Restriction_Spaces: a Fixed-Point Theory

Benoît Ballenghien Benjamin Puyobro Burkhart Wolff

September 1, 2025

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 **1**
- 1.1 Basic Notions for Restriction . . . 2
- 1.2 Restriction shift Maps . . . 6
  - 1.2.1 Definition . . . 7
  - 1.2.2 Three particular Cases . . . 8
  - 1.2.3 Properties . . . 12

2 Class Implementation **19**
- 2.1 Preliminaries . . . 19
- 2.2 Basic Notions for Restriction . . . 20
- 2.3 Definition of the Fixed-Point Operator . . . 24

2.3.1 Preliminaries . . . 24
2.3.2 Fixed-Point Operator . . . 31

3 Product over Restriction Spaces 35
3.1 Restriction Space . . . 35
3.2 Restriction shift Maps . . . 36
3.2.1 Domain is a Product . . . 36
3.2.2 Codomain is a Product . . . 39
3.3 Limits and Convergence . . . 40
3.4 Completeness . . . 41
3.5 Fixed Point . . . 41

4 Functions towards a Restriction Space 43
4.1 Restriction Space . . . 43
4.2 Restriction shift Maps . . . 44
4.3 Limits and Convergence . . . 46
4.4 Completeness . . . 47

5 Topological Notions 48
5.1 Continuity . . . 48
5.2 Balls . . . 50
5.3 Compactness . . . 59
5.4 Properties for Function and Product . . . 62

6 Induction in Restriction Space 66
6.1 Admissibility . . . 66
6.1.1 Definition . . . 66
6.1.2 Properties . . . 66
6.2 Induction . . . 71

7 Entry Point 73

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** ‹≲› *50*)
  assumes *restriction-restriction* [*simp*] : ‹*x* ↓ *n* ↓ *m* = *x* ↓ *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*›
by *blast*

lemma *disjoint-restriction-related-set-restriction-not-related-set* :
‹*restriction-related-set x y* ∩ *restriction-not-related-set x y* = {}› **by**
blast

lemma ‹*bdd-below* (*restriction-related-set x y*)› **by** (*fact bdd-below-bot*)

lemma ‹*bdd-below* (*restriction-not-related-set x y*)› **by** (*fact bdd-below-bot*)

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*›
using *restriction-0-related* **by** *blast*

lemma *all-restriction-related-iff-related* : ‹(∀ *n*. *x* ↓ *n* ≲≈ *y* ↓ *n*) ⟷ *x* ≲≈ *y*›
using *mono-restriction-related ex-not-restriction-related* **by** *blast*

lemma *restriction-related-le* : ‹*x* ↓ *n* ≲≈ *y* ↓ *n*› **if** ‹*n* ≤ *m*› **and** ‹*x* ↓ *m* ≲≈ *y* ↓ *m*›
proof −
from *mono-restriction-related*[*OF* ‹*x* ↓ *m* ≲≈ *y* ↓ *m*›] **have** ‹*x* ↓ *m* ↓ *n* ≲≈ *y* ↓ *m* ↓ *n*› **.**
also have ‹(λ*x*. *x* ↓ *m* ↓ *n*) = (λ*x*. *x* ↓ *n*)› **by** (*simp add*: ‹*n* ≤ *m*›)
finally show ‹*x* ↓ *n* ≲≈ *y* ↓ *n*› **.**
qed

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

by (*metis le-add2 plus-1-eq-Suc restriction-related-le*)

lemma *all-ge-restriction-related-iff-related* : ‹(∀ n≥m. x ↓ n ≲≈ y ↓ n) ⟷ x ≲≈ y›
by (*metis all-restriction-related-iff-related nle-le restriction-related-le*)

lemma *take-lemma-restriction* : ‹x ≲≈ y›
if ‹⋀n. ⟦⋀k. k ≤ n ⟹ x ↓ k ≲≈ y ↓ k⟧ ⟹ x ↓ Suc n ≲≈ y ↓ Suc n›
proof (*subst all-restriction-related-iff-related*[*symmetric*], *intro allI*)
show ‹x ↓ n ≲≈ y ↓ n› **for** *n*
by (*induct n rule*: *full-nat-induct*)
(*metis not-less-eq-eq restriction-0-related restriction-related-le that zero-induct*)
qed

lemma *ex-not-restriction-related-optimized* :
‹∃!n. ¬ x ↓ Suc n ≲≈ y ↓ Suc n ∧ (∀ m≤n. x ↓ m ≲≈ y ↓ m)› **if** ‹¬ x ≲≈ y›
proof (*rule ex1I*)
let *?S* = ‹{n. ¬ x ↓ Suc n ≲≈ y ↓ Suc n ∧ (∀ m≤n. x ↓ m ≲≈ y ↓ m)}›
let *?n* = ‹*Inf* {n. ¬ x ↓ Suc n ≲≈ y ↓ Suc n ∧ (∀ m≤n. x ↓ m ≲≈ y ↓ m)}›
from *restriction-related-le*[*of - - x y*] *take-lemma-restriction*[*of x y*] ‹¬ x ≲≈ y›
have ‹*?S* ≠ {}› **by** *auto*
from *Inf-nat-def1*[*OF this*] **have** ‹*?n* ∈ *?S*› **.**
thus ‹¬ x ↓ Suc *?n* ≲≈ y ↓ Suc *?n* ∧ (∀ m≤*?n*. x ↓ m ≲≈ y ↓ m)› **by** *blast*
thus ‹¬ x ↓ Suc n ≲≈ y ↓ Suc n ∧ (∀ m≤n. x ↓ m ≲≈ y ↓ m) ⟹ n = *?n*› **for** *n*
by (*meson not-less-eq-eq order-antisym-conv*)
qed

lemma *nonempty-restriction-related-set* : ‹*restriction-related-set x y* ≠ {}›
using *restriction-0-related* **by** *blast*

lemma *non-UNIV-restriction-not-related-set* : ‹*restriction-not-related-set x y* ≠ *UNIV*›
using *restriction-0-related* **by** *blast*

lemma *UNIV-restriction-related-set-iff* : ‹*restriction-related-set x y* = *UNIV* ⟷ x ≲≈ y›
using *all-restriction-related-iff-related* **by** *blast*

lemma *empty-restriction-not-related-set-iff*: ‹*restriction-not-related-set x y* = {} ⟷ *x* ≲≈ *y*›
by (*simp add*: *all-restriction-related-iff-related*)

lemma *finite-restriction-related-set-iff* :
‹*finite* (*restriction-related-set x y*) ⟷ ¬ *x* ≲≈ *y*›
proof (*rule iffI*)
assume ‹*finite* (*restriction-related-set x y*)›
obtain *n* **where** ‹¬ *x* ↓ *n* ≲≈ *y* ↓ *n*›
using ‹*finite* (*restriction-related-set x y*)› **by** *fastforce*
with *mono-restriction-related* **show** ‹¬ *x* ≲≈ *y*› **by** *blast*
next
assume ‹¬ *x* ≲≈ *y*›
then obtain *n* **where** ‹∀ *m*>*n*. ¬ *x* ↓ *m* ≲≈ *y* ↓ *m*›
by (*meson all-restriction-related-iff-related less-le-not-le restriction-related-le*)
hence ‹*restriction-related-set x y* ⊆ {*0*..*n*}›
by (*simp add*: *subset-iff*) (*meson linorder-not-le*)
thus ‹*finite* (*restriction-related-set x y*)›
by (*simp add*: *subset-eq-atLeast0-atMost-finite*)
qed

lemma *infinite-restriction-not-related-set-iff* :
‹*infinite* (*restriction-not-related-set x y*) ⟷ ¬ *x* ≲≈ *y*›
by (*metis empty-restriction-not-related-set-iff finite-restriction-related-set-iff finite.emptyI finite-Collect-not infinite-UNIV-char-0*)

lemma *bdd-above-restriction-related-set-iff* :
‹*bdd-above* (*restriction-related-set x y*) ⟷ ¬ *x* ≲≈ *y*›
by (*simp add*: *bdd-above-nat finite-restriction-related-set-iff*)

context fixes *x y* **assumes** ‹¬ *x* ≲≈ *y*› **begin**

lemma *Sup-in-restriction-related-set* :
‹*Sup* (*restriction-related-set x y*) ∈ *restriction-related-set x y*›
using *Max-in*[*OF finite-restriction-related-set-iff*[*THEN iffD2*, *OF* ‹¬ *x* ≲≈ *y*›]
nonempty-restriction-related-set]
cSup-eq-Max[*OF finite-restriction-related-set-iff*[*THEN iffD2*, *OF* ‹¬ *x* ≲≈ *y*›]
nonempty-restriction-related-set]
by *argo*

lemma *Inf-in-restriction-not-related-set* :
‹*Inf* (*restriction-not-related-set x y*) ∈ *restriction-not-related-set x y*›
by (*metis* ‹¬ *x* ≲≈ *y*› *Inf-nat-def1 finite.emptyI infinite-restriction-not-related-set-iff*)

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 −
let *?S-eq* = ‹*restriction-related-set x y*›
let *?S-neq* = ‹*restriction-not-related-set x y*›
from *Inf-in-restriction-not-related-set* **have** ‹*Inf ?S-neq* ∈ *?S-neq*› **by** *blast*
from *Sup-in-restriction-related-set* **have** ‹*Sup ?S-eq* ∈ *?S-eq*› **by** *blast*
hence ‹*Suc* (*Sup ?S-eq*) ∉ *?S-eq*›
by (*metis Suc-n-not-le-n* ‹¬ *x* ≲ *y*› *bdd-above-restriction-related-set-iff cSup-upper*)
with *restriction-related-set-Un-restriction-not-related-set*
have ‹*Suc* (*Sup ?S-eq*) ∈ *?S-neq*› **by** *auto*
show ‹*Inf ?S-neq* = *Suc* (*Sup ?S-eq*)›
proof (*rule order-antisym*)
show ‹*Inf ?S-neq* ≤ *Suc* (*Sup ?S-eq*)›
by (*fact wellorder-Inf-le1*[*OF* ‹*Suc* (*Sup ?S-eq*) ∈ *?S-neq*›])
next
from ‹*Inf ?S-neq* ∈ *?S-neq*› ‹*Sup ?S-eq* ∈ *?S-eq*› **show** ‹*Suc* (*Sup ?S-eq*) ≤ *Inf ?S-neq*›
by (*metis* (*mono-tags*, *lifting*) *mem-Collect-eq not-less-eq-eq restriction-related-le*)
qed
qed

end

lemma *restriction-related-set-is-atMost* :
‹*restriction-related-set x y* =
(*if x* ≲ *y then UNIV else* {*..Sup* (*restriction-related-set x y*)})›
proof (*split if-split*, *intro conjI impI*)
show ‹*x* ≲ *y* ⟹ *restriction-related-set x y* = *UNIV*›
by (*simp add*: *UNIV-restriction-related-set-iff*)
next
assume ‹¬ *x* ≲ *y*›
hence ∗ : ‹*Sup* (*restriction-related-set x y*) ∈ *restriction-related-set x y*›
by (*fact Sup-in-restriction-related-set*)
show ‹*restriction-related-set x y* = {*..Sup* (*restriction-related-set x y*)} ›
proof (*intro subset-antisym subsetI*)
show ‹*n* ∈ *restriction-related-set x y* ⟹ *n* ∈ {*..Sup* (*restriction-related-set x y*)}› **for** *n*
by (*simp add*: ‹¬ *x* ≲ *y*› *finite-restriction-related-set-iff le-cSup-finite*)

next
from ∗ **show** ‹$n \in \{..Sup\ (restriction\text{-}related\text{-}set\ x\ y)\} \Longrightarrow$
$n \in restriction\text{-}related\text{-}set\ x\ y$› **for** n
by *simp* (*meson mem-Collect-eq restriction-related-le*)
qed
qed

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 (*split if-split*, *intro conjI impI*)
from *empty-restriction-not-related-set-iff*
show ‹$x \lessapprox y \Longrightarrow$ *restriction-not-related-set x y* = {}› **by** *blast*
next
assume ‹$\neg\ x \lessapprox y$›
have ‹*restriction-not-related-set x y* = *UNIV* − *restriction-related-set*
x y› **by** *auto*
also have ‹. . . = *UNIV* − {..*Sup* (*restriction-related-set x y*)}›
by (*subst restriction-related-set-is-atMost*) (*simp add*: ‹$\neg\ x \lessapprox y$›)
also have ‹. . . = {*Suc* (*Sup* (*restriction-related-set x y*))..}› **by** *auto*
also have ‹*Suc* (*Sup* (*restriction-related-set x y*)) = *Inf* (*restriction-not-related-set*
x y)›
by (*simp add*: ‹$\neg\ x \lessapprox y$› *flip*: *Inf-restriction-not-related-set-eq-Suc-Sup-restriction-related-set*)
finally show ‹*restriction-not-related-set x y* = {*Inf* (*restriction-not-related-set*
x y)..}› .
qed

end

1.2 Restriction shift Maps

locale *Restriction-2-PreorderRestrictionSpace* =
R1 : *Restriction* ‹$(\downarrow_1)$› ‹$(\lessapprox_1)$› +
PRS2 : *PreorderRestrictionSpace* ‹$(\downarrow_2)$› ‹$(\lessapprox_2)$›
for $restriction_1$ :: ‹$'a \Rightarrow nat \Rightarrow 'a$› (**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

1.2.1 Definition

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

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

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*›
unfolding *restriction-shift-on-def*
by (*metis PRS2.all-restriction-related-iff-related*)

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*›
by (*unfold restriction-shift-def*, *rule restriction-shift-onI*)

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*)›
by (*unfold restriction-shift-on-def*) *blast*

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*)›
unfolding *restriction-shift-def* **using** *restriction-shift-onD* **by** *blast*

lemma *restriction-shift-on-subset* :
‹*restriction-shift-on f k B* ⟹ *A* ⊆ *B* ⟹ *restriction-shift-on f k A*›
by (*simp add*: *restriction-shift-on-def subset-iff*)

lemma *restriction-shift-imp-restriction-shift-on* [*restriction-shift-simpset*]
:
‹*restriction-shift f k* ⟹ *restriction-shift-on f k A*›
unfolding *restriction-shift-def* **using** *restriction-shift-on-subset* **by**
blast

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 (*rule restriction-shift-onI*)
fix *x y n* **assume** ‹*x* ∈ *A*› ‹*y* ∈ *A*› ‹*x* ↓$_1$ *n* ≲≈$_1$ *y* ↓$_1$ *n*›
from ‹*restriction-shift-on f k A*›[*THEN restriction-shift-onD*, *OF*
this]
have ‹*f x* ↓$_2$ *nat* (*int n* + *k*) ≲≈$_2$ *f y* ↓$_2$ *nat* (*int n* + *k*)› **.**

moreover have ‹*nat* (*int n* + *l*) ≤ *nat* (*int n* + *k*)› **by** (*simp add*: *nat-mono* ‹*l* ≤ *k*›)
ultimately show ‹*f x* ↓₂ *nat* (*int n* + *l*) ≲₂ *f y* ↓₂ *nat* (*int n* + *l*)›
using *PRS2.restriction-related-le* **by** *blast*
qed

corollary *restriction-shift-imp-restriction-shift-le* [*restriction-shift-simpset*]
:
‹*l* ≤ *k* ⟹ *restriction-shift f k* ⟹ *restriction-shift f l*›
unfolding *restriction-shift-def*
by (*fact restriction-shift-on-imp-restriction-shift-on-le*)

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 (λ*y*. *if P x then f x y else g x y*) *k A*›
by (*rule restriction-shift-onI*) (*auto dest*: *restriction-shift-onD*)

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 (λ*y*. *if P x then f x y else g x y*) *k*›
unfolding *restriction-shift-def* **by** (*fact restriction-shift-on-if-then-else*)

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 (*unfold non-too-destructive-on-def*, *rule restriction-shift-onI*)
fix *x y n*
show ‹⟦*x* ∈ *A*; *y* ∈ *A*; ¬ *f x* ≲₂ *f y*; *x* ↓₁ *n* ≲₁ *y* ↓₁ *n*⟧
⟹ *f x* ↓₂ *nat* (*int n* + − *1*) ≲₂ *f y* ↓₂ *nat* (*int n* + − *1*)›

by (*cases* ‹$n < 1$›) (*simp-all add*: *Suc-nat-eq-nat-zadd1 that*)
qed

lemma *non-too-destructiveI* :
‹$[\![\bigwedge n\ x\ y.\ [\![\neg\ f\ x \lessapprox_2 f\ y;\ x \downarrow_1 Suc\ n \lessapprox_1 y \downarrow_1 Suc\ n]\!] \Longrightarrow f\ x \downarrow_2 n \lessapprox_2 f\ y \downarrow_2 n]\!]$
$\Longrightarrow$ *non-too-destructive f*›
by (*unfold non-too-destructive-def*, *rule non-too-destructive-onI*)

lemma *non-too-destructive-onD* :
‹$[\![$*non-too-destructive-on f A*; $x \in A$; $y \in A$; $x \downarrow_1 Suc\ n \lessapprox_1 y \downarrow_1 Suc\ n]\!] \Longrightarrow f\ x \downarrow_2 n \lessapprox_2 f\ y \downarrow_2 n$›
unfolding *non-too-destructive-on-def* **using** *restriction-shift-onD* **by** *fastforce*

lemma *non-too-destructiveD* :
‹$[\![$*non-too-destructive f*; $x \downarrow_1 Suc\ n \lessapprox_1 y \downarrow_1 Suc\ n]\!] \Longrightarrow f\ x \downarrow_2 n \lessapprox_2 f\ y \downarrow_2 n$›
unfolding *non-too-destructive-def* **using** *non-too-destructive-onD*
by *simp*

lemma *non-too-destructive-on-subset* :
‹*non-too-destructive-on f B* $\Longrightarrow A \subseteq B \Longrightarrow$ *non-too-destructive-on f A*›
by (*meson non-too-destructive-on-def restriction-shift-on-subset*)

lemma *non-too-destructive-imp-non-too-destructive-on* [*restriction-shift-simpset*] :
‹*non-too-destructive f* $\Longrightarrow$ *non-too-destructive-on f A*›
unfolding *non-too-destructive-def* **using** *non-too-destructive-on-subset*
by *auto*

corollary *non-too-destructive-on-if-then-else* [*restriction-shift-simpset*, *restriction-shift-introset*]:
‹$[\![\bigwedge x.\ P\ x \Longrightarrow$ *non-too-destructive-on* $(f\ x)\ A;\ \bigwedge x.\ \neg\ P\ x \Longrightarrow$ *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* $(f\ x)$; $\bigwedge x.\ \neg\ P\ x \Longrightarrow$ *non-too-destructive* $(g\ x)]\!]$
$\Longrightarrow$ *non-too-destructive* $(\lambda y.$ *if P x then f x y else g x y*$)$›
by (*auto simp add*: *non-too-destructive-def non-too-destructive-on-def intro*: *restriction-shift-on-if-then-else*)

Non-destructive Map **definition** *non-destructive-on* :: ‹[$'a \Rightarrow 'b$, $'a$ *set*] $\Rightarrow$ *bool*›
where ‹*non-destructive-on f A* $\equiv$ *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* :
‹⟦⋀*n x y*. ⟦*n* ≠ *0*; *x* ∈ *A*; *y* ∈ *A*; ¬ *f x* ≲≈2 *f y*; *x* ↓1 *n* ≲≈1 *y* ↓1 *n*⟧
⟹ *f x* ↓2 *n* ≲≈2 *f y* ↓2 *n*⟧
⟹ *non-destructive-on f A*›
by (*unfold non-destructive-on-def*, *rule restriction-shift-onI*)
(*metis PRS2.restriction-0-related add.right-neutral nat-int*)

lemma *non-destructiveI* :
‹⟦⋀*n x y*. ⟦*n* ≠ *0*; ¬ *f x* ≲≈2 *f y*; *x* ↓1 *n* ≲≈1 *y* ↓1 *n*⟧ ⟹ *f x* ↓2 *n* ≲≈2 *f y* ↓2 *n*⟧
⟹ *non-destructive f*› **by** (*unfold non-destructive-def*, *rule non-destructive-onI*)

lemma *non-destructive-onD* :
‹⟦*non-destructive-on f A*; *x* ∈ *A*; *y* ∈ *A*; ¬ *f x* ≲≈2 *f y*; *x* ↓1 *n* ≲≈1 *y* ↓1 *n*⟧ ⟹ *f x* ↓2 *n* ≲≈2 *f y* ↓2 *n*›
by (*simp add*: *non-destructive-on-def restriction-shift-on-def*)

lemma *non-destructiveD* : ‹⟦*non-destructive f*; *x* ↓1 *n* ≲≈1 *y* ↓1 *n*⟧ ⟹ *f x* ↓2 *n* ≲≈2 *f y* ↓2 *n*›
by (*simp add*: *non-destructive-def non-destructive-on-def restriction-shift-on-def*)

lemma *non-destructive-on-subset* :
‹*non-destructive-on f B* ⟹ *A* ⊆ *B* ⟹ *non-destructive-on f A*›
by (*meson non-destructive-on-def restriction-shift-on-subset*)

lemma *non-destructive-imp-non-destructive-on* [*restriction-shift-simpset*]
:
‹*non-destructive f* ⟹ *non-destructive-on f A*›
unfolding *non-destructive-def* **using** *non-destructive-on-subset* **by** *auto*

lemma *non-destructive-on-imp-non-too-destructive-on* [*restriction-shift-simpset*]
:
‹*non-destructive-on f A* ⟹ *non-too-destructive-on f A*›
unfolding *non-destructive-on-def non-too-destructive-on-def*
by (*rule restriction-shift-on-imp-restriction-shift-on-le*[*of* ‹− *1*› *0 f A*, *simplified*])

corollary *non-destructive-imp-non-too-destructive* [*restriction-shift-simpset*]
:
‹*non-destructive f* ⟹ *non-too-destructive f*›
by (*unfold non-destructive-def non-too-destructive-def*)
(*fact non-destructive-on-imp-non-too-destructive-on*)

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*)›
by (*auto simp add*: *non-destructive-def non-destructive-on-def intro*: *restriction-shift-on-if-then-else*)

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*›
by (*simp add*: *Suc-as-int constructive-on-def restriction-shift-onI*)

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*› **by** (*unfold constructive-def*, *rule constructive-onI*)

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*›
unfolding *constructive-on-def* **by** (*metis Suc-as-int restriction-shift-onD*)

lemma *constructiveD* : ‹⟦*constructive f*; *x* ↓$_1$ *n* ≲≈$_1$ *y* ↓$_1$ *n*⟧ ⟹ *f x* ↓$_2$ *Suc n* ≲≈$_2$ *f y* ↓$_2$ *Suc n*›
unfolding *constructive-def* **using** *constructive-onD* **by** *blast*

lemma *constructive-on-subset* :
‹*constructive-on f B* ⟹ *A* ⊆ *B* ⟹ *constructive-on f A*›
by (*meson constructive-on-def restriction-shift-on-subset*)

lemma *constructive-imp-constructive-on* [*restriction-shift-simpset*] :

‹*constructive f* ⟹ *constructive-on f A*›
unfolding *constructive-def* **using** *constructive-on-subset* **by** *auto*

lemma *constructive-on-imp-non-destructive-on* [*restriction-shift-simpset*]
:
‹*constructive-on f A* ⟹ *non-destructive-on f A*›
by (*rule non-destructive-onI*)
(*meson PRS2.restriction-related-pred constructive-onD*)

corollary *constructive-imp-non-destructive* [*restriction-shift-simpset*]
:
‹*constructive f* ⟹ *non-destructive f*›
unfolding *constructive-def non-destructive-def*
by (*fact constructive-on-imp-non-destructive-on*)

