

Open Induction

Mizuhito Ogawa Christian Sternagel*

September 13, 2023

Abstract

A proof of the open induction schema based on [1].

Contents

1	Binary Predicates Restricted to Elements of a Given Set	1
1.1	Measures on Sets (Instead of Full Types)	7
1.2	Facts About Predecessor Sets	13
2	Open Induction	14
2.1	(Greatest) Lower Bounds and Chains	14
2.2	Open Properties	14
2.3	Downward Completeness	15
2.4	The Open Induction Principle	16
2.5	Open Induction on Universal Domains	17
2.6	Type Class of Downward Complete Orders	17

1 Binary Predicates Restricted to Elements of a Given Set

theory *Restricted-Predicates*

imports *Main*

begin

A subset C of A is a *chain* on A (w.r.t. P) iff for all pairs of elements of C , one is less than or equal to the other one.

abbreviation $\text{chain-on } P \ C \ A \equiv \text{pred-on.chain } A \ P \ C$

lemmas $\text{chain-on-def} = \text{pred-on.chain-def}$

lemma *chain-on-subset*:

$A \subseteq B \implies \text{chain-on } P \ C \ A \implies \text{chain-on } P \ C \ B$

*The research was partly funded by the Austrian Science Fund (FWF): J3202.

by (*force simp: chain-on-def*)

lemma *chain-on-imp-subset*:
chain-on P C A $\implies C \subseteq A$
by (*simp add: chain-on-def*)

lemma *subchain-on*:
assumes $C \subseteq D$ **and** *chain-on P D A*
shows *chain-on P C A*
using *assms* **by** (*auto simp: chain-on-def*)

definition *restrict-to* :: ($'a \Rightarrow 'a \Rightarrow \text{bool}$) $\Rightarrow 'a \text{ set} \Rightarrow ('a \Rightarrow 'a \Rightarrow \text{bool})$ **where**
restrict-to P A = $(\lambda x y. x \in A \wedge y \in A \wedge P x y)$

abbreviation *strict P* $\equiv \lambda x y. P x y \wedge \neg (P y x)$

abbreviation *incomparable P* $\equiv \lambda x y. \neg P x y \wedge \neg P y x$

abbreviation *antichain-on P f A* $\equiv \forall (i::\text{nat}) j. f i \in A \wedge (i < j \longrightarrow \text{incomparable } P (f i) (f j))$

lemma *strict-reflclp-conv* [*simp*]:
strict (P⁼⁼) = *strict P* **by** *auto*

lemma *reflp-on-reflclp-simp* [*simp*]:
assumes *reflp-on A P* **and** $a \in A$ **and** $b \in A$
shows $P^{==} a b = P a b$
using *assms* **by** (*auto simp: reflp-on-def*)

lemmas *reflp-on-converse-simp* = *reflp-on-conversp*
lemmas *irreflp-on-converse-simp* = *irreflp-on-converse*
lemmas *transp-on-converse-simp* = *transp-on-conversep*

lemma *transp-on-strict*:
transp-on A P $\implies \text{transp-on } A (\text{strict } P)$
unfolding *transp-on-def* **by** *blast*

definition *wfp-on* :: ($'a \Rightarrow 'a \Rightarrow \text{bool}$) $\Rightarrow 'a \text{ set} \Rightarrow \text{bool}$
where
wfp-on P A $\longleftrightarrow \neg (\exists f. \forall i. f i \in A \wedge P (f (Suc i)) (f i))$

definition *inductive-on* :: ($'a \Rightarrow 'a \Rightarrow \text{bool}$) $\Rightarrow 'a \text{ set} \Rightarrow \text{bool}$ **where**
inductive-on P A $\longleftrightarrow (\forall Q. (\forall y \in A. (\forall x \in A. P x y \longrightarrow Q x) \longrightarrow Q y) \longrightarrow (\forall x \in A. Q x))$

lemma *inductive-onI* [*Pure.intro*]:
assumes $\bigwedge Q x. \llbracket x \in A; (\bigwedge y. \llbracket y \in A; \bigwedge x. \llbracket x \in A; P x y \rrbracket \implies Q x \rrbracket \implies Q y \rrbracket$
 $\implies Q x$
shows *inductive-on P A*

using *assms* **unfolding** *inductive-on-def* **by** *metis*

If P is well-founded on A then every non-empty subset Q of A has a minimal element z w.r.t. P , i.e., all elements that are P -smaller than z are not in Q .

