Restriction_Spaces: a Fixed-Point Theory

Benoît Ballenghien Benjamin Puyobro Burkhart Wolff

February 6, 2026

Abstract

Fixed-point constructions are fundamental to defining recursive and co-recursive functions. However, a general axiom $Yf = f(Yf)$ leads to inconsistency, and definitions must therefore be based on theories guaranteeing existence under suitable conditions. In `Isabelle/HOL`, such constructions are typically based on sets, well-founded orders or domain-theoretic models such as for example `HOLCF`. In this submission we introduce `Restriction_Spaces`, a formalization of spaces equipped with a so-called restriction, denoted by $\downarrow$, satifying three properties:

$$x \downarrow 0 = y \downarrow 0$$
$$x \downarrow n \downarrow m = x \downarrow \min\ n\ m$$
$$x \neq y \Longrightarrow \exists n.\ x \downarrow n \neq y \downarrow n$$

They turn out to be cartesian closed and admit natural notions of constructiveness and completeness, enabling the definition of a fixed-point operator under verifiable side-conditions. This is achieved in our entry, from topological definitions to induction principles. Additionally, we configure the simplifier so that it can automatically solve both constructiveness and admissibility subgoals, as long as users write higher-order rules for their operators. Since our implementation relies on axiomatic type classes, the resulting library is a fully abstract, flexible and reusable framework.

Contents

1 Locales factorizing the proof Work

named-theorems *restriction-shift-simpset*
named-theorems *restriction-shift-introset* — Useful for future automation.

In order to factorize the proof work, we first work with locales and then with classes.

1.1 Basic Notions for Restriction

locale *Restriction* =
  fixes *restriction* :: ‹['a, *nat*] ⇒ 'a› (**infixl** ‹↓› *60*)
    and *relation* :: ‹['a, 'a] ⇒ *bool*› (**infixl** ‹$\lesssim$› *50*)
  assumes *restriction-restriction* [*simp*] : ‹$x \downarrow n \downarrow m = x \downarrow min\ n\ m$›
begin

abbreviation *restriction-related-set* :: ‹'*a* ⇒ 'a ⇒ *nat set*›
  where ‹*restriction-related-set x y* ≡ {*n*. *x* ↓ *n* ≲≈ *y* ↓ *n*}›

abbreviation *restriction-not-related-set* :: ‹'*a* ⇒ 'a ⇒ *nat set*›
  where ‹*restriction-not-related-set x y* ≡ {*n*. ¬ *x* ↓ *n* ≲≈ *y* ↓ *n*}›

lemma *restriction-related-set-Un-restriction-not-related-set* :
  ‹*restriction-related-set x y* ∪ *restriction-not-related-set x y* = *UNIV*›
⟨*proof*⟩

lemma *disjoint-restriction-related-set-restriction-not-related-set* :
  ‹*restriction-related-set x y* ∩ *restriction-not-related-set x y* = {}›
⟨*proof*⟩

lemma ‹*bdd-below* (*restriction-related-set x y*)› ⟨*proof*⟩

lemma ‹*bdd-below* (*restriction-not-related-set x y*)› ⟨*proof*⟩

end

locale *PreorderRestrictionSpace* = *Restriction* +
  assumes *restriction-0-related* [*simp*] : ‹*x* ↓ *0* ≲≈ *y* ↓ *0*›
    and *mono-restriction-related* : ‹ *x* ≲≈ *y* ⟹ *x* ↓ *n* ≲≈ *y* ↓ *n*›
    and *ex-not-restriction-related* : ‹¬ *x* ≲≈ *y* ⟹ ∃ *n*. ¬ *x* ↓ *n* ≲≈ *y* ↓ *n*›
    and *related-trans* : ‹*x* ≲≈ *y* ⟹ *y* ≲≈ *z* ⟹ *x* ≲≈ *z*›
begin

lemma *exists-restriction-related* [*simp*] : ‹∃ *n*. *x* ↓ *n* ≲≈ *y* ↓ *n*›
  ⟨*proof*⟩

lemma *all-restriction-related-iff-related* : ‹(∀ *n*. *x* ↓ *n* ≲≈ *y* ↓ *n*) ⟷ *x* ≲≈ *y*›
  ⟨*proof*⟩

lemma *restriction-related-le* : ‹*x* ↓ *n* ≲≈ *y* ↓ *n*› **if** ‹*n* ≤ *m*› **and** ‹*x* ↓ *m* ≲≈ *y* ↓ *m*›
⟨*proof*⟩

corollary *restriction-related-pred* : ‹*x* ↓ *Suc n* ≲≈ *y* ↓ *Suc n* ⟹ *x* ↓ *n* ≲≈ *y* ↓ *n*›
  ⟨*proof*⟩

lemma *all-ge-restriction-related-iff-related* : ‹(∀ *n*≥*m*. *x* ↓ *n* ≲≈ *y* ↓ *n*) ⟷ *x* ≲≈ *y*›
  ⟨*proof*⟩

lemma *take-lemma-restriction* : ‹$x \lessapprox y$›
if ‹$\bigwedge n.\ [\![\bigwedge k.\ k \leq n \Longrightarrow x \downarrow k \lessapprox y \downarrow k]\!] \Longrightarrow x \downarrow Suc\ n \lessapprox y \downarrow Suc\ n$›
⟨*proof*⟩

lemma *ex-not-restriction-related-optimized* :
‹$\exists!n.\ \neg\ x \downarrow Suc\ n \lessapprox y \downarrow Suc\ n \wedge (\forall\, m{\leq}n.\ x \downarrow m \lessapprox y \downarrow m)$› **if** ‹$\neg\ x \lessapprox y$›
⟨*proof*⟩

lemma *nonempty-restriction-related-set* : ‹*restriction-related-set x y* $\neq \{\}$›
⟨*proof*⟩

lemma *non-UNIV-restriction-not-related-set* : ‹*restriction-not-related-set x y* $\neq$ *UNIV*›
⟨*proof*⟩

lemma *UNIV-restriction-related-set-iff* : ‹*restriction-related-set x y* = *UNIV* $\longleftrightarrow$ $x \lessapprox y$›
⟨*proof*⟩

lemma *empty-restriction-not-related-set-iff*: ‹*restriction-not-related-set x y* $= \{\} \longleftrightarrow x \lessapprox y$›
⟨*proof*⟩

lemma *finite-restriction-related-set-iff* :
‹*finite* (*restriction-related-set x y*) $\longleftrightarrow \neg\ x \lessapprox y$›
⟨*proof*⟩

lemma *infinite-restriction-not-related-set-iff* :
‹*infinite* (*restriction-not-related-set x y*) $\longleftrightarrow \neg\ x \lessapprox y$›
⟨*proof*⟩

lemma *bdd-above-restriction-related-set-iff* :
‹*bdd-above* (*restriction-related-set x y*) $\longleftrightarrow \neg\ x \lessapprox y$›
⟨*proof*⟩

context fixes *x y* **assumes** ‹$\neg\ x \lessapprox y$› **begin**

lemma *Sup-in-restriction-related-set* :
‹*Sup* (*restriction-related-set x y*) $\in$ *restriction-related-set x y*›

⟨*proof*⟩

lemma *Inf-in-restriction-not-related-set* :
‹*Inf* (*restriction-not-related-set x y*) ∈ *restriction-not-related-set x y*›
⟨*proof*⟩

lemma *Inf-restriction-not-related-set-eq-Suc-Sup-restriction-related-set* :
‹*Inf* (*restriction-not-related-set x y*) = *Suc* (*Sup* (*restriction-related-set x y*))›
⟨*proof*⟩

end

lemma *restriction-related-set-is-atMost* :
‹*restriction-related-set x y* =
(*if x* $\lessapprox$ *y then UNIV else* {..*Sup* (*restriction-related-set x y*)})›
⟨*proof*⟩

lemma *restriction-not-related-set-is-atLeast* :
‹*restriction-not-related-set x y* =
(*if x* $\lessapprox$ *y then* {} *else* {*Inf* (*restriction-not-related-set x y*)..})›
⟨*proof*⟩

end

1.2 Restriction shift Maps

locale *Restriction-2-PreorderRestrictionSpace* =
R1 : *Restriction* ‹($\downarrow_1$)› ‹($\lessapprox_1$)› +
PRS2 : *PreorderRestrictionSpace* ‹($\downarrow_2$)› ‹($\lessapprox_2$)›
for *restriction*$_1$ :: ‹'*a* ⇒ *nat* ⇒ '*a*› (**infixl** ‹$\downarrow_1$› *60*)
and *relation*$_1$ :: ‹'*a* ⇒ '*a* ⇒ *bool*› (**infixl** ‹$\lessapprox_1$› *50*)
and *restriction*$_2$ :: ‹'*b* ⇒ *nat* ⇒ '*b*› (**infixl** ‹$\downarrow_2$› *60*)
and *relation*$_2$ :: ‹'*b* ⇒ '*b* ⇒ *bool*› (**infixl** ‹$\lessapprox_2$› *50*)
begin

1.2.1 Definition

This notion is a generalization of constructive map and non-destructive map.

definition *restriction-shift-on* :: ‹['*a* ⇒ '*b*, *int*, '*a set*] ⇒ *bool*›
where ‹*restriction-shift-on f k A* ≡
∀ *x*∈*A*. ∀ *y*∈*A*. ∀ *n*. *x* $\downarrow_1$ *n* $\lessapprox_1$ *y* $\downarrow_1$ *n* ⟶ *f x* $\downarrow_2$ *nat* (*int n* + *k*) $\lessapprox_2$ *f y* $\downarrow_2$ *nat* (*int n* + *k*)›

definition *restriction-shift* :: ‹['*a* ⇒ '*b*, *int*] ⇒ *bool*›

where ‹*restriction-shift f k* ≡ *restriction-shift-on f k UNIV*›

lemma *restriction-shift-onI* :
‹(⋀*x y n*. ⟦*x* ∈ *A*; *y* ∈ *A*; ¬ *f x* ≲≈$_2$ *f y*; *x* ↓$_1$ *n* ≲≈$_1$ *y* ↓$_1$ *n*⟧ ⟹
f x ↓$_2$ *nat* (*int n* + *k*) ≲≈$_2$ *f y* ↓$_2$ *nat* (*int n* + *k*))
⟹ *restriction-shift-on f k A*›
⟨*proof*⟩

corollary *restriction-shiftI* :
‹(⋀*x y n*. ⟦¬ *f x* ≲≈$_2$ *f y*; *x* ↓$_1$ *n* ≲≈$_1$ *y* ↓$_1$ *n*⟧ ⟹
f x ↓$_2$ *nat* (*int n* + *k*) ≲≈$_2$ *f y* ↓$_2$ *nat* (*int n* + *k*))
⟹ *restriction-shift f k*›
⟨*proof*⟩

lemma *restriction-shift-onD* :
‹⟦*restriction-shift-on f k A*; *x* ∈ *A*; *y* ∈ *A*; *x* ↓$_1$ *n* ≲≈$_1$ *y* ↓$_1$ *n*⟧
⟹ *f x* ↓$_2$ *nat* (*int n* + *k*) ≲≈$_2$ *f y* ↓$_2$ *nat* (*int n* + *k*)›
⟨*proof*⟩

lemma *restriction-shiftD* :
‹⟦*restriction-shift f k*; *x* ↓$_1$ *n* ≲≈$_1$ *y* ↓$_1$ *n*⟧ ⟹ *f x* ↓$_2$ *nat* (*int n* + *k*)
≲≈$_2$ *f y* ↓$_2$ *nat* (*int n* + *k*)›
⟨*proof*⟩

lemma *restriction-shift-on-subset* :
‹*restriction-shift-on f k B* ⟹ *A* ⊆ *B* ⟹ *restriction-shift-on f k A*›
⟨*proof*⟩

lemma *restriction-shift-imp-restriction-shift-on* [*restriction-shift-simpset*]
:
‹*restriction-shift f k* ⟹ *restriction-shift-on f k A*›
⟨*proof*⟩

lemma *restriction-shift-on-imp-restriction-shift-on-le* [*restriction-shift-simpset*]
:
‹*restriction-shift-on f l A*› **if** ‹*l* ≤ *k*› **and** ‹*restriction-shift-on f k A*›
⟨*proof*⟩

corollary *restriction-shift-imp-restriction-shift-le* [*restriction-shift-simpset*]
:
‹*l* ≤ *k* ⟹ *restriction-shift f k* ⟹ *restriction-shift f l*›
⟨*proof*⟩

lemma *restriction-shift-on-if-then-else* [*restriction-shift-simpset*, *restriction-shift-introset*] :
‹⟦⋀x. P x ⟹ restriction-shift-on (f x) k A;
⋀x. ¬ P x ⟹ restriction-shift-on (g x) k A⟧ ⟹
restriction-shift-on (λy. if P x then f x y else g x y) k A›
⟨proof⟩

corollary *restriction-shift-if-then-else* [*restriction-shift-simpset*, *restriction-shift-introset*] :
‹⟦⋀x. P x ⟹ restriction-shift (f x) k;
⋀x. ¬ P x ⟹ restriction-shift (g x) k⟧ ⟹
restriction-shift (λy. if P x then f x y else g x y) k›
⟨proof⟩

1.2.2 Three particular Cases

The shift is most often equal to *0*, *1* or $-$ *1*. We provide extra support in these three cases.

Non-too-destructive Map **definition** *non-too-destructive-on* ::
‹['a ⇒ 'b, 'a set] ⇒ bool›
where ‹non-too-destructive-on f A ≡ restriction-shift-on f (− 1) A›

definition *non-too-destructive* :: ‹['a ⇒ 'b] ⇒ bool›
where ‹non-too-destructive f ≡ non-too-destructive-on f UNIV›

lemma *non-too-destructive-onI* :
‹non-too-destructive-on f A›
if ‹⋀n x y. ⟦x ∈ A; y ∈ A; ¬ f x ≲≈2 f y; x ↓1 Suc n ≲≈1 y ↓1 Suc n⟧
⟹ f x ↓2 n ≲≈2 f y ↓2 n›
⟨proof⟩

lemma *non-too-destructiveI* :
‹⟦⋀n x y. ⟦¬ f x ≲≈2 f y; x ↓1 Suc n ≲≈1 y ↓1 Suc n⟧ ⟹ f x ↓2 n ≲≈2 f y ↓2 n⟧
⟹ non-too-destructive f›
⟨proof⟩

lemma *non-too-destructive-onD* :
‹⟦non-too-destructive-on f A; x ∈ A; y ∈ A; x ↓1 Suc n ≲≈1 y ↓1 Suc n⟧ ⟹ f x ↓2 n ≲≈2 f y ↓2 n›
⟨proof⟩

lemma *non-too-destructiveD* :
‹⟦non-too-destructive f; x ↓1 Suc n ≲≈1 y ↓1 Suc n⟧ ⟹ f x ↓2 n ≲≈2 f y ↓2 n›
⟨proof⟩

lemma *non-too-destructive-on-subset* :
‹*non-too-destructive-on f B* $\Longrightarrow$ $A \subseteq B$ $\Longrightarrow$ *non-too-destructive-on f A*›
⟨*proof*⟩

lemma *non-too-destructive-imp-non-too-destructive-on* [*restriction-shift-simpset*] :
‹*non-too-destructive f* $\Longrightarrow$ *non-too-destructive-on f A*›
⟨*proof*⟩

corollary *non-too-destructive-on-if-then-else* [*restriction-shift-simpset*, *restriction-shift-introset*]:
‹⟦$\bigwedge x.$ *P x* $\Longrightarrow$ *non-too-destructive-on* (*f x*) *A*; $\bigwedge x.$ ¬ *P x* $\Longrightarrow$ *non-too-destructive-on* (*g x*) *A*⟧
$\Longrightarrow$ *non-too-destructive-on* ($\lambda y.$ *if P x then f x y else g x y*) *A*›
and *non-too-destructive-if-then-else* [*restriction-shift-simpset*, *restriction-shift-introset*] :
‹⟦$\bigwedge x.$ *P x* $\Longrightarrow$ *non-too-destructive* (*f x*); $\bigwedge x.$ ¬ *P x* $\Longrightarrow$ *non-too-destructive* (*g x*)⟧
$\Longrightarrow$ *non-too-destructive* ($\lambda y.$ *if P x then f x y else g x y*)›
⟨*proof*⟩