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* (λ*y*. *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* (λ*y*. *if P x then f x y else g x y*)›
by (*auto simp add*: *constructive-def constructive-on-def*
intro: *restriction-shift-on-if-then-else*)

end

1.2.3 Properties

locale *PreorderRestrictionSpace-2-PreorderRestrictionSpace* =
PRS1 : *PreorderRestrictionSpace* ‹(↓$_1$)› ‹(≲$_1$)› +
PRS2 : *PreorderRestrictionSpace* ‹(↓$_2$)› ‹(≲$_2$)›
for *restriction*$_1$ :: ‹*'a* ⇒ *nat* ⇒ *'a*› (**infixl** ‹↓$_1$› *60*)
and *relation*$_1$ :: ‹*'a* ⇒ *'a* ⇒ *bool*› (**infixl** ‹≲$_1$› *50*)
and *restriction*$_2$ :: ‹*'b* ⇒ *nat* ⇒ *'b*› (**infixl** ‹↓$_2$› *60*)
and *relation*$_2$ :: ‹*'b* ⇒ *'b* ⇒ *bool*› (**infixl** ‹≲$_2$› *50*)
begin

sublocale *Restriction-2-PreorderRestrictionSpace* **by** *unfold-locales*

lemma *restriction-shift-on-restriction-restriction* :
‹*f* (*x* ↓$_1$ *n*) ↓$_2$ *nat* (*int n* + *k*) ≲$_2$ *f x* ↓$_2$ *nat* (*int n* + *k*)›
if ‹*restriction-shift-on f k A*› ‹*x* ↓$_1$ *n* ∈ *A*› ‹*x* ∈ *A*› ‹*x* ↓$_1$ *n* ≲$_1$ *x* ↓$_1$ *n*›

— the last assumption is trivial if ($\lessapprox_1$) is reflexive
by (*rule restriction-shift-onD*
[*OF* ‹*restriction-shift-on f k A*› ‹$x \downarrow_1 n \in A$› ‹$x \in A$›])
(*simp add*: ‹$x \downarrow_1 n \lessapprox_1 x \downarrow_1 n$›)

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** ‹$x \downarrow_1 n \lessapprox_1 x \downarrow_1 n$›
by (*rule restriction-shiftD*[*OF* ‹*restriction-shift f k*›])
(*simp add*: ‹$x \downarrow_1 n \lessapprox_1 x \downarrow_1 n$›)

corollary *constructive-on-restriction-restriction* :
‹$[\![$*constructive-on f A*; $x \downarrow_1 n \in A$; $x \in A$; $x \downarrow_1 n \lessapprox_1 x \downarrow_1 n]\!]$
$\Longrightarrow f\ (x \downarrow_1 n) \downarrow_2 Suc\ n \lessapprox_2 f\ x \downarrow_2 Suc\ n$›
using *restriction-shift-on-restriction-restriction*
restriction-shift-restriction-restriction Suc-as-int
unfolding *constructive-on-def* **by** *presburger*

corollary *constructive-restriction-restriction* :
‹*constructive f* $\Longrightarrow x \downarrow_1 n \lessapprox_1 x \downarrow_1 n \Longrightarrow f\ (x \downarrow_1 n) \downarrow_2 Suc\ n \lessapprox_2 f\ x \downarrow_2 Suc\ n$›
by (*simp add*: *constructive-def constructive-on-restriction-restriction*)

corollary *non-destructive-on-restriction-restriction* :
‹$[\![$*non-destructive-on f A*; $x \downarrow_1 n \in A$; $x \in A$; $x \downarrow_1 n \lessapprox_1 x \downarrow_1 n]\!]$
$\Longrightarrow f\ (x \downarrow_1 n) \downarrow_2 n \lessapprox_2 f\ x \downarrow_2 n$›
using *restriction-shift-on-restriction-restriction*
restriction-shift-restriction-restriction
unfolding *non-destructive-on-def* **by** (*metis add.commute add-0 nat-int*)

corollary *non-destructive-restriction-restriction* :
‹*non-destructive f* $\Longrightarrow x \downarrow_1 n \lessapprox_1 x \downarrow_1 n \Longrightarrow f\ (x \downarrow_1 n) \downarrow_2 n \lessapprox_2 f\ x \downarrow_2 n$›
by (*simp add*: *non-destructive-def non-destructive-on-restriction-restriction*)

corollary *non-too-destructive-on-restriction-restriction* :
‹$[\![$*non-too-destructive-on f A*; $x \downarrow_1 Suc\ n \in A$; $x \in A$; $x \downarrow_1 Suc\ n \lessapprox_1 x \downarrow_1 Suc\ n]\!]$
$\Longrightarrow f\ (x \downarrow_1 Suc\ n) \downarrow_2 n \lessapprox_2 f\ x \downarrow_2 n$›
using *restriction-shift-on-restriction-restriction*
restriction-shift-restriction-restriction
unfolding *non-too-destructive-on-def* **by** *fastforce*

corollary *non-too-destructive-restriction-restriction* :
‹*non-too-destructive f* $\Longrightarrow x \downarrow_1 Suc\ n \lessapprox_1 x \downarrow_1 Suc\ n \Longrightarrow f\ (x \downarrow_1 Suc$

$n) \downarrow_2 n \lessapprox_2 f\ x \downarrow_2 n\rangle$
by (*simp add*: *non-too-destructive-def non-too-destructive-on-restriction-restriction*)

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* $\langle(\downarrow_1)\rangle$
$\langle(\lessapprox_1)\rangle$ $\langle(\downarrow_3)\rangle$ $\langle(\lessapprox_3)\rangle$
by *unfold-locales*

interpretation *PRS2PRS3* : *PreorderRestrictionSpace-2-PreorderRestrictionSpace*
$\langle(\downarrow_2)\rangle$ $\langle(\lessapprox_2)\rangle$ $\langle(\downarrow_3)\rangle$ $\langle(\lessapprox_3)\rangle$
by *unfold-locales*

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 (*rule R2PRS2.restriction-shift-onI*)
fix *x y n* **assume** $\langle x \in A\rangle$ $\langle y \in A\rangle$ $\langle x \downarrow_1 n \lessapprox_1 y \downarrow_1 n\rangle$
from $\langle$*R2PRS1.restriction-shift-on f k A*$\rangle$[*THEN R2PRS1.restriction-shift-onD*,
OF this]
have $\langle f\ x \downarrow_2 nat\ (int\ n + k) \lessapprox_2 f\ y \downarrow_2 nat\ (int\ n + k)\rangle$ **.**
moreover from $\langle x \in A\rangle$ $\langle y \in A\rangle$ $\langle f\ `\ A \subseteq B\rangle$ **have** $\langle f\ x \in B\rangle$ $\langle f\ y$
$\in B\rangle$ **by** *auto*
ultimately have $\langle g\ (f\ x) \downarrow_3 nat\ (int\ (nat\ (int\ n + k)) + l) \lessapprox_3$
$g\ (f\ y) \downarrow_3 nat\ (int\ (nat\ (int\ n + k)) + l)\rangle$
using $\langle$*PRS2PRS3.restriction-shift-on g l B*$\rangle$[*THEN PRS2PRS3.restriction-shift-onD*]
by *blast*
moreover have $\langle nat\ (int\ n + (k + l)) \leq nat\ (int\ (nat\ (int\ n + k))$
$+ l)\rangle$
by (*simp add*: *nat-mono*)

ultimately show ‹$g\ (f\ x) \downarrow_3 nat\ (int\ n + (k + l)) \lessapprox_3 g\ (f\ y) \downarrow_3 nat\ (int\ n + (k + l))$›
by (*metis PRS2.restriction-related-le*)
qed

corollary *restriction-shift-comp-restriction-shift-on* [*restriction-shift-simpset*]
:
‹*PRS2PRS3.restriction-shift g l* ⟹ *R2PRS1.restriction-shift-on f k A* ⟹
R2PRS2.restriction-shift-on (λ*x*. *g* (*f x*)) (*k* + *l*) *A*›
using *PRS2PRS3.restriction-shift-imp-restriction-shift-on*
restriction-shift-on-comp-restriction-shift-on **by** *blast*

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*)›
by (*simp add*: *R2PRS1.restriction-shift-imp-restriction-shift-on*
R2PRS2.restriction-shift-def restriction-shift-comp-restriction-shift-on)

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*›
by (*simp add*: *R2PRS1.non-destructive-on-def R2PRS2.non-destructive-on-def*
R2PRS2.restriction-shift-on-def R2PRS1.restriction-shift-on-def)
(*meson PRS2.mono-restriction-related PRS2PRS3.non-destructive-onD image-subset-iff*)

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*›
by (*simp add*: *PRS2PRS3.non-destructiveD R2PRS1.non-destructive-on-def*
R2PRS2.non-destructive-on-def R2PRS2.restriction-shift-on-def
R2PRS1.restriction-shift-on-def)

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*))›
by (*simp add*: *PRS2PRS3.non-destructiveD R2PRS1.non-destructiveD*
R2PRS2.non-destructive-def R2PRS2.non-destructive-onI)

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*›
by (*metis PRS2PRS3.constructive-on-def R2PRS1.non-destructive-on-def R2PRS2.constructive-on-def add.commute add-cancel-left-right restriction-shift-on-comp-restriction-shift-on*)

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*›
by (*simp add*: *R2PRS1.Restriction-2-PreorderRestrictionSpace-axioms PRS2PRS3.constructiveD R2PRS1.non-destructive-on-def R2PRS2.constructive-onI Restriction-2-PreorderRestrictionSpace.restriction-shift-on-def*)

corollary *constructive-comp-non-destructive* [*restriction-shift-simpset*]
:
‹*PRS2PRS3.constructive g* ⟹ *R2PRS1.non-destructive f* ⟹
R2PRS2.constructive (λ*x*. *g* (*f x*))›
by (*simp add*: *PRS2PRS3.constructiveD R2PRS1.non-destructiveD R2PRS2.constructiveI*)

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*›
by (*simp add*: *PRS2PRS3.non-destructive-onD R2PRS1.constructive-onD R2PRS2.constructive-onI image-subset-iff*)

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*›
using *PRS2PRS3.non-destructive-def non-destructive-on-comp-constructive-on*
by *blast*

corollary *non-destructive-comp-constructive* [*restriction-shift-simpset*]
:
‹*PRS2PRS3.non-destructive g* ⟹ *R2PRS1.constructive f* ⟹
R2PRS2.constructive (λ*x*. *g* (*f x*))›
using *PRS2PRS3.non-destructiveD R2PRS1.constructiveD R2PRS2.constructiveI*
by *presburger*

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 (λx. g (f x)) A›
by (*metis PRS2PRS3.non-too-destructive-on-def R2PRS1.non-destructive-on-def R2PRS2.non-too-destructive-on-def add.commute add.right-neutral restriction-shift-on-comp-restriction-shift-on*)

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 (λx. *g* (*f x*)) *A*›
by (*metis PRS2PRS3.non-too-destructive-imp-non-too-destructive-on non-too-destructive-on-comp-non-destructive-on top-greatest*)

corollary *non-too-destructive-comp-non-destructive* [*restriction-shift-simpset*]
:
‹*PRS2PRS3.non-too-destructive g* ⟹ *R2PRS1.non-destructive f* ⟹
R2PRS2.non-too-destructive (λx. *g* (*f x*))›
by (*simp add*: *PRS2PRS3.non-too-destructiveD R2PRS1.non-destructiveD R2PRS2.non-too-destructiveI*)

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 (λx. g (f x)) A›
by (*simp add*: *PRS2PRS3.non-destructive-onD R2PRS1.non-too-destructive-onD R2PRS2.non-too-destructive-onI image-subset-iff*)

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 (λx. *g* (*f x*)) *A*›
by (*simp add*: *PRS2PRS3.non-destructiveD R2PRS1.non-too-destructive-onD R2PRS2.non-too-destructive-onI*)

corollary *non-destructive-comp-non-too-destructive* [*restriction-shift-simpset*]
:
‹*PRS2PRS3.non-destructive g* ⟹ *R2PRS1.non-too-destructive f* ⟹
R2PRS2.non-too-destructive (λx. *g* (*f x*))›
using *R2PRS1.non-too-destructive-imp-non-too-destructive-on R2PRS2.non-too-destructive-def non-destructive-comp-non-too-destructive-on* **by** *blast*

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*›
using *restriction-shift-on-comp-restriction-shift-on*[*of f A B g* ‹− *1*› *1*]
by (*simp add*: *PRS2PRS3.non-too-destructive-on-def*
R2PRS2.non-destructive-on-def R2PRS1.constructive-on-def)

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*›
by (*metis PRS2PRS3.non-too-destructive-imp-non-too-destructive-on*
non-too-destructive-on-comp-constructive-on top-greatest)

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*))›
by (*simp add*: *PRS2PRS3.non-too-destructiveD R2PRS1.constructiveD R2PRS2.non-destructiveI*)

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*›
using *restriction-shift-on-comp-restriction-shift-on*[*of f A B g 1* ‹− *1*›]
by (*simp add*: *R2PRS1.non-too-destructive-on-def*
PRS2PRS3.constructive-on-def R2PRS2.non-destructive-on-def)

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*›
using *PRS2PRS3.constructive-imp-constructive-on constructive-on-comp-non-too-destructive-on*
by *blast*

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*))›

by (*metis R2PRS1.non-too-destructive-imp-non-too-destructive-on R2PRS2.non-destructiveI*
R2PRS2.non-destructive-onD UNIV-I constructive-comp-non-too-destructive-on)

end

2 Class Implementation

2.1 Preliminaries

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

lemma *INFM-nat-le*: ‹$(\exists_\infty n :: nat.\ P\ n) \longleftrightarrow (\forall m.\ \exists n{\geq}m.\ P\ n)$›
unfolding *cofinite-eq-sequentially frequently-sequentially* **by** *simp*

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

fun *extraction-subseq* :: ‹$[nat \Rightarrow 'a, 'a \Rightarrow bool] \Rightarrow nat \Rightarrow nat$›
where ‹*extraction-subseq* $\sigma\ P\ 0 = (LEAST\ k.\ P\ (\sigma\ k))$›
| ‹*extraction-subseq* $\sigma\ P\ (Suc\ n) = (LEAST\ k.$ *extraction-subseq* $\sigma\ P\ n < k \wedge P\ (\sigma\ k))$›

lemma *exists-extraction-subseq* :
assumes ‹$\exists_\infty k.\ P\ (\sigma\ k)$›
defines *f-def* : ‹$f \equiv$ *extraction-subseq* $\sigma\ P$›
shows ‹*strict-mono f*› **and** ‹$P\ (\sigma\ (f\ k))$›
proof −
have ‹$f\ n < f\ (Suc\ n) \wedge P\ (\sigma\ (f\ n))$› **for** n
by (*cases n*, *simp-all add*: *f-def*)
(*metis* (*mono-tags*, *lifting*) ‹$\exists_\infty k.\ P\ (\sigma\ k)$›[*unfolded INFM-nat-le*] *LeastI-ex Suc-le-eq*)+
thus ‹*strict-mono f*› **and** ‹$P\ (\sigma\ (f\ k))$›
by (*simp-all add*: *strict-mono-Suc-iff*)
qed

lemma *extraction-subseqD* :
‹$\exists f :: nat \Rightarrow nat.$ *strict-mono* $f \wedge (\forall k.\ P\ (\sigma\ (f\ k)))$› **if** ‹$\exists_\infty k.\ P\ (\sigma\ k)$›
proof (*rule exI*)
show ‹*strict-mono* (*extraction-subseq* $\sigma\ P$) $\wedge$ ($\forall k.\ P\ (\sigma\ ((extraction\text{-}subseq\ \sigma\ P)\ k))$)›

by (*simp add*: ‹$\exists_\infty k.\ P\ (\sigma\ k)$› *exists-extraction-subseq*)
qed

lemma *extraction-subseqE* :
— The idea is to abstract the concrete construction of this extraction function, we only need the fact that there is one.
‹$\exists_\infty k.\ P\ (\sigma\ k) \Longrightarrow (\bigwedge f :: nat \Rightarrow nat.\ strict\text{-}mono\ f \Longrightarrow (\bigwedge k.\ P\ (\sigma\ (f\ k))) \Longrightarrow thesis) \Longrightarrow thesis$›
by (*blast dest*: *extraction-subseqD*)

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* ‹(↓)› ‹(=)› **by** *unfold-locales simp*
— Just to recover *local.restriction-related-set* and *local.restriction-not-related-set*.
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* ‹(↓)› ‹(=)›
by *unfold-locales* (*simp-all add*: *ex-not-restriction-eq*)

lemma *restriction-related-set-commute* :
‹*restriction-related-set x y* = *restriction-related-set y x*› **by** *auto*

lemma *restriction-not-related-set-commute* :
‹*restriction-not-related-set x y* = *restriction-not-related-set y x*› **by** *auto*

end

context *restriction-space* **begin**

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

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

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

‹(↓) :: 'a ⇒ nat ⇒ 'a› ‹(=)› **..**

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› ‹(=)› **..**

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*› **by** (*simp add*: *restriction-shiftI*)

lemma *constructive-const* [*restriction-shift-simpset*] :
‹*constructive* (λ*x*. *c*)› **by** (*simp add*: *constructiveI*)

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 (*rule restriction-shift-onI*)
fix *x y m* **assume** ‹*x* ∈ *A*› ‹*y* ∈ *A*› ‹*x* ↓ *m* = *y* ↓ *m*›
from *restriction-shift-onD*[*OF* ‹*restriction-shift-on f k A*› *this*]
have ‹*f x* ↓ *nat* (*int m* + *k*) = *f y* ↓ *nat* (*int m* + *k*)› **.**
hence ‹*f x* ↓ *nat* (*int m* + *k*) ↓ *n* = *f y* ↓ *nat* (*int m* + *k*) ↓ *n*› **by** *argo*
thus ‹*f x* ↓ *n* ↓ *nat* (*int m* + *k*) = *f y* ↓ *n* ↓ *nat* (*int m* + *k*)›
by (*simp add*: *min.commute*)
qed

lemma *restriction-shift-restricted* [*restriction-shift-simpset*] :
‹*restriction-shift f k* ⟹ *restriction-shift* (λ*x*. *f x* ↓ *n*) *k*›
using *restriction-shift-def restriction-shift-on-restricted* **by** *blast*

corollary *constructive-restricted* [*restriction-shift-simpset*] :
‹*constructive f* ⟹ *constructive* (λ*x*. *f x* ↓ *n*)›
by (*simp add*: *constructive-def constructive-on-def restriction-shift-on-restricted*)

corollary *non-destructive-restricted* [*restriction-shift-simpset*] :
‹*non-destructive f* ⟹ *non-destructive* (λ*x*. *f x* ↓ *n*)›
by (*simp add*: *non-destructive-def non-destructive-on-def restriction-shift-on-restricted*)

lemma *non-destructive-id* [*restriction-shift-simpset*] :
  ‹*non-destructive id*› ‹*non-destructive* (λ*x*. *x*)›
  by (*simp-all add*: *id-def non-destructiveI*)

interpretation *less-eqRS* : *Restriction* ‹(↓)› ‹(≤)› **by** *unfold-locales*
  — Just to recover *less-eqRS.restriction-related-set* and *less-eqRS.restriction-not-related-set*.

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 *unfold-locales*
  show ‹*x* ↓ *0* ≤ *y* ↓ *0*› **for** *x y* :: 'a **by** *simp*
next
  show ‹*x* ≤ *y* ⟹ *x* ↓ *n* ≤ *y* ↓ *n*› **for** *x y* :: 'a **and** *n*
    by (*fact mono-restriction-less-eq*)
next
  show ‹¬ *x* ≤ *y* ⟹ ∃ *n*. ¬ *x* ↓ *n* ≤ *y* ↓ *n*› **for** *x y* :: 'a
    by (*simp add*: *ex-not-restriction-less-eq*)
next
 show ‹*x* ≤ *y* ⟹ *y* ≤ *z* ⟹ *x* ≤ *z*› **for** *x y z* :: 'a **by** (*fact order-trans*)
qed

end

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

subclass *restriction-space*
proof *unfold-locales*
  show ‹*x* ↓ *0* = *y* ↓ *0*› **for** *x y* :: 'a **by** (*rule order-antisym*) *simp-all*
next
  show ‹*x* ≠ *y* ⟹ ∃ *n*. *x* ↓ *n* ≠ *y* ↓ *n*› **for** *x y* :: 'a
    by (*metis ex-not-restriction-less-eq order.eq-iff*)
qed

end

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

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

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› ‹(≤)› **..**

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› ‹(≤)› **..**

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* :
‹(⋀*x* *y* *n*. ⟦*x* ∈ *A*; *y* ∈ *A*; *f* *x* ≠ *f* *y*; *x* ↓ *n* = *y* ↓ *n*⟧ ⟹
f *x* ↓ *nat* (*int* *n* + *k*) ≤ *f* *y* ↓ *nat* (*int* *n* + *k*))
⟹ *restriction-shift-on* *f* *k* *A*›
by (*simp add*: *order-antisym* *restriction-shift-onI*)

corollary *order-restriction-shiftI* :
‹(⋀*x* *y* *n*. ⟦*f* *x* ≠ *f* *y*; *x* ↓ *n* = *y* ↓ *n*⟧ ⟹
f *x* ↓ *nat* (*int* *n* + *k*) ≤ *f* *y* ↓ *nat* (*int* *n* + *k*))
⟹ *restriction-shift* *f* *k*›
by (*simp add*: *order-antisym* *restriction-shiftI*)

corollary *order-non-too-destructive-onI* :
‹(⋀*x* *y* *n*. ⟦*x* ∈ *A*; *y* ∈ *A*; *f* *x* ≠ *f* *y*; *x* ↓ *Suc* *n* = *y* ↓ *Suc* *n*⟧ ⟹
f *x* ↓ *n* ≤ *f* *y* ↓ *n*)
⟹ *non-too-destructive-on* *f* *A*›

by (*simp add*: *order-antisym non-too-destructive-onI*)

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*›
by (*simp add*: *order-antisym non-too-destructiveI*)

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*›
by (*simp add*: *order-antisym non-destructive-onI*)

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*›
by (*simp add*: *order-antisym non-destructiveI*)

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*›
by (*simp add*: *order-antisym constructive-onI*)

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*›
by (*simp add*: *order-antisym constructiveI*)

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* $\sigma \Longrightarrow \sigma\ (Suc\ n) \downarrow n = \sigma\ n$›
by (*simp-all add*: *restriction-chain-def*)

end

```
context restriction-space begin

lemma (in restriction-space) restriction-chain-def-bis:
  ‹restriction-chain σ ⟷ (∀ n m. n < m ⟶ σ m ↓ n = σ n)›
proof (rule iffI)
  show ‹∀ n m. n < m ⟶ σ m ↓ n = σ n ⟹ restriction-chain σ›
    by (simp add: restriction-chainI)
next
  show ‹restriction-chain σ ⟹ ∀ n m. n < m ⟶ σ m ↓ n = σ n›
  proof (intro allI impI)
    fix n m
    assume restriction : ‹restriction-chain σ›
    show ‹n < m ⟹ σ m ↓ n = σ n›
    proof (induct ‹m − n› arbitrary: m)
      fix m
      assume ‹0 = m − n› and ‹n < m›
      hence False by simp
      thus ‹σ m ↓ n = σ n› by simp
    next
      fix x m
      assume prems : ‹Suc x = m − n› ‹n < m›
       assume hyp : ‹x = m' − n ⟹ n < m' ⟹ σ m' ↓ n = σ n›
for m'
       obtain m' where ‹m = Suc m'› by (meson less-imp-Suc-add
prems(2))
      from prems(2) ‹m = Suc m'› consider ‹n = m'› | ‹n < m'› by
linarith
      thus ‹σ m ↓ n = σ n›
      proof cases
        show ‹n = m' ⟹ σ m ↓ n = σ n›
          by (simp add: ‹m = Suc m'› restriction restriction-chainD)
      next
        assume ‹n < m'›
         have ∗ : ‹σ (Suc m') ↓ n = σ (Suc m') ↓ m' ↓ n› by (simp
add: ‹n < m'›)
          from ‹m = Suc m'› prems(1) have ∗∗ : ‹x = m' − n› by
linarith
        show ‹σ m ↓ n = σ n›
          by (simp add: ∗ ∗∗ ‹m = Suc m'› ‹n < m'› hyp restriction
restriction-chainD)
      qed
    qed
  qed
qed


lemma restricted-restriction-chain-is :
  ‹restriction-chain σ ⟹ (λn. σ n ↓ n) = σ›
 by (rule ext) (metis min.idem restriction-chainD restriction-restriction)
```

lemma *restriction-chain-def-ter*:
‹*restriction-chain* $\sigma \longleftrightarrow (\forall\, n\ m.\ n \leq m \longrightarrow \sigma\ m \downarrow n = \sigma\ n)$›
by (*metis le-eq-less-or-eq restricted-restriction-chain-is restriction-chain-def-bis*)

lemma *restriction-chain-restrictions* : ‹*restriction-chain* $((\downarrow)\ x)$›
by (*simp add*: *restriction-chainI*)

end

Iterations The sequence of restricted images of powers of a constructive function is a $chain_{\downarrow}$.

