Complete Non-Orders and Fixed Points

Akihisa Yamada and Jérémy Dubut

March 17, 2025

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 with only antisymmetry or attractivity, a mild condition implied by either antisymmetry or transitivity. In particular, we generalize various theorems ensuring the existence of a quasi-fixed point of monotone maps over complete relations, and show that the set of (quasi-)fixed points is itself complete. This result generalizes and strengthens theorems of Knaster–Tarski, Bourbaki–Witt, Kleene, Markowsky, Pataraia, Mashburn, Bhatta–George, and Stouti–Maaden.

Contents

1 Introduction

The main driving force towards mechanizing mathematics using proof assistants has been the reliability they offer, exemplified prominently by [10], [12], [15], etc. In this work, we utilize another aspect of proof assistants: they are also engineering tools for developing mathematical theories.

Fixed-point theorems are important in computer science, such as in denotational semantics [20] and in abstract interpretation [7], as they allow the definition of semantics of loops and recursive functions. The Knaster–Tarski theorem [23] shows that any monotone map $f : A \to A$ over complete lattice $(A, \sqsubseteq)$ has a fixed point, and the set of fixed points forms also a complete lattice. The result was generalized in various ways: Markowsky [16] showed a corresponding result for *chain-complete* posets. The proof uses the Bourbaki–Witt theorem [6], stating that any inflationary map over a chain-complete poset has a fixed point. The original proof of the latter is non-elementary in the sense that it relies on ordinals and Hartogs' theorem. Pataraia [18] gave an elementary proof that monotone maps over *pointed directed-complete* poset has a fixed point. Fixed points are studied also for *pseudo-orders* [21], relaxing transitivity. Stouti and Maaden [22] showed that every monotone map over a complete pseudo-order has a (least) fixed point. Markowsky's result was also generalized to *weak* chain-complete pseudo-orders by Bhatta and George [4, 5].

Another line of order-theoretic fixed points is the *iterative* approach. Kantorovitch showed that for *ω-continuous* map f over a complete lattice,[1] the iteration $\bot, f\ \bot, f^2\ \bot, \ldots$ converges to a fixed point [14, Theorem I]. Tarski [23] also claimed a similar result for a *countably distributive* map over a *countably complete Boolean algebra*. Kleene's fixed-point theorem states that, for *Scott-continuous* maps over pointed directed-complete posets, the iteration converges to the least fixed point. Finally, Mashburn [17] proved a version for ω-continuous maps over ω-complete posets, which covers Kantorovitch's, Tarski's and Kleene's claims.

In particular, we provide the following:

- Several *locales* that help organizing the different order-theoretic conditions, such as reflexivity, transitivity, antisymmetry, and their combination, as well as concepts such as connex and well-related sets, analogues of chains and well-ordered sets in a non-ordered context.

- Existence of fixed points: We provide two proof schemes to prove that monotone or inflationary mapping $f : A \to A$ over a complete related set $\langle A, \sqsubseteq \rangle$ has a *quasi-fixed point* $f\ x \sim x$, meaning $x \sqsubseteq f\ x \wedge f\ x \sqsubseteq x$, for various notions of completeness. The first one, similar to the original proof by Tarski [23], does not require any ordering assumptions, but relies on completeness with respect to all subsets. The second one, inspired by a *constructive* approach by Grall [11], is a proof scheme based on the notion of derivations. Here we demand antisymmetry (to avoid the necessity of the axiom of choice), but can be instantiated to *well-complete* sets, a generalization of weak chain-completeness. This also allows us to generalize Bourbaki–Witt theorem [6] to pseudo-orders.

- Completeness of the set of fixed points: if $(A, \sqsubseteq)$ satisfies a mild condition, which we call *attractivity* and which is implied by either transitivity or antisymmetry, then the set of quasi-fixed points inherits the completeness class from $(A, \sqsubseteq)$, if it is at least well-complete. The result instantiates to the full completeness (generalizing Knaster–Tarski and [22]), directed-completeness [18], chain-completeness [16], and weak chain-completeness [5].

[1] More precisely, he assumes a conditionally complete lattice defined over vectors and that $\bot \sqsubseteq f\ \bot$ and $f\ v' \sqsubseteq v'$. Hence f, which is monotone, is a map over the complete lattice $\{v \mid \bot \sqsubseteq v \sqsubseteq v'\}$.

- Iterative construction: For an ω-continuous map over an ω-complete related set, we show that suprema of $\{f^n \perp \mid n \in \mathbb{N}\}$ are quasi-fixed points. Under attractivity, the quasi-fixed points obtained from this scheme are precisely the least quasi-fixed points of f. This generalizes Mashburn's result, and thus ones by Kantorovitch, Tarski and Kleene.

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 aforementioned 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.

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 archive is the third version of this work. The first version has been published in the conference paper [24]. The second version has been published in the journal paper [8]. The third version is a restructuration of the second version for future formalizations, including [25].

2 Binary Relations

We start with basic properties of binary relations.

theory *Binary-Relations*
 imports

  Main
begin

unbundle *lattice-syntax*

lemma *conj-iff-conj-iff-imp-iff*: *Trueprop* $(x \wedge y \longleftrightarrow x \wedge z) \equiv (x \Longrightarrow (y \longleftrightarrow z))$
 by (*auto intro*!: *equal-intr-rule*)

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 ‘ $A \subseteq A \Longrightarrow (f\hat{}\hat{}n)$ ‘ $A \subseteq A$ **by** (*induct n*, *auto*)

lemma *image-subsetD*: f ‘ $A \subseteq B \Longrightarrow a \in A \Longrightarrow f\ a \in B$ **by** *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*{- /|./ (-/ ∈ -)})›)
syntax-consts
-range ⇌ *range* **and**
-image ⇌ *image*
translations
{*e* |. *p*} ⇌ *CONST range* (λ*p*. *e*)
{*e* |. *p* ∈ *A*} ⇌ *CONST image* (λ*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* (‹(-⁻)› [*1000*] *1000*) **where** $r^-\ x\ y \equiv r\ y\ x$

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

lemma *dual-inf*: $(r \sqcap s)^- = r^- \sqcap s^-$ **by** (*auto intro*!: *ext*)

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

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

lemma *monotone-on-dual*: *monotone-on X r s f* ⟹ *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* ≤ *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* ≡ *r x y* ∧ *r y x*

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

lemma *sympartpE*[*elim*]:
  fixes *r* (**infix** ‹⊑› *50*)
  assumes *sympartp* (⊑) *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 *sympartp-sympartp*[*simp*]: *sympartp* (*sympartp r*) = *sympartp r* **by** (*auto intro*!:*ext*)

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

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

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* ⟹ *equivpartp r x y*
  and *equivpartpCI*[*intro*]: ($x \neq y \Longrightarrow$ *sympartp r x y*) ⟹ *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* ⟹ *r y x* ⟹ *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** ‹⊑› *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*, *unfolded 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*)

lemma *asympartp-dual*: *asympartp* r^{-} = (*asympartp* $r)^{-}$ **by** *auto*

Restriction to a set:

definition *Restrp* (**infixl** ‹↾› *60*) **where** $(r \upharpoonright A)\ a\ b \equiv a \in A \wedge b \in A \wedge r\ a\ b$

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-simp*[*simp*]: $a \in A \Longrightarrow b \in A \Longrightarrow (r \upharpoonright A)\ a\ b \longleftrightarrow r\ a\ b$ **by** *auto*

lemma *Restrp-UNIV*[*simp*]: $r \upharpoonright UNIV \equiv r$ **by** (*auto simp*: *atomize-eq*)

lemma *Restrp-Restrp*[*simp*]: $r \upharpoonright A \upharpoonright B \equiv r \upharpoonright A \cap B$ **by** (*auto simp*: *atomize-eq Restrp-def*)

lemma *sympartp-Restrp*[*simp*]: *sympartp* $(r \upharpoonright A) \equiv$ *sympartp* $r \upharpoonright A$
by (*auto simp*: *atomize-eq*)

Relational images:

definition *Imagep* (**infixr** ‹‘‘‘› *59*) **where** r ‘‘‘ $A \equiv \{b.\ \exists\, a \in A.\ r\ a\ b\}$

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

lemma *in-Imagep*: $b \in r$ ‘‘‘ $A \longleftrightarrow (\exists\, a \in A.\ r\ a\ b)$ **by** (*auto simp*: *Imagep-def*)

lemma *ImagepI*: $a \in A \Longrightarrow r\ a\ b \Longrightarrow b \in r$ ‘‘‘ A **by** (*auto simp*: *in-Imagep*)

lemma *subset-Imagep*: $B \subseteq r$ ‘‘‘ $A \longleftrightarrow (\forall\, b {\in} B.\ \exists\, a {\in} A.\ r\ a\ b)$
by (*auto simp*: *Imagep-def*)

Bounds of a set:

definition *bound* X $(\sqsubseteq)$ $b \equiv \forall x \in X.\ x \sqsubseteq b$ **for** r (**infix** ‹$\sqsubseteq$› *50*)

lemma
fixes r (**infix** ‹$\sqsubseteq$› *50*)
shows *boundI*[*intro*!]: $(\bigwedge x.\ x \in X \Longrightarrow x \sqsubseteq b) \Longrightarrow$ *bound* X $(\sqsubseteq)$ b
and *boundE*[*elim*]: *bound* X $(\sqsubseteq)$ $b \Longrightarrow ((\bigwedge x.\ x \in X \Longrightarrow x \sqsubseteq b) \Longrightarrow$ *thesis*$) \Longrightarrow$ *thesis*
and *boundD*: *bound* X $(\sqsubseteq)$ $b \Longrightarrow a \in X \Longrightarrow a \sqsubseteq b$
by (*auto simp*: *bound-def*)

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

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

lemmas *bound-subset* = *bound-cmono*[*THEN le-funD*, *THEN le-funD*, *THEN le-boolD*, *folded atomize-imp*]

lemma *bound-un*: *bound* $(A \cup B) =$ *bound* $A \sqcap$ *bound* B
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*

lemma *bound-cong*:
assumes $A = A'$
and $b = b'$
and $\bigwedge a.\ a \in A' \Longrightarrow le\ a\ b' = le'\ a\ b'$
shows *bound* A *le* $b =$ *bound* A' le' b'
by (*auto simp*: *assms*)

lemma *bound-subsel*: $le \leq le' \Longrightarrow$ *bound* A *le* $\leq$ *bound* A le'
by (*auto simp add*: *bound-def*)

Extreme (greatest) elements in a set:

definition *extreme* X $(\sqsubseteq)$ $e \equiv e \in X \wedge (\forall x \in X.\ x \sqsubseteq e)$ **for** r (**infix** ‹$\sqsubseteq$› *50*)

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* (⊑) *t* ⟷ (∀ *x*. *x* ⊑ *t*) **by** *auto*

lemma *extreme-iff-bound*: *extreme X r e* ⟷ *bound X r e* ∧ *e* ∈ *X* **by** *auto*

lemma *extreme-imp-bound*: *extreme X r x* ⟹ *bound X r x* **by** *auto*

lemma *extreme-inf*: *extreme X* (*r* ⊓ *s*) *x* ⟷ *extreme X r x* ∧ *extreme X s x* **by** *auto*

lemma *extremes-equiv*: *extreme X r b* ⟹ *extreme X r c* ⟹ *sympartp r b c* **by** *blast*

lemma *extreme-cong*:
assumes *A* = *A′*
and *b* = *b′*
and ⋀*a*. *a* ∈ *A′* ⟹ *b′* ∈ *A′* ⟹ *le a b′* = *le′ a b′*
shows *extreme A le b* = *extreme A′ le′ b′*
by (*auto simp*: *assms extreme-def*)

lemma *extreme-subset*: *X* ⊆ *Y* ⟹ *extreme X r x* ⟹ *extreme Y r y* ⟹ *r x y* **by** *blast*

lemma *extreme-subrel*:
le ≤ *le′* ⟹ *extreme A le* ≤ *extreme A le′* **by** (*auto simp*: *extreme-def*)

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

definition
extreme-bound A (⊑) *X* ≡ *extreme* {*b* ∈ *A*. *bound X* (⊑) *b*} (⊑)$^-$ **for** *r* (**infix** ‹⊑› *50*)

lemmas *extreme-boundI-extreme* = *extreme-bound-def*[*unfolded atomize-eq*, *THEN fun-cong*, *THEN iffD2*]

lemmas *extreme-boundD-extreme* = *extreme-bound-def*[*unfolded atomize-eq*, *THEN fun-cong*, *THEN iffD1*]

context
fixes *A* :: ′*a set* **and** *less-eq* :: ′*a* ⇒ ′*a* ⇒ *bool* (**infix** ‹⊑› *50*)
begin

lemma *extreme-boundI*[*intro*]:
assumes ⋀*b*. *bound X* (⊑) *b* ⟹ *b* ∈ *A* ⟹ *s* ⊑ *b* **and** ⋀*x*. *x* ∈ *X* ⟹ *x* ⊑ *s* **and** *s* ∈ *A*
shows *extreme-bound A* (⊑) *X s*
using *assms* **by** (*auto simp*: *extreme-bound-def*)

lemma *extreme-boundD*:
assumes *extreme-bound A* (⊑) *X s*

shows $x \in X \Longrightarrow x \sqsubseteq s$
and *bound* X $(\sqsubseteq)$ $b \Longrightarrow b \in A \Longrightarrow s \sqsubseteq b$
and *extreme-bound-in*: $s \in A$
using *assms* **by** (*auto simp*: *extreme-bound-def*)

lemma *extreme-boundE*[*elim*]:
assumes *extreme-bound* A $(\sqsubseteq)$ X s
and $s \in A \Longrightarrow$ *bound* X $(\sqsubseteq)$ $s \Longrightarrow (\bigwedge b.$ *bound* X $(\sqsubseteq)$ $b \Longrightarrow b \in A \Longrightarrow s \sqsubseteq b)$ $\Longrightarrow$ *thesis*
shows *thesis*
using *assms* **by** (*auto simp*: *extreme-bound-def*)

lemma *extreme-bound-imp-bound*: *extreme-bound* A $(\sqsubseteq)$ X $s \Longrightarrow$ *bound* X $(\sqsubseteq)$ s
by *auto*

lemma *extreme-imp-extreme-bound*:
assumes *Xs*: *extreme* X $(\sqsubseteq)$ s **and** *XA*: $X \subseteq A$ **shows** *extreme-bound* A $(\sqsubseteq)$ X s
using *assms* **by** *force*

lemma *extreme-bound-subset-bound*:
assumes *XY*: $X \subseteq Y$
and *sX*: *extreme-bound* A $(\sqsubseteq)$ X s
and *b*: *bound* Y $(\sqsubseteq)$ b **and** *bA*: $b \in A$
shows $s \sqsubseteq b$
using *bound-subset*[*OF XY b*] *sX bA* **by** *auto*

lemma *extreme-bound-subset*:
assumes *XY*: $X \subseteq Y$
and *sX*: *extreme-bound* A $(\sqsubseteq)$ X *sX*
and *sY*: *extreme-bound* A $(\sqsubseteq)$ Y *sY*
shows $sX \sqsubseteq sY$
using *extreme-bound-subset-bound*[*OF XY sX*] *sY* **by** *auto*

lemma *extreme-bound-iff*:
extreme-bound A $(\sqsubseteq)$ 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-bound-def extreme-def*)

lemma *extreme-bound-empty*: *extreme-bound* A $(\sqsubseteq)$ $\{\}$ $x \longleftrightarrow$ *extreme* A $(\sqsubseteq)^-$ x
by *auto*

lemma *extreme-bound-singleton-refl*[*simp*]:
extreme-bound A $(\sqsubseteq)$ $\{x\}$ $x \longleftrightarrow x \in A \wedge x \sqsubseteq x$ **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* A $(\sqsubseteq)$ $(f$ ‘ $I)$ 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 A $(\sqsubseteq)$ $(\bigcup i{\in}I.\ \{y.\ P\ i\ y\})$ x
  by *auto*

lemma *extreme-bounds-equiv*:
  assumes *s*: *extreme-bound* A $(\sqsubseteq)$ X s **and** s': *extreme-bound* A $(\sqsubseteq)$ X s'
  shows *sympartp* $(\sqsubseteq)$ s s'
  using s s'
  apply (*unfold extreme-bound-def*)
  apply (*subst sympartp-dual*)
  by (*rule extremes-equiv*)

lemma *extreme-bound-sqweeze*:
  assumes *XY*: $X \subseteq Y$ **and** *YZ*: $Y \subseteq Z$
    and *Xs*: *extreme-bound* A $(\sqsubseteq)$ X s **and** *Zs*: *extreme-bound* A $(\sqsubseteq)$ Z s
  shows *extreme-bound* A $(\sqsubseteq)$ Y s
proof
  from *Xs* **show** $s \in A$ **by** *auto*
  fix b **assume** *Yb*: *bound* Y $(\sqsubseteq)$ b **and** *bA*: $b \in A$
  from *bound-subset*[*OF XY Yb*] **have** *bound* X $(\sqsubseteq)$ b**.**
  with *Xs bA*
  show $s \sqsubseteq b$ **by** *auto*
next
  fix y **assume** *yY*: $y \in Y$
  with *YZ* **have** $y \in Z$ **by** *auto*
  with *Zs* **show** $y \sqsubseteq s$ **by** *auto*
qed

lemma *bound-closed-imp-extreme-bound-eq-extreme*:
  assumes *closed*: $\forall\, b \in A.$ *bound* X $(\sqsubseteq)$ $b \longrightarrow b \in X$ **and** *XA*: $X \subseteq A$
  shows *extreme-bound* A $(\sqsubseteq)$ X = *extreme* X $(\sqsubseteq)$
proof (*intro ext iffI extreme-boundI extremeI*)
  fix e
  assume *extreme-bound* A $(\sqsubseteq)$ X e
  then have *Xe*: *bound* X $(\sqsubseteq)$ e **and** $e \in A$ **by** *auto*
  with *closed* **show** $e \in X$ **by** *auto*
  fix x **assume** $x \in X$
  with *Xe* **show** $x \sqsubseteq e$ **by** *auto*
next
  fix e
  assume *Xe*: *extreme* X $(\sqsubseteq)$ e
  then have *eX*: $e \in X$ **by** *auto*
  with *XA* **show** $e \in A$ **by** *auto*
  { fix b **assume** *Xb*: *bound* X $(\sqsubseteq)$ b **and** $b \in A$
    from *eX Xb* **show** $e \sqsubseteq b$ **by** *auto*
  }
  fix x **assume** *xX*: $x \in X$ **with** *Xe* **show** $x \sqsubseteq e$ **by** *auto*

qed

end

lemma *extreme-bound-cong*:
  assumes $A = A'$
    and $X = X'$
    and $\bigwedge a\ b.\ a \in A' \Longrightarrow b \in A' \Longrightarrow le\ a\ b \longleftrightarrow le'\ a\ b$
    and $\bigwedge a\ b.\ a \in X' \Longrightarrow b \in A' \Longrightarrow le\ a\ b \longleftrightarrow le'\ a\ b$
  shows $extreme\text{-}bound\ A\ le\ X\ s = extreme\text{-}bound\ A\ le'\ X\ s$
  apply (*unfold extreme-bound-def*)
  apply (*rule extreme-cong*)
  by (*auto simp*: *assms*)

Maximal or Minimal

definition $extremal\ X\ (\sqsubseteq)\ x \equiv x \in X \wedge (\forall y \in X.\ x \sqsubseteq y \longrightarrow y \sqsubseteq x)$ **for** *r* (**infix** ‹⊑› *50*)

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

lemma *extremalI*:
  assumes $x \in X\ \bigwedge y.\ y \in X \Longrightarrow x \sqsubseteq y \Longrightarrow y \sqsubseteq x$
  shows $extremal\ X\ (\sqsubseteq)\ x$
  using *assms* **by** (*auto simp*: *extremal-def*)

lemma *extremalE*:
  assumes $extremal\ X\ (\sqsubseteq)\ x$
    and $x \in X \Longrightarrow (\bigwedge y.\ y \in X \Longrightarrow x \sqsubseteq y \Longrightarrow y \sqsubseteq x) \Longrightarrow thesis$
  shows *thesis*
  using *assms* **by** (*auto simp*: *extremal-def*)

lemma *extremalD*:
  assumes $extremal\ X\ (\sqsubseteq)\ x$ **shows** $x \in X\ y \in X \Longrightarrow x \sqsubseteq y \Longrightarrow y \sqsubseteq x$
  using *assms* **by** (*auto elim*!: *extremalE*)

end

context
  fixes *ir* (**infix** ‹⪯› *50*) **and** *r* (**infix** ‹⊑› *50*) **and** *I f*
  assumes *mono*: $monotone\text{-}on\ I\ (\preceq)\ (\sqsubseteq)\ f$
begin

lemma *monotone-image-bound*:
  assumes $X \subseteq I$ **and** $b \in I$ **and** $bound\ X\ (\preceq)\ b$
  shows $bound\ (f\ `\ X)\ (\sqsubseteq)\ (f\ b)$
  using *assms monotone-onD*[*OF mono*]
  by (*auto simp*: *bound-def*)

```
lemma monotone-image-extreme:
  assumes e: extreme I (≼) e
  shows extreme (f ‘ I) (⊑) (f e)
  using e[unfolded extreme-iff-bound] monotone-image-bound[of I e] by auto

end

context
  fixes ir :: 'i ⇒ 'i ⇒ bool (infix ‹≼› 50)
    and r :: 'a ⇒ 'a ⇒ bool (infix ‹⊑› 50)
    and f and A and e and I
  assumes fIA: f ‘ I ⊆ A
    and mono: monotone-on I (≼) (⊑) f
    and e: extreme I (≼) e
begin

lemma monotone-extreme-imp-extreme-bound:
  extreme-bound A (⊑) (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 ⟹ extreme-bound A (⊑) (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* [13, 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 ⇒ 'a ⇒ bool (infix ‹⊑› 50)

locale less-syntax =
  fixes less :: 'a ⇒ 'a ⇒ bool (infix ‹⊏› 50)

locale equivalence-syntax =
  fixes equiv :: 'a ⇒ 'a ⇒ bool (infix ‹∼› 50)
begin

abbreviation equiv-class (‹[-]∼›) where [x]∼ ≡ { y. x ∼ 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-dualize* = *less-eq-syntax*
begin

abbreviation (*input*) *greater-eq* (**infix** ‹$\sqsupseteq$› *50*) **where** $x \sqsupseteq y \equiv y \sqsubseteq x$

end

locale *less-eq-symmetrize* = *less-eq-dualize*
begin

abbreviation *sym* (**infix** ‹$\sim$› *50*) **where** $(\sim) \equiv$ *sympartp* $(\sqsubseteq)$
abbreviation *equiv* (**infix** ‹$(\simeq)$› *50*) **where** $(\simeq) \equiv$ *equivpartp* $(\sqsubseteq)$

end

locale *less-eq-asymmetrize* = *less-eq-symmetrize*
begin

abbreviation *less* (**infix** ‹$\sqsubset$› *50*) **where** $(\sqsubset) \equiv$ *asympartp* $(\sqsubseteq)$
abbreviation *greater* (**infix** ‹$\sqsupset$› *50*) **where** $(\sqsupset) \equiv (\sqsubset)^{-}$

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-dualize* = *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 *reflexive-subset*: $B \subseteq A \Longrightarrow$ *reflexive* B $(\sqsubseteq)$ **apply** *unfold-locales* **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 *extreme-bound-cone*: $x \in A \Longrightarrow$ *extreme-bound* A $(\sqsubseteq)$ $\{a \in A.\ a \sqsubseteq x\}$ x
by *auto*

end

lemmas *reflexiveI*[*intro*!] = *reflexive.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** ‹⊏› *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

lemmas *irreflexiveI*[*intro*!] = *irreflexive.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*)

context *reflexive* **begin**

interpretation *less-eq-asymmetrize***.**

lemma *asympartp-irreflexive*: *irreflexive A* $(\sqsubset)$ **by** *auto*

end

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

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 *extreme-bound-mono*:
  assumes *XY*: $\forall x{\in}X.\ \exists y{\in}Y.\ x \sqsubseteq y$ **and** *XA*: $X \subseteq A$ **and** *YA*: $Y \subseteq A$
    and *sX*: *extreme-bound A* $(\sqsubseteq)$ *X sX*
    and *sY*: *extreme-bound A* $(\sqsubseteq)$ *Y sY*
  shows $sX \sqsubseteq sY$
proof (*intro extreme-boundD*(*2*)[*OF sX*] *CollectI conjI boundI*)
  from *sY* **show** *sYA*: $sY \in A$ **by** *auto*
  from *sY* **have** *bound Y* $(\sqsubseteq)$ *sY* **by** *auto*
  fix *x* **assume** *xX*: $x \in X$ **with** *XY* **obtain** *y* **where** *yY*: $y \in Y$ **and** *xy*: $x \sqsubseteq y$ **by** *auto*
  from *yY sY* **have** $y \sqsubseteq sY$ **by** *auto*
  from *trans*[*OF xy this*] *xX XA yY YA sYA* **show** $x \sqsubseteq sY$ **by** *auto*
qed

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* (*asympartp* $(\sqsubseteq)$)
  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

lemmas *transitiveI* = *transitive.intro*

lemma *transitive-ball*[*code*]:
  transitive A (⊑) ⟷ (∀ *x* ∈ *A*. ∀ *y* ∈ *A*. ∀ *z* ∈ *A*. *x* ⊑ *y* ⟶ *y* ⊑ *z* ⟶ *x* ⊑ *z*)
  for *less-eq* (**infix** ‹⊑› *50*)
  by (*auto simp*: *transitive-def*)

lemma *transitive-cong*:
  assumes *r*: ⋀*a b*. *a* ∈ *A* ⟹ *b* ∈ *A* ⟹ *r a b* ⟷ *r′ a b* **shows** *transitive A r* ⟷ *transitive A r′*
proof (*intro iffI*)
  show *transitive A r* ⟹ *transitive A r′*
    apply (*intro transitive.intro*)
    apply (*unfold r*[*symmetric*])
    using *transitive.trans*.
  show *transitive A r′* ⟹ *transitive A r*
    apply (*intro transitive.intro*)
    apply (*unfold r*)
    using *transitive.trans*.
qed

lemma *transitive-empty*[*intro*!]: *transitive* {} *r* **by** (*auto intro*!: *transitive.intro*)

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

lemmas *symmetricI*[*intro*] = *symmetric.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*)

lemma *symmetric-empty*[*intro*!]: *symmetric* {} *r* **by** *auto*

global-interpretation *sympartp*: *symmetric UNIV sympartp r*
rewrites $\bigwedge r.\ r \upharpoonright UNIV \equiv r$
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 *auto*

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

locale *antisymmetric* = *related-set* +
assumes *antisym*: $x \sqsubseteq y \Longrightarrow y \sqsubseteq x \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow x = y$
begin

interpretation *less-eq-symmetrize***.**

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

lemma *equiv-iff-eq*[*simp*]: $x \in A \Longrightarrow y \in A \Longrightarrow x \simeq y \longleftrightarrow x = y$ **by** (*auto dest*: *antisym elim*: *equivpartpE*)

lemma *extreme-unique*: $X \subseteq A \Longrightarrow extreme\ X\ (\sqsubseteq)\ x \Longrightarrow extreme\ X\ (\sqsubseteq)\ y \longleftrightarrow x = y$
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 *eq-The-extreme*: $X \subseteq A \Longrightarrow extreme\ X\ (\sqsubseteq)\ x \Longrightarrow x = The\ (extreme\ X\ (\sqsubseteq))$
by (*rule the1-equality*[*symmetric*], *auto simp*: *ex-extreme-iff-ex1*[*symmetric*])

lemma *Restrp-antisymmetric*: *antisymmetric UNIV* $((\sqsubseteq)\upharpoonright A)$
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

lemmas *antisymmetricI*[*intro*] = *antisymmetric.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-empty*[*intro*!]: *antisymmetric* {} *r* **by** *auto*

lemma *antisymmetric-union*:
  fixes *less-eq* (**infix** ‹⊑› *50*)
  assumes *A*: *antisymmetric* *A* (⊑) **and** *B*: *antisymmetric* *B* (⊑)
    and *AB*: $\forall a \in A.\ \forall b \in B.\ a \sqsubseteq b \longrightarrow b \sqsubseteq a \longrightarrow a = b$
  shows *antisymmetric* $(A \cup B)$ (⊑)
proof−
  interpret *A*: *antisymmetric* *A* (⊑) **using** *A*.
  interpret *B*: *antisymmetric* *B* (⊑) **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-symmetrize*.

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* (≃)
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 (∼)
proof unfold-locales
  fix x y z
  assume x: x ∈ A and y: y ∈ A and z: z ∈ A and xy: x ∼ y and yz: y ∼ z
  show x ∼ 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 ⊆ A and x: x ∈ A and C0: C ≠ {} and const: ∀ y ∈ C. y ∼ x
  shows extreme-bound A (⊑) C x
proof (intro extreme-boundI x)
  from C0 obtain c where cC: c ∈ C by auto
  with C have c: c ∈ A by auto
  from cC const have cx: c ∼ x by auto
  fix b assume b: b ∈ A and bound C (⊑) b
  with cC have cb: c ⊑ b by auto
  from attract[OF - - cb x c b] cx show x ⊑ b by auto
next
  fix c assume c ∈ C
  with const show c ⊑ x by auto
qed

lemma extreme-bound-quasi-const-iff:
  assumes C: C ⊆ A and x: x ∈ A and y: y ∈ A and C0: C ≠ {} and const:
∀ z ∈ C. z ∼ x
  shows extreme-bound A (⊑) C y ⟷ x ∼ y
proof (intro iffI)
  assume y: extreme-bound A (⊑) C y
  note x = extreme-bound-quasi-const[OF C x C0 const]
  from extreme-bounds-equiv[OF y x]
  show x ∼ y by auto
next
  assume xy: x ∼ y
  with const C sym.trans[OF - xy - x y] have Cy: ∀ z ∈ C. z ∼ y by auto
  show extreme-bound A (⊑) C y
    using extreme-bound-quasi-const[OF C y C0 Cy].
qed

lemma Restrp-semiattractive: semiattractive UNIV ((⊑)↾A)
  apply unfold-locales
  by (auto dest: attract)

lemma semiattractive-subset: B ⊆ A ⟹ semiattractive B (⊑)
  apply unfold-locales using attract by blast

end
```