Non-destructive Map **definition** *non-destructive-on* :: ‹['*a* ⇒ '*b*, '*a set*] ⇒ *bool*›
where ‹*non-destructive-on f A* ≡ *restriction-shift-on f 0 A*›

definition *non-destructive* :: ‹['*a* ⇒ '*b*] ⇒ *bool*›
where ‹*non-destructive f* ≡ *non-destructive-on f UNIV*›

lemma *non-destructive-onI* :
‹⟦$\bigwedge n\ x\ y.$ ⟦$n \neq 0$; $x \in A$; $y \in A$; ¬ *f x* $\lesssim_2$ *f y*; $x \downarrow_1 n \lesssim_1 y \downarrow_1 n$⟧ $\Longrightarrow$ $f\ x \downarrow_2 n \lesssim_2 f\ y \downarrow_2 n$⟧
$\Longrightarrow$ *non-destructive-on f A*›
⟨*proof*⟩

lemma *non-destructiveI* :
‹⟦$\bigwedge n\ x\ y.$ ⟦$n \neq 0$; ¬ $f\ x \lesssim_2 f\ y$; $x \downarrow_1 n \lesssim_1 y \downarrow_1 n$⟧ $\Longrightarrow$ $f\ x \downarrow_2 n \lesssim_2 f\ y \downarrow_2 n$⟧
$\Longrightarrow$ *non-destructive f*› ⟨*proof*⟩

lemma *non-destructive-onD* :
‹⟦*non-destructive-on f A*; $x \in A$; $y \in A$; ¬ $f\ x \lesssim_2 f\ y$; $x \downarrow_1 n \lesssim_1 y \downarrow_1 n$⟧ $\Longrightarrow$ $f\ x \downarrow_2 n \lesssim_2 f\ y \downarrow_2 n$›
⟨*proof*⟩

lemma *non-destructiveD* : ‹⟦*non-destructive f*; $x \downarrow_1 n \lesssim_1 y \downarrow_1 n$⟧ $\Longrightarrow$ $f\ x \downarrow_2 n \lesssim_2 f\ y \downarrow_2 n$›

⟨*proof*⟩

lemma *non-destructive-on-subset* :
‹*non-destructive-on f B* ⟹ *A* ⊆ *B* ⟹ *non-destructive-on f A*›
⟨*proof*⟩

lemma *non-destructive-imp-non-destructive-on* [*restriction-shift-simpset*]
:
‹*non-destructive f* ⟹ *non-destructive-on f A*›
⟨*proof*⟩

lemma *non-destructive-on-imp-non-too-destructive-on* [*restriction-shift-simpset*]
:
‹*non-destructive-on f A* ⟹ *non-too-destructive-on f A*›
⟨*proof*⟩

corollary *non-destructive-imp-non-too-destructive* [*restriction-shift-simpset*]
:
‹*non-destructive f* ⟹ *non-too-destructive f*›
⟨*proof*⟩

corollary *non-destructive-on-if-then-else* [*restriction-shift-simpset*, *restriction-shift-introset*] :
‹⟦⋀*x*. *P x* ⟹ *non-destructive-on* (*f x*) *A*; ⋀*x*. ¬ *P x* ⟹ *non-destructive-on* (*g x*) *A*⟧
⟹ *non-destructive-on* (λ*y*. *if P x then f x y else g x y*) *A*›
and *non-destructive-if-then-else* [*restriction-shift-simpset*, *restriction-shift-introset*] :
‹⟦⋀*x*. *P x* ⟹ *non-destructive* (*f x*); ⋀*x*. ¬ *P x* ⟹ *non-destructive* (*g x*)⟧
⟹ *non-destructive* (λ*y*. *if P x then f x y else g x y*)›
⟨*proof*⟩

Constructive Map **definition** *constructive-on* :: ‹['*a* ⇒ '*b*, '*a set*] ⇒ *bool*›
where ‹*constructive-on f A* ≡ *restriction-shift-on f 1 A*›

definition *constructive* :: ‹['*a* ⇒ '*b*] ⇒ *bool*›
where ‹*constructive f* ≡ *constructive-on f UNIV*›

lemma *constructive-onI* :
‹⟦⋀*n x y*. ⟦*x* ∈ *A*; *y* ∈ *A*; ¬ *f x* ≲₂ *f y*; *x* ↓₁ *n* ≲₁ *y* ↓₁ *n*⟧ ⟹ *f x* ↓₂ *Suc n* ≲₂ *f y* ↓₂ *Suc n*⟧
⟹ *constructive-on f A*›
⟨*proof*⟩

lemma *constructiveI* :

‹⟦⋀n x y. ⟦¬ f x ≲≈2 f y; x ↓1 n ≲≈1 y ↓1 n⟧ ⟹ f x ↓2 Suc n ≲≈2 f y ↓2 Suc n⟧
⟹ constructive f› ⟨proof⟩

lemma *constructive-onD* :
‹⟦constructive-on f A; x ∈ A; y ∈ A; x ↓1 n ≲≈1 y ↓1 n⟧ ⟹ f x ↓2 Suc n ≲≈2 f y ↓2 Suc n›
⟨proof⟩

lemma *constructiveD* : ‹⟦constructive f; x ↓1 n ≲≈1 y ↓1 n⟧ ⟹ f x ↓2 Suc n ≲≈2 f y ↓2 Suc n›
⟨proof⟩

lemma *constructive-on-subset* :
‹constructive-on f B ⟹ A ⊆ B ⟹ constructive-on f A›
⟨proof⟩

lemma *constructive-imp-constructive-on* [*restriction-shift-simpset*] :
‹constructive f ⟹ constructive-on f A›
⟨proof⟩

lemma *constructive-on-imp-non-destructive-on* [*restriction-shift-simpset*]
:
‹constructive-on f A ⟹ non-destructive-on f A›
⟨proof⟩

corollary *constructive-imp-non-destructive* [*restriction-shift-simpset*]
:
‹constructive f ⟹ non-destructive f›
⟨proof⟩

corollary *constructive-on-if-then-else* [*restriction-shift-simpset*, *restriction-shift-introset*] :
‹⟦⋀x. P x ⟹ constructive-on (f x) A; ⋀x. ¬ P x ⟹ constructive-on (g x) A⟧
⟹ constructive-on (λy. if P x then f x y else g x y) A›
and *constructive-if-then-else* [*restriction-shift-simpset*, *restriction-shift-introset*]
:
‹⟦⋀x. P x ⟹ constructive (f x); ⋀x. ¬ P x ⟹ constructive (g x)⟧
⟹ constructive (λy. if P x then f x y else g x y)›
⟨proof⟩

end

1.2.3 Properties

locale *PreorderRestrictionSpace-2-PreorderRestrictionSpace* =
  PRS1 : *PreorderRestrictionSpace* ‹$(\downarrow_1)$› ‹$(\lessapprox_1)$› +
  PRS2 : *PreorderRestrictionSpace* ‹$(\downarrow_2)$› ‹$(\lessapprox_2)$›
  for *restriction*$_1$ :: ‹$'a \Rightarrow nat \Rightarrow 'a$› (**infixl** ‹$\downarrow_1$› *60*)
    and *relation*$_1$ :: ‹$'a \Rightarrow 'a \Rightarrow bool$› (**infixl** ‹$\lessapprox_1$› *50*)
    and *restriction*$_2$ :: ‹$'b \Rightarrow nat \Rightarrow 'b$› (**infixl** ‹$\downarrow_2$› *60*)
    and *relation*$_2$ :: ‹$'b \Rightarrow 'b \Rightarrow bool$› (**infixl** ‹$\lessapprox_2$› *50*)
begin

sublocale *Restriction-2-PreorderRestrictionSpace* ⟨*proof*⟩

lemma *restriction-shift-on-restriction-restriction* :
  ‹$f\ (x \downarrow_1 n) \downarrow_2 nat\ (int\ n + k) \lessapprox_2 f\ x \downarrow_2 nat\ (int\ n + k)$›
  if ‹*restriction-shift-on f k A*› ‹$x \downarrow_1 n \in A$› ‹$x \in A$› ‹$x \downarrow_1 n \lessapprox_1 x \downarrow_1 n$›
    — the last assumption is trivial if $(\lessapprox_1)$ is reflexive
  ⟨*proof*⟩

corollary *restriction-shift-restriction-restriction* :
  ‹$f\ (x \downarrow_1 n) \downarrow_2 nat\ (int\ n + k) \lessapprox_2 f\ x \downarrow_2 nat\ (int\ n + k)$›
  if ‹*restriction-shift f k*› **and** ‹$x \downarrow_1 n \lessapprox_1 x \downarrow_1 n$›
  ⟨*proof*⟩

corollary *constructive-on-restriction-restriction* :
  ‹$[\![$*constructive-on f A*; $x \downarrow_1 n \in A$; $x \in A$; $x \downarrow_1 n \lessapprox_1 x \downarrow_1 n$$]\!]$
   $\Longrightarrow f\ (x \downarrow_1 n) \downarrow_2 Suc\ n \lessapprox_2 f\ x \downarrow_2 Suc\ n$›
  ⟨*proof*⟩

corollary *constructive-restriction-restriction* :
  ‹*constructive f* $\Longrightarrow x \downarrow_1 n \lessapprox_1 x \downarrow_1 n \Longrightarrow f\ (x \downarrow_1 n) \downarrow_2 Suc\ n \lessapprox_2 f\ x \downarrow_2 Suc\ n$›
  ⟨*proof*⟩

corollary *non-destructive-on-restriction-restriction* :
  ‹$[\![$*non-destructive-on f A*; $x \downarrow_1 n \in A$; $x \in A$; $x \downarrow_1 n \lessapprox_1 x \downarrow_1 n$$]\!]$
   $\Longrightarrow f\ (x \downarrow_1 n) \downarrow_2 n \lessapprox_2 f\ x \downarrow_2 n$›
  ⟨*proof*⟩

corollary *non-destructive-restriction-restriction* :
  ‹*non-destructive f* $\Longrightarrow x \downarrow_1 n \lessapprox_1 x \downarrow_1 n \Longrightarrow f\ (x \downarrow_1 n) \downarrow_2 n \lessapprox_2 f\ x \downarrow_2 n$›
  ⟨*proof*⟩

corollary *non-too-destructive-on-restriction-restriction* :
  ‹$[\![$*non-too-destructive-on f A*; $x \downarrow_1 Suc\ n \in A$; $x \in A$; $x \downarrow_1 Suc\ n \lessapprox_1$

$x \downarrow_1 Suc\ n]\!]$
$\Longrightarrow f\ (x \downarrow_1 Suc\ n) \downarrow_2 n \lessapprox_2 f\ x \downarrow_2 n\rangle$
⟨*proof*⟩

corollary *non-too-destructive-restriction-restriction* :
$\langle$*non-too-destructive* $f \Longrightarrow x \downarrow_1 Suc\ n \lessapprox_1 x \downarrow_1 Suc\ n \Longrightarrow f\ (x \downarrow_1 Suc\ n) \downarrow_2 n \lessapprox_2 f\ x \downarrow_2 n\rangle$
⟨*proof*⟩

end

locale *Restriction-2-PreorderRestrictionSpace-2-PreorderRestrictionSpace* =
R2PRS1 : *Restriction-2-PreorderRestrictionSpace* $\langle(\downarrow_1)\rangle$ $\langle(\lessapprox_1)\rangle$ $\langle(\downarrow_2)\rangle$ $\langle(\lessapprox_2)\rangle$ +
PRS2 : *PreorderRestrictionSpace* $\langle(\downarrow_3)\rangle$ $\langle(\lessapprox_3)\rangle$
for $restriction_1$ :: $\langle'a \Rightarrow nat \Rightarrow 'a\rangle$ (**infixl** $\langle\downarrow_1\rangle$ 60)
and $relation_1$ :: $\langle'a \Rightarrow 'a \Rightarrow bool\rangle$ (**infixl** $\langle\lessapprox_1\rangle$ 50)
and $restriction_2$ :: $\langle'b \Rightarrow nat \Rightarrow 'b\rangle$ (**infixl** $\langle\downarrow_2\rangle$ 60)
and $relation_2$ :: $\langle'b \Rightarrow 'b \Rightarrow bool\rangle$ (**infixl** $\langle\lessapprox_2\rangle$ 50)
and $restriction_3$ :: $\langle'c \Rightarrow nat \Rightarrow 'c\rangle$ (**infixl** $\langle\downarrow_3\rangle$ 60)
and $relation_3$ :: $\langle'c \Rightarrow 'c \Rightarrow bool\rangle$ (**infixl** $\langle\lessapprox_3\rangle$ 50)
begin

interpretation *R2PRS2* : *Restriction-2-PreorderRestrictionSpace* $\langle(\downarrow_1)\rangle$ $\langle(\lessapprox_1)\rangle$ $\langle(\downarrow_3)\rangle$ $\langle(\lessapprox_3)\rangle$
⟨*proof*⟩

interpretation *PRS2PRS3* : *PreorderRestrictionSpace-2-PreorderRestrictionSpace* $\langle(\downarrow_2)\rangle$ $\langle(\lessapprox_2)\rangle$ $\langle(\downarrow_3)\rangle$ $\langle(\lessapprox_3)\rangle$
⟨*proof*⟩

theorem *restriction-shift-on-comp-restriction-shift-on* [*restriction-shift-simpset*] :
$\langle$*R2PRS2.restriction-shift-on* $(\lambda x.\ g\ (f\ x))\ (k + l)\ A\rangle$
if $\langle f\ `\ A \subseteq B\rangle$ $\langle$*PRS2PRS3.restriction-shift-on* $g\ l\ B\rangle$ $\langle$*R2PRS1.restriction-shift-on* $f\ k\ A\rangle$
⟨*proof*⟩

corollary *restriction-shift-comp-restriction-shift-on* [*restriction-shift-simpset*] :
$\langle$*PRS2PRS3.restriction-shift* $g\ l \Longrightarrow$ *R2PRS1.restriction-shift-on* $f\ k\ A \Longrightarrow$
R2PRS2.restriction-shift-on $(\lambda x.\ g\ (f\ x))\ (k + l)\ A\rangle$
⟨*proof*⟩

corollary *restriction-shift-comp-restriction-shift* [*restriction-shift-simpset*]
:
‹*PRS2PRS3.restriction-shift g l* ⟹ *R2PRS1.restriction-shift f k* ⟹
R2PRS2.restriction-shift (λ*x*. *g* (*f x*)) (*k* + *l*)›
⟨*proof*⟩

corollary *non-destructive-on-comp-non-destructive-on* [*restriction-shift-simpset*]
:
‹⟦*f* ‘ *A* ⊆ *B*; *PRS2PRS3.non-destructive-on g B*; *R2PRS1.non-destructive-on f A*⟧ ⟹
R2PRS2.non-destructive-on (λ*x*. *g* (*f x*)) *A*›
⟨*proof*⟩

corollary *non-destructive-comp-non-destructive-on* [*restriction-shift-simpset*]
:
‹*PRS2PRS3.non-destructive g* ⟹ *R2PRS1.non-destructive-on f A* ⟹
R2PRS2.non-destructive-on (λ*x*. *g* (*f x*)) *A*›
⟨*proof*⟩

corollary *non-destructive-comp-non-destructive* [*restriction-shift-simpset*]
:
‹*PRS2PRS3.non-destructive g* ⟹ *R2PRS1.non-destructive f* ⟹
R2PRS2.non-destructive (λ*x*. *g* (*f x*))›
⟨*proof*⟩

corollary *constructive-on-comp-non-destructive-on* [*restriction-shift-simpset*]
:
‹⟦*f* ‘ *A* ⊆ *B*; *PRS2PRS3.constructive-on g B*; *R2PRS1.non-destructive-on f A*⟧ ⟹
R2PRS2.constructive-on (λ*x*. *g* (*f x*)) *A*›
⟨*proof*⟩

corollary *constructive-comp-non-destructive-on* [*restriction-shift-simpset*]
:
‹*PRS2PRS3.constructive g* ⟹ *R2PRS1.non-destructive-on f A* ⟹
R2PRS2.constructive-on (λ*x*. *g* (*f x*)) *A*›
⟨*proof*⟩

corollary *constructive-comp-non-destructive* [*restriction-shift-simpset*]
:
‹*PRS2PRS3.constructive g* ⟹ *R2PRS1.non-destructive f* ⟹
R2PRS2.constructive (λ*x*. *g* (*f x*))›
⟨*proof*⟩

corollary *non-destructive-on-comp-constructive-on* [*restriction-shift-simpset*]
:
‹⟦*f* ‘ *A* ⊆ *B*; *PRS2PRS3.non-destructive-on g B*; *R2PRS1.constructive-on f A*⟧ ⟹
R2PRS2.constructive-on (λ*x*. *g* (*f x*)) *A*›
⟨*proof*⟩

corollary *non-destructive-comp-constructive-on* [*restriction-shift-simpset*]
:
‹*PRS2PRS3.non-destructive g* ⟹ *R2PRS1.constructive-on f A* ⟹
R2PRS2.constructive-on (λ*x*. *g* (*f x*)) *A*›
⟨*proof*⟩