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

lemma *restriction-chain-funpow-restricted* [*simp*]:
‹*restriction-chain* $(\lambda n.\ (f \frown n)\ x \downarrow n)$› **if** ‹*constructive f*›
proof (*rule restriction-chainI*)
show ‹$(f \frown Suc\ n)\ x \downarrow Suc\ n \downarrow n = (f \frown n)\ x \downarrow n$› **for** n
proof (*induct n*)
show ‹$(f \frown Suc\ 0)\ x \downarrow Suc\ 0 \downarrow 0 = (f \frown 0)\ x \downarrow 0$› **by** *simp*
next
fix n **assume** ‹$(f \frown Suc\ n)\ x \downarrow Suc\ n \downarrow n = (f \frown n)\ x \downarrow n$›
from ‹*constructive f*›[*THEN constructiveD*, *OF this*[*simplified*]]
show ‹$(f \frown Suc\ (Suc\ n))\ x \downarrow Suc\ (Suc\ n) \downarrow Suc\ n = (f \frown Suc\ n)\ x \downarrow Suc\ n$› **by** *simp*
qed
qed

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 (*induct n arbitrary*: $k\ l$)
show ‹$(f \frown k)\ x \downarrow 0 = (f \frown l)\ y \downarrow 0$› **for** $k\ l$ **by** *simp*
next
fix $n\ k\ l$ **assume** *hyp* : ‹$n \leq k \Longrightarrow n \leq l \Longrightarrow (f \frown k)\ x \downarrow n = (f \frown l)\ y \downarrow n$› **for** $k\ l$
assume ‹$Suc\ n \leq k$› ‹$Suc\ n \leq l$›
then obtain $k'\ l'$ **where** ‹$k = Suc\ k'$› ‹$n \leq k'$› ‹$l = Suc\ l'$› ‹$n \leq l'$›
by (*metis Suc-le-D not-less-eq-eq*)
from ‹*constructive f*›[*THEN constructiveD*, *OF hyp*[*OF* ‹$n \leq k'$› ‹$n \leq l'$›]]
show ‹$(f \frown k)\ x \downarrow Suc\ n = (f \frown l)\ y \downarrow Suc\ n$›
by (*simp add*: ‹$k = Suc\ k'$› ‹$l = Suc\ l'$›)
qed

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›
  by (*simp add*: *restriction-tendsto-def*)

lemma *restriction-tendstoD* : ‹σ −↓→ Σ ⟹ ∃ *n0*. ∀ *k*≥*n0*. Σ ↓ *n* = σ *k* ↓ *n*›
  by (*simp add*: *restriction-tendsto-def*)

lemma *restriction-tendstoE* :
  ‹σ −↓→ Σ ⟹ (⋀*n0*. (⋀*k*. *n0* ≤ *k* ⟹ Σ ↓ *n* = σ *k* ↓ *n*) ⟹ *thesis*) ⟹ *thesis*›
  using *restriction-tendstoD* **by** *blast*

end

lemma (**in** *restriction-space*) *restriction-tendsto-unique* :
  ‹σ −↓→ Σ ⟹ σ −↓→ Σ' ⟹ Σ = Σ'›
  by (*metis ex-not-restriction-eq restriction-tendstoD nat-le-linear*)

context *restriction* **begin**

lemma *restriction-tendsto-const-restricted* :
  ‹σ −↓→ Σ ⟹ (λ*n*. σ *n* ↓ *k*) −↓→ Σ ↓ *k*›
  unfolding *restriction-tendsto-def* **by** *metis*

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

lemma *restriction-tendsto-const* : ‹(λ*n*. Σ) −↓→ Σ›
  by (*simp add*: *restriction-tendstoI*)

lemma (**in** *restriction-space*) *restriction-tendsto-restrictions* : ‹(λ*n*. Σ ↓ *n*) −↓→ Σ›
  by (*metis restriction-tendstoI restriction-chain-def-ter restriction-chain-restrictions*)

lemma *restriction-tendsto-shift-iff* : ‹(λ*n*. σ (*n* + *l*)) −↓→ Σ ⟷ σ −↓→ Σ›
proof (*rule iffI*)
  show ‹(λ*n*. σ (*n* + *l*)) −↓→ Σ› **if** ‹σ −↓→ Σ›

proof (*rule restriction-tendstoI*)
from ‹σ $-\downarrow\rightarrow$ Σ›[*THEN restriction-tendstoD*]
show ‹$\exists$ *n0*. $\forall$ $k{\geq}n0$. $\Sigma \downarrow n = \sigma\ (k + l) \downarrow n$› **for** n **by** (*meson trans-le-add1*)
qed
next
show ‹σ $-\downarrow\rightarrow$ Σ› **if** ‹$(\lambda n.\ \sigma\ (n + l))$ $-\downarrow\rightarrow$ Σ›
proof (*rule restriction-tendstoI*)
fix n
from ‹$(\lambda n.\ \sigma\ (n + l))$ $-\downarrow\rightarrow$ Σ›[*THEN restriction-tendstoD*]
obtain *n0* **where** ‹$\forall$ $k{\geq}n0$. $\Sigma \downarrow n = \sigma\ (k + l) \downarrow n$› **..**
hence ‹$\forall$ $k{\geq}n0 + l$. $\Sigma \downarrow n = \sigma\ k \downarrow n$›
by (*metis add.commute add-leD2 add-le-imp-le-left le-iff-add*)
thus ‹$\exists$ *n1*. $\forall$ $k{\geq}n1$. $\Sigma \downarrow n = \sigma\ k \downarrow n$› **..**
qed
qed

lemma *restriction-tendsto-shiftI* : ‹σ $-\downarrow\rightarrow$ $\Sigma \Longrightarrow (\lambda n.\ \sigma\ (n + l))$ $-\downarrow\rightarrow$ Σ›
by (*simp add*: *restriction-tendsto-shift-iff*)

lemma *restriction-tendsto-shiftD* : ‹$(\lambda n.\ \sigma\ (n + l))$ $-\downarrow\rightarrow$ $\Sigma \Longrightarrow \sigma$ $-\downarrow\rightarrow$ Σ›
by (*simp add*: *restriction-tendsto-shift-iff*)

lemma (**in** *restriction-space*) *restriction-tendsto-restricted-iff-restriction-tendsto* :
‹$(\lambda n.\ \sigma\ n \downarrow n)$ $-\downarrow\rightarrow$ $\Sigma \longleftrightarrow \sigma$ $-\downarrow\rightarrow$ Σ›
proof (*rule iffI*)
assume $*$: ‹$(\lambda n.\ \sigma\ n \downarrow n)$ $-\downarrow\rightarrow$ Σ›
show ‹σ $-\downarrow\rightarrow$ Σ›
proof (*rule restriction-tendstoI*)
fix n
from $*$ *restriction-tendstoE* **obtain** *n0* **where** ‹$n0 \leq k \Longrightarrow \Sigma \downarrow n = \sigma\ k \downarrow k \downarrow n$› **for** k **by** *blast*
hence ‹$max\ n0\ n \leq k \Longrightarrow \Sigma \downarrow n = \sigma\ k \downarrow n$› **for** k **by** *auto*
thus ‹$\exists$ *n0*. $\forall$ $k{\geq}n0$. $\Sigma \downarrow n = \sigma\ k \downarrow n$› **by** *blast*
qed
next
assume $*$: ‹σ $-\downarrow\rightarrow$ Σ›
show ‹$(\lambda n.\ \sigma\ n \downarrow n)$ $-\downarrow\rightarrow$ Σ›
proof (*rule restriction-tendstoI*)
fix n
from $*$ *restriction-tendstoE* **obtain** *n0* **where** ‹$n0 \leq k \Longrightarrow \Sigma \downarrow n = \sigma\ k \downarrow n$› **for** k **by** *blast*
hence ‹$max\ n0\ n \leq k \Longrightarrow \Sigma \downarrow n = \sigma\ k \downarrow k \downarrow n$› **for** k **by** *auto*
thus ‹$\exists$ *n0*. $\forall$ $k{\geq}n0$. $\Sigma \downarrow n = \sigma\ k \downarrow k \downarrow n$› **by** *blast*

qed
qed

lemma *restriction-tendsto-subseq* :
‹$(\sigma \circ f) \ -\downarrow\to \Sigma$› **if** ‹*strict-mono f*› **and** ‹$\sigma \ -\downarrow\to \Sigma$›
proof (*rule restriction-tendstoI*)
fix *n*
have ‹$n \leq f\, n$› **by** (*simp add*: *strict-mono-imp-increasing* ‹*strict-mono f*›)
moreover from ‹$\sigma \ -\downarrow\to \Sigma$› *restriction-tendstoE* **obtain** *n0* **where** ‹$n0 \leq k \Longrightarrow \Sigma \downarrow n = \sigma\ k \downarrow n$› **for** *k* **by** *blast*
ultimately have ‹$\forall\, k{\geq}n0.\ \Sigma \downarrow n = \sigma\ (f\, k) \downarrow n$›
by (*metis le-trans strict-mono-imp-increasing that(1)*)
thus ‹$\exists\, n0.\ \forall\, k{\geq}n0.\ \Sigma \downarrow n = (\sigma \circ f)\ k \downarrow n$› **by** *auto*
qed

end

context *restriction* **begin**

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

lemma *restriction-convergentI* : ‹$\sigma \ -\downarrow\to \Sigma \Longrightarrow$ *restriction-convergent* σ›
by (*auto simp add*: *restriction-convergent-def*)

lemma *restriction-convergentD'* : ‹*restriction-convergent* $\sigma \Longrightarrow \exists\, \Sigma.\ \sigma \ -\downarrow\to \Sigma$›
by (*simp add*: *restriction-convergent-def*)

end

context *restriction-space* **begin**

lemma *restriction-convergentD* :
‹*restriction-convergent* $\sigma \Longrightarrow \exists!\Sigma.\ \sigma \ -\downarrow\to \Sigma$›
unfolding *restriction-convergent-def* **using** *restriction-tendsto-unique*
by *blast*

lemma *restriction-convergentE* :
‹*restriction-convergent* $\sigma \Longrightarrow$
$(\bigwedge\Sigma.\ \sigma \ -\downarrow\to \Sigma \Longrightarrow (\bigwedge\Sigma'.\ \sigma \ -\downarrow\to \Sigma' \Longrightarrow \Sigma' = \Sigma) \Longrightarrow thesis) \Longrightarrow thesis$›
using *restriction-convergentD* **by** *blast*

lemma *restriction-tendsto-of-restriction-convergent* :

‹*restriction-convergent* $\sigma \Longrightarrow \sigma\ -\downarrow\rightarrow\ (\mathit{THE}\ \Sigma.\ \sigma\ -\downarrow\rightarrow\ \Sigma)$›
by (*simp add*: *restriction-convergentD the1I2*)

end

context *restriction* **begin**

lemma *restriction-convergent-const* [*simp*] : ‹$\mathit{convergent}_{\downarrow}\ (\lambda n.\ \Sigma)$›
unfolding *restriction-convergent-def restriction-tendsto-def* **by** *blast*

lemma (**in** *restriction-space*) *restriction-convergent-restrictions* [*simp*]
:
‹$\mathit{convergent}_{\downarrow}\ (\lambda n.\ \Sigma \downarrow n)$›
using *restriction-convergent-def restriction-tendsto-restrictions* **by**
blast

lemma *restriction-convergent-shift-iff* :
‹$\mathit{convergent}_{\downarrow}\ (\lambda n.\ \sigma\ (n + l)) \longleftrightarrow \mathit{convergent}_{\downarrow}\ \sigma$›
by (*simp add*: *restriction-convergent-def restriction-tendsto-shift-iff*)

lemma *restriction-convergent-shift-shiftI* :
‹$\mathit{convergent}_{\downarrow}\ \sigma \Longrightarrow \mathit{convergent}_{\downarrow}\ (\lambda n.\ \sigma\ (n + l))$›
by (*simp add*: *restriction-convergent-shift-iff*)

lemma *restriction-convergent-shift-shiftD* :
‹$\mathit{convergent}_{\downarrow}\ (\lambda n.\ \sigma\ (n + l)) \Longrightarrow \mathit{convergent}_{\downarrow}\ \sigma$›
by (*simp add*: *restriction-convergent-shift-iff*)

lemma (**in** *restriction-space*) *restriction-convergent-restricted-iff-restriction-convergent*
:
‹$\mathit{convergent}_{\downarrow}\ (\lambda n.\ \sigma\ n \downarrow n) \longleftrightarrow \mathit{convergent}_{\downarrow}\ \sigma$›
by (*simp add*: *restriction-convergent-def*
restriction-tendsto-restricted-iff-restriction-tendsto)

lemma *restriction-convergent-subseq* :
‹*strict-mono f* $\Longrightarrow$ *restriction-convergent* $\sigma \Longrightarrow$ *restriction-convergent*
$(\sigma \circ f)$›
unfolding *restriction-convergent-def* **using** *restriction-tendsto-subseq*
by *blast*

lemma (**in** *restriction-space*)
convergent-restriction-chain-imp-ex1 : ‹$\exists!\Sigma.\ \forall n.\ \Sigma \downarrow n = \sigma\ n$›
and *restriction-tendsto-of-convergent-restriction-chain* : ‹$\sigma\ -\downarrow\rightarrow$
$(\mathit{THE}\ \Sigma.\ \forall n.\ \Sigma \downarrow n = \sigma\ n)$›

if ‹*restriction-convergent* σ› **and** ‹*restriction-chain* σ›
proof −
from ‹*restriction-convergent* σ› *restriction-convergentE* **obtain** Σ
where ‹σ $-\downarrow\rightarrow$ Σ› ‹$\bigwedge\Sigma'.\ \sigma\ -\downarrow\rightarrow\ \Sigma' \Longrightarrow \Sigma' = \Sigma$› **by** *blast*

have $*$: ‹$\Sigma \downarrow n = \sigma\ n$› **for** n
proof −
from ‹σ $-\downarrow\rightarrow$ Σ› *restriction-tendstoE* **obtain** *n0* **where** ‹*n0* $\leq k$ $\Longrightarrow \Sigma \downarrow n = \sigma\ k \downarrow n$› **for** k **by** *blast*
hence ‹$\Sigma \downarrow n = \sigma\ (max\ n0\ n) \downarrow n$› **by** *simp*
thus ‹$\Sigma \downarrow n = \sigma\ n$›
by (*simp add*: *restriction-chain-def-ter*[*THEN iffD1*, *OF* ‹*restriction-chain* σ›])
qed
have $**$: ‹$\forall n.\ \Sigma' \downarrow n = \sigma\ n \Longrightarrow \Sigma' = \Sigma$› **for** Σ'
by (*metis* $*$ *ex-not-restriction-eq*)
from $*$ $**$ **show** ‹$\exists!\Sigma.\ \forall n.\ \Sigma \downarrow n = \sigma\ n$› **by** *blast*
from *theI'*[*OF this*] $**$ **have** ‹$\Sigma = (THE\ \Sigma.\ \forall n.\ \Sigma \downarrow n = \sigma\ n)$› **by** *simp*
with ‹σ $-\downarrow\rightarrow$ Σ› **show** ‹σ $-\downarrow\rightarrow$ $(THE\ \Sigma.\ \forall n.\ \Sigma \downarrow n = \sigma\ n)$› **by** *simp*
qed

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* : ‹$chain_{\downarrow}\ \sigma \Longrightarrow convergent_{\downarrow}\ \sigma$›

definition (**in** *complete-restriction-space*)
restriction-fix :: ‹$('a \Rightarrow 'a) \Rightarrow 'a$›
— We will use a syntax rather than a binder to be compatible with the product.
where ‹*restriction-fix* $(\lambda x.\ f\ x) \equiv THE\ \Sigma.\ (\lambda n.\ (f\ ^{\frown}\ n)\ undefined)$ $-\downarrow\rightarrow$ Σ›

syntax -*restriction-fix* :: ‹$[pttrn, 'a \Rightarrow 'a] \Rightarrow 'a$›
(‹(‹*indent=3 notation=‹binder restriction-fix›*›υ -./ -)› [*0*, *10*] *10*)
syntax-consts -*restriction-fix* $\rightleftharpoons$ *restriction-fix*
translations $\upsilon\ x.\ f \rightleftharpoons CONST\ restriction\text{-}fix\ (\lambda x.\ f)$
print-translation ‹
[(***const-syntax***‹*restriction-fix*›, *fn ctxt* => *fn* [*Abs abs*] =>

let val $(x,\ t)$ = *Syntax-Trans.atomic-abs-tr'* *ctxt abs*
in Syntax.const ***syntax-const***‹*-restriction-fix*› $ *x* $ *t end*)]
› — To avoid eta-contraction of body

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

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

lemma *restriction-chain-imp-ex1* : ‹$\exists!\Sigma.\ \forall n.\ \Sigma \downarrow n = \sigma\ n$›
and *restriction-tendsto-of-restriction-chain* : ‹$\sigma \longrightarrow\!\!\!\!\downarrow (THE\ \Sigma.\ \forall n.\ \Sigma \downarrow n = \sigma\ n)$›
if ‹*restriction-chain* σ›
by (*simp-all add*: *convergent-restriction-chain-imp-ex1*
restriction-tendsto-of-convergent-restriction-chain
restriction-chain-imp-restriction-convergent ‹*restriction-chain* σ›)

lemma *restriction-chain-is* :
‹$\sigma = (\downarrow)\ (THE\ \Sigma.\ \sigma \longrightarrow\!\!\!\!\downarrow \Sigma)$›
‹$\sigma = (\downarrow)\ (THE\ \Sigma.\ \forall n.\ \Sigma \downarrow n = \sigma\ n)$› **if** ‹*restriction-chain* σ›
proof −
from *restriction-tendsto-of-restriction-chain*[*OF* ‹*restriction-chain* σ›]
have ‹$\sigma \longrightarrow\!\!\!\!\downarrow (THE\ \Sigma.\ \forall n.\ \Sigma \downarrow n = \sigma\ n)$› .
with *restriction-tendsto-of-restriction-convergent*
restriction-convergentI restriction-tendsto-unique
have $*$: ‹$(THE\ \Sigma.\ \forall n.\ \Sigma \downarrow n = \sigma\ n) = (THE\ \Sigma.\ \sigma \longrightarrow\!\!\!\!\downarrow \Sigma)$› **by** *blast*

from *theI'*[*OF restriction-chain-imp-ex1*, *OF* ‹*restriction-chain* σ›]
show ‹$\sigma = (\downarrow)\ (THE\ \Sigma.\ \forall n.\ \Sigma \downarrow n = \sigma\ n)$› **by** (*intro ext*) *simp*
with $*$ **show** ‹$\sigma = (\downarrow)\ (THE\ \Sigma.\ \sigma \longrightarrow\!\!\!\!\downarrow \Sigma)$› **by** *auto*
qed

end

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

lemma *ex1-restriction-fix* :
‹$\exists!\Sigma.\ \forall x.\ (\lambda n.\ (f\ \widehat{\ }\widehat{\ }\ n)\ x) \longrightarrow\!\!\!\!\downarrow \Sigma$›
proof −
let *?*σ = ‹$\lambda x\ n.\ (f\ \widehat{\ }\widehat{\ }\ n)\ x \downarrow n$›
from *constructive-imp-eq-funpow-restricted*[*OF* ‹*constructive* f›]

have ‹*?σ x = ?σ y*› **for** *x y* **by** *blast*
moreover have ‹*restriction-chain* (*?σ x*)› **for** *x* **by** (*simp add*: ‹*constructive f*›)
ultimately obtain Σ **where** ‹∀ *x*. *?σ x* −↓→ Σ›
by (*metis* (*full-types*) *restriction-chain-is*(*1*) *restriction-tendsto-restrictions*)
with *restriction-tendsto-unique* **have** ‹∃!Σ. ∀ *x*. *?σ x* −↓→ Σ› **by** *blast*
thus ‹∃!Σ. ∀ *x*. (λ*n*. (*f* ^^ *n*) *x*) −↓→ Σ›
by (*simp add*: *restriction-tendsto-restricted-iff-restriction-tendsto*)
qed

lemma *ex1-restriction-fix-bis* :
‹∃!Σ. (λ*n*. (*f* ^^ *n*) *x*) −↓→ Σ›
using *ex1-restriction-fix restriction-tendsto-unique* **by** *blast*

lemma *restriction-fix-def-bis* :
‹(υ *x*. *f x*) = (*THE* Σ. (λ*n*. (*f* ^^ *n*) *x*) −↓→ Σ)›
proof −
have ‹(λΣ. (λ*n*. (*f* ^^ *n*) *undefined*) −↓→ Σ) = (λΣ. (λ*n*. (*f* ^^ *n*) *x*) −↓→ Σ)›
by (*metis ex1-restriction-fix restriction-tendsto-unique*)
from *arg-cong*[**where** *f* = *The*, *OF this*, *folded restriction-fix-def*]
show ‹(υ *x*. *f x*) = (*THE* Σ. (λ*n*. (*f* ^^ *n*) *x*) −↓→ Σ)› **.**
qed

lemma *funpow-restriction-tendsto-restriction-fix* : ‹(λ*n*. (*f* ^^ *n*) *x*) −↓→ (υ *x*. *f x*)›
by (*metis ex1-restriction-fix restriction-convergentI*
restriction-fix-def-bis restriction-tendsto-of-restriction-convergent)

lemma *restriction-restriction-fix-is* : ‹(υ *x*. *f x*) ↓ *n* = (*f* ^^ *n*) *x* ↓ *n*›
proof (*rule restriction-tendsto-unique*)
from *funpow-restriction-tendsto-restriction-fix*
show ‹(λ*k*. (*f* ^^ *k*) *x* ↓ *n*) −↓→ (υ *x*. *f x*) ↓ *n*›
by (*simp add*: *restriction-tendsto-def*)
next
from *restriction-tendsto-restrictions*
have ‹(λ*k*. (*f* ^^ *n*) *x* ↓ *n* ↓ *k*) −↓→ (*f* ^^ *n*) *x* ↓ *n*› **.**
then obtain *n0* **where** ∗ : ‹∀ *k*≥*n0*. (*f* ^^ *n*) *x* ↓ *n* = (*f* ^^ *n*) *x* ↓ *n* ↓ *k*›
by (*metis restriction-restriction min-def*)
show ‹(λ*k*. (*f* ^^ *k*) *x* ↓ *n*) −↓→ (*f* ^^ *n*) *x* ↓ *n*›
proof (*rule restriction-tendstoI*)
fix *m*
from ∗ **have** ‹∀ *k*≥*n* + *n0* + *m*. (*f* ^^ *n*) *x* ↓ *n* ↓ *m* = (*f* ^^ *k*) *x* ↓ *n* ↓ *m*›

by (*simp add*: ‹*constructive f*› *constructive-imp-eq-funpow-restricted*)
thus ‹∃ *n0*. ∀ *k*≥*n0*. (*f* ^^ *n*) *x* ↓ *n* ↓ *m* = (*f* ^^ *k*) *x* ↓ *n* ↓ *m*› **..**
qed
qed

lemma *restriction-fix-eq* : ‹(*υ x*. *f x*) = *f* (*υ x*. *f x*)›
proof (*subst restriction-fix-def*, *intro the1-equality restriction-tendstoI*)
show ‹∃!Σ. (λ*n*. (*f* ^^ *n*) *undefined*) −↓→ Σ›
by (*simp add*: *ex1-restriction-fix-bis*)
next
have ‹*n* ≤ *k* ⟹ *f* (*restriction-fix f*) ↓ *n* = (*f* ^^ *k*) *undefined* ↓ *n*›
for *n k*
by (*cases n*, *simp*, *cases k*, *simp-all*)
(*meson* ‹*constructive f*›[*THEN constructiveD*] *restriction-related-le*
restriction-restriction-fix-is)
thus ‹∃ *n0*. ∀ *k*≥*n0*. *f* (*restriction-fix f*) ↓ *n* = (*f* ^^ *k*) *undefined* ↓ *n*› **for** *n* **by** *blast*
qed

lemma *restriction-fix-unique* : ‹*f x* = *x* ⟹ (*υ x*. *f x*) = *x*›
by (*metis* (*no-types*, *opaque-lifting*) *restriction-fix-eq* ‹*constructive f*›
constructiveD dual-order.refl take-lemma-restriction)

lemma *restriction-fix-def-ter* : ‹(*υ x*. *f x*) = (*THE x*. *f x* = *x*)›
by (*metis* (*mono-tags*, *lifting*) *restriction-fix-eq restriction-fix-unique the-equality*)

end

3 Product over Restriction Spaces

3.1 Restriction Space

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

definition *restriction-prod* :: ‹'*a* × '*b* ⇒ *nat* ⇒ '*a* × '*b*›
where ‹*p* ↓ *n* ≡ (*fst p* ↓ *n*, *snd p* ↓ *n*)›

instance by *intro-classes* (*simp add*: *restriction-prod-def*)

end

instance *prod* :: (*restriction-space*, *restriction-space*) *restriction-space*
proof *intro-classes*
show ‹$p \downarrow 0 = q \downarrow 0$› **for** $p\ q$ :: ‹$'a \times 'b$›
by (*simp add*: *restriction-prod-def*)
next
show ‹$p \neq q \Longrightarrow \exists\, n.\ p \downarrow n \neq q \downarrow n$› **for** $p\ q$ :: ‹$'a \times 'b$›
by (*simp add*: *restriction-prod-def*)
(*meson ex-not-restriction-related prod.expand*)
qed