lemmas *semiattractiveI* = *semiattractive.intro*

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

lemma *semiattractive-empty*[*intro*!]: *semiattractive* {} *r*
  by (*auto intro*!: *semiattractiveI*)

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

interpretation *less-eq-symmetrize*.

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*

lemma *extreme-bound-sym-trans*:
  assumes *XA*: $X \subseteq A$ **and** *Xx*: *extreme-bound A* $(\sqsubseteq)$ *X x*
    and *xy*: $x \sim y$ **and** *yA*: $y \in A$
  shows *extreme-bound A* $(\sqsubseteq)$ *X y*

proof (*intro extreme-boundI yA*)
  from *Xx* **have** *xA*: $x \in A$ **by** *auto*
  {
    fix *b* **assume** *Xb*: *bound* X $(\sqsubseteq)$ b **and** *bA*: $b \in A$
    with *Xx* **have** *xb*: $x \sqsubseteq b$ **by** *auto*
    from *sym-order-trans*[*OF* - *xb yA xA bA*] *xy* **show** $y \sqsubseteq b$ **by** *auto*
  }
  fix *a* **assume** *aX*: $a \in X$
  with *Xx* **have** *ax*: $a \sqsubseteq x$ **by** *auto*
  from *aX XA* **have** *aA*: $a \in A$ **by** *auto*
  from *order-sym-trans*[*OF ax xy aA xA yA*] **show** $a \sqsubseteq y$ **by** *auto*
qed

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

lemma *Restrp-attractive*: *attractive UNIV* $((\sqsubseteq)\upharpoonright A)$
  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

lemmas *attractiveI* = *attractive.intro*[*OF* - *attractive-axioms.intro*]

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*)

lemma *attractive-empty*[*intro*!]: *attractive* {} *r*
  by (*auto intro*!: *attractiveI*)

context *antisymmetric* **begin**

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

end

context *transitive* **begin**

sublocale *attractive*
  rewrites $\bigwedge r.$ *sympartp* $(r \upharpoonright A) \equiv$ *sympartp* $r \upharpoonright A$
    and $\bigwedge r.$ *sympartp* (*sympartp* r) $\equiv$ *sympartp* r
    and *sympartp* $(\sqsubseteq)^{-} \equiv$ *sympartp* $(\sqsubseteq)$

and (*sympartp* (⊑))$^-$ ≡ *sympartp* (⊑)
and (*sympartp* (⊑) ↾ *A*)$^-$ ≡ *sympartp* (⊑) ↾ *A*
and *asympartp* (*asympartp* (⊑)) = *asympartp* (⊑)
and *asympartp* (*sympartp* (⊑)) = *bot*
and *asympartp* (⊑) ↾ *A* = *asympartp* ((⊑) ↾ *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* ⊏ *y* ⟹ *y* ⊏ *x* ⟹ *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *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* ⊆ *A* ⟹ *asymmetric B* (⊏)
apply *unfold-locales* **using** *asym* **by** *auto*

end

lemmas *asymmetricI* = *asymmetric.intro*

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*!:*asymmetricI 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* $A\ r \longleftrightarrow$ *asymmetric* $A\ r'$
by (*simp add*: *asymmetric-iff-irreflexive-antisymmetric r cong*: *irreflexive-cong antisymmetric-cong*)

lemma *asymmetric-empty*: *asymmetric* {} *r*
by (*auto simp*: *asymmetric-iff-irreflexive-antisymmetric*)

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

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

end

lemmas *quasi-ordered-setI* = *quasi-ordered-set.intro*

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*)

lemma *quasi-ordered-set-empty*[*intro*!]: *quasi-ordered-set* {} *r*
by (*auto intro*!: *quasi-ordered-set.intro*)

lemma *rtranclp-quasi-ordered*: *quasi-ordered-set* *A* (*rtranclp r*)
by (*unfold-locales*, *auto*)

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

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

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

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

end

lemmas *near-ordered-setI* = *near-ordered-set.intro*

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*)

lemma *near-ordered-set-empty*[*intro*!]: *near-ordered-set* {} *r*
by (*auto intro*!: *near-ordered-set.intro*)

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

interpretation *less-eq-symmetrize*.

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

lemmas *pseudo-ordered-setI* = *pseudo-ordered-set.intro*

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*)

lemma *pseudo-ordered-set-empty*[*intro*!]: *pseudo-ordered-set* {} *r*
by (*auto intro*!: *pseudo-ordered-setI*)

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

lemmas *partially-ordered-setI* = *partially-ordered-set.intro*

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*)

lemma *partially-ordered-set-empty*[*intro*!]: *partially-ordered-set* {} *r*
  by (*auto intro*!: *partially-ordered-setI*)

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

lemmas *strict-ordered-setI* = *strict-ordered-set.intro*

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*)

lemma *strict-ordered-set-empty*[*intro*!]: *strict-ordered-set* {} *r*
  by (*auto intro*!: *strict-ordered-set.intro*)

locale *tolerance* = *symmetric* + *reflexive A* (∼)
begin

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

end

lemmas *toleranceI* = *tolerance.intro*

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* ⟷ *tolerance A r′*
  by (*simp add*: *tolerance-def r cong*: *reflexive-cong symmetric-cong*)

lemma *tolerance-empty*[*intro*!]: *tolerance* {} *r* **by** (*auto intro*!: *toleranceI*)

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* (∼)
begin

sublocale *transitive A* (∼)
  rewrites *sympartp* (∼)↾*A* ≡ (∼)↾*A*
    and *sympartp* ((∼)↾*A*) ≡ (∼)↾*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* (∼)
  apply (*intro partial-equivalence.intro partial-equivalence-axioms.intro*)
  using *symmetric-subset transitive-subset* **by** *auto*

end

lemmas *partial-equivalenceI* = *partial-equivalence.intro*[*OF - partial-equivalence-axioms.intro*]

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* ⟷ *partial-equivalence A r′*
  by (*simp add*: *partial-equivalence-def partial-equivalence-axioms-def r*

cong: *transitive-cong symmetric-cong*)

lemma *partial-equivalence-empty*[*intro*!]: *partial-equivalence* {} *r*
by (*auto intro*!: *partial-equivalenceI*)

locale *equivalence* = *symmetric* + *reflexive* A (∼) + *transitive* A (∼)
begin

sublocale *tolerance* + *partial-equivalence* + *quasi-ordered-set* A (∼)**..**

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

end

lemmas *equivalenceI* = *equivalence.intro*

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** ‹⋈› *50*)
begin

proposition *reflexive-irreflexive-is-empty*:
assumes *r*: *reflexive* A (⋈) **and** *ir*: *irreflexive* A (⋈)
shows $A = \{\}$
proof (*rule ccontr*)
interpret *irreflexive* A (⋈) **using** *ir***.**
interpret *reflexive* A (⋈) **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 (⋈) **and** *as*: *antisymmetric* A (⋈)
shows (⋈)$\upharpoonright A \le (=)$
proof−
interpret *symmetric* A (⋈) + *antisymmetric* A (⋈) **using** *assms* **by** *auto*
show *?thesis* **using** *antisym* **by** (*auto dest*: *sym*)
qed

proposition *nontolerance*:
  shows *irreflexive A* (⋈) ∧ *symmetric A* (⋈) ⟷ *tolerance A* (λ*x y*. ¬ *x* ⋈ *y*)
proof (*intro iffI conjI*, *elim conjE*)
  assume *irreflexive A* (⋈) **and** *symmetric A* (⋈)
  then interpret *irreflexive A* (⋈) + *symmetric A* (⋈).
  show *tolerance A* (λ*x y*. ¬ *x* ⋈ *y*) **by** (*unfold-locales*, *auto dest*: *sym irrefl*)
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* - (⊏) + *less-syntax* +
  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

lemmas *semiconnexI*[*intro*] = *semiconnex.intro*

Totality is negated antisymmetry [19, Proposition 2.2.4].

proposition *semiconnex-iff-neg-antisymmetric*:
  fixes *less* (**infix** ‹⊏› *50*)
  shows *semiconnex A* (⊏) ⟷ *antisymmetric A* (λ*x y*. ¬ *x* ⊏ *y*) (**is** *?l* ⟷ *?r*)
proof (*intro iffI semiconnexI antisymmetricI*)
  assume *?l*
  then interpret *semiconnex*.
  fix *x y*
  assume *x* ∈ *A* *y* ∈ *A* ¬ *x* ⊏ *y* **and** ¬ *y* ⊏ *x*
  then show *x* = *y* **by** (*cases rule*: *cases*, *auto*)
next
  assume *?r*
  then interpret *neg*: *antisymmetric A* (λ*x y*. ¬ *x* ⊏ *y*).
  fix *x y*
  show *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *x* ⊏ *y* ∨ *x* = *y* ∨ *y* ⊏ *x* **using** *neg.antisym* **by** *auto*
qed

lemma *semiconnex-cong*:
  assumes *r*: ⋀*a b*. *a* ∈ *A* ⟹ *b* ∈ *A* ⟹ *r a b* ⟷ *r′ a b*
  shows *semiconnex A r* ⟷ *semiconnex A r′*
  by (*simp add*: *semiconnex-iff-neg-antisymmetric r cong*: *antisymmetric-cong*)

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

lemma *neq-iff*: *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *x* ≠ *y* ⟷ *x* ⊏ *y* ∨ *y* ⊏ *x* **by** (*auto elim*:*neqE dest*: *irrefl*)

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

end

lemmas *semiconnex-irreflexiveI* = *semiconnex-irreflexive.intro*

lemma *semiconnex-irreflexive-cong*:
  assumes *r*: ⋀*a b*. *a* ∈ *A* ⟹ *b* ∈ *A* ⟹ *r a b* ⟷ *r′ a b*
  shows *semiconnex-irreflexive A r* ⟷ *semiconnex-irreflexive A r′*
  by (*simp add*: *semiconnex-irreflexive-def r cong*: *semiconnex-cong irreflexive-cong*)

locale *connex* = *related-set* +
  assumes *comparable*: *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *x* ⊑ *y* ∨ *y* ⊑ *x*
begin

interpretation *less-eq-asymmetrize*.

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*: $\neg x \sqsubset y \longleftrightarrow y \sqsubseteq x$
  using *comparable*[*OF x y*] **by** *auto*

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

interpretation *less-eq-asymmetrize*.

end

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

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

lemma *connex-empty*: *connex* {} *A* **by** *auto*

context
  fixes *less-eq* :: $'a \Rightarrow 'a \Rightarrow$ *bool* (**infix** ‹⊑› *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* ≡ *connex A r*
by (*auto intro*!: *ext simp*: *atomize-eq connex-def Complete-Partial-Order.chain-def*)

lemma *connex-union*:
assumes *connex X* (⊑) **and** *connex Y* (⊑) **and** ∀ *x* ∈ *X*. ∀ *y* ∈ *Y*. *x* ⊑ *y* ∨ *y* ⊑ *x*
shows *connex* (*X*∪*Y*) (⊑)
using *assms* **by** (*auto simp*: *connex-def*)

end

lemma *connex-cong*:
assumes *r*: ⋀*a b*. *a* ∈ *A* ⟹ *b* ∈ *A* ⟹ *r a b* ⟷ *r′ a b*
shows *connex A r* ⟷ *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* ∈ *A* **and** *y*: *y* ∈ *A* **shows** ¬ *y* ⊑ *x* ⟷ *x* ⊑ *y* ∧ *x* ≠ *y*
using *x y* **by** (*cases rule*: *comparable-cases*, *auto intro*:*antisym*)

lemma *total-pseudo-ordered-subset*: *B* ⊆ *A* ⟹ *total-pseudo-ordered-set B* (⊑)
apply (*intro-locales*)
using *antisymmetric-subset connex-subset* **by** *auto*

interpretation *less-eq-asymmetrize***.**

interpretation *asympartp*: *semiconnex-irreflexive A* (⊏)
proof (*intro semiconnex-irreflexive.intro asympartp-irreflexive semiconnexI*)
fix *x y* **assume** *xA*: *x* ∈ *A* **and** *yA*: *y* ∈ *A*
with *comparable antisym*
show *x* ⊏ *y* ∨ *x* = *y* ∨ *y* ⊏ *x* **by** (*auto simp*: *asympartp-def*)
qed

lemmas *asympartp-semiconnex* = *asympartp.semiconnex-axioms*
lemmas *asympartp-semiconnex-irreflexive* = *asympartp.semiconnex-irreflexive-axioms*

end

lemmas *total-pseudo-ordered-setI* = *total-pseudo-ordered-set.intro*

lemma *total-pseudo-ordered-set-cong*:
assumes *r*: ⋀*a b*. *a* ∈ *A* ⟹ *b* ∈ *A* ⟹ *r a b* ⟷ *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

lemmas *total-quasi-ordered-setI* = *total-quasi-ordered-set.intro*

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*)

lemma *weak-semiconnex*: *semiconnex* A $(\sqsubseteq)$
using *connex-axioms* **by** (*simp add*: *connex-iff-semiconnex-reflexive*)

interpretation *less-eq-asymmetrize*.

end

lemmas *total-ordered-setI* = *total-ordered-set.intro*[*OF total-quasi-ordered-setI*]

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*)

lemma *monotone-connex-image*:
fixes *ir* (**infix** ‹$\preceq$› *50*) **and** r (**infix** ‹$\sqsubseteq$› *50*)
assumes *mono*: *monotone-on* I $(\preceq)$ $(\sqsubseteq)$ f **and** *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

2.5 Order Pairs

We pair a relation (weak part) with a well-behaving "strict" part. Here no assumption is put on the "weak" part.

locale *compatible-ordering* =
related-set + *irreflexive* +
assumes *strict-implies-weak*: $x \sqsubset y \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow x \sqsubseteq y$
assumes *weak-strict-trans*[*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 *strict-weak-trans*[*trans*]: $x \sqsubset y \Longrightarrow y \sqsubseteq z \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow z \in A \Longrightarrow x \sqsubset z$
begin

The following sequence of declarations are in order to obtain fact names in a manner similar to the Isabelle/HOL facts of orders.

The strict part is necessarily transitive.

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

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

thm *strict.trans asym irrefl*

lemma *Restrp-compatible-ordering*: *compatible-ordering UNIV* $((\sqsubseteq)\restriction A)$ $((\sqsubset)\restriction A)$
apply (*unfold-locales*)
by (*auto dest*: *weak-strict-trans strict-weak-trans strict-implies-weak*)

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

lemma *weak-implies-not-strict*:
assumes *xy*: $x \sqsubseteq y$ **and** [*simp*]: $x \in A$ $y \in A$
shows $\neg y \sqsubset x$
proof
assume $y \sqsubset x$
also note *xy*
finally show *False* **using** *irrefl* **by** *auto*
qed

lemma *compatible-ordering-subset*: **assumes** $X \subseteq A$ **shows** *compatible-ordering* X $(\sqsubseteq)$ $(\sqsubset)$

apply *unfold-locales*
using *assms strict-implies-weak* **by** (*auto intro*: *strict-weak-trans weak-strict-trans*)

end

context *transitive* **begin**

interpretation *less-eq-asymmetrize*.

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*)

lemma *asympartp-compatible-ordering*: *compatible-ordering* A ($\sqsubseteq$) ($\sqsubset$)
apply *unfold-locales*
by (*auto dest*: *asym-trans*)

end

locale *reflexive-ordering* = *reflexive* + *compatible-ordering*

locale *reflexive-attractive-ordering* = *reflexive-ordering* + *attractive*

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

sublocale *reflexive-attractive-ordering*..

end

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

sublocale *reflexive-attractive-ordering*..

lemma *quasi-ordering-subset*: **assumes** $X \subseteq A$ **shows** *quasi-ordering* X ($\sqsubseteq$) ($\sqsubset$)
by (*intro quasi-ordering.intro quasi-ordered-subset compatible-ordering-subset assms*)

end

context *quasi-ordered-set* **begin**

interpretation *less-eq-asymmetrize*.

lemma *asympartp-quasi-ordering*: *quasi-ordering* A ($\sqsubseteq$) ($\sqsubset$)
by (*intro quasi-ordering.intro quasi-ordered-set-axioms asympartp-compatible-ordering*)

end

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

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

lemma *partial-ordering-subset*: **assumes** $X \subseteq A$ **shows** *partial-ordering* X $(\sqsubseteq)$ $(\sqsubset)$
by (*intro partial-ordering.intro partially-ordered-subset compatible-ordering-subset assms*)

end

context *partially-ordered-set* **begin**

interpretation *less-eq-asymmetrize*.

lemma *asympartp-partial-ordering*: *partial-ordering* A $(\sqsubseteq)$ $(\sqsubset)$
by (*intro partial-ordering.intro partially-ordered-set-axioms asympartp-compatible-ordering*)

end

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

sublocale *quasi-ordering*..

lemma *total-quasi-ordering-subset*: **assumes** $X \subseteq A$ **shows** *total-quasi-ordering* X $(\sqsubseteq)$ $(\sqsubset)$
by (*intro total-quasi-ordering.intro total-quasi-ordered-subset compatible-ordering-subset assms*)

end

context *total-quasi-ordered-set* **begin**

interpretation *less-eq-asymmetrize*.

lemma *asympartp-total-quasi-ordering*: *total-quasi-ordering* A $(\sqsubseteq)$ $(\sqsubset)$
by (*intro total-quasi-ordering.intro total-quasi-ordered-set-axioms asympartp-compatible-ordering*)

end

Fixing the definition of the strict part is very common, though it looks restrictive to the author.

locale *strict-quasi-ordering* = *quasi-ordered-set* + *less-syntax* +
assumes *strict-iff*: $x \in A \Longrightarrow y \in A \Longrightarrow x \sqsubset y \longleftrightarrow x \sqsubseteq y \wedge \neg y \sqsubseteq x$
begin

sublocale *compatible-ordering*
proof *unfold-locales*
  fix $x\ y\ z$
  show $x \in A \Longrightarrow \neg\ x \sqsubset x$ **by** (*auto simp*: *strict-iff*)
  { assume *xy*: $x \sqsubseteq y$ **and** *yz*: $y \sqsubset z$ **and** *x*: $x \in A$ **and** *y*: $y \in A$ **and** *z*: $z \in A$
    from *yz y z* **have** *ywz*: $y \sqsubseteq z$ **and** *zy*: $\neg z \sqsubseteq y$ **by** (*auto simp*: *strict-iff*)
    from *trans*[*OF xy ywz*]*x y z* **have** *xz*: $x \sqsubseteq z$ **by** *auto*
    from *trans*[*OF - xy*] *x y z zy* **have** *zx*: $\neg z \sqsubseteq x$ **by** *auto*
    from *xz zx x z* **show** $x \sqsubset z$ **by** (*auto simp*: *strict-iff*)
  }
  { assume *xy*: $x \sqsubset y$ **and** *yz*: $y \sqsubseteq z$ **and** *x*: $x \in A$ **and** *y*: $y \in A$ **and** *z*: $z \in A$
    from *xy x y* **have** *xwy*: $x \sqsubseteq y$ **and** *yx*: $\neg y \sqsubseteq x$ **by** (*auto simp*: *strict-iff*)
    from *trans*[*OF xwy yz*]*x y z* **have** *xz*: $x \sqsubseteq z$ **by** *auto*
    from *trans*[*OF yz*] *x y z yx* **have** *zx*: $\neg z \sqsubseteq x$ **by** *auto*
    from *xz zx x z* **show** $x \sqsubset z$ **by** (*auto simp*: *strict-iff*)
  }
  { show $x \sqsubset y \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow x \sqsubseteq y$ **by** (*auto simp*: *strict-iff*) **}**
qed

end

locale *strict-partial-ordering* = *strict-quasi-ordering* + *antisymmetric*
begin

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

lemma *strict-iff-neq*: $x \in A \Longrightarrow y \in A \Longrightarrow x \sqsubset y \longleftrightarrow x \sqsubseteq y \wedge x \neq y$
  by (*auto simp*: *strict-iff antisym*)

end

locale *total-ordering* = *reflexive* + *compatible-ordering* + *semiconnex A* $(\sqsubset)$
begin

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

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

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

sublocale *strict-partial-ordering*
proof
  fix *a b*
  assume *a*: $a \in A$ **and** *b*: $b \in A$
  then show $a \sqsubset b \longleftrightarrow a \sqsubseteq b \wedge \neg\, b \sqsubseteq a$ **by** (*auto simp*: *not-strict*[*symmetric*] *dest*: *asym*)
next
  fix *x y z* **assume** *xy*: $x \sqsubseteq y$ **and** *yz*: $y \sqsubseteq z$ **and** *xA*: $x \in A$ **and** *yA*: $y \in A$ **and** *zA*: $z \in A$
  with *weak-strict-trans*[*OF yz*] **show** $x \sqsubseteq z$ **by** (*auto simp*: *not-strict*[*symmetric*])
next
  fix *x y* **assume** *xy*: $x \sqsubseteq y$ **and** *yx*: $y \sqsubseteq x$ **and** *xA*: $x \in A$ **and** *yA*: $y \in A$
  with *semiconnex* **show** $x = y$ **by** (*auto dest*: *weak-implies-not-strict*)
qed

sublocale *total-ordered-set*..