corollary *non-destructive-comp-constructive* [*restriction-shift-simpset*]
:
‹*PRS2PRS3.non-destructive g* ⟹ *R2PRS1.constructive f* ⟹
R2PRS2.constructive (λ*x*. *g* (*f x*))›
⟨*proof*⟩

corollary *non-too-destructive-on-comp-non-destructive-on* [*restriction-shift-simpset*]
:
‹⟦*f* ‘ *A* ⊆ *B*; *PRS2PRS3.non-too-destructive-on g B*; *R2PRS1.non-destructive-on f A*⟧ ⟹
R2PRS2.non-too-destructive-on (λ*x*. *g* (*f x*)) *A*›
⟨*proof*⟩

corollary *non-too-destructive-comp-non-destructive-on* [*restriction-shift-simpset*]
:
‹*PRS2PRS3.non-too-destructive g* ⟹ *R2PRS1.non-destructive-on f A* ⟹
R2PRS2.non-too-destructive-on (λ*x*. *g* (*f x*)) *A*›
⟨*proof*⟩

corollary *non-too-destructive-comp-non-destructive* [*restriction-shift-simpset*]
:
‹*PRS2PRS3.non-too-destructive g* ⟹ *R2PRS1.non-destructive f* ⟹
R2PRS2.non-too-destructive (λ*x*. *g* (*f x*))›
⟨*proof*⟩

corollary *non-destructive-on-comp-non-too-destructive-on* [*restriction-shift-simpset*]
:
‹⟦*f* ‘ *A* ⊆ *B*; *PRS2PRS3.non-destructive-on g B*; *R2PRS1.non-too-destructive-on f A*⟧ ⟹
R2PRS2.non-too-destructive-on (λ*x*. *g* (*f x*)) *A*›
⟨*proof*⟩

corollary *non-destructive-comp-non-too-destructive-on* [*restriction-shift-simpset*]
:
‹*PRS2PRS3.non-destructive g* ⟹ *R2PRS1.non-too-destructive-on f A* ⟹
R2PRS2.non-too-destructive-on (λ*x*. *g* (*f x*)) *A*›
⟨*proof*⟩

corollary *non-destructive-comp-non-too-destructive* [*restriction-shift-simpset*]
:
‹*PRS2PRS3.non-destructive g* ⟹ *R2PRS1.non-too-destructive f* ⟹
R2PRS2.non-too-destructive (λ*x*. *g* (*f x*))›
⟨*proof*⟩

corollary *non-too-destructive-on-comp-constructive-on* [*restriction-shift-simpset*]
:
‹⟦*f* ‘ *A* ⊆ *B*; *PRS2PRS3.non-too-destructive-on g B*; *R2PRS1.constructive-on f A*⟧ ⟹
R2PRS2.non-destructive-on (λ*x*. *g* (*f x*)) *A*›
⟨*proof*⟩

corollary *non-too-destructive-comp-constructive-on* [*restriction-shift-simpset*]
:
‹*PRS2PRS3.non-too-destructive g* ⟹ *R2PRS1.constructive-on f A* ⟹
R2PRS2.non-destructive-on (λ*x*. *g* (*f x*)) *A*›
⟨*proof*⟩

corollary *non-too-destructive-comp-constructive* [*restriction-shift-simpset*]
:
‹*PRS2PRS3.non-too-destructive g* ⟹ *R2PRS1.constructive f* ⟹
R2PRS2.non-destructive (λ*x*. *g* (*f x*))›
⟨*proof*⟩

corollary *constructive-on-comp-non-too-destructive-on* [*restriction-shift-simpset*]
:
‹⟦*f* ‘ *A* ⊆ *B*; *PRS2PRS3.constructive-on g B*; *R2PRS1.non-too-destructive-on f A*⟧ ⟹
R2PRS2.non-destructive-on (λ*x*. *g* (*f x*)) *A*›
⟨*proof*⟩

corollary *constructive-comp-non-too-destructive-on* [*restriction-shift-simpset*]
:
‹*PRS2PRS3.constructive g* ⟹ *R2PRS1.non-too-destructive-on f A* ⟹
R2PRS2.non-destructive-on (λ*x*. *g* (*f x*)) *A*›

⟨*proof*⟩

corollary *constructive-comp-non-too-destructive* [*restriction-shift-simpset*]
:
‹*PRS2PRS3.constructive g* ⟹ *R2PRS1.non-too-destructive f* ⟹
R2PRS2.non-destructive (λ*x*. *g* (*f x*))›
⟨*proof*⟩

end

2 Class Implementation

2.1 Preliminaries

Small lemma from `HOL-Library.Infinite_Set` to avoid dependency.

lemma *INFM-nat-le*: ‹(∃$_\infty$*n* :: *nat*. *P n*) ⟷ (∀ *m*. ∃ *n*≥*m*. *P n*)›
⟨*proof*⟩

We need to be able to extract a subsequence verifying a predicate.

fun *extraction-subseq* :: ‹[*nat* ⇒ *'a*, *'a* ⇒ *bool*] ⇒ *nat* ⇒ *nat*›
where ‹*extraction-subseq* σ *P* 0 = (*LEAST k*. *P* (σ *k*))›
| ‹*extraction-subseq* σ *P* (*Suc n*) = (*LEAST k*. *extraction-subseq* σ
P n < *k* ∧ *P* (σ *k*))›

lemma *exists-extraction-subseq* :
assumes ‹∃$_\infty$*k*. *P* (σ *k*)›
defines *f-def* : ‹*f* ≡ *extraction-subseq* σ *P*›
shows ‹*strict-mono f*› **and** ‹*P* (σ (*f k*))›
⟨*proof*⟩

lemma *extraction-subseqD* :
‹∃*f* :: *nat* ⇒ *nat*. *strict-mono f* ∧ (∀ *k*. *P* (σ (*f k*)))› **if** ‹∃$_\infty$*k*. *P* (σ
k)›
⟨*proof*⟩

lemma *extraction-subseqE* :
— The idea is to abstract the concrete construction of this extraction function, we only need the fact that there is one.

‹$\exists_\infty k.\ P\ (\sigma\ k) \Longrightarrow (\bigwedge f :: nat \Rightarrow nat.\ strict\text{-}mono\ f \Longrightarrow (\bigwedge k.\ P\ (\sigma\ (f\ k))) \Longrightarrow thesis) \Longrightarrow thesis$›
⟨*proof*⟩

2.2 Basic Notions for Restriction

class *restriction* =
fixes *restriction* :: ‹['a, *nat*] ⇒ 'a› (**infixl** ‹↓› *60*)
assumes [*simp*] : ‹$x \downarrow n \downarrow m = x \downarrow min\ n\ m$›
begin

sublocale *Restriction* ‹(↓)› ‹(=)› ⟨*proof*⟩
end

class *restriction-space* = *restriction* +
assumes [*simp*] : ‹$x \downarrow 0 = y \downarrow 0$›
and *ex-not-restriction-eq* : ‹$x \neq y \Longrightarrow \exists n.\ x \downarrow n \neq y \downarrow n$›
begin

sublocale *PreorderRestrictionSpace* ‹(↓)› ‹(=)›
⟨*proof*⟩

lemma *restriction-related-set-commute* :
‹*restriction-related-set x y* = *restriction-related-set y x*› ⟨*proof*⟩

lemma *restriction-not-related-set-commute* :
‹*restriction-not-related-set x y* = *restriction-not-related-set y x*› ⟨*proof*⟩

end

context *restriction-space* **begin**

sublocale *Restriction-2-PreorderRestrictionSpace*
‹(↓) :: 'b :: *restriction* ⇒ *nat* ⇒ 'b› ‹(=)›
‹(↓) :: 'a ⇒ *nat* ⇒ 'a› ‹(=)› ⟨*proof*⟩

With this we recover constants like *local.restriction-shift-on*.

sublocale *PreorderRestrictionSpace-2-PreorderRestrictionSpace*
‹(↓) :: 'b :: *restriction-space* ⇒ *nat* ⇒ 'b› ‹(=)›
‹(↓) :: 'a ⇒ *nat* ⇒ 'a› ‹(=)› ⟨*proof*⟩

With that we recover theorems like ⟦*Restriction-2-PreorderRestrictionSpace.constructive* (↓) (=) (↓) (=) *?f*; *?x* ↓ *?n* = *?x* ↓ *?n*⟧ ⟹ *?f* (*?x* ↓ *?n*) ↓ *Suc ?n* = *?f ?x* ↓ *Suc ?n*.

sublocale *Restriction-2-PreorderRestrictionSpace-2-PreorderRestrictionSpace*
‹(↓) :: 'c :: *restriction* ⇒ *nat* ⇒ 'c› ‹(=)›
‹(↓) :: 'b :: *restriction-space* ⇒ *nat* ⇒ 'b› ‹(=)›

‹(↓) :: 'a ⇒ nat ⇒ 'a› ‹(=)› ⟨proof⟩

And with that we recover theorems like ⟦*?f ‘ ?A ⊆ ?B*; *Restriction-2-PreorderRestrictionSpace.constructive-on* (↓) (=) (↓) (=) *?g ?B*; *R2PRS1.non-destructive-on ?f ?A*⟧ ⟹ *Restriction-2-PreorderRestrictionSpace.construct* (↓) (=) (↓) (=) (λ*x*. *?g* (*?f x*)) *?A*.

lemma *restriction-shift-const* [*restriction-shift-simpset*] :
‹*restriction-shift* (λ*x*. *c*) *k*› ⟨*proof*⟩

lemma *constructive-const* [*restriction-shift-simpset*] :
‹*constructive* (λ*x*. *c*)› ⟨*proof*⟩

end

lemma *restriction-shift-on-restricted* [*restriction-shift-simpset*] :
‹*restriction-shift-on* (λ*x*. *f x* ↓ *n*) *k A*› **if** ‹*restriction-shift-on f k A*›
⟨*proof*⟩

lemma *restriction-shift-restricted* [*restriction-shift-simpset*] :
‹*restriction-shift f k* ⟹ *restriction-shift* (λ*x*. *f x* ↓ *n*) *k*›
⟨*proof*⟩

corollary *constructive-restricted* [*restriction-shift-simpset*] :
‹*constructive f* ⟹ *constructive* (λ*x*. *f x* ↓ *n*)›
⟨*proof*⟩

corollary *non-destructive-restricted* [*restriction-shift-simpset*] :
‹*non-destructive f* ⟹ *non-destructive* (λ*x*. *f x* ↓ *n*)›
⟨*proof*⟩

lemma *non-destructive-id* [*restriction-shift-simpset*] :
‹*non-destructive id*› ‹*non-destructive* (λ*x*. *x*)›
⟨*proof*⟩

interpretation *less-eqRS* : *Restriction* ‹(↓)› ‹(≤)› ⟨*proof*⟩

class *preorder-restriction-space* = *restriction* + *preorder* +
assumes *restriction-0-less-eq* [*simp*] : ‹*x* ↓ *0* ≤ *y* ↓ *0*›
and *mono-restriction-less-eq* : ‹*x* ≤ *y* ⟹ *x* ↓ *n* ≤ *y* ↓ *n*›
and *ex-not-restriction-less-eq* : ‹¬ *x* ≤ *y* ⟹ ∃ *n*. ¬ *x* ↓ *n* ≤ *y* ↓ *n*›
begin

sublocale *less-eqRS* : *PreorderRestrictionSpace* ‹(↓) :: *'a* ⇒ *nat* ⇒ *'a*› ‹(≤)›
⟨*proof*⟩

end

class *order-restriction-space* = *preorder-restriction-space* + *order*
begin

subclass *restriction-space*
⟨*proof*⟩

end

context *preorder-restriction-space* **begin**

sublocale *less-eqRS* : *Restriction-2-PreorderRestrictionSpace*
‹(↓) :: *'b* :: {*restriction*, *ord*} ⇒ *nat* ⇒ *'b*› ‹(≤)›
‹(↓) :: *'a* ⇒ *nat* ⇒ *'a*› ‹(≤)› ⟨*proof*⟩

With this we recover constants like *local.less-eqRS.restriction-shift-on*.

sublocale *less-eqRS* : *PreorderRestrictionSpace-2-PreorderRestrictionSpace*
‹(↓) :: *'b* :: *preorder-restriction-space* ⇒ *nat* ⇒ *'b*› ‹(≤)›
‹(↓) :: *'a* ⇒ *nat* ⇒ *'a*› ‹(≤)› ⟨*proof*⟩

With that we recover theorems like ⟦*Restriction-2-PreorderRestrictionSpace.constructive* (↓) (≤) (↓) (≤) *?f*; *?x* ↓ *?n* ≤ *?x* ↓ *?n*⟧ ⟹ *?f* (*?x* ↓ *?n*) ↓ *Suc* *?n* ≤ *?f ?x* ↓ *Suc ?n*.

sublocale *less-eqRS* : *Restriction-2-PreorderRestrictionSpace-2-PreorderRestrictionSpace*
‹(↓) :: *'c* :: *restriction* ⇒ *nat* ⇒ *'c*› ‹(=)›
‹(↓) :: *'b* :: *preorder-restriction-space* ⇒ *nat* ⇒ *'b*› ‹(≤)›
‹(↓) :: *'a* ⇒ *nat* ⇒ *'a*› ‹(≤)› ⟨*proof*⟩

And with that we recover theorems like ⟦*?f* ‘ *?A* ⊆ *?B*; *Restriction-2-PreorderRestrictionSpace.constructive-on* (↓) (≤) (↓) (≤) *?g ?B*; *local.less-eqRS.R2PRS1.non-destructive-on ?f ?A*⟧ ⟹ *Restriction-2-PreorderRestrictionSpace.constructive-on* (↓) (=) (↓) (≤) (λ*x*. *?g* (*?f x*)) *?A*.

end

context *order-restriction-space* **begin**

From ⟦*?x* ≤ *?y*; *?y* ≤ *?x*⟧ ⟹ *?x* = *?y* we can obtain stronger lemmas.

corollary *order-restriction-shift-onI* :
‹($\bigwedge x\ y\ n.\ [\![x \in A;\ y \in A;\ f\ x \neq f\ y;\ x \downarrow n = y \downarrow n]\!] \Longrightarrow$
$f\ x \downarrow nat\ (int\ n + k) \leq f\ y \downarrow nat\ (int\ n + k)$)
$\Longrightarrow$ *restriction-shift-on f k A*›
⟨*proof*⟩

corollary *order-restriction-shiftI* :
‹($\bigwedge x\ y\ n.\ [\![f\ x \neq f\ y;\ x \downarrow n = y \downarrow n]\!] \Longrightarrow$
$f\ x \downarrow nat\ (int\ n + k) \leq f\ y \downarrow nat\ (int\ n + k)$)
$\Longrightarrow$ *restriction-shift f k*›
⟨*proof*⟩

corollary *order-non-too-destructive-onI* :
‹($\bigwedge x\ y\ n.\ [\![x \in A;\ y \in A;\ f\ x \neq f\ y;\ x \downarrow Suc\ n = y \downarrow Suc\ n]\!] \Longrightarrow$
$f\ x \downarrow n \leq f\ y \downarrow n$)
$\Longrightarrow$ *non-too-destructive-on f A*›
⟨*proof*⟩

corollary *order-non-too-destructiveI* :
‹($\bigwedge x\ y\ n.\ [\![f\ x \neq f\ y;\ x \downarrow Suc\ n = y \downarrow Suc\ n]\!] \Longrightarrow f\ x \downarrow n \leq f\ y \downarrow n$)
$\Longrightarrow$ *non-too-destructive f*›
⟨*proof*⟩

corollary *order-non-destructive-onI* :
‹($\bigwedge x\ y\ n.\ [\![n \neq 0;\ x \in A;\ y \in A;\ f\ x \neq f\ y;\ x \downarrow n = y \downarrow n]\!] \Longrightarrow f\ x \downarrow n \leq f\ y \downarrow n$)
$\Longrightarrow$ *non-destructive-on f A*›
⟨*proof*⟩

corollary *order-non-destructiveI* :
‹($\bigwedge x\ y\ n.\ [\![n \neq 0;\ f\ x \neq f\ y;\ x \downarrow n = y \downarrow n]\!] \Longrightarrow f\ x \downarrow n \leq f\ y \downarrow n$)
$\Longrightarrow$ *non-destructive f*›
⟨*proof*⟩

corollary *order-constructive-onI* :
‹($\bigwedge x\ y\ n.\ [\![x \in A;\ y \in A;\ f\ x \neq f\ y;\ x \downarrow n = y \downarrow n]\!] \Longrightarrow f\ x \downarrow Suc\ n \leq f\ y \downarrow Suc\ n$)
$\Longrightarrow$ *constructive-on f A*›
⟨*proof*⟩

corollary *order-constructiveI* :
‹($\bigwedge x\ y\ n.\ [\![f\ x \neq f\ y;\ x \downarrow n = y \downarrow n]\!] \Longrightarrow f\ x \downarrow Suc\ n \leq f\ y \downarrow Suc\ n$)
$\Longrightarrow$ *constructive f*›
⟨*proof*⟩

end

2.3 Definition of the Fixed-Point Operator

2.3.1 Preliminaries

Chain **context** *restriction* **begin**

definition *restriction-chain* :: ‹[*nat* ⇒ 'a] ⇒ *bool*› (‹*chain*$_\downarrow$›)
where ‹*restriction-chain* $\sigma \equiv \forall\, n.\ \sigma\ (Suc\ n) \downarrow n = \sigma\ n$›

lemma *restriction-chainI* : ‹$(\bigwedge n.\ \sigma\ (Suc\ n) \downarrow n = \sigma\ n) \Longrightarrow$ *restriction-chain* σ›
and *restriction-chainD* : ‹*restriction-chain* $\sigma \Longrightarrow \sigma\ (Suc\ n) \downarrow n = \sigma\ n$›
⟨*proof*⟩

end

context *restriction-space* **begin**

lemma (**in** *restriction-space*) *restriction-chain-def-bis*:
‹*restriction-chain* $\sigma \longleftrightarrow (\forall\, n\ m.\ n < m \longrightarrow \sigma\ m \downarrow n = \sigma\ n)$›
⟨*proof*⟩

lemma *restricted-restriction-chain-is* :
‹*restriction-chain* $\sigma \Longrightarrow (\lambda n.\ \sigma\ n \downarrow n) = \sigma$›
⟨*proof*⟩

lemma *restriction-chain-def-ter*:
‹*restriction-chain* $\sigma \longleftrightarrow (\forall\, n\ m.\ n \leq m \longrightarrow \sigma\ m \downarrow n = \sigma\ n)$›
⟨*proof*⟩

lemma *restriction-chain-restrictions* : ‹*restriction-chain* $((\downarrow)\ x)$›
⟨*proof*⟩

end

Iterations The sequence of restricted images of powers of a constructive function is a *chain*$_\downarrow$.

