PreOrders

Roger Bishop Jones

Date

Abstract

This is a theory of pre-orders, i.e. reflexive transitive relations, obtained in the first instance by taking the theory Orderings (which means partial orders), removing the antisymmetry axiom, defining a notion of equivalence and substituting equivalence for equality in the conclusions of theorems. Also covers linear pre-orders.

http://www.rbjones.com/rbjpub/isar/HOL/PreOrders.pdf
http://www.rbjones.com/rbjpub/isar/HOL/PreOrders.tgz

Id

Contents

1 Introduction

-

2 PreOrders

theory *PreOrders*
imports *Code-Setup*
uses
  ~~/src/Provers/order.ML
begin

2.1 pre-orders

adapted by rbj from Orderings.thy (by Tobias Nipkow, Markus Wenzel, and Larry Paulson)

class *preorder* = *ord* +
  assumes *less-le*: $x < y \longleftrightarrow x \leq y \wedge \neg\, y \leq x$
  and *preorder-refl* [*iff*]: $x \leq x$
  and *preorder-trans*: $x \leq y \Longrightarrow y \leq z \Longrightarrow x \leq z$

begin

definition
  pro-eq::$'a \Rightarrow 'a \Rightarrow bool$
where *pro-eq* $x\ y == x \leq y \wedge y \leq x$

notation
  pro-eq (*op* ~) **and**
  pro-eq ((-/ ~ -) [*50*, *51*] *50*)

lemma *pro-eq-refl* [*simp*]: $x \sim x$

⟨*proof*⟩

”AntiSymmetry”

lemma *antisym* [*intro*]: $x \le y \Longrightarrow y \le x \Longrightarrow x \sim y$
⟨*proof*⟩

Reflexivity.

lemma *eq-refl*: $x = y \Longrightarrow x \le y$
— This form is useful with the classical reasoner.
⟨*proof*⟩

lemma *less-irrefl* [*iff*]: $\neg\ x < x$
⟨*proof*⟩

lemma *le-less*: $x \le y \longleftrightarrow x < y \lor \textit{pro-eq}\ x\ y$
— NOT suitable for iff, since it can cause PROOF FAILED.
⟨*proof*⟩

lemma *le-imp-less-or-eq*: $x \le y \Longrightarrow x < y \lor x \sim y$
⟨*proof*⟩

lemma *less-imp-le*: $x < y \Longrightarrow x \le y$
⟨*proof*⟩

Useful for simplification, but too risky to include by default.

lemma *less-imp-not-eq*: $x < y \Longrightarrow (x \sim y) \longleftrightarrow \textit{False}$
⟨*proof*⟩

lemma *less-imp-not-eq2*: $x < y \Longrightarrow (y \sim x) \longleftrightarrow \textit{False}$
⟨*proof*⟩

Transitivity rules for calculational reasoning

lemma *neq-le-trans*: $\neg\ a \sim b \Longrightarrow a \le b \Longrightarrow a < b$
⟨*proof*⟩

lemma *le-neq-trans*: $a \le b \Longrightarrow \neg\ a \sim b \Longrightarrow a < b$
⟨*proof*⟩

”Asymmetry”

lemma *less-not-sym*: $x < y \Longrightarrow \neg\ (y < x)$
⟨*proof*⟩

lemma *less-asym*: $x < y \Longrightarrow (\neg\ P \Longrightarrow y < x) \Longrightarrow P$
⟨*proof*⟩

lemma *eq-iff*: $x \sim y \longleftrightarrow x \le y \land y \le x$
⟨*proof*⟩

lemma *antisym-conv*: $y \leq x \Longrightarrow x \leq y \longleftrightarrow x \sim y$
⟨*proof*⟩

lemma *less-imp-neq*: $x < y \Longrightarrow \neg\ x \sim y$
⟨*proof*⟩

Transitivity.

lemma *less-trans*: $x < y \Longrightarrow y < z \Longrightarrow x < z$
⟨*proof*⟩

lemma *le-less-trans*: $x \leq y \Longrightarrow y < z \Longrightarrow x < z$
⟨*proof*⟩

lemma *less-le-trans*: $x < y \Longrightarrow y \leq z \Longrightarrow x < z$
⟨*proof*⟩