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 *intro-classes*
show ‹$p < q \longleftrightarrow p \leq q \wedge \neg\ q \leq p$› **for** $p\ q$ :: ‹$'a \times 'b$›
by (*auto simp add*: *less-eq-prod-def less-prod-def less-le-not-le*)
next
show ‹$p \leq p$› **for** p :: ‹$'a \times 'b$›
by (*simp add*: *less-eq-prod-def*)
next
show ‹$p \leq q \Longrightarrow q \leq r \Longrightarrow p \leq r$› **for** $p\ q\ r$ :: ‹$'a \times 'b$›
by (*auto simp add*: *less-eq-prod-def intro*: *order-trans*)
next
show ‹$p \downarrow 0 \leq q \downarrow 0$› **for** $p\ q$ :: ‹$'a \times 'b$›
by (*simp add*: *less-eq-prod-def restriction-prod-def*)
next

show ‹$p \leq q \Longrightarrow p \downarrow n \leq q \downarrow n$› **for** $p\ q :: \langle 'a \times 'b \rangle$ **and** n
by (*simp add*: *less-eq-prod-def restriction-prod-def mono-restriction-less-eq*)
next
show ‹$\neg\ p \leq q \Longrightarrow \exists n.\ \neg\ p \downarrow n \leq q \downarrow n$› **for** $p\ q :: \langle 'a \times 'b \rangle$
by (*simp add*: *less-eq-prod-def restriction-prod-def*)
(*meson ex-not-restriction-less-eq*)
qed

end

instance *prod* :: (*order-restriction-space*, *order-restriction-space*) *order-restriction-space*
by *intro-classes* (*simp add*: *less-eq-prod-def order-antisym prod.expand*)

3.2 Restriction shift Maps

3.2.1 Domain is a Product

lemma *restriction-shift-on-prod-domain-iff* :
‹*restriction-shift-on f k* $(A \times B) \longleftrightarrow (\forall x{\in}A.$ *restriction-shift-on* $(\lambda y.\ f\ (x,\ y))\ k\ B) \wedge$
$(\forall y{\in}B.$ *restriction-shift-on* $(\lambda x.\ f\ (x,\ y))\ k\ A)$›
proof (*intro iffI conjI ballI*)
show ‹*restriction-shift-on* $(\lambda y.\ f\ (x,\ y))\ k\ B$›
if ‹*restriction-shift-on f k* $(A \times B)$› **and** ‹$x \in A$› **for** x
proof (*rule restriction-shift-onI*)
show ‹$y1 \in B \Longrightarrow y2 \in B \Longrightarrow y1 \downarrow n = y2 \downarrow n \Longrightarrow$
$f\ (x,\ y1) \downarrow nat\ (int\ n + k) = f\ (x,\ y2) \downarrow nat\ (int\ n + k)$› **for** *y1 y2 n*
by (*rule that*(*1*)[*THEN restriction-shift-onD*])
(*use that*(*2*) **in** ‹*simp-all add*: *restriction-prod-def*›)
qed
next
show ‹*restriction-shift-on* $(\lambda x.\ f\ (x,\ y))\ k\ A$›
if ‹*restriction-shift-on f k* $(A \times B)$› **and** ‹$y \in B$› **for** y
proof (*rule restriction-shift-onI*)
show ‹$x1 \in A \Longrightarrow x2 \in A \Longrightarrow x1 \downarrow n = x2 \downarrow n \Longrightarrow$
$f\ (x1,\ y) \downarrow nat\ (int\ n + k) = f\ (x2,\ y) \downarrow nat\ (int\ n + k)$› **for** *x1 x2 n*
by (*rule that*(*1*)[*THEN restriction-shift-onD*])
(*use that*(*2*) **in** ‹*simp-all add*: *restriction-prod-def*›)
qed
next
assume *assm* : ‹$(\forall x{\in}A.$ *restriction-shift-on* $(\lambda y.\ f\ (x,\ y))\ k\ B) \wedge$
$(\forall y{\in}B.$ *restriction-shift-on* $(\lambda x.\ f\ (x,\ y))\ k\ A)$›
show ‹*restriction-shift-on f k* $(A \times B)$›
proof (*rule restriction-shift-onI*)
fix $p\ q\ n$ **assume** ‹$p \in A \times B$› ‹$q \in A \times B$› ‹$p \downarrow n = q \downarrow n$›

then obtain *x1 y1 x2 y2*
where ‹$p = (x1,\ y1)$› ‹$q = (x2,\ y2)$› ‹$x1 \in A$› ‹$y1 \in B$›
‹$x2 \in A$› ‹$y2 \in B$› ‹$x1 \downarrow n = x2 \downarrow n$› ‹$y1 \downarrow n = y2 \downarrow n$›
by (*cases p*, *cases q*, *simp add*: *restriction-prod-def*)
have ‹$p = (x1,\ y1)$› **by** (*fact* ‹$p = (x1,\ y1)$›)
also have ‹$f\ (x1,\ y1) \downarrow nat\ (int\ n + k) = f\ (x1,\ y2) \downarrow nat\ (int\ n + k)$›
by (*rule assm*[*THEN conjunct1*, *rule-format*, *OF* ‹$x1 \in A$›, *THEN restriction-shift-onD*])
(*fact* ‹$y1 \in B$› ‹$y2 \in B$› ‹$y1 \downarrow n = y2 \downarrow n$›)+
also have ‹$f\ (x1,\ y2) \downarrow nat\ (int\ n + k) = f\ (x2,\ y2) \downarrow nat\ (int\ n + k)$›
by (*rule assm*[*THEN conjunct2*, *rule-format*, *OF* ‹$y2 \in B$›, *THEN restriction-shift-onD*])
(*fact* ‹$x1 \in A$› ‹$x2 \in A$› ‹$x1 \downarrow n = x2 \downarrow n$›)+
also have ‹$(x2,\ y2) = q$› **by** (*simp add*: ‹$q = (x2,\ y2)$›)
finally show ‹$f\ p \downarrow nat\ (int\ n + k) = f\ q \downarrow nat\ (int\ n + k)$› **.**
qed
qed

lemma *restriction-shift-prod-domain-iff*:
‹*restriction-shift f k* $\longleftrightarrow$ ($\forall\, x.$ *restriction-shift* ($\lambda y.$ *f* ($x,\ y$)) *k*) $\wedge$
($\forall\, y.$ *restriction-shift* ($\lambda x.$ *f* ($x,\ y$)) *k*)›
unfolding *restriction-shift-def*
by (*metis UNIV-I UNIV-Times-UNIV restriction-shift-on-prod-domain-iff*)

lemma *non-too-destructive-on-prod-domain-iff* :
‹*non-too-destructive-on f* ($A \times B$) $\longleftrightarrow$ ($\forall\, x{\in}A.$ *non-too-destructive-on* ($\lambda y.$ *f* ($x,\ y$)) *B*) $\wedge$
($\forall\, y{\in}B.$ *non-too-destructive-on* ($\lambda x.$ *f* ($x,\ y$)) *A*)›
by (*simp add*: *non-too-destructive-on-def restriction-shift-on-prod-domain-iff*)

lemma *non-too-destructive-prod-domain-iff* :
‹*non-too-destructive f* $\longleftrightarrow$ ($\forall\, x.$ *non-too-destructive* ($\lambda y.$ *f* ($x,\ y$))) $\wedge$
($\forall\, y.$ *non-too-destructive* ($\lambda x.$ *f* ($x,\ y$)))›
unfolding *non-too-destructive-def*
by (*metis UNIV-I UNIV-Times-UNIV non-too-destructive-on-prod-domain-iff*)

lemma *non-destructive-on-prod-domain-iff* :
‹*non-destructive-on f* ($A \times B$) $\longleftrightarrow$ ($\forall\, x{\in}A.$ *non-destructive-on* ($\lambda y.$ *f* ($x,\ y$)) *B*) $\wedge$
($\forall\, y{\in}B.$ *non-destructive-on* ($\lambda x.$ *f* ($x,\ y$)) *A*)›
by (*simp add*: *non-destructive-on-def restriction-shift-on-prod-domain-iff*)

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*)))›
unfolding *non-destructive-def*
by (*metis UNIV-I UNIV-Times-UNIV non-destructive-on-prod-domain-iff*)

lemma *constructive-on-prod-domain-iff* :
‹*constructive-on f* (*A* × *B*) ⟷ (∀ *x*∈*A*. *constructive-on* (λ*y*. *f* (*x*, *y*)) *B*) ∧
(∀ *y*∈*B*. *constructive-on* (λ*x*. *f* (*x*, *y*)) *A*)›
by (*simp add*: *constructive-on-def restriction-shift-on-prod-domain-iff*)

lemma *constructive-prod-domain-iff* :
‹*constructive f* ⟷ (∀ *x*. *constructive* (λ*y*. *f* (*x*, *y*))) ∧
(∀ *y*. *constructive* (λ*x*. *f* (*x*, *y*)))›
unfolding *constructive-def*
by (*metis UNIV-I UNIV-Times-UNIV constructive-on-prod-domain-iff*)

lemma *restriction-shift-prod-domain* [*restriction-shift-simpset*, *restriction-shift-introset*] :
‹⟦⋀*x*. *restriction-shift* (λ*y*. *f* (*x*, *y*)) *k*;
⋀*y*. *restriction-shift* (λ*x*. *f* (*x*, *y*)) *k*⟧ ⟹ *restriction-shift f k*›
and *non-too-destructive-prod-domain* [*restriction-shift-simpset*, *restriction-shift-introset*] :
‹⟦⋀*x*. *non-too-destructive* (λ*y*. *f* (*x*, *y*));
⋀*y*. *non-too-destructive* (λ*x*. *f* (*x*, *y*))⟧ ⟹ *non-too-destructive f*›
and *non-destructive-prod-domain* [*restriction-shift-simpset*, *restriction-shift-introset*] :
‹⟦⋀*x*. *non-destructive* (λ*y*. *f* (*x*, *y*));
⋀*y*. *non-destructive* (λ*x*. *f* (*x*, *y*))⟧ ⟹ *non-destructive f*›
and *constructive-prod-domain* [*restriction-shift-simpset*, *restriction-shift-introset*] :
‹⟦⋀*x*. *constructive* (λ*y*. *f* (*x*, *y*));
⋀*y*. *constructive* (λ*x*. *f* (*x*, *y*))⟧ ⟹ *constructive f*›
by (*simp-all add*: *restriction-shift-prod-domain-iff non-too-destructive-prod-domain-iff non-destructive-prod-domain-iff constructive-prod-domain-iff*)

3.2.2 Codomain is a Product

lemma *restriction-shift-on-prod-codomain-iff* :
‹*restriction-shift-on f k A* ⟷ (*restriction-shift-on* (λ*x*. *fst* (*f x*)) *k* *A*) ∧
(*restriction-shift-on* (λ*x*. *snd* (*f x*)) *k A*)›
by (*simp add*: *restriction-shift-on-def restriction-prod-def*) *blast*

lemma *restriction-shift-prod-codomain-iff*:
‹*restriction-shift f k* ⟷ (*restriction-shift* (λ*x*. *fst* (*f x*)) *k*) ∧
(*restriction-shift* (λ*x*. *snd* (*f x*)) *k*)›
unfolding *restriction-shift-def*
by (*fact restriction-shift-on-prod-codomain-iff*)

lemma *non-too-destructive-on-prod-codomain-iff* :
‹*non-too-destructive-on f A* ⟷ (*non-too-destructive-on* (λ*x*. *fst* (*f x*)) *A*) ∧
(*non-too-destructive-on* (λ*x*. *snd* (*f x*)) *A*)›
by (*simp add*: *non-too-destructive-on-def restriction-shift-on-prod-codomain-iff*)

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*)))›
by (*simp add*: *non-too-destructive-def non-too-destructive-on-prod-codomain-iff*)

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*)›

by (*simp add*: *non-destructive-on-def restriction-shift-on-prod-codomain-iff*)

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

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*)›
by (*simp add*: *constructive-on-def restriction-shift-on-prod-codomain-iff*)

lemma *constructive-prod-codomain-iff* :
‹*constructive f* ⟷ (*constructive* (λ*x*. *fst* (*f x*))) ∧
(*constructive* (λ*x*. *snd* (*f x*)))›
by (*simp add*: *constructive-def constructive-on-prod-codomain-iff*)

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*

$(\lambda x.\ (f\ x,\ g\ x))\rangle$
and *non-destructive-prod-codomain* [*restriction-shift-simpset*, *restriction-shift-introset*] :
$\langle [\![\textit{non-destructive}\ f;\ \textit{non-destructive}\ g]\!] \Longrightarrow \textit{non-destructive}\ (\lambda x.\ (f\ x,\ g\ x))\rangle$
and *constructive-prod-codomain* [*restriction-shift-simpset*, *restriction-shift-introset*] :
$\langle [\![\textit{constructive}\ f;\ \textit{constructive}\ g]\!] \Longrightarrow \textit{constructive}\ (\lambda x.\ (f\ x,\ g\ x))\rangle$
by (*simp-all add*: *restriction-shift-prod-codomain-iff non-too-destructive-prod-codomain-iff non-destructive-prod-codomain-iff constructive-prod-codomain-iff*)

3.3 Limits and Convergence

lemma *restriction-chain-prod-iff* :
$\langle \textit{restriction-chain}\ \sigma \longleftrightarrow \textit{restriction-chain}\ (\lambda n.\ \textit{fst}\ (\sigma\ n)) \wedge$
$\textit{restriction-chain}\ (\lambda n.\ \textit{snd}\ (\sigma\ n))\rangle$
by (*simp add*: *restriction-chain-def restriction-prod-def*)
(*metis fst-conv prod.collapse snd-conv*)

lemma *restriction-tendsto-prod-iff* :
$\langle \sigma \longrightarrow\!\downarrow \Sigma \longleftrightarrow (\lambda n.\ \textit{fst}\ (\sigma\ n)) \longrightarrow\!\downarrow \textit{fst}\ \Sigma \wedge (\lambda n.\ \textit{snd}\ (\sigma\ n)) \longrightarrow\!\downarrow \textit{snd}\ \Sigma\rangle$
by (*simp add*: *restriction-tendsto-def restriction-prod-def*)
(*meson nle-le order-trans*)

lemma *restriction-convergent-prod-iff* :
$\langle \textit{restriction-convergent}\ \sigma \longleftrightarrow \textit{restriction-convergent}\ (\lambda n.\ \textit{fst}\ (\sigma\ n)) \wedge$
$\textit{restriction-convergent}\ (\lambda n.\ \textit{snd}\ (\sigma\ n))\rangle$
by (*simp add*: *restriction-convergent-def restriction-tendsto-prod-iff*)

lemma *funpow-indep-prod-is* :
$\langle ((\lambda(x,\ y).\ (f\ x,\ g\ y))\ \widehat{\ }\widehat{\ }\ n)\ (x,\ y) = ((f\ \widehat{\ }\widehat{\ }\ n)\ x,\ (g\ \widehat{\ }\widehat{\ }\ n)\ y)\rangle$
for $f\ g :: \langle 'a \Rightarrow 'a\rangle$
by (*induct n*) *simp-all*

3.4 Completeness

instance *prod* :: (*complete-restriction-space*, *complete-restriction-space*) *complete-restriction-space*
proof *intro-classes*
fix $\sigma :: \langle nat \Rightarrow 'a :: \textit{complete-restriction-space} \times 'b :: \textit{complete-restriction-space}\rangle$
assume $\langle \textit{chain}_{\downarrow}\ \sigma\rangle$
hence $\langle \textit{chain}_{\downarrow}\ (\lambda n.\ \textit{fst}\ (\sigma\ n))\rangle\ \langle \textit{chain}_{\downarrow}\ (\lambda n.\ \textit{snd}\ (\sigma\ n))\rangle$
by (*simp-all add*: *restriction-chain-prod-iff*)
hence $\langle \textit{convergent}_{\downarrow}\ (\lambda n.\ \textit{fst}\ (\sigma\ n))\rangle\ \langle \textit{convergent}_{\downarrow}\ (\lambda n.\ \textit{snd}\ (\sigma\ n))\rangle$
by (*simp-all add*: *restriction-chain-imp-restriction-convergent*)
thus $\langle \textit{convergent}_{\downarrow}\ \sigma\rangle$
by (*simp add*: *restriction-convergent-prod-iff*)
qed

3.5 Fixed Point

lemma *restriction-fix-indep-prod-is* :
‹(υ (x, y). (f x, g y)) = (υ x. f x, υ y. g y)›
if *contructive* : ‹*constructive f*› ‹*constructive g*›
for *f* :: ‹'*a* ⇒ '*a* :: *complete-restriction-space*›
and *g* :: ‹'*b* ⇒ '*b* :: *complete-restriction-space*›
proof (*rule restriction-fix-unique*)
from *contructive* **show** ‹*constructive* (λ(x, y). (*f x*, *g y*))›
by (*simp add*: *constructive-prod-domain-iff constructive-prod-codomain-iff constructive-const*)
next
show ‹(*case* (υ x. *f x*, υ y. *g y*) *of* (x, y) ⇒ (*f x*, *g y*)) = (υ x. *f x*, υ y. *g y*)›
by *simp* (*metis restriction-fix-eq contructive*)
qed

lemma *non-destructive-fst* : ‹*non-destructive fst*›
by (*rule non-destructiveI*) (*simp add*: *restriction-prod-def*)

lemma *non-destructive-snd* : ‹*non-destructive snd*›
by (*rule non-destructiveI*) (*simp add*: *restriction-prod-def*)

lemma *constructive-restriction-fix-right* :
‹*constructive* (λx. υ y. *f* (x, y))› **if** ‹*constructive f*›
for *f* :: ‹'*a* :: *complete-restriction-space* × '*b* :: *complete-restriction-space* ⇒ '*b*›
proof (*rule constructiveI*)
fix *n* **and** *x x'* :: '*a* **assume** ‹*x* ↓ *n* = *x'* ↓ *n*›
show ‹(υ y. *f* (x, y)) ↓ *Suc n* = (υ y. *f* (*x'*, y)) ↓ *Suc n*›
proof (*subst* (*1 2*) *restriction-restriction-fix-is*)
show ‹*constructive* (λy. *f* (*x'*, y))› **and** ‹*constructive* (λy. *f* (x, y))›
using ‹*constructive f*› *constructive-prod-domain-iff* **by** *blast+*
next
from ‹*x* ↓ *n* = *x'* ↓ *n*› **show** ‹((λy. *f* (x, y)) ⁀⁀ *Suc n*) *undefined* ↓ *Suc n* =
((λy. *f* (*x'*, y)) ⁀⁀ *Suc n*) *undefined* ↓ *Suc n*›
by (*induct n*, *simp-all*)
(*use* ‹*constructive f*› *constructiveD restriction-0-related* **in** *blast*,
metis (*no-types*, *lifting*) ‹*constructive f*› *constructiveD prod.sel restriction-related-pred restriction-prod-def*)
qed
qed

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 (*rule constructiveI*)
  fix *n* **and** *y y*′ :: ′*b* **assume** ‹*y* ↓ *n* = *y*′ ↓ *n*›
  show ‹(∪ *x*. *f* (*x*, *y*)) ↓ *Suc n* = (∪ *x*. *f* (*x*, *y*′)) ↓ *Suc n*›
  proof (*subst* (*1 2*) *restriction-restriction-fix-is*)
    show ‹*constructive* (λ*x*. *f* (*x*, *y*′))› **and** ‹*constructive* (λ*x*. *f* (*x*,
y))›
      using ‹*constructive f*› *constructive-prod-domain-iff* **by** *blast*+
  next
    from ‹*y* ↓ *n* = *y*′ ↓ *n*› **show** ‹((λ*x*. *f* (*x*, *y*)) ⁀⁀ *Suc n*) *undefined*
↓ *Suc n* =
                                ((λ*x*. *f* (*x*, *y*′)) ⁀⁀ *Suc n*) *undefined* ↓ *Suc n*›
      by (*induct n*, *simp-all*)
       (*use restriction-0-related* ‹*constructive f*› *constructiveD* **in** *blast*,
         metis (*no-types*, *lifting*) ‹*constructive f*› *constructiveD prod.sel*
          restriction-related-pred restriction-prod-def)
  qed
qed