context fixes f :: ‹'a ⇒ 'a :: *restriction-space*› **begin**

lemma *restriction-chain-funpow-restricted* [*simp*]:
‹*restriction-chain* $(\lambda n.\ (f \frown n)\ x \downarrow n)$› **if** ‹*constructive* f›
⟨*proof*⟩

lemma *constructive-imp-eq-funpow-restricted* :
‹$n \leq k \Longrightarrow n \leq l \Longrightarrow (f \frown k)\ x \downarrow n = (f \frown l)\ y \downarrow n$› **if** ‹*constructive* f›

⟨*proof*⟩

end

Limits and Convergence **context** *restriction* **begin**

definition *restriction-tendsto* :: ‹[*nat* ⇒ *'a*, *'a*] ⇒ *bool*› (‹((-)/ −↓→ (-))› [*59*, *59*] *59*)
where ‹σ −↓→ Σ ≡ ∀ *n*. ∃ *n0*. ∀ *k*≥*n0*. Σ ↓ *n* = σ *k* ↓ *n*›

lemma *restriction-tendstoI* : ‹(⋀*n*. ∃ *n0*. ∀ *k*≥*n0*. Σ ↓ *n* = σ *k* ↓ *n*) ⟹ σ −↓→ Σ›
⟨*proof*⟩

lemma *restriction-tendstoD* : ‹σ −↓→ Σ ⟹ ∃ *n0*. ∀ *k*≥*n0*. Σ ↓ *n* = σ *k* ↓ *n*›
⟨*proof*⟩

lemma *restriction-tendstoE* :
‹σ −↓→ Σ ⟹ (⋀*n0*. (⋀*k*. *n0* ≤ *k* ⟹ Σ ↓ *n* = σ *k* ↓ *n*) ⟹ *thesis*) ⟹ *thesis*›
⟨*proof*⟩

end

lemma (**in** *restriction-space*) *restriction-tendsto-unique* :
‹σ −↓→ Σ ⟹ σ −↓→ Σ′ ⟹ Σ = Σ′›
⟨*proof*⟩

context *restriction* **begin**

lemma *restriction-tendsto-const-restricted* :
‹σ −↓→ Σ ⟹ (λ*n*. σ *n* ↓ *k*) −↓→ Σ ↓ *k*›
⟨*proof*⟩

lemma *restriction-tendsto-iff-eventually-in-restriction-eq-set* :
‹σ −↓→ Σ ⟷ (∀ *n*. ∃ *n0*. ∀ *k*≥*n0*. *n* ∈ *restriction-related-set* Σ (σ *k*))›
⟨*proof*⟩

lemma *restriction-tendsto-const* : ‹(λ*n*. Σ) −↓→ Σ›
⟨*proof*⟩

lemma (**in** *restriction-space*) *restriction-tendsto-restrictions* : ‹(λ*n*. Σ ↓ *n*) −↓→ Σ›
⟨*proof*⟩

lemma *restriction-tendsto-shift-iff* : ‹$(\lambda n.\ \sigma\ (n + l))\ -\downarrow\rightarrow \Sigma \longleftrightarrow \sigma\ -\downarrow\rightarrow \Sigma$›
⟨*proof*⟩

lemma *restriction-tendsto-shiftI* : ‹$\sigma\ -\downarrow\rightarrow \Sigma \Longrightarrow (\lambda n.\ \sigma\ (n + l))\ -\downarrow\rightarrow \Sigma$›
⟨*proof*⟩

lemma *restriction-tendsto-shiftD* : ‹$(\lambda n.\ \sigma\ (n + l))\ -\downarrow\rightarrow \Sigma \Longrightarrow \sigma\ -\downarrow\rightarrow \Sigma$›
⟨*proof*⟩

lemma (**in** *restriction-space*) *restriction-tendsto-restricted-iff-restriction-tendsto* :
‹$(\lambda n.\ \sigma\ n \downarrow n)\ -\downarrow\rightarrow \Sigma \longleftrightarrow \sigma\ -\downarrow\rightarrow \Sigma$›
⟨*proof*⟩

lemma *restriction-tendsto-subseq* :
‹$(\sigma \circ f)\ -\downarrow\rightarrow \Sigma$› **if** ‹*strict-mono f*› **and** ‹$\sigma\ -\downarrow\rightarrow \Sigma$›
⟨*proof*⟩

end

context *restriction* **begin**

definition *restriction-convergent* :: ‹$(nat \Rightarrow 'a) \Rightarrow bool$› (‹$convergent_{\downarrow}$›)
where ‹*restriction-convergent* $\sigma \equiv \exists \Sigma.\ \sigma\ -\downarrow\rightarrow \Sigma$›

lemma *restriction-convergentI* : ‹$\sigma\ -\downarrow\rightarrow \Sigma \Longrightarrow$ *restriction-convergent* σ›
⟨*proof*⟩

lemma *restriction-convergentD'* : ‹*restriction-convergent* $\sigma \Longrightarrow \exists \Sigma.\ \sigma\ -\downarrow\rightarrow \Sigma$›
⟨*proof*⟩

end

context *restriction-space* **begin**

lemma *restriction-convergentD* :
‹*restriction-convergent* $\sigma \Longrightarrow \exists !\Sigma.\ \sigma\ -\downarrow\rightarrow \Sigma$›
⟨*proof*⟩

lemma *restriction-convergentE* :
‹*restriction-convergent* $\sigma \Longrightarrow$
$(\bigwedge\Sigma.\ \sigma\ -\downarrow\rightarrow\ \Sigma \Longrightarrow (\bigwedge\Sigma'.\ \sigma\ -\downarrow\rightarrow\ \Sigma' \Longrightarrow \Sigma' = \Sigma) \Longrightarrow \mathit{thesis}) \Longrightarrow$
thesis›
⟨*proof*⟩

lemma *restriction-tendsto-of-restriction-convergent* :
‹*restriction-convergent* $\sigma \Longrightarrow \sigma\ -\downarrow\rightarrow\ (\mathit{THE}\ \Sigma.\ \sigma\ -\downarrow\rightarrow\ \Sigma)$›
⟨*proof*⟩

end

context *restriction* **begin**

lemma *restriction-convergent-const* [*simp*] : ‹$\mathit{convergent}_{\downarrow}\ (\lambda n.\ \Sigma)$›
⟨*proof*⟩

lemma (**in** *restriction-space*) *restriction-convergent-restrictions* [*simp*]
:
‹$\mathit{convergent}_{\downarrow}\ (\lambda n.\ \Sigma \downarrow n)$›
⟨*proof*⟩

lemma *restriction-convergent-shift-iff* :
‹$\mathit{convergent}_{\downarrow}\ (\lambda n.\ \sigma\ (n + l)) \longleftrightarrow \mathit{convergent}_{\downarrow}\ \sigma$›
⟨*proof*⟩

lemma *restriction-convergent-shift-shiftI* :
‹$\mathit{convergent}_{\downarrow}\ \sigma \Longrightarrow \mathit{convergent}_{\downarrow}\ (\lambda n.\ \sigma\ (n + l))$›
⟨*proof*⟩

lemma *restriction-convergent-shift-shiftD* :
‹$\mathit{convergent}_{\downarrow}\ (\lambda n.\ \sigma\ (n + l)) \Longrightarrow \mathit{convergent}_{\downarrow}\ \sigma$›
⟨*proof*⟩

lemma (**in** *restriction-space*) *restriction-convergent-restricted-iff-restriction-convergent*
:
‹$\mathit{convergent}_{\downarrow}\ (\lambda n.\ \sigma\ n \downarrow n) \longleftrightarrow \mathit{convergent}_{\downarrow}\ \sigma$›
⟨*proof*⟩

lemma *restriction-convergent-subseq* :
‹*strict-mono* $f \Longrightarrow$ *restriction-convergent* $\sigma \Longrightarrow$ *restriction-convergent*
$(\sigma \circ f)$›
⟨*proof*⟩

lemma (**in** *restriction-space*)
convergent-restriction-chain-imp-ex1 : ‹∃ !Σ. ∀ *n*. Σ ↓ *n* = σ *n*›
and *restriction-tendsto-of-convergent-restriction-chain* : ‹σ −↓→ (*THE* Σ. ∀ *n*. Σ ↓ *n* = σ *n*)›
if ‹*restriction-convergent* σ› **and** ‹*restriction-chain* σ›
⟨*proof*⟩

end

2.3.2 Fixed-Point Operator

Our definition only makes sense if such a fixed point exists and is unique. We will therefore directly add a completeness assumption, and define the fixed-point operator within this context. It will only be valid when the function f is *constructive*.

class *complete-restriction-space* = *restriction-space* +
assumes *restriction-chain-imp-restriction-convergent* : ‹*chain*↓ σ ⟹ *convergent*↓ σ›

definition (**in** *complete-restriction-space*)
restriction-fix :: ‹('*a* ⇒ '*a*) ⇒ '*a*›
— We will use a syntax rather than a binder to be compatible with the product.
where ‹*restriction-fix* (λ*x*. *f x*) ≡ *THE* Σ. (λ*n*. (*f* ^^ *n*) *undefined*) −↓→ Σ›

syntax -*restriction-fix* :: ‹[*pttrn*, '*a* ⇒ '*a*] ⇒ '*a*›
(‹(‹*indent*=3 *notation*=‹*binder restriction-fix*››υ -./ -)› [0, 10] 10)
syntax-consts -*restriction-fix* ⇌ *restriction-fix*
translations υ *x*. *f* ⇌ *CONST restriction-fix* (λ*x*. *f*)
⟨*ML*⟩

context *complete-restriction-space* **begin**

The following result is quite similar to the Banach's fixed point theorem.

lemma *restriction-chain-imp-ex1* : ‹∃ !Σ. ∀ *n*. Σ ↓ *n* = σ *n*›
and *restriction-tendsto-of-restriction-chain* : ‹σ −↓→ (*THE* Σ. ∀ *n*. Σ ↓ *n* = σ *n*)›
if ‹*restriction-chain* σ›
⟨*proof*⟩

lemma *restriction-chain-is* :
‹σ = (↓) (*THE* Σ. σ −↓→ Σ)›
‹σ = (↓) (*THE* Σ. ∀ *n*. Σ ↓ *n* = σ *n*)› **if** ‹*restriction-chain* σ›

⟨*proof*⟩

end

context
  fixes f :: ‹$'a \Rightarrow 'a$ :: *complete-restriction-space*›
  assumes ‹*constructive* f›
begin

lemma *ex1-restriction-fix* :
  ‹$\exists!\Sigma.\ \forall x.\ (\lambda n.\ (f\ \frown\frown\ n)\ x)\ -\downarrow\rightarrow\ \Sigma$›
⟨*proof*⟩

lemma *ex1-restriction-fix-bis* :
  ‹$\exists!\Sigma.\ (\lambda n.\ (f\ \frown\frown\ n)\ x)\ -\downarrow\rightarrow\ \Sigma$›
  ⟨*proof*⟩

lemma *restriction-fix-def-bis* :
  ‹$(\upsilon\ x.\ f\ x) = (THE\ \Sigma.\ (\lambda n.\ (f\ \frown\frown\ n)\ x)\ -\downarrow\rightarrow\ \Sigma)$›
⟨*proof*⟩

lemma *funpow-restriction-tendsto-restriction-fix* : ‹$(\lambda n.\ (f\ \frown\frown\ n)\ x)$
$-\downarrow\rightarrow (\upsilon\ x.\ f\ x)$›
  ⟨*proof*⟩

lemma *restriction-restriction-fix-is* : ‹$(\upsilon\ x.\ f\ x)\downarrow n = (f\ \frown\frown\ n)\ x\downarrow n$›
⟨*proof*⟩

lemma *restriction-fix-eq* : ‹$(\upsilon\ x.\ f\ x) = f\ (\upsilon\ x.\ f\ x)$›
⟨*proof*⟩

lemma *restriction-fix-unique* : ‹$f\ x = x \Longrightarrow (\upsilon\ x.\ f\ x) = x$›
  ⟨*proof*⟩

lemma *restriction-fix-def-ter* : ‹$(\upsilon\ x.\ f\ x) = (THE\ x.\ f\ x = x)$›
  ⟨*proof*⟩

end

3 Product over Restriction Spaces

3.1 Restriction Space

instantiation *prod* :: (*restriction*, *restriction*) *restriction*
begin

definition *restriction-prod* :: ‹$'a \times 'b \Rightarrow nat \Rightarrow 'a \times 'b$›
where ‹$p \downarrow n \equiv (fst\ p \downarrow n, snd\ p \downarrow n)$›

instance ⟨*proof*⟩

end

instance *prod* :: (*restriction-space*, *restriction-space*) *restriction-space*
⟨*proof*⟩

instantiation *prod* :: (*preorder-restriction-space*, *preorder-restriction-space*) *preorder-restriction-space*
begin

We might want to use lexicographic order :

- $p \leq q \equiv fst\ p < fst\ q \vee fst\ p = fst\ q \wedge snd\ p \leq snd\ q$
- $p < q \equiv fst\ p < fst\ q \vee fst\ p = fst\ q \wedge snd\ p < snd\ q$

but this is wrong since it is incompatible with $p \downarrow 0 \leq q \downarrow 0$, $\neg\ p \leq q \Longrightarrow \exists\, n.\ \neg\ p \downarrow n \leq q \downarrow n$ and $p \leq q \Longrightarrow p \downarrow n \leq q \downarrow n$.

definition *less-eq-prod* :: ‹$'a \times 'b \Rightarrow 'a \times 'b \Rightarrow bool$›
where ‹$p \leq q \equiv fst\ p \leq fst\ q \wedge snd\ p \leq snd\ q$›

definition *less-prod* :: ‹$'a \times 'b \Rightarrow 'a \times 'b \Rightarrow bool$›
where ‹$p < q \equiv fst\ p \leq fst\ q \wedge snd\ p < snd\ q \vee fst\ p < fst\ q \wedge snd\ p \leq snd\ q$›

instance
⟨*proof*⟩

end

instance *prod* :: (*order-restriction-space*, *order-restriction-space*) *order-restriction-space*
⟨*proof*⟩