Useful for simplification, but too risky to include by default.

lemma *less-imp-not-less*: $x < y \Longrightarrow (\neg\ y < x) \longleftrightarrow \mathit{True}$
⟨*proof*⟩

lemma *less-imp-triv*: $x < y \Longrightarrow (y < x \longrightarrow P) \longleftrightarrow \mathit{True}$
⟨*proof*⟩

Transitivity rules for calculational reasoning

lemma *less-asym'*: $a < b \Longrightarrow b < a \Longrightarrow P$
⟨*proof*⟩

Dual preorder

lemma *dual-preorder*:
  $\mathit{preorder}\ (\mathit{op}\ \geq)\ (\mathit{op}\ >)$
⟨*proof*⟩

end

2.2 Linear preorders

class *linpreorder* = *preorder* +
  assumes *linear*: $x \leq y \vee y \leq x$
begin

lemma *less-linear*: $x < y \vee x \sim y \vee y < x$
⟨*proof*⟩

lemma *le-less-linear*: $x \leq y \vee y < x$
⟨*proof*⟩

lemma *le-cases* [*case-names le ge*]:
  $(x \leq y \Longrightarrow P) \Longrightarrow (y \leq x \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

lemma *linpreorder-cases* [*case-names less equal greater*]:
$(x < y \Longrightarrow P) \Longrightarrow (x \sim y \Longrightarrow P) \Longrightarrow (y < x \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

lemma *not-less*: $\neg\ x < y \longleftrightarrow y \leq x$
⟨*proof*⟩

lemma *not-less-iff-gr-or-eq*:
$\neg(x < y) \longleftrightarrow (x > y \mid x \sim y)$
⟨*proof*⟩

lemma *not-le*: $\neg\ x \leq y \longleftrightarrow y < x$
⟨*proof*⟩

lemma *neq-iff*: $\neg\ x \sim y \longleftrightarrow x < y \lor y < x$
⟨*proof*⟩

lemma *neqE*: $\neg\ x \sim y \Longrightarrow (x < y \Longrightarrow R) \Longrightarrow (y < x \Longrightarrow R) \Longrightarrow R$
⟨*proof*⟩

lemma *antisym-conv1*: $\neg\ x < y \Longrightarrow x \leq y \longleftrightarrow x \sim y$
⟨*proof*⟩

lemma *antisym-conv2*: $x \leq y \Longrightarrow \neg\ x < y \longleftrightarrow x \sim y$
⟨*proof*⟩

lemma *antisym-conv3*: $\neg\ y < x \Longrightarrow \neg\ x < y \longleftrightarrow x \sim y$
⟨*proof*⟩

Replacing the old Nat.leI

lemma *leI*: $\neg\ x < y \Longrightarrow y \leq x$
⟨*proof*⟩

lemma *leD*: $y \leq x \Longrightarrow \neg\ x < y$
⟨*proof*⟩

lemma *not-leE*: $\neg\ y \leq x \Longrightarrow x < y$
⟨*proof*⟩

Dual order

lemma *dual-linpreorder*:
linpreorder $(op \geq)$ $(op >)$
⟨*proof*⟩

min/max

for historic reasons, definitions are done in context ord

definition (**in** *ord*)
$min :: 'a \Rightarrow 'a \Rightarrow 'a$ **where**
[*code unfold*, *code inline del*]: $min\ a\ b = (if\ a \leq b\ then\ a\ else\ b)$

definition (**in** *ord*)
$max :: 'a \Rightarrow 'a \Rightarrow 'a$ **where**
[*code unfold*, *code inline del*]: $max\ a\ b = (if\ a \leq b\ then\ b\ else\ a)$

lemma *min-le-iff-disj*:
$min\ x\ y \leq z \longleftrightarrow x \leq z \lor y \leq z$
⟨*proof*⟩

lemma *le-max-iff-disj*:
$z \leq max\ x\ y \longleftrightarrow z \leq x \lor z \leq y$
⟨*proof*⟩