— “Bekic's Theorem” in HOLCF

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 (*rule restriction-fix-unique*[*OF* ‹*constructive f*›])
  have ‹*constructive* (λ*p*. *snd* (*f p*))›
    by (*fact non-destructive-comp-constructive*[*OF non-destructive-snd*
‹*constructive f*›])
  hence ‹*constructive* (λ*x*. ∪ *y*. *snd* (*f* (*x*, *y*)))›
    by (*fact constructive-restriction-fix-right*)
  hence ‹*non-destructive* (λ*x*. ∪ *y*. *snd* (*f* (*x*, *y*)))›
    by (*fact constructive-imp-non-destructive*)
  hence ‹*non-destructive* (λ*x*. (*x*, ∪ *y*. *snd* (*f* (*x*, *y*))))›
    by (*fact non-destructive-prod-codomain*[*OF non-destructive-id*(*2*)])
  hence ‹*constructive* (λ*x*. *f* (*x*, ∪ *y*. *snd* (*f* (*x*, *y*))))›
    by (*fact constructive-comp-non-destructive*[*OF* ‹*constructive f*›])
  hence ∗ : ‹*constructive* (λ*x*. *fst* (*f* (*x*, ∪ *y*. *snd* (*f* (*x*, *y*)))))›
   by (*fact non-destructive-comp-constructive*[*OF non-destructive-fst*])

  have ‹*non-destructive* (λ*x*. ∪ *x*. *fst* (*f* (*x*, ∪ *y*. *snd* (*f* (*x*, *y*)))))›

by (*fact constructive-imp-non-destructive*[*OF constructive-const*])
hence ‹*non-destructive* (*Pair* (υ *x*. *fst* (*f* (*x*, υ *y*. *snd* (*f* (*x*, *y*))))))›
by (*fact non-destructive-prod-codomain*[*OF - non-destructive-id*(*2*)])
hence ‹*constructive* (λ*x*. *f* (υ *x*. *fst* (*f* (*x*, υ *y*. *snd* (*f* (*x*, *y*)))), *x*))›
by (*fact constructive-comp-non-destructive*[*OF* ‹*constructive f*›])
hence ∗∗ : ‹*constructive* (λ*x*. *snd* (*f* (υ *x*. *fst* (*f* (*x*, υ *y*. *snd* (*f* (*x*, *y*)))), *x*)))›
by (*fact non-destructive-comp-constructive*[*OF non-destructive-snd*])

have ‹*fst* (*f* (*?x*, *?y*)) = *?x*›
by (*rule trans* [*symmetric*, *OF restriction-fix-eq*[*OF* ∗]]) *simp*
moreover have ‹*snd* (*f* (*?x*, *?y*)) = *?y*›
by (*rule trans* [*symmetric*, *OF restriction-fix-eq*[*OF* ∗∗]]) *simp*
ultimately show ‹*f* (*?x*, *?y*) = (*?x*, *?y*)› **by** (*cases* ‹*f* (*?x*, *?y*)›) *simp*
qed

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 by *intro-classes* (*simp add*: *restriction-fun-def*)

end

instance ‹*fun*› :: (*type*, *restriction-space*) *restriction-space*
proof (*intro-classes*, *unfold restriction-fun-def*)
show ‹(λ*x*. *f x* ↓ *0*) = (λ*x*. *g x* ↓ *0*)›
for *f g* :: ‹'*a* ⇒ '*b*› **by** (*rule ext*) *simp*
next
fix *f g* :: ‹'*a* ⇒ '*b*› **assume** ‹*f* ≠ *g*›
then obtain *x* **where** ‹*f x* ≠ *g x*› **by** *fast*
with *ex-not-restriction-related* **obtain** *n*
where ‹*f x* ↓ *n* ≠ *g x* ↓ *n*› **by** *blast*
hence ‹(λ*x*. *f x* ↓ *n*) ≠ (λ*x*. *g x* ↓ *n*)› **by** *meson*
thus ‹∃ *n*. (λ*x*. *f x* ↓ *n*) ≠ (λ*x*. *g x* ↓ *n*)› **..**
qed

instance ‹*fun*› :: (*type*, *preorder-restriction-space*) *preorder-restriction-space*
proof *intro-classes*
show ‹$f \downarrow 0 \leq g \downarrow 0$› **for** $f\ g :: \langle 'a \Rightarrow 'b \rangle$
by (*simp add*: *le-fun-def restriction-fun-def*)
next
show ‹$f \leq g \Longrightarrow f \downarrow n \leq g \downarrow n$› **for** $f\ g :: \langle 'a \Rightarrow 'b \rangle$ **and** n
by (*simp add*: *restriction-fun-def le-fun-def mono-restriction-less-eq*)
next
show ‹$\neg\ f \leq g \Longrightarrow \exists\, n.\ \neg\ f \downarrow n \leq g \downarrow n$› **for** $f\ g :: \langle 'a \Rightarrow 'b \rangle$
by (*metis ex-not-restriction-less-eq le-funD le-funI restriction-fun-def*)
qed

instance ‹*fun*› :: (*type*, *order-restriction-space*) *order-restriction-space*
..

4.2 Restriction shift Maps

lemma *restriction-shift-on-fun-iff* :
‹*restriction-shift-on f k A* $\longleftrightarrow$ ($\forall\, z.$ *restriction-shift-on* ($\lambda x.$ *f x z*) *k A*)›
proof (*intro iffI allI*)
show ‹*restriction-shift-on* ($\lambda x.$ *f x z*) *k A*› **if** ‹*restriction-shift-on f k A*› **for** *z*
proof (*rule restriction-shift-onI*)
fix *x y n* **assume** ‹$x \in A$› ‹$y \in A$› ‹$x \downarrow n = y \downarrow n$›
from *restriction-shift-onD*[*OF* ‹*restriction-shift-on f k A*› *this*]
show ‹*f x z* $\downarrow$ *nat* (*int n* + *k*) = *f y z* $\downarrow$ *nat* (*int n* + *k*)›
by (*unfold restriction-fun-def*) (*blast dest*!: *fun-cong*)
qed
next
show ‹*restriction-shift-on f k A*› **if** ‹$\forall\, z.$ *restriction-shift-on* ($\lambda x.$ *f x z*) *k A*›
proof (*rule restriction-shift-onI*)
fix *x y n* **assume** ‹$x \in A$› ‹$y \in A$› ‹$x \downarrow n = y \downarrow n$›
with ‹$\forall\, z.$ *restriction-shift-on* ($\lambda x.$ *f x z*) *k A*› *restriction-shift-onD*
have ‹*f x z* $\downarrow$ *nat* (*int n* + *k*) = *f y z* $\downarrow$ *nat* (*int n* + *k*)› **for** *z* **by** *blast*
thus ‹*f x* $\downarrow$ *nat* (*int n* + *k*) = *f y* $\downarrow$ *nat* (*int n* + *k*)›
by (*simp add*: *restriction-fun-def*)
qed
qed

lemma *restriction-shift-fun-iff* : ‹*restriction-shift f k* $\longleftrightarrow$ ($\forall\, z.$ *restriction-shift* ($\lambda x.$ *f x z*) *k*)›
by (*unfold restriction-shift-def*, *fact restriction-shift-on-fun-iff*)

lemma *non-too-destructive-on-fun-iff*:
‹*non-too-destructive-on f A* $\longleftrightarrow$ ($\forall\, z.$ *non-too-destructive-on* ($\lambda x.$ *f*

x z) *A*)›
by (*simp add*: *non-too-destructive-on-def restriction-shift-on-fun-iff*)

lemma *non-too-destructive-fun-iff*:
‹*non-too-destructive f* ⟷ (∀ *z*. *non-too-destructive* (λ*x*. *f x z*))›
by (*unfold restriction-shift-def non-too-destructive-def*)
(*fact non-too-destructive-on-fun-iff*)

lemma *non-destructive-on-fun-iff*:
‹*non-destructive-on f A* ⟷ (∀ *z*. *non-destructive-on* (λ*x*. *f x z*) *A*)›
by (*simp add*: *non-destructive-on-def restriction-shift-on-fun-iff*)

lemma *non-destructive-fun-iff*:
‹*non-destructive f* ⟷ (∀ *z*. *non-destructive* (λ*x*. *f x z*))›
unfolding *non-destructive-def* **by** (*fact non-destructive-on-fun-iff*)

lemma *constructive-on-fun-iff*:
‹*constructive-on f A* ⟷ (∀ *z*. *constructive-on* (λ*x*. *f x z*) *A*)›
by (*simp add*: *constructive-on-def restriction-shift-on-fun-iff*)

lemma *constructive-fun-iff*:
‹*constructive f* ⟷ (∀ *z*. *constructive* (λ*x*. *f x z*))›
unfolding *constructive-def* **by** (*fact constructive-on-fun-iff*)

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* (λ*x*. *f x z*)) ⟹ *constructive f*›
by (*simp-all add*: *restriction-shift-fun-iff non-too-destructive-fun-iff*
non-destructive-fun-iff constructive-fun-iff)

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 −
let *?n* = ‹*Sup* (*restriction-related-set f g*)›
from *Sup-in-restriction-related-set*[*OF* ‹*f* ≠ *g*›]
have ‹*?n* ∈ *restriction-related-set f g*› **.**
with *restriction-related-le* **have** ‹∀ *m*≤*?n*. *f* ↓ *m* = *g* ↓ *m*› **by** *blast*
moreover have ‹*f* ↓ *Suc ?n* ≠ *g* ↓ *Suc ?n*›
using *cSup-upper*[*OF - bdd-above-restriction-related-set-iff*[*THEN iffD2*, *OF* ‹*f* ≠ *g*›], *of* ‹*Suc ?n*›]
by (*metis* (*mono-tags*, *lifting*) *dual-order.refl mem-Collect-eq not-less-eq-eq restriction-related-le*)
ultimately obtain *x* **where** ∗ : ‹∀ *m*≤*?n*. *f x* ↓ *m* = *g x* ↓ *m*› ‹*f x* ↓ *Suc ?n* ≠ *g x* ↓ *Suc ?n*›
unfolding *restriction-fun-def* **by** *meson*
from ∗(*2*) **have** ‹*f x* ≠ *g x*› **by** *auto*
moreover from ∗ **have** ‹*?n* = *Sup* (*restriction-related-set* (*f x*) (*g x*))›
by (*metis* (*no-types*, *lifting*) ‹∀ *m*≤*?n*. *f* ↓ *m* = *g* ↓ *m*›
‹*f* ↓ *Suc ?n* ≠ *g* ↓ *Suc ?n*› *not-less-eq-eq restriction-related-le*)
ultimately show *thesis* **using** *that* **by** *blast*
qed

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 (*cases* ‹*f* = *g*›)
from *that* **show** ‹*f* = *g* ⟹ *thesis*› **by** *simp*
next
from *that* **show** ‹*f* ≠ *g* ⟹ *thesis*›
by (*elim reached-dist-funE*) (*metis* (*full-types*) *restriction-related-set-is-atMost*)
qed

lemma *restriction-chain-fun-iff* :
‹*restriction-chain σ* ⟷ (∀ *z*. *restriction-chain* (λ*n*. *σ n z*))›
proof (*intro iffI allI*)
show ‹*restriction-chain σ* ⟹ *restriction-chain* (λ*n*. *σ n z*)› **for** *z*
by (*auto simp add*: *restriction-chain-def restriction-fun-def dest*!: *fun-cong*)
next
show ‹∀ *z*. *restriction-chain* (λ*n*. *σ n z*) ⟹ *restriction-chain σ*›
by (*simp add*: *restriction-chain-def restriction-fun-def*)
qed

lemma *restriction-tendsto-fun-imp* : ‹σ $-\downarrow\to$ Σ $\Longrightarrow$ $(\lambda n.\ \sigma\ n\ z)$ $-\downarrow\to$ Σ z›
by (*simp add*: *restriction-tendsto-def restriction-fun-def*) *meson*

lemma *restriction-convergent-fun-imp* :
‹*restriction-convergent* σ $\Longrightarrow$ *restriction-convergent* $(\lambda n.\ \sigma\ n\ z)$›
by (*metis restriction-convergent-def restriction-tendsto-fun-imp*)

4.4 Completeness

instance ‹*fun*› :: (*type*, *complete-restriction-space*) *complete-restriction-space*
proof *intro-classes*
fix σ :: ‹*nat* $\Rightarrow$ $'a$ $\Rightarrow$ $'b$ :: *complete-restriction-space*›
assume ‹*restriction-chain* σ›
hence $*$: ‹*restriction-chain* $(\lambda n.\ \sigma\ n\ x)$› **for** x
by (*simp add*: *restriction-chain-fun-iff*)
from *restriction-chain-imp-restriction-convergent*[*OF this*]
have $**$: ‹*restriction-convergent* $(\lambda n.\ \sigma\ n\ x)$› **for** x **.**
then obtain Σ **where** $***$: ‹$(\lambda n.\ \sigma\ n\ x)$ $-\downarrow\to$ Σ x› **for** x
by (*meson restriction-convergent-def*)
from $*$ **have** $****$: ‹$(\lambda n.\ \sigma\ n\ x \downarrow n) = (\lambda n.\ \sigma\ n\ x)$› **for** x
by (*simp add*: *restricted-restriction-chain-is*)
have ‹σ $-\downarrow\to$ Σ›
proof (*rule restriction-tendstoI*)
fix n
have ‹$\forall\, k{\geq}n.\ \Sigma\ x \downarrow n = \sigma\ k\ x \downarrow n$› **for** x
by (*metis* $*$ $**$ $***$ $****$ *restriction-related-le restriction-chain-is*(*1*)
restriction-tendsto-of-restriction-convergent restriction-tendsto-unique)
hence ‹$\forall\, k{\geq}n.\ \Sigma \downarrow n = \sigma\ k \downarrow n$› **by** (*simp add*: *restriction-fun-def*)
thus ‹$\exists\, n0.\ \forall\, k{\geq}n0.\ \Sigma \downarrow n = \sigma\ k \downarrow n$› **by** *blast*
qed

thus ‹*restriction-convergent* σ› **by** (*fact restriction-convergentI*)
qed

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 ⟶ (λn. *f* (σ *n*)) $-\downarrow\rightarrow$ *f* Σ›

lemma *restriction-cont-atI* : ‹(⋀σ. σ $-\downarrow\rightarrow$ Σ ⟹ (λn. *f* (σ *n*)) $-\downarrow\rightarrow$ *f* Σ) ⟹ $cont_{\downarrow}$ *f at* Σ›
by (*simp add*: *restriction-cont-at-def*)

lemma *restriction-cont-atD* : ‹$cont_{\downarrow}$ *f at* Σ ⟹ σ $-\downarrow\rightarrow$ Σ ⟹ (λn. *f* (σ *n*)) $-\downarrow\rightarrow$ *f* Σ›
by (*simp add*: *restriction-cont-at-def*)

lemma *restriction-cont-at-comp* [*restriction-cont-simpset*] :
‹$cont_{\downarrow}$ *f at* Σ ⟹ $cont_{\downarrow}$ *g at* (*f* Σ) ⟹ $cont_{\downarrow}$ (λx. *g* (*f x*)) *at* Σ›
by (*simp add*: *restriction-cont-at-def restriction-class.restriction-cont-at-def*)

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⟧
⟹ $cont_{\downarrow}$ (λy. *if P x then f x y else g x y*) *at* Σ›
by (*auto intro*!: *restriction-cont-atI*) (*blast dest*: *restriction-cont-atD*)+

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*›
by (*simp add*: *restriction-open-def*)

lemma *restriction-openD* : ‹$open_{\downarrow}$ *U* ⟹ Σ ∈ *U* ⟹ σ $-\downarrow\rightarrow$ Σ ⟹ ∃*n0*. ∀*k*≥*n0*. σ *k* ∈ *U*›
by (*simp add*: *restriction-open-def*)

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

lemma *restriction-open-UNIV* [*simp*] : ‹$open_{\downarrow}$ *UNIV*›
and *restriction-open-empty* [*simp*] : ‹$open_{\downarrow}$ {}›
by (*simp-all add*: *restriction-open-def*)

lemma *restriction-open-union* :
‹$open_{\downarrow}$ *U* ⟹ $open_{\downarrow}$ *V* ⟹ $open_{\downarrow}$ (*U* ∪ *V*)›
by (*metis Un-iff restriction-open-def*)

lemma *restriction-open-Union* :
‹$(\bigwedge i.\ i \in I \Longrightarrow open_{\downarrow}\ (U\ i)) \Longrightarrow open_{\downarrow}\ (\bigcup i{\in}I.\ U\ i)$›
by (*rule restriction-openI*) (*metis UN-iff restriction-openD*)

lemma *restriction-open-inter* :
‹$open_{\downarrow}\ (U \cap V)$› **if** ‹$open_{\downarrow}\ U$› **and** ‹$open_{\downarrow}\ V$›
proof (*rule restriction-openI*)
fix $\Sigma\ \sigma$ **assume** ‹$\Sigma \in U \cap V$› ‹$\sigma\ {-}{\downarrow}{\rightarrow}\ \Sigma$›
from ‹$\Sigma \in U \cap V$› **have** ‹$\Sigma \in U$› **and** ‹$\Sigma \in V$› **by** *simp-all*
from ‹$open_{\downarrow}\ U$› ‹$\Sigma \in U$› ‹$\sigma\ {-}{\downarrow}{\rightarrow}\ \Sigma$› *restriction-openD*
obtain *n0* **where** ‹$\forall\, k{\geq}n0.\ \sigma\ k \in U$› **by** *blast*
moreover from ‹$open_{\downarrow}\ V$› ‹$\Sigma \in V$› ‹$\sigma\ {-}{\downarrow}{\rightarrow}\ \Sigma$› *restriction-openD*
obtain *n1* **where** ‹$\forall\, k{\geq}n1.\ \sigma\ k \in V$› **by** *blast*
ultimately have ‹$\forall\, k{\geq}max\ n0\ n1.\ \sigma\ k \in U \cap V$› **by** *simp*
thus ‹$\exists\, n0.\ \forall\, k{\geq}n0.\ \sigma\ k \in U \cap V$› **by** *blast*
qed

lemma *restriction-open-finite-Inter* :
‹*finite* $I \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow open_{\downarrow}\ (U\ i)) \Longrightarrow open_{\downarrow}\ (\bigcap i{\in}I.\ U\ i)$›
by (*induct I rule*: *finite-induct*)
(*simp-all add*: *restriction-open-inter*)

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

lemma *restriction-closedI* : ‹$(\bigwedge\Sigma\ \sigma.\ \Sigma \notin S \Longrightarrow \sigma\ {-}{\downarrow}{\rightarrow}\ \Sigma \Longrightarrow \exists\, n0.\ \forall\, k{\geq}n0.\ \sigma\ k \notin S) \Longrightarrow closed_{\downarrow}\ S$›
by (*simp add*: *restriction-closed-def restriction-open-def*)

lemma *restriction-closedD* : ‹$closed_{\downarrow}\ S \Longrightarrow \Sigma \notin S \Longrightarrow \sigma\ {-}{\downarrow}{\rightarrow}\ \Sigma \Longrightarrow \exists\, n0.\ \forall\, k{\geq}n0.\ \sigma\ k \notin S$›
by (*simp add*: *restriction-closed-def restriction-open-def*)

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

lemma *restriction-closed-UNIV* [*simp*] : ‹$closed_{\downarrow}\ UNIV$›
and *restriction-closed-empty* [*simp*] : ‹$closed_{\downarrow}\ \{\}$›
by (*simp-all add*: *restriction-closed-def*)

end

5.2 Balls

context *restriction* **begin**

definition *restriction-cball* :: ‹'*a* ⇒ *nat* ⇒ '*a set*› (‹$\mathcal{B}_\downarrow$'(-, -')›)
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* : ‹$x \in \mathcal{B}_\downarrow(a,\ n) \Longrightarrow x \downarrow n = a \downarrow n$›
by (*simp-all add*: *restriction-cball-def*)

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

lemma ‹$x \in$ *restriction-ball a n* $\longleftrightarrow\ x \downarrow$ *Suc n* = *a* $\downarrow$ *Suc n*›
and ‹$x \downarrow$ *Suc n* = *a* $\downarrow$ *Suc n* $\Longrightarrow x \in$ *restriction-ball a n*›
and ‹$x \in$ *restriction-ball a n* $\Longrightarrow x \downarrow$ *Suc n* = *a* $\downarrow$ *Suc n*›
by (*simp-all add*: *restriction-cball-def*)

lemma ‹$a \in$ *restriction-ball a n*›
and *center-mem-restriction-cball* [*simp*] : ‹$a \in \mathcal{B}_\downarrow(a,\ n)$›
by (*simp-all add*: *restriction-cball-memI*)

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

lemma *every-point-of-restriction-cball-is-centre* :
‹$b \in \mathcal{B}_\downarrow(a,\ n) \Longrightarrow \mathcal{B}_\downarrow(a,\ n) = \mathcal{B}_\downarrow(b,\ n)$›
by (*simp add*: *restriction-cball-def*)

lemma ‹$b \in$ *restriction-ball a n* $\Longrightarrow$ *restriction-ball a n* = *restriction-ball b n*›
by (*simp add*: *every-point-of-restriction-cball-is-centre*)

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

lemma *restriction-sphere-mem-iff* : ‹$x \in \mathcal{S}_\downarrow(a,\ n) \longleftrightarrow x \downarrow n = a \downarrow n \wedge x \downarrow Suc\ n \neq a \downarrow Suc\ n$›
and *restriction-sphere-memI* : ‹$x \downarrow n = a \downarrow n \Longrightarrow x \downarrow Suc\ n \neq a$

$\downarrow$ *Suc n* $\Longrightarrow$ $x \in \mathcal{S}_{\downarrow}(a,\ n)$›
and *restriction-sphere-memD1* : ‹$x \in \mathcal{S}_{\downarrow}(a,\ n) \Longrightarrow x \downarrow n = a \downarrow n$›
and *restriction-sphere-memD2* : ‹$x \in \mathcal{S}_{\downarrow}(a,\ n) \Longrightarrow x \downarrow Suc\ n \neq a \downarrow Suc\ n$›
by (*simp-all add*: *restriction-sphere-def*)

lemma *restriction-sphere-is-diff* : ‹$\mathcal{S}_{\downarrow}(a,\ n) = \mathcal{B}_{\downarrow}(a,\ n) - \mathcal{B}_{\downarrow}(a,\ Suc\ n)$›
by (*simp add*: *set-eq-iff restriction-sphere-mem-iff restriction-cball-mem-iff*)

lemma *restriction-open-restriction-cball* [*simp*] : ‹$open_{\downarrow}\ \mathcal{B}_{\downarrow}(a,\ n)$›
by (*metis restriction-cball-mem-iff restriction-tendstoE restriction-openI*)

lemma *restriction-closed-restriction-cball* [*simp*] : ‹$closed_{\downarrow}\ \mathcal{B}_{\downarrow}(a,\ n)$›
by (*metis restriction-cball-mem-iff restriction-closedI restriction-tendstoE*)

lemma *restriction-open-Compl-iff* : ‹$open_{\downarrow}\ (-\ S) \longleftrightarrow closed_{\downarrow}\ S$›
by (*simp add*: *restriction-closed-def*)

lemma *restriction-open-restriction-sphere* [*simp*] : ‹$open_{\downarrow}\ \mathcal{S}_{\downarrow}(a,\ n)$›
by (*simp add*: *restriction-sphere-is-diff Diff-eq*
restriction-open-Compl-iff restriction-open-inter)

lemma *restriction-closed-restriction-sphere* : ‹$closed_{\downarrow}\ \mathcal{S}_{\downarrow}(a,\ n)$›
by (*simp add*: *restriction-closed-def restriction-sphere-is-diff*)
(*simp add*: *restriction-open-union restriction-open-Compl-iff*)

end

context *restriction-space* **begin**

lemma *restriction-cball-anti-mono* : ‹$n \leq m \Longrightarrow \mathcal{B}_{\downarrow}(a,\ m) \subseteq \mathcal{B}_{\downarrow}(a,\ n)$›
by (*meson restriction-cball-memD restriction-cball-memI restriction-related-le subsetI*)

lemma *inside-every-cball-iff-eq* : ‹$(\forall\, n.\ x \in \mathcal{B}_{\downarrow}(\Sigma,\ n)) \longleftrightarrow x = \Sigma$›
by (*simp add*: *all-restriction-related-iff-related restriction-cball-mem-iff*)

lemma *Inf-many-inside-cball-iff-eq* : ‹$(\exists_{\infty} n.\ x \in \mathcal{B}_{\downarrow}(\Sigma,\ n)) \longleftrightarrow x = \Sigma$›
by (*unfold INFM-nat-le*)
(*meson inside-every-cball-iff-eq nle-le restriction-cball-anti-mono*

subset-eq)