3.2 Restriction shift Maps

3.2.1 Domain is a Product

lemma *restriction-shift-on-prod-domain-iff* :
‹*restriction-shift-on f k* (*A* × *B*) ⟷ (∀ *x*∈*A*. *restriction-shift-on* (λ*y*. *f* (*x*, *y*)) *k B*) ∧
(∀ *y*∈*B*. *restriction-shift-on* (λ*x*. *f* (*x*, *y*)) *k A*)›
⟨*proof*⟩

lemma *restriction-shift-prod-domain-iff*:
‹*restriction-shift f k* ⟷ (∀ *x*. *restriction-shift* (λ*y*. *f* (*x*, *y*)) *k*) ∧
(∀ *y*. *restriction-shift* (λ*x*. *f* (*x*, *y*)) *k*)›
⟨*proof*⟩

lemma *non-too-destructive-on-prod-domain-iff* :
‹*non-too-destructive-on f* (*A* × *B*) ⟷ (∀ *x*∈*A*. *non-too-destructive-on* (λ*y*. *f* (*x*, *y*)) *B*) ∧
(∀ *y*∈*B*. *non-too-destructive-on* (λ*x*. *f* (*x*, *y*)) *A*)›
⟨*proof*⟩

lemma *non-too-destructive-prod-domain-iff* :
‹*non-too-destructive f* ⟷ (∀ *x*. *non-too-destructive* (λ*y*. *f* (*x*, *y*))) ∧
(∀ *y*. *non-too-destructive* (λ*x*. *f* (*x*, *y*)))›
⟨*proof*⟩

lemma *non-destructive-on-prod-domain-iff* :
‹*non-destructive-on f* (*A* × *B*) ⟷ (∀ *x*∈*A*. *non-destructive-on* (λ*y*. *f* (*x*, *y*)) *B*) ∧
(∀ *y*∈*B*. *non-destructive-on* (λ*x*. *f* (*x*, *y*)) *A*)›
⟨*proof*⟩

lemma *non-destructive-prod-domain-iff* :
‹*non-destructive f* ⟷ (∀ *x*. *non-destructive* (λ*y*. *f* (*x*, *y*))) ∧
(∀ *y*. *non-destructive* (λ*x*. *f* (*x*, *y*)))›
⟨*proof*⟩

lemma *constructive-on-prod-domain-iff* :
‹*constructive-on f* (*A* × *B*) ⟷ (∀ *x*∈*A*. *constructive-on* (λ*y*. *f* (*x*,

$y))\ B) \wedge$
$(\forall y{\in}B.\ constructive\text{-}on\ (\lambda x.\ f\ (x,\ y))\ A)\rangle$
⟨*proof*⟩

lemma *constructive-prod-domain-iff* :
‹*constructive f* ⟷ $(\forall x.\ constructive\ (\lambda y.\ f\ (x,\ y))) \wedge$
$(\forall y.\ constructive\ (\lambda x.\ f\ (x,\ y)))$›
⟨*proof*⟩

lemma *restriction-shift-prod-domain* [*restriction-shift-simpset*, *restriction-shift-introset*] :
‹⟦$\bigwedge x.\ restriction\text{-}shift\ (\lambda y.\ f\ (x,\ y))\ k$;
$\bigwedge y.\ restriction\text{-}shift\ (\lambda x.\ f\ (x,\ y))\ k$⟧ ⟹ *restriction-shift f k*›
and *non-too-destructive-prod-domain* [*restriction-shift-simpset*, *restriction-shift-introset*] :
‹⟦$\bigwedge x.\ non\text{-}too\text{-}destructive\ (\lambda y.\ f\ (x,\ y))$;
$\bigwedge y.\ non\text{-}too\text{-}destructive\ (\lambda x.\ f\ (x,\ y))$⟧ ⟹ *non-too-destructive f*›
and *non-destructive-prod-domain* [*restriction-shift-simpset*, *restriction-shift-introset*] :
‹⟦$\bigwedge x.\ non\text{-}destructive\ (\lambda y.\ f\ (x,\ y))$;
$\bigwedge y.\ non\text{-}destructive\ (\lambda x.\ f\ (x,\ y))$⟧ ⟹ *non-destructive f*›
and *constructive-prod-domain* [*restriction-shift-simpset*, *restriction-shift-introset*] :
‹⟦$\bigwedge x.\ constructive\ (\lambda y.\ f\ (x,\ y))$;
$\bigwedge y.\ constructive\ (\lambda x.\ f\ (x,\ y))$⟧ ⟹ *constructive f*›
⟨*proof*⟩

3.2.2 Codomain is a Product

lemma *restriction-shift-on-prod-codomain-iff* :
‹*restriction-shift-on f k A* ⟷ $(restriction\text{-}shift\text{-}on\ (\lambda x.\ fst\ (f\ x))\ k\ A) \wedge$
$(restriction\text{-}shift\text{-}on\ (\lambda x.\ snd\ (f\ x))\ k\ A)$›
⟨*proof*⟩

lemma *restriction-shift-prod-codomain-iff*:
‹*restriction-shift f k* ⟷ $(restriction\text{-}shift\ (\lambda x.\ fst\ (f\ x))\ k) \wedge$
$(restriction\text{-}shift\ (\lambda x.\ snd\ (f\ x))\ k)$›
⟨*proof*⟩

lemma *non-too-destructive-on-prod-codomain-iff* :
‹*non-too-destructive-on f A* ⟷ $(non\text{-}too\text{-}destructive\text{-}on\ (\lambda x.\ fst\ (f\ x))\ A) \wedge$
$(non\text{-}too\text{-}destructive\text{-}on\ (\lambda x.\ snd\ (f\ x))\ A)$›
⟨*proof*⟩

lemma *non-too-destructive-prod-codomain-iff* :
‹*non-too-destructive f* ⟷ (*non-too-destructive* (λ*x*. *fst* (*f x*))) ∧
(*non-too-destructive* (λ*x*. *snd* (*f x*)))›
⟨*proof*⟩

lemma *non-destructive-on-prod-codomain-iff* :
‹*non-destructive-on f A* ⟷ (*non-destructive-on* (λ*x*. *fst* (*f x*)) *A*) ∧
(*non-destructive-on* (λ*x*. *snd* (*f x*)) *A*)›

⟨*proof*⟩

lemma *non-destructive-prod-codomain-iff* :
‹*non-destructive f* ⟷ (*non-destructive* (λ*x*. *fst* (*f x*))) ∧
(*non-destructive* (λ*x*. *snd* (*f x*)))›
⟨*proof*⟩

lemma *constructive-on-prod-codomain-iff* :
‹*constructive-on f A* ⟷ (*constructive-on* (λ*x*. *fst* (*f x*)) *A*) ∧
(*constructive-on* (λ*x*. *snd* (*f x*)) *A*)›
⟨*proof*⟩

lemma *constructive-prod-codomain-iff* :
‹*constructive f* ⟷ (*constructive* (λ*x*. *fst* (*f x*))) ∧
(*constructive* (λ*x*. *snd* (*f x*)))›
⟨*proof*⟩

lemma *restriction-shift-prod-codomain* [*restriction-shift-simpset*, *restriction-shift-introset*] :
‹⟦*restriction-shift f k*; *restriction-shift g k*⟧ ⟹
restriction-shift (λ*x*. (*f x*, *g x*)) *k*›
and *non-too-destructive-prod-codomain* [*restriction-shift-simpset*, *restriction-shift-introset*] :
‹⟦*non-too-destructive f*; *non-too-destructive g*⟧ ⟹ *non-too-destructive* (λ*x*. (*f x*, *g x*))›
and *non-destructive-prod-codomain* [*restriction-shift-simpset*, *restriction-shift-introset*] :
‹⟦*non-destructive f*; *non-destructive g*⟧ ⟹ *non-destructive* (λ*x*. (*f x*, *g x*))›
and *constructive-prod-codomain* [*restriction-shift-simpset*, *restriction-shift-introset*] :
‹⟦*constructive f*; *constructive g*⟧ ⟹ *constructive* (λ*x*. (*f x*, *g x*))›
⟨*proof*⟩

3.3 Limits and Convergence

lemma *restriction-chain-prod-iff* :
‹*restriction-chain* $\sigma \longleftrightarrow$ *restriction-chain* $(\lambda n.\ fst\ (\sigma\ n)) \wedge$
restriction-chain $(\lambda n.\ snd\ (\sigma\ n))$›
⟨*proof*⟩

lemma *restriction-tendsto-prod-iff* :
‹$\sigma\ -\downarrow\rightarrow \Sigma \longleftrightarrow (\lambda n.\ fst\ (\sigma\ n))\ -\downarrow\rightarrow\ fst\ \Sigma \wedge (\lambda n.\ snd\ (\sigma\ n))\ -\downarrow\rightarrow\ snd\ \Sigma$›
⟨*proof*⟩

lemma *restriction-convergent-prod-iff* :
‹*restriction-convergent* $\sigma \longleftrightarrow$ *restriction-convergent* $(\lambda n.\ fst\ (\sigma\ n)) \wedge$
restriction-convergent $(\lambda n.\ snd\ (\sigma\ n))$›
⟨*proof*⟩

lemma *funpow-indep-prod-is* :
‹$((\lambda(x,\ y).\ (f\ x,\ g\ y))\ \widehat{\ }\ n)\ (x,\ y) = ((f\ \widehat{\ }\ n)\ x,\ (g\ \widehat{\ }\ n)\ y)$›
for $f\ g$ :: ‹$'a \Rightarrow 'a$›
⟨*proof*⟩

3.4 Completeness

instance *prod* :: (*complete-restriction-space*, *complete-restriction-space*) *complete-restriction-space*
⟨*proof*⟩

3.5 Fixed Point

lemma *restriction-fix-indep-prod-is* :
‹$(\upsilon\ (x,\ y).\ (f\ x,\ g\ y)) = (\upsilon\ x.\ f\ x,\ \upsilon\ y.\ g\ y)$›
if *contructive* : ‹*constructive* f› ‹*constructive* g›
for f :: ‹$'a \Rightarrow 'a$ :: *complete-restriction-space*›
and g :: ‹$'b \Rightarrow 'b$ :: *complete-restriction-space*›
⟨*proof*⟩

lemma *non-destructive-fst* : ‹*non-destructive fst*›
⟨*proof*⟩

lemma *non-destructive-snd* : ‹*non-destructive snd*›
⟨*proof*⟩

lemma *constructive-restriction-fix-right* :

‹*constructive* (λ*x*. ∪ *y*. *f* (*x*, *y*))› **if** ‹*constructive f*›
for *f* :: ‹′*a* :: *complete-restriction-space* × ′*b* :: *complete-restriction-space*
⇒ ′*b*›
⟨*proof*⟩

lemma *constructive-restriction-fix-left* :
‹*constructive* (λ*y*. ∪ *x*. *f* (*x*, *y*))› **if** ‹*constructive f*›
for *f* :: ‹′*a* :: *complete-restriction-space* × ′*b* :: *complete-restriction-space*
⇒ ′*a*›
⟨*proof*⟩

lemma *restriction-fix-prod-is* :
‹(∪ *p*. *f p*) = (∪ *x*. *fst* (*f* (*x*, ∪ *y*. *snd* (*f* (*x*, *y*)))),
∪ *y*. *snd* (*f* (∪ *x*. *fst* (*f* (*x*, ∪ *y*. *snd* (*f* (*x*, *y*)))), *y*)))›
(**is** ‹(∪ *p*. *f p*) = (*?x*, *?y*)›) **if** ‹*constructive f*›
for *f* :: ‹′*a* :: *complete-restriction-space* × ′*b* :: *complete-restriction-space*
⇒ ′*a* × ′*b*›
⟨*proof*⟩

4 Functions towards a Restriction Space

4.1 Restriction Space

instantiation ‹*fun*› :: (*type*, *restriction*) *restriction*
begin

definition *restriction-fun* :: ‹[′*a* ⇒ ′*b*, *nat*, ′*a*] ⇒ ′*b*›
where ‹*f* ↓ *n* ≡ (λ*x*. *f x* ↓ *n*)›

instance ⟨*proof*⟩

end

instance ‹*fun*› :: (*type*, *restriction-space*) *restriction-space*
⟨*proof*⟩

instance ‹*fun*› :: (*type*, *preorder-restriction-space*) *preorder-restriction-space*
⟨*proof*⟩

instance ‹*fun*› :: (*type*, *order-restriction-space*) *order-restriction-space*
⟨*proof*⟩

4.2 Restriction shift Maps

lemma *restriction-shift-on-fun-iff* :

‹*restriction-shift-on f k A* ⟷ (∀ *z*. *restriction-shift-on* (λ*x*. *f x z*) *k A*)›
⟨*proof*⟩

lemma *restriction-shift-fun-iff* : ‹*restriction-shift f k* ⟷ (∀ *z*. *restriction-shift* (λ*x*. *f x z*) *k*)›
⟨*proof*⟩

lemma *non-too-destructive-on-fun-iff*:
‹*non-too-destructive-on f A* ⟷ (∀ *z*. *non-too-destructive-on* (λ*x*. *f x z*) *A*)›
⟨*proof*⟩

lemma *non-too-destructive-fun-iff*:
‹*non-too-destructive f* ⟷ (∀ *z*. *non-too-destructive* (λ*x*. *f x z*))›
⟨*proof*⟩

lemma *non-destructive-on-fun-iff*:
‹*non-destructive-on f A* ⟷ (∀ *z*. *non-destructive-on* (λ*x*. *f x z*) *A*)›
⟨*proof*⟩

lemma *non-destructive-fun-iff*:
‹*non-destructive f* ⟷ (∀ *z*. *non-destructive* (λ*x*. *f x z*))›
⟨*proof*⟩

lemma *constructive-on-fun-iff*:
‹*constructive-on f A* ⟷ (∀ *z*. *constructive-on* (λ*x*. *f x z*) *A*)›
⟨*proof*⟩

lemma *constructive-fun-iff*:
‹*constructive f* ⟷ (∀ *z*. *constructive* (λ*x*. *f x z*))›
⟨*proof*⟩

lemma *restriction-shift-fun* [*restriction-shift-simpset*, *restriction-shift-introset*]
:
‹(⋀*z*. *restriction-shift* (λ*x*. *f x z*) *k*) ⟹ *restriction-shift f k*›
and *non-too-destructive-fun* [*restriction-shift-simpset*, *restriction-shift-introset*]
:
‹(⋀*z*. *non-too-destructive* (λ*x*. *f x z*)) ⟹ *non-too-destructive f*›
and *non-destructive-fun* [*restriction-shift-simpset*, *restriction-shift-introset*]
:
‹(⋀*z*. *non-destructive* (λ*x*. *f x z*)) ⟹ *non-destructive f*›
and *constructive-fun* [*restriction-shift-simpset*, *restriction-shift-introset*]
:

‹(⋀z. constructive (λx. f x z)) ⟹ constructive f›
⟨proof⟩

4.3 Limits and Convergence

lemma *reached-dist-funE* :
fixes *f g* :: ‹'a ⇒ 'b :: *restriction-space*› **assumes** ‹*f* ≠ *g*›
obtains *x* **where** ‹*f x* ≠ *g x*› ‹*Sup* (*restriction-related-set f g*) = *Sup* (*restriction-related-set* (*f x*) (*g x*))›
— Morally, we say here that the distance between two functions is reached. But we did not introduce the concept of distance.
⟨proof⟩

lemma *reached-restriction-related-set-funE* :
fixes *f g* :: ‹'a ⇒ 'b :: *restriction-space*›
obtains *x* **where** ‹*restriction-related-set f g* = *restriction-related-set* (*f x*) (*g x*)›
⟨proof⟩

lemma *restriction-chain-fun-iff* :
‹*restriction-chain* σ ⟷ (∀ z. *restriction-chain* (λn. σ n z))›
⟨proof⟩

lemma *restriction-tendsto-fun-imp* : ‹σ −↓→ Σ ⟹ (λn. σ n z) −↓→ Σ z›
⟨proof⟩

lemma *restriction-convergent-fun-imp* :
‹*restriction-convergent* σ ⟹ *restriction-convergent* (λn. σ n z)›
⟨proof⟩

4.4 Completeness

instance ‹*fun*› :: (*type*, *complete-restriction-space*) *complete-restriction-space*
⟨proof⟩

5 Topological Notions

named-theorems *restriction-cont-simpset* — For future automation.

