Complete Non-Orders and Fixed Points

Akihisa Yamada and Jérémy Dubut

February 23, 2021

Abstract

We develop an Isabelle/HOL library of order-theoretic concepts, such as various completeness conditions and fixed-point theorems. We keep our formalization as general as possible: we reprove several well-known results about complete orders, often without any properties of ordering, thus complete non-orders. In particular, we generalize the Knaster–Tarski theorem so that we ensure the existence of a quasi-fixed point of monotone maps over complete non-orders, and show that the set of quasi-fixed points is complete under a mild condition—attractivity—which is implied by either antisymmetry or transitivity. This result generalizes and strengthens a result by Stauti and Maaden. Finally, we recover Kleene's fixed-point theorem for omega-complete non-orders, again using attractivity to prove that Kleene's fixed points are least quasi-fixed points.

Contents

1 Introduction

The main driving force towards mechanizing mathematics using proof assistants has been the reliability they offer, exemplified prominently by [7], [8], [10], etc. In this work, we utilize another aspect of Isabelle/JEdit [17] as engineering tools for developing mathematical theories. We formalize order-theoretic concepts and results, adhering to an *as-general-as-possible* approach: most results concerning order-theoretic completeness and fixed-point theorems are proved without assuming the underlying relations to be orders (non-orders). In particular, we provide the following:

- A locale-based library for binary relations, as partly depicted in Figure 1.
- Various completeness results that generalize known theorems in order theory: Actually most relationships and duality of completeness conditions are proved without *any* properties of the underlying relations.
- Existence of fixed points: We show that a relation-preserving mapping $f : A \to A$ over a complete non-order $\langle A, \sqsubseteq \rangle$ admits a *quasi-fixed point*

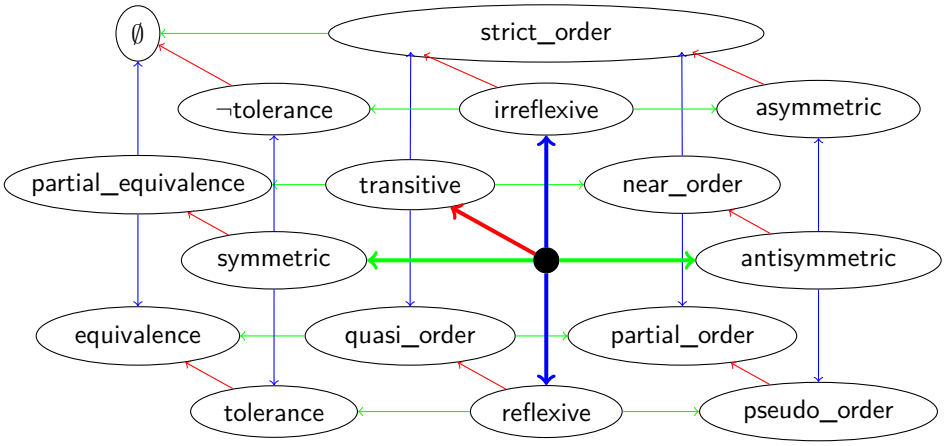

Figure 1: Combinations of basic properties. The black dot around the center represents arbitrary binary relations, and the five outgoing arrows indicate atomic assumptions. We do not present the combination of reflexive and irreflexive, which is empty, and one of symmetric and antisymmetric, which is a subset of equality. Node "¬tolerance" indicates the negated relation is tolerance, and "∅" is the empty relation.

$f(x) \sim x$, meaning $x \sqsubseteq f(x) \wedge f(x) \sqsubseteq x$. Clearly if $\sqsubseteq$ is antisymmetric then this implies the existence of fixed points $f(x) = x$.

- Completeness of the set of fixed points: We further show that if $\sqsubseteq$ satisfies a mild condition, which we call *attractivity* and which is implied by either transitivity or antisymmetry, then the set of quasi-fixed points is complete. Furthermore, we also show that if $\sqsubseteq$ is antisymmetric, then the set of *strict* fixed points $f(x) = x$ is complete.

- Kleene-style fixed-point theorems: For an ω-complete non-order $\langle A, \sqsubseteq \rangle$ with a bottom element $\bot \in A$ (not necessarily unique) and for every ω-continuous map $f : A \to A$, a supremum exists for the set $\{f^n(\bot) \mid n \in \mathbb{N}\}$, and it is a quasi-fixed point. If $\sqsubseteq$ is attractive, then the quasi-fixed points obtained this way are precisely the least quasi-fixed points.

We remark that all these results would have required much more effort than we spent (if possible at all), if we were not with the smart assistance by Isabelle. Our workflow was often the following: first we formalize existing proofs, try relaxing assumptions, see where proof breaks, and at some point ask for a counterexample.

Related Work Many attempts have been made to generalize the notion of completeness for lattices, conducted in different directions: by relaxing

the notion of order itself, removing transitivity (pseudo-orders [15]); by relaxing the notion of lattice, considering minimal upper bounds instead of least upper bounds (χ-posets [11]); by relaxing the notion of completeness, requiring the existence of least upper bounds for restricted classes of subsets (e.g., directed complete and ω-complete, see [5] for a textbook). Considering those generalizations, it was natural to prove new versions of classical fixed-point theorems for maps preserving those structures, e.g., existence of least fixed points for monotone maps on (weak chain) complete pseudo-orders [4, 16], construction of least fixed points for ω-continuous functions for ω-complete lattices [12], (weak chain) completeness of the set of fixed points for monotone functions on (weak chain) complete pseudo-orders [13].

Concerning Isabelle formalization, one can easily find several formalizations of complete partial orders or lattices in Isabelle's standard library. They are, however, defined on partial orders, either in form of classes or locales, and thus not directly reusable for non-orders. Nevertheless we tried to make our formalization compatible with the existing ones, and various correspondences are ensured.

This work has been published in the conference paper [18].

2 Binary Relations

We start with basic properties of binary relations.

theory *Binary-Relations*
imports
Main
begin

lemma *conj-imp-eq-imp-imp*: $(P \wedge Q \Longrightarrow PROP\ R) \equiv (P \Longrightarrow Q \Longrightarrow PROP\ R)$
by *standard simp-all*

lemma *tranclp-trancl*: $r^{++} = (\lambda x\ y.\ (x,y) \in \{(a,b).\ r\ a\ b\}^+)$
by (*auto simp*: *tranclp-trancl-eq*[*symmetric*])

lemma *tranclp-id*[*simp*]: *transp* $r \Longrightarrow$ *tranclp* $r = r$
using *trancl-id*[*of* $\{(x,y).\ r\ x\ y\}$, *folded transp-trans*] **by** (*auto simp*:*tranclp-trancl*)

lemma *transp-tranclp*[*simp*]: *transp* (*tranclp r*) **by** (*auto simp*: *tranclp-trancl transp-trans*)

lemma *funpow-dom*: $f\ `\ A \subseteq A \Longrightarrow (f\widehat{\ }\widehat{\ }n)\ `\ A \subseteq A$ **by** (*induct n*, *auto*)

Below we introduce an Isabelle-notation for $\{\ldots x \ldots \mid x \in X\}$.

syntax
-range :: *'a* ⇒ *idts* ⇒ *'a set* ((*1*{*-* /|./ *-*}))
-image :: *'a* ⇒ *pttrn* ⇒ *'a set* ⇒ *'a set* ((*1*{*-* /|./ (*-*/ ∈ *-*)}))
translations
$\{e \mid.\ p\} \rightleftharpoons \{e \mid p.\ CONST\ True\}$

$\{e \mid .\ p \in A\} \rightleftharpoons CONST\ image\ (\lambda p.\ e)\ A$

lemma *image-constant*:
assumes $\bigwedge i.\ i \in I \Longrightarrow f\ i = y$
shows $f\ `\ I = (if\ I = \{\}\ then\ \{\}\ else\ \{y\})$
using *assms* **by** *auto*

2.1 Various Definitions

Here we introduce various definitions for binary relations. The first one is our abbreviation for the dual of a relation.

abbreviation(*input*) *dual* $((\text{-}^{-})\ [1000]\ 1000)$ **where** $r^{-}\ x\ y \equiv r\ y\ x$

lemma *conversep-is-dual*[*simp*]: *conversep* = *dual* **by** *auto*

Monotonicity is already defined in the library, but we want one restricted to a domain.

definition *monotone-on* $X\ r\ s\ f \equiv \forall x\ y.\ x \in X \longrightarrow y \in X \longrightarrow r\ x\ y \longrightarrow s\ (f\ x)\ (f\ y)$

lemmas *monotone-onI* = *monotone-on-def*[*unfolded atomize-eq*, *THEN iffD2*, *rule-format*]
lemma *monotone-onD*: *monotone-on* $X\ r\ s\ f \Longrightarrow r\ x\ y \Longrightarrow x \in X \Longrightarrow y \in X \Longrightarrow s\ (f\ x)\ (f\ y)$
by (*auto simp*: *monotone-on-def*)

lemmas *monotone-onE* = *monotone-on-def*[*unfolded atomize-eq*, *THEN iffD1*, *elim-format*, *rule-format*]

lemma *monotone-on-UNIV*[*simp*]: *monotone-on UNIV* = *monotone*
by (*intro ext*, *auto simp*: *monotone-on-def monotone-def*)

lemma *monotone-on-dual*: *monotone-on* $X\ r\ s\ f \Longrightarrow$ *monotone-on* $X\ r^{-}\ s^{-}\ f$
by (*auto simp*: *monotone-on-def*)

lemma *monotone-on-id*: *monotone-on* $X\ r\ r\ id$
by (*auto simp*: *monotone-on-def*)

lemma *monotone-on-cmono*: $A \subseteq B \Longrightarrow$ *monotone-on* $B \leq$ *monotone-on* A
by (*intro le-funI*, *auto simp*: *monotone-on-def*)

Here we define the following notions in a standard manner

The symmetric part of a relation:

definition *sympartp* **where** *sympartp* $r\ x\ y \equiv r\ x\ y \wedge r\ y\ x$

lemma *sympartpI*[*intro*]:
fixes r (**infix** $\sqsubseteq$ *50*)
assumes $x \sqsubseteq y$ **and** $y \sqsubseteq x$ **shows** *sympartp* $(\sqsubseteq)\ x\ y$
using *assms* **by** (*auto simp*: *sympartp-def*)

lemma *sympartpE*[*elim*]:
fixes r (**infix** $\sqsubseteq$ *50*)
assumes *sympartp* ($\sqsubseteq$) x y **and** $x \sqsubseteq y \Longrightarrow y \sqsubseteq x \Longrightarrow$ *thesis* **shows** *thesis*
using *assms* **by** (*auto simp*: *sympartp-def*)

lemma *sympartp-dual*: *sympartp* r^{-} = *sympartp* r
by (*auto intro*!:*ext simp*: *sympartp-def*)

lemma *sympartp-eq*[*simp*]: *sympartp* (=) = (=) **by** *auto*

lemma *reflclp-sympartp*[*simp*]: (*sympartp* r)$^{==}$ = *sympartp* $r^{==}$ **by** *auto*

definition *equivpartp* r x $y \equiv x = y \vee r\ x\ y \wedge r\ y\ x$

lemma *sympartp-reflclp-equivp*[*simp*]: *sympartp* $r^{==}$ = *equivpartp* r **by** (*auto intro*!:*ext simp*: *equivpartp-def*)

lemma *equivpartI*[*simp*]: *equivpartp* r x x
and *sympartp-equivpartpI*: *sympartp* r x $y \Longrightarrow$ *equivpartp* r x y
and *equivpartpCI*[*intro*]: ($x \neq y \Longrightarrow$ *sympartp* r x y) $\Longrightarrow$ *equivpartp* r x y
by (*auto simp*:*equivpartp-def*)

lemma *equivpartpE*[*elim*]:
assumes *equivpartp* r x y
and $x = y \Longrightarrow$ *thesis*
and $r\ x\ y \Longrightarrow r\ y\ x \Longrightarrow$ *thesis*
shows *thesis*
using *assms* **by** (*auto simp*: *equivpartp-def*)

lemma *equivpartp-eq*[*simp*]: *equivpartp* (=) = (=) **by** *auto*

lemma *sympartp-equivpartp*[*simp*]: *sympartp* (*equivpartp* r) = (*equivpartp* r)
and *equivpartp-equivpartp*[*simp*]: *equivpartp* (*equivpartp* r) = (*equivpartp* r)
and *equivpartp-sympartp*[*simp*]: *equivpartp* (*sympartp* r) = (*equivpartp* r)
by (*auto 0 5 intro*!:*ext*)

lemma *equivpartp-dual*: *equivpartp* r^{-} = *equivpartp* r
by (*auto intro*!:*ext simp*: *equivpartp-def*)

The asymmetric part:

definition *asympartp* r x $y \equiv r\ x\ y \wedge \neg\ r\ y\ x$

lemma *asympartpE*[*elim*]:
fixes r (**infix** $\sqsubseteq$ *50*)
shows *asympartp* ($\sqsubseteq$) x $y \Longrightarrow (x \sqsubseteq y \Longrightarrow \neg y \sqsubseteq x \Longrightarrow$ *thesis*) $\Longrightarrow$ *thesis*
by (*auto simp*: *asympartp-def*)

lemmas *asympartpI*[*intro*] = *asympartp-def*[*unfolded atomize-eq*, *THEN iffD2*, *un-*

folded conj-imp-eq-imp-imp, *rule-format*]

lemma *asympartp-eq*[*simp*]: *asympartp* (=) = *bot* **by** *auto*

lemma *asympartp-sympartp* [*simp*]: *asympartp* (*sympartp r*) = *bot*
and *sympartp-asympartp* [*simp*]: *sympartp* (*asympartp r*) = *bot*
by (*auto intro*!: *ext*)

Restriction to a set:

definition *Restrp* (**infixl** ↾ *60*) **where** (*r* ↾ *A*) *a b* ≡ *a* ∈ *A* ∧ *b* ∈ *A* ∧ *r a b*

lemmas *RestrpI*[*intro*!] = *Restrp-def*[*unfolded atomize-eq*, *THEN iffD2*, *unfolded conj-imp-eq-imp-imp*]
lemmas *RestrpE*[*elim*!] = *Restrp-def*[*unfolded atomize-eq*, *THEN iffD1*, *elim-format*, *unfolded conj-imp-eq-imp-imp*]

lemma *Restrp-UNIV*[*simp*]: *r* ↾ *UNIV* ≡ *r* **by** (*auto simp*: *atomize-eq*)

lemma *Restrp-Restrp*[*simp*]: *r* ↾ *A* ↾ *B* ≡ *r* ↾ *A* ∩ *B* **by** (*auto simp*: *atomize-eq Restrp-def*)

lemma *sympartp-Restrp*[*simp*]: *sympartp* (*r* ↾ *A*) ≡ *sympartp r* ↾ *A*
by (*auto simp*: *atomize-eq*)

Relational images:

definition *Imagep* (**infixr** ‘‘‘ *59*) **where** *r* ‘‘‘ *A* ≡ {*b*. ∃ *a* ∈ *A*. *r a b*}

lemma *Imagep-Image*: *r* ‘‘‘ *A* = {(*a*,*b*). *r a b*} ‘‘ *A*
by (*auto simp*: *Imagep-def*)

lemma *in-Imagep*: *b* ∈ *r* ‘‘‘ *A* ⟷ (∃ *a* ∈ *A*. *r a b*) **by** (*auto simp*: *Imagep-def*)

lemma *ImagepI*: *a* ∈ *A* ⟹ *r a b* ⟹ *b* ∈ *r* ‘‘‘ *A* **by** (*auto simp*: *in-Imagep*)

lemma *subset-Imagep*: *B* ⊆ *r* ‘‘‘ *A* ⟷ (∀ *b*∈*B*. ∃ *a*∈*A*. *r a b*)
by (*auto simp*: *Imagep-def*)

Bounds of a set:

definition *bound X r b* ≡ ∀ *x* ∈ *X*. *r x b*

lemma
fixes *r* (**infix** ⊑ *50*)
shows *boundI*[*intro*!]: (⋀*x*. *x* ∈ *X* ⟹ *x* ⊑ *b*) ⟹ *bound X* (⊑) *b*
and *boundE*[*elim*]: *bound X* (⊑) *b* ⟹ ((⋀*x*. *x* ∈ *X* ⟹ *x* ⊑ *b*) ⟹ *thesis*) ⟹ *thesis*
by (*auto simp*: *bound-def*)

lemma *bound-empty*: *bound* {} = (λ*r x*. *True*) **by** *auto*

lemma *bound-insert*[*simp*]:
fixes r (**infix** $\sqsubseteq$ *50*)
shows *bound* (*insert x X*) ($\sqsubseteq$) $b \longleftrightarrow x \sqsubseteq b \wedge$ *bound X* ($\sqsubseteq$) b **by** *auto*

Extreme (greatest) elements in a set:

definition *extreme X r e* $\equiv e \in X \wedge (\forall x \in X.\ r\ x\ e)$

lemma
fixes r (**infix** $\sqsubseteq$ *50*)
shows *extremeI*[*intro*]: $e \in X \Longrightarrow (\bigwedge x.\ x \in X \Longrightarrow x \sqsubseteq e) \Longrightarrow$ *extreme X* ($\sqsubseteq$) e
and *extremeD*: *extreme X* ($\sqsubseteq$) $e \Longrightarrow e \in X$ *extreme X* ($\sqsubseteq$) $e \Longrightarrow (\bigwedge x.\ x \in X \Longrightarrow x \sqsubseteq e)$
and *extremeE*[*elim*]: *extreme X* ($\sqsubseteq$) $e \Longrightarrow (e \in X \Longrightarrow (\bigwedge x.\ x \in X \Longrightarrow x \sqsubseteq e) \Longrightarrow$ *thesis*$) \Longrightarrow$ *thesis*
by (*auto simp*: *extreme-def*)

lemma
fixes r (**infix** $\sqsubseteq$ *50*)
shows *extreme-UNIV*[*simp*]: *extreme UNIV* ($\sqsubseteq$) $t \longleftrightarrow (\forall x.\ x \sqsubseteq t)$ **by** *auto*

lemma *extremes-equiv*: *extreme X r b* $\Longrightarrow$ *extreme X r c* $\Longrightarrow$ *sympartp r b c* **by** *blast*

lemma *bound-cmono*: **assumes** $X \subseteq Y$ **shows** *bound Y* $\leq$ *bound X*
using *assms* **by** *auto*

lemma *sympartp-sympartp*[*simp*]: *sympartp* (*sympartp r*) = *sympartp r* **by** (*auto intro*!:*ext*)

Now suprema and infima are given uniformly as follows. The definition is restricted to a given set.

context
fixes $A :: 'a$ *set* **and** *less-eq* $:: 'a \Rightarrow 'a \Rightarrow$ *bool* (**infix** $\sqsubseteq$ *50*)
begin

abbreviation *extreme-bound X* $\equiv$ *extreme* $\{b \in A.$ *bound X* ($\sqsubseteq$) $b\}$ $(\lambda x\ y.\ y \sqsubseteq x)$

lemma *extreme-boundI*[*intro*]:
assumes $\bigwedge b.$ *bound X* ($\sqsubseteq$) $b \Longrightarrow b \in A \Longrightarrow s \sqsubseteq b$ **and** $\bigwedge x.\ x \in X \Longrightarrow x \sqsubseteq s$ **and** $s \in A$
shows *extreme-bound X s*
using *assms* **by** *auto*

lemma *extreme-bound-bound*: *extreme-bound X y* $\Longrightarrow x \in X \Longrightarrow x \sqsubseteq y$ **by** *auto*

lemma *extreme-bound-mono*:
assumes *XY*: $X \subseteq Y$
and *sX*: *extreme-bound X sX*
and *sY*: *extreme-bound Y sY*

shows $sX \sqsubseteq sY$
proof–
have *bound X* ($\sqsubseteq$) *sY* **using** *XY sY* **by** *force*
with *sX sY* **show** *?thesis* **by** (*auto 0 4*)
qed

lemma *extreme-bound-iff*:
shows *extreme-bound X s* $\longleftrightarrow$ $s \in A \wedge (\forall\, c \in A.\ (\forall\, x \in X.\ x \sqsubseteq c) \longrightarrow s \sqsubseteq c) \wedge (\forall\, x \in X.\ x \sqsubseteq s)$
by (*auto simp*: *extreme-def*)

lemma *extreme-bound-singleton-refl*[*simp*]:
extreme-bound $\{x\}$ *x* $\longleftrightarrow$ $x \in A \wedge x \sqsubseteq x$ **by** *auto*

lemma *extreme-bound-image-const*:
$x \sqsubseteq x \Longrightarrow I \neq \{\} \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow f\ i = x) \Longrightarrow x \in A \Longrightarrow$ *extreme-bound* (*f* ‘ *I*) *x*
by (*auto simp*: *image-constant*)

lemma *extreme-bound-UN-const*:
$x \sqsubseteq x \Longrightarrow I \neq \{\} \Longrightarrow (\bigwedge i\ y.\ i \in I \Longrightarrow P\ i\ y \longleftrightarrow x = y) \Longrightarrow x \in A \Longrightarrow$
extreme-bound $(\bigcup i{\in}I.\ \{y.\ P\ i\ y\})$ *x*
by *auto*

end

context
fixes *ir* :: $'i \Rightarrow 'i \Rightarrow$ *bool* (**infix** $\preceq$ *50*)
and *r* :: $'a \Rightarrow 'a \Rightarrow$ *bool* (**infix** $\sqsubseteq$ *50*)
and *f* **and** *A* **and** *e* **and** *I*
assumes *fIA*: *f* ‘ *I* $\subseteq$ *A*
and *mono*: *monotone-on I* ($\preceq$) ($\sqsubseteq$) *f*
and *e*: *extreme I* ($\preceq$) *e*
begin

lemma *monotone-extreme-imp-extreme-bound*:
extreme-bound A ($\sqsubseteq$) (*f* ‘ *I*) (*f e*)
using *monotone-onD*[*OF mono*] *e fIA*
by (*intro extreme-boundI*, *auto simp*: *image-def elim*!: *extremeE*)

lemma *monotone-extreme-extreme-boundI*:
x = *f e* $\Longrightarrow$ *extreme-bound A* ($\sqsubseteq$) (*f* ‘ *I*) *x*
using *monotone-extreme-imp-extreme-bound* **by** *auto*

end

2.2 Locales for Binary Relations

We now define basic properties of binary relations, in form of *locales* [9, 2].

2.2.1 Syntactic Locales

The following locales do not assume anything, but provide infix notations for relations.

locale *less-eq-syntax* =
  fixes *less-eq* :: $'a \Rightarrow 'a \Rightarrow$ *bool* (**infix** $\sqsubseteq$ *50*)

locale *less-syntax* =
  fixes *less* :: $'a \Rightarrow 'a \Rightarrow$ *bool* (**infix** $\sqsubset$ *50*)

locale *equivalence-syntax* =
  fixes *equiv* :: $'a \Rightarrow 'a \Rightarrow$ *bool* (**infix** $\sim$ *50*)
begin

abbreviation *equiv-class* ($[\text{-}]_\sim$) **where** $[x]_\sim \equiv \{\ y.\ x \sim y\ \}$

end

Next ones introduce abbreviations for dual etc. To avoid needless constants, one should be careful when declaring them as sublocales.

locale *less-eq-notations* = *less-eq-syntax*
begin

abbreviation (*input*) *greater-eq* (**infix** $\sqsupseteq$ *50*) **where** $x \sqsupseteq y \equiv y \sqsubseteq x$
abbreviation *sym* (**infix** $\sim$ *50*) **where** $(\sim) \equiv$ *sympartp* $(\sqsubseteq)$
abbreviation *less* (**infix** $\sqsubset$ *50*) **where** $(\sqsubset) \equiv$ *asympartp* $(\sqsubseteq)$
abbreviation *greater* (**infix** $\sqsupset$ *50*) **where** $(\sqsupset) \equiv (\sqsubset)^-$
abbreviation *equiv* (**infix** $(\simeq)$ *50*) **where** $(\simeq) \equiv$ *equivpartp* $(\sqsubseteq)$

lemma *asym-cases*[*consumes 1*, *case-names asym sym*]:
  assumes $x \sqsubseteq y$ **and** $x \sqsubset y \Longrightarrow$ *thesis* **and** $x \sim y \Longrightarrow$ *thesis*
  shows *thesis*
  using *assms* **by** *auto*

end

locale *less-notations* = *less-syntax*
begin

abbreviation (*input*) *greater* (**infix** $\sqsupset$ *50*) **where** $x \sqsupset y \equiv y \sqsubset x$

end

locale *related-set* =
  fixes *A* :: $'a$ *set* **and** *less-eq* :: $'a \Rightarrow 'a \Rightarrow$ *bool* (**infix** $\sqsubseteq$ *50*)

2.2.2 Basic Properties of Relations

In the following we define basic properties in form of locales.

Reflexivity restricted on a set:

locale *reflexive* = *related-set* +
  assumes *refl*[*intro*]: $x \in A \Longrightarrow x \sqsubseteq x$
begin

lemma *eq-implies*: $x = y \Longrightarrow x \in A \Longrightarrow x \sqsubseteq y$ **by** *auto*

lemma *extreme-singleton*[*simp*]: $x \in A \Longrightarrow$ *extreme* $\{x\}$ $(\sqsubseteq)$ $y \longleftrightarrow x = y$ **by** *auto*

lemma *extreme-bound-singleton*: $x \in A \Longrightarrow$ *extreme-bound* A $(\sqsubseteq)$ $\{x\}$ x **by** *auto*

lemma *reflexive-subset*: $B \subseteq A \Longrightarrow$ *reflexive* B $(\sqsubseteq)$ **apply** *unfold-locales* **by** *auto*

end

declare *reflexive.intro*[*intro*!]

lemma *reflexiveE*[*elim*]:
  assumes *reflexive* A r **and** $(\bigwedge x.\ x \in A \Longrightarrow r\ x\ x) \Longrightarrow$ *thesis* **shows** *thesis*
  using *assms* **by** (*auto simp*: *reflexive.refl*)

lemma *reflexive-cong*:
  $(\bigwedge a\ b.\ a \in A \Longrightarrow b \in A \Longrightarrow r\ a\ b \longleftrightarrow r'\ a\ b) \Longrightarrow$ *reflexive* A $r \longleftrightarrow$ *reflexive* A r'
  by (*simp add*: *reflexive-def*)

locale *irreflexive* = *related-set* A $(\sqsubset)$ **for** A **and** *less* (**infix** $\sqsubset$ *50*) +
  assumes *irrefl*: $x \in A \Longrightarrow \neg\ x \sqsubset x$
begin

lemma *irreflD*[*simp*]: $x \sqsubset x \Longrightarrow \neg x \in A$ **by** (*auto simp*: *irrefl*)

lemma *implies-not-eq*: $x \sqsubset y \Longrightarrow x \in A \Longrightarrow x \neq y$ **by** *auto*

lemma *Restrp-irreflexive*: *irreflexive UNIV* $((\sqsubset)\restriction A)$
  apply *unfold-locales* **by** *auto*

lemma *irreflexive-subset*: $B \subseteq A \Longrightarrow$ *irreflexive* B $(\sqsubset)$ **apply** *unfold-locales* **by** *auto*

end

declare *irreflexive.intro*[*intro*!]

lemma *irreflexive-cong*:
  $(\bigwedge a\ b.\ a \in A \Longrightarrow b \in A \Longrightarrow r\ a\ b \longleftrightarrow r'\ a\ b) \Longrightarrow$ *irreflexive* A $r \longleftrightarrow$ *irreflexive* A r'
  by (*simp add*: *irreflexive-def*)

locale *transitive* = *related-set* +
  assumes *trans*[*trans*]: $x \sqsubseteq y \Longrightarrow y \sqsubseteq z \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow z \in A \Longrightarrow x \sqsubseteq z$
begin

interpretation *less-eq-notations*.

lemma *Restrp-transitive*: *transitive UNIV* $((\sqsubseteq)\restriction A)$
  apply *unfold-locales*
  by (*auto intro*: *trans*)

lemma *bound-trans*[*trans*]: *bound X* $(\sqsubseteq)$ *b* $\Longrightarrow b \sqsubseteq c \Longrightarrow X \subseteq A \Longrightarrow b \in A \Longrightarrow c \in A \Longrightarrow$ *bound X* $(\sqsubseteq)$ *c*
  by (*auto 0 4 dest*: *trans*)

lemma *transitive-subset*:
  assumes *BA*: $B \subseteq A$ **shows** *transitive B* $(\sqsubseteq)$
  apply *unfold-locales*
  using *trans BA* **by** *blast*

lemma *asympartp-transitive*: *transitive A* $(\sqsubset)$
  apply *unfold-locales* **by** (*auto dest*:*trans*)

lemma *reflclp-transitive*: *transitive A* $(\sqsubseteq)^{==}$
  apply *unfold-locales* **by** (*auto dest*: *trans*)

The symmetric part is also transitive, but this is done in the later semi-attractive locale

end

declare *transitive.intro*[*intro?*]

lemma *transitive-cong*:
  assumes *r*: $\bigwedge a\ b.\ a \in A \Longrightarrow b \in A \Longrightarrow r\ a\ b \longleftrightarrow r'\ a\ b$ **shows** *transitive A r* $\longleftrightarrow$ *transitive A r'*
proof (*intro iffI*)
  show *transitive A r* $\Longrightarrow$ *transitive A r'*
    apply (*intro transitive.intro*)
    apply (*unfold r*[*symmetric*])
    using *transitive.trans*.
  show *transitive A r'* $\Longrightarrow$ *transitive A r*
    apply (*intro transitive.intro*)
    apply (*unfold r*)
    using *transitive.trans*.
qed

lemma *tranclp-transitive*: *transitive A* (*tranclp r*)
  using *tranclp-trans* **by** *unfold-locales*

locale *symmetric* = *related-set A* (∼) **for** *A* **and** *equiv* (**infix** ∼ *50*) +
  assumes *sym*[*sym*]: *x* ∼ *y* ⟹ *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *y* ∼ *x*
begin

lemma *sym-iff*: *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *x* ∼ *y* ⟷ *y* ∼ *x*
  by (*auto dest*: *sym*)

lemma *Restrp-symmetric*: *symmetric UNIV* ((∼)↾*A*)
  apply *unfold-locales* **by** (*auto simp*: *sym-iff*)

lemma *symmetric-subset*: *B* ⊆ *A* ⟹ *symmetric B* (∼)
  apply *unfold-locales* **by** (*auto dest*: *sym*)

end

declare *symmetric.intro*[*intro*]

lemma *symmetric-cong*:
  (⋀*a b*. *a* ∈ *A* ⟹ *b* ∈ *A* ⟹ *r a b* ⟷ *r*′ *a b*) ⟹ *symmetric A r* ⟷ *symmetric A r*′
  by (*auto simp*: *symmetric-def*)

global-interpretation *sympartp*: *symmetric UNIV sympartp r*
  rewrites ⋀*r*. *r* ↾ *UNIV* ≡ *r*
    and ⋀*x*. *x* ∈ *UNIV* ≡ *True*
    and ⋀*P1*. (*True* ⟹ *P1*) ≡ *Trueprop P1*
    and ⋀*P1 P2*. (*True* ⟹ *PROP P1* ⟹ *PROP P2*) ≡ (*PROP P1* ⟹ *PROP P2*)
  by *auto*

lemma *sympartp-symmetric*: *symmetric A* (*sympartp r*) **by** *auto*

locale *antisymmetric* = *related-set* +
  assumes *antisym*: *x* ⊑ *y* ⟹ *y* ⊑ *x* ⟹ *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *x* = *y*
begin

interpretation *less-eq-notations***.**

lemma *sym-iff-eq-refl*: *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *x* ∼ *y* ⟷ *x* = *y* ∧ *y* ⊑ *y* **by** (*auto dest*: *antisym*)

lemma *equiv-iff-eq*[*simp*]: *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *x* ≃ *y* ⟷ *x* = *y* **by** (*auto dest*: *antisym elim*: *equivpartpE*)

lemma *extreme-unique*: *X* ⊆ *A* ⟹ *extreme X* (⊑) *x* ⟹ *extreme X* (⊑) *y* ⟷ *x* = *y*
  by (*elim extremeE*, *auto dest*!: *antisym*[*OF* - - *subsetD*])

lemma *ex-extreme-iff-ex1*:
$X \subseteq A \Longrightarrow Ex\ (extreme\ X\ (\sqsubseteq)) \longleftrightarrow Ex1\ (extreme\ X\ (\sqsubseteq))$ **by** (*auto simp*: *extreme-unique*)

lemma *ex-extreme-iff-the*:
$X \subseteq A \Longrightarrow Ex\ (extreme\ X\ (\sqsubseteq)) \longleftrightarrow extreme\ X\ (\sqsubseteq)\ (The\ (extreme\ X\ (\sqsubseteq)))$
apply (*rule iffI*)
apply (*rule theI'*)
using *extreme-unique* **by** *auto*

lemma *Restrp-antisymmetric*: *antisymmetric UNIV* $((\sqsubseteq)\restriction A)$
apply *unfold-locales*
by (*auto dest*: *antisym*)

lemma *antisymmetric-subset*: $B \subseteq A \Longrightarrow antisymmetric\ B\ (\sqsubseteq)$
apply *unfold-locales* **using** *antisym* **by** *auto*

end

declare *antisymmetric.intro*[*intro*]

lemma *antisymmetric-cong*:
$(\bigwedge a\ b.\ a \in A \Longrightarrow b \in A \Longrightarrow r\ a\ b \longleftrightarrow r'\ a\ b) \Longrightarrow antisymmetric\ A\ r \longleftrightarrow antisymmetric\ A\ r'$
by (*auto simp*: *antisymmetric-def*)

lemma *antisymmetric-union*:
fixes *less-eq* (**infix** $\sqsubseteq$ *50*)
assumes *A*: *antisymmetric A* $(\sqsubseteq)$ **and** *B*: *antisymmetric B* $(\sqsubseteq)$
and *AB*: $\forall a \in A.\ \forall b \in B.\ a \sqsubseteq b \longrightarrow b \sqsubseteq a \longrightarrow a = b$
shows *antisymmetric* $(A \cup B)$ $(\sqsubseteq)$
proof−
interpret *A*: *antisymmetric A* $(\sqsubseteq)$ **using** *A*.
interpret *B*: *antisymmetric B* $(\sqsubseteq)$ **using** *B*.
show *?thesis* **by** (*auto dest*: *AB*[*rule-format*] *A.antisym B.antisym*)
qed

