Transport via Partial Galois Connections and Equivalences

Kevin Kappelmann

March 17, 2025

Abstract

This entry contains the accompanying formalisation of the paper "Transport via Partial Galois Connections and Equivalences" (APLAS 2023) [2]. It contains a theoretical framework to transport programs via equivalences, subsuming the theory of Isabelle's Lifting package [1]. It also contains a prototype to automate transports using this framework in Isabelle/HOL, but this prototype is not yet ready for production. Finally, it contains a library on top of Isabelle/HOL's axioms, including various relativised concepts on orders, functions, binary relations, and Galois connections and equivalences.

Contents

Chapter 1

HOL-Basics

theory *HOL-Basics-Base*
imports
HOL.HOL
begin

end

1.0.1 Lattice Syntax

theory *HOL-Syntax-Bundles-Lattices*
imports
HOL.Lattices
begin

open-bundle *lattice-syntax* — copied from theory Main
begin
notation *bot* (‹⊥›)
and *top* (‹⊤›)
and *inf* (**infixl** ‹⊓› *70*)
and *sup* (**infixl** ‹⊔› *65*)
end

end

1.0.2 Lattice

theory *Predicates-Lattice*
imports
HOL-Syntax-Bundles-Lattices
HOL.Boolean-Algebras
begin

lemma *inf-predI* [*intro*]:
assumes *P x*
and *Q x*

shows $(P \sqcap Q)\ x$
⟨*proof*⟩

lemma *inf-predE* [*elim*]:
assumes $(P \sqcap Q)\ x$
obtains $P\ x\ Q\ x$
⟨*proof*⟩

lemma *inf-predD*:
assumes $(P \sqcap Q)\ x$
shows $P\ x$ **and** $Q\ x$
⟨*proof*⟩

end

1.0.3 Bounded Quantifiers

theory *Bounded-Quantifiers*
imports
HOL-Basics-Base
Predicates-Lattice
ML-Unification.ML-Unification-HOL-Setup
begin

consts *ball* :: $'a \Rightarrow ('b \Rightarrow bool) \Rightarrow bool$

open-bundle *ball-syntax*
begin
syntax
-ball :: ‹[*idts*, $'a$, *bool*] ⇒ *bool*› (‹(*2*∀ - : -./ -)› *10*)
-ball2 :: ‹[*idts*, $'a$, *bool*] ⇒ *bool*›
notation *ball* (‹∀(_)›)
end

syntax-consts
-ball -ball2 ⇌ *ball*
translations
$\forall x\ xs : P.\ Q \rightharpoonup CONST\ ball\ P\ (\lambda x.\ \text{-}ball2\ xs\ P\ Q)$
-ball2 $x\ P\ Q \rightharpoonup \forall x : P.\ Q$
$\forall x : P.\ Q \rightleftharpoons CONST\ ball\ P\ (\lambda x.\ Q)$

consts *bex* :: $'a \Rightarrow ('b \Rightarrow bool) \Rightarrow bool$

open-bundle *bex-syntax*
begin
syntax
-bex :: ‹[*idts*, $'a$, *bool*] ⇒ *bool*› (‹(*2*∃ - : -./ -)› *10*)
-bex2 :: ‹[*idts*, $'a$, *bool*] ⇒ *bool*›

notation *bex* (‹∃(_)›)
end

syntax-consts
-bex -bex2 ⇌ *bex*
translations
∃ *x xs* : *P*. *Q* ⇀ *CONST bex P* (λ*x*. *-bex2 xs P Q*)
-bex2 x P Q ⇀ ∃ *x* : *P*. *Q*
∃ *x* : *P*. *Q* ⇌ *CONST bex P* (λ*x*. *Q*)

consts *bex1* :: *'a* ⇒ (*'b* ⇒ *bool*) ⇒ *bool*

open-bundle *bex1-syntax*
begin
syntax
-bex1 :: ‹[*idts*, *'a*, *bool*] ⇒ *bool*› (‹(*2*∃!- : -./ -)› *10*)
-bex12 :: ‹[*idts*, *'a*, *bool*] ⇒ *bool*›
notation *bex1* (‹∃!(_)›)
end

syntax-consts
-bex1 -bex12 ⇌ *bex1*
translations
∃!*x xs* : *P*. *Q* ⇀ *CONST bex1 P* (λ*x*. *-bex12 xs P Q*)
-bex12 x P Q ⇀ ∃!*x* : *P*. *Q*
∃!*x* : *P*. *Q* ⇌ *CONST bex1 P* (λ*x*. *Q*)

bundle *bounded-quantifier-syntax*
begin
unbundle *ball-syntax* **and** *bex-syntax* **and** *bex1-syntax*
end

definition *ball-pred P Q* ≡ ∀ *x*. *P x* ⟶ *Q x*
adhoc-overloading *ball* ⇌ *ball-pred*

definition *bex-pred P Q* ≡ ∃ *x*. *P x* ∧ *Q x*
adhoc-overloading *bex* ⇌ *bex-pred*

definition *bex1-pred P Q* ≡ ∃!*x*. *P x* ∧ *Q x*
adhoc-overloading *bex1* ⇌ *bex1-pred*

⟨*ML*⟩

lemma *ballI* [*intro*!]:
assumes ⋀*x*. *P x* ⟹ *Q x*
shows ∀ *x* : *P*. *Q x*
⟨*proof*⟩

lemma *ballE* [*elim*]:
assumes $\forall\, x : P.\ Q\ x$
obtains $\bigwedge x.\ P\ x \Longrightarrow Q\ x$
⟨*proof*⟩

lemma *ballE'*:
assumes $\forall\, x : P.\ Q\ x$
obtains $\neg(P\ x) \mid P\ x\ Q\ x$
⟨*proof*⟩

lemma *ballD*: $\forall\, x : P.\ Q\ x \Longrightarrow P\ x \Longrightarrow Q\ x$
⟨*proof*⟩

lemma *ball-cong*: $[\![P = P'; \bigwedge x.\ P'\ x \Longrightarrow Q\ x \longleftrightarrow Q'\ x]\!] \Longrightarrow (\forall\, x : P.\ Q\ x) \longleftrightarrow (\forall\, x : P'.\ Q'\ x)$
⟨*proof*⟩

lemma *ball-cong-simp* [*cong*]:
$[\![P = P'; \bigwedge x.\ P'\ x =simp=> Q\ x \longleftrightarrow Q'\ x]\!] \Longrightarrow (\forall\, x : P.\ Q\ x) \longleftrightarrow (\forall\, x : P'.\ Q'\ x)$
⟨*proof*⟩

⟨*ML*⟩

lemma *atomize-ball*: $(\bigwedge x.\ P\ x \Longrightarrow Q\ x) \equiv \mathit{Trueprop}\ (\forall\, x : P.\ Q\ x)$
⟨*proof*⟩

declare *atomize-ball*[*symmetric*, *rulify*]
declare *atomize-ball*[*symmetric*, *defn*]

lemma *bexI* [*intro*!]:

assumes $\exists\, x.\ Q\ x \wedge P\ x$
shows $\exists\, x : P.\ Q\ x$
⟨*proof*⟩

lemma *bexE* [*elim*!]:
assumes $\exists\, x : P.\ Q\ x$
obtains x **where** $P\ x\ Q\ x$
⟨*proof*⟩

lemma *bexD*:
assumes $\exists\, x : P.\ Q\ x$
shows $\exists\, x.\ P\ x \wedge Q\ x$
⟨*proof*⟩

lemma *bex-cong*:
$[\![P = P'; \bigwedge x.\ P'\ x \Longrightarrow Q\ x \longleftrightarrow Q'\ x]\!] \Longrightarrow (\exists\, x : P.\ Q\ x) \longleftrightarrow (\exists\, x : P'.\ Q'\ x)$

⟨*proof*⟩

lemma *bex-cong-simp* [*cong*]:
⟦$P = P'$; $\bigwedge x.\ P'\ x$ =*simp*=> $Q\ x \longleftrightarrow Q'\ x$⟧ ⟹ $(\exists\, x : P.\ Q\ x) \longleftrightarrow (\exists\, x : P'.\ Q'\ x)$
⟨*proof*⟩

lemma *bex1I* [*intro*]:

assumes $\exists !x.\ Q\ x \wedge P\ x$
shows $\exists !x : P.\ Q\ x$
⟨*proof*⟩

lemma *bex1D* [*dest*!]:
assumes $\exists !x : P.\ Q\ x$
shows $\exists !x.\ P\ x \wedge Q\ x$
⟨*proof*⟩

lemma *bex1-cong*: ⟦$P = P'$; $\bigwedge x.\ P\ x \Longrightarrow Q\ x \longleftrightarrow Q'\ x$⟧ ⟹ $(\exists !x : P.\ Q\ x) \longleftrightarrow (\exists !x : P'.\ Q'\ x)$
⟨*proof*⟩

lemma *bex1-cong-simp* [*cong*]:
⟦$P = P'$; $\bigwedge x.\ P\ x$ =*simp*=> $Q\ x \longleftrightarrow Q'\ x$⟧ ⟹ $(\exists !x : P.\ Q\ x) \longleftrightarrow (\exists !x : P'.\ Q'\ x)$
⟨*proof*⟩

lemma *ball-iff-ex-pred* [*iff*]: $(\forall\, x : P.\ Q) \longleftrightarrow ((\exists x.\ P\ x) \longrightarrow Q)$
⟨*proof*⟩

lemma *bex-iff-ex-and* [*iff*]: $(\exists\, x : P.\ Q) \longleftrightarrow ((\exists x.\ P\ x) \wedge Q)$
⟨*proof*⟩

lemma *ball-eq-imp-iff-imp* [*iff*]: $(\forall\, x : P.\ x = y \longrightarrow Q\ x) \longleftrightarrow (P\ y \longrightarrow Q\ y)$
⟨*proof*⟩

lemma *ball-eq-imp-iff-imp'* [*iff*]: $(\forall\, x : P.\ y = x \longrightarrow Q\ x) \longleftrightarrow (P\ y \longrightarrow Q\ y)$
⟨*proof*⟩

lemma *bex-eq-iff-pred* [*iff*]: $(\exists\, x : P.\ x = y) \longleftrightarrow P\ y$
⟨*proof*⟩

lemma *bex-eq-iff-pred'* [*iff*]: $(\exists\, x : P.\ y = x) \longleftrightarrow P\ y$
⟨*proof*⟩

lemma *bex-eq-and-iff-pred* [*iff*]: $(\exists\, x : P.\ x = y \wedge Q\ x) \longleftrightarrow P\ y \wedge Q\ y$
⟨*proof*⟩

lemma *bex-eq-and-iff-pred'* [*iff*]: $(\exists\, x : P.\ y = x \wedge Q\ x) \longleftrightarrow P\ y \wedge Q\ y$

⟨*proof*⟩

lemma *ball-and-iff-ball-and-ball*: $(\forall\, x : P.\ Q\ x \wedge U\ x) \longleftrightarrow (\forall\, x : P.\ Q\ x) \wedge (\forall\, x : P.\ U\ x)$
⟨*proof*⟩

lemma *bex-or-iff-bex-or-bex*: $(\exists\, x : P.\ Q\ x \vee U\ x) \longleftrightarrow (\exists\, x : P.\ Q\ x) \vee (\exists\, x : P.\ U\ x)$
⟨*proof*⟩

lemma *ball-or-iff-ball-or* [*iff*]: $(\forall\, x : P.\ Q\ x \vee U) \longleftrightarrow ((\forall\, x : P.\ Q\ x) \vee U)$
⟨*proof*⟩

lemma *ball-or-iff-or-ball* [*iff*]: $(\forall\, x : P.\ Q \vee U\ x) \longleftrightarrow (Q \vee (\forall\, x : P.\ U\ x))$
⟨*proof*⟩

lemma *ball-imp-iff-imp-ball* [*iff*]: $(\forall\, x : P.\ Q \longrightarrow U\ x) \longleftrightarrow (Q \longrightarrow (\forall\, x : P.\ U\ x))$
⟨*proof*⟩

lemma *bex-and-iff-bex-and* [*iff*]: $(\exists\, x : P.\ Q\ x \wedge U) \longleftrightarrow ((\exists\, x : P.\ Q\ x) \wedge U)$
⟨*proof*⟩

lemma *bex-and-iff-and-bex* [*iff*]: $(\exists\, x : P.\ Q \wedge U\ x) \longleftrightarrow (Q \wedge (\exists\, x : P.\ U\ x))$
⟨*proof*⟩

lemma *ball-imp-iff-bex-imp* [*iff*]: $(\forall\, x : P.\ Q\ x \longrightarrow U) \longleftrightarrow ((\exists\, x : P.\ Q\ x) \longrightarrow U)$
⟨*proof*⟩

lemma *not-ball-iff-bex-not* [*iff*]: $(\neg(\forall\, x : P.\ Q\ x)) \longleftrightarrow (\exists\, x : P.\ \neg(Q\ x))$
⟨*proof*⟩

lemma *not-bex-iff-ball-not* [*iff*]: $(\neg(\exists\, x : P.\ Q\ x)) \longleftrightarrow (\forall\, x : P.\ \neg(Q\ x))$
⟨*proof*⟩

lemma *bex1-iff-bex-and* [*iff*]: $(\exists!x : P.\ Q) \longleftrightarrow ((\exists!x.\ P\ x) \wedge Q)$
⟨*proof*⟩

lemma *ball-bottom-eq-top* [*simp*]: $\forall_{\bot} = \top$ ⟨*proof*⟩
lemma *bex-bottom-eq-bottom* [*simp*]: $\exists_{\bot} = \bot$ ⟨*proof*⟩
lemma *bex1-bottom-eq-bottom* [*simp*]: $\exists!_{\bot} = \bot$ ⟨*proof*⟩

lemma *ball-top-eq-all* [*simp*]: $\forall_{\top} = All$ ⟨*proof*⟩

lemma *ball-top-eq-all-uhint* [*uhint*]:
assumes $P \equiv (\top :: 'a \Rightarrow bool)$
shows $\forall_P \equiv All$
⟨*proof*⟩

lemma *bex-top-eq-ex* [*simp*]: $\exists_{\top} = Ex$ ⟨*proof*⟩

lemma *bex-top-eq-ex-uhint* [*uhint*]:
assumes $P \equiv (\top :: 'a \Rightarrow bool)$
shows $\exists_P \equiv Ex$
⟨*proof*⟩

lemma *bex1-top-eq-ex1* [*simp*]: $\exists!_{\top} = Ex1$ ⟨*proof*⟩

lemma *bex1-top-eq-ex1-uhint* [*uhint*]:
assumes $P \equiv (\top :: 'a \Rightarrow bool)$
shows $\exists!_P \equiv Ex1$
⟨*proof*⟩

end

1.1 Binary Relations

1.1.1 Basic Functions

theory *Binary-Relation-Functions*
imports
Bounded-Quantifiers
ML-Unification.Unify-Resolve-Tactics
begin

Summary Basic functions on binary relations.

Domain, Codomain, and Field

definition *in-dom* $R\ x \equiv \exists y.\ R\ x\ y$

lemma *in-domI* [*intro*]:
assumes $R\ x\ y$
shows *in-dom* $R\ x$
⟨*proof*⟩

lemma *in-domE* [*elim*]:
assumes *in-dom* $R\ x$
obtains y **where** $R\ x\ y$
⟨*proof*⟩

lemma *in-dom-bottom-eq-bottom* [*simp*]: *in-dom* $\bot = \bot$ ⟨*proof*⟩
lemma *in-dom-top-eq-top* [*simp*]: *in-dom* $\top = \top$ ⟨*proof*⟩

lemma *in-dom-sup-eq-in-dom-sup-in-dom* [*simp*]: *in-dom* $(R \sqcup S) =$ *in-dom* $R \sqcup$ *in-dom* S ⟨*proof*⟩

consts *in-codom-on* :: $'a \Rightarrow 'b \Rightarrow 'c \Rightarrow bool$

definition *in-codom-on-pred* $(P :: 'a \Rightarrow bool)\ R\ y \equiv \exists x : P.\ R\ x\ y$
adhoc-overloading *in-codom-on* $\rightleftharpoons$ *in-codom-on-pred*

lemma *in-codom-onI* [*intro*]:
assumes $R\ x\ y$
and $P\ x$
shows *in-codom-on* $P\ R\ y$
⟨*proof*⟩

lemma *in-codom-onE* [*elim*]:
assumes *in-codom-on* $P\ R\ y$
obtains x **where** $P\ x\ R\ x\ y$
⟨*proof*⟩

lemma *in-codom-on-pred-iff-bex-rel*: *in-codom-on* $P\ R\ y \longleftrightarrow (\exists x : P.\ R\ x\ y)$
⟨*proof*⟩

lemma *in-codom-bottom-pred-eq-bottom* [*simp*]: *in-codom-on* $\bot = \bot$ ⟨*proof*⟩
lemma *in-codom-bottom-rel-eq-bottom* [*simp*]: *in-codom-on* $P \bot = \bot$ ⟨*proof*⟩
lemma *in-codom-top-top-eq* [*simp*]: *in-codom-on* $\top\ \top = \top$ ⟨*proof*⟩

lemma *in-codom-on-eq-pred-eq* [*simp*]: *in-codom-on* $((=)\ P)\ R = R\ P$
⟨*proof*⟩

lemma *in-codom-on-sup-pred-eq-in-codom-on-sup-in-codom-on* [*simp*]:
in-codom-on $(P \sqcup Q) =$ *in-codom-on* $P \sqcup$ *in-codom-on* Q
⟨*proof*⟩

lemma *in-codom-on-sup-rel-eq-in-codom-on-sup-in-codom-on* [*simp*]:
in-codom-on $P\ (R \sqcup S) =$ *in-codom-on* $P\ R \sqcup$ *in-codom-on* $P\ S$
⟨*proof*⟩

definition *in-codom* $\equiv$ *in-codom-on* $(\top :: 'a \Rightarrow bool)$

lemma *in-codom-eq-in-codom-on-top*: *in-codom* $=$ *in-codom-on* $\top$ ⟨*proof*⟩

lemma *in-codom-eq-in-codom-on-top-uhint* [*uhint*]:
assumes $P \equiv (\top :: 'a \Rightarrow bool)$
shows *in-codom* $\equiv$ *in-codom-on* P
⟨*proof*⟩

lemma *in-codomI* [*intro*]:
assumes $R\ x\ y$
shows *in-codom* $R\ y$
⟨*proof*⟩

lemma *in-codomE* [*elim*]:

assumes *in-codom R y*
obtains *x* **where** *R x y*
⟨*proof*⟩

definition *in-field R x* ≡ *in-dom R x* ∨ *in-codom R x*

lemma *in-field-if-in-dom*:
assumes *in-dom R x*
shows *in-field R x*
⟨*proof*⟩

lemma *in-field-if-in-codom*:
assumes *in-codom R x*
shows *in-field R x*
⟨*proof*⟩

lemma *in-fieldE* [*elim*]:
assumes *in-field R x*
obtains (*in-dom*) *x′* **where** *R x x′* | (*in-codom*) *x′* **where** *R x′ x*
⟨*proof*⟩

lemma *in-fieldE′*:
assumes *in-field R x*
obtains (*in-dom*) *in-dom R x* | (*in-codom*) *in-codom R x*
⟨*proof*⟩

lemma *in-fieldI* [*intro*]:
assumes *R x y*
shows *in-field R x in-field R y*
⟨*proof*⟩

lemma *in-field-iff-in-dom-or-in-codom*:
in-field R x ⟷ *in-dom R x* ∨ *in-codom R x*
⟨*proof*⟩

lemma *in-field-eq-in-dom-if-in-codom-eq-in-dom*:
assumes *in-codom R* = *in-dom R*
shows *in-field R* = *in-dom R*
⟨*proof*⟩

lemma *in-field-bottom-eq-bottom* [*simp*]: *in-field* ⊥ = ⊥ ⟨*proof*⟩
lemma *in-field-top-eq-top* [*simp*]: *in-field* ⊤ = ⊤ ⟨*proof*⟩

lemma *in-field-sup-eq-in-field-sup-in-field* [*simp*]: *in-field* (*R* ⊔ *S*) = *in-field R* ⊔ *in-field S*
⟨*proof*⟩

Composition

consts *rel-comp* :: $'a \Rightarrow 'b \Rightarrow 'c$

open-bundle *rel-comp-syntax*
begin
notation *rel-comp* (**infixl** ‹∘∘› *55*)
end

definition *rel-comp-rel* $R\ S\ x\ y \equiv \exists z.\ R\ x\ z \wedge S\ z\ y$
adhoc-overloading *rel-comp* $\rightleftharpoons$ *rel-comp-rel*

lemma *rel-compI* [*intro*]:
  assumes $R\ x\ y$
  and $S\ y\ z$
  shows $(R \circ\circ S)\ x\ z$
  ⟨*proof*⟩

lemma *rel-compE* [*elim*]:
  assumes $(R \circ\circ S)\ x\ y$
  obtains z **where** $R\ x\ z\ S\ z\ y$
  ⟨*proof*⟩

lemma *rel-comp-assoc*: $R \circ\circ (S \circ\circ T) = (R \circ\circ S) \circ\circ T$
  ⟨*proof*⟩

lemma *in-dom-if-in-dom-rel-comp*:
  assumes *in-dom* $(R \circ\circ S)\ x$
  shows *in-dom* $R\ x$
  ⟨*proof*⟩

lemma *in-codom-if-in-codom-rel-comp*:
  assumes *in-codom* $(R \circ\circ S)\ y$
  shows *in-codom* $S\ y$
  ⟨*proof*⟩

lemma *in-field-compE* [*elim*]:
  assumes *in-field* $(R \circ\circ S)\ x$
  obtains (*in-dom*) *in-dom* $R\ x$ | (*in-codom*) *in-codom* $S\ x$
  ⟨*proof*⟩

Inverse

consts *rel-inv* :: $'a \Rightarrow 'b$

open-bundle *rel-inv-syntax*
begin
notation *rel-inv* (‹(-$^{-1}$)› [*1000*])
end

definition *rel-inv-rel* :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow 'b \Rightarrow 'a \Rightarrow bool$
  where *rel-inv-rel* $R\ x\ y \equiv R\ y\ x$
adhoc-overloading *rel-inv* $\rightleftharpoons$ *rel-inv-rel*

lemma *rel-invI* [*intro*]:
  assumes $R\ x\ y$
  shows $R^{-1}\ y\ x$
  ⟨*proof*⟩

lemma *rel-invD* [*dest*]:
  assumes $R^{-1}\ x\ y$
  shows $R\ y\ x$
  ⟨*proof*⟩

lemma *rel-inv-iff-rel* [*simp*]: $R^{-1}\ x\ y \longleftrightarrow R\ y\ x$
  ⟨*proof*⟩

lemma *in-dom-rel-inv-eq-in-codom* [*simp*]: *in-dom* $R^{-1} =$ *in-codom* R
  ⟨*proof*⟩

lemma *in-codom-rel-inv-eq-in-dom* [*simp*]: *in-codom* $R^{-1} =$ *in-dom* R
  ⟨*proof*⟩

lemma *in-field-rel-inv-eq-in-field* [*simp*]: *in-field* $R^{-1} =$ *in-field* R
  ⟨*proof*⟩

lemma *rel-inv-comp-eq* [*simp*]: $(R \circ\circ S)^{-1} = S^{-1} \circ\circ R^{-1}$
  ⟨*proof*⟩

lemma *rel-inv-inv-eq-self* [*simp*]: $R^{-1-1} = R$
  ⟨*proof*⟩

lemma *rel-inv-eq-iff-eq* [*iff*]: $R^{-1} = S^{-1} \longleftrightarrow R = S$
  ⟨*proof*⟩

lemma *rel-inv-le-rel-inv-iff* [*iff*]: $R^{-1} \leq S^{-1} \longleftrightarrow R \leq S$
  ⟨*proof*⟩

lemma *rel-inv-top-eq* [*simp*]: $\top^{-1} = \top$ ⟨*proof*⟩
lemma *rel-inv-bottom-eq* [*simp*]: $\bot^{-1} = \bot$ ⟨*proof*⟩

Restrictions

consts *rel-if* :: $bool \Rightarrow 'a \Rightarrow 'a$

open-bundle *rel-if-syntax*
begin
notation (**output**) *rel-if* (**infixl** ‹⟶› *50*)
end

definition *rel-if-rel B R x y* ≡ *B* ⟶ *R x y*
adhoc-overloading *rel-if* ⇌ *rel-if-rel*

lemma *rel-if-eq-rel-if-pred* [*simp*]:
assumes *B*
shows (*rel-if B R*) = *R*
⟨*proof*⟩

lemma *rel-if-eq-top-if-not-pred* [*simp*]:
assumes ¬*B*
shows (*rel-if B R*) = (λ- -. *True*)
⟨*proof*⟩

lemma *rel-if-if-impI* [*intro*]:
assumes *B* ⟹ *R x y*
shows (*rel-if B R*) *x y*
⟨*proof*⟩

lemma *rel-ifE* [*elim*]:
assumes (*rel-if B R*) *x y*
obtains ¬*B* | *B R x y*
⟨*proof*⟩

lemma *rel-ifD*:
assumes (*rel-if B R*) *x y*
and *B*
shows *R x y*
⟨*proof*⟩

consts *rel-restrict-left* :: *'a* ⇒ *'b* ⇒ *'a*
consts *rel-restrict-right* :: *'a* ⇒ *'b* ⇒ *'a*
consts *rel-restrict* :: *'a* ⇒ *'b* ⇒ *'a*

open-bundle *rel-restrict-syntax*
begin
notation *rel-restrict-left* (‹(-)⌈(_)› [*1000*])
notation *rel-restrict-right* (‹(-)⌉(_)› [*1000*])
notation *rel-restrict* (‹(-)↑(_)› [*1000*])
end

definition *rel-restrict-left-pred R P x y* ≡ *P x* ∧ *R x y*
adhoc-overloading *rel-restrict-left* ⇌ *rel-restrict-left-pred*
definition *rel-restrict-right-pred R P x y* ≡ *P y* ∧ *R x y*
adhoc-overloading *rel-restrict-right* ⇌ *rel-restrict-right-pred*
definition *rel-restrict-pred R P* ≡ *R*⌈*P*⌉*P*
adhoc-overloading *rel-restrict* ⇌ *rel-restrict-pred*

lemma *rel-restrict-leftI* [*intro*]:

assumes $R\ x\ y$
and $P\ x$
shows $R{\restriction_P}\ x\ y$
⟨*proof*⟩

lemma *rel-restrict-leftE* [*elim*]:
assumes $R{\restriction_P}\ x\ y$
obtains $P\ x\ R\ x\ y$
⟨*proof*⟩

lemma *rel-restrict-left-cong*:
assumes $\bigwedge x.\ P\ x \longleftrightarrow P'\ x$
and $\bigwedge x\ y.\ P'\ x \Longrightarrow R\ x\ y \longleftrightarrow R'\ x\ y$
shows $R{\restriction_P} = R'{\restriction_{P'}}$
⟨*proof*⟩

lemma *rel-restrict-left-restrict-left-eq-restrict-left* [*simp*]: $R{\restriction_P}{\restriction_P} = R{\restriction_P}$
⟨*proof*⟩

lemma *rel-restrict-left-top-eq* [*simp*]: $(R :: 'a \Rightarrow 'b \Rightarrow bool){\restriction_{\top\ ::\ 'a \Rightarrow bool}} = R$
⟨*proof*⟩

lemma *rel-restrict-left-top-eq-uhint* [*uhint*]:
assumes $P \equiv (\top :: 'a \Rightarrow bool)$
shows $(R :: 'a \Rightarrow 'b \Rightarrow bool){\restriction_P} \equiv R$
⟨*proof*⟩

lemma *rel-restrict-left-bottom-eq* [*simp*]: $(R :: 'a \Rightarrow 'b \Rightarrow bool){\restriction_{\bot\ ::\ 'a \Rightarrow bool}} = \bot$
⟨*proof*⟩

lemma *bottom-restrict-left-eq* [*simp*]: $\bot{\restriction_{P\ ::\ 'a \Rightarrow bool}} = \bot$
⟨*proof*⟩

lemma *rel-restrict-left-le-self*: $(R :: 'a \Rightarrow 'b \Rightarrow bool){\restriction_{(P\ ::\ 'a \Rightarrow bool)}} \leq R$
⟨*proof*⟩

lemma *le-rel-restrict-left-self-if-in-dom-le*:
assumes *in-dom* $R \leq P$
shows $R \leq (R :: 'a \Rightarrow 'b \Rightarrow bool){\restriction_{(P\ ::\ 'a \Rightarrow bool)}}$
⟨*proof*⟩

corollary *rel-restrict-left-eq-self-if-in-dom-le* [*simp*]:
assumes *in-dom* $R \leq P$
shows $R{\restriction_P} = R$
⟨*proof*⟩

lemma *ex-rel-restrict-left-iff-in-codom-on* [*iff*]: $(\exists\, x.\ R{\restriction_P}\ x\ y) \longleftrightarrow (in\text{-}codom\text{-}on\ P\ R\ y)$ ⟨*proof*⟩

lemma *in-dom-rel-restrict-leftI* [*intro*]:
assumes $R\ x\ y$
and $P\ x$
shows *in-dom* $R{\restriction}_P\ x$
⟨*proof*⟩

lemma *in-codom-rel-restrict-leftI* [*intro*]:
assumes $R\ x\ y$
and $P\ x$
shows *in-codom* $R{\restriction}_P\ y$
⟨*proof*⟩

lemma *rel-restrict-rightI* [*intro*]:
assumes $R\ x\ y$
and $P\ y$
shows $R{\restriction}_P\ x\ y$
⟨*proof*⟩

lemma *rel-restrict-rightE* [*elim*]:
assumes $R{\restriction}_P\ x\ y$
obtains $P\ y\ R\ x\ y$
⟨*proof*⟩

lemma *rel-restrict-right-cong*:
assumes $\bigwedge y.\ P\ y \longleftrightarrow P'\ y$
and $\bigwedge x\ y.\ P'\ y \Longrightarrow R\ x\ y \longleftrightarrow R'\ x\ y$
shows $R{\restriction}_P = R'{\restriction}_{P'}$
⟨*proof*⟩

lemma *rel-restrict-right-restrict-right-eq-restrict-right* [*simp*]: $R{\restriction}_P{\restriction}_P = R{\restriction}_P$
⟨*proof*⟩

lemma *rel-restrict-right-top-eq* [*simp*]: $(R :: 'a \Rightarrow 'b \Rightarrow bool){\restriction}_{\top :: 'b \Rightarrow bool} = R$
⟨*proof*⟩

lemma *rel-restrict-right-top-eq-uhint* [*uhint*]:
assumes $P \equiv (\top :: 'b \Rightarrow bool)$
shows $(R :: 'a \Rightarrow 'b \Rightarrow bool){\restriction}_P \equiv R$
⟨*proof*⟩

lemma *rel-restrict-right-bottom-eq* [*simp*]: $(R :: 'a \Rightarrow 'b \Rightarrow bool){\restriction}_{\bot\ ::\ 'b \Rightarrow bool} = \bot$
⟨*proof*⟩

lemma *bottom-restrict-right-eq* [*simp*]: $\bot{\restriction}_{P\ ::\ 'b \Rightarrow bool} = \bot$
⟨*proof*⟩

lemma *rel-restrict-right-le-self*: $(R :: 'a \Rightarrow 'b \Rightarrow bool){\restriction}_{(P\ ::\ 'b \Rightarrow bool)} \leq R$

⟨*proof*⟩

lemma *le-rel-restrict-right-self-if-in-codom-le*:
assumes *in-codom* $R \leq P$
shows $R \leq (R :: 'a \Rightarrow 'b \Rightarrow bool)\restriction_{(P :: 'b \Rightarrow bool)}$
⟨*proof*⟩

corollary *rel-restrict-right-eq-self-if-in-codom-le* [*simp*]:
assumes *in-codom* $R \leq P$
shows $R\restriction_P = R$
⟨*proof*⟩

context
fixes $R :: 'a \Rightarrow 'b \Rightarrow bool$
begin

lemma *rel-restrict-right-eq*: $R\restriction_{P :: 'b \Rightarrow bool} = ((R^{-1})\restriction^P)^{-1}$
⟨*proof*⟩

lemma *rel-inv-restrict-right-rel-inv-eq-restrict-left* [*simp*]: $((R^{-1})\restriction_{P :: 'a \Rightarrow bool})^{-1}$
$= R\restriction^P$
⟨*proof*⟩

lemma *rel-restrict-right-iff-restrict-left*: $R\restriction_{P :: 'b \Rightarrow bool}\ x\ y \longleftrightarrow (R^{-1})\restriction^P\ y\ x$
⟨*proof*⟩

end

lemma *rel-inv-rel-restrict-left-inv-rel-restrict-left-eq*:
fixes $R :: 'a \Rightarrow 'b \Rightarrow bool$ **and** $P :: 'a \Rightarrow bool$ **and** $Q :: 'b \Rightarrow bool$
shows $(((R\restriction^P)^{-1})\restriction^Q)^{-1} = (((R^{-1})\restriction^Q)^{-1})\restriction^P$
⟨*proof*⟩

lemma *rel-restrict-left-right-eq-restrict-right-left*:
fixes $R :: 'a \Rightarrow 'b \Rightarrow bool$ **and** $P :: 'a \Rightarrow bool$ **and** $Q :: 'b \Rightarrow bool$
shows $R\restriction^P\restriction_Q = R\restriction_Q\restriction^P$
⟨*proof*⟩

lemma *rel-restrictI* [*intro*]:
assumes $R\restriction^P\restriction_P\ x\ y$
shows $R\uparrow_P\ x\ y$
⟨*proof*⟩

lemma *rel-restrictD* [*dest*!]:
assumes $R\uparrow_P\ x\ y$
shows $R\restriction^P\restriction_P\ x\ y$
⟨*proof*⟩

lemma *rel-restrict-eq-restrict-left-right* [*simp*]: $R\uparrow_P = R\restriction^P\restriction_P$

⟨*proof*⟩

lemma *rel-restrict-cong*:
assumes $\bigwedge y.\ P\ y \longleftrightarrow P'\ y$
and $\bigwedge x\ y.\ P'\ x \Longrightarrow P'\ y \Longrightarrow R\ x\ y \longleftrightarrow R'\ x\ y$
shows $R{\uparrow}_P = R'{\uparrow}_{P'}$
⟨*proof*⟩

lemma *rel-restrict-restrict-eq-restrict* [*simp*]: $R{\uparrow}_P{\uparrow}_P = R{\uparrow}_P$
⟨*proof*⟩

lemma *rel-restrict-top-eq* [*simp*]: $(R :: 'a \Rightarrow 'a \Rightarrow bool){\uparrow}_{\top\ ::\ 'a \Rightarrow bool} = R$
⟨*proof*⟩

lemma *rel-restrict-top-eq-uhint* [*uhint*]:
assumes $P \equiv (\top ::'a \Rightarrow bool)$
shows $(R :: 'a \Rightarrow 'a \Rightarrow bool){\uparrow}_P \equiv R$
⟨*proof*⟩

lemma *rel-restrict-bottom-eq* [*simp*]: $(R :: 'a \Rightarrow 'a \Rightarrow bool){\uparrow}_{\bot\ ::\ 'a \Rightarrow bool} = \bot$
⟨*proof*⟩

lemma *bottom-restrict-eq* [*simp*]: $\bot{\uparrow}_{P\ ::\ 'a \Rightarrow bool} = \bot$
⟨*proof*⟩

lemma *rel-restrict-le-self*: $(R :: 'a \Rightarrow 'a \Rightarrow bool){\uparrow}_{(P\ ::\ 'a \Rightarrow bool)} \leq R$
⟨*proof*⟩

lemma *le-rel-restrict-self-if-in-field-le*:
assumes *in-field* $R \leq P$
shows $R \leq (R :: 'a \Rightarrow 'a \Rightarrow bool){\uparrow}_{(P\ ::\ 'a \Rightarrow bool)}$
⟨*proof*⟩

corollary *rel-restrict-eq-self-if-in-field-le* [*simp*]:
assumes *in-field* $R \leq P$
shows $R{\uparrow}_P = R$
⟨*proof*⟩

Mappers

definition *rel-bimap f g* $(R :: 'a \Rightarrow 'b \Rightarrow bool)\ x\ y \equiv R\ (f\ x)\ (g\ y)$

lemma *rel-bimap-eq* [*simp*]: *rel-bimap f g R x y* $= R\ (f\ x)\ (g\ y)$
⟨*proof*⟩

definition *rel-map f R* $\equiv$ *rel-bimap f f R*

lemma *rel-bimap-self-eq-rel-map* [*simp*]: *rel-bimap f f R* $=$ *rel-map f R*
⟨*proof*⟩

lemma *rel-map-eq* [*simp*]: *rel-map f R x y* = *R* (*f x*) (*f y*)
⟨*proof*⟩

end

1.1.2 Order

theory *Binary-Relations-Order-Base*
imports
Binary-Relation-Functions
HOL.Orderings
begin

lemma *le-relI* [*intro*]:
assumes ⋀*x y*. *R x y* ⟹ *S x y*
shows *R* ≤ *S*
⟨*proof*⟩

lemma *le-relD* [*dest*]:
assumes *R* ≤ *S*
and *R x y*
shows *S x y*
⟨*proof*⟩

lemma *le-relE*:
assumes *R* ≤ *S*
obtains ⋀*x y*. *R x y* ⟹ *S x y*
⟨*proof*⟩

end

1.2 Functions

1.2.1 Basic Functions

theory *Functions-Base*
imports
Binary-Relation-Functions
begin

definition *id x* ≡ *x*

lemma *id-eq-self* [*simp*]: *id x* = *x*
⟨*proof*⟩

lemma *rel-map-id-eq-self* [*simp*]: *rel-map id R* = *R* ⟨*proof*⟩

consts *comp* :: *'a* ⇒ *'b* ⇒ *'c*

open-bundle *comp-syntax*
begin
notation *comp* (**infixl** ‹○› *55*)
end

definition *comp-fun f g x* ≡ *f* (*g x*)
adhoc-overloading *comp* ⇌ *comp-fun*

lemma *comp-eq* [*simp*]: (*f* ∘ *g*) *x* = *f* (*g x*)
⟨*proof*⟩

lemma *id-comp-eq* [*simp*]: *id* ∘ *f* = *f*
⟨*proof*⟩

lemma *comp-id-eq* [*simp*]: *f* ∘ *id* = *f*
⟨*proof*⟩

lemma *bottom-comp-eq* [*simp*]: ⊥ ∘ *f* = ⊥ ⟨*proof*⟩
lemma *top-comp-eq* [*simp*]: ⊤ ∘ *f* = ⊤ ⟨*proof*⟩

definition *dep-fun-map f g h x* ≡ *g x* (*f x*) (*h* (*f x*))

definition *fun-map f g h* ≡ *dep-fun-map f* (λ- -. *g*) *h*

open-bundle *dep-fun-map-syntax*
begin
notation *fun-map* (**infixr** ‹⇝› *40*)
syntax
-*dep-fun-map* :: *idt* ⇒ ('*a* ⇒ '*b*) ⇒ ('*c* ⇒ '*d*) ⇒ ('*b* ⇒ '*c*) ⇒
('*a* ⇒ '*d*) (‹'(-/ : / -') ⇝ (-)› [*41*, *41*, *40*] *40*)
end
syntax-consts -*dep-fun-map* ⇌ *dep-fun-map*
translations
(*x* : *f*) ⇝ *g* ⇌ *CONST dep-fun-map f* (λ*x*. *g*)

lemma *fun-map-eq-dep-fun-map*: (*f* ⇝ *g*) = ((- : *f*) ⇝ (λ-. *g*))
⟨*proof*⟩

lemma *fun-map-eq-dep-fun-map-uhint* [*uhint*]:
assumes *f* ≡ *f'*
and *g'* ≡ (λ- -. *g*)
shows (*f* ⇝ *g*) = ((*x* : *f'*) ⇝ *g' x*)
⟨*proof*⟩

lemma *dep-fun-map-eq* [*simp*]: ((*x* : *f*) ⇝ *g x*) *h x* = *g x* (*f x*) (*h* (*f x*))
⟨*proof*⟩

lemma *fun-map-eq-comp* [*simp*]: $(f \rightsquigarrow g)\ h = g \circ h \circ f$
⟨*proof*⟩

lemma *fun-map-eq* [*simp*]: $(f \rightsquigarrow g)\ h\ x = g\ (h\ (f\ x))$
⟨*proof*⟩

lemma *fun-map-id-eq-comp* [*simp*]: *fun-map id* $= (\circ)$
⟨*proof*⟩

lemma *fun-map-id-eq-comp′* [*simp*]: $(f \rightsquigarrow id)\ h = h \circ f$
⟨*proof*⟩

consts *has-inverse-on* :: $'a \Rightarrow 'b \Rightarrow 'c \Rightarrow bool$

definition *has-inverse-on-pred* $(P :: 'a \Rightarrow bool)\ f \equiv$ *in-codom-on* $P\ ((=) \circ f)$
adhoc-overloading *has-inverse-on* $\rightleftharpoons$ *has-inverse-on-pred*

lemma *has-inverse-on-pred-eq-in-codom-on*: *has-inverse-on* $P =$ *in-codom-on* $P \circ ((\circ)\ (=))$
⟨*proof*⟩

lemma *has-inverse-onI* [*intro*]:
assumes $P\ x$
shows *has-inverse-on* $P\ f\ (f\ x)$
⟨*proof*⟩

lemma *has-inverse-on-if-pred-if-eq*:
assumes $y = f\ x$
and $P\ x$
shows *has-inverse-on* $P\ f\ y$
⟨*proof*⟩

lemma *has-inverse-onE* [*elim*]:
assumes *has-inverse-on* $P\ f\ y$
obtains x **where** $P\ x\ y = f\ x$
⟨*proof*⟩

context
notes *has-inverse-on-pred-eq-in-codom-on*[*simp*]
begin

lemma *has-inverse-on-bottom-eq* [*simp*]: *has-inverse-on* $\bot = \bot$
⟨*proof*⟩

lemma *has-inverse-on-eq-pred-eq-eq-app* [*simp*]: *has-inverse-on* $((=)\ P)\ f = (=)\ (f\ P)$
⟨*proof*⟩

lemma *has-inverse-on-sup-eq-has-inverse-on-sup-has-inverse-on* [*simp*]:
has-inverse-on $(P \sqcup Q)$ = *has-inverse-on* $P \sqcup$ *has-inverse-on* Q
⟨*proof*⟩

end

definition *has-inverse* ≡ *has-inverse-on* ⊤

lemma *has-inverse-eq-has-inverse-on-top*: *has-inverse* = *has-inverse-on* ⊤
⟨*proof*⟩

lemma *has-inverse-eq-has-inverse-on-top-uhint* [*uhint*]:
assumes $P \equiv \top$
shows *has-inverse* ≡ *has-inverse-on* P
⟨*proof*⟩

lemma *has-inverse-iff-has-inverse-on-top*: *has-inverse* f $y \longleftrightarrow$ *has-inverse-on* ⊤ f y
⟨*proof*⟩

lemma *has-inverseI* [*intro*]:
assumes $y = f\ x$
shows *has-inverse* f y
⟨*proof*⟩

lemma *has-inverseE* [*elim*]:
assumes *has-inverse* f y
obtains x **where** $y = f\ x$
⟨*proof*⟩

end

1.2.2 Relators

theory *Function-Relators*
imports
Binary-Relation-Functions
Functions-Base
Predicates-Lattice
ML-Unification.ML-Unification-HOL-Setup
ML-Unification.Unify-Resolve-Tactics
begin

Summary Introduces the concept of function relators. The slogan of function relators is "related functions map related inputs to related outputs".

consts *Dep-Fun-Rel* :: $'a \Rightarrow 'b \Rightarrow 'c \Rightarrow 'd \Rightarrow$ *bool*
consts *Fun-Rel* :: $'a \Rightarrow 'b \Rightarrow 'c \Rightarrow 'd \Rightarrow$ *bool*

open-bundle *Dep-Fun-Rel-syntax*

begin
notation *Fun-Rel* (**infixr** ‹⇛› *50*)
syntax
-Dep-Fun-Rel-rel :: *idt* ⇒ *idt* ⇒ *'a* ⇒ *'b* ⇒ *'c* ⇒ *'d* ⇒ *bool*
(‹*'*(-/ -/ ::/ -*'*) ⇛ (-)› [*51*, *51*, *50*, *50*] *50*)
-Dep-Fun-Rel-rel-if :: *idt* ⇒ *idt* ⇒ *'a* ⇒ *bool* ⇒ *'b* ⇒ *'c* ⇒ *'d* ⇒ *bool*
(‹*'*(-/ -/ ::/ -/ |/ -*'*) ⇛ (-)› [*51*, *51*, *50*, *50*, *50*] *50*)
-Dep-Fun-Rel-pred :: *idt* ⇒ *'a* ⇒ *'b* ⇒ *'c* ⇒ *'d* ⇒ *bool*
(‹*'*(-/ :/ -*'*) ⇛ (-)› [*51*, *50*, *50*] *50*)
-Dep-Fun-Rel-pred-if :: *idt* ⇒ *'a* ⇒ *bool* ⇒ *'b* ⇒ *'c* ⇒ *'d* ⇒ *bool*
(‹*'*(-/ :/ -/ |/ -*'*) ⇛ (-)› [*51*, *50*, *50*, *50*] *50*)
end
syntax-consts
-Dep-Fun-Rel-rel -Dep-Fun-Rel-rel-if -Dep-Fun-Rel-pred -Dep-Fun-Rel-pred-if ⇌ *Dep-Fun-Rel*
translations
(*x y* :: *R*) ⇛ *S* ⇌ *CONST Dep-Fun-Rel R* (λ*x y*. *S*)
(*x y* :: *R* | *B*) ⇛ *S* ⇌ *CONST Dep-Fun-Rel R* (λ*x y*. *CONST rel-if B S*)
(*x* : *P*) ⇛ *R* ⇌ *CONST Dep-Fun-Rel P* (λ*x*. *R*)
(*x* : *P* | *B*) ⇛ *R* ⇌ *CONST Dep-Fun-Rel P* (λ*x*. *CONST rel-if B R*)

definition *Dep-Fun-Rel-rel R S f g* ≡ ∀ *x y*. *R x y* ⟶ *S x y* (*f x*) (*g y*)
adhoc-overloading *Dep-Fun-Rel* ⇌ *Dep-Fun-Rel-rel*

definition *Fun-Rel-rel* (*R* :: *'a* ⇒ *'b* ⇒ *bool*) (*S* :: *'c* ⇒ *'d* ⇒ *bool*) ≡
((- - :: *R*) ⇛ *S*) :: (*'a* ⇒ *'c*) ⇒ (*'b* ⇒ *'d*) ⇒ *bool*
adhoc-overloading *Fun-Rel* ⇌ *Fun-Rel-rel*

definition *Dep-Fun-Rel-pred P R f g* ≡ ∀ *x*. *P x* ⟶ *R x* (*f x*) (*g x*)
adhoc-overloading *Dep-Fun-Rel* ⇌ *Dep-Fun-Rel-pred*

definition *Fun-Rel-pred* (*P* :: *'a* ⇒ *bool*) (*R* :: *'b* ⇒ *'c* ⇒ *bool*) ≡
((- : *P*) ⇛ *R*) :: (*'a* ⇒ *'b*) ⇒ (*'a* ⇒ *'c*) ⇒ *bool*
adhoc-overloading *Fun-Rel* ⇌ *Fun-Rel-pred*

lemma *Fun-Rel-rel-eq-Dep-Fun-Rel-rel*:
((*R* :: *'a* ⇒ *'b* ⇒ *bool*) ⇛ (*S* :: *'c* ⇒ *'d* ⇒ *bool*)) = ((- - :: *R*) ⇛ *S*)
⟨*proof*⟩

lemma *Fun-Rel-rel-eq-Dep-Fun-Rel-rel-uhint* [*uhint*]:
assumes *R* ≡ *R'*
and *S'* ≡ (λ(- :: *'a*) (- :: *'b*). *S*)
shows ((*R* :: *'a* ⇒ *'b* ⇒ *bool*) ⇛ (*S* :: *'c* ⇒ *'d* ⇒ *bool*)) = ((*x y* :: *R'*) ⇛ *S' x y*)
⟨*proof*⟩

lemma *Fun-Rel-rel-iff-Dep-Fun-Rel-rel*:
((*R* :: *'a* ⇒ *'b* ⇒ *bool*) ⇛ (*S* :: *'c* ⇒ *'d* ⇒ *bool*)) *f g* ⟷ ((- - :: *R*) ⇛ *S*) *f g*
⟨*proof*⟩

lemma *Fun-Rel-pred-eq-Dep-Fun-Rel-pred*:
$((P :: 'a \Rightarrow bool) \Rrightarrow (R :: 'b \Rightarrow 'c \Rightarrow bool)) = ((\text{-} : P) \Rrightarrow R)$
⟨*proof*⟩

lemma *Fun-Rel-pred-eq-Dep-Fun-Rel-pred-uhint* [*uhint*]:
assumes $P \equiv P'$
and $R' \equiv (\lambda(\text{-} :: 'a).\ R)$
shows $((P :: 'a \Rightarrow bool) \Rrightarrow (R :: 'b \Rightarrow 'c \Rightarrow bool)) = ((x : P') \Rrightarrow R'\ x)$
⟨*proof*⟩

lemma *Fun-Rel-pred-iff-Dep-Fun-Rel-pred*:
$((P :: 'a \Rightarrow bool) \Rrightarrow (R :: 'b \Rightarrow 'c \Rightarrow bool))\ f\ g \longleftrightarrow ((\text{-} : P) \Rrightarrow R)\ (f :: 'a \Rightarrow 'b)$
$(g :: 'a \Rightarrow 'c)$
⟨*proof*⟩

lemma *Dep-Fun-Rel-relI* [*intro*]:
assumes $\bigwedge x\ y.\ R\ x\ y \Longrightarrow S\ x\ y\ (f\ x)\ (g\ y)$
shows $((x\ y :: R) \Rrightarrow S\ x\ y)\ f\ g$
⟨*proof*⟩

lemma *Dep-Fun-Rel-relD* [*dest*]:
assumes $((x\ y :: R) \Rrightarrow S\ x\ y)\ f\ g$
and $R\ x\ y$
shows $S\ x\ y\ (f\ x)\ (g\ y)$
⟨*proof*⟩

lemma *Dep-Fun-Rel-relE*:
assumes $((x\ y :: R) \Rrightarrow S\ x\ y)\ f\ g$
obtains $\bigwedge x\ y.\ R\ x\ y \Longrightarrow S\ x\ y\ (f\ x)\ (g\ y)$
⟨*proof*⟩

lemma *Dep-Fun-Rel-rel-cong* [*cong*]:
assumes $R = R'$
and $\bigwedge x\ y.\ R'\ x\ y \Longrightarrow S\ x\ y = S'\ x\ y$
shows $((x\ y :: R) \Rrightarrow S\ x\ y) = ((x\ y :: R') \Rrightarrow S'\ x\ y)$
⟨*proof*⟩

lemma *Fun-Rel-relI* [*intro*]:
assumes $\bigwedge x\ y.\ R\ x\ y \Longrightarrow S\ (f\ x)\ (g\ y)$
shows $(R \Rrightarrow S)\ f\ g$
⟨*proof*⟩

lemma *Fun-Rel-relD* [*dest*]:
assumes $(R \Rrightarrow S)\ f\ g$
and $R\ x\ y$
shows $S\ (f\ x)\ (g\ y)$
⟨*proof*⟩

lemma *Fun-Rel-relE*:

assumes $((x\ y :: R) \Rrightarrow S\ x\ y)\ f\ g$
obtains $\bigwedge x\ y.\ R\ x\ y \Longrightarrow S\ x\ y\ (f\ x)\ (g\ y)$
⟨*proof*⟩

lemma *Dep-Fun-Rel-predI* [*intro*]:
assumes $\bigwedge x.\ P\ x \Longrightarrow R\ x\ (f\ x)\ (g\ x)$
shows $((x : P) \Rrightarrow R\ x)\ f\ g$
⟨*proof*⟩

lemma *Dep-Fun-Rel-predD* [*dest*]:
assumes $((x : P) \Rrightarrow R\ x)\ f\ g$
and $P\ x$
shows $R\ x\ (f\ x)\ (g\ x)$
⟨*proof*⟩

lemma *Dep-Fun-Rel-predE*:
assumes $((x : P) \Rrightarrow R\ x)\ f\ g$
obtains $\bigwedge x.\ P\ x \Longrightarrow R\ x\ (f\ x)\ (g\ x)$
⟨*proof*⟩

lemma *Dep-Fun-Rel-pred-cong* [*cong*]:
assumes $P = P'$
and $\bigwedge x.\ P'\ x \Longrightarrow R\ x = R'\ x$
shows $((x : P) \Rrightarrow R\ x) = ((x : P') \Rrightarrow R'\ x)$
⟨*proof*⟩

lemma *Fun-Rel-predI* [*intro*]:
assumes $\bigwedge x.\ P\ x \Longrightarrow R\ (f\ x)\ (g\ x)$
shows $(P \Rrightarrow R)\ f\ g$
⟨*proof*⟩

lemma *Fun-Rel-predD* [*dest*]:
assumes $(P \Rrightarrow R)\ f\ g$
and $P\ x$
shows $R\ (f\ x)\ (g\ x)$
⟨*proof*⟩

lemma *Fun-Rel-predE*:
assumes $(P \Rrightarrow R)\ f\ g$
obtains $\bigwedge x.\ P\ x \Longrightarrow R\ (f\ x)\ (g\ x)$
⟨*proof*⟩

lemma *rel-inv-Dep-Fun-Rel-rel-eq* [*simp*]:
fixes $R :: 'a \Rightarrow 'b \Rightarrow bool$ **and** $S :: 'a \Rightarrow 'b \Rightarrow 'c \Rightarrow 'd \Rightarrow bool$
shows $((x\ y :: R) \Rrightarrow S\ x\ y)^{-1} = ((y\ x :: R^{-1}) \Rrightarrow (S\ x\ y)^{-1})$
⟨*proof*⟩

lemma *rel-inv-rel-inv-Dep-Fun-Rel-rel-iff-Dep-Fun-Rel-rel* [*iff*]:
$((x\ y :: R^{-1}) \Rrightarrow (S\ x\ y)^{-1})\ f\ g \longleftrightarrow ((x\ y :: R) \Rrightarrow (S\ y\ x))\ g\ f$

⟨*proof*⟩

lemma *rel-inv-Dep-Fun-Rel-pred-eq* [*simp*]:
fixes $P :: 'a \Rightarrow bool$ **and** $R :: 'a \Rightarrow 'b \Rightarrow 'c \Rightarrow bool$
shows $((x : P) \Rrightarrow R\ x)^{-1} = ((x : P) \Rrightarrow (R\ x)^{-1})$
⟨*proof*⟩

lemma *rel-inv-Dep-Fun-Rel-pred-iff-Dep-Fun-Rel-pred* [*iff*]:
$((x : P) \Rrightarrow (R\ x)^{-1})\ f\ g \longleftrightarrow ((x : P) \Rrightarrow (R\ x))\ g\ f$
⟨*proof*⟩

lemma *Dep-Fun-Rel-pred-eq-Dep-Fun-Rel-rel*:
fixes $P :: 'a \Rightarrow bool$ **and** $R :: 'a \Rightarrow 'b \Rightarrow 'c \Rightarrow bool$
shows $((x : P) \Rrightarrow R\ x) = ((x\ (\text{-} :: 'a) :: (((\sqcap)\ P) \circ (=))) \Rrightarrow R\ x)$
⟨*proof*⟩

lemma *Fun-Rel-eq-eq-eq* [*simp*]: $((=) \Rrightarrow (=)) = (=)$
⟨*proof*⟩

Composition **lemma** *Dep-Fun-Rel-rel-compI*:
assumes $((x\ y :: R) \Rrightarrow S\ x\ y)\ f\ g$
and $\bigwedge x\ y.\ R\ x\ y \Longrightarrow ((x'\ y' :: T\ x\ y) \Rrightarrow U\ x\ y\ x'\ y')\ f'\ g'$
and $\bigwedge x\ y.\ R\ x\ y \Longrightarrow S\ x\ y\ (f\ x)\ (g\ y) \Longrightarrow T\ x\ y\ (f\ x)\ (g\ y)$
shows $((x\ y :: R) \Rrightarrow U\ x\ y\ (f\ x)\ (g\ y))\ (f' \circ f)\ (g' \circ g)$
⟨*proof*⟩

corollary *Dep-Fun-Rel-rel-compI'*:
assumes $((x\ y :: R) \Rrightarrow S\ x\ y)\ f\ g$
and $\bigwedge x\ y.\ R\ x\ y \Longrightarrow ((x'\ y' :: S\ x\ y) \Rrightarrow T\ x\ y\ x'\ y')\ f'\ g'$
shows $((x\ y :: R) \Rrightarrow T\ x\ y\ (f\ x)\ (g\ y))\ (f' \circ f)\ (g' \circ g)$
⟨*proof*⟩

lemma *Dep-Fun-Rel-pred-comp-Dep-Fun-Rel-rel-compI*:
assumes $((x : P) \Rrightarrow R\ x)\ f\ g$
and $\bigwedge x.\ P\ x \Longrightarrow ((x'\ y' :: S\ x) \Rrightarrow T\ x\ x'\ y')\ f'\ g'$
and $\bigwedge x.\ P\ x \Longrightarrow R\ x\ (f\ x)\ (g\ x) \Longrightarrow S\ x\ (f\ x)\ (g\ x)$
shows $((x : P) \Rrightarrow T\ x\ (f\ x)\ (g\ x))\ (f' \circ f)\ (g' \circ g)$
⟨*proof*⟩

corollary *Dep-Fun-Rel-pred-comp-Dep-Fun-Rel-rel-compI'*:
assumes $((x : P) \Rrightarrow R\ x)\ f\ g$
and $\bigwedge x.\ P\ x \Longrightarrow ((x'\ y' :: R\ x) \Rrightarrow S\ x\ x'\ y')\ f'\ g'$
shows $((x : P) \Rrightarrow S\ x\ (f\ x)\ (g\ x))\ (f' \circ f)\ (g' \circ g)$
⟨*proof*⟩

Restrictions **lemma** *Dep-Fun-Rel-restrict-left-restrict-left-eq*:
assumes $\bigwedge f\ x\ y.\ Q\ f \Longrightarrow R\ x\ y \Longrightarrow P\ x\ y\ (f\ x)$
shows $((x\ y :: R) \Rrightarrow (S\ x\ y)\restriction_{P\ x\ y})\restriction_Q = ((x\ y :: R) \Rrightarrow S\ x\ y)\restriction_Q$
⟨*proof*⟩

lemma *Dep-Fun-Rel-restrict-right-restrict-right-eq*:
assumes $\bigwedge f\ x\ y.\ Q\ f \Longrightarrow R\ x\ y \Longrightarrow P\ x\ y\ (f\ y)$
shows $((x\ y :: R) \Rrightarrow (S\ x\ y)\restriction_{P\ x\ y})\restriction_{Q} = (((x\ y :: R) \Rrightarrow S\ x\ y)\restriction_{Q})$
⟨*proof*⟩

end

1.2.3 Functions on Predicates

theory *Predicate-Functions*
imports
Functions-Base
begin

definition *pred-map f* $(P :: 'a \Rightarrow bool)\ x \equiv P\ (f\ x)$

lemma *pred-map-eq* [*simp*]: *pred-map f P x* $= P\ (f\ x)$
⟨*proof*⟩

lemma *comp-eq-pred-map* [*simp*]: $P \circ f =$ *pred-map f P*
⟨*proof*⟩

consts *pred-if* :: $bool \Rightarrow 'a \Rightarrow 'a$

open-bundle *pred-if-syntax*
begin
notation (**output**) *pred-if* (**infixl** ‹⟶› *50*)
end

definition *pred-if-pred B P x* $\equiv B \longrightarrow P\ x$
adhoc-overloading *pred-if* $\rightleftharpoons$ *pred-if-pred*

lemma *pred-if-eq-pred-if-pred* [*simp*]:
assumes *B*
shows (*pred-if B P*) $= P$
⟨*proof*⟩

lemma *pred-if-eq-top-if-not-pred* [*simp*]:
assumes $\neg B$
shows (*pred-if B P*) $= (\lambda\text{-}.\ True)$
⟨*proof*⟩

lemma *pred-if-if-impI* [*intro*]:
assumes $B \Longrightarrow P\ x$
shows (*pred-if B P*) *x*
⟨*proof*⟩

lemma *pred-ifE* [*elim*]:
assumes (*pred-if B P*) *x*
obtains ¬*B* | *B P x*
⟨*proof*⟩

lemma *pred-ifD*:
assumes (*pred-if B P*) *x*
and *B*
shows *P x*
⟨*proof*⟩

end

1.2.4 Orders

theory *Predicates-Order*
imports
HOL.Orderings
begin

lemma *le-predI* [*intro*]:
assumes ⋀*x*. *P x* ⟹ *Q x*
shows *P* ≤ *Q*
⟨*proof*⟩

lemma *le-predD* [*dest*]:
assumes *P* ≤ *Q*
and *P x*
shows *Q x*
⟨*proof*⟩

lemma *le-predE*:
assumes *P* ≤ *Q*
and *P x*
obtains *Q x*
⟨*proof*⟩

declare *le-boolD*[*dest*] **and** *le-boolI*[*intro*]

end

Monotonicity

theory *Functions-Monotone*
imports
Binary-Relations-Order-Base
Function-Relators
Predicate-Functions
Predicates-Order

begin

Summary Introduces the concept of monotone functions. A function is monotone if it is related to itself - see *Dep-Fun-Rel*.

declare *le-funI*[*intro*]
declare *le-funE*[*elim*]

consts *dep-mono-wrt* :: $'a \Rightarrow 'b \Rightarrow 'c$
consts *mono-wrt* :: $'a \Rightarrow 'b \Rightarrow 'c$

open-bundle *dep-mono-wrt-syntax*
begin
notation *mono-wrt* (**infixr** ‹⇒› *50*)
syntax
-dep-mono-wrt-rel :: *idt* ⇒ *idt* ⇒ $'a \Rightarrow 'b \Rightarrow 'c$
(‹'(-/ -/ ::/ -') ⇒ (-)› [*51*, *51*, *50*, *50*] *50*)
-dep-mono-wrt-rel-if :: *idt* ⇒ *idt* ⇒ $'a \Rightarrow$ *bool* ⇒ $'b \Rightarrow 'c$
(‹'(-/ -/ ::/ -/ |/ -') ⇒ (-)› [*51*, *51*, *50*, *50*, *50*] *50*)
-dep-mono-wrt-pred :: *idt* ⇒ $'a \Rightarrow 'b \Rightarrow 'c$ (‹'(-/ :/ -') ⇒ (-)› [*51*, *50*, *50*] *50*)
-dep-mono-wrt-pred-if :: *idt* ⇒ $'a \Rightarrow$ *bool* ⇒ $'b \Rightarrow 'c$
(‹'(-/ :/ -/ |/ -') ⇒ (-)› [*51*, *50*, *50*, *50*] *50*)
end
syntax-consts
-dep-mono-wrt-rel -dep-mono-wrt-rel-if -dep-mono-wrt-pred -dep-mono-wrt-pred-if ⇌ *dep-mono-wrt*
translations
(*x y* :: *R*) ⇒ *S* ⇌ *CONST dep-mono-wrt R* (λ*x y*. *S*)
(*x y* :: *R* | *B*) ⇒ *S* ⇌ *CONST dep-mono-wrt R* (λ*x y*. *CONST rel-if B S*)
(*x* : *P*) ⇒ *Q* ⇌ *CONST dep-mono-wrt P* (λ*x*. *Q*)
(*x* : *P* | *B*) ⇒ *Q* ⇌ *CONST dep-mono-wrt P* (λ*x*. *CONST pred-if B Q*)

definition *dep-mono-wrt-rel* (*R* :: $'a \Rightarrow 'a \Rightarrow$ *bool*)
(*S* :: $'a \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b \Rightarrow$ *bool*) (*f* :: $'a \Rightarrow 'b$) ≡ ((*x y* :: *R*) ⇛ *S x y*) *f f*
adhoc-overloading *dep-mono-wrt* ⇌ *dep-mono-wrt-rel*

definition *mono-wrt-rel* (*R* :: $'a \Rightarrow 'a \Rightarrow$ *bool*) (*S* :: $'b \Rightarrow 'b \Rightarrow$ *bool*) ≡
((- - :: *R*) ⇒ *S*) :: ($'a \Rightarrow 'b$) ⇒ *bool*
adhoc-overloading *mono-wrt* ⇌ *mono-wrt-rel*

definition *dep-mono-wrt-pred* (*P* :: $'a \Rightarrow$ *bool*) (*Q* :: $'a \Rightarrow 'b \Rightarrow$ *bool*) (*f* :: $'a \Rightarrow 'b$) ≡
((*x* : *P*) ⇛ (λ(- :: $'b$). *Q x*)) *f f*
adhoc-overloading *dep-mono-wrt* ⇌ *dep-mono-wrt-pred*

definition *mono-wrt-pred* (*P* :: $'a \Rightarrow$ *bool*) (*Q* :: $'b \Rightarrow$ *bool*) ≡
(((- :: $'a$) : *P*) ⇒ *Q*) :: ($'a \Rightarrow 'b$) ⇒ *bool*
adhoc-overloading *mono-wrt* ⇌ *mono-wrt-pred*

lemma *mono-wrt-rel-eq-dep-mono-wrt-rel*:

$((R :: 'a \Rightarrow 'a \Rightarrow bool) \Rightarrow (S :: 'b \Rightarrow 'b \Rightarrow bool)) = ((- - :: R) \Rightarrow S)$
⟨*proof*⟩

lemma *mono-wrt-rel-eq-dep-mono-wrt-rel-uhint* [*uhint*]:
assumes $R \equiv R'$
and $S' \equiv (\lambda(- :: 'a)\ (- :: 'a).\ S)$
shows $((R :: 'a \Rightarrow 'a \Rightarrow bool) \Rightarrow (S :: 'b \Rightarrow 'b \Rightarrow bool)) = ((x\ y :: R') \Rightarrow S'\ x\ y)$
⟨*proof*⟩

lemma *mono-wrt-rel-iff-dep-mono-wrt-rel*:
$((R :: 'a \Rightarrow 'a \Rightarrow bool) \Rightarrow (S :: 'b \Rightarrow 'b \Rightarrow bool))\ f \longleftrightarrow$
dep-mono-wrt $R\ (\lambda(- :: 'a)\ (- :: 'a).\ S)\ (f :: 'a \Rightarrow 'b)$
⟨*proof*⟩

lemma *mono-wrt-pred-eq-dep-mono-wrt-pred*:
$((P :: 'a \Rightarrow bool) \Rightarrow (Q :: 'b \Rightarrow bool)) = (((- :: 'a) : P) \Rightarrow Q)$
⟨*proof*⟩

lemma *mono-wrt-pred-eq-dep-mono-wrt-pred-uhint* [*uhint*]:
assumes $P \equiv P'$
and $\bigwedge x.\ Q \equiv Q'\ x$
shows $((P :: 'a \Rightarrow bool) \Rightarrow (Q :: 'b \Rightarrow bool)) = (((x : P') \Rightarrow Q'\ x) :: ('a \Rightarrow 'b) \Rightarrow bool)$
⟨*proof*⟩

lemma *mono-wrt-pred-iff-dep-mono-wrt-pred*:
$((P :: 'a \Rightarrow bool) \Rightarrow (Q :: 'b \Rightarrow bool))\ f \longleftrightarrow (((- :: 'a) : P) \Rightarrow Q)\ (f :: 'a \Rightarrow 'b)$
⟨*proof*⟩

lemma *dep-mono-wrt-relI* [*intro*]:
assumes $\bigwedge x\ y.\ R\ x\ y \Longrightarrow S\ x\ y\ (f\ x)\ (f\ y)$
shows $((x\ y :: R) \Rightarrow S\ x\ y)\ f$
⟨*proof*⟩

lemma *dep-mono-wrt-relE*:
assumes $((x\ y :: R) \Rightarrow S\ x\ y)\ f$
obtains $\bigwedge x\ y.\ R\ x\ y \Longrightarrow S\ x\ y\ (f\ x)\ (f\ y)$
⟨*proof*⟩

lemma *dep-mono-wrt-relD* [*dest*]:
assumes $((x\ y :: R) \Rightarrow S\ x\ y)\ f$
and $R\ x\ y$
shows $S\ x\ y\ (f\ x)\ (f\ y)$
⟨*proof*⟩

lemma *dep-mono-wrt-rel-cong* [*cong*]:
assumes $R = R'$
and $\bigwedge x\ y.\ R'\ x\ y \Longrightarrow S\ x\ y = S'\ x\ y$
shows $((x\ y :: R) \Rightarrow S\ x\ y) = ((x\ y :: R') \Rightarrow S'\ x\ y)$

⟨*proof*⟩

lemma *mono-wrt-relI* [*intro*]:
assumes $\bigwedge x\ y.\ R\ x\ y \Longrightarrow S\ (f\ x)\ (f\ y)$
shows $(R \Rightarrow S)\ f$
⟨*proof*⟩

lemma *mono-wrt-relE*:
assumes $(R \Rightarrow S)\ f$
obtains $\bigwedge x\ y.\ R\ x\ y \Longrightarrow S\ (f\ x)\ (f\ y)$
⟨*proof*⟩

lemma *mono-wrt-relD* [*dest*]:
assumes $(R \Rightarrow S)\ f$
and $R\ x\ y$
shows $S\ (f\ x)\ (f\ y)$
⟨*proof*⟩

lemma *dep-mono-wrt-predI* [*intro*]:
assumes $\bigwedge x.\ P\ x \Longrightarrow Q\ x\ (f\ x)$
shows $((x : P) \Rightarrow Q\ x)\ f$
⟨*proof*⟩

lemma *dep-mono-wrt-predE*:
assumes $((x : P) \Rightarrow Q\ x)\ f$
obtains $\bigwedge x.\ P\ x \Longrightarrow Q\ x\ (f\ x)$
⟨*proof*⟩

lemma *dep-mono-wrt-predD* [*dest*]:
assumes $((x : P) \Rightarrow Q\ x)\ f$
and $P\ x$
shows $Q\ x\ (f\ x)$
⟨*proof*⟩

lemma *dep-mono-wrt-pred-cong* [*cong*]:
assumes $P = P'$
and $\bigwedge x.\ P'\ x \Longrightarrow Q\ x = Q'\ x$
shows $((x : P) \Rightarrow Q\ x) = ((x : P') \Rightarrow Q'\ x)$
⟨*proof*⟩

lemma *dep-mono-wrt-pred-codom-iff-cong*:
assumes $P = P'$
and $\bigwedge x.\ P'\ x \Longrightarrow Q\ x\ (f\ x) \longleftrightarrow Q'\ x\ (f'\ x)$
shows $((x : P) \Rightarrow Q\ x)\ f \longleftrightarrow ((x : P') \Rightarrow Q'\ x)\ f'$
⟨*proof*⟩

lemma *mono-wrt-predI* [*intro*]:
assumes $\bigwedge x.\ P\ x \Longrightarrow Q\ (f\ x)$
shows $(P \Rightarrow Q)\ f$

⟨*proof*⟩

lemma *mono-wrt-predE*:
assumes $(P \Rightarrow Q)\ f$
obtains $\bigwedge x.\ P\ x \Longrightarrow Q\ (f\ x)$
⟨*proof*⟩

lemma *mono-wrt-predD* [*dest*]:
assumes $(P \Rightarrow Q)\ f$
and $P\ x$
shows $Q\ (f\ x)$
⟨*proof*⟩

context
fixes $R :: 'a \Rightarrow 'a \Rightarrow bool$ **and** $S :: 'a \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b \Rightarrow bool$ **and** $f :: 'a \Rightarrow 'b$
and $P :: 'a \Rightarrow bool$ **and** $Q :: 'a \Rightarrow 'b \Rightarrow bool$
begin

lemma *dep-mono-wrt-rel-if-Dep-Fun-Rel-rel-self*:
assumes $((x\ y :: R) \Rrightarrow S\ x\ y)\ f\ f$
shows $((x\ y :: R) \Rightarrow S\ x\ y)\ f$
⟨*proof*⟩

lemma *dep-mono-wrt-pred-if-Dep-Fun-Rel-pred-self*:
assumes $((x : P) \Rrightarrow (\lambda\text{-}.\ Q\ x))\ f\ f$
shows $((x : P) \Rightarrow Q\ x)\ f$
⟨*proof*⟩

lemma *Dep-Fun-Rel-rel-self-if-dep-mono-wrt-rel*:
assumes $((x\ y :: R) \Rightarrow S\ x\ y)\ f$
shows $((x\ y :: R) \Rrightarrow S\ x\ y)\ f\ f$
⟨*proof*⟩

lemma *Dep-Fun-Rel-pred-self-if-dep-mono-wrt-pred*:
assumes $((x : P) \Rightarrow Q\ x)\ f$
shows $((x : P) \Rrightarrow (\lambda\text{-}.\ Q\ x))\ f\ f$
⟨*proof*⟩

corollary *Dep-Fun-Rel-rel-self-iff-dep-mono-wrt-rel*:
$((x\ y :: R) \Rrightarrow S\ x\ y)\ f\ f \longleftrightarrow ((x\ y :: R) \Rightarrow S\ x\ y)\ f$
⟨*proof*⟩

corollary *Dep-Fun-Rel-pred-self-iff-dep-mono-wrt-pred*:
$((x : P) \Rrightarrow (\lambda(\text{-} :: 'b).\ Q\ x))\ f\ f \longleftrightarrow ((x : P) \Rightarrow Q\ x)\ f$
⟨*proof*⟩

lemma *dep-mono-wrt-rel-inv-eq* [*simp*]:
$((y\ x :: R^{-1}) \Rightarrow (S\ x\ y)^{-1}) = ((x\ y :: R) \Rightarrow S\ x\ y)$
⟨*proof*⟩

lemma *in-dom-if-rel-if-dep-mono-wrt-rel*:
assumes $((x\ y :: R) \Rightarrow S\ x\ y)\ f$
and $R\ x\ y$
shows *in-dom* $(S\ x\ y)\ (f\ x)$
⟨*proof*⟩

end

context
fixes $R :: 'a \Rightarrow 'a \Rightarrow bool$ **and** $f :: 'a \Rightarrow 'b$
begin

corollary *in-dom-if-in-dom-if-mono-wrt-rel*:
assumes $(R \Rightarrow S)\ f$
shows $(\textit{in-dom}\ R \Rightarrow \textit{in-dom}\ S)\ f$
⟨*proof*⟩

lemma *in-codom-if-rel-if-dep-mono-wrt-rel*:
assumes $((x\ y :: R) \Rightarrow S\ x\ y)\ f$
and $R\ x\ y$
shows *in-codom* $(S\ x\ y)\ (f\ y)$
⟨*proof*⟩

corollary *in-codom-if-in-codom-if-mono-wrt-rel*:
assumes $(R \Rightarrow S)\ f$
shows $(\textit{in-codom}\ R \Rightarrow \textit{in-codom}\ S)\ f$
⟨*proof*⟩

corollary *in-field-if-in-field-if-mono-wrt-rel*:
assumes $(R \Rightarrow S)\ f$
shows $(\textit{in-field}\ R \Rightarrow \textit{in-field}\ S)\ f$
⟨*proof*⟩

lemma *le-rel-map-if-mono-wrt-rel*:
assumes $(R \Rightarrow S)\ f$
shows $R \leq \textit{rel-map}\ f\ S$
⟨*proof*⟩

lemma *le-pred-map-if-mono-wrt-pred*:
assumes $(P \Rightarrow Q)\ f$
shows $P \leq \textit{pred-map}\ f\ Q$
⟨*proof*⟩

lemma *mono-wrt-rel-if-le-rel-map*:
assumes $R \leq \textit{rel-map}\ f\ S$
shows $(R \Rightarrow S)\ f$
⟨*proof*⟩

lemma *mono-wrt-pred-if-le-pred-map*:
assumes $P \leq$ *pred-map* $f\ Q$
shows $(P \Rightarrow Q)\ f$
⟨*proof*⟩

corollary *mono-wrt-rel-iff-le-rel-map*: $(R \Rightarrow S)\ f \longleftrightarrow R \leq$ *rel-map* $f\ S$
⟨*proof*⟩

corollary *mono-wrt-pred-iff-le-pred-map*: $(P \Rightarrow Q)\ f \longleftrightarrow P \leq$ *pred-map* $f\ Q$
⟨*proof*⟩

end

lemma *dep-mono-comp-iff-dep-mono-if-all-app-eq*:
assumes $\bigwedge x.\ A\ x \Longrightarrow f\ (m\ x) = g\ x$
shows $((x : (A :: - \Rightarrow bool)) \Rightarrow B\ x)\ (f \circ m) \longleftrightarrow ((x : A) \Rightarrow B\ x)\ g$
⟨*proof*⟩

lemma *dep-mono-pred-map-comp-iff-dep-mono-if-all-app-eq*:
assumes $\bigwedge x\ y.\ A\ x \Longrightarrow B\ x\ y \Longrightarrow f\ x\ (m\ y) = g\ x\ y$
shows $((x : A) \Rightarrow$ *pred-map* $(\lambda f.\ f \circ m)\ ((y : B\ x) \Rightarrow C\ x\ y))\ f$
$\longleftrightarrow ((x : A) \Rightarrow (y : (B\ x :: - \Rightarrow bool)) \Rightarrow C\ x\ y)\ g$
⟨*proof*⟩

definition *mono* $:: (('a :: ord) \Rightarrow ('b :: ord)) \Rightarrow bool$
$\equiv (((\leq) :: 'a \Rightarrow 'a \Rightarrow bool) \Rightarrow ((\leq) :: 'b \Rightarrow 'b \Rightarrow bool))$

lemma *mono-eq-mono-wrt-le* [*simp*]: $(mono :: (('a :: ord) \Rightarrow ('b :: ord)) \Rightarrow bool)$
$=$
$(((\leq) :: 'a \Rightarrow 'a \Rightarrow bool) \Rightarrow ((\leq) :: 'b \Rightarrow 'b \Rightarrow bool))$
⟨*proof*⟩

lemma *mono-eq-mono-wrt-le-uhint* [*uhint*]:
assumes $R \equiv (\leq) :: 'a \Rightarrow 'a \Rightarrow bool$
and $S \equiv (\leq) :: 'b \Rightarrow 'b \Rightarrow bool$
shows $mono :: (('a :: ord) \Rightarrow ('b :: ord)) \Rightarrow bool \equiv (R \Rightarrow S)$
⟨*proof*⟩

lemma *mono-iff-mono-wrt-le* [*iff*]: $mono\ f \longleftrightarrow ((\leq) \Rightarrow (\leq))\ f\ \langle proof \rangle$

lemma *monoI* [*intro*]:
assumes $\bigwedge x\ y.\ x \leq y \Longrightarrow f\ x \leq f\ y$
shows *mono f*
⟨*proof*⟩

lemma *monoE* [*elim*]:
assumes *mono f*
obtains $\bigwedge x\ y.\ x \leq y \Longrightarrow f\ x \leq f\ y$
⟨*proof*⟩

lemma *monoD*:
assumes *mono f*
and $x \leq y$
shows $f\ x \leq f\ y$
⟨*proof*⟩

definition *antimono* :: $((\prime a :: ord) \Rightarrow (\prime b :: ord)) \Rightarrow bool$
$\equiv (((\leq) :: \prime a \Rightarrow \prime a \Rightarrow bool) \Rightarrow ((\geq) :: \prime b \Rightarrow \prime b \Rightarrow bool))$

lemma *antimono-eq-mono-wrt-le-ge* [*simp*]: (*antimono* :: $((\prime a :: ord) \Rightarrow (\prime b :: ord))$
$\Rightarrow bool) =$
$(((\leq) :: \prime a \Rightarrow \prime a \Rightarrow bool) \Rightarrow ((\geq) :: \prime b \Rightarrow \prime b \Rightarrow bool))$
⟨*proof*⟩

lemma *antimono-eq-mono-wrt-le-ge-uhint* [*uhint*]:
assumes $R \equiv (\leq) :: \prime a \Rightarrow \prime a \Rightarrow bool$
and $S \equiv (\geq) :: \prime b \Rightarrow \prime b \Rightarrow bool$
shows *antimono* :: $((\prime a :: ord) \Rightarrow (\prime b :: ord)) \Rightarrow bool \equiv (R \Rightarrow S)$
⟨*proof*⟩

lemma *antimono-iff-mono-wrt-le-ge* [*iff*]: *antimono* $f \longleftrightarrow ((\leq) \Rightarrow (\geq))\ f$ ⟨*proof*⟩

lemma *antimonoI* [*intro*]:
assumes $\bigwedge x\ y.\ x \leq y \Longrightarrow f\ x \geq f\ y$
shows *antimono f*
⟨*proof*⟩

lemma *antimonoE* [*elim*]:
assumes *antimono f*
obtains $\bigwedge x\ y.\ x \leq y \Longrightarrow f\ x \geq f\ y$
⟨*proof*⟩

lemma *antimonoD*:
assumes *antimono f*
and $x \leq y$
shows $f\ x \geq f\ y$
⟨*proof*⟩

lemma *antimono-Dep-Fun-Rel-rel-left*: *antimono* $(\lambda(R :: \prime a \Rightarrow \prime b \Rightarrow bool).\ ((x\ y ::$
$R) \Rrightarrow S\ x\ y))$
⟨*proof*⟩

lemma *antimono-Dep-Fun-Rel-pred-left*: *antimono* $(\lambda(P :: \prime a \Rightarrow bool).\ ((x : P) \Rrightarrow$
$Q\ x))$
⟨*proof*⟩

lemma *antimono-dep-mono-wrt-rel-left*: *antimono* $(\lambda(R :: \prime a \Rightarrow \prime a \Rightarrow bool).\ ((x\ y$
$:: R) \Rightarrow S\ x\ y))$

⟨*proof*⟩

lemma *antimono-dep-mono-wrt-pred-left*: *antimono* (λ(*P* :: *'a* ⇒ *bool*). ((*x* : *P*) ⇒ *Q x*))
⟨*proof*⟩

lemma *Dep-Fun-Rel-rel-if-le-left-if-Dep-Fun-Rel-rel*:
fixes *R R'* :: *'a* ⇒ *'b* ⇒ *bool*
assumes ((*x y* :: *R*) ⇛ *S x y*) *f g*
and *R'* ≤ *R*
shows ((*x y* :: *R*) ⇛ *S x y*) *f g*
⟨*proof*⟩

lemma *Dep-Fun-Rel-pred-if-le-left-if-Dep-Fun-Rel-pred*:
fixes *P P'* :: *'a* ⇒ *bool*
assumes ((*x* : *P*) ⇛ *Q x*) *f g*
and *P'* ≤ *P*
shows ((*x* : *P'*) ⇛ *Q x*) *f g*
⟨*proof*⟩

lemma *dep-mono-wrt-rel-if-le-left-if-dep-mono-wrt-rel*:
fixes *R R'* :: *'a* ⇒ *'a* ⇒ *bool*
assumes ((*x y* :: *R*) ⇒ *S x y*) *f*
and *R'* ≤ *R*
shows ((*x y* :: *R'*) ⇒ *S x y*) *f*
⟨*proof*⟩

lemma *dep-mono-wrt-pred-if-le-left-if-dep-mono-wrt-pred*:
fixes *P P'* :: *'a* ⇒ *bool*
assumes ((*x* : *P*) ⇒ *Q x*) *f*
and *P'* ≤ *P*
shows ((*x* : *P'*) ⇒ *Q x*) *f*
⟨*proof*⟩

lemma *mono-Dep-Fun-Rel-rel-right*: *mono* (λ(*S* :: *'a* ⇒ *'b* ⇒ *'c* ⇒ *'d* ⇒ *bool*). ((*x y* :: *R*) ⇛ *S x y*))
⟨*proof*⟩

lemma *mono-Dep-Fun-Rel-pred-right*: *mono* (λ(*Q* :: *'a* ⇒ *'b* ⇒ *'c* ⇒ *bool*). ((*x* : *P*) ⇛ *Q x*))
⟨*proof*⟩

lemma *mono-dep-mono-wrt-rel-right*: *mono* (λ(*S* :: *'a* ⇒ *'a* ⇒ *'b* ⇒ *'b* ⇒ *bool*). ((*x y* :: *R*) ⇒ *S x y*))
⟨*proof*⟩

lemma *mono-dep-mono-wrt-pred-right*: *mono* (λ(*Q* :: *'a* ⇒ *'b* ⇒ *bool*). ((*x* : *P*) ⇒ *Q x*))
⟨*proof*⟩

lemma *Dep-Fun-Rel-rel-if-le-right-if-Dep-Fun-Rel-rel*:
assumes $((x\ y :: R) \Rrightarrow S\ x\ y)\ f\ g$
and $\bigwedge x\ y.\ R\ x\ y \Longrightarrow S\ x\ y\ (f\ x)\ (g\ y) \Longrightarrow T\ x\ y\ (f\ x)\ (g\ y)$
shows $((x\ y :: R) \Rrightarrow T\ x\ y)\ f\ g$
⟨*proof*⟩

lemma *Dep-Fun-Rel-pred-if-le-right-if-Dep-Fun-Rel-pred*:
assumes $((x : P) \Rrightarrow Q\ x)\ f\ g$
and $\bigwedge x.\ P\ x \Longrightarrow Q\ x\ (f\ x)\ (g\ x) \Longrightarrow T\ x\ (f\ x)\ (g\ x)$
shows $((x : P) \Rrightarrow T\ x)\ f\ g$
⟨*proof*⟩

lemma *dep-mono-wrt-rel-if-le-right-if-dep-mono-wrt-rel*:
assumes $((x\ y :: R) \Rightarrow S\ x\ y)\ f$
and $\bigwedge x\ y.\ R\ x\ y \Longrightarrow S\ x\ y\ (f\ x)\ (f\ y) \Longrightarrow T\ x\ y\ (f\ x)\ (f\ y)$
shows $((x\ y :: R) \Rightarrow T\ x\ y)\ f$
⟨*proof*⟩

lemma *dep-mono-wrt-pred-if-le-right-if-dep-mono-wrt-pred*:
assumes $((x : P) \Rightarrow Q\ x)\ f$
and $\bigwedge x.\ P\ x \Longrightarrow Q\ x\ (f\ x) \Longrightarrow T\ x\ (f\ x)$
shows $((x : P) \Rightarrow T\ x)\ f$
⟨*proof*⟩

Composition **lemma** *dep-mono-wrt-rel-compI*:
assumes $((x\ y :: R) \Rightarrow S\ x\ y)\ f$
and $\bigwedge x\ y.\ R\ x\ y \Longrightarrow ((x'\ y' :: T\ x\ y) \Rightarrow U\ x\ y\ x'\ y')\ f'$
and $\bigwedge x\ y.\ R\ x\ y \Longrightarrow S\ x\ y\ (f\ x)\ (f\ y) \Longrightarrow T\ x\ y\ (f\ x)\ (f\ y)$
shows $((x\ y :: R) \Rightarrow U\ x\ y\ (f\ x)\ (f\ y))\ (f' \circ f)$
⟨*proof*⟩

corollary *dep-mono-wrt-rel-compI'*:
assumes $((x\ y :: R) \Rightarrow S\ x\ y)\ f$
and $\bigwedge x\ y.\ R\ x\ y \Longrightarrow ((x'\ y' :: S\ x\ y) \Rightarrow T\ x\ y\ x'\ y')\ f'$
shows $((x\ y :: R) \Rightarrow T\ x\ y\ (f\ x)\ (f\ y))\ (f' \circ f)$
⟨*proof*⟩

lemma *dep-mono-wrt-pred-comp-dep-mono-wrt-rel-compI*:
assumes $((x : P) \Rightarrow Q\ x)\ f$
and $\bigwedge x.\ P\ x \Longrightarrow ((x'\ y' :: R\ x) \Rightarrow S\ x\ x'\ y')\ f'$
and $\bigwedge x.\ P\ x \Longrightarrow Q\ x\ (f\ x) \Longrightarrow R\ x\ (f\ x)\ (f\ x)$
shows $((x : P) \Rightarrow (\lambda y.\ S\ x\ (f\ x)\ (f\ x)\ y\ y))\ (f' \circ f)$
⟨*proof*⟩

lemma *dep-mono-wrt-pred-comp-dep-mono-wrt-pred-compI*:
assumes $((x : P) \Rightarrow Q\ x)\ f$
and $\bigwedge x.\ P\ x \Longrightarrow ((x' : R\ x) \Rightarrow S\ x\ x')\ f'$
and $\bigwedge x.\ P\ x \Longrightarrow Q\ x\ (f\ x) \Longrightarrow R\ x\ (f\ x)$

shows $((x : P) \Rightarrow S\ x\ (f\ x))\ (f' \circ f)$
⟨*proof*⟩

corollary *dep-mono-wrt-pred-comp-dep-mono-wrt-pred-compI'*:
assumes $((x : P) \Rightarrow Q\ x)\ f$
and $\bigwedge x.\ P\ x \Longrightarrow ((x' : Q\ x) \Rightarrow S\ x\ x')\ f'$
shows $((x : P) \Rightarrow S\ x\ (f\ x))\ (f' \circ f)$
⟨*proof*⟩

lemma *mono-wrt-rel-top* [*iff*]:
fixes $R :: 'a \Rightarrow 'a \Rightarrow bool$ **and** $f :: 'a \Rightarrow 'b$
shows $(R \Rightarrow (\top :: 'b \Rightarrow 'b \Rightarrow bool))\ f$
⟨*proof*⟩

lemma *mono-wrt-pred-top* [*iff*]:
fixes $P :: 'a \Rightarrow bool$ **and** $f :: 'a \Rightarrow 'b$
shows $(P \Rightarrow (\top :: 'b \Rightarrow bool))\ f$
⟨*proof*⟩

Instantiations **lemma** *mono-wrt-rel-self-id*:
fixes $R :: 'a \Rightarrow 'a \Rightarrow bool$
shows $(R \Rightarrow R)\ (id :: 'a \Rightarrow 'a)$
⟨*proof*⟩

lemma *mono-wrt-pred-self-id*:
fixes $P :: 'a \Rightarrow bool$
shows $(P \Rightarrow P)\ (id :: 'a \Rightarrow 'a)$
⟨*proof*⟩

lemma *mono-dep-mono-wrt-dep-mono-wrt-comp*:
$(((x : (B :: 'b \Rightarrow bool)) \Rightarrow C\ x) \Rightarrow (f : A \Rightarrow B) \Rightarrow (x : A) \Rightarrow C\ (f\ x))\ (\circ)$
⟨*proof*⟩

lemma *mono-wrt-dep-mono-wrt-dep-fun-map*:
fixes $A :: 'a \Rightarrow bool$
shows $((f : A \Rightarrow B) \Rightarrow ((x : A) \Rightarrow (y : B) \Rightarrow (z : C\ (f\ x)) \Rightarrow D\ x\ y\ z) \Rightarrow$
$(h : (x : B) \Rightarrow C\ x) \Rightarrow (x : A) \Rightarrow D\ x\ (f\ x)\ (h\ (f\ x)))$ *dep-fun-map*
⟨*proof*⟩

lemma *mono-wrt-dep-mono-wrt-fun-map*:
fixes $A :: 'a \Rightarrow bool$
shows $((f : A \Rightarrow B) \Rightarrow ((x : C) \Rightarrow D\ x) \Rightarrow (h : B \Rightarrow C) \Rightarrow (x : A) \Rightarrow D\ (h\ (f\ x)))$ *fun-map*
⟨*proof*⟩

lemma *mono-in-dom*: *mono in-dom* ⟨*proof*⟩
lemma *mono-in-codom*: *mono in-codom* ⟨*proof*⟩
lemma *mono-in-field*: *mono in-field* ⟨*proof*⟩
lemma *mono-rel-comp*: $((\leq) \Rightarrow (\leq) \Rrightarrow (\leq))\ (\circ\circ)$ ⟨*proof*⟩

lemma *mono-rel-inv*: *mono rel-inv* ⟨*proof*⟩

lemma *mono-rel-restrict-left*:
(($\leq$) ⇒ ($\leq$) ⇛ ($\leq$)) (*rel-restrict-left* :: ('*a* ⇒ '*b* ⇒ *bool*) ⇒ ('*a* ⇒ *bool*) ⇒ '*a* ⇒ '*b* ⇒ *bool*)
⟨*proof*⟩

lemma *mono-rel-restrict-right*:
(($\leq$) ⇒ ($\leq$) ⇛ ($\leq$)) (*rel-restrict-right* :: ('*a* ⇒ '*b* ⇒ *bool*) ⇒ ('*b* ⇒ *bool*) ⇒ '*a* ⇒ '*b* ⇒ *bool*)
⟨*proof*⟩

lemma *mono-ball*: (($\geq$) ⇒ ($\leq$) ⇛ ($\leq$)) *ball* ⟨*proof*⟩
lemma *mono-bex*: (($\leq$) ⇒ ($\leq$) ⇛ ($\leq$)) *bex* ⟨*proof*⟩

end

theory *Reverse-Implies*
imports *Binary-Relation-Functions*
begin

definition *rev-implies* ≡ $(\longrightarrow)^{-1}$

open-bundle *rev-implies-syntax*
begin
notation *rev-implies* (**infixr** ‹$\longleftarrow$› *25*)
end

lemma *rev-imp-eq-imp-inv* [*simp*]: ($\longleftarrow$) = $(\longrightarrow)^{-1}$
⟨*proof*⟩

lemma *rev-impI* [*intro*!]:
assumes $Q \Longrightarrow P$
shows $P \longleftarrow Q$
⟨*proof*⟩

lemma *rev-impD* [*dest*!]:
assumes $P \longleftarrow Q$
shows $Q \Longrightarrow P$
⟨*proof*⟩

lemma *rev-imp-iff-imp*: $(P \longleftarrow Q) \longleftrightarrow (Q \longrightarrow P)$ ⟨*proof*⟩

end

Injective

theory *Binary-Relations-Injective*
imports

Functions-Monotone
Reverse-Implies
begin

consts *rel-injective-on* :: $'a \Rightarrow 'b \Rightarrow bool$

overloading
rel-injective-on-pred $\equiv$ *rel-injective-on* :: $('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool$
begin
definition *rel-injective-on-pred* $P\ R \equiv \forall x\ x' : P.\ \forall y.\ R\ x\ y \wedge R\ x'\ y \longrightarrow x = x'$
end

lemma *rel-injective-onI* [*intro*]:
assumes $\bigwedge x\ x'\ y.\ P\ x \Longrightarrow P\ x' \Longrightarrow R\ x\ y \Longrightarrow R\ x'\ y \Longrightarrow x = x'$
shows *rel-injective-on* $P\ R$
⟨*proof*⟩

lemma *rel-injective-onD*:
assumes *rel-injective-on* $P\ R$
and $P\ x\ P\ x'$
and $R\ x\ y\ R\ x'\ y$
shows $x = x'$
⟨*proof*⟩

lemma *antimono-rel-injective-on*:
$((\leq) \Rightarrow (\leq) \Rightarrow (\geq))$ (*rel-injective-on* :: $('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool$)
⟨*proof*⟩

consts *rel-injective-at* :: $'a \Rightarrow 'b \Rightarrow bool$

overloading
rel-injective-at-pred $\equiv$ *rel-injective-at* :: $('a \Rightarrow bool) \Rightarrow ('b \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
begin
definition *rel-injective-at-pred* $P\ R \equiv \forall x\ x'\ y.\ P\ y \wedge R\ x\ y \wedge R\ x'\ y \longrightarrow x = x'$
end

lemma *rel-injective-atI* [*intro*]:
assumes $\bigwedge x\ x'\ y.\ P\ y \Longrightarrow R\ x\ y \Longrightarrow R\ x'\ y \Longrightarrow x = x'$
shows *rel-injective-at* $P\ R$
⟨*proof*⟩

lemma *rel-injective-atD*:
assumes *rel-injective-at* $P\ R$
and $P\ y$
and $R\ x\ y\ R\ x'\ y$
shows $x = x'$
⟨*proof*⟩

lemma *rel-injective-on-if-Fun-Rel-imp-if-rel-injective-at*:
assumes *rel-injective-at* $(Q :: 'b \Rightarrow bool)$ $(R :: 'a \Rightarrow 'b \Rightarrow bool)$
and $(R \Rrightarrow (\longrightarrow))$ P Q
shows *rel-injective-on* P R
⟨*proof*⟩

lemma *rel-injective-at-if-Fun-Rel-rev-imp-if-rel-injective-on*:
assumes *rel-injective-on* $(P :: 'a \Rightarrow bool)$ $(R :: 'a \Rightarrow 'b \Rightarrow bool)$
and $(R \Rrightarrow (\longleftarrow))$ P Q
shows *rel-injective-at* Q R
⟨*proof*⟩

consts *rel-injective* :: $'a \Rightarrow bool$

overloading
rel-injective ≡ *rel-injective* :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool$
begin
definition (*rel-injective* :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool$) ≡ *rel-injective-on* ($\top$ :: $'a \Rightarrow bool$)
end

lemma *rel-injective-eq-rel-injective-on*:
(*rel-injective* :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow$ -) = *rel-injective-on* ($\top$ :: $'a \Rightarrow bool$)
⟨*proof*⟩

lemma *rel-injective-eq-rel-injective-on-uhint* [*uhint*]:
assumes $P \equiv (\top :: 'a \Rightarrow bool)$
shows *rel-injective* :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow$ - ≡ *rel-injective-on* P
⟨*proof*⟩

lemma *rel-injectiveI* [*intro*]:
assumes $\bigwedge x\ x'\ y.\ R\ x\ y \Longrightarrow R\ x'\ y \Longrightarrow x = x'$
shows *rel-injective* R
⟨*proof*⟩

lemma *rel-injectiveD*:
assumes *rel-injective* R
and $R\ x\ y\ R\ x'\ y$
shows $x = x'$
⟨*proof*⟩

lemma *rel-injective-eq-rel-injective-at*:
(*rel-injective* :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool$) = *rel-injective-at* ($\top$ :: $'b \Rightarrow bool$)
⟨*proof*⟩

lemma *rel-injective-eq-rel-injective-at-uhint* [*uhint*]:
assumes $P \equiv (\top :: 'b \Rightarrow bool)$
shows *rel-injective* :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool \equiv$ *rel-injective-at* P

⟨*proof*⟩

lemma *rel-injective-on-if-rel-injective*:
fixes $P :: 'a \Rightarrow bool$ **and** $R :: 'a \Rightarrow 'b \Rightarrow bool$
assumes *rel-injective R*
shows *rel-injective-on P R*
⟨*proof*⟩

lemma *rel-injective-at-if-rel-injective*:
fixes $P :: 'a \Rightarrow bool$ **and** $R :: 'b \Rightarrow 'a \Rightarrow bool$
assumes *rel-injective R*
shows *rel-injective-at P R*
⟨*proof*⟩

lemma *rel-injective-if-rel-injective-on-in-dom*:
assumes *rel-injective-on* (*in-dom R*) *R*
shows *rel-injective R*
⟨*proof*⟩

lemma *rel-injective-if-rel-injective-at-in-codom*:
assumes *rel-injective-at* (*in-codom R*) *R*
shows *rel-injective R*
⟨*proof*⟩

corollary *rel-injective-on-in-dom-iff-rel-injective* [*simp*]:
rel-injective-on (*in-dom R*) *R* $\longleftrightarrow$ *rel-injective R*
⟨*proof*⟩

corollary *rel-injective-at-in-codom-iff-rel-injective* [*iff*]:
rel-injective-at (*in-codom R*) *R* $\longleftrightarrow$ *rel-injective R*
⟨*proof*⟩

lemma *rel-injective-on-compI*:
fixes $P :: 'a \Rightarrow bool$
assumes *rel-injective* $(R :: 'a \Rightarrow 'b \Rightarrow bool)$
and *rel-injective-on* (*in-codom R* $\sqcap$ *in-dom S*) $(S :: 'b \Rightarrow 'c \Rightarrow bool)$
shows *rel-injective-on P* (*R* $\circ\circ$ *S*)
⟨*proof*⟩

end

1.2.5 Agreement

theory *Binary-Relations-Agree*
imports
Functions-Monotone
begin

consts *rel-agree-on* :: $'a \Rightarrow 'b \Rightarrow bool$

overloading
  rel-agree-on-pred $\equiv$ *rel-agree-on* :: $('a \Rightarrow bool) \Rightarrow (('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool) \Rightarrow bool$
begin
  definition *rel-agree-on-pred* $(P :: 'a \Rightarrow bool)$ $(\mathcal{R} :: ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool) \equiv$
    $\forall R\ R' : \mathcal{R}.\ R{\restriction_P} = R'{\restriction_P}$
end

lemma *rel-agree-onI* [*intro*]:
  assumes $\bigwedge R\ R'\ x\ y.\ \mathcal{R}\ R \Longrightarrow \mathcal{R}\ R' \Longrightarrow P\ x \Longrightarrow R\ x\ y \Longrightarrow R'\ x\ y$
  shows *rel-agree-on* $P\ \mathcal{R}$
  ⟨*proof*⟩

lemma *rel-agree-onE*:
  assumes *rel-agree-on* $P\ \mathcal{R}$
  obtains $\bigwedge R\ R'.\ \mathcal{R}\ R \Longrightarrow \mathcal{R}\ R' \Longrightarrow R{\restriction_P} = R'{\restriction_P}$
  ⟨*proof*⟩

lemma *rel-restrict-left-eq-if-rel-agree-onI*:
  assumes *rel-agree-on* $P\ \mathcal{R}$
  and $\mathcal{R}\ R\ \mathcal{R}\ R'$
  shows $R{\restriction_P} = R'{\restriction_P}$
  ⟨*proof*⟩

lemma *rel-agree-onD*:
  assumes *rel-agree-on* $P\ \mathcal{R}$
  and $\mathcal{R}\ R\ \mathcal{R}\ R'$
  and $P\ x$
  and $R\ x\ y$
  shows $R'\ x\ y$
  ⟨*proof*⟩

lemma *antimono-rel-agree-on*:
  $((\leq) \Rightarrow (\leq) \Rightarrow (\geq))$ (*rel-agree-on* :: $('a \Rightarrow bool) \Rightarrow (('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool) \Rightarrow bool$)
  ⟨*proof*⟩

lemma *le-if-in-dom-le-if-rel-agree-onI*:
  assumes *rel-agree-on* $A\ \mathcal{R}$
  and $\mathcal{R}\ R\ \mathcal{R}\ R'$
  and *in-dom* $R \leq A$
  shows $R \leq R'$
  ⟨*proof*⟩

lemma *eq-if-in-dom-le-if-rel-agree-onI*:
  assumes *rel-agree-on* $A\ \mathcal{R}$
  and $\mathcal{R}\ R\ \mathcal{R}\ R'$

and *in-dom* $R \leq A$ *in-dom* $R' \leq A$
shows $R = R'$
⟨*proof*⟩

end

1.2.6 Dependent Binary Relations

theory *Dependent-Binary-Relations*
imports
Binary-Relations-Agree
begin

consts *dep-bin-rel* :: $'a \Rightarrow ('b \Rightarrow 'c) \Rightarrow 'd$
consts *bin-rel* :: $'a \Rightarrow 'b \Rightarrow 'c$

open-bundle *bin-rel-syntax*
begin
syntax *-dep-bin-rel* :: ‹*idt* ⇒ $'a$ ⇒ $'b$ ⇒ $'c$› (‹{∑}- : -./ -› [*51*, *50*, *50*] *51*)
notation *bin-rel* (**infixr** ‹{×}› *51*)
end

syntax-consts
-dep-bin-rel ⇌ *dep-bin-rel*
translations
{∑}x : A. B ⇌ *CONST dep-bin-rel* A (λx. B)

definition *dep-bin-rel-pred* (A :: $'a \Rightarrow$ *bool*) (B :: $'a \Rightarrow 'b \Rightarrow$ *bool*) (R :: $'a \Rightarrow 'b \Rightarrow$ *bool*) ≡
∀ x y. R x $y \longrightarrow A$ $x \wedge B$ x y
adhoc-overloading *dep-bin-rel* ⇌ *dep-bin-rel-pred*

definition *bin-rel-pred* (A :: $'a \Rightarrow$ *bool*) (B :: $'b \Rightarrow$ *bool*) :: ($'a \Rightarrow 'b \Rightarrow$ *bool*) ⇒ *bool* ≡
{∑}(- :: $'a$) : A. B
adhoc-overloading *bin-rel* ⇌ *bin-rel-pred*

lemma *bin-rel-pred-eq-dep-bin-rel-pred*: A {×} B = {∑}- : A. B
⟨*proof*⟩

lemma *bin-rel-pred-eq-dep-bin-rel-pred-uhint* [*uhint*]:
assumes $A \equiv A'$
and ⋀x. $B \equiv B'$ x
shows A {×} $B \equiv$ {∑}x : A'. B' x
⟨*proof*⟩

lemma *bin-rel-pred-iff-dep-bin-rel-pred*: (A {×} B) $R \longleftrightarrow$ ({∑}- : A. B) R
⟨*proof*⟩

lemma *dep-bin-relI* [*intro*]:
assumes $\bigwedge x\ y.\ R\ x\ y \Longrightarrow A\ x$
and $\bigwedge x\ y.\ R\ x\ y \Longrightarrow A\ x \Longrightarrow B\ x\ y$
shows $(\{\sum\}x : A.\ B\ x)\ R$
⟨*proof*⟩

lemma *dep-bin-rel-if-bin-rel-and*:
assumes $\bigwedge x\ y.\ R\ x\ y \Longrightarrow A\ x \wedge B\ x\ y$
shows $(\{\sum\}x : A.\ B\ x)\ R$
⟨*proof*⟩

lemma *dep-bin-relE* [*elim*]:
assumes $(\{\sum\}x : A.\ B\ x)\ R$
and $R\ x\ y$
obtains $A\ x\ B\ x\ y$
⟨*proof*⟩

lemma *dep-bin-relE'*:
assumes $(\{\sum\}x : A.\ B\ x)\ R$
obtains $\bigwedge x\ y.\ R\ x\ y \Longrightarrow A\ x \wedge B\ x\ y$
⟨*proof*⟩

lemma *bin-relI* [*intro*]:
assumes $\bigwedge x\ y.\ R\ x\ y \Longrightarrow A\ x$
and $\bigwedge x\ y.\ R\ x\ y \Longrightarrow A\ x \Longrightarrow B\ y$
shows $(A\ \{\times\}\ B)\ R$
⟨*proof*⟩

lemma *bin-rel-if-bin-rel-and*:
assumes $\bigwedge x\ y.\ R\ x\ y \Longrightarrow A\ x \wedge B\ y$
shows $(A\ \{\times\}\ B)\ R$
⟨*proof*⟩

lemma *bin-relE* [*elim*]:
assumes $(A\ \{\times\}\ B)\ R$
and $R\ x\ y$
obtains $A\ x\ B\ y$
⟨*proof*⟩

lemma *bin-relE'*:
assumes $(A\ \{\times\}\ B)\ R$
obtains $\bigwedge x\ y.\ R\ x\ y \Longrightarrow A\ x \wedge B\ y$
⟨*proof*⟩

lemma *dep-bin-rel-cong* [*cong*]:
$[\![A = A';\ \bigwedge x.\ A'\ x \Longrightarrow B\ x = B'\ x]\!] \Longrightarrow (\{\sum\}x : A.\ B\ x) = \{\sum\}x : A'.\ B'\ x$
⟨*proof*⟩

lemma *le-dep-bin-rel-if-le-dom*:

assumes $A \leq A'$
shows $(\{\sum\}x : A.\ B\ x) \leq (\{\sum\}x : A'.\ B\ x)$
⟨*proof*⟩

lemma *dep-bin-rel-covariant-codom*:
assumes $(\{\sum\}x : A.\ B\ x)\ R$
and $\bigwedge x\ y.\ R\ x\ y \Longrightarrow A\ x \Longrightarrow B\ x\ y \Longrightarrow B'\ x\ y$
shows $(\{\sum\}x : A.\ B'\ x)\ R$
⟨*proof*⟩

lemma *mono-dep-bin-rel*: $((\leq) \Rightarrow (\leq) \Rrightarrow (\geq) \Rrightarrow (\leq))$ *dep-bin-rel*
⟨*proof*⟩

lemma *mono-bin-rel*: $((\leq) \Rightarrow (\leq) \Rrightarrow (\geq) \Rrightarrow (\leq))\ (\{\times\})$
⟨*proof*⟩

lemma *in-dom-le-if-dep-bin-rel*:
assumes $(\{\sum\}x : A.\ B\ x)\ R$
shows *in-dom* $R \leq A$
⟨*proof*⟩

lemma *in-codom-le-in-codom-on-if-dep-bin-rel*:
assumes $(\{\sum\}x : A.\ B\ x)\ R$
shows *in-codom* $R \leq$ *in-codom-on* $A\ B$
⟨*proof*⟩

lemma *rel-restrict-left-eq-self-if-dep-bin-rel* [*simp*]:
assumes $(\{\sum\}x : A.\ B\ x)\ R$
shows $R\restriction_A = R$
⟨*proof*⟩

lemma *dep-bin-rel-bottom-dom-iff-eq-bottom* [*iff*]: $(\{\sum\}x : \bot.\ B\ x)\ R \longleftrightarrow R = \bot$
⟨*proof*⟩

lemma *dep-bin-rel-bottom-codom-iff-eq-bottom* [*iff*]: $(\{\sum\}x : A.\ \bot)\ R \longleftrightarrow R = \bot$
⟨*proof*⟩

lemma *mono-bin-rel-dep-bin-rel-bin-rel-in-codom-on-rel-comp*:
$(A\ \{\times\}\ B \Rightarrow (\{\sum\}x : B.\ C\ x) \Rightarrow A\ \{\times\}$ *in-codom-on* $B\ C)\ (\circ\circ)$
⟨*proof*⟩

lemma *mono-bin-rel-bin-rel-bin-rel-rel-comp*: $(A\ \{\times\}\ B \Rightarrow B\ \{\times\}\ C \Rightarrow A\ \{\times\}\ C)$ $(\circ\circ)$
⟨*proof*⟩

lemma *mono-dep-bin-rel-bin-rel-rel-inv*: $((\{\sum\}x : A.\ B\ x) \Rightarrow$ *in-codom-on* $A\ B$ $\{\times\}\ A)$ *rel-inv*
⟨*proof*⟩

lemma *mono-bin-rel-rel-inv*: $(A\ \{\times\}\ B \Rightarrow B\ \{\times\}\ A)$ *rel-inv*
⟨*proof*⟩

lemma *mono-dep-bin-rel-top-dep-bin-rel-inf-rel-restrict-left*:
$((\{\sum\}x : A.\ B\ x) \Rightarrow (P : \top) \Rightarrow (\{\sum\}x : A \sqcap P.\ B\ x))$ *rel-restrict-left*
⟨*proof*⟩

lemma *mono-dep-bin-rel-top-dep-bin-rel-inf-rel-restrict-right*:
$((\{\sum\}x : A.\ B\ x) \Rightarrow (P : \top) \Rightarrow (\{\sum\}x : A.\ B\ x \sqcap P))$ *rel-restrict-right*
⟨*proof*⟩

lemma *mono-dep-bin-rel-top-dep-bin-rel-inf-rel-restrict*:
$((\{\sum\}x : A.\ B\ x) \Rightarrow (P : \top) \Rightarrow (\{\sum\}x : A \sqcap P.\ B\ x \sqcap P))$ *rel-restrict*
⟨*proof*⟩

lemma *le-if-rel-agree-on-if-dep-bin-relI*:
assumes $(\{\sum\}x : A.\ B\ x)\ R$
and *rel-agree-on* $A\ \mathcal{R}$
and $\mathcal{R}\ R\ \mathcal{R}\ R'$
shows $R \leq R'$
⟨*proof*⟩

lemma *eq-if-rel-agree-on-if-dep-bin-relI*:
assumes $(\{\sum\}x : A.\ B\ x)\ R\ (\{\sum\}x : A.\ B'\ x)\ R'$
and *rel-agree-on* $A\ \mathcal{R}$
and $\mathcal{R}\ R\ \mathcal{R}\ R'$
shows $R = R'$
⟨*proof*⟩

end

1.2.7 Extensions

theory *Binary-Relations-Extend*
imports
Dependent-Binary-Relations
begin

consts *extend* :: $'a \Rightarrow 'b \Rightarrow 'c \Rightarrow 'c$

definition *extend-rel* $x\ y\ R\ x'\ y' \equiv (x = x' \wedge y = y') \vee R\ x'\ y'$
adhoc-overloading *extend* $\rightleftharpoons$ *extend-rel*

lemma *extend-leftI* [*iff*]: $(extend\ x\ y\ R)\ x\ y$
⟨*proof*⟩

lemma *extend-rightI* [*intro*]:
assumes $R\ x'\ y'$

shows (*extend x y R*) *x′ y′*
⟨*proof*⟩

lemma *extendE* [*elim*]:
assumes (*extend x y R*) *x′ y′*
obtains $x = x'\ y = y' \mid x \neq x' \vee y \neq y'\ R\ x'\ y'$
⟨*proof*⟩

lemma *extend-eq-self-if-rel* [*simp*]: *R x y* $\Longrightarrow$ *extend x y R* = *R*
⟨*proof*⟩

context
fixes *x* :: *′a* **and** *y* :: *′b* **and** *R* :: *′a* ⇒ *′b* ⇒ *bool*
begin

lemma *in-dom-extend-eq*: *in-dom* (*extend x y R*) = *in-dom R* ⊔ (=) *x*
⟨*proof*⟩

lemma *in-dom-extend-iff*: *in-dom* (*extend x y R*) *x′* $\longleftrightarrow$ *in-dom R x′* ∨ *x* = *x′*
⟨*proof*⟩

lemma *codom-extend-eq*: *in-codom* (*extend x y R*) = *in-codom R* ⊔ (=) *y*
⟨*proof*⟩

lemma *in-codom-extend-iff*: *in-codom* (*extend x y R*) *y′* $\longleftrightarrow$ *in-codom R y′* ∨ *y* = *y′*
⟨*proof*⟩

end

lemma *in-field-extend-eq*: *in-field* (*extend x y R*) = *in-field R* ⊔ (=) *x* ⊔ (=) *y*
⟨*proof*⟩

lemma *in-field-extend-iff*: *in-field* (*extend x y R*) *z* $\longleftrightarrow$ *in-field R z* ∨ *z* = *x* ∨ *z* = *y*
⟨*proof*⟩

lemma *mono-extend*: *mono* (*extend x y* :: (*′a* ⇒ *′b* ⇒ *bool*) ⇒ *′a* ⇒ *′b* ⇒ *bool*)
⟨*proof*⟩

lemma *dep-mono-dep-bin-rel-extend*:
((*x* : *A*) ⇒ *B x* ⇒ ({$\sum$}*x* : *A′*. *B′ x*) ⇒ ({$\sum$}*x* : *A* ⊔ *A′*. (*B* ⊔ *B′*) *x*)) *extend*
⟨*proof*⟩

consts *glue* :: *′a* ⇒ *′b*

definition *glue-rel* (*R* :: (*′a* ⇒ *′b* ⇒ *bool*) ⇒ *bool*) *x* ≡ *in-codom-on R* (*λR*. *R x*)
adhoc-overloading *glue* ⇌ *glue-rel*

lemma *glue-rel-eq-in-codom-on*: *glue* $\mathcal{R}$ x = *in-codom-on* $\mathcal{R}$ ($\lambda R.$ R x)
⟨*proof*⟩

lemma *glueI* [*intro*]:
assumes $\mathcal{R}$ R
and R x y
shows *glue* $\mathcal{R}$ x y
⟨*proof*⟩

lemma *glueE* [*elim*!]:
assumes *glue* $\mathcal{R}$ x y
obtains R **where** $\mathcal{R}$ R R x y
⟨*proof*⟩

lemma *glue-bottom-eq* [*simp*]: *glue* ($\bot$ :: ('a ⇒ 'b ⇒ *bool*) ⇒ *bool*) = $\bot$
⟨*proof*⟩

lemma *glue-eq-rel-eq-self* [*simp*]: *glue* ((=) R) = (R :: 'a ⇒ 'b ⇒ *bool*)
⟨*proof*⟩

lemma *glue-sup-eq-glue-sup-glue* [*simp*]: *glue* (A ⊔ B) = *glue* A ⊔ *glue* B
⟨*proof*⟩

lemma *mono-glue*: *mono* (*glue* :: (('a ⇒ 'b ⇒ *bool*) ⇒ *bool*) ⇒ ('a ⇒ 'b ⇒ *bool*))
⟨*proof*⟩

lemma *dep-bin-rel-glueI*:
fixes $\mathcal{R}$ **defines** [*simp*]: D ≡ *in-codom-on* $\mathcal{R}$ *in-dom*
assumes $\bigwedge R.$ $\mathcal{R}$ R ⟹ ∃ $A.$ ({$\sum$}x : $A.$ B x) R
shows ({$\sum$}x : $D.$ B x) (*glue* $\mathcal{R}$)
⟨*proof*⟩

end

Right Unique

theory *Binary-Relations-Right-Unique*
imports
Binary-Relations-Injective
Binary-Relations-Extend
begin

consts *right-unique-on* :: 'a ⇒ 'b ⇒ *bool*

overloading
right-unique-on-pred ≡ *right-unique-on* :: ('a ⇒ *bool*) ⇒ ('a ⇒ 'b ⇒ *bool*) ⇒ *bool*
begin
definition *right-unique-on-pred* P R ≡ ∀ x : $P.$ ∀ y y'. R x y ∧ R x y' ⟶ y = y'

end

lemma *right-unique-onI* [*intro*]:
assumes $\bigwedge x\ y\ y'.\ P\ x \Longrightarrow R\ x\ y \Longrightarrow R\ x\ y' \Longrightarrow y = y'$
shows *right-unique-on* $P\ R$
⟨*proof*⟩

lemma *right-unique-onD*:
assumes *right-unique-on* $P\ R$
and $P\ x$
and $R\ x\ y\ R\ x\ y'$
shows $y = y'$
⟨*proof*⟩

lemma *antimono-right-unique-on*:
$((\leq) \Rightarrow (\leq) \Rrightarrow (\geq))$ (*right-unique-on* :: $('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool$)
⟨*proof*⟩

lemma *mono-right-unique-on-top-right-unique-on-inf-rel-restrict-left*:
$((R : right\text{-}unique\text{-}on\ P) \Rightarrow (P' : \top) \Rightarrow right\text{-}unique\text{-}on\ (P \sqcap P'))\ rel\text{-}restrict\text{-}left$
⟨*proof*⟩

lemma *mono-right-unique-on-comp*:
$((R : right\text{-}unique\text{-}on\ P) \Rightarrow right\text{-}unique\text{-}on\ (in\text{-}codom\ (R\restriction_P)) \Rightarrow right\text{-}unique\text{-}on\ P)\ (\circ\circ)$
⟨*proof*⟩

context
fixes $P :: 'a \Rightarrow bool$ **and** $\mathcal{R} :: ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool$
begin

lemma *right-unique-on-glue-if-right-unique-on-sup*:
assumes $\bigwedge R\ R'.\ \mathcal{R}\ R \Longrightarrow \mathcal{R}\ R' \Longrightarrow right\text{-}unique\text{-}on\ P\ (R \sqcup R')$
shows *right-unique-on* P (*glue* $\mathcal{R}$)
⟨*proof*⟩

lemma *right-unique-on-if-right-unique-on-glue*:
assumes *right-unique-on* P (*glue* $\mathcal{R}$)
and $\mathcal{R}\ R$
shows *right-unique-on* $P\ R$
⟨*proof*⟩

end

consts *right-unique-at* :: $'a \Rightarrow 'b \Rightarrow bool$

overloading
right-unique-at-pred ≡ *right-unique-at* :: $('a \Rightarrow bool) \Rightarrow ('b \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
begin

definition *right-unique-at-pred* $P\ R \equiv \forall\, x\ y\ y'.\ P\ y \wedge P\ y' \wedge R\ x\ y \wedge R\ x\ y' \longrightarrow y = y'$
end

lemma *right-unique-atI* [*intro*]:
assumes $\bigwedge x\ y\ y'.\ P\ y \Longrightarrow P\ y' \Longrightarrow R\ x\ y \Longrightarrow R\ x\ y' \Longrightarrow y = y'$
shows *right-unique-at* $P\ R$
⟨*proof*⟩

lemma *right-unique-atD*:
assumes *right-unique-at* $P\ R$
and $P\ y$
and $P\ y'$
and $R\ x\ y\ R\ x\ y'$
shows $y = y'$
⟨*proof*⟩

lemma *right-unique-at-rel-inv-iff-rel-injective-on* [*iff*]:
right-unique-at $(P :: 'a \Rightarrow bool)$ $(R^{-1} :: 'b \Rightarrow 'a \Rightarrow bool) \longleftrightarrow$ *rel-injective-on* $P\ R$
⟨*proof*⟩

lemma *rel-injective-on-rel-inv-iff-right-unique-at* [*iff*]:
rel-injective-on $(P :: 'a \Rightarrow bool)$ $(R^{-1} :: 'a \Rightarrow 'b \Rightarrow bool) \longleftrightarrow$ *right-unique-at* $P\ R$
⟨*proof*⟩

lemma *right-unique-on-rel-inv-iff-rel-injective-at* [*iff*]:
right-unique-on $(P :: 'a \Rightarrow bool)$ $(R^{-1} :: 'a \Rightarrow 'b \Rightarrow bool) \longleftrightarrow$ *rel-injective-at* $P\ R$
⟨*proof*⟩

lemma *rel-injective-at-rel-inv-iff-right-unique-on* [*iff*]:
rel-injective-at $(P :: 'b \Rightarrow bool)$ $(R^{-1} :: 'a \Rightarrow 'b \Rightarrow bool) \longleftrightarrow$ *right-unique-on* $P\ R$
⟨*proof*⟩

lemma *right-unique-on-if-Fun-Rel-imp-if-right-unique-at*:
assumes *right-unique-at* $(Q :: 'b \Rightarrow bool)$ $(R :: 'a \Rightarrow 'b \Rightarrow bool)$
and $(R \Rrightarrow (\longrightarrow))\ P\ Q$
shows *right-unique-on* $P\ R$
⟨*proof*⟩

lemma *right-unique-at-if-Fun-Rel-rev-imp-if-right-unique-on*:
assumes *right-unique-on* $(P :: 'a \Rightarrow bool)$ $(R :: 'a \Rightarrow 'b \Rightarrow bool)$
and $(R \Rrightarrow (\longleftarrow))\ P\ Q$
shows *right-unique-at* $Q\ R$
⟨*proof*⟩

consts *right-unique* :: *'a ⇒ bool*

overloading
right-unique ≡ right-unique :: *('a ⇒ 'b ⇒ bool) ⇒ bool*
begin
definition (*right-unique* :: *('a ⇒ 'b ⇒ bool) ⇒ bool*) ≡ *right-unique-on* (⊤ :: *'a ⇒ bool*)
end

lemma *right-unique-eq-right-unique-on*:
(*right-unique* :: *('a ⇒ 'b ⇒ bool) ⇒ -*) = *right-unique-on* (⊤ :: *'a ⇒ bool*)
⟨*proof*⟩

lemma *right-unique-eq-right-unique-on-uhint* [*uhint*]:
assumes *P* ≡ (⊤ :: *'a ⇒ bool*)
shows *right-unique* :: *('a ⇒ 'b ⇒ bool) ⇒ -* ≡ *right-unique-on P*
⟨*proof*⟩

lemma *right-uniqueI* [*intro*]:
assumes ⋀*x y y'*. *R x y* ⟹ *R x y'* ⟹ *y* = *y'*
shows *right-unique R*
⟨*proof*⟩

lemma *right-uniqueD*:
assumes *right-unique R*
and *R x y R x y'*
shows *y* = *y'*
⟨*proof*⟩

lemma *right-unique-eq-right-unique-at*:
(*right-unique* :: *('a ⇒ 'b ⇒ bool) ⇒ bool*) = *right-unique-at* (⊤ :: *'b ⇒ bool*)
⟨*proof*⟩

lemma *right-unique-eq-right-unique-at-uhint* [*uhint*]:
assumes *P* ≡ (⊤ :: *'b ⇒ bool*)
shows *right-unique* :: *('a ⇒ 'b ⇒ bool) ⇒ -* ≡ *right-unique-at P*
⟨*proof*⟩

lemma *right-unique-on-if-right-unique*:
fixes *P* :: *'a ⇒ bool* **and** *R* :: *'a ⇒ 'b ⇒ bool*
assumes *right-unique R*
shows *right-unique-on P R*
⟨*proof*⟩

lemma *right-unique-at-if-right-unique*:
fixes *P* :: *'a ⇒ bool* **and** *R* :: *'b ⇒ 'a ⇒ bool*
assumes *right-unique R*
shows *right-unique-at P R*

⟨*proof*⟩

lemma *right-unique-if-right-unique-on-in-dom*:
assumes *right-unique-on* (*in-dom R*) *R*
shows *right-unique R*
⟨*proof*⟩

lemma *right-unique-if-right-unique-at-in-codom*:
assumes *right-unique-at* (*in-codom R*) *R*
shows *right-unique R*
⟨*proof*⟩

corollary *right-unique-on-in-dom-iff-right-unique* [*iff*]:
right-unique-on (*in-dom R*) *R* ⟷ *right-unique R*
⟨*proof*⟩

corollary *right-unique-at-in-codom-iff-right-unique* [*iff*]:
right-unique-at (*in-codom R*) *R* ⟷ *right-unique R*
⟨*proof*⟩

lemma *right-unique-rel-inv-iff-rel-injective* [*iff*]:
right-unique R^{-1} ⟷ *rel-injective R*
⟨*proof*⟩

lemma *rel-injective-rel-inv-iff-right-unique* [*iff*]:
rel-injective R^{-1} ⟷ *right-unique R*
⟨*proof*⟩

Instantiations **lemma** *right-unique-eq*: *right-unique* (=)
⟨*proof*⟩

end

1.2.8 Evaluation of Functions as Binary Relations

theory *Binary-Relations-Function-Evaluation*
imports
Binary-Relations-Right-Unique
Binary-Relations-Extend
begin

consts *eval* :: $'a \Rightarrow 'b \Rightarrow 'c$

definition *eval-rel R x* ≡ *THE y*. *R x y*
adhoc-overloading *eval* ⇌ *eval-rel*

open-bundle *eval-syntax*
begin

notation *eval* (‹$'($‘$'$)›)
notation *eval* (‹(-‘-)› [*999*, *1000*] *999*)
end

lemma *eval-eq-if-right-unique-onI*:
assumes *right-unique-on P R*
and *P x*
and *R x y*
shows *R*‘*x* = *y*
⟨*proof*⟩

lemma *eval-eq-if-right-unique-on-eqI*:
assumes *right-unique-on* ((=) *x*) *R*
and *R x y*
shows *R*‘*x* = *y*
⟨*proof*⟩

lemma *rel-eval-if-ex1*:
assumes ∃!*y*. *R x y*
shows *R x* (*R*‘*x*)
⟨*proof*⟩

lemma *rel-if-eval-eq-if-in-dom-if-right-unique-on-eq*:
assumes *right-unique-on* ((=) *x*) *R*
and *in-dom R x*
and *R*‘*x* = *y*
shows *R x y*
⟨*proof*⟩

corollary *rel-eval-if-in-dom-if-right-unique-on-eq*:
assumes *right-unique-on* ((=) *x*) *R*
and *in-dom R x*
shows *R x* (*R*‘*x*)
⟨*proof*⟩

lemma *eq-app-eval-eq-eq* [*simp*]: (λ*x*. (=) (*f x*))‘*x* = *f x*
⟨*proof*⟩

lemma *extend-eval-eq-if-not-in-dom* [*simp*]:
assumes ¬(*in-dom R x*)
shows (*extend x y R*)‘*x* = *y*
⟨*proof*⟩

corollary *extend-bottom-eval-eq* [*simp*]:
fixes *x* :: *'a* **and** *y* :: *'b*
shows (*extend x y* (⊥ :: *'a* ⇒ *'b* ⇒ *bool*))‘*x* = *y*
⟨*proof*⟩

lemma *glue-eval-eqI*:

assumes *right-unique-on P* (*glue* $\mathcal{R}$)
and $\mathcal{R}$ *R*
and *P x*
and *R x y*
shows (*glue* $\mathcal{R}$)‘*x* = *y*
⟨*proof*⟩

lemma *glue-eval-eq-evalI*:
assumes *right-unique-on P* (*glue* $\mathcal{R}$)
and $\mathcal{R}$ *R*
and *P x*
and *in-dom R x*
shows (*glue* $\mathcal{R}$)‘*x* = *R*‘*x*
⟨*proof*⟩

Note: the following rest on the definition of extend and eval:

lemma *extend-eval-eq-if-neq* [*simp*]:
fixes *R* :: *'a* ⇒ *'b* ⇒ *bool*
shows *x* ≠ *y* ⟹ (*extend y z R*)‘*x* = *R*‘*x*
⟨*proof*⟩

lemma *sup-eval-eq-left-eval-if-not-in-dom* [*simp*]:
fixes *R S* :: *'a* ⇒ *'b* ⇒ *bool*
shows ¬(*in-dom S x*) ⟹ (*R* ⊔ *S*)‘*x* = *R*‘*x*
⟨*proof*⟩

lemma *sup-eval-eq-right-eval-if-not-in-dom* [*simp*]:
fixes *R S* :: *'a* ⇒ *'b* ⇒ *bool*
shows ¬(*in-dom R x*) ⟹ (*R* ⊔ *S*)‘*x* = *S*‘*x*
⟨*proof*⟩

lemma *rel-restrict-left-eval-eq-if-pred* [*simp*]:
fixes *R* :: *'a* ⇒ *'b* ⇒ *bool*
assumes *P x*
shows (*R*↾$_P$)‘*x* = *R*‘*x*

⟨*proof*⟩

end

Left Total

theory *Binary-Relations-Left-Total*
imports
Functions-Monotone
begin

consts *left-total-on* :: *'a* ⇒ *'b* ⇒ *bool*

overloading

left-total-on-pred ≡ *left-total-on* :: ('*a* ⇒ *bool*) ⇒ ('*a* ⇒ '*b* ⇒ *bool*) ⇒ *bool*
begin
definition *left-total-on-pred P R* ≡ ∀ *x* : *P*. *in-dom R x*
end

lemma *left-total-onI* [*intro*]:
assumes ⋀*x*. *P x* ⟹ *in-dom R x*
shows *left-total-on P R*
⟨*proof*⟩

lemma *left-total-onE* [*elim*]:
assumes *left-total-on P R*
and *P x*
obtains *y* **where** *R x y*
⟨*proof*⟩

lemma *le-in-dom-if-left-total-on*:
assumes *left-total-on P R*
shows *P* ≤ *in-dom R*
⟨*proof*⟩

lemma *left-total-on-if-le-in-dom*:
assumes *P* ≤ *in-dom R*
shows *left-total-on P R*
⟨*proof*⟩

lemma *mono-left-total-on*:
((≥) ⇒ (≤) ⇛ (≤)) (*left-total-on* :: ('*a* ⇒ *bool*) ⇒ ('*a* ⇒ '*b* ⇒ *bool*) ⇒ *bool*)
⟨*proof*⟩

lemma *le-in-dom-iff-left-total-on*: *P* ≤ *in-dom R* ⟷ *left-total-on P R*
⟨*proof*⟩

lemma *mono-left-total-on-top-left-total-on-inf-rel-restrict-left*:
((*R* : *left-total-on P*) ⇒ (*P'* : ⊤) ⇒ *left-total-on* (*P* ⊓ *P'*)) *rel-restrict-left*
⟨*proof*⟩

lemma *mono-left-total-on-comp*:
((*R* : *left-total-on P*) ⇒ *left-total-on* (*in-codom* (*R*↾$_P$)) ⇒ *left-total-on P*) (∘∘)
⟨*proof*⟩

consts *left-total* :: '*a* ⇒ *bool*

overloading
left-total ≡ *left-total* :: ('*a* ⇒ '*b* ⇒ *bool*) ⇒ *bool*
begin
definition (*left-total* :: ('*a* ⇒ '*b* ⇒ *bool*) ⇒ *bool*) ≡ *left-total-on* (⊤ :: '*a* ⇒ *bool*)
end

lemma *left-total-eq-left-total-on*:
(*left-total* :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ -) = *left-total-on* (⊤ :: *'a* ⇒ *bool*)
⟨*proof*⟩

lemma *left-total-eq-left-total-on-uhint* [*uhint*]:
assumes *P* ≡ ⊤ :: *'a* ⇒ *bool*
shows *left-total* :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ - ≡ *left-total-on P*
⟨*proof*⟩

lemma *left-totalI* [*intro*]:
assumes ⋀*x*. *in-dom R x*
shows *left-total R*
⟨*proof*⟩

lemma *left-totalE*:
assumes *left-total R*
obtains *y* **where** *R x y*
⟨*proof*⟩

lemma *in-dom-if-left-total*:
assumes *left-total R*
shows *in-dom R x*
⟨*proof*⟩

lemma *left-total-on-if-left-total*:
fixes *P* :: *'a* ⇒ *bool* **and** *R* :: *'a* ⇒ *'b* ⇒ *bool*
assumes *left-total R*
shows *left-total-on P R*
⟨*proof*⟩

end

1.3 Functions as Binary Relations

theory *Binary-Relations-Function-Base*
imports
Binary-Relations-Function-Evaluation
Binary-Relations-Left-Total
begin

Relational functions may contain further elements outside their specification.