5.1 Continuity

context *restriction* **begin**

definition *restriction-cont-at* :: ‹['b :: *restriction* ⇒ 'a, 'b] ⇒ *bool*›
(‹$cont_{\downarrow}$ (-) *at* (-)› [*1000*, *1000*])
where ‹$cont_{\downarrow}$ *f at* Σ ≡ ∀σ. σ $-\downarrow\rightarrow$ Σ ⟶ (λ*n*. *f* (σ *n*)) $-\downarrow\rightarrow$ *f* Σ›

lemma *restriction-cont-atI* : ‹(⋀σ. σ $-\downarrow\rightarrow$ Σ ⟹ (λ*n*. *f* (σ *n*)) $-\downarrow\rightarrow$ *f* Σ) ⟹ $cont_{\downarrow}$ *f at* Σ›
⟨*proof*⟩

lemma *restriction-cont-atD* : ‹$cont_{\downarrow}$ *f at* Σ ⟹ σ $-\downarrow\rightarrow$ Σ ⟹ (λ*n*. *f* (σ *n*)) $-\downarrow\rightarrow$ *f* Σ›
⟨*proof*⟩

lemma *restriction-cont-at-comp* [*restriction-cont-simpset*] :
‹$cont_{\downarrow}$ *f at* Σ ⟹ $cont_{\downarrow}$ *g at* (*f* Σ) ⟹ $cont_{\downarrow}$ (λ*x*. *g* (*f x*)) *at* Σ›
⟨*proof*⟩

lemma *restriction-cont-at-if-then-else* [*restriction-cont-simpset*] :
‹⟦⋀*x*. *P x* ⟹ $cont_{\downarrow}$ (*f x*) *at* Σ; ⋀*x*. ¬ *P x* ⟹ $cont_{\downarrow}$ (*g x*) *at* Σ⟧
⟹ $cont_{\downarrow}$ (λ*y*. *if P x then f x y else g x y*) *at* Σ›
⟨*proof*⟩

definition *restriction-open* :: ‹'a *set* ⇒ *bool*› (‹$open_{\downarrow}$›)
where ‹$open_{\downarrow}$ *U* ≡ ∀Σ∈*U*. ∀σ. σ $-\downarrow\rightarrow$ Σ ⟶ (∃ *n0*. ∀ *k*≥*n0*. σ *k* ∈ *U*)›

lemma *restriction-openI* : ‹(⋀Σ σ. Σ ∈ *U* ⟹ σ $-\downarrow\rightarrow$ Σ ⟹ ∃ *n0*. ∀ *k*≥*n0*. σ *k* ∈ *U*) ⟹ $open_{\downarrow}$ *U*›
⟨*proof*⟩

lemma *restriction-openD* : ‹$open_{\downarrow}$ *U* ⟹ Σ ∈ *U* ⟹ σ $-\downarrow\rightarrow$ Σ ⟹ ∃ *n0*. ∀ *k*≥*n0*. σ *k* ∈ *U*›
⟨*proof*⟩

lemma *restriction-openE* :
‹$open_{\downarrow}$ *U* ⟹ Σ ∈ *U* ⟹ σ $-\downarrow\rightarrow$ Σ ⟹ (⋀*n0*. (⋀*n*. *n0* ≤ *k* ⟹ σ *k* ∈ *U*) ⟹ *thesis*) ⟹ *thesis*›
⟨*proof*⟩

lemma *restriction-open-UNIV* [*simp*] : ‹$open_{\downarrow}$ *UNIV*›
and *restriction-open-empty* [*simp*] : ‹$open_{\downarrow}$ {}›
⟨*proof*⟩

lemma *restriction-open-union* :

‹$open_{\downarrow}\ U \Longrightarrow open_{\downarrow}\ V \Longrightarrow open_{\downarrow}\ (U \cup V)$›
⟨*proof*⟩

lemma *restriction-open-Union* :
‹$(\bigwedge i.\ i \in I \Longrightarrow open_{\downarrow}\ (U\ i)) \Longrightarrow open_{\downarrow}\ (\bigcup i{\in}I.\ U\ i)$›
⟨*proof*⟩

lemma *restriction-open-inter* :
‹$open_{\downarrow}\ (U \cap V)$› **if** ‹$open_{\downarrow}\ U$› **and** ‹$open_{\downarrow}\ V$›
⟨*proof*⟩

lemma *restriction-open-finite-Inter* :
‹*finite* $I \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow open_{\downarrow}\ (U\ i)) \Longrightarrow open_{\downarrow}\ (\bigcap i{\in}I.\ U\ i)$›
⟨*proof*⟩

definition *restriction-closed* :: ‹$'a\ set \Rightarrow bool$› (‹$closed_{\downarrow}$›)
where ‹$closed_{\downarrow}\ S \equiv open_{\downarrow}\ (-\ S)$›

lemma *restriction-closedI* : ‹$(\bigwedge\Sigma\ \sigma.\ \Sigma \notin S \Longrightarrow \sigma\ {-}{\downarrow}{\rightarrow}\ \Sigma \Longrightarrow \exists\, n0.\ \forall\, k{\geq}n0.\ \sigma\ k \notin S) \Longrightarrow closed_{\downarrow}\ S$›
⟨*proof*⟩

lemma *restriction-closedD* : ‹$closed_{\downarrow}\ S \Longrightarrow \Sigma \notin S \Longrightarrow \sigma\ {-}{\downarrow}{\rightarrow}\ \Sigma \Longrightarrow \exists\, n0.\ \forall\, k{\geq}n0.\ \sigma\ k \notin S$›
⟨*proof*⟩

lemma *restriction-closedE* :
‹$closed_{\downarrow}\ S \Longrightarrow \Sigma \notin S \Longrightarrow \sigma\ {-}{\downarrow}{\rightarrow}\ \Sigma \Longrightarrow (\bigwedge n0.\ (\bigwedge n.\ n0 \leq k \Longrightarrow \sigma\ k \notin S) \Longrightarrow thesis) \Longrightarrow thesis$›
⟨*proof*⟩

lemma *restriction-closed-UNIV* [*simp*] : ‹$closed_{\downarrow}\ UNIV$›
and *restriction-closed-empty* [*simp*] : ‹$closed_{\downarrow}\ \{\}$›
⟨*proof*⟩

end

5.2 Balls

context *restriction* **begin**

definition *restriction-cball* :: ‹$'a \Rightarrow nat \Rightarrow 'a\ set$› (‹$\mathcal{B}_{\downarrow}'(\text{-}, \text{-}')$›)
where ‹$\mathcal{B}_{\downarrow}(a,\ n) \equiv \{x.\ x \downarrow n = a \downarrow n\}$›

lemma *restriction-cball-mem-iff* : ‹$x \in \mathcal{B}_{\downarrow}(a,\ n) \longleftrightarrow\ x \downarrow n = a \downarrow n$›
and *restriction-cball-memI* : ‹$x \downarrow n = a \downarrow n \Longrightarrow x \in \mathcal{B}_{\downarrow}(a,\ n)$›
and *restriction-cball-memD* : ‹$x \in \mathcal{B}_{\downarrow}(a,\ n) \Longrightarrow x \downarrow n = a \downarrow n$›

⟨*proof*⟩

abbreviation (*iff*) *restriction-ball* :: ‹'*a* ⇒ *nat* ⇒ '*a set*›
where ‹*restriction-ball a n* ≡ $\mathcal{B}_{\downarrow}$(*a*, *Suc n*)›

lemma ‹*x* ∈ *restriction-ball a n* ⟷ *x* ↓ *Suc n* = *a* ↓ *Suc n*›
and ‹*x* ↓ *Suc n* = *a* ↓ *Suc n* ⟹ *x* ∈ *restriction-ball a n*›
and ‹*x* ∈ *restriction-ball a n* ⟹ *x* ↓ *Suc n* = *a* ↓ *Suc n*›
⟨*proof*⟩

lemma ‹*a* ∈ *restriction-ball a n*›
and *center-mem-restriction-cball* [*simp*] : ‹*a* ∈ $\mathcal{B}_{\downarrow}$(*a*, *n*)›
⟨*proof*⟩

lemma (**in** *restriction-space*) *restriction-cball-0-is-UNIV* [*simp*] :
‹$\mathcal{B}_{\downarrow}$(*a*, *0*) = *UNIV*› ⟨*proof*⟩

lemma *every-point-of-restriction-cball-is-centre* :
‹*b* ∈ $\mathcal{B}_{\downarrow}$(*a*, *n*) ⟹ $\mathcal{B}_{\downarrow}$(*a*, *n*) = $\mathcal{B}_{\downarrow}$(*b*, *n*)›
⟨*proof*⟩

lemma ‹*b* ∈ *restriction-ball a n* ⟹ *restriction-ball a n* = *restriction-ball b n*›
⟨*proof*⟩

definition *restriction-sphere* :: ‹'*a* ⇒ *nat* ⇒ '*a set*› (‹$\mathcal{S}_{\downarrow}$'(-, -')›)
where ‹$\mathcal{S}_{\downarrow}$(*a*, *n*) ≡ {*x*. *x* ↓ *n* = *a* ↓ *n* ∧ *x* ↓ *Suc n* ≠ *a* ↓ *Suc n*}›

lemma *restriction-sphere-mem-iff* : ‹*x* ∈ $\mathcal{S}_{\downarrow}$(*a*, *n*) ⟷ *x* ↓ *n* = *a* ↓ *n* ∧ *x* ↓ *Suc n* ≠ *a* ↓ *Suc n*›
and *restriction-sphere-memI* : ‹*x* ↓ *n* = *a* ↓ *n* ⟹ *x* ↓ *Suc n* ≠ *a* ↓ *Suc n* ⟹ *x* ∈ $\mathcal{S}_{\downarrow}$(*a*, *n*)›
and *restriction-sphere-memD1* : ‹*x* ∈ $\mathcal{S}_{\downarrow}$(*a*, *n*) ⟹ *x* ↓ *n* = *a* ↓ *n*›
and *restriction-sphere-memD2* : ‹*x* ∈ $\mathcal{S}_{\downarrow}$(*a*, *n*) ⟹ *x* ↓ *Suc n* ≠ *a* ↓ *Suc n*›
⟨*proof*⟩

lemma *restriction-sphere-is-diff* : ‹$\mathcal{S}_{\downarrow}$(*a*, *n*) = $\mathcal{B}_{\downarrow}$(*a*, *n*) − $\mathcal{B}_{\downarrow}$(*a*, *Suc n*)›
⟨*proof*⟩

lemma *restriction-open-restriction-cball* [*simp*] : ‹$open_{\downarrow}\ \mathcal{B}_{\downarrow}(a,\ n)$›
⟨*proof*⟩

lemma *restriction-closed-restriction-cball* [*simp*] : ‹$closed_{\downarrow}\ \mathcal{B}_{\downarrow}(a,\ n)$›
⟨*proof*⟩

lemma *restriction-open-Compl-iff* : ‹$open_{\downarrow}\ (-\ S) \longleftrightarrow closed_{\downarrow}\ S$›
⟨*proof*⟩

lemma *restriction-open-restriction-sphere* [*simp*] : ‹$open_{\downarrow}\ \mathcal{S}_{\downarrow}(a,\ n)$›
⟨*proof*⟩

lemma *restriction-closed-restriction-sphere* : ‹$closed_{\downarrow}\ \mathcal{S}_{\downarrow}(a,\ n)$›
⟨*proof*⟩

end

context *restriction-space* **begin**

lemma *restriction-cball-anti-mono* : ‹$n \leq m \Longrightarrow \mathcal{B}_{\downarrow}(a,\ m) \subseteq \mathcal{B}_{\downarrow}(a,\ n)$›
⟨*proof*⟩

lemma *inside-every-cball-iff-eq* : ‹$(\forall\, n.\ x \in \mathcal{B}_{\downarrow}(\Sigma,\ n)) \longleftrightarrow x = \Sigma$›
⟨*proof*⟩

lemma *Inf-many-inside-cball-iff-eq* : ‹$(\exists_{\infty} n.\ x \in \mathcal{B}_{\downarrow}(\Sigma,\ n)) \longleftrightarrow x = \Sigma$›
⟨*proof*⟩

lemma *Inf-many-inside-cball-imp-eq* : ‹$\exists_{\infty} n.\ x \in \mathcal{B}_{\downarrow}(\Sigma,\ n) \Longrightarrow x = \Sigma$›
⟨*proof*⟩

lemma *restriction-cballs-disjoint-or-subset* :
‹$\mathcal{B}_{\downarrow}(a,\ n) \cap \mathcal{B}_{\downarrow}(b,\ m) = \{\} \vee \mathcal{B}_{\downarrow}(a,\ n) \subseteq \mathcal{B}_{\downarrow}(b,\ m) \vee \mathcal{B}_{\downarrow}(b,\ m) \subseteq \mathcal{B}_{\downarrow}(a,\ n)$›
⟨*proof*⟩

lemma *equal-restriction-to-cball* :
‹$a \notin \mathcal{B}_{\downarrow}(b,\ n) \Longrightarrow x \in \mathcal{B}_{\downarrow}(b,\ n) \Longrightarrow y \in \mathcal{B}_{\downarrow}(b,\ n) \Longrightarrow x \downarrow k = a \downarrow k \longleftrightarrow y \downarrow k = a \downarrow k$›
⟨*proof*⟩

end

context *restriction* **begin**

lemma *restriction-tendsto-iff-restriction-cball-characterization* :
‹$\sigma\ {-}{\downarrow}{\rightarrow}\ \Sigma \longleftrightarrow (\forall\, n.\ \exists\, n0.\ \forall\, k{\geq}n0.\ \sigma\ k \in \mathcal{B}_{\downarrow}(\Sigma,\ n))$›
⟨*proof*⟩

corollary *restriction-tendsto-restriction-cballI* : ‹$(\bigwedge n.\ \exists\, n0.\ \forall\, k{\geq}n0.\ \sigma\ k \in \mathcal{B}_{\downarrow}(\Sigma,\ n)) \Longrightarrow \sigma\ {-}{\downarrow}{\rightarrow}\ \Sigma$›
⟨*proof*⟩

corollary *restriction-tendsto-restriction-cballD* : ‹$\sigma\ {-}{\downarrow}{\rightarrow}\ \Sigma \Longrightarrow \exists\, n0.\ \forall\, k{\geq}n0.\ \sigma\ k \in \mathcal{B}_{\downarrow}(\Sigma,\ n)$›
⟨*proof*⟩

corollary *restriction-tendsto-restriction-cballE* :
‹$\sigma\ {-}{\downarrow}{\rightarrow}\ \Sigma \Longrightarrow (\bigwedge n0.\ (\bigwedge k.\ n0 \leq k \Longrightarrow \sigma\ k \in \mathcal{B}_{\downarrow}(\Sigma,\ n)) \Longrightarrow \mathit{thesis}) \Longrightarrow \mathit{thesis}$›
⟨*proof*⟩

end

context *restriction* **begin**

theorem *restriction-closed-iff-sequential-characterization* :
‹$\mathit{closed}_{\downarrow}\ S \longleftrightarrow (\forall\, \Sigma\ \sigma.\ \mathit{range}\ \sigma \subseteq S \longrightarrow \sigma\ {-}{\downarrow}{\rightarrow}\ \Sigma \longrightarrow \Sigma \in S)$›
⟨*proof*⟩

corollary *restriction-closed-sequentialI* :
‹$(\bigwedge \Sigma\ \sigma.\ \mathit{range}\ \sigma \subseteq S \Longrightarrow \sigma\ {-}{\downarrow}{\rightarrow}\ \Sigma \Longrightarrow \Sigma \in S) \Longrightarrow \mathit{closed}_{\downarrow}\ S$›
⟨*proof*⟩

corollary *restriction-closed-sequentialD* :
‹$\mathit{closed}_{\downarrow}\ S \Longrightarrow \mathit{range}\ \sigma \subseteq S \Longrightarrow \sigma\ {-}{\downarrow}{\rightarrow}\ \Sigma \Longrightarrow \Sigma \in S$›
⟨*proof*⟩

end

context *restriction-space* **begin**

theorem *restriction-open-iff-restriction-cball-characterization* :
‹$open_{\downarrow}\ U \longleftrightarrow (\forall\, \Sigma{\in}U.\ \exists\, n.\ \mathcal{B}_{\downarrow}(\Sigma,\ n) \subseteq U)$›
⟨*proof*⟩

corollary *restriction-open-restriction-cballI* :
‹$(\bigwedge\Sigma.\ \Sigma \in U \Longrightarrow \exists\, n.\ \mathcal{B}_{\downarrow}(\Sigma,\ n) \subseteq U) \Longrightarrow open_{\downarrow}\ U$›
⟨*proof*⟩

corollary *restriction-open-restriction-cballD* :
‹$open_{\downarrow}\ U \Longrightarrow \Sigma \in U \Longrightarrow \exists\, n.\ \mathcal{B}_{\downarrow}(\Sigma,\ n) \subseteq U$›
⟨*proof*⟩

corollary *restriction-open-restriction-cballE* :
‹$open_{\downarrow}\ U \Longrightarrow \Sigma \in U \Longrightarrow (\bigwedge n.\ \mathcal{B}_{\downarrow}(\Sigma,\ n) \subseteq U \Longrightarrow thesis) \Longrightarrow thesis$›
⟨*proof*⟩

end

context *restriction* **begin**

definition *restriction-cont-on* :: ‹$['b :: restriction \Rightarrow 'a,\ 'b\ set] \Rightarrow bool$›
($\langle cont_{\downarrow}$ (-) *on* (-)› [*1000*, *1000*])
where ‹$cont_{\downarrow}\ f\ on\ A \equiv \forall\, \Sigma{\in}A.\ cont_{\downarrow}\ f\ at\ \Sigma$›

lemma *restriction-cont-onI* : ‹$(\bigwedge\Sigma\ \sigma.\ \Sigma \in A \Longrightarrow \sigma\ {-}{\downarrow}{\rightarrow}\ \Sigma \Longrightarrow (\lambda n.\ f\ (\sigma\ n))\ {-}{\downarrow}{\rightarrow}\ f\ \Sigma) \Longrightarrow cont_{\downarrow}\ f\ on\ A$›
⟨*proof*⟩

lemma *restriction-cont-onD* : ‹$cont_{\downarrow}\ f\ on\ A \Longrightarrow \Sigma \in A \Longrightarrow \sigma\ {-}{\downarrow}{\rightarrow}\ \Sigma \Longrightarrow (\lambda n.\ f\ (\sigma\ n))\ {-}{\downarrow}{\rightarrow}\ f\ \Sigma$›
⟨*proof*⟩

lemma *restriction-cont-on-comp* [*restriction-cont-simpset*] :
‹$cont_{\downarrow}\ f\ on\ A \Longrightarrow cont_{\downarrow}\ g\ on\ B \Longrightarrow f\ `\ A \subseteq B \Longrightarrow cont_{\downarrow}\ (\lambda x.\ g\ (f\ x))\ on\ A$›
⟨*proof*⟩

lemma *restriction-cont-on-if-then-else* [*restriction-cont-simpset*] :
‹$[\![\bigwedge x.\ P\ x \Longrightarrow cont_{\downarrow}\ (f\ x)\ on\ A;\ \bigwedge x.\ \neg\ P\ x \Longrightarrow cont_{\downarrow}\ (g\ x)\ on\ A]\!] \Longrightarrow cont_{\downarrow}\ (\lambda y.\ if\ P\ x\ then\ f\ x\ y\ else\ g\ x\ y)\ on\ A$›
⟨*proof*⟩

lemma *restriction-cont-on-subset* [*restriction-cont-simpset*] :

