrightarrow$ *II1* $=$ *II2*)
proof
  have *II1* $=$ *shift cl n ?L* **using** *II1* **by** *simp*
  also assume *?L* $=$ *?R*
  also have *shift cl n ?R* $=$ *II2* **using** *II2* **by** *simp*
  finally show *II1* $=$ *II2* **.**
qed *auto*

lemma *back-mono*[*simp*]:
assumes *II1* $\subseteq \{$*brnL cl n* ..<+ *brn* (*cl*!*n*)$\}$
and *II2* $\subseteq \{$*brnL cl n* ..< *brnL cl n* + *brn* (*cl*!*n*)$\}$
shows *back cl n II1* $\subseteq$ *back cl n II2* $\longleftrightarrow$ *II1* $\subseteq$ *II2*
(**is** *?L* $\subseteq$ *?R* $\longleftrightarrow$ *II1* $\subseteq$ *II2*)
proof−
  have *?L* $\subseteq$ *?R* $\longleftrightarrow$ *shift cl n ?L* $\subseteq$ *shift cl n ?R* **by** *simp*
  also have ... $\longleftrightarrow$ *II1* $\subseteq$ *II2* **using** *assms* **by** *simp*
  finally show *?thesis* **.**
qed

lemma *back-Int*[*simp*]:
assumes *II1* $\subseteq \{$*brnL cl n* ..<+ *brn* (*cl*!*n*)$\}$
and *II2* $\subseteq \{$*brnL cl n* ..< *brnL cl n* + *brn* (*cl*!*n*)$\}$

shows *back cl n II1* ∩ *back cl n II2* = {} ⟷ *II1* ∩ *II2* = {}
(**is** *?L* ∩ *?R* = {} ⟷ *II1* ∩ *II2* = {})
proof−
  have *?L* ∩ *?R* = {} ⟷ *shift cl n ?L* ∩ *shift cl n ?R* = {} **by** *simp*
  also have ... ⟷ *II1* ∩ *II2* = {} **using** *assms* **by** *simp*
  finally show *?thesis* **.**
qed

lemma *inj-on-back*:
inj-on (*back cl n*) (*Pow* {*brnL cl n* ..<+ *brn* (*cl*!*n*)})
unfolding *inj-on-def* **by** *simp*

lemma *shift-surj*:
assumes *II* ⊆ {*brnL cl n* ..<+ *brn* (*cl*!*n*)}
shows ∃ *I*. *I* ⊆ {..< *brn* (*cl*!*n*)} ∧ *shift cl n I* = *II*
apply(*intro exI*[*of* - *back cl n II*]) **using** *assms* **by** *simp*

lemma *back-surj*:
assumes *I* ⊆ {..< *brn* (*cl*!*n*)}
shows ∃ *II*. *II* ⊆ {*brnL cl n* ..<+ *brn* (*cl*!*n*)} ∧ *back cl n II* = *I*
apply(*intro exI*[*of* - *shift cl n I*]) **using** *assms* **by** *simp*

lemma *shift-part*[*simp*]:
assumes *part* {..< *brn* (*cl*!*n*)} *P*
shows *part* {*brnL cl n* ..<+ *brn* (*cl*!*n*)} (*shift cl n* ‘ *P*)
unfolding *part-def* **proof**(*intro conjI allI impI*)
  show ⋃ (*shift cl n* ‘ *P*) = {*brnL cl n* ..<+ *brn* (*cl*!*n*)}
  proof *safe*
    fix *ii I* **assume** *ii*: *ii* ∈ *shift cl n I* **and** *I* ∈ *P*
    hence *I* ⊆ {..< *brn* (*cl*!*n*)} **using** *assms* **unfolding** *part-def* **by** *blast*
    thus *ii* ∈ {*brnL cl n* ..<+ *brn* (*cl*!*n*)} **using** *ii* **by** *simp*
  next
    fix *ii* **assume** *ii-in*: *ii* ∈ {*brnL cl n*..<+*brn* (*cl* ! *n*)}
    define *i* **where** *i* = *ii* − *brnL cl n*
    have *ii*: *ii* = *brnL cl n* + *i* **unfolding** *i-def* **using** *ii-in* **by** *force*
    have *i* ∈ {..< *brn* (*cl*!*n*)} **unfolding** *i-def* **using** *ii-in* **by** *auto*
    then obtain *I* **where** *i*: *i* ∈ *I* **and** *I*: *I* ∈ *P*
    using *assms* **unfolding** *part-def* **by** *blast*
    thus *ii* ∈ ⋃ (*shift cl n* ‘ *P*) **unfolding** *ii* **by** *force*
  qed
qed(*insert assms*, *unfold part-def*, *force*)

lemma *part-brn-disj1*:
assumes *P*: ⋀ *n*. *n* < *length cl* ⟹ *part* {..< *brn* (*cl*!*n*)} (*P n*)
and *n1*: *n1* < *length cl* **and** *n2*: *n2* < *length cl*
and *II1*: *II1* ∈ *shift cl n1* ‘ (*P n1*) **and** *II2*: *II2* ∈ *shift cl n2* ‘ (*P n2*) **and** *d*: *n1* ≠ *n2*
shows *II1* ∩ *II2* = {}
proof−

let *?N = length cl*
obtain *I1 I2* **where** *I1*: *I1 ∈ P n1* **and** *I2*: *I2 ∈ P n2*
and *II1*: *II1 = shift cl n1 I1* **and** *II2*: *II2 = shift cl n2 I2*
using *II1 II2* **by** *auto*
have *I1 ⊆ {..< brn (cl!n1)}* **and** *I2 ⊆ {..< brn (cl!n2)}*
using *n1 I1 n2 I2 P* **unfolding** *part-def* **by** *blast+*
hence *II1 ⊆ {brnL cl n1 ..<+ brn (cl!n1)}* **and** *II2 ⊆ {brnL cl n2 ..<+ brn (cl!n2)}*
unfolding *II1 II2* **by** *auto*
thus *?thesis* **using** *n1 n2 d brnL-Int* **by** *blast*
qed

lemma *part-brn-disj2*:
assumes *P*: *⋀ n. n < length cl ⟹ part {..< brn (cl!n)} (P n) ∧ {} ∉ P n*
and *n1*: *n1 < length cl* **and** *n2*: *n2 < length cl* **and** *d*: *n1 ≠ n2*
shows *shift cl n1 ‘ (P n1) ∩ shift cl n2 ‘ (P n2) = {}* (**is** *?L ∩ ?R = {}*)
proof−
{fix *II* **assume** *II*: *II ∈ ?L ∩ ?R*
hence *II = {}* **using** *part-brn-disj1*[*of cl P n1 n2 II II*] *assms* **by** *blast*
hence *{} ∈ ?L* **using** *II* **by** *blast*
then obtain *I* **where** *I*: *I ∈ P n1* **and** *II*: *{} = shift cl n1 I* **by** *blast*
hence *I = {}* **by** *simp*
hence *False* **using** *n1 P I* **by** *blast*
}
thus *?thesis* **by** *blast*
qed

lemma *part-brn-disj3*:
assumes *P*: *⋀ n. n < length cl ⟹ part {..< brn (cl!n)} (P n)*
and *n1*: *n1 < length cl* **and** *n2*: *n2 < length cl*
and *I1*: *I1 ∈ P n1* **and** *I2*: *I2 ∈ P n2* **and** *d*: *n1 ≠ n2*
shows *shift cl n1 I1 ∩ shift cl n2 I2 = {}*
apply(*rule part-brn-disj1*)
using *assms* **by** *auto*

lemma *sum-wt-Par-sub-shift*[*simp*]:
assumes *cl*: *properL cl* **and** *n*: *n < length cl* **and**
I: *I ⊆ {..< brn (cl ! n)}*
shows
sum (wt (Par cl) s) (shift cl n I) =
1 / (length cl) ∗ sum (wt (cl ! n) s) I
using *assms sum-wt-Par-sub* **unfolding** *shift-def* **by** *simp*

lemma *sum-wt-ParT-sub-WtFT-pickFT-0-shift*[*simp*]:
assumes *cl*: *properL cl* **and** *nf*: *WtFT cl = 1*
and *I*: *I ⊆ {..< brn (cl ! (pickFT cl))} 0 ∈ I*
shows
sum (wt (ParT cl) s) (shift cl (pickFT cl) I) = 1
using *assms sum-wt-ParT-sub-WtFT-pickFT-0*

unfolding *shift-def* **by** *simp*

lemma *sum-wt-ParT-sub-WtFT-notPickFT-0-shift*[*simp*]:
assumes *cl*: *properL cl* **and** *nf*: *WtFT cl* = *1* **and** *n*: *n* < *length cl*
and *I*: *I* ⊆ {..< *brn* (*cl* ! *n*)} **and** *nI*: *n* = *pickFT cl* ⟶ *0* ∉ *I*
shows *sum* (*wt* (*ParT cl*) *s*) (*shift cl n I*) = *0*
using *assms sum-wt-ParT-sub-WtFT-notPickFT-0* **unfolding** *shift-def* **by** *simp*

lemma *sum-wt-ParT-sub-notWtFT-finished-shift*[*simp*]:
assumes *cl*: *properL cl* **and** *nf*: *WtFT cl* ≠ *1* **and** *n*: *n* < *length cl* **and** *cln*: *finished* (*cl*!*n*)
and *I*: *I* ⊆ {..< *brn* (*cl* ! *n*)}
shows *sum* (*wt* (*ParT cl*) *s*) (*shift cl n I*) = *0*
using *assms sum-wt-ParT-sub-notWtFT-finished*
unfolding *shift-def* **by** *simp*

lemma *sum-wt-ParT-sub-notWtFT-notFinished-shift*[*simp*]:
assumes *cl*: *properL cl* **and** *nf*: *WtFT cl* ≠ *1*
and *n*: *n* < *length cl* **and** *cln*: ¬ *finished* (*cl*!*n*)
and *I*: *I* ⊆ {..< *brn* (*cl* ! *n*)}
shows
sum (*wt* (*ParT cl*) *s*) (*shift cl n I*) =
(*1* / (*length cl*)) / (*1* − *WtFT cl*) ∗ *sum* (*wt* (*cl* ! *n*) *s*) *I*
using *assms sum-wt-ParT-sub-notWtFT-notFinished*
unfolding *shift-def* **by** *simp*

definition *UNpart* **where**
UNpart cl P ≡ ⋃ *n* < *length cl*. *shift cl n* ‘ (*P n*)

lemma *UNpart-cases*[*elim*, *consumes 1*, *case-names Local*]:
assumes *II* ∈ *UNpart cl P* **and**
⋀ *n I*. ⟦*n* < *length cl*; *I* ∈ *P n*; *II* = *shift cl n I*⟧ ⟹ *phi*
shows *phi*
using *assms* **unfolding** *UNpart-def* **by** *auto*

lemma *emp-UNpart*:
assumes ⋀ *n*. *n* < *length cl* ⟹ {} ∉ *P n*
shows {} ∉ *UNpart cl P*
using *assms* **unfolding** *UNpart-def* **by** *auto*

lemma *part-UNpart*:
assumes *cl*: *properL cl* **and**
P: ⋀ *n*. *n* < *length cl* ⟹ *part* {..< *brn* (*cl*!*n*)} (*P n*)
shows *part* {..< *brnL cl* (*length cl*)} (*UNpart cl P*)
(**is** *part ?J ?Q*)
proof−
let *?N* = *length cl*

have *J*: *?J* = (⋃ *n* ∈ {..< *?N*}. {*brnL cl n* ..<+ *brn* (*cl*!*n*)})
using *cl brnL-UN* **by** *auto*
have *Q*: *?Q* = (⋃ *n* ∈ {..< *?N*}. *shift cl n* ‘ (*P n*))
unfolding *UNpart-def* **by** *auto*
show *?thesis* **unfolding** *J Q* **apply**(*rule part-UN*)
using *P brnL-Int* **by** *auto*
qed

definition *pickT-pred* **where**
pickT-pred cl P II n ≡ *n* < *length cl* ∧ *II* ∈ *shift cl n* ‘ (*P n*)

definition *pickT* **where**
pickT cl P II ≡ *SOME n*. *pickT-pred cl P II n*

lemma *pickT-pred*:
assumes *II* ∈ *UNpart cl P*
shows ∃ *n*. *pickT-pred cl P II n*
using *assms* **unfolding** *UNpart-def pickT-pred-def* **by** *auto*

lemma *pickT-pred-unique*:
assumes *P*: ⋀ *n*. *n* < *length cl* ⟹ *part* {..< *brn* (*cl*!*n*)} (*P n*) ∧ {} ∉ *P n*
and *1*: *pickT-pred cl P II n1* **and** *2*: *pickT-pred cl P II n2*
shows *n1* = *n2*
proof−
{assume *n1* ≠ *n2*
hence *shift cl n1* ‘ (*P n1*) ∩ *shift cl n2* ‘ (*P n2*) = {}
using *assms part-brn-disj2* **unfolding** *pickT-pred-def* **by** *blast*
hence *False* **using** *1 2* **unfolding** *pickT-pred-def* **by** *blast*
}
thus *?thesis* **by** *auto*
qed

lemma *pickT-pred-pickT*:
assumes *II* ∈ *UNpart cl P*
shows *pickT-pred cl P II* (*pickT cl P II*)
unfolding *pickT-def* **apply**(*rule someI-ex*)
using *assms pickT-pred* **by** *auto*

lemma *pickT-pred-pickT-unique*:
assumes *P*: ⋀ *n*. *n* < *length cl* ⟹ *part* {..< *brn* (*cl*!*n*)} (*P n*) ∧ {} ∉ *P n*
and *pickT-pred cl P II n*
shows *n* = *pickT cl P II*
unfolding *pickT-def* **apply**(*rule sym*, *rule some-equality*)
using *assms pickT-pred-unique*[*of cl P II*] **by** *auto*

lemma *pickT-length*[*simp*]:
assumes *II* ∈ *UNpart cl P*

shows *pickT cl P II* < *length cl*
using *assms pickT-pred-pickT* **unfolding** *pickT-pred-def* **by** *auto*

lemma *pickT-shift*[*simp*]:
assumes *II* ∈ *UNpart cl P*
shows *II* ∈ *shift cl* (*pickT cl P II*) ‘ (*P* (*pickT cl P II*))
using *assms pickT-pred-pickT* **unfolding** *pickT-pred-def* **by** *auto*

lemma *pickT-unique*:
assumes *P*: ⋀ *n*. *n* < *length cl* ⟹ *part* {..< *brn* (*cl*!*n*)} (*P n*) ∧ {} ∉ *P n*
and *n* < *length cl* **and** *II* ∈ *shift cl n* ‘ (*P n*)
shows *n* = *pickT cl P II*
using *assms pickT-pred-pickT-unique* **unfolding** *pickT-pred-def* **by** *auto*

definition *UNlift* **where**
UNlift cl dl P F II ≡
shift dl (*pickT cl P II*) (*F* (*pickT cl P II*) (*back cl* (*pickT cl P II*) *II*))

lemma *UNlift-shift*[*simp*]:
assumes *P*: ⋀ *n*. *n* < *length cl* ⟹ *part* {..< *brn* (*cl*!*n*)} (*P n*) ∧ {} ∉ *P n*
and *n*: *n* < *length cl* **and** *I*: *I* ∈ *P n*
shows *UNlift cl dl P F* (*shift cl n I*) = *shift dl n* (*F n I*)
proof−
let *?N* = *length cl*
define *II* **where** *II* = *shift cl n I*
have *II*: *shift cl n I* = *II* **using** *II-def* **by** *simp*
have *n*: *n* = *pickT cl P II* **apply**(*rule pickT-unique*)
using *assms* **unfolding** *II-def* **by** *auto*
have *back cl n II* = *I* **unfolding** *II-def* **by** *simp*
hence *shift dl n* (*F n* (*back cl n II*)) = *shift dl n* (*F n I*) **by** *simp*
thus *?thesis* **unfolding** *UNlift-def II n*[*THEN sym*] **.**
qed