The following notion is new, generalizing antisymmetry and transitivity.

locale *semiattractive* = *related-set* +
assumes *attract*: $x \sqsubseteq y \Longrightarrow y \sqsubseteq x \Longrightarrow y \sqsubseteq z \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow z \in A \Longrightarrow x \sqsubseteq z$
begin

interpretation *less-eq-notations*.

lemma *equiv-order-trans*[*trans*]:
assumes *xy*: $x \simeq y$ **and** *yz*: $y \sqsubseteq z$ **and** *x*: $x \in A$ **and** *y*: $y \in A$ **and** *z*: $z \in A$
shows $x \sqsubseteq z$
using *attract*[*OF - - - x y z*] *xy yz* **by** (*auto elim*: *equivpartpE*)

lemma *equiv-transitive*: *transitive A* ($\simeq$)
proof *unfold-locales*
fix *x y z*
assume *x*: $x \in A$ **and** *y*: $y \in A$ **and** *z*: $z \in A$ **and** *xy*: $x \simeq y$ **and** *yz*: $y \simeq z$
show $x \simeq z$
using *equiv-order-trans*[*OF xy - x y z*] *attract*[*OF - - - z y x*] *xy yz* **by** (*auto simp*:*equivpartp-def*)
qed

lemma *sym-order-trans*[*trans*]:
assumes *xy*: $x \sim y$ **and** *yz*: $y \sqsubseteq z$ **and** *x*: $x \in A$ **and** *y*: $y \in A$ **and** *z*: $z \in A$
shows $x \sqsubseteq z$
using *attract*[*OF - - - x y z*] *xy yz* **by** *auto*

interpretation *sym*: *transitive A* ($\sim$)
proof *unfold-locales*
fix *x y z*
assume *x*: $x \in A$ **and** *y*: $y \in A$ **and** *z*: $z \in A$ **and** *xy*: $x \sim y$ **and** *yz*: $y \sim z$
show $x \sim z$
using *sym-order-trans*[*OF xy - x y z*] *attract*[*OF - - - z y x*] *xy yz* **by** *auto*
qed

lemmas *sym-transitive* = *sym.transitive-axioms*

lemma *extreme-bound-quasi-const*:
assumes *C*: $C \subseteq A$ **and** *x*: $x \in A$ **and** *C0*: $C \neq \{\}$ **and** *const*: $\forall\, y \in C.\ y \sim x$
shows *extreme-bound A* ($\sqsubseteq$) *C x*
proof (*intro extreme-boundI x*)
from *C0* **obtain** *c* **where** *cC*: $c \in C$ **by** *auto*
with *C* **have** *c*: $c \in A$ **by** *auto*
from *cC const* **have** *cx*: $c \sim x$ **by** *auto*
fix *b* **assume** *b*: $b \in A$ **and** *bound C* ($\sqsubseteq$) *b*
with *cC* **have** *cb*: $c \sqsubseteq b$ **by** *auto*
from *attract*[*OF - - cb x c b*] *cx* **show** $x \sqsubseteq b$ **by** *auto*
next
fix *c* **assume** $c \in C$
with *const* **show** $c \sqsubseteq x$ **by** *auto*
qed

lemma *extreme-bound-quasi-const-iff*:
assumes *C*: $C \subseteq A$ **and** *x*: $x \in A$ **and** *y*: $y \in A$ **and** *C0*: $C \neq \{\}$ **and** *const*: $\forall\, z \in C.\ z \sim x$
shows *extreme-bound A* ($\sqsubseteq$) *C y* $\longleftrightarrow$ $x \sim y$
proof (*intro iffI*)
assume *y*: *extreme-bound A* ($\sqsubseteq$) *C y*
note x = *extreme-bound-quasi-const*[*OF C x C0 const*]
from *extremes-equiv*[*OF y x*]
show $x \sim y$ **by** *auto*

next
assume *xy*: $x \sim y$
with *const C sym.trans*[*OF - xy - x y*] **have** *Cy*: $\forall z \in C.\ z \sim y$ **by** *auto*
show *extreme-bound A* $(\sqsubseteq)$ *C y*
using *extreme-bound-quasi-const*[*OF C y C0 Cy*]**.**
qed

lemma *Restrp-semiattractive*: *semiattractive UNIV* $((\sqsubseteq)\restriction A)$
apply *unfold-locales*
by (*auto dest*: *attract*)

lemma *semiattractive-subset*: $B \subseteq A \Longrightarrow$ *semiattractive B* $(\sqsubseteq)$
apply *unfold-locales* **using** *attract* **by** *blast*

end

lemma *semiattractive-cong*:
assumes *r*: $\bigwedge a\ b.\ a \in A \Longrightarrow b \in A \Longrightarrow r\ a\ b \longleftrightarrow r'\ a\ b$
shows *semiattractive A r* $\longleftrightarrow$ *semiattractive A r'* (**is** *?l* $\longleftrightarrow$ *?r*)
proof
show *?l* $\Longrightarrow$ *?r*
apply (*intro semiattractive.intro*)
apply (*unfold r*[*symmetric*])
using *semiattractive.attract***.**
show *?r* $\Longrightarrow$ *?l*
apply (*intro semiattractive.intro*)
apply (*unfold r*)
using *semiattractive.attract***.**
qed

locale *attractive* = *semiattractive* +
assumes *semiattractive A* $(\sqsubseteq)^-$
begin

interpretation *less-eq-notations***.**

sublocale *dual*: *semiattractive A* $(\sqsubseteq)^-$
rewrites $\bigwedge r.$ *sympartp* $(r \restriction A) \equiv$ *sympartp* $r \restriction A$
and $\bigwedge r.$ *sympartp* (*sympartp r*) $\equiv$ *sympartp r*
and *sympartp* $((\sqsubseteq) \restriction A)^- \equiv (\sim) \restriction A$
and *sympartp* $(\sqsubseteq)^- \equiv (\sim)$
and *equivpartp* $(\sqsubseteq)^- \equiv (\simeq)$
using *attractive-axioms*[*unfolded attractive-def*]
by (*auto intro*!: *ext simp*: *attractive-axioms-def atomize-eq equivpartp-def*)

lemma *order-equiv-trans*[*trans*]:
assumes *xy*: $x \sqsubseteq y$ **and** *yz*: $y \simeq z$ **and** *x*: $x \in A$ **and** *y*: $y \in A$ **and** *z*: $z \in A$
shows $x \sqsubseteq z$
using *dual.attract*[*OF - - - z y x*] *xy yz* **by** *auto*

lemma *order-sym-trans*[*trans*]:
assumes *xy*: $x \sqsubseteq y$ **and** *yz*: $y \sim z$ **and** *x*: $x \in A$ **and** *y*: $y \in A$ **and** *z*: $z \in A$
shows $x \sqsubseteq z$
using *dual.attract*[*OF - - - z y x*] *xy yz* **by** *auto*

interpretation *Restrp*: *semiattractive UNIV* $(\sqsubseteq)\restriction A$ **using** *Restrp-semiattractive*.
interpretation *dual.Restrp*: *semiattractive UNIV* $(\sqsubseteq)^- \restriction A$ **using** *dual.Restrp-semiattractive*.

lemma *Restrp-attractive*: *attractive UNIV* $((\sqsubseteq)\restriction A)$
apply *unfold-locales*
using *dual.Restrp.attract* **by** *auto*

lemma *attractive-subset*: $B \subseteq A \Longrightarrow$ *attractive* B $(\sqsubseteq)$
apply (*intro attractive.intro attractive-axioms.intro*)
using *semiattractive-subset dual.semiattractive-subset* **by** *auto*

end

lemma *attractive-cong*:
assumes *r*: $\bigwedge a\ b.\ a \in A \Longrightarrow b \in A \Longrightarrow r\ a\ b \longleftrightarrow r'\ a\ b$
shows *attractive* $A\ r \longleftrightarrow$ *attractive* $A\ r'$
by (*simp add*: *attractive-def attractive-axioms-def r cong*: *semiattractive-cong*)

context *antisymmetric* **begin**

sublocale *attractive*
apply *unfold-locales* **by** (*auto dest*: *antisym*)

end

context *transitive* **begin**

sublocale *attractive*
rewrites $\bigwedge r.$ *sympartp* $(r \restriction A) \equiv$ *sympartp* $r \restriction A$
and $\bigwedge r.$ *sympartp* (*sympartp* r) $\equiv$ *sympartp* r
and *sympartp* $(\sqsubseteq)^- \equiv$ *sympartp* $(\sqsubseteq)$
and (*sympartp* $(\sqsubseteq)$)$^- \equiv$ *sympartp* $(\sqsubseteq)$
and (*sympartp* $(\sqsubseteq) \restriction A)^- \equiv$ *sympartp* $(\sqsubseteq) \restriction A$
and *asympartp* (*asympartp* $(\sqsubseteq)$) $=$ *asympartp* $(\sqsubseteq)$
and *asympartp* (*sympartp* $(\sqsubseteq)$) $=$ *bot*
and *asympartp* $(\sqsubseteq) \restriction A =$ *asympartp* $((\sqsubseteq) \restriction A)$
apply *unfold-locales*
by (*auto intro*!:*ext dest*: *trans simp*: *atomize-eq*)

end

2.3 Combined Properties

Some combinations of the above basic properties are given names.

locale *asymmetric* = *related-set A* (⊏) **for** *A* **and** *less* (**infix** ⊏ *50*) +
  assumes *asym*: $x \sqsubset y \Longrightarrow y \sqsubset x \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow$ *False*
begin

sublocale *irreflexive*
  apply *unfold-locales* **by** (*auto dest*: *asym*)

lemma *antisymmetric-axioms*: *antisymmetric A* (⊏)
  apply *unfold-locales* **by** (*auto dest*: *asym*)

lemma *Restrp-asymmetric*: *asymmetric UNIV* ((⊏)↾*A*)
  apply *unfold-locales*
  by (*auto dest*:*asym*)

lemma *asymmetric-subset*: $B \subseteq A \Longrightarrow$ *asymmetric B* (⊏)
  apply *unfold-locales* **using** *asym* **by** *auto*

end

lemma *asymmetric-iff-irreflexive-antisymmetric*:
  fixes *less* (**infix** ⊏ *50*)
  shows *asymmetric A* (⊏) ⟷ *irreflexive A* (⊏) ∧ *antisymmetric A* (⊏) (**is** *?l* ⟷ *?r*)
proof
  assume *?l*
  then interpret *asymmetric*.
  show *?r* **by** (*auto dest*: *asym*)
next
  assume *?r*
  then interpret *irreflexive* + *antisymmetric A* (⊏) **by** *auto*
  show *?l* **by** (*auto intro*!:*asymmetric.intro dest*: *antisym irrefl*)
qed

lemma *asymmetric-cong*:
  assumes *r*: $\bigwedge a\ b.\ a \in A \Longrightarrow b \in A \Longrightarrow r\ a\ b \longleftrightarrow r'\ a\ b$
  shows *asymmetric A r* ⟷ *asymmetric A r′*
  by (*simp add*: *asymmetric-iff-irreflexive-antisymmetric r cong*: *irreflexive-cong antisymmetric-cong*)

locale *quasi-ordered-set* = *reflexive* + *transitive*
begin

lemma *quasi-ordered-subset*: $B \subseteq A \Longrightarrow$ *quasi-ordered-set B* (⊑)
  apply *intro-locales*
  using *reflexive-subset transitive-subset* **by** *auto*

end

lemma *quasi-ordered-set-cong*:
assumes *r*: $\bigwedge a\ b.\ a \in A \Longrightarrow b \in A \Longrightarrow r\ a\ b \longleftrightarrow r'\ a\ b$
shows *quasi-ordered-set A r* $\longleftrightarrow$ *quasi-ordered-set A r'*
by (*simp add*: *quasi-ordered-set-def r cong*: *reflexive-cong transitive-cong*)

locale *near-ordered-set* = *antisymmetric* + *transitive*
begin

interpretation *Restrp*: *antisymmetric UNIV* $(\sqsubseteq)\restriction A$ **using** *Restrp-antisymmetric*.
interpretation *Restrp*: *transitive UNIV* $(\sqsubseteq)\restriction A$ **using** *Restrp-transitive*.

lemma *Restrp-near-order*: *near-ordered-set UNIV* $((\sqsubseteq)\restriction A)$..

lemma *near-ordered-subset*: $B \subseteq A \Longrightarrow$ *near-ordered-set B* $(\sqsubseteq)$
apply *intro-locales*
using *antisymmetric-subset transitive-subset* **by** *auto*

end

lemma *near-ordered-set-cong*:
assumes *r*: $\bigwedge a\ b.\ a \in A \Longrightarrow b \in A \Longrightarrow r\ a\ b \longleftrightarrow r'\ a\ b$
shows *near-ordered-set A r* $\longleftrightarrow$ *near-ordered-set A r'*
by (*simp add*: *near-ordered-set-def r cong*: *antisymmetric-cong transitive-cong*)

locale *pseudo-ordered-set* = *reflexive* + *antisymmetric*
begin

interpretation *less-eq-notations*.

lemma *sym-eq*[*simp*]: $x \in A \Longrightarrow y \in A \Longrightarrow x \sim y \longleftrightarrow x = y$
by (*auto simp*: *refl dest*: *antisym*)

lemma *extreme-bound-singleton-eq*[*simp*]: $x \in A \Longrightarrow$ *extreme-bound A* $(\sqsubseteq)$ $\{x\}$ $y \longleftrightarrow x = y$
by (*auto intro*!: *antisym*)

lemma *eq-iff*: $x \in A \Longrightarrow y \in A \Longrightarrow x = y \longleftrightarrow x \sqsubseteq y \wedge y \sqsubseteq x$ **by** (*auto dest*: *antisym simp*:*refl*)

lemma *extreme-order-iff-eq*: $e \in A \Longrightarrow$ *extreme* $\{x \in A.\ x \sqsubseteq e\}$ $(\sqsubseteq)$ $s \longleftrightarrow e = s$
by (*auto intro*!: *antisym*)

lemma *pseudo-ordered-subset*: $B \subseteq A \Longrightarrow$ *pseudo-ordered-set B* $(\sqsubseteq)$
apply *intro-locales*
using *reflexive-subset antisymmetric-subset* **by** *auto*

end

lemma *pseudo-ordered-set-cong*:
assumes *r*: $\bigwedge a\ b.\ a \in A \Longrightarrow b \in A \Longrightarrow r\ a\ b \longleftrightarrow r'\ a\ b$
shows *pseudo-ordered-set A r* $\longleftrightarrow$ *pseudo-ordered-set A r'*
by (*simp add*: *pseudo-ordered-set-def r cong*: *reflexive-cong antisymmetric-cong*)

locale *partially-ordered-set* = *reflexive* + *antisymmetric* + *transitive*
begin

sublocale *pseudo-ordered-set* + *quasi-ordered-set* + *near-ordered-set* **..**

lemma *partially-ordered-subset*: $B \subseteq A \Longrightarrow$ *partially-ordered-set B* $(\sqsubseteq)$
apply *intro-locales*
using *reflexive-subset transitive-subset antisymmetric-subset* **by** *auto*

end

lemma *partially-ordered-set-cong*:
assumes *r*: $\bigwedge a\ b.\ a \in A \Longrightarrow b \in A \Longrightarrow r\ a\ b \longleftrightarrow r'\ a\ b$
shows *partially-ordered-set A r* $\longleftrightarrow$ *partially-ordered-set A r'*
by (*simp add*: *partially-ordered-set-def r cong*: *reflexive-cong antisymmetric-cong transitive-cong*)

locale *strict-ordered-set* = *irreflexive* + *transitive A* $(\sqsubset)$
begin

sublocale *asymmetric*
proof
fix *x y*
assume *x*: $x \in A$ **and** *y*: $y \in A$
assume *xy*: $x \sqsubset y$
also assume *yx*: $y \sqsubset x$
finally have $x \sqsubset x$ **using** *x y* **by** *auto*
with *x* **show** *False* **by** *auto*
qed

lemma *near-ordered-set-axioms*: *near-ordered-set A* $(\sqsubset)$
using *antisymmetric-axioms* **by** *intro-locales*

interpretation *Restrp*: *asymmetric UNIV* $(\sqsubset)\restriction A$ **using** *Restrp-asymmetric*.
interpretation *Restrp*: *transitive UNIV* $(\sqsubset)\restriction A$ **using** *Restrp-transitive*.

lemma *Restrp-strict-order*: *strict-ordered-set UNIV* $((\sqsubset)\restriction A)$**..**

lemma *strict-ordered-subset*: $B \subseteq A \Longrightarrow$ *strict-ordered-set B* $(\sqsubset)$
apply *intro-locales*
using *irreflexive-subset transitive-subset* **by** *auto*

end

lemma *strict-ordered-set-cong*:
assumes *r*: $\bigwedge a\ b.\ a \in A \Longrightarrow b \in A \Longrightarrow r\ a\ b \longleftrightarrow r'\ a\ b$
shows *strict-ordered-set A r* $\longleftrightarrow$ *strict-ordered-set A r'*
by (*simp add*: *strict-ordered-set-def r cong*: *irreflexive-cong transitive-cong*)

locale *tolerance* = *symmetric* + *reflexive A* ($\sim$)
begin

lemma *tolerance-subset*: $B \subseteq A \Longrightarrow$ *tolerance B* ($\sim$)
apply *intro-locales*
using *symmetric-subset reflexive-subset* **by** *auto*

end

lemma *tolerance-cong*:
assumes *r*: $\bigwedge a\ b.\ a \in A \Longrightarrow b \in A \Longrightarrow r\ a\ b \longleftrightarrow r'\ a\ b$
shows *tolerance A r* $\longleftrightarrow$ *tolerance A r'*
by (*simp add*: *tolerance-def r cong*: *reflexive-cong symmetric-cong*)

global-interpretation *equiv*: *tolerance UNIV equivpartp r*
rewrites $\bigwedge r.\ r \restriction UNIV \equiv r$
and $\bigwedge x.\ x \in UNIV \equiv True$
and $\bigwedge P1.\ (True \Longrightarrow P1) \equiv Trueprop\ P1$
and $\bigwedge P1\ P2.\ (True \Longrightarrow PROP\ P1 \Longrightarrow PROP\ P2) \equiv (PROP\ P1 \Longrightarrow PROP\ P2)$
by *unfold-locales* (*auto simp*:*equivpartp-def*)

locale *partial-equivalence* = *symmetric* +
assumes *transitive A* ($\sim$)
begin

sublocale *transitive A* ($\sim$)
rewrites *sympartp* ($\sim$)$\restriction A \equiv$ ($\sim$)$\restriction A$
and *sympartp* (($\sim$)$\restriction A$) $\equiv$ ($\sim$)$\restriction A$
using *partial-equivalence-axioms*
unfolding *partial-equivalence-axioms-def partial-equivalence-def*
by (*auto simp*: *atomize-eq sym intro*!:*ext*)

lemma *partial-equivalence-subset*: $B \subseteq A \Longrightarrow$ *partial-equivalence B* ($\sim$)
apply (*intro partial-equivalence.intro partial-equivalence-axioms.intro*)
using *symmetric-subset transitive-subset* **by** *auto*

end

lemma *partial-equivalence-cong*:
assumes *r*: $\bigwedge a\ b.\ a \in A \Longrightarrow b \in A \Longrightarrow r\ a\ b \longleftrightarrow r'\ a\ b$
shows *partial-equivalence A r* $\longleftrightarrow$ *partial-equivalence A r'*
by (*simp add*: *partial-equivalence-def partial-equivalence-axioms-def r*

cong: *transitive-cong symmetric-cong*)

locale *equivalence* = *symmetric* + *reflexive* A ($\sim$) + *transitive* A ($\sim$)
begin

sublocale *tolerance* + *partial-equivalence* + *quasi-ordered-set* A ($\sim$)**..**

lemma *equivalence-subset*: $B \subseteq A \Longrightarrow$ *equivalence* B ($\sim$)
apply (*intro equivalence.intro*)
using *symmetric-subset transitive-subset* **by** *auto*

end

lemma *equivalence-cong*:
assumes *r*: $\bigwedge a\ b.\ a \in A \Longrightarrow b \in A \Longrightarrow r\ a\ b \longleftrightarrow r'\ a\ b$
shows *equivalence* $A\ r \longleftrightarrow$ *equivalence* $A\ r'$
by (*simp add*: *equivalence-def r cong*: *reflexive-cong transitive-cong symmetric-cong*)

Some combinations lead to uninteresting relations.

context
fixes $r :: 'a \Rightarrow 'a \Rightarrow bool$ (**infix** $\bowtie$ *50*)
begin

proposition *reflexive-irreflexive-is-empty*:
assumes *r*: *reflexive* A ($\bowtie$) **and** *ir*: *irreflexive* A ($\bowtie$)
shows $A = \{\}$
proof (*rule ccontr*)
interpret *irreflexive* A ($\bowtie$) **using** *ir*.
interpret *reflexive* A ($\bowtie$) **using** *r*.
assume $A \neq \{\}$
then obtain *a* **where** *a*: $a \in A$ **by** *auto*
from *a refl* **have** $a \bowtie a$ **by** *auto*
with *irrefl a* **show** *False* **by** *auto*
qed

proposition *symmetric-antisymmetric-imp-eq*:
assumes *s*: *symmetric* A ($\bowtie$) **and** *as*: *antisymmetric* A ($\bowtie$)
shows $(\bowtie)\restriction A \leq (=)$
proof−
interpret *symmetric* A ($\bowtie$) + *antisymmetric* A ($\bowtie$) **using** *assms* **by** *auto*
show *?thesis* **using** *antisym* **by** (*auto dest*: *sym*)
qed

proposition *nontolerance*:
shows *irreflexive* A ($\bowtie$) $\wedge$ *symmetric* A ($\bowtie$) $\longleftrightarrow$ *tolerance* A ($\lambda x\ y.\ \neg\ x \bowtie y$)
proof (*intro iffI conjI*, *elim conjE*)
assume *irreflexive* A ($\bowtie$) **and** *symmetric* A ($\bowtie$)
then interpret *irreflexive* A ($\bowtie$) + *symmetric* A ($\bowtie$)**.**

show *tolerance A* (λ*x y*. ¬ *x* ⋈ *y*) **by** (*unfold-locales*, *auto dest*: *sym irefl*)
next
assume *tolerance A* (λ*x y*. ¬ *x* ⋈ *y*)
then interpret *tolerance A* λ*x y*. ¬ *x* ⋈ *y***.**
show *irreflexive A* (⋈) **by** (*auto simp*: *eq-implies*)
show *symmetric A* (⋈) **using** *sym* **by** *auto*
qed

proposition *irreflexive-transitive-symmetric-is-empty*:
assumes *irr*: *irreflexive A* (⋈) **and** *tr*: *transitive A* (⋈) **and** *sym*: *symmetric A* (⋈)
shows (⋈)↾*A* = *bot*
proof (*intro ext*, *unfold bot-fun-def bot-bool-def eq-False*, *rule notI*, *erule RestrpE*)
interpret *strict-ordered-set A* (⋈) **using** *assms* **by** (*unfold strict-ordered-set-def*, *auto*)
interpret *symmetric A* (⋈) **using** *assms* **by** *auto*
fix *x y* **assume** *x*: *x* ∈ *A* **and** *y*: *y* ∈ *A*
assume *xy*: *x* ⋈ *y*
also note *sym*[*OF xy x y*]
finally have *x* ⋈ *x* **using** *x y* **by** *auto*
with *x* **show** *False* **by** *auto*
qed

end

2.4 Totality

locale *semiconnex* = *related-set A* (⊏) **for** *A* **and** *less* (**infix** ⊏ *50*) +
assumes *semiconnex*: *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *x* ⊏ *y* ∨ *x* = *y* ∨ *y* ⊏ *x*
begin

lemma *cases*[*consumes 2*, *case-names less eq greater*]:
assumes *x* ∈ *A* **and** *y* ∈ *A* **and** *x* ⊏ *y* ⟹ *P* **and** *x* = *y* ⟹ *P* **and** *y* ⊏ *x* ⟹ *P*
shows *P* **using** *semiconnex assms* **by** *auto*

lemma *neqE*:
assumes *x* ∈ *A* **and** *y* ∈ *A*
shows *x* ≠ *y* ⟹ (*x* ⊏ *y* ⟹ *P*) ⟹ (*y* ⊏ *x* ⟹ *P*) ⟹ *P*
by (*cases rule*: *cases*[*OF assms*], *auto*)

lemma *semiconnex-subset*: *B* ⊆ *A* ⟹ *semiconnex B* (⊏)
apply (*intro semiconnex.intro*)
using *semiconnex* **by** *auto*

end

declare *semiconnex.intro*[*intro*]

Totality is negated antisymmetry [14, Proposition 2.2.4].

proposition *semiconnex-iff-neg-antisymmetric*:
fixes *less* (**infix** $\sqsubset$ *50*)
shows *semiconnex A* ($\sqsubset$) $\longleftrightarrow$ *antisymmetric A* ($\lambda x\ y.\ \neg\ x \sqsubset y$) (**is** *?l* $\longleftrightarrow$ *?r*)
proof (*intro iffI semiconnex.intro antisymmetric.intro*)
assume *?l*
then interpret *semiconnex*.
fix *x y*
assume $x \in A$ $y \in A$ $\neg\ x \sqsubset y$ **and** $\neg\ y \sqsubset x$
then show $x = y$ **by** (*cases rule*: *cases*, *auto*)
next
assume *?r*
then interpret *neg*: *antisymmetric A* ($\lambda x\ y.\ \neg\ x \sqsubset y$).
fix *x y*
show $x \in A \Longrightarrow y \in A \Longrightarrow x \sqsubset y \vee x = y \vee y \sqsubset x$ **using** *neg.antisym* **by** *auto*
qed

lemma *semiconnex-cong*:
assumes *r*: $\bigwedge a\ b.\ a \in A \Longrightarrow b \in A \Longrightarrow r\ a\ b \longleftrightarrow r'\ a\ b$
shows *semiconnex A r* $\longleftrightarrow$ *semiconnex A r'*
by (*simp add*: *semiconnex-iff-neg-antisymmetric r cong*: *antisymmetric-cong*)

locale *semiconnex-irreflexive* = *semiconnex* + *irreflexive*
begin

lemma *neq-iff*: $x \in A \Longrightarrow y \in A \Longrightarrow x \neq y \longleftrightarrow x \sqsubset y \vee y \sqsubset x$ **by** (*auto elim*:*neqE dest*: *irrefl*)

lemma *semiconnex-irreflexive-subset*: $B \subseteq A \Longrightarrow$ *semiconnex-irreflexive B* ($\sqsubset$)
apply (*intro semiconnex-irreflexive.intro*)
using *semiconnex-subset irreflexive-subset* **by** *auto*

end

lemma *semiconnex-irreflexive-cong*:
assumes *r*: $\bigwedge a\ b.\ a \in A \Longrightarrow b \in A \Longrightarrow r\ a\ b \longleftrightarrow r'\ a\ b$
shows *semiconnex-irreflexive A r* $\longleftrightarrow$ *semiconnex-irreflexive A r'*
by (*simp add*: *semiconnex-irreflexive-def r cong*: *semiconnex-cong irreflexive-cong*)

locale *connex* = *related-set* +
assumes *comparable*: $x \in A \Longrightarrow y \in A \Longrightarrow x \sqsubseteq y \vee y \sqsubseteq x$
begin

interpretation *less-eq-notations*.