lemma *min-less-iff-disj*:
$min\ x\ y < z \longleftrightarrow x < z \lor y < z$
⟨*proof*⟩

lemma *less-max-iff-disj*:
$z < max\ x\ y \longleftrightarrow z < x \lor z < y$
⟨*proof*⟩

lemma *min-less-iff-conj* [*simp*]:
$z < min\ x\ y \longleftrightarrow z < x \land z < y$
⟨*proof*⟩

lemma *max-less-iff-conj* [*simp*]:
$max\ x\ y < z \longleftrightarrow x < z \land y < z$
⟨*proof*⟩

lemma *split-min* [*noatp*]:
$P\ (min\ i\ j) \longleftrightarrow (i \leq j \longrightarrow P\ i) \land (\neg\ i \leq j \longrightarrow P\ j)$
⟨*proof*⟩

lemma *split-max* [*noatp*]:
$P\ (max\ i\ j) \longleftrightarrow (i \leq j \longrightarrow P\ j) \land (\neg\ i \leq j \longrightarrow P\ i)$
⟨*proof*⟩

end

2.3 Reasoning tools setup

⟨*ML*⟩

Declarations to set up transitivity reasoner of partial and linear orders.

context *preorder*
begin

lemmas
[*preorder add less-reflE*: *preorder op* = :: ′*a* ⇒ ′*a* ⇒ *bool op* <= *op* <] =
less-irrefl [*THEN notE*]
lemmas
[*preorder add le-refl*: *preorder op* = :: ′*a* => ′*a* => *bool op* <= *op* <] =
preorder-refl
lemmas
[*preorder add less-imp-le*: *preorder op* = :: ′*a* => ′*a* => *bool op* <= *op* <] =
less-imp-le
lemmas
[*preorder add eqI*: *preorder op* = :: ′*a* => ′*a* => *bool op* <= *op* <] =
antisym
lemmas
[*preorder add eqD1*: *preorder op* = :: ′*a* => ′*a* => *bool op* <= *op* <] =
eq-refl
lemmas
[*preorder add eqD2*: *preorder op* = :: ′*a* => ′*a* => *bool op* <= *op* <] =
sym [*THEN eq-refl*]
lemmas
[*preorder add less-trans*: *preorder op* = :: ′*a* => ′*a* => *bool op* <= *op* <] =
less-trans
lemmas
[*preorder add less-le-trans*: *preorder op* = :: ′*a* => ′*a* => *bool op* <= *op* <] =
less-le-trans
lemmas
[*preorder add le-less-trans*: *preorder op* = :: ′*a* => ′*a* => *bool op* <= *op* <] =
le-less-trans
lemmas
[*preorder add le-trans*: *preorder op* = :: ′*a* => ′*a* => *bool op* <= *op* <] =
preorder-trans
lemmas
[*preorder add le-neq-trans*: *preorder op* = :: ′*a* => ′*a* => *bool op* <= *op* <] =
le-neq-trans
lemmas
[*preorder add neq-le-trans*: *preorder op* = :: ′*a* => ′*a* => *bool op* <= *op* <] =
neq-le-trans
lemmas
[*preorder add less-imp-neq*: *preorder op* = :: ′*a* => ′*a* => *bool op* <= *op* <] =
less-imp-neq
lemmas
[*preorder add eq-neq-eq-imp-neq*: *preorder op* = :: ′*a* => ′*a* => *bool op* <= *op*
<] =
eq-neq-eq-imp-neq
lemmas
[*preorder add not-sym*: *preorder op* = :: ′*a* => ′*a* => *bool op* <= *op* <] =
not-sym

end

context *linpreorder*
begin

lemmas
[*preorder del*: *preorder op* = :: *'a* => *'a* => *bool op* <= *op* <] = -