consts *rel-dep-mono-wrt* :: *'a* ⇒ *'b* ⇒ *'c*
consts *rel-mono-wrt* :: *'a* ⇒ *'b* ⇒ *'c*

open-bundle *rel-mono-wrt-syntax*
begin
notation *rel-mono-wrt* (**infixr** ‹→› *40*)

syntax
-rel-dep-mono-wrt-pred :: *idt* ⇒ *'a* ⇒ *'b* ⇒ *'c* ⇒ *bool* (‹*'*(*-*/ :/ *-'*) → (*-*)› [*41*, *41*, *40*] *40*)
end
syntax-consts *-rel-dep-mono-wrt-pred* ⇌ *rel-dep-mono-wrt*
translations
(*x* : *A*) → *B* ⇌ *CONST rel-dep-mono-wrt A* (λ*x*. *B*)

definition *rel-dep-mono-wrt-pred* (*A* :: *'a* ⇒ *bool*) (*B* :: *'a* ⇒ *'b* ⇒ *bool*) (*R* :: *'a* ⇒ *'b* ⇒ *bool*) ≡
left-total-on A R ∧ *right-unique-on A R* ∧ ((*x* : *A*) ⇒ *B x*) (*eval R*)
adhoc-overloading *rel-dep-mono-wrt* ⇌ *rel-dep-mono-wrt-pred*

definition *rel-mono-wrt-pred* (*A* :: *'a* ⇒ *bool*) (*B* :: *'b* ⇒ *bool*) :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ *bool* ≡
rel-dep-mono-wrt-pred A (λ(*-* :: *'a*). *B*)
adhoc-overloading *rel-mono-wrt* ⇌ *rel-mono-wrt-pred*

lemma *rel-mono-wrt-pred-eq-rel-dep-mono-wrt-pred*:
(((*A* :: *'a* ⇒ *bool*) → (*B* :: *'b* ⇒ *bool*)) :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ *bool*) = (((*-* :: *'a*) : *A*) → *B*)
⟨*proof*⟩

lemma *rel-mono-wrt-pred-eq-rel-dep-mono-wrt-pred-uhint* [*uhint*]:
assumes (*A* :: *'a* ⇒ *bool*) ≡ *A'*
and ⋀*x*. *B* :: *'b* ⇒ *bool* ≡ *B' x*
shows (*A* → *B*) ≡ ((*x* : *A'*) → *B' x*)
⟨*proof*⟩

lemma *rel-mono-wrt-pred-iff-rel-dep-mono-wrt-pred*:
((*A* :: *'a* ⇒ *bool*) → (*B* :: *'b* ⇒ *bool*)) (*R* :: *'a* ⇒ *'b* ⇒ *bool*) ⟷ (((*-* :: *'a*) : *A*) → *B*) *R*
⟨*proof*⟩

lemma *rel-dep-mono-wrt-predI* [*intro*]:
assumes *left-total-on A R*
and *right-unique-on A R*
and ((*x* : *A*) ⇒ *B x*) (*eval R*)
shows ((*x* : *A*) → *B x*) *R*
⟨*proof*⟩

lemma *rel-dep-mono-wrt-predE* [*elim*]:
assumes ((*x* : *A*) → *B x*) *R*
obtains *left-total-on A R right-unique-on A R* ((*x* : *A*) ⇒ *B x*) (*eval R*)
⟨*proof*⟩

lemma *rel-dep-mono-wrt-pred-cong* [*cong*]:
assumes *A* = *A'*
and ⋀*x y*. *A' x* ⟹ *B x* = *B' x*

shows $((x : A) \to B\ x) = ((x : A') \to B'\ x)$
⟨*proof*⟩

lemma *rel-mono-wrt-predI* [*intro*]:
assumes *left-total-on A R*
and *right-unique-on A R*
and $(A \Rightarrow B)$ $(eval\ R)$
shows $(A \to B)\ R$
⟨*proof*⟩

lemma *rel-mono-wrt-predE* [*elim*]:
assumes $(A \to B)\ R$
obtains *left-total-on A R right-unique-on A R* $(A \Rightarrow B)$ $(eval\ R)$
⟨*proof*⟩

lemma *mono-rel-dep-mono-wrt-pred-dep-mono-wrt-pred-eval*: $(((x : A) \to B\ x) \Rightarrow (x : A) \Rightarrow B\ x)$ *eval*
⟨*proof*⟩

lemma *ex1-rel-right-if-rel-dep-mono-wrt-predI*:
assumes $((x : A) \to B\ x)\ R$
and $A\ x$
shows $\exists!y.\ R\ x\ y$
⟨*proof*⟩

lemma *eq-if-rel-if-rel-if-rel-dep-mono-wrt-predI*:
assumes $((x : A) \to B\ x)\ R$
and $A\ x$
and $R\ x\ y\ R\ x\ y'$
shows $y = y'$
⟨*proof*⟩

lemma *eval-eq-if-rel-if-rel-dep-mono-wrt-predI* [*simp*]:
assumes $((x : A) \to B\ x)\ R$
and $A\ x$
and $R\ x\ y$
shows $R`x = y$
⟨*proof*⟩

lemma *rel-if-eval-eq-if-rel-dep-mono-wrt-predI*:
assumes $((x : A) \to B\ x)\ R$
and $A\ x$
and $R`x = y$
shows $R\ x\ y$
⟨*proof*⟩

corollary *rel-eval-if-rel-dep-mono-wrt-predI*:
assumes $((x : A) \to B\ x)\ R$
and $A\ x$

shows $R\ x\ (R`x)$
⟨*proof*⟩

corollary *rel-iff-eval-eq-if-rel-dep-mono-wrt-predI*:
assumes $((x : A) \to B\ x)\ R$
and $A\ x$
shows $R\ x\ y \longleftrightarrow R`x = y$
⟨*proof*⟩

lemma *rel-dep-mono-wrt-pred-relE*:
assumes $((x : A) \to B\ x)\ R$
and $A\ x$
and $R\ x\ y$
obtains $B\ x\ y\ R`x = y$
⟨*proof*⟩

lemma *rel-dep-mono-wrt-pred-relE'*:
assumes $((x : A) \to B\ x)\ R$
obtains $\bigwedge x\ y.\ A\ x \Longrightarrow R\ x\ y \Longrightarrow B\ x\ y \wedge R`x = y$
⟨*proof*⟩

lemma *rel-codom-if-rel-if-pred-if-rel-dep-mono-wrt-pred*:
assumes $((x : A) \to B\ x)\ R$
and $A\ x$
and $R\ x\ y$
shows $B\ x\ y$
⟨*proof*⟩

lemma *rel-dep-mono-wrt-pred-contravariant-dom*:
assumes $((x : A) \to B\ x)\ R$
and [*dest*]: $\bigwedge x.\ A'\ x \Longrightarrow A\ x$
shows $((x : A') \to B\ x)\ R$
⟨*proof*⟩

lemma *rel-dep-mono-wrt-pred-covariant-codom*:
assumes $((x : A) \to B\ x)\ R$
and $\bigwedge x.\ A\ x \Longrightarrow B\ x\ (R`x) \Longrightarrow B'\ x\ (R`x)$
shows $((x : A) \to B'\ x)\ R$
⟨*proof*⟩

lemma *rel-mono-wrt-pred-in-codom-on-if-rel-dep-mono-wrt-pred*:
assumes $((x : A) \to B\ x)\ R$
shows $(A \to \textit{in-codom-on}\ A\ B)\ R$
⟨*proof*⟩

lemma *eq-comp-eval-restrict-left-le-if-rel-dep-mono-wrt-pred*:
assumes $((x : A) \to B\ x)\ R$
shows $((=) \circ \textit{eval}\ R)\restriction_A \leq R\restriction_A$
⟨*proof*⟩

lemma *restrict-left-le-eq-comp-eval-restrict-left-if-rel-dep-mono-wrt-pred*:
assumes $((x : A) \to B\ x)\ R$
shows $R{\restriction_A} \leq ((=) \circ eval\ R){\restriction_A}$
⟨*proof*⟩

corollary *restrict-left-eq-eq-comp-eval-if-rel-dep-mono-wrt-pred*:
assumes $((x : A) \to B\ x)\ R$
shows $R{\restriction_A} = ((=) \circ eval\ R){\restriction_A}$
⟨*proof*⟩

lemma *eval-eq-if-rel-dep-mono-wrt-predI*:
fixes $R :: 'a \Rightarrow 'b \Rightarrow bool$
assumes $((x : A) \to B\ x)\ R\ ((x : A') \to B'\ x)\ R'$
and $R \leq R'$
and $(A \sqcap A')\ x$
shows $R`x = R'`x$
⟨*proof*⟩

lemma *rel-agree-on-if-eval-eq-if-rel-dep-mono-wrt-pred*:
assumes *rel-dep-mono-wrt-pred*: $\bigwedge R.\ \mathcal{R}\ R \Longrightarrow \exists B.\ ((x : A) \to B\ x)\ R$
and $\bigwedge R\ R'\ x.\ \mathcal{R}\ R \Longrightarrow \mathcal{R}\ R' \Longrightarrow A\ x \Longrightarrow R`x = R'`x$
shows *rel-agree-on A* $\mathcal{R}$
⟨*proof*⟩

lemma *mono-rel-dep-mono-wrt-pred-top-rel-dep-mono-wrt-pred-inf-rel-restrict-left*:
$(((x : A) \to B\ x) \Rightarrow (A' : \top) \Rightarrow ((x : A \sqcap A') \to B\ x))$ *rel-restrict-left*
⟨*proof*⟩

end

1.3.1 Clean Functions

theory *Binary-Relations-Clean-Functions*
imports
Binary-Relations-Function-Base
begin

Clean relational functions may not contain further elements outside their specification.

consts *crel-dep-mono-wrt* :: $'a \Rightarrow 'b \Rightarrow 'c$
consts *crel-mono-wrt* :: $'a \Rightarrow 'b \Rightarrow 'c$

open-bundle *crel-mono-wrt-syntax*
begin
notation *crel-mono-wrt* (**infixr** $\langle\to_c\rangle$ *50*)
syntax
-crel-dep-mono-wrt :: *idt* $\Rightarrow 'a \Rightarrow 'b \Rightarrow 'c \Rightarrow$ *bool* ($\langle'(-/\ :/\ -')\ \to_c\ (-)\rangle$ [*51*, *50*, *50*] *50*)

end
syntax-consts
-crel-dep-mono-wrt ⇌ *crel-dep-mono-wrt*
translations
$(x : A) \to_c B \rightleftharpoons CONST\ crel\text{-}dep\text{-}mono\text{-}wrt\ A\ (\lambda x.\ B)$

definition *crel-dep-mono-wrt-pred* $(A :: 'a \Rightarrow bool)\ B\ R \equiv ((x : A) \to B\ x)\ R \wedge in\text{-}dom\ R = A$
adhoc-overloading *crel-dep-mono-wrt* ⇌ *crel-dep-mono-wrt-pred*

definition *crel-mono-wrt-pred* $(A :: 'a \Rightarrow bool)\ B \equiv (((- :: 'a) : A) \to_c B)$
adhoc-overloading *crel-mono-wrt* ⇌ *crel-mono-wrt-pred*

lemma *crel-mono-wrt-pred-eq-crel-dep-mono-wrt-pred*:
$(((A :: 'a \Rightarrow bool) \to_c (B :: 'b \Rightarrow bool)) :: ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool) = (((- :: 'a) : A) \to_c B)$
⟨*proof*⟩

lemma *crel-mono-wrt-pred-eq-crel-dep-mono-wrt-pred-uhint* [*uhint*]:
assumes $(A :: 'a \Rightarrow bool) \equiv A'$
and $\bigwedge x.\ B \equiv B'\ x$
shows $(A \to_c B) \equiv ((x : A') \to_c B'\ x)$
⟨*proof*⟩

lemma *crel-mono-wrt-pred-iff-crel-dep-mono-wrt-pred*:
$((A :: 'a \Rightarrow bool) \to_c (B :: 'b \Rightarrow bool))\ (R :: 'a \Rightarrow 'b \Rightarrow bool) \longleftrightarrow (((- :: 'a) : A) \to_c B)\ R$
⟨*proof*⟩

lemma *crel-dep-mono-wrt-predI* [*intro*]:
assumes $((x : A) \to B\ x)\ R$
and $in\text{-}dom\ R \leq A$
shows $((x : A) \to_c B\ x)\ R$
⟨*proof*⟩

lemma *crel-dep-mono-wrt-predI'*:
assumes *left-total-on A R*
and *right-unique-on A R*
and $(\{\sum\}x : A.\ B\ x)\ R$
shows $((x : A) \to_c B\ x)\ R$
⟨*proof*⟩

lemma *crel-dep-mono-wrt-predE*:
assumes $((x : A) \to_c B\ x)\ R$
obtains $((x : A) \to B\ x)\ R\ in\text{-}dom\ R = A$
⟨*proof*⟩

lemma *crel-dep-mono-wrt-predE'* [*elim*]:
notes *crel-dep-mono-wrt-predE*[*elim*]

assumes $((x : A) \to_c B\ x)\ R$
obtains $((x : A) \to B\ x)\ R\ (\{\sum\}x : A.\ B\ x)\ R$
⟨*proof*⟩

lemma *crel-dep-mono-wrt-pred-cong* [*cong*]:
assumes $A = A'$
and $\bigwedge x\ y.\ A'\ x \Longrightarrow B\ x = B'\ x$
shows $((x : A) \to_c B\ x) = ((x : A') \to_c B'\ x)$
⟨*proof*⟩

lemma *in-dom-eq-if-crel-dep-mono-wrt-pred* [*simp*]:
assumes $((x : A) \to_c B\ x)\ R$
shows *in-dom* $R = A$
⟨*proof*⟩

lemma *in-codom-le-in-codom-on-if-crel-dep-mono-wrt-pred*:
assumes $((x : A) \to_c B\ x)\ R$
shows *in-codom* $R \leq$ *in-codom-on* $A\ B$
⟨*proof*⟩

lemma *crel-mono-wrt-predI* [*intro*]:
assumes $(A \to B)\ R$
and *in-dom* $R \leq A$
shows $(A \to_c B)\ R$
⟨*proof*⟩

lemma *crel-mono-wrt-predI'*:
assumes *left-total-on* $A\ R$
and *right-unique-on* $A\ R$
and $(A\ \{\times\}\ B)\ R$
shows $(A \to_c B)\ R$
⟨*proof*⟩

lemma *crel-mono-wrt-predE*:
assumes $(A \to_c B)\ R$
obtains $(A \to B)\ R$ *in-dom* $R = A$
⟨*proof*⟩

lemma *crel-mono-wrt-predE'* [*elim*]:
assumes $(A \to_c B)\ R$
obtains $(A \to B)\ R\ (A\ \{\times\}\ B)\ R$
⟨*proof*⟩

lemma *in-dom-eq-if-crel-mono-wrt-pred* [*simp*]:
assumes $(A \to_c B)\ R$
shows *in-dom* $R = A$
⟨*proof*⟩

lemma *eq-if-rel-if-rel-if-crel-dep-mono-wrt-predI*:

assumes $((x : A) \to_c B\ x)\ R$
and $R\ x\ y\ R\ x\ y'$
shows $y = y'$
⟨*proof*⟩

lemma *eval-eq-if-rel-if-crel-dep-mono-wrt-predI* [*simp*]:
assumes $((x : A) \to_c B\ x)\ R$
and $R\ x\ y$
shows $R`x = y$
⟨*proof*⟩

lemma *crel-dep-mono-wrt-pred-relE*:
assumes $((x : A) \to_c B\ x)\ R$
and $R\ x\ y$
obtains $A\ x\ B\ x\ y\ R`x = y$
⟨*proof*⟩

lemma *crel-dep-mono-wrt-pred-relE'*:
assumes $((x : A) \to_c B\ x)\ R$
obtains $\bigwedge x\ y.\ R\ x\ y \Longrightarrow A\ x \wedge B\ x\ y \wedge R`x = y$
⟨*proof*⟩

lemma *rel-restrict-left-eq-self-if-crel-dep-mono-wrt-pred* [*simp*]:
assumes $((x : A) \to_c B\ x)\ R$
shows $R\restriction_A = R$
⟨*proof*⟩

Note: clean function relations are not contravariant on their domain.

lemma *crel-dep-mono-wrt-pred-covariant-codom*:
assumes $((x : A) \to_c B\ x)\ R$
and $\bigwedge x.\ A\ x \Longrightarrow B\ x\ (R`x) \Longrightarrow B'\ x\ (R`x)$
shows $((x : A) \to_c B'\ x)\ R$
⟨*proof*⟩

lemma *eq-comp-eval-restrict-left-le-if-crel-dep-mono-wrt-pred*:
assumes [*uhint*]: $((x : A) \to_c B\ x)\ R$
shows $((=) \circ eval\ R)\restriction_A \leq R$
⟨*proof*⟩

lemma *le-eq-comp-eval-restrict-left-if-rel-dep-mono-wrt-pred*:
assumes [*uhint*]: $((x : A) \to_c B\ x)\ R$
shows $R \leq ((=) \circ eval\ R)\restriction_A$
⟨*proof*⟩

corollary *restrict-left-eq-eq-comp-eval-if-crel-dep-mono-wrt-pred*:
assumes $((x : A) \to_c B\ x)\ R$
shows $R = ((=) \circ eval\ R)\restriction_A$
⟨*proof*⟩

lemma *eval-eq-if-crel-dep-mono-wrt-pred-if-rel-dep-mono-wrt-predI*:
fixes $R :: 'a \Rightarrow 'b \Rightarrow bool$
assumes $((x : A) \to B\ x)\ R\ ((x : A') \to_c B'\ x)\ R'$
and $R \leq R'$
and $A\ x$
shows $R`x = R'`x$
⟨*proof*⟩

lemma *crel-dep-mono-wrt-pred-ext*:
assumes $((x : A) \to_c B\ x)\ R\ ((x : A) \to_c B'\ x)\ R'$
and $\bigwedge x.\ A\ x \Longrightarrow R`x = R'`x$
shows $R = R'$
⟨*proof*⟩

lemma *eq-if-le-if-crel-dep-mono-wrt-pred-if-rel-dep-mono-wrt-pred*:
assumes $((x : A) \to B\ x)\ R\ ((x : A) \to_c B'\ x)\ R'$
and $R \leq R'$
shows $R = R'$
⟨*proof*⟩

lemma *ex-dom-crel-dep-mono-wrt-pred-iff-crel-dep-mono-wrt-pred-in-dom*:
$(\exists (A :: 'a \Rightarrow bool).\ ((x : A) \to_c B\ x)\ R) \longleftrightarrow (((x : in\text{-}dom\ R) \to_c B\ x)\ R)$
⟨*proof*⟩

lemma *crel-mono-wrt-pred-bottom-bottom*: $((\bot :: 'a \Rightarrow bool) \to_c A)\ (\bot :: 'a \Rightarrow 'b \Rightarrow bool)$
⟨*proof*⟩

lemma *crel-dep-mono-wrt-pred-bottom-iff-eq-bottom* [*iff*]: $((x : (\bot :: 'a \Rightarrow bool)) \to_c B\ x)\ R \longleftrightarrow R = \bot$
⟨*proof*⟩

lemma *mono-crel-dep-mono-wrt-pred-top-crel-dep-mono-wrt-pred-inf-rel-restrict-left*:
$(((x : A) \to_c B\ x) \Rightarrow (A' : \top) \Rightarrow (x : A \sqcap A') \to_c B\ x)\ rel\text{-}restrict\text{-}left$
⟨*proof*⟩

lemma *mono-rel-dep-mono-wrt-pred-ge-crel-dep-mono-wrt-pred-rel-restrict-left*:
$(((x : A) \to B\ x) \Rightarrow (A' : (\geq)\ A) \Rightarrow (x : A') \to_c B\ x)\ rel\text{-}restrict\text{-}left$
⟨*proof*⟩

lemma *crel-dep-mono-wrt-pred-eq-restrict*: $((x : (A :: 'a \Rightarrow bool)) \to_c (=)\ x)\ (=)\restriction_A$
⟨*proof*⟩

end

Symmetric

theory *Binary-Relations-Symmetric*
imports

Functions-Monotone
begin

consts *symmetric-on* :: $'a \Rightarrow 'b \Rightarrow bool$

overloading
symmetric-on-pred $\equiv$ *symmetric-on* :: $('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
begin
definition *symmetric-on-pred* $P\ R \equiv \forall x\ y : P.\ R\ x\ y \longrightarrow R\ y\ x$
end

lemma *symmetric-onI* [*intro*]:
assumes $\bigwedge x\ y.\ P\ x \Longrightarrow P\ y \Longrightarrow R\ x\ y \Longrightarrow R\ y\ x$
shows *symmetric-on* $P\ R$
⟨*proof*⟩

lemma *symmetric-onD*:
assumes *symmetric-on* $P\ R$
and $P\ x\ P\ y$
and $R\ x\ y$
shows $R\ y\ x$
⟨*proof*⟩

lemma *symmetric-on-rel-inv-iff-symmetric-on* [*iff*]:
symmetric-on $P\ R^{-1} \longleftrightarrow$ *symmetric-on* $(P :: 'a \Rightarrow bool)\ (R :: 'a \Rightarrow 'a \Rightarrow bool)$
⟨*proof*⟩

lemma *antimono-symmetric-on*: *antimono* (*symmetric-on* :: $('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$)
⟨*proof*⟩

lemma *symmetric-on-if-le-pred-if-symmetric-on*:
fixes $P' :: 'a \Rightarrow bool$ **and** $R :: 'a \Rightarrow 'a \Rightarrow bool$
assumes *symmetric-on* $P\ R$
and $P' \leq P$
shows *symmetric-on* $P'\ R$
⟨*proof*⟩

consts *symmetric* :: $'a \Rightarrow bool$

overloading
symmetric $\equiv$ *symmetric* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
begin
definition (*symmetric* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow$ -) $\equiv$ *symmetric-on* $(\top :: 'a \Rightarrow bool)$
end

lemma *symmetric-eq-symmetric-on*:
(*symmetric* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow$ -) = *symmetric-on* $(\top :: 'a \Rightarrow bool)$
⟨*proof*⟩

lemma *symmetric-eq-symmetric-on-uhint* [*uhint*]:
$P \equiv (\top :: 'a \Rightarrow bool) \Longrightarrow (symmetric :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow \text{-}) \equiv symmetric\text{-}on\ P$
⟨*proof*⟩

lemma *symmetricI* [*intro*]:
assumes $\bigwedge x\ y.\ R\ x\ y \Longrightarrow R\ y\ x$
shows *symmetric R*
⟨*proof*⟩

lemma *symmetricD*:
assumes *symmetric R*
and *R x y*
shows *R y x*
⟨*proof*⟩

lemma *symmetric-on-if-symmetric*:
fixes $P :: 'a \Rightarrow bool$ **and** $R :: 'a \Rightarrow 'a \Rightarrow bool$
assumes *symmetric R*
shows *symmetric-on P R*
⟨*proof*⟩

lemma *symmetric-rel-inv-iff-symmetric* [*iff*]: $symmetric\ R^{-1} \longleftrightarrow symmetric\ (R :: 'a \Rightarrow 'a \Rightarrow bool)$
⟨*proof*⟩

lemma *rel-inv-eq-self-if-symmetric* [*simp*]:
assumes *symmetric R*
shows $R^{-1} = R$
⟨*proof*⟩

lemma *rel-iff-rel-if-symmetric*:
assumes *symmetric R*
shows $R\ x\ y \longleftrightarrow R\ y\ x$
⟨*proof*⟩

lemma *symmetric-if-rel-inv-eq-self*:
assumes $R^{-1} = R$
shows *symmetric R*
⟨*proof*⟩

lemma *symmetric-iff-rel-inv-eq-self*: $symmetric\ R \longleftrightarrow R^{-1} = R$
⟨*proof*⟩

lemma *symmetric-if-symmetric-on-in-field*:
assumes *symmetric-on* (*in-field R*) *R*
shows *symmetric R*
⟨*proof*⟩

corollary *symmetric-on-in-field-iff-symmetric* [*iff*]:
symmetric-on (*in-field R*) *R* ⟷ *symmetric R*
⟨*proof*⟩

Instantiations **lemma** *symmetric-eq* [*iff*]: *symmetric* (=)
⟨*proof*⟩

lemma *symmetric-top*: *symmetric* (⊤ :: *'a* ⇒ *'a* ⇒ *bool*)
⟨*proof*⟩

end

Reflexive

theory *Binary-Relations-Reflexive*
imports
Functions-Monotone
ML-Unification.ML-Unification-HOL-Setup
ML-Unification.Unify-Resolve-Tactics
begin

consts *reflexive-on* :: *'a* ⇒ *'b* ⇒ *bool*

overloading
reflexive-on-pred ≡ *reflexive-on* :: (*'a* ⇒ *bool*) ⇒ (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*
begin
definition *reflexive-on-pred P R* ≡ ∀ *x*. *P x* ⟶ *R x x*
end

lemma *reflexive-onI* [*intro*]:
assumes ⋀*x*. *P x* ⟹ *R x x*
shows *reflexive-on P R*
⟨*proof*⟩

lemma *reflexive-onD* [*dest*]:
assumes *reflexive-on P R*
and *P x*
shows *R x x*
⟨*proof*⟩

context
fixes *R* :: *'a* ⇒ *'a* ⇒ *bool* **and** *P* :: *'a* ⇒ *bool*
begin

lemma *le-in-dom-if-reflexive-on*:
assumes *reflexive-on P R*
shows *P* ≤ *in-dom R*
⟨*proof*⟩

lemma *le-in-codom-if-reflexive-on*:
assumes *reflexive-on P R*
shows $P \leq$ *in-codom R*
⟨*proof*⟩

lemma *in-codom-eq-in-dom-if-reflexive-on-in-field*:
assumes *reflexive-on* (*in-field R*) *R*
shows *in-codom R* = *in-dom R*
⟨*proof*⟩

lemma *reflexive-on-rel-inv-iff-reflexive-on* [*iff*]:
reflexive-on P $R^{-1} \longleftrightarrow$ *reflexive-on P R*
⟨*proof*⟩

lemma *mono-reflexive-on*:
$((\geq) \Rightarrow (\leq) \Rrightarrow (\leq))$ (*reflexive-on* :: $('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$)
⟨*proof*⟩

lemma *reflexive-on-if-le-pred-if-reflexive-on*:
fixes $P' :: 'a \Rightarrow bool$
assumes *reflexive-on P R*
and $P' \leq P$
shows *reflexive-on* P' *R*
⟨*proof*⟩

end

lemma *reflexive-on-sup-eq* [*simp*]:
(*reflexive-on* :: $('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$) $(P \sqcup Q)$
= *reflexive-on P* $\sqcap$ *reflexive-on Q*
⟨*proof*⟩

lemma *reflexive-on-iff-eq-restrict-le*:
reflexive-on $(P :: 'a \Rightarrow bool)$ $(R :: 'a \Rightarrow$ -) $\longleftrightarrow$ $((=)\restriction_P \leq R)$
⟨*proof*⟩

consts *reflexive* :: $'a \Rightarrow bool$

overloading
reflexive $\equiv$ *reflexive* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
begin
definition *reflexive* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool \equiv$ *reflexive-on* $(\top :: 'a \Rightarrow bool)$
end

lemma *reflexive-eq-reflexive-on*:
(*reflexive* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow$ -) = *reflexive-on* $(\top :: 'a \Rightarrow bool)$
⟨*proof*⟩

lemma *reflexive-eq-reflexive-on-uhint* [*uhint*]:
  $P \equiv (\top :: 'a \Rightarrow bool) \Longrightarrow (reflexive :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow \text{-}) \equiv reflexive\text{-}on\ P$
  ⟨*proof*⟩

lemma *reflexiveI* [*intro*]:
  assumes $\bigwedge x.\ R\ x\ x$
  shows *reflexive R*
  ⟨*proof*⟩

lemma *reflexiveD*:
  assumes *reflexive R*
  shows *R x x*
  ⟨*proof*⟩

lemma *reflexive-on-if-reflexive*:
  fixes $P :: 'a \Rightarrow bool$ **and** $R :: 'a \Rightarrow 'a \Rightarrow bool$
  assumes *reflexive R*
  shows *reflexive-on P R*
  ⟨*proof*⟩

lemma *reflexive-rel-inv-iff-reflexive* [*iff*]:
  $reflexive\ (R :: 'a \Rightarrow 'a \Rightarrow bool)^{-1} \longleftrightarrow reflexive\ R$
  ⟨*proof*⟩

lemma *reflexive-iff-eq-le*: $reflexive\ R \longleftrightarrow ((=) \leq R)$
  ⟨*proof*⟩

Instantiations **lemma** *reflexive-eq*: *reflexive* (=)
  ⟨*proof*⟩

lemma *reflexive-top*: $reflexive\ (\top :: 'a \Rightarrow 'a \Rightarrow bool)$
  ⟨*proof*⟩

end

Transitive

theory *Binary-Relations-Transitive*
  imports
    Functions-Monotone
begin

consts $transitive\text{-}on :: 'a \Rightarrow 'b \Rightarrow bool$

overloading
  $transitive\text{-}on\text{-}pred \equiv transitive\text{-}on :: ('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
begin

definition *transitive-on-pred* $P\ R \equiv \forall\, x\ y\ z : P.\ R\ x\ y \wedge R\ y\ z \longrightarrow R\ x\ z$
end

lemma *transitive-onI* [*intro*]:
assumes $\bigwedge x\ y\ z.\ P\ x \Longrightarrow P\ y \Longrightarrow P\ z \Longrightarrow R\ x\ y \Longrightarrow R\ y\ z \Longrightarrow R\ x\ z$
shows *transitive-on* $P\ R$
⟨*proof*⟩

lemma *transitive-onD*:
assumes *transitive-on* $P\ R$
and $P\ x\ P\ y\ P\ z$
and $R\ x\ y\ R\ y\ z$
shows $R\ x\ z$
⟨*proof*⟩

lemma *transitive-on-if-rel-comp-self-imp*:
assumes $\bigwedge x\ y.\ P\ x \Longrightarrow P\ y \Longrightarrow (R \circ\circ R)\ x\ y \Longrightarrow R\ x\ y$
shows *transitive-on* $P\ R$
⟨*proof*⟩

lemma *transitive-on-rel-inv-iff-transitive-on* [*iff*]:
transitive-on $P\ R^{-1} \longleftrightarrow$ *transitive-on* $(P :: 'a \Rightarrow bool)\ (R :: 'a \Rightarrow 'a \Rightarrow bool)$
⟨*proof*⟩

lemma *antimono-transitive-on*: *antimono* $(transitive\text{-}on :: ('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool)$
⟨*proof*⟩

lemma *transitive-on-rel-map-if-mono-wrt-pred-if-transitive-on*:
fixes $P :: 'a \Rightarrow bool$ **and** $R :: 'a \Rightarrow 'a \Rightarrow bool$
assumes *transitive-on* $P\ R$
and $(Q \Rightarrow P)\ f$
shows *transitive-on* Q $(rel\text{-}map\ f\ R)$
⟨*proof*⟩

consts *transitive* :: $'a \Rightarrow bool$

overloading
transitive $\equiv$ *transitive* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
begin
definition $(transitive :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool) \equiv transitive\text{-}on\ (\top :: 'a \Rightarrow bool)$
end

lemma *transitive-eq-transitive-on*:
$(transitive :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow \text{-}) = transitive\text{-}on\ (\top :: 'a \Rightarrow bool)$
⟨*proof*⟩

lemma *transitive-eq-transitive-on-uhint* [*uhint*]:

$P \equiv (\top :: 'a \Rightarrow bool) \Longrightarrow (transitive :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow \text{-}) \equiv transitive\text{-}on\ P$
⟨*proof*⟩

lemma *transitiveI* [*intro*]:
assumes $\bigwedge x\ y\ z.\ R\ x\ y \Longrightarrow R\ y\ z \Longrightarrow R\ x\ z$
shows *transitive R*
⟨*proof*⟩

lemma *transitiveD* [*dest*]:
assumes *transitive R*
and *R x y R y z*
shows *R x z*
⟨*proof*⟩

lemma *transitive-on-if-transitive*:
fixes $P :: 'a \Rightarrow bool$ **and** $R :: 'a \Rightarrow 'a \Rightarrow bool$
assumes *transitive R*
shows *transitive-on P R*
⟨*proof*⟩

lemma *transitive-if-rel-comp-le-self*:
assumes $R \circ\circ R \leq R$
shows *transitive R*
⟨*proof*⟩

lemma *rel-comp-le-self-if-transitive*:
assumes *transitive R*
shows $R \circ\circ R \leq R$
⟨*proof*⟩

corollary *transitive-iff-rel-comp-le-self*: $transitive\ R \longleftrightarrow R \circ\circ R \leq R$
⟨*proof*⟩

lemma *transitive-if-transitive-on-in-field*:
assumes *transitive-on* (*in-field R*) *R*
shows *transitive R*
⟨*proof*⟩

corollary *transitive-on-in-field-iff-transitive* [*iff*]:
transitive-on (*in-field R*) $R \longleftrightarrow$ *transitive R*
⟨*proof*⟩

lemma *transitive-rel-inv-iff-transitive* [*iff*]: $transitive\ R^{-1} \longleftrightarrow transitive\ (R :: 'a \Rightarrow 'a \Rightarrow bool)$
⟨*proof*⟩

Instantiations **lemma** *transitive-eq*: *transitive* (=)
⟨*proof*⟩

lemma *transitive-top*: *transitive* ($\top$ :: $'a \Rightarrow 'a \Rightarrow bool$)
⟨*proof*⟩

end

1.3.2 Preorders

theory *Preorders*
imports
Binary-Relations-Reflexive
Binary-Relations-Transitive
begin

definition *preorder-on* $\equiv$ *reflexive-on* $\sqcap$ *transitive-on*

lemma *preorder-onI* [*intro*]:
assumes *reflexive-on P R*
and *transitive-on P R*
shows *preorder-on P R*
⟨*proof*⟩

lemma *preorder-onE* [*elim*]:
assumes *preorder-on P R*
obtains *reflexive-on P R transitive-on P R*
⟨*proof*⟩

lemma *reflexive-on-if-preorder-on*:
assumes *preorder-on P R*
shows *reflexive-on P R*
⟨*proof*⟩

lemma *transitive-on-if-preorder-on*:
assumes *preorder-on P R*
shows *transitive-on P R*
⟨*proof*⟩

lemma *transitive-if-preorder-on-in-field*:
assumes *preorder-on* (*in-field R*) *R*
shows *transitive R*
⟨*proof*⟩

corollary *preorder-on-in-fieldE* [*elim*]:
assumes *preorder-on* (*in-field R*) *R*
obtains *reflexive-on* (*in-field R*) *R transitive R*
⟨*proof*⟩

lemma *preorder-on-rel-inv-if-preorder-on* [*iff*]:
preorder-on P $R^{-1} \longleftrightarrow$ *preorder-on* (*P* :: $'a \Rightarrow bool$) (*R* :: $'a \Rightarrow 'a \Rightarrow bool$)

⟨*proof*⟩

lemma *rel-if-all-rel-if-rel-if-reflexive-on*:
assumes *reflexive-on P R*
and $\bigwedge z.\ P\ z \Longrightarrow R\ x\ z \Longrightarrow R\ y\ z$
and $P\ x$
shows $R\ y\ x$
⟨*proof*⟩

lemma *rel-if-all-rel-if-rel-if-reflexive-on'*:
assumes *reflexive-on P R*
and $\bigwedge z.\ P\ z \Longrightarrow R\ z\ x \Longrightarrow R\ z\ y$
and $P\ x$
shows $R\ x\ y$
⟨*proof*⟩

consts *preorder* :: $'a \Rightarrow bool$

overloading
preorder ≡ *preorder* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
begin
definition (*preorder* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$) ≡ *preorder-on* ($\top$ :: $'a \Rightarrow bool$)
end

lemma *preorder-eq-preorder-on*:
(*preorder* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$) = *preorder-on* ($\top$ :: $'a \Rightarrow bool$)
⟨*proof*⟩

lemma *preorder-eq-preorder-on-uhint* [*uhint*]:
assumes $P \equiv \top$ :: $'a \Rightarrow bool$
shows (*preorder* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$) ≡ *preorder-on P*
⟨*proof*⟩

context
fixes R :: $'a \Rightarrow 'a \Rightarrow bool$
begin

lemma *preorderI* [*intro*]:
assumes *reflexive R*
and *transitive R*
shows *preorder R*
⟨*proof*⟩

lemma *preorderE* [*elim*]:
assumes *preorder R*
obtains *reflexive R transitive R*
⟨*proof*⟩

lemma *preorder-on-if-preorder*:

fixes $P :: 'a \Rightarrow bool$
assumes *preorder* R
shows *preorder-on* P R
⟨*proof*⟩

end

Instantiations **lemma** *preorder-eq*: *preorder* (=)
⟨*proof*⟩

end

1.3.3 Partial Equivalence Relations

theory *Partial-Equivalence-Relations*
imports
Binary-Relations-Symmetric
Preorders
begin

definition *partial-equivalence-rel-on* ≡ *transitive-on* ⊓ *symmetric-on*

lemma *partial-equivalence-rel-onI* [*intro*]:
assumes *transitive-on* P R
and *symmetric-on* P R
shows *partial-equivalence-rel-on* P R
⟨*proof*⟩

lemma *partial-equivalence-rel-onE* [*elim*]:
assumes *partial-equivalence-rel-on* P R
obtains *transitive-on* P R *symmetric-on* P R
⟨*proof*⟩

lemma *partial-equivalence-rel-on-rel-self-if-rel-dom*:
assumes *partial-equivalence-rel-on* $(P :: 'a \Rightarrow bool)$ $(R :: 'a \Rightarrow 'a \Rightarrow bool)$
and $P\ x$ $P\ y$
and $R\ x\ y$
shows $R\ x\ x$
⟨*proof*⟩

lemma *partial-equivalence-rel-on-rel-self-if-rel-codom*:
assumes *partial-equivalence-rel-on* $(P :: 'a \Rightarrow bool)$ $(R :: 'a \Rightarrow 'a \Rightarrow bool)$
and $P\ x$ $P\ y$
and $R\ x\ y$
shows $R\ y\ y$
⟨*proof*⟩

lemma *partial-equivalence-rel-on-rel-inv-iff-partial-equivalence-rel-on* [*iff*]:

partial-equivalence-rel-on P R^{-1} $\longleftrightarrow$ *partial-equivalence-rel-on* (*P* :: $'a \Rightarrow$ *bool*) (*R* :: $'a \Rightarrow 'a \Rightarrow$ *bool*)
⟨*proof*⟩

consts *partial-equivalence-rel* :: $'a \Rightarrow$ *bool*

overloading
partial-equivalence-rel ≡ *partial-equivalence-rel* :: $('a \Rightarrow 'a \Rightarrow$ *bool*$) \Rightarrow$ *bool*
begin
definition (*partial-equivalence-rel* :: $('a \Rightarrow 'a \Rightarrow$ *bool*$) \Rightarrow$ *bool*) ≡ *partial-equivalence-rel-on* (⊤ :: $'a \Rightarrow$ *bool*)
end

lemma *partial-equivalence-rel-eq-partial-equivalence-rel-on*:
(*partial-equivalence-rel* :: $('a \Rightarrow 'a \Rightarrow$ *bool*$) \Rightarrow$ *bool*) = *partial-equivalence-rel-on* (⊤ :: $'a \Rightarrow$ *bool*)
⟨*proof*⟩

lemma *partial-equivalence-rel-eq-partial-equivalence-rel-on-uhint* [*uhint*]:
assumes *P* ≡ ⊤ :: $'a \Rightarrow$ *bool*
shows (*partial-equivalence-rel* :: $('a \Rightarrow 'a \Rightarrow$ *bool*$) \Rightarrow$ *bool*) ≡ *partial-equivalence-rel-on P*
⟨*proof*⟩

context
fixes *R* :: $'a \Rightarrow 'a \Rightarrow$ *bool*
begin

lemma *partial-equivalence-relI* [*intro*]:
assumes *transitive R*
and *symmetric R*
shows *partial-equivalence-rel R*
⟨*proof*⟩

lemma *reflexive-on-in-field-if-partial-equivalence-rel*:
assumes *partial-equivalence-rel R*
shows *reflexive-on* (*in-field R*) *R*
⟨*proof*⟩

lemma *partial-equivalence-relE* [*elim*]:
assumes *partial-equivalence-rel R*
obtains *preorder-on* (*in-field R*) *R symmetric R*
⟨*proof*⟩

lemma *partial-equivalence-rel-on-if-partial-equivalence-rel*:
fixes *P* :: $'a \Rightarrow$ *bool*
assumes *partial-equivalence-rel R*
shows *partial-equivalence-rel-on P R*
⟨*proof*⟩

lemma *partial-equivalence-rel-rel-inv-iff-partial-equivalence-rel* [*iff*]:
partial-equivalence-rel $R^{-1} \longleftrightarrow$ *partial-equivalence-rel* R
⟨*proof*⟩

corollary *in-codom-eq-in-dom-if-partial-equivalence-rel*:
assumes *partial-equivalence-rel* R
shows *in-codom* R = *in-dom* R
⟨*proof*⟩

lemma *partial-equivalence-rel-rel-comp-self-eq-self*:
assumes *partial-equivalence-rel* R
shows $(R \circ\circ R) = R$
⟨*proof*⟩

lemma *partial-equivalence-rel-if-partial-equivalence-rel-on-in-field*:
assumes *partial-equivalence-rel-on* (*in-field* R) R
shows *partial-equivalence-rel* R
⟨*proof*⟩

corollary *partial-equivalence-rel-on-in-field-iff-partial-equivalence-rel* [*iff*]:
partial-equivalence-rel-on (*in-field* R) $R \longleftrightarrow$ *partial-equivalence-rel* R
⟨*proof*⟩

end

Instantiations **lemma** *partial-equivalence-rel-eq*: *partial-equivalence-rel* (=)
⟨*proof*⟩

lemma *partial-equivalence-rel-top*: *partial-equivalence-rel* ($\top$ :: $'a \Rightarrow 'a \Rightarrow$ *bool*)
⟨*proof*⟩

end

1.3.4 Equivalences

theory *Equivalence-Relations*
imports
Partial-Equivalence-Relations
begin

definition *equivalence-rel-on* ≡ *partial-equivalence-rel-on* ⊓ *reflexive-on*

lemma *equivalence-rel-onI* [*intro*]:
assumes *partial-equivalence-rel-on* P R
and *reflexive-on* P R
shows *equivalence-rel-on* P R
⟨*proof*⟩

lemma *equivalence-rel-onE* [*elim*]:
assumes *equivalence-rel-on P R*
obtains *partial-equivalence-rel-on P R reflexive-on P R*
⟨*proof*⟩

lemma *equivalence-rel-on-in-field-if-partial-equivalence-rel*:
assumes *partial-equivalence-rel R*
shows *equivalence-rel-on* (*in-field R*) *R*
⟨*proof*⟩

corollary *partial-equivalence-rel-iff-equivalence-rel-on-in-field*:
partial-equivalence-rel R ⟷ *equivalence-rel-on* (*in-field R*) *R*
⟨*proof*⟩

consts *equivalence-rel* :: *'a* ⇒ *bool*

overloading
equivalence-rel ≡ *equivalence-rel* :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*
begin
definition (*equivalence-rel* :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*) ≡ *equivalence-rel-on* (⊤ :: *'a* ⇒ *bool*)
end

lemma *equivalence-rel-eq-equivalence-rel-on*:
(*equivalence-rel* :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*) = *equivalence-rel-on* (⊤ :: *'a* ⇒ *bool*)
⟨*proof*⟩

lemma *equivalence-rel-eq-equivalence-rel-on-uhint* [*uhint*]:
assumes *P* ≡ ⊤ :: *'a* ⇒ *bool*
shows *equivalence-rel* :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool* ≡ *equivalence-rel-on P*
⟨*proof*⟩

context
fixes *R* :: *'a* ⇒ *'a* ⇒ *bool*
begin

lemma *equivalence-relI* [*intro*]:
assumes *partial-equivalence-rel R*
and *reflexive R*
shows *equivalence-rel R*
⟨*proof*⟩

lemma *equivalence-relE* [*elim*]:
assumes *equivalence-rel R*
obtains *partial-equivalence-rel R reflexive R*
⟨*proof*⟩

lemma *equivalence-rel-on-if-equivalence*:

fixes $P :: 'a \Rightarrow bool$
assumes *equivalence-rel* R
shows *equivalence-rel-on* P R
⟨*proof*⟩

end

Instantiations **lemma** *equivalence-eq*: *equivalence-rel* (=)
⟨*proof*⟩

lemma *equivalence-top*: *equivalence-rel* ($\top :: 'a \Rightarrow 'a \Rightarrow bool$)
⟨*proof*⟩

end

Antisymmetric

theory *Binary-Relations-Antisymmetric*
imports
Binary-Relation-Functions
Functions-Monotone
begin

consts *antisymmetric-on* :: $'a \Rightarrow 'b \Rightarrow bool$

overloading
antisymmetric-on-pred $\equiv$ *antisymmetric-on* :: $('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
begin
definition *antisymmetric-on-pred* P $R \equiv \forall x\ y : P.\ R\ x\ y \wedge R\ y\ x \longrightarrow x = y$
end

lemma *antisymmetric-onI* [*intro*]:
assumes $\bigwedge x\ y.\ P\ x \Longrightarrow P\ y \Longrightarrow R\ x\ y \Longrightarrow R\ y\ x \Longrightarrow x = y$
shows *antisymmetric-on* P R
⟨*proof*⟩

lemma *antisymmetric-onD*:
assumes *antisymmetric-on* P R
and $P\ x\ P\ y$
and $R\ x\ y\ R\ y\ x$
shows $x = y$
⟨*proof*⟩

lemma *antisymmetric-onE*:
assumes *antisymmetric-on* P R
obtains $\bigwedge x\ y.\ P\ x \Longrightarrow P\ y \Longrightarrow R\ x\ y \Longrightarrow R\ y\ x \Longrightarrow x = y$
⟨*proof*⟩

lemma *antimono-antisymmetric-on*:
(($\leq$) $\Rightarrow$ ($\leq$) $\Rrightarrow$ ($\geq$)) (*antisymmetric-on* :: ('*a* $\Rightarrow$ *bool*) $\Rightarrow$ ('*a* $\Rightarrow$ '*a* $\Rightarrow$ *bool*) $\Rightarrow$ *bool*)
⟨*proof*⟩

consts *antisymmetric* :: '*a* $\Rightarrow$ *bool*

overloading
antisymmetric $\equiv$ *antisymmetric* :: ('*a* $\Rightarrow$ '*a* $\Rightarrow$ *bool*) $\Rightarrow$ *bool*
begin
definition *antisymmetric* :: ('*a* $\Rightarrow$ '*a* $\Rightarrow$ *bool*) $\Rightarrow$ - $\equiv$ *antisymmetric-on* ($\top$ :: '*a* $\Rightarrow$ *bool*)
end

lemma *antisymmetric-eq-antisymmetric-on*:
(*antisymmetric* :: ('*a* $\Rightarrow$ '*a* $\Rightarrow$ *bool*) $\Rightarrow$ -) = *antisymmetric-on* ($\top$ :: '*a* $\Rightarrow$ *bool*)
⟨*proof*⟩

lemma *antisymmetric-eq-antisymmetric-on-uhint* [*uhint*]:
$P \equiv (\top :: 'a \Rightarrow bool) \Longrightarrow$ (*antisymmetric* :: ('*a* $\Rightarrow$ '*a* $\Rightarrow$ *bool*) $\Rightarrow$ -) $\equiv$ *antisymmetric-on P*
⟨*proof*⟩

lemma *antisymmetricI* [*intro*]:
assumes $\bigwedge x\ y.\ R\ x\ y \Longrightarrow R\ y\ x \Longrightarrow x = y$
shows *antisymmetric R*
⟨*proof*⟩

lemma *antisymmetricD*:
assumes *antisymmetric R*
and *R x y R y x*
shows $x = y$
⟨*proof*⟩

lemma *antisymmetricE*:
assumes *antisymmetric R*
obtains $\bigwedge x\ y.\ R\ x\ y \Longrightarrow R\ y\ x \Longrightarrow x = y$
⟨*proof*⟩

lemma *antisymmetric-on-if-antisymmetric*:
fixes *P* :: '*a* $\Rightarrow$ *bool* **and** *R* :: '*a* $\Rightarrow$ '*a* $\Rightarrow$ *bool*
assumes *antisymmetric R*
shows *antisymmetric-on P R*
⟨*proof*⟩

lemma *antisymmetric-if-antisymmetric-on-in-field*:
assumes *antisymmetric-on* (*in-field R*) *R*
shows *antisymmetric R*
⟨*proof*⟩

corollary *antisymmetric-on-in-field-iff-antisymmetric* [*iff*]:
antisymmetric-on (*in-field R*) *R* ⟷ *antisymmetric R*
⟨*proof*⟩

end

1.3.5 Partial Orders

theory *Partial-Orders*
imports
Binary-Relations-Antisymmetric
Preorders
begin

definition *partial-order-on* ≡ *preorder-on* ⊓ *antisymmetric-on*

lemma *partial-order-onI* [*intro*]:
assumes *preorder-on P R*
and *antisymmetric-on P R*
shows *partial-order-on P R*
⟨*proof*⟩

lemma *partial-order-onE* [*elim*]:
assumes *partial-order-on P R*
obtains *preorder-on P R antisymmetric-on P R*
⟨*proof*⟩

lemma *transitive-if-partial-order-on-in-field*:
assumes *partial-order-on* (*in-field R*) *R*
shows *transitive R*
⟨*proof*⟩

lemma *antisymmetric-if-partial-order-on-in-field*:
assumes *partial-order-on* (*in-field R*) *R*
shows *antisymmetric R*
⟨*proof*⟩

consts *partial-order* :: *'a* ⇒ *bool*

overloading
partial-order ≡ *partial-order* :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*
begin
definition (*partial-order* :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*) ≡ *partial-order-on* (⊤ :: *'a* ⇒ *bool*)
end

lemma *partial-order-eq-partial-order-on*:

(*partial-order* :: ('*a* ⇒ '*a* ⇒ *bool*) ⇒ *bool*) = *partial-order-on* (⊤ :: '*a* ⇒ *bool*)
⟨*proof*⟩

lemma *partial-order-eq-partial-order-on-uhint* [*uhint*]:
assumes $P \equiv \top :: 'a \Rightarrow bool$
shows (*partial-order* :: ('*a* ⇒ '*a* ⇒ *bool*) ⇒ *bool*) ≡ *partial-order-on P*
⟨*proof*⟩

context
fixes $R :: 'a \Rightarrow 'a \Rightarrow bool$
begin

lemma *partial-orderI* [*intro*]:
assumes *preorder R*
and *antisymmetric R*
shows *partial-order R*
⟨*proof*⟩

lemma *partial-orderE* [*elim*]:
assumes *partial-order R*
obtains *preorder R antisymmetric R*
⟨*proof*⟩

lemma *partial-order-on-if-partial-order*:
fixes $P :: 'a \Rightarrow bool$
assumes *partial-order R*
shows *partial-order-on P R*
⟨*proof*⟩

end

end

1.3.6 Restricted Equality

theory *Restricted-Equality*
imports
Binary-Relations-Order-Base
Binary-Relation-Functions
Equivalence-Relations
Partial-Orders
begin

Summary Introduces notations and theorems for restricted equalities. An equality (=) can be restricted to only apply to a subset of its elements. The restriction can be formulated, for example, by a predicate or a set.

open-bundle *eq-rel-restrict-syntax*
begin
syntax

-eq-restrict-infix :: *'a* ⇒ *'b* ⇒ *'a* ⇒ *bool* (‹(-) =(_) (-)› [*51*,*51*,*51*] *50*)
-eq-restrict :: *'b* ⇒ *'a* ⇒ *bool* (‹*'*(=(_)*'*)›)
end

syntax-consts
-eq-restrict-infix ⇌ *rel-restrict-left* **and**
-eq-restrict ⇌ *rel-restrict-left*
translations
(=$_P$) ⇌ *CONST rel-restrict-left* (=) *P*
x =$_P$ *y* ⇌ *CONST rel-restrict-left* (=) *P x y*

lemma *in-dom-eq-restrict-eq* [*simp*]: *in-dom* (=$_P$) = *P* ⟨*proof*⟩
lemma *in-codom-eq-restrict-eq* [*simp*]: *in-codom* (=$_P$) = *P* ⟨*proof*⟩
lemma *in-field-eq-restrict-eq* [*simp*]: *in-field* (=$_P$) = *P* ⟨*proof*⟩

Order Properties **context**
fixes *P* :: *'a* ⇒ *bool*
begin

context
begin
lemma *reflexive-on-eq-restrict*: *reflexive-on P* ((=$_P$) :: *'a* ⇒ -) ⟨*proof*⟩
lemma *transitive-eq-restrict*: *transitive* ((=$_P$) :: *'a* ⇒ -) ⟨*proof*⟩
lemma *symmetric-eq-restrict*: *symmetric* ((=$_P$) :: *'a* ⇒ -) ⟨*proof*⟩
lemma *antisymmetric-eq-restrict*: *antisymmetric* ((=$_P$) :: *'a* ⇒ -) ⟨*proof*⟩
end

context
begin
lemma *preorder-on-eq-restrict*: *preorder-on P* ((=$_P$) :: *'a* ⇒ -)
⟨*proof*⟩
lemma *partial-equivalence-rel-eq-restrict*: *partial-equivalence-rel* ((=$_P$) :: *'a* ⇒ -)
⟨*proof*⟩
end

lemma *partial-order-on-eq-restrict*: *partial-order-on P* ((=$_P$) :: *'a* ⇒ -)
⟨*proof*⟩
lemma *equivalence-rel-on-eq-restrict*: *equivalence-rel-on P* ((=$_P$) :: *'a* ⇒ -)
⟨*proof*⟩
end

end

1.3.7 Composing Functions

theory *Binary-Relations-Function-Composition*
imports
Binary-Relations-Clean-Functions

Restricted-Equality
begin

lemma *dep-bin-rel-eval-rel-comp-if-dep-bin-rel-if-crel-dep-mono-wrt-pred*:
assumes $((x : A) \rightarrow_c B\ x)\ R$
and $\bigwedge x.\ A\ x \Longrightarrow (\{\sum\}y : B\ x.\ C\ x\ y)\ R'$
shows $(\{\sum\}x : A.\ C\ x\ (R`x))\ (R \circ\circ R')$
⟨*proof*⟩

lemma *crel-dep-mono-wrt-pred-eval-rel-comp-if-rel-dep-mono-wrt-pred-if-crel-dep-mono-wrt-pred*:
assumes $((x : A) \rightarrow_c B\ x)\ R$
and $\bigwedge x.\ A\ x \Longrightarrow ((y : B\ x) \rightarrow C\ x\ y)\ R'$
shows $((x : A) \rightarrow_c C\ x\ (R`x))\ (R \circ\circ R')$
⟨*proof*⟩

corollary *mono-crel-mono-rel-mono-crel-mono-rel-comp*: $((A \rightarrow_c B) \Rightarrow (B \rightarrow C) \Rightarrow A \rightarrow_c C)\ (\circ\circ)$
⟨*proof*⟩

lemma *rel-comp-eval-eq-if-rel-dep-mono-wrt-pred-if-crel-dep-mono-wrt-predI* [*simp*]:
assumes $((x : A) \rightarrow_c B\ x)\ R$
and $\bigwedge x.\ A\ x \Longrightarrow ((y : B\ x) \rightarrow C\ x\ y)\ R'$
and $A\ x$
shows $(R \circ\circ R')`x = R'`(R`x)$
⟨*proof*⟩

lemma *eq-restrict-comp-eq-if-crel-dep-mono-wrt-pred* [*simp*]:
assumes $((x : A) \rightarrow_c B\ x)\ R$
shows $(=_A) \circ\circ R = R$
⟨*proof*⟩

lemma *comp-eq-restrict-if-crel-dep-mono-wrt-pred* [*simp*]:
assumes $(A \rightarrow_c B)\ R$
shows $R \circ\circ (=_B) = R$
⟨*proof*⟩

end

1.3.8 Extending Functions

theory *Binary-Relations-Function-Extend*
imports
Binary-Relations-Clean-Functions
begin

lemma *left-total-on-eq-sup-extend-if-left-total-on*:
assumes *left-total-on A R*
shows *left-total-on* $((=)\ x \sqcup A)$ $(extend\ x\ y\ R)$
⟨*proof*⟩

lemma *right-unique-on-eq-sup-extend-if-not-in-dom-if-right-unique-on*:
assumes *right-unique-on A R*
and ¬(*in-dom R x*)
shows *right-unique-on* ((=) *x* ⊔ *A*) (*extend x y R*)
⟨*proof*⟩

lemma *rel-dep-mono-wrt-eq-sup-if-extend-if-rel-dep-mono-wrt-predI*:
assumes ((*x* : *A*) → *B x*) *R*
and ¬(*in-dom R x*)
shows ((*x'* : (=) *x* ⊔ *A*) → (*if x' = x then* (=) *y else B x'*)) (*extend x y R*)
⟨*proof*⟩

lemma *rel-dep-mono-wrt-eq-sup-extend-if-rel-dep-mono-wrt-predI*:
assumes ((*x* : *A*) → *B x*) *R*
and ¬(*in-dom R x*)
and *B x y*
shows ((*x'* : (=) *x* ⊔ *A*) → *B x'*) (*extend x y R*)
⟨*proof*⟩

lemma *crel-dep-mono-wrt-eq-sup-if-extend-if-crel-dep-mono-wrt-predI*:
assumes ((*x* : *A*) →$_c$ *B x*) *R*
and ¬(*in-dom R x*)
shows ((*x'* : (=) *x* ⊔ *A*) →$_c$ (*if x' = x then* (=) *y else B x'*)) (*extend x y R*)
⟨*proof*⟩

lemma *crel-dep-mono-wrt-eq-sup-extend-if-rel-dep-mono-wrt-predI*:
assumes ((*x* : *A*) →$_c$ *B x*) *R*
and ¬(*in-dom R x*)
and *B x y*
shows ((*x'* : (=) *x* ⊔ *A*) →$_c$ *B x'*) (*extend x y R*)
⟨*proof*⟩

context
fixes $\mathcal{R}$ :: ('*a* ⇒ '*b* ⇒ *bool*) ⇒ *bool* **and** *A* :: ('*a* ⇒ '*b* ⇒ *bool*) ⇒ '*a* ⇒ *bool* **and** *D*
defines [*simp*]: *D* ≡ *in-codom-on* $\mathcal{R}$ *A*
begin

lemma *rel-dep-mono-wrt-pred-glue-if-right-unique-if-rel-dep-mono-wrt-pred*:
assumes *funs*: ⋀*R*. $\mathcal{R}$ *R* ⟹ ((*x* : *A R*) → *B x*) *R*
and *runique*: *right-unique-on D* (*glue* $\mathcal{R}$)
shows ((*x* : *D*) → *B x*) (*glue* $\mathcal{R}$)
⟨*proof*⟩

lemma *crel-dep-mono-wrt-pred-glue-if-right-unique-if-crel-dep-mono-wrt-pred*:
assumes ⋀*R*. $\mathcal{R}$ *R* ⟹ ((*x* : *A R*) →$_c$ *B x*) *R*
and *right-unique-on D* (*glue* $\mathcal{R}$)
shows ((*x* : *D*) →$_c$ *B x*) (*glue* $\mathcal{R}$)

⟨*proof*⟩

end

lemma *right-unique-on-sup-if-rel-agree-on-sup-if-right-unique-on*:
assumes *right-unique-on P R right-unique-on P R′*
and *rel-agree-on* (*P* ⊓ *in-dom R* ⊓ *in-dom R′*) ((=) *R* ⊔ (=) *R′*)
shows *right-unique-on P* (*R* ⊔ *R′*)
⟨*proof*⟩

lemma *crel-dep-mono-wrt-pred-sup-if-eval-eq-if-crel-dep-mono-wrt-pred*:
assumes *dep-funs*: ((*x* : *A*) →$_c$ *B x*) *R* ((*x* : *A′*) →$_c$ *B x*) *R′*
and ⋀*x*. *A x* ⟹ *A′ x* ⟹ *R*‘*x* = *R′*‘*x*
shows ((*x* : *A* ⊔ *A′*) →$_c$ *B x*) (*R* ⊔ *R′*)
⟨*proof*⟩

end

1.3.9 Lambda Abstractions

theory *Binary-Relations-Function-Lambda*
imports *Binary-Relations-Clean-Functions*
begin

consts *rel-lambda* :: *′a* ⇒ (*′b* ⇒ *′c*) ⇒ *′d*

definition *rel-lambda-pred A f x y* ≡ *A x* ∧ *f x* = *y*
adhoc-overloading *rel-lambda* ⇌ *rel-lambda-pred*

open-bundle *rel-lambda-syntax*
begin
syntax
-rel-lambda :: *pttrns* ⇒ *′a* ⇒ *′b* ⇒ *′c* (‹(*2*λ- : -./ -)› *60*)
end

syntax-consts
-rel-lambda ⇌ *rel-lambda*
translations
λ*x xs* : *A*. *f* ⇀ *CONST rel-lambda A* (λ*x*. (λ*xs* : *A*. *f*))
λ*x* : *A*. *f* ⇌ *CONST rel-lambda A* (λ*x*. *f*)

lemma *rel-lambdaI* [*intro*]:
assumes *A x*
and *f x* = *y*
shows (λ*x* : *A*. *f x*) *x y*
⟨*proof*⟩

lemma *rel-lambda-appI*:
assumes *A x*

shows $(\lambda x : A.\ f\ x)\ x\ (f\ x)$
⟨*proof*⟩

lemma *rel-lambdaE* [*elim*!]:
assumes $(\lambda x : A.\ f\ x)\ x\ y$
obtains $A\ x\ y = f\ x$
⟨*proof*⟩

lemma *rel-lambda-cong* [*cong*]:
$[\![\bigwedge x.\ A\ x \longleftrightarrow A'\ x;\ \bigwedge x.\ A'\ x \Longrightarrow f\ x = f'\ x]\!] \Longrightarrow (\lambda x : A.\ f\ x) = \lambda x : A'.\ f'\ x$
⟨*proof*⟩

lemma *in-dom-rel-lambda-eq* [*simp*]: *in-dom* $(\lambda x : A.\ f\ x) = A$
⟨*proof*⟩

lemma *in-codom-rel-lambda-eq-has-inverse-on* [*simp*]: *in-codom* $(\lambda x : A.\ f\ x) =$ *has-inverse-on* $A\ f$
⟨*proof*⟩

lemma *left-total-on-rel-lambda*: *left-total-on* $A\ (\lambda x : A.\ f\ x)$
⟨*proof*⟩

lemma *right-unique-on-rel-lambda*: *right-unique-on* $A\ (\lambda x : A.\ f\ x)$
⟨*proof*⟩

lemma *crel-dep-mono-wrt-pred-rel-lambda*: $((x : A) \to_c ((=)\ (f\ x)))\ (\lambda x : A.\ f\ x)$
⟨*proof*⟩

Compare the following with (*rel-dep-mono-wrt ?A ?B* ⇒ *dep-mono-wrt ?A ?B*) (‘).

lemma *mono-dep-mono-wrt-pred-crel-dep-mono-wrt-pred-rel-lambda*:
$((A : \top) \Rightarrow ((x : A) \Rightarrow B\ x) \Rightarrow (x : A) \to_c B\ x)$ *rel-lambda*
⟨*proof*⟩

lemma *rel-lambda-eval-eq* [*simp*]:
assumes $A\ x$
shows $(\lambda x : A.\ f\ x)\text{‘}x = f\ x$
⟨*proof*⟩

lemma *app-eq-if-rel-lambda-eqI*:
assumes $(\lambda x : A.\ f\ x) = (\lambda x : A.\ g\ x)$
and $A\ x$
shows $f\ x = g\ x$
⟨*proof*⟩

lemma *crel-dep-mono-wrt-pred-inf-rel-lambda-inf-if-rel-dep-mono-wrt-pred*:
assumes $((x : A) \to B\ x)\ R$
shows $((x : A \sqcap A') \to_c B\ x)\ (\lambda x : A \sqcap A'.\ R\text{‘}x)$
⟨*proof*⟩

corollary *crel-dep-mono-wrt-pred-rel-lambda-if-le-if-rel-dep-mono-wrt-pred*:
assumes $((x : A) \to B\ x)\ R$
and [*uhint*]: $A' \leq A$
shows $((x : A') \to_c B\ x)\ (\lambda x : A'.\ R`x)$
⟨*proof*⟩

lemma *rel-lambda-ext*:
assumes $((x : A) \to_c B\ x)\ R$
and $\bigwedge x.\ A\ x \Longrightarrow f\ x = R`x$
shows $(\lambda x : A.\ f\ x) = R$
⟨*proof*⟩

lemma *rel-lambda-eval-eq-if-crel-dep-mono-wrt-pred* [*simp*]: $((x : A) \to_c B\ x)\ R \Longrightarrow (\lambda x : A.\ R`x) = R$
⟨*proof*⟩

Every element of *crel-dep-mono-wrt A B* may be expressed as a lambda abstraction.

lemma *eq-rel-lambda-if-crel-dep-mono-wrt-predE*:
assumes $((x : A) \to_c B\ x)\ R$
obtains f **where** $R = (\lambda x : A.\ f\ x)$
⟨*proof*⟩

lemma *rel-restrict-left-eq-rel-lambda-if-le-if-rel-dep-mono-wrt-pred*:
assumes $((x : A) \to B\ x)\ R$
and $A' \leq A$
shows $R\restriction_{A'} = (\lambda x : A'.\ R`x)$
⟨*proof*⟩

lemma *mono-rel-lambda*: *mono* $(\lambda A.\ \lambda x : A.\ f\ x)$
⟨*proof*⟩

end

theory *Binary-Relations-Functions*
imports
Binary-Relations-Clean-Functions
Binary-Relations-Function-Base
Binary-Relations-Function-Composition
Binary-Relations-Function-Evaluation
Binary-Relations-Function-Extend
Binary-Relations-Function-Lambda
begin

end

theory *Binary-Relations-Order*
imports
Binary-Relations-Order-Base
Binary-Relations-Reflexive
Binary-Relations-Symmetric
Binary-Relations-Transitive
begin

Summary Basic results about the order on binary relations.