sublocale *reflexive* **apply** *unfold-locales* **using** *comparable* **by** *auto*

lemma *comparable-cases*[*consumes 2*, *case-names le ge*]:
assumes $x \in A$ **and** $y \in A$ **and** $x \sqsubseteq y \Longrightarrow P$ **and** $y \sqsubseteq x \Longrightarrow P$ **shows** *P*
using *assms comparable* **by** *auto*

lemma *comparable-three-cases*[*consumes 2*, *case-names less eq greater*]:
assumes $x \in A$ **and** $y \in A$ **and** $x \sqsubset y \Longrightarrow P$ **and** $x \sim y \Longrightarrow P$ **and** $y \sqsubset x \Longrightarrow P$ **shows** P
using *assms comparable* **by** *auto*

lemma
assumes *x*: $x \in A$ **and** *y*: $y \in A$
shows *not-iff-asym*: $\neg x \sqsubseteq y \longleftrightarrow y \sqsubset x$
and *not-asym-iff*[*simp*]: $\neg x \sqsubset y \longleftrightarrow y \sqsubseteq x$
using *comparable*[*OF x y*] **by** *auto*

lemma *connex-subset*: $B \subseteq A \Longrightarrow$ *connex B* ($\sqsubseteq$)
by (*intro connex.intro comparable*, *auto*)

end

lemmas *connexE* = *connex.comparable-cases*

lemmas *connexI*[*intro*] = *connex.intro*

context
fixes *less-eq* :: $'a \Rightarrow 'a \Rightarrow$ *bool* (**infix** $\sqsubseteq$ *50*)
begin

lemma *connex-iff-semiconnex-reflexive*: *connex A* ($\sqsubseteq$) $\longleftrightarrow$ *semiconnex A* ($\sqsubseteq$) $\wedge$ *reflexive A* ($\sqsubseteq$)
(**is** *?c* $\longleftrightarrow$ *?t* $\wedge$ *?r*)
proof (*intro iffI conjI*; (*elim conjE*)*?*)
assume *?c* **then interpret** *connex*.
show *?t* **apply** *unfold-locales* **using** *comparable* **by** *auto*
show *?r* **by** *unfold-locales*
next
assume *?t* **then interpret** *semiconnex A* ($\sqsubseteq$).
assume *?r* **then interpret** *reflexive*.
from *semiconnex* **show** *?c* **by** *auto*
qed

lemma *chain-connect*: *Complete-Partial-Order.chain r A* $\equiv$ *connex A r*
by (*auto intro*!: *ext simp*: *atomize-eq connex-def Complete-Partial-Order.chain-def*)

lemma *connex-union*:
assumes *connex X* ($\sqsubseteq$) **and** *connex Y* ($\sqsubseteq$) **and** $\forall x \in X.\ \forall y \in Y.\ x \sqsubseteq y \vee y \sqsubseteq x$
shows *connex* ($X \cup Y$) ($\sqsubseteq$)
using *assms* **by** (*auto simp*: *connex-def*)

end

lemma *connex-cong*:
assumes *r*: $\bigwedge a\ b.\ a \in A \Longrightarrow b \in A \Longrightarrow r\ a\ b \longleftrightarrow r'\ a\ b$
shows *connex A r* $\longleftrightarrow$ *connex A r'*
by (*simp add*: *connex-iff-semiconnex-reflexive r cong*: *semiconnex-cong reflexive-cong*)

locale *total-pseudo-ordered-set* = *connex* + *antisymmetric*
begin

sublocale *pseudo-ordered-set* **..**

lemma *not-weak-iff*:
assumes *x*: $x \in A$ **and** *y*: $y \in A$ **shows** $\neg\ y \sqsubseteq x \longleftrightarrow x \sqsubseteq y \wedge x \neq y$
using *x y* **by** (*cases rule*: *comparable-cases*, *auto intro*:*antisym*)

lemma *total-pseudo-ordered-subset*: $B \subseteq A \Longrightarrow$ *total-pseudo-ordered-set B* ($\sqsubseteq$)
apply (*intro-locales*)
using *antisymmetric-subset connex-subset* **by** *auto*

end

lemma *total-pseudo-ordered-set-cong*:
assumes *r*: $\bigwedge a\ b.\ a \in A \Longrightarrow b \in A \Longrightarrow r\ a\ b \longleftrightarrow r'\ a\ b$
shows *total-pseudo-ordered-set A r* $\longleftrightarrow$ *total-pseudo-ordered-set A r'*
by (*simp add*: *total-pseudo-ordered-set-def r cong*: *connex-cong antisymmetric-cong*)

locale *total-quasi-ordered-set* = *connex* + *transitive*
begin

sublocale *quasi-ordered-set* **..**

lemma *total-quasi-ordered-subset*: $B \subseteq A \Longrightarrow$ *total-quasi-ordered-set B* ($\sqsubseteq$)
using *transitive-subset connex-subset* **by** *intro-locales*

end

lemma *total-quasi-ordered-set-cong*:
assumes *r*: $\bigwedge a\ b.\ a \in A \Longrightarrow b \in A \Longrightarrow r\ a\ b \longleftrightarrow r'\ a\ b$
shows *total-quasi-ordered-set A r* $\longleftrightarrow$ *total-quasi-ordered-set A r'*
by (*simp add*: *total-quasi-ordered-set-def r cong*: *connex-cong transitive-cong*)

locale *total-ordered-set* = *total-quasi-ordered-set* + *antisymmetric*
begin

sublocale *partially-ordered-set* + *total-pseudo-ordered-set* **..**

lemma *total-ordered-subset*: $B \subseteq A \Longrightarrow$ *total-ordered-set B* ($\sqsubseteq$)
using *total-quasi-ordered-subset antisymmetric-subset* **by** (*intro total-ordered-set.intro*)

end

lemma *total-ordered-set-cong*:
assumes *r*: $\bigwedge a\ b.\ a \in A \Longrightarrow b \in A \Longrightarrow r\ a\ b \longleftrightarrow r'\ a\ b$
shows *total-ordered-set A r* $\longleftrightarrow$ *total-ordered-set A r'*
by (*simp add*: *total-ordered-set-def r cong*: *total-quasi-ordered-set-cong antisymmetric-cong*)

2.5 Well-Foundedness

locale *well-founded* = *related-set A* ($\sqsubset$) **for** *A* **and** *less* (**infix** $\sqsubset$ *50*) +
assumes *induct*[*consumes 1*, *case-names less*, *induct set*]:
$a \in A \Longrightarrow (\bigwedge x.\ x \in A \Longrightarrow (\bigwedge y.\ y \in A \Longrightarrow y \sqsubset x \Longrightarrow P\ y) \Longrightarrow P\ x) \Longrightarrow P\ a$
begin

sublocale *asymmetric*
proof (*intro asymmetric.intro notI*)
fix *x y*
assume *xA*: $x \in A$
then show $y \in A \Longrightarrow x \sqsubset y \Longrightarrow y \sqsubset x \Longrightarrow$ *False*
by (*induct arbitrary*: *y rule*: *induct*, *auto*)
qed

lemma *prefixed-Imagep-imp-empty*:
assumes *a*: $X \subseteq ((\sqsubset)\ ``` X) \cap A$ **shows** $X = \{\}$
proof −
from *a* **have** *XA*: $X \subseteq A$ **by** *auto*
have $x \in A \Longrightarrow x \notin X$ **for** *x*
proof (*induct x rule*: *induct*)
case (*less x*)
with *a* **show** *?case* **by** (*auto simp*: *Imagep-def*)
qed
with *XA* **show** *?thesis* **by** *auto*
qed

lemma *nonempty-imp-ex-extremal*:
assumes *QA*: $Q \subseteq A$ **and** *Q*: $Q \neq \{\}$
shows $\exists z \in Q.\ \forall y \in Q.\ \neg\ y \sqsubset z$
using *Q prefixed-Imagep-imp-empty*[*of Q*] *QA* **by** (*auto simp*: *Imagep-def*)

interpretation *Restrp*: *well-founded UNIV* $(\sqsubset)\restriction A$
rewrites $\bigwedge x.\ x \in UNIV \equiv True$
and $(\sqsubset)\restriction A\restriction UNIV = (\sqsubset)\restriction A$
and $\bigwedge P1.\ (True \Longrightarrow PROP\ P1) \equiv PROP\ P1$
and $\bigwedge P1.\ (True \Longrightarrow P1) \equiv Trueprop\ P1$
and $\bigwedge P1\ P2.\ (True \Longrightarrow PROP\ P1 \Longrightarrow PROP\ P2) \equiv (PROP\ P1 \Longrightarrow PROP\ P2)$
proof −

```
  have (⋀x. (⋀y. ((⊏) ↾ A) y x ⟹ P y) ⟹ P x) ⟹ P a for a P
    using induct[of a P] by (auto simp: Restrp-def)
  then show well-founded UNIV ((⊏)↾A) apply unfold-locales by auto
qed auto

lemmas Restrp-well-founded = Restrp.well-founded-axioms
lemmas Restrp-induct[consumes 0, case-names less] = Restrp.induct

interpretation Restrp.tranclp: well-founded UNIV ((⊏)↾A)^++
  rewrites ⋀x. x ∈ UNIV ≡ True
    and ((⊏)↾A)^++ ↾ UNIV = ((⊏)↾A)^++
    and (((⊏)↾A)^++)^++ = ((⊏)↾A)^++
    and ⋀P1. (True ⟹ PROP P1) ≡ PROP P1
    and ⋀P1. (True ⟹ P1) ≡ Trueprop P1
    and ⋀P1 P2. (True ⟹ PROP P1 ⟹ PROP P2) ≡ (PROP P1 ⟹ PROP
P2)
proof−
  { fix P x
    assume induct-step: ⋀x. (⋀y. ((⊏)↾A)^++ y x ⟹ P y) ⟹ P x
    have P x
    proof (rule induct-step)
      show ⋀y. ((⊏)↾A)^++ y x ⟹ P y
      proof (induct x rule: Restrp-induct)
        case (less x)
        from ‹((⊏)↾A)^++ y x›
        show ?case
        proof (cases rule: tranclp.cases)
          case r-into-trancl
          with induct-step less show ?thesis by auto
        next
          case (trancl-into-trancl b)
          with less show ?thesis by auto
        qed
      qed
    qed
  }
  then show well-founded UNIV ((⊏)↾A)^++ by unfold-locales auto
qed auto

lemmas Restrp-tranclp-well-founded = Restrp.tranclp.well-founded-axioms
lemmas Restrp-tranclp-induct[consumes 0, case-names less] = Restrp.tranclp.induct

end

context
  fixes A :: 'a set and less :: 'a ⇒ 'a ⇒ bool (infix ⊏ 50)
begin

lemma well-foundedI-pf:
```

assumes *pre*: $\bigwedge X.\ X \subseteq A \Longrightarrow X \subseteq ((\sqsubset)\ ``` X) \cap A \Longrightarrow X = \{\}$
shows *well-founded A* $(\sqsubset)$
proof
fix *P a* **assume** *aA*: $a \in A$ **and** *Ind*: $\bigwedge x.\ x \in A \Longrightarrow (\bigwedge y.\ y \in A \Longrightarrow y \sqsubset x \Longrightarrow P\ y) \Longrightarrow P\ x$
from *Ind* **have** $\{a{\in}A.\ \neg P\ a\} \subseteq ((\sqsubset)\ ``` \{a{\in}A.\ \neg P\ a\}) \cap A$ **by** (*auto simp*: *Imagep-def*)
from *pre*[*OF - this*] *aA*
show *P a* **by** *auto*
qed

lemma *well-foundedI-extremal*:
assumes *a*: $\bigwedge X.\ X \subseteq A \Longrightarrow X \neq \{\} \Longrightarrow \exists\, x \in X.\ \forall\, y \in X.\ \neg\ y \sqsubset x$
shows *well-founded A* $(\sqsubset)$
proof (*rule well-foundedI-pf*)
fix *X* **assume** *XA*: $X \subseteq A$ **and** *pf*: $X \subseteq ((\sqsubset)\ ``` X) \cap A$
from *a*[*OF XA*] *pf* **show** $X = \{\}$ **by** (*auto simp*: *Imagep-def*)
qed

lemma *well-founded-iff-ex-extremal*:
well-founded A $(\sqsubset) \longleftrightarrow (\forall\, X \subseteq A.\ X \neq \{\} \longrightarrow (\exists\, x \in X.\ \forall\, z \in X.\ \neg\ z \sqsubset x))$
using *well-founded.nonempty-imp-ex-extremal well-foundedI-extremal* **by** *blast*

end

lemma *well-founded-cong*:
assumes *r*: $\bigwedge a\ b.\ a \in A \Longrightarrow b \in A \Longrightarrow r\ a\ b \longleftrightarrow r'\ a\ b$
and *A*: $\bigwedge a\ b.\ r'\ a\ b \Longrightarrow a \in A \longleftrightarrow a \in A'$
and *B*: $\bigwedge a\ b.\ r'\ a\ b \Longrightarrow b \in A \longleftrightarrow b \in A'$
shows *well-founded A r* $\longleftrightarrow$ *well-founded A' r'*
proof (*intro iffI*)
assume *wf*: *well-founded A r*
show *well-founded A' r'*
proof (*intro well-foundedI-extremal*)
fix *X*
assume *X*: $X \subseteq A'$ **and** *X0*: $X \neq \{\}$
show $\exists\, x{\in}X.\ \forall\, y{\in}X.\ \neg\ r'\ y\ x$
proof (*cases* $X \cap A = \{\}$)
case *True*
from *X0* **obtain** *x* **where** *xX*: $x \in X$ **by** *auto*
with *True* **have** $x \notin A$ **by** *auto*
with *xX X* **have** $\forall\, y{\in}X.\ \neg\ r'\ y\ x$ **by** (*auto simp*: *B*)
with *xX* **show** *?thesis* **by** *auto*
next
case *False*
from *well-founded.nonempty-imp-ex-extremal*[*OF wf - this*]
obtain *x* **where** *x*: $x \in X \cap A$ **and** *Ar*: $\bigwedge y.\ y \in X \Longrightarrow y \in A \Longrightarrow \neg\ r\ y\ x$
by *auto*
have $\forall\, y \in X.\ \neg\ r'\ y\ x$

```
      proof (intro ballI notI)
        fix y assume yX: y ∈ X and yx: r' y x
        from yX X have yA': y ∈ A' by auto
        show False
        proof (cases y ∈ A)
          case True with x Ar[OF yX] yx r show ?thesis by auto
        next
          case False with yA' x A[OF yx] r X show ?thesis by (auto simp:)
        qed
      qed
      with x show ∃ x ∈ X. ∀ y ∈ X. ¬ r' y x by auto
    qed
  qed
next
  assume wf: well-founded A' r'
  show well-founded A r
  proof (intro well-foundedI-extremal)
    fix X
    assume X: X ⊆ A and X0: X ≠ {}
    show ∃ x∈X. ∀ y∈X. ¬ r y x
    proof (cases X ∩ A' = {})
      case True
      from X0 obtain x where xX: x ∈ X by auto
      with True have x ∉ A' by auto
      with xX X B have ∀ y∈X. ¬ r y x by (auto simp: r in-mono)
      with xX show ?thesis by auto
    next
      case False
      from well-founded.nonempty-imp-ex-extremal[OF wf - this]
      obtain x where x: x ∈ X ∩ A' and Ar: ⋀y. y ∈ X ⟹ y ∈ A' ⟹ ¬ r' y x
by auto
      have ∀ y ∈ X. ¬ r y x
      proof (intro ballI notI)
        fix y assume yX: y ∈ X and yx: r y x
        from yX X have y: y ∈ A by auto
        show False
        proof (cases y ∈ A')
          case True with x Ar[OF yX] yx r X y show ?thesis by auto
        next
          case False with y x A yx r X show ?thesis by auto
        qed
      qed
      with x show ∃ x ∈ X. ∀ y ∈ X. ¬ r y x by auto
    qed
  qed
qed

lemma wfP-iff-well-founded-UNIV: wfP r ⟷ well-founded UNIV r
  by (auto simp: wfP-def wf-def well-founded-def)
```

lemma *well-founded-singleton*:
assumes $\neg r\ x\ x$ **shows** *well-founded* $\{x\}\ r$
using *assms* **by** (*auto simp*: *well-founded-iff-ex-extremal*)

lemma *well-founded-Restrp*[*simp*]: *well-founded* $A\ (r{\restriction}B) \longleftrightarrow$ *well-founded* $(A{\cap}B)$ r (**is** *?l* $\longleftrightarrow$ *?r*)
proof (*intro iffI well-foundedI-extremal*)
assume *l*: *?l*
fix X **assume** *XAB*: $X \subseteq A \cap B$ **and** *X0*: $X \neq \{\}$
with *l*[*THEN well-founded.nonempty-imp-ex-extremal*]
have $\exists\, x{\in}X.\ \forall\, z{\in}X.\ \neg\ (r \restriction B)\ z\ x$ **by** *auto*
with *XAB* **show** $\exists\, x{\in}X.\ \forall\, y{\in}X.\ \neg\ r\ y\ x$ **by** (*auto simp*: *Restrp-def*)
next
assume *r*: *?r*
fix X **assume** *XA*: $X \subseteq A$ **and** *X0*: $X \neq \{\}$
show $\exists\, x{\in}X.\ \forall\, y{\in}X.\ \neg\ (r \restriction B)\ y\ x$
proof (*cases* $X \subseteq B$)
case *True*
with *r*[*THEN well-founded.nonempty-imp-ex-extremal*, *of X*] *XA X0*
have $\exists\, z{\in}X.\ \forall\, y{\in}X.\ \neg\ r\ y\ z$ **by** *auto*
then show *?thesis* **by** *auto*
next
case *False*
then obtain x **where** *x*: $x \in X - B$ **by** *auto*
then have $\forall\, y{\in}X.\ \neg\ (r \restriction B)\ y\ x$ **by** *auto*
with x **show** *?thesis* **by** *auto*
qed
qed

lemma (**in** *well-founded*) *well-founded-subset*:
assumes $B \subseteq A$ **shows** *well-founded* $B\ (\sqsubset)$
using *assms well-founded-axioms* **by** (*auto simp*: *well-founded-iff-ex-extremal*)

lemma *well-founded-extend*:
fixes *less* (**infix** $\sqsubset$ *50*)
assumes *A*: *well-founded* $A\ (\sqsubset)$
assumes *B*: *well-founded* $B\ (\sqsubset)$
assumes *AB*: $\forall\, a \in A.\ \forall\, b \in B.\ \neg b \sqsubset a$
shows *well-founded* $(A \cup B)\ (\sqsubset)$
proof (*intro well-foundedI-extremal*)
interpret *A*: *well-founded* $A\ (\sqsubset)$ **using** *A*.
interpret *B*: *well-founded* $B\ (\sqsubset)$ **using** *B*.
fix X **assume** *XAB*: $X \subseteq A \cup B$ **and** *X0*: $X \neq \{\}$
show $\exists\, x{\in}X.\ \forall\, y{\in}X.\ \neg\ y \sqsubset x$
proof (*cases* $X \cap A = \{\}$)
case *True*
with *XAB* **have** *XB*: $X \subseteq B$ **by** *auto*
from *B.nonempty-imp-ex-extremal*[*OF XB X0*] **show** *?thesis*.

```
  next
    case False
    with A.nonempty-imp-ex-extremal[OF - this]
    obtain e where XAe: e ∈ X ∩ A ∀ y∈X ∩ A. ¬ y ⊏ e by auto
    then have eX: e ∈ X and eA: e ∈ A by auto
    { fix x assume xX: x ∈ X
      have ¬x ⊏ e
      proof (cases x ∈ A)
        case True with XAe xX show ?thesis by auto
      next
        case False
        with xX XAB have x ∈ B by auto
        with AB eA show ?thesis by auto
      qed
    }
    with eX show ?thesis by auto
  qed
qed

lemma closed-UN-well-founded:
  fixes r (infix ⊏ 50)
  assumes XX: ∀ X∈XX. well-founded X (⊏) ∧ (∀ x∈X. ∀ y∈⋃ XX. y ⊏ x ⟶ y
∈ X)
  shows well-founded (⋃ XX) (⊏)
proof (intro well-foundedI-extremal)
  have ∗: X ∈ XX ⟹ x∈X ⟹ y ∈ ⋃ XX ⟹ y ⊏ x ⟹ y ∈ X for X x y using
XX by blast
  fix S
  assume S: S ⊆ ⋃ XX and S0: S ≠ {}
  from S0 obtain x where xS: x ∈ S by auto
  with S obtain X where X: X ∈ XX and xX: x ∈ X by auto
  from xS xX have Sx0: S ∩ X ≠ {} by auto
  from X XX interpret well-founded X (⊏) by auto
  from nonempty-imp-ex-extremal[OF - Sx0]
  obtain z where zS: z ∈ S and zX: z ∈ X and min: ∀ y ∈ S ∩ X. ¬ y ⊏ z by
auto
  show ∃ x∈S. ∀ y∈S. ¬ y ⊏ x
  proof (intro bexI[OF - zS] ballI notI)
    fix y
    assume yS: y ∈ S and yz: y ⊏ z
    have yXX: y ∈ ⋃ XX using S yS by auto
    from ∗[OF X zX yXX yz] yS have y ∈ X ∩ S by auto
    with min yz show False by auto
  qed
qed

lemma well-founded-cmono:
  assumes r': r' ≤ r and wf: well-founded A r
  shows well-founded A r'
```

proof (*intro well-foundedI-extremal*)
fix X **assume** $X \subseteq A$ **and** $X \neq \{\}$
from *well-founded.nonempty-imp-ex-extremal*[*OF wf this*]
show $\exists\, x{\in}X.\ \forall\, y{\in}X.\ \neg\ r'\ y\ x$ **using** r' **by** *auto*
qed

locale *well-founded-ordered-set* = *well-founded* + *transitive* - $(\sqsubset)$
begin

sublocale *strict-ordered-set*..

interpretation *Restrp*: *strict-ordered-set UNIV* $(\sqsubset)\restriction A$ + *Restrp*: *well-founded UNIV* $(\sqsubset)\restriction A$
using *Restrp-strict-order Restrp-well-founded* .

lemma *Restrp-well-founded-order*: *well-founded-ordered-set UNIV* $((\sqsubset)\restriction A)$..

lemma *well-founded-ordered-subset*: $B \subseteq A \Longrightarrow$ *well-founded-ordered-set* B $(\sqsubset)$
apply *intro-locales*
using *well-founded-subset transitive-subset* **by** *auto*

end

lemma (**in** *well-founded*) *Restrp-tranclp-well-founded-ordered*: *well-founded-ordered-set UNIV* $((\sqsubset)\restriction A)^{++}$
using *Restrp-tranclp-well-founded tranclp-transitive* **by** *intro-locales*

locale *well-related-set* = *related-set* +
assumes *nonempty-imp-ex-extreme*: $X \subseteq A \Longrightarrow X \neq \{\} \Longrightarrow \exists\, e.$ *extreme* X $(\sqsubseteq)^{-}$ e
begin

sublocale *connex*
proof
fix $x\ y$ **assume** $x \in A$ **and** $y \in A$
with *nonempty-imp-ex-extreme*[*of* $\{x,y\}$] **show** $x \sqsubseteq y \vee y \sqsubseteq x$ **by** *auto*
qed

lemmas *connex-axioms* = *connex-axioms*

interpretation *less-eq-notations*.

sublocale *asym*: *well-founded* A $(\sqsubset)$
proof (*unfold well-founded-iff-ex-extremal*, *intro allI impI*)
fix X
assume *XA*: $X \subseteq A$ **and** *X0*: $X \neq \{\}$
from *nonempty-imp-ex-extreme*[*OF XA X0*] **obtain** e **where** *extreme* X $(\sqsubseteq)^{-}$ e **by** *auto*
then show $\exists\, x{\in}X.\ \forall\, z{\in}X.\ \neg z \sqsubset x$ **by** (*auto intro*!: *bexI*[*of* - e])

qed

lemma *well-related-subset*: $B \subseteq A \Longrightarrow$ *well-related-set* B $(\sqsubseteq)$
by (*auto intro*!: *well-related-set.intro nonempty-imp-ex-extreme*)

end

context
fixes *less-eq* :: $'a \Rightarrow 'a \Rightarrow$ *bool* (**infix** $\sqsubseteq$ *50*)
begin

lemma *well-related-iff-neg-well-founded*:
well-related-set A $(\sqsubseteq) \longleftrightarrow$ *well-founded* A $(\lambda x\ y.\ \neg\ y \sqsubseteq x)$
by (*simp add*: *well-related-set-def well-founded-iff-ex-extremal extreme-def Bex-def*)

lemma *well-related-singleton-refl*:
assumes $x \sqsubseteq x$ **shows** *well-related-set* $\{x\}$ $(\sqsubseteq)$
by (*intro well-related-set.intro exI*[*of* - x], *auto simp*: *subset-singleton-iff assms*)

lemma *closed-UN-well-related*:
assumes *XX*: $\forall X{\in}XX.$ *well-related-set* X $(\sqsubseteq) \wedge (\forall x{\in}X.\ \forall y{\in}\bigcup XX.\ \neg x \sqsubseteq y \longrightarrow y \in X)$
shows *well-related-set* $(\bigcup XX)$ $(\sqsubseteq)$
using *XX*
apply (*unfold well-related-iff-neg-well-founded*)
using *closed-UN-well-founded*[*of* - $\lambda x\ y.\ \neg\ y \sqsubseteq x$]**.**

end

lemma *well-related-extend*:
fixes r (**infix** $\sqsubseteq$ *50*)
assumes *well-related-set* A $(\sqsubseteq)$ **and** *well-related-set* B $(\sqsubseteq)$ **and** $\forall a \in A.\ \forall b \in B.\ a \sqsubseteq b$
shows *well-related-set* $(A \cup B)$ $(\sqsubseteq)$
using *well-founded-extend*[*of* - $\lambda x\ y.\ \neg\ y \sqsubseteq x$, *folded well-related-iff-neg-well-founded*]
using *assms* **by** *auto*

locale *pre-well-ordered-set* = *semiattractive* + *well-related-set*
begin

interpretation *less-eq-notations***.**

sublocale *transitive*
proof
fix $x\ y\ z$ **assume** *xy*: $x \sqsubseteq y$ **and** *yz*: $y \sqsubseteq z$ **and** x: $x \in A$ **and** y: $y \in A$ **and** z: $z \in A$
from $x\ y\ z$ **have** $\exists e.$ *extreme* $\{x,y,z\}$ $(\sqsupseteq)$ e (**is** $\exists e.$ *?P* e) **by** (*auto intro*!: *nonempty-imp-ex-extreme*)
then have *?P* $x \vee$ *?P* $y \vee$ *?P* z **by** *auto*

then show $x \sqsubseteq z$
proof (*elim disjE*)
assume *?P x*
then show *?thesis* **by** *auto*
next
assume *?P y*
then have $y \sqsubseteq x$ **by** *auto*
from *attract*[*OF xy this yz*] *x y z* **show** *?thesis* **by** *auto*
next
assume *?P z*
then have *zx*: $z \sqsubseteq x$ **and** *zy*: $z \sqsubseteq y$ **by** *auto*
from *attract*[*OF yz zy zx*] *x y z* **have** *yx*: $y \sqsubseteq x$ **by** *auto*
from *attract*[*OF xy yx yz*] *x y z* **show** *?thesis* **by** *auto*
qed
qed

sublocale *total-quasi-ordered-set*..

lemmas *connex-axioms* = *connex-axioms*

lemma *strict-weak-trans*[*trans*]:
assumes *xy*: $x \sqsubset y$ **and** *yz*: $y \sqsubseteq z$ **and** *x*: $x \in A$ **and** *y*: $y \in A$ **and** *z*: $z \in A$
shows $x \sqsubset z$
proof (*intro asympartpI notI*)
from *trans xy yz x y z* **show** $x \sqsubseteq z$ **by** *auto*
assume $z \sqsubseteq x$
from *trans*[*OF yz this*] *y z x* **have** $y \sqsubseteq x$ **by** *auto*
with *xy* **show** *False* **by** *auto*
qed

lemma *weak-strict-trans*[*trans*]:
assumes *xy*: $x \sqsubseteq y$ **and** *yz*: $y \sqsubset z$ **and** *x*: $x \in A$ **and** *y*: $y \in A$ **and** *z*: $z \in A$
shows $x \sqsubset z$
proof (*intro asympartpI notI*)
from *trans xy yz x y z* **show** $x \sqsubseteq z$ **by** *auto*
assume $z \sqsubseteq x$
from *trans*[*OF this xy*] *z x y* **have** $z \sqsubseteq y$ **by** *auto*
with *yz* **show** *False* **by** *auto*
qed

end

lemma *pre-well-ordered-iff*:
pre-well-ordered-set A r $\longleftrightarrow$ *total-quasi-ordered-set A r* $\wedge$ *well-founded A* (*asympartp r*)
(**is** *?p* $\longleftrightarrow$ *?t* $\wedge$ *?w*)
proof *safe*
assume *?p*
then interpret *pre-well-ordered-set A r*.

show *?t ?w* **by** *unfold-locales*
next
assume *?t*
then interpret *total-quasi-ordered-set A r*.
assume *?w*
then have *well-founded UNIV* (*asympartp r* ↾ *A*) **by** *simp*
also have *asympartp r* ↾ *A* = (λ*x y*. ¬ *r y x*) ↾ *A* **by** (*intro ext*, *auto simp*: *not-iff-asym*)
finally have *well-related-set A r* **by** (*simp add*: *well-related-iff-neg-well-founded*)
then show *?p* **by** *intro-locales*
qed

lemma (**in** *semiattractive*) *pre-well-ordered-iff-well-related*:
assumes *XA*: *X* ⊆ *A*
shows *pre-well-ordered-set X* (⊑) ⟷ *well-related-set X* (⊑) (**is** *?l* ⟷ *?r*)
proof
interpret *X*: *semiattractive X* **using** *semiattractive-subset*[*OF XA*].
{ assume *?l*
then interpret *X*: *pre-well-ordered-set X*.
show *?r* **by** *unfold-locales*
}
assume *?r*
then interpret *X*: *well-related-set X*.
show *?l* **by** *unfold-locales*
qed

lemma *semiattractive-extend*:
fixes *r* (**infix** ⊑ *50*)
assumes *A*: *semiattractive A* (⊑) **and** *B*: *semiattractive B* (⊑)
and *AB*: ∀ *a* ∈ *A*. ∀ *b* ∈ *B*. *a* ⊑ *b* ∧ ¬ *b* ⊑ *a*
shows *semiattractive* (*A* ∪ *B*) (⊑)
proof−
interpret *A*: *semiattractive A* (⊑) **using** *A*.
interpret *B*: *semiattractive B* (⊑) **using** *B*.
{
fix *x y z*
assume *yB*: *y* ∈ *B* **and** *zA*: *z* ∈ *A* **and** *yz*: *y* ⊑ *z*
have *False* **using** *AB*[*rule-format*, *OF zA yB*] *yz* **by** *auto*
}
note ∗ = *this*
show *?thesis*
by (*auto intro*!: *semiattractive.intro dest*:∗ *AB*[*rule-format*] *A.attract B.attract*)
qed

lemma *pre-well-order-extend*:
fixes *r* (**infix** ⊑ *50*)
assumes *A*: *pre-well-ordered-set A* (⊑) **and** *B*: *pre-well-ordered-set B* (⊑)
and *AB*: ∀ *a* ∈ *A*. ∀ *b* ∈ *B*. *a* ⊑ *b* ∧ ¬ *b* ⊑ *a*
shows *pre-well-ordered-set* (*A*∪*B*) (⊑)

proof–
interpret A: *pre-well-ordered-set* A ($\sqsubseteq$) **using** A**.**
interpret B: *pre-well-ordered-set* B ($\sqsubseteq$) **using** B**.**
show *?thesis*
apply (*intro pre-well-ordered-set.intro well-related-extend semiattractive-extend*)
apply *unfold-locales*
by (*auto dest*: AB[*rule-format*])
qed

locale *well-ordered-set* = *antisymmetric* + *well-related-set*
begin

sublocale *pre-well-ordered-set*..

sublocale *total-ordered-set*..