lemma *UNlift-inj-on*:
assumes *l*: *length cl* = *length dl*
and *P*: ⋀ *n*. *n* < *length cl* ⟹ *part* {..< *brn* (*cl*!*n*)} (*P n*) ∧ {} ∉ *P n*
and *FP*: ⋀ *n*. *n* < *length dl* ⟹ *part* {..< *brn* (*dl*!*n*)} (*F n* ‘ (*P n*)) ∧ {} ∉ *F n* ‘
(*P n*)
and *F*: ⋀ *n*. *n* < *length cl* ⟹ *inj-on* (*F n*) (*P n*)
shows *inj-on* (*UNlift cl dl P F*) (*UNpart cl P*) (**is** *inj-on ?G ?Q*)
unfolding *inj-on-def* **proof** *clarify*
fix *II1 II2*
assume *II1*: *II1* ∈ *?Q* **and** *II2*: *II2* ∈ *?Q* **and** *G*: *?G II1* = *?G II2*
from *II1* **show** *II1* = *II2*
proof(*cases rule*: *UNpart-cases*)
case (*Local n1 I1*)
hence *n1*: *n1* < *length cl n1* < *length dl* **and** *I1*: *I1* ∈ *P n1*
and *II1*: *II1* = *shift cl n1 I1* **using** *l* **by** *auto*
hence *G1-def*: *?G II1* = *shift dl n1* (*F n1 I1*) **using** *P* **by** *simp*

```
    have Pn1: part {..< brn (dl!n1)} (F n1 ‘ (P n1)) {} ∉ F n1 ‘ (P n1)
    using n1 FP by auto
    have F1-in: F n1 I1 ∈ F n1 ‘ (P n1) using I1 by simp
    hence Fn1I1: F n1 I1 ≠ {} F n1 I1 ⊆ {..< brn (dl!n1)}
    using Pn1 by (blast, unfold part-def, blast)
    hence G1: ?G II1 ≠ {} ?G II1 ⊆ {brnL dl n1 ..<+ brn (dl!n1)}
    unfolding G1-def by simp-all
    from II2 show ?thesis
    proof(cases rule: UNpart-cases)
      case (Local n2 I2)
      hence n2: n2 < length cl n2 < length dl and I2: I2 ∈ P n2
      and II2: II2 = shift cl n2 I2 using l by auto
      hence G2-def: ?G II2 = shift dl n2 (F n2 I2) using P by simp
      have Pn2: part {..< brn (dl!n2)} (F n2 ‘ (P n2)) {} ∉ F n2 ‘ (P n2)
      using n2 FP by auto
      have F2-in: F n2 I2 ∈ F n2 ‘ (P n2) using I2 by simp
      hence Fn2I2: F n2 I2 ≠ {} F n2 I2 ⊆ {..< brn (dl!n2)}
      using Pn2 by (blast, unfold part-def, blast)
      hence G2: ?G II2 ≠ {} ?G II2 ⊆ {brnL dl n2 ..<+ brn (dl!n2)}
      unfolding G2-def by simp-all

      have n12: n1 = n2 using n1 n2 G1 G2 G brnL-Int by blast
      have F n1 I1 = F n2 I2 using G unfolding G1-def G2-def n12 by simp
      hence I1 = I2 using I1 I2 n1 F unfolding n12 inj-on-def by simp
      thus ?thesis unfolding II1 II2 n12 by simp
    qed
  qed
qed

lemma UNlift-UNpart:
assumes l: length cl = length dl
and P: ⋀ n. n < length cl ⟹ part {..< brn (cl!n)} (P n) ∧ {} ∉ P n
shows (UNlift cl dl P F) ‘ (UNpart cl P) = UNpart dl (%n. F n ‘ (P n)) (is ?G
‘ ?Q = ?R)
proof safe
  fix II assume II: II ∈ ?Q
  thus ?G II ∈ ?R
  proof(cases rule: UNpart-cases)
    case (Local n I)
    hence n: n < length cl n < length dl and I: I ∈ P n
    and II: II = shift cl n I using l by auto
    hence G: ?G II = shift dl n (F n I) using P by simp
    show ?thesis using n I unfolding G UNpart-def by auto
  qed
next
  fix JJ assume JJ: JJ ∈ ?R
  thus JJ ∈ ?G ‘ ?Q
  proof(cases rule: UNpart-cases)
    case (Local n J)
```

hence *n*: $n < length\ cl\ n < length\ dl$ **and** *J*: $J \in F\ n\ `\ (P\ n)$
and *JJ*: $JJ = shift\ dl\ n\ J$ **using** *l* **by** *auto*
then obtain *I* **where** *I*: $I \in P\ n$ **and** $J = F\ n\ I$ **by** *auto*
hence $JJ = shift\ dl\ n\ (F\ n\ I)$ **using** *JJ* **by** *simp*
also have ... $= UNlift\ cl\ dl\ P\ F\ (shift\ cl\ n\ I)$ **using** *n I P* **by** *simp*
finally have *JJ*: $JJ = UNlift\ cl\ dl\ P\ F\ (shift\ cl\ n\ I)$ **.**
show *?thesis* **using** *n I* **unfolding** *JJ UNpart-def* **by** *auto*
qed
qed

lemma *emp-UNlift-UNpart*:
assumes *l*: $length\ cl = length\ dl$
and *P*: $\bigwedge n.\ n < length\ cl \Longrightarrow part\ \{..<brn\ (cl!n)\}\ (P\ n) \wedge \{\} \notin P\ n$
and *FP*: $\bigwedge n.\ n < length\ dl \Longrightarrow \{\} \notin F\ n\ `\ (P\ n)$
shows $\{\} \notin (UNlift\ cl\ dl\ P\ F)\ `\ (UNpart\ cl\ P)$ (**is** $\{\} \notin ?R$)
proof−
have *R*: $?R = UNpart\ dl\ (\%n.\ F\ n\ `\ (P\ n))$
apply(*rule UNlift-UNpart*) **using** *assms* **by** *auto*
show *?thesis* **unfolding** *R* **apply**(*rule emp-UNpart*) **using** *FP* **by** *simp*
qed

lemma *part-UNlift-UNpart*:
assumes *l*: $length\ cl = length\ dl$ **and** *dl*: *properL dl*
and *P*: $\bigwedge n.\ n < length\ cl \Longrightarrow part\ \{..<brn\ (cl!n)\}\ (P\ n) \wedge \{\} \notin P\ n$
and *FP*: $\bigwedge n.\ n < length\ dl \Longrightarrow part\ \{..<brn\ (dl!n)\}\ (F\ n\ `\ (P\ n))$
shows $part\ \{..<brnL\ dl\ (length\ dl)\}\ ((UNlift\ cl\ dl\ P\ F)\ `\ (UNpart\ cl\ P))$ (**is** *part ?C ?R*)
proof−
have *R*: $?R = UNpart\ dl\ (\%n.\ F\ n\ `\ (P\ n))$
apply(*rule UNlift-UNpart*) **using** *assms* **by** *auto*
show *?thesis* **unfolding** *R* **apply**(*rule part-UNpart*) **using** *dl FP* **by** *auto*
qed

lemma *ss-wt-Par-UNlift*:
assumes *l*: $length\ cl = length\ dl$
and *cldl*: *properL cl properL dl* **and** *II*: $II \in UNpart\ cl\ P$
and *P*: $\bigwedge n.\ n < length\ cl \Longrightarrow part\ \{..<brn\ (cl!n)\}\ (P\ n) \wedge \{\} \notin P\ n$
and *FP*: $\bigwedge n.\ n < length\ dl \Longrightarrow part\ \{..<brn\ (dl!n)\}\ (F\ n\ `\ (P\ n))$
and *sw*:
$\bigwedge n\ I.\ [\![n < length\ cl;\ I \in P\ n]\!] \Longrightarrow$
$sum\ (wt\ (cl\ !\ n)\ s)\ I =$
$sum\ (wt\ (dl\ !\ n)\ t)\ (F\ n\ I)$
and *st*: $s \approx t$
shows
$sum\ (wt\ (Par\ cl)\ s)\ II =$
$sum\ (wt\ (Par\ dl)\ t)\ (UNlift\ cl\ dl\ P\ F\ II)$ (**is** $?L = ?R$)
proof−
let *?N* = *length cl*
let *?p* = %*n. 1 / ?N* **let** *?q* = %*n. 1 / (length dl)*

let *?ss* = %*n*. *s* **let** *?tt* = %*n*. *t*
have *sstt*: ⋀ *n*. *n* < *?N* ⟹ *?ss n* ≈ *?tt n* **using** *st* **by** *auto*
have *pq*: ⋀ *n*. *n* < *?N* ⟹ *?p n* = *?q n* **and** *sstt*: ⋀ *n*. *n* < *?N* ⟹ *?ss n* ≈ *?tt n*
using *assms l* **by** *auto*
from *II* **show** *?thesis*
proof(*cases rule*: *UNpart-cases*)
case (*Local n I*)
hence *n*: *n* < *?N n* < *length dl* **and** *I*: *I* ∈ *P n*
and *II*: *II* = *shift cl n I* **using** *l* **by** *auto*
have *I-sub*: *I* ⊆ {..< *brn* (*cl*!*n*)} **using** *n I P* **unfolding** *part-def* **by** *blast*
hence *FnI-sub*: *F n I* ⊆ {..< *brn* (*dl*!*n*)} **using** *n I FP* **unfolding** *part-def* **by** *blast*
have *?L* = (*?p n*) ∗ *sum* (*wt* (*cl* ! *n*) (*?ss n*)) *I*
unfolding *II* **using** *n cldl I-sub* **by** *simp*
also have ... = (*?q n*) ∗ *sum* (*wt* (*dl* ! *n*) (*?tt n*)) (*F n I*)
using *n pq* **apply** *simp* **using** *I sw*[*of n I*] **unfolding** *l* **by** *auto*
also have ... = *?R*
unfolding *II* **using** *l cldl n FnI-sub P I* **by** *simp*
finally show *?thesis* **.**
qed
qed

definition *thetaSPar* **where**
thetaSPar ≡
{(*Par cl*, *Par dl*) |
cl dl. *properL cl* ∧ *properL dl* ∧ *SbisL cl dl*}

lemma *cont-eff-Par-UNlift*:
assumes *l*: *length cl* = *length dl*
and *cldl*: *properL cl properL dl SbisL cl dl*
and *II*: *II* ∈ *UNpart cl P* **and** *ii*: *ii* ∈ *II* **and** *jj*: *jj* ∈ *UNlift cl dl P F II*
and *P*: ⋀ *n*. *n* < *length cl* ⟹ *part* {..< *brn* (*cl*!*n*)} (*P n*) ∧ {} ∉ *P n*
and *FP*: ⋀ *n*. *n* < *length dl* ⟹ *part* {..< *brn* (*dl*!*n*)} (*F n* ‘ (*P n*))
and *eff-cont*:
⋀*n I i j*. ⟦*n* < *length cl*; *I* ∈ *P n*; *i* ∈ *I*; *j* ∈ *F n I*⟧ ⟹
eff (*cl*!*n*) *s i* ≈ *eff* (*dl*!*n*) *t j* ∧
cont (*cl*!*n*) *s i* ≈*s cont* (*dl*!*n*) *t j*
and *st*: *s* ≈ *t*
shows
eff (*Par cl*) *s ii* ≈ *eff* (*Par dl*) *t jj* ∧
(*cont* (*Par cl*) *s ii*, *cont* (*Par dl*) *t jj*) ∈ *thetaSPar*
(**is** *?eff* ∧ *?cont*)
proof−
let *?N* = *length cl*
let *?p* = %*n*. *1*/*?N* **let** *?q* = %*n*. *1*/(*length dl*)
let *?ss* = %*n*. *s* **let** *?tt* = %*n*. *t*

```
  have sstt: ⋀ n. n < ?N ⟹ ?ss n ≈ ?tt n
  using st l by auto
  have pq: ⋀ n. n < ?N ⟹ ?p n = ?q n and sstt: ⋀ n. n < ?N ⟹ ?ss n ≈ ?tt
n
  using assms l by auto
  from II show ?thesis
  proof(cases rule: UNpart-cases)
    case (Local n I)
    hence n: n < length cl n < length dl and I: I ∈ P n
    and II: II = shift cl n I using l by auto
    from ii II obtain i where i: i ∈ I and ii: ii = brnL cl n + i
    unfolding shift-def by auto
    have i < brn (cl!n) using i I n P unfolding part-def by blast note i = this i
    have jj: jj ∈ shift dl n (F n I) using jj P n I unfolding II by simp
    from jj II obtain j where j: j ∈ F n I and jj: jj = brnL dl n + j
    unfolding shift-def by auto
    have j < brn (dl!n) using j I n FP unfolding part-def by blast note j = this
j
    show ?thesis
    proof
      have eff (cl!n) (?ss n) i ≈ eff (dl!n) (?tt n) j
      using n I i j eff-cont by blast
      thus ?eff unfolding ii jj using st cldl n i j by simp
    next
      have cont (cl!n) (?ss n) i ≈s cont (dl!n) (?tt n) j
      using n I i j eff-cont by blast
      thus ?cont unfolding ii jj thetaSPar-def using n i j l cldl by simp
    qed
  qed
qed

lemma thetaSPar-Sretr: thetaSPar ⊆ Sretr (thetaSPar)
unfolding Sretr-def matchC-C-def proof safe
  fix c d s t
  assume c-d: (c, d) ∈ thetaSPar and st: s ≈ t
  then obtain cl dl where
  c: c = Par cl and d: d = Par dl and
  cldl: properL cl properL dl SbisL cl dl
  unfolding thetaSPar-def by blast
  let ?N = length cl
  let ?ss = %n. s let ?tt = %n. t
  have N: ?N = length dl using cldl by simp
  have sstt: ⋀ n. n < ?N ⟹ ?ss n ≈ ?tt n
  using st N by auto
  let ?phi = %n PFn. mC-C Sbis (cl ! n) (dl ! n) (?ss n) (?tt n) (fst PFn) (snd
PFn)
  {fix n assume n: n < ?N
   hence cl ! n ≈s dl ! n using cldl by auto
   hence ∃ PFn. ?phi n PFn using n Sbis-mC-C sstt by fastforce
```

```
  }
  then obtain PF where phi: ⋀n. n < ?N ⟹ ?phi n (PF n)
  using bchoice[of {..< ?N} ?phi] by blast
  define P F where P = fst o PF and F = snd o PF
  have m: ⋀n. n < ?N ⟹ mC-C Sbis (cl ! n) (dl ! n) (?ss n) (?tt n) (P n) (F
n)
  using phi unfolding P-def F-def by auto

  have brn-c: brn c = brnL cl ?N unfolding c by simp
  have brn-d: brn d = brnL dl (length dl) unfolding d by simp
  have P: ⋀n. n < ?N ⟹ part {..< brn (cl ! n)} (P n) ∧ {} ∉ (P n)
  using m unfolding m-defsAll part-def by auto
  have FP: ⋀n. n < length dl ⟹ part {..< brn (dl ! n)} (F n ‘ (P n)) ∧ {} ∉ F
n ‘ (P n)
  using m N unfolding m-defsAll part-def by auto
  have F: ⋀n. n < ?N ⟹ inj-on (F n) (P n) using m unfolding m-defsAll by
auto
  have sw: ⋀n I. ⟦n < length cl; I ∈ P n⟧ ⟹
    sum (wt (cl ! n) (?ss n)) I = sum (wt (dl ! n) (?tt n)) (F n I)
  using m unfolding mC-C-def mC-C-wt-def by auto
  have eff-cont: ⋀n I i j. ⟦n < length cl; I ∈ P n; i ∈ I; j ∈ F n I⟧ ⟹
    eff (cl!n) (?ss n) i ≈ eff (dl!n) (?tt n) j ∧
    cont (cl!n) (?ss n) i ≈s cont (dl!n) (?tt n) j
  using m unfolding mC-C-def mC-C-eff-cont-def by auto

  define Q G where Q = UNpart cl P and G = UNlift cl dl P F
  note defi = Q-def G-def brn-c brn-d
  show ∃ Q G. mC-C (thetaSPar) c d s t Q G
  apply(rule exI[of - Q]) apply(rule exI[of - G])
  unfolding mC-C-def proof (intro conjI)
   show mC-C-part c d Q G unfolding mC-C-part-def proof(intro conjI)
    show {} ∉ Q unfolding defi apply(rule emp-UNpart) using P by simp
     show {} ∉ G ‘ Q unfolding defi apply(rule emp-UNlift-UNpart) using N
P FP by auto
    show part {..<brn c} Q
    unfolding defi apply(rule part-UNpart) using cldl P by auto
    show part {..<brn d} (G ‘ Q)
    unfolding defi apply(rule part-UNlift-UNpart) using N cldl P FP by auto
   qed
  next
   show inj-on G Q
   unfolding defi apply(rule UNlift-inj-on) using N P FP F by auto
  next
   show mC-C-wt c d s t Q G
   unfolding mC-C-wt-def defi proof clarify
    fix I assume I ∈ UNpart cl P
    thus sum (wt c s) I = sum (wt d t) (UNlift cl dl P F I)
    unfolding c d apply(intro ss-wt-Par-UNlift)
    using N cldl P FP sw st by auto
```

```
    qed
  next
    show mC-C-eff-cont (thetaSPar) c d s t Q G
    unfolding mC-C-eff-cont-def proof clarify
      fix II ii jj assume II: II ∈ Q and ii: ii ∈ II and jj: jj ∈ G II
      thus eff c s ii ≈ eff d t jj ∧ (cont c s ii, cont d t jj) ∈ thetaSPar
      unfolding defi c d apply(intro cont-eff-Par-UNlift)
      using N cldl P FP eff-cont st by blast+
    qed
  qed
qed

lemma thetaSPar-Sbis: thetaSPar ⊆ Sbis
using Sbis-raw-coind thetaSPar-Sretr by blast

theorem Par-Sbis[simp]:
assumes properL cl and properL dl SbisL cl dl
shows Par cl ≈s Par dl
using assms thetaSPar-Sbis unfolding thetaSPar-def by blast
```