lemmas
[*preorder add less-reflE*: *linpreorder op* = :: *'a* => *'a* => *bool op* <= *op* <] = *less-irrefl* [*THEN notE*]
lemmas
[*preorder add le-refl*: *linpreorder op* = :: *'a* => *'a* => *bool op* <= *op* <] = *preorder-refl*
lemmas
[*preorder add less-imp-le*: *linpreorder op* = :: *'a* => *'a* => *bool op* <= *op* <] = *less-imp-le*
lemmas
[*preorder add not-lessI*: *linpreorder op* = :: *'a* => *'a* => *bool op* <= *op* <] = *not-less* [*THEN iffD2*]
lemmas
[*preorder add not-leI*: *linpreorder op* = :: *'a* => *'a* => *bool op* <= *op* <] = *not-le* [*THEN iffD2*]
lemmas
[*preorder add not-lessD*: *linpreorder op* = :: *'a* => *'a* => *bool op* <= *op* <] = *not-less* [*THEN iffD1*]
lemmas
[*preorder add not-leD*: *linpreorder op* = :: *'a* => *'a* => *bool op* <= *op* <] = *not-le* [*THEN iffD1*]
lemmas
[*preorder add eqI*: *linpreorder op* = :: *'a* => *'a* => *bool op* <= *op* <] = *antisym*
lemmas
[*preorder add eqD1*: *linpreorder op* = :: *'a* => *'a* => *bool op* <= *op* <] = *eq-refl*
lemmas
[*preorder add eqD2*: *linpreorder op* = :: *'a* => *'a* => *bool op* <= *op* <] = *sym* [*THEN eq-refl*]
lemmas
[*preorder add less-trans*: *linpreorder op* = :: *'a* => *'a* => *bool op* <= *op* <] = *less-trans*
lemmas
[*preorder add less-le-trans*: *linpreorder op* = :: *'a* => *'a* => *bool op* <= *op* <] = *less-le-trans*
lemmas
[*preorder add le-less-trans*: *linpreorder op* = :: *'a* => *'a* => *bool op* <= *op* <] = *le-less-trans*
lemmas
[*preorder add le-trans*: *linpreorder op* = :: *'a* => *'a* => *bool op* <= *op* <] =

preorder-trans
lemmas
[*preorder add le-neq-trans*: *linpreorder op* = :: *'a* => *'a* => *bool op* <= *op* <] =
le-neq-trans
lemmas
[*preorder add neq-le-trans*: *linpreorder op* = :: *'a* => *'a* => *bool op* <= *op* <] =
neq-le-trans
lemmas
[*preorder add less-imp-neq*: *linpreorder op* = :: *'a* => *'a* => *bool op* <= *op* <] =
less-imp-neq
lemmas
[*preorder add eq-neq-eq-imp-neq*: *linpreorder op* = :: *'a* => *'a* => *bool op* <= *op*
<] =
eq-neq-eq-imp-neq
lemmas
[*preorder add not-sym*: *linpreorder op* = :: *'a* => *'a* => *bool op* <= *op* <] =
not-sym

end

⟨*ML*⟩

2.4 Name duplicates

lemmas *preorder-less-le* = *less-le*
lemmas *preorder-eq-refl* = *preorder-class.eq-refl*
lemmas *preorder-less-irrefl* = *preorder-class.less-irrefl*
lemmas *preorder-le-less* = *preorder-class.le-less*
lemmas *preorder-le-imp-less-or-eq* = *preorder-class.le-imp-less-or-eq*
lemmas *preorder-less-imp-le* = *preorder-class.less-imp-le*
lemmas *preorder-less-imp-not-eq* = *preorder-class.less-imp-not-eq*
lemmas *preorder-less-imp-not-eq2* = *preorder-class.less-imp-not-eq2*
lemmas *preorder-neq-le-trans* = *preorder-class.neq-le-trans*
lemmas *preorder-le-neq-trans* = *preorder-class.le-neq-trans*

lemmas *preorder-antisym* = *antisym*
lemmas *preorder-less-not-sym* = *preorder-class.less-not-sym*
lemmas *preorder-less-asym* = *preorder-class.less-asym*
lemmas *preorder-eq-iff* = *preorder-class.eq-iff*
lemmas *preorder-antisym-conv* = *preorder-class.antisym-conv*
lemmas *preorder-less-trans* = *preorder-class.less-trans*
lemmas *preorder-le-less-trans* = *preorder-class.le-less-trans*
lemmas *preorder-less-le-trans* = *preorder-class.less-le-trans*
lemmas *preorder-less-imp-not-less* = *preorder-class.less-imp-not-less*
lemmas *preorder-less-imp-triv* = *preorder-class.less-imp-triv*
lemmas *preorder-less-asym'* = *preorder-class.less-asym'*