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

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 (*unfold disj-imp*, *intro impI*)
assume ‹$\mathcal{B}_\downarrow(a,\ n) \cap \mathcal{B}_\downarrow(b,\ m) \neq \{\}$› ‹$\neg\ \mathcal{B}_\downarrow(a,\ n) \subseteq \mathcal{B}_\downarrow(b,\ m)$›
from ‹$\mathcal{B}_\downarrow(a,\ n) \cap \mathcal{B}_\downarrow(b,\ m) \neq \{\}$› **obtain** x **where** ‹$x \in \mathcal{B}_\downarrow(a,\ n)$› ‹$x \in \mathcal{B}_\downarrow(b,\ m)$› **by** *blast*
with *every-point-of-restriction-cball-is-centre*
have ‹$\mathcal{B}_\downarrow(a,\ n) = \mathcal{B}_\downarrow(x,\ m)$› ‹$\mathcal{B}_\downarrow(b,\ m) = \mathcal{B}_\downarrow(x,\ m)$› **by** *auto*
with ‹$\neg\ \mathcal{B}_\downarrow(a,\ n) \subseteq \mathcal{B}_\downarrow(b,\ m)$› **show** ‹$\mathcal{B}_\downarrow(b,\ m) \subseteq \mathcal{B}_\downarrow(a,\ n)$›
by (*metis nle-le restriction-cball-anti-mono*)
qed

lemma *equal-restriction-to-cball* :
‹$a \notin \mathcal{B}_\downarrow(b,\ n) \Longrightarrow x \in \mathcal{B}_\downarrow(b,\ n) \Longrightarrow y \in \mathcal{B}_\downarrow(b,\ n) \Longrightarrow x \downarrow k = a \downarrow k \longleftrightarrow y \downarrow k = a \downarrow k$›
by (*metis nat-le-linear restriction-cball-memD restriction-cball-memI restriction-related-le*)

end

context *restriction* **begin**

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

corollary *restriction-tendsto-restriction-cballI* : ‹$(\bigwedge n.\ \exists\, n0.\ \forall\, k{\geq}n0.\ \sigma\ k \in \mathcal{B}_\downarrow(\Sigma,\ n)) \Longrightarrow \sigma\ -\downarrow\rightarrow \Sigma$›
by (*simp add*: *restriction-tendsto-iff-restriction-cball-characterization*)

corollary *restriction-tendsto-restriction-cballD* : ‹$\sigma\ -\downarrow\rightarrow \Sigma \Longrightarrow \exists\, n0.\ \forall\, k{\geq}n0.\ \sigma\ k \in \mathcal{B}_\downarrow(\Sigma,\ n)$›
by (*simp add*: *restriction-tendsto-iff-restriction-cball-characterization*)

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

using *restriction-tendsto-restriction-cballD* **by** *blast*

end

context *restriction* **begin**

theorem *restriction-closed-iff-sequential-characterization* :
‹$closed_{\downarrow}\ S \longleftrightarrow (\forall \Sigma\ \sigma.\ range\ \sigma \subseteq S \longrightarrow \sigma\ -\downarrow\rightarrow \Sigma \longrightarrow \Sigma \in S)$›
proof (*intro iffI allI impI*)
show ‹*restriction-closed* $S \Longrightarrow$ *range* $\sigma \subseteq S \Longrightarrow \sigma\ -\downarrow\rightarrow \Sigma \Longrightarrow \Sigma \in S$› **for** $\Sigma\ \sigma$
by (*meson le-add1 range-subsetD restriction-closedD*)
next
assume $*$: ‹$\forall \Sigma\ \sigma.$ *range* $\sigma \subseteq S \longrightarrow \sigma\ -\downarrow\rightarrow \Sigma \longrightarrow \Sigma \in S$›
show ‹$closed_{\downarrow}\ S$›
proof (*rule restriction-closedI*, *rule ccontr*)
fix $\Sigma\ \sigma$ **assume** ‹$\Sigma \notin S$› ‹$\sigma\ -\downarrow\rightarrow \Sigma$› ‹$\nexists n0.\ \forall k{\geq}n0.\ \sigma\ k \notin S$›
from ‹$\nexists n0.\ \forall k{\geq}n0.\ \sigma\ k \notin S$› *INFM-nat-le* **have** ‹$\exists_{\infty}k.\ \sigma\ k \in S$›
by *auto*
from *this*[*THEN extraction-subseqD*[*of* ‹$\lambda x.\ x \in S$›]]
obtain f :: ‹*nat* ⇒ *nat*› **where** ‹*strict-mono* f› ‹$\forall k.\ \sigma\ (f\ k) \in S$›
by *blast*
from ‹$\forall k.\ \sigma\ (f\ k) \in S$› **have** ‹*range* $(\sigma \circ f) \subseteq S$› **by** *auto*
moreover from ‹*strict-mono* f› ‹$\sigma\ -\downarrow\rightarrow \Sigma$› **have** ‹$(\sigma \circ f)\ -\downarrow\rightarrow \Sigma$›
by (*fact restriction-tendsto-subseq*)
ultimately have ‹$\Sigma \in S$› **by** (*fact* $*$[*rule-format*])
with ‹$\Sigma \notin S$› **show** *False* **..**
qed
qed

corollary *restriction-closed-sequentialI* :
‹$(\bigwedge \Sigma\ \sigma.$ *range* $\sigma \subseteq S \Longrightarrow \sigma\ -\downarrow\rightarrow \Sigma \Longrightarrow \Sigma \in S) \Longrightarrow closed_{\downarrow}\ S$›
by (*simp add*: *restriction-closed-iff-sequential-characterization*)

corollary *restriction-closed-sequentialD* :
‹$closed_{\downarrow}\ S \Longrightarrow$ *range* $\sigma \subseteq S \Longrightarrow \sigma\ -\downarrow\rightarrow \Sigma \Longrightarrow \Sigma \in S$›
by (*simp add*: *restriction-closed-iff-sequential-characterization*)

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 (*intro iffI ballI*)
show ‹$open_{\downarrow}\ U \Longrightarrow \Sigma \in U \Longrightarrow \exists\,n.\ \mathcal{B}_{\downarrow}(\Sigma,\ n) \subseteq U$› **for** Σ
proof (*rule ccontr*)
assume ‹$open_{\downarrow}\ U$› ‹$\Sigma \in U$› ‹$\nexists\,n.\ \mathcal{B}_{\downarrow}(\Sigma,\ n) \subseteq U$›
from ‹$\nexists\,n.\ \mathcal{B}_{\downarrow}(\Sigma,\ n) \subseteq U$› **have** ‹$\forall\,n.\ \exists\,\sigma.\ \sigma \in \mathcal{B}_{\downarrow}(\Sigma,\ n) \cap -\ U$›
by *auto*
then obtain σ **where** ‹$\sigma\ n \in \mathcal{B}_{\downarrow}(\Sigma,\ n)$› ‹$\sigma\ n \in -\ U$› **for** n **by**
(*metis IntE*)
from ‹$\bigwedge n.\ \sigma\ n \in \mathcal{B}_{\downarrow}(\Sigma,\ n)$› **have** ‹$\sigma \mathrel{-\!\!\downarrow\!\!\to} \Sigma$›
by (*metis restriction-cball-memD restriction-related-le restriction-tendstoI*)
moreover from ‹$open_{\downarrow}\ U$› **have** ‹$closed_{\downarrow}\ (-\ U)$›
by (*simp add*: *restriction-closed-def*)
ultimately have ‹$\Sigma \in -\ U$›
using ‹$\bigwedge n.\ \sigma\ n \in -\ U$› *restriction-closedD* **by** *blast*
with ‹$\Sigma \in U$› **show** *False* **by** *simp*
qed
next
show ‹$\forall\,\Sigma{\in}U.\ \exists\,n.\ \mathcal{B}_{\downarrow}(\Sigma,\ n) \subseteq U \Longrightarrow open_{\downarrow}\ U$›
by (*metis center-mem-restriction-cball restriction-open-def*
restriction-open-restriction-cball subset-iff)
qed

corollary *restriction-open-restriction-cballI* :
‹$(\bigwedge\Sigma.\ \Sigma \in U \Longrightarrow \exists\,n.\ \mathcal{B}_{\downarrow}(\Sigma,\ n) \subseteq U) \Longrightarrow open_{\downarrow}\ U$›
by (*simp add*: *restriction-open-iff-restriction-cball-characterization*)

corollary *restriction-open-restriction-cballD* :
‹$open_{\downarrow}\ U \Longrightarrow \Sigma \in U \Longrightarrow \exists\,n.\ \mathcal{B}_{\downarrow}(\Sigma,\ n) \subseteq U$›
by (*simp add*: *restriction-open-iff-restriction-cball-characterization*)

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

end

context *restriction* **begin**

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

lemma *restriction-cont-onI* : ‹$(\bigwedge\Sigma\ \sigma.\ \Sigma \in A \Longrightarrow \sigma \mathrel{-\!\!\downarrow\!\!\to} \Sigma \Longrightarrow (\lambda n.\ f\ (\sigma\ n)) \mathrel{-\!\!\downarrow\!\!\to} f\ \Sigma) \Longrightarrow cont_{\downarrow}\ f\ on\ A$›
by (*simp add*: *restriction-cont-on-def restriction-cont-atI*)

lemma *restriction-cont-onD* : ‹$cont_{\downarrow}\ f\ on\ A \Longrightarrow \Sigma \in A \Longrightarrow \sigma \longrightarrow_{\downarrow} \Sigma \Longrightarrow (\lambda n.\ f\ (\sigma\ n)) \longrightarrow_{\downarrow} f\ \Sigma$›
by (*simp add*: *restriction-cont-on-def restriction-cont-atD*)

lemma *restriction-cont-on-comp* [*restriction-cont-simpset*] :
‹$cont_{\downarrow}\ f\ on\ A \Longrightarrow cont_{\downarrow}\ g\ on\ B \Longrightarrow f\ `\ A \subseteq B \Longrightarrow cont_{\downarrow}\ (\lambda x.\ g\ (f\ x))\ on\ A$›
by (*simp add*: *image-subset-iff restriction-cont-at-comp restriction-cont-on-def restriction-class.restriction-cont-on-def*)

lemma *restriction-cont-on-if-then-else* [*restriction-cont-simpset*] :
‹$[\![\bigwedge x.\ P\ x \Longrightarrow cont_{\downarrow}\ (f\ x)\ on\ A;\ \bigwedge x.\ \neg\ P\ x \Longrightarrow cont_{\downarrow}\ (g\ x)\ on\ A]\!] \Longrightarrow cont_{\downarrow}\ (\lambda y.\ if\ P\ x\ then\ f\ x\ y\ else\ g\ x\ y)\ on\ A$›
by (*auto intro*!: *restriction-cont-onI*) (*blast dest*: *restriction-cont-onD*)+

lemma *restriction-cont-on-subset* [*restriction-cont-simpset*] :
‹$cont_{\downarrow}\ f\ on\ B \Longrightarrow A \subseteq B \Longrightarrow cont_{\downarrow}\ f\ on\ A$›
by (*simp add*: *restriction-cont-on-def subset-iff*)

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

lemma *restriction-contI* : ‹$(\bigwedge\Sigma\ \sigma.\ \sigma \longrightarrow_{\downarrow} \Sigma \Longrightarrow (\lambda n.\ f\ (\sigma\ n)) \longrightarrow_{\downarrow} f\ \Sigma) \Longrightarrow cont_{\downarrow}\ f$›
by (*simp add*: *restriction-cont-onI*)

lemma *restriction-contD* : ‹$cont_{\downarrow}\ f \Longrightarrow \sigma \longrightarrow_{\downarrow} \Sigma \Longrightarrow (\lambda n.\ f\ (\sigma\ n)) \longrightarrow_{\downarrow} f\ \Sigma$›
by (*simp add*: *restriction-cont-onD*)

lemma *restriction-cont-comp* [*restriction-cont-simpset*] :
‹$cont_{\downarrow}\ g \Longrightarrow cont_{\downarrow}\ f \Longrightarrow cont_{\downarrow}\ (\lambda x.\ g\ (f\ x))$›
by (*simp add*: *restriction-cont-on-comp*)

lemma *restriction-cont-if-then-else* [*restriction-cont-simpset*] :
‹$[\![\bigwedge x.\ P\ x \Longrightarrow cont_{\downarrow}\ (f\ x);\ \bigwedge x.\ \neg\ P\ x \Longrightarrow cont_{\downarrow}\ (g\ x)]\!] \Longrightarrow cont_{\downarrow}\ (\lambda y.\ if\ P\ x\ then\ f\ x\ y\ else\ g\ x\ y)$›
by (*auto intro*!: *restriction-contI*) (*blast dest*: *restriction-contD*)+

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 $f :: \langle 'b :: \textit{restriction-space} \Rightarrow 'a \rangle$
proof (*intro iffI allI*)
show $\langle \exists k.\ f\ `\ \mathcal{B}_\downarrow(\Sigma,\ k) \subseteq \mathcal{B}_\downarrow(f\ \Sigma,\ n) \rangle$ **if** $\langle \textit{cont}_\downarrow\ f\ \textit{at}\ \Sigma \rangle$ **for** n
proof (*rule ccontr*)
assume $\langle \nexists k.\ f\ `\ \mathcal{B}_\downarrow(\Sigma,\ k) \subseteq \mathcal{B}_\downarrow(f\ \Sigma,\ n) \rangle$
hence $\langle \forall k.\ \exists \psi.\ \psi \in f\ `\ \mathcal{B}_\downarrow(\Sigma,\ k) \wedge \psi \notin \mathcal{B}_\downarrow(f\ \Sigma,\ n) \rangle$ **by** *auto*
then obtain ψ **where** $* : \langle \psi\ k \in f\ `\ \mathcal{B}_\downarrow(\Sigma,\ k) \rangle\ \langle \psi\ k \notin \mathcal{B}_\downarrow(f\ \Sigma,\ n) \rangle$
for k **by** *metis*
from $*(1)$ **obtain** σ **where** $** : \langle \sigma\ k \in \mathcal{B}_\downarrow(\Sigma,\ k) \rangle\ \langle \psi\ k = f\ (\sigma\ k) \rangle$
for k
by (*simp add*: *image-iff*) *metis*
have $\langle \sigma\ -\downarrow\rightarrow \Sigma \rangle$
by (*rule restriction-class.restriction-tendsto-restriction-cballI*)
(*use* $**(1)$ *restriction-space-class.restriction-cball-anti-mono* **in** *blast*)
with *restriction-cont-atD* $\langle$*restriction-cont-at f* $\Sigma \rangle$
have $\langle (\lambda k.\ f\ (\sigma\ k))\ -\downarrow\rightarrow f\ \Sigma \rangle$ **by** *blast*
hence $\langle \psi\ -\downarrow\rightarrow f\ \Sigma \rangle$ **by** (*fold* $**(2)$)
with $*(2)$ *restriction-tendsto-restriction-cballD* **show** *False* **by** *blast*
qed
next
show $\langle \forall n.\ \exists k.\ f\ `\ \mathcal{B}_\downarrow(\Sigma,\ k) \subseteq \mathcal{B}_\downarrow(f\ \Sigma,\ n) \Longrightarrow \textit{cont}_\downarrow\ f\ \textit{at}\ \Sigma \rangle$
by (*intro restriction-cont-atI restriction-tendsto-restriction-cballI*)
(*meson image-iff restriction-class.restriction-tendsto-restriction-cballD subset-eq*)
qed

corollary *restriction-cont-at-restriction-cballI* :
$\langle (\bigwedge n.\ \exists k.\ f\ `\ \mathcal{B}_\downarrow(\Sigma,\ k) \subseteq \mathcal{B}_\downarrow(f\ \Sigma,\ n)) \Longrightarrow \textit{cont}_\downarrow\ f\ \textit{at}\ \Sigma \rangle$
for $f :: \langle 'b :: \textit{restriction-space} \Rightarrow 'a \rangle$
by (*simp add*: *restriction-cont-at-iff-restriction-cball-characterization*)

corollary *restriction-cont-at-restriction-cballD* :
$\langle \textit{cont}_\downarrow\ f\ \textit{at}\ \Sigma \Longrightarrow \exists k.\ f\ `\ \mathcal{B}_\downarrow(\Sigma,\ k) \subseteq \mathcal{B}_\downarrow(f\ \Sigma,\ n) \rangle$
for $f :: \langle 'b :: \textit{restriction-space} \Rightarrow 'a \rangle$
by (*simp add*: *restriction-cont-at-iff-restriction-cball-characterization*)

corollary *restriction-cont-at-restriction-cballE* :
$\langle \textit{cont}_\downarrow\ f\ \textit{at}\ \Sigma \Longrightarrow (\bigwedge k.\ f\ `\ \mathcal{B}_\downarrow(\Sigma,\ k) \subseteq \mathcal{B}_\downarrow(f\ \Sigma,\ n) \Longrightarrow \textit{thesis}) \Longrightarrow \textit{thesis} \rangle$
for $f :: \langle 'b :: \textit{restriction-space} \Rightarrow 'a \rangle$
using *restriction-cont-at-restriction-cballD* **by** *blast*

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

proof (*intro iffI allI impI*)
fix $U :: \langle 'a\ set \rangle$ **assume** $\langle cont_{\downarrow}\ f \rangle\ \langle open_{\downarrow}\ U \rangle$
show $\langle open_{\downarrow}\ (f\ -`\ U) \rangle$
proof (*rule restriction-space-class.restriction-open-restriction-cballI*)
fix Σ **assume** $\langle \Sigma \in f\ -`\ U \rangle$
hence $\langle f\ \Sigma \in U \rangle$ **by** *simp*
with $\langle open_{\downarrow}\ U \rangle$ *restriction-open-restriction-cballD*
obtain n **where** $\langle \mathcal{B}_{\downarrow}(f\ \Sigma,\ n) \subseteq U \rangle$ **by** *blast*
moreover obtain k **where** $\langle f\ `\ \mathcal{B}_{\downarrow}(\Sigma,\ k) \subseteq \mathcal{B}_{\downarrow}(f\ \Sigma,\ n) \rangle$
by (*meson UNIV-I* $\langle cont_{\downarrow}\ f \rangle$ *restriction-cont-at-restriction-cballE restriction-cont-on-def*)
ultimately have $\langle \mathcal{B}_{\downarrow}(\Sigma,\ k) \subseteq f\ -`\ U \rangle$ **by** *blast*
thus $\langle \exists k.\ \mathcal{B}_{\downarrow}(\Sigma,\ k) \subseteq f\ -`\ U \rangle$ **..**
qed
next
show $\langle \forall U.\ open_{\downarrow}\ U \longrightarrow open_{\downarrow}\ (f\ -`\ U) \Longrightarrow cont_{\downarrow}\ f \rangle$
by (*unfold restriction-cont-on-def*, *intro ballI restriction-cont-at-restriction-cballI*)
(*simp add*: *image-subset-iff-subset-vimage restriction-space-class.restriction-open-restriction-cballD*)
qed

corollary *restriction-cont-restriction-openI* :
$\langle (\bigwedge U.\ open_{\downarrow}\ U \Longrightarrow open_{\downarrow}\ (f\ -`\ U)) \Longrightarrow cont_{\downarrow}\ f \rangle$
for $f :: \langle 'b :: restriction\text{-}space \Rightarrow 'a \rangle$
by (*simp add*: *restriction-cont-iff-restriction-open-characterization*)

corollary *restriction-cont-restriction-openD* :
$\langle cont_{\downarrow}\ f \Longrightarrow open_{\downarrow}\ U \Longrightarrow open_{\downarrow}\ (f\ -`\ U) \rangle$
for $f :: \langle 'b :: restriction\text{-}space \Rightarrow 'a \rangle$
by (*simp add*: *restriction-cont-iff-restriction-open-characterization*)

theorem *restriction-cont-iff-restriction-closed-characterization* :
$\langle cont_{\downarrow}\ f \longleftrightarrow (\forall S.\ closed_{\downarrow}\ S \longrightarrow closed_{\downarrow}\ (f\ -`\ S)) \rangle$
for $f :: \langle 'b :: restriction\text{-}space \Rightarrow 'a \rangle$
by (*metis boolean-algebra-class.boolean-algebra.double-compl local.restriction-closed-def restriction-class.restriction-closed-def restriction-cont-iff-restriction-open-characterization vimage-Compl*)

corollary *restriction-cont-restriction-closedI* :
$\langle (\bigwedge U.\ closed_{\downarrow}\ U \Longrightarrow closed_{\downarrow}\ (f\ -`\ U)) \Longrightarrow cont_{\downarrow}\ f \rangle$
for $f :: \langle 'b :: restriction\text{-}space \Rightarrow 'a \rangle$
by (*simp add*: *restriction-cont-iff-restriction-closed-characterization*)

corollary *restriction-cont-restriction-closedD* :
$\langle cont_{\downarrow}\ f \Longrightarrow closed_{\downarrow}\ U \Longrightarrow closed_{\downarrow}\ (f\ -`\ U) \rangle$
for $f :: \langle 'b :: restriction\text{-}space \Rightarrow 'a \rangle$
by (*simp add*: *restriction-cont-iff-restriction-closed-characterization*)

theorem *restriction-shift-on-restriction-open-imp-restriction-cont-on* :
‹$cont_{\downarrow}$ f *on* U› **if** ‹$open_{\downarrow}$ U› **and** ‹*restriction-shift-on* f k U›
proof (*intro restriction-cont-onI restriction-tendstoI*)
fix Σ σ **and** n :: *nat* **assume** ‹$\Sigma \in U$› ‹σ $-\downarrow\rightarrow$ Σ›
with ‹$open_{\downarrow}$ U› **obtain** *n0* **where** ‹$\forall\, l{\geq}n0.\ \sigma\ l \in U$›
by (*meson restriction-class.restriction-openD*)
moreover from ‹σ $-\downarrow\rightarrow$ Σ›[*THEN restriction-class.restriction-tendstoD*]
obtain *n1* **where** ‹$\forall\, l{\geq}n1.\ \Sigma \downarrow nat\ (int\ n - k) = \sigma\ l \downarrow nat\ (int\ n - k)$› **..**
ultimately have ‹$\forall\, l{\geq}max\ n0\ n1.\ \sigma\ l \in U \wedge \Sigma \downarrow nat\ (int\ n - k) = \sigma\ l \downarrow nat\ (int\ n - k)$› **by** *simp*
with ‹$\Sigma \in U$› ‹*restriction-shift-on* f k U› *restriction-shift-onD*
have ‹$\forall\, l{\geq}max\ n0\ n1.\ f\ \Sigma \downarrow nat\ (int\ (nat\ (int\ n - k)) + k) = f\ (\sigma\ l) \downarrow nat\ (int\ (nat\ (int\ n - k)) + k)$› **by** *blast*
moreover have ‹$n \leq nat\ (int\ (nat\ (int\ n - k)) + k)$› **by** *auto*
ultimately have ‹$\forall\, l{\geq}max\ n0\ n1.\ f\ \Sigma \downarrow n = f\ (\sigma\ l) \downarrow n$› **by** (*meson restriction-related-le*)
thus ‹$\exists\, n2.\ \forall\, l{\geq}n2.\ f\ \Sigma \downarrow n = f\ (\sigma\ l) \downarrow n$› **by** *blast*
qed

corollary *restriction-shift-imp-restriction-cont* [*restriction-cont-simpset*] :
‹*restriction-shift* f k $\Longrightarrow$ $cont_{\downarrow}$ f›
by (*simp add*: *restriction-shift-def*
restriction-shift-on-restriction-open-imp-restriction-cont-on)

corollary *non-too-destructive-imp-restriction-cont* [*restriction-cont-simpset*] :
‹*non-too-destructive* f $\Longrightarrow$ $cont_{\downarrow}$ f›
by (*simp add*: *non-too-destructive-def non-too-destructive-on-def*
restriction-shift-on-restriction-open-imp-restriction-cont-on)