4.5 01-bisimilarity versus language constructs

```
lemma ZObis-pres-discr-L: c ≈01 d ⟹ discr d ⟹ discr c
proof (coinduction arbitrary: d c, clarsimp)
  fix s i c d assume i: i < brn c and d: discr d and c-d: c ≈01 d
  then obtain I0 P F where
    match: mC-ZOC ZObis c d s s I0 P F
    using ZObis-mC-ZOC[of c d s s] by blast
  hence ⋃P = {..<brn c}
    using i unfolding mC-ZOC-def mC-ZOC-part-def part-def by simp
  then obtain I where I: I ∈ P and i: i ∈ I using i by auto
  show s ≈ eff c s i ∧ ((∃ d. cont c s i ≈01 d ∧ discr d) ∨ discr (cont c s i))
  proof(cases I = I0)
    case False
    then obtain j where j: j ∈ F I
      using match I False unfolding mC-ZOC-def mC-ZOC-part-def by blast
    hence j < brn d using I match
      unfolding mC-ZOC-def mC-ZOC-part-def part-def apply simp by blast
    hence md: discr (cont d s j) and s: s ≈ eff d s j
      using d discr-cont[of d j s] discr-eff-indis[of d j s] by auto
    have eff c s i ≈ eff d s j and md2: cont c s i ≈01 cont d s j
      using I i j match False unfolding mC-ZOC-def mC-ZOC-eff-cont-def by
auto
    hence s ≈ eff c s i using s indis-sym indis-trans by blast
    thus ?thesis using md md2 by blast
  next
    case True
    hence s ≈ eff c s i ∧ cont c s i ≈01 d
      using match i ZObis-sym unfolding mC-ZOC-def mC-ZOC-eff-cont0-def by
```

blast
  thus *?thesis* **using** *d* **by** *blast*
 qed
qed

theorem *ZObis-pres-discr-R*:
assumes *discr c* **and** *c* ≈*01 d*
shows *discr d*
using *assms ZObis-pres-discr-L ZObis-sym* **by** *blast*

theorem *ZObis-finished-discr-L*:
assumes *c* ≈*01 d* **and** *proper d* **and** *finished d*
shows *discr c*
using *assms ZObis-pres-discr-L* **by** *auto*

theorem *ZObis-finished-discr-R*:
assumes *proper c* **and** *finished c* **and** *c* ≈*01 d*
shows *discr d*
using *assms ZObis-pres-discr-R*[*of c d*] **by** *auto*

theorem *discr-ZObis*[*simp*]:
assumes *proper c* **and** *proper d* **and** *discr c* **and** *discr d*
shows *c* ≈*01 d*
using *assms* **by** *auto*

theorem *Done-ZObis*[*simp*]:
Done ≈*01 Done*
by *simp*

theorem *Atm-ZObis*[*simp*]:
assumes *compatAtm atm*
shows *Atm atm* ≈*01 Atm atm*
using *assms* **by** *simp*

definition *thetaZOSeqI* **where**
thetaZOSeqI ≡
{(*e* ;; *c*, *e* ;; *d*) | *e c d* . *siso e* ∧ *c* ≈*01 d*}

lemma *thetaZOSeqI-ZOretr*:
thetaZOSeqI ⊆ *ZOretr* (*thetaZOSeqI Un ZObis*)
unfolding *ZOretr-def matchC-LC-def* **proof** *safe*
 fix *c d s t*
 assume *c-d*: (*c*, *d*) ∈ *thetaZOSeqI* **and** *st*: *s* ≈ *t*
 then obtain *e c1 d1* **where** *e*: *siso e* **and** *c1d1*: *c1* ≈*01 d1*
 and *c*: *c* = *e* ;; *c1* **and** *d*: *d* = *e* ;; *d1*

unfolding *thetaZOSeqI-def* **by** *auto*
let *?I0* = {} **let** *?J0* = {}
let *?P* = {*?I0*} *Un* {{*i*} | *i* . *i* < *brn e*}
let *?F* = %*I*. *I*
show ∃ *I0 P F*. *mC-ZOC* (*thetaZOSeqI Un ZObis*) *c d s t I0 P F*
apply(*rule exI*[*of* - *?I0*])
apply(*rule exI*[*of* - *?P*]) **apply**(*rule exI*[*of* - *?F*])
unfolding *mC-ZOC-def* **proof** (*intro conjI*)
show *mC-ZOC-part c d s t ?I0 ?P ?F*
unfolding *mC-ZOC-part-def* **proof** (*intro conjI*)
show *part* {..<*brn c*} *?P*
unfolding *part-def* **proof** *safe*
fix *i* **assume** *i* < *brn c*
thus *i* ∈ ⋃ *?P* **using** *c e st siso-cont-indis*[*of e s t*] **by** *auto*
qed (*unfold c*, *simp*)

thus *part* {..<*brn d*} (*?F* ‘ *?P*) **unfolding** *c d* **by** *auto*
qed *auto*
next
show *mC-ZOC-eff-cont* (*thetaZOSeqI Un ZObis*) *c d s t ?I0 ?P ?F*
unfolding *mC-ZOC-eff-cont-def* **proof**(*intro allI impI*, *elim conjE*)
fix *I i j*
assume *I*: *I* ∈ *?P* − {*?I0*} **and** *i*: *i* ∈ *I* **and** *j*: *j* ∈ *I*
then obtain *i′* **where** *j* ∈ *?F* {*i′*} **and** *i′* < *brn e*
using *I j* **by** *auto*
thus *eff c s i* ≈ *eff d t j* ∧ (*cont c s i*, *cont d t j*) ∈ *thetaZOSeqI* ∪ *ZObis*
using *st c-d e i j I* **unfolding** *c d thetaZOSeqI-def*
by (*cases finished* (*cont e s i′*)) *auto*
qed
qed (*insert st c-d c*, *unfold m-defsAll thetaZOSeqI-def part-def*, *auto*)
qed

lemma *thetaZOSeqI-ZObis*:
thetaZOSeqI ⊆ *ZObis*
using *ZObis-coind thetaZOSeqI-ZOretr* **by** *blast*

theorem *Seq-siso-ZObis*[*simp*]:
assumes *siso e* **and** *c2* ≈*01 d2*
shows *e* ;; *c2* ≈*01 e* ;; *d2*
using *assms thetaZOSeqI-ZObis* **unfolding** *thetaZOSeqI-def* **by** *auto*

definition *thetaZOSeqD* **where**
thetaZOSeqD ≡
{(*c1* ;; *c2*, *d1* ;; *d2*) |
c1 c2 d1 d2.
proper c1 ∧ *proper d1* ∧ *proper c2* ∧ *proper d2* ∧
discr c2 ∧ *discr d2* ∧

```
      c1 ≈01 d1}

lemma thetaZOSeqD-ZOretr:
thetaZOSeqD ⊆ ZOretr (thetaZOSeqD Un ZObis)
unfolding ZOretr-def matchC-LC-def proof safe
  fix c d s t
  assume c-d: (c, d) ∈ thetaZOSeqD and st: s ≈ t
  then obtain c1 c2 d1 d2 where
  c1d1: proper c1 proper d1 c1 ≈01 d1 and
  c2d2: proper c2 proper d2 discr c2 discr d2
  and c: c = c1 ;; c2 and d: d = d1 ;; d2
  unfolding thetaZOSeqD-def by auto
  from c1d1 st obtain P F I0
  where match: mC-ZOC ZObis c1 d1 s t I0 P F
  using ZObis-mC-ZOC by blast
  have P: ⋃P = {..<brn c1} and FP: ⋃(F ‘ P) = {..<brn d1}
  using match unfolding mC-ZOC-def mC-ZOC-part-def part-def by metis+
  show ∃I0 P F. mC-ZOC (thetaZOSeqD Un ZObis) c d s t I0 P F
  apply(intro exI[of - I0] exI[of - P] exI[of - F])
  unfolding mC-ZOC-def proof (intro conjI)
    have I0: I0 ∈ P using match unfolding m-defsAll by blast
    show mC-ZOC-eff-cont0 (thetaZOSeqD ∪ ZObis) c d s t I0 F
    unfolding mC-ZOC-eff-cont0-def proof(intro conjI ballI)
      fix i assume i: i ∈ I0
      let ?c1' = cont c1 s i let ?s' = eff c1 s i
      have i < brn c1 using i I0 P by blast note i = this i
      have c1'd1: ?c1' ≈01 d1 proper ?c1'
       using c1d1 i I0 match unfolding mC-ZOC-def mC-ZOC-eff-cont0-def by
auto
      show s ≈ eff c s i
        using i match unfolding c mC-ZOC-def mC-ZOC-eff-cont0-def by simp
      show (cont c s i, d) ∈ thetaZOSeqD ∪ ZObis
      proof(cases finished ?c1')
        case False note f-c1' = False
        hence csi: cont c s i = ?c1' ;; c2 using i unfolding c by simp
        hence (cont c s i, d) ∈ thetaZOSeqD
        using c1'd1 c1d1 c2d2 f-c1' i match
         unfolding csi d thetaZOSeqD-def mC-ZOC-def mC-ZOC-eff-cont0-def by
blast
        thus ?thesis by simp
      next
        case True note f-c1' = True
        hence csi: cont c s i = c2 using i unfolding c by simp
        have discr d1 using f-c1' c1'd1 ZObis-finished-discr-R by blast
        hence c2 ≈01 d using c2d2 c1d1 unfolding d by simp
        thus ?thesis unfolding csi by simp
      qed
    next
      fix j assume j: j ∈ F I0
```

```
      let ?d1' = cont d1 t j let ?t' = eff d1 t j
      have j < brn d1 using j I0 FP by blast note j = this j
      have c1d1': c1 ≈01 ?d1' proper ?d1'
       using c1d1 j I0 match unfolding mC-ZOC-def mC-ZOC-eff-cont0-def by
auto
      show t ≈ eff d t j
        using j match unfolding d mC-ZOC-def mC-ZOC-eff-cont0-def by simp
      show (c, cont d t j) ∈ thetaZOSeqD ∪ ZObis
      proof (cases finished ?d1')
        case False note f-d1' = False
        hence dtj: cont d t j = ?d1' ;; d2 using j unfolding d by simp
        hence (c, cont d t j) ∈ thetaZOSeqD
        using c1d1' c1d1 c2d2 f-d1' j match
         unfolding c dtj thetaZOSeqD-def mC-ZOC-def mC-ZOC-eff-cont0-def by
blast
        thus ?thesis by simp
      next
        case True note f-d1' = True
        hence dtj: cont d t j = d2 using j unfolding d by simp
        hence discr c1 using f-d1' c1d1' ZObis-finished-discr-L by blast
        hence c ≈01 d2 using c2d2 c1d1 unfolding c by simp
        thus ?thesis unfolding dtj by simp
      qed
    qed
  next
    show mC-ZOC-eff-cont (thetaZOSeqD ∪ ZObis) c d s t I0 P F
    unfolding mC-ZOC-eff-cont-def proof(intro allI impI, elim conjE)
      fix i j I assume I : I ∈ P − {I0} and i: i ∈ I and j: j ∈ F I
      let ?c1' = cont c1 s i  let ?d1' = cont d1 t j
      let ?s' = eff c1 s i  let ?t' = eff d1 t j
      have i < brn c1 using i I P by blast note i = this i
      have j < brn d1 using j I FP by blast note j = this j
      have c1'd1': ?c1' ≈01 ?d1' proper ?c1' proper ?d1'
       using c1d1 i j I match unfolding c mC-ZOC-def mC-ZOC-eff-cont-def by
auto
      show eff c s i ≈ eff d t j ∧ (cont c s i, cont d t j) ∈ thetaZOSeqD ∪ ZObis
      (is ?eff ∧ ?cont) proof
       show ?eff using match I i j unfolding c d m-defsAll apply simp by blast
      next
        show ?cont
        proof(cases finished ?c1')
          case True note c1' = True
         hence csi: cont c s i = c2 using i match unfolding c m-defsAll by simp
          show ?thesis
          proof(cases finished ?d1')
            case True
            hence cont d t j = d2 using j match unfolding d m-defsAll by simp
            thus ?thesis using csi c2d2 by simp
          next
```

```
          case False
          hence dtj: cont d t j = ?d1' ;; d2
          using j match unfolding d m-defsAll by simp
          have discr ?d1' using c1'd1' c1' ZObis-finished-discr-R by blast
          thus ?thesis using c1'd1' c2d2 unfolding csi dtj by simp
        qed
      next
        case False
        hence csi: cont c s i = ?c1' ;; c2
        using i match unfolding c m-defsAll by simp
        show ?thesis
        proof(cases finished (cont d1 t j))
          case True note d1' = True
           hence dtj: cont d t j = d2 using j match unfolding d m-defsAll by
simp
          have discr ?c1' using c1'd1' d1' ZObis-finished-discr-L by blast
          thus ?thesis using c1'd1' c2d2 unfolding csi dtj by simp
        next
          case False
         hence dtj: cont d t j = ?d1' ;; d2 using j match unfolding d m-defsAll
by simp
          thus ?thesis unfolding csi dtj thetaZOSeqD-def
          using c1'd1' c2d2 by blast
        qed
      qed
    qed
  qed
qed

lemma thetaZOSeqD-ZObis:
thetaZOSeqD ⊆ ZObis
using ZObis-coind thetaZOSeqD-ZOretr by blast

theorem Seq-ZObis[simp]:
assumes proper c1 and proper d1 and proper c2 and proper d2
and c1 ≈01 d1 and discr c2 and discr d2
shows c1 ;; c2 ≈01 d1 ;; d2
using assms thetaZOSeqD-ZObis unfolding thetaZOSeqD-def by auto


definition thetaZOCh where
thetaZOCh ch c1 c2 d1 d2 ≡ {(Ch ch c1 c2, Ch ch d1 d2)}

lemma thetaZOCh-Sretr:
assumes compatCh ch and c1 ≈01 d1 and c2 ≈01 d2
shows thetaZOCh ch c1 c2 d1 d2 ⊆
      Sretr (thetaZOCh ch c1 c2 d1 d2 ∪ ZObis)
(is ?th ⊆ Sretr (?th ∪ ZObis))
```

unfolding *Sretr-def matchC-C-def* **proof** *safe*
fix *c d s t*
assume *c-d*: $(c, d) \in ?th$ **and** *st*: $s \approx t$
hence *c*: $c = Ch\ ch\ c1\ c2\ brn\ c = 2$
and *d*: $d = Ch\ ch\ d1\ d2\ brn\ d = 2$
unfolding *thetaZOCh-def* **by** *auto*
let $?P = \{\{0\}, \{1\}\}$
let $?F = \%I.\ I$
show $\exists P\ F.\ mC\text{-}C\ (?th\ Un\ ZObis)\ c\ d\ s\ t\ P\ F$
apply(*rule exI*[*of* - *?P*]) **apply**(*rule exI*[*of* - *?F*])
using *assms st c-d c* **unfolding** *m-defsAll thetaZOCh-def part-def* **by** *auto*
qed

lemma *thetaZOCh-ZOretr*:
assumes *compatCh ch* **and** *c1* $\approx$*01 d1* **and** *c2* $\approx$*01 d2*
shows *thetaZOCh ch c1 c2 d1 d2* $\subseteq$
ZOretr (*thetaZOCh ch c1 c2 d1 d2* $\cup$ *ZObis*)
using *thetaZOCh-Sretr*[*OF assms*]
by (*metis* (*no-types*) *Retr-incl subset-trans*)

lemma *thetaZOCh-ZObis*:
assumes *compatCh ch* **and** *c1* $\approx$*01 d1* **and** *c2* $\approx$*01 d2*
shows *thetaZOCh ch c1 c2 d1 d2* $\subseteq$ *ZObis*
using *ZObis-coind thetaZOCh-ZOretr*[*OF assms*] **by** *blast*

theorem *Ch-siso-ZObis*[*simp*]:
assumes *compatCh ch* **and** *c1* $\approx$*01 d1* **and** *c2* $\approx$*01 d2*
shows *Ch ch c1 c2* $\approx$*01 Ch ch d1 d2*
using *thetaZOCh-ZObis*[*OF assms*] **unfolding** *thetaZOCh-def* **by** *auto*