lemmas *linpreorder-linear* = *linear*

lemmas *linpreorder-less-linear* = *linpreorder-class.less-linear*
lemmas *linpreorder-le-less-linear* = *linpreorder-class.le-less-linear*
lemmas *linpreorder-le-cases* = *linpreorder-class.le-cases*
lemmas *linpreorder-not-less* = *linpreorder-class.not-less*
lemmas *linpreorder-not-le* = *linpreorder-class.not-le*
lemmas *linpreorder-neq-iff* = *linpreorder-class.neq-iff*
lemmas *linpreorder-neqE* = *linpreorder-class.neqE*
lemmas *linpreorder-antisym-conv1* = *linpreorder-class.antisym-conv1*
lemmas *linpreorder-antisym-conv2* = *linpreorder-class.antisym-conv2*
lemmas *linpreorder-antisym-conv3* = *linpreorder-class.antisym-conv3*

2.5 Bounded quantifiers

syntax
-All-less :: [idt, 'a, bool] => bool ((3ALL -<-./ -) [0, 0, 10] 10)
-Ex-less :: [idt, 'a, bool] => bool ((3EX -<-./ -) [0, 0, 10] 10)
-All-less-eq :: [idt, 'a, bool] => bool ((3ALL -<=-./ -) [0, 0, 10] 10)
-Ex-less-eq :: [idt, 'a, bool] => bool ((3EX -<=-./ -) [0, 0, 10] 10)

-All-greater :: [idt, 'a, bool] => bool ((3ALL ->-./ -) [0, 0, 10] 10)
-Ex-greater :: [idt, 'a, bool] => bool ((3EX ->-./ -) [0, 0, 10] 10)
-All-greater-eq :: [idt, 'a, bool] => bool ((3ALL ->=-./ -) [0, 0, 10] 10)
-Ex-greater-eq :: [idt, 'a, bool] => bool ((3EX ->=-./ -) [0, 0, 10] 10)

syntax (*xsymbols*)
-All-less :: [idt, 'a, bool] => bool ((3∀ -<-./ -) [0, 0, 10] 10)
-Ex-less :: [idt, 'a, bool] => bool ((3∃ -<-./ -) [0, 0, 10] 10)
-All-less-eq :: [idt, 'a, bool] => bool ((3∀ -≤-./ -) [0, 0, 10] 10)
-Ex-less-eq :: [idt, 'a, bool] => bool ((3∃ -≤-./ -) [0, 0, 10] 10)

-All-greater :: [idt, 'a, bool] => bool ((3∀ ->-./ -) [0, 0, 10] 10)
-Ex-greater :: [idt, 'a, bool] => bool ((3∃ ->-./ -) [0, 0, 10] 10)
-All-greater-eq :: [idt, 'a, bool] => bool ((3∀ -≥-./ -) [0, 0, 10] 10)
-Ex-greater-eq :: [idt, 'a, bool] => bool ((3∃ -≥-./ -) [0, 0, 10] 10)

syntax (*HOL*)
-All-less :: [idt, 'a, bool] => bool ((3! -<-./ -) [0, 0, 10] 10)
-Ex-less :: [idt, 'a, bool] => bool ((3? -<-./ -) [0, 0, 10] 10)
-All-less-eq :: [idt, 'a, bool] => bool ((3! -<=-./ -) [0, 0, 10] 10)
-Ex-less-eq :: [idt, 'a, bool] => bool ((3? -<=-./ -) [0, 0, 10] 10)

syntax (*HTML* **output**)
-All-less :: [idt, 'a, bool] => bool ((3∀ -<-./ -) [0, 0, 10] 10)
-Ex-less :: [idt, 'a, bool] => bool ((3∃ -<-./ -) [0, 0, 10] 10)
-All-less-eq :: [idt, 'a, bool] => bool ((3∀ -≤-./ -) [0, 0, 10] 10)
-Ex-less-eq :: [idt, 'a, bool] => bool ((3∃ -≤-./ -) [0, 0, 10] 10)