context
  fixes *s*
  assumes *s*: $\forall x \in A.\ x \sqsubset s \longrightarrow (\exists z \in A.\ x \sqsubset z \wedge z \sqsubset s)$ **and** *sA*: $s \in A$
begin

lemma *dense-weakI*:
  assumes *bound*: $\bigwedge x.\ x \sqsubset s \Longrightarrow x \in A \Longrightarrow x \sqsubseteq y$ **and** *yA*: $y \in A$
  shows $s \sqsubseteq y$
proof (*rule ccontr*)
  assume ¬ *?thesis*
  with *yA sA* **have** $y \sqsubset s$ **by** (*simp add*: *not-weak*)
  from *s*[*rule-format*, *OF yA this*]
  obtain *x* **where** *xA*: $x \in A$ **and** *xs*: $x \sqsubset s$ **and** *yx*: $y \sqsubset x$ **by** *safe*
  have *xy*: $x \sqsubseteq y$ **using** *bound*[*OF xs xA*] **.**
  from *yx xy xA yA*
  show *False* **by** (*simp add*: *weak-implies-not-strict*)
qed

lemma *dense-bound-iff*:
  assumes *bA*: $b \in A$ **shows** *bound* $\{x{\in}A.\ x \sqsubset s\}$ $(\sqsubseteq)$ $b \longleftrightarrow s \sqsubseteq b$
  using *assms sA*
  by (*auto simp*: *bound-def intro*: *strict-implies-weak strict-weak-trans dense-weakI*)

lemma *dense-extreme-bound*:
  extreme-bound A $(\sqsubseteq)$ $\{x \in A.\ x \sqsubset s\}$ *s*
  by (*auto intro*!: *extreme-boundI intro*: *strict-implies-weak simp*: *dense-bound-iff sA*)

end

lemma *ordinal-cases*[*consumes 1*, *case-names suc lim*]:

assumes *aA*: $a \in A$
and *suc*: $\bigwedge p.$ *extreme* $\{x \in A.\ x \sqsubset a\}$ $(\sqsubseteq)$ $p \Longrightarrow$ *thesis*
and *lim*: *extreme-bound* A $(\sqsubseteq)$ $\{x \in A.\ x \sqsubset a\}$ $a \Longrightarrow$ *thesis*
shows *thesis*
proof (*cases* $\exists p.$ *extreme* $\{x \in A.\ x \sqsubset a\}$ $(\sqsubseteq)$ p)
case *True*
with *suc* **show** *?thesis* **by** *auto*
next
case *False*
show *?thesis*
proof (*rule lim*, *rule dense-extreme-bound*, *safe intro*!: *aA*)
fix x **assume** *xA*: $x \in A$ **and** *xa*: $x \sqsubset a$
show $\exists z \in A.\ x \sqsubset z \wedge z \sqsubset a$
proof (*rule ccontr*)
assume $\neg$*?thesis*
with *xA xa* **have** *extreme* $\{x \in A.\ x \sqsubset a\}$ $(\sqsubseteq)$ x **by** (*auto simp*: *not-strict*)
with *False* **show** *False* **by** *auto*
qed
qed
qed

end

context *total-ordered-set* **begin**

interpretation *less-eq-asymmetrize*.

lemma *asympartp-total-ordering*: *total-ordering* A $(\sqsubseteq)$ $(\sqsubset)$
by (*intro total-ordering.intro reflexive-axioms asympartp-compatible-ordering asympartp-semiconnex*)

end

2.6 Functions

definition *pointwise* $I\ r\ f\ g \equiv \forall i \in I.\ r\ (f\ i)\ (g\ i)$

lemmas *pointwiseI* = *pointwise-def*[*unfolded atomize-eq*, *THEN iffD2*, *rule-format*]

lemmas *pointwiseD*[*simp*] = *pointwise-def*[*unfolded atomize-eq*, *THEN iffD1*, *rule-format*]

lemma *pointwise-cong*:
assumes $r = r'$ $\bigwedge i.\ i \in I \Longrightarrow f\ i = f'\ i$ $\bigwedge i.\ i \in I \Longrightarrow g\ i = g'\ i$
shows *pointwise* $I\ r\ f\ g$ = *pointwise* $I\ r'\ f'\ g'$
using *assms* **by** (*auto simp*: *pointwise-def*)

lemma *pointwise-empty*[*simp*]: *pointwise* $\{\} = \top$ **by** (*auto intro*!: *ext pointwiseI*)

lemma *dual-pointwise*[*simp*]: (*pointwise* $I\ r)^{-}$ = *pointwise* $I\ r^{-}$

by (*auto intro*!: *ext pointwiseI dest*: *pointwiseD*)

lemma *pointwise-dual*: *pointwise I* r^{-} *f g* $\Longrightarrow$ *pointwise I r g f* **by** (*auto simp*: *pointwise-def*)

lemma *pointwise-un*: *pointwise* ($I \cup J$) *r* = *pointwise I r* $\sqcap$ *pointwise J r*
by (*auto intro*!: *ext pointwiseI*)

lemma *pointwise-unI*[*intro*!]: *pointwise I r f g* $\Longrightarrow$ *pointwise J r f g* $\Longrightarrow$ *pointwise* ($I \cup J$) *r f g*
by (*auto simp*: *pointwise-un*)

lemma *pointwise-bound*: *bound F* (*pointwise I r*) *f* $\longleftrightarrow$ ($\forall i \in I$. *bound* {*f i* |. *f* $\in$ *F*} *r* (*f i*))
by (*auto intro*!:*pointwiseI elim*!: *boundE*)

lemma *pointwise-extreme*:
shows *extreme F* (*pointwise X r*) *e* $\longleftrightarrow$ *e* $\in$ *F* $\wedge$ ($\forall x \in X$. *extreme* {*f x* |. *f* $\in$ *F*} *r* (*e x*))
by (*auto intro*!: *pointwiseI extremeI elim*!: *extremeE*)

lemma *pointwise-extreme-bound*:
fixes *r* (**infix** ‹$\sqsubseteq$› *50*)
assumes *F*: $F \subseteq \{f.\ f\ `\ X \subseteq A\}$
shows *extreme-bound* $\{f.\ f\ `\ X \subseteq A\}$ (*pointwise X* ($\sqsubseteq$)) *F s* $\longleftrightarrow$
($\forall x \in X$. *extreme-bound A* ($\sqsubseteq$) {*f x* |. *f* $\in$ *F*} (*s x*)) (**is** *?p* $\longleftrightarrow$ *?a*)
proof (*safe intro*!: *extreme-boundI pointwiseI*)
fix *x*
assume *s*: *?p* **and** *xX*: $x \in X$
{ fix *b*
assume *b*: *bound* {*f x* |. *f* $\in$ *F*} ($\sqsubseteq$) *b* **and** *bA*: $b \in A$
have *pointwise X* ($\sqsubseteq$) *s* (*s*(*x*:=*b*))
proof (*rule extreme-boundD*(*2*)[*OF s*], *safe intro*!: *pointwiseI*)
fix *f y*
assume *fF*: $f \in F$ **and** *yX*: $y \in X$
show *f y* $\sqsubseteq$ (*s*(*x*:=*b*)) *y*
proof (*cases x* = *y*)
case *True*
with *b fF* **show** *?thesis* **by** *auto*
next
case *False*
with *s*[*THEN extreme-bound-imp-bound*] *fF yX* **show** *?thesis* **by** (*auto dest*: *boundD*)
qed
next
fix *y* **assume** $y \in X$ **with** *bA s* **show** (*s*(*x* := *b*)) $y \in A$ **by** *auto*
qed
with *xX* **show** *s x* $\sqsubseteq$ *b* **by** (*auto dest*: *pointwiseD*)
next

    fix *f* **assume** $f \in F$
    from *extreme-boundD*(*1*)[*OF s this*] *F xX*
    show $f\ x \sqsubseteq s\ x$ **by** *auto*
  next
    show $s\ x \in A$ **using** *s xX* **by** *auto*
  }
next
  fix *x*
  assume *s*: *?a* **and** *xX*: $x \in X$
  { from *s xX* **show** $s\ x \in A$ **by** *auto*
  next
    fix *b* **assume** *b*: *bound F* (*pointwise X* ($\sqsubseteq$)) *b* **and** *bA*: b ‘ $X \subseteq A$
    with *xX* **have** *bound* $\{f\ x \mid . f \in F\}$ ($\sqsubseteq$) (*b x*) **by** (*auto simp*: *pointwise-bound*)
    with *s*[*rule-format*, *OF xX*] *bA xX* **show** $s\ x \sqsubseteq b\ x$ **by** *auto*
  next
    fix *f* **assume** $f \in F$
    with *s*[*rule-format*, *OF xX*] **show** $f\ x \sqsubseteq s\ x$ **by** *auto*
  }
qed

lemma *dual-pointwise-extreme-bound*:
  extreme-bound FA (*pointwise X r*)$^-$ *F* = *extreme-bound FA* (*pointwise X r*$^-$) *F*
  by (*simp*)

lemma *pointwise-monotone-on*:
  fixes *less-eq* (**infix** ‹$\sqsubseteq$› *50*) **and** *prec-eq* (**infix** ‹$\preceq$› *50*)
  shows *monotone-on I* ($\preceq$) (*pointwise A* ($\sqsubseteq$)) *f* $\longleftrightarrow$
   ($\forall a \in A$. *monotone-on I* ($\preceq$) ($\sqsubseteq$) (λi. *f i a*)) (**is** *?l* $\longleftrightarrow$ *?r*)
proof (*safe intro*!: *monotone-onI pointwiseI*)
  fix *a i j* **assume** *aA*: $a \in A$ **and** $*$: *?l* $i \preceq j$ $i \in I$ $j \in I$
  then
  show $f\ i\ a \sqsubseteq f\ j\ a$ **by** (*auto dest*: *monotone-onD*)
next
  fix *a i j* **assume** *?r* **and** $a \in A$ **and** *ij*: $i \preceq j$ $i \in I$ $j \in I$
  then have *monotone-on I* ($\preceq$) ($\sqsubseteq$) (λi. *f i a*) **by** *auto*
  from *monotone-onD*[*OF this*]*ij*
  show $f\ i\ a \sqsubseteq f\ j\ a$ **by** *auto*
qed

lemmas *pointwise-monotone* = *pointwise-monotone-on*[*of UNIV*]

lemma (**in** *reflexive*) *pointwise-reflexive*: *reflexive* $\{f.\ f$ ‘ $I \subseteq A\}$ (*pointwise I* ($\sqsubseteq$))
  apply *unfold-locales* **by** (*auto intro*!: *pointwiseI simp*: *subsetD*[*OF - imageI*])

lemma (**in** *irreflexive*) *pointwise-irreflexive*:
  assumes *I0*: $I \neq \{\}$ **shows** *irreflexive* $\{f.\ f$ ‘ $I \subseteq A\}$ (*pointwise I* ($\sqsubset$))
proof (*safe intro*!: *irreflexive.intro*)
  fix *f*
  assume *f*: *f* ‘ $I \subseteq A$ **and** *ff*: *pointwise I* ($\sqsubset$) *f f*

from *I0* **obtain** *i* **where** *i*: $i \in I$ **by** *auto*
with *ff* **have** $f\ i \sqsubset f\ i$ **by** *auto*
with *f i* **show** *False* **by** *auto*
qed

lemma (**in** *semiattractive*) *pointwise-semiattractive*: *semiattractive* $\{f.\ f\ `\ I \subseteq A\}$ (*pointwise* I $(\sqsubseteq)$)
proof (*unfold-locales*, *safe intro*!: *pointwiseI*)
fix *f g h i*
assume *fg*: *pointwise* I $(\sqsubseteq)$ *f g* **and** *gf*: *pointwise* I $(\sqsubseteq)$ *g f* **and** *gh*: *pointwise* I $(\sqsubseteq)$ *g h*
and [*simp*]: $i \in I$ **and** *f*: $f\ `\ I \subseteq A$ **and** *g*: $g\ `\ I \subseteq A$ **and** *h*: $h\ `\ I \subseteq A$
show $f\ i \sqsubseteq h\ i$
proof (*rule attract*)
from *fg* **show** $f\ i \sqsubseteq g\ i$ **by** *auto*
from *gf* **show** $g\ i \sqsubseteq f\ i$ **by** *auto*
from *gh* **show** $g\ i \sqsubseteq h\ i$ **by** *auto*
qed (*insert f g h*, *auto simp*: *subsetD*[*OF* - *imageI*])
qed

lemma (**in** *attractive*) *pointwise-attractive*: *attractive* $\{f.\ f\ `\ I \subseteq A\}$ (*pointwise* I $(\sqsubseteq)$)
apply (*intro attractive.intro attractive-axioms.intro*)
using *pointwise-semiattractive dual.pointwise-semiattractive* **by** *auto*

Antisymmetry will not be preserved by pointwise extension over restricted domain.

lemma (**in** *antisymmetric*) *pointwise-antisymmetric*:
antisymmetric $\{f.\ f\ `\ I \subseteq A\}$ (*pointwise* I $(\sqsubseteq)$)
oops

lemma (**in** *transitive*) *pointwise-transitive*: *transitive* $\{f.\ f\ `\ I \subseteq A\}$ (*pointwise* I $(\sqsubseteq)$)
proof (*unfold-locales*, *safe intro*!: *pointwiseI*)
fix *f g h i*
assume *fg*: *pointwise* I $(\sqsubseteq)$ *f g* **and** *gh*: *pointwise* I $(\sqsubseteq)$ *g h*
and [*simp*]: $i \in I$ **and** *f*: $f\ `\ I \subseteq A$ **and** *g*: $g\ `\ I \subseteq A$ **and** *h*: $h\ `\ I \subseteq A$
from *fg* **have** $f\ i \sqsubseteq g\ i$ **by** *auto*
also from *gh* **have** $g\ i \sqsubseteq h\ i$ **by** *auto*
finally show $f\ i \sqsubseteq h\ i$ **using** *f g h* **by** (*auto simp*: *subsetD*[*OF* - *imageI*])
qed

lemma (**in** *quasi-ordered-set*) *pointwise-quasi-order*:
quasi-ordered-set $\{f.\ f\ `\ I \subseteq A\}$ (*pointwise* I $(\sqsubseteq)$)
by (*intro quasi-ordered-setI pointwise-transitive pointwise-reflexive*)

lemma (**in** *compatible-ordering*) *pointwise-compatible-ordering*:
assumes *I0*: $I \neq \{\}$
shows *compatible-ordering* $\{f.\ f\ `\ I \subseteq A\}$ (*pointwise* I $(\sqsubseteq)$) (*pointwise* I $(\sqsubset)$)

proof (*intro compatible-ordering.intro compatible-ordering-axioms.intro pointwise-irreflexive*[*OF I0*], *safe intro*!: *pointwiseI*)
 fix *f g h i*
 assume *fg*: *pointwise I* ($\sqsubseteq$) *f g* **and** *gh*: *pointwise I* ($\sqsubset$) *g h*
  and [*simp*]: $i \in I$ **and** *f*: *f* ‘ $I \subseteq A$ **and** *g*: *g* ‘ $I \subseteq A$ **and** *h*: *h* ‘ $I \subseteq A$
 from *fg* **have** $f\ i \sqsubseteq g\ i$ **by** *auto*
 also from *gh* **have** $g\ i \sqsubset h\ i$ **by** *auto*
 finally show $f\ i \sqsubset h\ i$ **using** *f g h* **by** (*auto simp*: *subsetD*[*OF* - *imageI*])
next
 fix *f g h i*
 assume *fg*: *pointwise I* ($\sqsubset$) *f g* **and** *gh*: *pointwise I* ($\sqsubseteq$) *g h*
  and [*simp*]: $i \in I$ **and** *f*: *f* ‘ $I \subseteq A$ **and** *g*: *g* ‘ $I \subseteq A$ **and** *h*: *h* ‘ $I \subseteq A$
 from *fg* **have** $f\ i \sqsubset g\ i$ **by** *auto*
 also from *gh* **have** $g\ i \sqsubseteq h\ i$ **by** *auto*
 finally show $f\ i \sqsubset h\ i$ **using** *f g h* **by** (*auto simp*: *subsetD*[*OF* - *imageI*])
next
 fix *f g i*
 assume *fg*: *pointwise I* ($\sqsubset$) *f g*
  and [*simp*]: $i \in I$
  and *f*: *f* ‘ $I \subseteq A$ **and** *g*: *g* ‘ $I \subseteq A$
 from *fg* **have** $f\ i \sqsubset g\ i$ **by** *auto*
 with *f g* **show** $f\ i \sqsubseteq g\ i$ **by** (*auto simp*: *subsetD*[*OF* - *imageI*] *strict-implies-weak*)
qed

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* ($\lambda x\ y.\ y \le x$)

abbreviation *greatest* **where** *greatest X* ≡ *extreme X* ($\le$)

abbreviation *supremum* **where** *supremum X* ≡ *least* (*Collect* (*bound X* ($\le$)))

abbreviation *infimum* **where** *infimum X* ≡ *greatest* (*Collect* (*bound X* ($\lambda x\ y.\ y \le x$)))

lemma *supremumI*: *bound X* ($\le$) $s \Longrightarrow (\bigwedge b.$ *bound X* ($\le$) $b \Longrightarrow s \le b) \Longrightarrow$ *supremum X s*
 and *infimumI*: *bound X* ($\ge$) $i \Longrightarrow (\bigwedge b.$ *bound X* ($\ge$) $b \Longrightarrow b \le i) \Longrightarrow$ *infimum X i*
 by (*auto intro*!: *extremeI*)

lemma *supremumE*: *supremum X s* $\Longrightarrow$
  (*bound X* ($\le$) $s \Longrightarrow (\bigwedge b.$ *bound X* ($\le$) $b \Longrightarrow s \le b) \Longrightarrow$ *thesis*) $\Longrightarrow$ *thesis*
 and *infimumE*: *infimum X i* $\Longrightarrow$
  (*bound X* ($\ge$) $i \Longrightarrow (\bigwedge b.$ *bound X* ($\ge$) $b \Longrightarrow b \le i) \Longrightarrow$ *thesis*) $\Longrightarrow$ *thesis*

by (*auto*)

lemma *extreme-bound-supremum*[*simp*]: *extreme-bound UNIV* (≤) = *supremum* **by** (*auto intro*!: *ext*)
lemma *extreme-bound-infimum*[*simp*]: *extreme-bound UNIV* (≥) = *infimum* **by** (*auto intro*!: *ext*)

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 *The-least-eq-Least*: *The* (*least X*) = *Least* (λ*x*. *x* ∈ *X*)
by (*simp add*: *Least-eq-The-least*)

lemma *least-imp-infimum*: **assumes** *least X x* **shows** *infimum X x*
using *extreme-imp-extreme-bound*[*OF assms*, *of UNIV*] **by** *simp*

lemma *least-LeastI-ex1*:
assumes *ex1*: ∃!*x*. *least* {*x*. *P x*} *x*
shows *least* {*x*. *P x*} (*LEAST x*. *P x*)
using *theI'*[*OF ex1*] **by** (*simp add*: *Least-eq-The-least*)

end
context *order* **begin**

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

lemma *The-greatest-eq-Greatest*: *The* (*greatest X*) = *Greatest* (λ*x*. *x* ∈ *X*)
by (*simp add*: *Greatest-eq-The-greatest*)

lemma *greatest-imp-supremum*: **assumes** *greatest X x* **shows** *supremum X x*
using *extreme-imp-extreme-bound*[*OF assms*, *of UNIV*] **by** *simp*

lemma *greatest-GreatestI-ex1*:
assumes *ex1*: ∃!*x*. *greatest* {*x*. *P x*} *x*
shows *greatest* {*x*. *P x*} (*GREATEST x*. *P x*)
using *theI'*[*OF ex1*] **by** (*simp add*: *Greatest-eq-The-greatest*)

end

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

lemma *pointwise-UNIV-le*[*simp*]: *pointwise UNIV* (≤) = (≤) **by** (*intro ext*, *simp add*: *pointwise-def le-fun-def*)
lemma *pointwise-UNIV-ge*[*simp*]: *pointwise UNIV* (≥) = (≥) **by** (*intro ext*, *simp add*: *pointwise-def le-fun-def*)

lemma *fun-supremum-iff*: *supremum F e* ⟷ (∀ *x*. *supremum* {*f x* |. *f* ∈ *F*} (*e*

x))
  using *pointwise-extreme-bound*[*of F UNIV UNIV* (≤)] **by** *simp*

lemma *fun-infimum-iff*: *infimum F e* ⟷ (∀ *x*. *infimum* {*f x* |. *f* ∈ *F*} (*e x*))
  using *pointwise-extreme-bound*[*of F UNIV UNIV* (≥)] **by** *simp*

class *reflorder* = *ord* + **assumes** *reflexive-ordering UNIV* (≤) (<)
begin

sublocale *order*: *reflexive-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 *reflorder-axioms* **unfolding** *class.reflorder-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 *attrorder* = *ord* +
  assumes *reflexive-attractive-ordering UNIV* (≤) (<)
begin

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

subclass *reflorder*
proof−
  interpret *reflexive-attractive-ordering UNIV*
    using *attrorder-axioms* **unfolding** *class.attrorder-def* **by** *auto*
  show *class.reflorder* (≤) (<)**..**
qed

sublocale *order*: *reflexive-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 ⋀*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 *attrorder-axioms* **unfolding** *class.attrorder-def*
by (*auto simp*:*atomize-eq*)

end

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

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

subclass *attrorder*
proof−
interpret *pseudo-ordering UNIV*
using *psorder-axioms* **unfolding** *class.psorder-def* **by** *auto*
show *class.attrorder* (≤) (<)**..**
qed

sublocale *order*: *pseudo-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 *psorder-axioms* **unfolding** *class.psorder-def* **by** (*auto simp*:*atomize-eq*)

end

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

subclass *attrorder*
proof−
interpret *quasi-ordering UNIV*
using *qorder-axioms* **unfolding** *class.qorder-def* **by** *auto*
show *class.attrorder* (≤) (<)**..**
qed

sublocale *order*: *quasi-ordering UNIV*

rewrites $\bigwedge x.\ x \in UNIV \equiv True$
and $\bigwedge X.\ X \subseteq UNIV \equiv True$
and $\bigwedge r.\ r \restriction UNIV \equiv r$
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 $\bigwedge P1\ P2.\ (True \Longrightarrow PROP\ P1 \Longrightarrow PROP\ P2) \equiv (PROP\ P1 \Longrightarrow PROP\ P2)$
using *qorder-axioms* **unfolding** *class.qorder-def* **by** (*auto simp*:*atomize-eq*)

lemmas [*intro*!] = *order.quasi-ordered-subset*

end

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

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

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 $\bigwedge r.\ r \restriction UNIV \equiv r$
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

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

interpretation *total-quasi-ordering UNIV*
using *linqorder-axioms* **unfolding** *class.linqorder-def* **by** *auto*

subclass *qorder*..

sublocale *order*: *total-quasi-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* ≡ *All*
    and *Bex UNIV* ≡ *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 $\bigwedge P1\ P2.\ (True \Longrightarrow PROP\ P1 \Longrightarrow PROP\ P2) \equiv (PROP\ P1 \Longrightarrow PROP\ P2)$
    using *linqorder-axioms* **unfolding** *class.linqorder-def*
    by (*auto simp*:*atomize-eq*)

lemmas *asympartp-le* = *order.not-iff-asym*[*symmetric*, *abs-def*]

end

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

context *preorder* **begin**

The relation (<) is defined as the antisymmetric part of (≤).

lemma [*simp*]:
  shows *asympartp-le*: *asympartp* (≤) = (<)
    and *asympartp-ge*: *asympartp* (≥) = (>)
  by (*intro ext*, *auto simp*: *asympartp-def less-le-not-le*)

interpretation *strict-quasi-ordering UNIV* (≤) (<)
  apply *unfold-locales*
  using *order-refl* **apply** *assumption*
  using *order-trans* **apply** *assumption*
  using *less-le-not-le* **apply** *assumption*
  done

subclass *qorder*..

sublocale *order*: *strict-quasi-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* ≡ *All*
    and *Bex UNIV* ≡ *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

context *order* **begin**

interpretation *strict-partial-ordering UNIV* $(\leq)$ $(<)$
apply *unfold-locales*
using *order-antisym* **by** *assumption*

subclass *porder*..

sublocale *order*: *strict-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

context *order* **begin**

lemma *ex-greatest-iff-Greatest*:
$Ex\ (greatest\ X) \longleftrightarrow greatest\ X\ (Greatest\ (\lambda x.\ x \in X))$
using *order.ex-extreme-iff-the*[*of X*]
by (*simp add*: *The-greatest-eq-Greatest*)

lemma *greatest-imp-supremum-Greatest*:
$greatest\ X\ x \Longrightarrow supremum\ X\ (Greatest\ (\lambda x.\ x \in X))$
using *ex-greatest-iff-Greatest*[*THEN iffD1*, *THEN greatest-imp-supremum*]
by *auto*

end

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

context *linorder* **begin**

subclass *linqorder* **apply** *unfold-locales* **by** *auto*

sublocale *order*: *total-ordering UNIV*
  rewrites $\bigwedge x.\ x \in UNIV \equiv True$
    and $\bigwedge X.\ X \subseteq UNIV \equiv True$
    and $\bigwedge r.\ r \upharpoonright UNIV \equiv r$
    and $\bigwedge P.\ True \wedge P \equiv P$
    and *Ball UNIV* ≡ *All*
    and *Bex UNIV* ≡ *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::reflorder*]
thm *order.extreme-bound-quasi-const*[**where** *'a='a::attrorder*]
thm *order.extreme-bound-singleton-eq*[**where** *'a='a::psorder*]
thm *order.trans*[**where** *'a='a::qorder*]
thm *order.comparable-cases*[**where** *'a='a::linqorder*]
thm *order.cases*[**where** *'a='a::linorder*]

2.8 Declaring Duals

sublocale *reflexive* ⊆ *sym*: *reflexive A sympartp* (⊑)
  rewrites *sympartp* $(\sqsubseteq)^{-} \equiv$ *sympartp* (⊑)
    and $\bigwedge r.$ *sympartp* (*sympartp r*) ≡ *sympartp r*
    and $\bigwedge r.$ *sympartp r* ↾ *A* ≡ *sympartp* (*r* ↾ *A*)
  by (*auto 0 4 simp*:*atomize-eq*)

sublocale *quasi-ordered-set* ⊆ *sym*: *quasi-ordered-set A sympartp* (⊑)
  rewrites *sympartp* $(\sqsubseteq)^{-}$ = *sympartp* (⊑)
    and *sympartp* (*sympartp* (⊑)) = *sympartp* (⊑)
   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* ⊆ *dual*: *reflexive A* $(\sqsubseteq)^{-}$
  rewrites *sympartp* $(\sqsubseteq)^{-} \equiv$ *sympartp* (⊑)
    and $\bigwedge r.$ *sympartp* (*r* ↾ *A*) ≡ *sympartp r* ↾ *A*
    and $(\sqsubseteq)^{-} \upharpoonright A \equiv ((\sqsubseteq) \upharpoonright A)^{-}$
  by (*auto simp*: *atomize-eq*)

context *attractive* **begin**

interpretation *less-eq-symmetrize*.