lemma *in-dom-if-rel-if-rel-comp-le*:
assumes $(R \circ\circ S) \leq (S \circ\circ R)$
and $R\ x\ y\ S\ y\ z$
shows *in-dom* $S\ x$
⟨*proof*⟩

lemma *in-codom-if-rel-if-rel-comp-le*:
assumes $(R \circ\circ S) \leq (S \circ\circ R)$
and $R\ x\ y\ S\ y\ z$
shows *in-codom* $R\ z$
⟨*proof*⟩

lemma *rel-comp-le-rel-inv-if-rel-comp-le-if-symmetric*:
assumes *symms*: *symmetric R1 symmetric R2*
and *le*: $(R1 \circ\circ R2) \leq R3$
shows $(R2 \circ\circ R1) \leq R3^{-1}$
⟨*proof*⟩

lemma *rel-inv-le-rel-comp-if-le-rel-comp-if-symmetric*:
assumes *symms*: *symmetric R1 symmetric R2*
and *le*: $R3 \leq (R1 \circ\circ R2)$
shows $R3^{-1} \leq (R2 \circ\circ R1)$
⟨*proof*⟩

corollary *rel-comp-le-rel-comp-if-rel-comp-le-rel-comp-if-symmetric*:
assumes *symmetric* $(R1 :: 'a \Rightarrow 'a \Rightarrow bool)$ *symmetric R2 symmetric R3 symmetric R4*
and $(R1 \circ\circ R2) \leq (R3 \circ\circ R4)$
shows $(R2 \circ\circ R1) \leq (R4 \circ\circ R3)$
⟨*proof*⟩

corollary *rel-comp-le-rel-comp-iff-if-symmetric*:
assumes *symmetric* $(R1 :: 'a \Rightarrow 'a \Rightarrow bool)$ *symmetric R2 symmetric R3 symmetric R4*
shows $(R1 \circ\circ R2) \leq (R3 \circ\circ R4) \longleftrightarrow (R2 \circ\circ R1) \leq (R4 \circ\circ R3)$
⟨*proof*⟩

corollary *eq-if-le-rel-if-symmetric*:
assumes *symmetric R symmetric S*
and $(R \circ\circ S) \leq (S \circ\circ R)$

shows $(R \circ\circ S) = (S \circ\circ R)$
⟨*proof*⟩

lemma *rel-comp-le-rel-comp-if-le-rel-if-reflexive-on-in-codom-if-transitive*:
assumes *trans*: *transitive S*
and *refl-on*: *reflexive-on* (*in-codom S*) *R*
and *le-rel*: $R \leq S$
shows $R \circ\circ S \leq S \circ\circ R$
⟨*proof*⟩

end

Irreflexive

theory *Binary-Relations-Irreflexive*
imports
Functions-Monotone
begin

consts *irreflexive-on* :: $'a \Rightarrow 'b \Rightarrow bool$

overloading
irreflexive-on-pred ≡ *irreflexive-on* :: $('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
begin
definition *irreflexive-on-pred P R* ≡ $\forall x : P.\ \neg(R\ x\ x)$
end

lemma *irreflexive-onI* [*intro*]:
assumes $\bigwedge x.\ P\ x \Longrightarrow \neg(R\ x\ x)$
shows *irreflexive-on P R*
⟨*proof*⟩

lemma *irreflexive-onD* [*dest*]:
assumes *irreflexive-on P R*
and *P x*
shows $\neg(R\ x\ x)$
⟨*proof*⟩

lemma *antimono-irreflexive-on*:
$((\leq) \Rightarrow (\leq) \Rrightarrow (\geq))$ (*irreflexive-on* :: $('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$)
⟨*proof*⟩

consts *irreflexive* :: $'a \Rightarrow bool$

overloading
irreflexive ≡ *irreflexive* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
begin

definition (*irreflexive* :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*) ≡ *irreflexive-on* (⊤ :: *'a* ⇒ *bool*)
end

lemma *irreflexive-eq-irreflexive-on*:
(*irreflexive* :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ -) = *irreflexive-on* (⊤ :: *'a* ⇒ *bool*)
⟨*proof*⟩

lemma *irreflexive-eq-irreflexive-on-uhint* [*uhint*]:
assumes *P* ≡ (⊤ :: *'a* ⇒ *bool*)
shows *irreflexive* :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ - ≡ *irreflexive-on P*
⟨*proof*⟩

lemma *irreflexiveI* [*intro*]:
assumes ⋀*x*. ¬(*R x x*)
shows *irreflexive R*
⟨*proof*⟩

lemma *irreflexiveD*:
assumes *irreflexive R*
shows ¬(*R x x*)
⟨*proof*⟩

lemma *irreflexive-on-if-irreflexive*:
fixes *P* :: *'a* ⇒ *bool* **and** *R* :: *'a* ⇒ *'a* ⇒ *bool*
assumes *irreflexive R*
shows *irreflexive-on P R*
⟨*proof*⟩

end

theory *Binary-Relations-Asymmetric*
imports
Binary-Relations-Irreflexive
Binary-Relations-Antisymmetric
Functions-Monotone
begin

consts *asymmetric-on* :: *'a* ⇒ *'b* ⇒ *bool*

overloading
asymmetric-on-pred ≡ *asymmetric-on* :: (*'a* ⇒ *bool*) ⇒ (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*
begin
definition *asymmetric-on-pred* (*P* :: *'a* ⇒ *bool*) (*R* :: *'a* ⇒ *'a* ⇒ *bool*)
≡ ∀ *x y* : *P*. *R x y* ⟶ ¬(*R y x*)
end

lemma *asymmetric-onI* [*intro*]:
assumes $\bigwedge x\ y.\ P\ x \Longrightarrow P\ y \Longrightarrow R\ x\ y \Longrightarrow \neg(R\ y\ x)$
shows *asymmetric-on P R*
⟨*proof*⟩

lemma *asymmetric-onD* [*dest*]:
assumes *asymmetric-on P R*
and *P x P y*
and *R x y*
shows $\neg(R\ y\ x)$
⟨*proof*⟩

lemma *asymmetric-onE*:
assumes *asymmetric-on P R*
obtains $\bigwedge x\ y.\ P\ x \Longrightarrow P\ y \Longrightarrow R\ x\ y \Longrightarrow \neg(R\ y\ x)$
⟨*proof*⟩

lemma *asymmetric-on-le-irreflexive-on*:
(*asymmetric-on* :: (*'a* ⇒ *bool*) ⇒ (*'a* ⇒ *'a* ⇒ *bool*) ⇒ -) ≤ *irreflexive-on*
⟨*proof*⟩

lemma *asymmetric-on-le-antisymmetric-on*:
(*asymmetric-on* :: (*'a* ⇒ *bool*) ⇒ (*'a* ⇒ *'a* ⇒ *bool*) ⇒ -) ≤ *antisymmetric-on*
⟨*proof*⟩

lemma *antimono-asymmetric-on*:
((≤) ⇒ (≤) ⇛ (≥)) (*asymmetric-on* :: (*'a* ⇒ *bool*) ⇒ (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*)
⟨*proof*⟩

lemma *asymmetric-on-rel-map-if-mono-wrt-pred-if-asymmetric-on*:
fixes *P* :: *'a* ⇒ *bool* **and** *R* :: *'a* ⇒ *'a* ⇒ *bool*
assumes *asymmetric-on P R*
and (*Q* ⇒ *P*) *f*
shows *asymmetric-on Q* (*rel-map f R*)
⟨*proof*⟩

consts *asymmetric* :: *'a* ⇒ *bool*

overloading
asymmetric ≡ *asymmetric* :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*
begin
definition *asymmetric* :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ - ≡ *asymmetric-on* (⊤ :: *'a* ⇒ *bool*)
end

lemma *asymmetric-eq-asymmetric-on*:
(*asymmetric* :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ -) = *asymmetric-on* (⊤ :: *'a* ⇒ *bool*)
⟨*proof*⟩

lemma *asymmetric-eq-asymmetric-on-uhint* [*uhint*]:
$P \equiv (\top :: 'a \Rightarrow bool) \Longrightarrow (asymmetric :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow \text{-}) \equiv asymmetric\text{-}on\ P$
⟨*proof*⟩

lemma *asymmetricI* [*intro*]:
assumes $\bigwedge x\ y.\ R\ x\ y \Longrightarrow \neg(R\ y\ x)$
shows *asymmetric R*
⟨*proof*⟩

lemma *asymmetricD* [*dest*]:
assumes *asymmetric R*
and *R x y*
shows $\neg(R\ y\ x)$
⟨*proof*⟩

lemma *asymmetricE*:
assumes *asymmetric R*
obtains $\bigwedge x\ y.\ R\ x\ y \Longrightarrow \neg(R\ y\ x)$
⟨*proof*⟩

lemma *asymmetric-on-if-asymmetric*:
fixes $P :: 'a \Rightarrow bool$ **and** $R :: 'a \Rightarrow 'a \Rightarrow bool$
assumes *asymmetric R*
shows *asymmetric-on P R*
⟨*proof*⟩

lemma *asymmetric-if-asymmetric-on-in-field*:
assumes *asymmetric-on* (*in-field R*) *R*
shows *asymmetric R*
⟨*proof*⟩

corollary *asymmetric-on-in-field-iff-asymmetric* [*iff*]:
asymmetric-on (*in-field R*) $R \longleftrightarrow$ *asymmetric R*
⟨*proof*⟩

lemma *asymmetric-on-if-asymmetric-restrict*:
assumes *asymmetric* $R{\restriction_P}$
shows *asymmetric-on P R*
⟨*proof*⟩

lemma *asymmetric-restrict-if-asymmetric-on*:
assumes *asymmetric-on P R*
shows *asymmetric* $R{\restriction_P}$
⟨*proof*⟩

corollary *asymmetric-restrict-iff-asymmetric-on* [*iff*]: *asymmetric* $R{\restriction_P} \longleftrightarrow$ *asymmetric-on P R*
⟨*proof*⟩

lemma *asymmetric-le-irreflexive*:
(*asymmetric* :: ('*a* ⇒ '*a* ⇒ *bool*) ⇒ -) ≤ *irreflexive*
⟨*proof*⟩

lemma *asymmetric-le-antisymmetric*:
(*asymmetric* :: ('*a* ⇒ '*a* ⇒ *bool*) ⇒ -) ≤ *antisymmetric*
⟨*proof*⟩

lemma *antimono-asymmetric*: *antimono* (*asymmetric* :: ('*a* ⇒ '*a* ⇒ *bool*) ⇒ -)
⟨*proof*⟩

end

Surjective

theory *Binary-Relations-Surjective*
imports
Binary-Relations-Left-Total
HOL-Syntax-Bundles-Lattices
begin

consts *rel-surjective-at* :: '*a* ⇒ '*b* ⇒ *bool*

overloading
rel-surjective-at-pred ≡ *rel-surjective-at* :: ('*a* ⇒ *bool*) ⇒ ('*b* ⇒ '*a* ⇒ *bool*) ⇒ *bool*
begin
definition *rel-surjective-at-pred P R* ≡ ∀ *y* : *P*. *in-codom R y*
end

lemma *rel-surjective-atI* [*intro*]:
assumes ⋀*y*. *P y* ⟹ *in-codom R y*
shows *rel-surjective-at P R*
⟨*proof*⟩

lemma *rel-surjective-atE* [*elim*]:
assumes *rel-surjective-at P R*
and *P y*
obtains *x* **where** *R x y*
⟨*proof*⟩

lemma *in-codom-if-rel-surjective-at*:
assumes *rel-surjective-at P R*
and *P y*
shows *in-codom R y*
⟨*proof*⟩

lemma *rel-surjective-at-rel-inv-iff-left-total-on* [*iff*]:
rel-surjective-at $(P :: 'a \Rightarrow bool)$ $(R^{-1} :: 'b \Rightarrow 'a \Rightarrow bool) \longleftrightarrow$ *left-total-on P R*
⟨*proof*⟩

lemma *left-total-on-rel-inv-iff-rel-surjective-at* [*iff*]:
left-total-on $(P :: 'a \Rightarrow bool)$ $(R^{-1} :: 'a \Rightarrow 'b \Rightarrow bool) \longleftrightarrow$ *rel-surjective-at P R*
⟨*proof*⟩

lemma *mono-rel-surjective-at*:
$((\geq) \Rightarrow (\leq) \Rrightarrow (\leq))$ $(rel\text{-}surjective\text{-}at :: ('b \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool)$
⟨*proof*⟩

lemma *rel-surjective-at-iff-le-codom*:
rel-surjective-at $(P :: 'b \Rightarrow bool)$ $(R :: 'a \Rightarrow 'b \Rightarrow bool) \longleftrightarrow P \leq$ *in-codom R*
⟨*proof*⟩

lemma *rel-surjective-at-compI*:
fixes $P :: 'c \Rightarrow bool$ **and** $R :: 'a \Rightarrow 'b \Rightarrow bool$ **and** $S :: 'b \Rightarrow 'c \Rightarrow bool$
assumes *surj-R*: *rel-surjective-at* (*in-dom S*) *R*
and *surj-S*: *rel-surjective-at P S*
shows *rel-surjective-at P* $(R \circ\circ S)$
⟨*proof*⟩

consts *rel-surjective* $:: 'a \Rightarrow bool$

overloading
rel-surjective $\equiv$ *rel-surjective* $:: ('b \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
begin
definition (*rel-surjective* $:: ('b \Rightarrow 'a \Rightarrow bool) \Rightarrow$ -) $\equiv$ *rel-surjective-at* $(\top :: 'a \Rightarrow bool)$
end

lemma *rel-surjective-eq-rel-surjective-at*:
(*rel-surjective* $:: ('b \Rightarrow 'a \Rightarrow bool) \Rightarrow$ -) $=$ *rel-surjective-at* $(\top :: 'a \Rightarrow bool)$
⟨*proof*⟩

lemma *rel-surjective-eq-rel-surjective-at-uhint* [*uhint*]:
assumes $P \equiv (\top :: 'a \Rightarrow bool)$
shows (*rel-surjective* $:: ('b \Rightarrow 'a \Rightarrow bool) \Rightarrow$ -) $\equiv$ *rel-surjective-at P*
⟨*proof*⟩

lemma *rel-surjectiveI*:
assumes $\bigwedge y.$ *in-codom R y*
shows *rel-surjective R*
⟨*proof*⟩

lemma *rel-surjectiveE*:
assumes *rel-surjective R*
obtains *x* **where** *R x y*

⟨*proof*⟩

lemma *in-codom-if-rel-surjective*:
assumes *rel-surjective R*
shows *in-codom R y*
⟨*proof*⟩

lemma *rel-surjective-rel-inv-iff-left-total* [*iff*]: *rel-surjective* R^{-1} $\longleftrightarrow$ *left-total R*
⟨*proof*⟩

lemma *left-total-rel-inv-iff-rel-surjective* [*iff*]: *left-total* R^{-1} $\longleftrightarrow$ *rel-surjective R*
⟨*proof*⟩

lemma *rel-surjective-at-if-surjective*:
fixes $P :: 'a \Rightarrow bool$ **and** $R :: 'b \Rightarrow 'a \Rightarrow bool$
assumes *rel-surjective R*
shows *rel-surjective-at P R*
⟨*proof*⟩

end

Bi-Total

theory *Binary-Relations-Bi-Total*
imports
Binary-Relations-Left-Total
Binary-Relations-Surjective
begin

definition *bi-total-on P Q* $\equiv$ *left-total-on P* $\sqcap$ *rel-surjective-at Q*

lemma *bi-total-onI* [*intro*]:
assumes *left-total-on P R*
and *rel-surjective-at Q R*
shows *bi-total-on P Q R*
⟨*proof*⟩

lemma *bi-total-onE* [*elim*]:
assumes *bi-total-on P Q R*
obtains *left-total-on P R rel-surjective-at Q R*
⟨*proof*⟩

lemma *bi-total-on-restricts-if-Fun-Rel-iff-if-bi-total-on*:
assumes *bi-total-on P Q R*
and $(R \Rrightarrow (\longleftrightarrow))\ P\ Q$
shows *bi-total-on P Q* $R\restriction_P\restriction_Q$
⟨*proof*⟩

definition *bi-total* ≡ *bi-total-on* (⊤ :: *'a* ⇒ *bool*) (⊤ :: *'b* ⇒ *bool*) :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ *bool*

lemma *bi-total-eq-bi-total-on*:
(*bi-total* :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ *bool*) = *bi-total-on* (⊤ :: *'a* ⇒ *bool*) (⊤ :: *'b* ⇒ *bool*)
⟨*proof*⟩

lemma *bi-total-eq-bi-total-on-uhint* [*uhint*]:
assumes *P* ≡ (⊤ :: *'a* ⇒ *bool*)
and *Q* ≡ (⊤ :: *'b* ⇒ *bool*)
shows (*bi-total* :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ *bool*) ≡ *bi-total-on P Q*
⟨*proof*⟩

lemma *bi-totalI* [*intro*]:
assumes *left-total R*
and *rel-surjective R*
shows *bi-total R*
⟨*proof*⟩

lemma *bi-totalE* [*elim*]:
assumes *bi-total R*
obtains *left-total R rel-surjective R*
⟨*proof*⟩

end

Bi-Unique

theory *Binary-Relations-Bi-Unique*
imports
Binary-Relations-Injective
Binary-Relations-Right-Unique
begin

definition *bi-unique-on* ≡ *right-unique-on* ⊓ *rel-injective-on*

lemma *bi-unique-onI* [*intro*]:
assumes *right-unique-on P R*
and *rel-injective-on P R*
shows *bi-unique-on P R*
⟨*proof*⟩

lemma *bi-unique-onE* [*elim*]:
assumes *bi-unique-on P R*
obtains *right-unique-on P R rel-injective-on P R*
⟨*proof*⟩

lemma *bi-unique-on-rel-inv-if-Fun-Rel-iff-if-bi-unique-on*:

assumes *bi-unique-on P R*
and $(R \Rrightarrow (\longleftrightarrow))$ *P Q*
shows *bi-unique-on Q* R^{-1}
⟨*proof*⟩

definition *bi-unique* ≡ *bi-unique-on* $(\top :: 'a \Rightarrow bool) :: ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool$

lemma *bi-unique-eq-bi-unique-on*:
bi-unique = (*bi-unique-on* $(\top :: 'a \Rightarrow bool) :: ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool$)
⟨*proof*⟩

lemma *bi-unique-eq-bi-unique-on-uhint* [*uhint*]:
assumes $P \equiv (\top :: 'a \Rightarrow bool)$
shows *bi-unique* ≡ (*bi-unique-on P* $:: ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool$)
⟨*proof*⟩

lemma *bi-uniqueI* [*intro*]:
assumes *right-unique R*
and *rel-injective R*
shows *bi-unique R*
⟨*proof*⟩

lemma *bi-uniqueE* [*elim*]:
assumes *bi-unique R*
obtains *right-unique R rel-injective R*
⟨*proof*⟩

end

Injective

theory *Functions-Injective*
imports
Bounded-Quantifiers
Functions-Monotone
HOL-Syntax-Bundles-Lattices
begin

consts *injective-on* $:: 'a \Rightarrow 'b \Rightarrow bool$

overloading
injective-on-pred ≡ *injective-on* $:: ('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool$
begin
definition *injective-on-pred P f* $\equiv \forall\, x\ x' : P.\ f\ x = f\ x' \longrightarrow x = x'$
end

lemma *injective-onI* [*intro*]:
assumes $\bigwedge x\ x'.\ P\ x \Longrightarrow P\ x' \Longrightarrow f\ x = f\ x' \Longrightarrow x = x'$
shows *injective-on P f*

⟨*proof*⟩

lemma *injective-onD*:
assumes *injective-on P f*
and *P x P x'*
and *f x = f x'*
shows *x = x'*
⟨*proof*⟩

lemma *injective-on-comp-if-injective-onI*:
assumes *injective-on* (*P* :: *'a* ⇒ *bool*) *f injective-on Q g*
and (*P* ⇒ *Q*) *f*
shows *injective-on P* (*g* ∘ *f*)
⟨*proof*⟩

consts *injective* :: *'a* ⇒ *bool*

overloading
injective ≡ *injective* :: (*'a* ⇒ *'b*) ⇒ *bool*
begin
definition (*injective* :: (*'a* ⇒ *'b*) ⇒ *bool*) ≡ *injective-on* (⊤ :: *'a* ⇒ *bool*)
end

lemma *injective-eq-injective-on*:
(*injective* :: (*'a* ⇒ *'b*) ⇒ *bool*) = *injective-on* (⊤ :: *'a* ⇒ *bool*)
⟨*proof*⟩

lemma *injective-eq-injective-on-uhint* [*uhint*]:
assumes *P* ≡ (⊤ :: *'a* ⇒ *bool*)
shows *injective* :: (*'a* ⇒ *'b*) ⇒ *bool* ≡ *injective-on P*
⟨*proof*⟩

lemma *injectiveI* [*intro*]:
assumes ⋀*x x'*. *f x = f x'* ⟹ *x = x'*
shows *injective f*
⟨*proof*⟩

lemma *injectiveD*:
assumes *injective f*
and *f x = f x'*
shows *x = x'*
⟨*proof*⟩

lemma *injective-on-if-injective*:
fixes *P* :: *'a* ⇒ *bool* **and** *f* :: *'a* ⇒ -
assumes *injective f*
shows *injective-on P f*
⟨*proof*⟩

Instantiations **lemma** *injective-id*: *injective id* ⟨*proof*⟩

end

Connected

theory *Binary-Relations-Connected*
imports
Binary-Relation-Functions
Functions-Injective
begin

consts *connected-on* :: $'a \Rightarrow 'b \Rightarrow bool$

overloading
connected-on-pred ≡ *connected-on* :: $('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
begin
definition *connected-on-pred* $P\ R \equiv \forall x\ y : P.\ x \neq y \longrightarrow R\ x\ y \vee R\ y\ x$
end

lemma *connected-onI* [*intro*]:
assumes $\bigwedge x\ y.\ P\ x \Longrightarrow P\ y \Longrightarrow x \neq y \Longrightarrow R\ x\ y \vee R\ y\ x$
shows *connected-on* $P\ R$
⟨*proof*⟩

lemma *connected-onE* [*elim*]:
assumes *connected-on* $P\ R$
and $P\ x\ P\ y$
obtains $x = y \mid R\ x\ y \mid R\ y\ x$
⟨*proof*⟩

lemma *connected-on-rel-inv-iff-connected-on* [*iff*]:
connected-on $(P :: 'a \Rightarrow bool)\ (R :: 'a \Rightarrow 'a \Rightarrow bool)^{-1} \longleftrightarrow$ *connected-on* $P\ R$
⟨*proof*⟩

lemma *connected-on-rel-map-if-injective-on-if-mono-wrt-pred-if-connected-on*:
fixes $P :: 'a \Rightarrow bool$ **and** $R :: 'a \Rightarrow 'a \Rightarrow bool$
assumes *connected-on* $P\ R$
and $(Q \Rightarrow P)\ f$
and *injective-on* $Q\ f$
shows *connected-on* Q (*rel-map* $f\ R$)
⟨*proof*⟩

lemma *antimono-connected-on*: *antimono* (*connected-on* :: $('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$)
⟨*proof*⟩

lemma *mono-connected-on*:

((≤) ⇒ (≥) ⇛ (≥)) (*connected-on* :: ('*a* ⇒ *bool*) ⇒ ('*a* ⇒ '*a* ⇒ *bool*) ⇒ *bool*)
⟨*proof*⟩

lemma *in-field-restrict-eq-if-ex-ne-if-connected-on*:
assumes *conn*: *connected-on P R*
and *eq-ne*: ∃ *x y* : *P*. *x* ≠ *y*
shows *in-field* $R{\uparrow}_P$ = *P*
⟨*proof*⟩

consts *connected* :: '*a* ⇒ *bool*

overloading
connected ≡ *connected* :: ('*a* ⇒ '*a* ⇒ *bool*) ⇒ *bool*
begin
definition *connected* :: ('*a* ⇒ '*a* ⇒ *bool*) ⇒ *bool* ≡ *connected-on* (⊤ :: '*a* ⇒ *bool*)
end

lemma *connected-eq-connected-on*:
(*connected* :: ('*a* ⇒ '*a* ⇒ *bool*) ⇒ -) = *connected-on* (⊤ :: '*a* ⇒ *bool*)
⟨*proof*⟩

lemma *connected-eq-connected-on-uhint* [*uhint*]:
P ≡ (⊤ :: '*a* ⇒ *bool*) ⟹ (*connected* :: ('*a* ⇒ '*a* ⇒ *bool*) ⇒ -) ≡ *connected-on P*
⟨*proof*⟩

lemma *connectedI* [*intro*]:
assumes ⋀*x y*. *x* ≠ *y* ⟹ *R x y* ∨ *R y x*
shows *connected R*
⟨*proof*⟩

lemma *connectedE* [*elim*]:
assumes *connected R*
and *x* ≠ *y*
obtains *R x y* | *R y x*
⟨*proof*⟩

lemma *connected-on-if-connected*:
fixes *P* :: '*a* ⇒ *bool* **and** *R* :: '*a* ⇒ '*a* ⇒ *bool*
assumes *connected R*
shows *connected-on P R*
⟨*proof*⟩

end

Well-Founded Relations

theory *Binary-Relations-Wellfounded*
  imports
    Functions-Monotone
begin

consts *wellfounded-on* :: $'a \Rightarrow 'b \Rightarrow bool$

overloading
  wellfounded-on-pred $\equiv$ *wellfounded-on* :: $('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
begin
  definition *wellfounded-on-pred* $(P :: 'a \Rightarrow bool)$ $(R :: 'a \Rightarrow 'a \Rightarrow bool)$
    $\equiv \forall\, Q.\ (\exists\, x : P.\ Q\ x) \longrightarrow (\exists\, m : P.\ Q\ m \wedge (\forall\, y : P.\ R\ y\ m \longrightarrow \neg(Q\ y)))$
end

lemma *wellfounded-onI*:
  assumes $\bigwedge Q\ x.\ P\ x \Longrightarrow Q\ x \Longrightarrow \exists\, m : P.\ Q\ m \wedge (\forall\, y : P.\ R\ y\ m \longrightarrow \neg(Q\ y))$
  shows *wellfounded-on* $P\ R$
  ⟨*proof*⟩

lemma *wellfounded-onE*:
  assumes *wellfounded-on* $P\ R\ P\ x\ Q\ x$
  obtains m **where** $P\ m\ Q\ m\ \bigwedge y.\ P\ y \Longrightarrow R\ y\ m \Longrightarrow \neg(Q\ y)$
⟨*proof*⟩

lemma *wellfounded-on-induct* [*consumes 1*, *case-names step*]:
  assumes *wellfounded-on* $P\ R\ P\ z$
  assumes *step*: $\bigwedge x.\ P\ x \Longrightarrow (\bigwedge y.\ P\ y \Longrightarrow R\ y\ x \Longrightarrow Q\ y) \Longrightarrow Q\ x$
  shows $Q\ z$
⟨*proof*⟩

lemma *wellfounded-on-rel-map-if-mono-wrt-pred-if-wellfounded-on*:
  fixes $A :: 'a \Rightarrow bool$ **and** $B :: 'b \Rightarrow bool$
  assumes *wf*: *wellfounded-on* $A\ R$
  and $(B \Rightarrow A)\ f$
  shows *wellfounded-on* B $(rel\text{-}map\ f\ R)$
⟨*proof*⟩

corollary *wellfounded-on-if-le-pred-if-wellfounded-on*:
  fixes $P :: 'a \Rightarrow bool$ **and** $R :: 'a \Rightarrow 'a \Rightarrow bool$
  assumes *wellfounded-on* $P\ R\ Q \leq P$
  shows *wellfounded-on* $Q\ R$
⟨*proof*⟩

lemma *wellfounded-on-if-le-rel-if-wellfounded-on*:
  fixes $P :: 'a \Rightarrow bool$ **and** $R :: 'a \Rightarrow 'a \Rightarrow bool$
  assumes *wellfounded-on* $P\ R\ S \leq R$
  shows *wellfounded-on* $P\ S$
⟨*proof*⟩

corollary *antimono-wellfounded-on*:
(($\leq$) $\Rightarrow$ ($\leq$) $\Rrightarrow$ ($\geq$)) (*wellfounded-on* :: ('*a* $\Rightarrow$ *bool*) $\Rightarrow$ ('*a* $\Rightarrow$ '*a* $\Rightarrow$ *bool*) $\Rightarrow$ *bool*)
⟨*proof*⟩

consts *wellfounded* :: '*a* $\Rightarrow$ *bool*

overloading
wellfounded $\equiv$ *wellfounded* :: ('*a* $\Rightarrow$ '*a* $\Rightarrow$ *bool*) $\Rightarrow$ *bool*
begin
definition (*wellfounded* :: ('*a* $\Rightarrow$ '*a* $\Rightarrow$ *bool*) $\Rightarrow$ *bool*) $\equiv$ *wellfounded-on* ($\top$ :: '*a* $\Rightarrow$ *bool*)
end

lemma *wellfounded-eq-wellfounded-on*:
(*wellfounded* :: ('*a* $\Rightarrow$ '*a* $\Rightarrow$ *bool*) $\Rightarrow$ -) = *wellfounded-on* ($\top$ :: '*a* $\Rightarrow$ *bool*)
⟨*proof*⟩

lemma *wellfounded-eq-wellfounded-on-uhint* [*uhint*]:
P $\equiv$ ($\top$ :: '*a* $\Rightarrow$ *bool*) $\Longrightarrow$ (*wellfounded* :: ('*a* $\Rightarrow$ '*a* $\Rightarrow$ *bool*) $\Rightarrow$ -) $\equiv$ *wellfounded-on* *P*
⟨*proof*⟩

lemma *wellfoundedI* [*intro*]:
assumes $\bigwedge Q\ x.\ Q\ x \Longrightarrow (\exists\, m : Q.\ \forall\, y.\ R\ y\ m \longrightarrow \neg(Q\ y))$
shows *wellfounded R*
⟨*proof*⟩

lemma *wellfoundedE*:
assumes *wellfounded R Q x*
obtains *m* **where** *Q m* $\bigwedge y.\ R\ y\ m \Longrightarrow \neg(Q\ y)$
⟨*proof*⟩

lemma *wellfounded-on-if-wellfounded*:
fixes *P* :: '*a* $\Rightarrow$ *bool* **and** *R* :: '*a* $\Rightarrow$ '*a* $\Rightarrow$ *bool*
assumes *wellfounded R*
shows *wellfounded-on P R*
⟨*proof*⟩

lemma *wellfounded-induct* [*consumes 1*, *case-names step*]:
assumes *wellfounded R*
assumes $\bigwedge x.\ (\bigwedge y.\ R\ y\ x \Longrightarrow Q\ y) \Longrightarrow Q\ x$
shows *Q x*
⟨*proof*⟩

lemma *wellfounded-rel-restrict-if-wellfounded-on*:
assumes *wellfounded-on P R*
shows *wellfounded* $R{\uparrow}_P$

⟨*proof*⟩

end

1.3.10 Basic Properties

theory *Binary-Relation-Properties*
imports
Binary-Relations-Antisymmetric
Binary-Relations-Asymmetric
Binary-Relations-Bi-Total
Binary-Relations-Bi-Unique
Binary-Relations-Connected
Binary-Relations-Injective
Binary-Relations-Irreflexive
Binary-Relations-Left-Total
Binary-Relations-Reflexive
Binary-Relations-Right-Unique
Binary-Relations-Surjective
Binary-Relations-Symmetric
Binary-Relations-Transitive
Binary-Relations-Wellfounded
begin

end

1.3.11 Bounded Definite Description

theory *Bounded-Definite-Description*
imports
Bounded-Quantifiers
begin

consts *bthe* :: $'a \Rightarrow ('b \Rightarrow bool) \Rightarrow 'b$

open-bundle *bounded-the-syntax*
begin
syntax *-bthe* :: [*idt*, $'a$, *bool*] $\Rightarrow 'b$ (‹(*3THE* - : -./ -)› [*0*, *0*, *10*] *10*)
end

syntax-consts *-bthe* $\rightleftharpoons$ *bthe*
translations *THE* x : P. $Q \rightleftharpoons$ *CONST bthe* P (λx. Q)

definition *bthe-pred* P $Q \equiv$ *The* ($P \sqcap Q$)
adhoc-overloading *bthe* $\rightleftharpoons$ *bthe-pred*

lemma *bthe-eqI* [*intro*]:
assumes Q a

and $P\ a$
and $\bigwedge x.\ [\![P\ x;\ Q\ x]\!] \Longrightarrow x = a$
shows $(THE\ x : P.\ Q\ x) = a$
⟨*proof*⟩

lemma *pred-bthe-if-ex1E*:
assumes $\exists!x : P.\ Q\ x$
obtains $P\ (THE\ x : P.\ Q\ x)\ Q\ (THE\ x : P.\ Q\ x)$
⟨*proof*⟩

lemma *pred-btheI*:
assumes $\exists!x : P.\ Q\ x$
shows $P\ (THE\ x : P.\ Q\ x)$
⟨*proof*⟩

lemma *pred-btheI'*:
assumes $\exists!x : P.\ Q\ x$
shows $Q\ (THE\ x : P.\ Q\ x)$
⟨*proof*⟩

end

1.3.12 Least Element With Respect To Some Relation

theory *Binary-Relations-Least*
imports
Binary-Relations-Antisymmetric
Bounded-Definite-Description
begin

definition *is-minimal-wrt-rel* $R\ (P :: 'a \Rightarrow bool)\ x \equiv P\ x \wedge (\forall\, y : P.\ R\ x\ y)$

lemma *is-minimal-wrt-relI* [*intro*]:
assumes $P\ x$
and $\bigwedge y.\ P\ y \Longrightarrow R\ x\ y$
shows *is-minimal-wrt-rel* $R\ P\ x$
⟨*proof*⟩

lemma *is-minimal-wrt-relE* [*elim*]:
assumes *is-minimal-wrt-rel* $R\ P\ x$
obtains $P\ x\ \bigwedge y.\ P\ y \Longrightarrow R\ x\ y$
⟨*proof*⟩

definition *is-least-wrt-rel* $R\ (P :: 'a \Rightarrow bool)\ x \equiv$
is-minimal-wrt-rel $R\ P\ x \wedge (\forall\, x'.$ *is-minimal-wrt-rel* $R\ P\ x' \longrightarrow x' = x)$

lemma *is-least-wrt-relI* [*intro*]:

assumes *is-minimal-wrt-rel R P x*
and $\bigwedge x'$. *is-minimal-wrt-rel R P x'* $\Longrightarrow x' = x$
shows *is-least-wrt-rel R P x*
⟨*proof*⟩

lemma *is-least-wrt-relE* [*elim*]:
assumes *is-least-wrt-rel R P x*
obtains *is-minimal-wrt-rel R P x* $\bigwedge x'$. *is-minimal-wrt-rel R P x'* $\Longrightarrow x' = x$
⟨*proof*⟩

lemma *is-least-wrt-rel-if-antisymmetric-onI*:
assumes *antisymmetric-on P R*
and *is-minimal-wrt-rel R P x*
shows *is-least-wrt-rel R P x*
⟨*proof*⟩

corollary *is-least-wrt-rel-eq-is-minimal-wrt-rel-if-antisymmetric-on* [*simp*]:
assumes *antisymmetric-on P R*
shows *is-least-wrt-rel R P* = *is-minimal-wrt-rel R P*
⟨*proof*⟩

definition *least-wrt-rel R* (*P* :: *'a* ⇒ *bool*) ≡ *THE x*. *is-least-wrt-rel R P x*

lemma *least-wrt-rel-eqI*:
assumes *is-least-wrt-rel R P x*
shows *least-wrt-rel R P* = *x*
⟨*proof*⟩

lemma *least-wrt-rel-eq-if-antisymmetric-onI*:
assumes *antisymmetric-on P R*
and *is-minimal-wrt-rel R P x*
shows *least-wrt-rel R P* = *x*
⟨*proof*⟩

lemma *pred-least-wrt-relI*:
assumes *is-least-wrt-rel R P x*
shows *P* (*least-wrt-rel R P*)
⟨*proof*⟩

lemma *pred-least-wrt-rel-if-antisymmetric-onI*:
assumes *antisymmetric-on P R*
and *is-minimal-wrt-rel R P x*
shows *P* (*least-wrt-rel R P*)
⟨*proof*⟩

end

1.3.13 Transitive Closure

theory *Binary-Relations-Transitive-Closure*
imports
Binary-Relations-Least
Binary-Relations-Transitive
begin

consts *trans-closure-on* :: $'a \Rightarrow 'b \Rightarrow 'b$

definition *trans-closure-on-pred* $(P :: 'a \Rightarrow bool)\ R\ x\ y \equiv$
$\forall (R' :: 'a \Rightarrow 'a \Rightarrow bool) : transitive\text{-}on\ P.\ R \leq R' \longrightarrow R'\ x\ y$
adhoc-overloading *trans-closure-on* $\rightleftharpoons$ *trans-closure-on-pred*

lemma *trans-closure-on-if-all-trans-relI*:
assumes $\bigwedge R'.\ transitive\text{-}on\ P\ R' \Longrightarrow R \leq R' \Longrightarrow R'\ x\ y$
shows *trans-closure-on P R x y*
⟨*proof*⟩

lemma *all-rel-if-trans-closure-onE*:
assumes *trans-closure-on P R x y*
obtains $\bigwedge R'.\ transitive\text{-}on\ P\ R' \Longrightarrow R \leq R' \Longrightarrow R'\ x\ y$
⟨*proof*⟩

lemma *transitive-on-trans-closure-on*: *transitive-on P* (*trans-closure-on P R*)
⟨*proof*⟩

lemma *trans-closure-on-le-if-le-if-transitive-on*:
assumes *transitive-on P S*
and $R \leq S$
shows *trans-closure-on P R* $\leq S$
⟨*proof*⟩

lemma *trans-closure-on-if-rel*: $R\ x\ y \Longrightarrow$ *trans-closure-on P R x y*
⟨*proof*⟩

corollary *le-trans-closure-on*: $R \leq$ *trans-closure-on P R*
⟨*proof*⟩

corollary *is-least-wrt-rel-trans-closure-on*:
is-least-wrt-rel $(\leq)$ $((\leq)\ R \sqcap$ *transitive-on P*) (*trans-closure-on P R*)
⟨*proof*⟩

corollary *trans-closure-on-eq-least-wrt-rel*:
trans-closure-on P R = *least-wrt-rel* $(\leq)$ $((\leq)\ R \sqcap$ *transitive-on P*)
⟨*proof*⟩

lemma *trans-closure-on-le-sup*:

fixes P R **defines** $S \equiv R \sqcup (\lambda x\ y.\ P\ x \wedge P\ y \wedge (\exists\, z : P.\ \textit{trans-closure-on}\ P\ R\ x\ z \wedge R\ z\ y))$
shows $\textit{trans-closure-on}\ P\ R \leq S$
⟨*proof*⟩

lemma *trans-closure-on-cases*:
assumes $\textit{trans-closure-on}\ P\ R\ x\ y$
obtains (*rel*) $R\ x\ y$ | (*step*) z **where** $P\ x\ P\ z\ P\ y\ \textit{trans-closure-on}\ P\ R\ x\ z\ R\ z\ y$
⟨*proof*⟩

lemma *trans-closure-on-eq-rel-sup-trans-closure-on*:
$\textit{trans-closure-on}\ P\ R = R \sqcup \textit{trans-closure-on}\ P\ R\restriction_P\restriction_P$
⟨*proof*⟩

consts *trans-closure* :: $'a \Rightarrow 'a$

definition $\textit{trans-closure-rel} \equiv \textit{trans-closure-on}\ \top$
adhoc-overloading $\textit{trans-closure} \rightleftharpoons \textit{trans-closure-rel}$

lemma *trans-closure-eq-trans-closure-on*: $\textit{trans-closure} = \textit{trans-closure-on}\ \top$
⟨*proof*⟩

lemma *trans-closure-eq-trans-closure-on-uhint* [*uhint*]:
assumes $P \equiv \top$
shows $\textit{trans-closure} = \textit{trans-closure-on}\ P$
⟨*proof*⟩

lemma *trans-closure-iff-trans-closure-on*: $\textit{trans-closure}\ R\ x\ y = \textit{trans-closure-on}\ \top\ R\ x\ y$
⟨*proof*⟩

lemma *all-rel-if-trans-closureE*:
assumes $\textit{trans-closure}\ R\ x\ y$
obtains $\bigwedge R'.\ \textit{transitive}\ R' \Longrightarrow R \leq R' \Longrightarrow R'\ x\ y$
⟨*proof*⟩

lemma *transitive-trans-closure*: $\textit{transitive}\ (\textit{trans-closure}\ R)$
⟨*proof*⟩

lemma *trans-closure-le-if-le-if-transitive*:
assumes $\textit{transitive}\ S$
and $R \leq S$
shows $\textit{trans-closure}\ R \leq S$
⟨*proof*⟩

lemma *trans-closure-if-rel*: $R\ x\ y \Longrightarrow \textit{trans-closure}\ R\ x\ y$
⟨*proof*⟩

corollary *trans-closure-eq-least-wrt-rel*: $\textit{trans-closure}\ R = \textit{least-wrt-rel}\ (\leq)\ ((\leq)\ R$

$\sqcap$ *transitive*)
⟨*proof*⟩

lemma *trans-closure-cases*:
assumes *trans-closure R x y*
obtains (*rel*) *R x y* | (*step*) *z* **where** *trans-closure R x z R z y*
⟨*proof*⟩

end

theory *Functions-Restrict*
imports *HOL-Basics-Base*
begin

consts *fun-restrict* :: *'a* ⇒ *'b* ⇒ *'a*

bundle *fun-restrict-syntax*
begin
notation *fun-restrict* (‹(-)↾(_)› [*1000*])
end

definition *fun-restrict-pred f P x* ≡ *if P x then f x else undefined*
adhoc-overloading *fun-restrict* ⇌ *fun-restrict-pred*

context
includes *fun-restrict-syntax*
begin

lemma *fun-restrict-eq* [*simp*]:
assumes *P x*
shows $f{\restriction}_P\ x = f\ x$
⟨*proof*⟩

lemma *fun-restrict-eq-if-not* [*simp*]:
assumes ¬(*P x*)
shows $f{\restriction}_P\ x$ = *undefined*
⟨*proof*⟩

lemma *fun-restrict-eq-if*: $f{\restriction}_P\ x$ = (*if P x then f x else undefined*)
⟨*proof*⟩

lemma *fun-restrict-cong* [*cong*]:
assumes *P* = *P'*
and ⋀*x*. *P' x* ⟹ *f x* = *g x*
shows $f{\restriction}_P = g{\restriction}_{P'}$
⟨*proof*⟩

end

end

1.3.14 Well-Founded Transitive Recursion

theory *Wellfounded-Transitive-Recursion*
imports
Binary-Relations-Transitive
Binary-Relations-Wellfounded
Binary-Relation-Functions
Functions-Restrict
begin

Summary Well-founded recursion on transitive, well-founded relations. One can extend this to well-founded recursion on non-transitive, well-founded relations since the transitive closure of a well-founded relation is well-founded.

context
includes *fun-restrict-syntax* **and** *no rel-restrict-syntax*
begin

definition *fun-rel-restrict f R y* $\equiv f\restriction_{R^{-1}}\ y :: 'a \Rightarrow bool$

lemma *fun-rel-restrict-eq* [*simp*]:
assumes *R x y*
shows *fun-rel-restrict f R y x* = *f x*
⟨*proof*⟩

lemma *fun-rel-restrict-if-not* [*simp*]:
assumes ¬(*R x y*)
shows *fun-rel-restrict f R y x* = *undefined*
⟨*proof*⟩

lemma *fun-rel-restrict-eq-fun-restrict*: *fun-rel-restrict f R y* = $f\restriction_{R^{-1}}\ y$
⟨*proof*⟩

lemma *fun-rel-restrict-cong* [*cong*]:
assumes *y* = *y'*
and $\bigwedge x.\ R\ x\ y' \longleftrightarrow R'\ x\ y'$
and $\bigwedge x.\ R'\ x\ y' \Longrightarrow f\ x = g\ x$
shows *fun-rel-restrict f R y* = *fun-rel-restrict g R' y*
⟨*proof*⟩

lemma *fun-rel-restrict-rel-restrict-eq-fun-restrict-fun-rel-restrictI* [*simp*]:
assumes *P x*
shows *fun-rel-restrict f* (*rel-restrict R P*) *x* = (*fun-rel-restrict f R x*)$\restriction_P$
⟨*proof*⟩

context
  fixes $R :: 'a \Rightarrow 'a \Rightarrow bool$ **and** $step :: ('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b$
begin

definition *wf-rec-step* $f\ x = step\ (fun\text{-}rel\text{-}restrict\ (f\ x)\ R\ x)\ x$

lemma *wf-rec-step-eq*: $wf\text{-}rec\text{-}step\ f\ x = step\ (fun\text{-}rel\text{-}restrict\ (f\ x)\ R\ x)\ x$
  ⟨*proof*⟩

definition *is-recfun* $X\ f \equiv f = fun\text{-}rel\text{-}restrict\ (wf\text{-}rec\text{-}step\ (\lambda\text{-}.\ f))\ R\ X$

definition *the-recfun* $X = (THE\ f.\ is\text{-}recfun\ X\ f)$

lemma *is-recfunI* [*intro*]:
  assumes $\bigwedge x.\ R\ x\ X \Longrightarrow f\ x = wf\text{-}rec\text{-}step\ (\lambda\text{-}.\ f)\ x$
  and $\bigwedge x.\ \neg(R\ x\ X) \Longrightarrow f\ x = undefined$
  shows *is-recfun* $X\ f$
  ⟨*proof*⟩

lemma *is-recfunE* [*elim*]:
  assumes *is-recfun* $X\ f$
  obtains $\bigwedge x.\ R\ x\ X \Longrightarrow f\ x = wf\text{-}rec\text{-}step\ (\lambda\text{-}.\ f)\ x$
    $\bigwedge x.\ \neg(R\ x\ X) \Longrightarrow f\ x = undefined$
  ⟨*proof*⟩

lemma *eq-if-rel-if-is-recfun*:
  assumes *is-recfun* $X\ f$
  and $R\ x\ X$
  shows $f\ x = wf\text{-}rec\text{-}step\ (\lambda\text{-}.\ f)\ x$
  ⟨*proof*⟩

lemma *eq-if-not-rel-if-is-recfun*:
  assumes *is-recfun* $X\ f$
  and $\neg(R\ x\ X)$
  shows $f\ x = undefined$
  ⟨*proof*⟩

context
  assumes *wf*: *wellfounded* R
  and *trans*: *transitive* R
begin

lemma *app-eq-app-if-is-recfunI*:
  assumes *is-recfun* $X\ f$ *is-recfun* $Y\ g$
  and $R\ Z\ X\ R\ Z\ Y$
  shows $f\ Z = g\ Z$
⟨*proof*⟩

corollary *eq-if-is-recfunI*:
assumes *is-recfun X f is-recfun X g*
shows $f = g$
⟨*proof*⟩

corollary *is-recfun-the-recfun-if-is-recfunI*:
assumes *is-recfun X f*
shows *is-recfun X* (*the-recfun X*)
⟨*proof*⟩

corollary *fun-rel-restrict-eq-fun-rel-restrict-if-is-recfunI*:
assumes *recfuns*: *is-recfun x f is-recfun X g*
and *R x X*
shows *fun-rel-restrict f R x* = *fun-rel-restrict g R x*
⟨*proof*⟩

definition *wft-rec* ≡ *wf-rec-step the-recfun*

lemma *ex-is-recfunI*: ∃ *f*. *is-recfun X f*
⟨*proof*⟩

corollary *is-recfun-the-recfunI*: *is-recfun X* (*the-recfun X*)
⟨*proof*⟩

theorem *wft-rec-eq-wf-rec-stepI*: *wft-rec X* = *wf-rec-step* (λ-. *wft-rec*) *X*
⟨*proof*⟩

end
end
end

end

1.3.15 Well-Founded Recursion

theory *Wellfounded-Recursion*
imports
Binary-Relations-Transitive-Closure
Wellfounded-Transitive-Recursion
begin

Summary We use *wft-rec* to define well-founded recursion on non-transitive, well-founded relations. The fact that the transitive closure of a well-founded relation is itself well-founded can be used to remove the transitivity assumption of ⟦*wellfounded ?R*; *transitive ?R*⟧ ⟹ *wft-rec ?R ?step ?X* = *wf-rec-step ?R ?step* (λ-. *wft-rec ?R ?step*) *?X*.

lemma *wellfounded-trans-closure-if-wellfounded*:
  assumes *wellfounded R*
  shows *wellfounded* (*trans-closure R*)
⟨*proof*⟩

consts *wf-rec* :: *'a* ⇒ *'b* ⇒ *'c*

definition *wf-rec-rel R step* = *wft-rec* (*trans-closure R*) (λ*f*. *wf-rec-step R step* (λ-. *f*))
adhoc-overloading *wf-rec* ⇌ *wf-rec-rel*

theorem *wf-rec-eq-wf-rec-stepI*:
  assumes *wellfounded R*
  shows *wf-rec R step X* = *wf-rec-step R step* (λ-. *wf-rec R step*) *X*
⟨*proof*⟩

consts *wf-rec-on* :: *'a* ⇒ *'b* ⇒ *'c* ⇒ *'d*

definition *wf-rec-on-pred* :: (*'a* ⇒ *bool*) ⇒ (*'a* ⇒ *'a* ⇒ *bool*) ⇒ ((*'a* ⇒ *'b*) ⇒ *'a* ⇒ *'b*) ⇒ *'a* ⇒ *'b* **where**
  wf-rec-on-pred P R ≡ *wf-rec R*↑$_P$
adhoc-overloading *wf-rec-on* ⇌ *wf-rec-on-pred*

lemma *wf-rec-on-eq-wf-rec-restrict*: *wf-rec-on P R* = *wf-rec R*↑$_P$
  ⟨*proof*⟩

theorem *wf-rec-on-eq-wf-rec-step-restrictI*:
  assumes *wellfounded-on P R*
  shows *wf-rec-on P R step X* = *wf-rec-step R*↑$_P$ *step* (λ-. *wf-rec-on P R step*) *X*
⟨*proof*⟩

lemma *wf-rec-on-eq-app-fun-restrict-fun-rel-restrictI*:
  fixes *step*
  assumes *wf*: *wellfounded-on P R*
  and *P x*
  defines *f* ≡ *wf-rec-on P R step*
  shows *f x* = *step* (*fun-restrict* (*fun-rel-restrict f R x*) *P*) *x*
⟨*proof*⟩

end

1.3.16 Lattice

theory *Binary-Relations-Lattice*
  imports
    Binary-Relations-Order-Base
    HOL.Boolean-Algebras
begin

Summary Basic results about the lattice structure on binary relations.

lemma *rel-infI* [*intro*]:
assumes $R\ x\ y$
and $S\ x\ y$
shows $(R \sqcap S)\ x\ y$
⟨*proof*⟩

lemma *rel-infE* [*elim*]:
assumes $(R \sqcap S)\ x\ y$
obtains $R\ x\ y\ S\ x\ y$
⟨*proof*⟩

lemma *rel-infD*:
assumes $(R \sqcap S)\ x\ y$
shows $R\ x\ y$ **and** $S\ x\ y$
⟨*proof*⟩

lemma *in-dom-rel-infI* [*intro*]:
assumes $R\ x\ y$
and $S\ x\ y$
shows *in-dom* $(R \sqcap S)\ x$
⟨*proof*⟩

lemma *in-dom-rel-infE* [*elim*]:
assumes *in-dom* $(R \sqcap S)\ x$
obtains y **where** $R\ x\ y\ S\ x\ y$
⟨*proof*⟩

lemma *in-codom-rel-infI* [*intro*]:
assumes $R\ x\ y$
and $S\ x\ y$
shows *in-codom* $(R \sqcap S)\ y$
⟨*proof*⟩

lemma *in-codom-rel-infE* [*elim*]:
assumes *in-codom* $(R \sqcap S)\ y$
obtains x **where** $R\ x\ y\ S\ x\ y$
⟨*proof*⟩

lemma *in-field-eq-in-dom-sup-in-codom*: *in-field* $L = (\textit{in-dom}\ L \sqcup \textit{in-codom}\ L)$
⟨*proof*⟩

lemma *in-dom-rel-restrict-left-eq* [*simp*]: *in-dom* $R{\restriction}_P = (\textit{in-dom}\ R \sqcap P)$
⟨*proof*⟩

lemma *in-codom-rel-restrict-left-eq* [*simp*]: *in-codom* $R{\restriction}_P = (\textit{in-codom}\ R \sqcap P)$
⟨*proof*⟩

lemma *rel-restrict-left-restrict-left-eq* [*simp*]:

fixes $R :: 'a \Rightarrow 'b \Rightarrow bool$ **and** $P\ Q :: 'a \Rightarrow bool$
shows $R\restriction_P\restriction_Q = R\restriction_P \sqcap R\restriction_Q$
⟨*proof*⟩

lemma *rel-restrict-left-restrict-right-eq* [*simp*]:
fixes $R :: 'a \Rightarrow 'b \Rightarrow bool$ **and** $P :: 'a \Rightarrow bool$ **and** $Q :: 'b \Rightarrow bool$
shows $R\restriction_P\upharpoonright_Q = R\restriction_P \sqcap R\upharpoonright_Q$
⟨*proof*⟩

lemma *rel-restrict-right-restrict-left-eq* [*simp*]:
fixes $R :: 'a \Rightarrow 'b \Rightarrow bool$ **and** $P :: 'b \Rightarrow bool$ **and** $Q :: 'a \Rightarrow bool$
shows $R\upharpoonright_P\restriction_Q = R\upharpoonright_P \sqcap R\restriction_Q$
⟨*proof*⟩

lemma *rel-restrict-right-restrict-right-eq* [*simp*]:
fixes $R :: 'a \Rightarrow 'b \Rightarrow bool$ **and** $P\ Q :: 'b \Rightarrow bool$
shows $R\upharpoonright_P\upharpoonright_Q = R\upharpoonright_P \sqcap R\upharpoonright_Q$
⟨*proof*⟩

lemma *rel-restrict-left-sup-eq* [*simp*]:
$(R :: 'a \Rightarrow 'b \Rightarrow bool)\restriction_{((P :: 'a \Rightarrow bool) \sqcup Q)} = R\restriction_P \sqcup R\restriction_Q$
⟨*proof*⟩

lemma *rel-restrict-left-inf-eq* [*simp*]:
$(R :: 'a \Rightarrow 'b \Rightarrow bool)\restriction_{((P :: 'a \Rightarrow bool) \sqcap Q)} = R\restriction_P \sqcap R\restriction_Q$
⟨*proof*⟩

lemma *inf-rel-bimap-and-eq-restrict-left-restrict-right*:
$R \sqcap (\textit{rel-bimap}\ P\ Q\ (\wedge)) = R\restriction_P\upharpoonright_Q$
⟨*proof*⟩

end

1.3.17 Reflexive Closure

theory *Binary-Relations-Reflexive-Closure*
imports
Binary-Relations-Reflexive
Restricted-Equality
begin

consts *refl-closure-on* :: $'a \Rightarrow 'b \Rightarrow 'b$

overloading
refl-closure-on-pred ≡ *refl-closure-on* :: $('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a \Rightarrow bool)$
begin
definition *refl-closure-on-pred* $(P :: 'a \Rightarrow bool)\ (R :: 'a \Rightarrow 'a \Rightarrow bool) \equiv R \sqcup$

$(=_P)$
end

lemma *refl-closure-on-if-pred* [*intro*]:
assumes $P\ x$
shows *refl-closure-on* $P\ R\ x\ x$
⟨*proof*⟩

lemma *refl-closure-on-if-rel* [*intro*]:
assumes $(R :: 'a \Rightarrow 'a \Rightarrow bool)\ x\ y$
shows *refl-closure-on* $(P :: 'a \Rightarrow bool)\ R\ x\ y$
⟨*proof*⟩

corollary *le-refl-closure-on-self*: $(R :: 'a \Rightarrow 'a \Rightarrow bool) \leq$ *refl-closure-on* $(P :: 'a \Rightarrow bool)\ R$
⟨*proof*⟩

lemma *refl-closure-onE* [*elim*]:
assumes *refl-closure-on* $P\ R\ x\ y$
obtains $P\ x\ x = y \mid R\ x\ y$
⟨*proof*⟩

lemma *reflexive-on-refl-closure-on*:
reflexive-on $(P :: 'a \Rightarrow bool)$ (*refl-closure-on* P $(R :: 'a \Rightarrow 'a \Rightarrow bool)$)
⟨*proof*⟩

definition *refl-closure-field* $R \equiv$ *refl-closure-on* (*in-field* R) R

open-bundle *refl-closure-field-syntax*
begin
notation *refl-closure-field* (‹(-$^+$)› [*1000*])
end

lemma *refl-closure-fieldI* [*intro*]:
assumes *refl-closure-on* (*in-field* R) $R\ x\ y$
shows $R^+\ x\ y$
⟨*proof*⟩

lemma *refl-closure-fieldD* [*dest*]:
assumes $R^+\ x\ y$
shows *refl-closure-on* (*in-field* R) $R\ x\ y$
⟨*proof*⟩

lemma *refl-closure-field-eq-refl-closure-on-in-field*: $R^+ =$ *refl-closure-on* (*in-field* R) R
⟨*proof*⟩

end

theory *LBinary-Relations*
imports
Binary-Relations-Functions
Binary-Relations-Order
Binary-Relation-Properties
Wellfounded-Recursion
Wellfounded-Transitive-Recursion
Binary-Relation-Functions
Binary-Relations-Agree
Binary-Relations-Extend
Binary-Relations-Lattice
Binary-Relations-Least
Binary-Relations-Reflexive-Closure
Binary-Relations-Transitive-Closure
Dependent-Binary-Relations
Restricted-Equality
Reverse-Implies
begin

Summary Basic concepts on binary relations.

end

Inverse

theory *Functions-Inverse*
imports
Functions-Injective
Binary-Relations-Function-Evaluation
Bounded-Definite-Description
begin

consts *the-inverse-on* :: $'a \Rightarrow 'b \Rightarrow 'c$

definition *the-inverse-on-pred* $P\ f \equiv \lambda y.\ THE\ x : P.\ y = f\ x$
adhoc-overloading *the-inverse-on* $\rightleftharpoons$ *the-inverse-on-pred*

lemma *the-inverse-on-eq-if-injective-onI*:
assumes *injective-on* $P\ f$
and $y = f\ x$
and $P\ x$
shows *the-inverse-on* $P\ f\ y = x$
⟨*proof*⟩

lemma *the-inverse-on-app-eq-if-injective-onI* [*simp*]:
assumes *injective-on* $P\ f$
and $P\ x$
shows *the-inverse-on* $P\ f\ (f\ x) = x$

⟨*proof*⟩

consts *the-inverse* :: $'a \Rightarrow 'b$

definition *the-inverse-fun* ≡ *the-inverse-on* (⊤ :: $'a \Rightarrow$ *bool*)
adhoc-overloading *the-inverse* ⇌ *the-inverse-fun*

lemma *the-inverse-eq-the-inverse-on*:
the-inverse = *the-inverse-on* (⊤ :: $'a \Rightarrow$ *bool*)
⟨*proof*⟩

lemma *the-inverse-eq-the-inverse-on-uhint* [*uhint*]:
assumes $P \equiv \top :: 'a \Rightarrow$ *bool*
shows *the-inverse* :: $('a \Rightarrow 'b) \Rightarrow 'b \Rightarrow 'a \equiv$ *the-inverse-on* P
⟨*proof*⟩

lemma *the-inverse-eq-if-injectiveI*:
assumes *injective* f
and $y = f\ x$
shows *the-inverse* $f\ y = x$
⟨*proof*⟩

lemma *the-inverse-app-eq-if-injectiveI* [*simp*]:
assumes *injective* f
shows *the-inverse* $f\ (f\ x) = x$
⟨*proof*⟩

consts *inverse-on* :: $'a \Rightarrow 'b \Rightarrow 'c \Rightarrow$ *bool*

overloading
inverse-on-pred ≡ *inverse-on* :: $('a \Rightarrow$ *bool*$) \Rightarrow ('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow 'a) \Rightarrow$ *bool*
begin
definition *inverse-on-pred* $P\ f\ g \equiv \forall x : P.\ g\ (f\ x) = x$
end

lemma *inverse-onI* [*intro*]:
assumes $\bigwedge x.\ P\ x \Longrightarrow g\ (f\ x) = x$
shows *inverse-on* $P\ f\ g$
⟨*proof*⟩

lemma *inverse-onD*:
assumes *inverse-on* $P\ f\ g$
and $P\ x$
shows $g\ (f\ x) = x$
⟨*proof*⟩

lemma *inverse-onE*:
assumes *inverse-on* $P\ f\ g$
obtains $\bigwedge x.\ P\ x \Longrightarrow g\ (f\ x) = x$

⟨*proof*⟩

lemma *injective-on-if-inverse-on*:
assumes *inv*: *inverse-on* (*P* :: ′*a* ⇒ *bool*) (*f* :: ′*a* ⇒ ′*b*) (*g* :: ′*b* ⇒ ′*a*)
shows *injective-on P f*
⟨*proof*⟩

lemma *inverse-on-the-inverse-on-if-injective-on*:
fixes *P* :: ′*a* ⇒ *bool* **and** *f* :: ′*a* ⇒ ′*b*
assumes *injective-on P f*
shows *inverse-on P f* (*the-inverse-on P f*)
⟨*proof*⟩

lemma *inverse-on-has-inverse-on-the-inverse-on-if-injective-on*:
assumes *injective-on P f*
shows *inverse-on* (*has-inverse-on P f*) (*the-inverse-on P f*) *f*
⟨*proof*⟩

lemma *antimono-inverse-on*: *antimono* (*inverse-on* :: (′*a* ⇒ *bool*) ⇒ (′*a* ⇒ ′*b*) ⇒ (′*b* ⇒ ′*a*) ⇒ *bool*)
⟨*proof*⟩

lemma *inverse-on-compI*:
fixes *P* :: ′*a* ⇒ *bool* **and** *P*′ :: ′*b* ⇒ *bool*
and *f* :: ′*a* ⇒ ′*b* **and** *g* :: ′*b* ⇒ ′*a* **and** *f*′ :: ′*b* ⇒ ′*c* **and** *g*′ :: ′*c* ⇒ ′*b*
assumes (*P* ⇒ *P*′) *f*
and *inverse-on P f g*
and *inverse-on P*′ *f*′ *g*′
shows *inverse-on P* (*f*′ ∘ *f*) (*g* ∘ *g*′)
⟨*proof*⟩

consts *inverse* :: ′*a* ⇒ ′*b* ⇒ *bool*

overloading
inverse ≡ *inverse* :: (′*a* ⇒ ′*b*) ⇒ (′*b* ⇒ ′*a*) ⇒ *bool*
begin
definition (*inverse* :: (′*a* ⇒ ′*b*) ⇒ (′*b* ⇒ ′*a*) ⇒ *bool*) ≡ *inverse-on* (⊤ :: ′*a* ⇒ *bool*)
end

lemma *inverse-eq-inverse-on*:
(*inverse* :: (′*a* ⇒ ′*b*) ⇒ (′*b* ⇒ ′*a*) ⇒ *bool*) = *inverse-on* (⊤ :: ′*a* ⇒ *bool*)
⟨*proof*⟩

lemma *inverse-eq-inverse-on-uhint* [*uhint*]:
assumes *P* ≡ ⊤ :: ′*a* ⇒ *bool*
shows *inverse* :: (′*a* ⇒ ′*b*) ⇒ (′*b* ⇒ ′*a*) ⇒ *bool* ≡ *inverse-on P*
⟨*proof*⟩

lemma *inverseI* [*intro*]:
assumes $\bigwedge x.\ g\ (f\ x) = x$
shows *inverse f g*
⟨*proof*⟩

lemma *inverseD*:
assumes *inverse f g*
shows $g\ (f\ x) = x$
⟨*proof*⟩

lemma *inverseE*:
assumes *inverse f g*
obtains $\bigwedge x.\ g\ (f\ x) = x$
⟨*proof*⟩

lemma *inverse-on-if-inverse*:
fixes $P :: 'a \Rightarrow bool$ **and** $f :: 'a \Rightarrow 'b$ **and** $g :: 'b \Rightarrow 'a$
assumes *inverse f g*
shows *inverse-on P f g*
⟨*proof*⟩

lemma *injective-if-inverse*:
assumes *inverse* $(f :: 'a \Rightarrow 'b)\ (g :: 'b \Rightarrow 'a)$
shows *injective f*
⟨*proof*⟩

lemma *inverse-the-inverse-if-injective*:
assumes *injective f*
shows *inverse f* (*the-inverse f*)
⟨*proof*⟩

lemma *inverse-on-has-inverse-the-inverse-if-injective*:
assumes *injective f*
shows *inverse-on* (*has-inverse f*) (*the-inverse f*) *f*
⟨*proof*⟩

end

Bijections

theory *Functions-Bijection*
imports
Functions-Inverse
Functions-Monotone
begin

consts *bijection-on* $:: 'a \Rightarrow 'b \Rightarrow 'c \Rightarrow 'd \Rightarrow bool$

definition *bijection-on-pred* $(P :: 'a \Rightarrow bool)\ (Q :: 'b \Rightarrow bool)\ f\ g \equiv$

$(P \Rightarrow Q)\ f \wedge$
$(Q \Rightarrow P)\ g \wedge$
inverse-on $P\ f\ g \wedge$
inverse-on $Q\ g\ f$
adhoc-overloading *bijection-on* $\rightleftharpoons$ *bijection-on-pred*

context
fixes $P :: 'a \Rightarrow bool$ **and** $Q :: 'b \Rightarrow bool$ **and** $f :: 'a \Rightarrow 'b$ **and** $g :: 'b \Rightarrow 'a$
begin

lemma *bijection-onI* [*intro*]:
assumes $(P \Rightarrow Q)\ f$
and $(Q \Rightarrow P)\ g$
and *inverse-on* $P\ f\ g$
and *inverse-on* $Q\ g\ f$
shows *bijection-on* $P\ Q\ f\ g$
⟨*proof*⟩

lemma *bijection-onE* [*elim*]:
assumes *bijection-on* $P\ Q\ f\ g$
obtains $(P \Rightarrow Q)\ f\ (Q \Rightarrow P)\ g$
inverse-on $P\ f\ g$ *inverse-on* $Q\ g\ f$
⟨*proof*⟩

lemma *mono-wrt-pred-if-bijection-on-left*:
assumes *bijection-on* $P\ Q\ f\ g$
shows $(P \Rightarrow Q)\ f$
⟨*proof*⟩

lemma *mono-wrt-pred-if-bijection-on-right*:
assumes *bijection-on* $P\ Q\ f\ g$
shows $(Q \Rightarrow P)\ g$
⟨*proof*⟩

lemma *bijection-on-pred-right*:
assumes *bijection-on* $P\ Q\ f\ g$
and $P\ x$
shows $Q\ (f\ x)$
⟨*proof*⟩

lemma *bijection-on-pred-left*:
assumes *bijection-on* $P\ Q\ f\ g$
and $Q\ y$
shows $P\ (g\ y)$
⟨*proof*⟩

lemma *inverse-on-if-bijection-on-left-right*:
assumes *bijection-on* $P\ Q\ f\ g$
shows *inverse-on* $P\ f\ g$