-All-greater :: [idt, 'a, bool] => bool ((3∀ ->-./ -) [0, 0, 10] 10)
-Ex-greater :: [idt, 'a, bool] => bool ((3∃ ->-./ -) [0, 0, 10] 10)

$-All\text{-}greater\text{-}eq :: [idt, 'a, bool] => bool \quad ((3\forall \text{-}\geq\text{-}./ \text{-}) [0, 0, 10] 10)$
$-Ex\text{-}greater\text{-}eq :: [idt, 'a, bool] => bool \quad ((3\exists \text{-}\geq\text{-}./ \text{-}) [0, 0, 10] 10)$

translations
$ALL\ x<y.\ P \ => \ ALL\ x.\ x < y \longrightarrow P$
$EX\ x<y.\ P \ => \ EX\ x.\ x < y \wedge P$
$ALL\ x<=y.\ P \ => \ ALL\ x.\ x <= y \longrightarrow P$
$EX\ x<=y.\ P \ => \ EX\ x.\ x <= y \wedge P$
$ALL\ x>y.\ P \ => \ ALL\ x.\ x > y \longrightarrow P$
$EX\ x>y.\ P \ => \ EX\ x.\ x > y \wedge P$
$ALL\ x>=y.\ P \ => \ ALL\ x.\ x >= y \longrightarrow P$
$EX\ x>=y.\ P \ => \ EX\ x.\ x >= y \wedge P$

⟨*ML*⟩

2.6 Transitivity reasoning

context *ord*
begin

lemma *prord-le-eq-trans*: $a \leq b \Longrightarrow b = c \Longrightarrow a \leq c$
⟨*proof*⟩

lemma *prord-eq-le-trans*: $a = b \Longrightarrow b \leq c \Longrightarrow a \leq c$
⟨*proof*⟩

lemma *prord-less-eq-trans*: $a < b \Longrightarrow b = c \Longrightarrow a < c$
⟨*proof*⟩

lemma *prord-eq-less-trans*: $a = b \Longrightarrow b < c \Longrightarrow a < c$
⟨*proof*⟩

end

lemma *preorder-less-subst2*: $(a::'a::preorder) < b ==> f\ b < (c::'c::preorder) ==>$
$(!!x\ y.\ x < y ==> f\ x < f\ y) ==> f\ a < c$
⟨*proof*⟩

lemma *preorder-less-subst1*: $(a::'a::preorder) < f\ b ==> (b::'b::preorder) < c ==>$
$(!!x\ y.\ x < y ==> f\ x < f\ y) ==> a < f\ c$
⟨*proof*⟩

lemma *preorder-le-less-subst2*: $(a::'a::preorder) <= b ==> f\ b < (c::'c::preorder) ==>$
$(!!x\ y.\ x <= y ==> f\ x <= f\ y) ==> f\ a < c$
⟨*proof*⟩

lemma *preorder-le-less-subst1*: $(a::'a::preorder) <= f\ b ==> (b::'b::preorder) < c ==>$
$(!!x\ y.\ x < y ==> f\ x < f\ y) ==> a < f\ c$
⟨*proof*⟩

lemma *preorder-less-le-subst2*: $(a::'a::preorder) < b ==> f\ b <= (c::'c::preorder) ==>$
$(!!x\ y.\ x < y ==> f\ x < f\ y) ==> f\ a < c$
⟨*proof*⟩

lemma *preorder-less-le-subst1*: $(a::'a::preorder) < f\ b ==> (b::'b::preorder) <= c ==>$
$(!!x\ y.\ x <= y ==> f\ x <= f\ y) ==> a < f\ c$
⟨*proof*⟩

lemma *preorder-subst1*: $(a::'a::preorder) <= f\ b ==> (b::'b::preorder) <= c ==>$
$(!!x\ y.\ x <= y ==> f\ x <= f\ y) ==> a <= f\ c$
⟨*proof*⟩