lemma *wfp-on-imp-minimal*:

assumes *wfp-on* P A

shows $\forall Q$ $x. x \in Q \wedge Q \subseteq A \longrightarrow (\exists z \in Q. \forall y. P y z \longrightarrow y \notin Q)$

proof (*rule ccontr*)

assume \neg *?thesis*

then obtain Q x **where** $*$: $x \in Q$ $Q \subseteq A$

and $\forall z. \exists y. z \in Q \longrightarrow P y z \wedge y \in Q$ **by** *metis*

from *choice* [*OF this(3)*] **obtain** f

where $**$: $\forall x \in Q. P (f x) x \wedge f x \in Q$ **by** *blast*

let $?S = \lambda i. (f \smallfrown i) x$

have $***$: $\forall i. ?S i \in Q$

proof

fix i **show** $?S i \in Q$ **by** (*induct i*) (*auto simp: * ***)

qed

then have $\forall i. ?S i \in A$ **using** $*$ **by** *blast*

moreover have $\forall i. P (?S (Suc i)) (?S i)$

proof

fix i **show** $P (?S (Suc i)) (?S i)$

by (*induct i*) (*auto simp: * ** ****)

qed

ultimately have $\forall i. ?S i \in A \wedge P (?S (Suc i)) (?S i)$ **by** *blast*

with *assms(1)* **show** *False*

unfolding *wfp-on-def* **by** *fast*

qed

lemma *minimal-imp-inductive-on*:

assumes $\forall Q$ $x. x \in Q \wedge Q \subseteq A \longrightarrow (\exists z \in Q. \forall y. P y z \longrightarrow y \notin Q)$

shows *inductive-on* P A

proof (*rule ccontr*)

assume \neg *?thesis*

then obtain Q x

where $*$: $\forall y \in A. (\forall x \in A. P x y \longrightarrow Q x) \longrightarrow Q y$

and $**$: $x \in A \wedge \neg Q x$

by (*auto simp: inductive-on-def*)

let $?Q = \{x \in A. \neg Q x\}$

from $**$ **have** $x \in ?Q$ **by** *auto*

moreover have $?Q \subseteq A$ **by** *auto*

ultimately obtain z **where** $z \in ?Q$

and min : $\forall y. P y z \longrightarrow y \notin ?Q$

using *assms* [*THEN spec [of - ?Q]*, *THEN spec [of - x]*] **by** *blast*

from $\langle z \in ?Q \rangle$ **have** $z \in A$ **and** $\neg Q z$ **by** *auto*

with $*$ **obtain** y **where** $y \in A$ **and** $P y z$ **and** $\neg Q y$ **by** *auto*

then have $y \in ?Q$ **by** *auto*

with $\langle P y z \rangle$ **and** min **show** *False* **by** *auto*

qed

lemmas *wfp-on-imp-inductive-on* =
wfp-on-imp-minimal [THEN *minimal-imp-inductive-on*]

lemma *inductive-on-induct* [*consumes 2, case-names less, induct pred: inductive-on*]:
assumes *inductive-on P A* **and** $x \in A$
and $\bigwedge y. \llbracket y \in A; \bigwedge x. \llbracket x \in A; P x y \rrbracket \implies Q x \rrbracket \implies Q y$
shows $Q x$
using *assms unfolding inductive-on-def by metis*

lemma *inductive-on-imp-wfp-on*:
assumes *inductive-on P A*
shows *wfp-on P A*
proof –
let $?Q = \lambda x. \neg (\exists f. f 0 = x \wedge (\forall i. f i \in A \wedge P (f (Suc i)) (f i)))$
{ **fix** x **assume** $x \in A$
with *assms* **have** $?Q x$
proof (*induct rule: inductive-on-induct*)
fix y **assume** $y \in A$ **and** *IH*: $\bigwedge x. x \in A \implies P x y \implies ?Q x$
show $?Q y$
proof (*rule ccontr*)
assume $\neg ?Q y$
then obtain f **where** $*$: $f 0 = y$
 $\forall i. f i \in A \wedge P (f (Suc i)) (f i)$ **by** *auto*
then have $P (f (Suc 0)) (f 0)$ **and** $f (Suc 0) \in A$ **by** *auto*
with *IH* **and** $*$ **have** $?Q (f (Suc 0))$ **by** *auto*
with $*$ **show** *False* **by** *auto*
qed
qed }
then show *?thesis* **unfolding** *wfp-on-def* **by** *blast*
qed

definition *qo-on* :: $('a \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow 'a \text{ set} \Rightarrow \text{bool}$ **where**
qo-on P A \longleftrightarrow *reflp-on A P* \wedge *transp-on A P*

definition *po-on* :: $('a \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow 'a \text{ set} \Rightarrow \text{bool}$ **where**
po-on P A \longleftrightarrow (*irreflp-on A P* \wedge *transp-on A P*)

lemma *po-onI* [*Pure.intro*]:
 $\llbracket \text{irreflp-on } A P; \text{transp-on } A P \rrbracket \implies \text{po-on } P A$
by (*auto simp: po-on-def*)

lemma *po-on-converse-simp* [*simp*]:
 $\text{po-on } P^{-1-1} A \longleftrightarrow \text{po-on } P A$
by (*simp add: po-on-def*)

lemma *po-on-imp-qo-on*:
 $\text{po-on } P A \implies \text{qo-on } (P^{==}) A$
unfolding *po-on-def qo-on-def*

by (metis reflp-on-reflclp transp-on-reflclp)

lemma *po-on-imp-irreflp-on*:
po-on $P A \implies$ irreflp-on $A P$
by (auto simp: po-on-def)

lemma *po-on-imp-transp-on*:
po-on $P A \implies$ transp-on $A P$
by (auto simp: po-on-def)