definition *theFTOne* **where**
theFTOne cl dl $\equiv$ *theFT cl* $\cup$ *theFT dl*

definition *theNFTBoth* **where**
theNFTBoth cl dl $\equiv$ *theNFT cl* $\cap$ *theNFT dl*

lemma *theFTOne-sym*: *theFTOne cl dl* = *theFTOne dl cl*
unfolding *theFTOne-def* **by** *auto*

lemma *finite-theFTOne*[*simp*]:
finite (*theFTOne cl dl*)
unfolding *theFTOne-def* **by** *simp*

lemma *theFTOne-length-finished*[*simp*]:
assumes $n \in$ *theFTOne cl dl*
shows ($n <$ *length cl* $\wedge$ *finished* (*cl*!*n*)) $\vee$ ($n <$ *length dl* $\wedge$ *finished* (*dl*!*n*))
using *assms* **unfolding** *theFTOne-def* **by** *auto*

lemma *theFTOne-length*[*simp*]:
assumes *length cl* = *length dl* **and** *n* ∈ *theFTOne cl dl*
shows *n* < *length cl* **and** *n* < *length dl*
using *assms theFTOne-length-finished*[*of n cl dl*] **by** *auto*

lemma *theFTOne-intro*[*intro*]:
assumes ⋀ *n*. (*n* < *length cl* ∧ *finished* (*cl*!*n*)) ∨ (*n* < *length dl* ∧ *finished* (*dl*!*n*))
shows *n* ∈ *theFTOne cl dl*
using *assms* **unfolding** *theFTOne-def* **by** *auto*

lemma *pickFT-theFTOne*[*simp*]:
assumes *WtFT cl* = *1*
shows *pickFT cl* ∈ *theFTOne cl dl*
using *assms* **unfolding** *theFTOne-def* **by** *auto*

lemma *finite-theNFTBoth*[*simp*]:
finite (*theNFTBoth cl dl*)
unfolding *theNFTBoth-def* **by** *simp*

lemma *theNFTBoth-sym*: *theNFTBoth cl dl* = *theNFTBoth dl cl*
unfolding *theNFTBoth-def* **by** *auto*

lemma *theNFTBoth-length-finished*[*simp*]:
assumes *n* ∈ *theNFTBoth cl dl*
shows *n* < *length cl* **and** ¬ *finished* (*cl*!*n*)
and *n* < *length dl* **and** ¬ *finished* (*dl*!*n*)
using *assms* **unfolding** *theNFTBoth-def* **by** *auto*

lemma *theNFTBoth-intro*[*intro*]:
assumes ⋀ *n*. *n* < *length cl* ∧ ¬ *finished* (*cl*!*n*) ∧ *n* < *length dl* ∧ ¬ *finished* (*dl*!*n*)
shows *n* ∈ *theNFTBoth cl dl*
using *assms* **unfolding** *theNFTBoth-def* **by** *auto*

lemma *theFTOne-Int-theNFTBoth*[*simp*]:
theFTOne cl dl ∩ *theNFTBoth cl dl* = {}
and *theNFTBoth cl dl* ∩ *theFTOne cl dl* = {}
unfolding *theFTOne-def theNFTBoth-def theFT-def theNFT-def* **by** *auto*

lemma *theFT-Un-theNFT-One-Both*[*simp*]:
assumes *length cl* = *length dl*
shows
theFTOne cl dl ∪ *theNFTBoth cl dl* = {..< *length cl*} **and**
theNFTBoth cl dl ∪ *theFTOne cl dl* = {..< *length cl*}
using *assms*
unfolding *theFTOne-def theNFTBoth-def theFT-def theNFT-def* **by** *auto*

lemma *in-theFTOne-theNFTBoth*[*simp*]:
assumes *n1* ∈ *theFTOne cl dl* **and** *n2* ∈ *theNFTBoth cl dl*

shows *n1* $\neq$ *n2* **and** *n2* $\neq$ *n1*
using *assms theFTOne-Int-theNFTBoth* **by** *blast+*

definition *BrnFT* **where**
BrnFT cl dl $\equiv \bigcup$ *n* $\in$ *theFTOne cl dl*. {*brnL cl n* ..<+ *brn* (*cl*!*n*)}

definition *BrnNFT* **where**
BrnNFT cl dl $\equiv \bigcup$ *n* $\in$ *theNFTBoth cl dl*. {*brnL cl n* ..<+ *brn* (*cl*!*n*)}

lemma *BrnFT-elim*[*elim*, *consumes 1*, *case-names Local*]:
assumes *ii* $\in$ *BrnFT cl dl*
and $\bigwedge$ *n i*. ⟦*n* $\in$ *theFTOne cl dl*; *i* < *brn* (*cl*!*n*); *ii* = *brnL cl n* + *i*⟧ $\Longrightarrow$ *phi*
shows *phi*
using *assms* **unfolding** *BrnFT-def* **by** *auto*

lemma *finite-BrnFT*[*simp*]:
finite (*BrnFT cl dl*)
unfolding *BrnFT-def* **by** *auto*

lemma *BrnFT-incl-brnL*[*simp*]:
assumes *l*: *length cl* = *length dl* **and** *cl*: *properL cl*
shows *BrnFT cl dl* $\subseteq$ {..< *brnL cl* (*length cl*)} (**is** *?L* $\subseteq$ *?R*)
proof−
have *?L* $\subseteq$ ($\bigcup$ *n*<*length cl*. {*brnL cl n*..<+*brn* (*cl* ! *n*)})
using *l* **unfolding** *BrnFT-def theFTOne-def theFT-def* **by** *auto*
also have ... = *?R* **using** *cl brnL-UN* **by** *auto*
finally show *?thesis* **.**
qed

lemma *BrnNFT-elim*[*elim*, *consumes 1*, *case-names Local*]:
assumes *ii* $\in$ *BrnNFT cl dl*
and $\bigwedge$ *n i*. ⟦*n* $\in$ *theNFTBoth cl dl*; *i* < *brn* (*cl*!*n*); *ii* = *brnL cl n* + *i*⟧ $\Longrightarrow$ *phi*
shows *phi*
using *assms* **unfolding** *BrnNFT-def* **by** *auto*

lemma *finite-BrnNFT*[*simp*]:
finite (*BrnNFT cl dl*)
unfolding *BrnNFT-def* **by** *auto*

lemma *BrnNFT-incl-brnL*[*simp*]:
assumes *cl*: *properL cl*
shows *BrnNFT cl dl* $\subseteq$ {..< *brnL cl* (*length cl*)} (**is** *?L* $\subseteq$ *?R*)
proof−
have *?L* $\subseteq$ ($\bigcup$ *n*<*length cl*. {*brnL cl n*..<+*brn* (*cl* ! *n*)})
unfolding *BrnNFT-def theNFTBoth-def theNFT-def* **by** *auto*
also have ... = *?R* **using** *cl brnL-UN* **by** *auto*

```
  finally show ?thesis .
qed

lemma BrnFT-Int-BrnNFT[simp]:
assumes l: length cl = length dl
shows
BrnFT cl dl ∩ BrnNFT cl dl = {} (is ?L)
and BrnNFT cl dl ∩ BrnFT cl dl = {} (is ?R)
proof−
  {fix ii assume 1: ii ∈ BrnFT cl dl and 2: ii ∈ BrnNFT cl dl
   from 1 have False
   proof (cases rule: BrnFT-elim)
     case (Local n1 i1)
     hence n1: n1 < length cl n1 < length dl n1 ∈ theFTOne cl dl
     and i1: i1 < brn (cl ! n1)
     and ii1: ii = brnL cl n1 + i1 using l by auto
     from 2 show ?thesis
     proof (cases rule: BrnNFT-elim)
       case (Local n2 i2)
       hence n2: n2 ∈ theNFTBoth cl dl and i2: i2 < brn (cl ! n2)
       and ii2: ii = brnL cl n2 + i2 by auto
       have n12: n1 ≠ n2 using n1 n2 by simp
       show ?thesis
       proof(cases n1 < n2)
         case True hence Suc: Suc n1 ≤ n2 by simp
         have ii < brnL cl (Suc n1) unfolding ii1 using i1 n1 by simp
         also have ... ≤ brnL cl n2 using Suc by auto
         also have ... ≤ ii unfolding ii2 by simp
         finally show False by simp
       next
         case False hence Suc: Suc n2 ≤ n1 using n12 by simp
         have ii < brnL cl (Suc n2) unfolding ii2 using i2 n2 by simp
         also have ... ≤ brnL cl n1 using Suc by auto
         also have ... ≤ ii unfolding ii1 by simp
         finally show False by simp
       qed
     qed
   qed
  }
  thus ?L by blast  thus ?R by blast
qed

lemma BrnFT-Un-BrnNFT[simp]:
assumes l: length cl = length dl and cl: properL cl
shows BrnFT cl dl ∪ BrnNFT cl dl = {..< brnL cl (length cl)} (is ?L1 = ?R)
and BrnNFT cl dl ∪ BrnFT cl dl = {..< brnL cl (length cl)} (is ?L2 = ?R)
proof−
  have R: ?R = (⋃ n<length cl. {brnL cl n..<+brn (cl ! n)})
  using cl brnL-UN by auto
```

thus *?L1 = ?R* **unfolding** *R*
unfolding *l BrnFT-def BrnNFT-def*
theFTOne-def theNFTBoth-def theFT-def theNFT-def **by** *blast*
thus *?L2 = ?R* **by** *blast*
qed

lemma *BrnFT-part*:
assumes *l*: *length cl = length dl*
and *P*: $\bigwedge$ *n. n < length cl* $\Longrightarrow$ *part* {..< *brn* (*cl*!*n*)} (*P n*)
shows *BrnFT cl dl* = ($\bigcup$ *n* ∈ *theFTOne cl dl. Union* (*shift cl n* ‘ (*P n*))) (**is** *?L = ?R*)
proof (*safe elim*!: *BrnFT-elim*)
fix *x n i* **assume** *n*: *n* ∈ *theFTOne cl dl* **and** *i*: *i < brn* (*cl* ! *n*)
hence *n < length cl n < length dl* **using** *l* **by** *auto* **note** *n = this n*
hence *i* ∈ *Union* (*P n*) **using** *P i* **unfolding** *part-def* **by** *auto*
then obtain *I* **where** *i* ∈ *I* **and** *I* ∈ *P n* **by** *blast*
thus *brnL cl n + i* ∈ *?R* **using** *n* **unfolding** *shift-def* **by** *auto*
next
fix *n ii I* **assume** *n*: *n* ∈ *theFTOne cl dl* **and** *ii*: *ii* ∈ *shift cl n I* **and** *I*: *I* ∈ *P n*
hence *n < length cl* **using** *l* **by** *simp*
hence *I* ⊆ {..< *brn* (*cl* ! *n*)} **using** *I P* **unfolding** *part-def* **by** *blast*
hence *ii* ∈ {*brnL cl n*..<+*brn* (*cl* ! *n*)} **using** *ii* **unfolding** *shift-def* **by** *auto*
thus *ii* ∈ *?L* **using** *n* **unfolding** *BrnFT-def* **by** *auto*
qed

lemma *brnL-pickFT-BrnFT*[*simp*]:
assumes *properL cl* **and** *WtFT cl = 1*
shows *brnL cl* (*pickFT cl*) ∈ *BrnFT cl dl*
using *assms brn-gt-0-L* **unfolding** *BrnFT-def* **by** *auto*

lemma *WtFT-ParT-BrnFT*[*simp*]:
assumes *length cl = length dl properL cl* **and** *WtFT cl = 1*
shows *sum* (*wt* (*ParT cl*) *s*) (*BrnFT cl dl*) = *1*
proof−
have *brnL cl* (*pickFT cl*) ∈ *BrnFT cl dl* **and**
BrnFT cl dl ⊆ {..<*brnL cl* (*length cl*)}
using *assms BrnFT-incl-brnL* **by** (*simp*, *blast*)
thus *?thesis* **using** *assms* **by** *simp*
qed

definition *UNpart1* **where**
UNpart1 cl dl P ≡ $\bigcup$ *n* ∈ *theNFTBoth cl dl. shift cl n* ‘ (*P n*)

definition *UNpart01* **where**
UNpart01 cl dl P ≡ {*BrnFT cl dl*} ∪ *UNpart1 cl dl P*

lemma *BrnFT-UNpart01*[*simp*]:
BrnFT cl dl ∈ *UNpart01 cl dl P*
unfolding *UNpart01-def* **by** *simp*

lemma *UNpart1-cases*[*elim*, *consumes 1*, *case-names Local*]:
assumes *II* ∈ *UNpart1 cl dl P*
⋀ *n I*. ⟦*n* ∈ *theNFTBoth cl dl*; *I* ∈ *P n*; *II* = *shift cl n I*⟧ ⟹ *phi*
shows *phi*
using *assms* **unfolding** *UNpart1-def* **by** *auto*

lemma *UNpart01-cases*[*elim*, *consumes 1*, *case-names Local0 Local*]:
assumes *II* ∈ *UNpart01 cl dl P* **and** *II* = *BrnFT cl dl* ⟹ *phi*
⋀ *n I*. ⟦*n* ∈ *theNFTBoth cl dl*; *I* ∈ *P n*; *II* = *shift cl n I*; *II* ∈ *UNpart1 cl dl P*⟧
⟹ *phi*
shows *phi*
using *assms* **unfolding** *UNpart01-def UNpart1-def* **by** *auto*

lemma *emp-UNpart1*:
assumes ⋀ *n*. *n* < *length cl* ⟹ {} ∉ *P n*
shows {} ∉ *UNpart1 cl dl P*
using *assms* **unfolding** *UNpart1-def* **by** *auto*

lemma *emp-UNpart01*:
assumes ⋀ *n*. *n* < *length cl* ⟹ {} ∉ *P n*
shows {} ∉ *UNpart01 cl dl P* − {*BrnFT cl dl*}
using *assms emp-UNpart1* **unfolding** *UNpart01-def* **by** *auto*

lemma *BrnFT-Int-UNpart1*[*simp*]:
assumes *l*: *length cl* = *length dl*
and *P*: ⋀ *n*. *n* < *length cl* ⟹ *part* {..< *brn* (*cl*!*n*)} (*P n*) ∧ {} ∉ *P n*
and *II*: *II* ∈ *UNpart1 cl dl P*
shows *BrnFT cl dl* ∩ *II* = {}
using *II* **proof**(*cases rule*: *UNpart1-cases*)
have *1*: *BrnFT cl dl* = (⋃ *n* ∈ *theFTOne cl dl*. *Union* (*shift cl n* ‘ (*P n*)))
apply(*rule BrnFT-part*) **using** *l P* **by** *auto*
case (*Local n I*)
hence *n*: *n* < *length cl n* < *length dl n* ∈ *theNFTBoth cl dl*
and *I*: *I* ∈ *P n* **and** *II*: *II* = *shift cl n I* **by** *auto*
{fix *n0* **assume** *n0*: *n0* ∈ *theFTOne cl dl*
hence *n0* < *length cl n0* < *length dl* **using** *l* **by** *auto* **note** *n0* = *this n0*
{fix *JJ* **assume** *JJ* ∈ *shift cl n0* ‘ (*P n0*)
then obtain *J* **where** *J*: *J* ∈ *P n0* **and** *JJ*: *JJ* = *shift cl n0 J* **by** *auto*
have *n* ≠ *n0* **using** *n n0* **by** *simp*
have *JJ Int II* = {} **unfolding** *JJ II*
apply(*rule part-brn-disj3*) **using** *P I J n n0* **by** *auto*
}
hence *Union* (*shift cl n0* ‘ (*P n0*)) *Int II* = {} **by** *blast*
}
thus *?thesis* **unfolding** *II 1* **by** *blast*

qed

lemma *BrnFT-notIn-UNpart1*:
assumes *l*: *length cl* = *length dl*
and *P*: $\bigwedge$ *n*. *n* < *length cl* $\Longrightarrow$ *part* {..< *brn* (*cl*!*n*)} (*P n*) ∧ {} ∉ *P n*
shows *BrnFT cl dl* ∉ *UNpart1 cl dl P*
using *assms BrnFT-Int-UNpart1 emp-UNpart1* **by** (*metis Int-absorb*)

lemma *UNpart1-UNpart01*:
assumes *l*: *length cl* = *length dl*
and *P*: $\bigwedge$ *n*. *n* < *length cl* $\Longrightarrow$ *part* {..< *brn* (*cl*!*n*)} (*P n*) ∧ {} ∉ *P n*
shows *UNpart1 cl dl P* = *UNpart01 cl dl P* − {*BrnFT cl dl*}
proof−
have *BrnFT cl dl* ∉ *UNpart1 cl dl P*
apply(*rule BrnFT-notIn-UNpart1*) **using** *assms* **by** *auto*
thus *?thesis* **unfolding** *UNpart01-def* **by** *auto*
qed