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*)

context *antisymmetric* **begin**

lemma *extreme-bound-unique*:
  extreme-bound A $(\sqsubseteq)$ X $x \Longrightarrow$ *extreme-bound* A $(\sqsubseteq)$ X $y \longleftrightarrow x = y$
  apply (*unfold extreme-bound-def*)
  apply (*rule dual.extreme-unique*) **by** *auto*

lemma *ex-extreme-bound-iff-ex1*:
  Ex (*extreme-bound* A $(\sqsubseteq)$ X) $\longleftrightarrow$ *Ex1* (*extreme-bound* A $(\sqsubseteq)$ X)
  apply (*unfold extreme-bound-def*)
  apply (*rule dual.ex-extreme-iff-ex1*) **by** *auto*

lemma *ex-extreme-bound-iff-the*:
  Ex (*extreme-bound* A $(\sqsubseteq)$ X) $\longleftrightarrow$ *extreme-bound* A $(\sqsubseteq)$ X (*The* (*extreme-bound* A $(\sqsubseteq)$ X))
  apply (*rule iffI*)

apply (*rule theI′*)
using *extreme-bound-unique* **by** *auto*

end

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

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

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

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

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

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

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

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

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

lemmas(**in** *qorder*) [*intro*!] = *order.dual.quasi-ordered-subset*

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

sublocale *reflexive-attractive-ordering* ⊆ *dual*: *reflexive-attractive-ordering A* $(\sqsubseteq)^-$

$(\sqsubset)^{-}$
rewrites *sympartp* $(\sqsubseteq)^{-}$ = *sympartp* $(\sqsubseteq)$
by *unfold-locales auto*

sublocale *pseudo-ordering* ⊆ *dual*: *pseudo-ordering A* $(\sqsubseteq)^{-}$ $(\sqsubset)^{-}$
rewrites *sympartp* $(\sqsubseteq)^{-}$ = *sympartp* $(\sqsubseteq)$
by *unfold-locales auto*

lemma (**in** *psorder*) *least-Least*:
fixes *X* :: *'a set*
shows *Ex* (*least X*) ⟷ *least X* (*LEAST x. x* ∈ *X*)
using *order.dual.ex-extreme-iff-the*[*of X*, *unfolded The-least-eq-Least*].

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

sublocale *partial-ordering* ⊆ *dual*: *partial-ordering A* $(\sqsubseteq)^{-}$ $(\sqsubset)^{-}$
rewrites *sympartp* $(\sqsubseteq)^{-}$ = *sympartp* $(\sqsubseteq)$
by *unfold-locales auto*

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

sublocale *total-ordering* ⊆ *dual*: *total-ordering A* $(\sqsubseteq)^{-}$ $(\sqsubset)^{-}$
rewrites *sympartp* $(\sqsubseteq)^{-}$ = *sympartp* $(\sqsubseteq)$
by *unfold-locales auto*

sublocale *strict-quasi-ordering* ⊆ *dual*: *strict-quasi-ordering A* $(\sqsubseteq)^{-}$ $(\sqsubset)^{-}$
rewrites *sympartp* $(\sqsubseteq)^{-}$ = *sympartp* $(\sqsubseteq)$
by *unfold-locales* (*auto simp*: *strict-iff*)

sublocale *strict-partial-ordering* ⊆ *dual*: *strict-partial-ordering A* $(\sqsubseteq)^{-}$ $(\sqsubset)^{-}$
rewrites *sympartp* $(\sqsubseteq)^{-}$ = *sympartp* $(\sqsubseteq)$
by *unfold-locales auto*

sublocale *total-ordering* ⊆ *dual*: *total-ordering A* $(\sqsubseteq)^{-}$ $(\sqsubset)^{-}$
rewrites *sympartp* $(\sqsubseteq)^{-}$ = *sympartp* $(\sqsubseteq)$
by *unfold-locales auto*

lemma(**in** *antisymmetric*) *monotone-extreme-imp-extreme-bound-iff*:
fixes *ir* (**infix** ‹⪯› *50*)
assumes *f* ' *C* ⊆ *A* **and** *monotone-on C* (⪯) $(\sqsubseteq)$ *f* **and** *i*: *extreme C* (⪯) *i*
shows *extreme-bound A* $(\sqsubseteq)$ (*f* ' *C*) *x* ⟷ *f i* = *x*
using *dual.extreme-unique monotone-extreme-extreme-boundI*[*OF assms*]
by (*auto simp*: *extreme-bound-def*)

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*) *reflorder*
proof (*intro-classes*, *unfold-locales*)
  note [*simp*] = *le-fun-def less-fun-def*
  fix $f\ g\ h :: 'a \Rightarrow 'b$
  { assume *fg*: $f \le g$ **and** *gh*: $g < h$
    show $f < h$
    proof (*unfold less-fun-def*, *intro conjI le-funI notI*)
      from *fg* **have** $f\ x \le g\ x$ **for** x **by** *auto*
      also from *gh* **have** $g\ x \le h\ x$ **for** x **by** *auto*
      finally (*order.trans*) **show** $f\ x \le h\ x$ **for** x**.**
      assume *hf*: $h \le f$
      then have $h\ x \le f\ x$ **for** x **by** *auto*
      also from *fg* **have** $f\ x \le g\ x$ **for** x **by** *auto*
      finally have $h \le g$ **by** *auto*
      with *gh* **show** *False* **by** *auto*
    qed
  }
  { assume *fg*: $f < g$ **and** *gh*: $g \le h$
    show $f < h$
    proof (*unfold less-fun-def*, *intro conjI le-funI notI*)
      from *fg* **have** $f\ x \le g\ x$ **for** x **by** *auto*
      also from *gh* **have** $g\ x \le h\ x$ **for** x **by** *auto*
      finally show $f\ x \le h\ x$ **for** x**.**
      from *gh* **have** $g\ x \le h\ x$ **for** x **by** *auto*
      also assume *hf*: $h \le f$
      then have $h\ x \le f\ x$ **for** x **by** *auto*
      finally have $g \le f$ **by** *auto*
      with *fg* **show** *False* **by** *auto*
    qed
  }
  show $f < g \Longrightarrow f \le g$ **by** *auto*
  show $\neg f < f$ **by** *auto*
  show $f \le f$ **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
theory *Well-Relations*
imports *Binary-Relations*
begin

3 Well-Relations

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 *extremedI*: *extreme* $A\ r\ e \Longrightarrow$ *extremed* $A\ r$
by (*auto simp*: *extremed-def*)

lemma *extremedE*: *extremed* $A\ r \Longrightarrow (\bigwedge e.$ *extreme* $A\ r\ e \Longrightarrow$ *thesis*$) \Longrightarrow$ *thesis*
by (*auto simp*: *extremed-def*)

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*)

locale *well-founded* = *related-set* - $(\sqsubset)$ + *less-syntax* +
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 $(\bigwedge x.\ (\bigwedge y.\ ((\sqsubset) \restriction A)\ y\ x \Longrightarrow P\ y) \Longrightarrow P\ x) \Longrightarrow P\ a$ **for** *a P*
using *induct*[*of a P*] **by** (*auto simp*: *Restrp-def*)
then show *well-founded UNIV* $((\sqsubset)\restriction A)$ **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* $((\sqsubset)\restriction A)^{++}$
rewrites $\bigwedge x.\ x \in UNIV \equiv True$
and $((\sqsubset)\restriction A)^{++} \restriction UNIV = ((\sqsubset)\restriction A)^{++}$
and $(((\sqsubset)\restriction A)^{++})^{++} = ((\sqsubset)\restriction A)^{++}$
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−
{ fix *P x*
assume *induct-step*: $\bigwedge x.\ (\bigwedge y.\ ((\sqsubset)\restriction A)^{++}\ y\ x \Longrightarrow P\ y) \Longrightarrow P\ x$
have *P x*
proof (*rule induct-step*)
show $\bigwedge y.\ ((\sqsubset)\restriction A)^{++}\ y\ x \Longrightarrow P\ y$
proof (*induct x rule*: *Restrp-induct*)
case (*less x*)
from ‹$((\sqsubset)\restriction A)^{++}\ y\ x$›
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: ⋀X. X ⊆ A ⟹ X ⊆ ((⊏) `` X) ∩ A ⟹ X = {}
  shows well-founded A (⊏)
proof
  fix P a assume aA: a ∈ A and Ind: ⋀x. x ∈ A ⟹ (⋀y. y ∈ A ⟹ y ⊏ x ⟹
P y) ⟹ P x
   from Ind have {a∈A. ¬P a} ⊆ ((⊏) `` {a∈A. ¬P a}) ∩ A by (auto simp:
Imagep-def)
  from pre[OF - this] aA
  show P a by auto
qed

lemma well-foundedI-extremal:
  assumes a: ⋀X. X ⊆ A ⟹ X ≠ {} ⟹ ∃x ∈ X. ∀y ∈ X. ¬ y ⊏ x
  shows well-founded A (⊏)
proof (rule well-foundedI-pf)
  fix X assume XA: X ⊆ A and pf: X ⊆ ((⊏) `` X) ∩ A
  from a[OF XA] pf show X = {} by (auto simp: Imagep-def)
qed

lemma well-founded-iff-ex-extremal:
  well-founded A (⊏) ⟷ (∀X ⊆ A. X ≠ {} ⟶ (∃x ∈ X. ∀z ∈ X. ¬ z ⊏ x))
  using well-founded.nonempty-imp-ex-extremal well-foundedI-extremal by blast

end

lemma well-founded-cong:
  assumes r: ⋀a b. a ∈ A ⟹ b ∈ A ⟹ r a b ⟷ r' a b
    and A: ⋀a b. r' a b ⟹ a ∈ A ⟷ a ∈ A'
    and B: ⋀a b. r' a b ⟹ b ∈ A ⟷ b ∈ 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*: $y \in X$ **and** *yx*: $r'\ y\ x$
from *yX* X **have** *yA'*: $y \in A'$ **by** *auto*
show *False*
proof (*cases* $y \in A$)
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** $\exists\, x \in X.\ \forall\, y \in X.\ \neg\ r'\ y\ x$ **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: $X \subseteq A$ **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 B **have** $\forall y{\in}X.\ \neg\ r\ y\ x$ **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: $x \in X \cap A'$ **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: $y \in X$ **and** yx: $r\ y\ x$
from $yX\ X$ **have** y: $y \in A$ **by** *auto*
show *False*
proof (*cases* $y \in A'$)
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** $\exists\, x \in X.\ \forall\, y \in X.\ \neg\ r\ y\ x$ **by** *auto*
qed
qed
qed

lemma *wfP-iff-well-founded-UNIV*: *wfP* $r \longleftrightarrow$ *well-founded UNIV* r
by (*auto simp*: *wfp-def wf-def well-founded-def*)

lemma *well-founded-empty*[*intro*!]: *well-founded* $\{\}$ r
by (*auto simp*: *well-founded-iff-ex-extremal*)

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 *Restrp-tranclp-well-founded-iff*:
fixes *less* (**infix** ‹⊏› *50*)
shows *well-founded UNIV* ((⊏) ↾ *A*)$^{++}$ ⟷ *well-founded A* (⊏) (**is** *?l* ⟷ *?r*)
proof (*rule iffI*)
show *?r* ⟹ *?l* **by** (*fact well-founded.Restrp-tranclp-well-founded*)
assume *?l*
then interpret *well-founded UNIV* ((⊏) ↾ *A*)$^{++}$**.**
show *?r*
proof (*unfold well-founded-iff-ex-extremal*, *intro allI impI*)
fix *X* **assume** *XA*: *X* ⊆ *A* **and** *X0*: *X* ≠ {}
from *nonempty-imp-ex-extremal*[*OF - X0*]
obtain *z* **where** *zX*: *z* ∈ *X* **and** *Xz*: ∀ *y*∈*X*. ¬ ((⊏) ↾ *A*)$^{++}$ *y z* **by** *auto*
show ∃ *z* ∈ *X*. ∀ *y* ∈ *X*. ¬ *y* ⊏ *z*
proof (*intro bexI*[*OF - zX*] *ballI notI*)
fix *y* **assume** *yX*: *y* ∈ *X* **and** *yz*: *y* ⊏ *z*
from *yX yz zX XA* **have** ((⊏) ↾ *A*) *y z* **by** *auto*
with *yX Xz* **show** *False* **by** *auto*
qed
qed
qed

lemma (**in** *well-founded*) *well-founded-subset*:
assumes *B* ⊆ *A* **shows** *well-founded B* (⊏)
using *assms well-founded-axioms* **by** (*auto simp*: *well-founded-iff-ex-extremal*)

lemma *well-founded-extend*:
fixes *less* (**infix** ‹⊏› *50*)
assumes *A*: *well-founded A* (⊏)
assumes *B*: *well-founded B* (⊏)
assumes *AB*: ∀ *a* ∈ *A*. ∀ *b* ∈ *B*. ¬*b* ⊏ *a*
shows *well-founded* (*A* ∪ *B*) (⊏)
proof (*intro well-foundedI-extremal*)
interpret *A*: *well-founded A* (⊏) **using** *A***.**
interpret *B*: *well-founded B* (⊏) **using** *B***.**
fix *X* **assume** *XAB*: *X* ⊆ *A* ∪ *B* **and** *X0*: *X* ≠ {}
show ∃ *x*∈*X*. ∀ *y*∈*X*. ¬ *y* ⊏ *x*
proof (*cases X* ∩ *A* = {})
case *True*
with *XAB* **have** *XB*: *X* ⊆ *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 ⊆ A and X ≠ {}
  from well-founded.nonempty-imp-ex-extremal[OF wf this]
  show ∃x∈X. ∀y∈X. ¬ 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* ($\sqsubset$)$\upharpoonright$*A* + *Restrp*: *well-founded UNIV* ($\sqsubset$)$\upharpoonright$*A*
  using *Restrp-strict-order Restrp-well-founded* .

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

lemmas *well-founded-ordered-setI* = *well-founded-ordered-set.intro*

lemma *well-founded-ordered-set-empty*[*intro*!]: *well-founded-ordered-set* {} *r*
  by (*auto intro*!: *well-founded-ordered-setI*)

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* = *connex-axioms*

interpretation *less-eq-asymmetrize*.

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*)

lemma *monotone-image-well-related*:
fixes *leB* (**infix** ‹⊴› *50*)
assumes *mono*: *monotone-on A* (⊑) (⊴) *f* **shows** *well-related-set* (*f* ‘ *A*) (⊴)
proof (*intro well-related-set.intro*)
interpret *less-eq-dualize*.
fix *X'* **assume** *X'fA*: *X'* ⊆ *f* ‘ *A* **and** *X'0*: *X'* ≠ {}
then obtain *X* **where** *XA*: *X* ⊆ *A* **and** *X'*: *X'* = *f* ‘ *X* **and** *X0*: *X* ≠ {}
by (*auto elim*!: *subset-imageE*)
from *nonempty-imp-ex-extreme*[*OF XA X0*]
obtain *e* **where** *Xe*: *extreme X* (⊒) *e* **by** *auto*
note *monotone-on-subset*[*OF mono XA*]
note *monotone-on-dual*[*OF this*]
from *monotone-image-extreme*[*OF this Xe*]
show ∃ *e'*. *extreme X'* (⊴)$^-$ *e'* **by** (*auto simp*: *X'*)
qed

end

sublocale *well-related-set* ⊆ *reflexive* **using** *local.reflexive-axioms*.

lemmas *well-related-setI* = *well-related-set.intro*

lemmas *well-related-iff-ex-extreme* = *well-related-set-def*

lemma *well-related-set-empty*[*intro*!]: *well-related-set* {} *r*
by (*auto intro*!: *well-related-setI*)

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

lemma *well-related-iff-neg-well-founded*:
well-related-set A (⊑) ⟷ *well-founded A* (λ*x y*. ¬ *y* ⊑ *x*)
by (*simp add*: *well-related-set-def well-founded-iff-ex-extremal extreme-def Bex-def*)

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

lemma *closed-UN-well-related*:
assumes *XX*: ∀ *X*∈*XX*. *well-related-set X* (⊑) ∧ (∀ *x*∈*X*. ∀ *y*∈⋃ *XX*. ¬*x* ⊑ *y*
⟶ *y* ∈ *X*)
shows *well-related-set* (⋃ *XX*) (⊑)
using *XX*
apply (*unfold well-related-iff-neg-well-founded*)
using *closed-UN-well-founded*[*of* - λ*x y*. ¬ *y* ⊑ *x*].

end

lemma *well-related-extend*:
  fixes *r* (**infix** ‹⊑› *50*)
  assumes *well-related-set A* (⊑) **and** *well-related-set B* (⊑) **and** ∀ *a* ∈ *A*. ∀ *b* ∈ *B*. *a* ⊑ *b*
  shows *well-related-set* (*A* ∪ *B*) (⊑)
  using *well-founded-extend*[*of - λx y*. ¬ *y* ⊑ *x*, *folded well-related-iff-neg-well-founded*]
  using *assms* **by** *auto*

lemma *pair-well-related*:
  fixes *less-eq* (**infix** ‹⊑› *50*)
  assumes *i* ⊑ *i* **and** *i* ⊑ *j* **and** *j* ⊑ *j*
  shows *well-related-set* {*i*, *j*} (⊑)
proof (*intro well-related-setI*)
  fix *X* **assume** *X* ⊆ {*i*,*j*} **and** *X* ≠ {}
  then have *X* = {*i*,*j*} ∨ *X* = {*i*} ∨ *X* = {*j*} **by** *auto*
  with *assms* **show** ∃ *e*. *extreme X* (⊑)⁻ *e* **by** *auto*
qed

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

interpretation *less-eq-asymmetrize*.

sublocale *transitive*
proof
  fix *x y z* **assume** *xy*: *x* ⊑ *y* **and** *yz*: *y* ⊑ *z* **and** *x*: *x* ∈ *A* **and** *y*: *y* ∈ *A* **and** *z*: *z* ∈ *A*
   from *x y z* **have** ∃ *e*. *extreme* {*x*,*y*,*z*} (⊒) *e* (**is** ∃ *e*. *?P e*) **by** (*auto intro*!: *nonempty-imp-ex-extreme*)
  then have *?P x* ∨ *?P y* ∨ *?P z* **by** *auto*
  then show *x* ⊑ *z*
  proof (*elim disjE*)
    assume *?P x*
    then show *?thesis* **by** *auto*
  next
    assume *?P y*
    then have *y* ⊑ *x* **by** *auto*
    from *attract*[*OF xy this yz*] *x y z* **show** *?thesis* **by** *auto*
  next
    assume *?P z*
    then have *zx*: *z* ⊑ *x* **and** *zy*: *z* ⊑ *y* **by** *auto*
    from *attract*[*OF yz zy zx*] *x y z* **have** *yx*: *y* ⊑ *x* **by** *auto*
    from *attract*[*OF xy yx yz*] *x y z* **show** *?thesis* **by** *auto*
  qed
qed

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

end

lemmas *pre-well-ordered-iff-semiattractive-well-related* =
pre-well-ordered-set-def[*unfolded atomize-eq*]

lemma *pre-well-ordered-set-empty*[*intro*!]: *pre-well-ordered-set* {} *r*
by (*auto simp*: *pre-well-ordered-iff-semiattractive-well-related*)

lemma *pre-well-ordered-iff*:
pre-well-ordered-set A r ⟷ *total-quasi-ordered-set A r* ∧ *well-founded A* (*asympartp r*)
(**is** *?p* ⟷ *?t* ∧ *?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 (⊑) using A.
  interpret B: pre-well-ordered-set B (⊑) 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

lemma (in well-related-set) monotone-image-pre-well-ordered:
  fixes leB (infix ‹⊑′› 50)
  assumes mono: monotone-on A (⊑) (⊑′) f
    and image: semiattractive (f ‘ A) (⊑′)
  shows pre-well-ordered-set (f ‘ A) (⊑′)
   by (intro pre-well-ordered-set.intro monotone-image-well-related[OF mono] image)

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

sublocale pre-well-ordered-set..

sublocale total-ordered-set..

lemma well-ordered-subset: B ⊆ A =⇒ well-ordered-set B (⊑)
  using well-related-subset antisymmetric-subset by (intro well-ordered-set.intro)

sublocale asym: well-founded-ordered-set A asympartp (⊑)
 by (intro well-founded-ordered-set.intro asym.well-founded-axioms asympartp-transitive)