lemma *po-on-subset*:
assumes $A \subseteq B$ and po-on $P B$
shows po-on $P A$
using transp-on-subset and irreflp-on-subset and assms
unfolding po-on-def by blast

lemma *transp-on-irreflp-on-imp-antisymp-on*:
assumes transp-on $A P$ and irreflp-on $A P$
shows antisymp-on $A (P^{==})$
proof (rule antisymp-onI)
fix $a b$ assume $a \in A$
and $b \in A$ and $P^{==} a b$ and $P^{==} b a$
show $a = b$
proof (rule ccontr)
assume $a \neq b$
with $\langle P^{==} a b \rangle$ and $\langle P^{==} b a \rangle$ have $P a b$ and $P b a$ by auto
with $\langle \text{transp-on } A P \rangle$ and $\langle a \in A \rangle$ and $\langle b \in A \rangle$ have $P a a$ unfolding
transp-on-def by blast
with $\langle \text{irreflp-on } A P \rangle$ and $\langle a \in A \rangle$ show False unfolding irreflp-on-def by
blast
qed
qed

lemma *po-on-imp-antisymp-on*:
assumes po-on $P A$
shows antisymp-on $A P$
using transp-on-irreflp-on-imp-antisymp-on [of $A P$] and assms by (auto simp:
po-on-def)

lemma *strict-reflclp [simp]*:
assumes $x \in A$ and $y \in A$
and transp-on $A P$ and irreflp-on $A P$
shows strict ($P^{==}$) $x y = P x y$
using assms unfolding transp-on-def irreflp-on-def
by blast

lemma *qo-on-imp-reflp-on*:
qo-on $P A \implies$ reflp-on $A P$
by (auto simp: qo-on-def)

lemma *go-on-imp-transp-on*:
 $go-on\ P\ A \implies transp-on\ A\ P$
by (*auto simp: go-on-def*)

lemma *go-on-subset*:
 $A \subseteq B \implies go-on\ P\ B \implies go-on\ P\ A$
unfolding *go-on-def*
using *reflp-on-subset*
and *transp-on-subset* **by** *blast*

Quasi-orders are instances of the *preorder* class.

lemma *go-on-UNIV-conv*:
 $go-on\ P\ UNIV \longleftrightarrow class.preorder\ P\ (strict\ P)\ (is\ ?lhs = ?rhs)$
proof
assume *?lhs* **then show** *?rhs*
unfolding *go-on-def class.preorder-def*
using *go-on-imp-reflp-on [of P UNIV]*
and *go-on-imp-transp-on [of P UNIV]*
by (*auto simp: reflp-on-def*) (*unfold transp-on-def, blast*)
next
assume *?rhs* **then show** *?lhs*
unfolding *class.preorder-def*
by (*auto simp: go-on-def reflp-on-def transp-on-def*)
qed

lemma *wfp-on-iff-inductive-on*:
 $wfp-on\ P\ A \longleftrightarrow inductive-on\ P\ A$
by (*blast intro: inductive-on-imp-wfp-on wfp-on-imp-inductive-on*)

lemma *wfp-on-iff-minimal*:
 $wfp-on\ P\ A \longleftrightarrow (\forall Q\ x.$
 $x \in Q \wedge Q \subseteq A \longrightarrow$
 $(\exists z \in Q. \forall y. P\ y\ z \longrightarrow y \notin Q))$
using *wfp-on-imp-minimal [of P A]*
and *minimal-imp-inductive-on [of A P]*
and *inductive-on-imp-wfp-on [of P A]*
by *blast*

Every non-empty well-founded set A has a minimal element, i.e., an element that is not greater than any other element.

lemma *wfp-on-imp-has-min-elt*:
assumes *wfp-on P A* **and** $A \neq \{\}$
shows $\exists x \in A. \forall y \in A. \neg P\ y\ x$
using *assms* **unfolding** *wfp-on-iff-minimal* **by** *force*

lemma *wfp-on-induct [consumes 2, case-names less, induct pred: wfp-on]*:
assumes *wfp-on P A* **and** $x \in A$
and $\bigwedge y. \llbracket y \in A; \bigwedge x. \llbracket x \in A; P\ x\ y \rrbracket \implies Q\ x \rrbracket \implies Q\ y$

shows $Q\ x$
 using *assms* and *inductive-on-induct* [of $P\ A\ x$]
 unfolding *wfp-on-iff-inductive-on* by *blast*

lemma *wfp-on-UNIV* [simp]:
wfp-on $P\ UNIV \longleftrightarrow wfP\ P$
 unfolding *wfp-on-iff-inductive-on* *inductive-on-def* *wfP-def* *wf-def* by *force*