lemma *well-ordered-subset*: $B \subseteq A \Longrightarrow$ *well-ordered-set* B ($\sqsubseteq$)
using *well-related-subset antisymmetric-subset* **by** (*intro well-ordered-set.intro*)

lemmas *connex-axioms* = *connex-axioms*

end

lemma (**in** *antisymmetric*) *well-ordered-iff-well-related*:
assumes XA: $X \subseteq A$
shows *well-ordered-set* X ($\sqsubseteq$) $\longleftrightarrow$ *well-related-set* X ($\sqsubseteq$) (**is** *?l* $\longleftrightarrow$ *?r*)
proof
interpret X: *antisymmetric* X **using** *antisymmetric-subset*[*OF* XA]**.**
{ assume *?l*
then interpret X: *well-ordered-set* X**.**
show *?r* **by** *unfold-locales*
}
assume *?r*
then interpret X: *well-related-set* X**.**
show *?l* **by** *unfold-locales*
qed

context
fixes A **and** *less-eq* (**infix** $\sqsubseteq$ *50*)
assumes A: $\forall\, a \in A.\ \forall\, b \in A.\ a \sqsubseteq b$
begin

interpretation *well-related-set* A ($\sqsubseteq$)
apply *unfold-locales*
using A **by** *blast*

lemmas *trivial-well-related* = *well-related-set-axioms*

lemma *trivial-pre-well-order*: *pre-well-ordered-set* A ($\sqsubseteq$)

apply *unfold-locales*
using *A* **by** *blast*

end

context
fixes *less-eq* :: $'a \Rightarrow 'a \Rightarrow bool$ (**infix** $\sqsubseteq$ *50*)
begin

interpretation *less-eq-notations***.**

lemma *well-order-extend*:
assumes *A*: *well-ordered-set A* ($\sqsubseteq$) **and** *B*: *well-ordered-set B* ($\sqsubseteq$)
and *ABa*: $\forall\, a \in A.\ \forall\, b \in B.\ a \sqsubseteq b \longrightarrow b \sqsubseteq a \longrightarrow a = b$
and *AB*: $\forall\, a \in A.\ \forall\, b \in B.\ a \sqsubseteq b$
shows *well-ordered-set* ($A \cup B$) ($\sqsubseteq$)
proof−
interpret *A*: *well-ordered-set A* ($\sqsubseteq$) **using** *A***.**
interpret *B*: *well-ordered-set B* ($\sqsubseteq$) **using** *B***.**
show *?thesis*
apply (*intro well-ordered-set.intro antisymmetric-union well-related-extend ABa AB*)
by *unfold-locales*
qed

interpretation *singleton*: *antisymmetric* $\{a\}$ ($\sqsubseteq$) **for** *a* **apply** *unfold-locales* **by** *auto*

lemmas *singleton-antisymmetric*[*intro*!] = *singleton.antisymmetric-axioms*

lemma *singleton-well-ordered*[*intro*!]: $a \sqsubseteq a \Longrightarrow$ *well-ordered-set* $\{a\}$ ($\sqsubseteq$)
apply *unfold-locales* **by** *auto*

lemma *closed-UN-well-ordered*:
assumes *anti*: *antisymmetric* ($\bigcup$ *XX*) ($\sqsubseteq$)
and *XX*: $\forall\, X{\in}XX.$ *well-ordered-set X* ($\sqsubseteq$) $\wedge$ ($\forall\, x{\in}X.\ \forall\, y{\in}\bigcup XX.\ \neg\ x \sqsubseteq y \longrightarrow y \in X$)
shows *well-ordered-set* ($\bigcup XX$) ($\sqsubseteq$)
apply (*intro well-ordered-set.intro closed-UN-well-related anti*)
using *XX well-ordered-set.axioms* **by** *fast*

end

Directed sets:

definition *directed A r* $\equiv \forall\, x \in A.\ \forall\, y \in A.\ \exists\, z \in A.\ r\ x\ z \wedge r\ y\ z$

lemmas *directedI*[*intro*] = *directed-def*[*unfolded atomize-eq*, *THEN iffD2*, *rule-format*]

lemmas *directedD* = *directed-def*[*unfolded atomize-eq*, *THEN iffD1*, *rule-format*]

context
  fixes *less-eq* :: *'a* ⇒ *'a* ⇒ *bool* (**infix** ⊑ *50*)
begin

lemma *directedE*:
  assumes *directed A* (⊑) **and** *x* ∈ *A* **and** *y* ∈ *A*
    and ⋀*z*. *z* ∈ *A* ⟹ *x* ⊑ *z* ⟹ *y* ⊑ *z* ⟹ *thesis*
  shows *thesis*
  using *assms* **by** (*auto dest*: *directedD*)

lemma *directed-empty*[*simp*]: *directed* {} (⊑) **by** *auto*

lemma *directed-union*:
  assumes *dX*: *directed X* (⊑) **and** *dY*: *directed Y* (⊑)
    and *XY*: ∀ *x*∈*X*. ∀ *y*∈*Y*. ∃ *z* ∈ *X* ∪ *Y*. *x* ⊑ *z* ∧ *y* ⊑ *z*
  shows *directed* (*X* ∪ *Y*) (⊑)
  using *directedD*[*OF dX*] *directedD*[*OF dY*] *XY*
  apply (*intro directedI*) **by** *blast*

lemma *directed-extend*:
  assumes *X*: *directed X* (⊑) **and** *Y*: *directed Y* (⊑) **and** *XY*: ∀ *x*∈*X*. ∀ *y*∈*Y*. *x* ⊑ *y*
  shows *directed* (*X* ∪ *Y*) (⊑)
proof −
  { fix *x y*
    assume *xX*: *x* ∈ *X* **and** *yY*: *y* ∈ *Y*
    let *?g* = ∃ *z*∈*X* ∪ *Y*. *x* ⊑ *z* ∧ *y* ⊑ *z*
    from *directedD*[*OF Y yY yY*] **obtain** *z* **where** *zY*: *z* ∈ *Y* **and** *yz*: *y* ⊑ *z* **by** *auto*
    from *xX XY zY yz* **have** *?g* **by** *auto*
  }
  then show *?thesis* **by** (*auto intro*!: *directed-union*[*OF X Y*])
qed

end

context *connex* **begin**

lemma *directed*: *directed A* (⊑)
proof
  fix *x y*
  assume *x*: *x* ∈ *A* **and** *y*: *y* ∈ *A*
  then show ∃ *z*∈*A*. *x* ⊑ *z* ∧ *y* ⊑ *z*
  proof (*cases rule*: *comparable-cases*)
    case *le*
    with *refl*[*OF y*] *y* **show** *?thesis* **by** (*intro bexI*[*of* - *y*], *auto*)
  next
    case *ge*

with *refl*[*OF x*] *x* **show** *?thesis* **by** (*intro bexI*[*of - x*], *auto*)
qed
qed

end

context
fixes *ir* :: $'i \Rightarrow 'i \Rightarrow$ *bool* (**infix** $\preceq$ *50*)
fixes *r* :: $'a \Rightarrow 'a \Rightarrow$ *bool* (**infix** $\sqsubseteq$ *50*)
begin

lemma *monotone-connex-image*:
assumes *mono*: *monotone-on I* ($\preceq$) ($\sqsubseteq$) *f*
assumes *connex*: *connex I* ($\preceq$)
shows *connex* (*f* ‘ *I*) ($\sqsubseteq$)
proof (*rule connexI*)
fix *x y*
assume $x \in f$ ‘ I **and** $y \in f$ ‘ I
then obtain *i j* **where** *ij*: $i \in I\ j \in I$ **and** [*simp*]: $x = f\ i\ y = f\ j$ **by** *auto*
from *connex ij* **have** $i \preceq j \vee j \preceq i$ **by** (*auto elim*: *connexE*)
with *ij mono* **show** $x \sqsubseteq y \vee y \sqsubseteq x$ **by** (*elim disjE*, *auto dest*: *monotone-onD*)
qed

lemma *monotone-directed-image*:
assumes *mono*: *monotone-on I* ($\preceq$) ($\sqsubseteq$) *f*
assumes *dir*: *directed I* ($\preceq$) **shows** *directed* (*f* ‘ *I*) ($\sqsubseteq$)
proof (*rule directedI*, *safe*)
fix *x y* **assume** *x*: $x \in I$ **and** *y*: $y \in I$
with *dir* **obtain** *z* **where** *z*: $z \in I$ **and** $x \preceq z$ **and** $y \preceq z$ **by** (*auto elim*: *directedE*)
with *mono x y* **have** $f\ x \sqsubseteq f\ z$ **and** $f\ y \sqsubseteq f\ z$ **by** (*auto dest*: *monotone-onD*)
with *z* **show** $\exists fz \in f$ ‘ $I.\ f\ x \sqsubseteq fz \wedge f\ y \sqsubseteq fz$ **by** *auto*
qed

end

2.6 Order Pairs

locale *compatible* = *related-set* + *related-set A* ($\sqsubset$) **for** *less* (**infix** $\sqsubset$ *50*) +
assumes *compat-right*[*trans*]: $x \sqsubseteq y \Longrightarrow y \sqsubset z \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow z \in A \Longrightarrow x \sqsubset z$
assumes *compat-left*[*trans*]: $x \sqsubset y \Longrightarrow y \sqsubseteq z \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow z \in A \Longrightarrow x \sqsubset z$

locale *compatible-ordering* = *reflexive* + *irreflexive* + *compatible* +
assumes *strict-implies-weak*: $x \sqsubset y \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow x \sqsubseteq y$
begin

The strict part is necessarily transitive.

The following sequence of declarations are in order to obtain fact names

in a manner similar to the Isabelle/HOL facts of orders.

sublocale *strict*: *transitive A* ($\sqsubset$)
  using *compat-right*[*OF strict-implies-weak*] **by** *unfold-locales*

sublocale *strict-ordered-set A* ($\sqsubset$) **..**

thm *strict.trans asym irrefl*

lemma *strict-implies-not-weak*: $x \sqsubset y \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow \neg\ y \sqsubseteq x$
  using *irrefl compat-left* **by** *blast*

end

context *transitive* **begin**

interpretation *less-eq-notations*.

lemma *asym-trans*[*trans*]:
  shows $x \sqsubset y \Longrightarrow y \sqsubseteq z \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow z \in A \Longrightarrow x \sqsubset z$
    and $x \sqsubseteq y \Longrightarrow y \sqsubset z \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow z \in A \Longrightarrow x \sqsubset z$
  by (*auto 0 3 dest*: *trans*)

end

locale *attractive-ordering* = *compatible-ordering* + *attractive*

locale *pseudo-ordering* = *compatible-ordering* + *pseudo-ordered-set*
begin

sublocale *attractive-ordering* **..**

end

locale *quasi-ordering* = *compatible-ordering* + *quasi-ordered-set*
begin

sublocale *attractive-ordering* **..**

end

locale *partial-ordering* = *compatible-ordering* + *partially-ordered-set*
begin

sublocale *pseudo-ordering* + *quasi-ordering* **..**

end

locale *well-founded-ordering* = *quasi-ordering* + *well-founded*

locale *total-ordering* = *compatible-ordering* + *total-ordered-set*
begin

sublocale *partial-ordering* **..**

end

locale *strict-total-ordering* = *partial-ordering* + *semiconnex A* (⊏)
begin

sublocale *semiconnex-irreflexive* **..**

sublocale *connex*
proof
  fix *x y* **assume** *x*: $x \in A$ **and** *y*: $y \in A$
  then show $x \sqsubseteq y \vee y \sqsubseteq x$
    apply (*cases rule*: *cases*[*OF x y*])
    by (*auto dest*: *strict-implies-weak*)
qed

sublocale *total-ordering* **..**

lemma *not-weak*[*simp*]:
  assumes $x \in A$ **and** $y \in A$ **shows** $\neg\ x \sqsubseteq y \longleftrightarrow y \sqsubset x$
 using *assms* **by** (*cases rule*:*cases*, *auto simp*: *strict-implies-not-weak dest*: *strict-implies-weak*)

lemma *not-strict*[*simp*]: $x \in A \Longrightarrow y \in A \Longrightarrow \neg\ x \sqsubset y \longleftrightarrow y \sqsubseteq x$
  using *not-weak* **by** *blast*

end

2.7 Relating to Classes

In Isabelle 2020, we should declare sublocales in class before declaring dual sublocales, since otherwise facts would be prefixed by "dual.dual."

context *ord* **begin**

abbreviation *least* **where** *least X* ≡ *extreme X* (λ*x y*. *y* ≤ *x*)

abbreviation *greatest* **where** *greatest X* ≡ *extreme X* (≤)

abbreviation *supremum* **where** *supremum X* ≡ *least* (*Collect* (*bound X* (≤)))

abbreviation *infimum* **where** *infimum X* ≡ *greatest* (*Collect* (*bound X* (λ*x y*. *y* ≤ *x*)))

lemma *Least-eq-The-least*: *Least P* = *The* (*least* {*x*. *P x*})
  by (*auto simp*: *Least-def extreme-def*[*unfolded atomize-eq*, *THEN ext*])

lemma *Greatest-eq-The-greatest*: *Greatest P = The (greatest {x. P x})*
by (*auto simp*: *Greatest-def extreme-def*[*unfolded atomize-eq*, *THEN ext*])

end

lemma *Ball-UNIV*[*simp*]: *Ball UNIV = All* **by** *auto*
lemma *Bex-UNIV*[*simp*]: *Bex UNIV = Ex* **by** *auto*

class *compat = ord* + **assumes** *compatible-ordering UNIV* (≤) (<)
begin

sublocale *order*: *compatible-ordering UNIV*
rewrites ⋀*x*. *x* ∈ *UNIV* ≡ *True*
and ⋀*X*. *X* ⊆ *UNIV* ≡ *True*
and ⋀*r*. *r* ↾ *UNIV* ≡ *r*
and ⋀*P*. *True* ∧ *P* ≡ *P*
and *Ball UNIV* ≡ *All*
and *Bex UNIV* ≡ *Ex*
and *sympartp* (≤)⁻ ≡ *sympartp* (≤)
and ⋀*P1*. (*True* ⟹ *PROP P1*) ≡ *PROP P1*
and ⋀*P1*. (*True* ⟹ *P1*) ≡ *Trueprop P1*
and ⋀*P1 P2*. (*True* ⟹ *PROP P1* ⟹ *PROP P2*) ≡ (*PROP P1* ⟹ *PROP P2*)
using *compat-axioms* **unfolding** *class.compat-def* **by** (*auto 0 4 simp*:*atomize-eq*)

end

We should have imported locale-based facts in classes, e.g.:

thm *order.trans order.strict.trans order.refl order.irrefl order.asym order.extreme-bound-singleton*

class *attractive-order = ord* + **assumes** *attractive-ordering UNIV* (≤) (<)
begin

We need to declare subclasses before sublocales in order to preserve facts for superclasses.

interpretation *attractive-ordering UNIV*
using *attractive-order-axioms* **unfolding** *class.attractive-order-def*.

subclass *compat* ..

sublocale *order*: *attractive-ordering UNIV*
rewrites ⋀*x*. *x* ∈ *UNIV* ≡ *True*
and ⋀*X*. *X* ⊆ *UNIV* ≡ *True*
and ⋀*r*. *r* ↾ *UNIV* ≡ *r*
and ⋀*P*. *True* ∧ *P* ≡ *P*
and *Ball UNIV* ≡ *All*
and *Bex UNIV* ≡ *Ex*
and *sympartp* (≤)⁻ ≡ *sympartp* (≤)

and $\bigwedge P1.\ (True \Longrightarrow PROP\ P1) \equiv PROP\ P1$
and $\bigwedge P1.\ (True \Longrightarrow P1) \equiv Trueprop\ P1$
and $\bigwedge P1\ P2.\ (True \Longrightarrow PROP\ P1 \Longrightarrow PROP\ P2) \equiv (PROP\ P1 \Longrightarrow PROP\ P2)$
apply *unfold-locales* **by** (*auto simp*:*atomize-eq*)

end

thm *order.extreme-bound-quasi-const*

class *psorder* = *ord* + **assumes** *pseudo-ordering UNIV* (≤) (<)
begin

interpretation *pseudo-ordering UNIV* **using** *psorder-axioms* **unfolding** *class.psorder-def*.

subclass *attractive-order* ..

sublocale *order*: *pseudo-ordering UNIV*
rewrites $\bigwedge x.\ x \in UNIV \equiv True$
and $\bigwedge X.\ X \subseteq UNIV \equiv True$
and $\bigwedge r.\ r \upharpoonright UNIV \equiv r$
and $\bigwedge P.\ True \wedge P \equiv P$
and $Ball\ UNIV \equiv All$
and $Bex\ UNIV \equiv Ex$
and $sympartp\ (\leq)^{-} \equiv sympartp\ (\leq)$
and $\bigwedge P1.\ (True \Longrightarrow PROP\ P1) \equiv PROP\ P1$
and $\bigwedge P1.\ (True \Longrightarrow P1) \equiv Trueprop\ P1$
and $\bigwedge P1\ P2.\ (True \Longrightarrow PROP\ P1 \Longrightarrow PROP\ P2) \equiv (PROP\ P1 \Longrightarrow PROP\ P2)$
apply *unfold-locales* **by** (*auto simp*:*atomize-eq*)

end

class *qorder* = *ord* + **assumes** *quasi-ordering UNIV* (≤) (<)
begin

interpretation *quasi-ordering UNIV* **using** *qorder-axioms* **unfolding** *class.qorder-def*.

subclass *attractive-order* ..

sublocale *order*: *quasi-ordering UNIV*
rewrites $\bigwedge x.\ x \in UNIV \equiv True$
and $\bigwedge X.\ X \subseteq UNIV \equiv True$
and $\bigwedge r.\ r \upharpoonright UNIV \equiv r$
and $\bigwedge P.\ True \wedge P \equiv P$
and $Ball\ UNIV \equiv All$
and $Bex\ UNIV \equiv Ex$
and $sympartp\ (\leq)^{-} \equiv sympartp\ (\leq)$
and $\bigwedge P1.\ (True \Longrightarrow PROP\ P1) \equiv PROP\ P1$

and $\bigwedge P1.\ (True \Longrightarrow P1) \equiv Trueprop\ P1$
and $\bigwedge P1\ P2.\ (True \Longrightarrow PROP\ P1 \Longrightarrow PROP\ P2) \equiv (PROP\ P1 \Longrightarrow PROP\ P2)$
apply *unfold-locales* **by** (*auto simp*:*atomize-eq*)

end

class *porder* = *ord* + **assumes** *partial-ordering UNIV* (≤) (<)
begin

interpretation *partial-ordering UNIV*
using *porder-axioms* **unfolding** *class.porder-def*.

subclass *psorder* ..
subclass *qorder* ..

sublocale *order*: *partial-ordering UNIV*
rewrites $\bigwedge x.\ x \in UNIV \equiv True$
and $\bigwedge X.\ X \subseteq UNIV \equiv True$
and $\bigwedge r.\ r \upharpoonright UNIV \equiv r$
and $\bigwedge P.\ True \wedge P \equiv P$
and $Ball\ UNIV \equiv All$
and $Bex\ UNIV \equiv Ex$
and $sympartp\ (\leq)^{-} \equiv sympartp\ (\leq)$
and $\bigwedge P1.\ (True \Longrightarrow PROP\ P1) \equiv PROP\ P1$
and $\bigwedge P1.\ (True \Longrightarrow P1) \equiv Trueprop\ P1$
and $\bigwedge P1\ P2.\ (True \Longrightarrow PROP\ P1 \Longrightarrow PROP\ P2) \equiv (PROP\ P1 \Longrightarrow PROP\ P2)$
apply *unfold-locales* **by** (*auto simp*:*atomize-eq*)

end

class *wf-qorder* = *ord* + **assumes** *well-founded-ordering UNIV* (≤) (<)
begin

interpretation *well-founded-ordering UNIV*
using *wf-qorder-axioms* **unfolding** *class.wf-qorder-def*.

subclass *qorder* ..

sublocale *order*: *well-founded-ordering UNIV*
rewrites $\bigwedge x.\ x \in UNIV \equiv True$
and $\bigwedge X.\ X \subseteq UNIV \equiv True$
and $\bigwedge r.\ r \upharpoonright UNIV \equiv r$
and $\bigwedge P.\ True \wedge P \equiv P$
and $Ball\ UNIV \equiv All$
and $Bex\ UNIV \equiv Ex$
and $sympartp\ (\leq)^{-} \equiv sympartp\ (\leq)$
and $\bigwedge P1.\ (True \Longrightarrow PROP\ P1) \equiv PROP\ P1$

and $\bigwedge P1.\ (True \Longrightarrow P1) \equiv Trueprop\ P1$
and $\bigwedge P1\ P2.\ (True \Longrightarrow PROP\ P1 \Longrightarrow PROP\ P2) \equiv (PROP\ P1 \Longrightarrow PROP\ P2)$
apply *unfold-locales* **by** (*auto simp*:*atomize-eq*)

end

class *totalorder* = *ord* + **assumes** *total-ordering UNIV* ($\leq$) ($<$)
begin

interpretation *total-ordering UNIV*
using *totalorder-axioms* **unfolding** *class.totalorder-def*.

subclass *porder* ..

sublocale *order*: *total-ordering UNIV*
rewrites $\bigwedge x.\ x \in UNIV \equiv True$
and $\bigwedge X.\ X \subseteq UNIV \equiv True$
and $\bigwedge r.\ r \restriction UNIV \equiv r$
and $\bigwedge P.\ True \wedge P \equiv P$
and *Ball UNIV* $\equiv$ *All*
and *Bex UNIV* $\equiv$ *Ex*
and *sympartp* $(\leq)^{-} \equiv$ *sympartp* $(\leq)$
and $\bigwedge P1.\ (True \Longrightarrow PROP\ P1) \equiv PROP\ P1$
and $\bigwedge P1.\ (True \Longrightarrow P1) \equiv Trueprop\ P1$
and $\bigwedge P1\ P2.\ (True \Longrightarrow PROP\ P1 \Longrightarrow PROP\ P2) \equiv (PROP\ P1 \Longrightarrow PROP\ P2)$
apply *unfold-locales* **by** (*auto simp*:*atomize-eq*)

end

Isabelle/HOL's *preorder* belongs to *qorder*, but not vice versa.

subclass (**in** *preorder*) *qorder*
apply *unfold-locales*
using *order-refl* **apply** *assumption*
apply *simp*
using *le-less-trans* **apply** *assumption*
using *less-le-trans* **apply** *assumption*
using *less-imp-le* **apply** *assumption*
using *order-trans* **apply** *assumption*
done

subclass (**in** *order*) *porder* **by** (*unfold-locales*, *auto*)

subclass (**in** *wellorder*) *wf-qorder*
apply (*unfold-locales*)
using *less-induct* **by** *auto*

Isabelle/HOL's *linorder* is equivalent to our locale *strict-total-ordering*.

context *linorder* **begin**

interpretation *strict-total-ordering UNIV*
apply *unfold-locales* **by** *auto*

subclass *totalorder* **..**

sublocale *order*: *strict-total-ordering UNIV*
rewrites $\bigwedge x.\ x \in UNIV \equiv True$
and $\bigwedge X.\ X \subseteq UNIV \equiv True$
and $\bigwedge r.\ r \restriction UNIV \equiv r$
and $\bigwedge P.\ True \wedge P \equiv P$
and *Ball UNIV* $\equiv$ *All*
and *Bex UNIV* $\equiv$ *Ex*
and *sympartp* $(\leq)^- \equiv$ *sympartp* $(\leq)$
and $\bigwedge P1.\ (True \Longrightarrow PROP\ P1) \equiv PROP\ P1$
and $\bigwedge P1.\ (True \Longrightarrow P1) \equiv Trueprop\ P1$
and $\bigwedge P1\ P2.\ (True \Longrightarrow PROP\ P1 \Longrightarrow PROP\ P2) \equiv (PROP\ P1 \Longrightarrow PROP\ P2)$
apply *unfold-locales* **by** (*auto simp*:*atomize-eq*)

end

Tests: facts should be available in the most general classes.

thm *order.strict.trans*[**where** *'a='a*::*compat*]
thm *order.extreme-bound-quasi-const*[**where** *'a='a*::*attractive-order*]
thm *order.extreme-bound-singleton-eq*[**where** *'a='a*::*psorder*]
thm *order.trans*[**where** *'a='a*::*qorder*]
thm *order.comparable-cases*[**where** *'a='a*::*totalorder*]
thm *order.cases*[**where** *'a='a*::*linorder*]

2.8 Declaring Duals

sublocale *reflexive* $\subseteq$ *sym*: *reflexive A sympartp* $(\sqsubseteq)$
rewrites *sympartp* $(\sqsubseteq)^- \equiv$ *sympartp* $(\sqsubseteq)$
and $\bigwedge r.$ *sympartp* (*sympartp r*) $\equiv$ *sympartp r*
and $\bigwedge r.$ *sympartp r* $\restriction A \equiv$ *sympartp* $(r \restriction A)$
by (*auto 0 4 simp*:*atomize-eq*)

sublocale *quasi-ordered-set* $\subseteq$ *sym*: *quasi-ordered-set A sympartp* $(\sqsubseteq)$
rewrites *sympartp* $(\sqsubseteq)^- =$ *sympartp* $(\sqsubseteq)$
and *sympartp* (*sympartp* $(\sqsubseteq)$) = *sympartp* $(\sqsubseteq)$
apply *unfold-locales* **by** (*auto 0 4 dest*: *trans*)

At this point, we declare dual as sublocales. In the following, "rewrites" eventually cleans up redundant facts.

sublocale *reflexive* $\subseteq$ *dual*: *reflexive A* $(\sqsubseteq)^-$
rewrites *sympartp* $(\sqsubseteq)^- \equiv$ *sympartp* $(\sqsubseteq)$
and $\bigwedge r.$ *sympartp* $(r \restriction A) \equiv$ *sympartp r* $\restriction A$
and $(\sqsubseteq)^- \restriction A \equiv ((\sqsubseteq) \restriction A)^-$

by (*auto simp*: *atomize-eq*)

context *attractive* **begin**

interpretation *less-eq-notations*.

sublocale *dual*: *attractive A* $(\sqsupseteq)$
rewrites *sympartp* $(\sqsupseteq) = (\sim)$
and *equivpartp* $(\sqsupseteq) \equiv (\simeq)$
and $\bigwedge r$. *sympartp* $(r \restriction A) \equiv$ *sympartp* $r \restriction A$
and $\bigwedge r$. *sympartp* (*sympartp* r) $\equiv$ *sympartp* r
and $(\sqsubseteq)^- \restriction A \equiv ((\sqsubseteq) \restriction A)^-$
apply *unfold-locales* **by** (*auto intro*!: *ext dest*: *attract dual.attract simp*: *atomize-eq*)

end

context *irreflexive* **begin**

sublocale *dual*: *irreflexive A* $(\sqsubset)^-$
rewrites $(\sqsubset)^- \restriction A \equiv ((\sqsubset) \restriction A)^-$
apply *unfold-locales* **by** (*auto dest*: *irrefl simp*: *atomize-eq*)

end

sublocale *transitive* $\subseteq$ *dual*: *transitive A* $(\sqsubseteq)^-$
rewrites $(\sqsubseteq)^- \restriction A \equiv ((\sqsubseteq) \restriction A)^-$
and *sympartp* $(\sqsubseteq)^-$ = *sympartp* $(\sqsubseteq)$
and *asympartp* $(\sqsubseteq)^-$ = (*asympartp* $(\sqsubseteq)$)$^-$
apply *unfold-locales* **by** (*auto dest*: *trans simp*: *atomize-eq intro*!:*ext*)

sublocale *antisymmetric* $\subseteq$ *dual*: *antisymmetric A* $(\sqsubseteq)^-$
rewrites $(\sqsubseteq)^- \restriction A \equiv ((\sqsubseteq) \restriction A)^-$
and *sympartp* $(\sqsubseteq)^-$ = *sympartp* $(\sqsubseteq)$
by (*auto dest*: *antisym simp*: *atomize-eq*)

sublocale *semiconnex* $\subseteq$ *dual*: *semiconnex A* $(\sqsubset)^-$
rewrites *sympartp* $(\sqsubset)^-$ = *sympartp* $(\sqsubset)$
using *semiconnex* **by** *auto*

sublocale *connex* $\subseteq$ *dual*: *connex A* $(\sqsubseteq)^-$
rewrites *sympartp* $(\sqsubseteq)^-$ = *sympartp* $(\sqsubseteq)$
by (*auto intro*!: *chainI dest*:*comparable*)

sublocale *semiconnex-irreflexive* $\subseteq$ *dual*: *semiconnex-irreflexive A* $(\sqsubset)^-$
rewrites *sympartp* $(\sqsubset)^-$ = *sympartp* $(\sqsubset)$
by *unfold-locales auto*

sublocale *pseudo-ordered-set* $\subseteq$ *dual*: *pseudo-ordered-set A* $(\sqsubseteq)^-$

rewrites *sympartp* $(\sqsubseteq)^-$ = *sympartp* $(\sqsubseteq)$
by *unfold-locales* (*auto 0 4*)

sublocale *quasi-ordered-set* $\subseteq$ *dual*: *quasi-ordered-set A* $(\sqsubseteq)^-$
rewrites *sympartp* $(\sqsubseteq)^-$ = *sympartp* $(\sqsubseteq)$
by *unfold-locales auto*

sublocale *partially-ordered-set* $\subseteq$ *dual*: *partially-ordered-set A* $(\sqsubseteq)^-$
rewrites *sympartp* $(\sqsubseteq)^-$ = *sympartp* $(\sqsubseteq)$
by *unfold-locales* (*auto 0 4*)

sublocale *total-pseudo-ordered-set* $\subseteq$ *dual*: *total-pseudo-ordered-set A* $(\sqsubseteq)^-$
rewrites *sympartp* $(\sqsubseteq)^-$ = *sympartp* $(\sqsubseteq)$
by *unfold-locales* (*auto 0 4*)

sublocale *total-quasi-ordered-set* $\subseteq$ *dual*: *total-quasi-ordered-set A* $(\sqsubseteq)^-$
rewrites *sympartp* $(\sqsubseteq)^-$ = *sympartp* $(\sqsubseteq)$
by *unfold-locales auto*

sublocale *compatible-ordering* $\subseteq$ *dual*: *compatible-ordering A* $(\sqsubseteq)^-$ $(\sqsubset)^-$
rewrites *sympartp* $(\sqsubseteq)^-$ = *sympartp* $(\sqsubseteq)$
apply *unfold-locales*
by (*auto dest*: *compat-left compat-right strict-implies-weak*)

sublocale *attractive-ordering* $\subseteq$ *dual*: *attractive-ordering A* $(\sqsubseteq)^-$ $(\sqsubset)^-$
rewrites *sympartp* $(\sqsubseteq)^-$ = *sympartp* $(\sqsubseteq)$
by *unfold-locales auto*

sublocale *pseudo-ordering* $\subseteq$ *dual*: *pseudo-ordering A* $(\sqsubseteq)^-$ $(\sqsubset)^-$
rewrites *sympartp* $(\sqsubseteq)^-$ = *sympartp* $(\sqsubseteq)$
by *unfold-locales* (*auto 0 4*)

sublocale *quasi-ordering* $\subseteq$ *dual*: *quasi-ordering A* $(\sqsubseteq)^-$ $(\sqsubset)^-$
rewrites *sympartp* $(\sqsubseteq)^-$ = *sympartp* $(\sqsubseteq)$
by *unfold-locales auto*

sublocale *partial-ordering* $\subseteq$ *dual*: *partial-ordering A* $(\sqsubseteq)^-$ $(\sqsubset)^-$
rewrites *sympartp* $(\sqsubseteq)^-$ = *sympartp* $(\sqsubseteq)$
by *unfold-locales* (*auto 0 4*)

sublocale *total-ordering* $\subseteq$ *dual*: *total-ordering A* $(\sqsubseteq)^-$ $(\sqsubset)^-$
rewrites *sympartp* $(\sqsubseteq)^-$ = *sympartp* $(\sqsubseteq)$
by *unfold-locales* (*auto 0 4*)

lemma(**in** *antisymmetric*) *monotone-extreme-imp-extreme-bound-iff*:
fixes *ir* (**infix** $\preceq$ *50*)
assumes *f* ‘ *C* $\subseteq$ *A* **and** *monotone-on C* $(\preceq)$ $(\sqsubseteq)$ *f* **and** *i*: *extreme C* $(\preceq)$ *i*
shows *extreme-bound A* $(\sqsubseteq)$ (*f* ‘ *C*) *x* $\longleftrightarrow$ *f i* = *x*
using *dual.extreme-unique monotone-extreme-extreme-boundI*[*OF assms*] **by** *auto*

2.9 Instantiations

Finally, we instantiate our classes for sanity check.

instance *nat* :: *linorder* **..**

Pointwise ordering of functions are compatible only if the weak part is transitive.

```
instance fun :: (type,qorder) compat
proof (intro-classes, unfold-locales)
  note [simp] = le-fun-def less-fun-def
  fix f g h :: 'a ⇒ 'b
  { assume fg: f ≤ g and gh: g < h
    show f < h
    proof (unfold less-fun-def, intro conjI le-funI notI)
      from fg have f x ≤ g x for x by auto
      also from gh have g x ≤ h x for x by auto
      finally (order.trans) show f x ≤ h x for x.
      assume hf: h ≤ f
      then have h x ≤ f x for x by auto
      also from fg have f x ≤ g x for x by auto
      finally have h ≤ g by auto
      with gh show False by auto
    qed
  }
  { assume fg: f < g and gh: g ≤ h
    show f < h
    proof (unfold less-fun-def, intro conjI le-funI notI)
      from fg have f x ≤ g x for x by auto
      also from gh have g x ≤ h x for x by auto
      finally show f x ≤ h x for x.
      from gh have g x ≤ h x for x by auto
      also assume hf: h ≤ f
      then have h x ≤ f x for x by auto
      finally have g ≤ f by auto
      with fg show False by auto
    qed
  }
  show f < g ⟹ f ≤ g by auto
  show ¬f < f by auto
  show f ≤ f by auto
qed

instance fun :: (type,qorder) qorder
  apply intro-classes
  apply unfold-locales
  by (auto simp: le-fun-def dest: order.trans)

instance fun :: (type,porder) porder
  apply intro-classes
```

apply *unfold-locales*
proof (*intro ext*)
fix $f\ g :: 'a \Rightarrow 'b$ **and** $x :: 'a$
assume *fg*: $f \leq g$ **and** *gf*: $g \leq f$
then have $f\ x \leq g\ x$ **and** $g\ x \leq f\ x$ **by** (*auto elim*: *le-funE*)
from *order.antisym*[*OF this*] **show** $f\ x = g\ x$ **by** *auto*
qed

end

3 Completeness of Relations

Here we formalize various order-theoretic completeness conditions.