‹$cont_{\downarrow}\ f\ on\ B \Longrightarrow A \subseteq B \Longrightarrow cont_{\downarrow}\ f\ on\ A$›
⟨*proof*⟩

abbreviation *restriction-cont* :: ‹['b :: *restriction* ⇒ 'a] ⇒ *bool*› (‹$cont_{\downarrow}$›)
where ‹$cont_{\downarrow}\ f \equiv cont_{\downarrow}\ f\ on\ UNIV$›

lemma *restriction-contI* : ‹$(\bigwedge\Sigma\ \sigma.\ \sigma\ -\downarrow\rightarrow \Sigma \Longrightarrow (\lambda n.\ f\ (\sigma\ n))\ -\downarrow\rightarrow f\ \Sigma) \Longrightarrow cont_{\downarrow}\ f$›
⟨*proof*⟩

lemma *restriction-contD* : ‹$cont_{\downarrow}\ f \Longrightarrow \sigma\ -\downarrow\rightarrow \Sigma \Longrightarrow (\lambda n.\ f\ (\sigma\ n))\ -\downarrow\rightarrow f\ \Sigma$›
⟨*proof*⟩

lemma *restriction-cont-comp* [*restriction-cont-simpset*] :
‹$cont_{\downarrow}\ g \Longrightarrow cont_{\downarrow}\ f \Longrightarrow cont_{\downarrow}\ (\lambda x.\ g\ (f\ x))$›
⟨*proof*⟩

lemma *restriction-cont-if-then-else* [*restriction-cont-simpset*] :
‹$[\![\bigwedge x.\ P\ x \Longrightarrow cont_{\downarrow}\ (f\ x);\ \bigwedge x.\ \neg\ P\ x \Longrightarrow cont_{\downarrow}\ (g\ x)]\!] \Longrightarrow cont_{\downarrow}\ (\lambda y.\ if\ P\ x\ then\ f\ x\ y\ else\ g\ x\ y)$›
⟨*proof*⟩

end

context *restriction-space* **begin**

theorem *restriction-cont-at-iff-restriction-cball-characterization* :
‹$cont_{\downarrow}\ f\ at\ \Sigma \longleftrightarrow (\forall\, n.\ \exists\, k.\ f\ `\ \mathcal{B}_{\downarrow}(\Sigma,\ k) \subseteq \mathcal{B}_{\downarrow}(f\ \Sigma,\ n))$›
for *f* :: ‹'b :: *restriction-space* ⇒ 'a›
⟨*proof*⟩

corollary *restriction-cont-at-restriction-cballI* :
‹$(\bigwedge n.\ \exists\, k.\ f\ `\ \mathcal{B}_{\downarrow}(\Sigma,\ k) \subseteq \mathcal{B}_{\downarrow}(f\ \Sigma,\ n)) \Longrightarrow cont_{\downarrow}\ f\ at\ \Sigma$›
for *f* :: ‹'b :: *restriction-space* ⇒ 'a›
⟨*proof*⟩

corollary *restriction-cont-at-restriction-cballD* :
‹$cont_{\downarrow}\ f\ at\ \Sigma \Longrightarrow \exists\, k.\ f\ `\ \mathcal{B}_{\downarrow}(\Sigma,\ k) \subseteq \mathcal{B}_{\downarrow}(f\ \Sigma,\ n)$›
for *f* :: ‹'b :: *restriction-space* ⇒ 'a›
⟨*proof*⟩

corollary *restriction-cont-at-restriction-cballE* :
‹$cont_{\downarrow}\ f\ at\ \Sigma \Longrightarrow (\bigwedge k.\ f\ `\ \mathcal{B}_{\downarrow}(\Sigma,\ k) \subseteq \mathcal{B}_{\downarrow}(f\ \Sigma,\ n) \Longrightarrow thesis) \Longrightarrow thesis$›

for $f :: \langle 'b :: \mathit{restriction\text{-}space} \Rightarrow 'a \rangle$
⟨*proof*⟩

theorem *restriction-cont-iff-restriction-open-characterization* :
$\langle \mathit{cont}_{\downarrow}\ f \longleftrightarrow (\forall\, U.\ \mathit{open}_{\downarrow}\ U \longrightarrow \mathit{open}_{\downarrow}\ (f\ -`\ U)) \rangle$
for $f :: \langle 'b :: \mathit{restriction\text{-}space} \Rightarrow 'a \rangle$
⟨*proof*⟩

corollary *restriction-cont-restriction-openI* :
$\langle (\bigwedge U.\ \mathit{open}_{\downarrow}\ U \Longrightarrow \mathit{open}_{\downarrow}\ (f\ -`\ U)) \Longrightarrow \mathit{cont}_{\downarrow}\ f \rangle$
for $f :: \langle 'b :: \mathit{restriction\text{-}space} \Rightarrow 'a \rangle$
⟨*proof*⟩

corollary *restriction-cont-restriction-openD* :
$\langle \mathit{cont}_{\downarrow}\ f \Longrightarrow \mathit{open}_{\downarrow}\ U \Longrightarrow \mathit{open}_{\downarrow}\ (f\ -`\ U) \rangle$
for $f :: \langle 'b :: \mathit{restriction\text{-}space} \Rightarrow 'a \rangle$
⟨*proof*⟩

theorem *restriction-cont-iff-restriction-closed-characterization* :
$\langle \mathit{cont}_{\downarrow}\ f \longleftrightarrow (\forall\, S.\ \mathit{closed}_{\downarrow}\ S \longrightarrow \mathit{closed}_{\downarrow}\ (f\ -`\ S)) \rangle$
for $f :: \langle 'b :: \mathit{restriction\text{-}space} \Rightarrow 'a \rangle$
⟨*proof*⟩

corollary *restriction-cont-restriction-closedI* :
$\langle (\bigwedge U.\ \mathit{closed}_{\downarrow}\ U \Longrightarrow \mathit{closed}_{\downarrow}\ (f\ -`\ U)) \Longrightarrow \mathit{cont}_{\downarrow}\ f \rangle$
for $f :: \langle 'b :: \mathit{restriction\text{-}space} \Rightarrow 'a \rangle$
⟨*proof*⟩

corollary *restriction-cont-restriction-closedD* :
$\langle \mathit{cont}_{\downarrow}\ f \Longrightarrow \mathit{closed}_{\downarrow}\ U \Longrightarrow \mathit{closed}_{\downarrow}\ (f\ -`\ U) \rangle$
for $f :: \langle 'b :: \mathit{restriction\text{-}space} \Rightarrow 'a \rangle$
⟨*proof*⟩

theorem *restriction-shift-on-restriction-open-imp-restriction-cont-on* :
$\langle \mathit{cont}_{\downarrow}\ f\ \mathit{on}\ U \rangle$ **if** $\langle \mathit{open}_{\downarrow}\ U \rangle$ **and** $\langle \mathit{restriction\text{-}shift\text{-}on}\ f\ k\ U \rangle$
⟨*proof*⟩

corollary *restriction-shift-imp-restriction-cont* [*restriction-cont-simpset*]
:
$\langle \mathit{restriction\text{-}shift}\ f\ k \Longrightarrow \mathit{cont}_{\downarrow}\ f \rangle$
⟨*proof*⟩

corollary *non-too-destructive-imp-restriction-cont* [*restriction-cont-simpset*]
:
$\langle \mathit{non\text{-}too\text{-}destructive}\ f \Longrightarrow \mathit{cont}_{\downarrow}\ f \rangle$

⟨*proof*⟩

end

5.3 Compactness

context *restriction* **begin**

definition *restriction-compact* :: ‹'*a set* ⇒ *bool*› (‹*compact*$_\downarrow$›)
where ‹*compact*$_\downarrow$ $K \equiv$
$\forall \sigma.$ *range* $\sigma \subseteq K \longrightarrow$
$(\exists f ::$ *nat* ⇒ *nat*. $\exists \Sigma.\ \Sigma \in K \wedge$ *strict-mono* $f \wedge (\sigma \circ f) -\downarrow\rightarrow \Sigma)$›

lemma *restriction-compactI* :
‹$(\bigwedge \sigma.$ *range* $\sigma \subseteq K \Longrightarrow \exists f ::$ *nat* ⇒ *nat*. $\exists \Sigma.\ \Sigma \in K \wedge$ *strict-mono* $f \wedge (\sigma \circ f) -\downarrow\rightarrow \Sigma)$
$\Longrightarrow$ *compact*$_\downarrow$ K› ⟨*proof*⟩

lemma *restriction-compactD* :
‹*compact*$_\downarrow$ $K \Longrightarrow$ *range* $\sigma \subseteq K \Longrightarrow$
$\exists f ::$ *nat* ⇒ *nat*. $\exists \Sigma.\ \Sigma \in K \wedge$ *strict-mono* $f \wedge (\sigma \circ f) -\downarrow\rightarrow \Sigma$›
⟨*proof*⟩

lemma *restriction-compactE* :
assumes ‹*compact*$_\downarrow$ K› **and** ‹*range* $\sigma \subseteq K$›
obtains $f ::$ ‹*nat* ⇒ *nat*› **and** Σ **where** ‹$\Sigma \in K$› ‹*strict-mono* f› ‹$(\sigma \circ f) -\downarrow\rightarrow \Sigma$›
⟨*proof*⟩

lemma *restriction-compact-empty* [*simp*] : ‹*compact*$_\downarrow$ {}›
⟨*proof*⟩

lemma (**in** *restriction-space*) *restriction-compact-imp-restriction-closed* :
‹*closed*$_\downarrow$ K› **if** ‹*compact*$_\downarrow$ K›
⟨*proof*⟩

lemma *restriction-compact-union* : ‹*compact*$_\downarrow$ $(K \cup L)$›
if ‹*compact*$_\downarrow$ K› **and** ‹*compact*$_\downarrow$ L›
⟨*proof*⟩

lemma *restriction-compact-finite-Union* :
‹$[\![$*finite* I; $\bigwedge i.\ i \in I \Longrightarrow$ *compact*$_\downarrow$ $(K\ i)]\!] \Longrightarrow$ *compact*$_\downarrow$ $(\bigcup i \in I.\ K$

i)›
⟨*proof*⟩

lemma (**in** *restriction-space*) *restriction-compact-Inter* :
‹$compact_{\downarrow}$ ($\bigcap i.\ K\ i$)› **if** ‹$\bigwedge i.\ compact_{\downarrow}\ (K\ i)$›
⟨*proof*⟩

lemma *finite-imp-restriction-compact* : ‹$compact_{\downarrow}\ K$› **if** ‹*finite* K›
⟨*proof*⟩

lemma *restriction-compact-restriction-closed-subset* : ‹$compact_{\downarrow}\ L$›
if ‹$L \subseteq K$› ‹$compact_{\downarrow}\ K$› ‹$closed_{\downarrow}\ L$›
⟨*proof*⟩

lemma *restriction-cont-image-of-restriction-compact* :
‹$compact_{\downarrow}\ (f\ `\ K)$› **if** ‹$compact_{\downarrow}\ K$› **and** ‹$cont_{\downarrow}\ f$ *on* K›
⟨*proof*⟩

end

5.4 Properties for Function and Product

lemma *restriction-cball-fun-is* : ‹$\mathcal{B}_{\downarrow}(f,\ n) = \{g.\ \forall x.\ g\ x \in \mathcal{B}_{\downarrow}(f\ x,\ n)\}$›
⟨*proof*⟩

lemma *restriction-cball-prod-is* :
‹$\mathcal{B}_{\downarrow}(\Sigma,\ n) = \mathcal{B}_{\downarrow}(fst\ \Sigma,\ n) \times \mathcal{B}_{\downarrow}(snd\ \Sigma,\ n)$›
⟨*proof*⟩

lemma *restriction-open-prod-imp-restriction-open-image-fst* :
‹$open_{\downarrow}\ (fst\ `\ U)$› **if** ‹$open_{\downarrow}\ U$›
⟨*proof*⟩

lemma *restriction-open-prod-imp-restriction-open-image-snd* :
‹$open_{\downarrow}\ (snd\ `\ U)$› **if** ‹$open_{\downarrow}\ U$›
⟨*proof*⟩

lemma *restriction-open-prod-iff* :
‹$open_{\downarrow}\ (U \times V) \longleftrightarrow (V = \{\} \vee open_{\downarrow}\ U) \wedge (U = \{\} \vee open_{\downarrow}\ V)$›
⟨*proof*⟩

lemma *restriction-cont-at-prod-codomain-iff*:
‹$cont_{\downarrow}\ f\ at\ \Sigma \longleftrightarrow cont_{\downarrow}\ (\lambda x.\ fst\ (f\ x))\ at\ \Sigma \wedge cont_{\downarrow}\ (\lambda x.\ snd\ (f\ x))$

at Σ›
⟨*proof*⟩

lemma *restriction-cont-on-prod-codomain-iff*:
‹$cont_{\downarrow}$ f *on* $A \longleftrightarrow$ $cont_{\downarrow}$ $(\lambda x.$ *fst* $(f\ x))$ *on* $A \wedge$ $cont_{\downarrow}$ $(\lambda x.$ *snd* $(f\ x))$ *on* A›
⟨*proof*⟩

lemma *restriction-cont-prod-codomain-iff*:
‹$cont_{\downarrow}$ $f \longleftrightarrow$ $cont_{\downarrow}$ $(\lambda x.$ *fst* $(f\ x)) \wedge$ $cont_{\downarrow}$ $(\lambda x.$ *snd* $(f\ x))$›
⟨*proof*⟩

lemma *restriction-cont-at-prod-codomain-imp* [*restriction-cont-simpset*]
:
‹$cont_{\downarrow}$ f *at* $\Sigma \Longrightarrow$ $cont_{\downarrow}$ $(\lambda x.$ *fst* $(f\ x))$ *at* Σ›
‹$cont_{\downarrow}$ f *at* $\Sigma \Longrightarrow$ $cont_{\downarrow}$ $(\lambda x.$ *snd* $(f\ x))$ *at* Σ›
⟨*proof*⟩

lemma *restriction-cont-on-prod-codomain-imp* [*restriction-cont-simpset*]
:
‹$cont_{\downarrow}$ f *on* $A \Longrightarrow$ $cont_{\downarrow}$ $(\lambda x.$ *fst* $(f\ x))$ *on* A›
‹$cont_{\downarrow}$ f *on* $A \Longrightarrow$ $cont_{\downarrow}$ $(\lambda x.$ *snd* $(f\ x))$ *on* A›
⟨*proof*⟩

lemma *restriction-cont-prod-codomain-imp* [*restriction-cont-simpset*]
:
‹$cont_{\downarrow}$ $f \Longrightarrow$ $cont_{\downarrow}$ $(\lambda x.$ *fst* $(f\ x))$›
‹$cont_{\downarrow}$ $f \Longrightarrow$ $cont_{\downarrow}$ $(\lambda x.$ *snd* $(f\ x))$›
⟨*proof*⟩

lemma *restriction-cont-at-fun-imp* [*restriction-cont-simpset*] :
‹$cont_{\downarrow}$ f *at* $A \Longrightarrow$ $cont_{\downarrow}$ $(\lambda x.\ f\ x\ y)$ *at* A›
⟨*proof*⟩

lemma *restriction-cont-on-fun-imp* [*restriction-cont-simpset*] :
‹$cont_{\downarrow}$ f *on* $A \Longrightarrow$ $cont_{\downarrow}$ $(\lambda x.\ f\ x\ y)$ *on* A›
⟨*proof*⟩

corollary *restriction-cont-fun-imp* [*restriction-cont-simpset*] :
‹$cont_{\downarrow}$ $f \Longrightarrow$ $cont_{\downarrow}$ $(\lambda x.\ f\ x\ y)$›
⟨*proof*⟩

lemma *restriction-cont-at-prod-domain-imp* [*restriction-cont-simpset*]