1.1 Measures on Sets (Instead of Full Types)

definition

inv-image-betw ::
 $('b \Rightarrow 'a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'a\ set \Rightarrow 'b\ set \Rightarrow ('a \Rightarrow 'a \Rightarrow bool)$

where

inv-image-betw $P\ f\ A\ B = (\lambda x\ y. x \in A \wedge y \in A \wedge f\ x \in B \wedge f\ y \in B \wedge P\ (f\ x)\ (f\ y))$

definition

measure-on :: $('a \Rightarrow nat) \Rightarrow 'a\ set \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$

where

measure-on $f\ A = inv-image-betw\ (<)\ f\ A\ UNIV$

lemma *in-inv-image-betw* [simp]:

inv-image-betw $P\ f\ A\ B\ x\ y \longleftrightarrow x \in A \wedge y \in A \wedge f\ x \in B \wedge f\ y \in B \wedge P\ (f\ x)\ (f\ y)$
 by (*auto simp: inv-image-betw-def*)

lemma *in-measure-on* [simp, code-unfold]:

measure-on $f\ A\ x\ y \longleftrightarrow x \in A \wedge y \in A \wedge f\ x < f\ y$
 by (*simp add: measure-on-def*)

lemma *wfp-on-inv-image-betw* [simp, intro!]:

assumes *wfp-on* $P\ B$

shows *wfp-on* $(inv-image-betw\ P\ f\ A\ B)\ A$ (**is** *wfp-on* $?P\ A$)

proof (*rule ccontr*)

assume $\neg\ ?thesis$

then obtain g **where** $\forall i. g\ i \in A \wedge ?P\ (g\ (Suc\ i))\ (g\ i)$ **by** (*auto simp: wfp-on-def*)

with *assms* **show** *False* **by** (*auto simp: wfp-on-def*)

qed

lemma *wfp-less*:

wfp-on $(<)\ (UNIV :: nat\ set)$

using *wf-less* **by** (*auto simp: wfP-def*)

lemma *wfp-on-measure-on* [iff]:

wfp-on $(measure-on\ f\ A)\ A$

unfolding *measure-on-def*

by (*rule wfp-less [THEN wfp-on-inv-image-betw]*)

lemma *wfp-on-mono*:

$A \subseteq B \implies (\bigwedge x y. x \in A \implies y \in A \implies P x y \implies Q x y) \implies \text{wfp-on } Q B \implies \text{wfp-on } P A$

unfolding *wfp-on-def* **by** (*metis subsetD*)

lemma *wfp-on-subset*:

$A \subseteq B \implies \text{wfp-on } P B \implies \text{wfp-on } P A$

using *wfp-on-mono* **by** *blast*

lemma *restrict-to-iff* [*iff*]:

$\text{restrict-to } P A x y \longleftrightarrow x \in A \wedge y \in A \wedge P x y$

by (*simp add: restrict-to-def*)

lemma *wfp-on-restrict-to* [*simp*]:

$\text{wfp-on } (\text{restrict-to } P A) A = \text{wfp-on } P A$

by (*auto simp: wfp-on-def*)

lemma *irreflp-on-strict* [*simp, intro*]:

irreflp-on A (*strict* P)

by (*auto simp: irreflp-on-def*)

lemma *transp-on-map'*:

assumes *transp-on* $B Q$

and $g \text{ ' } A \subseteq B$

and $h \text{ ' } A \subseteq B$

and $\bigwedge x. x \in A \implies Q = (h x) (g x)$

shows *transp-on* $A (\lambda x y. Q (g x) (h y))$

using *assms* **unfolding** *transp-on-def*

by *auto* (*metis imageI subsetD*)

lemma *transp-on-map*:

assumes *transp-on* $B Q$

and $h \text{ ' } A \subseteq B$

shows *transp-on* $A (\lambda x y. Q (h x) (h y))$

using *transp-on-map'* [*of* $B Q h A h$, *simplified, OF assms*] **by** *blast*

lemma *irreflp-on-map*:

assumes *irreflp-on* $B Q$

and $h \text{ ' } A \subseteq B$

shows *irreflp-on* $A (\lambda x y. Q (h x) (h y))$

using *assms* **unfolding** *irreflp-on-def* **by** *auto*

lemma *po-on-map*:

assumes *po-on* $Q B$

and $h \text{ ' } A \subseteq B$

shows *po-on* $(\lambda x y. Q (h x) (h y)) A$

using *assms* **and** *transp-on-map* **and** *irreflp-on-map*

unfolding *po-on-def* **by** *auto*

lemma *chain-transp-on-less*:
assumes $\forall i. f\ i \in A \wedge P\ (f\ i)\ (f\ (Suc\ i))$ **and** *transp-on A P* **and** $i < j$
shows $P\ (f\ i)\ (f\ j)$
using $\langle i < j \rangle$
proof (*induct j*)
 case 0 **then show** *?case* **by** *simp*
next
 case (*Suc j*)
 show *?case*
 proof (*cases i = j*)
 case True
 with Suc **show** *?thesis* **using** *assms(1)* **by** *simp*
 next
 case False
 with Suc **have** $P\ (f\ i)\ (f\ j)$ **by** *force*
 moreover from *assms* **have** $P\ (f\ j)\ (f\ (Suc\ j))$ **by** *auto*
 ultimately show *?thesis* **using** *assms(1, 2)* **unfolding** *transp-on-def* **by** *blast*
qed
qed