lemma *preorder-subst2*: $(a::'a::preorder) <= b ==> f\ b <= (c::'c::preorder) ==>$
$(!!x\ y.\ x <= y ==> f\ x <= f\ y) ==> f\ a <= c$
⟨*proof*⟩

lemma *prord-le-eq-subst*: $a <= b ==> f\ b = c ==>$
$(!!x\ y.\ x <= y ==> f\ x <= f\ y) ==> f\ a <= c$
⟨*proof*⟩

lemma *prord-eq-le-subst*: $a = f\ b ==> b <= c ==>$
$(!!x\ y.\ x <= y ==> f\ x <= f\ y) ==> a <= f\ c$
⟨*proof*⟩

lemma *prord-less-eq-subst*: $a < b ==> f\ b = c ==>$
$(!!x\ y.\ x < y ==> f\ x < f\ y) ==> f\ a < c$
⟨*proof*⟩

lemma *prord-eq-less-subst*: $a = f\ b ==> b < c ==>$
$(!!x\ y.\ x < y ==> f\ x < f\ y) ==> a < f\ c$
⟨*proof*⟩

Note that this list of rules is in reverse order of priorities.

lemmas *preorder-trans-rules* [*trans*] =
preorder-less-subst2
preorder-less-subst1
preorder-le-less-subst2
preorder-le-less-subst1
preorder-less-le-subst2
preorder-less-le-subst1
preorder-subst2
preorder-subst1

prord-le-eq-subst
prord-eq-le-subst
prord-less-eq-subst
prord-eq-less-subst
forw-subst
back-subst
rev-mp
mp
preorder-neq-le-trans
preorder-le-neq-trans
preorder-less-trans
preorder-less-asym'
preorder-le-less-trans
preorder-less-le-trans
preorder-trans
preorder-antisym
prord-le-eq-trans
prord-eq-le-trans
prord-less-eq-trans
prord-eq-less-trans
trans

These support proving chains of decreasing inequalities a ¿= b ¿= c ... in Isar proofs.

lemma *xt1*:
$a = b ==> b > c ==> a > c$
$a > b ==> b = c ==> a > c$
$a = b ==> b >= c ==> a >= c$
$a >= b ==> b = c ==> a >= c$
$(x::'a::preorder) >= y ==> y >= z ==> x >= z$
$(x::'a::preorder) > y ==> y >= z ==> x > z$
$(x::'a::preorder) >= y ==> y > z ==> x > z$
$(a::'a::preorder) > b ==> b > a ==> P$
$(x::'a::preorder) > y ==> y > z ==> x > z$
$(a::'a::preorder) >= b ==> \neg\ b >= a ==> a > b$
$\neg(a::'a::preorder) >= b ==> b >= a ==> b > a$
$a = f\ b ==> b > c ==> (!!x\ y.\ x > y ==> f\ x > f\ y) ==> a > f\ c$
$a > b ==> f\ b = c ==> (!!x\ y.\ x > y ==> f\ x > f\ y) ==> f\ a > c$
$a = f\ b ==> b >= c ==> (!!x\ y.\ x >= y ==> f\ x >= f\ y) ==> a >= f\ c$
$a >= b ==> f\ b = c ==> (!!\ x\ y.\ x >= y ==> f\ x >= f\ y) ==> f\ a >= c$
⟨*proof*⟩

lemma *xt2*:
$(a::'a::preorder) >= f\ b ==> b >= c ==> (!!x\ y.\ x >= y ==> f\ x >= f\ y) ==> a >= f\ c$
⟨*proof*⟩

lemma *xt3*: $(a::'a::preorder) >= b ==> (f\ b::'b::preorder) >= c ==>$
$(!!x\ y.\ x >= y ==> f\ x >= f\ y) ==> f\ a >= c$