lemma *part-UNpart1*[*simp*]:
assumes *l*: *length cl* = *length dl*
and *P*: $\bigwedge$ *n*. *n* < *length cl* $\Longrightarrow$ *part* {..< *brn* (*cl*!*n*)} (*P n*)
shows *part* (*BrnNFT cl dl*) (*UNpart1 cl dl P*)
unfolding *BrnNFT-def UNpart1-def* **apply**(*rule part-UN*)
using *l P* **apply** *fastforce*
apply(*rule brnL-Int*) **using** *l* **by** *auto*

lemma *part-UNpart01*:
assumes *cl*: *properL cl* **and** *l*: *length cl* = *length dl*
and *P*: $\bigwedge$ *n*. *n* < *length cl* $\Longrightarrow$ *part* {..< *brn* (*cl*!*n*)} (*P n*) ∧ {} ∉ *P n*
shows *part* {..< *brnL cl* (*length cl*)} (*UNpart01 cl dl P*)
unfolding *UNpart01-def* **apply**(*rule part-Un-singl2*[*of* - - *BrnNFT cl dl*])
using *assms* **using** *BrnFT-Int-UNpart1* **by** (*simp*, *simp*, *blast*)

definition *UNlift01* **where**
UNlift01 cl dl P F II ≡
if II = *BrnFT cl dl*
then BrnFT dl cl
else shift dl (*pickT cl P II*) (*F* (*pickT cl P II*) (*back cl* (*pickT cl P II*) *II*))

lemma *UNlift01-BrnFT*[*simp*]:
UNlift01 cl dl P F (*BrnFT cl dl*) = *BrnFT dl cl*
unfolding *UNlift01-def* **by** *simp*

lemma *UNlift01-shift*[*simp*]:
assumes *l*: *length cl* = *length dl*
and *P*: $\bigwedge$ *n*. *n* < *length cl* $\Longrightarrow$ *part* {..< *brn* (*cl*!*n*)} (*P n*) ∧ {} ∉ *P n*
and *n*: *n* ∈ *theNFTBoth cl dl* **and** *I*: *I* ∈ *P n*

shows *UNlift01 cl dl P F* (*shift cl n I*) = *shift dl n* (*F n I*)
proof−
let *?N* = *length cl*
define *II* **where** *II* = *shift cl n I*
have *n* < *length cl* **using** *n l* **by** *auto* **note** *n* = *this n*
have *II*: *shift cl n I* = *II* **using** *II-def* **by** *simp*
have *II* ∈ *UNpart1 cl dl P* **unfolding** *II-def UNpart1-def* **using** *n I* **by** *auto*
hence *II* ≠ *BrnFT cl dl* **using** *BrnFT-notIn-UNpart1*[*of cl dl P*] *l n P* **by** *auto*
hence *1*: *UNlift01 cl dl P F II* =
shift dl (*pickT cl P II*) (*F* (*pickT cl P II*) (*back cl* (*pickT cl P II*) *II*))
unfolding *UNlift01-def* **by** *simp*
have *n*: *n* = *pickT cl P II* **apply**(*rule pickT-unique*)
using *assms* **unfolding** *II-def* **by** *auto*
have *back cl n II* = *I* **unfolding** *II-def* **by** *simp*
hence *shift dl n* (*F n* (*back cl n II*)) = *shift dl n* (*F n I*) **by** *simp*
thus *?thesis* **unfolding** *1 II n*[*THEN sym*] **.**
qed

lemma *UNlift01-inj-on-UNpart1*:
assumes *l*: *length cl* = *length dl*
and *P*: ⋀ *n*. *n* < *length cl* ⟹ *part* {..< *brn* (*cl*!*n*)} (*P n*) ∧ {} ∉ *P n*
and *FP*: ⋀ *n*. *n* < *length dl* ⟹ *part* {..< *brn* (*dl*!*n*)} (*F n* ‘ (*P n*)) ∧ {} ∉ *F n* ‘ (*P n*)
and *F*: ⋀ *n*. *n* < *length cl* ⟹ *inj-on* (*F n*) (*P n*)
shows *inj-on* (*UNlift01 cl dl P F*) (*UNpart1 cl dl P*) (**is** *inj-on ?G ?Q*)
unfolding *inj-on-def* **proof** *clarify*
fix *II1 II2*
assume *II1*: *II1* ∈ *?Q* **and** *II2*: *II2* ∈ *?Q* **and** *G*: *?G II1* = *?G II2*
from *II1* **show** *II1* = *II2*
proof(*cases rule*: *UNpart1-cases*)
case (*Local n1 I1*)
hence *n1*: *n1* ∈ *theNFTBoth cl dl n1* < *length cl n1* < *length dl* **and** *I1*: *I1* ∈ *P n1*
and *II1*: *II1* = *shift cl n1 I1* **using** *l* **by** *auto*
hence *G1-def*: *?G II1* = *shift dl n1* (*F n1 I1*) **using** *l P* **by** *simp*
have *Pn1*: *part* {..< *brn* (*dl*!*n1*)} (*F n1* ‘ (*P n1*)) {} ∉ *F n1* ‘ (*P n1*)
using *n1 FP* **by** *auto*
have *F1-in*: *F n1 I1* ∈ *F n1* ‘ (*P n1*) **using** *I1* **by** *simp*
hence *Fn1I1*: *F n1 I1* ≠ {} *F n1 I1* ⊆ {..< *brn* (*dl*!*n1*)}
using *Pn1* **by** (*blast*, *unfold part-def*, *blast*)
hence *G1*: *?G II1* ≠ {} *?G II1* ⊆ {*brnL dl n1* ..<+ *brn* (*dl*!*n1*)}
unfolding *G1-def* **by** *simp-all*
from *II2* **show** *?thesis*
proof(*cases rule*: *UNpart1-cases*)
case (*Local n2 I2*)
hence *n2*: *n2* ∈ *theNFTBoth cl dl n2* < *length cl n2* < *length dl*
and *I2*: *I2* ∈ *P n2* **and** *II2*: *II2* = *shift cl n2 I2* **using** *l* **by** *auto*
hence *G2-def*: *?G II2* = *shift dl n2* (*F n2 I2*) **using** *l P* **by** *auto*
have *Pn2*: *part* {..< *brn* (*dl*!*n2*)} (*F n2* ‘ (*P n2*)) {} ∉ *F n2* ‘ (*P n2*)

```
    using n2 FP by auto
    have F2-in: F n2 I2 ∈ F n2 ‘ (P n2) using I2 by simp
    hence Fn2I2: F n2 I2 ≠ {} F n2 I2 ⊆ {..< brn (dl!n2)}
    using Pn2 by (blast, unfold part-def, blast)
    hence G2: ?G II2 ≠ {} ?G II2 ⊆ {brnL dl n2 ..<+ brn (dl!n2)}
    unfolding G2-def by simp-all

    have n12: n1 = n2 using n1 n2 G1 G2 G brnL-Int by blast
    have F n1 I1 = F n2 I2 using G unfolding G1-def G2-def n12 by simp
    hence I1 = I2 using I1 I2 n1 F unfolding n12 inj-on-def by blast
    thus ?thesis unfolding II1 II2 n12 by simp
   qed
  qed
qed

lemma inj-on-singl:
assumes inj-on f A and a0 ∉ A and ⋀ a. a ∈ A ⟹ f a ≠ f a0
shows inj-on f ({a0} Un A)
using assms unfolding inj-on-def by fastforce

lemma UNlift01-inj-on:
assumes l: length cl = length dl
and P: ⋀ n. n < length cl ⟹ part {..< brn (cl!n)} (P n) ∧ {} ∉ P n
and FP: ⋀ n. n < length dl ⟹ part {..< brn (dl!n)} (F n ‘ (P n)) ∧ {} ∉ F n ‘
(P n)
and F: ⋀ n. n < length cl ⟹ inj-on (F n) (P n)
shows inj-on (UNlift01 cl dl P F) (UNpart01 cl dl P)
unfolding UNpart01-def proof(rule inj-on-singl)
  show inj-on (UNlift01 cl dl P F) (UNpart1 cl dl P)
  apply (rule UNlift01-inj-on-UNpart1) using assms by auto
next
  show BrnFT cl dl ∉ UNpart1 cl dl P
  apply(rule BrnFT-notIn-UNpart1) using l P by auto
next
  let ?Q = %n. F n ‘ (P n)
  fix II assume II ∈ UNpart1 cl dl P
  hence UNlift01 cl dl P F II ≠ BrnFT dl cl
  proof(cases rule: UNpart1-cases)
    case (Local n I)
   hence n: n ∈ theNFTBoth cl dl n ∈ theNFTBoth dl cl n < length cl n < length
dl
    and I: I ∈ P n and II: II = shift cl n I using l theNFTBoth-sym by auto
    have shift dl n (F n I) ∈ UNpart1 dl cl ?Q
    unfolding UNpart1-def shift-def using n I by auto
    hence shift dl n (F n I) ≠ BrnFT dl cl
    using BrnFT-notIn-UNpart1[of dl cl ?Q] n l FP by auto
    thus ?thesis unfolding II using n I l P by simp
  qed
  thus UNlift01 cl dl P F II ≠ UNlift01 cl dl P F (BrnFT cl dl) by simp
```

qed

lemma *UNlift01-UNpart1*:
assumes *l*: *length cl* = *length dl*
and *P*: ⋀ *n*. *n* < *length cl* ⟹ *part* {..< *brn* (*cl*!*n*)} (*P n*) ∧ {} ∉ *P n*
shows (*UNlift01 cl dl P F*) ‘ (*UNpart1 cl dl P*) = *UNpart1 dl cl* (%*n*. *F n* ‘ (*P n*)) (**is** *?G* ‘ *?Q* = *?R*)
proof *safe*
  fix *II* **assume** *II*: *II* ∈ *?Q*
  thus *?G II* ∈ *?R*
  proof(*cases rule*: *UNpart1-cases*)
    case (*Local n I*)
    hence *n*: *n* ∈ *theNFTBoth cl dl n* ∈ *theNFTBoth dl cl n* < *length cl*
    n < *length dl* **and** *I*: *I* ∈ *P n*
    and *II*: *II* = *shift cl n I* **using** *l theNFTBoth-sym* **by** *auto*
    hence *G*: *?G II* = *shift dl n* (*F n I*) **using** *l P* **by** *simp*
    show *?thesis* **using** *n I* **unfolding** *G UNpart1-def* **by** *auto*
  qed
next
  fix *JJ* **assume** *JJ*: *JJ* ∈ *?R*
  thus *JJ* ∈ *?G* ‘ *?Q*
  proof(*cases rule*: *UNpart1-cases*)
    case (*Local n J*)
   hence *n*: *n* ∈ *theNFTBoth cl dl n* ∈ *theNFTBoth dl cl n* < *length cl n* < *length dl*
    and *J*: *J* ∈ *F n* ‘ (*P n*)
    and *JJ*: *JJ* = *shift dl n J* **using** *l theNFTBoth-sym* **by** *auto*
    then obtain *I* **where** *I*: *I* ∈ *P n* **and** *J* = *F n I* **by** *auto*
    hence *JJ* = *shift dl n* (*F n I*) **using** *JJ* **by** *simp*
    also have ... = *UNlift01 cl dl P F* (*shift cl n I*) **using** *n I l P* **by** *simp*
    finally have *JJ*: *JJ* = *UNlift01 cl dl P F* (*shift cl n I*) **.**
    show *?thesis* **using** *n l I* **unfolding** *JJ UNpart1-def* **by** *auto*
  qed
qed

lemma *UNlift01-UNpart01*:
assumes *l*: *length cl* = *length dl*
and *P*: ⋀ *n*. *n* < *length cl* ⟹ *part* {..< *brn* (*cl*!*n*)} (*P n*) ∧ {} ∉ *P n*
shows (*UNlift01 cl dl P F*) ‘ (*UNpart01 cl dl P*) = *UNpart01 dl cl* (%*n*. *F n* ‘ (*P n*))
using *assms UNlift01-UNpart1*[*of cl dl P*] **unfolding** *UNpart01-def* **by** *auto*

lemma *emp-UNlift01-UNpart1*:
assumes *l*: *length cl* = *length dl*
and *P*: ⋀ *n*. *n* < *length cl* ⟹ *part* {..< *brn* (*cl*!*n*)} (*P n*) ∧ {} ∉ *P n*
and *FP*: ⋀ *n*. *n* < *length dl* ⟹ {} ∉ *F n* ‘ (*P n*)
shows {} ∉ (*UNlift01 cl dl P F*) ‘ (*UNpart1 cl dl P*) (**is** {} ∉ *?R*)
proof−
  have *R*: *?R* = *UNpart1 dl cl* (%*n*. *F n* ‘ (*P n*))

apply(*rule UNlift01-UNpart1*) **using** *assms* **by** *auto*
show *?thesis* **unfolding** *R* **apply**(*rule emp-UNpart1*) **using** *FP* **by** *simp*
qed

lemma *emp-UNlift01-UNpart01*:
assumes *l*: *length cl* = *length dl*
and *P*: ⋀ *n*. *n* < *length cl* ⟹ *part* {..< *brn* (*cl*!*n*)} (*P n*) ∧ {} ∉ *P n*
and *FP*: ⋀ *n*. *n* < *length dl* ⟹ {} ∉ *F n* ‘ (*P n*)
shows {} ∉ (*UNlift01 cl dl P F*) ‘ (*UNpart01 cl dl P* − {*BrnFT cl dl*})
(**is** {} ∉ *?U* ‘ *?V*)
proof−
have *V*: *?V* = *UNpart1 cl dl P* **apply**(*rule UNpart1-UNpart01*[*THEN sym*])
using *assms* **by** *auto*
show *?thesis* **unfolding** *V* **apply**(*rule emp-UNlift01-UNpart1*)
using *assms* **by** *auto*
qed

lemma *part-UNlift01-UNpart1*:
assumes *l*: *length cl* = *length dl* **and** *dl*: *properL dl*
and *P*: ⋀ *n*. *n* < *length cl* ⟹ *part* {..< *brn* (*cl*!*n*)} (*P n*) ∧ {} ∉ *P n*
and *FP*: ⋀ *n*. *n* < *length dl* ⟹ *part* {..< *brn* (*dl*!*n*)} (*F n* ‘ (*P n*))
shows *part* (*BrnNFT dl cl*) ((*UNlift01 cl dl P F*) ‘ (*UNpart1 cl dl P*)) (**is** *part ?C ?R*)
proof−
let *?Q* = %*n*. *F n* ‘ (*P n*)
have *R*: *?R* = *UNpart1 dl cl ?Q*
apply(*rule UNlift01-UNpart1*[*of cl dl P F*]) **using** *assms* **by** *auto*
show *?thesis* **unfolding** *R* **apply**(*rule part-UNpart1*) **using** *dl l FP* **by** *auto*
qed

lemma *part-UNlift01-UNpart01*:
assumes *l*: *length cl* = *length dl* **and** *dl*: *properL dl*
and *P*: ⋀ *n*. *n* < *length cl* ⟹ *part* {..< *brn* (*cl*!*n*)} (*P n*) ∧ {} ∉ *P n*
and *FP*: ⋀ *n*. *n* < *length dl* ⟹ *part* {..< *brn* (*dl*!*n*)} (*F n* ‘ (*P n*)) ∧ {} ∉ (*F n* ‘ (*P n*))
shows *part* {..< *brnL dl* (*length dl*)} ((*UNlift01 cl dl P F*) ‘ (*UNpart01 cl dl P*))
(**is** *part ?K ?R*)
proof−
let *?G* = *UNlift01 cl dl P F* **let** *?Q* = %*n*. *F n* ‘ (*P n*)
have *R*: *?R* = {*?G* (*BrnFT cl dl*)} ∪ *?G* ‘ (*UNpart1 cl dl P*)
unfolding *UNpart01-def* **by** *simp*
show *?thesis* **unfolding** *R* **apply**(*rule part-Un-singl2*[*of - - BrnNFT dl cl*])
using *assms part-UNlift01-UNpart1*
apply(*force*, *force*)
using *assms* **apply** *simp* **apply**(*rule BrnFT-Int-UNpart1*[*of dl cl ?Q*])
apply(*force*, *force*) **using** *UNlift01-UNpart1* **by** *auto*
qed

lemma *diff-frac-eq-1*:
assumes $b \neq (0::real)$
shows $1 - a \;/\; b = (b - a) \;/\; b$
by (*metis assms diff-divide-distrib divide-self-if*)