theory *Complete-Relations*
imports *Binary-Relations*
begin

3.1 Completeness Conditions

Order-theoretic completeness demands certain subsets of elements to admit suprema or infima.

definition *complete* (*-−complete*[*999*]*1000*) **where**
$CC{-}complete\ A\ r \equiv \forall X \subseteq A.\ X \in CC \longrightarrow (\exists s.\ extreme\text{-}bound\ A\ r\ X\ s)$

lemmas *completeI* = *complete-def*[*unfolded atomize-eq*, *THEN iffD2*, *rule-format*]
lemmas *completeD* = *complete-def*[*unfolded atomize-eq*, *THEN iffD1*, *rule-format*]

lemma *complete-cmono*: $CC \subseteq DD \Longrightarrow DD{-}complete \leq CC{-}complete$
by (*force simp*: *complete-def*)

lemma *complete-empty*[*simp*]: $CC{-}complete\ \{\}\ r \longleftrightarrow \{\} \notin CC$ **by** (*auto simp*: *complete-def*)

A related set $\langle A, \sqsubseteq \rangle$ is called *topped* if there is a "top" element $\top \in A$, a greatest element in A. Note that there might be multiple tops if $(\sqsubseteq)$ is not antisymmetric.

definition $extremed\ A\ r \equiv \exists e.\ extreme\ A\ r\ e$

lemma *extremed-imp-ex-bound*: $extremed\ A\ r \Longrightarrow X \subseteq A \Longrightarrow \exists b \in A.\ bound\ X\ r\ b$
by (*auto simp*: *extremed-def*)

context
fixes *less-eq* :: $'a \Rightarrow 'a \Rightarrow bool$ (**infix** $\sqsubseteq$ *50*)
begin

Toppedness can be also seen as a completeness condition, since it is equivalent to saying that the universe has a supremum.

lemma *extremed-iff-UNIV-complete*: *extremed A* ($\sqsubseteq$) $\longleftrightarrow$ $\{A\}-$*complete A* ($\sqsubseteq$) (**is** *?l* $\longleftrightarrow$ *?r*)
proof
assume *?l*
then obtain *e* **where** *extreme A* ($\sqsubseteq$) *e* **by** (*auto simp*: *extremed-def*)
then have *extreme-bound A* ($\sqsubseteq$) *A e* **by** *auto*
then show *?r* **by** (*auto intro*!: *completeI*)
next
assume *?r*
from *completeD*[*OF this*] **obtain** *s* **where** *extreme-bound A* ($\sqsubseteq$) *A s* **by** *auto*
then have *extreme A* ($\sqsubseteq$) *s* **by** *auto*
then show *?l* **by** (*auto simp*: *extremed-def*)
qed

The dual notion of topped is called "pointed", equivalently ensuring a supremum of the empty set.

lemma *pointed-iff-empty-complete*: *extremed A* ($\sqsubseteq$) $\longleftrightarrow$ $\{\{\}\}-$*complete A* $(\sqsubseteq)^-$
by (*auto simp*: *complete-def extremed-def*)

end

One of the most well-studied notion of completeness would be the semilattice condition: every pair of elements x and y has a supremum $x \sqcup y$ (not necessarily unique if the underlying relation is not antisymmetric).

definition *pair-complete* $\equiv \{\{x,y\} \mid. x\ y :: 'a\}-$*complete*

lemma *pair-completeI*:
assumes $\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow \exists s.$ *extreme-bound A r* $\{x,y\}$ *s*
shows *pair-complete A r*
using *assms* **by** (*auto simp*: *pair-complete-def complete-def*)

lemma *pair-completeD*:
assumes *pair-complete A r*
shows $x \in A \Longrightarrow y \in A \Longrightarrow \exists s.$ *extreme-bound A r* $\{x,y\}$ *s*
by (*intro completeD*[*OF assms*[*unfolded pair-complete-def*]], *auto*)

context
fixes *less-eq* :: $'a \Rightarrow 'a \Rightarrow$ *bool* (**infix** $\sqsubseteq$ *50*)
begin

lemma *pair-complete-imp-directed*:
assumes *comp*: *pair-complete A* ($\sqsubseteq$) **shows** *directed A* ($\sqsubseteq$)
proof
fix *x y* :: $'a$
assume $x \in A\ y \in A$
with *pair-completeD*[*OF comp this*] **show** $\exists z \in A.\ x \sqsubseteq z \wedge y \sqsubseteq z$ **by** *auto*
qed

end

lemma (**in** *connex*) *pair-complete*: *pair-complete A* (⊑)
proof (*safe intro*!: *pair-completeI*)
  fix *x y*
  assume *x*: *x* ∈ *A* **and** *y*: *y* ∈ *A*
  then show ∃ *s*. *extreme-bound A* (⊑) {*x*, *y*} *s*
  proof (*cases rule*:*comparable-cases*)
    case *le*
    with *x y* **show** *?thesis* **by** (*auto intro*!: *exI*[*of* - *y*])
  next
    case *ge*
    with *x y* **show** *?thesis* **by** (*auto intro*!: *exI*[*of* - *x*])
  qed
qed

The next one assumes that every nonempty finite set has a supremum.

abbreviation *finite-complete* ≡ {*X*. *finite X* ∧ *X* ≠ {}}−*complete*

lemma *finite-complete-le-pair-complete*: *finite-complete* ≤ *pair-complete*
  by (*unfold pair-complete-def*, *rule complete-cmono*, *auto*)

The next one assumes that every nonempty bounded set has a supremum. It is also called the Dedekind completeness.

abbreviation *conditionally-complete A r* ≡ {*X*. ∃ *b* ∈ *A*. *bound X r b* ∧ *X* ≠ {}}−*complete A r*

lemma *conditionally-complete-imp-nonempty-imp-ex-extreme-bound-iff-ex-bound*:
  assumes *comp*: *conditionally-complete A r*
  assumes *X* ⊆ *A* **and** *X* ≠ {}
  shows (∃ *s*. *extreme-bound A r X s*) ⟷ (∃ *b*∈*A*. *bound X r b*)
  using *assms* **by** (*auto 0 4 intro*!:*completeD*[*OF comp*])

The ω-completeness condition demands a supremum for an ω-chain, $a_1 \sqsubseteq a_2 \sqsubseteq \ldots$. We model ω-chain as the range of a monotone map $f : i \mapsto a_i$.

definition *omega-complete A r* ≡ {*range f* | *f* :: *nat* ⇒ *'a*. *monotone* (≤) *r f*}−*complete A r*

lemma (**in** *transitive*) *local-chain*:
  assumes *CA*: *range C* ⊆ *A*
  shows (∀ *i*::*nat*. *C i* ⊑ *C* (*Suc i*)) ⟷ *monotone* (<) (⊑) *C*
proof (*intro iffI allI monotoneI*)
  fix *i j* :: *nat*
  assume *loc*: ∀ *i*. *C i* ⊑ *C* (*Suc i*) **and** *ij*: *i* < *j*
  have *C i* ⊑ *C* (*i*+*k*+*1*) **for** *k*
  proof (*induct k*)
    case *0*
    from *loc* **show** *?case* **by** *auto*
  next

```
    case (Suc k)
    also have C (i+k+1) ⊑ C (i+k+Suc 1) using loc by auto
    finally show ?case using CA by auto
  qed
  from this[of j−i−1] ij show C i ⊑ C j by auto
next
  fix i
  assume monotone (<) (⊑) C
  then show C i ⊑ C (Suc i) by (auto dest: monotoneD)
qed
```

Directed completeness is an important notion in domain theory [1], asserting that every nonempty directed set has a supremum. Here, a set X is *directed* if any pair of two elements in X has a bound in X.

```
definition directed-complete A r ≡ {X. directed X r ∧ X ≠ {}}−complete A r

lemma monotone-directed-complete:
  assumes comp: directed-complete A r
   assumes fI: f ‘ I ⊆ A and dir: directed I ri and I0: I ≠ {} and mono:
monotone-on I ri r f
  shows ∃ s. extreme-bound A r (f ‘ I) s
  apply (rule completeD[OF comp[unfolded directed-complete-def], OF fI])
  using monotone-directed-image[OF mono dir] I0 by auto
```

The next one is quite complete, only the empty set may fail to have a supremum. The terminology follows [3], although there it is defined more generally depending on a cardinal α such that a nonempty set X of cardinality below α has a supremum.

```
abbreviation semicomplete ≡ {X. X ≠ {}}−complete

lemma semicomplete-nonempty-imp-extremed:
  semicomplete A r ⟹ A ≠ {} ⟹ extremed A r
  unfolding extremed-iff-UNIV-complete
  using complete-cmono[of {A} {X. X ≠ {}}]
  by auto

lemma connex-dual-semicomplete: semicomplete {C. connex C r} (⊇)
proof (intro completeI, elim CollectE)
  fix X
  assume X ⊆ {C. connex C r} and X ≠ {}
  then have connex (⋂ X) r by (auto simp: connex-def)
  then have extreme-bound {C. connex C r} (⊇) X (⋂ X) by auto
  then show ∃ S. extreme-bound {C. connex C r} (⊇) X S by auto
qed
```

3.2 Pointed Ones

The term ‘pointed’ refers to the dual notion of toppedness, i.e., there is a global least element. This serves as the supremum of the empty set.

lemma *complete-union*: $(CC \cup CC')-complete\ A\ r \longleftrightarrow CC-complete\ A\ r \wedge CC'-complete\ A\ r$
by (*auto simp*: *complete-def*)

lemma *pointed-directed-complete*:
$\{X.\ directed\ X\ r\}-complete\ A\ r \longleftrightarrow directed\text{-}complete\ A\ r \wedge extremed\ A\ r^-$
proof−
have [*simp*]: $\{X.\ directed\ X\ r \wedge X \neq \{\} \vee X = \{\}\} = \{X.\ directed\ X\ r\}$ **by** *auto*
show *?thesis*
by (*auto simp*: *directed-complete-def pointed-iff-empty-complete complete-union*[*symmetric*] *Un-def*)
qed

"Bounded complete" refers to pointed conditional complete, but this notion is just the dual of semicompleteness. We prove this later.

abbreviation *bounded-complete* $A\ r \equiv \{X.\ \exists\, b{\in}A.\ bound\ X\ r\ b\}-complete\ A\ r$

3.3 Relations between Completeness Conditions

context
fixes *less-eq* :: $'a \Rightarrow 'a \Rightarrow bool$ (**infix** $\sqsubseteq$ *50*)
begin

interpretation *less-eq-notations*.

Pair-completeness implies that the universe is directed. Thus, with directed completeness implies toppedness.

proposition *directed-complete-pair-complete-imp-extremed*:
assumes *dc*: *directed-complete A* ($\sqsubseteq$) **and** *pc*: *pair-complete A* ($\sqsubseteq$) **and** *A*: $A \neq \{\}$
shows *extremed A* ($\sqsubseteq$)
proof−
have $\exists\, s.$ *extreme-bound A* ($\sqsubseteq$) *A s*
apply (*rule completeD*[*OF dc*[*unfolded directed-complete-def*]])
using *pair-complete-imp-directed*[*OF pc*] *A* **by** *auto*
then obtain *t* **where** *extreme-bound A* ($\sqsubseteq$) *A t* **by** *auto*
then have $\forall\, x \in A.\ x \sqsubseteq t$ **and** $t \in A$ **by** *auto*
then show *?thesis* **by** (*auto simp*: *extremed-def*)
qed

Semicomplete is conditional complete and topped.

proposition *semicomplete-iff-conditionally-complete-extremed*:
assumes *A*: $A \neq \{\}$
shows *semicomplete A* ($\sqsubseteq$) $\longleftrightarrow$ *conditionally-complete A* ($\sqsubseteq$) $\wedge$ *extremed A* ($\sqsubseteq$)
(**is** *?l* $\longleftrightarrow$ *?r*)
proof (*intro iffI*)
assume *r*: *?r*
then have *cc*: *conditionally-complete A* ($\sqsubseteq$) **and** *e*: *extremed A* ($\sqsubseteq$) **by** *auto*
show *?l*

```
  proof (intro completeI, unfold mem-Collect-eq)
    fix X
    assume X: X ⊆ A and X ≠ {}
    with extremed-imp-ex-bound[OF e X]
    show ∃ s. extreme-bound A (⊑) X s by (auto intro!: completeD[OF cc X])
  qed
next
  assume l: ?l
  with semicomplete-nonempty-imp-extremed[OF l] A
  show ?r by (auto intro!: completeI dest: completeD)
qed

proposition complete-iff-pointed-semicomplete:
  UNIV−complete A (⊑) ⟷ semicomplete A (⊑) ∧ extremed A (⊒) (is ?l ⟷
?r)
  by (unfold pointed-iff-empty-complete complete-union[symmetric] Un-def, auto)
```

Conditional completeness only lacks top and bottom to be complete.

```
proposition complete-iff-conditionally-complete-extremed-pointed:
  UNIV−complete A (⊑) ⟷ conditionally-complete A (⊑) ∧ extremed A (⊑) ∧
extremed A (⊒)
  unfolding complete-iff-pointed-semicomplete
  apply (cases A = {})
   apply (auto intro!: completeI dest: extremed-imp-ex-bound)[1]
  unfolding semicomplete-iff-conditionally-complete-extremed
  apply (auto intro:completeI)
  done
```

If the universe is directed, then every pair is bounded, and thus has a supremum. On the other hand, supremum gives an upper bound, witnessing directedness.

```
proposition conditionally-complete-imp-pair-complete-iff-directed:
  assumes comp: conditionally-complete A (⊑)
  shows pair-complete A (⊑) ⟷ directed A (⊑) (is ?l ⟷ ?r)
proof(intro iffI)
  assume ?r
  then show ?l
    by (auto intro!: pair-completeI completeD[OF comp] elim: directedE)
next
  assume pc: ?l
  show ?r
  proof (intro directedI)
    fix x y assume x ∈ A and y ∈ A
      with pc obtain z where extreme-bound A (⊑) {x,y} z by (auto dest:
pair-completeD)
    then show ∃ z∈A. x ⊑ z ∧ y ⊑ z by auto
  qed
qed
```

end

Following is a new generalization of (weak) chain-completeness:

abbreviation *well-complete A r* ≡ {*X*. *well-related-set X r*}−*complete A r*

3.4 Duality of Completeness Conditions

Conditional completeness is symmetric.

context fixes *less-eq* :: *'a* ⇒ *'a* ⇒ *bool* (**infix** ⊑ *50*)
begin

interpretation *less-eq-notations***.**

lemma *conditionally-complete-dual*:
assumes *comp*: *conditionally-complete A* (⊑) **shows** *conditionally-complete A* (⊒)
proof (*intro completeI*, *elim CollectE*)
fix *X* **assume** *XA*: *X* ⊆ *A*
define *B* **where** [*simp*]: *B* ≡ {*b*∈*A*. *bound X* (⊒) *b*}
assume *bound*: ∃ *b*∈*A*. *bound X* (⊒) *b* ∧ *X* ≠ {}
with *in-mono*[*OF XA*] **have** *B*: *B* ⊆ *A* **and** ∃ *b* ∈ *A*. *bound B* (⊑) *b* **and** *B* ≠ {} **by** *auto*
from *comp*[*THEN completeD*, *OF B*] *this*
obtain *s* **where** *s* ∈ *A* **and** *extreme-bound A* (⊑) *B s* **by** *auto*
with *in-mono*[*OF XA*] **show** ∃ *s*. *extreme-bound A* (⊒) *X s* **by** (*intro exI*[*of - s*] *extremeI*, *auto*)
qed

Full completeness is symmetric.

lemma *complete-dual*:
UNIV−*complete A* (⊑) ⟹ *UNIV*−*complete A* (⊒)
apply (*unfold complete-iff-conditionally-complete-extremed-pointed*)
using *conditionally-complete-dual* **by** *auto*

Now we show that bounded completeness is the dual of semicompleteness.

lemma *pointed-conditionally-complete-iff-bounded-complete*:
assumes *A*: *A* ≠ {}
shows *conditionally-complete A* (⊑) ∧ *extremed A* (⊒) ⟷ *bounded-complete A* (⊑)
apply (*unfold pointed-iff-empty-complete*)
apply (*fold complete-union*)
apply (*unfold Un-def*)
apply (*rule arg-cong*[*of - - λCC*. *CC*−*complete A* (⊑)])
using *A* **by** *auto*

proposition *bounded-complete-iff-dual-semicomplete*:
bounded-complete A (⊑) ⟷ *semicomplete A* (⊒)
proof (*cases A* = {})

case *True*
then show *?thesis* **by** *auto*
next
case *False*
then show *?thesis*
apply (*fold pointed-conditionally-complete-iff-bounded-complete*[*OF False*])
apply (*unfold semicomplete-iff-conditionally-complete-extremed*)
using *Complete-Relations.conditionally-complete-dual* **by** *auto*
qed

end

lemmas *connex-bounded-complete* = *connex-dual-semicomplete*[*folded bounded-complete-iff-dual-semicomplete*]

3.5 Completeness Results Requiring Order-Like Properties

Above results hold without any assumption on the relation. This part demands some order-like properties.

It is well known that in a semilattice, i.e., a pair-complete partial order, every finite nonempty subset of elements has a supremum. We prove the result assuming transitivity, but only that.

lemma (**in** *transitive*) *pair-complete-iff-finite-complete*:
pair-complete A (⊑) ⟷ *finite-complete A* (⊑) (**is** *?l* ⟷ *?r*)
proof (*intro iffI completeI*, *elim CollectE conjE*)
fix *X*
assume *pc*: *?l*
show *finite X* ⟹ *X* ⊆ *A* ⟹ *X* ≠ {} ⟹ *Ex* (*extreme-bound A* (⊑) *X*)
proof (*induct X rule*: *finite-induct*)
case *empty*
then show *?case* **by** *auto*
next
case (*insert x X*)
then have *x*: *x* ∈ *A* **and** *X*: *X* ⊆ *A* **by** *auto*
show *?case*
proof (*cases X* = {})
case *True*
obtain *x′* **where** *extreme-bound A* (⊑) {*x*,*x*} *x′* **using** *pc*[*THEN pair-completeD*, *OF x x*] **by** *auto*
with *True* **show** *?thesis* **by** (*auto intro*!: *exI*[*of* - *x′*])
next
case *False*
with *insert* **obtain** *b* **where** *b*: *extreme-bound A* (⊑) *X b* **by** *auto*
with *pc*[*THEN pair-completeD*] *x* **obtain** *c* **where** *c*: *extreme-bound A* (⊑) {*x*,*b*} *c* **by** *auto*
from *c* **have** *cA*: *c* ∈ *A* **and** *xc*: *x* ⊑ *c* **and** *bc*: *b* ⊑ *c* **by** *auto*
from *b* **have** *bA*: *b* ∈ *A* **and** *bX*: *bound X* (⊑) *b* **by** *auto*
show *?thesis*
proof (*intro exI extreme-boundI*)

```
      fix xb assume xb: xb ∈ insert x X
      from bound-trans[OF bX bc X bA cA] have bound X (⊑) c.
      with xb xc show xb ⊑ c by auto
    next
      fix d assume bound (insert x X) (⊑) d and dA: d ∈ A
      with b have bound {x,b} (⊑) d by auto
      with c show c ⊑ d using dA by auto
    qed (fact cA)
  qed
 qed
qed (insert finite-complete-le-pair-complete, auto)
```

Gierz et al. [6] showed that a directed complete partial order is semicomplete if and only if it is also a semilattice. We generalize the claim so that the underlying relation is only transitive.

```
proposition(in transitive) semicomplete-iff-directed-complete-pair-complete:
  semicomplete A (⊑) ⟷ directed-complete A (⊑) ∧ pair-complete A (⊑) (is ?l
⟷ ?r)
proof (intro iffI)
 assume l: ?l
 then show ?r by (auto simp: directed-complete-def intro!: completeI pair-completeI
completeD[OF l])
next
 assume ?r
 then have dc: directed-complete A (⊑) and pc: pair-complete A (⊑) by auto
 with pair-complete-iff-finite-complete have fc: finite-complete A (⊑) by auto
 show ?l
 proof (intro completeI, elim CollectE)
  fix X assume XA: X ⊆ A
  have 1: directed {x. ∃ Y ⊆ X. finite Y ∧ Y ≠ {} ∧ extreme-bound A (⊑) Y x}
(⊑) (is directed ?B -)
   proof (intro directedI)
    fix a b assume a: a ∈ ?B and b: b ∈ ?B
    from a obtain Y where Y: extreme-bound A (⊑) Y a finite Y Y ≠ {} Y ⊆
X by auto
    from b obtain B where B: extreme-bound A (⊑) B b finite B B ≠ {} B ⊆
X by auto
    from XA Y B have AB: Y ⊆ A B ⊆ A finite (Y ∪ B) Y ∪ B ≠ {} Y ∪ B
⊆ X by auto
    with fc[THEN completeD] have Ex (extreme-bound A (⊑) (Y ∪ B)) by auto
    then obtain c where c: extreme-bound A (⊑) (Y ∪ B) c by auto
    show ∃ c ∈ ?B. a ⊑ c ∧ b ⊑ c
    proof (intro bexI conjI)
     from Y B c show a ⊑ c and b ⊑ c by (auto simp: extreme-bound-iff)
     from AB c show c ∈ ?B by (auto intro!: exI[of - Y ∪ B])
    qed
   qed
   have B: ?B ⊆ A by auto
   assume X ≠ {}
```

then obtain x **where** xX: $x \in X$ **by** *auto*
with *fc*[*THEN completeD*, *of* $\{x\}$] *XA*
obtain x' **where** *extreme-bound A* ($\sqsubseteq$) $\{x\}$ x' **by** *auto*
with xX **have** $x'B$: $x' \in$ *?B* **by** (*auto intro*!: *exI*[*of* - $\{x\}$] *extreme-boundI*)
then have *2*: *?B* $\neq \{\}$ **by** *auto*
from *dc*[*unfolded directed-complete-def*, *THEN completeD*, *of ?B*] *1 2*
obtain b **where** b: *extreme-bound A* ($\sqsubseteq$) *?B b* **by** *auto*
then have bA: $b \in A$ **by** *auto*
show *Ex* (*extreme-bound A* ($\sqsubseteq$) *X*)
proof (*intro exI extreme-boundI UNIV-I*)
fix x
assume xX: $x \in X$
with *XA fc*[*THEN completeD*, *of* $\{x\}$]
obtain c **where** c: *extreme-bound A* ($\sqsubseteq$) $\{x\}$ c **by** *auto*
then have cA: $c \in A$ **and** xc: $x \sqsubseteq c$ **by** *auto*
from *c xX* **have** *cB*: $c \in$ *?B* **by** (*auto intro*!: *exI*[*of* - $\{x\}$] *extreme-boundI*)
with b **have** bA: $b \in A$ **and** cb: $c \sqsubseteq b$ **by** *auto*
from *xX XA cA bA trans*[*OF xc cb*]
show $x \sqsubseteq b$ **by** *auto*

Here transitivity is needed.

next
fix x
assume xA: $x \in A$ **and** Xx: *bound X* ($\sqsubseteq$) x
have *bound ?B* ($\sqsubseteq$) x
proof (*intro boundI UNIV-I*, *clarify*)
fix c Y
assume *finite* Y **and** YX: $Y \subseteq X$ **and** $Y \neq \{\}$ **and** c: *extreme-bound A* ($\sqsubseteq$) Y c
from *YX Xx* **have** *bound Y* ($\sqsubseteq$) x **by** *auto*
with *c xA* **show** $c \sqsubseteq x$ **by** *auto*
qed
with b *xA* **show** $b \sqsubseteq x$ **by** *auto*
qed (*fact bA*)
qed
qed

The last argument in the above proof requires transitivity, but if we had reflexivity then x itself is a supremum of $\{x\}$ (see $[\![$*reflexive ?A ?less-eq*; *?x* $\in$ *?A*$]\!] \Longrightarrow$ *extreme-bound ?A ?less-eq* $\{$*?x*$\}$ *?x*) and so $x \sqsubseteq s$ would be immediate. Thus we can replace transitivity by reflexivity, but then pair-completeness does not imply finite completeness. We obtain the following result.

proposition (**in** *reflexive*) *semicomplete-iff-directed-complete-finite-complete*:
semicomplete A ($\sqsubseteq$) $\longleftrightarrow$ *directed-complete A* ($\sqsubseteq$) $\wedge$ *finite-complete A* ($\sqsubseteq$) (**is** *?l* $\longleftrightarrow$ *?r*)
proof (*intro iffI*)
assume *l*: *?l*

then show *?r* **by** (*auto simp*: *directed-complete-def intro*!: *completeI pair-completeI completeD*[*OF l*])
next
assume *?r*
then have *dc*: *directed-complete A* (⊑) **and** *fc*: *finite-complete A* (⊑) **by** *auto*
show *?l*
proof (*intro completeI*, *elim CollectE*)
fix *X* **assume** *XA*: $X \subseteq A$
have *1*: *directed* {*x*. ∃ *Y* ⊆ *X*. *finite Y* ∧ $Y \neq \{\}$ ∧ *extreme-bound A* (⊑) *Y x*} (⊑) (**is** *directed ?B* -)
proof (*intro directedI*)
fix *a b* **assume** *a*: $a \in ?B$ **and** *b*: $b \in ?B$
from *a* **obtain** *Y* **where** *Y*: *extreme-bound A* (⊑) *Y a finite Y Y* ≠ {} *Y* ⊆ *X* **by** *auto*
from *b* **obtain** *B* **where** *B*: *extreme-bound A* (⊑) *B b finite B B* ≠ {} *B* ⊆ *X* **by** *auto*
from *XA Y B* **have** *AB*: *Y* ⊆ *A B* ⊆ *A finite* (*Y* ∪ *B*) *Y* ∪ *B* ≠ {} *Y* ∪ *B* ⊆ *X* **by** *auto*
with *fc*[*THEN completeD*] **have** *Ex* (*extreme-bound A* (⊑) (*Y* ∪ *B*)) **by** *auto*
then obtain *c* **where** *c*: *extreme-bound A* (⊑) (*Y* ∪ *B*) *c* **by** *auto*
show ∃ *c* ∈ *?B*. *a* ⊑ *c* ∧ *b* ⊑ *c*
proof (*intro bexI conjI*)
from *Y B c* **show** *a* ⊑ *c* **and** *b* ⊑ *c* **by** (*auto simp*: *extreme-bound-iff*)
from *AB c* **show** *c* ∈ *?B* **by** (*auto intro*!: *exI*[*of* - *Y* ∪ *B*])
qed
qed
have *B*: $?B \subseteq A$ **by** *auto*
assume $X \neq \{\}$
then obtain *x* **where** *xX*: $x \in X$ **by** *auto*
with *XA* **have** *extreme-bound A* (⊑) {*x*} *x*
by (*intro extreme-bound-singleton*, *auto*)
with *xX* **have** *xB*: $x \in ?B$ **by** (*auto intro*!: *exI*[*of* - {*x*}])
then have *2*: $?B \neq \{\}$ **by** *auto*
from *dc*[*unfolded directed-complete-def*, *THEN completeD*, *of ?B*] *B 1 2*
obtain *b* **where** *b*: *extreme-bound A* (⊑) *?B b* **by** *auto*
then have *bA*: $b \in A$ **by** *auto*
show *Ex* (*extreme-bound A* (⊑) *X*)
proof (*intro exI extreme-boundI UNIV-I*)
fix *x*
assume *xX*: $x \in X$
with *XA* **have** *x*: *extreme-bound A* (⊑) {*x*} *x* **by** (*intro extreme-bound-singleton*, *auto*)
from *x xX* **have** *cB*: $x \in ?B$ **by** (*auto intro*!: *exI*[*of* - {*x*}])
with *b* **show** *x* ⊑ *b* **by** *auto*
next
fix *x*
assume *Xx*: *bound X* (⊑) *x* **and** *xA*: $x \in A$
have *bound ?B* (⊑) *x*
proof (*intro boundI UNIV-I*, *clarify*)

fix c Y
assume *finite* Y **and** YX: $Y \subseteq X$ **and** $Y \neq \{\}$ **and** c: *extreme-bound* A $(\sqsubseteq)$ Y c
from YX Xx **have** *bound* Y $(\sqsubseteq)$ x **by** *auto*
with YX XA xA c **show** $c \sqsubseteq x$ **by** *auto*
qed
with xA b **show** $b \sqsubseteq x$ **by** *auto*
qed (*fact* bA)
qed
qed

3.6 Relating to Classes

Isabelle's class *complete-lattice* is $UNIV-complete$.

lemma (**in** *complete-lattice*) $UNIV-complete$ $UNIV$ $(\leq)$
by (*auto intro*!: *completeI Sup-upper Sup-least Inf-lower Inf-greatest*)

3.7 Set-wise Completeness

lemma *directed-sets-directed-complete*:
assumes *cl*: $\forall DC.\ DC \subseteq AA \longrightarrow (\forall X{\in}DC.\ directed\ X\ r) \longrightarrow (\bigcup DC) \in AA$
shows $\{DC.\ directed\ DC\ (\subseteq)\}-complete\ \{X \in AA.\ directed\ X\ r\}\ (\subseteq)$
proof (*intro completeI*, *safe*)
fix XX
assume *ch*: $XX \subseteq \{X \in AA.\ directed\ X\ r\}$ **and** *dir*: $directed\ XX\ (\subseteq)$
with *cl* **have** $(\bigcup XX) \in AA$ **by** *auto*
moreover have *directed* $(\bigcup XX)$ r
proof (*intro directedI*, *elim UnionE*)
fix x y X Y **assume** xX: $x \in X$ **and** X: $X \in XX$ **and** yY: $y \in Y$ **and** Y: $Y \in XX$
from *directedD*[*OF dir X Y*]
obtain Z **where** $X \subseteq Z$ $Y \subseteq Z$ **and** Z: $Z \in XX$ **by** *auto*
with *ch* xX yY **have** *directed* Z r $x \in Z$ $y \in Z$ **by** *auto*
then obtain z **where** $z \in Z$ $r\ x\ z \wedge r\ y\ z$ **by** (*auto elim*:*directedE*)
with Z **show** $\exists z{\in}\bigcup XX.\ r\ x\ z \wedge r\ y\ z$ **by** *auto*
qed
ultimately have *extreme-bound* $\{X \in AA.\ directed\ X\ r\}$ $(\subseteq)$ XX $(\bigcup XX)$ **by** *auto*
then show *Ex* (*extreme-bound* $\{X \in AA.\ directed\ X\ r\}$ $(\subseteq)$ XX) **by** *auto*
qed

lemma *connex-directed-Un*:
assumes *ch*: $CC \subseteq \{C.\ connex\ C\ r\}$ **and** *dir*: $directed\ CC\ (\subseteq)$
shows *connex* $(\bigcup CC)$ r
proof (*intro connexI*, *elim UnionE*)
fix x y X Y **assume** xX: $x \in X$ **and** X: $X \in CC$ **and** yY: $y \in Y$ **and** Y: $Y \in CC$
from *directedD*[*OF dir X Y*]
obtain Z **where** $X \subseteq Z$ $Y \subseteq Z$ $Z \in CC$ **by** *auto*

with *xX yY ch* **have** $x \in Z$ $y \in Z$ *connex Z r* **by** *auto*
then show *r x y* $\vee$ *r y x* **by** (*auto elim*:*connexE*)
qed

lemma *connex-directed-complete*: {*DC*. *directed DC* ($\subseteq$)}−*complete* {*C*. *connex C r*} ($\subseteq$)
by (*intro completeI*, *force dest*!: *connex-directed-Un*)

end

theory *Fixed-Points*
imports *Complete-Relations*
begin

4 Existence of Fixed Points in Complete Related Sets

The following proof is simplified and generalized from Stouti–Maaden [16]. We construct some set whose extreme bounds – if they exist, typically when the underlying related set is complete – are fixed points of a monotone or inflationary function on any related set. When the related set is attractive, those are actually the least fixed points. This generalizes [16], relaxing reflexivity and antisymmetry.