end

lemmas well-ordered-iff-antisymmetric-well-related = well-ordered-set-def[unfolded
```

atomize-eq]

lemma *well-ordered-set-empty*[*intro*!]: *well-ordered-set* {} *r*
  by (*auto simp*: *well-ordered-iff-antisymmetric-well-related*)

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 :: *'a set* **and** *less-eq* :: $'a \Rightarrow 'a \Rightarrow$ *bool* (**infix** ‹$\sqsubseteq$› *50*)
begin

context
  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

interpretation *less-eq-asymmetrize***.**

lemma *well-ordered-iff-well-founded-total-ordered*:
  well-ordered-set A ($\sqsubseteq$) $\longleftrightarrow$ *total-ordered-set* A ($\sqsubseteq$) $\land$ *well-founded* A ($\sqsubset$)
proof (*safe*)
  assume *well-ordered-set* A ($\sqsubseteq$)
  then interpret *well-ordered-set* A ($\sqsubseteq$)**.**
  show *total-ordered-set* A ($\sqsubseteq$) *well-founded* A ($\sqsubset$) **by** *unfold-locales*
next
  assume *total-ordered-set* A ($\sqsubseteq$) **and** *well-founded* A ($\sqsubset$)

then interpret *total-ordered-set A* (⊑) + *asympartp*: *well-founded A* (⊏)**.**
show *well-ordered-set A* (⊑)
proof
fix *X* **assume** *XA*: $X \subseteq A$ **and** $X \neq \{\}$
from *XA asympartp.nonempty-imp-ex-extremal*[*OF this*]
show ∃ *e*. *extreme X* (⊒) *e* **by** (*auto simp*: *not-asym-iff subsetD*)
qed
qed

end

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

lemma *well-order-extend*:
assumes *A*: *well-ordered-set A* (⊑) **and** *B*: *well-ordered-set B* (⊑)
and *ABa*: ∀ *a* ∈ *A*. ∀ *b* ∈ *B*. *a* ⊑ *b* ⟶ *b* ⊑ *a* ⟶ *a* = *b*
and *AB*: ∀ *a* ∈ *A*. ∀ *b* ∈ *B*. *a* ⊑ *b*
shows *well-ordered-set* (*A*∪*B*) (⊑)
proof−
interpret *A*: *well-ordered-set A* (⊑) **using** *A***.**
interpret *B*: *well-ordered-set B* (⊑) **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*} (⊑) **for** *a* **apply** *unfold-locales* **by** *auto*

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

lemma *singleton-well-ordered*[*intro*!]: *a* ⊑ *a* ⟹ *well-ordered-set* {*a*} (⊑)
apply *unfold-locales* **by** *auto*

lemma *closed-UN-well-ordered*:
assumes *anti*: *antisymmetric* (⋃ *XX*) (⊑)
and *XX*: ∀ *X*∈*XX*. *well-ordered-set X* (⊑) ∧ (∀ *x*∈*X*. ∀ *y*∈⋃ *XX*. ¬ *x* ⊑ *y* ⟶ *y* ∈ *X*)
shows *well-ordered-set* (⋃ *XX*) (⊑)
apply (*intro well-ordered-set.intro closed-UN-well-related anti*)
using *XX well-ordered-set.axioms* **by** *fast*

end

lemma (**in** *well-related-set*) *monotone-image-well-ordered*:
fixes *leB* (**infix** ‹⊑′′› *50*)

assumes *mono*: *monotone-on* A ($\sqsubseteq$) ($\sqsubseteq'$) f
and *image*: *antisymmetric* (f ‘ A) ($\sqsubseteq'$)
shows *well-ordered-set* (f ‘ A) ($\sqsubseteq'$)
by (*intro well-ordered-set.intro monotone-image-well-related*[*OF mono*] *image*)

3.1 Relating to Classes

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

lemma *well-founded-quasi-ordering-subset*:
assumes $X \subseteq A$ **shows** *well-founded-quasi-ordering* X ($\sqsubseteq$) ($\sqsubset$)
by (*intro well-founded-quasi-ordering.intro quasi-ordering-subset well-founded-subset assms*)

end

class *wf-qorder* = *ord* +
assumes *well-founded-quasi-ordering UNIV* ($\leq$) ($<$)
begin

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

subclass *qorder* **..**

sublocale *order*: *well-founded-quasi-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

context *wellorder* **begin**

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

sublocale *order*: *well-ordered-set UNIV*

rewrites $\bigwedge x.\ x \in \mathit{UNIV} \equiv \mathit{True}$
and $\bigwedge X.\ X \subseteq \mathit{UNIV} \equiv \mathit{True}$
and $\bigwedge r.\ r \upharpoonright \mathit{UNIV} \equiv r$
and $\bigwedge P.\ \mathit{True} \wedge P \equiv P$
and $\mathit{Ball\ UNIV} \equiv \mathit{All}$
and $\mathit{Bex\ UNIV} \equiv \mathit{Ex}$
and $\mathit{sympartp}\ (\le)^{-} \equiv \mathit{sympartp}\ (\le)$
and $\bigwedge P1.\ (\mathit{True} \Longrightarrow \mathit{PROP}\ P1) \equiv \mathit{PROP}\ P1$
and $\bigwedge P1.\ (\mathit{True} \Longrightarrow P1) \equiv \mathit{Trueprop}\ P1$
and $\bigwedge P1\ P2.\ (\mathit{True} \Longrightarrow \mathit{PROP}\ P1 \Longrightarrow \mathit{PROP}\ P2) \equiv (\mathit{PROP}\ P1 \Longrightarrow \mathit{PROP}\ P2)$
apply (*unfold well-ordered-iff-well-founded-total-ordered*)
apply (*intro conjI order.total-ordered-set-axioms*)
by (*auto simp*: *order.well-founded-axioms atomize-eq*)

end

thm *order.nonempty-imp-ex-extreme*

3.2 omega-Chains

definition *omega-chain A r* $\equiv \exists f :: \mathit{nat} \Rightarrow 'a.\ \mathit{monotone}\ (\le)\ r\ f \wedge \mathit{range}\ f = A$

lemma *omega-chainI*:
fixes $f :: \mathit{nat} \Rightarrow 'a$
assumes *monotone* $(\le)$ *r f range f* $= A$ **shows** *omega-chain A r*
using *assms* **by** (*auto simp*: *omega-chain-def*)

lemma *omega-chainE*:
assumes *omega-chain A r*
and $\bigwedge f :: \mathit{nat} \Rightarrow 'a.\ \mathit{monotone}\ (\le)\ r\ f \Longrightarrow \mathit{range}\ f = A \Longrightarrow \mathit{thesis}$
shows *thesis*
using *assms* **by** (*auto simp*: *omega-chain-def*)

lemma (**in** *transitive*) *local-chain*:
assumes *CA*: *range* $C \subseteq A$
shows $(\forall i{::}\mathit{nat}.\ C\ i \sqsubseteq C\ (\mathit{Suc}\ i)) \longleftrightarrow \mathit{monotone}\ (<)\ (\sqsubseteq)\ C$
proof (*intro iffI allI monotoneI*)
fix $i\ j :: \mathit{nat}$
assume *loc*: $\forall i.\ C\ i \sqsubseteq C\ (\mathit{Suc}\ i)$ **and** *ij*: $i < j$
have $C\ i \sqsubseteq C\ (i{+}k{+}1)$ **for** *k*
proof (*induct k*)
case *0*
from *loc* **show** *?case* **by** *auto*
next
case (*Suc k*)
also have $C\ (i{+}k{+}1) \sqsubseteq C\ (i{+}k{+}\mathit{Suc}\ 1)$ **using** *loc* **by** *auto*
finally show *?case* **using** *CA* **by** *auto*
qed

from *this*[*of j−i−1*] *ij* **show** $C\ i \sqsubseteq C\ j$ **by** *auto*
next
fix *i*
assume *monotone* (<) (⊑) *C*
then show $C\ i \sqsubseteq C\ (Suc\ i)$ **by** (*auto dest*: *monotoneD*)
qed

lemma *pair-omega-chain*:
assumes *r a a r b b r a b* **shows** *omega-chain* {*a*,*b*} *r*
using *assms* **by** (*auto intro*!: *omega-chainI*[*of r* λ*i*. *if i* = *0 then a else b*] *monotoneI*)

Every omega-chain is a well-order.

lemma *omega-chain-imp-well-related*:
fixes *less-eq* (**infix** ‹⊑› *50*)
assumes *A*: *omega-chain A* (⊑) **shows** *well-related-set A* (⊑)
proof
interpret *less-eq-dualize***.**
from *A* **obtain** *f* :: *nat* ⇒ *'a* **where** *mono*: *monotone-on UNIV* (≤) (⊑) *f* **and** *A*: *A* = *range f*
by (*auto elim*!: *omega-chainE*)
fix *X* **assume** *XA*: *X* ⊆ *A* **and** *X* ≠ {}
then obtain *I* **where** *X*: *X* = *f* ‘ *I* **and** *I0*: *I* ≠ {} **by** (*auto simp*: *A subset-image-iff*)
from *order.nonempty-imp-ex-extreme*[*OF I0*]
obtain *i* **where** *least I i* **by** *auto*
with *mono*
show ∃ *e*. *extreme X* (⊒) *e* **by** (*auto intro*!: *exI*[*of - f i*] *extremeI simp*: *X monotoneD*)
qed

lemma (**in** *semiattractive*) *omega-chain-imp-pre-well-ordered*:
assumes *omega-chain A* (⊑) **shows** *pre-well-ordered-set A* (⊑)
apply (*intro pre-well-ordered-set.intro omega-chain-imp-well-related assms*)**..**

lemma (**in** *antisymmetric*) *omega-chain-imp-well-ordered*:
assumes *omega-chain A* (⊑) **shows** *well-ordered-set A* (⊑)
by (*intro well-ordered-set.intro omega-chain-imp-well-related assms antisymmetric-axioms*)

3.2.1 Relation image that preserves well-orderedness.

definition *well-image f A* (⊑) *fa fb* ≡
∀ *a b*. *extreme* {*x*∈*A*. *fa* = *f x*} (⊑)⁻ *a* ⟶ *extreme* {*y*∈*A*. *fb* = *f y*} (⊑)⁻ *b* ⟶ *a* ⊑ *b*
for *less-eq* (**infix** ‹⊑› *50*)

lemmas *well-imageI* = *well-image-def*[*unfolded atomize-eq*, *THEN iffD2*, *rule-format*]
lemmas *well-imageD* = *well-image-def*[*unfolded atomize-eq*, *THEN iffD1*, *rule-format*]

```
lemma (in pre-well-ordered-set)
  well-image-well-related: pre-well-ordered-set (f`A) (well-image f A (⊑))
proof−
  interpret less-eq-dualize.
  interpret im: transitive f`A well-image f A (⊑)
  proof (safe intro!: transitiveI well-imageI)
    interpret less-eq-dualize.
    fix x y z a c
    assume fxfy: well-image f A (⊑) (f x) (f y)
      and fyfz: well-image f A (⊑) (f y) (f z)
      and xA: x ∈ A and yA: y ∈ A and zA: z ∈ A
      and a: extreme {a ∈ A. f x = f a} (⊒) a
      and c: extreme {c ∈ A. f z = f c} (⊒) c
    have ∃ b. extreme {b ∈ A. f y = f b} (⊒) b
      apply (rule nonempty-imp-ex-extreme) using yA by auto
    then obtain b where b: extreme {b ∈ A. f y = f b} (⊒) b by auto
    from trans[OF well-imageD[OF fxfy a b] well-imageD[OF fyfz b c]] a b c
    show a ⊑ c by auto
  qed
  interpret im: well-related-set f`A well-image f A (⊑)
  proof
    fix fX
    assume fXfA: fX ⊆ f ` A and fX0: fX ≠ {}
    define X where X ≡ {x∈A. f x ∈ fX}
    with fXfA fX0 have XA: X ⊆ A and X ≠ {} by (auto simp: ex-in-conv[symmetric])
    from nonempty-imp-ex-extreme[OF this] obtain e where Xe: extreme X (⊒)
e by auto
    with XA have eA: e ∈ A by auto
    from fXfA have fX: f ` X = fX by (auto simp: X-def intro!: equalityI)
    show ∃ fe. extreme fX (well-image f A (⊑))⁻ fe
    proof (safe intro!: exI extremeI elim!: subset-imageE)
      from Xe fX show fefX: f e ∈ fX by auto
      fix fx assume fxfX: fx ∈ fX
      show well-image f A (⊑) (f e) fx
      proof (intro well-imageI)
        fix a b
        assume fea: extreme {a ∈ A. f e = f a} (⊒) a
          and feb: extreme {b ∈ A. fx = f b} (⊒) b
        from fea eA have aA: a ∈ A and ae: a ⊑ e by auto
        from feb fxfX have bA: b ∈ A and bX: b ∈ X by (auto simp: X-def)
        with Xe have eb: e ⊑ b by auto
        from trans[OF ae eb aA eA bA]
        show a ⊑ b.
      qed
    qed
  qed
  show ?thesis by unfold-locales
qed
```

end
theory *Directedness*
  imports *Binary-Relations Well-Relations*
begin

Directed sets:

locale *directed* =
  fixes *A* **and** *less-eq* (**infix** ‹⊑› *50*)
  assumes *pair-bounded*: $x \in A \Longrightarrow y \in A \Longrightarrow \exists z \in A.\ x \sqsubseteq z \wedge y \sqsubseteq z$

lemmas *directedI*[*intro*] = *directed.intro*

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** $x \in A$ **and** $y \in A$
    and $\bigwedge z.\ z \in A \Longrightarrow x \sqsubseteq z \Longrightarrow y \sqsubseteq z \Longrightarrow$ *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*: $\forall x \in X.\ \forall y \in Y.\ \exists z \in X \cup Y.\ x \sqsubseteq z \wedge y \sqsubseteq z$
  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*: $\forall x \in X.\ \forall y \in Y.\ x$
$\sqsubseteq y$
  shows *directed* (*X* ∪ *Y*) (⊑)
proof −
  { **fix** *x y*
    assume *xX*: $x \in X$ **and** *yY*: $y \in Y$
    let *?g* = $\exists z \in X \cup Y.\ x \sqsubseteq z \wedge y \sqsubseteq z$
    from *directedD*[*OF Y yY yY*] **obtain** *z* **where** *zY*: $z \in Y$ **and** *yz*: $y \sqsubseteq z$ **by**
auto
    from *xX XY zY yz* **have** *?g* **by** *auto*
  }
  then show *?thesis* **by** (*auto intro*!: *directed-union*[*OF X Y*])
qed

end

sublocale *connex* ⊆ *directed*
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

lemmas(**in** *connex*) *directed* = *directed-axioms*

lemma *monotone-directed-image*:
  fixes *ir* (**infix** ‹⪯› *50*) **and** *r* (**infix** ‹⊑› *50*)
  assumes *mono*: *monotone-on I* (⪯) (⊑) *f* **and** *dir*: *directed I* (⪯)
  shows *directed* (*f* ‘ *I*) (⊑)
proof (*rule directedI*, *safe*)
  fix *x y* **assume** *x*: *x* ∈ *I* **and** *y*: *y* ∈ *I*
  with *dir* **obtain** *z* **where** *z*: *z* ∈ *I* **and** *x* ⪯ *z* **and** *y* ⪯ *z* **by** (*auto elim*: *directedE*)
  with *mono x y* **have** *f x* ⊑ *f z* **and** *f y* ⊑ *f z* **by** (*auto dest*: *monotone-onD*)
  with *z* **show** ∃ *fz* ∈ *f* ‘ *I*. *f x* ⊑ *fz* ∧ *f y* ⊑ *fz* **by** *auto*
qed

definition *directed-set A* (⊑) ≡ ∀ *X* ⊆ *A*. *finite X* ⟶ (∃ *b* ∈ *A*. *bound X* (⊑) *b*)
  for *less-eq* (**infix** ‹⊑› *50*)

lemmas *directed-setI* = *directed-set-def*[*unfolded atomize-eq*, *THEN iffD2*, *rule-format*]
lemmas *directed-setD* = *directed-set-def*[*unfolded atomize-eq*, *THEN iffD1*, *rule-format*]

lemma *directed-imp-nonempty*:
  fixes *less-eq* (**infix** ‹⊑› *50*)
  shows *directed-set A* (⊑) ⟹ *A* ≠ {}
  by (*auto simp*: *directed-set-def*)

lemma *directedD2*:
  fixes *less-eq* (**infix** ‹⊑› *50*)
  assumes *dir*: *directed-set A* (⊑) **and** *xA*: *x* ∈ *A* **and** *yA*: *y* ∈ *A*
  shows ∃ *z* ∈ *A*. *x* ⊑ *z* ∧ *y* ⊑ *z*
  using *directed-setD*[*OF dir*, *of* {*x*,*y*}] *xA yA* **by** *auto*

lemma *monotone-directed-set-image*:

fixes *ir* (**infix** ‹⪯› *50*) **and** *r* (**infix** ‹⊑› *50*)
assumes *mono*: *monotone-on I* (⪯) (⊑) *f* **and** *dir*: *directed-set I* (⪯)
shows *directed-set* (*f* ‘ *I*) (⊑)
proof (*rule directed-setI*)
fix *X* **assume** *finite X* **and** $X \subseteq f$ ‘ *I*
from *finite-subset-image*[*OF this*]
obtain *J* **where** *JI*: $J \subseteq I$ **and** *Jfin*: *finite J* **and** *X*: $X = f$ ‘ *J* **by** *auto*
from *directed-setD*[*OF dir JI Jfin*]
obtain *b* **where** *bI*: $b \in I$ **and** *Jb*: *bound J* (⪯) *b* **by** *auto*
from *monotone-image-bound*[*OF mono JI bI Jb*] *bI*
show *Bex* (*f* ‘ *I*) (*bound X* (⊑)) **by** (*auto simp*: *X*)
qed

lemma *directed-set-iff-extremed*:
fixes *less-eq* (**infix** ‹⊑› *50*)
assumes *Dfin*: *finite D*
shows *directed-set D* (⊑) ⟷ *extremed D* (⊑)
proof (*intro iffI directed-setI conjI*)
assume *directed-set D* (⊑)
from *directed-setD*[*OF this order.refl Dfin*]
show *extremed D* (⊑) **by** (*auto intro*: *extremedI*)
next
fix *X* **assume** *XD*: $X \subseteq D$ **and** *Xfin*: *finite X*
assume *extremed D* (⊑)
then obtain *b* **where** $b \in D$ **and** *extreme D* (⊑) *b* **by** (*auto elim*!: *extremedE*)
with *XD* **show** $\exists\, b \in D$. *bound X* (⊑) *b* **by** *auto*
qed

lemma (**in** *transitive*) *directed-iff-nonempty-pair-bounded*:
directed-set A (⊑) ⟷ $A \neq \{\} \wedge (\forall x{\in}A.\ \forall y{\in}A.\ \exists z{\in}A.\ x \sqsubseteq z \wedge y \sqsubseteq z)$
(**is** *?l* ⟷ - ∧ *?r*)
proof (*safe intro*!: *directed-setI del*: *notI subset-antisym*)
assume *dir*: *?l*
from *directed-imp-nonempty*[*OF dir*] **show** $A \neq \{\}$**.**
fix *x y* **assume** $x \in A$ $y \in A$
with *directed-setD*[*OF dir*, *of* {*x*,*y*}]
show $\exists z{\in}A.\ x \sqsubseteq z \wedge y \sqsubseteq z$ **by** *auto*
next
fix *X*
assume *A0*: $A \neq \{\}$ **and** *r*: *?r*
assume *finite X* **and** $X \subseteq A$
then show *Bex A* (*bound X* (⊑))
proof (*induct*)
case *empty*
with *A0* **show** *?case* **by** (*auto simp*: *bound-empty ex-in-conv*)
next
case (*insert x X*)
then obtain *y* **where** *xA*: $x \in A$ **and** *XA*: $X \subseteq A$ **and** *yA*: $y \in A$ **and** *Xy*:

bound X (⊑) *y* **by** *auto*
  from *r yA xA* **obtain** *z* **where** *zA*: *z* ∈ *A* **and** *xz*: *x* ⊑ *z* **and** *yz*: *y* ⊑ *z* **by** *auto*
  from *bound-trans*[*OF Xy yz XA yA zA*] *xz zA*
  show *?case* **by** *auto*
 qed
qed

lemma (**in** *transitive*) *directed-set-iff-nonempty-directed*:
 directed-set A (⊑) ⟷ *A* ≠ {} ∧ *directed A* (⊑)
 apply (*unfold directed-iff-nonempty-pair-bounded*)
 by (*auto simp*: *directed-def*)

lemma (**in** *well-related-set*) *finite-sets-extremed*:
 assumes *fin*: *finite X* **and** *X0*: *X* ≠ {} **and** *XA*: *X* ⊆ *A*
 shows *extremed X* (⊑)
proof−
 interpret *less-eq-asymmetrize*.
 from *fin X0 XA* **show** *?thesis*
 proof (*induct card X arbitrary*: *X*)
  case *0*
  then show *?case* **by** *auto*
 next
  case (*Suc n*)
  note *XA* = ‹*X* ⊆ *A*› **and** *X0* = ‹*X* ≠ {}› **and** *Sn* = ‹*Suc n* = *card X*› **and** *finX* = ‹*finite X*›
  note *IH* = *Suc*(*1*)
  from *nonempty-imp-ex-extreme*[*OF XA X0*]
  obtain *l* **where** *l*: *extreme X* (⊒) *l* **by** *auto*
  from *l* **have** *lX*: *l* ∈ *X* **by** *auto*
  with *XA* **have** *lA*: *l* ∈ *A* **by** *auto*
  from *Sn lX* **have** *n*: *n* = *card* (*X*−{*l*}) **by** *auto*
  show *?case*
  proof (*cases X* − {*l*} = {})
   case *True*
   with *lA lX* **show** *?thesis* **by** (*auto intro*!: *extremedI*)
  next
   case *False*
   from *IH*[*OF n - this*] *finX XA*
   obtain *e* **where** *e*: *extreme* (*X* − {*l*}) (⊑) *e* **by** (*auto elim*!: *extremedE*)
   with *l* **show** *?thesis* **by** (*auto intro*!: *extremedI*[*of - - e*] *extremeI*)
  qed
 qed
qed

lemma (**in** *well-related-set*) *directed-set*:
 assumes *A0*: *A* ≠ {} **shows** *directed-set A* (⊑)
proof (*intro directed-setI*)
 fix *X* **assume** *XA*: *X* ⊆ *A* **and** *Xfin*: *finite X*

show *Bex A* (*bound X* (⊑))
proof (*cases X* = {})
case *True*
with *A0* **show** *?thesis* **by** (*auto simp*: *bound-empty ex-in-conv*)
next
case *False*
from *finite-sets-extremed*[*OF Xfin this*] *XA*
show *?thesis* **by** (*auto elim*!: *extremedE*)
qed
qed

lemma *prod-directed*:
fixes *leA* (**infix** ‹$\sqsubseteq_A$› *50*) **and** *leB* (**infix** ‹$\sqsubseteq_B$› *50*)
assumes *dir*: *directed X* (*rel-prod* ($\sqsubseteq_A$) ($\sqsubseteq_B$))
shows *directed* (*fst* ‘ *X*) ($\sqsubseteq_A$) **and** *directed* (*snd* ‘ *X*) ($\sqsubseteq_B$)
proof (*safe intro*!: *directedI*)
fix *a b x y* **assume** (*a*,*x*) ∈ *X* **and** (*b*,*y*) ∈ *X*
from *directedD*[*OF dir this*]
obtain *c z* **where** *cz*: (*c*,*z*) ∈ *X* **and** *ac*: *a* $\sqsubseteq_A$ *c* **and** *bc*: *b* $\sqsubseteq_A$ *c* **and** *x* $\sqsubseteq_B$ *z* **and** *y* $\sqsubseteq_B$ *z* **by** *auto*
then show ∃ *z*∈*fst* ‘ *X*. *fst* (*a*,*x*) $\sqsubseteq_A$ *z* ∧ *fst* (*b*,*y*) $\sqsubseteq_A$ *z*
and ∃ *z*∈*snd* ‘ *X*. *snd* (*a*,*x*) $\sqsubseteq_B$ *z* ∧ *snd* (*b*,*y*) $\sqsubseteq_B$ *z*
by (*auto intro*!: *bexI*[*OF - cz*])
qed

class *dir* = *ord* + **assumes** *directed UNIV* (≤)
begin

sublocale *order*: *directed 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 *dir-axioms*[*unfolded class.dir-def*]
by (*auto simp*: *atomize-eq*)

end

class *filt* = *ord* +
assumes *directed UNIV* (≥)
begin

sublocale *order.dual*: *directed 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* ≡ *All*
and *Bex UNIV* ≡ *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 $\bigwedge P1\ P2.\ (True \Longrightarrow PROP\ P1 \Longrightarrow PROP\ P2) \equiv (PROP\ P1 \Longrightarrow PROP\ P2)$
using *filt-axioms*[*unfolded class.filt-def*]
by (*auto simp*: *atomize-eq*)

end

subclass (**in** *linqorder*) *dir*..

subclass (**in** *linqorder*) *filt*..

thm *order.directed-axioms*[**where** *'a* = *'a* ::*dir*]

thm *order.dual.directed-axioms*[**where** *'a* = *'a* ::*filt*]

end

4 Completeness of Relations

Here we formalize various order-theoretic completeness conditions.

theory *Complete-Relations*
imports *Well-Relations Directedness*
begin

4.1 Completeness Conditions

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

definition *complete* (‹-—*complete*›[*999*]*1000*) **where**
$\mathcal{C}$−*complete* *A* (⊑) ≡ ∀ *X* ⊆ *A*. $\mathcal{C}$ *X* (⊑) ⟶ (∃ *s*. *extreme-bound A* (⊑) *X s*) **for** *less-eq* (**infix** ‹⊑› *50*)

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

lemma *complete-cmono*: *CC* ≤ *DD* ⟹ *DD*−*complete* ≤ *CC*−*complete*

by (*force simp*: *complete-def*)

lemma *complete-subclass*:
fixes *less-eq* (**infix** ‹⊑› *50*)
assumes $\mathcal{C}$−*complete* *A* (⊑) **and** ∀ *X* ⊆ *A*. $\mathcal{D}$ *X* (⊑) ⟶ $\mathcal{C}$ *X* (⊑)
shows $\mathcal{D}$−*complete* *A* (⊑)
using *assms* **by** (*auto simp*: *complete-def*)

lemma *complete-empty*[*simp*]: $\mathcal{C}$−*complete* {} *r* ⟷ ¬ $\mathcal{C}$ {} *r* **by** (*auto simp*: *complete-def*)

context
fixes *less-eq* :: ′*a* ⇒ ′*a* ⇒ *bool* (**infix** ‹⊑› *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* (⊑) ⟷ (λ*X r*. *X* = *A*)−*complete* *A* (⊑) (**is** *?l* ⟷ *?r*)
proof
assume *?l*
then obtain *e* **where** *extreme A* (⊑) *e* **by** (*erule extremedE*)
then have *extreme-bound A* (⊑) *A e* **by** *auto*
then show *?r* **by** (*auto intro*!: *completeI*)
next
assume *?r*
from *completeD*[*OF this*] **obtain** *s* **where** *extreme-bound A* (⊑) *A s* **by** *auto*
then have *extreme A* (⊑) *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* (⊑) ⟷ (λ*X r*. *X* = {})−*complete* *A* (⊑)$^{-}$
by (*auto simp*: *complete-def extremed-def*)

Downward closure is topped.

lemma *dual-closure-is-extremed*:
assumes *bA*: *b* ∈ *A* **and** *b* ⊑ *b*
shows *extremed* {*a* ∈ *A*. *a* ⊑ *b*} (⊑)
using *assms* **by** (*auto intro*!: *extremedI*[*of* - - *b*])

Downward closure preserves completeness.

lemma *dual-closure-is-complete*:
assumes *A*: $\mathcal{C}$−*complete* *A* (⊑) **and** *bA*: *b* ∈ *A*
shows $\mathcal{C}$−*complete* {*x*∈*A*. *x* ⊑ *b*} (⊑)
proof (*intro completeI*)

fix X **assume** XAb:$X \subseteq \{x \in A.\ x \sqsubseteq b\}$ **and** X: $\mathcal{C}\ X\ (\sqsubseteq)$
with *completeD*[*OF A*] **obtain** x **where** x: *extreme-bound* A $(\sqsubseteq)$ X x **by** *auto*
then have xA: $x \in A$ **by** *auto*
from XAb **have** $x \sqsubseteq b$ **by** (*auto intro*!: *extreme-boundD*[*OF x*] *bA*)
with xA x **show** $\exists x.$ *extreme-bound* $\{x \in A.\ x \sqsubseteq b\}$ $(\sqsubseteq)$ X x **by** (*auto intro*!: *exI*[*of - x*])
qed

interpretation *less-eq-dualize*.

Upward closure preserves completeness, under a condition.

lemma *closure-is-complete*:
assumes A: $\mathcal{C}-$*complete* A $(\sqsubseteq)$ **and** bA: $b \in A$
and Cb: $\forall X \subseteq A.\ \mathcal{C}\ X\ (\sqsubseteq) \longrightarrow$ *bound* X $(\sqsupseteq)$ $b \longrightarrow \mathcal{C}\ (X \cup \{b\})\ (\sqsubseteq)$
shows $\mathcal{C}-$*complete* $\{x{\in}A.\ b \sqsubseteq x\}$ $(\sqsubseteq)$
proof (*intro completeI*)
fix X **assume** XAb:$X \subseteq \{x \in A.\ b \sqsubseteq x\}$ **and** X: $\mathcal{C}\ X\ (\sqsubseteq)$
with Cb **have** XbC: $\mathcal{C}\ (X \cup \{b\})\ (\sqsubseteq)$ **by** *auto*
from XAb bA **have** XbA: $X \cup \{b\} \subseteq A$ **by** *auto*
with *completeD*[*OF A XbA*] XbC
obtain x **where** x: *extreme-bound* A $(\sqsubseteq)$ $(X{\cup}\{b\})$ x **by** *auto*
then show $\exists x.$ *extreme-bound* $\{x \in A.\ b \sqsubseteq x\}$ $(\sqsubseteq)$ X x
by (*auto intro*!: *exI*[*of - x*] *extreme-boundI*)
qed

lemma *biclosure-is-complete*:
assumes A: $\mathcal{C}-$*complete* A $(\sqsubseteq)$ **and** aA: $a \in A$ **and** bA: $b \in A$ **and** ab: $a \sqsubseteq b$
and Ca: $\forall X \subseteq A.\ \mathcal{C}\ X\ (\sqsubseteq) \longrightarrow$ *bound* X $(\sqsupseteq)$ $a \longrightarrow \mathcal{C}\ (X \cup \{a\})\ (\sqsubseteq)$
shows $\mathcal{C}-$*complete* $\{x{\in}A.\ a \sqsubseteq x \wedge x \sqsubseteq b\}$ $(\sqsubseteq)$
proof−
note *closure-is-complete*[*OF A aA Ca*]
from *dual-closure-is-complete*[*OF this*, *of b*] *bA ab* **show** *?thesis* **by** *auto*
qed

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* $\equiv$ $(\lambda X\ r.\ \exists x\ y.\ X = \{x,y\})-$*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** ‹⊑› *50*)
begin

lemma *pair-complete-imp-directed*:
assumes *comp*: *pair-complete A* (⊑) **shows** *directed A* (⊑)
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*: $x \in A$ **and** *y*: $y \in A$
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* ≡ (λ*X r*. *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* ≡ (λ*X r*. ∃ *b* ∈ *A*. *bound X r b* ∧ *X* ≠ {})−*complete A*

lemma *conditionally-complete-imp-nonempty-imp-ex-extreme-bound-iff-ex-bound*:
assumes *comp*: *conditionally-complete A r*
assumes $X \subseteq A$ **and** $X \neq \{\}$
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.

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* ≡ (λ*X r*. *directed X r* ∧ *X* ≠ {})−*complete*

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*

lemma (**in** *reflexive*) *dual-closure-is-directed-complete*:
  assumes *comp*: *directed*−*complete A* (⊑) **and** *bA*: *b* ∈ *A*
  shows *directed*−*complete* {*X* ∈ *A*. *b* ⊑ *X*} (⊑)
  apply (*rule closure-is-complete*[*OF comp bA*])
proof (*intro allI impI directedI CollectI*)
  interpret *less-eq-dualize*.
  fix *X x y* **assume** *Xdir*: *directed X* (⊑) **and** *X*: *X* ⊆ *A*
    and *bX*: *bound X* (⊒) *b* **and** *x*: *x* ∈ *X* ∪ {*b*} **and** *y*: *y* ∈ *X* ∪ {*b*}
  from *x y X bA* **have** *xA*: *x* ∈ *A* **and** *yA*: *y* ∈ *A* **by** *auto*
  show ∃ *z*∈*X* ∪ {*b*}. *x* ⊑ *z* ∧ *y* ⊑ *z*
  proof (*cases x* = *b*)
    case [*simp*]: *True*
    with *y bX bA* **have** *b* ⊑ *y* **by** *auto*
    with *y yA bA* **show** *?thesis* **by** (*auto intro*!: *bexI*[*of* - *y*])
  next
    case *False*
    with *x* **have** *x*: *x* ∈ *X* **by** *auto*
    show *?thesis*
    proof (*cases y* = *b*)
      case [*simp*]: *True*
      with *x bX* **have** *b* ⊑ *x* **by** *auto*
      with *x xA bA* **show** *?thesis* **by** (*auto intro*!: *bexI*[*of* - *x*])
    next
      case *False*
      with *y* **have** *y*: *y* ∈ *X* **by** *auto*
      from *directedD*[*OF Xdir x y*] **show** *?thesis* **by** *simp*
    qed
  qed
qed

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* ≡ (λ*X r*. *X* ≠ {})−*complete*

lemma *semicomplete-nonempty-imp-extremed*:
  $semicomplete\ A\ r \Longrightarrow A \neq \{\} \Longrightarrow extremed\ A\ r$
  unfolding *extremed-iff-UNIV-complete*
  using *complete-cmono*[*of* $\lambda X\ r.\ X = A\ \lambda X\ r.\ X \neq \{\}$]
  by *auto*

lemma *connex-dual-semicomplete*: $semicomplete\ \{C.\ connex\ C\ r\}\ (\supseteq)$
proof (*intro completeI*)
  fix X
  assume $X \subseteq \{C.\ connex\ C\ r\}$ **and** $X \neq \{\}$
  then have $connex\ (\bigcap X)\ r$ **by** (*auto simp*: *connex-def*)
  then have $extreme\text{-}bound\ \{C.\ connex\ C\ r\}\ (\supseteq)\ X\ (\bigcap X)$ **by** *auto*
  then show $\exists S.\ extreme\text{-}bound\ \{C.\ connex\ C\ r\}\ (\supseteq)\ X\ S$ **by** *auto*
qed

4.2 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-sup*: $(CC \sqcup CC')-complete\ A\ r \longleftrightarrow CC-complete\ A\ r \wedge CC'-complete\ A\ r$
  by (*auto simp*: *complete-def*)

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

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

abbreviation $bounded\text{-}complete\ A \equiv (\lambda X\ r.\ \exists\, b{\in}A.\ bound\ X\ r\ b)-complete\ A$

4.3 Relations between Completeness Conditions

context
  fixes $less\text{-}eq :: 'a \Rightarrow 'a \Rightarrow bool$ (**infix** ‹⊑› *50*)
begin

interpretation *less-eq-dualize*.

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* (⊑) **and** *pc*: *pair-complete A* (⊑) **and** *A*: *A* ≠ {}
shows *extremed A* (⊑)
proof−
have ∃ *s*. *extreme-bound A* (⊑) *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* (⊑) *A t* **by** *auto*
then have ∀ *x* ∈ *A*. *x* ⊑ *t* **and** *t* ∈ *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* ≠ {}
shows *semicomplete A* (⊑) ⟷ *conditionally-complete A* (⊑) ∧ *extremed A* (⊑) (**is** *?l* ⟷ *?r*)
proof (*intro iffI*)
assume *r*: *?r*
then have *cc*: *conditionally-complete A* (⊑) **and** *e*: *extremed A* (⊑) **by** *auto*
show *?l*
proof (*intro completeI*)
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*:
⊤−*complete A* (⊑) ⟷ *semicomplete A* (⊑) ∧ *extremed A* (⊒) (**is** *?l* ⟷ *?r*)
by (*unfold pointed-iff-empty-complete complete-sup*[*symmetric*], *auto simp*: *sup-fun-def top-fun-def*)

Conditional completeness only lacks top and bottom to be complete.

proposition *complete-iff-conditionally-complete-extremed-pointed*:
⊤−*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

4.4 Duality of Completeness Conditions

Conditional completeness is symmetric.

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

interpretation *less-eq-dualize*.

lemma *conditionally-complete-dual*:
  assumes *comp*: *conditionally-complete A* (⊑) **shows** *conditionally-complete A* (⊒)
proof (*intro completeI*)
  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*] *extreme-boundI*, *auto*)
qed

Full completeness is symmetric.

lemma *complete-dual*:
  ⊤−*complete A* (⊑) ⟹ ⊤−*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 *bounded-complete-iff-pointed-conditionally-complete*:
assumes *A*: $A \neq \{\}$
shows *bounded-complete* A ($\sqsubseteq$) $\longleftrightarrow$ *conditionally-complete* A ($\sqsubseteq$) $\wedge$ *extremed* A ($\sqsupseteq$)
apply (*unfold pointed-iff-empty-complete*)
apply (*fold complete-sup*)
apply (*unfold sup-fun-def*)
apply (*rule arg-cong*[*of - -* λCC. $CC-$*complete* A ($\sqsubseteq$)])
using *A* **by** *auto*

proposition *bounded-complete-iff-dual-semicomplete*:
bounded-complete A ($\sqsubseteq$) $\longleftrightarrow$ *semicomplete* A ($\sqsupseteq$)
proof (*cases* $A = \{\}$)
case *True*
then show *?thesis* **by** *auto*
next
case *False*
then show *?thesis*
apply (*unfold bounded-complete-iff-pointed-conditionally-complete*[*OF False*])
apply (*unfold semicomplete-iff-conditionally-complete-extremed*)
using *Complete-Relations.conditionally-complete-dual* **by** *auto*
qed

lemma *bounded-complete-imp-conditionally-complete*:
assumes *bounded-complete* A ($\sqsubseteq$) **shows** *conditionally-complete* A ($\sqsubseteq$)
using *assms* **by** (*cases* $A = \{\}$, *auto simp*:*bounded-complete-iff-pointed-conditionally-complete*)

Completeness in downward-closure:

lemma *conditionally-complete-imp-semicomplete-in-dual-closure*:
assumes *A*: *conditionally-complete* A ($\sqsubseteq$) **and** *bA*: $b \in A$
shows *semicomplete* $\{a \in A.\ a \sqsubseteq b\}$ ($\sqsubseteq$)
proof (*intro completeI*)
fix X **assume** *X*: $X \subseteq \{a \in A.\ a \sqsubseteq b\}$ **and** *X0*: $X \neq \{\}$
then have $X \subseteq A$ **and** *Xb*: *bound* X ($\sqsubseteq$) b **by** *auto*
with *bA completeD*[*OF A*] *X0* **obtain** s **where** *Xs*: *extreme-bound* A ($\sqsubseteq$) X s
by *auto*
with *Xb bA* **have** *sb*: $s \sqsubseteq b$ **by** *auto*
with *Xs* **have** *extreme-bound* $\{a \in A.\ a \sqsubseteq b\}$ ($\sqsubseteq$) X s
by (*intro extreme-boundI*, *auto*)
then show $\exists s.$ *extreme-bound* $\{a \in A.\ a \sqsubseteq b\}$ ($\sqsubseteq$) X s **by** *auto*
qed

end

Completeness in intervals:

lemma *conditionally-complete-imp-complete-in-interval*:

fixes *less-eq* (**infix** ‹⊑› *50*)
assumes *comp*: *conditionally-complete A* (⊑) **and** *aA*: $a \in A$ **and** *bA*: $b \in A$
and *aa*: $a \sqsubseteq a$ **and** *ab*: $a \sqsubseteq b$
shows ⊤−*complete* $\{x \in A.\ a \sqsubseteq x \wedge x \sqsubseteq b\}$ (⊑)
proof (*intro completeI*)
fix *X* **assume** *X*: $X \subseteq \{x \in A.\ a \sqsubseteq x \wedge x \sqsubseteq b\}$
note *conditionally-complete-imp-semicomplete-in-dual-closure*[*OF comp bA*]
from *closure-is-complete*[*OF this*, *of a*,*simplified*] *aA ab*
have *semi*: *semicomplete* $\{x \in A.\ a \sqsubseteq x \wedge x \sqsubseteq b\}$ (⊑) **by** (*simp add*: *conj-commute cong*: *Collect-cong*)
show *Ex* (*extreme-bound* $\{x \in A.\ a \sqsubseteq x \wedge x \sqsubseteq b\}$ (⊑) *X*)
proof (*cases* $X = \{\}$)
case *True*
with *aA aa ab* **have** *extreme-bound* $\{x \in A.\ a \sqsubseteq x \wedge x \sqsubseteq b\}$ (⊑) *X a* **by** (*auto simp*: *bound-empty*)
then show *?thesis* **by** *auto*
next
case *False*
with *completeD*[*OF semi X*] **show** *?thesis* **by** *simp*
qed
qed

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

4.5 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* $\Longrightarrow X \subseteq A \Longrightarrow X \neq \{\} \Longrightarrow$ *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*: $x \in A$ **and** *X*: $X \subseteq A$ **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. [9] 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*)
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* ∈ *X* **by** *auto*
with *fc*[*THEN completeD*, *of* {*x*}] *XA*
obtain *x′* **where** *extreme-bound A* (⊑) {*x*} *x′* **by** *auto*
with *xX* **have** *x′B*: *x′* ∈ *?B* **by** (*auto intro*!: *exI*[*of* - {*x*}] *extreme-boundI*)
then have *2*: *?B* ≠ {} **by** *auto*
from *dc*[*unfolded directed-complete-def*, *THEN completeD*, *of ?B*] *1 2*
obtain *b* **where** *b*: *extreme-bound A* (⊑) *?B b* **by** *auto*
then have *bA*: *b* ∈ *A* **by** *auto*
show *Ex* (*extreme-bound A* (⊑) *X*)
proof (*intro exI extreme-boundI UNIV-I*)
fix *x*
assume *xX*: *x* ∈ *X*
with *XA fc*[*THEN completeD*, *of* {*x*}]
obtain *c* **where** *c*: *extreme-bound A* (⊑) {*x*} *c* **by** *auto*
then have *cA*: *c* ∈ *A* **and** *xc*: *x* ⊑ *c* **by** *auto*
from *c xX* **have** *cB*: *c* ∈ *?B* **by** (*auto intro*!: *exI*[*of* - {*x*}] *extreme-boundI*)
with *b* **have** *bA*: *b* ∈ *A* **and** *cb*: *c* ⊑ *b* **by** *auto*
from *xX XA cA bA trans*[*OF xc cb*]
show *x* ⊑ *b* **by** *auto*

Here transitivity is needed.

next
fix *x*
assume *xA*: *x* ∈ *A* **and** *Xx*: *bound X* (⊑) *x*
have *bound ?B* (⊑) *x*
proof (*intro boundI UNIV-I*, *clarify*)
fix *c Y*
assume *finite Y* **and** *YX*: *Y* ⊆ *X* **and** *Y* ≠ {} **and** *c*: *extreme-bound A* (⊑) *Y c*
from *YX Xx* **have** *bound Y* (⊑) *x* **by** *auto*
with *c xA* **show** *c* ⊑ *x* **by** *auto*
qed
with *b xA* **show** *b* ⊑ *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* ∈ *?A*⟧ ⟹ *extreme-bound ?A ?less-eq* {*?x*} *?x*) and so *x* ⊑ *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 (⊑) ⟷ *directed-complete A* (⊑) ∧ *finite-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** *fc*: *finite-complete A* (⊑) **by** *auto*
show *?l*
proof (*intro completeI*)
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* ∈ *X* **by** *auto*
with *XA* **have** *extreme-bound A* (⊑) {*x*} *x*
by (*intro extreme-bound-singleton*, *auto*)
with *xX* **have** *xB*: *x* ∈ *?B* **by** (*auto intro*!: *exI*[*of -* {*x*}])
then have *2*: *?B* ≠ {} **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* ∈ *A* **by** *auto*
show *Ex* (*extreme-bound A* (⊑) *X*)
proof (*intro exI extreme-boundI UNIV-I*)
fix *x*
assume *xX*: *x* ∈ *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** $x \sqsubseteq b$ **by** *auto*
  next
    fix *x*
    assume *Xx*: *bound X* ($\sqsubseteq$) *x* **and** *xA*: $x \in A$
    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 *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

4.6 Relating to Classes

Isabelle's class *complete-lattice* is $\top$−*complete*.

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

4.7 Set-wise Completeness

lemma *Pow-extreme-bound*: $X \subseteq Pow\ A \Longrightarrow$ *extreme-bound* (*Pow A*) ($\subseteq$) *X* ($\bigcup X$)
 by (*intro extreme-boundI*, *auto 2 3*)

lemma *Pow-complete*: $\mathcal{C}$−*complete* (*Pow A*) ($\subseteq$)
 by (*auto intro*!: *completeI dest*: *Pow-extreme-bound*)

lemma *directed-directed-Un*:
assumes *ch*: $XX \subseteq \{X.\ directed\ X\ r\}$ **and** *dir*: *directed XX* ($\subseteq$)
  shows *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

lemmas *directed-connex-Un* = *directed-directed-Un*[*OF* - *connex.directed*]

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 *directed−complete* $\{X \in AA.\ directed\ X\ r\}$ $(\subseteq)$
proof (*intro completeI*)
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*
apply (*rule directed-directed-Un*) **using** *ch* **by** (*auto simp*: *dir*)
ultimately show *Ex* (*extreme-bound* $\{X \in AA.\ directed\ X\ r\}$ $(\subseteq)$ *XX*)
by (*auto intro*!: *exI*[*of* - $\bigcup XX$])
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 $r\ x\ y \lor r\ y\ x$ **by** (*auto elim*:*connexE*)
qed

lemma *connex-is-directed-complete*: *directed−complete* $\{C.\ C \subseteq A \land connex\ C\ r\}$ $(\subseteq)$
proof (*intro completeI*)
fix *CC* **assume** *CC*: $CC \subseteq \{C.\ C \subseteq A \land connex\ C\ r\}$ **and** *directed CC* $(\subseteq)$
with *connex-directed-Un* **have** *Scon*: *connex* $(\bigcup CC)$ *r* **by** *auto*
from *CC* **have** *SA*: $\bigcup CC \subseteq A$ **by** *auto*
from *Scon SA* **show** $\exists S.$ *extreme-bound* $\{C.\ C \subseteq A \land connex\ C\ r\}$ $(\subseteq)$ *CC S*
by (*auto intro*!: *exI*[*of* - $\bigcup CC$] *extreme-boundI*)
qed

lemma (**in** *well-ordered-set*) *well-ordered-set-insert*:
assumes *aA*: *total-ordered-set* (*insert a A*) $(\sqsubseteq)$
shows *well-ordered-set* (*insert a A*) $(\sqsubseteq)$
proof−
interpret *less-eq-asymmetrize*.
interpret *aA*: *total-ordered-set insert a A* $(\sqsubseteq)$ **using** *aA*.
show *?thesis*
proof (*intro well-ordered-set.intro aA.antisymmetric-axioms well-related-setI*)
fix *X* **assume** *XaA*: $X \subseteq$ *insert a A* **and** *X0*: $X \neq \{\}$
show $\exists e.$ *extreme X* $(\sqsupseteq)$ *e*
proof (*cases* $a \in X$)
case *False*
with *XaA* **have** $X \subseteq A$ **by** *auto*
from *nonempty-imp-ex-extreme*[*OF this X0*] **show** *?thesis*.
next