lemma *diff-frac-eq-2*:
assumes $b \neq (1::real)$
shows $1 - (a - b) \;/\; (1 - b) = (1 - a) \;/\; (1 - b)$
(**is** *?L = ?R*)
proof−
  have *b*: $1 - b \neq 0$ **using** *assms* **by** *simp*
  hence *?L* = $(1 - b - (a - b)) \;/\; (1 - b)$ (**is** *?L = ?A / ?B*)
  using *diff-frac-eq-1* **by** *blast*
  also have *?A* = $1 - a$ **by** *simp*
  finally show *?thesis* **by** *simp*
qed

lemma *triv-div-mult*:
assumes *vSF*: $vSF \neq (1::real)$
and *L*: $L = (K - vSF) \;/\; (1 - vSF)$ **and** *Ln*: $L \neq 1$
shows $(VS \;/\; (1 - vSF) * V) \;/\; (1 - L) = (VS * V) \;/\; (1 - K)$
(**is** *?A = ?B*)
proof−
  have *vSF-0*: $1 - vSF \neq 0$ **using** *vSF* **by** *simp*
  {assume $K = 1$
   hence $L = 1$ **using** *L vSF* **by** *simp*
   hence *False* **using** *Ln* **by** *simp*
  }
  hence *Kn*: $K \neq 1$ **by** *auto*
  hence *K-0*: $1 - K \neq 0$ **by** *simp*
  have $1 - L = (1 - K) \;/\; (1 - vSF)$ **unfolding** *L* **using** *vSF diff-frac-eq-2* **by** *blast*
  hence *?A* = $(VS \;/\; (1 - vSF) * V) \;/\; ((1 - K) \;/\; (1 - vSF))$ **by** *simp*
  also have *...* = *?B* **using** *vSF-0 K-0* **by** *auto*
  finally show *?thesis* **.**
qed

lemma *ss-wt-ParT-UNlift01*:
assumes *l*: *length cl* = *length dl*
and *cldl*: *properL cl properL dl* **and** *II*: $II \in UNpart01\ cl\ dl\ P - \{BrnFT\ cl\ dl\}$
and *P*: $\bigwedge n.\ n < length\ cl \Longrightarrow part\ \{..<\ brn\ (cl!n)\}\ (P\ n) \wedge \{\} \notin P\ n$
and *FP*: $\bigwedge n.\ n < length\ dl \Longrightarrow part\ \{..<\ brn\ (dl!n)\}\ (F\ n\ `\ (P\ n))$
and *sw*:
$\bigwedge n\ I.\ [\![n < length\ cl;\ I \in P\ n]\!] \Longrightarrow$
    sum $(wt\ (cl\ !\ n)\ s)\ I =$
    sum $(wt\ (dl\ !\ n)\ t)\ (F\ n\ I)$
and *st*: $s \approx t$
and *le1*: *sum* $(wt\ (ParT\ cl)\ s)\ (BrnFT\ cl\ dl) < 1$

sum (*wt* (*ParT dl*) *t*) (*BrnFT dl cl*) < *1*
shows
sum (*wt* (*ParT cl*) *s*) *II* /
(*1* − *sum* (*wt* (*ParT cl*) *s*) (*BrnFT cl dl*)) =
sum (*wt* (*ParT dl*) *t*) (*UNlift01 cl dl P F II*) /
(*1* − *sum* (*wt* (*ParT dl*) *t*) (*BrnFT dl cl*))
(**is** *sum ?vP II* / (*1* − *sum ?vP ?II-0*) =
sum ?wP ?JJ / (*1* − *sum ?wP ?JJ-0*))
proof−
let *?N* = *length cl*
let *?vS* = %*n*. *1* / *?N* **let** *?wS* = %*n*. *1* / (*length dl*)
let *?vSF* = *WtFT cl* **let** *?wSF* = *WtFT dl*
let *?ss* = %*n*. *s* **let** *?tt* = %*n*. *t*
let *?v* = %*n*. *wt* (*cl* ! *n*) (*?ss n*) **let** *?w* = %*n*. *wt* (*dl* ! *n*) (*?tt n*)

have *sstt*: ⋀ *n*. *n* < *?N* ⟹ *?ss n* ≈ *?tt n*
using *st l* **by** *auto*
have *vSwS*: ⋀ *n*. *n* < *?N* ⟹ *?vS n* = *?wS n* **and** *sstt*: ⋀ *n*. *n* < *?N* ⟹ *?ss n* ≈ *?tt n*
using *assms* **by** *auto*
have *nf*: *?vSF* ≠ *1* *?wSF* ≠ *1* **using** *le1 cldl l* **by** *auto*
have *theFT-theNFT*[*simp*]:
⋀ *n*. *n* ∈ *theFT dl* − *theFT cl* ⟹ *n* < *length cl* ∧ ¬ *finished* (*cl* ! *n*)
⋀ *n*. *n* ∈ *theFT cl* − *theFT dl* ⟹ *n* < *length dl* ∧ ¬ *finished* (*dl* ! *n*)
unfolding *theFT-def* **using** *l* **by** *auto*
have *sum-v*[*simp*]:
⋀ *n*. *n* < *length cl* ⟹ *sum* (*?v n*) {..< *brn* (*cl* ! *n*)} = *1*
using *cldl* **by** *auto*
have *sum-w*[*simp*]:
⋀ *n*. *n* < *length dl* ⟹ *sum* (*?w n*) {..< *brn* (*dl* ! *n*)} = *1*
using *cldl* **by** *auto*
have *theFTOne*: *theFTOne cl dl* = *theFT dl* − *theFT cl* ∪ *theFT cl*
theFTOne dl cl = *theFT cl* − *theFT dl* ∪ *theFT dl*
unfolding *theFTOne-def* **by** *blast*+
have *sum-vS-wS*: *sum ?vS* (*theFTOne cl dl*) = *sum ?wS* (*theFTOne dl cl*)
unfolding *theFTOne-sym*[*of cl dl*] **apply** (*rule sum.cong*)
using *vSwS l* **unfolding** *theFTOne-def theFT-def theNFT-def* **by** *auto*

have *II*: *II* ∈ *UNpart1 cl dl P* **using** *II l P UNpart1-UNpart01* **by** *blast*
thus *?thesis*
proof(*cases rule*: *UNpart1-cases*)
case (*Local n I*)
hence *n*: *n* < *?N n* < *length dl n* ∈ *theNFTBoth cl dl*
¬ *finished* (*cl*!*n*) ¬ *finished* (*dl*!*n*)
and *I*: *I* ∈ *P n*
and *II*: *II* = *shift cl n I* **using** *l* **by** *auto*
have *I-sub*: *I* ⊆ {..< *brn* (*cl*!*n*)} **using** *n I P* **unfolding** *part-def* **by** *blast*
hence *FnI-sub*: *F n I* ⊆ {..< *brn* (*dl*!*n*)} **using** *n I FP* **unfolding** *part-def* **by** *blast*

have *JJ*: *?JJ* = *shift dl n* (*F n I*)
unfolding *II* **using** *l P n I* **by** *simp*

have *sum ?vP ?II-0* =
sum (%*n*. *sum ?vP* {*brnL cl n..<+brn* (*cl* ! *n*)}) (*theFTOne cl dl*)
unfolding *BrnFT-def* **apply**(*rule sum.UNION-disjoint*)
using *brnL-Int l* **by** *auto*
also have ... =
sum
(%*n*. *sum ?vP* {*brnL cl n..<+brn* (*cl* ! *n*)})
((*theFT dl* − *theFT cl*) ∪ *theFT cl*) **unfolding** *theFTOne-def*
by (*metis Un-Diff-cancel2 Un-commute*)
also have ... =
sum
(%*n*. *sum ?vP* {*brnL cl n..<+brn* (*cl* ! *n*)})
(*theFT dl* − *theFT cl*) +
sum
(%*n*. *sum ?vP* {*brnL cl n..<+brn* (*cl* ! *n*)})
(*theFT cl*) (**is** ... = *?L* + *?R*)
apply(*rule sum.union-disjoint*) **by** *auto*
also have *?R* = *0* **apply**(*rule sum.neutral*) **using** *cldl nf* **by** *auto*
finally have *sum ?vP ?II-0* = *?L* **by** *simp*
also have *?L* =
sum
(%*n*. *?vS n* / (*1* − *?vSF*) ∗ *sum* (*?v n*) {*..<* *brn* (*cl* ! *n*)})
(*theFT dl* − *theFT cl*)
apply(*intro sum.cong*) **using** *cldl nf*
using *theFT-theNFT sum-wt-ParT-notWtFT-notFinished*[*of cl*] **by** *metis*+
also have ... =
sum
(%*n*. *?vS n* ∗ *sum* (*?v n*) {*..<* *brn* (*cl* ! *n*)})
(*theFT dl* − *theFT cl*) / (*1* − *?vSF*) (**is** ... = *?L* / (*1* − *?vSF*))
unfolding *times-divide-eq-left sum-divide-distrib* **by** *simp*
also have *?L* = *sum ?vS* (*theFT dl* − *theFT cl*)
apply(*intro sum.cong*) **by** *auto*
finally have
sum ?vP ?II-0 = (*sum ?vS* (*theFT dl* − *theFT cl*)) / (*1* − *?vSF*)
(**is** ... = *?L* / *?R*) **by** *simp*
also have *?L* = *sum ?vS* (*theFTOne cl dl*) − *?vSF*
unfolding *eq-diff-eq WtFT-def theFTOne*
apply(*rule sum.union-disjoint*[*THEN sym*]) **by** *auto*
finally have *vPII0*: *sum ?vP ?II-0* =
(*sum ?vS* (*theFTOne cl dl*) − *?vSF*) / (*1* − *?vSF*) **by** *simp*

have *sum ?wP ?JJ-0* =
sum (%*n*. *sum ?wP* {*brnL dl n..<+brn* (*dl* ! *n*)}) (*theFTOne dl cl*)
unfolding *BrnFT-def* **apply**(*rule sum.UNION-disjoint*)
unfolding *theFTOne-def theFT-def* **apply** (*force*, *force*, *clarify*)

apply(*rule brnL-Int*) **using** *l* **by** *auto*
also have ... =
sum
(%*n*. *sum ?wP* {*brnL dl n..<+ brn* (*dl* ! *n*)})
((*theFT cl* − *theFT dl*) ∪ *theFT dl*) **unfolding** *theFTOne-def*
by (*metis Un-Diff-cancel2 Un-commute*)
also have ... =
sum
(%*n*. *sum ?wP* {*brnL dl n..<+brn* (*dl* ! *n*)})
(*theFT cl* − *theFT dl*) +
sum
(%*n*. *sum ?wP* {*brnL dl n..<+brn* (*dl* ! *n*)})
(*theFT dl*) (**is** ... = *?L* + *?R*)
apply(*rule sum.union-disjoint*) **by** *auto*
also have *?R* = *0* **apply**(*rule sum.neutral*) **using** *cldl nf* **by** *auto*
finally have *sum ?wP ?JJ-0* = *?L* **by** *simp*
also have *?L* =
sum
(%*n*. *?wS n* / (*1* − *?wSF*) ∗ *sum* (*?w n*) {..< *brn* (*dl* ! *n*)})
(*theFT cl* − *theFT dl*)
apply(*intro sum.cong*) **using** *cldl nf*
using *theFT-theNFT sum-wt-ParT-notWtFT-notFinished*[*of dl*] **by** *metis+*
also have ... =
sum
(%*n*. *?wS n* ∗ *sum* (*?w n*) {..< *brn* (*dl* ! *n*)})
(*theFT cl* − *theFT dl*) / (*1* − *?wSF*) (**is** ... = *?L* / (*1* − *?wSF*))
unfolding *times-divide-eq-left sum-divide-distrib* **by** *simp*
also have *?L* = *sum ?wS* (*theFT cl* − *theFT dl*)
apply(*intro sum.cong*) **by** *auto*
finally have
sum ?wP ?JJ-0 = (*sum ?wS* (*theFT cl* − *theFT dl*)) / (*1* − *?wSF*)
(**is** ... = *?L* / *?R*) **by** *simp*
also have *?L* = *sum ?wS* (*theFTOne dl cl*) − *?wSF*
unfolding *eq-diff-eq WtFT-def theFTOne*
apply(*rule sum.union-disjoint*[*THEN sym*]) **by** *auto*
finally have *wPJJ0*: *sum ?wP ?JJ-0* =
(*sum ?wS* (*theFTOne dl cl*) − *?wSF*) / (*1* − *?wSF*) **by** *simp*