locale *fixed-point-proof* = *related-set* +
fixes *f*
assumes *f*: *f* ‘ *A* $\subseteq$ *A*
begin

sublocale *less-eq-notations*.

definition *AA* **where** *AA* $\equiv$
{*X*. *X* $\subseteq$ *A* $\wedge$ *f* ‘ *X* $\subseteq$ *X* $\wedge$ ($\forall$ *Y s*. *Y* $\subseteq$ *X* $\longrightarrow$ *extreme-bound A* ($\sqsubseteq$) *Y s* $\longrightarrow$ *s* $\in$ *X*)}

lemma *AA-I*:
X $\subseteq$ *A* $\Longrightarrow$ *f* ‘ *X* $\subseteq$ *X* $\Longrightarrow$ ($\bigwedge$ *Y s*. *Y* $\subseteq$ *X* $\Longrightarrow$ *extreme-bound A* ($\sqsubseteq$) *Y s* $\Longrightarrow$ *s* $\in$ *X*) $\Longrightarrow$ *X* $\in$ *AA*
by (*unfold AA-def*, *safe*)

lemma *AA-E*:
X $\in$ *AA* $\Longrightarrow$
(*X* $\subseteq$ *A* $\Longrightarrow$ *f* ‘ *X* $\subseteq$ *X* $\Longrightarrow$ ($\bigwedge$ *Y s*. *Y* $\subseteq$ *X* $\Longrightarrow$ *extreme-bound A* ($\sqsubseteq$) *Y s* $\Longrightarrow$ *s* $\in$ *X*) $\Longrightarrow$ *thesis*) $\Longrightarrow$ *thesis*
by (*auto simp*: *AA-def*)

definition *C* **where** *C* $\equiv \bigcap$ *AA*

lemma *A-AA*: $A \in AA$ **by** (*auto intro*!:*AA-I f*)

lemma *C-AA*: $C \in AA$
proof (*intro AA-I*)
  show $C \subseteq A$ **using** *C-def A-AA f* **by** *auto*
  show $f \mathbin{`} C \subseteq C$ **unfolding** *C-def AA-def* **by** *auto*
  fix B b **assume** B: $B \subseteq C$ *extreme-bound A* ($\sqsubseteq$) B b
  { fix X **assume** X: $X \in AA$
    with B **have** $B \subseteq X$ **by** (*auto simp*: *C-def*)
    with X B **have** $b \in X$ **by** (*auto elim*!: *AA-E*)
  }
  then show $b \in C$ **by** (*auto simp*: *C-def AA-def*)
qed

lemma *CA*: $C \subseteq A$ **using** *A-AA* **by** (*auto simp*: *C-def*)

lemma *fC*: $f \mathbin{`} C \subseteq C$ **using** *C-AA* **by** (*auto elim*!: *AA-E*)

context
  fixes c **assumes** *Cc*: *extreme-bound A* ($\sqsubseteq$) C c
begin

private lemma *cA*: $c \in A$ **using** *Cc* **by** *auto*
private lemma *cC*: $c \in C$ **using** *Cc C-AA* **by** (*blast elim*!:*AA-E*)
private lemma *fcC*: $f\ c \in C$ **using** *cC AA-def C-AA* **by** *auto*
private lemma *fcA*: $f\ c \in A$ **using** *fcC CA* **by** *auto*

lemma *qfp-as-extreme-bound*:
  assumes *infl-mono*: $\forall x \in A.\ x \sqsubseteq f\ x \vee (\forall y \in A.\ y \sqsubseteq x \longrightarrow f\ y \sqsubseteq f\ x)$
  shows $f\ c \sim c$
proof (*intro conjI bexI sympartpI*)
  show $f\ c \sqsubseteq c$ **using** *fcC Cc* **by** *auto*
  from *infl-mono*[*rule-format*, *OF cA*]
  show $c \sqsubseteq f\ c$
  proof (*safe*)

Monotone case:

    assume *mono*: $\forall b \in A.\ b \sqsubseteq c \longrightarrow f\ b \sqsubseteq f\ c$
    define D **where** $D \equiv \{x \in C.\ x \sqsubseteq f\ c\}$
    have $D \in AA$
    proof (*intro AA-I*)
      show $D \subseteq A$ **unfolding** *D-def C-def* **using** *A-AA f* **by** *auto*
      have *fxC*: $x \in C \Longrightarrow x \sqsubseteq f\ c \Longrightarrow f\ x \in C$ **for** x **using** *C-AA* **by** (*auto simp*: *AA-def*)
      show $f \mathbin{`} D \subseteq D$
      proof (*unfold D-def*, *safe intro*!: *fxC*)
        fix x **assume** *xC*: $x \in C$
        have $x \sqsubseteq c\ x \in A$ **using** *Cc xC CA* **by** *auto*

```
      then show f x ⊑ f c using mono by (auto dest:monotoneD)
    qed
    have DC: D ⊆ C unfolding D-def by auto
    fix B b assume BD: B ⊆ D and Bb: extreme-bound A (⊑) B b
    have B ⊆ C using DC BD by auto
    then have bC: b ∈ C using C-AA Bb BD by (auto elim!: AA-E)
    have bfc: ∀ a∈B. a ⊑ f c using BD unfolding D-def by auto
    with f cA Bb
    have b ⊑ f c by (auto simp: extreme-def image-subset-iff)
    with bC show b ∈ D unfolding D-def by auto
  qed
  then have C ⊆ D unfolding C-def by auto
  then show c ⊑ f c using cC unfolding D-def by auto
 qed
qed

lemma extreme-qfp:
 assumes attract: ∀ q ∈ A. ∀ x ∈ A. f q ∼ q ⟶ x ⊑ f q ⟶ x ⊑ q
   and mono: monotone-on A (⊑) (⊑) f
 shows extreme {q ∈ A. f q ∼ q ∨ f q = q} (⊒) c
proof−
 have fcc: f c ∼ c
   apply (rule qfp-as-extreme-bound)
   using mono by (auto elim!: monotone-onE)
 define L where [simp]: L ≡ {a ∈ A. ∀ s ∈ A. (f s ∼ s ∨ f s = s) ⟶ a ⊑ s}
 have L ∈ AA
 proof (unfold AA-def, intro CollectI conjI allI impI)
   show XA: L ⊆ A by auto
   show f ‘ L ⊆ L
   proof safe
     fix x assume xL: x ∈ L
     show f x ∈ L
     proof (unfold L-def, safe)
       have xA: x ∈ A using xL by auto
       then show fxA: f x ∈ A using f by auto
       { fix s assume sA: s ∈ A and sf: f s ∼ s ∨ f s = s
         then have x ⊑ s using xL sA sf by auto
       then have f x ⊑ f s using mono fxA sA xA by (auto elim!:monotone-onE)}
       note fxfs = this
       { fix s assume sA: s ∈ A and sf: f s ∼ s
           then show f x ⊑ s using fxfs attract mono sf fxA sA xA by (auto
elim!:monotone-onE)
       }
       { fix s assume sA: s ∈ A and sf: f s = s
         with fxfs[OF sA] show f x ⊑ s by simp}
     qed
   qed
   fix B b assume BL: B ⊆ L and b: extreme-bound A (⊑) B b
   then have BA: B ⊆ A by auto
```

```
    with BL b have bA: b ∈ A by auto
    show b ∈ L
    proof (unfold L-def, safe intro!: bA)
      { fix s assume sA: s ∈ A and sf: f s ∼ s ∨ f s = s
        have bound B (⊑) s using sA BL b sf by auto
      }
      note Bs = this
      { fix s assume sA: s ∈ A and sf: f s ∼ s
        with b sA Bs show b ⊑ s by auto
      }
      { fix s assume sA: s ∈ A and sf: f s = s
        with b sA Bs show b ⊑ s by auto
      }
    qed
  qed
  then have C ⊆ L by (simp add: C-def Inf-lower)
  with cC have c ∈ L by auto
  with L-def fcc
  show ?thesis by auto
qed

end

lemma ex-qfp:
  assumes comp: CC−complete A (⊑) and C: C ∈ CC
    and infl-mono: ∀ a ∈ A. a ⊑ f a ∨ (∀ b ∈ A. b ⊑ a ⟶ f b ⊑ f a)
  shows ∃ s ∈ A. f s ∼ s
  using qfp-as-extreme-bound[OF - infl-mono] completeD[OF comp CA C] by auto

lemma ex-extreme-qfp-fp:
  assumes comp: CC−complete A (⊑) and C: C ∈ CC
    and attract: ∀ q ∈ A. ∀ x ∈ A. f q ∼ q ⟶ x ⊑ f q ⟶ x ⊑ q
    and mono: monotone-on A (⊑) (⊑) f
  shows ∃ c. extreme {q ∈ A. f q ∼ q ∨ f q = q} (⊒) c
  using extreme-qfp[OF - attract mono] completeD[OF comp CA C] by auto

lemma ex-extreme-qfp:
  assumes comp: CC−complete A (⊑) and C: C ∈ CC
    and attract: ∀ q ∈ A. ∀ x ∈ A. f q ∼ q ⟶ x ⊑ f q ⟶ x ⊑ q
    and mono: monotone-on A (⊑) (⊑) f
  shows ∃ c. extreme {q ∈ A. f q ∼ q} (⊒) c
proof−
  from completeD[OF comp CA C]
  obtain c where Cc: extreme-bound A (⊑) C c by auto
  from extreme-qfp[OF Cc attract mono]
  have Qc: bound {q ∈ A. f q ∼ q} (⊒) c by auto
  have fcc: f c ∼ c
    apply (rule qfp-as-extreme-bound[OF Cc])
    using mono by (auto simp: monotone-onD)
```

from *Cc CA* **have** *cA*: $c \in A$ **by** *auto*
from *Qc fcc cA* **show** *?thesis* **by** (*auto intro*!: *exI*[*of* - *c*])
qed

end

context
fixes *less-eq* :: $'a \Rightarrow 'a \Rightarrow$ *bool* (**infix** $\sqsubseteq$ *50*) **and** A :: $'a$ *set* **and** f
assumes *f*: f ' $A \subseteq A$
begin

interpretation *less-eq-notations*.
interpretation *fixed-point-proof* A ($\sqsubseteq$) f **using** f **by** *unfold-locales*

theorem *complete-infl-mono-imp-ex-qfp*:
assumes *comp*: *UNIV*−*complete* A ($\sqsubseteq$) **and** *infl-mono*: $\forall\, a{\in}A.\ a \sqsubseteq f\ a \vee (\forall\, b{\in}A.\ b \sqsubseteq a \longrightarrow f\ b \sqsubseteq f\ a)$
shows $\exists\, s{\in}A.\ f\ s \sim s$
apply (*rule ex-qfp*[*OF comp* - *infl-mono*]) **by** *auto*

end

corollary (**in** *antisymmetric*) *complete-infl-mono-imp-ex-fp*:
assumes *comp*: *UNIV*−*complete* A ($\sqsubseteq$) **and** *f*: f ' $A \subseteq A$
and *infl-mono*: $\forall\, a{\in}A.\ a \sqsubseteq f\ a \vee (\forall\, b{\in}A.\ b \sqsubseteq a \longrightarrow f\ b \sqsubseteq f\ a)$
shows $\exists\, s \in A.\ f\ s = s$
proof−
interpret *less-eq-notations*.
from *complete-infl-mono-imp-ex-qfp*[*OF f comp infl-mono*]
obtain *s* **where** *sA*: $s \in A$ **and** *fss*: $f\ s \sim s$ **by** *auto*
from *f sA* **have** *fsA*: $f\ s \in A$ **by** *auto*
have $f\ s = s$ **using** *antisym fsA sA fss* **by** *auto*
with *sA* **show** *?thesis* **by** *auto*
qed

context *semiattractive* **begin**

interpretation *less-eq-notations*.

theorem *complete-mono-imp-ex-extreme-qfp*:
assumes *comp*: *UNIV*−*complete* A ($\sqsubseteq$) **and** *f*: f ' $A \subseteq A$
and *mono*: *monotone-on* A ($\sqsubseteq$) ($\sqsubseteq$) f
shows $\exists\, s.$ *extreme* $\{p \in A.\ f\ p \sim p\}$ ($\sqsubseteq$) s
proof−
interpret *dual*: *fixed-point-proof* A ($\sqsupseteq$) **rewrites** *dual.sym* = ($\sim$)
using f **by** *unfold-locales* (*auto intro*!:*ext*)
show *?thesis*
apply (*rule dual.ex-extreme-qfp*[*OF complete-dual*[*OF comp*] - - *monotone-on-dual*[*OF mono*]])

apply *simp*
using *f sym-order-trans* **by** *blast*
qed

end

corollary (**in** *antisymmetric*) *complete-mono-imp-ex-extreme-fp*:
assumes *comp*: *UNIV*−*complete A* (⊑) **and** *f*: *f* ‘ *A* ⊆ *A*
and *mono*: *monotone-on A* (⊑) (⊑) *f*
shows ∃ *s*. *extreme* {*s* ∈ *A*. *f s* = *s*} (⊑)$^-$ *s*
proof−
interpret *less-eq-notations*.
interpret *fixed-point-proof A* (⊑) *f* **using** *f* **by** *unfold-locales*
have ∃ *c*. *extreme* {*q* ∈ *A*. *f q* ∼ *q* ∨ *f q* = *q*} (⊒) *c*
apply (*rule ex-extreme-qfp-fp*[*OF comp* - - *mono*])
using *antisym f* **by** (*auto dest*: *order-sym-trans*)
then obtain *c* **where** *c*: *extreme* {*q* ∈ *A*. *f q* ∼ *q* ∨ *f q* = *q*} (⊒) *c* **by** *auto*
then have *f c* = *c* **using** *antisym f* **by** *blast*
with *c* **have** *extreme* {*q* ∈ *A*. *f q* = *q*} (⊒) *c* **by** *auto*
then show *?thesis* **by** *auto*
qed

5 Fixed Points in Well-Complete Antisymmetric Sets

In this section, we prove that an inflationary or monotone map over a well-complete antisymmetric set has a fixed point.

In order to formalize such a theorem in Isabelle, we followed Grall's [**?**] elementary proof for Bourbaki–Witt and Markowsky's theorems. His idea is to consider well-founded derivation trees over A, where from a set $C \subseteq A$ of premises one can derive $f\,(\bigsqcup C)$ if C is a chain. The main observation is as follows: Let D be the set of all the derivable elements; that is, for each $d \in D$ there exists a well-founded derivation whose root is d. It is shown that D is a chain, and hence one can build a derivation yielding $f\,(\bigsqcup D)$, and $f\,(\bigsqcup D)$ is shown to be a fixed point.

lemma *bound-monotone-on*:
assumes *mono*: *monotone-on A r s f* **and** *XA*: *X* ⊆ *A* **and** *aA*: *a* ∈ *A* **and** *rXa*: *bound X r a*
shows *bound* (*f*‘*X*) *s* (*f a*)
proof (*safe*)
fix *x* **assume** *xX*: *x* ∈ *X*
from *rXa xX* **have** *r x a* **by** *auto*
with *xX XA mono aA* **show** *s* (*f x*) (*f a*) **by** (*auto elim*!:*monotone-onE*)
qed

context *fixed-point-proof* **begin**

To avoid the usage of the axiom of choice, we carefully define derivations so that any derivable element determines its lower set. This led to the following definition:

definition *derivation* $X \equiv X \subseteq A \wedge$ *well-ordered-set* X $(\sqsubseteq) \wedge$
$(\forall x \in X.$ *let* $Y = \{y \in X.\ y \sqsubset x\}$ *in*
$(\exists y.$ *extreme* Y $(\sqsubseteq)$ $y \wedge x = f\ y) \vee$
f ‘ $Y \subseteq Y \wedge$ *extreme-bound* A $(\sqsubseteq)$ Y $x)$

lemma assumes *derivation* P
shows *derivation-A*: $P \subseteq A$ **and** *derivation-well-ordered*: *well-ordered-set* P $(\sqsubseteq)$
using *assms* **by** (*auto simp*: *derivation-def*)

lemma *derivation-cases*[*consumes 2*, *case-names suc lim*]:
assumes *derivation* X **and** $x \in X$
and $\bigwedge Y\ y.\ Y = \{y \in X.\ y \sqsubset x\} \Longrightarrow$ *extreme* Y $(\sqsubseteq)$ $y \Longrightarrow x = f\ y \Longrightarrow$ *thesis*
and $\bigwedge Y.\ Y = \{y \in X.\ y \sqsubset x\} \Longrightarrow f$ ‘ $Y \subseteq Y \Longrightarrow$ *extreme-bound* A $(\sqsubseteq)$ Y x
$\Longrightarrow$ *thesis*
shows *thesis*
using *assms* **unfolding** *derivation-def Let-def* **by** *auto*

definition *derivable* $x \equiv \exists X.$ *derivation* $X \wedge x \in X$

lemma *derivableI*[*intro?*]: *derivation* $X \Longrightarrow x \in X \Longrightarrow$ *derivable* x **by** (*auto simp*: *derivable-def*)
lemma *derivableE*: *derivable* $x \Longrightarrow (\bigwedge P.$ *derivation* $P \Longrightarrow x \in P \Longrightarrow$ *thesis*$) \Longrightarrow$ *thesis*
by (*auto simp*: *derivable-def*)

lemma *derivable-A*: *derivable* $x \Longrightarrow x \in A$ **by** (*auto elim*: *derivableE dest*:*derivation-A*)

lemma *UN-derivations-eq-derivable*: $(\bigcup \{P.$ *derivation* $P\}) = \{x.$ *derivable* $x\}$
by (*auto simp*: *derivable-def*)

context
assumes *ord*: *antisymmetric* A $(\sqsubseteq)$
begin

interpretation *antisymmetric* **using** *ord*.

lemma *derivation-lim*:
assumes *P*: *derivation* P **and** *fP*: f ‘ $P \subseteq P$ **and** *Pp*: *extreme-bound* A $(\sqsubseteq)$ P p
shows *derivation* $(P \cup \{p\})$
proof (*cases* $p \in P$)
case *True*
with P **show** *?thesis* **by** (*auto simp*: *insert-absorb*)
next
case *pP*: *False*
interpret *P*: *well-ordered-set* P $(\sqsubseteq)$ **using** *derivation-well-ordered*[*OF P*].
have *PA*: $P \subseteq A$ **using** *derivation-A*[*OF P*].

from *Pp* **have** *pA*: $p \in A$ **by** *auto*
have *bp*: *bound P* ($\sqsubseteq$) *p* **using** *Pp* **by** *auto*
then have *pp*: $p \sqsubseteq p$ **using** *Pp* **by** *auto*
have *1*: $y \in P \longrightarrow \{x.\ (x = p \vee x \in P) \wedge x \sqsubset y\} = \{x \in P.\ x \sqsubset y\}$ **for** *y*
using *Pp* **by** (*auto dest*:*extreme-bound-bound*)
{ fix *x* **assume** *xP*: $x \in P$ **and** *px*: $p \sqsubseteq x$
from *xP Pp* **have** $x \sqsubseteq p$ **by** *auto*
with *px* **have** $p = x$ **using** *xP PA pA* **by** (*auto intro*!: *antisym*)
with *xP pP*
have *False* **by** *auto*
}
note *2* = *this*
then have *3*: $\{x.\ (x = p \vee x \in P) \wedge x \sqsubset p\} = P$ **using** *Pp* **by** (*auto intro*!: *asympartpI*)
have *wr*: *well-ordered-set* ($P \cup \{p\}$) ($\sqsubseteq$)
apply (*rule well-order-extend*[*OF P.well-ordered-set-axioms*])
using *pp bp pP 2* **by** *auto*
from *P fP Pp*
show *derivation* ($P \cup \{p\}$) **by** (*auto simp*: *derivation-def pA wr*[*simplified*] *1 3*)
qed

context
assumes *derivation-infl*: $\forall X\ x\ y.$ *derivation* $X \longrightarrow x \in X \longrightarrow y \in X \longrightarrow x \sqsubseteq y \longrightarrow x \sqsubseteq f\ y$
and *derivation-f-refl*: $\forall X\ x.$ *derivation* $X \longrightarrow x \in X \longrightarrow f\ x \sqsubseteq f\ x$
begin

lemma *derivation-suc*:
assumes *P*: *derivation P* **and** *Pp*: *extreme P* ($\sqsubseteq$) *p* **shows** *derivation* ($P \cup \{f\ p\}$)
proof (*cases f p* $\in$ *P*)
case *True*
with *P* **show** *?thesis* **by** (*auto simp*: *insert-absorb*)
next
case *fpP*: *False*
interpret *P*: *well-ordered-set P* ($\sqsubseteq$) **using** *derivation-well-ordered*[*OF P*]**.**
have *PA*: $P \subseteq A$ **using** *derivation-A*[*OF P*]**.**
with *Pp* **have** *pP*: $p \in P$ **and** *pA*: $p \in A$ **by** *auto*
with *f* **have** *fpA*: $f\ p \in A$ **by** *auto*
from *pP* **have** *pp*: $p \sqsubseteq p$ **by** *auto*
from *derivation-infl*[*rule-format*, *OF P pP pP pp*] **have** $p \sqsubseteq f\ p$**.**
{ fix *x* **assume** *xP*: $x \in P$
then have *xA*: $x \in A$ **using** *PA* **by** *auto*
have *xp*: $x \sqsubseteq p$ **using** *xP Pp* **by** *auto*
from *derivation-infl*[*rule-format*, *OF P xP pP this*]
have $x \sqsubseteq f\ p$**.**
}
note *Pfp* = *this*
then have *bfp*: *bound P* ($\sqsubseteq$) (*f p*) **by** *auto*

```
  { fix y assume yP: y ∈ P
    note yfp = Pfp[OF yP]
    { assume fpy: f p ⊑ y
      with yfp have f p = y using yP PA pA fpA antisym by auto
      with yP fpP have False by auto
    }
    with Pfp yP have y ⊏ f p by auto
  }
  note Pfp = this
  have 1: ⋀y. y ∈ P ⟶ {x. (x = f p ∨ x ∈ P) ∧ x ⊏ y} = {x ∈ P. x ⊏ y}
   and 2: {x. (x = f p ∨ x ∈ P) ∧ x ⊏ f p} = P using Pfp by auto
  have wr: well-ordered-set (P ∪ {f p}) (⊑)
   apply (rule well-order-extend[OF P.well-ordered-set-axioms singleton-well-ordered])
    using Pfp derivation-f-refl[rule-format, OF P pP] by auto
  from P Pp
   show derivation (P ∪ {f p}) by (auto simp: derivation-def wr[simplified] 1 2
fpA)
qed

lemma derivable-closed:
  assumes x: derivable x shows derivable (f x)
proof (insert x, elim derivableE)
  fix P
  assume P: derivation P and xP: x ∈ P
  note PA = derivation-A[OF P]
  then have xA: x ∈ A using xP by auto
  interpret P: well-ordered-set P (⊑) using derivation-well-ordered[OF P].
  interpret P.asympartp: transitive P (⊏) using P.asympartp-transitive.
  define Px where Px ≡ {y. y ∈ P ∧ y ⊏ x} ∪ {x}
  then have PxP: Px ⊆ P using xP by auto
  have x ⊑ x using xP by auto
  then have Pxx: extreme Px (⊑) x using xP PA by (auto simp: Px-def)
  have wr: well-ordered-set Px (⊑) using P.well-ordered-subset[OF PxP].
  { fix z y assume zPx: z ∈ Px and yP: y ∈ P and yz: y ⊏ z
    then have zP: z ∈ P using PxP by auto
    have y ⊏ x
    proof (cases z = x)
      case True
      then show ?thesis using yz by auto
    next
      case False
      then have zx: z ⊏ x using zPx by (auto simp: Px-def)
      from P.asympartp.trans[OF yz zx yP zP xP] show ?thesis.
    qed
  }
  then have 1: ⋀z. z ∈ Px ⟶ {y ∈ Px. y ⊏ z} = {y ∈ P. y ⊏ z} using Px-def
by blast
  have Px: derivation Px using PxP PA P by (auto simp: wr derivation-def 1)
  from derivation-suc[OF Px Pxx]
```

```
  show ?thesis by (auto intro!: derivableI)
qed
```

The following lemma is derived from Grall's proof. We simplify the claim so that we consider two elements from one derivation, instead of two derivations.

```
lemma derivation-useful:
  assumes X: derivation X and xX: x ∈ X and yX: y ∈ X and xy: x ⊏ y
  shows f x ⊑ y
proof−
  interpret X: well-ordered-set X (⊑) using derivation-well-ordered[OF X].
  note XA = derivation-A[OF X]
  { fix x y assume xX: x ∈ X and yX: y ∈ X
   from xX yX have (x ⊏ y ⟶ f x ⊑ y ∧ f x ∈ X) ∧ (y ⊏ x ⟶ f y ⊑ x ∧ f y
∈ X)
    proof (induct x arbitrary: y)
      case (less x)
      note xX = ‹x ∈ X› and IHx = this(2)
      with XA have xA: x ∈ A by auto
      from ‹y ∈ X› show ?case
      proof (induct y)
        case (less y)
        note yX = ‹y ∈ X› and IHy = this(2)
        with XA have yA: y ∈ A by auto
        show ?case
        proof (rule conjI; intro impI)
          assume xy: x ⊏ y
          from X yX
          show f x ⊑ y ∧ f x ∈ X
          proof (cases rule:derivation-cases)
            case (suc Z z)
            with XA have zX: z ∈ X and zA: z ∈ A and zy: z ⊏ y and yfz: y = f
z by auto
            from xX zX show ?thesis
            proof (cases rule: X.comparable-three-cases)
              case xz: less
              with IHy[OF zX zy] have fxz: f x ⊑ z and fxX: f x ∈ X by auto
               from derivation-infl[rule-format, OF X fxX zX fxz] have f x ⊑ y by
(auto simp: yfz)
              with fxX show ?thesis by auto
            next
              case eq
              with xX zX have x = z by auto
              with yX yfz show ?thesis by auto
            next
              case zx: greater
              with IHy[OF zX zy] yfz xy have False by auto
              then show ?thesis by auto
            qed
```

```
        next
          case (lim Z)
          note Z = ‹Z = {z ∈ X. z ⊏ y}› and fZ = ‹f ‘ Z ⊆ Z›
          from xX xy have x ∈ Z by (auto simp: Z)
          with fZ have f x ∈ Z by auto
          then have f x ⊏ y and f x ∈ X by (auto simp: Z)
          then show ?thesis by auto
        qed
      next
        assume yx: y ⊏ x
        from X xX
        show f y ⊑ x ∧ f y ∈ X
        proof (cases rule:derivation-cases)
          case (suc Z z)
          with XA have zX: z ∈ X and zA: z ∈ A and zx: z ⊏ x and xfz: x = f
z by auto
          from yX zX show ?thesis
          proof (cases rule: X.comparable-three-cases)
            case yz: less
            with IHx[OF zX zx yX] have fyz: f y ⊑ z and fyX: f y ∈ X by auto
             from derivation-infl[rule-format, OF X fyX zX fyz] have f y ⊑ x by
(auto simp: xfz)
            with fyX show ?thesis by auto
          next
            case eq
            with yX zX have y = z by auto
            with xX xfz show ?thesis by auto
          next
            case greater
            with IHx[OF zX zx yX] xfz yx have False by auto
            then show ?thesis by auto
          qed
        next
          case (lim Z)
          note Z = ‹Z = {z ∈ X. z ⊏ x}› and fZ = ‹f ‘ Z ⊆ Z›
          from yX yx have y ∈ Z by (auto simp: Z)
          with fZ have f y ∈ Z by auto
          then have f y ⊏ x and f y ∈ X by (auto simp: Z)
          then show ?thesis by auto
        qed
      qed
    qed
  qed
 }
 with assms show f x ⊑ y by auto
qed
```

Next one is the main lemma of this section, stating that elements from two possibly different derivations are comparable, and moreover the lower one is in the derivation of the upper one. The latter claim, not found in

Grall's proof, is crucial in proving that the union of all derivations is well-related.