end

5.3 Compactness

context *restriction* **begin**

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

$\Longrightarrow$ *compact*$_\downarrow$ *K*› **by** (*simp add*: *restriction-compact-def*)

lemma *restriction-compactD* :
‹*compact*$_\downarrow$ *K* $\Longrightarrow$ *range* $\sigma \subseteq K \Longrightarrow$
$\exists f :: nat \Rightarrow nat.\ \exists \Sigma.\ \Sigma \in K \wedge$ *strict-mono* $f \wedge (\sigma \circ f) \longrightarrow\!\!\downarrow \rightarrow \Sigma$›
by (*simp add*: *restriction-compact-def*)

lemma *restriction-compactE* :
assumes ‹*compact*$_\downarrow$ *K*› **and** ‹*range* $\sigma \subseteq K$›
obtains f :: ‹*nat* $\Rightarrow$ *nat*› **and** Σ **where** ‹$\Sigma \in K$› ‹*strict-mono* f›
‹$(\sigma \circ f) \longrightarrow\!\!\downarrow \rightarrow \Sigma$›
by (*meson assms restriction-compactD*)

lemma *restriction-compact-empty* [*simp*] : ‹*compact*$_\downarrow$ {}›
by (*simp add*: *restriction-compact-def*)

lemma (**in** *restriction-space*) *restriction-compact-imp-restriction-closed*
:
‹*closed*$_\downarrow$ *K*› **if** ‹*compact*$_\downarrow$ *K*›
proof (*rule restriction-closed-sequentialI*)
fix σ Σ **assume** ‹*range* $\sigma \subseteq K$› ‹$\sigma \longrightarrow\!\!\downarrow \rightarrow \Sigma$›
from *restriction-compactD* ‹*compact*$_\downarrow$ *K*› ‹*range* $\sigma \subseteq K$›
obtain f **and** Σ' **where** ‹$\Sigma' \in K$› ‹*strict-mono* f› ‹$(\sigma \circ f) \longrightarrow\!\!\downarrow \rightarrow \Sigma'$›
by *blast*
from *restriction-tendsto-subseq* ‹*strict-mono* f› ‹$\sigma \longrightarrow\!\!\downarrow \rightarrow \Sigma$›
have ‹$(\sigma \circ f) \longrightarrow\!\!\downarrow \rightarrow \Sigma$› **by** *blast*
with ‹$(\sigma \circ f) \longrightarrow\!\!\downarrow \rightarrow \Sigma'$› **have** ‹$\Sigma' = \Sigma$› **by** (*fact restriction-tendsto-unique*)
with ‹$\Sigma' \in K$› **show** ‹$\Sigma \in K$› **by** *simp*
qed

lemma *restriction-compact-union* : ‹*compact*$_\downarrow$ $(K \cup L)$›
if ‹*compact*$_\downarrow$ *K*› **and** ‹*compact*$_\downarrow$ *L*›
proof (*rule restriction-compactI*)
fix σ :: ‹*nat* $\Rightarrow$ -› **assume** ‹*range* $\sigma \subseteq K \cup L$›
{ fix *K L* **and** f :: ‹*nat* $\Rightarrow$ *nat*›
assume ‹*compact*$_\downarrow$ *K*› ‹*strict-mono* f› ‹$\sigma\ (f\ n) \in K$› **for** n
from ‹$(\bigwedge n.\ \sigma\ (f\ n) \in K)$› **have** ‹*range* $(\sigma \circ f) \subseteq K$› **by** *auto*
with ‹*compact*$_\downarrow$ *K*› *restriction-compactD* **obtain** g Σ
where ‹$\Sigma \in K$› ‹*strict-mono* g› ‹$(\sigma \circ f \circ g) \longrightarrow\!\!\downarrow \rightarrow \Sigma$› **by** *blast*
hence ‹$\Sigma \in K \cup L \wedge$ *strict-mono* $(f \circ g) \wedge (\sigma \circ (f \circ g)) \longrightarrow\!\!\downarrow \rightarrow \Sigma$›
by (*metis* (*no-types*, *lifting*) *Un-iff* ‹*strict-mono* f› *comp-assoc*
monotone-on-o subset-UNIV)
hence ‹$\exists f\ \Sigma.\ \Sigma \in K \cup L \wedge$ *strict-mono* $f \wedge (\sigma \circ f) \longrightarrow\!\!\downarrow \rightarrow \Sigma$› **by**
blast
} **note** $* =$ *this*
have ‹$(\exists_\infty n.\ \sigma\ n \in K) \vee (\exists_\infty n.\ \sigma\ n \in L)$›

proof (*rule ccontr*)
assume ‹¬ ((∃$_\infty$*n*. *σ n* ∈ *K*) ∨ (∃$_\infty$*n*. *σ n* ∈ *L*))›
hence ‹*finite* {*n*. *σ n* ∈ *K*} ∧ *finite* {*n*. *σ n* ∈ *L*}›
using *frequently-cofinite* **by** *blast*
then obtain *n* **where** ‹*n* ∉ {*n*. *σ n* ∈ *K*} ∧ *n* ∉ {*n*. *σ n* ∈ *L*}›
by (*metis* (*mono-tags*, *lifting*) *INFM-nat-le dual-order.refl*
frequently-cofinite le-sup-iff mem-Collect-eq)
hence ‹*σ n* ∉ *K* ∪ *L*› **by** *simp*
with ‹*range σ* ⊆ *K* ∪ *L*› **show** *False* **by** *blast*
qed
thus ‹∃*f* Σ. Σ ∈ *K* ∪ *L* ∧ *strict-mono f* ∧ (*σ* ∘ *f*) −↓→ Σ›
by (*elim disjE extraction-subseqE*)
(*use* ∗ ‹*compact*$_\downarrow$ *K*› **in** *blast*, *metis* ∗ *Un-iff* ‹*compact*$_\downarrow$ *L*›)
qed

lemma *restriction-compact-finite-Union* :
‹⟦*finite I*; ⋀*i*. *i* ∈ *I* ⟹ *compact*$_\downarrow$ (*K i*)⟧ ⟹ *compact*$_\downarrow$ (⋃ *i*∈*I*. *K i*)›
by (*induct I rule*: *finite-induct*)
(*simp-all add*: *restriction-compact-union*)

lemma (**in** *restriction-space*) *restriction-compact-Inter* :
‹*compact*$_\downarrow$ (⋂ *i*. *K i*)› **if** ‹⋀*i*. *compact*$_\downarrow$ (*K i*)›
proof (*rule restriction-compactI*)
fix *σ* :: ‹*nat* ⇒ *'a*› **assume** ‹*range σ* ⊆ ⋂ (*range K*)›
hence ‹*range σ* ⊆ *K i*› **for** *i* **by** *blast*
with ‹⋀*i*. *compact*$_\downarrow$ (*K i*)› *restriction-compactD*
obtain *f* Σ **where** ‹*strict-mono f*› ‹(*σ* ∘ *f*) −↓→ Σ› **by** *blast*
from ‹⋀*i*. *compact*$_\downarrow$ (*K i*)› **have** ‹*closed*$_\downarrow$ (*K i*)› **for** *i*
by (*simp add*: *restriction-compact-imp-restriction-closed*)
moreover from ‹⋀*i*. *range σ* ⊆ *K i*› **have** ‹*range* (*σ* ∘ *f*) ⊆ *K i*›
for *i* **by** *auto*
ultimately have ‹Σ ∈ *K i*› **for** *i*
by (*meson* ‹(*σ* ∘ *f*) −↓→ Σ› *restriction-closed-sequentialD*)
with ‹*strict-mono f*› ‹(*σ* ∘ *f*) −↓→ Σ›
show ‹∃*f* Σ. Σ ∈ ⋂ (*range K*) ∧ *strict-mono f* ∧ (*σ* ∘ *f*) −↓→ Σ›
by *blast*
qed

lemma *finite-imp-restriction-compact* : ‹*compact*$_\downarrow$ *K*› **if** ‹*finite K*›
proof (*rule restriction-compactI*)
fix *σ* :: ‹*nat* ⇒ -› **assume** ‹*range σ* ⊆ *K*›
have ‹∃Σ∈*K*. ∃$_\infty$*n*. *σ n* = Σ›
proof (*rule ccontr*)
assume ‹¬ (∃Σ∈*K*. ∃$_\infty$*n*. *σ n* = Σ)›
hence ‹∀Σ∈*K*. *finite* {*n*. *σ n* = Σ}› **by** (*simp add*: *frequently-cofinite*)
with ‹*finite K*› **have** ‹*finite* (⋃Σ∈*K*. {*n*. *σ n* = Σ})› **by** *blast*
also from ‹*range σ* ⊆ *K*› **have** ‹(⋃Σ∈*K*. {*n*. *σ n* = Σ}) = *UNIV*›

by *auto*
finally show *False* **by** *simp*
qed
then obtain Σ **where** ‹$\Sigma \in K$› ‹$\exists_\infty n.\ \sigma\ n = \Sigma$› **..**
from *extraction-subseqD*[*of* - σ, *OF* ‹$\exists_\infty n.\ \sigma\ n = \Sigma$›]
obtain f :: ‹*nat* ⇒ *nat*› **where** ‹*strict-mono* f› ‹$\sigma\ (f\ n) = \Sigma$› **for** n
by *blast*
from ‹$\bigwedge n.\ \sigma\ (f\ n) = \Sigma$› **have** ‹$(\sigma \circ f) \longrightarrow\!\!\downarrow \Sigma$›
by (*simp add*: *restriction-tendstoI*)
with ‹*strict-mono* f› ‹$\Sigma \in K$›
show ‹$\exists f\ \Sigma.\ \Sigma \in K \wedge$ *strict-mono* $f \wedge (\sigma \circ f) \longrightarrow\!\!\downarrow \Sigma$› **by** *blast*
qed

lemma *restriction-compact-restriction-closed-subset* : ‹$compact_\downarrow\ L$›
if ‹$L \subseteq K$› ‹$compact_\downarrow\ K$› ‹$closed_\downarrow\ L$›
proof (*rule restriction-compactI*)
fix σ :: ‹*nat* ⇒ -› **assume** ‹*range* $\sigma \subseteq L$›
with ‹$L \subseteq K$› **have** ‹*range* $\sigma \subseteq K$› **by** *blast*
with ‹$compact_\downarrow\ K$› *restriction-compactD*
obtain $f\ \Sigma$ **where** ‹$\Sigma \in K$› ‹*strict-mono* f› ‹$(\sigma \circ f) \longrightarrow\!\!\downarrow \Sigma$› **by**
blast
from ‹*range* $\sigma \subseteq L$› **have** ‹*range* $(\sigma \circ f) \subseteq L$› **by** *auto*
from *restriction-closed-sequentialD* ‹*restriction-closed* L›
‹$(\sigma \circ f) \longrightarrow\!\!\downarrow \Sigma$› ‹*range* $(\sigma \circ f) \subseteq L$› **have** ‹$\Sigma \in L$› **by** *blast*
with ‹*strict-mono* f› ‹$(\sigma \circ f) \longrightarrow\!\!\downarrow \Sigma$›
show ‹$\exists f\ \Sigma.\ \Sigma \in L \wedge$ *strict-mono* $f \wedge (\sigma \circ f) \longrightarrow\!\!\downarrow \Sigma$› **by** *blast*
qed

lemma *restriction-cont-image-of-restriction-compact* :
‹$compact_\downarrow\ (f\ `\ K)$› **if** ‹$compact_\downarrow\ K$› **and** ‹$cont_\downarrow\ f$ *on* K›
proof (*rule restriction-compactI*)
fix σ :: ‹*nat* ⇒ -› **assume** ‹*range* $\sigma \subseteq f\ `\ K$›
hence ‹$\forall n.\ \exists \gamma.\ \gamma \in K \wedge \sigma\ n = f\ \gamma$› **by** (*meson imageE range-subsetD*)
then obtain γ :: ‹*nat* ⇒ -› **where** ‹*range* $\gamma \subseteq K$› ‹$\sigma\ n = f\ (\gamma\ n)$›
for n
by (*metis image-subsetI*)
from *restriction-class.restriction-compactD*[*OF* ‹$compact_\downarrow\ K$› ‹*range*
$\gamma \subseteq K$›]
obtain $g\ \Sigma$ **where** ‹$\Sigma \in K$› ‹*strict-mono* g› ‹$(\gamma \circ g) \longrightarrow\!\!\downarrow \Sigma$› **by**
blast
from ‹$cont_\downarrow\ f$ *on* K› ‹$\Sigma \in K$›
have ‹$cont_\downarrow\ f$ *at* Σ› **by** (*simp add*: *restriction-cont-on-def*)
with ‹$(\gamma \circ g) \longrightarrow\!\!\downarrow \Sigma$› *restriction-cont-atD*
have ‹$(\lambda n.\ f\ ((\gamma \circ g)\ n)) \longrightarrow\!\!\downarrow f\ \Sigma$› **by** *blast*
also have ‹$(\lambda n.\ f\ ((\gamma \circ g)\ n)) = (\sigma \circ g)$›
by (*simp add*: ‹$\bigwedge n.\ \sigma\ n = f\ (\gamma\ n)$› *comp-def*)
finally have ‹$(\sigma \circ g) \longrightarrow\!\!\downarrow f\ \Sigma$› **.**

with ‹$\Sigma \in K$› ‹*strict-mono g*›
show ‹$\exists\, g\ \Sigma.\ \Sigma \in f\ `\ K \wedge$ *strict-mono g* $\wedge\ (\sigma \circ g) \longrightarrow \Sigma$› **by** *blast*
qed

end

5.4 Properties for Function and Product

lemma *restriction-cball-fun-is* : ‹$\mathcal{B}_{\downarrow}(f,\ n) = \{g.\ \forall x.\ g\ x \in \mathcal{B}_{\downarrow}(f\ x,\ n)\}$›
by (*simp add*: *set-eq-iff restriction-cball-mem-iff restriction-fun-def*) *metis*

lemma *restriction-cball-prod-is* :
‹$\mathcal{B}_{\downarrow}(\Sigma,\ n) = \mathcal{B}_{\downarrow}(fst\ \Sigma,\ n) \times \mathcal{B}_{\downarrow}(snd\ \Sigma,\ n)$›
by (*simp add*: *set-eq-iff restriction-cball-def restriction-prod-def*)

lemma *restriction-open-prod-imp-restriction-open-image-fst* :
‹$open_{\downarrow}\ (fst\ `\ U)$› **if** ‹$open_{\downarrow}\ U$›
proof (*rule restriction-openI*)
fix $\Sigma\ \sigma$ **assume** ‹$\Sigma \in fst\ `\ U$› **and** ‹$\sigma \longrightarrow \Sigma$›
from ‹$\Sigma \in fst\ `\ U$› **obtain** v **where** ‹$(\Sigma,\ v) \in U$› **by** *auto*
from ‹$\sigma \longrightarrow \Sigma$› **have** ‹$(\lambda n.\ (\sigma\ n,\ v)) \longrightarrow (\Sigma,\ v)$›
by (*simp add*: *restriction-tendsto-prod-iff restriction-tendsto-const*)
from *restriction-openD*[*OF* ‹*restriction-open U*› ‹$(\Sigma,\ v) \in U$› *this*]
obtain *n0* **where** ‹$\forall\, k \geq n0.\ (\sigma\ k,\ v) \in U$› **..**
thus ‹$\exists\, n0.\ \forall\, k \geq n0.\ \sigma\ k \in fst\ `\ U$› **by** (*metis fst-conv imageI*)
qed

lemma *restriction-open-prod-imp-restriction-open-image-snd* :
‹$open_{\downarrow}\ (snd\ `\ U)$› **if** ‹$open_{\downarrow}\ U$›
proof (*rule restriction-openI*)
fix $\Sigma\ \sigma$ **assume** ‹$\Sigma \in snd\ `\ U$› **and** ‹$\sigma \longrightarrow \Sigma$›
from ‹$\Sigma \in snd\ `\ U$› **obtain** u **where** ‹$(u,\ \Sigma) \in U$› **by** *auto*
from ‹$\sigma \longrightarrow \Sigma$› **have** ‹$(\lambda n.\ (u,\ \sigma\ n)) \longrightarrow (u,\ \Sigma)$›
by (*simp add*: *restriction-tendsto-prod-iff restriction-tendsto-const*)
from *restriction-openD*[*OF* ‹*restriction-open U*› ‹$(u,\ \Sigma) \in U$› *this*]
obtain *n0* **where** ‹$\forall\, k \geq n0.\ (u,\ \sigma\ k) \in U$› **..**
thus ‹$\exists\, n0.\ \forall\, k \geq n0.\ \sigma\ k \in snd\ `\ U$› **by** (*metis snd-conv imageI*)
qed

lemma *restriction-open-prod-iff* :
‹$open_{\downarrow}\ (U \times V) \longleftrightarrow (V = \{\} \vee open_{\downarrow}\ U) \wedge (U = \{\} \vee open_{\downarrow}\ V)$›
proof (*intro iffI conjI*)
show ‹$open_{\downarrow}\ (U \times V) \Longrightarrow V = \{\} \vee open_{\downarrow}\ U$›
by (*metis fst-image-times restriction-open-prod-imp-restriction-open-image-fst*)
next
show ‹$open_{\downarrow}\ (U \times V) \Longrightarrow U = \{\} \vee open_{\downarrow}\ V$›
by (*metis restriction-open-prod-imp-restriction-open-image-snd snd-image-times*)

next
assume ‹$(V = \{\} \vee open_{\downarrow}\ U) \wedge (U = \{\} \vee open_{\downarrow}\ V)$›
then consider ‹$U = \{\}$› | ‹$V = \{\}$› | ‹$open_{\downarrow}\ U \wedge open_{\downarrow}\ V$› **by** *fast*
thus ‹$open_{\downarrow}\ (U \times V)$›
proof *cases*
show ‹$U = \{\} \Longrightarrow open_{\downarrow}\ (U \times V)$› **by** *simp*
next
show ‹$V = \{\} \Longrightarrow open_{\downarrow}\ (U \times V)$› **by** *simp*
next
show ‹$open_{\downarrow}\ (U \times V)$› **if** $*$: ‹$open_{\downarrow}\ U \wedge open_{\downarrow}\ V$›
proof (*rule restriction-openI*)
fix $\Sigma\ \sigma$ **assume** ‹$\Sigma \in U \times V$› **and** ‹$\sigma\ -\downarrow\rightarrow \Sigma$›
from ‹$\Sigma \in U \times V$› **have** ‹$fst\ \Sigma \in U$› ‹$snd\ \Sigma \in V$› **by** *auto*
from ‹$\sigma\ -\downarrow\rightarrow \Sigma$› **have** ‹$(\lambda n.\ fst\ (\sigma\ n))\ -\downarrow\rightarrow fst\ \Sigma$› ‹$(\lambda n.\ snd\ (\sigma\ n))\ -\downarrow\rightarrow snd\ \Sigma$›
by (*simp-all add*: *restriction-tendsto-prod-iff*)
from *restriction-openD*[*OF* $*$[*THEN conjunct1*] ‹$fst\ \Sigma \in U$› ‹$(\lambda n.\ fst\ (\sigma\ n))\ -\downarrow\rightarrow fst\ \Sigma$›]
obtain *n0* **where** ‹$\forall\, k{\geq}n0.\ fst\ (\sigma\ k) \in U$› **..**
moreover from *restriction-openD*[*OF* $*$[*THEN conjunct2*] ‹$snd\ \Sigma \in V$› ‹$(\lambda n.\ snd\ (\sigma\ n))\ -\downarrow\rightarrow snd\ \Sigma$›]
obtain *n1* **where** ‹$\forall\, k{\geq}n1.\ snd\ (\sigma\ k) \in V$› **..**
ultimately have ‹$\forall\, k{\geq}max\ n0\ n1.\ \sigma\ k \in U \times V$› **by** (*simp add*: *mem-Times-iff*)
thus ‹$\exists\, n2.\ \forall\, k{\geq}n2.\ \sigma\ k \in U \times V$› **by** *blast*
qed
qed
qed

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

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

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

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

‹$cont_{\downarrow}\ f\ at\ \Sigma \Longrightarrow cont_{\downarrow}\ (\lambda x.\ snd\ (f\ x))\ at\ \Sigma$›
by (*simp-all add*: *restriction-cont-at-prod-codomain-iff*)

lemma *restriction-cont-on-prod-codomain-imp* [*restriction-cont-simpset*]
:
‹$cont_{\downarrow}\ f\ on\ A \Longrightarrow cont_{\downarrow}\ (\lambda x.\ fst\ (f\ x))\ on\ A$›
‹$cont_{\downarrow}\ f\ on\ A \Longrightarrow cont_{\downarrow}\ (\lambda x.\ snd\ (f\ x))\ on\ A$›
by (*simp-all add*: *restriction-cont-on-prod-codomain-iff*)

lemma *restriction-cont-prod-codomain-imp* [*restriction-cont-simpset*]
:
‹$cont_{\downarrow}\ f \Longrightarrow cont_{\downarrow}\ (\lambda x.\ fst\ (f\ x))$›
‹$cont_{\downarrow}\ f \Longrightarrow cont_{\downarrow}\ (\lambda x.\ snd\ (f\ x))$›
by (*simp-all add*: *restriction-cont-prod-codomain-iff*)

lemma *restriction-cont-at-fun-imp* [*restriction-cont-simpset*] :
‹$cont_{\downarrow}\ f\ at\ A \Longrightarrow cont_{\downarrow}\ (\lambda x.\ f\ x\ y)\ at\ A$›
by (*rule restriction-cont-atI*)
(*metis restriction-cont-atD restriction-tendsto-fun-imp*)

lemma *restriction-cont-on-fun-imp* [*restriction-cont-simpset*] :
‹$cont_{\downarrow}\ f\ on\ A \Longrightarrow cont_{\downarrow}\ (\lambda x.\ f\ x\ y)\ on\ A$›
by (*simp add*: *restriction-cont-at-fun-imp restriction-cont-on-def*)

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

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 $f :: \langle 'a :: restriction\text{-}space \times 'b :: restriction\text{-}space \Rightarrow 'c :: restriction\text{-}space\rangle$
by (*simp add*: *restriction-cball-prod-is subset-iff image-iff*
restriction-cont-at-iff-restriction-cball-characterization,
meson center-mem-restriction-cball)+

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 $f :: \langle 'a :: restriction\text{-}space \times 'b :: restriction\text{-}space \Rightarrow 'c :: restriction\text{-}space\rangle$

proof −
show ‹$cont_{\downarrow}$ ($\lambda x.\ f\ (x,\ y)$) *on* {$x.\ (x,\ y) \in A$}›
proof (*unfold restriction-cont-on-def*, *rule ballI*)
fix x **assume** ‹$x \in$ {$x.\ (x,\ y) \in A$}›
with ‹$cont_{\downarrow}$ f *on* A› **have** ‹$cont_{\downarrow}$ f *at* $(x,\ y)$›
unfolding *restriction-cont-on-def* **by** *simp*
thus ‹$cont_{\downarrow}$ ($\lambda x.\ f\ (x,\ y)$) *at* x›
by (*fact restriction-cont-at-prod-domain-imp*[*of f* ‹$(x,\ y)$›, *simplified*])
qed
next
show ‹$cont_{\downarrow}$ ($\lambda y.\ f\ (x,\ y)$) *on* {$y.\ (x,\ y) \in A$}›
proof (*unfold restriction-cont-on-def*, *rule ballI*)
fix y **assume** ‹$y \in$ {$y.\ (x,\ y) \in A$}›
with ‹$cont_{\downarrow}$ f *on* A› **have** ‹$cont_{\downarrow}$ f *at* $(x,\ y)$›
unfolding *restriction-cont-on-def* **by** *simp*
thus ‹$cont_{\downarrow}$ ($\lambda y.\ f\ (x,\ y)$) *at* y›
by (*fact restriction-cont-at-prod-domain-imp*[*of f* ‹$(x,\ y)$›, *simplified*])
qed
qed