⟨*proof*⟩

lemma *inverse-on-if-bijection-on-right-left*:
assumes *bijection-on P Q f g*
shows *inverse-on Q g f*
⟨*proof*⟩

lemma *bijection-on-left-right-eq-self*:
assumes *bijection-on P Q f g*
and *P x*
shows *g* (*f x*) = *x*
⟨*proof*⟩

lemma *bijection-on-right-left-eq-self'*:
assumes *bijection-on P Q f g*
and *Q y*
shows *f* (*g y*) = *y*
⟨*proof*⟩

end

lemma *bijection-on-has-inverse-on-the-inverse-on-if-injective-on*:
assumes *injective-on P f*
shows *bijection-on P* (*has-inverse-on P f*) *f* (*the-inverse-on P f*)
⟨*proof*⟩

context
fixes *P* :: *'a* ⇒ *bool* **and** *Q* :: *'b* ⇒ *bool* **and** *f* :: *'a* ⇒ *'b* **and** *g* :: *'b* ⇒ *'a*
begin

lemma *bijection-on-right-left-if-bijection-on-left-right*:
assumes *bijection-on P Q f g*
shows *bijection-on Q P g f*
⟨*proof*⟩

lemma *injective-on-if-bijection-on-left*:
assumes *bijection-on P Q f g*
shows *injective-on P f*
⟨*proof*⟩

lemma *injective-on-if-bijection-on-right*:
assumes *bijection-on P Q f g*
shows *injective-on Q g*
⟨*proof*⟩

end

lemma *bijection-on-compI*:
fixes *P* :: *'a* ⇒ *bool* **and** *P'* :: *'b* ⇒ *bool* **and** *Q* :: *'c* ⇒ *bool*

assumes *bijection-on P P′ f g*
and *bijection-on P′ Q f′ g′*
shows *bijection-on P Q* $(f' \circ f)$ $(g \circ g')$
⟨*proof*⟩

consts *bijection* :: $'a \Rightarrow 'b \Rightarrow$ *bool*

definition (*bijection-rel* :: $('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow 'a) \Rightarrow$ *bool*) ≡
bijection-on (⊤ :: $'a \Rightarrow$ *bool*) (⊤ :: $'b \Rightarrow$ *bool*)
adhoc-overloading *bijection* ⇌ *bijection-rel*

lemma *bijection-eq-bijection-on*:
(*bijection* :: $('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow 'a) \Rightarrow$ *bool*) = *bijection-on* (⊤ :: $'a \Rightarrow$ *bool*) (⊤ :: $'b \Rightarrow$ *bool*)
⟨*proof*⟩

lemma *bijection-eq-bijection-on-uhint* [*uhint*]:
assumes $P \equiv$ (⊤ :: $'a \Rightarrow$ *bool*)
and $Q \equiv$ (⊤ :: $'b \Rightarrow$ *bool*)
shows (*bijection* :: $('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow 'a) \Rightarrow$ *bool*) = *bijection-on P Q*
⟨*proof*⟩

context
fixes P :: $'a \Rightarrow$ *bool* **and** Q :: $'b \Rightarrow$ *bool* **and** f :: $'a \Rightarrow 'b$ **and** g :: $'b \Rightarrow 'a$
begin

lemma *bijectionI* [*intro*]:
assumes *inverse f g*
and *inverse g f*
shows *bijection f g*
⟨*proof*⟩

lemma *bijectionE* [*elim*]:
assumes *bijection f g*
obtains *inverse f g inverse g f*
⟨*proof*⟩

lemma *inverse-if-bijection-left-right*:
assumes *bijection f g*
shows *inverse f g*
⟨*proof*⟩

lemma *inverse-if-bijection-right-left*:
assumes *bijection f g*
shows *inverse g f*
⟨*proof*⟩

end

lemma *bijection-right-left-if-bijection-left-right*:
fixes $f :: 'a \Rightarrow 'b$ **and** $g :: 'b \Rightarrow 'a$
assumes *bijection f g*
shows *bijection g f*
⟨*proof*⟩

Instantiations **lemma** *bijection-on-self-id*: *bijection-on* $(P :: 'a \Rightarrow bool)$ *P id id*
⟨*proof*⟩

end

Surjective

theory *Functions-Surjective*
imports
Functions-Base
begin

consts *surjective-at* $:: 'a \Rightarrow 'b \Rightarrow bool$

overloading
surjective-at-pred $\equiv$ *surjective-at* $:: ('a \Rightarrow bool) \Rightarrow ('b \Rightarrow 'a) \Rightarrow bool$
begin
definition *surjective-at-pred* $(P :: 'a \Rightarrow bool)$ $(f :: 'b \Rightarrow 'a) \equiv \forall\, y : P.$ *has-inverse f y*
end

lemma *surjective-atI* [*intro*]:
assumes $\bigwedge y.$ *P y* $\Longrightarrow$ *has-inverse f y*
shows *surjective-at P f*
⟨*proof*⟩

lemma *surjective-atE* [*elim*]:
assumes *surjective-at P f*
and *P y*
obtains *x* **where** $y = f\, x$
⟨*proof*⟩

lemma *has-inverse-if-pred-if-surjective-at*:
assumes *surjective-at P f*
and *P y*
shows *has-inverse f y*
⟨*proof*⟩

consts *surjective* $:: 'a \Rightarrow bool$

overloading
  surjective ≡ *surjective* :: (*'b* ⇒ *'a*) ⇒ *bool*
begin
  definition (*surjective* :: (*'b* ⇒ *'a*) ⇒ *bool*) ≡ *surjective-at* (⊤ :: *'a* ⇒ *bool*)
end

lemma *surjective-eq-surjective-at*:
  (*surjective* :: (*'b* ⇒ *'a*) ⇒ *bool*) = *surjective-at* (⊤ :: *'a* ⇒ *bool*)
  ⟨*proof*⟩

lemma *surjective-eq-surjective-at-uhint* [*uhint*]:
  assumes *P* ≡ ⊤ :: *'a* ⇒ *bool*
  shows *surjective* :: (*'b* ⇒ *'a*) ⇒ *bool* ≡ *surjective-at P*
  ⟨*proof*⟩

lemma *surjectiveI* [*intro*]:
  assumes ⋀*y*. *has-inverse f y*
  shows *surjective f*
  ⟨*proof*⟩

lemma *surjectiveE*:
  assumes *surjective f*
  obtains ⋀*y*. *has-inverse f y*
  ⟨*proof*⟩

lemma *surjective-at-if-surjective*:
  fixes *P* :: *'a* ⇒ *bool* **and** *f* :: *'b* ⇒ *'a*
  assumes *surjective f*
  shows *surjective-at P f*
  ⟨*proof*⟩

end

1.3.18 Basic Properties

theory *Function-Properties*
  imports
    Functions-Bijection
    Functions-Injective
    Functions-Inverse
    Functions-Monotone
    Functions-Surjective
begin

Summary Basic properties on functions.

end

theory *LFunctions*
  imports

Function-Properties
Function-Relators
Functions-Restrict
begin

Summary Basic concepts on functions.

end

1.3.19 Functions On Orders

Basics

theory *Order-Functions-Base*
imports
Functions-Monotone
Binary-Relations-Antisymmetric
Binary-Relations-Symmetric
Preorders
begin

Bi-Relation **consts** *bi-related* :: $'a \Rightarrow 'b \Rightarrow 'b \Rightarrow bool$

overloading
bi-related $\equiv$ *bi-related* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$
begin
definition *bi-related* $(R :: 'a \Rightarrow 'a \Rightarrow bool)\ x\ y \equiv R\ x\ y \wedge R\ y\ x$
end

open-bundle *bi-related-syntax*
begin
syntax
-bi-related :: $'a \Rightarrow 'b \Rightarrow 'a \Rightarrow bool$ $(\langle(\text{-})\ \equiv_{(\text{-})}\ (\text{-})\rangle\ [51,51,51]\ 50)$
notation *bi-related* $(\langle'(\equiv(\_)')\rangle)$
end

syntax-consts
-bi-related $\rightleftharpoons$ *bi-related*
translations
$x \equiv_R y \rightleftharpoons CONST\ bi\text{-}related\ R\ x\ y$

lemma *bi-relatedI* [*intro*]:
assumes $R\ x\ y$
and $R\ y\ x$
shows $x \equiv_R y$
⟨*proof*⟩

lemma *bi-relatedE* [*elim*]:
assumes $x \equiv_R y$

obtains $R\ x\ y \mid R\ y\ x$
⟨*proof*⟩

context
fixes $P :: 'a \Rightarrow bool$ **and** $R\ S :: 'a \Rightarrow 'a \Rightarrow bool$ **and** $x\ y :: 'a$
begin

lemma *symmetric-bi-related* [*iff*]: *symmetric* $((\equiv_R) :: 'a \Rightarrow 'a \Rightarrow bool)$
⟨*proof*⟩

lemma *reflexive-bi-related-if-reflexive* [*intro*]:
assumes *reflexive* R
shows *reflexive* $((\equiv_R) :: 'a \Rightarrow 'a \Rightarrow bool)$
⟨*proof*⟩

lemma *transitive-bi-related-if-transitive* [*intro*]:
assumes *transitive* R
shows *transitive* $((\equiv_R) :: 'a \Rightarrow 'a \Rightarrow bool)$
⟨*proof*⟩

lemma *mono-bi-related*: *mono* $(bi\text{-}related :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a \Rightarrow 'a \Rightarrow bool)$
⟨*proof*⟩

lemma *bi-related-if-le-rel-if-bi-related*:
assumes $x \equiv_R y$
and $R \leq S$
shows $x \equiv_S y$
⟨*proof*⟩

lemma *eq-if-bi-related-if-antisymmetric-on*:
assumes *antisymmetric-on* $P\ R$
and $x \equiv_R y$
and $P\ x\ P\ y$
shows $x = y$
⟨*proof*⟩

lemma *eq-if-bi-related-if-in-field-le-if-antisymmetric-on*:
assumes *antisymmetric-on* $P\ R$
and *in-field* $R \leq P$
and $x \equiv_R y$
shows $x = y$
⟨*proof*⟩

lemma *bi-related-if-all-rel-iff-if-reflexive-on*:
assumes *reflexive-on* $P\ R$
and $\bigwedge z.\ P\ z \Longrightarrow R\ x\ z \longleftrightarrow R\ y\ z$
and $P\ x\ P\ y$
shows $x \equiv_R y$
⟨*proof*⟩

lemma *bi-related-if-all-rel-iff-if-reflexive-on'*:
assumes *reflexive-on P R*
and $\bigwedge z.\ P\ z \Longrightarrow R\ z\ x \longleftrightarrow R\ z\ y$
and $P\ x\ P\ y$
shows $x \equiv_R y$
⟨*proof*⟩

corollary *eq-if-all-rel-iff-if-antisymmetric-on-if-reflexive-on*:
assumes *reflexive-on P R* **and** *antisymmetric-on P R*
and $\bigwedge z.\ P\ z \Longrightarrow R\ x\ z \longleftrightarrow R\ y\ z$
and $P\ x\ P\ y$
shows $x = y$
⟨*proof*⟩

corollary *eq-if-all-rel-iff-if-antisymmetric-on-if-reflexive-on'*:
assumes *reflexive-on P R* **and** *antisymmetric-on P R*
and $\bigwedge z.\ P\ z \Longrightarrow R\ z\ x \longleftrightarrow R\ z\ y$
and $P\ x\ P\ y$
shows $x = y$
⟨*proof*⟩

end

lemma *bi-related-le-eq-if-antisymmetric-on-in-field*:
assumes *antisymmetric-on* (*in-field R*) ($R :: 'a \Rightarrow 'a \Rightarrow bool$)
shows $((\equiv_R) :: 'a \Rightarrow 'a \Rightarrow bool) \leq (=)$
⟨*proof*⟩

Inflationary **consts** *inflationary-on* :: $'a \Rightarrow 'b \Rightarrow 'c \Rightarrow bool$

overloading
inflationary-on-pred ≡ *inflationary-on* :: $('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool$
begin

Often also called "extensive".

definition *inflationary-on-pred P* ($R :: 'a \Rightarrow 'b \Rightarrow bool$) ($f :: 'a \Rightarrow 'b$) ≡ $\forall x : P.\ R\ x\ (f\ x)$
end

lemma *inflationary-onI* [*intro*]:
assumes $\bigwedge x.\ P\ x \Longrightarrow R\ x\ (f\ x)$
shows *inflationary-on P R f*
⟨*proof*⟩

lemma *inflationary-onD* [*dest*]:
assumes *inflationary-on P R f*
and *P x*

shows $R\ x\ (f\ x)$
⟨*proof*⟩

lemma *inflationary-on-eq-dep-mono-wrt-pred*: *inflationary-on* = *dep-mono-wrt-pred*
⟨*proof*⟩

lemma *inflationary-on-if-le-rel-if-inflationary-on*:
assumes *inflationary-on* $P\ R\ f$
and $\bigwedge x.\ P\ x \Longrightarrow R\ x\ (f\ x) \Longrightarrow R'\ x\ (f\ x)$
shows *inflationary-on* $P\ R'\ f$
⟨*proof*⟩

lemma *mono-inflationary-on-rel*:
$((\geq) \Rightarrow (\leq) \Rrightarrow (\leq))$ (*inflationary-on* :: $('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool$)
⟨*proof*⟩

context
fixes $P\ P' :: 'a \Rightarrow bool$ **and** $R :: 'a \Rightarrow 'b \Rightarrow bool$ **and** $f :: 'a \Rightarrow 'b$
begin

lemma *inflationary-on-if-le-pred-if-inflationary-on*:
assumes *inflationary-on* $P\ R\ f$
and $P' \leq P$
shows *inflationary-on* $P'\ R\ f$
⟨*proof*⟩

lemma *le-in-dom-if-inflationary-on*:
assumes *inflationary-on* $P\ R\ f$
shows $P \leq$ *in-dom* R
⟨*proof*⟩
end

lemma *inflationary-on-sup-eq* [*simp*]:
(*inflationary-on* :: $('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool$) $(P \sqcup Q)$
= *inflationary-on* $P \sqcap$ *inflationary-on* Q
⟨*proof*⟩

consts *inflationary* :: $'a \Rightarrow 'b \Rightarrow bool$

overloading
inflationary ≡ *inflationary* :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool$
begin
definition (*inflationary* :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool$) ≡
inflationary-on ($\top$ :: $'a \Rightarrow bool$)
end

lemma *inflationary-eq-inflationary-on*:
(*inflationary* :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ (*'a* ⇒ *'b*) ⇒ *bool*) = *inflationary-on* (⊤ :: *'a* ⇒ *bool*)
⟨*proof*⟩

lemma *inflationary-eq-inflationary-on-uhint* [*uhint*]:
assumes *P* ≡ ⊤ :: *'a* ⇒ *bool*
shows *inflationary* :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ (*'a* ⇒ *'b*) ⇒ *bool* ≡ *inflationary-on P*
⟨*proof*⟩

lemma *inflationaryI* [*intro*]:
assumes ⋀*x*. *R x* (*f x*)
shows *inflationary R f*
⟨*proof*⟩

lemma *inflationaryD*:
assumes *inflationary R f*
shows *R x* (*f x*)
⟨*proof*⟩

lemma *inflationary-on-if-inflationary*:
fixes *P* :: *'a* ⇒ *bool* **and** *R* :: *'a* ⇒ *'b* ⇒ *bool* **and** *f* :: *'a* ⇒ *'b*
assumes *inflationary R f*
shows *inflationary-on P R f*
⟨*proof*⟩

lemma *inflationary-eq-dep-mono-wrt-pred*: *inflationary* = *dep-mono-wrt-pred* ⊤
⟨*proof*⟩

Deflationary **consts** *deflationary-on* :: *'a* ⇒ *'b* ⇒ *'c* ⇒ *bool*

overloading
deflationary-on-pred ≡ *deflationary-on* :: (*'a* ⇒ *bool*) ⇒ (*'b* ⇒ *'a* ⇒ *bool*) ⇒ (*'a* ⇒ *'b*) ⇒ *bool*
begin
definition *deflationary-on-pred* (*P* :: *'a* ⇒ *bool*) (*R* :: *'b* ⇒ *'a* ⇒ *bool*) :: (*'a* ⇒ *'b*) ⇒ *bool* ≡
inflationary-on P R^{-1}
end

context
fixes *P* :: *'a* ⇒ *bool* **and** *R* :: *'b* ⇒ *'a* ⇒ *bool* **and** *f* :: *'a* ⇒ *'b*
begin

lemma *deflationary-on-eq-inflationary-on-rel-inv*:
(*deflationary-on P R* :: (*'a* ⇒ *'b*) ⇒ *bool*) = *inflationary-on P* R^{-1}
⟨*proof*⟩

declare *deflationary-on-eq-inflationary-on-rel-inv*[*symmetric*, *simp*]

lemma *deflationary-onI* [*intro*]:
assumes $\bigwedge x.\ P\ x \Longrightarrow R\ (f\ x)\ x$
shows *deflationary-on P R f*
⟨*proof*⟩

lemma *deflationary-onD* [*dest*]:
assumes *deflationary-on P R f*
and $P\ x$
shows $R\ (f\ x)\ x$
⟨*proof*⟩

end

context
fixes $P\ P' :: 'a \Rightarrow bool$ **and** $R :: 'b \Rightarrow 'a \Rightarrow bool$ **and** $f :: 'a \Rightarrow 'b$
begin

corollary *deflationary-on-rel-inv-eq-inflationary-on* [*simp*]:
$(deflationary\text{-}on\ P\ (S :: 'a \Rightarrow 'b \Rightarrow bool)^{-1} :: ('a \Rightarrow 'b) \Rightarrow bool) = inflationary\text{-}on\ P\ S$
⟨*proof*⟩

lemma *deflationary-on-eq-dep-mono-wrt-pred-rel-inv*:
$(deflationary\text{-}on\ P\ R :: ('a \Rightarrow 'b) \Rightarrow bool) = ((x : P) \Rightarrow R^{-1}\ x)$
⟨*proof*⟩

lemma *deflationary-on-if-le-rel-if-deflationary-on*:
assumes *deflationary-on P R f*
and $\bigwedge x.\ P\ x \Longrightarrow R\ (f\ x)\ x \Longrightarrow R'\ (f\ x)\ x$
shows *deflationary-on P R' f*
⟨*proof*⟩

lemma *mono-deflationary-on*:
$((\geq) \Rightarrow (\leq) \Rrightarrow (\leq))\ (deflationary\text{-}on :: ('a \Rightarrow bool) \Rightarrow ('b \Rightarrow 'a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool)$
⟨*proof*⟩

lemma *deflationary-on-if-le-pred-if-deflationary-on*:
assumes *deflationary-on P R f*
and $P' \leq P$
shows *deflationary-on P' R f*
⟨*proof*⟩

lemma *le-in-dom-if-deflationary-on*:
assumes *deflationary-on P R f*
shows $P \leq in\text{-}codom\ R$
⟨*proof*⟩

lemma *deflationary-on-sup-eq* [*simp*]:
(*deflationary-on* :: ('*a* ⇒ *bool*) ⇒ ('*b* ⇒ '*a* ⇒ *bool*) ⇒ ('*a* ⇒ '*b*) ⇒ *bool*) (*P* ⊔ *Q*)
= *deflationary-on P* ⊓ *deflationary-on Q*
⟨*proof*⟩

end

consts *deflationary* :: '*a* ⇒ '*b* ⇒ *bool*

overloading
deflationary ≡ *deflationary* :: ('*b* ⇒ '*a* ⇒ *bool*) ⇒ ('*a* ⇒ '*b*) ⇒ *bool*
begin
definition (*deflationary* :: ('*b* ⇒ '*a* ⇒ *bool*) ⇒ ('*a* ⇒ '*b*) ⇒ *bool*) ≡
deflationary-on (⊤ :: '*a* ⇒ *bool*)
end

lemma *deflationary-eq-deflationary-on*:
(*deflationary* :: ('*b* ⇒ '*a* ⇒ *bool*) ⇒ ('*a* ⇒ '*b*) ⇒ *bool*) = *deflationary-on* (⊤ :: '*a* ⇒ *bool*)
⟨*proof*⟩

lemma *deflationary-eq-deflationary-on-uhint* [*uhint*]:
assumes *P* ≡ ⊤ :: '*a* ⇒ *bool*
shows *deflationary* :: ('*b* ⇒ '*a* ⇒ *bool*) ⇒ ('*a* ⇒ '*b*) ⇒ *bool* ≡ *deflationary-on P*
⟨*proof*⟩

lemma *deflationaryI* [*intro*]:
assumes ⋀*x*. *R* (*f x*) *x*
shows *deflationary R f*
⟨*proof*⟩

lemma *deflationaryD*:
assumes *deflationary R f*
shows *R* (*f x*) *x*
⟨*proof*⟩

lemma *deflationary-on-if-deflationary*:
fixes *P* :: '*a* ⇒ *bool* **and** *R* :: '*b* ⇒ '*a* ⇒ *bool* **and** *f* :: '*a* ⇒ '*b*
assumes *deflationary R f*
shows *deflationary-on P R f*
⟨*proof*⟩

lemma *deflationary-eq-dep-mono-wrt-pred-rel-inv*:
deflationary R = *dep-mono-wrt-pred* ⊤ R^{-1}
⟨*proof*⟩

Relational Equivalence **definition** *rel-equivalence-on* $\equiv$ *inflationary-on* $\sqcap$ *deflationary-on*

lemma *rel-equivalence-on-eq*:
rel-equivalence-on $=$ *inflationary-on* $\sqcap$ *deflationary-on*
⟨*proof*⟩

lemma *rel-equivalence-onI* [*intro*]:
assumes *inflationary-on* P R f
and *deflationary-on* P R f
shows *rel-equivalence-on* P R f
⟨*proof*⟩

lemma *rel-equivalence-onE* [*elim*]:
assumes *rel-equivalence-on* P R f
obtains *inflationary-on* P R f *deflationary-on* P R f
⟨*proof*⟩

lemma *rel-equivalence-on-eq-dep-mono-wrt-pred-inf*:
rel-equivalence-on P R $=$ *dep-mono-wrt-pred* P $(R \sqcap R^{-1})$
⟨*proof*⟩

context
fixes P P' :: $'a \Rightarrow$ *bool* **and** R :: $'a \Rightarrow 'a \Rightarrow$ *bool* **and** f :: $'a \Rightarrow 'a$
begin

lemma *bi-related-if-rel-equivalence-on*:
assumes *rel-equivalence-on* P R f
and P x
shows $x \equiv_R f\ x$
⟨*proof*⟩

lemma *rel-equivalence-on-if-all-bi-related*:
assumes $\bigwedge x.\ P\ x \Longrightarrow x \equiv_R f\ x$
shows *rel-equivalence-on* P R f
⟨*proof*⟩

corollary *rel-equivalence-on-iff-all-bi-related*:
rel-equivalence-on P R $f \longleftrightarrow (\forall x.\ P\ x \longrightarrow x \equiv_R f\ x)$
⟨*proof*⟩

lemma *rel-equivalence-onD* [*dest*]:
assumes *rel-equivalence-on* P R f
and P x
shows $x \equiv_R f\ x$
⟨*proof*⟩

lemma *rel-equivalence-on-rel-inv-eq-rel-equivalence-on* [*simp*]:
(*rel-equivalence-on* P R^{-1} :: $('a \Rightarrow 'a) \Rightarrow$ *bool*) $=$ *rel-equivalence-on* P R

⟨*proof*⟩

lemma *mono-rel-equivalence-on*:
((≥) ⇒ (≤) ⇛ (≤)) (*rel-equivalence-on* :: ('*a* ⇒ *bool*) ⇒ ('*a* ⇒ '*a* ⇒ *bool*) ⇒ ('*a* ⇒ '*a*) ⇒ *bool*)
⟨*proof*⟩

lemma *rel-equivalence-on-if-le-pred-if-rel-equivalence-on*:
assumes *rel-equivalence-on P R f*
and $P' \leq P$
shows *rel-equivalence-on P' R f*
⟨*proof*⟩

lemma *rel-equivalence-on-sup-eq* [*simp*]:
(*rel-equivalence-on* :: ('*a* ⇒ *bool*) ⇒ ('*a* ⇒ '*a* ⇒ *bool*) ⇒ ('*a* ⇒ '*a*) ⇒ *bool*) (*P* ⊔ *Q*)
= *rel-equivalence-on P* ⊓ *rel-equivalence-on Q*
⟨*proof*⟩

lemma *in-codom-eq-in-dom-if-rel-equivalence-on-in-field*:
assumes *rel-equivalence-on* (*in-field R*) *R f*
shows *in-codom R* = *in-dom R*
⟨*proof*⟩

lemma *reflexive-on-if-transitive-on-if-mon-wrt-pred-if-rel-equivalence-on*:
assumes *rel-equivalence-on P R f*
and (*P* ⇒ *P*) *f*
and *transitive-on P R*
shows *reflexive-on P R*
⟨*proof*⟩

lemma *inflationary-on-eq-rel-equivalence-on-if-symmetric*:
assumes *symmetric R*
shows (*inflationary-on P R* :: ('*a* ⇒ '*a*) ⇒ *bool*) = *rel-equivalence-on P R*
⟨*proof*⟩

lemma *deflationary-on-eq-rel-equivalence-on-if-symmetric*:
assumes *symmetric R*
shows (*deflationary-on P R* :: ('*a* ⇒ '*a*) ⇒ *bool*) = *rel-equivalence-on P R*
⟨*proof*⟩

end

consts *rel-equivalence* :: '*a* ⇒ '*b* ⇒ *bool*

overloading
rel-equivalence ≡ *rel-equivalence* :: ('*a* ⇒ '*a* ⇒ *bool*) ⇒ ('*a* ⇒ '*a*) ⇒ *bool*
begin

definition (*rel-equivalence* :: ('*a* ⇒ '*a* ⇒ *bool*) ⇒ ('*a* ⇒ '*a*) ⇒ *bool*) ≡
rel-equivalence-on (⊤ :: '*a* ⇒ *bool*)
end

lemma *rel-equivalence-eq-rel-equivalence-on*:
(*rel-equivalence* :: ('*a* ⇒ '*a* ⇒ *bool*) ⇒ ('*a* ⇒ '*a*) ⇒ *bool*) = *rel-equivalence-on* (⊤ :: '*a* ⇒ *bool*)
⟨*proof*⟩

lemma *rel-equivalence-eq-rel-equivalence-on-uhint* [*uhint*]:
assumes *P* ≡ ⊤ :: '*a* ⇒ *bool*
shows *rel-equivalence* :: ('*a* ⇒ '*a* ⇒ *bool*) ⇒ ('*a* ⇒ '*a*) ⇒ *bool* ≡ *rel-equivalence-on P*
⟨*proof*⟩

context
fixes *P* :: '*a* ⇒ *bool* **and** *R* :: '*a* ⇒ '*a* ⇒ *bool* **and** *f* :: '*a* ⇒ '*a*
begin

lemma *rel-equivalenceI* [*intro*]:
assumes *inflationary R f*
and *deflationary R f*
shows *rel-equivalence R f*
⟨*proof*⟩

lemma *rel-equivalenceE* [*elim*]:
assumes *rel-equivalence R f*
obtains *inflationary R f deflationary R f*
⟨*proof*⟩

lemma *inflationary-if-rel-equivalence*:
assumes *rel-equivalence R f*
shows *inflationary R f*
⟨*proof*⟩

lemma *deflationary-if-rel-equivalence*:
assumes *rel-equivalence R f*
shows *deflationary R f*
⟨*proof*⟩

lemma *rel-equivalence-on-if-rel-equivalence*:
assumes *rel-equivalence R f*
shows *rel-equivalence-on P R f*
⟨*proof*⟩

lemma *bi-related-if-rel-equivalence*:
assumes *rel-equivalence R f*
shows *x* ≡$_R$ *f x*
⟨*proof*⟩

lemma *rel-equivalence-if-all-bi-related*:
assumes $\bigwedge x.\ x \equiv_R f\ x$
shows *rel-equivalence* R f
⟨*proof*⟩

lemma *rel-equivalenceD*:
assumes *rel-equivalence* R f
shows $R\ x\ (f\ x)\ R\ (f\ x)\ x$
⟨*proof*⟩

lemma *reflexive-on-in-field-if-transitive-if-rel-equivalence-on*:
assumes *rel-equivalence-on* (*in-field* R) R f
and *transitive* R
shows *reflexive-on* (*in-field* R) R
⟨*proof*⟩

corollary *preorder-on-in-field-if-transitive-if-rel-equivalence-on*:
assumes *rel-equivalence-on* (*in-field* R) R f
and *transitive* R
shows *preorder-on* (*in-field* R) R
⟨*proof*⟩

end

end

1.3.20 Order Functors

Basic Setup and Results

theory *Order-Functors-Base*
imports
Functions-Inverse
Order-Functions-Base
begin

In the following, we do not add any assumptions to our locales but rather add them as needed to the theorem statements. This allows consumers to state preciser results; particularly, the development of Transport depends on this setup.

locale *orders* =
fixes $L :: 'a \Rightarrow 'b \Rightarrow bool$
and $R :: 'c \Rightarrow 'd \Rightarrow bool$
begin

notation L (**infix** $\langle \leq_L \rangle$ *50*)
notation R (**infix** $\langle \leq_R \rangle$ *50*)

We call $(\leq_L)$ the *left relation* and $(\leq_R)$ the *right relation*.

abbreviation (*input*) *ge-left* $\equiv (\leq_L)^{-1}$
notation *ge-left* (**infix** $\langle\geq_L\rangle$ *50*)

abbreviation (*input*) *ge-right* $\equiv (\leq_R)^{-1}$
notation *ge-right* (**infix** $\langle\geq_R\rangle$ *50*)

end

Homogeneous orders

locale *hom-orders* = *orders L R*
  for $L :: 'a \Rightarrow 'a \Rightarrow bool$
  and $R :: 'b \Rightarrow 'b \Rightarrow bool$

locale *order-functor* = *hom-orders L R*
  for $L :: 'a \Rightarrow 'a \Rightarrow bool$
  and $R :: 'b \Rightarrow 'b \Rightarrow bool$
  and $l :: 'a \Rightarrow 'b$
begin

lemma *left-right-rel-left-self-if-reflexive-on-left-if-mono-left*:
  assumes $((\leq_L) \Rightarrow (\leq_R))\ l$
  and *reflexive-on* $P\ (\leq_L)$
  and $P\ x$
  shows $l\ x \leq_R l\ x$
  ⟨*proof*⟩

lemma *left-right-rel-left-self-if-reflexive-on-in-dom-right-if-mono-left*:
  assumes $((\leq_L) \Rightarrow (\leq_R))\ l$
  and *reflexive-on* (*in-dom* $(\leq_R)$) $(\leq_R)$
  and *in-dom* $(\leq_L)\ x$
  shows $l\ x \leq_R l\ x$
  ⟨*proof*⟩

lemma *left-right-rel-left-self-if-reflexive-on-in-codom-right-if-mono-left*:
  assumes $((\leq_L) \Rightarrow (\leq_R))\ l$
  and *reflexive-on* (*in-codom* $(\leq_R)$) $(\leq_R)$
  and *in-codom* $(\leq_L)\ x$
  shows $l\ x \leq_R l\ x$
  ⟨*proof*⟩

lemma *left-right-rel-left-self-if-reflexive-on-in-field-right-if-mono-left*:
  assumes $((\leq_L) \Rightarrow (\leq_R))\ l$
  and *reflexive-on* (*in-field* $(\leq_R)$) $(\leq_R)$
  and *in-field* $(\leq_L)\ x$
  shows $l\ x \leq_R l\ x$
  ⟨*proof*⟩

lemma *mono-wrt-rel-left-if-reflexive-on-if-le-eq-if-mono-wrt-in-field*:
  assumes (*in-field* $(\leq_L) \Rightarrow P$) l

and $(\leq_L) \leq (=)$
and *reflexive-on* P $(\leq_R)$
shows $((\leq_L) \Rightarrow (\leq_R))$ l
⟨*proof*⟩

end

locale *order-functors* = *order-functor* L R l + *flip-of* : *order-functor* R L r
for L R l r
begin

We call the composition $r \circ l$ the *unit* and the term $l \circ r$ the *counit* of the order functors pair. This is terminology is borrowed from category theory - the functors are an *adjoint*.

definition *unit* $\equiv r \circ l$

notation *unit* (‹η›)

lemma *unit-eq-comp*: $\eta = r \circ l$ ⟨*proof*⟩

lemma *unit-eq* [*simp*]: $\eta\ x = r\ (l\ x)$ ⟨*proof*⟩

context
begin

Note that by flipping the roles of the left and rights functors, we obtain a flipped interpretation of *order-functors*. In many cases, this allows us to obtain symmetric definitions and theorems for free. As such, in many cases, we do we do not explicitly state those free results but users can obtain them as needed by creating said flipped interpretation.

interpretation *flip* : *order-functors* R L r l ⟨*proof*⟩

definition *counit* $\equiv$ *flip.unit*

notation *counit* (‹ε›)

lemma *counit-eq-comp*: $\varepsilon = l \circ r$ ⟨*proof*⟩

lemma *counit-eq* [*simp*]: $\varepsilon\ x = l\ (r\ x)$ ⟨*proof*⟩

end

context
begin

interpretation *flip* : *order-functors* R L r l ⟨*proof*⟩

lemma *flip-counit-eq-unit*: *flip.counit* $= \eta$

⟨*proof*⟩

lemma *flip-unit-eq-counit*: *flip.unit* = ε
⟨*proof*⟩

lemma *inflationary-on-unit-if-left-rel-right-if-left-right-relI*:
assumes $((\leq_L) \Rightarrow (\leq_R))\ l$
and *reflexive-on* $P\ (\leq_L)$
and $\bigwedge x\ y.\ P\ x \Longrightarrow l\ x \leq_R y \Longrightarrow x \leq_L r\ y$
shows *inflationary-on* $P\ (\leq_L)\ \eta$
⟨*proof*⟩

lemma *deflationary-on-unit-if-right-left-rel-if-right-rel-leftI*:
assumes $((\leq_L) \Rightarrow (\leq_R))\ l$
and *reflexive-on* $P\ (\leq_L)$
and $\bigwedge x\ y.\ P\ x \Longrightarrow y \leq_R l\ x \Longrightarrow r\ y \leq_L x$
shows *deflationary-on* $P\ (\leq_L)\ \eta$
⟨*proof*⟩

context
fixes $P :: 'a \Rightarrow bool$
begin

lemma *rel-equivalence-on-unit-iff-inflationary-on-if-inverse-on*:
assumes *inverse-on* $P\ l\ r$
shows *rel-equivalence-on* $P\ (\leq_L)\ \eta \longleftrightarrow$ *inflationary-on* $P\ (\leq_L)\ \eta$
⟨*proof*⟩

lemma *reflexive-on-left-if-inflationary-on-unit-if-inverse-on*:
assumes *inverse-on* $P\ l\ r$
and *inflationary-on* $P\ (\leq_L)\ \eta$
shows *reflexive-on* $P\ (\leq_L)$
⟨*proof*⟩

lemma *rel-equivalence-on-unit-if-reflexive-on-if-inverse-on*:
assumes *inverse-on* $P\ l\ r$
and *reflexive-on* $P\ (\leq_L)$
shows *rel-equivalence-on* $P\ (\leq_L)\ \eta$
⟨*proof*⟩

end

corollary *rel-equivalence-on-unit-iff-reflexive-on-if-inverse-on*:
fixes $P :: 'a \Rightarrow bool$
assumes *inverse-on* $P\ l\ r$
shows *rel-equivalence-on* $P\ (\leq_L)\ \eta \longleftrightarrow$ *reflexive-on* $P\ (\leq_L)$
⟨*proof*⟩

end

Here is an example of a free theorem.

notepad
begin
⟨*proof*⟩
end

end

end

1.4 Galois

1.4.1 Basic Abbreviations

theory *Galois-Base*
imports
Order-Functors-Base
begin

locale *galois* = *order-functors*
begin

The locale *galois* serves to define concepts that ultimately lead to the definition of Galois connections and Galois equivalences. Galois connections and equivalences are special cases of adjoints and adjoint equivalences, respectively, known from category theory. As such, in what follows, we sometimes borrow vocabulary from category theory to highlight this connection.

A *Galois connection* between two relations $(\leq_L)$ and $(\leq_R)$ consists of two monotone functions (i.e. order functors) l and r such that $(x \leq_L r\ y) = (l\ x \leq_R y)$. We call this the *Galois property*. l is called the *left adjoint* and r the *right adjoint*. We call $(\leq_L)$ the *left relation* and $(\leq_R)$ the *right relation*. By composing the adjoints, we obtain the unit η and counit ε of the Galois connection.

end

end

1.4.2 Basics For Relator For Galois Connections

theory *Galois-Relator-Base*
imports
Galois-Base
begin

locale *galois-rel* = *orders L R*
for *L* :: $'a \Rightarrow 'b \Rightarrow$ *bool*

and $R :: 'c \Rightarrow 'd \Rightarrow bool$
and $r :: 'd \Rightarrow 'b$
begin

Morally speaking, the Galois relator characterises when two terms x and y are "similar".

definition *Galois* $x\ y \equiv$ *in-codom* $(\leq_R)\ y \wedge x \leq_L r\ y$

abbreviation *left-Galois* $\equiv$ *Galois*
notation *left-Galois* (**infix** $\langle {}_L\lessapprox\rangle$ *50*)

abbreviation (*input*) *ge-Galois-left* $\equiv ({}_L\lessapprox)^{-1}$
notation *ge-Galois-left* (**infix** $\langle \gtrapprox_L\rangle$ *50*)

Here we only introduced the (left) Galois relator $({}_L\lessapprox)$. All other variants can be introduced by considering suitable flipped and inversed interpretations (see `Half_Galois_Property.thy`).

lemma *left-GaloisI* [*intro*]:
assumes *in-codom* $(\leq_R)\ y$
and $x \leq_L r\ y$
shows $x\ {}_L\lessapprox\ y$
⟨*proof*⟩

lemma *left-GaloisE* [*elim*]:
assumes $x\ {}_L\lessapprox\ y$
obtains *in-codom* $(\leq_R)\ y\ x \leq_L r\ y$
⟨*proof*⟩

corollary *in-dom-left-if-left-Galois*:
assumes $x\ {}_L\lessapprox\ y$
shows *in-dom* $(\leq_L)\ x$
⟨*proof*⟩

corollary *left-Galois-iff-in-codom-and-left-rel-right*:
$x\ {}_L\lessapprox\ y \longleftrightarrow$ *in-codom* $(\leq_R)\ y \wedge x \leq_L r\ y$
⟨*proof*⟩

lemma *left-Galois-restrict-left-eq-left-Galois-left-restrict-left*:
$({}_L\lessapprox)\restriction_P :: 'a \Rightarrow bool = galois\text{-}rel.Galois\ (\leq_L)\restriction_P\ (\leq_R)\ r$
⟨*proof*⟩

lemma *left-Galois-restrict-right-eq-left-Galois-right-restrict-right*:
$({}_L\lessapprox)\downharpoonright_P :: 'd \Rightarrow bool = galois\text{-}rel.Galois\ (\leq_L)\ (\leq_R)\downharpoonright_P\ r$
⟨*proof*⟩

end

end

Equivalences

theory *Order-Equivalences*
  imports
    Order-Functors-Base
    Partial-Equivalence-Relations
    Preorders
begin

context *order-functors*
begin

definition *order-equivalence* $\equiv$
  $((\leq_L) \Rightarrow (\leq_R))\ l \wedge$
  $((\leq_R) \Rightarrow (\leq_L))\ r \wedge$
  rel-equivalence-on (*in-field* $(\leq_L)$) $(\leq_L)\ \eta \wedge$
  rel-equivalence-on (*in-field* $(\leq_R)$) $(\leq_R)\ \varepsilon$

notation *order-functors.order-equivalence* (**infix** $\langle\equiv_o\rangle$ *50*)

lemma *order-equivalenceI* [*intro*]:
  assumes $((\leq_L) \Rightarrow (\leq_R))\ l$
  and $((\leq_R) \Rightarrow (\leq_L))\ r$
  and *rel-equivalence-on* (*in-field* $(\leq_L)$) $(\leq_L)\ \eta$
  and *rel-equivalence-on* (*in-field* $(\leq_R)$) $(\leq_R)\ \varepsilon$
  shows $((\leq_L) \equiv_o (\leq_R))\ l\ r$
  ⟨*proof*⟩

lemma *order-equivalenceE* [*elim*]:
  assumes $((\leq_L) \equiv_o (\leq_R))\ l\ r$
  obtains $((\leq_L) \Rightarrow (\leq_R))\ l\ ((\leq_R) \Rightarrow (\leq_L))\ r$
    rel-equivalence-on (*in-field* $(\leq_L)$) $(\leq_L)\ \eta$
    rel-equivalence-on (*in-field* $(\leq_R)$) $(\leq_R)\ \varepsilon$
  ⟨*proof*⟩

interpretation *of* : *order-functors S T f g* **for** *S T f g* ⟨*proof*⟩

lemma *rel-inv-order-equivalence-eq-order-equivalence* [*simp*]:
  $((\leq_R) \equiv_o (\leq_L))^{-1} = ((\leq_L) \equiv_o (\leq_R))$
  ⟨*proof*⟩

corollary *order-equivalence-right-left-iff-order-equivalence-left-right*:
  $((\leq_R) \equiv_o (\leq_L))\ r\ l \longleftrightarrow ((\leq_L) \equiv_o (\leq_R))\ l\ r$
  ⟨*proof*⟩

Due to the symmetry given by $((\leq_R) \equiv_o (\leq_L))\ r\ l =$ *order-equivalence*, for any theorem on $(\leq_L)$, we obtain a corresponding theorem on $(\leq_R)$ by flipping the roles of the two functors. As such, in what follows, we do not explicitly state these free theorems but users can obtain them as needed by creating a flipped interpretation of *order-functors*.

lemma *order-equivalence-rel-inv-eq-order-equivalence* [*simp*]:
$((\geq_L) \equiv_o (\geq_R)) = ((\leq_L) \equiv_o (\leq_R))$
⟨*proof*⟩

lemma *in-codom-left-eq-in-dom-left-if-order-equivalence*:
assumes $((\leq_L) \equiv_o (\leq_R))$ *l r*
shows *in-codom* $(\leq_L)$ = *in-dom* $(\leq_L)$
⟨*proof*⟩

corollary *preorder-on-in-field-left-if-transitive-if-order-equivalence*:
assumes $((\leq_L) \equiv_o (\leq_R))$ *l r*
and *transitive* $(\leq_L)$
shows *preorder-on* (*in-field* $(\leq_L)$) $(\leq_L)$
⟨*proof*⟩

lemma *order-equivalence-partial-equivalence-rel-not-reflexive-not-transitive*:
assumes $\exists (y :: 'b)\ y'.\ y \neq y'$
shows $\exists (L :: 'a \Rightarrow 'a \Rightarrow bool)\ (R :: 'b \Rightarrow 'b \Rightarrow bool)\ l\ r.$
$(L \equiv_o R)\ l\ r \wedge$ *partial-equivalence-rel L* $\wedge$
$\neg$(*reflexive-on* (*in-field R*) *R*) $\wedge$ $\neg$(*transitive-on* (*in-field R*) *R*)
⟨*proof*⟩

end

end

1.4.3 Half Galois Property

theory *Half-Galois-Property*
imports
Galois-Relator-Base
Order-Equivalences
begin

As the definition of the Galois property also works on heterogeneous relations, we define the concepts in a locale that generalises *galois*.

locale *galois-prop* = *orders L R*
for $L :: 'a \Rightarrow 'b \Rightarrow bool$
and $R :: 'c \Rightarrow 'd \Rightarrow bool$
and $l :: 'a \Rightarrow 'c$
and $r :: 'd \Rightarrow 'b$
begin

sublocale *galois-rel L R r* ⟨*proof*⟩

interpretation *gr-flip-inv* : *galois-rel* $(\geq_R)$ $(\geq_L)$ *l* ⟨*proof*⟩

abbreviation *right-ge-Galois* ≡ *gr-flip-inv.Galois*

notation *right-ge-Galois* (**infix** $\langle_R \gtrsim\rangle$ *50*)

abbreviation (*input*) *Galois-right* $\equiv$ *gr-flip-inv.ge-Galois-left*
notation *Galois-right* (**infix** $\langle \lesssim_R\rangle$ *50*)

lemma *Galois-rightI* [*intro*]:
assumes *in-dom* $(\leq_L)$ x
and $l\ x \leq_R y$
shows $x \lesssim_R y$
⟨*proof*⟩

lemma *Galois-rightE* [*elim*]:
assumes $x \lesssim_R y$
obtains *in-dom* $(\leq_L)$ x $l\ x \leq_R y$
⟨*proof*⟩

corollary *Galois-right-iff-in-dom-and-left-right-rel*:
$x \lesssim_R y \longleftrightarrow$ *in-dom* $(\leq_L)$ $x \wedge l\ x \leq_R y$
⟨*proof*⟩

Unlike common literature, we split the definition of the Galois property into two halves. This has its merits in modularity of proofs and preciser statement of required assumptions.

definition *half-galois-prop-left* $\equiv \forall x\ y.\ x\ {}_L\lesssim\ y \longrightarrow l\ x \leq_R y$

notation *galois-prop.half-galois-prop-left* (**infix** $\langle_h \trianglelefteq\rangle$ *50*)

lemma *half-galois-prop-leftI* [*intro*]:
assumes $\bigwedge x\ y.\ x\ {}_L\lesssim\ y \Longrightarrow l\ x \leq_R y$
shows $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$
⟨*proof*⟩

lemma *half-galois-prop-leftD* [*dest*]:
assumes $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$
and $x\ {}_L\lesssim\ y$
shows $l\ x \leq_R y$
⟨*proof*⟩

Observe that the second half can be obtained by creating an appropriately flipped and inverted interpretation of *galois-prop*. Indeed, many concepts in our formalisation are "closed" under inversion, i.e. taking their inversion yields a statement for a related concept. Many theorems can thus be derived for free by inverting (and flipping) the concepts at hand. In such cases, we only state those theorems that require some non-trivial setup. All other theorems can simply be obtained by creating a suitable locale interpretation.

interpretation *flip-inv* : *galois-prop* $(\geq_R)$ $(\geq_L)$ r l ⟨*proof*⟩

definition *half-galois-prop-right* ≡ *flip-inv.half-galois-prop-left*

notation *galois-prop.half-galois-prop-right* (**infix** ‹$\trianglelefteq_h$› *50*)

lemma *half-galois-prop-rightI* [*intro*]:
assumes $\bigwedge x\ y.\ x \lessapprox_R y \Longrightarrow x \leq_L r\ y$
shows $((\leq_L) \trianglelefteq_h (\leq_R))\ l\ r$
⟨*proof*⟩

lemma *half-galois-prop-rightD* [*dest*]:
assumes $((\leq_L) \trianglelefteq_h (\leq_R))\ l\ r$
and $x \lessapprox_R y$
shows $x \leq_L r\ y$
⟨*proof*⟩

interpretation *g* : *galois-prop S T f g* **for** *S T f g* ⟨*proof*⟩

lemma *rel-inv-half-galois-prop-right-eq-half-galois-prop-left-rel-inv* [*simp*]:
$((\leq_R) \trianglelefteq_h (\leq_L))^{-1} = ((\geq_L)\ {}_h\trianglerighteq (\geq_R))$
⟨*proof*⟩

corollary *half-galois-prop-left-rel-inv-iff-half-galois-prop-right* [*iff*]:
$((\geq_L)\ {}_h\trianglerighteq (\geq_R))\ f\ g \longleftrightarrow ((\leq_R) \trianglelefteq_h (\leq_L))\ g\ f$
⟨*proof*⟩

lemma *rel-inv-half-galois-prop-left-eq-half-galois-prop-right-rel-inv* [*simp*]:
$((\leq_R)\ {}_h\trianglerighteq (\leq_L))^{-1} = ((\geq_L) \trianglelefteq_h (\geq_R))$
⟨*proof*⟩

corollary *half-galois-prop-right-rel-inv-iff-half-galois-prop-left* [*iff*]:
$((\geq_L) \trianglelefteq_h (\geq_R))\ f\ g \longleftrightarrow ((\leq_R)\ {}_h\trianglerighteq (\leq_L))\ g\ f$
⟨*proof*⟩

end

context *galois*
begin

sublocale *galois-prop L R l r* ⟨*proof*⟩

interpretation *flip* : *galois R L r l* ⟨*proof*⟩

abbreviation *right-Galois* ≡ *flip.Galois*
notation *right-Galois* (**infix** ‹$_R\gtrsim$› *50*)

abbreviation (*input*) *ge-Galois-right* ≡ *flip.ge-Galois-left*
notation *ge-Galois-right* (**infix** ‹$\gtrsim_R$› *50*)

abbreviation *left-ge-Galois* ≡ *flip.right-ge-Galois*

notation *left-ge-Galois* (**infix** $\langle_L \gtrsim\rangle$ *50*)

abbreviation (*input*) *Galois-left* ≡ *flip.Galois-right*
notation *Galois-left* (**infix** $\langle \lesssim_L \rangle$ *50*)

context
begin

interpretation *flip-inv* : *galois* $(\geq_R)$ $(\geq_L)$ *r l* ⟨*proof*⟩

lemma *rel-unit-if-left-rel-if-mono-wrt-relI*:
assumes $((\leq_L) \Rightarrow (\leq_R))$ *l*
and $x \lesssim_R l\ x' \Longrightarrow x \leq_L \eta\ x'$
and $x \leq_L x'$
shows $x \leq_L \eta\ x'$
⟨*proof*⟩

corollary *rel-unit-if-left-rel-if-half-galois-prop-right-if-mono-wrt-rel*:
assumes $((\leq_L) \Rightarrow (\leq_R))$ *l*
and $((\leq_L) \trianglelefteq_h (\leq_R))$ *l r*
and $x \leq_L x'$
shows $x \leq_L \eta\ x'$
⟨*proof*⟩

corollary *rel-unit-if-reflexive-on-if-half-galois-prop-right-if-mono-wrt-rel*:
assumes $((\leq_L) \Rightarrow (\leq_R))$ *l*
and $((\leq_L) \trianglelefteq_h (\leq_R))$ *l r*
and *reflexive-on P* $(\leq_L)$
and *P x*
shows $x \leq_L \eta\ x$
⟨*proof*⟩

corollary *inflationary-on-unit-if-reflexive-on-if-half-galois-prop-rightI*:
fixes *P* :: *'a* ⇒ *bool*
assumes $((\leq_L) \Rightarrow (\leq_R))$ *l*
and $((\leq_L) \trianglelefteq_h (\leq_R))$ *l r*
and *reflexive-on P* $(\leq_L)$
shows *inflationary-on P* $(\leq_L)$ η
⟨*proof*⟩

interpretation *flip* : *galois-prop R L r l* ⟨*proof*⟩

lemma *right-rel-if-Galois-left-right-if-deflationary-onI*:
assumes $((\leq_R) \Rightarrow (\leq_L))$ *r*
and $((\leq_R) \trianglelefteq_h (\leq_L))$ *r l*
and *deflationary-on P* $(\leq_R)$ ε
and *transitive* $(\leq_R)$
and $y \lesssim_L r\ y'$
and *P y'*

shows $y \leq_R y'$
⟨*proof*⟩

lemma *half-galois-prop-left-left-right-if-transitive-if-deflationary-on-if-mono-wrt-rel*:
assumes $((\leq_L) \Rightarrow (\leq_R))\ l$
and *deflationary-on* $(\textit{in-codom}\ (\leq_R))\ (\leq_R)\ \varepsilon$
and *transitive* $(\leq_R)$
shows $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$
⟨*proof*⟩

end

interpretation *flip-inv* : *galois* $(\geq_R)\ (\geq_L)\ r\ l$
rewrites *flip-inv.unit* $\equiv \varepsilon$ **and** *flip-inv.counit* $\equiv \eta$
and $\bigwedge R\ S.\ (R^{-1} \Rightarrow S^{-1}) \equiv (R \Rightarrow S)$
and $\bigwedge R\ S\ f\ g.\ (R^{-1} \trianglelefteq_h S^{-1})\ f\ g \equiv (S\ {}_h\trianglelefteq\ R)\ g\ f$
and $((\geq_R)\ {}_h\trianglelefteq\ (\geq_L))\ r\ l \equiv ((\leq_L) \trianglelefteq_h (\leq_R))\ l\ r$
and $\bigwedge R.\ R^{-1-1} \equiv R$
and $\bigwedge(P :: 'c \Rightarrow \textit{bool})\ (R :: 'c \Rightarrow 'c \Rightarrow \textit{bool}).$
$(\textit{inflationary-on}\ P\ R^{-1} :: ('c \Rightarrow 'c) \Rightarrow \textit{bool}) \equiv \textit{deflationary-on}\ P\ R$
and $\bigwedge(P :: 'c \Rightarrow \textit{bool})\ (R :: 'c \Rightarrow 'c \Rightarrow \textit{bool}).$
$(\textit{deflationary-on}\ P\ R^{-1}\ :: ('c \Rightarrow 'c) \Rightarrow \textit{bool}) \equiv \textit{inflationary-on}\ P\ R$
and $\bigwedge(P :: 'b \Rightarrow \textit{bool}).\ \textit{reflexive-on}\ P\ (\geq_R) \equiv \textit{reflexive-on}\ P\ (\leq_R)$
and $\bigwedge(R :: 'a \Rightarrow 'a \Rightarrow \textit{bool}).\ \textit{transitive}\ R^{-1} \equiv \textit{transitive}\ R$
and $\bigwedge R.\ \textit{in-codom}\ R^{-1} \equiv \textit{in-dom}\ R$
⟨*proof*⟩

corollary *counit-rel-if-right-rel-if-mono-wrt-relI*:
assumes $((\leq_R) \Rightarrow (\leq_L))\ r$
and $r\ y\ {}_L\lessapprox y' \Longrightarrow \varepsilon\ y \leq_R y'$
and $y \leq_R y'$
shows $\varepsilon\ y \leq_R y'$
⟨*proof*⟩

corollary *counit-rel-if-right-rel-if-half-galois-prop-left-if-mono-wrt-rel*:
assumes $((\leq_R) \Rightarrow (\leq_L))\ r$
and $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$
and $y \leq_R y'$
shows $\varepsilon\ y \leq_R y'$
⟨*proof*⟩

corollary *counit-rel-if-reflexive-on-if-half-galois-prop-left-if-mono-wrt-rel*:
assumes $((\leq_R) \Rightarrow (\leq_L))\ r$
and $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$
and *reflexive-on* $P\ (\leq_R)$
and $P\ y$
shows $\varepsilon\ y \leq_R y$
⟨*proof*⟩

corollary *deflationary-on-counit-if-reflexive-on-if-half-galois-prop-leftI*:
fixes $P :: 'b \Rightarrow bool$
assumes $((\leq_R) \Rightarrow (\leq_L))\ r$
and $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$
and *reflexive-on* $P\ (\leq_R)$
shows *deflationary-on* $P\ (\leq_R)\ \varepsilon$
⟨*proof*⟩

corollary *left-rel-if-left-right-Galois-if-inflationary-onI*:
assumes $((\leq_L) \Rightarrow (\leq_R))\ l$
and $((\leq_R)\ {}_h\trianglelefteq\ (\leq_L))\ r\ l$
and *inflationary-on* $P\ (\leq_L)\ \eta$
and *transitive* $(\leq_L)$
and $l\ x\ {}_R\overset{\leq}{\approx}\ x'$
and $P\ x$
shows $x \leq_L x'$
⟨*proof*⟩

corollary *half-galois-prop-right-left-right-if-transitive-if-inflationary-on-if-mono-wrt-rel*:
assumes $((\leq_R) \Rightarrow (\leq_L))\ r$
and *inflationary-on* $(\textit{in-dom}\ (\leq_L))\ (\leq_L)\ \eta$
and *transitive* $(\leq_L)$
shows $((\leq_L)\ \trianglelefteq_h\ (\leq_R))\ l\ r$
⟨*proof*⟩

end

context *order-functors*
begin

interpretation g : *galois* $L\ R\ l\ r$ ⟨*proof*⟩
interpretation *flip-g* : *galois* $R\ L\ r\ l$
rewrites *flip-g.unit* $\equiv \varepsilon$ **and** *flip-g.counit* $\equiv \eta$
⟨*proof*⟩

lemma *left-rel-if-left-right-rel-left-if-order-equivalenceI*:
assumes $((\leq_L) \equiv_o (\leq_R))\ l\ r$
and *transitive* $(\leq_L)$
and $l\ x \leq_R l\ x'$
and *in-dom* $(\leq_L)\ x$
and *in-codom* $(\leq_L)\ x'$
shows $x \leq_L x'$
⟨*proof*⟩

end

end

1.4.4 Galois Property

theory *Galois-Property*
imports
Half-Galois-Property
begin

context *galois-prop*
begin

definition *galois-prop* $\equiv ((\leq_L)\ {}_h\trianglelefteq\ (\leq_R)) \sqcap ((\leq_L)\ \trianglelefteq_h\ (\leq_R))$

notation *galois-prop.galois-prop* (**infix** $\langle\trianglelefteq\rangle$ *50*)

lemma *galois-propI* [*intro*]:
assumes $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$
and $((\leq_L)\ \trianglelefteq_h\ (\leq_R))\ l\ r$
shows $((\leq_L) \trianglelefteq (\leq_R))\ l\ r$
⟨*proof*⟩

lemma *galois-propI'*:
assumes $\bigwedge x\ y.$ *in-dom* $(\leq_L)\ x \Longrightarrow$ *in-codom* $(\leq_R)\ y \Longrightarrow x \leq_L r\ y \longleftrightarrow l\ x \leq_R y$
shows $((\leq_L) \trianglelefteq (\leq_R))\ l\ r$
⟨*proof*⟩

lemma *galois-propE* [*elim*]:
assumes $((\leq_L) \trianglelefteq (\leq_R))\ l\ r$
obtains $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r\ ((\leq_L)\ \trianglelefteq_h\ (\leq_R))\ l\ r$
⟨*proof*⟩

interpretation g : *galois-prop* $S\ T\ f\ g$ **for** $S\ T\ f\ g$⟨*proof*⟩

lemma *galois-prop-eq-half-galois-prop-left-rel-inf-half-galois-prop-right*:
$((\leq_L) \trianglelefteq (\leq_R)) = ((\leq_L)\ {}_h\trianglelefteq\ (\leq_R)) \sqcap ((\leq_L)\ \trianglelefteq_h\ (\leq_R))$
⟨*proof*⟩

lemma *galois-prop-left-rel-right-iff-left-right-rel*:
assumes $((\leq_L) \trianglelefteq (\leq_R))\ l\ r$
and *in-dom* $(\leq_L)\ x$ *in-codom* $(\leq_R)\ y$
shows $x \leq_L r\ y \longleftrightarrow l\ x \leq_R y$
⟨*proof*⟩

lemma *rel-inv-galois-prop-eq-galois-prop-rel-inv* [*simp*]:
$((\leq_R) \trianglelefteq (\leq_L))^{-1} = ((\geq_L) \trianglelefteq (\geq_R))$
⟨*proof*⟩

corollary *galois-prop-rel-inv-iff-galois-prop* [*iff*]:
$((\geq_L) \trianglelefteq (\geq_R))\ f\ g \longleftrightarrow ((\leq_R) \trianglelefteq (\leq_L))\ g\ f$
⟨*proof*⟩

end

context *galois*
begin

lemma *galois-prop-left-right-if-transitive-if-deflationary-on-if-inflationary-on-if-mono-wrt-rel*:
assumes $((\leq_L) \Rightarrow (\leq_R))\ l$ **and** $((\leq_R) \Rightarrow (\leq_L))\ r$
and *inflationary-on* $(\textit{in-dom}\ (\leq_L))\ (\leq_L)\ \eta$
and *deflationary-on* $(\textit{in-codom}\ (\leq_R))\ (\leq_R)\ \varepsilon$
and *transitive* $(\leq_L)$ *transitive* $(\leq_R)$
shows $((\leq_L) \trianglelefteq (\leq_R))\ l\ r$
⟨*proof*⟩

end

end

1.4.5 Galois Connections

theory *Galois-Connections*
imports
Galois-Property
begin

context *galois*
begin

definition *galois-connection* $\equiv ((\leq_L) \Rightarrow (\leq_R))\ l \wedge ((\leq_R) \Rightarrow (\leq_L))\ r \wedge ((\leq_L) \trianglelefteq (\leq_R))\ l\ r$

notation *galois.galois-connection* (**infix** $\langle\dashv\rangle$ *50*)

lemma *galois-connectionI* [*intro*]:
assumes $((\leq_L) \Rightarrow (\leq_R))\ l$ **and** $((\leq_R) \Rightarrow (\leq_L))\ r$
and $((\leq_L) \trianglelefteq (\leq_R))\ l\ r$
shows $((\leq_L) \dashv (\leq_R))\ l\ r$
⟨*proof*⟩

lemma *galois-connectionE* [*elim*]:
assumes $((\leq_L) \dashv (\leq_R))\ l\ r$
obtains $((\leq_L) \Rightarrow (\leq_R))\ l\ ((\leq_R) \Rightarrow (\leq_L))\ r\ ((\leq_L) \trianglelefteq (\leq_R))\ l\ r$
⟨*proof*⟩

context
begin

interpretation g : *galois* $S\ T\ f\ g$ **for** $S\ T\ f\ g$⟨*proof*⟩

lemma *rel-inv-galois-connection-eq-galois-connection-rel-inv* [*simp*]:
$((\leq_R) \dashv (\leq_L))^{-1} = ((\geq_L) \dashv (\geq_R))$
⟨*proof*⟩

corollary *galois-connection-rel-inv-iff-galois-connection* [*iff*]:
$((\geq_L) \dashv (\geq_R))\ l\ r \longleftrightarrow ((\leq_R) \dashv (\leq_L))\ r\ l$
⟨*proof*⟩

lemma *rel-unit-if-left-rel-if-galois-connection*:
assumes $((\leq_L) \dashv (\leq_R))\ l\ r$
and $x \leq_L x'$
shows $x \leq_L \eta\ x'$
⟨*proof*⟩

end

lemma *counit-rel-if-right-rel-if-galois-connection*:
assumes $((\leq_L) \dashv (\leq_R))\ l\ r$
and $y \leq_R y'$
shows $\varepsilon\ y \leq_R y'$
⟨*proof*⟩

lemma *rel-unit-if-reflexive-on-if-galois-connection*:
assumes $((\leq_L) \dashv (\leq_R))\ l\ r$
and *reflexive-on* $P\ (\leq_L)$
and $P\ x$
shows $x \leq_L \eta\ x$
⟨*proof*⟩

lemma *counit-rel-if-reflexive-on-if-galois-connection*:
assumes $((\leq_L) \dashv (\leq_R))\ l\ r$
and *reflexive-on* $P\ (\leq_R)$
and $P\ y$
shows $\varepsilon\ y \leq_R y$
⟨*proof*⟩

lemma *inflationary-on-unit-if-reflexive-on-if-galois-connection*:
fixes $P :: 'a \Rightarrow bool$
assumes $((\leq_L) \dashv (\leq_R))\ l\ r$
and *reflexive-on* $P\ (\leq_L)$
shows *inflationary-on* $P\ (\leq_L)\ \eta$
⟨*proof*⟩

lemma *deflationary-on-counit-if-reflexive-on-if-galois-connection*:
fixes $P :: 'b \Rightarrow bool$
assumes $((\leq_L) \dashv (\leq_R))\ l\ r$
and *reflexive-on* $P\ (\leq_R)$
shows *deflationary-on* $P\ (\leq_R)\ \varepsilon$

⟨*proof*⟩

end

end

1.4.6 Galois Equivalences

theory *Galois-Equivalences*
imports
Galois-Connections
Order-Equivalences
Partial-Equivalence-Relations
begin

context *galois*
begin

In the literature, an adjoint equivalence is an adjunction for which the unit and counit are natural isomorphisms. Translated to the category of orders, this means that a *Galois equivalence* between two relations $(\leq_L)$ and $(\leq_R)$ is a Galois connection for which the unit η is *deflationary* and the counit ε is *inflationary*.