```
      case aX: True
      show ?thesis
      proof (cases X − {a} = {})
        case True
        with aX XaA have Xa: X = {a} by auto
        from aA.refl[of a]
        have a ⊑ a by auto
        then show ?thesis by (auto simp: Xa)
      next
        case False
        from nonempty-imp-ex-extreme[OF - False] XaA
        obtain e where Xae: extreme (X−{a}) (⊒) e by auto
        with Xae XaA have eaA: e ∈ insert a A by auto
        then have e ⊑ a ∨ a ⊑ e by (intro aA.comparable, auto)
        then show ?thesis
        proof (elim disjE)
          assume ea: e ⊑ a
          with Xae show ?thesis by (auto intro!:exI[of - e])
        next
          assume ae: a ⊑ e
          show ?thesis
          proof (intro exI[of - a] extremeI aX)
            fix x assume xX: x ∈ X
            show a ⊑ x
            proof (cases a = x)
              case True with aA.refl[of a] show ?thesis by auto
            next
              case False
              with xX have x ∈ X − {a} by auto
              with Xae have e ⊑ x by auto
              from aA.trans[OF ae this - eaA] xX XaA
              show ?thesis by auto
            qed
          qed
        qed
      qed
    qed
  qed
qed
```

The following should be true in general, but here we use antisymmetry to avoid the axiom of choice.