have *sum ?vP II* / (*1* − *sum ?vP ?II-0*) =
(*?vS n*) / (*1* − *?vSF*) ∗ (*sum* (*?v n*) *I*) / (*1* − *sum ?vP ?II-0*)
unfolding *II* **using** *n nf cldl I-sub* **by** *simp*
also have ... =
(*?vS n*) ∗ (*sum* (*?v n*) *I*) / (*1* − *sum ?vS* (*theFTOne cl dl*))
using *nf*(*1*) *vPII0* **by** (*rule triv-div-mult*) (*insert le1*, *auto*)
also have ... =
(*?wS n*) ∗ (*sum* (*?w n*) (*F n I*)) / (*1* − *sum ?wS* (*theFTOne dl cl*))
using *n vSwS*[*of n*] *sw*[*of n I*] *I* **unfolding** *sum-vS-wS* **by** *simp*
also have ... =
(*?wS n*) / (*1* − *?wSF*) ∗ (*sum* (*?w n*) (*F n I*)) / (*1* − *sum ?wP ?JJ-0*)

```
    using nf(2) wPJJ0 by (rule triv-div-mult[THEN sym]) (insert le1, auto)
   also have ... = sum ?wP ?JJ / (1 − sum ?wP ?JJ-0)
   unfolding JJ using n nf cldl FnI-sub by simp
   finally show ?thesis .
 qed
qed



definition thetaZOParT where
thetaZOParT ≡
 {(ParT cl, ParT dl) |
    cl dl.
     properL cl ∧ properL dl ∧ SbisL cl dl}

lemma cont-eff-ParT-BrnFT-L:
assumes l: length cl = length dl
and cldl: properL cl properL dl SbisL cl dl
and ii: ii ∈ BrnFT cl dl
and eff-cont:
⋀n I i j. ⟦n < length cl; I ∈ P n; i ∈ I; j ∈ F n I⟧ ⟹
  eff (cl!n) s i ≈ eff (dl!n) t j ∧
  cont (cl!n) s i ≈s cont (dl!n) t j
shows
s ≈ eff (ParT cl) s ii ∧
 (cont (ParT cl) s ii, ParT dl) ∈ thetaZOParT
(is ?eff ∧ ?cont)
proof−
  let ?N = length cl let ?p = %n. 1 / length cl let ?ss = %n. s
  from ii show ?thesis
  proof(cases rule: BrnFT-elim)
    case (Local n i)
    hence n: n ∈ theFTOne cl dl
    and i: i < brn (cl ! n) and ii: ii = brnL cl n + i by auto
    from n have n < length cl n < length dl using l cldl
    unfolding theFTOne-def theFT-def by auto note n = this n
    have discr: discr (cl!n)
    proof(cases finished (cl!n))
      case True
      thus ?thesis using n cldl discr-finished by auto
    next
      case False
      hence finished (dl!n) using n unfolding theFTOne-def theFT-def by auto
      moreover have proper (cl!n) and proper (dl!n) and cl!n ≈01 dl!n
      using n cldl by auto
      ultimately show ?thesis using ZObis-finished-discr-L by blast
    qed
    hence eff: ?ss n ≈ eff (cl!n) (?ss n) i
    and cont: proper (cont (cl!n) (?ss n) i) ∧ discr (cont (cl!n) (?ss n) i)
```

```
    using i cldl n by auto
    show ?thesis
    proof
      have s ≈ eff (cl!n) (?ss n) i using eff n indis-trans by blast
      thus ?eff using i n cldl unfolding ii by simp
    next
      have cont (cl!n) (?ss n) i ≈s cl!n using discr cont cldl n by auto
      moreover have cl!n ≈s dl!n using cldl n by auto
      ultimately have cont (cl!n) (?ss n) i ≈s dl!n using Sbis-trans by blast
      thus ?cont using i n cldl unfolding ii thetaZOParT-def by auto
    qed
  qed
qed

lemma cont-eff-ParT-BrnFT-R:
assumes l: length cl = length dl
and cldl: properL cl properL dl SbisL cl dl
and jj: jj ∈ BrnFT dl cl
and eff-cont:
⋀n I i j. ⟦n < length cl; I ∈ P n; i ∈ I; j ∈ F n I⟧ ⟹
  eff (cl!n) s i ≈ eff (dl!n) t j ∧ cont (cl!n) s i ≈s cont (dl!n) t j
shows
t ≈ eff (ParT dl) t jj ∧
 (ParT cl, cont (ParT dl) t jj) ∈ thetaZOParT
(is ?eff ∧ ?cont)
proof−
  let ?N = length dl  let ?q = %n. 1 /?N let ?tt = %n. t
  from jj show ?thesis
  proof(cases rule: BrnFT-elim)
    case (Local n j)
    hence n: n ∈ theFTOne dl cl
    and j: j < brn (dl ! n) and jj: jj = brnL dl n + j by auto
    from n have n < length cl n < length dl using l cldl
    unfolding theFTOne-def theFT-def by auto note n = this n
    have discr: discr (dl!n)
    proof(cases finished (dl!n))
      case True
      thus ?thesis using n cldl discr-finished by auto
    next
      case False
      hence finished (cl!n) using n unfolding theFTOne-def theFT-def by auto
      moreover have proper (cl!n) and proper (dl!n) and cl!n ≈01 dl!n
      using n cldl by auto
      ultimately show ?thesis using ZObis-finished-discr-R by blast
    qed
    hence eff: ?tt n ≈ eff (dl!n) (?tt n) j
    and cont: proper (cont (dl!n) (?tt n) j) ∧ discr (cont (dl!n) (?tt n) j)
    using j cldl n by auto
    show ?thesis
```

```
    proof
      have t ≈ eff (dl!n) (?tt n) j using eff n indis-trans by blast
      thus ?eff using j n cldl unfolding jj by simp
    next
      have cl!n ≈s dl!n using cldl n by auto
     moreover have dl!n ≈s cont (dl!n) (?tt n) j using discr cont cldl n by auto
      ultimately have cl!n ≈s cont (dl!n) (?tt n) j using Sbis-trans by blast
      thus ?cont using j n cldl unfolding jj thetaZOParT-def by simp
    qed
  qed
qed

lemma cont-eff-ParT-UNlift01:
assumes l: length cl = length dl
and cldl: properL cl properL dl SbisL cl dl
and II: II ∈ UNpart01 cl dl P − {BrnFT cl dl}
and ii: ii ∈ II and jj: jj ∈ UNlift01 cl dl P F II
and P: ⋀ n. n < length cl ⟹ part {..< brn (cl!n)} (P n) ∧ {} ∉ P n
and FP: ⋀ n. n < length dl ⟹ part {..< brn (dl!n)} (F n ‘ (P n))
and eff-cont:
⋀n I i j. ⟦n < length cl; I ∈ P n; i ∈ I; j ∈ F n I⟧ ⟹
  eff (cl!n) s i ≈
  eff (dl!n) t j ∧
  cont (cl!n) s i ≈s
  cont (dl!n) t j
and st: s ≈ t
shows
eff (ParT cl) s ii ≈ eff (ParT dl) t jj ∧
 (cont (ParT cl) s ii, cont (ParT dl) t jj) ∈ thetaZOParT
(is ?eff ∧ ?cont)
proof−
  let ?N = length cl
  let ?p = %n. 1 / ?N let ?q = %n. 1 / (length dl)
  let ?ss = %n. s let ?tt = %n. t
  have sstt: ⋀ n. n < ?N ⟹ ?ss n ≈ ?tt n using st l by auto
  have pq: ⋀ n. n < ?N ⟹ ?p n = ?q n and sstt: ⋀ n. n < ?N ⟹ ?ss n ≈ ?tt
n
  using assms l by auto
  have II: II ∈ UNpart1 cl dl P using II l P UNpart1-UNpart01 by blast
  thus ?thesis
  proof(cases rule: UNpart1-cases)
    case (Local n I)
    hence n: n < ?N n < length dl n ∈ theNFTBoth cl dl
    ¬ finished (cl!n)  ¬ finished (dl!n)
    and I: I ∈ P n and II: II = shift cl n I using l by auto
    from ii II obtain i where i: i ∈ I and ii: ii = brnL cl n + i
    unfolding shift-def by auto
    have i < brn (cl!n) using i I n P unfolding part-def by blast note i = this i
    have jj: jj ∈ shift dl n (F n I) using jj P n I l unfolding II by simp
```

```
    from jj II obtain j where j: j ∈ F n I and jj: jj = brnL dl n + j
    unfolding shift-def by auto
    have j < brn (dl!n) using j I n FP unfolding part-def by blast note j = this
j
    show ?thesis
    proof
      have eff (cl!n) (?ss n) i ≈ eff (dl!n) (?tt n) j
      using n I i j eff-cont by blast
      thus ?eff unfolding ii jj using st cldl n i j by simp
    next
      have 1: cont (cl!n) (?ss n) i ≈s cont (dl!n) (?tt n) j
      using n I i j eff-cont by blast
      have (cont (ParT cl) s ii, cont (ParT dl) t jj) =
            (ParT (cl[n := cont (cl ! n) (?ss n) i]),
              ParT (dl[n := cont (dl ! n) (?tt n) j]))
      (is ?A = ?B)
      unfolding ii jj using n i j cldl by simp
      moreover have ?B ∈ thetaZOParT
      unfolding thetaZOParT-def apply (simp, safe)
      apply(intro properL-update)
        using cldl apply force
        apply(rule proper-cont) using cldl i n apply (force,force)
      apply(intro properL-update)
        using cldl apply force
        apply(rule proper-cont) using cldl j n apply (force,force)
      apply(intro SbisL-update)
      using 1 cldl n i apply (force,force)
      done
      ultimately show ?cont by auto
    qed
  qed
qed

lemma thetaZOParT-ZOretr: thetaZOParT ⊆ ZOretr (thetaZOParT)
unfolding ZOretr-def matchC-LC-def proof safe
  fix c d s t
  assume c-d: (c, d) ∈ thetaZOParT and st: s ≈ t
  then obtain cl dl where
  c: c = ParT cl and d: d = ParT dl and
  cldl: properL cl properL dl SbisL cl dl
  unfolding thetaZOParT-def by blast
  let ?N = length cl
  let ?ss = %n. s let ?tt = %n. t
  have N: ?N = length dl using cldl by simp
  have sstt: ⋀ n. n < ?N ⟹ ?ss n ≈ ?tt n using st N by auto
  let ?phi = %n PFn. mC-C Sbis (cl ! n) (dl ! n) (?ss n) (?tt n) (fst PFn) (snd
PFn)
  {fix n assume n: n < ?N
   hence cl ! n ≈s dl ! n using cldl by auto
```

hence $\exists$ *PFn. ?phi n PFn* **using** *n Sbis-mC-C sstt* **by** *fastforce*
}
then obtain *PF* **where** *phi*: $\bigwedge n.\ n < ?N \Longrightarrow ?phi\ n\ (PF\ n)$
using *bchoice*[*of* {..< *?N*} *?phi*] **by** *blast*
define *P F* **where** *P* = *fst o PF* **and** *F* = *snd o PF*
have *m*: $\bigwedge n.\ n < ?N \Longrightarrow$ *mC-C Sbis* (*cl* ! *n*) (*dl* ! *n*) (*?ss n*) (*?tt n*) (*P n*) (*F n*)
using *phi* **unfolding** *P-def F-def* **by** *auto*

have *brn-c*: *brn c* = *brnL cl ?N* **unfolding** *c* **by** *simp*
have *brn-d*: *brn d* = *brnL dl* (*length dl*) **unfolding** *d* **by** *simp*
have *P*: $\bigwedge n.\ n < ?N \Longrightarrow$ *part* {..< *brn* (*cl* ! *n*)} (*P n*) $\wedge$ {} $\notin$ (*P n*)
using *m* **unfolding** *m-defsAll part-def* **by** *auto*
have *FP*: $\bigwedge n.\ n <$ *length dl* $\Longrightarrow$ *part* {..< *brn* (*dl* ! *n*)} (*F n* ‘ (*P n*)) $\wedge$ {} $\notin$ *F n* ‘ (*P n*)
using *m N* **unfolding** *m-defsAll part-def* **by** *auto*
have *F*: $\bigwedge n.\ n < ?N \Longrightarrow$ *inj-on* (*F n*) (*P n*) **using** *m* **unfolding** *m-defsAll* **by** *auto*
have *sw*: $\bigwedge n\ I.\ [\![n <$ *length cl*; *I* $\in$ *P n*$]\!] \Longrightarrow$
sum (*wt* (*cl* ! *n*) (*?ss n*)) *I* = *sum* (*wt* (*dl* ! *n*) (*?tt n*)) (*F n I*)
using *m* **unfolding** *mC-C-def mC-C-wt-def* **by** *auto*
have *eff-cont*: $\bigwedge n\ I\ i\ j.\ [\![n <$ *length cl*; *I* $\in$ *P n*; *i* $\in$ *I*; *j* $\in$ *F n I*$]\!] \Longrightarrow$
eff (*cl*!*n*) (*?ss n*) *i* $\approx$ *eff* (*dl*!*n*) (*?tt n*) *j* $\wedge$ *cont* (*cl*!*n*) (*?ss n*) *i* $\approx s$ *cont* (*dl*!*n*) (*?tt n*) *j*
using *m* **unfolding** *mC-C-def mC-C-eff-cont-def* **by** *auto*

define *II0* **where** *II0* = *BrnFT cl dl*
define *Q G* **where** *Q* = *UNpart01 cl dl P* **and** *G* = *UNlift01 cl dl P F*
note *defi* = *II0-def Q-def G-def brn-c brn-d*
show $\exists$ *II0 Q G. mC-ZOC* (*thetaZOParT*) *c d s t II0 Q G*
apply(*rule exI*[*of - II0*]) **apply**(*rule exI*[*of - Q*]) **apply**(*rule exI*[*of - G*])
unfolding *mC-ZOC-def* **proof** (*intro conjI*)
show *mC-ZOC-part c d s t II0 Q G* **unfolding** *mC-ZOC-part-def* **proof**(*intro conjI*)
show {} $\notin$ *Q* $-$ {*II0*} **unfolding** *defi* **apply**(*rule emp-UNpart01*) **using** *P* **by** *simp*
show {} $\notin$ *G* ‘ (*Q* $-$ {*II0*}) **unfolding** *defi*
apply(*rule emp-UNlift01-UNpart01*) **using** *N P FP* **by** *auto*
show *II0* $\in$ *Q* **unfolding** *defi* **by** *simp*
show *part* {..<*brn c*} *Q*
unfolding *defi* **apply**(*rule part-UNpart01*) **using** *cldl P* **by** *auto*
show *part* {..<*brn d*} (*G* ‘ *Q*)
unfolding *defi* **apply**(*rule part-UNlift01-UNpart01*) **using** *N cldl P FP* **by** *auto*
qed
next
show *inj-on G Q*
unfolding *defi* **apply**(*rule UNlift01-inj-on*) **using** *N P FP F* **by** *auto*
next

```
    show mC-ZOC-wt c d s t II0 Q G
    unfolding mC-ZOC-wt-def proof (intro impI ballI, elim conjE)
      fix II assume II: II ∈ Q − {II0} and
      le1: sum (wt c s) II0 < 1 sum (wt d t) (G II0) < 1
      thus
      sum (wt c s) II / (1 − sum (wt c s) II0) =
       sum (wt d t) (G II) / (1 − sum (wt d t) (G II0))
      unfolding c d defi UNlift01-BrnFT apply(intro ss-wt-ParT-UNlift01)
      using N cldl II P FP sw st by auto
    qed
  next
    show mC-ZOC-eff-cont0 (thetaZOParT) c d s t II0 G
    unfolding mC-ZOC-eff-cont0-def
    proof(intro conjI[OF ballI ballI])
      fix ii assume ii ∈ II0 thus s ≈ eff c s ii ∧ (cont c s ii, d) ∈ thetaZOParT
      unfolding defi c d apply(intro cont-eff-ParT-BrnFT-L)
      using N cldl P FP eff-cont st by (auto intro!: )
    next
      fix jj assume jj ∈ G II0
      hence jj ∈ BrnFT dl cl unfolding defi UNlift01-BrnFT by simp
      thus t ≈ eff d t jj ∧ (c, cont d t jj) ∈ thetaZOParT
      unfolding defi c d apply(intro cont-eff-ParT-BrnFT-R)
      using N cldl P FP eff-cont st by auto
    qed
  next
    show mC-ZOC-eff-cont (thetaZOParT) c d s t II0 Q G
    unfolding mC-ZOC-eff-cont-def proof (intro allI impI, elim conjE)
      fix II ii jj assume II: II ∈ Q − {II0} and ii: ii ∈ II and jj: jj ∈ G II
      thus eff c s ii ≈ eff d t jj ∧ (cont c s ii, cont d t jj) ∈ thetaZOParT
      unfolding defi c d apply(intro cont-eff-ParT-UNlift01)
      using N cldl P FP eff-cont st by auto
    qed
  qed
qed

lemma thetaZOParT-ZObis: thetaZOParT ⊆ ZObis
using ZObis-raw-coind thetaZOParT-ZOretr by auto

theorem ParT-ZObis[simp]:
assumes properL cl and properL dl and SbisL cl dl
shows ParT cl ≈01 ParT dl
using assms thetaZOParT-ZObis unfolding thetaZOParT-def by blast


end



end
```

5 Syntactic Criteria

theory *Syntactic-Criteria*
imports *Compositionality*
begin

context *PL-Indis*
begin

lemma *proper-intros*[*intro*]:
proper Done
proper (*Atm atm*)
proper c1 $\Longrightarrow$ *proper c2* $\Longrightarrow$ *proper* (*Seq c1 c2*)
proper c1 $\Longrightarrow$ *proper c2* $\Longrightarrow$ *proper* (*Ch ch c1 c2*)
proper c $\Longrightarrow$ *proper* (*While tst c*)
properL cs $\Longrightarrow$ *proper* (*Par cs*)
properL cs $\Longrightarrow$ *proper* (*ParT cs*)
($\bigwedge$*c*. *c* $\in$ *set cs* $\Longrightarrow$ *proper c*) $\Longrightarrow$ *cs* $\neq$ [] $\Longrightarrow$ *properL cs*
by *auto*

lemma *discr*:
discr Done
presAtm atm $\Longrightarrow$ *discr* (*Atm atm*)
discr c1 $\Longrightarrow$ *discr c2* $\Longrightarrow$ *discr* (*Seq c1 c2*)
discr c1 $\Longrightarrow$ *discr c2* $\Longrightarrow$ *discr* (*Ch ch c1 c2*)
discr c $\Longrightarrow$ *discr* (*While tst c*)
properL cs $\Longrightarrow$ ($\bigwedge$*c*. *c* $\in$ *set cs* $\Longrightarrow$ *discr c*) $\Longrightarrow$ *discr* (*Par cs*)
properL cs $\Longrightarrow$ ($\bigwedge$*c*. *c* $\in$ *set cs* $\Longrightarrow$ *discr c*) $\Longrightarrow$ *discr* (*ParT cs*)
by (*auto intro*!: *discr-Par discr-ParT*)

lemma *siso*:
compatAtm atm $\Longrightarrow$ *siso* (*Atm atm*)
siso c1 $\Longrightarrow$ *siso c2* $\Longrightarrow$ *siso* (*Seq c1 c2*)
compatCh ch $\Longrightarrow$ *siso c1* $\Longrightarrow$ *siso c2* $\Longrightarrow$ *siso* (*Ch ch c1 c2*)
compatTst tst $\Longrightarrow$ *siso c* $\Longrightarrow$ *siso* (*While tst c*)
properL cs $\Longrightarrow$ ($\bigwedge$*c*. *c* $\in$ *set cs* $\Longrightarrow$ *siso c*) $\Longrightarrow$ *siso* (*Par cs*)
properL cs $\Longrightarrow$ ($\bigwedge$*c*. *c* $\in$ *set cs* $\Longrightarrow$ *siso c*) $\Longrightarrow$ *siso* (*ParT cs*)
by (*auto intro*!: *siso-Par siso-ParT*)

lemma *Sbis*:
compatAtm atm $\Longrightarrow$ *Atm atm* $\approx$*s Atm atm*
siso c1 $\Longrightarrow$ *c2* $\approx$*s c2* $\Longrightarrow$ *Seq c1 c2* $\approx$*s Seq c1 c2*
proper c1 $\Longrightarrow$ *proper c2* $\Longrightarrow$ *c1* $\approx$*s c1* $\Longrightarrow$ *discr c2* $\Longrightarrow$ *Seq c1 c2* $\approx$*s Seq c1 c2*
compatCh ch $\Longrightarrow$ *c1* $\approx$*s c1* $\Longrightarrow$ *c2* $\approx$*s c2* $\Longrightarrow$ *Ch ch c1 c2* $\approx$*s Ch ch c1 c2*
properL cs $\Longrightarrow$ ($\bigwedge$*c*. *c* $\in$ *set cs* $\Longrightarrow$ *c* $\approx$*s c*) $\Longrightarrow$ *Par cs* $\approx$*s Par cs*
by (*auto intro*!: *Par-Sbis*)