For reasons of symmetry, we give a different definition which next to *galois-connection* requires *galois-prop l r*. In other words, a Galois equivalence is a Galois connection for which the left and right adjoints are also right and left adjoints, respectively. As shown below, in the case of preorders, the definitions coincide.

definition *galois-equivalence* $\equiv ((\leq_L) \dashv (\leq_R))\ l\ r \wedge ((\leq_R) \trianglelefteq (\leq_L))\ r\ l$

notation *galois.galois-equivalence* (**infix** $\langle\equiv_G\rangle$ *50*)

lemma *galois-equivalenceI* [*intro*]:
assumes $((\leq_L) \dashv (\leq_R))\ l\ r$
and $((\leq_R) \trianglelefteq (\leq_L))\ r\ l$
shows $((\leq_L) \equiv_G (\leq_R))\ l\ r$
⟨*proof*⟩

lemma *galois-equivalenceE* [*elim*]:
assumes $((\leq_L) \equiv_G (\leq_R))\ l\ r$
obtains $((\leq_L) \dashv (\leq_R))\ l\ r\ ((\leq_R) \dashv (\leq_L))\ r\ l$
⟨*proof*⟩

context
begin

interpretation g : *galois* $S\ T\ f\ g$ **for** $S\ T\ f\ g$⟨*proof*⟩

lemma *galois-equivalence-eq-galois-connection-rel-inf-galois-prop*:
$((\leq_L) \equiv_G (\leq_R)) = ((\leq_L) \dashv (\leq_R)) \sqcap ((\geq_L) \trianglelefteq (\geq_R))$
⟨*proof*⟩

lemma *rel-inv-galois-equivalence-eq-galois-equivalence* [*simp*]:
$((\leq_R) \equiv_G (\leq_L))^{-1} = ((\leq_L) \equiv_G (\leq_R))$
⟨*proof*⟩

corollary *galois-equivalence-right-left-iff-galois-equivalence-left-right*:
$((\leq_R) \equiv_G (\leq_L))\ r\ l \longleftrightarrow ((\leq_L) \equiv_G (\leq_R))\ l\ r$
⟨*proof*⟩

lemma *galois-equivalence-rel-inv-eq-galois-equivalence* [*simp*]:
$((\geq_L) \equiv_G (\geq_R)) = ((\leq_L) \equiv_G (\leq_R))$
⟨*proof*⟩

lemma *inflationary-on-unit-if-reflexive-on-if-galois-equivalence*:
fixes $P :: 'a \Rightarrow bool$
assumes $((\leq_L) \equiv_G (\leq_R))\ l\ r$
and *reflexive-on* $P\ (\leq_L)$
shows *inflationary-on* $P\ (\leq_L)\ \eta$
⟨*proof*⟩

end

lemma *deflationary-on-unit-if-reflexive-on-if-galois-equivalence*:
fixes $P :: 'a \Rightarrow bool$
assumes $((\leq_L) \equiv_G (\leq_R))\ l\ r$
and *reflexive-on* $P\ (\leq_L)$
shows *deflationary-on* $P\ (\leq_L)\ \eta$
⟨*proof*⟩

Every *galois-equivalence* on reflexive orders is a Galois equivalence in the sense of the common literature.

lemma *rel-equivalence-on-unit-if-reflexive-on-if-galois-equivalence*:
fixes $P :: 'a \Rightarrow bool$
assumes $((\leq_L) \equiv_G (\leq_R))\ l\ r$
and *reflexive-on* $P\ (\leq_L)$
shows *rel-equivalence-on* $P\ (\leq_L)\ \eta$
⟨*proof*⟩

lemma *galois-equivalence-partial-equivalence-rel-not-reflexive-not-transitive*:
assumes $\exists (y :: 'b)\ y'.\ y \neq y'$
shows $\exists (L :: 'a \Rightarrow 'a \Rightarrow bool)\ (R :: 'b \Rightarrow 'b \Rightarrow bool)\ l\ r.$
$(L \equiv_G R)\ l\ r \wedge$ *partial-equivalence-rel* $L \wedge$
$\neg($*reflexive-on* $($*in-field* $R)\ R) \wedge \neg($*transitive-on* $($*in-field* $R)\ R)$
⟨*proof*⟩

1.4.7 Equivalence of Order Equivalences and Galois Equivalences

In general categories, every adjoint equivalence is an equivalence but not vice versa. In the category of preorders, however, they are morally the same: the adjoint zigzag equations are satisfied up to unique isomorphism rather than equality. In the category of partial orders, the concepts coincide.

lemma *half-galois-prop-left-left-right-if-transitive-if-order-equivalence*:
assumes $((\leq_L) \equiv_o (\leq_R))$ *l r*
and *transitive* $(\leq_L)$ *transitive* $(\leq_R)$
shows $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))$ *l r*
⟨*proof*⟩

lemma *half-galois-prop-right-left-right-if-transitive-if-order-equivalence*:
assumes $((\leq_L) \equiv_o (\leq_R))$ *l r*
and *transitive* $(\leq_L)$ *transitive* $(\leq_R)$
shows $((\leq_L) \trianglelefteq_h (\leq_R))$ *l r*
⟨*proof*⟩

lemma *galois-prop-left-right-if-transitive-if-order-equivalence*:
assumes $((\leq_L) \equiv_o (\leq_R))$ *l r*
and *transitive* $(\leq_L)$ *transitive* $(\leq_R)$
shows $((\leq_L) \trianglelefteq (\leq_R))$ *l r*
⟨*proof*⟩

corollary *galois-connection-left-right-if-transitive-if-order-equivalence*:
assumes $((\leq_L) \equiv_o (\leq_R))$ *l r*
and *transitive* $(\leq_L)$ *transitive* $(\leq_R)$
shows $((\leq_L) \dashv (\leq_R))$ *l r*
⟨*proof*⟩

interpretation *flip* : *galois R L r l*
rewrites *flip.unit* $\equiv \varepsilon$
⟨*proof*⟩

corollary *galois-equivalence-left-right-if-transitive-if-order-equivalence*:
assumes $((\leq_L) \equiv_o (\leq_R))$ *l r*
and *transitive* $(\leq_L)$ *transitive* $(\leq_R)$
shows $((\leq_L) \equiv_G (\leq_R))$ *l r*
⟨*proof*⟩

lemma *order-equivalence-if-reflexive-on-in-field-if-galois-equivalence*:
assumes $((\leq_L) \equiv_G (\leq_R))$ *l r*
and *reflexive-on* (*in-field* $(\leq_L)$) $(\leq_L)$ *reflexive-on* (*in-field* $(\leq_R)$) $(\leq_R)$
shows $((\leq_L) \equiv_o (\leq_R))$ *l r*
⟨*proof*⟩

corollary *galois-equivalence-eq-order-equivalence-if-preorder-on-in-field*:
assumes *preorder-on* (*in-field* $(\leq_L)$) $(\leq_L)$ *preorder-on* (*in-field* $(\leq_R)$) $(\leq_R)$

shows $((\leq_L) \equiv_G (\leq_R)) = ((\leq_L) \equiv_o (\leq_R))$
⟨*proof*⟩

end

end

1.4.8 Relator For Galois Connections

theory *Galois-Relator*
imports
Galois-Relator-Base
Galois-Property
begin

context *galois-prop*
begin

interpretation *flip-inv* : *galois-rel* $(\geq_R)$ $(\geq_L)$ *l* ⟨*proof*⟩

lemma *left-Galois-if-Galois-right-if-half-galois-prop-right*:
assumes $((\leq_L) \trianglelefteq_h (\leq_R))$ *l r*
and $x \lessapprox_R y$
shows $x \ {}_L\lessapprox y$
⟨*proof*⟩

lemma *Galois-right-if-left-Galois-if-half-galois-prop-left*:
assumes $((\leq_L) \ {}_h\trianglelefteq (\leq_R))$ *l r*
and $x \ {}_L\lessapprox y$
shows $x \lessapprox_R y$
⟨*proof*⟩

corollary *Galois-right-iff-left-Galois-if-galois-prop* [*iff*]:
assumes $((\leq_L) \trianglelefteq (\leq_R))$ *l r*
shows $x \lessapprox_R y \longleftrightarrow x \ {}_L\lessapprox y$
⟨*proof*⟩

lemma *rel-inv-Galois-eq-flip-Galois-rel-inv-if-galois-prop* [*simp*]:
assumes $((\leq_L) \trianglelefteq (\leq_R))$ *l r*
shows $(\gtrsim_L) = ({}_R\gtrsim)$
⟨*proof*⟩

corollary *flip-Galois-rel-inv-iff-Galois-if-galois-prop* [*iff*]:
assumes $((\leq_L) \trianglelefteq (\leq_R))$ *l r*
shows $y \ {}_R\gtrsim x \longleftrightarrow x \ {}_L\lessapprox y$
⟨*proof*⟩

corollary *inv-flip-Galois-rel-inv-eq-Galois-if-galois-prop* [*simp*]:

assumes $((\leq_L) \trianglelefteq (\leq_R))$ l r
shows $(\lessapprox_R) = ({}_L\lessapprox)$ — Note that $(galois\text{-}rel.Galois\ (\leq_R)^{-1}\ (\leq_L)^{-1}\ l)^{-1} = (galois\text{-}rel.Galois\ (\leq_R)^{-1}\ (\leq_L)^{-1}\ l)^{-1}$
⟨*proof*⟩

end

context *galois*
begin

interpretation *flip-inv* : *galois* $(\geq_R)$ $(\geq_L)$ r l ⟨*proof*⟩

context
begin

interpretation *flip* : *galois* R L r l ⟨*proof*⟩

lemma *left-Galois-left-if-left-relI*:
assumes $((\leq_L) \Rightarrow (\leq_R))$ l
and $((\leq_L) \trianglelefteq_h (\leq_R))$ l r
and $x \leq_L x'$
shows $x\ {}_L\lessapprox\ l\ x'$
⟨*proof*⟩

corollary *left-Galois-left-if-reflexive-on-if-half-galois-prop-rightI*:
assumes $((\leq_L) \Rightarrow (\leq_R))$ l
and $((\leq_L) \trianglelefteq_h (\leq_R))$ l r
and *reflexive-on* P $(\leq_L)$
and P x
shows $x\ {}_L\lessapprox\ l\ x$
⟨*proof*⟩

lemma *left-Galois-left-if-in-codom-if-inflationary-onI*:
assumes $((\leq_L) \Rightarrow (\leq_R))$ l
and *inflationary-on* P $(\leq_L)$ η
and *in-codom* $(\leq_L)$ x
and P x
shows $x\ {}_L\lessapprox\ l\ x$
⟨*proof*⟩

lemma *left-Galois-left-if-in-codom-if-inflationary-on-in-codomI*:
assumes $((\leq_L) \Rightarrow (\leq_R))$ l
and *inflationary-on* (*in-codom* $(\leq_L)$) $(\leq_L)$ η
and *in-codom* $(\leq_L)$ x
shows $x\ {}_L\lessapprox\ l\ x$
⟨*proof*⟩

lemma *left-Galois-left-if-left-rel-if-inflationary-on-in-fieldI*:
assumes $((\leq_L) \Rightarrow (\leq_R))$ l

and *inflationary-on* (*in-field* ($\leq_L$)) ($\leq_L$) η
and $x \leq_L x$
shows $x \ {}_L\lessapprox\ l\ x$
⟨*proof*⟩

lemma *right-left-Galois-if-right-relI*:
assumes (($\leq_R$) $\Rightarrow$ ($\leq_L$)) r
and $y \leq_R y'$
shows $r\ y\ {}_L\lessapprox\ y'$
⟨*proof*⟩

corollary *right-left-Galois-if-reflexive-onI*:
assumes (($\leq_R$) $\Rightarrow$ ($\leq_L$)) r
and *reflexive-on* P ($\leq_R$)
and $P\ y$
shows $r\ y\ {}_L\lessapprox\ y$
⟨*proof*⟩

lemma *left-Galois-if-right-rel-if-left-GaloisI*:
assumes (($\leq_R$) $\Rightarrow$ ($\leq_L$)) r
and *transitive* ($\leq_L$)
and $x \ {}_L\lessapprox\ y$
and $y \leq_R z$
shows $x \ {}_L\lessapprox\ z$
⟨*proof*⟩

lemma *left-Galois-if-left-Galois-if-left-relI*:
assumes *transitive* ($\leq_L$)
and $x \leq_L y$
and $y \ {}_L\lessapprox\ z$
shows $x \ {}_L\lessapprox\ z$
⟨*proof*⟩

lemma *left-rel-if-right-Galois-if-left-GaloisI*:
assumes (($\leq_R$) ${}_h\trianglelefteq$ ($\leq_L$)) $r\ l$
and *transitive* ($\leq_L$)
and $x \ {}_L\lessapprox\ y$
and $y \ {}_R\lessapprox\ z$
shows $x \leq_L z$
⟨*proof*⟩

lemma *Dep-Fun-Rel-left-Galois-right-Galois-if-mono-wrt-rel* [*intro*]:
assumes (($\leq_L$) $\Rightarrow$ ($\leq_R$)) l
shows (({}$_L\lessapprox$) $\Rrightarrow$ ({}$_R\lessapprox$)) $l\ r$
⟨*proof*⟩

lemma *left-ge-Galois-eq-left-Galois-if-in-codom-eq-in-dom-if-symmetric*:
assumes *symmetric* ($\leq_L$)
and *in-codom* ($\leq_R$) = *in-dom* ($\leq_R$)

shows $({}_L\gtrsim) = ({}_L\lesssim)$ — Note that *galois-rel.Galois* $(\leq_L)^{-1}$ $(\leq_R)^{-1}$ r = *galois-rel.Galois* $(\leq_L)^{-1}$ $(\leq_R)^{-1}$ r
⟨*proof*⟩

end

interpretation *flip* : *galois* R L r l ⟨*proof*⟩

lemma *ge-Galois-right-eq-left-Galois-if-symmetric-if-in-codom-eq-in-dom-if-galois-prop*:
assumes $((\leq_L) \trianglelefteq (\leq_R))$ l r
and *in-codom* $(\leq_L)$ = *in-dom* $(\leq_L)$
and *symmetric* $(\leq_R)$
shows $(\gtrsim_R) = ({}_L\lesssim)$ — Note that *flip.left-Galois*$^{-1}$ = *flip.left-Galois*$^{-1}$
⟨*proof*⟩

interpretation *gp* : *galois-prop* $({}_L\lesssim)$ $({}_R\lesssim)$ l l ⟨*proof*⟩

lemma *half-galois-prop-left-left-Galois-right-Galois-if-half-galois-prop-leftI* [*intro*]:
assumes $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))$ l r
shows $(({}_L\lesssim)\ {}_h\trianglelefteq\ ({}_R\lesssim))$ l l
⟨*proof*⟩

lemma *half-galois-prop-right-left-Galois-right-Galois-if-half-galois-prop-rightI* [*intro*]:
assumes $((\leq_L) \trianglelefteq_h (\leq_R))$ l r
shows $(({}_L\lesssim) \trianglelefteq_h ({}_R\lesssim))$ l l
⟨*proof*⟩

corollary *galois-prop-left-Galois-right-Galois-if-galois-prop* [*intro*]:
assumes $((\leq_L) \trianglelefteq (\leq_R))$ l r
shows $(({}_L\lesssim) \trianglelefteq ({}_R\lesssim))$ l l
⟨*proof*⟩

end

end

Chapter 2

Transport

2.1 Basic Setup

theory *Transport-Base*
  imports
    Galois-Equivalences
    Galois-Relator
begin

Summary Basic setup for commonly used concepts in Transport, including a suitable locale.

locale *transport* = *galois* L R l r
  for $L :: 'a \Rightarrow 'a \Rightarrow bool$
  and $R :: 'b \Rightarrow 'b \Rightarrow bool$
  and $l :: 'a \Rightarrow 'b$
  and $r :: 'b \Rightarrow 'a$
begin

2.1.1 Ordered Galois Connections

definition *preorder-galois-connection* $\equiv$
  $((\leq_L) \dashv (\leq_R))\ l\ r$
  $\wedge$ *preorder-on* (*in-field* $(\leq_L)$) $(\leq_L)$
  $\wedge$ *preorder-on* (*in-field* $(\leq_R)$) $(\leq_R)$

notation *transport.preorder-galois-connection* (**infix** $\langle\dashv_{pre}\rangle$ *50*)

lemma *preorder-galois-connectionI* [*intro*]:
  assumes $((\leq_L) \dashv (\leq_R))\ l\ r$
  and *preorder-on* (*in-field* $(\leq_L)$) $(\leq_L)$
  and *preorder-on* (*in-field* $(\leq_R)$) $(\leq_R)$
  shows $((\leq_L) \dashv_{pre} (\leq_R))\ l\ r$
  ⟨*proof*⟩

lemma *preorder-galois-connectionE* [*elim*]:

assumes $((\leq_L) \dashv_{pre} (\leq_R))$ *l r*
obtains $((\leq_L) \dashv (\leq_R))$ *l r preorder-on* (*in-field* $(\leq_L)$) $(\leq_L)$
preorder-on (*in-field* $(\leq_R)$) $(\leq_R)$
⟨*proof*⟩

context
begin

interpretation *t* : *transport S T f g* **for** *S T f g* ⟨*proof*⟩

lemma *rel-inv-preorder-galois-connection-eq-preorder-galois-connection-rel-inv* [*simp*]:
$((\leq_R) \dashv_{pre} (\leq_L))^{-1} = ((\geq_L) \dashv_{pre} (\geq_R))$
⟨*proof*⟩

end

corollary *preorder-galois-connection-rel-inv-iff-preorder-galois-connection* [*iff*]:
$((\geq_L) \dashv_{pre} (\geq_R))$ *l r* $\longleftrightarrow$ $((\leq_R) \dashv_{pre} (\leq_L))$ *r l*
⟨*proof*⟩

definition *partial-equivalence-rel-galois-connection* $\equiv$
$((\leq_L) \dashv (\leq_R))$ *l r*
$\wedge$ *partial-equivalence-rel* $(\leq_L)$
$\wedge$ *partial-equivalence-rel* $(\leq_R)$

notation *transport.partial-equivalence-rel-galois-connection* (**infix** $\langle\dashv_{PER}\rangle$ *50*)

lemma *partial-equivalence-rel-galois-connectionI* [*intro*]:
assumes $((\leq_L) \dashv (\leq_R))$ *l r*
and *partial-equivalence-rel-on* (*in-field* $(\leq_L)$) $(\leq_L)$
and *partial-equivalence-rel-on* (*in-field* $(\leq_R)$) $(\leq_R)$
shows $((\leq_L) \dashv_{PER} (\leq_R))$ *l r*
⟨*proof*⟩

lemma *partial-equivalence-rel-galois-connectionE* [*elim*]:
assumes $((\leq_L) \dashv_{PER} (\leq_R))$ *l r*
obtains $((\leq_L) \dashv_{pre} (\leq_R))$ *l r symmetric* $(\leq_L)$ *symmetric* $(\leq_R)$
⟨*proof*⟩

context
begin

interpretation *t* : *transport S T f g* **for** *S T f g* ⟨*proof*⟩

lemma *rel-inv-partial-equivalence-rel-galois-connection-eq-partial-equivalence-rel-galois-connection-rel-inv*
[*simp*]: $((\leq_R) \dashv_{PER} (\leq_L))^{-1} = ((\geq_L) \dashv_{PER} (\geq_R))$
⟨*proof*⟩

end

corollary *partial-equivalence-rel-galois-connection-rel-inv-iff-partial-equivalence-rel-galois-connection* [*iff*]: $((\geq_L) \dashv_{PER} (\geq_R))\ l\ r \longleftrightarrow ((\leq_R) \dashv_{PER} (\leq_L))\ r\ l$
⟨*proof*⟩

lemma *left-Galois-comp-ge-Galois-left-eq-left-if-partial-equivalence-rel-galois-connection*:
assumes $((\leq_L) \dashv_{PER} (\leq_R))\ l\ r$
shows $((_L\lesssim) \circ\circ (\gtrsim_L)) = (\leq_L)$
⟨*proof*⟩

2.1.2 Ordered Equivalences

definition *preorder-equivalence* $\equiv$
$((\leq_L) \equiv_G (\leq_R))\ l\ r$
$\wedge$ *preorder-on* (*in-field* $(\leq_L)$) $(\leq_L)$
$\wedge$ *preorder-on* (*in-field* $(\leq_R)$) $(\leq_R)$

notation *transport.preorder-equivalence* (**infix** $\langle\equiv_{pre}\rangle$ *50*)

lemma *preorder-equivalence-if-galois-equivalenceI* [*intro*]:
assumes $((\leq_L) \equiv_G (\leq_R))\ l\ r$
and *preorder-on* (*in-field* $(\leq_L)$) $(\leq_L)$
and *preorder-on* (*in-field* $(\leq_R)$) $(\leq_R)$
shows $((\leq_L) \equiv_{pre} (\leq_R))\ l\ r$
⟨*proof*⟩

lemma *preorder-equivalence-if-order-equivalenceI*:
assumes $((\leq_L) \equiv_o (\leq_R))\ l\ r$
and *transitive* $(\leq_L)$
and *transitive* $(\leq_R)$
shows $((\leq_L) \equiv_{pre} (\leq_R))\ l\ r$
⟨*proof*⟩

lemma *preorder-equivalence-galois-equivalenceE* [*elim*]:
assumes $((\leq_L) \equiv_{pre} (\leq_R))\ l\ r$
obtains $((\leq_L) \equiv_G (\leq_R))\ l\ r$ *preorder-on* (*in-field* $(\leq_L)$) $(\leq_L)$
preorder-on (*in-field* $(\leq_R)$) $(\leq_R)$
⟨*proof*⟩

lemma *preorder-equivalence-order-equivalenceE*:
assumes $((\leq_L) \equiv_{pre} (\leq_R))\ l\ r$
obtains $((\leq_L) \equiv_o (\leq_R))\ l\ r$ *preorder-on* (*in-field* $(\leq_L)$) $(\leq_L)$
preorder-on (*in-field* $(\leq_R)$) $(\leq_R)$
⟨*proof*⟩

context
begin

interpretation t : *transport* $S\ T\ f\ g$ **for** $S\ T\ f\ g$ ⟨*proof*⟩

lemma *rel-inv-preorder-equivalence-eq-preorder-equivalence* [*simp*]:
$((\leq_R) \equiv_{pre} (\leq_L))^{-1} = ((\leq_L) \equiv_{pre} (\leq_R))$
⟨*proof*⟩

end

corollary *preorder-equivalence-right-left-iff-preorder-equivalence-left-right*:
$((\leq_R) \equiv_{pre} (\leq_L))\ r\ l \longleftrightarrow ((\leq_L) \equiv_{pre} (\leq_R))\ l\ r$
⟨*proof*⟩

lemma *preorder-equivalence-rel-inv-eq-preorder-equivalence* [*simp*]:
$((\geq_L) \equiv_{pre} (\geq_R)) = ((\leq_L) \equiv_{pre} (\leq_R))$
⟨*proof*⟩

definition *partial-equivalence-rel-equivalence* $\equiv$
$((\leq_L) \equiv_G (\leq_R))\ l\ r$
$\wedge$ *partial-equivalence-rel* $(\leq_L)$
$\wedge$ *partial-equivalence-rel* $(\leq_R)$

notation *transport.partial-equivalence-rel-equivalence* (**infix** $\langle\equiv_{PER}\rangle$ *50*)

lemma *partial-equivalence-rel-equivalence-if-galois-equivalenceI* [*intro*]:
assumes $((\leq_L) \equiv_G (\leq_R))\ l\ r$
and *partial-equivalence-rel* $(\leq_L)$
and *partial-equivalence-rel* $(\leq_R)$
shows $((\leq_L) \equiv_{PER} (\leq_R))\ l\ r$
⟨*proof*⟩

lemma *partial-equivalence-rel-equivalence-if-order-equivalenceI*:
assumes $((\leq_L) \equiv_o (\leq_R))\ l\ r$
and *partial-equivalence-rel* $(\leq_L)$
and *partial-equivalence-rel* $(\leq_R)$
shows $((\leq_L) \equiv_{PER} (\leq_R))\ l\ r$
⟨*proof*⟩

lemma *partial-equivalence-rel-equivalenceE* [*elim*]:
assumes $((\leq_L) \equiv_{PER} (\leq_R))\ l\ r$
obtains $((\leq_L) \equiv_{pre} (\leq_R))\ l\ r$ *symmetric* $(\leq_L)$ *symmetric* $(\leq_R)$
⟨*proof*⟩

context
begin

interpretation t : *transport* $S\ T\ f\ g$ **for** $S\ T\ f\ g$ ⟨*proof*⟩

lemma *rel-inv-partial-equivalence-rel-equivalence-eq-partial-equivalence-rel-equivalence* [*simp*]:
$((\leq_R) \equiv_{PER} (\leq_L))^{-1} = ((\leq_L) \equiv_{PER} (\leq_R))$

⟨*proof*⟩

end

corollary *partial-equivalence-rel-equivalence-right-left-iff-partial-equivalence-rel-equivalence-left-right*:
((≤$_R$) ≡$_{PER}$ (≤$_L$)) *r l* ⟷ ((≤$_L$) ≡$_{PER}$ (≤$_R$)) *l r*
⟨*proof*⟩

lemma *partial-equivalence-rel-equivalence-rel-inv-eq-partial-equivalence-rel-equivalence* [*simp*]:
((≥$_L$) ≡$_{PER}$ (≥$_R$)) = ((≤$_L$) ≡$_{PER}$ (≤$_R$))
⟨*proof*⟩

end

end

2.2 Compositions With Agreeing Relations

2.2.1 Basic Setup

theory *Transport-Compositions-Agree-Base*
imports
Transport-Base
begin

locale *transport-comp-agree* =
g1 : *galois L1 R1 l1 r1* + *g2* : *galois L2 R2 l2 r2*
for *L1* :: *'a* ⇒ *'a* ⇒ *bool*
and *R1* :: *'b* ⇒ *'b* ⇒ *bool*
and *l1* :: *'a* ⇒ *'b*
and *r1* :: *'b* ⇒ *'a*
and *L2* :: *'b* ⇒ *'b* ⇒ *bool*
and *R2* :: *'c* ⇒ *'c* ⇒ *bool*
and *l2* :: *'b* ⇒ *'c*
and *r2* :: *'c* ⇒ *'b*
begin

This locale collects results about the composition of transportable components under the assumption that the relations *R1* and *L2* agree (in one sense or another) whenever required. Such an agreement may not necessarily hold in practice, and the resulting theorems are not particularly pretty. However, in the special case where *R1* = *L2*, most side-conditions disappear and the results are very simple.

notation *L1* (**infix** ‹≤$_{L1}$› *50*)
notation *R1* (**infix** ‹≤$_{R1}$› *50*)
notation *L2* (**infix** ‹≤$_{L2}$› *50*)

notation *R2* (**infix** ‹$\leq_{R2}$› *50*)

notation *g1.ge-left* (**infix** ‹$\geq_{L1}$› *50*)
notation *g1.ge-right* (**infix** ‹$\geq_{R1}$› *50*)
notation *g2.ge-left* (**infix** ‹$\geq_{L2}$› *50*)
notation *g2.ge-right* (**infix** ‹$\geq_{R2}$› *50*)

notation *g1.left-Galois* (**infix** ‹$_{L1}\lessapprox$› *50*)
notation *g1.right-Galois* (**infix** ‹$_{R1}\lessapprox$› *50*)
notation *g2.left-Galois* (**infix** ‹$_{L2}\lessapprox$› *50*)
notation *g2.right-Galois* (**infix** ‹$_{R2}\lessapprox$› *50*)

notation *g1.ge-Galois-left* (**infix** ‹$\gtrapprox_{L1}$› *50*)
notation *g1.ge-Galois-right* (**infix** ‹$\gtrapprox_{R1}$› *50*)
notation *g2.ge-Galois-left* (**infix** ‹$\gtrapprox_{L2}$› *50*)
notation *g2.ge-Galois-right* (**infix** ‹$\gtrapprox_{R2}$› *50*)

notation *g1.right-ge-Galois* (**infix** ‹$_{R1}\gtrapprox$› *50*)
notation *g1.Galois-right* (**infix** ‹$\lessapprox_{R1}$› *50*)
notation *g2.right-ge-Galois* (**infix** ‹$_{R2}\gtrapprox$› *50*)
notation *g2.Galois-right* (**infix** ‹$\lessapprox_{R2}$› *50*)

notation *g1.left-ge-Galois* (**infix** ‹$_{L1}\gtrapprox$› *50*)
notation *g1.Galois-left* (**infix** ‹$\lessapprox_{L1}$› *50*)
notation *g2.left-ge-Galois* (**infix** ‹$_{L2}\gtrapprox$› *50*)
notation *g2.Galois-left* (**infix** ‹$\lessapprox_{L2}$› *50*)

notation *g1.unit* (‹η_1›)
notation *g1.counit* (‹ε_1›)
notation *g2.unit* (‹η_2›)
notation *g2.counit* (‹ε_2›)

abbreviation (*input*) $L \equiv L1$

definition $l \equiv l2 \circ l1$

lemma *left-eq-comp*: $l = l2 \circ l1$
⟨*proof*⟩

lemma *left-eq* [*simp*]: $l\ x = l2\ (l1\ x)$
⟨*proof*⟩

context
begin

interpretation *flip* : *transport-comp-agree R2 L2 r2 l2 R1 L1 r1 l1* ⟨*proof*⟩

abbreviation (*input*) $R \equiv$ *flip.L*
abbreviation $r \equiv$ *flip.l*

lemma *right-eq-comp*: $r = r1 \circ r2$
⟨*proof*⟩

lemma *right-eq* [*simp*]: $r\ z = r1\ (r2\ z)$
⟨*proof*⟩

lemmas *transport-defs* = *left-eq-comp right-eq-comp*

end

sublocale *transport L R l r* ⟨*proof*⟩

notation *L* (**infix** $\langle\leq_L\rangle$ *50*)
notation *R* (**infix** $\langle\leq_R\rangle$ *50*)

end

locale *transport-comp-same* =
transport-comp-agree L1 R1 l1 r1 R1 R2 l2 r2
for *L1* :: $'a \Rightarrow 'a \Rightarrow$ *bool*
and *R1* :: $'b \Rightarrow 'b \Rightarrow$ *bool*
and *l1* :: $'a \Rightarrow 'b$
and *r1* :: $'b \Rightarrow 'a$
and *R2* :: $'c \Rightarrow 'c \Rightarrow$ *bool*
and *l2* :: $'b \Rightarrow 'c$
and *r2* :: $'c \Rightarrow 'b$
begin

This locale is a special case of *transport-comp-agree* where the left and right components both use ($\leq_{R1}$) as their right and left relation, respectively. This is the special case that is most prominent in the literature. The resulting theorems are quite simple, but often not applicable in practice.

end

end

2.2.2 Monotonicity

theory *Transport-Compositions-Agree-Monotone*
imports
Transport-Compositions-Agree-Base
begin

context *transport-comp-agree*
begin

lemma *mono-wrt-rel-leftI*:
assumes $((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1* $((\leq_{L2}) \Rightarrow (\leq_{R2}))$ *l2*
and $\bigwedge x\ y.\ x \leq_{L1} y \Longrightarrow l1\ x \leq_{R1} l1\ y \Longrightarrow l1\ x \leq_{L2} l1\ y$
shows $((\leq_L) \Rightarrow (\leq_R))$ *l*
⟨*proof*⟩

end

context *transport-comp-same*
begin

lemma *mono-wrt-rel-leftI*:
assumes $((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1* $((\leq_{R1}) \Rightarrow (\leq_{R2}))$ *l2*
shows $((\leq_L) \Rightarrow (\leq_R))$ *l*
⟨*proof*⟩

end

end

2.2.3 Galois Property

theory *Transport-Compositions-Agree-Galois-Property*
imports
Transport-Compositions-Agree-Base
begin

context *transport-comp-agree*
begin

lemma *galois-propI*:
assumes *galois1*: $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
and *galois2*: $((\leq_{L2}) \trianglelefteq (\leq_{R2}))$ *l2 r2*
and *mono-l1*: (*in-dom* $(\leq_{L1}) \Rightarrow$ *in-dom* $(\leq_{L2}))$ *l1*
and *mono-r2*: (*in-codom* $(\leq_{R2}) \Rightarrow$ *in-codom* $(\leq_{R1}))$ *r2*
and *agree*: (*in-dom* $(\leq_{L1}) \Rrightarrow$ *in-codom* $(\leq_{R2}) \Rrightarrow (\longleftrightarrow)$)
(*rel-bimap l1 r2* $(\leq_{R1})$) (*rel-bimap l1 r2* $(\leq_{L2})$)
shows $((\leq_L) \trianglelefteq (\leq_R))$ *l r*
⟨*proof*⟩

end

context *transport-comp-same*
begin

corollary *galois-propI*:
assumes $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
and $((\leq_{R1}) \trianglelefteq (\leq_{R2}))$ *l2 r2*

and (*in-dom* ($\leq_{L1}$) $\Rightarrow$ *in-dom* ($\leq_{R1}$)) *l1*
and (*in-codom* ($\leq_{R2}$) $\Rightarrow$ *in-codom* ($\leq_{R1}$)) *r2*
shows (($\leq_L$) $\trianglelefteq$ ($\leq_R$)) *l r*
⟨*proof*⟩

end

end

2.2.4 Galois Connection

theory *Transport-Compositions-Agree-Galois-Connection*
imports
Transport-Compositions-Agree-Monotone
Transport-Compositions-Agree-Galois-Property
begin

context *transport-comp-agree*
begin

interpretation *flip* : *transport-comp-agree R2 L2 r2 l2 R1 L1 r1 l1* ⟨*proof*⟩

lemma *galois-connectionI*:
assumes *galois*: (($\leq_{L1}$) $\dashv$ ($\leq_{R1}$)) *l1 r1* (($\leq_{L2}$) $\dashv$ ($\leq_{R2}$)) *l2 r2*
and *mono-L1-L2-l1*: $\bigwedge x\ y.\ x \leq_{L1} y \Longrightarrow$ *l1* $x \leq_{R1}$ *l1* $y \Longrightarrow$ *l1* $x \leq_{L2}$ *l1* y
and *mono-R2-R1-r2*: $\bigwedge x\ y.\ x \leq_{R2} y \Longrightarrow$ *r2* $x \leq_{L2}$ *r2* $y \Longrightarrow$ *r2* $x \leq_{R1}$ *r2* y
and (*in-dom* ($\leq_{L1}$) $\Rrightarrow$ *in-codom* ($\leq_{R2}$) $\Rrightarrow$ ($\longleftrightarrow$)) (*rel-bimap l1 r2* ($\leq_{R1}$))
(*rel-bimap l1 r2* ($\leq_{L2}$))
shows (($\leq_L$) $\dashv$ ($\leq_R$)) *l r*
⟨*proof*⟩

lemma *galois-connectionI'*:
assumes (($\leq_{L1}$) $\dashv$ ($\leq_{R1}$)) *l1 r1* (($\leq_{L2}$) $\dashv$ ($\leq_{R2}$)) *l2 r2*
and (($\leq_{L1}$) $\Rightarrow$ ($\leq_{L2}$)) *l1* (($\leq_{R2}$) $\Rightarrow$ ($\leq_{R1}$)) *r2*
and (*in-dom* ($\leq_{L1}$) $\Rrightarrow$ *in-codom* ($\leq_{R2}$) $\Rrightarrow$ ($\longleftrightarrow$))
(*rel-bimap l1 r2* ($\leq_{R1}$)) (*rel-bimap l1 r2* ($\leq_{L2}$))
shows (($\leq_L$) $\dashv$ ($\leq_R$)) *l r*
⟨*proof*⟩

end

context *transport-comp-same*
begin

corollary *galois-connectionI*:
assumes (($\leq_{L1}$) $\dashv$ ($\leq_{R1}$)) *l1 r1* (($\leq_{R1}$) $\dashv$ ($\leq_{R2}$)) *l2 r2*
shows (($\leq_L$) $\dashv$ ($\leq_R$)) *l r*
⟨*proof*⟩

end

end

2.2.5 Galois Equivalence

theory *Transport-Compositions-Agree-Galois-Equivalence*
imports
Transport-Compositions-Agree-Galois-Connection
begin

context *transport-comp-agree*
begin

interpretation *flip* : *transport-comp-agree R2 L2 r2 l2 R1 L1 r1 l1* ⟨*proof*⟩

lemma *galois-equivalenceI*:
assumes *galois*: $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1* $((\leq_{L2}) \equiv_G (\leq_{R2}))$ *l2 r2*
and *mono-L1-L2-l1*: $\bigwedge x\ y.\ x \leq_{L1} y \Longrightarrow$ *l1 x* $\leq_{R1}$ *l1 y* $\Longrightarrow$ *l1 x* $\leq_{L2}$ *l1 y*
and *mono-R2-R1-r2*: $\bigwedge x\ y.\ x \leq_{R2} y \Longrightarrow$ *r2 x* $\leq_{L2}$ *r2 y* $\Longrightarrow$ *r2 x* $\leq_{R1}$ *r2 y*
and (*in-dom* $(\leq_{L1}) \Rrightarrow$ *in-codom* $(\leq_{R2}) \Rrightarrow (\longleftrightarrow)$)
(*rel-bimap l1 r2* $(\leq_{R1})$) (*rel-bimap l1 r2* $(\leq_{L2})$)
and *mono-iff2*: (*in-dom* $(\leq_{R2}) \Rrightarrow$ *in-codom* $(\leq_{L1}) \Rrightarrow (\longleftrightarrow)$)
(*rel-bimap r2 l1* $(\leq_{R1})$) (*rel-bimap r2 l1* $(\leq_{L2})$)
shows $((\leq_L) \equiv_G (\leq_R))$ *l r*
⟨*proof*⟩

lemma *galois-equivalenceI'*:
assumes $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1* $((\leq_{L2}) \equiv_G (\leq_{R2}))$ *l2 r2*
and $((\leq_{L1}) \Rightarrow (\leq_{L2}))$ *l1* $((\leq_{R2}) \Rightarrow (\leq_{R1}))$ *r2*
and (*in-dom* $(\leq_{L1}) \Rrightarrow$ *in-codom* $(\leq_{R2}) \Rrightarrow (\longleftrightarrow)$)
(*rel-bimap l1 r2* $(\leq_{R1})$) (*rel-bimap l1 r2* $(\leq_{L2})$)
and (*in-dom* $(\leq_{R2}) \Rrightarrow$ *in-codom* $(\leq_{L1}) \Rrightarrow (\longleftrightarrow)$)
(*rel-bimap r2 l1* $(\leq_{R1})$) (*rel-bimap r2 l1* $(\leq_{L2})$)
shows $((\leq_L) \equiv_G (\leq_R))$ *l r*
⟨*proof*⟩

end

context *transport-comp-same*
begin

lemma *galois-equivalenceI*:
assumes $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1* $((\leq_{R1}) \equiv_G (\leq_{R2}))$ *l2 r2*
shows $((\leq_L) \equiv_G (\leq_R))$ *l r*
⟨*proof*⟩

end

end

theory *Galois-Equivalent*
imports
Transport-Compositions-Agree-Galois-Equivalence
begin

context
begin

interpretation *galois L R l r* **for** *L R l r* ⟨*proof*⟩

definition *galois-equivalent L R* ≡ ∃ *l r*. (*L* ≡$_G$ *R*) *l r*

lemma *galois-equivalentI* [*intro*]:
assumes (*L* ≡$_G$ *R*) *l r*
shows *galois-equivalent L R*
⟨*proof*⟩

lemma *galois-equivalentE* [*elim*]:
assumes *galois-equivalent L R*
obtains *l r* **where** (*L* ≡$_G$ *R*) *l r*
⟨*proof*⟩

lemma *galois-equivalent-if-galois-equivalent*:
assumes *galois-equivalent L R*
shows *galois-equivalent R L*
⟨*proof*⟩

lemma *galois-equivalent-trans* [*trans*]:
assumes *galois-equivalent L M*
and *galois-equivalent M R*
shows *galois-equivalent L R*
⟨*proof*⟩

end

end

theory *Galois*
imports
Galois-Equivalences
Galois-Equivalent
Galois-Relator
begin

Summary We define the concept of (partial) Galois connections, Galois equivalences, and the Galois relator. For details refer to [2].

end

2.2.6 Linear Orders

theory *Linear-Orders*
imports
Binary-Relations-Connected
Partial-Orders
begin

definition *linear-order-on* ≡ *partial-order-on* ⊓ *connected-on*

lemma *linear-order-onI* [*intro*]:
assumes *partial-order-on P R*
and *connected-on P R*
shows *linear-order-on P R*
⟨*proof*⟩

lemma *linear-order-onE* [*elim*]:
assumes *linear-order-on P R*
obtains *partial-order-on P R connected-on P R*
⟨*proof*⟩

consts *linear-order* :: *'a* ⇒ *bool*

overloading
linear-order ≡ *linear-order* :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*
begin
definition (*linear-order* :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*) ≡ *linear-order-on* (⊤ :: *'a* ⇒ *bool*)
end

lemma *linear-order-eq-linear-order-on*:
(*linear-order* :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*) = *linear-order-on* (⊤ :: *'a* ⇒ *bool*)
⟨*proof*⟩

lemma *linear-order-eq-linear-order-on-uhint* [*uhint*]:
assumes *P* ≡ ⊤ :: *'a* ⇒ *bool*
shows (*linear-order* :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*) ≡ *linear-order-on P*
⟨*proof*⟩

context
fixes *R* :: *'a* ⇒ *'a* ⇒ *bool*
begin

lemma *linear-orderI* [*intro*]:
assumes *partial-order R*

and *connected R*
shows *linear-order R*
⟨*proof*⟩

lemma *linear-orderE* [*elim*]:
assumes *linear-order R*
obtains *partial-order R connected R*
⟨*proof*⟩

lemma *linear-order-on-if-linear-order*:
fixes *P* :: *'a* ⇒ *bool*
assumes *linear-order R*
shows *linear-order-on P R*
⟨*proof*⟩

end

end

Closure Operators

theory *Closure-Operators*
imports
Order-Functions-Base
begin

consts *idempotent-on* :: *'a* ⇒ *'b* ⇒ *'c* ⇒ *bool*

overloading
idempotent-on-pred ≡ *idempotent-on* :: (*'a* ⇒ *bool*) ⇒ (*'a* ⇒ *'a* ⇒ *bool*) ⇒ (*'a* ⇒ *'a*) ⇒ *bool*
begin
definition *idempotent-on-pred* (*P* :: *'a* ⇒ *bool*) (*R* :: *'a* ⇒ *'a* ⇒ *bool*) (*f* :: *'a* ⇒ *'a*) ≡
rel-equivalence-on P (*rel-map f R*) *f*
end

context
fixes *P* :: *'a* ⇒ *bool* **and** *R* :: *'a* ⇒ *'a* ⇒ *bool* **and** *f* :: *'a* ⇒ *'a*
begin

lemma *idempotent-onI* [*intro*]:
assumes ⋀*x*. *P x* ⟹ *f x* ≡$_R$ *f* (*f x*)
shows *idempotent-on P R f*
⟨*proof*⟩

lemma *idempotent-onE* [*elim*]:
assumes *idempotent-on P R f*
and *P x*

obtains $f\ x \equiv_R f\ (f\ x)$
⟨*proof*⟩

lemma *rel-equivalence-on-rel-map-iff-idempotent-on* [*iff*]:
rel-equivalence-on P (*rel-map f R*) *f* ⟷ *idempotent-on P R f*
⟨*proof*⟩

lemma *idempotent-onD*:
assumes *idempotent-on P R f*
and *P x*
shows $f\ x \equiv_R f\ (f\ x)$
⟨*proof*⟩

end

consts *idempotent* :: $'a \Rightarrow 'b \Rightarrow$ *bool*

overloading
idempotent ≡ *idempotent* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a) \Rightarrow bool$
begin
definition (*idempotent* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a) \Rightarrow bool$) ≡ *idempotent-on* (⊤ :: $'a \Rightarrow bool$)
end

lemma *idempotent-eq-idempotent-on*:
(*idempotent* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a) \Rightarrow bool$) = *idempotent-on* (⊤ :: $'a \Rightarrow bool$)
⟨*proof*⟩

lemma *idempotent-eq-idempotent-on-uhint* [*uhint*]:
assumes *P* ≡ (⊤ :: $'a \Rightarrow bool$)
shows *idempotent* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a) \Rightarrow bool \equiv$ *idempotent-on P*
⟨*proof*⟩

context
fixes *P* :: $'a \Rightarrow bool$ **and** *R* :: $'a \Rightarrow 'a \Rightarrow bool$ **and** *f* :: $'a \Rightarrow 'a$
begin

lemma *idempotentI* [*intro*]:
assumes $\bigwedge x.\ f\ x \equiv_R f\ (f\ x)$
shows *idempotent R f*
⟨*proof*⟩

lemma *idempotentD* [*dest*]:
assumes *idempotent R f*
shows $f\ x \equiv_R f\ (f\ x)$
⟨*proof*⟩

lemma *idempotent-on-if-idempotent*:

assumes *idempotent R f*
shows *idempotent-on P R f*
⟨*proof*⟩

end

consts *closure-operator* :: $'a \Rightarrow 'b \Rightarrow bool$

overloading
closure-operator ≡ *closure-operator* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a) \Rightarrow bool$
begin
definition *closure-operator* $(R :: 'a \Rightarrow 'a \Rightarrow bool) :: ('a \Rightarrow 'a) \Rightarrow bool \equiv$
$(R \Rightarrow R) \sqcap$ *inflationary-on* (*in-field R*) *R* $\sqcap$ *idempotent-on* (*in-field R*) *R*
end

lemma *closure-operatorI* [*intro*]:
assumes $(R \Rightarrow R)$ *f*
and *inflationary-on* (*in-field R*) *R f*
and *idempotent-on* (*in-field R*) *R f*
shows *closure-operator R f*
⟨*proof*⟩

lemma *closure-operatorE* [*elim*]:
assumes *closure-operator R f*
obtains $(R \Rightarrow R)$ *f inflationary-on* (*in-field R*) *R f*
idempotent-on (*in-field R*) *R f*
⟨*proof*⟩

lemma *mono-wrt-rel-if-closure-operator*:
assumes *closure-operator* $(R :: 'a \Rightarrow 'a \Rightarrow bool)$ *f*
shows $(R \Rightarrow R)$ *f*
⟨*proof*⟩

context
fixes $R :: 'a \Rightarrow 'a \Rightarrow bool$ **and** $f :: 'a \Rightarrow 'a$
begin

lemma *inflationary-on-in-field-if-closure-operator*:
assumes *closure-operator R f*
shows *inflationary-on* (*in-field R*) *R f*
⟨*proof*⟩

lemma *idempotent-on-in-field-if-closure-operator*:
assumes *closure-operator R f*
shows *idempotent-on* (*in-field R*) *R f*
⟨*proof*⟩

end

end

theory *Order-Functions*
imports
Order-Functions-Base
Closure-Operators
begin

Summary Basic functions on orders.

end

2.3 Transport using Bijections

theory *Transport-Bijections*
imports
Restricted-Equality
Functions-Bijection
Transport-Base
Binary-Relations-Asymmetric
Binary-Relations-Connected
begin

Summary Setup for Transport using bijective transport functions.

locale *transport-bijection* =
fixes $L :: 'a \Rightarrow 'a \Rightarrow bool$
and $R :: 'b \Rightarrow 'b \Rightarrow bool$
and $l :: 'a \Rightarrow 'b$
and $r :: 'b \Rightarrow 'a$
assumes *mono-wrt-rel-left*: $(L \Rightarrow R)\ l$
and *mono-wrt-rel-right*: $(R \Rightarrow L)\ r$
and *inverse-left-right*: *inverse-on* (*in-field* L) $l\ r$
and *inverse-right-left*: *inverse-on* (*in-field* R) $r\ l$
begin

interpretation *transport* $L\ R\ l\ r$ ⟨*proof*⟩
interpretation *g-flip-inv* : *galois* $(\geq_R)$ $(\geq_L)\ r\ l$ ⟨*proof*⟩

lemma *bijection-on-in-field*: *bijection-on* (*in-field* $(\leq_L)$) (*in-field* $(\leq_R)$) $l\ r$
⟨*proof*⟩

lemma *half-galois-prop-left*: $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$
⟨*proof*⟩

lemma *half-galois-prop-right*: $((\leq_L)\ \trianglelefteq_h\ (\leq_R))\ l\ r$
⟨*proof*⟩

lemma *galois-prop*: $((\leq_L) \trianglelefteq (\leq_R))\ l\ r$

⟨*proof*⟩

lemma *galois-connection*: $((\leq_L) \dashv (\leq_R))\ l\ r$
⟨*proof*⟩

lemma *rel-equivalence-on-unitI*:
assumes *reflexive-on* (*in-field* $(\leq_L)$) $(\leq_L)$
shows *rel-equivalence-on* (*in-field* $(\leq_L)$) $(\leq_L)$ η
⟨*proof*⟩

interpretation *flip* : *transport-bijection* $R\ L\ r\ l$
rewrites *order-functors.unit* $r\ l \equiv \varepsilon$
⟨*proof*⟩

lemma *galois-equivalence*: $((\leq_L) \equiv_G (\leq_R))\ l\ r$
⟨*proof*⟩

lemmas *rel-equivalence-on-counitI* = *flip.rel-equivalence-on-unitI*

lemma *order-equivalenceI*:
assumes *reflexive-on* (*in-field* $(\leq_L)$) $(\leq_L)$
and *reflexive-on* (*in-field* $(\leq_R)$) $(\leq_R)$
shows $((\leq_L) \equiv_o (\leq_R))\ l\ r$
⟨*proof*⟩

lemma *preorder-equivalenceI*:
assumes *preorder-on* (*in-field* $(\leq_L)$) $(\leq_L)$
and *preorder-on* (*in-field* $(\leq_R)$) $(\leq_R)$
shows $((\leq_L) \equiv_{pre} (\leq_R))\ l\ r$
⟨*proof*⟩

lemma *partial-equivalence-rel-equivalenceI*:
assumes *partial-equivalence-rel* $(\leq_L)$
and *partial-equivalence-rel* $(\leq_R)$
shows $((\leq_L) \equiv_{PER} (\leq_R))\ l\ r$
⟨*proof*⟩

end

locale *transport-reflexive-on-in-field-bijection* =
fixes $L :: 'a \Rightarrow 'a \Rightarrow bool$
and $R :: 'b \Rightarrow 'b \Rightarrow bool$
and $l :: 'a \Rightarrow 'b$
and $r :: 'b \Rightarrow 'a$
assumes *reflexive-on-in-field-left*: *reflexive-on* (*in-field L*) *L*
and *reflexive-on-in-field-right*: *reflexive-on* (*in-field R*) *R*
and *transport-bijection*: *transport-bijection L R l r*
begin

sublocale *tbij?* : *transport-bijection L R l r*
  rewrites *reflexive-on* (*in-field L*) *L* ≡ *True*
  and *reflexive-on* (*in-field R*) *R* ≡ *True*
  and ⋀*P*. (*True* ⟹ *P*) ≡ *Trueprop P*
  ⟨*proof*⟩

lemmas *rel-equivalence-on-unit* = *rel-equivalence-on-unitI*
lemmas *rel-equivalence-on-counit* = *rel-equivalence-on-counitI*
lemmas *order-equivalence* = *order-equivalenceI*

end

locale *transport-preorder-on-in-field-bijection* =
  fixes *L* :: *'a* ⇒ *'a* ⇒ *bool*
  and *R* :: *'b* ⇒ *'b* ⇒ *bool*
  and *l* :: *'a* ⇒ *'b*
  and *r* :: *'b* ⇒ *'a*
  assumes *preorder-on-in-field-left*: *preorder-on* (*in-field L*) *L*
  and *preorder-on-in-field-right*: *preorder-on* (*in-field R*) *R*
  and *transport-bijection*: *transport-bijection L R l r*
begin

sublocale *trefl-bij?* : *transport-reflexive-on-in-field-bijection L R l r*
  rewrites *preorder-on* (*in-field L*) *L* ≡ *True*
  and *preorder-on* (*in-field R*) *R* ≡ *True*
  and ⋀*P*. (*True* ⟹ *P*) ≡ *Trueprop P*
  ⟨*proof*⟩

lemmas *preorder-equivalence* = *preorder-equivalenceI*

end

locale *transport-partial-equivalence-rel-bijection* =
  fixes *L* :: *'a* ⇒ *'a* ⇒ *bool*
  and *R* :: *'b* ⇒ *'b* ⇒ *bool*
  and *l* :: *'a* ⇒ *'b*
  and *r* :: *'b* ⇒ *'a*
  assumes *partial-equivalence-rel-left*: *partial-equivalence-rel L*
  and *partial-equivalence-rel-right*: *partial-equivalence-rel R*
  and *transport-bijection*: *transport-bijection L R l r*
begin

sublocale *tpre-bij?* : *transport-preorder-on-in-field-bijection L R l r*
  rewrites *partial-equivalence-rel L* ≡ *True*
  and *partial-equivalence-rel R* ≡ *True*
  and ⋀*P*. (*True* ⟹ *P*) ≡ *Trueprop P*
  ⟨*proof*⟩

lemmas *partial-equivalence-rel-equivalence* = *partial-equivalence-rel-equivalenceI*

end

locale *transport-eq-restrict-bijection* =
  fixes $P :: 'a \Rightarrow bool$
  and $Q :: 'b \Rightarrow bool$
  and $l :: 'a \Rightarrow 'b$
  and $r :: 'b \Rightarrow 'a$
  assumes *bijection-on-in-field*:
    bijection-on (*in-field* $((=_P) :: 'a \Rightarrow -)$) (*in-field* $((=_Q) :: 'b \Rightarrow -)$) *l r*
begin

interpretation *transport* $(=_P)$ $(=_Q)$ *l r* ⟨*proof*⟩

sublocale *tper-bij?* : *transport-partial-equivalence-rel-bijection* $(=_P)$ $(=_Q)$ *l r*
  ⟨*proof*⟩

lemma *left-Galois-eq-Galois-eq-eq-restrict*: $({}_L\lessapprox) = (\textit{galois-rel.Galois}\ (=)\ (=)\ r)\restriction_P\restriction_Q$
  ⟨*proof*⟩

end

locale *transport-eq-bijection* =
  fixes $l :: 'a \Rightarrow 'b$
  and $r :: 'b \Rightarrow 'a$
  assumes *bijection-on-in-field*:
    bijection-on (*in-field* $((=) :: 'a \Rightarrow -)$) (*in-field* $((=) :: 'b \Rightarrow -)$) *l r*
begin

sublocale *teq-restr-bij?* : *transport-eq-restrict-bijection* $\top$ $\top$ *l r*
  rewrites $(=_\top :: 'a \Rightarrow bool) = ((=) :: 'a \Rightarrow -)$
  and $(=_\top :: 'b \Rightarrow bool) = ((=) :: 'b \Rightarrow -)$
  ⟨*proof*⟩

end

lemma *mono-wrt-rel-if-connected-on-if-asymmetric-if-mono-if-inverse-on*:
  assumes *conn*: *connected-on* (*in-field L*) *L*
  and *asym*: *asymmetric R*
  and *monol*: $(L \Rightarrow R)$ *l*
  and *monor*: (*in-field R* ⇒ *in-field L*) *r*
  and *inv*: *inverse-on* (*in-field R*) *r l*
  shows $(R \Rightarrow L)$ *r*
⟨*proof*⟩

context *galois*
begin

lemma *transport-bijection-if-asymmetric-if-connected-on-if-mono-if-bijection-on*:

assumes *bijection-on* (*in-field* ($\leq_L$)) (*in-field* ($\leq_R$)) *l r*
and (($\leq_L$) $\Rightarrow$ ($\leq_R$)) *l*
and *connected-on* (*in-field* ($\leq_L$)) ($\leq_L$)
and *asymmetric* ($\leq_R$)
shows *transport-bijection* ($\leq_L$) ($\leq_R$) *l r*
⟨*proof*⟩

lemma *inverse-on-if-connected-on-if-asymmetric-if-galois-connection*:
assumes *gconn*: (($\leq_L$) $\dashv$ ($\leq_R$)) *l r*
and *conn*: *connected-on* (*in-field* ($\leq_L$)) ($\leq_L$)
and *asym*: *asymmetric* ($\leq_R$)
shows *inverse-on* (*in-field* ($\leq_L$)) *l r*
⟨*proof*⟩

interpretation *flip* : *galois R L r l* ⟨*proof*⟩

corollary *bijection-on-if-connected-on-if-asymmetric-if-galois-equivalence*:
assumes (($\leq_L$) $\equiv_G$ ($\leq_R$)) *l r*
and *asymmetric* ($\leq_L$) *asymmetric* ($\leq_R$)
and *connected-on* (*in-field* ($\leq_L$)) ($\leq_L$) *connected-on* (*in-field* ($\leq_R$)) ($\leq_R$)
shows *bijection-on* (*in-field* ($\leq_L$)) (*in-field* ($\leq_R$)) *l r*
⟨*proof*⟩

end

end

theory *Order-Isomorphisms*
imports
Transport-Bijections
begin

consts *order-isomorphism-on* :: $'a \Rightarrow 'b \Rightarrow 'c \Rightarrow 'd \Rightarrow 'e \Rightarrow 'f \Rightarrow$ *bool*

definition *order-isomorphism-on-pred* :: ($'a \Rightarrow$ *bool*) $\Rightarrow$ ($'b \Rightarrow$ *bool*) $\Rightarrow$
($'a \Rightarrow 'a \Rightarrow$ *bool*) $\Rightarrow$ ($'b \Rightarrow 'b \Rightarrow$ *bool*) $\Rightarrow$ ($'a \Rightarrow 'b$) $\Rightarrow$ ($'b \Rightarrow 'a$) $\Rightarrow$ *bool* **where**
order-isomorphism-on-pred P Q L R l r $\equiv$ *bijection-on P Q l r* $\wedge$ ($\forall x\ y : P.\ L\ x\ y \longleftrightarrow R\ (l\ x)\ (l\ y)$)
adhoc-overloading *order-isomorphism-on* $\rightleftharpoons$ *order-isomorphism-on-pred*

context *order-functors*
begin

lemma *order-isomorphism-onI* [*intro*]:
assumes *bij*: *bijection-on P Q l r*
and *monol*: (($\leq_L$)$\uparrow_P \Rightarrow$ ($\leq_R$)) *l*
and *monor*: (($\leq_R$)$\uparrow_Q \Rightarrow$ ($\leq_L$)) *r*

shows *order-isomorphism-on P Q* $(\leq_L)$ $(\leq_R)$ *l r*
⟨*proof*⟩

lemma *order-isomorphism-onE*:
assumes *order-isomorphism-on P Q* $(\leq_L)$ $(\leq_R)$ *l r*
obtains *bijection-on P Q l r* $\bigwedge x\ y.\ P\ x \Longrightarrow P\ y \Longrightarrow L\ x\ y \longleftrightarrow R\ (l\ x)\ (l\ y)$
⟨*proof*⟩

context
notes *order-isomorphism-onE*[*elim*!]
begin

lemma *right-rel-right-if-right-rel-if-preds-if-order-isomorphism-on*:
assumes *order-isomorphism-on P Q* $(\leq_L)$ $(\leq_R)$ *l r*
and *Q y Q* y'
and $y \leq_R y'$
shows *r y* $\leq_L$ *r* y'
⟨*proof*⟩

lemma *order-isomorphism-on-right-left-if-order-isomorphism-on-left-right*:
assumes *order-isomorphism-on P Q* $(\leq_L)$ $(\leq_R)$ *l r*
shows *order-isomorphism-on Q P* $(\leq_R)$ $(\leq_L)$ *r l*
⟨*proof*⟩

lemma *mono-wrt-restrict-restrict-left-if-order-isomorphism-on*:
assumes *order-isomorphism-on P Q* $(\leq_L)$ $(\leq_R)$ *l r*
shows $((\leq_L)\!\upharpoonright_P \Rightarrow (\leq_R)\!\upharpoonright_Q)$ *l*
⟨*proof*⟩

end
end

context *galois*
begin

interpretation *flip* : *galois R L r l* ⟨*proof*⟩

lemma *order-isomorphism-onE* [*elim*]:
assumes *order-isomorphism-on P Q* $(\leq_L)$ $(\leq_R)$ *l r*
obtains *bijection-on P Q l r* $((\leq_L)\!\upharpoonright_P \equiv_G (\leq_R)\!\upharpoonright_Q)$ *l r*
⟨*proof*⟩

lemma *order-isomorphism-on-in-field-if-connected-on-if-asymmetric-if-galois-equivalence*:
assumes $((\leq_L) \equiv_G (\leq_R))$ *l r*
and *asymmetric* $(\leq_L)$ *asymmetric* $(\leq_R)$
and *connected-on* (*in-field* $(\leq_L)$) $(\leq_L)$ *connected-on* (*in-field* $(\leq_R)$) $(\leq_R)$
shows *order-isomorphism-on* (*in-field* $(\leq_L)$) (*in-field* $(\leq_R)$) $(\leq_L)$ $(\leq_R)$ *l r*
⟨*proof*⟩

end

context *order-functors*
begin

interpretation *of* : *order-functors A B a b* **for** *A B a b* ⟨*proof*⟩

lemma *order-isomorphism-on-compI*:
  assumes *order-isomorphism-on P Q L M l1 r1*
  and *order-isomorphism-on Q U M R l2 r2*
  shows *order-isomorphism-on P U L R* (*l2* ∘ *l1*) (*r1* ∘ *r2*)
⟨*proof*⟩

lemma *order-isomorphism-on-self-id*: *order-isomorphism-on P P R R id id*
  ⟨*proof*⟩

end

consts *order-isomorphic-on* :: *'a* ⇒ *'b* ⇒ *'c* ⇒ *'d* ⇒ *bool*

definition *order-isomorphic-on-pred* ::
  (*'a* ⇒ *bool*) ⇒ (*'b* ⇒ *bool*) ⇒ (*'a* ⇒ *'a* ⇒ *bool*) ⇒ (*'b* ⇒ *'b* ⇒ *bool*) ⇒ *bool*
where
  order-isomorphic-on-pred P Q L R ≡ ∃ *l r*. *order-isomorphism-on P Q L R l r*
adhoc-overloading *order-isomorphic-on* ⇌ *order-isomorphic-on-pred*

lemma *order-isomorphic-onI* [*intro*]:
  assumes *order-isomorphism-on P Q L R l r*
  shows *order-isomorphic-on P Q L R*
  ⟨*proof*⟩

lemma *order-isomorphic-onE* [*elim*]:
  assumes *order-isomorphic-on P Q L R*
  obtains *l r* **where** *order-isomorphism-on P Q L R l r*
  ⟨*proof*⟩

lemma *order-isomorphic-on-if-order-isomorphic-on*:
  assumes *order-isomorphic-on P Q L R*
  shows *order-isomorphic-on Q P R L*
  ⟨*proof*⟩

lemma *order-isomorphic-on-trans*:
  assumes *order-isomorphic-on P Q L M*
  and *order-isomorphic-on Q U M R*
  shows *order-isomorphic-on P U L R*
  ⟨*proof*⟩

lemma *order-isomorphic-on-self*: *order-isomorphic-on P P R R*
  ⟨*proof*⟩

end

theory *Order-Functors*
imports
Order-Functors-Base
Order-Equivalences
Order-Isomorphisms
begin

Summary Functors between orders aka. order-homomorphisms aka. monotone functions.

end

2.3.1 Strict Partial Orders

theory *Strict-Partial-Orders*
imports
Binary-Relations-Asymmetric
Binary-Relations-Transitive
begin

definition *strict-partial-order-on* ≡ *transitive-on* ⊓ *asymmetric-on*

lemma *strict-partial-order-onI* [*intro*]:
assumes *transitive-on P R*
and *asymmetric-on P R*
shows *strict-partial-order-on P R*
⟨*proof*⟩

lemma *strict-partial-order-onE* [*elim*]:
assumes *strict-partial-order-on P R*
obtains *transitive-on P R asymmetric-on P R*
⟨*proof*⟩

lemma *transitive-if-strict-partial-order-on-in-field*:
assumes *strict-partial-order-on* (*in-field R*) *R*
shows *transitive R*
⟨*proof*⟩

lemma *asymmetric-if-strict-partial-order-on-in-field*:
assumes *strict-partial-order-on* (*in-field R*) *R*
shows *asymmetric R*
⟨*proof*⟩

lemma *antimono-strict-partial-order-on*:
antimono (*strict-partial-order-on* :: ('*a* ⇒ *bool*) ⇒ ('*a* ⇒ '*a* ⇒ *bool*) ⇒ *bool*)
⟨*proof*⟩

lemma *strict-partial-order-on-rel-map-if-mono-wrt-pred-if-strict-partial-order-on*:
fixes $P :: 'a \Rightarrow bool$ **and** $R :: 'a \Rightarrow 'a \Rightarrow bool$
assumes *strict-partial-order-on P R*
and $(Q \Rightarrow P)$ *f*
shows *strict-partial-order-on Q* (*rel-map f R*)
⟨*proof*⟩

consts *strict-partial-order* :: $'a \Rightarrow bool$

overloading
strict-partial-order ≡ *strict-partial-order* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
begin
definition (*strict-partial-order* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$) ≡ *strict-partial-order-on* ($\top$ :: $'a \Rightarrow bool$)
end

lemma *strict-partial-order-eq-strict-partial-order-on*:
(*strict-partial-order* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$) = *strict-partial-order-on* ($\top$ :: $'a \Rightarrow bool$)
⟨*proof*⟩

lemma *strict-partial-order-eq-strict-partial-order-on-uhint* [*uhint*]:
assumes $P \equiv \top :: 'a \Rightarrow bool$
shows (*strict-partial-order* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$) ≡ *strict-partial-order-on P*
⟨*proof*⟩

context
fixes $R :: 'a \Rightarrow 'a \Rightarrow bool$
begin

lemma *strict-partial-orderI* [*intro*]:
assumes *transitive R*
and *asymmetric R*
shows *strict-partial-order R*
⟨*proof*⟩

lemma *strict-partial-orderE* [*elim*]:
assumes *strict-partial-order R*
obtains *transitive R asymmetric R*
⟨*proof*⟩

lemma *strict-partial-order-on-if-strict-partial-order*:
fixes $P :: 'a \Rightarrow bool$
assumes *strict-partial-order R*
shows *strict-partial-order-on P R*
⟨*proof*⟩

end

end

2.3.2 Strict Linear Orders

theory *Strict-Linear-Orders*
imports
Binary-Relations-Connected
Strict-Partial-Orders
begin

definition *strict-linear-order-on* ≡ *strict-partial-order-on* ⊓ *connected-on*

lemma *strict-linear-order-onI* [*intro*]:
assumes *strict-partial-order-on P R connected-on P R*
shows *strict-linear-order-on P R*
⟨*proof*⟩

lemma *strict-linear-order-onE* [*elim*]:
assumes *strict-linear-order-on P R*
obtains *strict-partial-order-on P R connected-on P R*
⟨*proof*⟩

lemma *antimono-strict-linear-order-on*:
antimono (*strict-linear-order-on* :: ('*a* ⇒ *bool*) ⇒ ('*a* ⇒ '*a* ⇒ *bool*) ⇒ *bool*)
⟨*proof*⟩

lemma *strict-linear-order-on-rel-map-if-injective-on-if-mono-wrt-pred-if-strict-linear-order-on*:
fixes *P* :: '*a* ⇒ *bool* **and** *R* :: '*a* ⇒ '*a* ⇒ *bool*
assumes *strict-linear-order-on P R*
and (*Q* ⇒ *P*) *f*
and *injective-on Q f*
shows *strict-linear-order-on Q* (*rel-map f R*)
⟨*proof*⟩

consts *strict-linear-order* :: '*a* ⇒ *bool*

overloading
strict-linear-order ≡ *strict-linear-order* :: ('*a* ⇒ '*a* ⇒ *bool*) ⇒ *bool*
begin
definition (*strict-linear-order* :: ('*a* ⇒ '*a* ⇒ *bool*) ⇒ *bool*) ≡ *strict-linear-order-on*
(⊤ :: '*a* ⇒ *bool*)
end

lemma *strict-linear-order-eq-strict-linear-order-on*:
(*strict-linear-order* :: ('*a* ⇒ '*a* ⇒ *bool*) ⇒ *bool*) = *strict-linear-order-on* (⊤ :: '*a*
⇒ *bool*)

⟨*proof*⟩

lemma *strict-linear-order-eq-strict-linear-order-on-uhint* [*uhint*]:
assumes $P \equiv \top :: 'a \Rightarrow bool$
shows (*strict-linear-order* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$) $\equiv$ *strict-linear-order-on P*
⟨*proof*⟩

context
fixes $R :: 'a \Rightarrow 'a \Rightarrow bool$
begin

lemma *strict-linear-orderI* [*intro*]:
assumes *strict-partial-order R*
and *connected R*
shows *strict-linear-order R*
⟨*proof*⟩

lemma *strict-linear-orderE* [*elim*]:
assumes *strict-linear-order R*
obtains *strict-partial-order R connected R*
⟨*proof*⟩

lemma *strict-linear-order-on-if-strict-linear-order*:
fixes $P :: 'a \Rightarrow bool$
assumes *strict-linear-order R*
shows *strict-linear-order-on P R*
⟨*proof*⟩

end

end

Well-Orders

theory *Wellorders*
imports
Binary-Relations-Wellfounded
Strict-Linear-Orders
begin

definition *wellorder-on* $\equiv$ *strict-linear-order-on* $\sqcap$ *wellfounded-on*

lemma *wellorder-onI* [*intro*]:
assumes *strict-linear-order-on P R wellfounded-on P R*
shows *wellorder-on P R*
⟨*proof*⟩

lemma *wellorder-onE* [*elim*]:
assumes *wellorder-on P R*
obtains *strict-linear-order-on P R wellfounded-on P R*
⟨*proof*⟩

lemma *antimono-wellorder-on*:
antimono (*wellorder-on* :: (*'a* ⇒ *bool*) ⇒ (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*)
⟨*proof*⟩

lemma *wellorder-on-rel-map-if-injective-on-if-mono-wrt-pred-if-wellorder-on*:
fixes *P* :: *'a* ⇒ *bool*
assumes *wellorder-on P R* (*Q* ⇒ *P*) *f injective-on Q f*
shows *wellorder-on Q* (*rel-map f R*)
⟨*proof*⟩

consts *wellorder* :: *'a* ⇒ *bool*

overloading
wellorder ≡ *wellorder* :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*
begin
definition (*wellorder* :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*) ≡ *wellorder-on* (⊤ :: *'a* ⇒ *bool*)
end

lemma *wellorder-eq-wellorder-on*:
(*wellorder* :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*) = *wellorder-on* (⊤ :: *'a* ⇒ *bool*)
⟨*proof*⟩

lemma *wellorder-eq-wellorder-on-uhint* [*uhint*]:
assumes *P* ≡ ⊤ :: *'a* ⇒ *bool*
shows (*wellorder* :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*) ≡ *wellorder-on P*
⟨*proof*⟩

context
fixes *R* :: *'a* ⇒ *'a* ⇒ *bool*
begin

lemma *wellorderI* [*intro*]:
assumes *strict-linear-order R*
and *wellfounded R*
shows *wellorder R*
⟨*proof*⟩

lemma *wellorderE* [*elim*]:
assumes *wellorder R*
obtains *strict-linear-order R wellfounded R*
⟨*proof*⟩

lemma *wellorder-on-if-wellorder*:
fixes $P :: 'a \Rightarrow bool$
assumes *wellorder R*
shows *wellorder-on P R*
⟨*proof*⟩

end

end

2.4 Orders

theory *Orders*
imports
Equivalence-Relations
Linear-Orders
Order-Functions
Order-Functors
Partial-Equivalence-Relations
Partial-Orders
Preorders
Strict-Linear-Orders
Strict-Partial-Orders
Wellorders
begin