lemma *wfp-on-imp-irreflp-on*:
assumes *wfp-on P A*
shows *irreflp-on A P*
proof (*rule irreflp-onI*)
 fix x
 assume $x \in A$
 show $\neg P\ x\ x$
 proof
 let $?f = \lambda-. x$
 assume $P\ x\ x$
 then have $\forall i. P\ (?f\ (Suc\ i))\ (?f\ i)$ **by** *blast*
 with $\langle x \in A \rangle$ **have** $\neg wfp\text{-on}\ P\ A$ **by** (*auto simp: wfp-on-def*)
 with *assms* **show** *False* **by** *contradiction*
 qed
qed

inductive
 accessible-on $:: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ set \Rightarrow 'a \Rightarrow bool$
 for P **and** A
where
 accessible-onI [*Pure.intro*]:
 $\llbracket x \in A; \bigwedge y. \llbracket y \in A; P\ y\ x \rrbracket \implies accessible\text{-on}\ P\ A\ y \rrbracket \implies accessible\text{-on}\ P\ A\ x$

lemma *accessible-on-imp-mem*:
assumes *accessible-on P A a*
shows $a \in A$
using *assms* **by** (*induct*) *auto*

```

lemma accessible-on-induct [consumes 1, induct pred: accessible-on]:
  assumes *: accessible-on P A a
    and IH:  $\bigwedge x. \llbracket \text{accessible-on } P \ A \ x; \bigwedge y. \llbracket y \in A; P \ y \ x \rrbracket \implies Q \ y \rrbracket \implies Q \ x$ 
  shows Q a
  by (rule * [THEN accessible-on.induct]) (auto intro: IH accessible-onI)

lemma accessible-on-downward:
  accessible-on P A b  $\implies$  a  $\in$  A  $\implies$  P a b  $\implies$  accessible-on P A a
  by (cases rule: accessible-on.cases) fast

lemma accessible-on-restrict-to-downwards:
  assumes (restrict-to P A)++ a b and accessible-on P A b
  shows accessible-on P A a
  using assms by (induct) (auto dest: accessible-on-imp-mem accessible-on-downward)

lemma accessible-on-imp-inductive-on:
  assumes  $\forall x \in A. \text{accessible-on } P \ A \ x$ 
  shows inductive-on P A
proof
  fix Q x
  assume x  $\in$  A
    and *:  $\bigwedge y. \llbracket y \in A; \bigwedge x. \llbracket x \in A; P \ x \ y \rrbracket \implies Q \ x \rrbracket \implies Q \ y$ 
  with assms have accessible-on P A x by auto
  then show Q x
  proof (induct)
    case (1 z)
    then have z  $\in$  A by (blast dest: accessible-on-imp-mem)
    show ?case by (rule *) fact+
  qed
qed

lemmas accessible-on-imp-wfp-on = accessible-on-imp-inductive-on [THEN inductive-on-imp-wfp-on]

lemma wfp-on-tranclp-imp-wfp-on:
  assumes wfp-on (P++) A
  shows wfp-on P A
  by (rule ccontr) (insert assms, auto simp: wfp-on-def)

lemma inductive-on-imp-accessible-on:
  assumes inductive-on P A
  shows  $\forall x \in A. \text{accessible-on } P \ A \ x$ 
proof
  fix x
  assume x  $\in$  A
  with assms show accessible-on P A x
  by (induct) (auto intro: accessible-onI)
qed

```

lemma *inductive-on-accessible-on-conv*:
inductive-on $P A \longleftrightarrow (\forall x \in A. \text{accessible-on } P A x)$
using *inductive-on-imp-accessible-on*
and *accessible-on-imp-inductive-on*
by *blast*

lemmas *wfp-on-imp-accessible-on* =
wfp-on-imp-inductive-on [*THEN inductive-on-imp-accessible-on*]

lemma *wfp-on-accessible-on-iff*:
wfp-on $P A \longleftrightarrow (\forall x \in A. \text{accessible-on } P A x)$
by (*blast dest: wfp-on-imp-accessible-on accessible-on-imp-wfp-on*)

lemma *accessible-on-tranclp*:
assumes *accessible-on* $P A x$
shows *accessible-on* $((\text{restrict-to } P A)^{++}) A x$
(is accessible-on ?P A x)
using *assms*
proof (*induct*)
case (*1 x*)
then have $x \in A$ **by** (*blast dest: accessible-on-imp-mem*)
then show *?case*
proof (*rule accessible-onI*)
fix y
assume $y \in A$
assume *?P y x*
then show *accessible-on ?P A y*
proof (*cases*)
assume *restrict-to P A y x*
with 1 and $\langle y \in A \rangle$ **show** *?thesis by blast*
next
fix z
assume *?P y z and restrict-to P A z x*
with 1 have *accessible-on ?P A z* **by** (*auto simp: restrict-to-def*)
from *accessible-on-downward* [*OF this* $\langle y \in A \rangle \langle ?P y z \rangle$]
show *?thesis* .
qed
qed
qed