:
‹$cont_{\downarrow}\ f\ at\ \Sigma \Longrightarrow cont_{\downarrow}\ (\lambda x.\ f\ (x,\ snd\ \Sigma))\ at\ (fst\ \Sigma)$›
‹$cont_{\downarrow}\ f\ at\ \Sigma \Longrightarrow cont_{\downarrow}\ (\lambda y.\ f\ (fst\ \Sigma,\ y))\ at\ (snd\ \Sigma)$›
for f :: ‹$'a$ :: *restriction-space* × $'b$ :: *restriction-space* ⇒ $'c$ :: *restriction-space*›
⟨*proof*⟩

lemma *restriction-cont-on-prod-domain-imp* [*restriction-cont-simpset*]
:
‹$cont_{\downarrow}\ (\lambda x.\ f\ (x,\ y))\ on\ \{x.\ (x,\ y) \in A\}$›
‹$cont_{\downarrow}\ (\lambda y.\ f\ (x,\ y))\ on\ \{y.\ (x,\ y) \in A\}$› **if** ‹$cont_{\downarrow}\ f\ on\ A$›
for f :: ‹$'a$ :: *restriction-space* × $'b$ :: *restriction-space* ⇒ $'c$ :: *restriction-space*›
⟨*proof*⟩

lemma *restriction-cont-prod-domain-imp* [*restriction-cont-simpset*] :
‹$cont_{\downarrow}\ f \Longrightarrow cont_{\downarrow}\ (\lambda x.\ f\ (x,\ y))$›
‹$cont_{\downarrow}\ f \Longrightarrow cont_{\downarrow}\ (\lambda y.\ f\ (x,\ y))$›
for f :: ‹$'a$ :: *restriction-space* × $'b$ :: *restriction-space* ⇒ $'c$ :: *restriction-space*›
⟨*proof*⟩

6 Induction in Restriction Space

6.1 Admissibility

named-theorems *restriction-adm-simpset* — For future automation.

6.1.1 Definition

We start by defining the notion of admissible predicate. The idea is that if this predicates holds for each value of a convergent sequence, it also holds for its limit.

context *restriction* **begin**

definition *restriction-adm* :: ‹($'a$ ⇒ *bool*) ⇒ *bool*› (‹$adm_{\downarrow}$›)
where ‹*restriction-adm* $P \equiv \forall \sigma\ \Sigma.\ \sigma\ -\downarrow\rightarrow \Sigma \longrightarrow (\forall n.\ P\ (\sigma\ n)) \longrightarrow P\ \Sigma$›

lemma *restriction-admI* :
‹$(\bigwedge \sigma\ \Sigma.\ \sigma\ -\downarrow\rightarrow \Sigma \Longrightarrow (\bigwedge n.\ P\ (\sigma\ n)) \Longrightarrow P\ \Sigma) \Longrightarrow$ *restriction-adm* P›
⟨*proof*⟩

lemma *restriction-admD* :
‹$[\![$*restriction-adm* P; $\sigma\ -\downarrow\rightarrow \Sigma$; $\bigwedge n.\ P\ (\sigma\ n)]\!] \Longrightarrow P\ \Sigma$›
⟨*proof*⟩

6.1.2 Properties

lemma *restriction-adm-const* [*restriction-adm-simpset*] :
‹$adm_{\downarrow}\ (\lambda x.\ t)$›
⟨*proof*⟩

lemma *restriction-adm-conj* [*restriction-adm-simpset*] :
‹$adm_{\downarrow}\ (\lambda x.\ P\ x) \Longrightarrow adm_{\downarrow}\ (\lambda x.\ Q\ x) \Longrightarrow adm_{\downarrow}\ (\lambda x.\ P\ x \wedge Q\ x)$›
⟨*proof*⟩

lemma *restriction-adm-all* [*restriction-adm-simpset*] :
‹$(\bigwedge y.\ adm_{\downarrow}\ (\lambda x.\ P\ x\ y)) \Longrightarrow adm_{\downarrow}\ (\lambda x.\ \forall y.\ P\ x\ y)$›
⟨*proof*⟩

lemma *restriction-adm-ball* [*restriction-adm-simpset*] :
‹$(\bigwedge y.\ y \in A \Longrightarrow adm_{\downarrow}\ (\lambda x.\ P\ x\ y)) \Longrightarrow adm_{\downarrow}\ (\lambda x.\ \forall y{\in}A.\ P\ x\ y)$›
⟨*proof*⟩

lemma *restriction-adm-disj* [*restriction-adm-simpset*] :
‹$adm_{\downarrow}\ (\lambda x.\ P\ x \vee Q\ x)$› **if** ‹$adm_{\downarrow}\ (\lambda x.\ P\ x)$› ‹$adm_{\downarrow}\ (\lambda x.\ Q\ x)$›
⟨*proof*⟩

lemma *restriction-adm-imp* [*restriction-adm-simpset*] :
‹$adm_{\downarrow}\ (\lambda x.\ \neg\ P\ x) \Longrightarrow adm_{\downarrow}\ (\lambda x.\ Q\ x) \Longrightarrow adm_{\downarrow}\ (\lambda x.\ P\ x \longrightarrow Q\ x)$›
⟨*proof*⟩

lemma *restriction-adm-iff* [*restriction-adm-simpset*] :
‹$adm_{\downarrow}\ (\lambda x.\ P\ x \longrightarrow Q\ x) \Longrightarrow adm_{\downarrow}\ (\lambda x.\ Q\ x \longrightarrow P\ x) \Longrightarrow adm_{\downarrow}\ (\lambda x.\ P\ x \longleftrightarrow Q\ x)$›
⟨*proof*⟩

lemma *restriction-adm-if-then-else* [*restriction-adm-simpset*]:
‹$[\![P \Longrightarrow adm_{\downarrow}\ (\lambda x.\ Q\ x);\ \neg\ P \Longrightarrow adm_{\downarrow}\ (\lambda x.\ R\ x)]\!] \Longrightarrow$
$adm_{\downarrow}\ (\lambda x.\ \mathit{if}\ P\ \mathit{then}\ Q\ x\ \mathit{else}\ R\ x)$›
⟨*proof*⟩

end

The notion of continuity is of course strongly related to the notion of admissibility.

lemma *restriction-adm-eq* [*restriction-adm-simpset*] :
‹$adm_{\downarrow}\ (\lambda x.\ f\ x = g\ x)$› **if** ‹$cont_{\downarrow}\ f$› **and** ‹$cont_{\downarrow}\ g$›
for $f\ g$ :: ‹$'a$ :: *restriction* ⇒ $'b$ :: *restriction-space*›
⟨*proof*⟩

lemma *restriction-adm-subst* [*restriction-adm-simpset*] :
‹$adm_{\downarrow}\ (\lambda x.\ P\ (t\ x))$› **if** ‹$cont_{\downarrow}\ (\lambda x.\ t\ x)$› **and** ‹$adm_{\downarrow}\ P$›
⟨*proof*⟩

lemma *restriction-adm-prod-domainD* [*restriction-adm-simpset*] :
‹$adm_{\downarrow}$ ($\lambda x.$ P (x, y))› **and** ‹$adm_{\downarrow}$ ($\lambda y.$ P (x, y))› **if** ‹$adm_{\downarrow}$ P›
⟨*proof*⟩

lemma *restriction-adm-restriction-shift-on* [*restriction-adm-simpset*] :
‹$adm_{\downarrow}$ ($\lambda f.$ *restriction-shift-on* f k A)›
⟨*proof*⟩

lemma *restriction-adm-constructive-on* [*restriction-adm-simpset*] :
‹$adm_{\downarrow}$ ($\lambda f.$ *constructive-on* f A)›
⟨*proof*⟩

lemma *restriction-adm-non-destructive-on* [*restriction-adm-simpset*] :
‹$adm_{\downarrow}$ ($\lambda f.$ *non-destructive-on* f A)›
⟨*proof*⟩

lemma *restriction-adm-restriction-cont-at* [*restriction-adm-simpset*] :
‹$adm_{\downarrow}$ ($\lambda f.$ $cont_{\downarrow}$ f *at* a)›
⟨*proof*⟩

lemma *restriction-adm-restriction-cont-on* [*restriction-adm-simpset*] :
‹$adm_{\downarrow}$ ($\lambda f.$ $cont_{\downarrow}$ f *on* A)›
⟨*proof*⟩

corollary *restriction-adm-restriction-shift* [*restriction-adm-simpset*] :
‹$adm_{\downarrow}$ ($\lambda f.$ *restriction-shift* f k)›
and *restriction-adm-constructive* [*restriction-adm-simpset*] :
‹$adm_{\downarrow}$ ($\lambda f.$ *constructive* f)›
and *restriction-adm-non-destructive* [*restriction-adm-simpset*] :
‹$adm_{\downarrow}$ ($\lambda f.$ *non-destructive* f)›
and *restriction-adm-restriction-cont* [*restriction-adm-simpset*] :
‹$adm_{\downarrow}$ ($\lambda f.$ $cont_{\downarrow}$ f)›
⟨*proof*⟩

lemma (**in** *restriction*) *restriction-adm-mem-restriction-closed* [*restriction-adm-simpset*]
:
‹$closed_{\downarrow}$ $K \Longrightarrow adm_{\downarrow}$ ($\lambda x.$ $x \in K$)›
⟨*proof*⟩

lemma (**in** *restriction-space*) *restriction-adm-mem-restriction-compact* [*restriction-adm-simpset*] :
‹$compact_{\downarrow}\ K \Longrightarrow adm_{\downarrow}\ (\lambda x.\ x \in K)$›
⟨*proof*⟩

lemma (**in** *restriction-space*) *restriction-adm-mem-finite* [*restriction-adm-simpset*] :
‹*finite* $S \Longrightarrow adm_{\downarrow}\ (\lambda x.\ x \in S)$›
⟨*proof*⟩

lemma *restriction-adm-restriction-tendsto* [*restriction-adm-simpset*] :
‹$adm_{\downarrow}\ (\lambda\sigma.\ \sigma\ -\!\!\downarrow\!\!\rightarrow \Sigma)$›
⟨*proof*⟩

lemma *restriction-adm-lim* [*restriction-adm-simpset*] :
‹$adm_{\downarrow}\ (\lambda\Sigma.\ \sigma\ -\!\!\downarrow\!\!\rightarrow \Sigma)$›
⟨*proof*⟩

lemma *restriction-restriction-cont-on* [*restriction-cont-simpset*] :
‹$cont_{\downarrow}\ f\ on\ A \Longrightarrow cont_{\downarrow}\ (\lambda x.\ f\ x \downarrow n)\ on\ A$›
⟨*proof*⟩

lemma *restriction-cont-on-id* [*restriction-cont-simpset*] : ‹$cont_{\downarrow}\ (\lambda x.\ x)\ on\ A$›
⟨*proof*⟩

lemma *restriction-cont-on-const* [*restriction-cont-simpset*] : ‹$cont_{\downarrow}\ (\lambda x.\ c)\ on\ A$›
⟨*proof*⟩

lemma *restriction-cont-on-fun* [*restriction-cont-simpset*] : ‹$cont_{\downarrow}\ (\lambda f.\ f\ x)\ on\ A$›
⟨*proof*⟩

lemma *restriction-cont2cont-on-fun* [*restriction-cont-simpset*] :
‹$cont_{\downarrow}\ f\ on\ A \Longrightarrow cont_{\downarrow}\ (\lambda x.\ f\ x\ y)\ on\ A$›
⟨*proof*⟩

6.2 Induction

Now that we have the concept of admissibility, we can formalize an induction rule for fixed points. Considering a *constructive* function f of type $'a \Rightarrow 'a$ (where $'a$ is instance of the class *complete-restriction-space*) and a predicate P which is admissible, and assuming that :

- P holds for a certain element x

- for any element x, if P holds for x then it still holds for $f\ x$ we can have that P holds for the fixed point $\upsilon\ x.\ P\ x$.

lemma *restriction-fix-ind'* [*case-names constructive adm steps*] :
‹*constructive f* ⟹ *adm*$_{\downarrow}$ *P* ⟹ (⋀*n*. *P* ((*f* ^^ *n*) *x*)) ⟹ *P* (*υ x*. *f x*)›
⟨*proof*⟩

lemma *restriction-fix-ind* [*case-names constructive adm base step*] :
‹*P* (*υ x*. *f x*)› **if** ‹*constructive f*› ‹*adm*$_{\downarrow}$ *P*› ‹*P x*› ‹⋀*x*. *P x* ⟹ *P* (*f x*)›
⟨*proof*⟩

lemma *restriction-fix-ind2* [*case-names constructive adm base0 base1 step*] :
‹*P* (*υ x*. *f x*)› **if** ‹*constructive f*› ‹*adm*$_{\downarrow}$ *P*› ‹*P x*› ‹*P* (*f x*)›
‹⋀*x*. ⟦*P x*; *P* (*f x*)⟧ ⟹ *P* (*f* (*f x*))›
⟨*proof*⟩

We can rewrite the fixed point over a product to obtain this parallel fixed point induction rule.

lemma *parallel-restriction-fix-ind* [*case-names constructiveL constructiveR adm base step*] :
fixes *f* :: ‹'*a* :: *complete-restriction-space* ⇒ '*a*›
and *g* :: ‹'*b* :: *complete-restriction-space* ⇒ '*b*›
assumes *constructive* : ‹*constructive f*› ‹*constructive g*›
and *adm* : ‹*restriction-adm* (λ*p*. *P* (*fst p*) (*snd p*))›
and *base* : ‹*P x y*› **and** *step* : ‹⋀*x y*. *P x y* ⟹ *P* (*f x*) (*g y*)›
shows ‹*P* (*υ x*. *f x*) (*υ y*. *g y*)›
⟨*proof*⟩

k-steps induction

lemma *restriction-fix-ind-k-steps* [*case-names constructive adm base-k-steps step*] :
assumes ‹*constructive f*›
and ‹*adm*$_{\downarrow}$ *P*›
and ‹∀ *i* < *k*. *P* ((*f* ^^ *i*) *x*)›
and ‹⋀*x*. ∀ *i* < *k*. *P* ((*f* ^^ *i*) *x*) ⟹ *P* ((*f* ^^ *k*) *x*)›
shows ‹*P* (*υ x*. *f x*)›
⟨*proof*⟩

7 Entry Point

This is the file `Restriction_Spaces` should be imported from.

declare

restriction-shift-introset [*intro*!]
restriction-shift-simpset [*simp*]
restriction-cont-simpset [*simp*]
restriction-adm-simpset [*simp*]

We already have *non-destructive* ($\lambda x.\ x$), and can easily notice *non-destructive* ($\lambda f.\ f\ x$), but also *non-destructive* ($\lambda f.\ f\ x\ y$), etc. We add a **simproc-setup** to enable the simplifier to automatically handle goals of this form, regardless of the number of arguments on which the function is applied.

⟨*ML*⟩

lemma ‹*non-destructive* ($\lambda f.\ f\ a\ b\ c\ d\ e\ f'\ g\ h\ i\ j\ k\ l\ m\ n\ o'\ p\ q\ r\ s\ t\ u\ v\ w\ x\ y\ z$)›
⟨*proof*⟩