```
lemma (in antisymmetric) pointwise-connex-complete:
  assumes comp: connex−complete A (⊑)
  shows connex−complete {f. f ‘ X ⊆ A} (pointwise X (⊑))
proof (safe intro!: completeI exI)
  fix F
  assume FXA: F ⊆ {f. f ‘ X ⊆ A} and F: connex F (pointwise X (⊑))
  show extreme-bound {f. f ‘ X ⊆ A} (pointwise X (⊑)) F (λx. The (extreme-bound
```

A (⊑) {f x \|. f ∈ F}))
  proof (*unfold pointwise-extreme-bound*[*OF FXA*], *safe*)
    fix x **assume** *xX*: $x \in X$
    from *FXA xX* **have** *FxA*: {f x \|. f ∈ F} ⊆ A **by** *auto*
    have *Ex* (*extreme-bound* A (⊑) {f x \|. f ∈ F})
     proof (*intro completeD*[*OF comp*] *FxA CollectI connexI*, *elim imageE*, *fold atomize-eq*)
      fix f g **assume** *fF*: $f \in F$ **and** *gF*: $g \in F$
      from *connex.comparable*[*OF F this*] *xX* **show** $f\ x \sqsubseteq g\ x \vee g\ x \sqsubseteq f\ x$ **by** *auto*
    qed
    also note *ex-extreme-bound-iff-the*
    finally
    show *extreme-bound* A (⊑) {f x \|. f ∈ F} (*The* (*extreme-bound* A (⊑) {f x \|. f ∈ F})).
  qed
qed

Our supremum/infimum coincides with those of Isabelle's *complete-lattice*.

lemma *complete-UNIV*: ⊤−*complete* (*UNIV*::′*a*::*complete-lattice set*) (≤)
proof−
  have *Ex* (*supremum* X) **for** X :: ′*a set*
    by (*auto intro*!: *exI*[*of* - ⨆X] *supremumI simp*:*Sup-upper Sup-least*)
  then show *?thesis* **by** (*auto intro*!: *completeI*)
qed

context
  fixes X :: ′*a* :: *complete-lattice set*
begin

lemma *supremum-Sup*: *supremum* X (⨆X)
proof−
  define *it* **where** *it* ≡ *The* (*supremum* X)
  note *completeD*[*OF complete-UNIV*,*simplified*, *of* X]
  from *this*[*unfolded order.dual.ex-extreme-iff-the*]
  have *1*: *supremum* X *it* **by** (*simp add*: *it-def*)
  then have ⨆X = *it* **by** (*intro Sup-eqI*, *auto*)
  with *1* **show** *?thesis* **by** *auto*
qed

lemmas *Sup-eq-The-supremum* = *order.dual.eq-The-extreme*[*OF supremum-Sup*]

lemma *supremum-eq-Sup*: *supremum* X x ⟷ ⨆X = x
  using *order.dual.eq-The-extreme supremum-Sup* **by** *auto*

lemma *infimum-Inf*:
  shows *infimum* X (⨅X)
proof−
  define *it* **where** *it* ≡ *The* (*infimum* X)
  note *completeD*[*OF complete-dual*[*OF complete-UNIV*],*simplified*, *of* X]

from *this*[*unfolded order.ex-extreme-iff-the*]
have *1*: *infimum X it* **by** (*simp add*: *it-def*)
then have $\bigsqcap X = it$ **by** (*intro Inf-eqI*, *auto*)
with *1* **show** *?thesis* **by** *auto*
qed

lemmas *Inf-eq-The-infimum* = *order.eq-The-extreme*[*OF infimum-Inf*]

lemma *infimum-eq-Inf*: *infimum X x* $\longleftrightarrow \bigsqcap X = x$
using *order.eq-The-extreme infimum-Inf* **by** *auto*

end

end

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

5 Existence of Fixed Points in Complete Related Sets

The following proof is simplified and generalized from Stouti–Maaden [22]. 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 [22], relaxing reflexivity and antisymmetry.

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

sublocale *less-eq-asymmetrize*.

definition *AA* **where** $AA \equiv$
$\{X.\ X \subseteq A \land f$ ‘ $X \subseteq X \land (\forall Y\ s.\ Y \subseteq X \longrightarrow$ *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*:
$X \in AA \Longrightarrow$

$(X \subseteq A \Longrightarrow f \text{ ` } X \subseteq X \Longrightarrow (\bigwedge Y\ s.\ Y \subseteq X \Longrightarrow \textit{extreme-bound } A\ (\sqsubseteq)\ Y\ s \Longrightarrow s \in X) \Longrightarrow \textit{thesis}) \Longrightarrow \textit{thesis}$
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 $f \text{ ` } C \subseteq C$ **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*: $f \text{ ` } C \subseteq C$ **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 ‘ $D \subseteq D$
  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 $f\ x \sqsubseteq f\ c$ **using** *mono* **by** (*auto dest*:*monotoneD*)
  qed
  have *DC*: $D \subseteq C$ **unfolding** *D-def* **by** *auto*
  fix B b **assume** *BD*: $B \subseteq D$ **and** *Bb*: *extreme-bound* A ($\sqsubseteq$) B b
  have $B \subseteq C$ **using** *DC BD* **by** *auto*
  then have *bC*: $b \in C$ **using** *C-AA Bb BD* **by** (*auto elim*!: *AA-E*)
  have *bfc*: $\forall a{\in}B.\ a \sqsubseteq f\ c$ **using** *BD* **unfolding** *D-def* **by** *auto*
  with *f cA Bb*
  have $b \sqsubseteq f\ c$ **by** (*auto simp*: *extreme-def image-subset-iff*)
  with *bC* **show** $b \in D$ **unfolding** *D-def* **by** *auto*
 qed
 then have $C \subseteq D$ **unfolding** *C-def* **by** *auto*
 then show $c \sqsubseteq f\ c$ **using** *cC* **unfolding** *D-def* **by** *auto*
 qed
qed

lemma *extreme-qfp*:
 assumes *attract*: $\forall q \in A.\ \forall x \in A.\ f\ q \sim q \longrightarrow x \sqsubseteq f\ q \longrightarrow x \sqsubseteq q$
  and *mono*: *monotone-on* A ($\sqsubseteq$) ($\sqsubseteq$) f
 shows *extreme* $\{q \in A.\ f\ q \sim q \lor f\ q = q\}$ ($\sqsupseteq$) c
proof−
 have *fcc*: $f\ c \sim c$
  apply (*rule qfp-as-extreme-bound*)
  using *mono* **by** (*auto elim*!: *monotone-onE*)
 define L **where** [*simp*]: $L \equiv \{a \in A.\ \forall s \in A.\ (f\ s \sim s \lor f\ s = s) \longrightarrow a \sqsubseteq s\}$
 have $L \in AA$
 proof (*unfold AA-def*, *intro CollectI conjI allI impI*)
  show *XA*: $L \subseteq A$ **by** *auto*
  show f ‘ $L \subseteq L$
  proof *safe*
   fix x **assume** *xL*: $x \in L$
   show $f\ x \in L$
   proof (*unfold L-def*, *safe*)
    have *xA*: $x \in A$ **using** *xL* **by** *auto*
    then show *fxA*: $f\ x \in A$ **using** f **by** *auto*
    { fix s **assume** *sA*: $s \in A$ **and** *sf*: $f\ s \sim s \lor f\ s = s$
     then have $x \sqsubseteq s$ **using** *xL sA sf* **by** *auto*
    then have $f\ x \sqsubseteq f\ s$ **using** *mono fxA sA xA* **by** (*auto elim*!:*monotone-onE*)**}**
    note *fxfs* = *this*
    { fix s **assume** *sA*: $s \in A$ **and** *sf*: $f\ s \sim s$
      then show $f\ x \sqsubseteq s$ **using** *fxfs attract mono sf fxA sA xA* **by** (*auto elim*!:*monotone-onE*)
    }
    { fix s **assume** *sA*: $s \in A$ **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*: *CC C* (⊑)
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*, *OF C*]
by *auto*

lemma *ex-extreme-qfp-fp*:
assumes *comp*: *CC−complete A* (⊑) **and** *C*: *CC C* (⊑)
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*, *OF C*] **by** *auto*

lemma *ex-extreme-qfp*:
assumes *comp*: *CC−complete A* (⊑) **and** *C*: *CC C* (⊑)
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*, *OF 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* ∈ *A* **by** *auto*
from *Qc fcc cA* **show** *?thesis* **by** (*auto intro*!: *exI*[*of* - *c*])
qed

end

context
fixes *less-eq* :: *'a* ⇒ *'a* ⇒ *bool* (**infix** ‹⊑› *50*) **and** *A* :: *'a set* **and** *f*
assumes *f*: *f* ‘ *A* ⊆ *A*
begin

interpretation *less-eq-symmetrize*.
interpretation *fixed-point-proof A* (⊑) *f* **using** *f* **by** *unfold-locales*

theorem *complete-infl-mono-imp-ex-qfp*:
assumes *comp*: ⊤−*complete A* (⊑) **and** *infl-mono*: ∀ *a*∈*A*. *a* ⊑ *f a* ∨ (∀ *b*∈*A*. *b* ⊑ *a* ⟶ *f b* ⊑ *f a*)
shows ∃ *s*∈*A*. *f s* ∼ *s*
apply (*rule ex-qfp*[*OF comp* - *infl-mono*]) **by** *auto*

end

corollary (**in** *antisymmetric*) *complete-infl-mono-imp-ex-fp*:
assumes *comp*: ⊤−*complete A* (⊑) **and** *f*: *f* ‘ *A* ⊆ *A*
and *infl-mono*: ∀ *a*∈*A*. *a* ⊑ *f a* ∨ (∀ *b*∈*A*. *b* ⊑ *a* ⟶ *f b* ⊑ *f a*)
shows ∃ *s* ∈ *A*. *f s* = *s*
proof−
interpret *less-eq-symmetrize*.
from *complete-infl-mono-imp-ex-qfp*[*OF f comp infl-mono*]
obtain *s* **where** *sA*: *s* ∈ *A* **and** *fss*: *f s* ∼ *s* **by** *auto*
from *f sA* **have** *fsA*: *f s* ∈ *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-symmetrize*.

theorem *complete-mono-imp-ex-extreme-qfp*:
assumes *comp*: ⊤−*complete A* (⊑) **and** *f*: *f* ‘ *A* ⊆ *A*
and *mono*: *monotone-on A* (⊑) (⊑) *f*
shows ∃ *s*. *extreme* {*p* ∈ *A*. *f p* ∼ *p*} (⊑) *s*

proof−
interpret *dual*: *fixed-point-proof A* (⊒) **rewrites** *dual.sym* = (∼)
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*: ⊤−*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-symmetrize*.
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

6 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 [11] 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: $x \in X$
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 *empty-derivation*: *derivation* $\{\}$ **by** (*auto simp*: *derivation-def*)

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*)

end

locale *fixed-point-proof2* = *fixed-point-proof* + *antisymmetric* +
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-lim*:
  assumes *P*: *derivation P* **and** *fP*: *f ‘ P* ⊆ *P* **and** *Pp*: *extreme-bound A* (⊑) *P p*
  shows *derivation* (*P* ∪ {*p*})
proof (*cases p* ∈ *P*)
  case *True*
  with *P* **show** *?thesis* **by** (*auto simp*: *insert-absorb*)
next
  case *pP*: *False*
  interpret *P*: *well-ordered-set P* (⊑) **using** *derivation-well-ordered*[*OF P*].
  have *PA*: *P* ⊆ *A* **using** *derivation-A*[*OF P*].
  from *Pp* **have** *pA*: *p* ∈ *A* **by** *auto*
  have *bp*: *bound P* (⊑) *p* **using** *Pp* **by** *auto*
  then have *pp*: *p* ⊑ *p* **using** *Pp* **by** *auto*
  have *1*: *y* ∈ *P* ⟶ {*x*. (*x* = *p* ∨ *x* ∈ *P*) ∧ *x* ⊏ *y*} = {*x* ∈ *P*. *x* ⊏ *y*} **for** *y*
    using *Pp* **by** (*auto dest*!: *extreme-bound-imp-bound*)
  { fix *x* **assume** *xP*: *x* ∈ *P* **and** *px*: *p* ⊑ *x*
    from *xP Pp* **have** *x* ⊑ *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* ∨ *x* ∈ *P*) ∧ *x* ⊏ *p*} = *P* **using** *Pp* **by** (*auto intro*!:
asympartpI)
  have *wr*: *well-ordered-set* (*P* ∪ {*p*}) (⊑)
    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* ∪ {*p*}) **by** (*auto simp*: *derivation-def pA wr*[*simplified*] *1 3*)
qed

lemma *derivation-suc*:
  assumes *P*: *derivation P* **and** *Pp*: *extreme P* (⊑) *p* **shows** *derivation* (*P* ∪ {*f*
p})
proof (*cases f p* ∈ *P*)
  case *True*
  with *P* **show** *?thesis* **by** (*auto simp*: *insert-absorb*)
next
  case *fpP*: *False*
  interpret *P*: *well-ordered-set P* (⊑) **using** *derivation-well-ordered*[*OF P*].
  have *PA*: *P* ⊆ *A* **using** *derivation-A*[*OF P*].
  with *Pp* **have** *pP*: *p* ∈ *P* **and** *pA*: *p* ∈ *A* **by** *auto*
  with *f* **have** *fpA*: *f p* ∈ *A* **by** *auto*
  from *pP* **have** *pp*: *p* ⊑ *p* **by** *auto*
  from *derivation-infl*[*rule-format*, *OF P pP pP pp*] **have** *p* ⊑ *f p*.
  { fix *x* **assume** *xP*: *x* ∈ *P*
    then have *xA*: *x* ∈ *A* **using** *PA* **by** *auto*
    have *xp*: *x* ⊑ *p* **using** *xP Pp* **by** *auto*

```
    from derivation-infl[rule-format, OF P xP pP this]
    have x ⊑ f p.
  }
  note Pfp = this
  then have bfp: bound P (⊑) (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.asym.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 ∈ X and yY: y ∈ Y
  shows (x ⊏ y ∧ x ∈ Y) ∨ x = y ∨ (y ⊏ x ∧ y ∈ X)
proof–
  { fix X Y x y
    assume X: derivation X and Y: derivation Y and xX: x ∈ X and yY: y ∈ Y
    interpret X: well-ordered-set X (⊑) using derivation-well-ordered[OF X].
    interpret X.asympartp: transitive X (⊏) using X.asympartp-transitive.
    interpret Y: well-ordered-set Y (⊑) using derivation-well-ordered[OF Y].
    have XA: X ⊆ A using derivation-A[OF X].
    then have xA: x ∈ A using xX by auto
    with f have fxA: f x ∈ A by auto
    have YA: Y ⊆ A using derivation-A[OF Y].
    then have yA: y ∈ A using yY by auto
    with f have fyA: f y ∈ A by auto
    { fix Z
      assume Z: Z = {z ∈ X. z ⊏ x}
        and fZ: f ‘ Z ⊆ Z
        and Zx: extreme-bound A (⊑) Z x
        and IHx: ∀ z ∈ X. z ⊏ x ⟶ (z ⊏ y ∧ z ∈ Y) ∨ z = y ∨ (y ⊏ z ∧ y ∈ X)
      have (y ⊏ x ∧ y ∈ X) ∨ x ⊑ y
      proof (cases ∃ z ∈ Z. y ⊏ z)
        case True
        then obtain z where zZ: z ∈ Z and yz: y ⊏ z by auto
        from zZ Z have zX: z ∈ X and zx: z ⊏ x by auto
        from IHx[rule-format, OF zX zx] yz have yX: y ∈ X by auto
        from X.asym.trans[OF yz zx yX zX xX] have y ⊏ x.
        with yX show ?thesis by auto
      next
        case False
        have bound Z (⊑) y
        proof
          fix z assume z ∈ Z
          then have zX: z ∈ X and zx: z ⊏ x and nyz: ¬ y ⊏ z using Z False by
auto
          with IHx[rule-format, OF zX zx] X show z ⊑ y by auto
        qed
        with yA Zx have xy: x ⊑ y by auto
        then show ?thesis by auto
      qed
    }
```

```
    note lim-any = this
    { fix z Z
      assume Z: Z = {z ∈ X. z ⊏ x}
        and Zz: extreme Z (⊑) z
        and xfz: x = f z
        and IHx: (z ⊏ y ∧ z ∈ Y) ∨ z = y ∨ (y ⊏ z ∧ y ∈ 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.asym.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 $W = ‹W = \{w \in Y.\ w \sqsubset y\}›$ **and** $Ww = ‹extreme\ W\ (\sqsubseteq)\ w›$ **and** $yfw = ‹y = f\ w›$
then have wY: $w \in Y$ **and** wy: $w \sqsubset y$ **by** *auto*
have *IHw*: $w \sqsubset x \wedge w \in X \vee w = x \vee x \sqsubset w \wedge x \in Y$ **using** *IHy*[*OF wY wy*] **by** *auto*
have $x \sqsubset y \wedge x \in Y \vee y \sqsubseteq x$ **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 $W = ‹W = \{w \in Y.\ w \sqsubset y\}›$ **and** $fW = ‹f\ ‘\ W \subseteq W›$ **and** $Wy = ‹extreme\text{-}bound\ A\ (\sqsubseteq)\ W\ y›$
have $x \sqsubset y \wedge x \in Y \vee y \sqsubseteq x$ **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 $Z = ‹Z = \{z \in X.\ z \sqsubset x\}›$ **and** $fZ = ‹f\ ‘\ Z \subseteq Z›$ **and** $Zx = ‹extreme\text{-}bound\ A\ (\sqsubseteq)\ Z\ x›$
have *1*: $y \sqsubset x \wedge y \in X \vee x \sqsubseteq y$ **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 $W = ‹W = \{w \in Y.\ w \sqsubset y\}›$ **and** $Ww = ‹extreme\ W\ (\sqsubseteq)\ w›$ **and** $yfw = ‹y = f\ w›$
then have wY: $w \in Y$ **and** wy: $w \sqsubset y$ **by** *auto*
have *IHw*: $w \sqsubset x \wedge w \in X \vee w = x \vee x \sqsubset w \wedge x \in Y$ **using** *IHy*[*OF wY wy*] **by** *auto*
have $x \sqsubset y \wedge x \in Y \vee y \sqsubseteq x$ **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 $W = ‹W = \{w \in Y.\ w \sqsubset y\}›$ **and** $fW = ‹f\ ‘\ W \subseteq W›$ **and** $Wy = ‹extreme\text{-}bound\ A\ (\sqsubseteq)\ W\ y›$
have $x \sqsubset y \wedge x \in Y \vee y \sqsubseteq x$ **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

sublocale *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

lemma *pred-unique*:
  assumes *X*: *derivation X* **and** *xX*: *x* ∈ *X*
  shows {*z* ∈ *X*. *z* ⊏ *x*} = {*z*. *derivable z* ∧ *z* ⊏ *x*}
proof
  { **fix** *z* **assume** *z* ∈ *X* **and** *z* ⊏ *x*
    then have *derivable z* ∧ *z* ⊏ *x* **using** *X* **by** (*auto simp*: *derivable-def*)
  }
  then show {*z* ∈ *X*. *z* ⊏ *x*} ⊆ {*z*. *derivable z* ∧ *z* ⊏ *x*} **by** *auto*
  { **fix** *z* **assume** *derivable z* **and** *zx*: *z* ⊏ *x*
    then obtain *Y* **where** *Y*: *derivation Y* **and** *zY*: *z* ∈ *Y* **by** (*auto simp*: *derivable-def*)
    then have *z* ∈ *X* **using** *derivations-cross-compare*[*OF X Y xX zY*] *zx* **by** *auto*
  }
  then show {*z* ∈ *X*. *z* ⊏ *x*} ⊇ {*z*. *derivable z* ∧ *z* ⊏ *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.

context
  fixes *p*
  assumes *p*: *extreme-bound A* (⊑) {*x*. *derivable x*} *p*
begin

lemma *sup-derivable-derivable*: *derivable p*
  using *derivation-lim*[*OF derivation-derivable - p*] *derivable-closed*
  by (*auto intro*: *derivableI*)

private lemmas *sucp* = *sup-derivable-derivable*[*THEN derivable-closed*]

lemma *sup-derivable-prefixed*: *f p* ⊑ *p* **using** *sucp p* **by** *auto*

lemma *sup-derivable-postfixed*: $p \sqsubseteq f\ p$
apply (*rule derivation-infl*[*rule-format*, *OF derivation-derivable*])
using *sup-derivable-derivable* **by** *auto*

lemma *sup-derivable-qfp*: $f\ p \sim p$
using *sup-derivable-prefixed sup-derivable-postfixed* **by** *auto*

lemma *sup-derivable-fp*: $f\ p = p$
using *sup-derivable-derivable sucp*
by (*auto intro*!: *antisym sup-derivable-prefixed sup-derivable-postfixed simp*: *derivable-A*)

end

end

The assumptions are satisfied by monotone functions.

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

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

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

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

interpretation *fixed-point-proof2*
proof
show *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** $xX = \langle x \in X \rangle$ **and** $yX = \langle y \in X \rangle$ **and** $xy = \langle x \sqsubseteq y \rangle$
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 ⊑ y.
      from zX XA have zA: z ∈ A by auto
      from zy monotone-onD[OF mono] zA yA xfz show x ⊑ f y by auto
    next
      case (lim Z)
      note Z = ‹Z = {z ∈ X. z ⊏ x}› and Zx = ‹extreme-bound A (⊑) Z x›
      from f yA have fyA: f y ∈ A by auto
      have bound Z (⊑) (f y)
      proof
        fix z assume zZ: z ∈ Z
        with Z xX have zsx: z ⊏ x and zX: z ∈ X by auto
        then have zx: z ⊑ x by auto
        from X.trans[OF zx xy zX xX yX] have zy: z ⊑ y.
        from IH[OF zX zsx yX] zy show z ⊑ f y by auto
      qed
      with Zx fyA show ?thesis by auto
    qed
  qed
qed
show mono-imp-derivation-f-refl:
  ∀ X x. derivation X ⟶ x ∈ X ⟶ f x ⊑ f x
proof (intro allI impI)
  fix X x
  assume X: derivation X and xX: x ∈ X
  interpret X: well-ordered-set X (⊑) using derivation-well-ordered[OF X].
  note XA = derivation-A[OF X]
  from monotone-onD[OF mono] xX XA show f x ⊑ f x by auto
qed
qed

lemmas mono-imp-fixed-point-proof2 = fixed-point-proof2-axioms

corollary mono-imp-sup-derivable-fp:
  assumes p: extreme-bound A (⊑) {x. derivable x} p
  shows f p = p
  by (simp add: sup-derivable-fp[OF p])

lemma mono-imp-sup-derivable-lfp:
  assumes p: extreme-bound A (⊑) {x. derivable x} p
  shows extreme {q ∈ A. f q = q} (⊒) p
proof (safe intro!: extremeI)
  from p show p ∈ A by auto
  from sup-derivable-fp[OF p]
  show f p = p.
  fix q assume qA: q ∈ A and fqq: f q = q
  have bound {x. derivable x} (⊑) q
  proof (safe intro!: boundI elim!:derivableE)
    fix x X
    assume X: derivation X and xX: x ∈ X
```

```
    from X interpret well-ordered-set X (⊑) by (rule derivation-well-ordered)
    from xX show x ⊑ 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 ⊑ q and zX: z ∈ X by auto
        from monotone-onD[OF mono - qA zq] zX derivation-A[OF X]
        show ?thesis by (auto simp: fqq suc)
      next
        case lim
        with IH have bound {z ∈ X. z ⊏ x} (⊑) q by auto
        with lim qA show ?thesis by auto
      qed
    qed
  qed
  with p qA show p ⊑ q by auto
qed