Summary Basic order-theoretic concepts.

end

2.5 Predicates

theory *Predicates*
imports
Bounded-Definite-Description
Bounded-Quantifiers
Predicate-Functions
Predicates-Lattice
Predicates-Order
begin

Summary Basic concepts on predicates.

end

2.6 HOL-Basics

theory *HOL-Basics*
imports
LBinary-Relations
LFunctions
Galois
Orders
Predicates
begin

Summary Library on top of HOL axioms, as required for Transport [2]. Requires *only* the HOL axioms, nothing else. Includes:

1. Basic concepts on binary relations, relativised properties, and restricted equalities e.g. *left-total-on* and *eq-restrict*.
2. Basic concepts on functions, relativised properties, and relators, e.g. *injective-on* and *dep-mono-wrt*.
3. Basic concepts on orders and relativised order-theoretic properties, e.g. *partial-equivalence-rel-on*.
4. Galois connections, Galois equivalences, order equivalences, and other related concepts on order functors, e.g. *galois.galois-equivalence*.
5. Basic concepts on predicates.
6. Syntax bundles for HOL `HOL_Syntax_Bundles`.
7. Alignments for concepts that have counterparts in the HOL library - see `HOL_Alignments`.

end

theory *HOL-Mem-Of*
imports
HOL.Set
ML-Unification.ML-Unification-HOL-Setup
begin

definition *mem-of* $A\ x \equiv x \in A$
lemma *mem-of-eq*: *mem-of* $= (\lambda A\ x.\ x \in A)$ ⟨*proof*⟩
lemma *mem-of-iff* [*iff*]: *mem-of* $A\ x \longleftrightarrow x \in A$ ⟨*proof*⟩

lemma *mem-of-eq-mem-uhint* [*uhint*]:
assumes $x \equiv x'$
and $A \equiv A'$
shows *mem-of* $A\ x \equiv x' \in A'$

⟨*proof*⟩

lemma *mem-of-UNIV-eq-top* [*simp*]: *mem-of UNIV* = ⊤
⟨*proof*⟩

lemma *mem-of-empty-eq-bot* [*simp*]: *mem-of* {} = ⊥
⟨*proof*⟩

end

2.6.1 Alignment With Binary Relation Definitions from HOL.Main

theory *HOL-Alignment-Binary-Relations*
imports
Main
HOL-Mem-Of
LBinary-Relations
begin

unbundle *no relcomp-syntax* **and** *no converse-syntax*

named-theorems *HOL-bin-rel-alignment*

Properties

Antisymmetric **overloading**
antisymmetric-on-set ≡ *antisymmetric-on* :: *'a set* ⇒ (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*
begin
definition *antisymmetric-on-set* (*S* :: *'a set*) :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ - ≡
antisymmetric-on (*mem-of S*)
end

lemma *antisymmetric-on-set-eq-antisymmetric-on-pred* [*simp*]:
(*antisymmetric-on* (*S* :: *'a set*) :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*) = *antisymmetric-on* (*mem-of S*)
⟨*proof*⟩

lemma *antisymmetric-on-set-eq-antisymmetric-on-pred-uhint* [*uhint*]:
assumes *P* ≡ *mem-of S*
shows *antisymmetric-on* (*S* :: *'a set*) :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool* ≡ *antisymmetric-on P*
⟨*proof*⟩

lemma *antisymmetric-on-set-iff-antisymmetric-on-pred* [*iff*]:
antisymmetric-on (*S* :: *'a set*) (*R* :: *'a* ⇒ *'a* ⇒ *bool*) ⟷ *antisymmetric-on* (*mem-of S*) *R*
⟨*proof*⟩

lemma *antisymp-on-eq-antisymmetric-on* [*HOL-bin-rel-alignment*]:
antisymp-on = *antisymmetric-on*
⟨*proof*⟩

lemma *antisymp-eq-antisymmetric* [*HOL-bin-rel-alignment*]:
antisymp = *antisymmetric*
⟨*proof*⟩

Asymmetric overloading
asymmetric-on-set ≡ *asymmetric-on* :: *'a set* ⇒ (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*
begin
definition *asymmetric-on-set* (*S* :: *'a set*) :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ - ≡ *asymmetric-on* (*mem-of S*)
end

lemma *asymmetric-on-set-eq-asymmetric-on-pred* [*simp*]:
(*asymmetric-on* (*S* :: *'a set*) :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*) = *asymmetric-on* (*mem-of S*)
⟨*proof*⟩

lemma *asymmetric-on-set-eq-asymmetric-on-pred-uhint* [*uhint*]:
assumes *P* ≡ *mem-of S*
shows *asymmetric-on* (*S* :: *'a set*) :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool* ≡ *asymmetric-on P*
⟨*proof*⟩

lemma *asymmetric-on-set-iff-asymmetric-on-pred* [*iff*]:
asymmetric-on (*S* :: *'a set*) (*R* :: *'a* ⇒ *'a* ⇒ *bool*) ⟷ *asymmetric-on* (*mem-of S*) *R*
⟨*proof*⟩

lemma *asymp-on-eq-asymmetric-on* [*HOL-bin-rel-alignment*]: *asymp-on* = *asymmetric-on*
⟨*proof*⟩

lemma *asymp-eq-asymmetric* [*HOL-bin-rel-alignment*]: *asymp* = *asymmetric*
⟨*proof*⟩

Injective overloading
rel-injective-on-set ≡ *rel-injective-on* :: *'a set* ⇒ (*'a* ⇒ *'b* ⇒ *bool*) ⇒ *bool*
rel-injective-at-set ≡ *rel-injective-at* :: *'a set* ⇒ (*'b* ⇒ *'a* ⇒ *bool*) ⇒ *bool*
begin
definition *rel-injective-on-set* (*S* :: *'a set*) :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ - ≡
rel-injective-on (*mem-of S*)
definition *rel-injective-at-set* (*S* :: *'a set*) :: (*'b* ⇒ *'a* ⇒ *bool*) ⇒ - ≡
rel-injective-at (*mem-of S*)
end

lemma *rel-injective-on-set-eq-rel-injective-on-pred* [*simp*]:

(*rel-injective-on* (*S* :: 'a *set*) :: ('a ⇒ 'b ⇒ *bool*) ⇒ *bool*) =
rel-injective-on (*mem-of S*)
⟨*proof*⟩

lemma *rel-injective-on-set-eq-rel-injective-on-pred-uhint* [*uhint*]:
assumes *P* ≡ *mem-of S*
shows *rel-injective-on* (*S* :: 'a *set*) :: ('a ⇒ 'b ⇒ *bool*) ⇒ *bool* ≡ *rel-injective-on* *P*
⟨*proof*⟩

lemma *rel-injective-on-set-iff-rel-injective-on-pred* [*iff*]:
rel-injective-on (*S* :: 'a *set*) (*R* :: 'a ⇒ 'b ⇒ *bool*) ⟷ *rel-injective-on* (*mem-of* *S*) *R*
⟨*proof*⟩

lemma *rel-injective-at-set-eq-rel-injective-at-pred* [*simp*]:
(*rel-injective-at* (*S* :: 'a *set*) :: ('b ⇒ 'a ⇒ *bool*) ⇒ *bool*) =
rel-injective-at (*mem-of S*)
⟨*proof*⟩

lemma *rel-injective-at-set-eq-rel-injective-at-pred-uhint* [*uhint*]:
assumes *P* ≡ *mem-of S*
shows *rel-injective-at* (*S* :: 'a *set*) :: ('b ⇒ 'a ⇒ *bool*) ⇒ *bool* ≡ *rel-injective-at* *P*
⟨*proof*⟩

lemma *rel-injective-at-set-iff-rel-injective-at-pred* [*iff*]:
rel-injective-at (*S* :: 'a *set*) (*R* :: 'b ⇒ 'a ⇒ *bool*) ⟷ *rel-injective-at* (*mem-of* *S*) *R*
⟨*proof*⟩

lemma *left-unique-eq-rel-injective* [*HOL-bin-rel-alignment*]:
left-unique = *rel-injective*
⟨*proof*⟩

Irreflexive overloading

irreflexive-on-set ≡ *irreflexive-on* :: 'a *set* ⇒ ('a ⇒ 'a ⇒ *bool*) ⇒ *bool*
begin
definition *irreflexive-on-set* (*S* :: 'a *set*) :: ('a ⇒ 'a ⇒ *bool*) ⇒ *bool* ≡
irreflexive-on (*mem-of S*)
end

lemma *irreflexive-on-set-eq-irreflexive-on-pred* [*simp*]:
(*irreflexive-on* (*S* :: 'a *set*) :: ('a ⇒ 'a ⇒ *bool*) ⇒ *bool*) =
irreflexive-on (*mem-of S*)
⟨*proof*⟩

lemma *irreflexive-on-set-eq-irreflexive-on-pred-uhint* [*uhint*]:
assumes *P* ≡ *mem-of S*

shows *irreflexive-on* (*S* :: *'a set*) :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool* ≡ *irreflexive-on P*
⟨*proof*⟩

lemma *irreflexive-on-set-iff-irreflexive-on-pred* [*iff*]:
irreflexive-on (*S* :: *'a set*) (*R* :: *'a* ⇒ *'a* ⇒ *bool*) ⟷ *irreflexive-on* (*mem-of S*) *R*
⟨*proof*⟩

lemma *irreflp-on-eq-irreflexive-on* [*HOL-bin-rel-alignment*]: *irreflp-on* = *irreflexive-on*
⟨*proof*⟩

lemma *irreflp-eq-irreflexive* [*HOL-bin-rel-alignment*]: *irreflp* = *irreflexive*
⟨*proof*⟩

Left-Total **overloading**
left-total-on-set ≡ *left-total-on* :: *'a set* ⇒ (*'a* ⇒ *'b* ⇒ *bool*) ⇒ *bool*
begin
definition *left-total-on-set* (*S* :: *'a set*) :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ - ≡
left-total-on (*mem-of S*)
end

lemma *left-total-on-set-eq-left-total-on-pred* [*simp*]:
(*left-total-on* (*S* :: *'a set*) :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ *bool*) =
left-total-on (*mem-of S*)
⟨*proof*⟩

lemma *left-total-on-set-eq-left-total-on-pred-uhint* [*uhint*]:
assumes *P* ≡ *mem-of S*
shows *left-total-on* (*S* :: *'a set*) :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ *bool* ≡ *left-total-on P*
⟨*proof*⟩

lemma *left-total-on-set-iff-left-total-on-pred* [*iff*]:
left-total-on (*S* :: *'a set*) (*R* :: *'a* ⇒ *'b* ⇒ *bool*) ⟷ *left-total-on* (*mem-of S*) *R*
⟨*proof*⟩

lemma *Transfer-left-total-eq-left-total* [*HOL-bin-rel-alignment*]:
Transfer.left-total = *Binary-Relations-Left-Total.left-total*
⟨*proof*⟩

Reflexive **overloading**
reflexive-on-set ≡ *reflexive-on* :: *'a set* ⇒ (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*
begin
definition *reflexive-on-set* (*S* :: *'a set*) :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ - ≡
reflexive-on (*mem-of S*)
end

lemma *reflexive-on-set-eq-reflexive-on-pred* [*simp*]:
(*reflexive-on* (*S* :: *'a set*) :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*) = *reflexive-on* (*mem-of S*)

⟨*proof*⟩

lemma *reflexive-on-set-eq-reflexive-on-pred-uhint* [*uhint*]:
assumes $P \equiv$ *mem-of* S
shows *reflexive-on* $(S :: 'a\ set) :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool \equiv$ *reflexive-on* P
⟨*proof*⟩

lemma *reflexive-on-set-iff-reflexive-on-pred* [*iff*]:
reflexive-on $(S :: 'a\ set)\ (R :: 'a \Rightarrow 'a \Rightarrow bool) \longleftrightarrow$ *reflexive-on* (*mem-of* S) R
⟨*proof*⟩

lemma *reflp-on-eq-reflexive-on* [*HOL-bin-rel-alignment*]:
reflp-on = *reflexive-on*
⟨*proof*⟩

lemma *reflp-eq-reflexive* [*HOL-bin-rel-alignment*]: *reflp* = *reflexive*
⟨*proof*⟩

Right-Unique **overloading**

right-unique-on-set $\equiv$ *right-unique-on* $:: 'a\ set \Rightarrow ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool$
right-unique-at-set $\equiv$ *right-unique-at* $:: 'a\ set \Rightarrow ('b \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
begin
definition *right-unique-on-set* $(S :: 'a\ set) :: ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow$ - $\equiv$
right-unique-on (*mem-of* S)
definition *right-unique-at-set* $(S :: 'a\ set) :: ('b \Rightarrow 'a \Rightarrow bool) \Rightarrow$ - $\equiv$
right-unique-at (*mem-of* S)
end

lemma *right-unique-on-set-eq-right-unique-on-pred* [*simp*]:
(*right-unique-on* $(S :: 'a\ set) :: ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool$) = *right-unique-on* (*mem-of* S)
⟨*proof*⟩

lemma *right-unique-on-set-eq-right-unique-on-pred-uhint* [*uhint*]:
assumes $P \equiv$ *mem-of* S
shows *right-unique-on* $(S :: 'a\ set) :: ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool \equiv$ *right-unique-on* P
⟨*proof*⟩

lemma *right-unique-on-set-iff-right-unique-on-pred* [*iff*]:
right-unique-on $(S :: 'a\ set)\ (R :: 'a \Rightarrow 'b \Rightarrow bool) \longleftrightarrow$ *right-unique-on* (*mem-of* S) R
⟨*proof*⟩

lemma *right-unique-at-set-eq-right-unique-at-pred* [*simp*]:
(*right-unique-at* $(S :: 'a\ set) :: ('b \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$) =
right-unique-at (*mem-of* S)
⟨*proof*⟩

lemma *right-unique-at-set-iff-right-unique-at-pred* [*iff*]:
right-unique-at (*S* :: *'a set*) (*R* :: *'b* ⇒ *'a* ⇒ *bool*) ⟷ *right-unique-at* (*mem-of S*) *R*
⟨*proof*⟩

lemma *Transfer-right-unique-eq-right-unique* [*HOL-bin-rel-alignment*]:
Transfer.right-unique = *Binary-Relations-Right-Unique.right-unique*
⟨*proof*⟩

Surjective overloading
rel-surjective-at-set ≡ *rel-surjective-at* :: *'a set* ⇒ (*'b* ⇒ *'a* ⇒ *bool*) ⇒ *bool*
begin
definition *rel-surjective-at-set* (*S* :: *'a set*) :: (*'b* ⇒ *'a* ⇒ *bool*) ⇒ - ≡
rel-surjective-at (*mem-of S*)
end

lemma *rel-surjective-at-set-eq-rel-surjective-at-pred* [*simp*]:
(*rel-surjective-at* (*S* :: *'a set*) :: (*'b* ⇒ *'a* ⇒ *bool*) ⇒ *bool*) = *rel-surjective-at* (*mem-of S*)
⟨*proof*⟩

lemma *rel-surjective-at-set-eq-rel-surjective-at-pred-uhint* [*uhint*]:
assumes *P* ≡ *mem-of S*
shows *rel-surjective-at* (*S* :: *'a set*) :: (*'b* ⇒ *'a* ⇒ *bool*) ⇒ *bool* ≡ *rel-surjective-at P*
⟨*proof*⟩

lemma *rel-surjective-at-set-iff-rel-surjective-at-pred* [*iff*]:
rel-surjective-at (*S* :: *'a set*) (*R* :: *'b* ⇒ *'a* ⇒ *bool*) ⟷ *rel-surjective-at* (*mem-of S*) *R*
⟨*proof*⟩

lemma *Transfer-right-total-eq-rel-surjective* [*HOL-bin-rel-alignment*]:
Transfer.right-total = *rel-surjective*
⟨*proof*⟩

Symmetric overloading
symmetric-on-set ≡ *symmetric-on* :: *'a set* ⇒ (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*
begin
definition *symmetric-on-set* (*S* :: *'a set*) :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ - ≡
symmetric-on (*mem-of S*)
end

lemma *symmetric-on-set-eq-symmetric-on-pred* [*simp*]:
(*symmetric-on* (*S* :: *'a set*) :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*) = *symmetric-on* (*mem-of S*)
⟨*proof*⟩

lemma *symmetric-on-set-eq-symmetric-on-pred-uhint* [*uhint*]:

assumes $P \equiv mem\text{-}of\ S$
shows $symmetric\text{-}on\ (S :: 'a\ set) :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool \equiv symmetric\text{-}on\ P$
⟨*proof*⟩

lemma *symmetric-on-set-iff-symmetric-on-pred* [*iff*]:
$symmetric\text{-}on\ (S :: 'a\ set)\ (R :: 'a \Rightarrow 'a \Rightarrow bool) \longleftrightarrow symmetric\text{-}on\ (mem\text{-}of\ S)\ R$
⟨*proof*⟩

lemma *symp-on-eq-symmetric-on* [*HOL-bin-rel-alignment*]: $symp\text{-}on = symmetric\text{-}on$
⟨*proof*⟩

lemma *symp-eq-symmetric* [*HOL-bin-rel-alignment*]: $symp = symmetric$
⟨*proof*⟩

Transitive **overloading**
$transitive\text{-}on\text{-}set \equiv transitive\text{-}on :: 'a\ set \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
begin
definition $transitive\text{-}on\text{-}set\ (S :: 'a\ set) :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow \text{-} \equiv transitive\text{-}on\ (mem\text{-}of\ S)$
end

lemma *transitive-on-set-eq-transitive-on-pred* [*simp*]:
$(transitive\text{-}on\ (S :: 'a\ set) :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool) = transitive\text{-}on\ (mem\text{-}of\ S)$
⟨*proof*⟩

lemma *transitive-on-set-eq-transitive-on-pred-uhint* [*uhint*]:
assumes $P \equiv mem\text{-}of\ S$
shows $transitive\text{-}on\ (S :: 'a\ set) :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool \equiv transitive\text{-}on\ P$
⟨*proof*⟩

lemma *transitive-on-set-iff-transitive-on-pred* [*iff*]:
$transitive\text{-}on\ (S :: 'a\ set)\ (R :: 'a \Rightarrow 'a \Rightarrow bool) \longleftrightarrow transitive\text{-}on\ (mem\text{-}of\ S)\ R$
⟨*proof*⟩

lemma *transp-on-eq-transitive-on* [*HOL-bin-rel-alignment*]: $transp\text{-}on = transitive\text{-}on$
⟨*proof*⟩

lemma *transp-eq-transitive* [*HOL-bin-rel-alignment*]: $transp = transitive$
⟨*proof*⟩

Well-Founded **overloading**
$wellfounded\text{-}on\text{-}set \equiv wellfounded\text{-}on :: 'a\ set \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
begin
definition $wellfounded\text{-}on\text{-}set\ (S :: 'a\ set) :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow \text{-} \equiv wellfounded\text{-}on\ (mem\text{-}of\ S)$
end

lemma *wellfounded-on-set-eq-wellfounded-on-pred* [*simp*]:
(*wellfounded-on* (*S* :: *'a set*) :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*) = *wellfounded-on* (*mem-of S*)
⟨*proof*⟩

lemma *wellfounded-on-set-eq-wellfounded-on-pred-uhint* [*uhint*]:
assumes *P* ≡ *mem-of S*
shows *wellfounded-on* (*S* :: *'a set*) :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool* ≡ *wellfounded-on P*
⟨*proof*⟩

lemma *wellfounded-on-set-iff-wellfounded-on-pred* [*iff*]:
wellfounded-on (*S* :: *'a set*) (*R* :: *'a* ⇒ *'a* ⇒ *bool*) ⟷ *wellfounded-on* (*mem-of S*) *R*
⟨*proof*⟩

lemma *wfp-on-eq-wellfounded-on* [*HOL-bin-rel-alignment*]: *wfp-on* = *wellfounded-on*
⟨*proof*⟩

lemma *wfp-eq-wellfounded* [*HOL-bin-rel-alignment*]: *wfp* = *wellfounded*
⟨*proof*⟩

Bi-Total **lemma** *bi-total-on-set-eq-bi-total-on-pred* [*simp*]:
(*bi-total-on* (*S* :: *'a set*) (*T* :: *'b set*) :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ *bool*) =
bi-total-on (*mem-of S*) (*mem-of T*)
⟨*proof*⟩

lemma *bi-total-on-set-eq-bi-total-on-pred-uhint* [*uhint*]:
assumes *P* ≡ *mem-of S*
and *Q* ≡ *mem-of T*
shows *bi-total-on* (*S* :: *'a set*) (*T* :: *'b set*) :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ *bool* ≡ *bi-total-on P Q*
⟨*proof*⟩

lemma *bi-total-on-set-iff-bi-total-on-pred* [*iff*]:
bi-total-on (*S* :: *'a set*) (*T* :: *'b set*) (*R* :: *'a* ⇒ *'b* ⇒ *bool*) ⟷
bi-total-on (*mem-of S*) (*mem-of T*) *R*
⟨*proof*⟩

lemma *Transfer-bi-total-eq-bi-total* [*HOL-bin-rel-alignment*]:
Transfer.bi-total = *Binary-Relations-Bi-Total.bi-total*
⟨*proof*⟩

Bi-Unique **lemma** *bi-unique-on-set-eq-bi-unique-on-pred* [*simp*]:
(*bi-unique-on* (*S* :: *'a set*) :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ *bool*) = *bi-unique-on* (*mem-of S*)
⟨*proof*⟩

lemma *bi-unique-on-set-eq-bi-unique-on-pred-uhint* [*uhint*]:
assumes $P \equiv$ *mem-of* S
shows *bi-unique-on* $(S :: 'a\ set) :: ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool \equiv$ *bi-unique-on* P
⟨*proof*⟩

lemma *bi-unique-on-set-iff-bi-unique-on-pred* [*iff*]:
bi-unique-on $(S :: 'a\ set)$ $(R :: 'a \Rightarrow 'b \Rightarrow bool) \longleftrightarrow$ *bi-unique-on* (*mem-of* S) R
⟨*proof*⟩

lemma *Transfer-bi-unique-eq-bi-unique* [*HOL-bin-rel-alignment*]:
Transfer.bi-unique = *Binary-Relations-Bi-Unique.bi-unique*
⟨*proof*⟩

Functions **lemma** *Domainp-eq-in-dom* [*HOL-bin-rel-alignment*]: *Domainp* = *in-dom*
⟨*proof*⟩

lemma *Rangep-eq-in-codom* [*HOL-bin-rel-alignment*]: *Rangep* = *in-codom*
⟨*proof*⟩

lemma *relcompp-eq-rel-comp* [*HOL-bin-rel-alignment*]: *relcompp* = *rel-comp*
⟨*proof*⟩

lemma *conversep-eq-rel-inv* [*HOL-bin-rel-alignment*]: *conversep* = *rel-inv*
⟨*proof*⟩

lemma *eq-onp-eq-eq-restrict* [*HOL-bin-rel-alignment*]: *eq-onp* = *rel-restrict-left* (=)
⟨*proof*⟩

definition *rel-restrict-left-set* $(R :: 'a \Rightarrow 'b \Rightarrow bool)$ $(S :: 'a\ set) \equiv R{\restriction}_{mem\text{-}of\ S}$
adhoc-overloading *rel-restrict-left* ⇌ *rel-restrict-left-set*

definition *rel-restrict-right-set* $(R :: 'a \Rightarrow 'b \Rightarrow bool)$ $(S :: 'b\ set) \equiv R{\restriction}_{mem\text{-}of\ S}$
adhoc-overloading *rel-restrict-right* ⇌ *rel-restrict-right-set*

definition *rel-restrict-set* $(R :: 'a \Rightarrow 'a \Rightarrow bool)$ $(S :: 'a\ set) \equiv R{\restriction}_{mem\text{-}of\ S}$
adhoc-overloading *rel-restrict* ⇌ *rel-restrict-set*

lemma *rel-restrict-left-set-eq-restrict-left-pred* [*simp*]:
$R{\restriction}_S = R{\restriction}_{mem\text{-}of\ S}$
⟨*proof*⟩

lemma *rel-restrict-left-set-eq-restrict-left-pred-uhint* [*uhint*]:
assumes $R \equiv R'$
and $P \equiv$ *mem-of* S
shows $R{\restriction}_S \equiv R'{\restriction}_P$
⟨*proof*⟩

lemma *rel-restrict-left-set-iff-restrict-left-pred* [*iff*]: $R{\restriction}_S\ x\ y \longleftrightarrow R{\restriction}_{mem\text{-}of\ S}\ x\ y$

⟨*proof*⟩

lemma *rel-restrict-right-set-eq-restrict-right-pred* [*simp*]:
$R|_S = R|_{\textit{mem-of } S}$
⟨*proof*⟩

lemma *rel-restrict-right-set-eq-restrict-right-pred-uhint* [*uhint*]:
assumes $R \equiv R'$
and $P \equiv \textit{mem-of } S$
shows $R|_S \equiv R'|_P$
⟨*proof*⟩

lemma *rel-restrict-right-set-iff-restrict-right-pred* [*iff*]: $R|_S\ x\ y \longleftrightarrow R|_{\textit{mem-of } S}\ x\ y$
⟨*proof*⟩

lemma *rel-restrict-set-eq-restrict-pred* [*simp*]:
$R\uparrow_S = R\uparrow_{\textit{mem-of } S}$
⟨*proof*⟩

lemma *rel-restrict-set-eq-restrict-pred-uhint* [*uhint*]:
assumes $R \equiv R'$
and $P \equiv \textit{mem-of } S$
shows $R\uparrow_S \equiv R'\uparrow_P$
⟨*proof*⟩

lemma *rel-restrict-set-iff-restrict-pred* [*iff*]: $R\uparrow_S\ x\ y \longleftrightarrow R\uparrow_{\textit{mem-of } S}\ x\ y$
⟨*proof*⟩

end

2.6.2 Function Syntax

theory *HOL-Syntax-Bundles-Functions*
imports *HOL.Fun*
begin

bundle *HOL-function-syntax*
begin
notation *comp* (**infixl** ‹∘› *55*)
end

end

2.6.3 Alignment With Function Definitions from HOL.Main

theory *HOL-Alignment-Functions*
imports
HOL-Alignment-Binary-Relations

HOL-Syntax-Bundles-Functions
LFunctions
begin

unbundle *no HOL-function-syntax*

named-theorems *HOL-fun-alignment*

Functions

Bijection **definition** *bijection-on-set* $(S :: 'a\ set)\ (S' :: 'b\ set) :: ('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow 'a) \Rightarrow bool \equiv$
bijection-on $(mem\text{-}of\ S)\ (mem\text{-}of\ S')$
adhoc-overloading *bijection-on* $\rightleftharpoons$ *bijection-on-set*

lemma *bijection-on-set-eq-bijection-on-pred* [*simp*]:
bijection-on $(S :: 'a\ set)\ (S' :: 'b\ set) = bijection\text{-}on\ (mem\text{-}of\ S)\ (mem\text{-}of\ S')$
⟨*proof*⟩

lemma *bijection-on-set-eq-bijection-on-pred-uhint* [*uhint*]:
assumes $P \equiv mem\text{-}of\ S$
and $Q \equiv mem\text{-}of\ S'$
shows *bijection-on* $S\ S' \equiv bijection\text{-}on\ P\ Q$
⟨*proof*⟩

lemma *bijection-on-set-iff-bijection-on-pred* [*iff*]:
bijection-on $(S :: 'a\ set)\ (S' :: 'b\ set)\ (f :: 'a \Rightarrow 'b)\ (g :: 'b \Rightarrow 'a) \longleftrightarrow$
bijection-on $(mem\text{-}of\ S)\ (mem\text{-}of\ S')\ f\ g$
⟨*proof*⟩

lemma *bij-betw-bijection-onE*:
assumes *bij-betw* $(f :: 'a \Rightarrow 'b)\ S\ S'$
obtains $g :: 'b \Rightarrow 'a$ **where** *bijection-on* $S\ S'\ f\ g$
⟨*proof*⟩

lemma *bij-betw-if-bijection-on*:
assumes *bijection-on* $S\ S'\ (f :: 'a \Rightarrow 'b)\ (g :: 'b \Rightarrow 'a)$
shows *bij-betw* $f\ S\ S'$
⟨*proof*⟩

corollary *bij-betw-iff-ex-bijection-on* [*HOL-fun-alignment*]:
bij-betw $(f :: 'a \Rightarrow 'b)\ S\ S' \longleftrightarrow (\exists (g :: 'b \Rightarrow 'a).\ bijection\text{-}on\ S\ S'\ f\ g)$
⟨*proof*⟩

Injective **overloading**
injective-on-set $\equiv$ *injective-on* $:: 'a\ set \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool$
begin
definition *injective-on-set* $(S :: 'a\ set) :: ('a \Rightarrow 'b) \Rightarrow bool \equiv$
injective-on $(mem\text{-}of\ S)$

end

lemma *injective-on-set-eq-injective-on-pred* [*simp*]:
(*injective-on* (*S* :: *'a set*) :: (*'a* ⇒ *'b*) ⇒ -) = *injective-on* (*mem-of S*)
⟨*proof*⟩

lemma *injective-on-set-eq-injective-on-pred-uhint* [*uhint*]:
assumes *P* ≡ *mem-of S*
shows *injective-on* (*S* :: *'a set*) :: (*'a* ⇒ *'b*) ⇒ - ≡ *injective-on P*
⟨*proof*⟩

lemma *injective-on-set-iff-injective-on-pred* [*iff*]:
injective-on (*S* :: *'a set*) (*f* :: *'a* ⇒ *'b*) ⟷ *injective-on* (*mem-of S*) *f*
⟨*proof*⟩

lemma *inj-on-iff-injective-on* [*HOL-fun-alignment*]: *inj-on f P* ⟷ *injective-on P*
f
⟨*proof*⟩

lemma *inj-eq-injective* [*HOL-fun-alignment*]: *inj* = *injective*
⟨*proof*⟩

Inverse **overloading**
inverse-on-set ≡ *inverse-on* :: *'a set* ⇒ (*'a* ⇒ *'b*) ⇒ (*'b* ⇒ *'a*) ⇒ *bool*
begin
definition *inverse-on-set* (*S* :: *'a set*) :: (*'a* ⇒ *'b*) ⇒ (*'b* ⇒ *'a*) ⇒ *bool* ≡
inverse-on (*mem-of S*)
end

lemma *inverse-on-set-eq-inverse-on-pred* [*simp*]:
(*inverse-on* (*S* :: *'a set*) :: (*'a* ⇒ *'b*) ⇒ (*'b* ⇒ *'a*) ⇒ -) = *inverse-on* (*mem-of S*)
⟨*proof*⟩

lemma *inverse-on-set-eq-inverse-on-pred-uhint* [*uhint*]:
assumes *P* ≡ *mem-of S*
shows *inverse-on* (*S* :: *'a set*) :: (*'a* ⇒ *'b*) ⇒ (*'b* ⇒ *'a*) ⇒ - ≡ *inverse-on P*
⟨*proof*⟩

lemma *inverse-on-set-iff-inverse-on-pred* [*iff*]:
inverse-on (*S* :: *'a set*) (*f* :: *'a* ⇒ *'b*) (*g* :: *'b* ⇒ *'a*) ⟷ *inverse-on* (*mem-of S*)
f g
⟨*proof*⟩

Monotone **lemma** *monotone-on-eq-mono-wrt-rel-restrict-left-right* [*HOL-fun-alignment*]:
monotone-on S R = *mono-wrt-rel R*⌈*S*⌉*S*
⟨*proof*⟩

lemma *monotone-eq-mono-wrt-rel* [*HOL-fun-alignment*]: *monotone* = *mono-wrt-rel*
⟨*proof*⟩

lemma *pred-fun-eq-mono-wrt-pred* [*HOL-fun-alignment*]: *pred-fun* = *mono-wrt-pred*
⟨*proof*⟩

lemma *Fun-mono-eq-mono* [*HOL-fun-alignment*]: *Fun.mono* = *mono*
⟨*proof*⟩

lemma *Fun-antimono-eq-antimono* [*HOL-fun-alignment*]: *Fun.antimono* = *antimono*
⟨*proof*⟩

Surjective **overloading**
surjective-at-set ≡ *surjective-at* :: *'a set* ⇒ (*'b* ⇒ *'a*) ⇒ *bool*
begin
definition *surjective-at-set* (*S* :: *'a set*) :: (*'b* ⇒ *'a*) ⇒ *bool* ≡
surjective-at (*mem-of S*)
end

lemma *surjective-at-set-eq-surjective-at-pred* [*simp*]:
(*surjective-at* (*S* :: *'a set*) :: (*'b* ⇒ *'a*) ⇒ -) = *surjective-at* (*mem-of S*)
⟨*proof*⟩

lemma *surjective-at-set-eq-surjective-at-pred-uhint* [*uhint*]:
assumes *P* ≡ *mem-of S*
shows *surjective-at* (*S* :: *'a set*) :: (*'b* ⇒ *'a*) ⇒ - ≡ *surjective-at P*
⟨*proof*⟩

lemma *surjective-at-set-iff-surjective-at-pred* [*iff*]:
surjective-at (*S* :: *'a set*) (*f* :: *'b* ⇒ *'a*) ⟷ *surjective-at* (*mem-of S*) *f*
⟨*proof*⟩

lemma *surj-eq-surjective* [*HOL-fun-alignment*]: *surj* = *surjective*
⟨*proof*⟩

Functions **lemma** *Fun-id-eq-id* [*HOL-fun-alignment*]: *Fun.id* = *Functions-Base.id*
⟨*proof*⟩

lemma *Fun-comp-eq-comp* [*HOL-fun-alignment*]: *Fun.comp* = *Functions-Base.comp*
⟨*proof*⟩

lemma *map-fun-eq-fun-map* [*HOL-fun-alignment*]: *map-fun* = *fun-map*
⟨*proof*⟩

Relators **lemma** *rel-fun-eq-Fun-Rel-rel* [*HOL-fun-alignment*]: *BNF-Def.rel-fun* = *Fun-Rel*
⟨*proof*⟩

end

2.7 Order Syntax

theory *HOL-Syntax-Bundles-Orders*
imports *HOL.Orderings*
begin

bundle *HOL-order-syntax*
begin
notation
less-eq (‹'(≤')›) **and**
less-eq (‹(‹*notation*=‹*infix* ≤››-/ ≤ -)› [*51*, *51*] *50*) **and**
less (‹'(<')›) **and**
less (‹(‹*notation*=‹*infix* <››-/ < -)› [*51*, *51*] *50*)
notation (*input*) *greater-eq* (**infix** ‹≥› *50*)
notation (*input*) *greater* (**infix** ‹>› *50*)
notation (*ASCII*)
less-eq (‹'(<=')›) **and**
less-eq (‹(‹*notation*=‹*infix* <=››-/ <= -)› [*51*, *51*] *50*)
notation (*input*) *greater-eq* (**infix** ‹>=› *50*)
end

end

2.7.1 Alignment With Order Definitions from HOL

theory *HOL-Alignment-Orders*
imports
HOL−Library.Preorder
HOL-Alignment-Binary-Relations
HOL-Syntax-Bundles-Orders
Orders
begin

named-theorems *HOL-order-alignment*

Functions **definition** *rel R x y* ≡ (*x*, *y*) ∈ *R*
lemma *rel-of-eq* [*simp*]: *rel* = (λ*R x y*. (*x*, *y*) ∈ *R*) ⟨*proof*⟩
lemma *rel-of-iff* [*iff*]: *rel R x y* ⟷ (*x*, *y*) ∈ *R* ⟨*proof*⟩

Bi-Related **overloading**
bi-related-set ≡ *bi-related* :: *'a rel* ⇒ *'a* ⇒ *'a* ⇒ *bool*
begin
definition *bi-related-set* (*S* :: *'a rel*) ≡ *bi-related* (*rel S*) :: *'a* ⇒ *'a* ⇒ *bool*
end

lemma *bi-related-set-eq-bi-related-pred* [*simp*]:
((≡$_{S}$ :: *'a rel*) :: *'a* ⇒ *'a* ⇒ *bool*) = (≡$_{rel\ S}$)
⟨*proof*⟩

lemma *bi-related-set-eq-bi-related-pred-uhint* [*uhint*]:
assumes $R \equiv rel\ S$
shows $(\equiv_{S\ ::\ 'a\ rel}) :: 'a \Rightarrow 'a \Rightarrow bool \equiv (\equiv_R)$
⟨*proof*⟩

lemma *bi-related-set-iff-bi-related-pred* [*iff*]: $(x :: 'a) \equiv_{(S\ ::\ 'a\ rel)} (y :: 'a) \longleftrightarrow x \equiv_{rel\ S} y$
⟨*proof*⟩

lemma (**in** *preorder-equiv*) *equiv-eq-bi-related* [*HOL-order-alignment*]:
$equiv = bi\text{-}related\ (\leq)$
⟨*proof*⟩

Inflationary overloading
$inflationary\text{-}on\text{-}set \equiv inflationary\text{-}on :: 'a\ set \Rightarrow ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool$
begin
definition $inflationary\text{-}on\text{-}set\ (S :: 'a\ set) :: ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool \equiv$
$inflationary\text{-}on\ (mem\text{-}of\ S)$
end

lemma *inflationary-on-set-eq-inflationary-on-pred* [*simp*]:
$(inflationary\text{-}on\ (S :: 'a\ set) :: ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool) =$
$inflationary\text{-}on\ (mem\text{-}of\ S)$
⟨*proof*⟩

lemma *inflationary-on-set-eq-inflationary-on-pred-uhint* [*uhint*]:
assumes $P \equiv mem\text{-}of\ S$
shows $inflationary\text{-}on\ (S :: 'a\ set) :: ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool \equiv inflationary\text{-}on\ P$
⟨*proof*⟩

lemma *inflationary-on-set-iff-inflationary-on-pred* [*iff*]:
$inflationary\text{-}on\ (S :: 'a\ set)\ (R :: 'a \Rightarrow 'b \Rightarrow bool)\ (f :: 'a \Rightarrow 'b) \longleftrightarrow$
$inflationary\text{-}on\ (mem\text{-}of\ S)\ R\ f$
⟨*proof*⟩

Deflationary overloading
$deflationary\text{-}on\text{-}set \equiv deflationary\text{-}on :: 'a\ set \Rightarrow ('b \Rightarrow 'a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool$
begin
definition $deflationary\text{-}on\text{-}set\ (S :: 'a\ set) :: ('b \Rightarrow 'a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool \equiv$
$deflationary\text{-}on\ (mem\text{-}of\ S)$
end

lemma *deflationary-on-set-eq-deflationary-on-pred* [*simp*]:
$(deflationary\text{-}on\ (S :: 'a\ set) :: ('b \Rightarrow 'a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool) = defla-$

tionary-on (*mem-of S*)
⟨*proof*⟩

lemma *deflationary-on-set-eq-deflationary-on-pred-uhint* [*uhint*]:
assumes $P \equiv$ *mem-of S*
shows *deflationary-on* (*S* :: *'a set*) :: (*'b* ⇒ *'a* ⇒ *bool*) ⇒ (*'a* ⇒ *'b*) ⇒ *bool* ≡ *deflationary-on P*
⟨*proof*⟩

lemma *deflationary-on-set-iff-deflationary-on-pred* [*iff*]:
deflationary-on (*S* :: *'a set*) (*R* :: *'b* ⇒ *'a* ⇒ *bool*) (*f* :: *'a* ⇒ *'b*) ⟷ *deflationary-on* (*mem-of S*) *R f*
⟨*proof*⟩

Idempotent **overloading**
idempotent-on-set ≡ *idempotent-on* :: *'a set* ⇒ (*'a* ⇒ *'a* ⇒ *bool*) ⇒ (*'a* ⇒ *'a*) ⇒ *bool*
begin
definition *idempotent-on-set* (*S* :: *'a set*) :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ (*'a* ⇒ *'a*) ⇒ *bool* ≡
idempotent-on (*mem-of S*)
end

lemma *idempotent-on-set-eq-idempotent-on-pred* [*simp*]:
(*idempotent-on* (*S* :: *'a set*) :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ (*'a* ⇒ *'a*) ⇒ *bool*) = *idempotent-on* (*mem-of S*)
⟨*proof*⟩

lemma *idempotent-on-set-iff-idempotent-on-pred* [*iff*]:
idempotent-on (*S* :: *'a set*) (*R* :: *'a* ⇒ *'a* ⇒ *bool*) (*f* :: *'a* ⇒ *'a*) ⟷ *idempotent-on* (*mem-of S*) *R f*
⟨*proof*⟩

Properties

Equivalence Relations **lemma** *equiv-eq-equivalence-rel* [*HOL-order-alignment*]:
equivp = *equivalence-rel*
⟨*proof*⟩

Partial Equivalence Relations **lemma** *part-equiv-eq-partial-equivalence-rel-if-rel* [*HOL-order-alignment*]:
assumes *R x y*
shows *part-equivp R* = *partial-equivalence-rel R*
⟨*proof*⟩

Partial Orders **lemma** (**in** *order*) *partial-order* [*HOL-order-alignment*]: *partial-order* (≤)
⟨*proof*⟩

Preorders **lemma** (**in** *partial-preordering*) *preorder* [*HOL-order-alignment*]: *preorder* ($\leq$)
⟨*proof*⟩

lemma *partial-preordering-eq* [*HOL-order-alignment*]:
partial-preordering = *Preorders.preorder*
⟨*proof*⟩

Linear Orders **lemma** (**in** *linorder*) *linear-order*: *linear-order* ($\leq$)
⟨*proof*⟩

Strict Parital Orders **lemma** (**in** *preordering*) *linear-order*: *strict-partial-order* ($<$)
⟨*proof*⟩

Strict Linear Orders **lemma** (**in** *linorder*) *strict-linear-order*: *strict-linear-order* ($<$)
⟨*proof*⟩

Well-Orders **lemma** (**in** *wellorder*) *wellorder*: *wellorder* ($<$)
⟨*proof*⟩

end

2.7.2 Alignment With Predicate Definitions from HOL

theory *HOL-Alignment-Predicates*
imports
Main
HOL-Mem-Of
Predicates
begin

named-theorems *HOL-predicate-alignment*

Quantifiers **adhoc-overloading** *ball* $\rightleftharpoons$ *Ball*

lemma *Ball-eq-ball-pred* [*HOL-predicate-alignment*]: $\forall_A = \forall_{\textit{mem-of } A}$ ⟨*proof*⟩

lemma *Ball-eq-ball-pred-uhint* [*uhint*]:
assumes $P \equiv$ *mem-of A*
shows $\forall_A = \forall_P$
⟨*proof*⟩

lemma *Ball-iff-ball-pred* [*HOL-predicate-alignment*]: $(\forall x : A.\ Q\ x) \longleftrightarrow (\forall x : \textit{mem-of } A.\ Q\ x)$
⟨*proof*⟩

adhoc-overloading *bex* $\rightleftharpoons$ *Bex*

lemma *Bex-eq-bex-pred* [*HOL-predicate-alignment*]: $\exists_A = \exists_{mem\text{-}of\ A}$ ⟨*proof*⟩

lemma *Bex-eq-bex-pred-uhint* [*uhint*]:
assumes $P \equiv mem\text{-}of\ A$
shows $\exists_A = \exists_P$
⟨*proof*⟩

lemma *Bex-iff-bex-pred* [*HOL-predicate-alignment*]: $(\exists\, x : A.\ Q\ x) \longleftrightarrow (\exists\, x : mem\text{-}of\ A.\ Q\ x)$
⟨*proof*⟩

end

2.8 HOL Alignments

theory *HOL-Alignments*
imports
HOL-Alignment-Binary-Relations
HOL-Alignment-Functions
HOL-Alignment-Orders
HOL-Alignment-Predicates
begin

Summary Alignment of concepts with HOL counterparts

end

2.8.1 Alignment With Order Definitions from HOL-Algebra

theory *HOL-Algebra-Alignment-Orders*
imports
HOL−Algebra.Order
HOL-Alignment-Orders
begin

named-theorems *HOL-Algebra-order-alignment*

context *equivalence*
begin

lemma *reflexive-on-carrier* [*HOL-Algebra-order-alignment*]:
reflexive-on (*carrier S*) (.=)
⟨*proof*⟩

lemma *transitive-on-carrier* [*HOL-Algebra-order-alignment*]:
transitive-on (*carrier S*) (.=)
⟨*proof*⟩

lemma *preorder-on-carrier* [*HOL-Algebra-order-alignment*]:
preorder-on (*carrier S*) (.=)
⟨*proof*⟩

lemma *symmetric-on-carrier* [*HOL-Algebra-order-alignment*]:
symmetric-on (*carrier S*) (.=)
⟨*proof*⟩

lemma *partial-equivalence-rel-on-carrier* [*HOL-Algebra-order-alignment*]:
partial-equivalence-rel-on (*carrier S*) (.=)
⟨*proof*⟩

lemma *equivalence-rel-on-carrier* [*HOL-Algebra-order-alignment*]:
equivalence-rel-on (*carrier S*) (.=)
⟨*proof*⟩

end

lemma *equivalence-iff-equivalence-rel-on-carrier* [*HOL-Algebra-order-alignment*]:
equivalence S ⟷ *equivalence-rel-on* (*carrier S*) ($.=_S$)
⟨*proof*⟩

context *partial-order*
begin

lemma *reflexive-on-carrier* [*HOL-Algebra-order-alignment*]:
reflexive-on (*carrier L*) (⊑)
⟨*proof*⟩

lemma *transitive-on-carrier* [*HOL-Algebra-order-alignment*]:
transitive-on (*carrier L*) (⊑)
⟨*proof*⟩

lemma *preorder-on-carrier* [*HOL-Algebra-order-alignment*]:
preorder-on (*carrier L*) (⊑)
⟨*proof*⟩

lemma *antisymmetric-on-carrier* [*HOL-Algebra-order-alignment*]:
antisymmetric-on (*carrier L*) (⊑)
⟨*proof*⟩

lemma *partial-order-on-carrier* [*HOL-Algebra-order-alignment*]:
partial-order-on (*carrier L*) (⊑)
⟨*proof*⟩

end

end

2.8.2 Alignment With Galois Definitions from HOL-Algebra

theory *HOL-Algebra-Alignment-Galois*
imports
HOL−Algebra.Galois-Connection
HOL-Algebra-Alignment-Orders
Galois
begin

named-theorems *HOL-Algebra-galois-alignment*

context *galois-connection*
begin

context
fixes L R l r
defines $L \equiv (\sqsubseteq_{\mathcal{X}})\restriction_{carrier\ \mathcal{X}}{}^{\uparrow}{}_{carrier\ \mathcal{X}}$ **and** $R \equiv (\sqsubseteq_{\mathcal{Y}})\restriction_{carrier\ \mathcal{Y}}{}^{\uparrow}{}_{carrier\ \mathcal{Y}}$
and $l \equiv \pi^*$ **and** $r \equiv \pi_*$
notes *defs*[*simp*] = *L-def R-def l-def r-def* **and** *rel-restrict-right-eq*[*simp*]
and *rel-restrict-leftI*[*intro*!] *rel-restrict-leftE*[*elim*!]
begin

interpretation *galois* L R l r ⟨*proof*⟩

lemma *mono-wrt-rel-lower* [*HOL-Algebra-galois-alignment*]: $(L \Rightarrow R)\ l$
⟨*proof*⟩

lemma *mono-wrt-rel-upper* [*HOL-Algebra-galois-alignment*]: $(R \Rightarrow L)\ r$
⟨*proof*⟩

lemma *half-galois-prop-left* [*HOL-Algebra-galois-alignment*]: $(L\ {}_h\trianglelefteq\ R)\ l\ r$
⟨*proof*⟩

lemma *half-galois-prop-right* [*HOL-Algebra-galois-alignment*]: $(L \trianglelefteq_h\ R)\ l\ r$
⟨*proof*⟩

lemma *galois-prop* [*HOL-Algebra-galois-alignment*]: $(L \trianglelefteq R)\ l\ r$
⟨*proof*⟩

lemma *galois-connection* [*HOL-Algebra-galois-alignment*]: $(L \dashv R)\ l\ r$
⟨*proof*⟩

end
end

context *galois-bijection*
begin

context
fixes L R l r
defines $L \equiv (\sqsubseteq_{\mathcal{X}})\restriction_{carrier\ \mathcal{X}}\restriction_{carrier\ \mathcal{X}}$ **and** $R \equiv (\sqsubseteq_{\mathcal{Y}})\restriction_{carrier\ \mathcal{Y}}\restriction_{carrier\ \mathcal{Y}}$
and $l \equiv \pi^*$ **and** $r \equiv \pi_*$
notes *defs*[*simp*] = *L-def R-def l-def r-def* **and** *rel-restrict-right-eq*[*simp*]
and *rel-restrict-leftI*[*intro*!] *rel-restrict-leftE*[*elim*!] *in-codomE*[*elim*!]
begin

interpretation *galois R L r l* ⟨*proof*⟩

lemma *half-galois-prop-left-right-left* [*HOL-Algebra-galois-alignment*]: $(R\ {}_h\trianglelefteq\ L)\ r\ l$
⟨*proof*⟩

lemma *half-galois-prop-right-right-left* [*HOL-Algebra-galois-alignment*]: $(R\ \trianglelefteq_h\ L)\ r\ l$
⟨*proof*⟩

lemma *prop-right-right-left* [*HOL-Algebra-galois-alignment*]: $(R \trianglelefteq L)\ r\ l$
⟨*proof*⟩

lemma *galois-equivalence* [*HOL-Algebra-galois-alignment*]: $(L \equiv_G R)\ l\ r$
⟨*proof*⟩

end
end

end

2.9 HOL-Algebra Alignments

theory *HOL-Algebra-Alignments*
imports
HOL-Algebra-Alignment-Galois
HOL-Algebra-Alignment-Orders
begin

Summary Alignment of concepts with HOL-Algebra counterparts

end

2.10 HOL Syntax Bundles

2.10.1 Basic Syntax

theory *HOL-Syntax-Bundles-Base*
imports *HOL-Basics-Base*

begin

bundle *HOL-ascii-syntax*
begin
notation (*ASCII*) *Not* (‹~ -› [*40*] *40*)
  and *conj* (**infixr** ‹&› *35*)
  and *disj* (**infixr** ‹|› *30*)
  and *implies* (**infixr** ‹−−>› *25*)
  and *not-equal* (**infixl** ‹~=› *50*)
syntax -*Let* :: [*letbinds*, *'a*] ⇒ *'a* (‹(*let* (-)/ *in* (-))› *10*)
end

end

2.10.2 Group Syntax

theory *HOL-Syntax-Bundles-Groups*
  imports *HOL.Groups*
begin

bundle *HOL-groups-syntax*
begin
notation *Groups.zero* (‹*0*›)
notation *Groups.one* (‹*1*›)
notation *Groups.plus* (**infixl** ‹+› *65*)
notation *Groups.minus* (**infixl** ‹−› *65*)
notation *Groups.uminus* (‹(‹*open-block notation*=‹*prefix* −››− -)› [*81*] *80*)
notation *Groups.times* (**infixl** ‹*› *70*)
notation *abs* (‹|-|›)
end

end

theory *HOL-Syntax-Bundles*
  imports
    HOL-Syntax-Bundles-Base
    HOL-Syntax-Bundles-Functions
    HOL-Syntax-Bundles-Groups
    HOL-Syntax-Bundles-Lattices
    HOL-Syntax-Bundles-Orders
begin

Summary Bundles to enable and disable syntax from HOL.

end

2.10.3 Galois Relator

theory *Transport-Compositions-Agree-Galois-Relator*
  imports

Transport-Compositions-Agree-Base
begin

context *transport-comp-agree*
begin

lemma *left-Galois-le-comp-left-GaloisI*:
assumes *in-codom-mono-r2*: $(\textit{in-codom}\ (\leq_{R2}) \Rightarrow \textit{in-codom}\ (\leq_{R1}))\ r2$
and *r2-L2-self-if-in-codom*: $\bigwedge z.\ \textit{in-codom}\ (\leq_{R2})\ z \Longrightarrow r2\ z \leq_{L2} r2\ z$
shows $({}_{L}\lessapprox) \leq (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$
⟨*proof*⟩

lemma *comp-left-Galois-le-left-GaloisI*:
assumes *mono-r1*: $((\leq_{R1}) \Rightarrow (\leq_{L1}))\ r1$
and *trans-L1*: *transitive* $(\leq_{L1})$
and *R1-r2-if-in-codom*: $\bigwedge y\ z.\ \textit{in-codom}\ (\leq_{R2})\ z \Longrightarrow y \leq_{L2} r2\ z \Longrightarrow y \leq_{R1} r2\ z$
shows $(({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox)) \leq ({}_{L}\lessapprox)$
⟨*proof*⟩

corollary *left-Galois-eq-comp-left-GaloisI*:
assumes $((\leq_{R1}) \Rightarrow (\leq_{L1}))\ r1$
and *transitive* $(\leq_{L1})$
and $\bigwedge z.\ \textit{in-codom}\ (\leq_{R2})\ z \Longrightarrow r2\ z \leq_{L2} r2\ z$
and $\bigwedge y\ z.\ \textit{in-codom}\ (\leq_{R2})\ z \Longrightarrow y \leq_{L2} r2\ z \Longrightarrow y \leq_{R1} r2\ z$
shows $({}_{L}\lessapprox) = (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$
⟨*proof*⟩

corollary *left-Galois-eq-comp-left-GaloisI'*:
assumes $((\leq_{R1}) \Rightarrow (\leq_{L1}))\ r1$
and *transitive* $(\leq_{L1})$
and $((\leq_{R2}) \Rightarrow (\leq_{L2}))\ r2$
and *reflexive-on* $(\textit{in-codom}\ (\leq_{R2}))\ (\leq_{R2})$
and $\bigwedge y\ z.\ \textit{in-codom}\ (\leq_{R2})\ z \Longrightarrow y \leq_{L2} r2\ z \Longrightarrow y \leq_{R1} r2\ z$
shows $({}_{L}\lessapprox) = (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$
⟨*proof*⟩

corollary *left-Galois-eq-comp-left-GaloisI''*:
assumes $((\leq_{R1}) \Rightarrow (\leq_{L1}))\ r1$
and *transitive* $(\leq_{L1})$
and $((\leq_{R2}) \Rightarrow (\leq_{L2}))\ r2$
and *reflexive-on* $(\textit{in-codom}\ (\leq_{L2}))\ (\leq_{L2})$
and $\bigwedge y\ z.\ \textit{in-codom}\ (\leq_{R2})\ z \Longrightarrow y \leq_{L2} r2\ z \Longrightarrow y \leq_{R1} r2\ z$
shows $({}_{L}\lessapprox) = (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$
⟨*proof*⟩

end

context *transport-comp-same*
begin

lemma *left-Galois-eq-comp-left-GaloisI*:
assumes $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*
and *transitive* $(\leq_{L1})$
and $((\leq_{R2}) \Rightarrow (\leq_{R1}))$ *r2*
and *reflexive-on* (*in-codom* $(\leq_{R2})$) $(\leq_{R2})$
shows $({}_{L}\overset{\leq}{\approx}) = (({}_{L1}\overset{\leq}{\approx}) \circ\circ ({}_{L2}\overset{\leq}{\approx}))$
⟨*proof*⟩

lemma *left-Galois-eq-comp-left-GaloisI′*:
assumes $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*
and *transitive* $(\leq_{L1})$
and *reflexive-on* (*in-codom* $(\leq_{R1})$) $(\leq_{R1})$
and $((\leq_{R2}) \Rightarrow (\leq_{R1}))$ *r2*
shows $({}_{L}\overset{\leq}{\approx}) = (({}_{L1}\overset{\leq}{\approx}) \circ\circ ({}_{L2}\overset{\leq}{\approx}))$
⟨*proof*⟩

end

end

2.10.4 Order Equivalence

theory *Transport-Compositions-Agree-Order-Equivalence*
imports
Transport-Compositions-Agree-Monotone
begin

context *transport-comp-agree*
begin

Unit

Inflationary **lemma** *inflationary-on-unitI*:
assumes *mono-l1*: $(P \Rightarrow P')$ *l1*
and *mono-r1*: $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*
and *inflationary-unit1*: *inflationary-on* P $(\leq_{L1})$ η_1
and *trans-L1*: *transitive* $(\leq_{L1})$
and *inflationary-unit2*: *inflationary-on* P' $(\leq_{L2})$ η_2
and *L2-le-R1*: $\bigwedge x.\ P\ x \Longrightarrow l1\ x \leq_{L2} r2\ (l\ x) \Longrightarrow l1\ x \leq_{R1} r2\ (l\ x)$
shows *inflationary-on* P $(\leq_{L})$ η
⟨*proof*⟩

corollary *inflationary-on-in-field-unitI*:
assumes $((\leq_{L1}) \Rightarrow (\leq_{L2}))$ *l1*
and $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*
and *inflationary-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$ η_1
and *transitive* $(\leq_{L1})$
and *inflationary-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$ η_2

and $\bigwedge x.$ *in-field* $(\leq_{L1})$ $x \Longrightarrow$ *l1* $x \leq_{L2}$ *r2* $(l\ x) \Longrightarrow$ *l1* $x \leq_{R1}$ *r2* $(l\ x)$
shows *inflationary-on* (*in-field* $(\leq_L)$) $(\leq_L)$ η
⟨*proof*⟩

Deflationary context
begin

interpretation *inv* :
transport-comp-agree $(\geq_{L1})$ $(\geq_{R1})$ *l1 r1* $(\geq_{L2})$ $(\geq_{R2})$ *l2 r2*
rewrites $\bigwedge R\ S.\ (R^{-1} \Rightarrow S^{-1}) \equiv (R \Rightarrow S)$
and $\bigwedge (P :: 'i \Rightarrow bool)\ (R :: 'j \Rightarrow 'i \Rightarrow bool).$
$(inflationary\text{-}on\ P\ R^{-1} :: ('i \Rightarrow 'j) \Rightarrow bool) \equiv deflationary\text{-}on\ P\ R$
and $\bigwedge (R :: 'i \Rightarrow 'i \Rightarrow bool).\ transitive\ R^{-1} \equiv transitive\ R$
and $\bigwedge R.\ in\text{-}field\ R^{-1} \equiv in\text{-}field\ R$
⟨*proof*⟩

lemma *deflationary-on-in-field-unitI*:
assumes $((\leq_{L1}) \Rightarrow (\leq_{L2}))$ *l1*
and $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*
and *deflationary-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$ η_1
and *transitive* $(\leq_{L1})$
and *deflationary-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$ η_2
and $\bigwedge x.$ *in-field* $(\leq_{L1})$ $x \Longrightarrow$ *r2* $(l\ x) \leq_{L2}$ *l1* $x \Longrightarrow$ *r2* $(l\ x) \leq_{R1}$ *l1* x
shows *deflationary-on* (*in-field* $(\leq_L)$) $(\leq_L)$ η
⟨*proof*⟩

end

Relational Equivalence

corollary *rel-equivalence-on-in-field-unitI*:
assumes $((\leq_{L1}) \Rightarrow (\leq_{L2}))$ *l1*
and $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*
and *rel-equivalence-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$ η_1
and *transitive* $(\leq_{L1})$
and *rel-equivalence-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$ η_2
and $\bigwedge x.$ *in-field* $(\leq_{L1})$ $x \Longrightarrow$ *l1* $x \leq_{L2}$ *r2* $(l\ x) \Longrightarrow$ *l1* $x \leq_{R1}$ *r2* $(l\ x)$
and $\bigwedge x.$ *in-field* $(\leq_{L1})$ $x \Longrightarrow$ *r2* $(l\ x) \leq_{L2}$ *l1* $x \Longrightarrow$ *r2* $(l\ x) \leq_{R1}$ *l1* x
shows *rel-equivalence-on* (*in-field* $(\leq_L)$) $(\leq_L)$ η
⟨*proof*⟩

Counit

Corresponding lemmas for the counit can be obtained by flipping the interpretation of the locale.

Order Equivalence

interpretation *flip* : *transport-comp-agree R2 L2 r2 l2 R1 L1 r1 l1*
rewrites *flip.g1.unit* $\equiv \varepsilon_2$ **and** *flip.g2.unit* $\equiv \varepsilon_1$ **and** *flip.unit* $\equiv \varepsilon$

⟨*proof*⟩

lemma *order-equivalenceI*:
assumes $((\leq_{L1}) \equiv_o (\leq_{R1}))$ *l1 r1*
and *transitive* $(\leq_{L1})$
and $((\leq_{L2}) \equiv_o (\leq_{R2}))$ *l2 r2*
and *transitive* $(\leq_{R2})$
and $\bigwedge x\ y.\ x \leq_{L1} y \Longrightarrow l1\ x \leq_{R1} l1\ y \Longrightarrow l1\ x \leq_{L2} l1\ y$
and $\bigwedge x\ y.\ x \leq_{R2} y \Longrightarrow r2\ x \leq_{L2} r2\ y \Longrightarrow r2\ x \leq_{R1} r2\ y$
and $\bigwedge x.$ *in-field* $(\leq_{L1})\ x \Longrightarrow l1\ x \leq_{L2} r2\ (l\ x) \Longrightarrow l1\ x \leq_{R1} r2\ (l\ x)$
and $\bigwedge x.$ *in-field* $(\leq_{L1})\ x \Longrightarrow r2\ (l\ x) \leq_{L2} l1\ x \Longrightarrow r2\ (l\ x) \leq_{R1} l1\ x$
and $\bigwedge x.$ *in-field* $(\leq_{R2})\ x \Longrightarrow r2\ x \leq_{R1} l1\ (r\ x) \Longrightarrow r2\ x \leq_{L2} l1\ (r\ x)$
and $\bigwedge x.$ *in-field* $(\leq_{R2})\ x \Longrightarrow l1\ (r\ x) \leq_{R1} r2\ x \Longrightarrow l1\ (r\ x) \leq_{L2} r2\ x$
shows $((\leq_L) \equiv_o (\leq_R))\ l\ r$
⟨*proof*⟩

end

context *transport-comp-same*
begin

lemma *order-equivalenceI*:
assumes $((\leq_{L1}) \equiv_o (\leq_{R1}))$ *l1 r1*
and *transitive* $(\leq_{L1})$
and $((\leq_{R1}) \equiv_o (\leq_{R2}))$ *l2 r2*
and *transitive* $(\leq_{R2})$
shows $((\leq_L) \equiv_o (\leq_R))\ l\ r$
⟨*proof*⟩

end

end

theory *Transport-Compositions-Agree*
imports
Transport-Compositions-Agree-Galois-Equivalence
Transport-Compositions-Agree-Galois-Relator
Transport-Compositions-Agree-Order-Equivalence
begin

Summary The general - though probably not very useful - results for the composition of transportable components under the condition of agreeing middle relations can be found in *transport-comp-agree*. The special case of a coinciding middle relation can be found in *transport-comp-same*. The latter corresponds to the well-know result in the literature, generalised to partial Galois connections and equivalences.

end

2.11 Generic Compositions

2.11.1 Basic Setup

theory *Transport-Compositions-Generic-Base*
imports
Equivalence-Relations
Transport-Base
begin

locale *transport-comp* =
t1 : *transport L1 R1 l1 r1* + *t2* : *transport L2 R2 l2 r2*
for *L1* :: $'a \Rightarrow 'a \Rightarrow bool$
and *R1* :: $'b \Rightarrow 'b \Rightarrow bool$
and *l1* :: $'a \Rightarrow 'b$
and *r1* :: $'b \Rightarrow 'a$
and *L2* :: $'b \Rightarrow 'b \Rightarrow bool$
and *R2* :: $'c \Rightarrow 'c \Rightarrow bool$
and *l2* :: $'b \Rightarrow 'c$
and *r2* :: $'c \Rightarrow 'b$
begin

This locale collects results about the composition of transportable components under some generic compatibility conditions on *R1* and *L2* (cf. below). The composition is rather subtle, but in return can cover quite general cases.