lemma *ZObis*:
compatAtm atm $\Longrightarrow$ *Atm atm* $\approx$*01 Atm atm*

siso c1 $\Longrightarrow$ *c2* $\approx$*01 c2* $\Longrightarrow$ *Seq c1 c2* $\approx$*01 Seq c1 c2*
proper c1 $\Longrightarrow$ *proper c2* $\Longrightarrow$ *c1* $\approx$*01 c1* $\Longrightarrow$ *discr c2* $\Longrightarrow$ *Seq c1 c2* $\approx$*01 Seq c1 c2*
compatCh ch $\Longrightarrow$ *c1* $\approx$*01 c1* $\Longrightarrow$ *c2* $\approx$*01 c2* $\Longrightarrow$ *Ch ch c1 c2* $\approx$*01 Ch ch c1 c2*
properL cs $\Longrightarrow$ ($\bigwedge$*c*. *c* $\in$ *set cs* $\Longrightarrow$ *c* $\approx$*s c*) $\Longrightarrow$ *ParT cs* $\approx$*01 ParT cs*
by (*auto intro*!: *ParT-ZObis*)

lemma *discr-imp-Sbis*: *proper c* $\Longrightarrow$ *discr c* $\Longrightarrow$ *c* $\approx$*s c*
by *auto*

lemma *siso-imp-Sbis*: *siso c* $\Longrightarrow$ *c* $\approx$*s c*
by *auto*

lemma *Sbis-imp-ZObis*: *c* $\approx$*s c* $\Longrightarrow$ *c* $\approx$*01 c*
by *auto*

fun *SC-discr* **where**
SC-discr Done $\longleftrightarrow$ *True*
| *SC-discr* (*Atm atm*) $\longleftrightarrow$ *presAtm atm*
| *SC-discr* (*Seq c1 c2*) $\longleftrightarrow$ *SC-discr c1* $\wedge$ *SC-discr c2*
| *SC-discr* (*Ch ch c1 c2*) $\longleftrightarrow$ *SC-discr c1* $\wedge$ *SC-discr c2*
| *SC-discr* (*While tst c*) $\longleftrightarrow$ *SC-discr c*
| *SC-discr* (*ParT cs*) $\longleftrightarrow$ ($\forall$ *c*$\in$*set cs*. *SC-discr c*)
| *SC-discr* (*Par cs*) $\longleftrightarrow$ ($\forall$ *c*$\in$*set cs*. *SC-discr c*)

theorem *SC-discr-discr*[*intro*]: *proper c* $\Longrightarrow$ *SC-discr c* $\Longrightarrow$ *discr c*
by (*induct c*) (*auto intro*!: *discr*)

fun *SC-siso* **where**
SC-siso Done $\longleftrightarrow$ *True*
| *SC-siso* (*Atm atm*) $\longleftrightarrow$ *compatAtm atm*
| *SC-siso* (*Seq c1 c2*) $\longleftrightarrow$ *SC-siso c1* $\wedge$ *SC-siso c2*
| *SC-siso* (*Ch ch c1 c2*) $\longleftrightarrow$ *compatCh ch* $\wedge$ *SC-siso c1* $\wedge$ *SC-siso c2*
| *SC-siso* (*While tst c*) $\longleftrightarrow$ *compatTst tst* $\wedge$ *SC-siso c*
| *SC-siso* (*Par cs*) $\longleftrightarrow$ ($\forall$ *c*$\in$*set cs*. *SC-siso c*)
| *SC-siso* (*ParT cs*) $\longleftrightarrow$ ($\forall$ *c*$\in$*set cs*. *SC-siso c*)

theorem *SC-siso-siso*[*intro*]: *proper c* $\Longrightarrow$ *SC-siso c* $\Longrightarrow$ *siso c*
by (*induct c*) (*auto intro*!: *siso*)

fun *SC-Sbis* **where**
SC-Sbis Done $\longleftrightarrow$ *True*
| *SC-Sbis* (*Atm atm*) $\longleftrightarrow$ *compatAtm atm*
| *SC-Sbis* (*Seq c1 c2*) $\longleftrightarrow$ (*SC-siso c1* $\wedge$ *SC-Sbis c2*) $\vee$
(*SC-Sbis c1* $\wedge$ *SC-discr c2*) $\vee$
SC-discr (*Seq c1 c2*) $\vee$ *SC-siso* (*Seq c1 c2*)

```
| SC-Sbis (Ch ch c1 c2) ⟷ (if compatCh ch
                          then SC-Sbis c1 ∧ SC-Sbis c2
                          else (SC-discr (Ch ch c1 c2) ∨ SC-siso (Ch ch c1 c2)))
| SC-Sbis (While tst c) ⟷ SC-discr (While tst c) ∨ SC-siso (While tst c)
| SC-Sbis (Par cs)   ⟷ (∀ c∈set cs. SC-Sbis c)
| SC-Sbis (ParT cs) ⟷ SC-siso (ParT cs) ∨ SC-discr (ParT cs)

theorem SC-siso-SCbis[intro]: SC-siso c ⟹ SC-Sbis c
  by (induct c) auto

theorem SC-discr-SCbis[intro]: SC-discr c ⟹ SC-Sbis c
  by (induct c) auto

declare SC-siso.simps[simp del]

declare SC-discr.simps[simp del]

theorem SC-Sbis-Sbis[intro]: proper c ⟹ SC-Sbis c ⟹ c ≈s c
  by (induct c)
     (auto intro: Sbis discr-imp-Sbis siso-imp-Sbis
           split: if-split-asm)

fun SC-ZObis where
  SC-ZObis Done          ⟷ True
| SC-ZObis (Atm atm)      ⟷ compatAtm atm
| SC-ZObis (Seq c1 c2)    ⟷ (SC-siso c1 ∧ SC-ZObis c2) ∨
                          (SC-ZObis c1 ∧ SC-discr c2) ∨
                          SC-Sbis (Seq c1 c2)
| SC-ZObis (Ch ch c1 c2) ⟷ (if compatCh ch
                          then SC-ZObis c1 ∧ SC-ZObis c2
                          else SC-Sbis (Ch ch c1 c2))
| SC-ZObis (While tst c) ⟷ SC-Sbis (While tst c)
| SC-ZObis (Par cs)   ⟷ SC-Sbis (Par cs)
| SC-ZObis (ParT cs) ⟷ (∀ c∈set cs. SC-Sbis c)

theorem SC-Sbis-SC-ZObis[intro]: SC-Sbis c ⟹ SC-ZObis c
  by (induct c) (auto simp: SC-siso.simps SC-discr.simps)

declare SC-Sbis.simps[simp del]

theorem SC-ZObis-ZObis: proper c ⟹ SC-ZObis c ⟹ c ≈01 c
  apply (induct c)
  apply (auto intro: Sbis-imp-ZObis ZObis split: if-split-asm)
  apply (auto intro!: ZObis(5))
  done

end

end
```

6 Concrete setting

theory *Concrete*
imports *Syntactic-Criteria*
begin

datatype *level* = *Lo* | *Hi*

lemma [*simp*]: $\bigwedge$ *l*. *l* ≠ *Hi* ⟷ *l* = *Lo* **and**
[*simp*]: $\bigwedge$ *l*. *Hi* ≠ *l* ⟷ *Lo* = *l* **and**
[*simp*]: $\bigwedge$ *l*. *l* ≠ *Lo* ⟷ *l* = *Hi* **and**
[*simp*]: $\bigwedge$ *l*. *Lo* ≠ *l* ⟷ *Hi* = *l*
by (*metis level.exhaust level.simps*(*2*))+

lemma [*dest*]: $\bigwedge$ *l A*. ⟦*l* ∈ *A*; *Lo* ∉ *A*⟧ ⟹ *l* = *Hi* **and**
[*dest*]: $\bigwedge$ *l A*. ⟦*l* ∈ *A*; *Hi* ∉ *A*⟧ ⟹ *l* = *Lo*
by (*metis level.exhaust*)+

declare *level.split*[*split*]

instantiation *level* :: *complete-lattice*
begin
definition *top-level*: *top* ≡ *Hi*
definition *bot-level*: *bot* ≡ *Lo*
definition *inf-level*: *inf l1 l2* ≡ *if Lo* ∈ {*l1*,*l2*} *then Lo else Hi*
definition *sup-level*: *sup l1 l2* ≡ *if Hi* ∈ {*l1*,*l2*} *then Hi else Lo*
definition *less-eq-level*: *less-eq l1 l2* ≡ (*l1* = *Lo* ∨ *l2* = *Hi*)
definition *less-level*: *less l1 l2* ≡ *l1* = *Lo* ∧ *l2* = *Hi*
definition *Inf-level*: *Inf L* ≡ *if Lo* ∈ *L then Lo else Hi*
definition *Sup-level*: *Sup L* ≡ *if Hi* ∈ *L then Hi else Lo*
instance
proof qed (*auto simp*: *top-level bot-level inf-level sup-level*
less-eq-level less-level Inf-level Sup-level)
end

lemma *sup-eq-Lo*[*simp*]: *sup a b* = *Lo* ⟷ *a* = *Lo* ∧ *b* = *Lo*
by (*auto simp*: *sup-level*)

datatype *var* = *h* | *h′* | *l* | *l′*
datatype *exp* = *Ct nat* | *Var var* | *Plus exp exp* | *Minus exp exp*
datatype *test* = *Tr* | *Eq exp exp* | *Gt exp exp* | *Non test*
datatype *atom* = *Assign var exp*
type-synonym *choice* = *real* + *test*
type-synonym *state* = *var* ⇒ *nat*

syntax

```
 -assign :: 'a ⇒ 'a ⇒ 'a  (‹- ::= -› [1000, 61] 61)

syntax-consts
  -assign == Assign

translations
  x ::= expr == CONST Atm (CONST Assign x expr)

primrec sec where
  sec h  = Hi
| sec h' = Hi
| sec l  = Lo
| sec l' = Lo

fun eval where
 eval (Ct n) s = n
|eval (Var x) s = s x
|eval (Plus e1 e2) s = eval e1 s + eval e2 s
|eval (Minus e1 e2) s = eval e1 s − eval e2 s

fun tval where
 tval Tr s = True
|tval (Eq e1 e2) s = (eval e1 s = eval e2 s)
|tval (Gt e1 e2) s = (eval e1 s > eval e2 s)
|tval (Non e) s = (¬ tval e s)

fun aval where
aval (Assign x e) s = (s (x := eval e s))

fun cval where
 cval (Inl p) s = min 1 (max 0 p)
|cval (Inr tst) s = (if tval tst s then 1 else 0)

definition indis :: (state ∗ state) setwhere
indis ≡ {(s,t). ALL x. sec x = Lo ⟶ s x = t x}

interpretation Example-PL: PL-Indis aval tval cval indis
proof
  fix ch :: choice and  s show 0 ≤ cval ch s ∧ cval ch s ≤ 1
    by (cases ch) auto
next
  show equiv UNIV indis
    unfolding refl-on-def sym-def trans-def equiv-def indis-def by auto
qed


fun exprSec where
 exprSec (Ct n) = Lo
|exprSec (Var x) = sec x
```

|*exprSec* (*Plus e1 e2*) = *sup* (*exprSec e1*) (*exprSec e2*)
|*exprSec* (*Minus e1 e2*) = *sup* (*exprSec e1*) (*exprSec e2*)

fun *tstSec* **where**
tstSec Tr = *Lo*
|*tstSec* (*Eq e1 e2*) = *sup* (*exprSec e1*) (*exprSec e2*)
|*tstSec* (*Gt e1 e2*) = *sup* (*exprSec e1*) (*exprSec e2*)
|*tstSec* (*Non e*) = *tstSec e*

lemma *exprSec-Lo-eval-eq*: *exprSec expr* = *Lo* ⟹ (*s*, *t*) ∈ *indis* ⟹ *eval expr s* = *eval expr t*
by (*induct expr*) (*auto simp*: *indis-def*)

lemma *compatAtmSyntactic*[*simp*]: *exprSec expr* = *Lo* ∨ *sec v* = *Hi* ⟹ *Example-PL.compatAtm* (*Assign v expr*)
unfolding *Example-PL.compatAtm-def*
by (*induct expr*)
(*auto simp*: *indis-def intro*!: *arg-cong2*[**where** *f*=(+)] *arg-cong2*[**where** *f*=(−)] *exprSec-Lo-eval-eq*)

lemma *presAtmSyntactic*[*simp*]: *sec v* = *Hi* ⟹ *Example-PL.presAtm* (*Assign v expr*)
unfolding *Example-PL.presAtm-def* **by** (*simp add*: *indis-def*)

lemma *compatTstSyntactic*[*simp*]: *tstSec tst* = *Lo* ⟹ *Example-PL.compatTst tst*
unfolding *Example-PL.compatTst-def*
by (*induct tst*)
(*simp-all*, *safe del*: *iffI*
intro!: *arg-cong2*[**where** *f*=(=)] *arg-cong2*[**where** *f*=(<) :: *nat* ⇒ *nat* ⇒ *bool*] *exprSec-Lo-eval-eq*)

lemma *compatPrchSyntactic*[*simp*]: *Example-PL.compatCh* (*Inl p*)
unfolding *Example-PL.compatCh-def* **by** *auto*

lemma *compatIfchSyntactic*[*simp*]: *Example-PL.compatCh* (*Inr tst*) ⟷ *Example-PL.compatTst tst*
unfolding *Example-PL.compatCh-def Example-PL.compatTst-def* **by** *auto*

abbreviation *Ch-half* (‹*Ch*½›) **where** *Ch*½ ≡ *Ch* (*Inl* (*1*/*2*))
abbreviation *If* **where** *If tst* ≡ *Ch* (*Inr tst*)

abbreviation *siso c* ≡ *Example-PL.siso c*
abbreviation *discr c* ≡ *Example-PL.discr c*
abbreviation *Sbis-abbrev* (**infix** ‹≈*s*› *55*) **where** *c1* ≈*s c2* ≡ (*c1*,*c2*) ∈ *Example-PL.Sbis*
abbreviation *ZObis-abbrev* (**infix** ‹≈*01*› *55*) **where** *c1* ≈*01 c2* ≡ (*c1*,*c2*) ∈

Example-PL.ZObis

abbreviation *SC-siso c* ≡ *Example-PL.SC-siso c*
abbreviation *SC-discr c* ≡ *Example-PL.SC-discr c*
abbreviation *SC-Sbis c* ≡ *Example-PL.SC-Sbis c*
abbreviation *SC-ZObis c* ≡ *Example-PL.SC-ZObis c*

lemma *SC-discr* (*h* ::= *Ct 0*)
by (*simp add*: *Example-PL.SC-discr.simps*)

6.1 The secure programs from the paper's Example 3

definition [*simp*]: *d0* =
h′ ::= *Ct 0* ;;
While (*Gt* (*Var h*) (*Ct 0*))
(*Ch½* (*h* ::= *Ct 0*)
(*h′* ::= *Plus* (*Var h′*) (*Ct 1*)))

definition [*simp*]: *d1* =
While (*Gt* (*Var h*) (*Ct 0*))
(*Ch½* (*h* ::= *Minus* (*Var h*) (*Ct 1*))
(*h* ::= *Plus* (*Var h*) (*Ct 1*)))

definition [*simp*]: *d2* =
If (*Eq* (*Var l*) (*Ct 0*))
(*l′* ::= *Ct 1*)
d0

definition [*simp*]: *d3* =
h ::= *Ct 5* ;;
ParT [*d0*, (*l* ::= *Ct 1*)]

theorem *SC-discr d0*
SC-discr d1
SC-Sbis d2
SC-ZObis d2
by (*auto simp*: *Example-PL.SC-discr.simps Example-PL.SC-Sbis.simps Example-PL.SC-ZObis.simps*)

theorem *discr d0*
discr d1
d2 ≈*s d2*
d3 ≈*01 d3*
by (*auto intro*!: *compatAtmSyntactic*
Example-PL.ZObis Example-PL.proper-intros
Example-PL.Atm-Sbis)

end

References

[1] A. Popescu, J. Hölzl, and T. Nipkow. Formalizing probabilistic noninterference. In G. Gonthier and M. Norrish, editors, *Certified Programs and Proofs (CPP 2013)*, volume 8307 of *LNCS*, pages 259–275. Springer, 2013.

[2] A. Popescu, J. Hölzl, and T. Nipkow. Noninterfering schedulers. In R. Heckel and S. Milius, editors, *Algebra and Coalgebra in Computer Science (CALCO 2013)*, volume 8089 of *LNCS*, pages 236–252. Springer, 2013.