lemma mono-imp-ex-least-fp:
  assumes comp: well-related-set−complete A (⊑)
  shows ∃ p. extreme {q ∈ A. f q = q} (⊒) p
proof−
  interpret fixed-point-proof using f by unfold-locales
  have ∃ p. extreme-bound A (⊑) {x. derivable x} p
    apply (rule completeD[OF comp])
    using derivable-A derivable.well-related-set-axioms by auto
  then obtain p where p: extreme-bound A (⊑) {x. derivable x} p by auto
  from p mono-imp-sup-derivable-lfp[OF p] sup-derivable-qfp[OF 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-related-set−complete A (⊑)
    and f: f ‘ A ⊆ A and infl: ∀ x ∈ A. ∀ y ∈ A. x ⊑ y ⟶ x ⊑ f y
  shows ∃ p ∈ A. f p = p
proof−
  interpret fixed-point-proof using f by unfold-locales
  interpret fixed-point-proof2
  proof
```

show *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*)
show *drefl*: $\forall X\ x.$ *derivation* $X \longrightarrow x \in X \longrightarrow f\ x \sqsubseteq f\ x$
using *f* **by** (*auto dest*!: *derivation-A*)
qed
have $\exists p.$ *extreme-bound* A ($\sqsubseteq$) $\{x.$ *derivable* $x\}$ p
apply (*rule completeD*[*OF comp*])
using *derivable.well-related-set-axioms derivable-A* **by** *auto*
with *sup-derivable-fp*
show *?thesis* **by** *auto*
qed

Bourbaki-Witt Theorem on posets:

corollary (**in** *partially-ordered-set*) *well-complete-infl-imp-ex-fp*:
assumes *comp*: *well-related-set−complete* A ($\sqsubseteq$)
and *f*: f ‘ $A \subseteq A$ **and** *infl*: $\forall x \in A.\ x \sqsubseteq f\ x$
shows $\exists p \in A.\ f\ p = p$
proof (*intro well-complete-infl′-imp-ex-fp*[*OF comp f*] *ballI impI*)
fix $x\ y$ **assume** *x*: $x \in A$ **and** *y*: $y \in A$ **and** *xy*: $x \sqsubseteq y$
from *y infl* **have** $y \sqsubseteq f\ y$ **by** *auto*
from *trans*[*OF xy this x y*] $f\ y$ **show** $x \sqsubseteq f\ y$ **by** *auto*
qed

7 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* ($\sqsubseteq$)

7.1 Least Quasi-Fixed Points for Attractive Relations.

lemma *attract-mono-imp-least-qfp*:
assumes *attract*: *attractive* A ($\sqsubseteq$)
and *comp*: *well-related-set−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* (‹[-]$_\sim$›) **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-related-set−complete* Q $(\sqsubseteq^s)$
proof (*intro completeI*)
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** $F\ X \in Q$ **by** (*auto simp*: *Q-def ecl-def*)
qed
then have *F*: $F\ `\ Q \subseteq Q$ **by** *auto*
then interpret *Q*: *fixed-point-proof Q* $(\sqsubseteq^s)$ *F* **by** *unfold-locales*
have *monoQ*: *monotone-on Q* $(\sqsubseteq^s)$ $(\sqsubseteq^s)$ *F*
proof (*intro monotone-onI*)
fix *X Y* **assume** *XQ*: $X \in Q$ **and** *YQ*: $Y \in Q$ **and** *XY*: $X \sqsubseteq^s Y$
then obtain *x y* **where** *xX*: $x \in X$ **and** *yY*: $y \in Y$ **using** *Q-def* **by** (*auto simp*: *ecl-def*)
then have *xA*: $x \in A$ **and** *yA*: $y \in A$ **using** *XQ YQ* **by** (*auto simp*: *Q-def ecl-def*)
have $x \sqsubseteq y$ **using** *XY xX yY* **by** *auto*
then have *fxfy*: $f\ x \sqsubseteq f\ y$ **using** *monotone-on-def*[*of A* $(\sqsubseteq)$ $(\sqsubseteq)$ *f*] *xA yA mono* **by** *auto*
have *fxgX*: $f\ x \in F\ X$ **using** *xX F-def* **by** *blast*
have *fygY*: $f\ y \in F\ Y$ **using** *yY F-def* **by** *blast*
show $F\ X \sqsubseteq^s F\ Y$ **using** *RSLE-eq*[*OF XQFXQ*[*OF XQ*] *XQFXQ*[*OF YQ*]

fxgX fygY fxfy].
  qed
  have *QdA*: $\{x.\ Q.derivable\ x\} \subseteq Q$ **using** *Q.derivable-A* **by** *auto*
  interpret *Q*: *fixed-point-proof2 Q* $(\sqsubseteq^s)$ *F*
    using *Q.mono-imp-fixed-point-proof2*[*OF Q.antisymmetric-axioms monoQ*].
  from *Q.mono-imp-ex-least-fp*[*OF Q.antisymmetric-axioms monoQ compQ*]
  obtain *P* **where** *P*: *extreme* $\{q \in Q.\ F\ q = q\}$ $(\sqsubseteq^s)^-$ *P* **by** *auto*
  then have *PQ*: $P \in Q$ **by** (*auto simp*: *extreme-def*)
  from *P* **have** *FPP*: $F\ P = P$ **using** *PQ* **by** *auto*
  with *P* **have** *PP*: $P \sqsubseteq^s P$ **by** *auto*
  from *P* **obtain** *p* **where** *pA*: $p \in A$ **and** *Pp*: $P = [p]_\sim$ **using** *Q-def* **by** *auto*
  then have *pP*: $p \in P$ **by** (*auto simp*: *ecl-def*)
  then have *fpA*: $f\ p \in A$ **using** *pA f* **by** *auto*
  have $f\ p \in F\ P$ **using** *pP F-def fpA* **by** *auto*
  then have $F\ P = [f\ p]_\sim$ **using** *XQx XQFXQ*[*OF PQ*] **by** *auto*
  then have *fp*: $f\ p \sim p \lor f\ p = p$ **using** *pP FPP* **by** (*auto simp*: *ecl-def*)
  have $p \sqsubseteq p$ **using** *PP pP* **by** *auto*
  with *fp* **have** *fpp*: $f\ p \sim p$ **by** *auto*
  have *e*: *extreme* $\{p \in A.\ f\ p \sim p \lor f\ p = p\}$ $(\sqsupseteq)$ *p*
  proof (*intro extremeI CollectI conjI pA fp*, *elim CollectE conjE*)
    fix *q* **assume** *qA*: $q \in A$ **and** *fq*: $f\ q \sim q \lor f\ q = q$
    define *Z* **where** $Z \equiv \{z \in A.\ q \sim z\} \cup \{q\}$
    then have *qZ*: $q \in Z$ **using** *qA* **by** *auto*
    then have *ZQ*: $Z \in Q$ **using** *qA* **by** (*auto simp*: *Z-def Q-def ecl-def*)
    have *fqA*: $f\ q \in A$ **using** *qA f* **by** *auto*
    then have $f\ q \in Z$ **using** *fq* **by** (*auto simp*: *Z-def*)
    then have *1*: $Z = [f\ q]_\sim$ **using** *XQx ZQ* **by** *auto*
    then have $f\ q \in F\ Z$ **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

7.2 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* $(\sqsubseteq) \longrightarrow CC\ C\ (\sqsubseteq)$
    and *extend*: $\forall X\ Y.\ CC\ X\ (\sqsubseteq) \longrightarrow CC\ Y\ (\sqsubseteq) \longrightarrow X \sqsubseteq^s Y \longrightarrow CC\ (X \cup Y)$
$(\sqsubseteq)$
    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 ⊆ {q ∈ A. f q ∼ q} ∪ P and XCC: CC X (⊑)
with P have XA: X ⊆ A by auto
define B where B ≡ {b ∈ A. ∀ a ∈ X. a ⊑ b}
{ fix s a assume sA: s ∈ A and as: ∀ a ∈ X. a ⊑ s and aX: a ∈ X
  then have aA: a ∈ A using XA by auto
 then have fafs: f a ⊑ f s using mono f aX sA as by (auto elim!: monotone-onE)
  have a ⊑ f s
  proof (cases f a = a)
    case True
    then show ?thesis using fafs by auto
  next
    case False
    then have a ∼ f a using P aX Xfix by auto
    also from fafs have f a ⊑ f s by auto
    finally show ?thesis using f aA sA by auto
  qed
}
with f have fBB: f ‘ B ⊆ B unfolding B-def by auto
have BA: B ⊆ A by (auto simp: B-def)
have compB: CC−complete B (⊑)
proof (unfold complete-def, intro allI impI)
  fix Y assume YS: Y ⊆ B and YCC: CC Y (⊑)
  with BA have YA: Y ⊆ A by auto
  define C where C ≡ X∪Y
  then have CA: C ⊆ A using XA YA C-def by auto
  have XY: X ⊑^s Y using B-def YS by auto
  then have CCC: CC C (⊑) using extend XA YA XCC YCC C-def by auto
  then obtain s where s: extreme-bound A (⊑) C s
    using completeD[OF comp CA, OF CCC] by auto
  then have sA: s ∈ A by auto
  show Ex (extreme-bound B (⊑) 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-related-set−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* (⊑) ⟶ *CC C* (⊑)
    and ∀ *X Y*. *CC X* (⊑) ⟶ *CC Y* (⊑) ⟶ *X* ⊑^{s} *Y* ⟶ *CC* (*X* ∪ *Y*) (⊑)
    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* (⊑)
    and *wr-CC*: ∀ *C* ⊆ *A*. *well-related-set C* (⊑) ⟶ *CC C* (⊑)
    and *extend*: ∀ *X Y*. *CC X* (⊑) ⟶ *CC Y* (⊑) ⟶ *X* ⊑^{s} *Y* ⟶ *CC* (*X* ∪ *Y*) (⊑)
    and *mono*: *monotone-on A* (⊑) (⊑) *f*
  shows *CC−complete* {*p* ∈ *A*. *f p* = *p*} (⊑)
proof−
  interpret *antisymmetric* **using** *anti*.
  have ∗: {*q* ∈ *A*. *f q* ∼ *q*} ⊆ {*p* ∈ *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*∈*A*. *f p* = *p*}]
  show *?thesis* **by** (*auto simp*: *subset-Un-eq*)
qed

end

7.3 Instances

7.3.1 Instances under attractivity

context *attractive* **begin**

interpretation *less-eq-symmetrize*.

Full completeness

theorem *mono-imp-qfp-complete*:
assumes *comp*: $\top$−*complete* A ($\sqsubseteq$) **and** f: f ‘ $A \subseteq A$ **and** *mono*: *monotone-on* A ($\sqsubseteq$) ($\sqsubseteq$) f
shows $\top$−*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*: *connex*−*complete* A ($\sqsubseteq$)
and f: f ‘ $A \subseteq A$ **and** *mono*: *monotone-on* A ($\sqsubseteq$) ($\sqsubseteq$) f
shows *connex*−*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*)

Directed completeness

theorem *mono-imp-qfp-directed-complete*:
assumes *comp*: *directed*−*complete* A ($\sqsubseteq$)
and f: f ‘ $A \subseteq A$ **and** *mono*: *monotone-on* A ($\sqsubseteq$) ($\sqsubseteq$) f
shows *directed*−*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 connex.directed*)

Well Completeness

theorem *mono-imp-qfp-well-complete*:
assumes *comp*: *well-related-set*−*complete* A ($\sqsubseteq$)
and f: f ‘ $A \subseteq A$ **and** *mono*: *monotone-on* A ($\sqsubseteq$) ($\sqsubseteq$) f
shows *well-related-set*−*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 *well-related-extend*)

end

7.3.2 Usual instances under antisymmetry

context *antisymmetric* **begin**

Knaster–Tarski

theorem *mono-imp-fp-complete*:
  assumes *comp*: $\top$−*complete* *A* ($\sqsubseteq$)
    and *f*: *f* ‘ *A* $\subseteq$ *A* **and** *mono*: *monotone-on* *A* ($\sqsubseteq$) ($\sqsubseteq$) *f*
  shows $\top$−*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*
qed

Markowsky 1976

theorem *mono-imp-fp-connex-complete*:
  assumes *comp*: *connex*−*complete* *A* ($\sqsubseteq$)
    and *f*: *f* ‘ *A* $\subseteq$ *A* **and** *mono*: *monotone-on* *A* ($\sqsubseteq$) ($\sqsubseteq$) *f*
  shows *connex*−*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 antisymmetric-axioms mono comp*)
    by (*auto intro*: *connex-union well-related-set.connex*)
qed

Pataraia

theorem *mono-imp-fp-directed-complete*:
  assumes *comp*: *directed*−*complete* *A* ($\sqsubseteq$)
    and *f*: *f* ‘ *A* $\subseteq$ *A* **and** *mono*: *monotone-on* *A* ($\sqsubseteq$) ($\sqsubseteq$) *f*
  shows *directed*−*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*: *directed-extend connex.directed well-related-set.connex*)
qed

Bhatta & George 2011

theorem *mono-imp-fp-well-complete*:
  assumes *comp*: *well-related-set*−*complete* *A* ($\sqsubseteq$)
    and *f*: *f* ‘ *A* $\subseteq$ *A* **and** *mono*: *monotone-on* *A* ($\sqsubseteq$) ($\sqsubseteq$) *f*
  shows *well-related-set*−*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 *Continuity*
  imports *Complete-Relations*
begin

7.4 Scott Continuity, ω-Continuity

In this Section, we formalize 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.

definition *continuous* (‹-−*continuous*› [*1000*]*1000*) **where**
  $\mathcal{C}$−*continuous* A ($\sqsubseteq$) B ($\trianglelefteq$) f ≡
   f ‘ $A \subseteq B$ ∧
    (∀ X s. $\mathcal{C}$ X ($\sqsubseteq$) ⟶ $X \neq \{\}$ ⟶ $X \subseteq A$ ⟶ *extreme-bound* A ($\sqsubseteq$) X s ⟶ *extreme-bound* B ($\trianglelefteq$) (f‘X) (f s))
  for *leA* (**infix** ‹$\sqsubseteq$› *50*) **and** *leB* (**infix** ‹$\trianglelefteq$› *50*)

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

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

lemma
  fixes *prec-eq* (**infix** ‹$\preceq$› *50*) **and** *less-eq* (**infix** ‹$\sqsubseteq$› *50*)
  assumes $\mathcal{C}$−*continuous* I ($\preceq$) A ($\sqsubseteq$) f
  shows *continuous-carrierD*[*dest*]: f ‘ $I \subseteq A$
    and *continuousD*: $\mathcal{C}$ X ($\preceq$) ⟹ $X \neq \{\}$ ⟹ $X \subseteq I$ ⟹ *extreme-bound* I ($\preceq$) X b ⟹ *extreme-bound* A ($\sqsubseteq$) (f ‘ X) (f b)
  using *assms* **by** (*auto elim*!: *continuousE*)

lemma *continuous-comp*:
  fixes *leA* (**infix** ‹$\sqsubseteq_A$› *50*) **and** *leB* (**infix** ‹$\sqsubseteq_B$› *50*) **and** *leC* (**infix** ‹$\sqsubseteq_C$› *50*)
  assumes *KfL*: ∀ $X \subseteq A$. $\mathcal{K}$ X ($\sqsubseteq_A$) ⟶ $\mathcal{L}$ (f ‘ X) ($\sqsubseteq_B$)
  assumes f: $\mathcal{K}$−*continuous* A ($\sqsubseteq_A$) B ($\sqsubseteq_B$) f **and** g: $\mathcal{L}$−*continuous* B ($\sqsubseteq_B$) C ($\sqsubseteq_C$) g
  shows $\mathcal{K}$−*continuous* A ($\sqsubseteq_A$) C ($\sqsubseteq_C$) ($g \circ f$)
  apply (*intro continuousI*)
proof −
  from f g **have** *fAB*: f ‘ $A \subseteq B$ **and** *gBC*: g ‘ $B \subseteq C$ **by** *auto*
  then show ($g \circ f$) ‘ $A \subseteq C$ **by** *auto*
  fix X s
  assume *XA*: $X \subseteq A$ **and** *X0*: $X \neq \{\}$ **and** *XK*: $\mathcal{K}$ X ($\sqsubseteq_A$) **and** *Xs*: *extreme-bound* A ($\sqsubseteq_A$) X s
  from *fAB XA* **have** *fXB*: f ‘ $X \subseteq B$ **by** *auto*
  from *X0* **have** *fX0*: f ‘ $X \neq \{\}$ **by** *auto*
  from *KfL XA XK* **have** *fXL*: $\mathcal{L}$ (f ‘ X) ($\sqsubseteq_B$) **by** *auto*

from *continuousD*[*OF f XK X0 XA Xs*]
have *extreme-bound B* ($\sqsubseteq_B$) (*f* ‘ *X*) (*f s*)**.**
from *continuousD*[*OF g fXL fX0 fXB this*]
show *extreme-bound C* ($\sqsubseteq_C$) ((*g*∘*f*)‘*X*) ((*g*∘*f*) *s*) **by** (*auto simp*: *image-comp*)
qed

lemma *continuous-comp-top*:
fixes *leA* (**infix** ‹$\sqsubseteq_A$› *50*) **and** *leB* (**infix** ‹$\sqsubseteq_B$› *50*) **and** *leC* (**infix** ‹$\sqsubseteq_C$› *50*)
assumes *f*: $\mathcal{K}$−*continuous A* ($\sqsubseteq_A$) *B* ($\sqsubseteq_B$) *f* **and** *g*: ⊤−*continuous B* ($\sqsubseteq_B$) *C* ($\sqsubseteq_C$) *g*
shows $\mathcal{K}$−*continuous A* ($\sqsubseteq_A$) *C* ($\sqsubseteq_C$) (*g* ∘ *f*)
by (*rule continuous-comp*[*OF - f g*], *auto*)

lemma *id-continuous*:
fixes *leA* (**infix** ‹$\sqsubseteq_A$› *50*)
shows $\mathcal{K}$−*continuous A* ($\sqsubseteq_A$) *A* ($\sqsubseteq_A$) (λ*x*. *x*)
by (*auto intro*: *continuousI*)

lemma *cst-continuous*:
fixes *leA* (**infix** ‹$\sqsubseteq_A$› *50*) **and** *leB* (**infix** ‹$\sqsubseteq_B$› *50*)
assumes *b* ∈ *B* **and** *bb*: *b* $\sqsubseteq_B$ *b*
shows $\mathcal{K}$−*continuous A* ($\sqsubseteq_A$) *B* ($\sqsubseteq_B$) (λ*x*. *b*)
apply (*intro continuousI*)
using *assms*(*1*) **apply** *auto*
using *assms extreme-bound-singleton-refl*[*of B* ($\sqsubseteq_B$) *b*] **by** *blast*

lemma *continuous-cmono*:
assumes *CD*: $\mathcal{C} \le \mathcal{D}$ **shows** $\mathcal{D}$−*continuous* ≤ $\mathcal{C}$−*continuous*
proof (*safe intro*!: *le-funI le-boolI*)
fix *I A f* **and** *prec-eq* (**infix** ‹⪯› *50*) **and** *less-eq* (**infix** ‹⊑› *50*)
assume *cont*: $\mathcal{D}$−*continuous I* (⪯) *A* (⊑) *f*
show $\mathcal{C}$−*continuous I* (⪯) *A* (⊑) *f*
proof (*rule continuousI*)
from *cont* **show** *f* ‘ *I* ⊆ *A* **by** *auto*
fix *X s* **assume** *X*: $\mathcal{C}$ *X* (⪯) **and** *X0*: *X* ≠ {} **and** *XI*: *X* ⊆ *I* **and** *Xs*: *extreme-bound I* (⪯) *X s*
from *CD X* **have** $\mathcal{D}$ *X* (⪯) **by** *auto*
from *continuousD*[*OF cont*, *OF this X0 XI Xs*]
show *extreme-bound A* (⊑) (*f* ‘ *X*) (*f s*)**.**
qed
qed

context
fixes *prec-eq* :: ′*i* ⇒ ′*i* ⇒ *bool* (**infix** ‹⪯› *50*) **and** *less-eq* :: ′*a* ⇒ ′*a* ⇒ *bool* (**infix** ‹⊑› *50*)
begin

lemma *continuous-subclass*:

assumes *CD*: ∀ *X*⊆*I*. *X* ≠ {} ⟶ 𝒞 *X* (≼) ⟶ 𝒟 *X* (≼) **and** *cont*: 𝒟−*continuous* *I* (≼) *A* (⊑) *f*
shows 𝒞−*continuous* *I* (≼) *A* (⊑) *f*
using *cont* **by** (*auto simp*: *continuous-def CD*[*rule-format*])

lemma *chain-continuous-imp-well-continuous*:
assumes *cont*: *connex*−*continuous* *I* (≼) *A* (⊑) *f*
shows *well-related-set*−*continuous* *I* (≼) *A* (⊑) *f*
by (*rule continuous-subclass*[*OF* - *cont*], *auto simp*: *well-related-set.connex*)

lemma *well-continuous-imp-omega-continous*:
assumes *cont*: *well-related-set*−*continuous* *I* (≼) *A* (⊑) *f*
shows *omega-chain*−*continuous* *I* (≼) *A* (⊑) *f*
by (*rule continuous-subclass*[*OF* - *cont*], *auto simp*: *omega-chain-imp-well-related*)

end

abbreviation *scott-continuous* *I* (≼) ≡ *directed-set*−*continuous* *I* (≼)
for *prec-eq* (**infix** ‹≼› *50*)

lemma *scott-continuous-imp-well-continuous*:
fixes *prec-eq* :: *'i* ⇒ *'i* ⇒ *bool* (**infix** ‹≼› *50*) **and** *less-eq* :: *'a* ⇒ *'a* ⇒ *bool* (**infix** ‹⊑› *50*)
assumes *cont*: *scott-continuous* *I* (≼) *A* (⊑) *f*
shows *well-related-set*−*continuous* *I* (≼) *A* (⊑) *f*
by (*rule continuous-subclass*[*OF* - *cont*], *auto simp*: *well-related-set.directed-set*)

lemmas *scott-continuous-imp-omega-continuous* =
scott-continuous-imp-well-continuous[*THEN well-continuous-imp-omega-continous*]

7.4.1 Continuity implies monotonicity

lemma *continuous-imp-mono-refl*:
fixes *prec-eq* (**infix** ‹≼› *50*) **and** *less-eq* (**infix** ‹⊑› *50*)
assumes *cont*: 𝒞−*continuous* *I* (≼) *A* (⊑) *f* **and** *xyC*: 𝒞 {*x*,*y*} (≼)
and *xy*: *x* ≼ *y* **and** *yy*: *y* ≼ *y*
and *x*: *x* ∈ *I* **and** *y*: *y* ∈ *I*
shows *f x* ⊑ *f y*
proof−
have *fboy*: *extreme-bound* *A* (⊑) (*f* ‘ {*x*,*y*}) (*f y*)
proof (*intro continuousD*[*OF cont*] *xyC*)
from *x y* **show** *CI*: {*x*,*y*} ⊆ *I* **by** *auto*
show *Cy*: *extreme-bound* *I* (≼) {*x*,*y*} *y* **using** *xy yy x y* **by** *auto*
qed *auto*
then show *?thesis* **by** *auto*
qed

lemma *omega-continuous-imp-mono-refl*:
fixes *prec-eq* (**infix** ‹≼› *50*) **and** *less-eq* (**infix** ‹⊑› *50*)

assumes *cont*: *omega-chain−continuous I* ($\preceq$) *A* ($\sqsubseteq$) *f*
and *xx*: $x \preceq x$ **and** *xy*: $x \preceq y$ **and** *yy*: $y \preceq y$
and *x*: $x \in I$ **and** *y*: $y \in I$
shows $f\ x \sqsubseteq f\ y$
apply (*rule continuous-imp-mono-refl*[*OF cont*, *OF pair-omega-chain*])
using *assms* **by** *auto*

context *reflexive* **begin**

lemma *continuous-imp-monotone-on*:
fixes *leB* (**infix** ‹$\trianglelefteq$› *50*)
assumes *cont*: $\mathcal{C}$*−continuous A* ($\sqsubseteq$) *B* ($\trianglelefteq$) *f*
and *II*: $\forall\, i \in A.\ \forall\ j \in A.\ i \sqsubseteq j \longrightarrow \mathcal{C}\ \{i,j\}$ ($\sqsubseteq$)
shows *monotone-on A* ($\sqsubseteq$) ($\trianglelefteq$) *f*
proof−
show *?thesis*
apply (*intro monotone-onI continuous-imp-mono-refl*[*OF cont*])
using *II* **by** *auto*
qed

lemma *well-complete-imp-monotone-on*:
fixes *leB* (**infix** ‹$\trianglelefteq$› *50*)
assumes *cont*: *well-related-set−continuous A* ($\sqsubseteq$) *B* ($\trianglelefteq$) *f*
shows *monotone-on A* ($\sqsubseteq$) ($\trianglelefteq$) *f*
by (*auto intro*!: *continuous-imp-monotone-on cont pair-well-related*)

end

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

8 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 [17] 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.

8.1 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 monotone-on-funpow: assumes f: f ‘ A ⊆ A and mono: monotone-on A
r r f
  shows monotone-on A r r (f^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 (‹⊥›)

context
  fixes A and less-eq (infix ‹⊑› 50) and bot (‹⊥›) and f
  assumes bot: ⊥ ∈ A ∀ q ∈ A. ⊥ ⊑ q
  assumes cont: omega-chain−continuous A (⊑) A (⊑) f
begin

interpretation less-eq-symmetrize.

private lemma f: f ‘ A ⊆ A using cont by auto

private abbreviation(input) Fn ≡ {f^n ⊥ |. n :: nat}

private lemma fn-ref: f^n ⊥ ⊑ f^n ⊥ and fnA: f^n ⊥ ∈ A
proof (atomize(full), induct n)
  case 0
  from bot show ?case by simp
next
  case (Suc n)
  then have fn: f^n ⊥ ∈ A and fnfn: f^n ⊥ ⊑ f^n ⊥ by auto
  from f fn omega-continuous-imp-mono-refl[OF cont fnfn fnfn fnfn]
  show ?case by auto
qed

private lemma FnA: Fn ⊆ A using fnA by auto

private lemma Fn-chain: omega-chain Fn (⊑)
proof (intro omega-chainI)
```

```
  show fn-monotone: monotone (≤) (⊑) (λn. f^n ⊥)
  proof
    fix n m :: nat
    assume n ≤ 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^n ⊥ ⊑ f^m ⊥ by (unfold m, induct n, auto)
  qed
qed auto

private lemma Fn: Fn = range (λn. f^n ⊥) by auto

theorem kleene-qfp:
  assumes q: extreme-bound A (⊑) Fn q
  shows f q ∼ q
proof
  have fq: extreme-bound A (⊑) (f ‘ Fn) (f q)
    apply (rule continuousD[OF cont - - FnA q])
    using Fn-chain by auto
  with bot have nq: f^n ⊥ ⊑ f q for n by (induct n, auto simp: extreme-bound-iff)
  then show q ⊑ f q using f q by blast
  have f (f^n ⊥) ∈ Fn for n by (auto intro!: range-eqI[of - - Suc n])
  then have f ‘ Fn ⊆ Fn by auto
  from extreme-bound-subset[OF this fq q]
  show f q ⊑ q.
qed

lemma ex-kleene-qfp:
  assumes comp: omega-chain−complete A (⊑)
  shows ∃p. extreme-bound A (⊑) Fn p
  apply (intro comp[THEN completeD, OF FnA])
  using Fn-chain
  by auto
```

8.2 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: ∀q ∈ A. ∀x ∈ A. f q ∼ q ⟶ x ⊑ f q ⟶ x ⊑ q
  assumes q: extreme-bound A (⊑) Fn q
  shows extreme {s ∈ A. f s ∼ s} (⊒) q
proof(safe intro!: extremeI kleene-qfp[OF q])
  from q
  show q ∈ A by auto
  fix c assume c: c ∈ A and cqfp: f c ∼ c
  {
    fix n::nat
    have f^n ⊥ ⊑ c
```

```
    proof(induct n)
      case 0
      show ?case using bot c by auto
    next
      case IH: (Suc n)
      have c ⊑ c using attract cqfp c by auto
      with IH have f^(Suc n) ⊥ ⊑ 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 ⊑ c using q c by auto
qed

lemma kleene-qfp-iff-least:
  assumes comp: omega-chain−complete A (⊑)
  assumes attract: ∀ q ∈ A. ∀ x ∈ A. f q ∼ q ⟶ x ⊑ f q ⟶ x ⊑ q
  assumes dual-attract: ∀ p ∈ A. ∀ q ∈ A. ∀ x ∈ A. p ∼ q ⟶ q ⊑ x ⟶ p ⊑ x
  shows extreme-bound A (⊑) Fn = extreme {s ∈ A. f s ∼ s} (⊒)
proof (intro ext iffI kleene-qfp-is-least[OF attract])
  fix q
  assume q: extreme {s ∈ A. f s ∼ s} (⊒) q
  from q have qA: q ∈ A by auto
  from q have qq: q ⊑ 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* $(\sqsubseteq)$ x **and** x: $x \in A$
with k **have** *kx*: $k \sqsubseteq x$ **by** *auto*
with *dual-attract*[*rule-format*, *OF* - - x *qk*] q k
show $q \sqsubseteq x$ **by** *auto*
next
from q **show** $q \in A$ **by** *auto*
qed
qed

end

context *attractive* **begin**

interpretation *less-eq-dualize* + *less-eq-symmetrize*.

theorem *kleene-qfp-is-dual-extreme*:
assumes *comp*: *omega-chain−complete* A $(\sqsubseteq)$
and *cont*: *omega-chain−continuous* A $(\sqsubseteq)$ A $(\sqsubseteq)$ f **and** *bA*: $b \in A$ **and** *bot*: $\forall x \in A.\ b \sqsubseteq x$
shows *extreme-bound* A $(\sqsubseteq)$ $\{f\hat{\ }n\ b\ |.\ n :: nat\}$ = *extreme* $\{s \in A.\ f\ s \sim s\}$ $(\sqsupseteq)$
apply (*rule kleene-qfp-iff-least*[*OF bA bot cont comp*])
using *continuous-carrierD*[*OF cont*]
by (*auto dest*: *sym-order-trans order-sym-trans*)

end

corollary(**in** *antisymmetric*) *kleene-fp*:
assumes *cont*: *omega-chain−continuous* A $(\sqsubseteq)$ A $(\sqsubseteq)$ f
and b: $b \in A$ $\forall x \in A.\ b \sqsubseteq x$
and p: *extreme-bound* A $(\sqsubseteq)$ $\{f\hat{\ }n\ b\ |.\ n :: nat\}$ p
shows $f\ p = p$
using *kleene-qfp*[*OF b cont*] p *cont*[*THEN continuous-carrierD*]
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, *Proceedings of the 5th International Conference on Mathematical Knowledge Man-*

agement (MKM 2006), volume 4108 of *LNCS*, pages 31–43. Springer Berlin Heidelberg, 2006.

[3] G. M. Bergman. *An Invitation to General Algebra and Universal Constructions.* 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] S. P. Bhatta and S. George. Some fixed point theorems for pseudo ordered sets. *Algebra and Discrete Mathematics*, 11(1):17–22, 2011.

[6] N. Bourbaki. Sur le théorème de zorn. *Archiv der Mathematik*, 2(6):434–437, 1949.

[7] P. Cousot and R. Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Proceedings of the 4th ACM Symposium on Principles of Programming Languages (POPL'77)*, pages 238–252. ACM Press, 1977.

[8] J. Dubut and A. Yamada. Fixed Points Theorems for Non-Transitive Relations. *Logical Methods in Computer Science*, 18(1), 2021.

[9] 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.

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

[11] H. Grall. Proving fixed points. In *Fixed Points in Computer Science 2010*, pages 41–46, 2010.

[12] 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:e2, 2017.

[13] F. Kammüller. Modular reasoning in Isabelle. In D. McAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction (CADE-17)*, volume 1831 of *LNCS*, pages 99–114. Springer Berlin Heidelberg, 2000.

[14] L. Kantorovitch. The method of successive approximations for functional equations. *Acta Math.*, 71:63–97, 1939.

[15] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Wiwood. seL4: Formal verification of an OS kernel. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP 2009)*, pages 207–220. ACM, 2009.

[16] G. Markowsky. Chain-complete posets and directed sets with applications. *Algebra Universalis*, 6:53–68, 1976.

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

[18] D. Pataraia. A constructive proof of Tarski's fixed-point theorem for dcpo's. Presented in the 65th Peripatetic Seminar on Sheaves and Logic, in Aarhus, Denmark, 1997.

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

[20] D. Scott and C. Strachey. Toward a mathematical semantics for computer languages. Technical Monograph PRG-6, Oxford Programming Research Group, 1971.

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

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

[23] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285–309, 1955.

[24] A. Yamada and J. Dubut. Complete Non-Orders and Fixed Points. In *Proceedings of the 10th International Conference on Interactive Theorem Proving (ITP 2019)*, volume 141 of *Leibniz International Proceedings in Informatics*, pages 30:1–30:16. Leibniz-Zentrum für Informatik, 2019.

[25] A. Yamada and J. Dubut. Formalizing Results on Directed Sets in Isabelle/HOL. In *Proceedings of the fourteenth conference on Interactive Theorem Proving (ITP'23)*, 2023.