lemma *derivations-cross-compare*:
assumes X: *derivation* X **and** Y: *derivation* Y **and** xX: $x \in X$ **and** yY: $y \in Y$
shows $(x \sqsubset y \wedge x \in Y) \vee x = y \vee (y \sqsubset x \wedge y \in X)$
proof−
{ fix X Y x y
assume X: *derivation* X **and** Y: *derivation* Y **and** xX: $x \in X$ **and** yY: $y \in Y$
interpret X: *well-ordered-set* X $(\sqsubseteq)$ **using** *derivation-well-ordered*[*OF* X]**.**
interpret X.*asympartp*: *transitive* X $(\sqsubset)$ **using** X.*asympartp-transitive***.**
interpret Y: *well-ordered-set* Y $(\sqsubseteq)$ **using** *derivation-well-ordered*[*OF* Y]**.**
have XA: $X \subseteq A$ **using** *derivation-A*[*OF* X]**.**
then have xA: $x \in A$ **using** xX **by** *auto*
with f **have** fxA: $f\ x \in A$ **by** *auto*
have YA: $Y \subseteq A$ **using** *derivation-A*[*OF* Y]**.**
then have yA: $y \in A$ **using** yY **by** *auto*
with f **have** fyA: $f\ y \in A$ **by** *auto*
{ fix Z
assume Z: $Z = \{z \in X.\ z \sqsubset x\}$
and fZ: $f\ `\ Z \subseteq Z$
and Zx: *extreme-bound* A $(\sqsubseteq)$ Z x
and IHx: $\forall z \in X.\ z \sqsubset x \longrightarrow (z \sqsubset y \wedge z \in Y) \vee z = y \vee (y \sqsubset z \wedge y \in X)$
have $(y \sqsubset x \wedge y \in X) \vee x \sqsubseteq y$
proof (*cases* $\exists z \in Z.\ y \sqsubset z$)
case *True*
then obtain z **where** zZ: $z \in Z$ **and** yz: $y \sqsubset z$ **by** *auto*
from zZ Z **have** zX: $z \in X$ **and** zx: $z \sqsubset x$ **by** *auto*
from IHx[*rule-format*, *OF* zX zx] yz **have** yX: $y \in X$ **by** *auto*
from X.*asympartp.trans*[*OF* yz zx yX zX xX] **have** $y \sqsubset x$**.**
with yX **show** *?thesis* **by** *auto*
next
case *False*
have *bound* Z $(\sqsubseteq)$ y
proof
fix z **assume** $z \in Z$
then have zX: $z \in X$ **and** zx: $z \sqsubset x$ **and** nyz: $\neg\ y \sqsubset z$ **using** Z *False* **by** *auto*
with IHx[*rule-format*, *OF* zX zx] X **show** $z \sqsubseteq y$ **by** *auto*
qed
with yA Zx **have** xy: $x \sqsubseteq y$ **by** *auto*
then show *?thesis* **by** *auto*
qed
}
note *lim-any* = *this*
{ fix z Z
assume Z: $Z = \{z \in X.\ z \sqsubset x\}$
and Zz: *extreme* Z $(\sqsubseteq)$ z
and xfz: $x = f\ z$
and IHx: $(z \sqsubset y \wedge z \in Y) \vee z = y \vee (y \sqsubset z \wedge y \in X)$

```
    have zX: z ∈ X and zx: z ⊏ x using Zz Z by (auto simp: extreme-def)
    then have zA: z ∈ A using XA by auto
    from IHx have (y ⊏ x ∧ y ∈ X) ∨ x ⊑ y
    proof (elim disjE conjE)
      assume zy: z ⊏ y and zY: z ∈ Y
      from derivation-useful[OF Y zY yY zy] xfz have xy: x ⊑ y by auto
      then show ?thesis by auto
    next
      assume zy: z = y
      then have y ⊏ x using zx by auto
      with zy zX show ?thesis by auto
    next
      assume yz: y ⊏ z and yX: y ∈ X
      from X.asympartp.trans[OF yz zx yX zX xX] have y ⊏ x.
      with yX show ?thesis by auto
    qed
  }
  note lim-any this
 }
 note lim-any = this(1) and suc-any = this(2)
 interpret X: well-ordered-set X (⊑) using derivation-well-ordered[OF X].
 interpret Y: well-ordered-set Y (⊑) using derivation-well-ordered[OF Y].
 have XA: X ⊆ A using derivation-A[OF X].
 have YA: Y ⊆ A using derivation-A[OF Y].
 from xX yY show ?thesis
 proof (induct x arbitrary: y)
  case (less x)
  note xX = ‹x ∈ X› and IHx = this(2)
  from xX XA f have xA: x ∈ A and fxA: f x ∈ A by auto
  from ‹y ∈ Y›
  show ?case
  proof (induct y)
   case (less y)
   note yY = ‹y ∈ Y› and IHy = less(2)
   from yY YA f have yA: y ∈ A and fyA: f y ∈ A by auto
   from X xX show ?case
   proof (cases rule: derivation-cases)
    case (suc Z z)
    note Z = ‹Z = {z ∈ X. z ⊏ x}› and Zz = ‹extreme Z (⊑) z› and xfz = ‹x
= f z›
    then have zx: z ⊏ x and zX: z ∈ X by auto
    note IHz = IHx[OF zX zx yY]
     have 1: y ⊏ x ∧ y ∈ X ∨ x ⊑ y using suc-any[OF X Y xX yY Z Zz xfz
IHz] IHy by auto
    from Y yY show ?thesis
    proof (cases rule: derivation-cases)
     case (suc W w)
     note W = ‹W = {w ∈ Y. w ⊏ y}› and Ww = ‹extreme W (⊑) w› and
yfw = ‹y = f w›
```

```
        then have wY: w ∈ Y and wy: w ⊏ y by auto
         have IHw: w ⊏ x ∧ w ∈ X ∨ w = x  ∨ x ⊏ w ∧ x ∈ Y using IHy[OF
wY wy] by auto
        have x ⊏ y ∧ x ∈ Y ∨ y ⊑ x using suc-any[OF Y X yY xX W Ww yfw
IHw] by auto
        with 1 show ?thesis using antisym xA yA by auto
       next
        case (lim W)
        note W = ‹W = {w ∈ Y. w ⊏ y}› and fW = ‹f ‘ W ⊆ W› and Wy =
‹extreme-bound A (⊑) W y›
        have x ⊏ y ∧ x ∈ Y ∨ y ⊑ x using lim-any[OF Y X yY xX W fW Wy]
IHy by auto
        with 1 show ?thesis using antisym xA yA by auto
       qed
     next
       case (lim Z)
         note Z = ‹Z = {z ∈ X. z ⊏ x}› and fZ = ‹f ‘ Z ⊆ Z› and Zx =
‹extreme-bound A (⊑) Z x›
        have 1: y ⊏ x ∧ y ∈ X ∨ x ⊑ y using lim-any[OF X Y xX yY Z fZ Zx]
IHx[OF - - yY] by auto
       from Y yY show ?thesis
       proof (cases rule: derivation-cases)
        case (suc W w)
        note W = ‹W = {w ∈ Y. w ⊏ y}› and Ww = ‹extreme W (⊑) w› and
yfw = ‹y = f w›
        then have wY: w ∈ Y and wy: w ⊏ y by auto
        have IHw: w ⊏ x ∧ w ∈ X ∨  w = x  ∨  x ⊏ w ∧ x ∈ Y using IHy[OF
wY wy] by auto
        have x ⊏ y ∧ x ∈ Y ∨ y ⊑ x using suc-any[OF Y X yY xX W Ww yfw
IHw] by auto
        with 1 show ?thesis using antisym xA yA by auto
       next
        case (lim W)
        note W = ‹W = {w ∈ Y. w ⊏ y}› and fW = ‹f ‘ W ⊆ W› and Wy =
‹extreme-bound A (⊑) W y›
        have x ⊏ y ∧ x ∈ Y ∨ y ⊑ x using lim-any[OF Y X yY xX W fW Wy]
IHy by auto
        with 1 show ?thesis using antisym xA yA by auto
       qed
     qed
   qed
  qed
qed

interpretation derivable: well-ordered-set {x. derivable x} (⊑)
proof (rule well-ordered-set.intro)
  show antisymmetric {x. derivable x} (⊑)
    apply unfold-locales by (auto dest: derivable-A antisym)
  show well-related-set {x. derivable x} (⊑)
```

apply (*fold UN-derivations-eq-derivable*)
apply (*rule closed-UN-well-related*)
by (*auto dest*: *derivation-well-ordered derivations-cross-compare well-ordered-set.axioms*)
qed

lemmas *derivable-well-ordered* = *derivable.well-ordered-set-axioms*
lemmas *derivable-total-ordered* = *derivable.total-ordered-set-axioms*
lemmas *derivable-well-related* = *derivable.well-related-set-axioms*

lemma *pred-unique*:
assumes *X*: *derivation X* **and** *xX*: $x \in X$
shows $\{z \in X.\ z \sqsubset x\} = \{z.\ derivable\ z \wedge z \sqsubset x\}$
proof
{ fix *z* **assume** $z \in X$ **and** $z \sqsubset x$
then have *derivable z* $\wedge$ $z \sqsubset x$ **using** *X* **by** (*auto simp*: *derivable-def*)
}
then show $\{z \in X.\ z \sqsubset x\} \subseteq \{z.\ derivable\ z \wedge z \sqsubset x\}$ **by** *auto*
{ fix *z* **assume** *derivable z* **and** *zx*: $z \sqsubset x$
then obtain *Y* **where** *Y*: *derivation Y* **and** *zY*: $z \in Y$ **by** (*auto simp*: *derivable-def*)
then have $z \in X$ **using** *derivations-cross-compare*[*OF X Y xX zY*] *zx* **by** *auto*
}
then show $\{z \in X.\ z \sqsubset x\} \supseteq \{z.\ derivable\ z \wedge z \sqsubset x\}$ **by** *auto*
qed

The set of all derivable elements is itself a derivation.

lemma *derivation-derivable*: *derivation* $\{x.\ derivable\ x\}$
apply (*unfold derivation-def*)
apply (*safe intro*!: *derivable-A derivable.well-ordered-set-axioms elim*!: *derivableE*)
apply (*unfold mem-Collect-eq pred-unique*[*symmetric*])
by (*auto simp*: *derivation-def*)

Finally, if the set of all derivable elements admits a supremum, then it is a fixed point.

lemma
assumes *p*: *extreme-bound A* ($\sqsubseteq$) $\{x.\ derivable\ x\}$ *p*
shows *sup-derivable-qfp*: $f\ p \sim p$ **and** *sup-derivable-fp*: $f\ p = p$
proof (*intro antisym sympartpI*)
define *P* **where** $P \equiv \{x.\ derivable\ x\}$
have *pA*: $p \in A$ **using** *p* **by** *auto*
have *P*: *derivation P* **using** *derivation-derivable* **by** (*simp add*: *P-def*)
from *derivable-closed* **have** *fP*: $f\ `\ P \subseteq P$ **by** (*auto simp*: *P-def*)
from *derivation-lim*[*OF P fP*] *p*
have *pP*: $p \in P$ **by** (*auto simp*: *P-def intro*:*derivableI*)
with *fP* **have** $f\ p \in P$ **by** *auto*
with *p* **show** *fpp*: $f\ p \sqsubseteq p$ **by** (*auto simp*: *P-def*)
show *pfp*: $p \sqsubseteq f\ p$ **apply** (*rule derivation-infl*[*rule-format*, *OF P*]) **using** *pP* **by** (*auto simp*: *P-def*)

from *fpp pfp p f* **show** $f\ p = p$ **by** (*auto intro*!: *antisym*)
qed

end

The assumptions are satisfied by monotone functions.

context
assumes *mono*: *monotone-on A* $(\sqsubseteq)$ $(\sqsubseteq)$ *f*
begin

lemma *mono-imp-derivation-infl*:
$\forall X\ x\ y.$ *derivation* $X \longrightarrow x \in X \longrightarrow y \in X \longrightarrow x \sqsubseteq y \longrightarrow x \sqsubseteq f\ y$
proof (*intro allI impI*)
fix *X x y*
assume *X*: *derivation X* **and** *xX*: $x \in X$ **and** *yX*: $y \in X$ **and** *xy*: $x \sqsubseteq y$
interpret *X*: *well-ordered-set X* $(\sqsubseteq)$ **using** *derivation-well-ordered*[*OF X*].
note *XA* = *derivation-A*[*OF X*]
from *xX yX xy* **show** $x \sqsubseteq f\ y$
proof (*induct x*)
case (*less x*)
note *IH* = *this*(*2*) **and** *xX* = ‹$x \in X$› **and** *yX* = ‹$y \in X$› **and** *xy* = ‹$x \sqsubseteq y$›
from *xX yX XA* **have** *xA*: $x \in A$ **and** *yA*: $y \in A$ **by** *auto*
from *X xX* **show** *?case*
proof (*cases rule*: *derivation-cases*)
case (*suc Z z*)
then have *zX*: $z \in X$ **and** *zsx*: $z \sqsubset x$ **and** *xfz*: $x = f\ z$ **by** *auto*
then have *zx*: $z \sqsubseteq x$ **by** *auto*
from *X.trans*[*OF zx xy zX xX yX*] **have** *zy*: $z \sqsubseteq y$.
from *zX XA* **have** *zA*: $z \in A$ **by** *auto*
from *zy monotone-onD*[*OF mono*] *zA yA xfz* **show** $x \sqsubseteq f\ y$ **by** *auto*
next
case (*lim Z*)
note *Z* = ‹$Z = \{z \in X.\ z \sqsubset x\}$› **and** *Zx* = ‹*extreme-bound A* $(\sqsubseteq)$ *Z x*›
from *f yA* **have** *fyA*: $f\ y \in A$ **by** *auto*
have *bound Z* $(\sqsubseteq)$ $(f\ y)$
proof
fix *z* **assume** *zZ*: $z \in Z$
with *Z xX* **have** *zsx*: $z \sqsubset x$ **and** *zX*: $z \in X$ **by** *auto*
then have *zx*: $z \sqsubseteq x$ **by** *auto*
from *X.trans*[*OF zx xy zX xX yX*] **have** *zy*: $z \sqsubseteq y$.
from *IH*[*OF zX zsx yX*] *zy* **show** $z \sqsubseteq f\ y$ **by** *auto*
qed
with *Zx fyA* **show** *?thesis* **by** *auto*
qed
qed
qed

lemma *mono-imp-derivation-f-refl*:
$\forall X\ x.$ *derivation* $X \longrightarrow x \in X \longrightarrow f\ x \sqsubseteq f\ x$

proof (*intro allI impI*)
fix X x
assume X: *derivation* X **and** xX: $x \in X$
interpret X: *well-ordered-set* X $(\sqsubseteq)$ **using** *derivation-well-ordered*[*OF* X]**.**
note XA = *derivation-A*[*OF* X]
from xX **have** $x \sqsubseteq x$ **by** *auto*
from *monotone-onD*[*OF mono this*] xX XA **show** $f\ x \sqsubseteq f\ x$ **by** *auto*
qed

corollary *mono-imp-sup-derivable-fp*:
assumes p: *extreme-bound* A $(\sqsubseteq)$ $\{x.\ derivable\ x\}$ p
shows $f\ p = p$
by (*simp add*: *sup-derivable-fp*[*OF mono-imp-derivation-infl mono-imp-derivation-f-refl p*])

lemma *mono-imp-sup-derivable-lfp*:
assumes p: *extreme-bound* A $(\sqsubseteq)$ $\{x.\ derivable\ x\}$ p
shows *extreme* $\{q \in A.\ f\ q = q\}$ $(\sqsupseteq)$ p
proof (*safe intro*!: *extremeI*)
from p **show** $p \in A$ **by** *auto*
from *sup-derivable-fp*[*OF mono-imp-derivation-infl mono-imp-derivation-f-refl p*]
show $f\ p = p$**.**
fix q **assume** qA: $q \in A$ **and** fqq: $f\ q = q$
have *bound* $\{x.\ derivable\ x\}$ $(\sqsubseteq)$ q
proof (*safe intro*!: *boundI elim*!:*derivableE*)
fix x X
assume X: *derivation* X **and** xX: $x \in X$
from X **interpret** *well-ordered-set* X $(\sqsubseteq)$ **by** (*rule derivation-well-ordered*)
from xX **show** $x \sqsubseteq q$
proof (*induct x*)
case (*less x*)
note xP = *this*(*1*) **and** IH = *this*(*2*)
with X **show** *?case*
proof (*cases rule*: *derivation-cases*)
case (*suc Z z*)
with IH[*of z*] **have** zq: $z \sqsubseteq q$ **and** zX: $z \in X$ **by** *auto*
from *monotone-onD*[*OF mono zq*] zX qA *derivation-A*[*OF* X]
show *?thesis* **by** (*auto simp*: *fqq suc*)
next
case *lim*
with IH **have** *bound* $\{z \in X.\ z \sqsubset x\}$ $(\sqsubseteq)$ q **by** *auto*
with *lim* qA **show** *?thesis* **by** *auto*
qed
qed
qed
with p qA **show** $p \sqsubseteq q$ **by** *auto*
qed

lemma *mono-imp-ex-least-fp*:

assumes *comp*: *well-complete A* ($\sqsubseteq$)
shows $\exists\, p.$ *extreme* $\{q \in A.\ f\ q = q\}$ ($\sqsupseteq$) p
proof–
interpret *fixed-point-proof* **using** f **by** *unfold-locales*
note *as* = *antisymmetric-axioms*
note *infl* = *mono-imp-derivation-infl*
note *refl* = *mono-imp-derivation-f-refl*
have *wr*: *well-related-set* $\{x.$ *derivable* $x\}$ ($\sqsubseteq$)
using *derivable-well-related*[*OF infl refl*]**.**
have $\exists\, p.$ *extreme-bound A* ($\sqsubseteq$) $\{x.$ *derivable* $x\}$ p
apply (*rule completeD*[*OF comp*])
using *derivable-A wr* **by** *auto*
then obtain p **where** p: *extreme-bound A* ($\sqsubseteq$) $\{x.$ *derivable* $x\}$ p **by** *auto*
from p *mono-imp-sup-derivable-lfp*[*OF p*] *sup-derivable-qfp*[*OF infl refl p*]
show *?thesis* **by** *auto*
qed

end

end

end

Bourbaki-Witt Theorem on well-complete pseudo-ordered set:

theorem (**in** *pseudo-ordered-set*) *well-complete-infl-imp-ex-fp*:
assumes *comp*: *well-complete A* ($\sqsubseteq$)
and f: f ' $A \subseteq A$ **and** *infl*: $\forall\, x \in A.\ \forall\, y \in A.\ x \sqsubseteq y \longrightarrow x \sqsubseteq f\ y$
shows $\exists\, p \in A.\ f\ p = p$
proof–
note *as* = *antisymmetric-axioms*
interpret *fixed-point-proof* **using** f **by** *unfold-locales*
have *dinfl*: $\forall\, X\ x\ y.$ *derivation* $X \longrightarrow x \in X \longrightarrow y \in X \longrightarrow x \sqsubseteq y \longrightarrow x \sqsubseteq f\ y$
using *infl* **by** (*auto dest*!:*derivation-A*)
have *drefl*: $\forall\, X\ x.$ *derivation* $X \longrightarrow x \in X \longrightarrow f\ x \sqsubseteq f\ x$ **using** f **by** (*auto dest*!: *derivation-A*)
have $\exists\, p.$ *extreme-bound A* ($\sqsubseteq$) $\{x.$ *derivable* $x\}$ p
apply (*rule completeD*[*OF comp*])
using *derivable-well-related*[*OF as dinfl drefl*] *derivable-A* **by** *auto*
with *sup-derivable-fp*[*OF as dinfl drefl*]
show *?thesis* **by** *auto*
qed

6 Completeness of (Quasi-)Fixed Points

We now prove that, under attractivity, the set of quasi-fixed points is complete.

definition *setwise* **where** *setwise r X Y* $\equiv \forall\, x{\in}X.\ \forall\, y{\in}Y.\ r\ x\ y$

lemmas *setwiseI*[*intro*] = *setwise-def*[*unfolded atomize-eq*, *THEN iffD2*, *rule-format*]
lemmas *setwiseE*[*elim*] = *setwise-def*[*unfolded atomize-eq*, *THEN iffD1*, *elim-format*, *rule-format*]

context *fixed-point-proof* **begin**

abbreviation *setwise-less-eq* (**infix** $\sqsubseteq^s$ *50*) **where** $(\sqsubseteq^s) \equiv$ *setwise* $(\sqsubseteq)$

6.1 Least Quasi-Fixed Points for Attractive Relations.

lemma *attract-mono-imp-least-qfp*:
assumes *attract*: *attractive* A $(\sqsubseteq)$
and *comp*: *well-complete* A $(\sqsubseteq)$
and *mono*: *monotone-on* A $(\sqsubseteq)$ $(\sqsubseteq)$ f
shows $\exists\, c.$ *extreme* $\{p \in A.\ f\ p \sim p \vee f\ p = p\}$ $(\sqsupseteq)$ $c \wedge f\ c \sim c$
proof–
interpret *attractive* **using** *attract* **by** *auto*
interpret *sym*: *transitive* A $(\sim)$ **using** *sym-transitive*.
define *ecl* $([\text{-}]_\sim)$ **where** $[x]_\sim \equiv \{y \in A.\ x \sim y\} \cup \{x\}$ **for** x
define Q **where** $Q \equiv \{[x]_\sim \mid .\ x \in A\}$
{ fix $X\ x$ **assume** XQ: $X \in Q$ **and** xX: $x \in X$
then have XA: $X \subseteq A$ **by** (*auto simp*: *Q-def ecl-def*)
then have xA: $x \in A$ **using** xX **by** *auto*
obtain q **where** qA: $q \in A$ **and** X: $X = [q]_\sim$ **using** XQ **by** (*auto simp*: *Q-def*)
have $xqqx$: $x \sim q \vee x = q$ **using** $X\ xX$ **by** (*auto simp*: *ecl-def*)
{fix y **assume** yX: $y \in X$
then have yA: $y \in A$ **using** XA **by** *auto*
have $y \sim q \vee y = q$ **using** $yX\ X$ **by** (*auto simp*: *ecl-def*)
then have $x \sim y \vee y = x$ **using** *sym-order-trans* $xqqx\ xA\ qA\ yA$ **by** *blast*
}
then have *1*: $X \subseteq [x]_\sim$ **using** $X\ qA$ **by** (*auto simp*: *ecl-def*)
{ fix y **assume** $y \in A$ **and** $x \sim y \vee y = x$
then have $q \sim y \vee y = q$ **using** *sym-order-trans* $xqqx\ xA\ qA$ **by** *blast*
}
then have *2*: $X \supseteq [x]_\sim$ **using** $X\ xX$ **by** (*auto simp*: *ecl-def*)
from *1 2* **have** $X = [x]_\sim$ **by** *auto*
}
then have XQx: $\forall\, X \in Q.\ \forall\, x \in X.\ X = [x]_\sim$ **by** *auto*
have *RSLE-eq*: $X \in Q \Longrightarrow Y \in Q \Longrightarrow x \in X \Longrightarrow y \in Y \Longrightarrow x \sqsubseteq y \Longrightarrow X \sqsubseteq^s Y$ **for** $X\ Y\ x\ y$
proof–
assume XQ: $X \in Q$ **and** YQ: $Y \in Q$ **and** xX: $x \in X$ **and** yY: $y \in Y$ **and** xy: $x \sqsubseteq y$
then have XA: $X \subseteq A$ **and** YA: $Y \subseteq A$ **by** (*auto simp*: *Q-def ecl-def*)
then have xA: $x \in A$ **and** yA: $y \in A$ **using** $xX\ yY$ **by** *auto*
{ fix $xp\ yp$ **assume** xpX: $xp \in X$ **and** ypY: $yp \in Y$
then have xpA: $xp \in A$ **and** ypA: $yp \in A$ **using** $XA\ YA$ **by** *auto*
then have $xp \sim x \vee xp = x$ **using** $xpX\ XQx\ xX\ XQ$ **by** (*auto simp*: *ecl-def*)
then have xpy: $xp \sqsubseteq y$ **using** *attract*[*OF - - xy xpA xA yA*] xy **by** *blast*

have $yp \sim y \vee yp = y$ **using** *ypY XQx yY YQ* **by** (*auto simp*: *ecl-def*)
then have $xp \sqsubseteq yp$ **using** *dual.attract*[*OF - - xpy ypA yA xpA*] *xpy* **by** *blast*
}
then show $X \sqsubseteq^s Y$ **using** *XQ YQ XA YA* **by** *auto*
qed
have *compQ*: *well-complete Q* ($\sqsubseteq^s$)
proof (*intro completeI*, *safe*)
fix *XX* **assume** *XXQ*: $XX \subseteq Q$ **and** *XX*: *well-related-set XX* ($\sqsubseteq^s$)
have *BA*: $\bigcup XX \subseteq A$ **using** *XXQ* **by** (*auto simp*: *Q-def ecl-def*)
from *XX* **interpret** *XX*: *well-related-set XX* ($\sqsubseteq^s$)**.**
interpret *UXX*: *semiattractive* $\bigcup XX$ ($\sqsubseteq$) **by** (*rule semiattractive-subset*[*OF BA*])
have *well-related-set* ($\bigcup XX$) ($\sqsubseteq$)
proof(*unfold-locales*)
fix *Y* **assume** *YXX*: $Y \subseteq \bigcup XX$ **and** *Y0*: $Y \neq \{\}$
have $\{X \in XX.\ X \cap Y \neq \{\}\} \neq \{\}$ **using** *YXX Y0* **by** *auto*
from *XX.nonempty-imp-ex-extreme*[*OF - this*]
obtain *E* **where** *E*: *extreme* $\{X \in XX.\ X \cap Y \neq \{\}\}$ $(\sqsubseteq^s)^-$ *E* **by** *auto*
then have $E \cap Y \neq \{\}$ **by** *auto*
then obtain *e* **where** *eE*: $e \in E$ **and** *eX*: $e \in Y$ **by** *auto*
have *extreme Y* ($\sqsupseteq$) *e*
proof (*intro extremeI eX*)
fix *x* **assume** *xY*: $x \in Y$
with *YXX* **obtain** *X* **where** *XXX*: $X \in XX$ **and** *xX*: $x \in X$ **by** *auto*
with *xY E XXX* **have** $E \sqsubseteq^s X$ **by** *auto*
with *eE xX* **show** $e \sqsubseteq x$ **by** *auto*
qed
then show $\exists\, e.$ *extreme Y* ($\sqsupseteq$) *e* **by** *auto*
qed
with *completeD*[*OF comp BA*]
obtain *b* **where** *extb*: *extreme-bound A* ($\sqsubseteq$) ($\bigcup XX$) *b* **by** *auto*
then have *bb*: $b \sqsubseteq b$ **using** *extreme-def bound-def* **by** *auto*
have *bA*: $b \in A$ **using** *extb extreme-def* **by** *auto*
then have *XQ*: $[b]_\sim \in Q$ **using** *Q-def bA* **by** *auto*
have *bX*: $b \in [b]_\sim$ **by** (*auto simp*: *ecl-def*)
have *extreme-bound Q* ($\sqsubseteq^s$) *XX* $[b]_\sim$
proof(*intro extreme-boundI*)
show $[b]_\sim \in Q$ **using** *XQ***.**
next
fix *Y* **assume** *YXX*: $Y \in XX$
then have *YQ*: $Y \in Q$ **using** *XXQ* **by** *auto*
then obtain *y* **where** *yA*: $y \in A$ **and** *Yy*: $Y = [y]_\sim$ **by** (*auto simp*: *Q-def*)
then have *yY*: $y \in Y$ **by** (*auto simp*: *ecl-def*)
then have $y \in \bigcup XX$ **using** *yY YXX* **by** *auto*
then have $y \sqsubseteq b$ **using** *extb* **by** *auto*
then show $Y \sqsubseteq^s [b]_\sim$ **using** *RSLE-eq*[*OF YQ XQ yY bX*] **by** *auto*
next
fix *Z* **assume** *boundZ*: *bound XX* ($\sqsubseteq^s$) *Z* **and** *ZQ*: $Z \in Q$
then obtain *z* **where** *zA*: $z \in A$ **and** *Zz*: $Z = [z]_\sim$ **by** (*auto simp*: *Q-def*)

then have zZ: $z \in Z$ **by** (*auto simp*: *ecl-def*)
{ fix y **assume** $y \in \bigcup XX$
then obtain Y **where** yY: $y \in Y$ **and** YXX: $Y \in XX$ **by** *auto*
then have YA: $Y \subseteq A$ **using** XXQ Q-*def* **by** (*auto simp*: *ecl-def*)
then have $Y \sqsubseteq^s Z$ **using** YXX *boundZ bound-def* **by** *auto*
then have $y \sqsubseteq z$ **using** yY zZ **by** *auto*
}
then have *bound* $(\bigcup XX)$ $(\sqsubseteq)$ z **by** *auto*
then have $b \sqsubseteq z$ **using** *extb* zA **by** *auto*
then show $[b]_\sim \sqsubseteq^s Z$ **using** *RSLE-eq*[*OF* XQ ZQ bX zZ] **by** *auto*
qed
then show *Ex* (*extreme-bound* Q $(\sqsubseteq^s)$ XX) **by** *auto*
qed
interpret Q: *antisymmetric* Q $(\sqsubseteq^s)$
proof
fix X Y **assume** XY: $X \sqsubseteq^s Y$ **and** YX: $Y \sqsubseteq^s X$ **and** XQ: $X \in Q$ **and** YQ: $Y \in Q$
then obtain q **where** qA: $q \in A$ **and** X: $X = [q]_\sim$ **using** Q-*def* **by** *auto*
then have qX: $q \in X$ **using** X **by** (*auto simp*: *ecl-def*)
then obtain p **where** pA: $p \in A$ **and** Y: $Y = [p]_\sim$ **using** YQ Q-*def* **by** *auto*
then have pY: $p \in Y$ **using** X **by** (*auto simp*: *ecl-def*)
have pq: $p \sqsubseteq q$ **using** XQ YQ YX qX pY **by** *auto*
have $q \sqsubseteq p$ **using** XQ YQ XY qX pY **by** *auto*
then have $p \in X$ **using** pq X pA **by** (*auto simp*: *ecl-def*)
then have $X = [p]_\sim$ **using** XQ XQx **by** *auto*
then show $X = Y$ **using** Y **by** (*auto simp*: *ecl-def*)
qed
define F **where** $F\ X \equiv \{y \in A.\ \exists x \in X.\ y \sim f\ x\} \cup f\ `\ X$ **for** X
have $XQFXQ$: $\bigwedge X.\ X \in Q \Longrightarrow F\ X \in Q$
proof–
fix X **assume** XQ: $X \in Q$
then obtain x **where** xA: $x \in A$ **and** X: $X = [x]_\sim$ **using** Q-*def* **by** *auto*
then have xX: $x \in X$ **by** (*auto simp*: *ecl-def*)
have fxA: $f\ x \in A$ **using** xA f **by** *auto*
have FXA: $F\ X \subseteq A$ **using** f fxA X **by** (*auto simp*: F-*def ecl-def*)
have $F\ X = [f\ x]_\sim$
proof (*unfold* X, *intro equalityI subsetI*)
fix z **assume** zFX: $z \in F\ [x]_\sim$
then obtain y **where** yX: $y \in [x]_\sim$ **and** zfy: $z \sim f\ y \vee z = f\ y$ **by** (*auto simp*: F-*def*)
have yA: $y \in A$ **using** yX xA **by** (*auto simp*: *ecl-def*)
with f **have** fyA: $f\ y \in A$ **by** *auto*
have zA: $z \in A$ **using** zFX FXA **by** (*auto simp*: X)
have $y \sim x \vee y = x$ **using** X yX **by** (*auto simp*: *ecl-def*)
then have $f\ y \sim f\ x \vee f\ y = f\ x$ **using** *mono* xA yA **by** (*auto simp*: *monotone-on-def*)
then have $z \sim f\ x \vee z = f\ x$ **using** zfy *sym.trans*[*OF* - - zA fyA fxA] **by** (*auto simp*:)
with zA **show** $z \in [f\ x]_\sim$ **by** (*auto simp*: *ecl-def*)

qed (*auto simp*: *xX F-def ecl-def*)
with *FXA* **show** *F X* ∈ *Q* **by** (*auto simp*: *Q-def ecl-def*)
qed
then have *F*: *F* ‘ *Q* ⊆ *Q* **by** *auto*
then interpret *Q*: *fixed-point-proof Q* (⊑^{s}) *F* **by** *unfold-locales*
have *monoQ*: *monotone-on Q* (⊑^{s}) (⊑^{s}) *F*
proof (*intro monotone-onI*)
fix *X Y* **assume** *XQ*: *X* ∈ *Q* **and** *YQ*: *Y* ∈ *Q* **and** *XY*: *X* ⊑^{s} *Y*
then obtain *x y* **where** *xX*: *x* ∈ *X* **and** *yY*: *y* ∈ *Y* **using** *Q-def* **by** (*auto simp*: *ecl-def*)
then have *xA*: *x* ∈ *A* **and** *yA*: *y* ∈ *A* **using** *XQ YQ* **by** (*auto simp*: *Q-def ecl-def*)
have *x* ⊑ *y* **using** *XY xX yY* **by** *auto*
then have *fxfy*: *f x* ⊑ *f y* **using** *monotone-on-def*[*of A* (⊑) (⊑) *f*] *xA yA mono* **by** *auto*
have *fxgX*: *f x* ∈ *F X* **using** *xX F-def* **by** *blast*
have *fygY*: *f y* ∈ *F Y* **using** *yY F-def* **by** *blast*
show *F X* ⊑^{s} *F Y* **using** *RSLE-eq*[*OF XQFXQ*[*OF XQ*] *XQFXQ*[*OF YQ*] *fxgX fygY fxfy*]**.**
qed
have *QdA*: {*x*. *Q.derivable x*} ⊆ *Q* **using** *Q.derivable-A* **by** *auto*
note *asQ* = *Q.antisymmetric-axioms*
note *infl* = *Q.mono-imp-derivation-infl*[*OF asQ monoQ*]
note *f-refl* = *Q.mono-imp-derivation-f-refl*[*OF asQ monoQ*]
from *Q.mono-imp-ex-least-fp*[*OF asQ monoQ compQ*]
obtain *P* **where** *P*: *extreme* {*q* ∈ *Q*. *F q* = *q*} (⊑^{s})$^-$ *P* **by** *auto*
then have *PQ*: *P* ∈ *Q* **by** (*auto simp*: *extreme-def*)
from *P* **have** *FPP*: *F P* = *P* **using** *PQ* **by** *auto*
with *P* **have** *PP*: *P* ⊑^{s} *P* **by** *auto*
from *P* **obtain** *p* **where** *pA*: *p* ∈ *A* **and** *Pp*: *P* = [*p*]$_\sim$ **using** *Q-def* **by** *auto*
then have *pP*: *p* ∈ *P* **by** (*auto simp*: *ecl-def*)
then have *fpA*: *f p* ∈ *A* **using** *pA f* **by** *auto*
have *f p* ∈ *F P* **using** *pP F-def fpA* **by** *auto*
then have *F P* = [*f p*]$_\sim$ **using** *XQx XQFXQ*[*OF PQ*] **by** *auto*
then have *fp*: *f p* ∼ *p* ∨ *f p* = *p* **using** *pP FPP* **by** (*auto simp*: *ecl-def*)
have *p* ⊑ *p* **using** *PP pP* **by** *auto*
with *fp* **have** *fpp*: *f p* ∼ *p* **by** *auto*
have *e*: *extreme* {*p* ∈ *A*. *f p* ∼ *p* ∨ *f p* = *p*} (⊒) *p*
proof (*intro extremeI CollectI conjI pA fp*, *elim CollectE conjE*)
fix *q* **assume** *qA*: *q* ∈ *A* **and** *fq*: *f q* ∼ *q* ∨ *f q* = *q*
define *Z* **where** *Z* ≡ {*z* ∈ *A*. *q* ∼ *z*}∪{*q*}
then have *qZ*: *q* ∈ *Z* **using** *qA* **by** *auto*
then have *ZQ*: *Z* ∈ *Q* **using** *qA* **by** (*auto simp*: *Z-def Q-def ecl-def*)
have *fqA*: *f q* ∈ *A* **using** *qA f* **by** *auto*
then have *f q* ∈ *Z* **using** *fq* **by** (*auto simp*: *Z-def*)
then have *1*: *Z* = [*f q*]$_\sim$ **using** *XQx ZQ* **by** *auto*
then have *f q* ∈ *F Z* **using** *qZ fqA* **by** (*auto simp*: *F-def*)
then have *F Z* = [*f q*]$_\sim$ **using** *XQx XQFXQ*[*OF ZQ*] **by** *auto*
with *1* **have** *Z* = *F Z* **by** *auto*

then have $P \sqsubseteq^s Z$ **using** $P\ ZQ$ **by** *auto*
then show $p \sqsubseteq q$ **using** $pP\ qZ$ **by** *auto*
qed
with *fpp* **show** *?thesis* **using** e **by** *auto*
qed