lemma *wfp-on-restrict-to-tranclp*:
assumes *wfp-on* $P A$
shows *wfp-on* $((\text{restrict-to } P A)^{++}) A$
using *wfp-on-imp-accessible-on* [*OF assms*]
and *accessible-on-tranclp* [*of P A*]
and *accessible-on-imp-wfp-on* [*of A (restrict-to P A)^{++}*]
by *blast*

lemma *wfp-on-restrict-to-tranclp'*:

assumes $wfp\text{-on } (restrict\text{-to } P A)^{++} A$
shows $wfp\text{-on } P A$
by (*rule ccontr*) (*insert assms, auto simp: wfp-on-def*)

lemma *wfp-on-restrict-to-tranclp-wfp-on-conv*:
 $wfp\text{-on } (restrict\text{-to } P A)^{++} A \longleftrightarrow wfp\text{-on } P A$
using *wfp-on-restrict-to-tranclp* [*of P A*]
and *wfp-on-restrict-to-tranclp'* [*of P A*]
by *blast*

lemma *tranclp-idemp* [*simp*]:
 $(P^{++})^{++} = P^{++}$ (**is** $?l = ?r$)
proof (*intro ext*)
fix $x y$
show $?l x y = ?r x y$
proof
assume $?l x y$ **then show** $?r x y$ **by** (*induct*) *auto*
next
assume $?r x y$ **then show** $?l x y$ **by** (*induct*) *auto*
qed
qed

lemma *stepfun-imp-tranclp*:
assumes $f 0 = x$ **and** $f (Suc n) = z$
and $\forall i \leq n. P (f i) (f (Suc i))$
shows $P^{++} x z$
using *assms*
by (*induct n arbitrary: x z*)
(*auto intro: tranclp.trancl-into-trancl*)

lemma *tranclp-imp-stepfun*:
assumes $P^{++} x z$
shows $\exists f n. f 0 = x \wedge f (Suc n) = z \wedge (\forall i \leq n. P (f i) (f (Suc i)))$
(**is** $\exists f n. ?P x z f n$)
using *assms*
proof (*induct rule: tranclp-induct*)
case (*base y*)
let $?f = (\lambda-. y)(0 := x)$
have $?f 0 = x$ **and** $?f (Suc 0) = y$ **by** *auto*
moreover have $\forall i \leq 0. P (?f i) (?f (Suc i))$
using *base* **by** *auto*
ultimately show $?case$ **by** *blast*
next
case (*step y z*)
then obtain $f n$ **where** *IH*: $?P x y f n$ **by** *blast*
then have $*$: $\forall i \leq n. P (f i) (f (Suc i))$
and [*simp*]: $f 0 = x \wedge f (Suc n) = z$
by *auto*

let $?n = \text{Suc } n$
let $?f = f(\text{Suc } ?n := z)$
have $?f\ 0 = x$ **and** $?f (\text{Suc } ?n) = z$ **by** *auto*
moreover have $\forall i \leq ?n. P (?f\ i) (?f (\text{Suc } i))$
using $\langle P\ y\ z \rangle$ **and** $*$ **by** *auto*
ultimately show $?case$ **by** *blast*
qed

lemma *tranclp-stepfun-conv*:
 $P^{++}\ x\ y \longleftrightarrow (\exists f\ n. f\ 0 = x \wedge f (\text{Suc } n) = y \wedge (\forall i \leq n. P (f\ i) (f (\text{Suc } i))))$
using *tranclp-imp-stepfun* **and** *stepfun-imp-tranclp* **by** *metis*

1.2 Facts About Predecessor Sets

lemma *qo-on-predecessor-subset-conv'*:
assumes $qo\text{-on } P\ A$ **and** $B \subseteq A$ **and** $C \subseteq A$
shows $\{x \in A. \exists y \in B. P\ x\ y\} \subseteq \{x \in A. \exists y \in C. P\ x\ y\} \longleftrightarrow (\forall x \in B. \exists y \in C. P\ x\ y)$
using *assms*
by (*auto simp: subset-eq qo-on-def reflp-on-def, unfold transp-on-def*) *metis+*

lemma *qo-on-predecessor-subset-conv*:
 $\llbracket qo\text{-on } P\ A; x \in A; y \in A \rrbracket \implies \{z \in A. P\ z\ x\} \subseteq \{z \in A. P\ z\ y\} \longleftrightarrow P\ x\ y$
using *qo-on-predecessor-subset-conv'* [of $P\ A\ \{x\}\ \{y\}$] **by** *simp*

lemma *po-on-predecessors-eq-conv*:
assumes $po\text{-on } P\ A$ **and** $x \in A$ **and** $y \in A$
shows $\{z \in A. P^{==}\ z\ x\} = \{z \in A. P^{==}\ z\ y\} \longleftrightarrow x = y$
using *assms(2-)*
and *reflp-on-reflclp* [of $A\ P$]
and *po-on-imp-antisymp-on* [OF $\langle po\text{-on } P\ A \rangle$]
unfolding *antisymp-on-def reflp-on-def*
by *blast*

lemma *restrict-to-rtranclp*:
assumes $transp\text{-on } A\ P$
and $x \in A$ **and** $y \in A$
shows $(\text{restrict-to } P\ A)^{**}\ x\ y \longleftrightarrow P^{==}\ x\ y$