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*›
by (*metis UNIV-I restriction-cont-at-prod-domain-imp*(*1*) *restriction-cont-on-def split-pairs*)
(*metis UNIV-I restriction-cont-at-prod-domain-imp*(*2*) *restriction-cont-on-def split-pairs*)

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›
by (*simp add*: *restriction-adm-def*)

lemma *restriction-admD* :
‹$[\![$*restriction-adm* P; $\sigma\ -\downarrow\rightarrow\ \Sigma$; $\bigwedge n.\ P\ (\sigma\ n)]\!] \Longrightarrow P\ \Sigma$›
by (*simp add*: *restriction-adm-def*)

6.1.2 Properties

lemma *restriction-adm-const* [*restriction-adm-simpset*] :
‹$adm_{\downarrow}\ (\lambda x.\ t)$›
by (*simp add*: *restriction-admI*)

lemma *restriction-adm-conj* [*restriction-adm-simpset*] :
‹$adm_{\downarrow}\ (\lambda x.\ P\ x) \Longrightarrow adm_{\downarrow}\ (\lambda x.\ Q\ x) \Longrightarrow adm_{\downarrow}\ (\lambda x.\ P\ x \wedge Q\ x)$›
by (*fast intro*: *restriction-admI elim*: *restriction-admD*)

lemma *restriction-adm-all* [*restriction-adm-simpset*] :
‹$(\bigwedge y.\ adm_{\downarrow}\ (\lambda x.\ P\ x\ y)) \Longrightarrow adm_{\downarrow}\ (\lambda x.\ \forall y.\ P\ x\ y)$›
by (*fast intro*: *restriction-admI elim*: *restriction-admD*)

lemma *restriction-adm-ball* [*restriction-adm-simpset*] :
‹$(\bigwedge y.\ y \in A \Longrightarrow adm_{\downarrow}\ (\lambda x.\ P\ x\ y)) \Longrightarrow adm_{\downarrow}\ (\lambda x.\ \forall y{\in}A.\ P\ x\ y)$›
by (*fast intro*: *restriction-admI elim*: *restriction-admD*)

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 (*rule restriction-admI*)
fix $\sigma\ \Sigma$
assume $*$: ‹$\sigma\ -\downarrow\rightarrow\ \Sigma$› ‹$\bigwedge n.\ P\ (\sigma\ n) \vee Q\ (\sigma\ n)$›
from $*(2)$ **have** $**$: ‹$(\forall i.\ \exists j{\geq}i.\ P\ (\sigma\ j)) \vee (\forall i.\ \exists j{\geq}i.\ Q\ (\sigma\ j))$›
by (*meson nat-le-linear*)

{ fix P **assume** \$: ‹$adm_{\downarrow}\ (\lambda x.\ P\ x)$› ‹$\forall i.\ \exists j{\geq}i.\ P\ (\sigma\ j)$›
define f **where** ‹$f\ i = (LEAST\ j.\ i \leq j \wedge P\ (\sigma\ j))$› **for** i
have *f1*: ‹$\bigwedge i.\ i \leq f\ i$› **and** *f2*: ‹$\bigwedge i.\ P\ (\sigma\ (f\ i))$›
using *LeastI-ex* [*OF* \$(*2*)[*rule-format*]] **by** (*simp-all add*: *f-def*)
have *f3* : ‹$(\lambda n.\ \sigma\ (f\ n))\ -\downarrow\rightarrow\ \Sigma$›
proof (*rule restriction-tendstoI*)
fix n
from ‹$\sigma\ -\downarrow\rightarrow\ \Sigma$› *restriction-tendstoD* **obtain** *n0* **where** ‹$\forall k{\geq}n0.\ \Sigma \downarrow n = \sigma\ k \downarrow n$› **by** *blast*
hence ‹$\forall k{\geq}max\ n0\ n.\ \Sigma \downarrow n = \sigma\ (f\ k) \downarrow n$› **by** (*meson f1 le-trans*

max.boundedE)
thus ‹$\exists\, n0.\ \forall\, k{\geq}n0.\ \Sigma \downarrow n = \sigma\ (f\ k) \downarrow n$› **by** *blast*
qed
have ‹$P\ \Sigma$› **by** (*fact restriction-admD*[*OF* $(1) *f3 f2*])
}

with ** ‹$adm_{\downarrow}\ (\lambda x.\ P\ x)$› ‹$adm_{\downarrow}\ (\lambda x.\ Q\ x)$› **show** ‹$P\ \Sigma \vee Q\ \Sigma$› **by** *blast*
qed

lemma *restriction-adm-imp* [*restriction-adm-simpset*] :
‹$adm_{\downarrow}\ (\lambda x.\ \neg\ P\ x) \Longrightarrow adm_{\downarrow}\ (\lambda x.\ Q\ x) \Longrightarrow adm_{\downarrow}\ (\lambda x.\ P\ x \longrightarrow Q\ x)$›
by (*subst imp-conv-disj*) (*rule restriction-adm-disj*)

lemma *restriction-adm-iff* [*restriction-adm-simpset*] :
‹$adm_{\downarrow}\ (\lambda x.\ P\ x \longrightarrow Q\ x) \Longrightarrow adm_{\downarrow}\ (\lambda x.\ Q\ x \longrightarrow P\ x) \Longrightarrow adm_{\downarrow}\ (\lambda x.\ P\ x \longleftrightarrow Q\ x)$›
by (*subst iff-conv-conj-imp*) (*rule restriction-adm-conj*)

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

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* $\Rightarrow$ $'b$ :: *restriction-space*›
proof (*rule restriction-admI*)
fix $\sigma\ \Sigma$ **assume** ‹$\sigma\ {-}{\downarrow}{\rightarrow}\ \Sigma$› **and** ‹$\bigwedge n.\ f\ (\sigma\ n) = g\ (\sigma\ n)$›
from *restriction-contD*[*OF* ‹$cont_{\downarrow}\ f$› ‹$\sigma\ {-}{\downarrow}{\rightarrow}\ \Sigma$›] **have** ‹$(\lambda n.\ f\ (\sigma\ n))\ {-}{\downarrow}{\rightarrow}\ f\ \Sigma$› **.**
hence ‹$(\lambda n.\ g\ (\sigma\ n))\ {-}{\downarrow}{\rightarrow}\ f\ \Sigma$› **by** (*unfold* ‹$\bigwedge n.\ f\ (\sigma\ n) = g\ (\sigma\ n)$›)
moreover from *restriction-contD*[*OF* ‹$cont_{\downarrow}\ g$› ‹$\sigma\ {-}{\downarrow}{\rightarrow}\ \Sigma$›] **have** ‹$(\lambda n.\ g\ (\sigma\ n))\ {-}{\downarrow}{\rightarrow}\ g\ \Sigma$› **.**
ultimately show ‹$f\ \Sigma = g\ \Sigma$› **by** (*fact restriction-tendsto-unique*)
qed

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 (*rule restriction-admI*)
fix $\sigma\ \Sigma$ **assume** ‹$\sigma\ {-}{\downarrow}{\rightarrow}\ \Sigma$› ‹$\bigwedge n.\ P\ (t\ (\sigma\ n))$›
from *restriction-contD*[*OF* ‹$cont_{\downarrow}\ (\lambda x.\ t\ x)$› ‹$\sigma\ {-}{\downarrow}{\rightarrow}\ \Sigma$›]

have ‹$(\lambda n.\ t\ (\sigma\ n))\ -\downarrow\rightarrow\ t\ \Sigma$› **.**
from *restriction-admD*[*OF* ‹*restriction-adm P*› ‹$(\lambda n.\ t\ (\sigma\ n))\ -\downarrow\rightarrow\ t\ \Sigma$› ‹$\bigwedge n.\ P\ (t\ (\sigma\ n))$›]
show ‹$P\ (t\ \Sigma)$› **.**
qed

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 −
show ‹$adm_\downarrow\ (\lambda x.\ P\ (x,\ y))$›
proof (*rule restriction-admI*)
show ‹$P\ (\Sigma,\ y)$› **if** ‹$\sigma\ -\downarrow\rightarrow\ \Sigma$› ‹$\bigwedge n.\ P\ (\sigma\ n,\ y)$› **for** $\sigma\ \Sigma$
proof (*rule restriction-admD*[*OF* ‹$adm_\downarrow\ P$› - ‹$\bigwedge n.\ P\ (\sigma\ n,\ y)$›])
show ‹$(\lambda n.\ (\sigma\ n,\ y))\ -\downarrow\rightarrow\ (\Sigma,\ y)$›
by (*simp add*: *restriction-tendsto-prod-iff restriction-tendsto-const* ‹$\sigma\ -\downarrow\rightarrow\ \Sigma$›)
qed
qed
next
show ‹$adm_\downarrow\ (\lambda y.\ P\ (x,\ y))$›
proof (*rule restriction-admI*)
show ‹$P\ (x,\ \Sigma)$› **if** ‹$\sigma\ -\downarrow\rightarrow\ \Sigma$› ‹$\bigwedge n.\ P\ (x,\ \sigma\ n)$› **for** $\sigma\ \Sigma$
proof (*rule restriction-admD*[*OF* ‹$adm_\downarrow\ P$› - ‹$\bigwedge n.\ P\ (x,\ \sigma\ n)$›])
show ‹$(\lambda n.\ (x,\ \sigma\ n))\ -\downarrow\rightarrow\ (x,\ \Sigma)$›
by (*simp add*: *restriction-tendsto-prod-iff restriction-tendsto-const* ‹$\sigma\ -\downarrow\rightarrow\ \Sigma$›)
qed
qed
qed

lemma *restriction-adm-restriction-shift-on* [*restriction-adm-simpset*] :
‹$adm_\downarrow\ (\lambda f.\ restriction\text{-}shift\text{-}on\ f\ k\ A)$›
proof (*rule restriction-admI*)
fix σ :: ‹$nat \Rightarrow\ 'a \Rightarrow\ 'b$› **and** Σ :: ‹$'a \Rightarrow\ 'b$›
assume ‹$\sigma\ -\downarrow\rightarrow\ \Sigma$› **and** *hyp* : ‹*restriction-shift-on* $(\sigma\ n)\ k\ A$› **for** n
show ‹*restriction-shift-on* $\Sigma\ k\ A$›
proof (*rule restriction-shift-onI*)
fix $x\ y\ n$ **assume** ‹$x \in A$› ‹$y \in A$› ‹$x \downarrow n = y \downarrow n$›
from *hyp*[*THEN restriction-shift-onD*, *OF this*]
have $*$: ‹$\sigma\ m\ x \downarrow nat\ (int\ n + k) = \sigma\ m\ y \downarrow nat\ (int\ n + k)$› **for** m **.**
show ‹$\Sigma\ x \downarrow nat\ (int\ n + k) = \Sigma\ y \downarrow nat\ (int\ n + k)$›
proof (*rule restriction-tendsto-unique*)
show ‹$(\lambda m.\ \sigma\ m\ x \downarrow nat\ (int\ n + k))\ -\downarrow\rightarrow\ \Sigma\ x \downarrow nat\ (int\ n + k)$›

by (*simp add*: ‹σ $-\downarrow\rightarrow$ Σ› *restriction-tendsto-const-restricted restriction-tendsto-fun-imp*)
next
show ‹($\lambda m.$ σ m x $\downarrow$ *nat* (*int* n + k)) $-\downarrow\rightarrow$ Σ y $\downarrow$ *nat* (*int* n + k)›
by (*simp add*: ∗ ‹σ $-\downarrow\rightarrow$ Σ› *restriction-tendsto-const-restricted restriction-tendsto-fun-imp*)
qed
qed
qed

lemma *restriction-adm-constructive-on* [*restriction-adm-simpset*] :
‹$adm_{\downarrow}$ ($\lambda f.$ *constructive-on* f A)›
by (*simp add*: *constructive-on-def restriction-adm-restriction-shift-on*)

lemma *restriction-adm-non-destructive-on* [*restriction-adm-simpset*] :
‹$adm_{\downarrow}$ ($\lambda f.$ *non-destructive-on* f A)›
by (*simp add*: *non-destructive-on-def restriction-adm-restriction-shift-on*)

lemma *restriction-adm-restriction-cont-at* [*restriction-adm-simpset*] :
‹$adm_{\downarrow}$ ($\lambda f.$ $cont_{\downarrow}$ f *at* a)›
proof (*rule restriction-admI*)
fix σ :: ‹*nat* ⇒ $'a$ ⇒ $'b$› **and** Σ :: ‹$'a$ ⇒ $'b$›
assume ‹σ $-\downarrow\rightarrow$ Σ› **and** *hyp* : ‹$cont_{\downarrow}$ (σ n) *at* a› **for** n
show ‹$cont_{\downarrow}$ Σ *at* a›
proof (*rule restriction-cont-atI*)
fix γ **assume** ‹γ $-\downarrow\rightarrow$ a›
from *hyp*[*THEN restriction-cont-atD*, *OF this*, *THEN restriction-tendstoD*]
have ‹∃ *n0*. ∀ $k{\geq}n0$. σ m a $\downarrow$ n = σ m (γ k) $\downarrow$ n› **for** m n **.**
moreover from ‹σ $-\downarrow\rightarrow$ Σ›[*THEN restriction-tendstoD*]
have ‹∃ *n0*. ∀ $k{\geq}n0$. Σ $\downarrow$ n = σ k $\downarrow$ n› **for** n **.**
ultimately show ‹($\lambda n.$ Σ (γ n)) $-\downarrow\rightarrow$ Σ a›
by (*intro restriction-tendstoI*) (*metis restriction-fun-def*)
qed
qed

lemma *restriction-adm-restriction-cont-on* [*restriction-adm-simpset*] :
‹$adm_{\downarrow}$ ($\lambda f.$ $cont_{\downarrow}$ f *on* A)›
unfolding *restriction-cont-on-def*
by (*intro restriction-adm-ball restriction-adm-restriction-cont-at*)

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*] :
‹$adm_{\downarrow}$ (λf. *non-destructive* f)›
and *restriction-adm-restriction-cont* [*restriction-adm-simpset*] :
‹$adm_{\downarrow}$ (λf. $cont_{\downarrow}$ f)›
by (*simp-all add*: *restriction-adm-simpset restriction-shift-def constructive-def non-destructive-def*)

lemma (**in** *restriction*) *restriction-adm-mem-restriction-closed* [*restriction-adm-simpset*] :
‹$closed_{\downarrow}$ $K \Longrightarrow adm_{\downarrow}$ (λx. $x \in K$)›
by (*auto intro*!: *restriction-admI dest*: *restriction-closed-sequentialD*)

lemma (**in** *restriction-space*) *restriction-adm-mem-restriction-compact* [*restriction-adm-simpset*] :
‹$compact_{\downarrow}$ $K \Longrightarrow adm_{\downarrow}$ (λx. $x \in K$)›
by (*simp add*: *restriction-adm-mem-restriction-closed restriction-compact-imp-restriction-closed*)

lemma (**in** *restriction-space*) *restriction-adm-mem-finite* [*restriction-adm-simpset*] :
‹*finite* $S \Longrightarrow adm_{\downarrow}$ (λx. $x \in S$)›
by (*simp add*: *finite-imp-restriction-compact restriction-adm-mem-restriction-compact*)

lemma *restriction-adm-restriction-tendsto* [*restriction-adm-simpset*] :
‹$adm_{\downarrow}$ ($\lambda\sigma$. σ $-\downarrow\rightarrow$ Σ)›
by (*intro restriction-admI restriction-tendstoI*)
(*metis* (*no-types*, *opaque-lifting*) *restriction-fun-def restriction-tendsto-def*)

lemma *restriction-adm-lim* [*restriction-adm-simpset*] :
‹$adm_{\downarrow}$ ($\lambda\Sigma$. σ $-\downarrow\rightarrow$ Σ)›
by (*metis restriction-admI restriction-openD restriction-open-restriction-cball restriction-tendsto-iff-restriction-cball-characterization*)

lemma *restriction-restriction-cont-on* [*restriction-cont-simpset*] :
‹$cont_{\downarrow}$ f *on* $A \Longrightarrow cont_{\downarrow}$ (λx. f $x \downarrow n$) *on* A›
by (*rule restriction-cont-onI*)
(*simp add*: *restriction-cont-onD restriction-tendsto-const-restricted*)

lemma *restriction-cont-on-id* [*restriction-cont-simpset*] : ‹$cont_{\downarrow}$ (λx. x) *on* A›
by (*simp add*: *restriction-cont-onI*)

lemma *restriction-cont-on-const* [*restriction-cont-simpset*] : ‹$cont_{\downarrow}$ (λx. c) *on* A›
by (*simp add*: *restriction-cont-onI restriction-tendstoI*)

lemma *restriction-cont-on-fun* [*restriction-cont-simpset*] : ‹$cont_{\downarrow}$ ($\lambda f.$ $f\ x$) *on* A›
by (*rule restriction-cont-onI*) (*simp add*: *restriction-tendsto-fun-imp*)

lemma *restriction-cont2cont-on-fun* [*restriction-cont-simpset*] :
‹$cont_{\downarrow}$ f *on* $A \Longrightarrow$ $cont_{\downarrow}$ ($\lambda x.$ $f\ x\ y$) *on* A›
by (*rule restriction-cont-onI*)
(*metis restriction-cont-onD restriction-tendsto-fun-imp*)

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* $f \Longrightarrow$ $adm_{\downarrow}$ $P \Longrightarrow$ ($\bigwedge n.$ P ((f ^^ n) x)) $\Longrightarrow$ P ($\upsilon\ x.\ f$ x)›
using *restriction-admD funpow-restriction-tendsto-restriction-fix* **by** *blast*

lemma *restriction-fix-ind* [*case-names constructive adm base step*] :
‹P ($\upsilon\ x.\ f\ x$)› **if** ‹*constructive* f› ‹$adm_{\downarrow}$ P› ‹$P\ x$› ‹$\bigwedge x.\ P\ x \Longrightarrow P$ (f x)›
proof (*induct rule*: *restriction-fix-ind'*)
show ‹*constructive* f› **by** (*fact* ‹*constructive* f›)
next
show ‹*restriction-adm* P› **by** (*fact* ‹*restriction-adm* P›)
next
show ‹P ((f ^^ n) x)› **for** n
by (*induct n*) (*simp-all add*: ‹$P\ x$› ‹$\bigwedge x.\ P\ x \Longrightarrow P$ ($f\ x$)›)
qed

lemma *restriction-fix-ind2* [*case-names constructive adm base0 base1 step*] :
‹P ($\upsilon\ x.\ f\ x$)› **if** ‹*constructive* f› ‹$adm_{\downarrow}$ P› ‹$P\ x$› ‹P ($f\ x$)›
‹$\bigwedge x.$ ⟦$P\ x$; P ($f\ x$)⟧ $\Longrightarrow$ P (f ($f\ x$))›
proof (*induct rule*: *restriction-fix-ind'*)
show ‹*constructive* f› **by** (*fact* ‹*constructive* f›)
next
show ‹*restriction-adm* P› **by** (*fact* ‹*restriction-adm* P›)
next

show ‹$P\ ((f\ \widehat{\ }\ n)\ x)$› **for** n
by (*induct n rule*: *induct-nat-012*) (*simp-all add*: *that*(*3*−*5*))
qed

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* : ‹$\bigwedge x\ y.\ P\ x\ y \Longrightarrow P\ (f\ x)\ (g\ y)$›
shows ‹$P\ (\upsilon\ x.\ f\ x)\ (\upsilon\ y.\ g\ y)$›
proof −
define F **where** ‹$F \equiv \lambda(x,\ y).\ (f\ x,\ g\ y)$›
define Q **where** ‹$Q \equiv \lambda(x,\ y).\ P\ x\ y$›

have ‹$P\ (\upsilon\ x.\ f\ x)\ (\upsilon\ y.\ g\ y) = Q\ (\upsilon\ p.\ F\ p)$›
by (*simp add*: *F-def Q-def constructive restriction-fix-indep-prod-is*)
also have ‹$Q\ (\upsilon\ p.\ F\ p)$›
proof (*induct F rule* : *restriction-fix-ind*)
show ‹*constructive F*›
by (*simp add*: *F-def constructive-prod-codomain-iff constructive-prod-domain-iff constructive constructive-const*)
next
show ‹*restriction-adm Q*›
by (*unfold Q-def*) (*metis* (*mono-tags*, *lifting*) *adm case-prod-beta restriction-adm-def*)
next
show ‹$Q\ (x,\ y)$› **by** (*simp add*: *Q-def base*)
next
show ‹$Q\ p \Longrightarrow Q\ (F\ p)$› **for** p **by** (*simp add*: *Q-def F-def step split-beta*)
qed
finally show ‹$P\ (\upsilon\ x.\ f\ x)\ (\upsilon\ y.\ g\ y)$› **.**
qed

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 ‹$\forall\, i < k.\ P\ ((f\ \widehat{\ }\ i)\ x)$›
and ‹$\bigwedge x.\ \forall\, i < k.\ P\ ((f\ \widehat{\ }\ i)\ x) \Longrightarrow P\ ((f\ \widehat{\ }\ k)\ x)$›
shows ‹$P\ (\upsilon\ x.\ f\ x)$›
proof (*rule restriction-fix-ind′*)
show ‹*constructive f*› **by** (*fact* ‹*constructive f*›)
next

show ‹$adm_{\downarrow}$ P› **by** (*fact* ‹$adm_{\downarrow}$ P›)
next
have *nat-k-induct* :
‹P n› **if** ‹$\forall$ i<k. P i› **and** ‹$\forall$ n_0. ($\forall$ i<k. P (n_0 + i)) $\longrightarrow$ P (n_0 + k)› **for** k n :: *nat* **and** P
proof (*induct rule*: *nat-less-induct*)
fix n **assume** ‹$\forall$ m<n. P m›
show ‹P n›
proof (*cases* ‹n < k›)
from *that*(*1*) **show** ‹n < k $\Longrightarrow$ P n› **by** *blast*
next
from ‹$\forall$ m<n. P m› *that*(*2*)[*rule-format*, *of* ‹n − k›]
show ‹$\neg$ n < k $\Longrightarrow$ P n› **by** *auto*
qed
qed
show ‹P ((f ^^ i) x)› **for** i
proof (*induct rule*: *nat-k-induct*)
show ‹$\forall$ i<k. P ((f ^^ i) x)› **by** (*simp add*: *assms*(*3*))
next
show ‹$\forall$ n_0. ($\forall$ i<k. P ((f ^^ (n_0 + i)) x)) $\longrightarrow$ P ((f ^^ (n_0 + k)) x)›
by (*smt* (*verit*, *del-insts*) *add.commute assms*(*4*) *funpow-add o-apply*)
qed
qed

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.

simproc-setup *apply-non-destructiveness* (‹*non-destructive* ($\lambda f.\ E\ f$)›)
= ‹

```
  K (fn ctxt => fn ct =>
    let val t = Thm.term-of ct
        val foo = case t of - $ foo => foo
    in  case foo of Abs (-, -, expr) =>
          if  fst (strip-comb expr) = Bound 0
         (* since ‹λf. E f› is too permissive, we ensure that the term
is of the
              form ‹λf. f ...› (for example ‹λf. f x›, or ‹λf. f x y›,
etc.) *)
          then let val tac = Metis-Tactic.metis-tac [no-types] combs
ctxt
                                              @{thms non-destructive-fun-iff
non-destructive-id(2)}
                   val rwrt-ct  = HOLogic.mk-judgment (Thm.apply
cterm‹λlhs. lhs = True› ct)
                   val rwrt-thm = Goal.prove-internal ctxt [] rwrt-ct
(fn - => tac 1)
               in  SOME (mk-meta-eq rwrt-thm)
               end
          else NONE
        | - => NONE
    end
   )
›
```

lemma ‹*non-destructive* (λ*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*)›
  using [[*simp-trace*]] **by** *simp* — test