6.2 General Completeness

lemma *attract-mono-imp-fp-qfp-complete*:
assumes *attract*: *attractive* A $(\sqsubseteq)$
and *comp*: $CC-$*complete* A $(\sqsubseteq)$
and *wr-CC*: $\forall\, C \subseteq A.$ *well-related-set* C $(\sqsubseteq) \longrightarrow C \in CC$
and *extend*: $\forall\, X \in CC.\ \forall\, Y \in CC.\ X \sqsubseteq^s Y \longrightarrow X \cup Y \in CC$
and *mono*: *monotone-on* A $(\sqsubseteq)$ $(\sqsubseteq)$ f
and P: $P \subseteq \{x \in A.\ f\, x = x\}$
shows $CC-$*complete* $(\{q \in A.\ f\, q \sim q\} \cup P)$ $(\sqsubseteq)$
proof (*intro completeI*)
interpret *attractive* **using** *attract*.
fix X **assume** *Xfix*: $X \subseteq \{q \in A.\ f\, q \sim q\} \cup P$ **and** *XCC*: $X \in CC$
with P **have** *XA*: $X \subseteq A$ **by** *auto*
define B **where** $B \equiv \{b \in A.\ \forall\, a \in X.\ a \sqsubseteq b\}$
{ fix $s\ a$ **assume** *sA*: $s \in A$ **and** *as*: $\forall\, a \in X.\ a \sqsubseteq s$ **and** *aX*: $a \in X$
then have *aA*: $a \in A$ **using** *XA* **by** *auto*
then have *fafs*: $f\, a \sqsubseteq f\, s$ **using** *mono f aX sA as* **by** (*auto elim*!: *monotone-onE*)
have $a \sqsubseteq f\, s$
proof (*cases* $f\, a = a$)
case *True*
then show *?thesis* **using** *fafs* **by** *auto*
next
case *False*
then have $a \sim f\, a$ **using** $P\ aX\ Xfix$ **by** *auto*
also from *fafs* **have** $f\, a \sqsubseteq f\, s$ **by** *auto*
finally show *?thesis* **using** $f\ aA\ sA$ **by** *auto*
qed
}
with f **have** *fBB*: $f\ `\ B \subseteq B$ **unfolding** *B-def* **by** *auto*
have *BA*: $B \subseteq A$ **by** (*auto simp*: *B-def*)
have *compB*: $CC-$*complete* B $(\sqsubseteq)$
proof (*unfold complete-def*, *intro allI impI*)
fix Y **assume** *YS*: $Y \subseteq B$ **and** *YCC*: $Y \in CC$
with *BA* **have** *YA*: $Y \subseteq A$ **by** *auto*
define C **where** $C \equiv X \cup Y$
then have *CA*: $C \subseteq A$ **using** *XA YA C-def* **by** *auto*
have *XY*: $X \sqsubseteq^s Y$ **using** *B-def YS* **by** *auto*
then have *CCC*: $C \in CC$ **using** *extend XA YA XCC YCC C-def* **by** *auto*
then obtain s **where** s: *extreme-bound* A $(\sqsubseteq)$ $C\ s$
using *completeD*[*OF comp CA CCC*] **by** *auto*
then have *sA*: $s \in A$ **by** *auto*
show *Ex* (*extreme-bound* B $(\sqsubseteq)$ Y)

```
    proof (intro exI extreme-boundI)
      { fix x assume x ∈ X
        then have x ⊑ s using s C-def by auto
      }
      then show s ∈ B using sA B-def by auto
    next
      fix y assume y ∈ Y
      then show y ⊑ s using s C-def using extremeD by auto
    next
      fix c assume cS: c ∈ B and bound Y (⊑) c
      then have bound C (⊑) c using C-def B-def by auto
      then show s ⊑ c using s BA cS by auto
    qed
  qed
  from fBB interpret B: fixed-point-proof B (⊑) f by unfold-locales
  from BA have ∗: {x ∈ A. f x ∼ x} ∩ B = {x ∈ B. f x ∼ x} by auto
  have asB: attractive B (⊑) using attractive-subset[OF BA] by auto
   have monoB: monotone-on B (⊑) (⊑) f using monotone-on-cmono[OF BA]
mono by (auto dest!: le-funD)
  have compB: well-complete B (⊑)
    using wr-CC compB BA by (simp add: complete-def)
  from B.attract-mono-imp-least-qfp[OF asB compB monoB]
  obtain l where extreme {p ∈ B. f p ∼ p ∨ f p = p} (⊒) l and fll: f l ∼ l by
auto
  with P have l: extreme ({p ∈ B. f p ∼ p} ∪ P ∩ B) (⊒) l by auto
  show Ex (extreme-bound ({q ∈ A. f q ∼ q} ∪ P) (⊑) X)
  proof (intro exI extreme-boundI)
    show l ∈ {q ∈ A. f q ∼ q} ∪ P using l BA by auto
    fix a assume a ∈ X
    with l show a ⊑ l by (auto simp: B-def)
  next
    fix c assume c: bound X (⊑) c and cfix: c ∈ {q ∈ A. f q ∼ q} ∪ P
    with P have cA: c ∈ A by auto
    with c have c ∈ B by (auto simp: B-def)
    with cfix l show l ⊑ c by auto
  qed
qed

lemma attract-mono-imp-qfp-complete:
  assumes attractive A (⊑)
    and CC−complete A (⊑)
    and ∀ C ⊆ A. well-related-set C (⊑) ⟶ C ∈ CC
    and ∀ X ∈ CC. ∀ Y ∈ CC. X ⊑^s Y ⟶ X ∪ Y ∈ CC
    and monotone-on A (⊑) (⊑) f
  shows CC−complete {p ∈ A. f p ∼ p} (⊑)
  using attract-mono-imp-fp-qfp-complete[OF assms, of {}] by simp

lemma antisym-mono-imp-fp-complete:
  assumes anti: antisymmetric A (⊑)
```

and *comp*: *CC−complete A* ($\sqsubseteq$)
and *wr-CC*: $\forall\, C \subseteq A.$ *well-related-set C* ($\sqsubseteq$) $\longrightarrow C \in CC$
and *extend*: $\forall\, X \in CC.\ \forall\, Y \in CC.\ X \sqsubseteq^s Y \longrightarrow X \cup Y \in CC$
and *mono*: *monotone-on A* ($\sqsubseteq$) ($\sqsubseteq$) *f*
shows *CC−complete* $\{p \in A.\ f\ p = p\}$ ($\sqsubseteq$)
proof−
interpret *antisymmetric* **using** *anti*.
have $*$: $\{q \in A.\ f\ q \sim q\} \subseteq \{p \in A.\ f\ p = p\}$ **using** *f* **by** (*auto intro*!: *antisym*)
from $*$ *attract-mono-imp-fp-qfp-complete*[*OF attractive-axioms comp wr-CC extend mono*, *of* $\{p{\in}A.\ f\ p = p\}$]
show *?thesis* **by** (*auto simp*: *subset-Un-eq*)
qed

end

6.3 Instances

6.3.1 Instances under attractivity

context *attractive* **begin**

interpretation *less-eq-notations*.

Full completeness

theorem *mono-imp-qfp-UNIV-complete*:
assumes *comp*: *UNIV−complete A* ($\sqsubseteq$) **and** *f*: *f ' A* $\subseteq$ *A* **and** *mono*: *monotone-on A* ($\sqsubseteq$) ($\sqsubseteq$) *f*
shows *UNIV−complete* $\{p \in A.\ f\ p \sim p\}$ ($\sqsubseteq$)
apply (*intro fixed-point-proof.attract-mono-imp-qfp-complete comp mono*)
apply *unfold-locales*
by (*auto simp*: *f*)

Connex completeness

theorem *mono-imp-qfp-connex-complete*:
assumes *comp*: $\{X.$ *connex X* ($\sqsubseteq$)$\}$*−complete A* ($\sqsubseteq$)
and *f*: *f ' A* $\subseteq$ *A* **and** *mono*: *monotone-on A* ($\sqsubseteq$) ($\sqsubseteq$) *f*
shows $\{X.$ *connex X* ($\sqsubseteq$)$\}$*−complete* $\{p \in A.\ f\ p \sim p\}$ ($\sqsubseteq$)
apply (*intro fixed-point-proof.attract-mono-imp-qfp-complete mono comp*)
apply *unfold-locales*
by (*auto simp*: *f intro*: *connex-union well-related-set.connex-axioms*)

Directed completeness

theorem *mono-imp-qfp-directed-complete*:
assumes *comp*: $\{X.$ *directed X* ($\sqsubseteq$)$\}$*−complete A* ($\sqsubseteq$)
and *f*: *f ' A* $\subseteq$ *A* **and** *mono*: *monotone-on A* ($\sqsubseteq$) ($\sqsubseteq$) *f*
shows $\{X.$ *directed X* ($\sqsubseteq$)$\}$*−complete* $\{p \in A.\ f\ p \sim p\}$ ($\sqsubseteq$)
apply (*intro fixed-point-proof.attract-mono-imp-qfp-complete mono comp*)
apply *unfold-locales*
by (*auto simp*: *f intro*!: *directed-extend intro*: *well-related-set.connex-axioms connex.directed*)

Well Completeness

theorem *mono-imp-qfp-well-complete*:
assumes *comp*: *well-complete A* (⊑)
and *f*: *f ‘ A* ⊆ *A* **and** *mono*: *monotone-on A* (⊑) (⊑) *f*
shows *well-complete* {*p* ∈ *A*. *f p* ∼ *p*} (⊑)
apply (*intro fixed-point-proof.attract-mono-imp-qfp-complete mono comp*)
apply *unfold-locales*
by (*auto simp*: *f well-related-extend*)

end

6.3.2 Usual instances under antisymmetry

context *antisymmetric* **begin**

Knaster–Tarski

theorem *mono-imp-fp-complete*:
assumes *comp*: *UNIV*−*complete A* (⊑)
and *f*: *f ‘ A* ⊆ *A* **and** *mono*: *monotone-on A* (⊑) (⊑) *f*
shows *UNIV*−*complete* {*p* ∈ *A*. *f p* = *p*} (⊑)
proof−
interpret *fixed-point-proof* **using** *f* **by** *unfold-locales*
show *?thesis*
apply (*intro antisym-mono-imp-fp-complete mono antisymmetric-axioms comp*)
by *auto*
qed

Markowsky 1976

theorem *mono-imp-fp-connex-complete*:
assumes *comp*: {*X*. *connex X* (⊑)}−*complete A* (⊑)
and *f*: *f ‘ A* ⊆ *A* **and** *mono*: *monotone-on A* (⊑) (⊑) *f*
shows {*X*. *connex X* (⊑)}−*complete* {*p* ∈ *A*. *f p* = *p*} (⊑)
proof−
interpret *fixed-point-proof* **using** *f* **by** *unfold-locales*
show *?thesis*
apply (*intro antisym-mono-imp-fp-complete antisymmetric-axioms mono comp*)
by (*auto intro*: *connex-union well-related-set.connex-axioms*)
qed

Pataraia

theorem *mono-imp-fp-directed-complete*:
assumes *comp*: {*X*. *directed X* (⊑)}−*complete A* (⊑)
and *f*: *f ‘ A* ⊆ *A* **and** *mono*: *monotone-on A* (⊑) (⊑) *f*
shows {*X*. *directed X* (⊑)}−*complete* {*p* ∈ *A*. *f p* = *p*} (⊑)
proof−
interpret *fixed-point-proof* **using** *f* **by** *unfold-locales*
show *?thesis*
apply (*intro antisym-mono-imp-fp-complete mono antisymmetric-axioms comp*)
by (*auto intro*: *directed-extend connex.directed well-related-set.connex-axioms*)

qed

Bhatta & George 2011

theorem *mono-imp-fp-well-complete*:
assumes *comp*: *well-complete* A $(\sqsubseteq)$
and f: f ‘ $A \subseteq A$ **and** *mono*: *monotone-on* A $(\sqsubseteq)$ $(\sqsubseteq)$ f
shows *well-complete* $\{p \in A.\ f\ p = p\}$ $(\sqsubseteq)$
proof−
interpret *fixed-point-proof* **using** f **by** *unfold-locales*
show *?thesis*
apply (*intro antisym-mono-imp-fp-complete mono antisymmetric-axioms comp*)
by (*auto intro*!: *antisym well-related-extend*)
qed

end

end
theory *Kleene-Fixed-Point*
imports *Complete-Relations*
begin

7 Iterative Fixed Point Theorem

Kleene's fixed-point theorem states that, for a pointed directed complete partial order $\langle A, \sqsubseteq \rangle$ and a Scott-continuous map $f : A \to A$, the supremum of $\{f^n(\bot) \mid n \in \mathbb{N}\}$ exists in A and is a least fixed point. Mashburn [12] generalized the result so that $\langle A, \sqsubseteq \rangle$ is a ω-complete partial order and f is ω-continuous.

In this section we further generalize the result and show that for ω-complete relation $\langle A, \sqsubseteq \rangle$ and for every bottom element $\bot \in A$, the set $\{f^n(\bot) \mid n \in \mathbb{N}\}$ has suprema (not necessarily unique, of course) and, they are quasi-fixed points. Moreover, if $(\sqsubseteq)$ is attractive, then the suprema are precisely the least quasi-fixed points.

7.1 Scott Continuity, ω-Completeness, ω-Continuity

In this Section, we formalize ω-completeness, Scott continuity and ω-continuity. We then prove that a Scott continuous map is ω-continuous and that an ω-continuous map is "nearly" monotone.

context
fixes A :: *'a set* **and** *less-eq* :: *'a* ⇒ *'a* ⇒ *bool* (**infix** $\sqsubseteq$ *50*)
begin

definition *omega-continuous* $f \equiv$
f ‘ $A \subseteq A \wedge$
$(\forall\, c :: nat \Rightarrow {}'a.\ \forall\ \ b \in A.$

range $c \subseteq A \longrightarrow$
monotone $(\leq)$ $(\sqsubseteq)$ $c \longrightarrow$
extreme-bound A $(\sqsubseteq)$ (*range* c) $b \longrightarrow$ *extreme-bound* A $(\sqsubseteq)$ (f ‘ *range* c) (f b))

lemmas *omega-continuousI*[*intro?*] =
omega-continuous-def[*unfolded atomize-eq*, *THEN iffD2*, *unfolded conj-imp-eq-imp-imp*, *rule-format*]

lemmas *omega-continuousDdom* =
omega-continuous-def[*unfolded atomize-eq*, *THEN iffD1*, *unfolded conj-imp-eq-imp-imp*, *THEN conjunct1*]

lemmas *omega-continuousD* =
omega-continuous-def[*unfolded atomize-eq*, *THEN iffD1*, *unfolded conj-imp-eq-imp-imp*, *THEN conjunct2*, *rule-format*]

lemmas *omega-continuousE*[*elim*] =
omega-continuous-def[*unfolded atomize-eq*, *THEN iffD1*, *elim-format*, *unfolded conj-imp-eq-imp-imp*, *rule-format*]

lemma *omega-continuous-imp-mono-refl*:
assumes *cont*: *omega-continuous* f
and x: $x \in A$ **and** y: $y \in A$
and xy: $x \sqsubseteq y$ **and** xx: $x \sqsubseteq x$ **and** yy: $y \sqsubseteq y$
shows $f\ x \sqsubseteq f\ y$
proof−
define c :: *nat* $\Rightarrow$ $'a$ **where** $c \equiv \lambda i.$ *if* $i = 0$ *then* x *else* y
from x y xx xy yy **have** c: *range* $c \subseteq A$ *monotone* $(\leq)$ $(\sqsubseteq)$ c
by (*auto simp*: *c-def intro*!: *monotoneI*)
have *extreme-bound* A $(\sqsubseteq)$ (*range* c) y **using** xy yy x y **by** (*auto simp*: *c-def*)
then have *fboy*: *extreme-bound* A $(\sqsubseteq)$ (f ‘ *range* c) (f y) **using** c *cont* y **by** *auto*
then show $f\ x \sqsubseteq f\ y$ **by** (*auto simp*: *c-def*)
qed

definition *scott-continuous* $f \equiv$
f ‘ $A \subseteq A \wedge$
$(\forall X\ s.\ X \subseteq A \longrightarrow$ *directed* X $(\sqsubseteq) \longrightarrow X \neq \{\} \longrightarrow$ *extreme-bound* A $(\sqsubseteq)$ X s
$\longrightarrow$ *extreme-bound* A $(\sqsubseteq)$ (f ‘ X) (f s))

lemmas *scott-continuousI*[*intro?*] =
scott-continuous-def[*unfolded atomize-eq*, *THEN iffD2*, *unfolded conj-imp-eq-imp-imp*, *rule-format*]

lemmas *scott-continuousE* =
scott-continuous-def[*unfolded atomize-eq*, *THEN iffD1*, *elim-format*, *unfolded conj-imp-eq-imp-imp*, *rule-format*]

lemma *scott-continuous-imp-mono-refl*:
assumes *scott*: *scott-continuous* f

and x: $x \in A$ **and** y: $y \in A$ **and** xy: $x \sqsubseteq y$ **and** yy: $y \sqsubseteq y$
shows $f\ x \sqsubseteq f\ y$
proof−
define D **where** $D \equiv \{x,y\}$
from $x\ y\ xy\ yy$ **have** *dir-D*: $D \subseteq A$ *directed* D $(\sqsubseteq)$ $D \neq \{\}$
by (*auto simp*: *D-def intro*!: *bexI*[*of* - y] *directedI*)
have *extreme-bound* A $(\sqsubseteq)$ $D\ y$ **using** $xy\ yy\ x\ y$ **by** (*auto simp*: *D-def*)
then have *fboy*: *extreme-bound* A $(\sqsubseteq)$ $(f\ `\ D)$ $(f\ y)$ **using** *dir-D scott* **by** (*auto elim*!: *scott-continuousE*)
then show $f\ x \sqsubseteq f\ y$ **by** (*auto simp*: *D-def*)
qed

lemma *scott-continuous-imp-omega-continuous*:
assumes *scott*: *scott-continuous* f **shows** *omega-continuous* f
proof
from *scott* **show** $f\ `\ A \subseteq A$ **by** (*auto elim*!: *scott-continuousE*)
fix c :: *nat* $\Rightarrow$ $'a$
assume *mono*: *monotone* $(\leq)$ $(\sqsubseteq)$ c **and** c: *range* $c \subseteq A$
from *monotone-directed-image*[*OF mono*[*folded monotone-on-UNIV*] *order.directed*] *scott* c
show *extreme-bound* A $(\sqsubseteq)$ (*range* c) $b \Longrightarrow$ *extreme-bound* A $(\sqsubseteq)$ $(f\ `\ range\ c)$ $(f\ b)$ **for** b
by (*auto elim*!: *scott-continuousE*)
qed

end

7.2 Existence of Iterative Fixed Points

The first part of Kleene's theorem demands to prove that the set $\{f^n(\bot) \mid n \in \mathbb{N}\}$ has a supremum and that all such are quasi-fixed points. We prove this claim without assuming anything on the relation $\sqsubseteq$ besides ω-completeness and one bottom element.

notation *compower* (-^[*1000*,*1000*]*1000*)

lemma *mono-funpow*: **assumes** f: $f\ `\ A \subseteq A$ **and** *mono*: *monotone-on* A r r f
shows *monotone-on* A r r $(f\hat{}n)$
proof (*induct* n)
case *0*
show *?case* **using** *monotone-on-id* **by** (*auto simp*: *id-def*)
next
case (*Suc* n)
with *funpow-dom*[*OF* f] **show** *?case*
by (*auto intro*!: *monotone-onI monotone-onD*[*OF mono*] *elim*!:*monotone-onE*)
qed

no-notation *bot* $(\bot)$

context

fixes A **and** *less-eq* (**infix** $\sqsubseteq$ *50*) **and** *bot* ($\bot$) **and** f
assumes *bot*: $\bot \in A\ \forall\, q \in A.\ \bot \sqsubseteq q$
assumes *cont*: *omega-continuous* A ($\sqsubseteq$) f
begin

interpretation *less-eq-notations*.

private lemma f: $f\ `\ A \subseteq A$ **using** *cont* **by** *auto*

private abbreviation(*input*) $Fn \equiv \{f\hat{}n\ \bot \mid .\ n :: nat\}$

private lemma *fn-ref*: $f\hat{}n\ \bot \sqsubseteq f\hat{}n\ \bot$ **and** *fnA*: $f\hat{}n\ \bot \in A$
proof (*atomize*(*full*), *induct* n)
case *0*
from *bot* **show** *?case* **by** *simp*
next
case (*Suc* n)
then have *fn*: $f\hat{}n\ \bot \in A$ **and** *fnfn*: $f\hat{}n\ \bot \sqsubseteq f\hat{}n\ \bot$ **by** *auto*
from *f fn omega-continuous-imp-mono-refl*[*OF cont fn fn fnfn fnfn fnfn*]
show *?case* **by** *auto*
qed

private lemma *FnA*: $Fn \subseteq A$ **using** *fnA* **by** *auto*

private lemma *fn-monotone*: *monotone* ($\leq$) ($\sqsubseteq$) ($\lambda n.\ f\hat{}n\ \bot$)
proof
fix $n\ m :: nat$
assume $n \leq m$
from *le-Suc-ex*[*OF this*] **obtain** k **where** m: $m = n + k$ **by** *auto*
from *bot fn-ref fnA omega-continuous-imp-mono-refl*[*OF cont*]
show $f\hat{}n\ \bot \sqsubseteq f\hat{}m\ \bot$ **by** (*unfold* m, *induct* n, *auto*)
qed

private lemma *Fn*: $Fn = range\ (\lambda n.\ f\hat{}n\ \bot)$ **by** *auto*

theorem *kleene-qfp*:
assumes q: *extreme-bound* A ($\sqsubseteq$) $Fn\ q$
shows $f\ q \sim q$
proof
have *fq*: *extreme-bound* A ($\sqsubseteq$) ($f\ `\ Fn$) ($f\ q$)
apply (*unfold Fn*)
apply (*rule omega-continuousD*[*OF cont*])
using *FnA fn-monotone q* **by** (*unfold Fn*, *auto*)
with *bot* **have** *nq*: $f\hat{}n\ \bot \sqsubseteq f\ q$ **for** n
by(*induct* n, *auto simp*: *extreme-bound-iff*)
then show $q \sqsubseteq f\ q$ **using** $f\ q$ **by** *blast*
have $f\ (f\hat{}n\ \bot) \in Fn$ **for** n **by** (*auto intro*!: *exI*[*of - Suc n*])
then have $f\ `\ Fn \subseteq Fn$ **by** *auto*
from *extreme-bound-mono*[*OF this fq q*]

show $f\ q \sqsubseteq q$.
qed

lemma *ex-kleene-qfp*:
assumes *comp*: *omega-complete A* ($\sqsubseteq$)
shows $\exists\, p.$ *extreme-bound A* ($\sqsubseteq$) *Fn p*
using *fn-monotone*
apply (*intro comp*[*unfolded omega-complete-def*, *THEN completeD*, *OF FnA*])
by *fast*

7.3 Iterative Fixed Points are Least.

Kleene's theorem also states that the quasi-fixed point found this way is a least one. Again, attractivity is needed to prove this statement.

lemma *kleene-qfp-is-least*:
assumes *attract*: $\forall\, q \in A.\ \forall\, x \in A.\ f\ q \sim q \longrightarrow x \sqsubseteq f\ q \longrightarrow x \sqsubseteq q$
assumes *q*: *extreme-bound A* ($\sqsubseteq$) *Fn q*
shows *extreme* $\{s \in A.\ f\ s \sim s\}$ ($\sqsupseteq$) *q*
proof(*safe intro*!: *extremeI kleene-qfp*[*OF q*])
from *q*
show $q \in A$ **by** *auto*
fix *c* **assume** *c*: $c \in A$ **and** *cqfp*: $f\ c \sim c$
{
fix *n*::*nat*
have $f\hat{}n\ \bot \sqsubseteq c$
proof(*induct n*)
case *0*
show *?case* **using** *bot c* **by** *auto*
next
case *IH*: (*Suc n*)
have $c \sqsubseteq c$ **using** *attract cqfp c* **by** *auto*
with *IH* **have** $f\hat{}(Suc\ n)\ \bot \sqsubseteq f\ c$
using *omega-continuous-imp-mono-refl*[*OF cont*] *fn-ref fnA c* **by** *auto*
then show *?case* **using** *attract cqfp c fnA* **by** *blast*
qed
}
then show $q \sqsubseteq c$ **using** *q c* **by** *auto*
qed

lemma *kleene-qfp-iff-least*:
assumes *comp*: *omega-complete A* ($\sqsubseteq$)
assumes *attract*: $\forall\, q \in A.\ \forall\, x \in A.\ f\ q \sim q \longrightarrow x \sqsubseteq f\ q \longrightarrow x \sqsubseteq q$
assumes *dual-attract*: $\forall\, p \in A.\ \forall\, q \in A.\ \forall\, x \in A.\ p \sim q \longrightarrow q \sqsubseteq x \longrightarrow p \sqsubseteq x$
shows *extreme-bound A* ($\sqsubseteq$) *Fn* = *extreme* $\{s \in A.\ f\ s \sim s\}$ ($\sqsupseteq$)
proof (*intro ext iffI kleene-qfp-is-least*[*OF attract*])
fix *q*
assume *q*: *extreme* $\{s \in A.\ f\ s \sim s\}$ ($\sqsupseteq$) *q*
from *q* **have** *qA*: $q \in A$ **by** *auto*
from *q* **have** *qq*: $q \sqsubseteq q$ **by** *auto*

```
  from q have fqq: f q ∼ q by auto
  from ex-kleene-qfp[OF comp]
  obtain k where k: extreme-bound A (⊑) Fn k by auto
  have qk: q ∼ k
  proof
    from kleene-qfp[OF k] q k
    show q ⊑ k by auto
    from kleene-qfp-is-least[OF - k] q attract
    show k ⊑ q by blast
  qed
  show extreme-bound A (⊑) Fn q
  proof (intro extreme-boundI,safe)
    fix n
    show f^n ⊥ ⊑ q
    proof (induct n)
      case 0
      from bot q show ?case by auto
    next
      case S:(Suc n)
      from fnA f have fsnbA: f (f^n ⊥) ∈ A by auto
      have fnfn: f^n ⊥ ⊑ f^n ⊥ using fn-ref by auto
      have f (f^n ⊥) ⊑ f q
        using omega-continuous-imp-mono-refl[OF cont fnA qA S fnfn qq] by auto
      then show ?case using fsnbA qA attract fqq by auto
    qed
  next
    fix x
    assume bound Fn (⊑) x and x: x ∈ A
    with k have kx: k ⊑ x by auto
    with dual-attract[rule-format, OF - - x qk] q k
    show q ⊑ x by auto
  next
    from q show q ∈ A by auto
  qed
qed

end

context attractive begin

interpretation less-eq-notations.

theorem kleene-qfp-is-dual-extreme:
  assumes comp: omega-complete A (⊑)
    and cont: omega-continuous A (⊑) f and bA: b ∈ A and bot: ∀ x ∈ A. b ⊑ x
  shows extreme-bound A (⊑) {f^n b |. n :: nat} = extreme {s ∈ A. f s ∼ s} (⊒)
  apply (rule kleene-qfp-iff-least[OF bA bot cont comp])
  using cont[THEN omega-continuousDdom]
  by (auto dest: sym-order-trans order-sym-trans)
```

end

corollary(**in** *antisymmetric*) *kleene-fp*:
assumes *cont*: *omega-continuous A* (⊑) *f*
and *b*: *b* ∈ *A* ∀ *x* ∈ *A*. *b* ⊑ *x*
and *p*: *extreme-bound A* (⊑) {*f*^*n b* |. *n* :: *nat*} *p*
shows *f p* = *p*
using *kleene-qfp*[*OF b cont*] *p cont*[*THEN omega-continuousDdom*]
by (*auto 2 3 intro*!:*antisym*)

no-notation *compower* (-^-[*1000*,*1000*]*1000*)

end

References

[1] S. Abramsky and A. Jung. *Domain Theory.* Number III in Handbook of Logic in Computer Science. Oxford University Press, 1994.

[2] C. Ballarin. Interpretation of locales in Isabelle: Theories and proof contexts. In J. M. Borwein and W. M. Farmer, editors, *Mathematical Knowledge Management*, pages 31–43, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[3] G. M. Bergman. *Lattices, Closure Operators, and Galois Connections*, pages 173–212. Springer International Publishing, Cham, 2015.

[4] S. P. Bhatta. Weak chain-completeness and fixed point property for pseudo-ordered sets. *Czechoslovak Mathematical Journal*, 55(2):365–369, 2005.

[5] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order.* Cambridge University Press, 2002.

[6] G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. W. Mislove, and D. S. Scott. *Continuous Lattices and Domains.* Cambridge University Press, 2003.

[7] G. Gonthier. Formal proof – the four-color theorem. *Notices of the AMS*, 55(11):1382–1393, 2008.

[8] T. Hales, M. Adams, G. Bauer, T. D. Dang, J. Harrison, H. Le Truong, C. Kaliszyk, V. Magron, S. McLaughlin, T. T. Nguyen, et al. A formal proof of the Kepler conjecture. *Forum of Mathematics, Pi*, 5, 2017.

[9] F. Kammüller. Modular reasoning in Isabelle. In D. McAllester, editor, *Automated Deduction - CADE-17*, pages 99–114, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[10] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, et al. seL4: Formal verification of an OS kernel. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems principles*, pages 207–220. ACM, 2009.

[11] K. Leutola and J. Nieminen. Posets and generalized lattices. *Algebra Universalis*, 16(1):344–354, 1983.

[12] J. D. Mashburn. The least fixed point property for omega-chain continuous functions. *Houston Journal of Mathematics*, 9(2):231–244, 1983.

[13] S. Parameshwara Bhatta and S. George. Some fixed point theorems for pseudo ordered sets. *Algebra and Discrete Mathematics*, 11(1):17–22, 2011.

[14] G. Schmidt and T. Ströhlein. *Relations and Graphs: Discrete Mathematics for Computer Scientists*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1993.

[15] H. Skala. Trellis theory. *Algebra Univ.*, 1:218–233, 1971.

[16] A. Stouti and A. Maaden. Fixed points and common fixed points theorems in pseudo-ordered sets. *Proyecciones. Revista de Matemática*, 32(4):409–418, 2013.

[17] M. Wenzel. Isabelle/jEdit–a prover IDE within the PIDE framework. In *International Conference on Intelligent Computer Mathematics*, pages 468–471. Springer, 2012.

[18] A. Yamada and J. Dubut. Complete Non-Orders and Fixed Points. In J. Harrison, J. O'Leary, and A. Tolmach, editors, *10th International Conference on Interactive Theorem Proving (ITP 2019)*, volume 141 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:16, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.