proof –
{ **assume** $(\text{restrict-to } P\ A)^{**}\ x\ y$
then have $P^{==}\ x\ y$ **using** *assms*
by (*induct*) (*auto, unfold transp-on-def, blast*) **}**
with *assms* **show** $?thesis$ **by** *auto*
qed

lemma *reflp-on-restrict-to-rtranclp*:
assumes $reflp\text{-on } A\ P$ **and** $transp\text{-on } A\ P$
and $x \in A$ **and** $y \in A$
shows $(\text{restrict-to } P\ A)^{**}\ x\ y \longleftrightarrow P\ x\ y$
unfolding *restrict-to-rtranclp* [OF *assms(2-)*]

unfolding *reflp-on-reflclp-simp* [*OF assms(1, 3-)*] ..

end

2 Open Induction

theory *Open-Induction*
imports *Restricted-Predicates*
begin

2.1 (Greatest) Lower Bounds and Chains

A set B has the *lower bound* x iff x is less than or equal to every element of B .

definition $lb\ P\ B\ x \longleftrightarrow (\forall y \in B. P^{==}\ x\ y)$

lemma *lbI* [*Pure.intro*]:

$(\bigwedge y. y \in B \implies P^{==}\ x\ y) \implies lb\ P\ B\ x$
by (*auto simp: lb-def*)

A set B has the *greatest lower bound* x iff x is a lower bound of B and less than or equal to every other lower bound of B .

definition $glb\ P\ B\ x \longleftrightarrow lb\ P\ B\ x \wedge (\forall y. lb\ P\ B\ y \longrightarrow P^{==}\ y\ x)$

lemma *glbI* [*Pure.intro*]:

$lb\ P\ B\ x \implies (\bigwedge y. lb\ P\ B\ y \implies P^{==}\ y\ x) \implies glb\ P\ B\ x$
by (*auto simp: glb-def*)

Antisymmetric relations have unique glbs.

lemma *glb-unique*:

$antisymp\text{-}on\ A\ P \implies x \in A \implies y \in A \implies glb\ P\ B\ x \implies glb\ P\ B\ y \implies x = y$
by (*auto simp: glb-def antisymp-on-def*)

context *pred-on*

begin

lemma *chain-glb*:

assumes *transp-on* $A\ (\sqsubset)$
shows $chain\ C \implies glb\ (\sqsubset)\ C\ x \implies x \in A \implies y \in A \implies y \sqsubset x \implies chain\ (\{y\} \cup C)$
using *assms* [*unfolded transp-on-def*]
unfolding *chain-def glb-def lb-def*
by (*cases* $C = \{\}$) *blast+*

2.2 Open Properties

definition *open* $Q \longleftrightarrow (\forall C. chain\ C \wedge C \neq \{\} \wedge (\exists x \in A. glb\ (\sqsubset)\ C\ x \wedge Q\ x) \longrightarrow (\exists y \in C. Q\ y))$

lemma *openI* [*Pure.intro*]:
 $(\bigwedge C. \text{chain } C \implies C \neq \{\}) \implies \exists x \in A. \text{glb } (\sqsubset) C x \wedge Q x \implies \exists y \in C. Q y \implies$
open Q
by (*auto simp: open-def*)

lemma *open-glb*:
 $\llbracket \text{chain } C; C \neq \{\}; \text{open } Q; \forall x \in C. \neg Q x; x \in A; \text{glb } (\sqsubset) C x \rrbracket \implies \neg Q x$
by (*auto simp: open-def*)

2.3 Downward Completeness

A relation \sqsubset is *downward-complete* iff every non-empty \sqsubset -chain has a greatest lower bound.

definition *downward-complete* $\longleftrightarrow (\forall C. \text{chain } C \wedge C \neq \{\} \longrightarrow (\exists x \in A. \text{glb } (\sqsubset) C x))$

lemma *downward-completeI* [*Pure.intro*]:
assumes $\bigwedge C. \text{chain } C \implies C \neq \{\} \implies \exists x \in A. \text{glb } (\sqsubset) C x$
shows *downward-complete*
using *assms* **by** (*auto simp: downward-complete-def*)

end

abbreviation *open-on P Q A* $\equiv \text{pred-on.open } A P Q$
abbreviation *dc-on P A* $\equiv \text{pred-on.downward-complete } A P$
lemmas *open-on-def = pred-on.open-def*
and *dc-on-def = pred-on.downward-complete-def*

lemma *dc-onI* [*Pure.intro*]:
assumes $\bigwedge C. \text{chain-on } P C A \implies C \neq \{\} \implies \exists x \in A. \text{glb } P C x$
shows *dc-on P A*
using *assms* **by** (*auto simp: dc-on-def*)

lemma *open-onI* [*Pure.intro*]:
 $(\bigwedge C. \text{chain-on } P C A \implies C \neq \{\} \implies \exists x \in A. \text{glb } P C x \wedge Q x \implies \exists y \in C. Q y) \implies \text{open-on } P Q A$
by (*auto simp: open-on-def*)