$\langle proof \rangle$

lemma *xt4*: $(a::'a::preorder) > f\ b$ ==> $(b::'b::preorder) >= c$ ==>
  $(!!x\ y.\ x >= y$ ==> $f\ x >= f\ y)$ ==> $a > f\ c$
$\langle proof \rangle$

lemma *xt5*: $(a::'a::preorder) > b$ ==> $(f\ b::'b::preorder) >= c$==>
  $(!!x\ y.\ x > y$ ==> $f\ x > f\ y)$ ==> $f\ a > c$
$\langle proof \rangle$

lemma *xt6*: $(a::'a::preorder) >= f\ b$ ==> $b > c$ ==>
  $(!!x\ y.\ x > y$ ==> $f\ x > f\ y)$ ==> $a > f\ c$
$\langle proof \rangle$

lemma *xt7*: $(a::'a::preorder) >= b$ ==> $(f\ b::'b::preorder) > c$ ==>
  $(!!x\ y.\ x >= y$ ==> $f\ x >= f\ y)$ ==> $f\ a > c$
$\langle proof \rangle$

lemma *xt8*: $(a::'a::preorder) > f\ b$ ==> $(b::'b::preorder) > c$ ==>
  $(!!x\ y.\ x > y$ ==> $f\ x > f\ y)$ ==> $a > f\ c$
$\langle proof \rangle$

lemma *xt9*: $(a::'a::preorder) > b$ ==> $(f\ b::'b::preorder) > c$ ==>
  $(!!x\ y.\ x > y$ ==> $f\ x > f\ y)$ ==> $f\ a > c$
$\langle proof \rangle$

lemmas *xtrans* = *xt1 xt2 xt3 xt4 xt5 xt6 xt7 xt8 xt9*

2.7 Order on functions

instantiation *fun* :: (*type*, *ord*) *ord*
begin

definition
  le-fun-def [*code func del*]: $f \le g \longleftrightarrow (\forall x.\ f\ x \le g\ x)$

definition
  less-fun-def [*code func del*]: $(f::'a \Rightarrow 'b) < g \longleftrightarrow f \le g \wedge f \ne g$

instance $\langle proof \rangle$

end

2.8 Monotonicity, least value operator and min/max

context *preorder*
begin

definition
  mono :: $('a \Rightarrow 'b::preorder) \Rightarrow bool$

where
  $mono\ f \longleftrightarrow (\forall\, x\ y.\ x \le y \longrightarrow f\, x \le f\, y)$

lemma *monoI* [*intro?*]:
  fixes $f\ ::\ 'a \Rightarrow\ 'b{::}preorder$
  shows $(\bigwedge x\ y.\ x \le y \Longrightarrow f\, x \le f\, y) \Longrightarrow mono\ f$
  ⟨*proof*⟩

lemma *monoD* [*dest?*]:
  fixes $f\ ::\ 'a \Rightarrow\ 'b{::}preorder$
  shows $mono\ f \Longrightarrow x \le y \Longrightarrow f\, x \le f\, y$
  ⟨*proof*⟩

end

context *linpreorder*
begin

lemma *min-of-mono*:
  fixes $f\ ::\ 'a \Rightarrow\ 'b\ ::\ linpreorder$
  shows $mono\ f \Longrightarrow min\ (f\ m)\ (f\ n) \sim f\ (min\ m\ n)$
⟨*proof*⟩

lemma *max-of-mono*:
  fixes $f\ ::\ 'a \Rightarrow\ 'b{::}linpreorder$
  shows $mono\ f \Longrightarrow max\ (f\ m)\ (f\ n) \sim f\ (max\ m\ n)$
  ⟨*proof*⟩

end

lemma *min-leastL*: $(!!x.\ least <= x) ==> min\ least\ x = least$
⟨*proof*⟩

lemma *max-leastL*: $(!!x.\ least <= x) ==> max\ least\ x = x$
⟨*proof*⟩

lemma *min-leastR*: $(\bigwedge x{::}'a{::}preorder.\ least \le x) \Longrightarrow min\ x\ least \sim least$
⟨*proof*⟩

lemma *max-leastR*: $(\bigwedge x{::}'a{::}preorder.\ least \le x) \Longrightarrow max\ x\ least \sim x$
⟨*proof*⟩

end

References

[1] Thomas Jech, *Set Theory*, The Third Millenium Edition, Springer 2002