lemma *chain-on-reflcp*:
 $\text{chain-on } P^{==} A C \longleftrightarrow \text{chain-on } P A C$
by (*auto simp: pred-on.chain-def*)

lemma *lb-reflcp*:
 $\text{lb } P^{==} B x \longleftrightarrow \text{lb } P B x$
by (*auto simp: lb-def*)

lemma *glb-reflcp*:
 $\text{glb } P^{==} B x \longleftrightarrow \text{glb } P B x$

by (auto simp: glb-def lb-reflclp)

lemma *dc-on-reflclp*:

dc-on $P \stackrel{=}{=} A \iff \text{dc-on } P \ A$

by (auto simp: dc-on-def chain-on-reflclp glb-reflclp)

2.4 The Open Induction Principle

lemma *open-induct-on* [*consumes 4, case-names less*]:

assumes *qo*: *qo-on* $P \ A$ and *dc-on* $P \ A$ and *open-on* $P \ Q \ A$

and $x \in A$

and *ind*: $\bigwedge x. [x \in A; \bigwedge y. [y \in A; \text{strict } P \ y \ x]] \implies Q \ y] \implies Q \ x$

shows $Q \ x$

proof (*rule ccontr*)

assume $\neg Q \ x$

let $?B = \{x \in A. \neg Q \ x\}$

have $?B \subseteq A$ by *blast*

interpret B : *pred-on* $?B \ P$.

from *B.Hausdorff* obtain M

where *chain*: B .*chain* M

and *max*: $\bigwedge C. B$.*chain* $C \implies M \subseteq C \implies M = C$ by (auto simp: *B.maxchain-def*)

then have $M \subseteq ?B$ by (auto simp: *B.chain-def*)

show *False*

proof (*cases* $M = \{\}$)

assume $M = \{\}$

moreover have B .*chain* $\{x\}$ using $\langle x \in A \rangle$ and $\langle \neg Q \ x \rangle$ by (*simp add*: *B.chain-def*)

ultimately show *False* using *max* by *blast*

next

interpret A : *pred-on* $A \ P$.

assume $M \neq \{\}$

have A .*chain* M using *chain* by (auto simp: A .*chain-def* B .*chain-def*)

moreover with $\langle \text{dc-on } P \ A \rangle$ and $\langle M \neq \{\} \rangle$ obtain m

where $m \in A$ and *glb* $P \ M \ m$ by (auto simp: A .*downward-complete-def*)

ultimately have $\neg Q \ m$ and $m \in ?B$

using A .*open-glb* [*OF* - $\langle M \neq \{\} \rangle$ $\langle \text{open-on } P \ Q \ A \rangle$ - - $\langle \text{glb } P \ M \ m \rangle$]

and $\langle M \subseteq ?B \rangle$ by *auto*

from *ind* [*OF* $\langle m \in A \rangle$] and $\langle \neg Q \ m \rangle$ obtain y

where $y \in A$ and *strict* $P \ y \ m$ and $\neg Q \ y$ by *blast*

then have $P \ y \ m$ and $y \in ?B$ by *simp+*

from *transp-on-subset* [*OF* *qo-on-imp-transp-on* [*OF* *qo*] $\langle ?B \subseteq A \rangle$]

have *transp-on* $?B \ P$.

from B .*chain-glb* [*OF* *this chain* $\langle \text{glb } P \ M \ m \rangle$ $\langle m \in ?B \rangle$ $\langle y \in ?B \rangle$ $\langle P \ y \ m \rangle$]

have B .*chain* $(\{y\} \cup M)$.

then show *False*

using $\langle \text{glb } P \ M \ m \rangle$ and $\langle \text{strict } P \ y \ m \rangle$ by (*cases* $y \in M$) (auto *dest*: *max simp*: *glb-def lb-def*)

qed

qed

2.5 Open Induction on Universal Domains

Open induction on quasi-orders (i.e., *preorder*).

```
lemma (in preorder) dc-open-induct [consumes 2, case-names less]:  
  assumes dc-on ( $\leq$ ) UNIV  
    and open-on ( $\leq$ ) Q UNIV  
    and  $\bigwedge x. (\bigwedge y. y < x \implies Q y) \implies Q x$   
  shows  $Q x$   
proof –  
  have qo-on ( $\leq$ ) UNIV by (auto simp: qo-on-def transp-on-def reflp-on-def dest:  
order-trans)  
  from open-induct-on [OF this assms(1,2)]  
  show  $Q x$  using assms(3) unfolding less-le-not-le by blast  
qed
```

2.6 Type Class of Downward Complete Orders

```
class dcorder = preorder +  
  assumes dc-on-UNIV: dc-on ( $\leq$ ) UNIV  
begin
```

Open induction on downward-complete orders.

```
lemmas open-induct [consumes 1, case-names less] = dc-open-induct [OF dc-on-UNIV]  
end  
end
```

References

- [1] J.-C. Raoult. Proving open properties by induction. *Information Processing Letters*, 29(1):19–23, 1988. doi:10.1016/0020-0190(88)90126-3.