Explanations and intuition about the construction can be found in [2].

notation *L1* (**infix** $\langle \leq_{L1} \rangle$ *50*)
notation *R1* (**infix** $\langle \leq_{R1} \rangle$ *50*)
notation *L2* (**infix** $\langle \leq_{L2} \rangle$ *50*)
notation *R2* (**infix** $\langle \leq_{R2} \rangle$ *50*)

notation *t1.ge-left* (**infix** $\langle \geq_{L1} \rangle$ *50*)
notation *t1.ge-right* (**infix** $\langle \geq_{R1} \rangle$ *50*)
notation *t2.ge-left* (**infix** $\langle \geq_{L2} \rangle$ *50*)
notation *t2.ge-right* (**infix** $\langle \geq_{R2} \rangle$ *50*)

notation *t1.left-Galois* (**infix** $\langle {}_{L1}\lessapprox \rangle$ *50*)
notation *t1.right-Galois* (**infix** $\langle {}_{R1}\lessapprox \rangle$ *50*)
notation *t2.left-Galois* (**infix** $\langle {}_{L2}\lessapprox \rangle$ *50*)
notation *t2.right-Galois* (**infix** $\langle {}_{R2}\lessapprox \rangle$ *50*)

notation *t1.ge-Galois-left* (**infix** $\langle \gtrapprox_{L1} \rangle$ *50*)
notation *t1.ge-Galois-right* (**infix** $\langle \gtrapprox_{R1} \rangle$ *50*)
notation *t2.ge-Galois-left* (**infix** $\langle \gtrapprox_{L2} \rangle$ *50*)
notation *t2.ge-Galois-right* (**infix** $\langle \gtrapprox_{R2} \rangle$ *50*)

notation *t1.right-ge-Galois* (**infix** $\langle {}_{R1}\gtrapprox \rangle$ *50*)
notation *t1.Galois-right* (**infix** $\langle \lessapprox_{R1} \rangle$ *50*)

notation *t2.right-ge-Galois* (**infix** $\langle_{R2}\gtrsim\rangle$ *50*)
notation *t2.Galois-right* (**infix** $\langle\lesssim_{R2}\rangle$ *50*)

notation *t1.left-ge-Galois* (**infix** $\langle_{L1}\gtrsim\rangle$ *50*)
notation *t1.Galois-left* (**infix** $\langle\lesssim_{L1}\rangle$ *50*)
notation *t2.left-ge-Galois* (**infix** $\langle_{L2}\gtrsim\rangle$ *50*)
notation *t2.Galois-left* (**infix** $\langle\lesssim_{L2}\rangle$ *50*)

notation *t1.unit* ($\langle\eta_1\rangle$)
notation *t1.counit* ($\langle\varepsilon_1\rangle$)
notation *t2.unit* ($\langle\eta_2\rangle$)
notation *t2.counit* ($\langle\varepsilon_2\rangle$)

definition $L \equiv (_{L1}\lesssim) \circ\circ (\leq_{L2}) \circ\circ (_{R1}\lesssim)$

lemma *left-rel-eq-comp*: $L = (_{L1}\lesssim) \circ\circ (\leq_{L2}) \circ\circ (_{R1}\lesssim)$
⟨*proof*⟩

definition $l \equiv l2 \circ l1$

lemma *left-eq-comp*: $l = l2 \circ l1$
⟨*proof*⟩

lemma *left-eq* [*simp*]: $l\ x = l2\ (l1\ x)$
⟨*proof*⟩

context
begin

interpretation *flip* : *transport-comp R2 L2 r2 l2 R1 L1 r1 l1* ⟨*proof*⟩

abbreviation $R \equiv flip.L$
abbreviation $r \equiv flip.l$

lemma *right-rel-eq-comp*: $R = (_{R2}\lesssim) \circ\circ (\leq_{R1}) \circ\circ (_{L2}\lesssim)$
⟨*proof*⟩

lemma *right-eq-comp*: $r = r1 \circ r2$
⟨*proof*⟩

lemma *right-eq* [*simp*]: $r\ z = r1\ (r2\ z)$
⟨*proof*⟩

lemmas *transport-defs* = *left-rel-eq-comp left-eq-comp right-rel-eq-comp right-eq-comp*

end

sublocale *transport L R l r* ⟨*proof*⟩

notation L (**infix** $\langle \leq_L \rangle$ *50*)
notation R (**infix** $\langle \leq_R \rangle$ *50*)

lemma *left-relI* [*intro*]:
assumes $x\ {}_{L1}\overset{\leq}{\approx}\ y$
and $y \leq_{L2} y'$
and $y'\ {}_{R1}\overset{\leq}{\approx}\ x'$
shows $x \leq_L x'$
⟨*proof*⟩

lemma *left-relE* [*elim*]:
assumes $x \leq_L x'$
obtains $y\ y'$ **where** $x\ {}_{L1}\overset{\leq}{\approx}\ y\ y \leq_{L2} y'\ y'\ {}_{R1}\overset{\leq}{\approx}\ x'$
⟨*proof*⟩

context
begin

interpretation *flip* : *transport-comp R2 L2 r2 l2 R1 L1 r1 l1* ⟨*proof*⟩
interpretation *inv* : *transport-comp* $(\geq_{L1})$ $(\geq_{R1})$ *l1 r1* $(\geq_{L2})$ $(\geq_{R2})$ *l2 r2* ⟨*proof*⟩

lemma *ge-left-rel-eq-left-rel-inv-if-galois-prop* [*simp*]:
assumes $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1* $((\leq_{R1}) \trianglelefteq (\leq_{L1}))$ *r1 l1*
shows $(\geq_L)$ = *transport-comp.L* $(\geq_{L1})$ $(\geq_{R1})$ *l1 r1* $(\geq_{L2})$
⟨*proof*⟩

corollary *left-rel-inv-iff-left-rel-if-galois-prop* [*iff*]:
assumes $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1* $((\leq_{R1}) \trianglelefteq (\leq_{L1}))$ *r1 l1*
shows (*transport-comp.L* $(\geq_{L1})$ $(\geq_{R1})$ *l1 r1* $(\geq_{L2})$) *x x'* $\longleftrightarrow$ $x' \leq_L x$
⟨*proof*⟩

Simplification of Relations

lemma *left-rel-le-left-rel1I*:
assumes $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*
and $((\leq_{R1}) \trianglelefteq_h (\leq_{L1}))$ *r1 l1*
and *trans-L1*: *transitive* $(\leq_{L1})$
and *mono-l1*: $((\leq_L) \Rightarrow ((\leq_{R1}) \circ\circ (\leq_{R1})))$ *l1*
shows $(\leq_L) \leq (\leq_{L1})$
⟨*proof*⟩

lemma *left-rel1-le-left-relI*:
assumes $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*
and *mono-l1*: $((\leq_{L1}) \Rightarrow ((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})))$ *l1*
shows $(\leq_{L1}) \leq (\leq_L)$
⟨*proof*⟩

corollary *left-rel-eq-left-rel1I*:

assumes $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*
and $((\leq_{R1}) \ {}_h\trianglerighteq (\leq_{L1}))$ *r1 l1*
and *transitive* $(\leq_{L1})$
and $((\leq_L) \Rightarrow ((\leq_{R1}) \circ\circ (\leq_{R1})))$ *l1*
and $((\leq_{L1}) \Rightarrow ((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})))$ *l1*
shows $(\leq_L) = (\leq_{L1})$
⟨*proof*⟩

Note that we may not necessarily have $\mathit{flip}.R = (\leq_{L1})$, even in case of equivalence relations. Depending on the use case, one thus may wish to use an alternative composition operation.

lemma *ex-order-equiv-left-rel-neq-left-rel1*:
$\exists (L1 :: bool \Rightarrow \text{-}) \ (R1 :: bool \Rightarrow \text{-}) \ l1 \ r1$
$(L2 :: bool \Rightarrow \text{-}) \ (R2 :: bool \Rightarrow \text{-}) \ l2 \ r2.$
$(L1 \equiv_o R1) \ l1 \ r1$
$\wedge$ *equivalence-rel L1* $\wedge$ *equivalence-rel R1*
$\wedge \ (L2 \equiv_o R2) \ l2 \ r2$
$\wedge$ *equivalence-rel L2* $\wedge$ *equivalence-rel R2*
$\wedge$ *transport-comp.L L1 R1 l1 r1 L2* $\neq$ *L1*
⟨*proof*⟩

end

Generic Left to Right Introduction Rules

The following lemmas generalise the proof outline used, for example, when proving monotonicity and the Galois property (cf. the paper [2]).

interpretation *flip* : *transport-comp R2 L2 r2 l2 R1 L1 r1 l1* ⟨*proof*⟩

lemma *right-rel-if-left-relI*:
assumes $x \leq_L x'$
and *l1-R1-if-L1-r1*: $\bigwedge y.$ *in-codom* $(\leq_{R1}) \ y \Longrightarrow x \leq_{L1} r1 \ y \Longrightarrow l1 \ x \leq_{R1} y$
and *t-R1-if-l1-R1*: $\bigwedge y.$ *l1* $x \leq_{R1} y \Longrightarrow t \ y \leq_{R1} y$
and *R2-l2-if-t-L2-if-l1-R1*:
$\bigwedge y \ y'.$ *l1* $x \leq_{R1} y \Longrightarrow t \ y \leq_{L2} y' \Longrightarrow z \leq_{R2} l2 \ y'$
and *R1-b-if-R1-l1-if-R1-l1*:
$\bigwedge y \ y'. \ y \leq_{R1} l1 \ x' \Longrightarrow y' \leq_{R1} l1 \ x' \Longrightarrow y' \leq_{R1} b \ y$
and *b-L2-r2-if-in-codom-L2-b-if-R1-l1*:
$\bigwedge y. \ y \leq_{R1} l1 \ x' \Longrightarrow$ *in-codom* $(\leq_{L2}) \ (b \ y) \Longrightarrow b \ y \leq_{L2} r2 \ z'$
and *in-codom-R2-if-in-codom-L2-b-if-R1-l1*:
$\bigwedge y. \ y \leq_{R1} l1 \ x' \Longrightarrow$ *in-codom* $(\leq_{L2}) \ (b \ y) \Longrightarrow$ *in-codom* $(\leq_{R2}) \ z'$
and *rel-comp-le*: $(\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1}) \leq (\leq_{L2}) \circ\circ (\leq_{R1})$
and *in-codom-rel-comp-le*: *in-codom* $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-codom* $(\leq_{L2})$
shows $z \leq_R z'$
⟨*proof*⟩

lemma *right-rel-if-left-relI'*:
assumes $x \leq_L x'$

and *l1-R1-if-L1-r1*: $\bigwedge y.\ \textit{in-codom}\ (\leq_{R1})\ y \Longrightarrow x \leq_{L1} r1\ y \Longrightarrow l1\ x \leq_{R1} y$
and *R1-b-if-R1-l1*: $\bigwedge y.\ y \leq_{R1} l1\ x' \Longrightarrow y \leq_{R1} b\ y$
and *L2-r2-if-L2-b-if-R1-l1*:
$\bigwedge y\ y'.\ y \leq_{R1} l1\ x' \Longrightarrow y' \leq_{L2} b\ y \Longrightarrow y' \leq_{L2} r2\ z'$
and *in-codom-R2-if-L2-b-if-R1-l1*:
$\bigwedge y\ y'.\ y \leq_{R1} l1\ x' \Longrightarrow y' \leq_{L2} b\ y \Longrightarrow \textit{in-codom}\ (\leq_{R2})\ z'$
and *t-R1-if-R1-l1-if-l1-R1*:
$\bigwedge y\ y'\ y''.\ l1\ x \leq_{R1} y \Longrightarrow l1\ x \leq_{R1} y' \Longrightarrow t\ y \leq_{R1} y'$
and *R2-l2-t-if-in-dom-L2-t-if-l1-R1*:
$\bigwedge y\ y'.\ l1\ x \leq_{R1} y \Longrightarrow \textit{in-dom}\ (\leq_{L2})\ (t\ y) \Longrightarrow z \leq_{R2} l2\ (t\ y)$
and *in-codom-L2-t-if-in-dom-L2-t-if-l1-R1*:
$\bigwedge y\ y'.\ l1\ x \leq_{R1} y \Longrightarrow \textit{in-dom}\ (\leq_{L2})\ (t\ y) \Longrightarrow \textit{in-codom}\ (\leq_{L2})\ (t\ y)$
and *rel-comp-le*: $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{R1}) \circ\circ (\leq_{L2}))$
and *in-dom-rel-comp-le*: $\textit{in-dom}\ ((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq \textit{in-dom}\ (\leq_{L2})$
shows $z \leq_R z'$
⟨*proof*⟩

Simplification of Monotonicity Assumptions

Some sufficient conditions for monotonicity assumptions that repeatedly arise in various places.

lemma *mono-in-dom-left-rel-left1-if-in-dom-rel-comp-le*:
assumes $((\leq_{L1})\ {}_h\trianglelefteq\ (\leq_{R1}))\ l1\ r1$
and $\textit{in-dom}\ ((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq \textit{in-dom}\ (\leq_{L2})$
shows $(\textit{in-dom}\ (\leq_L) \Rightarrow \textit{in-dom}\ (\leq_{L2}))\ l1$
⟨*proof*⟩

lemma *mono-in-codom-left-rel-left1-if-in-codom-rel-comp-le*:
assumes $((\leq_{L1})\ {}_h\trianglelefteq\ (\leq_{R1}))\ l1\ r1$
and $\textit{in-codom}\ ((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq \textit{in-codom}\ (\leq_{L2})$
shows $(\textit{in-codom}\ (\leq_L) \Rightarrow \textit{in-codom}\ (\leq_{L2}))\ l1$
⟨*proof*⟩

Simplification of Compatibility Conditions

Most results will depend on certain compatibility conditions between $(\leq_{R1})$ and $(\leq_{L2})$. We next derive some sufficient assumptions for these conditions.

end

lemma *rel-comp-comp-le-rel-comp-if-rel-comp-comp-if-in-dom-leI*:
assumes *trans-R*: *transitive R*
and *refl-S*: *reflexive-on P S*
and *in-dom-le*: $\textit{in-dom}\ (R \circ\circ S \circ\circ R) \leq P$
and *rel-comp-le*: $(S \circ\circ R \circ\circ S) \leq (S \circ\circ R)$
shows $(R \circ\circ S \circ\circ R) \leq (S \circ\circ R)$
⟨*proof*⟩

lemma *rel-comp-comp-le-rel-comp-if-rel-comp-comp-if-in-codom-leI*:

assumes *trans-R*: *transitive R*
and *refl-S*: *reflexive-on P S*
and *in-codom-le*: *in-codom* $(R \circ\circ S \circ\circ R) \leq P$
and *rel-comp-le*: $(S \circ\circ R \circ\circ S) \leq (R \circ\circ S)$
shows $(R \circ\circ S \circ\circ R) \leq (R \circ\circ S)$
⟨*proof*⟩
thm *mono-rel-comp*
lemma *rel-comp-comp-le-rel-comp-if-rel-comp-le-if-transitive*:
assumes *trans-R*: *transitive R*
and *R-S-le*: $(R \circ\circ S) \leq (S \circ\circ R)$
shows $(R \circ\circ S \circ\circ R) \leq (S \circ\circ R)$
⟨*proof*⟩

lemma *rel-comp-comp-le-rel-comp-if-rel-comp-le-if-transitive'*:
assumes *trans-R*: *transitive R*
and *S-R-le*: $(S \circ\circ R) \leq (R \circ\circ S)$
shows $(R \circ\circ S \circ\circ R) \leq (R \circ\circ S)$
⟨*proof*⟩

lemma *rel-comp-eq-rel-comp-if-le-if-transitive-if-reflexive*:
assumes *refl-R*: *reflexive-on* (*in-field S*) *R*
and *trans-S*: *transitive S*
and *R-le*: $R \leq S \sqcup (=)$
shows $(R \circ\circ S) = (S \circ\circ R)$
⟨*proof*⟩

lemma *rel-comp-eq-rel-comp-if-in-field-le-if-le-eq*:
assumes *le-eq*: $R \leq (=)$
and *in-field-le*: *in-field S* ≤ *in-field R*
shows $(R \circ\circ S) = (S \circ\circ R)$
⟨*proof*⟩

context *transport-comp*
begin

lemma *left2-right1-left2-le-left2-right1-if-right1-left2-right1-le-left2-right1*:
assumes *reflexive-on* (*in-codom* $(\leq_{R1})$) $(\leq_{R1})$
and *transitive* $(\leq_{L2})$
and $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{L2}) \circ\circ (\leq_{R1}))$
and *in-codom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-codom* $(\leq_{R1})$
shows $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq ((\leq_{L2}) \circ\circ (\leq_{R1}))$
⟨*proof*⟩

lemma *left2-right1-left2-le-right1-left2-if-right1-left2-right1-le-right1-left2I*:
assumes *reflexive-on* (*in-dom* $(\leq_{R1})$) $(\leq_{R1})$
and *transitive* $(\leq_{L2})$
and $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{R1}) \circ\circ (\leq_{L2}))$
and *in-dom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-dom* $(\leq_{R1})$
shows $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq ((\leq_{R1}) \circ\circ (\leq_{L2}))$

⟨*proof*⟩

lemma *in-dom-right1-left2-right1-le-if-right1-left2-right1-le*:
assumes $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{L2}) \circ\circ (\leq_{R1}))$
shows *in-dom* $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-dom* $(\leq_{L2})$
⟨*proof*⟩

lemma *in-codom-right1-left2-right1-le-if-right1-left2-right1-le*:
assumes $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{R1}) \circ\circ (\leq_{L2}))$
shows *in-codom* $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-codom* $(\leq_{L2})$
⟨*proof*⟩

Our main results will be derivable for two different sets of compatibility conditions. The next two lemmas show the equivalence between those two sets under certain assumptions. In cases where these assumptions are met, we will only state the result for one of the two compatibility conditions. The other one will then be derivable using one of the following lemmas.

definition *middle-compatible-dom* ≡
$(\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1}) \leq (\leq_{R1}) \circ\circ (\leq_{L2})$
$\wedge$ *in-dom* $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-dom* $(\leq_{L2})$
$\wedge ((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq ((\leq_{L2}) \circ\circ (\leq_{R1}))$
$\wedge$ *in-dom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-dom* $(\leq_{R1})$

lemma *middle-compatible-domI* [*intro*]:
assumes $(\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1}) \leq (\leq_{R1}) \circ\circ (\leq_{L2})$
and *in-dom* $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-dom* $(\leq_{L2})$
and $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq ((\leq_{L2}) \circ\circ (\leq_{R1}))$
and *in-dom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-dom* $(\leq_{R1})$
shows *middle-compatible-dom*
⟨*proof*⟩

lemma *middle-compatible-domE* [*elim*]:
assumes *middle-compatible-dom*
obtains $(\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1}) \leq (\leq_{R1}) \circ\circ (\leq_{L2})$
and *in-dom* $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-dom* $(\leq_{L2})$
and $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq ((\leq_{L2}) \circ\circ (\leq_{R1}))$
and *in-dom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-dom* $(\leq_{R1})$
⟨*proof*⟩

definition *middle-compatible-codom* ≡
$((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{L2}) \circ\circ (\leq_{R1}))$
$\wedge$ *in-codom* $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-codom* $(\leq_{L2})$
$\wedge (\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2}) \leq (\leq_{R1}) \circ\circ (\leq_{L2})$
$\wedge$ *in-codom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-codom* $(\leq_{R1})$

lemma *middle-compatible-codomI* [*intro*]:
assumes $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{L2}) \circ\circ (\leq_{R1}))$
and *in-codom* $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-codom* $(\leq_{L2})$
and $(\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2}) \leq (\leq_{R1}) \circ\circ (\leq_{L2})$

and *in-codom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-codom* $(\leq_{R1})$
shows *middle-compatible-codom*
⟨*proof*⟩

lemma *middle-compatible-codomE* [*elim*]:
assumes *middle-compatible-codom*
obtains $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{L2}) \circ\circ (\leq_{R1}))$
and *in-codom* $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-codom* $(\leq_{L2})$
and $(\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2}) \leq (\leq_{R1}) \circ\circ (\leq_{L2})$
and *in-codom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-codom* $(\leq_{R1})$
⟨*proof*⟩

context
begin

interpretation *flip* : *transport-comp R2 L2 r2 l2 R1 L1 r1 l1* ⟨*proof*⟩

lemma *rel-comp-comp-le-assms-if-in-codom-rel-comp-comp-leI*:
assumes *preorder-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and *preorder-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$
and *middle-compatible-codom*
shows *middle-compatible-dom*
⟨*proof*⟩

lemma *rel-comp-comp-le-assms-if-in-dom-rel-comp-comp-leI*:
assumes *preorder-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and *preorder-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$
and *middle-compatible-dom*
shows *middle-compatible-codom*
⟨*proof*⟩

lemma *middle-compatible-dom-iff-middle-compatible-codom-if-preorder-on*:
assumes *preorder-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and *preorder-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$
shows *middle-compatible-dom* $\longleftrightarrow$ *middle-compatible-codom*
⟨*proof*⟩

end

Finally we derive some sufficient assumptions for the compatibility conditions.

lemma *right1-left2-right1-le-assms-if-right1-left2-eqI*:
assumes *transitive* $(\leq_{R1})$
and $((\leq_{R1}) \circ\circ (\leq_{L2})) = ((\leq_{L2}) \circ\circ (\leq_{R1}))$
shows $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{L2}) \circ\circ (\leq_{R1}))$
and $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{R1}) \circ\circ (\leq_{L2}))$
⟨*proof*⟩

interpretation *flip* : *transport-comp R2 L2 r2 l2 R1 L1 r1 l1*

rewrites $((\leq_{L2}) \circ\circ (\leq_{R1})) = ((\leq_{R1}) \circ\circ (\leq_{L2})) \equiv ((\leq_{R1}) \circ\circ (\leq_{L2})) = ((\leq_{L2}) \circ\circ (\leq_{R1}))$
⟨*proof*⟩

lemma *middle-compatible-codom-if-rel-comp-eq-if-transitive*:
assumes *transitive* $(\leq_{R1})$ *transitive* $(\leq_{L2})$
and $((\leq_{R1}) \circ\circ (\leq_{L2})) = ((\leq_{L2}) \circ\circ (\leq_{R1}))$
shows *middle-compatible-codom*
⟨*proof*⟩

lemma *middle-compatible-codom-if-right1-le-left2-eqI*:
assumes *preorder-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$ *transitive* $(\leq_{L2})$
and $(\leq_{R1}) \leq (\leq_{L2}) \sqcup (=)$
and *in-field* $(\leq_{L2}) \leq$ *in-field* $(\leq_{R1})$
shows *middle-compatible-codom*
⟨*proof*⟩

lemma *middle-compatible-codom-if-right1-le-eqI*:
assumes $(\leq_{R1}) \leq (=)$
and *transitive* $(\leq_{L2})$
and *in-field* $(\leq_{L2}) \leq$ *in-field* $(\leq_{R1})$
shows *middle-compatible-codom*
⟨*proof*⟩

end

end

2.11.2 Galois Property

theory *Transport-Compositions-Generic-Galois-Property*
imports
Transport-Compositions-Generic-Base
begin

context *transport-comp*
begin

interpretation *flip* : *transport-comp R2 L2 r2 l2 R1 L1 r1 l1*
rewrites *flip.t2.unit* $= \varepsilon_1$ **and** *flip.t1.counit* $\equiv \eta_2$
⟨*proof*⟩

lemma *half-galois-prop-left-left-rightI*:
assumes $((\leq_{L1})\ {}_h\trianglelefteq\ (\leq_{R1}))$ *l1 r1*
and *deflationary-counit1*: *deflationary-on* (*in-codom* $(\leq_{R1})$) $(\leq_{R1})$ ε_1
and *trans-R1*: *transitive* $(\leq_{R1})$
and $((\leq_{L2}) \Rightarrow (\leq_{R2}))$ *l2*
and *reflexive-on* (*in-codom* $(\leq_{L2})$) $(\leq_{L2})$

and $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{L2}) \circ\circ (\leq_{R1}))$
and *in-codom* $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-codom* $(\leq_{L2})$
and *mono-in-codom-r2*: $(\textit{in-codom}\ (\leq_R) \Rightarrow \textit{in-codom}\ (\leq_{R1}))\ r2$
shows $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$
⟨*proof*⟩

lemma *half-galois-prop-left-left-rightI′*:
assumes $((\leq_{L1})\ {}_h\trianglelefteq\ (\leq_{R1}))\ l1\ r1$
and *deflationary-counit1*: *deflationary-on* $(\textit{in-codom}\ (\leq_{R1}))\ (\leq_{R1})\ \varepsilon_1$
and *trans-R1*: *transitive* $(\leq_{R1})$
and $((\leq_{L2}) \Rightarrow (\leq_{R2}))\ l2$
and *refl-L2*: *reflexive-on* $(\textit{in-dom}\ (\leq_{L2}))\ (\leq_{L2})$
and $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{R1}) \circ\circ (\leq_{L2}))$
and *in-dom* $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-dom* $(\leq_{L2})$
and *mono-in-codom-r2*: $(\textit{in-codom}\ (\leq_R) \Rightarrow \textit{in-codom}\ (\leq_{R1}))\ r2$
shows $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$
⟨*proof*⟩

lemma *half-galois-prop-right-left-rightI*:
assumes $((\leq_{R1}) \Rightarrow (\leq_{L1}))\ r1$
and $((\leq_{L1})\ \trianglelefteq_h\ (\leq_{R1}))\ l1\ r1$
and *inflationary-counit1*: *inflationary-on* $(\textit{in-codom}\ (\leq_{R1}))\ (\leq_{R1})\ \varepsilon_1$
and $((\leq_{R2})\ {}_h\trianglelefteq\ (\leq_{L2}))\ r2\ l2$
and *inflationary-unit2*: *inflationary-on* $(\textit{in-dom}\ (\leq_{L2}))\ (\leq_{L2})\ \eta_2$
and *trans-L2*: *transitive* $(\leq_{L2})$
and *mono-in-dom-l1*: $(\textit{in-dom}\ (\leq_L) \Rightarrow \textit{in-dom}\ (\leq_{L2}))\ l1$
and $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq ((\leq_{R1}) \circ\circ (\leq_{L2}))$
and *in-codom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-codom* $(\leq_{R1})$
shows $((\leq_L)\ \trianglelefteq_h\ (\leq_R))\ l\ r$
⟨*proof*⟩

lemma *half-galois-prop-right-left-rightI′*:
assumes $((\leq_{R1}) \Rightarrow (\leq_{L1}))\ r1$
and *inflationary-unit1*: *inflationary-on* $(\textit{in-dom}\ (\leq_{L1}))\ (\leq_{L1})\ \eta_1$
and *inflationary-counit1*: $\bigwedge y\ z.\ y \leq_{R1} r2\ z \Longrightarrow y \leq_{R1} l1\ (r\ z)$
and *in-dom* $(\leq_{R1}) \leq$ *in-codom* $(\leq_{R1})$
and $((\leq_{R2})\ {}_h\trianglelefteq\ (\leq_{L2}))\ r2\ l2$
and *inflationary-unit2*: *inflationary-on* $(\textit{in-dom}\ (\leq_{L2}))\ (\leq_{L2})\ \eta_2$
and *trans-L2*: *transitive* $(\leq_{L2})$
and *mono-in-dom-l1*: $(\textit{in-dom}\ (\leq_L) \Rightarrow \textit{in-dom}\ (\leq_{L2}))\ l1$
and $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq ((\leq_{L2}) \circ\circ (\leq_{R1}))$
and *in-dom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-dom* $(\leq_{R1})$
shows $((\leq_L) \trianglelefteq_h (\leq_R))\ l\ r$
⟨*proof*⟩

lemma *galois-prop-left-rightI*:
assumes $((\leq_{R1}) \Rightarrow (\leq_{L1}))\ r1$
and $((\leq_{L1}) \trianglelefteq (\leq_{R1}))\ l1\ r1$
and *rel-equivalence-on* $(\textit{in-codom}\ (\leq_{R1}))\ (\leq_{R1})\ \varepsilon_1$

and *transitive* $(\leq_{R1})$
and $((\leq_{L2}) \Rightarrow (\leq_{R2}))$ *l2*
and $((\leq_{R2})\ {}_h\trianglelefteq\ (\leq_{L2}))$ *r2 l2*
and *inflationary-on* (*in-dom* $(\leq_{L2})$) $(\leq_{L2})$ η_2
and *preorder-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$
and *middle-compatible-codom*
shows $((\leq_L) \trianglelefteq (\leq_R))$ *l r*
⟨*proof*⟩

lemma *galois-prop-left-rightI'*:
assumes $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*
and $((\leq_{L1})\ {}_h\trianglelefteq\ (\leq_{R1}))$ *l1 r1*
and *inflationary-on* (*in-dom* $(\leq_{L1})$) $(\leq_{L1})$ η_1
and *rel-equiv-counit1*: *rel-equivalence-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$ ε_1
and *trans-R1*: *transitive* $(\leq_{R1})$
and $((\leq_{L2}) \Rightarrow (\leq_{R2}))$ *l2*
and $((\leq_{R2})\ {}_h\trianglelefteq\ (\leq_{L2}))$ *r2 l2*
and *inflationary-on* (*in-dom* $(\leq_{L2})$) $(\leq_{L2})$ η_2
and *preorder-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$
and *middle-compatible-dom*
shows $((\leq_L) \trianglelefteq (\leq_R))$ *l r*
⟨*proof*⟩

end

end

2.11.3 Monotonicity

theory *Transport-Compositions-Generic-Monotone*
imports
Transport-Compositions-Generic-Base
begin

context *transport-comp*
begin

lemma *mono-wrt-rel-leftI*:
assumes $((\leq_{L1})\ {}_h\trianglelefteq\ (\leq_{R1}))$ *l1 r1*
and $((\leq_{L2}) \Rightarrow (\leq_{R2}))$ *l2*
and *inflationary-unit2*: *inflationary-on* (*in-codom* $(\leq_{L2})$) $(\leq_{L2})$ η_2
and $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{L2}) \circ\circ (\leq_{R1}))$
and *in-codom* $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-codom* $(\leq_{L2})$
shows $((\leq_L) \Rightarrow (\leq_R))$ *l*
⟨*proof*⟩

lemma *mono-wrt-rel-leftI'*:
assumes $((\leq_{L1})\ {}_h\trianglelefteq\ (\leq_{R1}))$ *l1 r1*

and $((\leq_{L2}) \Rightarrow (\leq_{R2}))$ *l2*
and $((\leq_{L2}) \trianglelefteq_h (\leq_{R2}))$ *l2 r2*
and *refl-L2*: *reflexive-on* (*in-dom* $(\leq_{L2})$) $(\leq_{L2})$
and $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{R1}) \circ\circ (\leq_{L2}))$
and *in-dom* $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-dom* $(\leq_{L2})$
shows $((\leq_L) \Rightarrow (\leq_R))$ *l*
⟨*proof*⟩

end

end

2.11.4 Galois Connection

theory *Transport-Compositions-Generic-Galois-Connection*
imports
Transport-Compositions-Generic-Galois-Property
Transport-Compositions-Generic-Monotone
begin

context *transport-comp*
begin

interpretation *flip* : *transport-comp R2 L2 r2 l2 R1 L1 r1 l1*
rewrites *flip.t2.unit* $= \varepsilon_1$ **and** *flip.t1.counit* $\equiv \eta_2$
⟨*proof*⟩

lemma *galois-connection-left-rightI*:
assumes $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*
and $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
and *rel-equivalence-on* (*in-codom* $(\leq_{R1})$) $(\leq_{R1})$ ε_1
and *transitive* $(\leq_{R1})$
and $((\leq_{L2}) \Rightarrow (\leq_{R2}))$ *l2*
and $((\leq_{R2}) \ {}_h\trianglelefteq (\leq_{L2}))$ *r2 l2*
and *inflationary-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$ η_2
and *preorder-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$
and *middle-compatible-codom*
shows $((\leq_L) \dashv (\leq_R))$ *l r*
⟨*proof*⟩

lemma *galois-connection-left-rightI′*:
assumes $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*
and $((\leq_{L1}) \ {}_h\trianglelefteq (\leq_{R1}))$ *l1 r1*
and $((\leq_{R1}) \trianglelefteq_h (\leq_{L1}))$ *r1 l1*
and *inflationary-on* (*in-dom* $(\leq_{L1})$) $(\leq_{L1})$ η_1
and *rel-equivalence-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$ ε_1
and *transitive* $(\leq_{R1})$
and $((\leq_{L2}) \Rightarrow (\leq_{R2}))$ *l2*

and $((\leq_{L2}) \trianglelefteq_h (\leq_{R2}))$ *l2 r2*
and $((\leq_{R2}) \ {}_h\trianglerighteq (\leq_{L2}))$ *r2 l2*
and *inflationary-on* (*in-dom* $(\leq_{L2})$) $(\leq_{L2})$ η_2
and *preorder-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$
and *middle-compatible-dom*
shows $((\leq_L) \dashv (\leq_R))$ *l r*
⟨*proof*⟩

corollary *galois-connection-left-right-if-galois-equivalenceI*:
assumes $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1*
and *preorder-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and $((\leq_{L2}) \equiv_G (\leq_{R2}))$ *l2 r2*
and *preorder-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$
and *middle-compatible-codom*
shows $((\leq_L) \dashv (\leq_R))$ *l r*
⟨*proof*⟩

corollary *galois-connection-left-right-if-order-equivalenceI*:
assumes $((\leq_{L1}) \equiv_o (\leq_{R1}))$ *l1 r1*
and *transitive* $(\leq_{R1})$
and $((\leq_{L2}) \equiv_o (\leq_{R2}))$ *l2 r2*
and *transitive* $(\leq_{L2})$
and *middle-compatible-codom*
shows $((\leq_L) \dashv (\leq_R))$ *l r*
⟨*proof*⟩

end

end

2.11.5 Galois Equivalence

theory *Transport-Compositions-Generic-Galois-Equivalence*
imports
Transport-Compositions-Generic-Galois-Connection
begin

context *transport-comp*
begin

interpretation *flip* : *transport-comp R2 L2 r2 l2 R1 L1 r1 l1*
rewrites *flip.t2.unit* $= \varepsilon_1$ **and** *flip.t1.counit* $\equiv \eta_2$ **and** *flip.t1.unit* $\equiv \varepsilon_2$
⟨*proof*⟩

lemma *galois-equivalenceI*:
assumes $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*
and $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
and *rel-equivalence-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$ ε_1

and *transitive* $(\leq_{R1})$
and $((\leq_{L2}) \Rightarrow (\leq_{R2}))$ *l2*
and $((\leq_{R2}) \trianglelefteq (\leq_{L2}))$ *r2 l2*
and *rel-equivalence-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$ η_2
and *transitive* $(\leq_{L2})$
and *middle-compatible-codom*
shows $((\leq_L) \equiv_G (\leq_R))$ *l r*
⟨*proof*⟩

lemma *galois-equivalenceI′*:
assumes $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*
and $((\leq_{L1}) \ {}_h\trianglelefteq (\leq_{R1}))$ *l1 r1*
and $((\leq_{R1}) \trianglelefteq_h (\leq_{L1}))$ *r1 l1*
and *inflationary-on* (*in-dom* $(\leq_{L1})$) $(\leq_{L1})$ η_1
and *rel-equivalence-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$ ε_1
and *transitive* $(\leq_{R1})$
and $((\leq_{L2}) \Rightarrow (\leq_{R2}))$ *l2*
and $((\leq_{L2}) \trianglelefteq_h (\leq_{R2}))$ *l2 r2*
and $((\leq_{R2}) \ {}_h\trianglelefteq (\leq_{L2}))$ *r2 l2*
and *rel-equivalence-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$ η_2
and *inflationary-on* (*in-dom* $(\leq_{R2})$) $(\leq_{R2})$ ε_2
and *transitive* $(\leq_{L2})$
and *middle-compatible-dom*
shows $((\leq_L) \equiv_G (\leq_R))$ *l r*
⟨*proof*⟩

corollary *galois-equivalence-if-galois-equivalenceI*:
assumes $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1*
and *preorder-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and $((\leq_{L2}) \equiv_G (\leq_{R2}))$ *l2 r2*
and *preorder-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$
and *middle-compatible-codom*
shows $((\leq_L) \equiv_G (\leq_R))$ *l r*
⟨*proof*⟩

corollary *galois-equivalence-if-order-equivalenceI*:
assumes $((\leq_{L1}) \equiv_o (\leq_{R1}))$ *l1 r1*
and *transitive* $(\leq_{R1})$
and $((\leq_{L2}) \equiv_o (\leq_{R2}))$ *l2 r2*
and *transitive* $(\leq_{L2})$
and *middle-compatible-codom*
shows $((\leq_L) \equiv_G (\leq_R))$ *l r*
⟨*proof*⟩

end

end

2.11.6 Galois Relator

theory *Transport-Compositions-Generic-Galois-Relator*
imports
Transport-Compositions-Generic-Base
begin

context *transport-comp*
begin

interpretation *flip* : *transport-comp R2 L2 r2 l2 R1 L1 r1 l1*
rewrites *flip.t2.unit* $\equiv \varepsilon_1$
⟨*proof*⟩

lemma *left-Galois-le-comp-left-GaloisI*:
assumes *mono-r1*: $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*
and *galois-prop1*: $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
and *preorder-R1*: *preorder-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and *rel-comp-le*: $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{R1}) \circ\circ (\leq_{L2}))$
and *mono-in-codom-r2*: (*in-codom* $(\leq_R) \Rightarrow$ *in-codom* $(\leq_{R1})$) *r2*
shows $({}_L\lessapprox) \leq (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$
⟨*proof*⟩

lemma *comp-left-Galois-le-left-GaloisI*:
assumes *mono-r1*: $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*
and *half-galois-prop-left1*: $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*
and *half-galois-prop-right1*: $((\leq_{R1}) \trianglelefteq_h (\leq_{L1}))$ *r1 l1*
and *refl-R1*: *reflexive-on* (*in-codom* $(\leq_{R1})$) $(\leq_{R1})$
and *mono-l2*: $((\leq_{L2}) \Rightarrow (\leq_{R2}))$ *l2*
and *refl-L2*: *reflexive-on* (*in-dom* $(\leq_{L2})$) $(\leq_{L2})$
and *in-codom-rel-comp-le*: *in-codom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-codom* $(\leq_{R1})$
shows $(({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox)) \leq ({}_L\lessapprox)$
⟨*proof*⟩

corollary *left-Galois-eq-comp-left-GaloisI*:
assumes $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*
and $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
and $((\leq_{R1}) \trianglelefteq_h (\leq_{L1}))$ *r1 l1*
and *preorder-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and $((\leq_{L2}) \Rightarrow (\leq_{R2}))$ *l2*
and *reflexive-on* (*in-dom* $(\leq_{L2})$) $(\leq_{L2})$
and $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{R1}) \circ\circ (\leq_{L2}))$
and (*in-codom* $(\leq_R) \Rightarrow$ *in-codom* $(\leq_{R1})$) *r2*
and *in-codom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-codom* $(\leq_{R1})$
shows $({}_L\lessapprox) = (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$
⟨*proof*⟩

corollary *left-Galois-eq-comp-left-GaloisI'*:
assumes $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*

and $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
and $((\leq_{R1}) \trianglelefteq_h (\leq_{L1}))$ *r1 l1*
and *preorder-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and $((\leq_{L2}) \Rightarrow (\leq_{R2}))$ *l2*
and $((\leq_{R2}) \ {}_h\trianglelefteq (\leq_{L2}))$ *r2 l2*
and *reflexive-on* (*in-dom* $(\leq_{L2})$) $(\leq_{L2})$
and $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{R1}) \circ\circ (\leq_{L2}))$
and *in-codom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-codom* $(\leq_{R1})$
shows $({}_L\lessapprox) = (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$
⟨*proof*⟩

theorem *left-Galois-eq-comp-left-Galois-if-galois-connection-if-galois-equivalenceI'*:
assumes $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1*
and *preorder-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and $((\leq_{R2}) \dashv (\leq_{L2}))$ *r2 l2*
and *reflexive-on* (*in-dom* $(\leq_{L2})$) $(\leq_{L2})$
and $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{R1}) \circ\circ (\leq_{L2}))$
and *in-codom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-codom* $(\leq_{R1})$
shows $({}_L\lessapprox) = (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$
⟨*proof*⟩

corollary *left-Galois-eq-comp-left-Galois-if-galois-connection-if-galois-equivalenceI*:
assumes $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1*
and *preorder-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and $((\leq_{R2}) \dashv (\leq_{L2}))$ *r2 l2*
and *reflexive-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$
and *in-codom* $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-codom* $(\leq_{L2})$
and $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq ((\leq_{R1}) \circ\circ (\leq_{L2}))$
and *in-codom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-codom* $(\leq_{R1})$
shows $({}_L\lessapprox) = (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$
⟨*proof*⟩

corollary *left-Galois-eq-comp-left-Galois-if-preorder-equivalenceI*:
assumes $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and $((\leq_{R2}) \equiv_{pre} (\leq_{L2}))$ *r2 l2*
and *in-codom* $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-codom* $(\leq_{L2})$
and $(\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2}) \leq (\leq_{R1}) \circ\circ (\leq_{L2})$
and *in-codom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-codom* $(\leq_{R1})$
shows $({}_L\lessapprox) = (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$
⟨*proof*⟩

end

end

2.11.7 Basic Order Properties

theory *Transport-Compositions-Generic-Order-Base*

imports
Transport-Compositions-Generic-Base
begin

context *transport-comp*
begin

interpretation *flip1* : *galois R1 L1 r1 l1* ⟨*proof*⟩

Reflexivity

lemma *reflexive-on-in-dom-leftI*:
assumes *galois-prop*: $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
and *in-dom-L1-le*: *in-dom* $(\leq_{L1}) \leq$ *in-codom* $(\leq_{L1})$
and *refl-R1*: *reflexive-on* (*in-dom* $(\leq_{R1})$) $(\leq_{R1})$
and *refl-L2*: *reflexive-on* (*in-dom* $(\leq_{L2})$) $(\leq_{L2})$
and *mono-in-dom-l1*: (*in-dom* $(\leq_L) \Rightarrow$ *in-dom* $(\leq_{L2})$) *l1*
shows *reflexive-on* (*in-dom* $(\leq_L)$) $(\leq_L)$
⟨*proof*⟩

lemma *reflexive-on-in-codom-leftI*:
assumes *L1-r1-l1I*: $\bigwedge x.$ *in-dom* $(\leq_{L1})$ $x \Longrightarrow$ *l1* $x \leq_{R1}$ *l1* $x \Longrightarrow x \leq_{L1}$ *r1* (*l1* x)
and *in-codom-L1-le*: *in-codom* $(\leq_{L1}) \leq$ *in-dom* $(\leq_{L1})$
and *refl-R1*: *reflexive-on* (*in-codom* $(\leq_{R1})$) $(\leq_{R1})$
and *refl-L2*: *reflexive-on* (*in-codom* $(\leq_{L2})$) $(\leq_{L2})$
and *mono-in-codom-l1*: (*in-codom* $(\leq_L) \Rightarrow$ *in-codom* $(\leq_{L2})$) *l1*
shows *reflexive-on* (*in-codom* $(\leq_L)$) $(\leq_L)$
⟨*proof*⟩

corollary *reflexive-on-in-field-leftI*:
assumes $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
and *in-codom* $(\leq_{L1}) =$ *in-dom* $(\leq_{L1})$
and *reflexive-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and *reflexive-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$
and (*in-field* $(\leq_L) \Rightarrow$ *in-field* $(\leq_{L2})$) *l1*
shows *reflexive-on* (*in-field* $(\leq_L)$) $(\leq_L)$
⟨*proof*⟩

Transitivity

There are many similar proofs for transitivity. They slightly differ in their assumptions, particularly which of $(\leq_{R1})$ and $(\leq_{L2})$ has to be transitive and the order of commutativity for the relations.

In the following, we just give two of them that suffice for many purposes.

lemma *transitive-leftI*:
assumes $((\leq_{L1}) \ {}_h\trianglelefteq (\leq_{R1}))$ *l1 r1*
and *trans-L2*: *transitive* $(\leq_{L2})$
and *R1-L2-R1-le*: $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{L2}) \circ\circ (\leq_{R1}))$
shows *transitive* $(\leq_L)$

⟨*proof*⟩

lemma *transitive-leftI'*:
  assumes *galois-prop*: $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
  and *trans-L2*: *transitive* $(\leq_{L2})$
  and *R1-L2-R1-le*: $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{R1}) \circ\circ (\leq_{L2}))$
  shows *transitive* $(\leq_L)$
⟨*proof*⟩

Preorders

lemma *preorder-on-in-field-leftI*:
  assumes $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
  and *in-codom* $(\leq_{L1})$ = *in-dom* $(\leq_{L1})$
  and *reflexive-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
  and *preorder-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$
  and *mono-in-codom-l1*: (*in-codom* $(\leq_L) \Rightarrow$ *in-codom* $(\leq_{L2})$) *l1*
  and *R1-L2-R1-le*: $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{L2}) \circ\circ (\leq_{R1}))$
  shows *preorder-on* (*in-field* $(\leq_L)$) $(\leq_L)$
⟨*proof*⟩

lemma *preorder-on-in-field-leftI'*:
  assumes $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
  and *in-codom* $(\leq_{L1})$ = *in-dom* $(\leq_{L1})$
  and *reflexive-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
  and *preorder-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$
  and *mono-in-dom-l1*: (*in-dom* $(\leq_L) \Rightarrow$ *in-dom* $(\leq_{L2})$) *l1*
  and *R1-L2-R1-le*: $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{R1}) \circ\circ (\leq_{L2}))$
  shows *preorder-on* (*in-field* $(\leq_L)$) $(\leq_L)$
⟨*proof*⟩

Symmetry

lemma *symmetric-leftI*:
  assumes $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
  and *in-codom* $(\leq_{L1})$ = *in-dom* $(\leq_{L1})$
  and *symmetric* $(\leq_{R1})$
  and *symmetric* $(\leq_{L2})$
  shows *symmetric* $(\leq_L)$
⟨*proof*⟩

lemma *partial-equivalence-rel-leftI*:
  assumes $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
  and *in-codom* $(\leq_{L1})$ = *in-dom* $(\leq_{L1})$
  and *symmetric* $(\leq_{R1})$
  and *partial-equivalence-rel* $(\leq_{L2})$
  and $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{L2}) \circ\circ (\leq_{R1}))$
  shows *partial-equivalence-rel* $(\leq_L)$
  ⟨*proof*⟩

lemma *partial-equivalence-rel-leftI'*:
assumes $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
and *in-codom* $(\leq_{L1})$ = *in-dom* $(\leq_{L1})$
and *symmetric* $(\leq_{R1})$
and *partial-equivalence-rel* $(\leq_{L2})$
and $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{R1}) \circ\circ (\leq_{L2}))$
shows *partial-equivalence-rel* $(\leq_L)$
⟨*proof*⟩

end

end

2.11.8 Order Equivalence

theory *Transport-Compositions-Generic-Order-Equivalence*
imports
Transport-Compositions-Generic-Monotone
begin

context *transport-comp*
begin

context
begin

interpretation *flip* : *transport-comp R2 L2 r2 l2 R1 L1 r1 l1* ⟨*proof*⟩

Unit

Inflationary **lemma** *inflationary-on-in-dom-unitI*:
assumes $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*
and $((\leq_{L1}) \ _h\trianglelefteq (\leq_{R1}))$ *l1 r1*
and *inflationary-unit1*: *inflationary-on* (*in-dom* $(\leq_{L1})$) $(\leq_{L1})$ η_1
and *inflationary-counit1*: *inflationary-on* (*in-codom* $(\leq_{R1})$) $(\leq_{R1})$ ε_1
and *refl-R1*: *reflexive-on* (*in-dom* $(\leq_{R1})$) $(\leq_{R1})$
and *inflationary-unit2*: *inflationary-on* (*in-dom* $(\leq_{L2})$) $(\leq_{L2})$ η_2
and *refl-L2*: *reflexive-on* (*in-dom* $(\leq_{L2})$) $(\leq_{L2})$
and *mono-in-dom-l1*: (*in-dom* $(\leq_L)$ ⇒ *in-dom* $(\leq_{L2})$) *l1*
and *in-codom-rel-comp-le*: *in-codom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-codom* $((\leq_{R1}))$
shows *inflationary-on* (*in-dom* $(\leq_L)$) $(\leq_L)$ η
⟨*proof*⟩

lemma *inflationary-on-in-codom-unitI*:
assumes $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*
and *inflationary-unit1*: *inflationary-on* (*in-codom* $(\leq_{L1})$) $(\leq_{L1})$ η_1
and *inflationary-counit1*: *inflationary-on* (*in-codom* $(\leq_{R1})$) $(\leq_{R1})$ ε_1
and *refl-R1*: *reflexive-on* (*in-codom* $(\leq_{R1})$) $(\leq_{R1})$

and *inflationary-unit2*: *inflationary-on* (*in-codom* ($\leq_{L2}$)) ($\leq_{L2}$) η_2
and *refl-L2*: *reflexive-on* (*in-codom* ($\leq_{L2}$)) ($\leq_{L2}$)
and *mono-in-codom-l1*: (*in-codom* ($\leq_L$) $\Rightarrow$ *in-codom* ($\leq_{L2}$)) *l1*
and *in-codom-rel-comp-le*: *in-codom* (($\leq_{L2}$) $\circ\circ$ ($\leq_{R1}$) $\circ\circ$ ($\leq_{L2}$)) $\leq$ *in-codom* (($\leq_{R1}$))
shows *inflationary-on* (*in-codom* ($\leq_L$)) ($\leq_L$) η
⟨*proof*⟩

corollary *inflationary-on-in-field-unitI*:
assumes (($\leq_{R1}$) $\Rightarrow$ ($\leq_{L1}$)) *r1*
and (($\leq_{L1}$) $_h\trianglelefteq$ ($\leq_{R1}$)) *l1 r1*
and *inflationary-on* (*in-field* ($\leq_{L1}$)) ($\leq_{L1}$) η_1
and *inflationary-on* (*in-codom* ($\leq_{R1}$)) ($\leq_{R1}$) ε_1
and *reflexive-on* (*in-field* ($\leq_{R1}$)) ($\leq_{R1}$)
and *inflationary-on* (*in-field* ($\leq_{L2}$)) ($\leq_{L2}$) η_2
and *reflexive-on* (*in-field* ($\leq_{L2}$)) ($\leq_{L2}$)
and (*in-dom* ($\leq_L$) $\Rightarrow$ *in-dom* ($\leq_{L2}$)) *l1*
and (*in-codom* ($\leq_L$) $\Rightarrow$ *in-codom* ($\leq_{L2}$)) *l1*
and *in-codom* (($\leq_{L2}$) $\circ\circ$ ($\leq_{R1}$) $\circ\circ$ ($\leq_{L2}$)) $\leq$ *in-codom* (($\leq_{R1}$))
shows *inflationary-on* (*in-field* ($\leq_L$)) ($\leq_L$) η
⟨*proof*⟩

Deflationary

lemma *deflationary-on-in-dom-unitI*:
assumes (($\leq_{L1}$) $\Rightarrow$ ($\leq_{R1}$)) *l1* (($\leq_{R1}$) $\Rightarrow$ ($\leq_{L1}$)) *r1*
and *refl-L1*: *reflexive-on* (*in-dom* ($\leq_{L1}$)) ($\leq_{L1}$)
and *in-dom-R1-le-in-codom-R1*: *in-dom* ($\leq_{R1}$) $\leq$ *in-codom* ($\leq_{R1}$)
and *deflationary-L2*: *deflationary-on* (*in-dom* ($\leq_{L2}$)) ($\leq_{L2}$) η_2
and *refl-L2*: *reflexive-on* (*in-dom* ($\leq_{L2}$)) ($\leq_{L2}$)
and *mono-in-dom-l1*: (*in-dom* ($\leq_L$) $\Rightarrow$ *in-dom* ($\leq_{L2}$)) *l1*
and *in-dom-rel-comp-le*: *in-dom* (($\leq_{L2}$) $\circ\circ$ ($\leq_{R1}$) $\circ\circ$ ($\leq_{L2}$)) $\leq$ *in-dom* (($\leq_{R1}$))
shows *deflationary-on* (*in-dom* ($\leq_L$)) ($\leq_L$) η
⟨*proof*⟩

lemma *deflationary-on-in-codom-unitI*:
assumes (($\leq_{L1}$) $\Rightarrow$ ($\leq_{R1}$)) *l1* (($\leq_{R1}$) $\Rightarrow$ ($\leq_{L1}$)) *r1*
and *refl-L1*: *reflexive-on* (*in-codom* ($\leq_{L1}$)) ($\leq_{L1}$)
and *in-dom-R1-le-in-codom-R1*: *in-dom* ($\leq_{R1}$) $\leq$ *in-codom* ($\leq_{R1}$)
and *deflationary-L2*: *deflationary-on* (*in-codom* ($\leq_{L2}$)) ($\leq_{L2}$) η_2
and *refl-L2*: *reflexive-on* (*in-codom* ($\leq_{L2}$)) ($\leq_{L2}$)
and *mono-in-codom-l1*: (*in-codom* ($\leq_L$) $\Rightarrow$ *in-codom* ($\leq_{L2}$)) *l1*
and *in-dom-rel-comp-le*: *in-dom* (($\leq_{L2}$) $\circ\circ$ ($\leq_{R1}$) $\circ\circ$ ($\leq_{L2}$)) $\leq$ *in-dom* (($\leq_{R1}$))
shows *deflationary-on* (*in-codom* ($\leq_L$)) ($\leq_L$) η
⟨*proof*⟩

corollary *deflationary-on-in-field-unitI*:
assumes (($\leq_{L1}$) $\Rightarrow$ ($\leq_{R1}$)) *l1* (($\leq_{R1}$) $\Rightarrow$ ($\leq_{L1}$)) *r1*
and *reflexive-on* (*in-field* ($\leq_{L1}$)) ($\leq_{L1}$)
and *in-dom* ($\leq_{R1}$) $\leq$ *in-codom* ($\leq_{R1}$)

and *deflationary-on* (*in-field* ($\leq_{L2}$)) ($\leq_{L2}$) η_2
and *reflexive-on* (*in-field* ($\leq_{L2}$)) ($\leq_{L2}$)
and (*in-dom* ($\leq_L$) $\Rightarrow$ *in-dom* ($\leq_{L2}$)) *l1*
and (*in-codom* ($\leq_L$) $\Rightarrow$ *in-codom* ($\leq_{L2}$)) *l1*
and *in-dom* (($\leq_{L2}$) $\circ\circ$ ($\leq_{R1}$) $\circ\circ$ ($\leq_{L2}$)) $\leq$ *in-dom* (($\leq_{R1}$))
shows *deflationary-on* (*in-field* ($\leq_L$)) ($\leq_L$) η
⟨*proof*⟩

Relational Equivalence

corollary *rel-equivalence-on-in-field-unitI*:
assumes (($\leq_{L1}$) $\Rightarrow$ ($\leq_{R1}$)) *l1* (($\leq_{R1}$) $\Rightarrow$ ($\leq_{L1}$)) *r1*
and (($\leq_{L1}$) $_h\trianglelefteq$ ($\leq_{R1}$)) *l1 r1*
and *inflationary-on* (*in-field* ($\leq_{L1}$)) ($\leq_{L1}$) η_1
and *inflationary-on* (*in-codom* ($\leq_{R1}$)) ($\leq_{R1}$) ε_1
and *reflexive-on* (*in-field* ($\leq_{L1}$)) ($\leq_{L1}$)
and *reflexive-on* (*in-field* ($\leq_{R1}$)) ($\leq_{R1}$)
and *rel-equivalence-on* (*in-field* ($\leq_{L2}$)) ($\leq_{L2}$) η_2
and *reflexive-on* (*in-field* ($\leq_{L2}$)) ($\leq_{L2}$)
and (*in-dom* ($\leq_L$) $\Rightarrow$ *in-dom* ($\leq_{L2}$)) *l1*
and (*in-codom* ($\leq_L$) $\Rightarrow$ *in-codom* ($\leq_{L2}$)) *l1*
and *in-dom* (($\leq_{L2}$) $\circ\circ$ ($\leq_{R1}$) $\circ\circ$ ($\leq_{L2}$)) $\leq$ *in-dom* (($\leq_{R1}$))
and *in-codom* (($\leq_{L2}$) $\circ\circ$ ($\leq_{R1}$) $\circ\circ$ ($\leq_{L2}$)) $\leq$ *in-codom* (($\leq_{R1}$))
shows *rel-equivalence-on* (*in-field* ($\leq_L$)) ($\leq_L$) η
⟨*proof*⟩

Counit

Corresponding lemmas for the counit can be obtained by flipping the interpretation of the locale, i.e. *interpretation flip* : *transport-comp R2 L2 r2 l2 R1 L1 r1 l1 rewrites flip.t2.unit* $\equiv \varepsilon_1$ *and flip.t2.counit* $\equiv \eta_1$ *and flip.t1.unit* $\equiv \varepsilon_2$ *and flip.t1.counit* $\equiv \eta_2$ *and flip.unit* $\equiv \varepsilon$ *and flip.counit* $\equiv \eta$ *unfolding transport-comp.transport-defs by* (*auto simp*: *order-functors.flip-counit-eq-unit*)

end

Order Equivalence

interpretation *flip* : *transport-comp R2 L2 r2 l2 R1 L1 r1 l1*
rewrites *flip.t2.unit* $\equiv \varepsilon_1$ **and** *flip.t2.counit* $\equiv \eta_1$
and *flip.t1.unit* $\equiv \varepsilon_2$ **and** *flip.t1.counit* $\equiv \eta_2$
and *flip.counit* $\equiv \eta$ **and** *flip.unit* $\equiv \varepsilon$
⟨*proof*⟩

lemma *order-equivalenceI*:
assumes (($\leq_{L1}$) $\Rightarrow$ ($\leq_{R1}$)) *l1* (($\leq_{R1}$) $\Rightarrow$ ($\leq_{L1}$)) *r1*
and (($\leq_{L1}$) $_h\trianglelefteq$ ($\leq_{R1}$)) *l1 r1*
and *inflationary-on* (*in-field* ($\leq_{L1}$)) ($\leq_{L1}$) η_1
and *rel-equiv-counit1*: *rel-equivalence-on* (*in-field* ($\leq_{R1}$)) ($\leq_{R1}$) ε_1
and *reflexive-on* (*in-field* ($\leq_{L1}$)) ($\leq_{L1}$)
and *reflexive-on* (*in-field* ($\leq_{R1}$)) ($\leq_{R1}$)

and $((\leq_{R2}) \Rightarrow (\leq_{L2}))$ *r2* $((\leq_{L2}) \Rightarrow (\leq_{R2}))$ *l2*
and $((\leq_{R2}) \ _h\trianglelefteq (\leq_{L2}))$ *r2 l2*
and *rel-equiv-unit2*: *rel-equivalence-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$ η_2
and *inflationary-on* (*in-field* $(\leq_{R2})$) $(\leq_{R2})$ ε_2
and *reflexive-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$
and *reflexive-on* (*in-field* $(\leq_{R2})$) $(\leq_{R2})$
and *middle-compatible*: *middle-compatible-codom*
shows $((\leq_L) \equiv_o (\leq_R))$ *l r*
⟨*proof*⟩

corollary *order-equivalence-if-order-equivalenceI*:
assumes $((\leq_{L1}) \equiv_o (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and *transitive* $(\leq_{R1})$
and $((\leq_{L2}) \equiv_o (\leq_{R2}))$ *l2 r2*
and *transitive* $(\leq_{L2})$
and *reflexive-on* (*in-field* $(\leq_{R2})$) $(\leq_{R2})$
and *middle-compatible-codom*
shows $((\leq_L) \equiv_o (\leq_R))$ *l r*
⟨*proof*⟩

corollary *order-equivalence-if-galois-equivalenceI*:
assumes $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and *reflexive-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and $((\leq_{L2}) \equiv_G (\leq_{R2}))$ *l2 r2*
and *reflexive-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$
and *reflexive-on* (*in-field* $(\leq_{R2})$) $(\leq_{R2})$
and *middle-compatible-codom*
shows $((\leq_L) \equiv_o (\leq_R))$ *l r*
⟨*proof*⟩

end

end

theory *Transport-Compositions-Generic*
imports
Transport-Compositions-Generic-Galois-Equivalence
Transport-Compositions-Generic-Galois-Relator
Transport-Compositions-Generic-Order-Base
Transport-Compositions-Generic-Order-Equivalence
begin

Summary of Main Results

Closure of Order and Galois Concepts **context** *transport-comp*
begin

interpretation *flip* : *transport-comp R2 L2 r2 l2 R1 L1 r1 l1* ⟨*proof*⟩

lemma *preorder-galois-connection-if-galois-equivalenceI*:
assumes $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and *preorder-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and $((\leq_{L2}) \equiv_G (\leq_{R2}))$ *l2 r2*
and *preorder-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$
and *reflexive-on* (*in-field* $(\leq_{R2})$) $(\leq_{R2})$
and *middle-compatible-codom*
shows $((\leq_L) \dashv_{pre} (\leq_R))$ *l r*
⟨*proof*⟩

theorem *preorder-galois-connection-if-preorder-equivalenceI*:
assumes $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and $((\leq_{L2}) \equiv_{pre} (\leq_{R2}))$ *l2 r2*
and *middle-compatible-codom*
shows $((\leq_L) \dashv_{pre} (\leq_R))$ *l r*
⟨*proof*⟩

lemma *preorder-equivalence-if-galois-equivalenceI*:
assumes $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and *preorder-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and $((\leq_{L2}) \equiv_G (\leq_{R2}))$ *l2 r2*
and *preorder-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$
and *reflexive-on* (*in-field* $(\leq_{R2})$) $(\leq_{R2})$
and *middle-compatible-codom*
shows $((\leq_L) \equiv_{pre} (\leq_R))$ *l r*
⟨*proof*⟩

theorem *preorder-equivalenceI*:
assumes $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and $((\leq_{L2}) \equiv_{pre} (\leq_{R2}))$ *l2 r2*
and *middle-compatible-codom*
shows $((\leq_L) \equiv_{pre} (\leq_R))$ *l r*
⟨*proof*⟩

theorem *partial-equivalence-rel-equivalenceI*:
assumes $((\leq_{L1}) \equiv_{PER} (\leq_{R1}))$ *l1 r1*
and $((\leq_{L2}) \equiv_{PER} (\leq_{R2}))$ *l2 r2*
and *middle-compatible-codom*
shows $((\leq_L) \equiv_{PER} (\leq_R))$ *l r*
⟨*proof*⟩

Simplification of Galois relator **theorem** *left-Galois-eq-comp-left-GaloisI*:
assumes $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and $((\leq_{R2}) \dashv_{pre} (\leq_{L2}))$ *r2 l2*

and *middle-compatible-codom*
shows $({}_{L}\lessapprox) = (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$
⟨*proof*⟩

For theorems with weaker assumptions, see ⟦$((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*; *t1.galois-prop l1 r1*; *flip.t2.half-galois-prop-right*; *preorder-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$; $((\leq_{L2}) \Rightarrow (\leq_{R2}))$ *l2*; *flip.t1.half-galois-prop-left*; *reflexive-on* (*in-dom* $(\leq_{L2})$) $(\leq_{L2})$; $(\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1}) \leq (\leq_{R1}) \circ\circ (\leq_{L2})$; *in-codom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-codom* $(\leq_{R1})$⟧ $\Longrightarrow$ *flip.right-Galois* = *flip.t2.right-Galois* $\circ\circ$ *flip.t1.right-Galois*

⟦*t1.galois-equivalence*; *preorder-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$; *flip.t1.galois-connection*; *reflexive-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$; *in-codom* $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-codom* $(\leq_{L2})$; $(\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2}) \leq (\leq_{R1}) \circ\circ (\leq_{L2})$; *in-codom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-codom* $(\leq_{R1})$⟧ $\Longrightarrow$ *flip.right-Galois* = *flip.t2.right-Galois* $\circ\circ$ *flip.t1.right-Galois*.

Simplification of Compatibility Assumption See *Transport.Transport-Compositions-Gener*

end

end

2.12 Transport For Compositions

theory *Transport-Compositions*
imports
Transport-Compositions-Agree
Transport-Compositions-Generic
begin

Summary We provide two ways to compose transportable components: a slightly intricate, generic one in *transport-comp* and another straightforward but less general one in *transport-comp-agree*. As a special case from the latter, we obtain *transport-comp-same*, which includes the cases most prominently covered in the literature.

Refer to [2] for more details.

end

2.13 Reflexive Relator

theory *Reflexive-Relator*
imports
Galois-Equivalences
Galois-Relator
begin

definition *Refl-Rel* $R\ x\ y \equiv R\ x\ x \wedge R\ y\ y \wedge R\ x\ y$

open-bundle *Refl-Rel-syntax*
begin
notation *Refl-Rel* ($\langle$(-$^{\oplus}$)$\rangle$ [*1000*])
end

lemma *Refl-RelI* [*intro*]:
assumes $R\ x\ x$
and $R\ y\ y$
and $R\ x\ y$
shows $R^{\oplus}\ x\ y$
⟨*proof*⟩

lemma *Refl-Rel-selfI* [*intro*]:
assumes $R\ x\ x$
shows $R^{\oplus}\ x\ x$
⟨*proof*⟩

lemma *Refl-RelE* [*elim*]:
assumes $R^{\oplus}\ x\ y$
obtains $R\ x\ x\ R\ y\ y\ R\ x\ y$
⟨*proof*⟩

lemma *Refl-Rel-reflexive-on-in-field* [*iff*]:
reflexive-on (*in-field* $R^{\oplus}$) $R^{\oplus}$
⟨*proof*⟩

lemma *Refl-Rel-le-self* [*iff*]: $R^{\oplus} \leq R$ ⟨*proof*⟩

lemma *Refl-Rel-eq-self-if-reflexive-on* [*simp*]:
assumes *reflexive-on* (*in-field* R) R
shows $R^{\oplus} = R$
⟨*proof*⟩

lemma *reflexive-on-in-field-if-Refl-Rel-eq-self*:
assumes $R^{\oplus} = R$
shows *reflexive-on* (*in-field* R) R
⟨*proof*⟩

corollary *Refl-Rel-eq-self-iff-reflexive-on*:
$R^{\oplus} = R \longleftrightarrow$ *reflexive-on* (*in-field* R) R
⟨*proof*⟩

lemma *Refl-Rel-Refl-Rel-eq* [*simp*]: $(R^{\oplus})^{\oplus} = R^{\oplus}$
⟨*proof*⟩

lemma *rel-inv-Refl-Rel-eq* [*simp*]: $(R^{\oplus})^{-1} = (R^{-1})^{\oplus}$

⟨*proof*⟩

lemma *Refl-Rel-transitive-onI* [*intro*]:
assumes *transitive-on* (*P* :: *'a* ⇒ *bool*) (*R* :: *'a* ⇒ -)
shows *transitive-on P* $R^{\oplus}$
⟨*proof*⟩

corollary *Refl-Rel-transitiveI* [*intro*]:
assumes *transitive R*
shows *transitive* $R^{\oplus}$
⟨*proof*⟩

corollary *Refl-Rel-preorder-onI*:
assumes *transitive-on P R*
and *P* ≤ *in-field* $R^{\oplus}$
shows *preorder-on P* $R^{\oplus}$
⟨*proof*⟩

corollary *Refl-Rel-preorder-on-in-fieldI* [*intro*]:
assumes *transitive R*
shows *preorder-on* (*in-field* $R^{\oplus}$) $R^{\oplus}$
⟨*proof*⟩

lemma *Refl-Rel-symmetric-onI* [*intro*]:
assumes *symmetric-on* (*P* :: *'a* ⇒ *bool*) (*R* :: *'a* ⇒ -)
shows *symmetric-on P* $R^{\oplus}$
⟨*proof*⟩

lemma *Refl-Rel-symmetricI* [*intro*]:
assumes *symmetric R*
shows *symmetric* $R^{\oplus}$
⟨*proof*⟩

lemma *Refl-Rel-partial-equivalence-rel-onI* [*intro*]:
assumes *partial-equivalence-rel-on* (*P* :: *'a* ⇒ *bool*) (*R* :: *'a* ⇒ -)
shows *partial-equivalence-rel-on P* $R^{\oplus}$
⟨*proof*⟩

lemma *Refl-Rel-partial-equivalence-relI* [*intro*]:
assumes *partial-equivalence-rel R*
shows *partial-equivalence-rel* $R^{\oplus}$
⟨*proof*⟩

lemma *Refl-Rel-app-leftI*:
assumes *R* (*f x*) *y*
and *in-field* $S^{\oplus}$ *x*
and *in-field* $R^{\oplus}$ *y*
and (*S* ⇒ *R*) *f*
shows $R^{\oplus}$ (*f x*) *y*

⟨*proof*⟩

corollary *Refl-Rel-app-rightI*:
assumes $R\ x\ (f\ y)$
and *in-field* $S^{\oplus}\ y$
and *in-field* $R^{\oplus}\ x$
and $(S \Rightarrow R)\ f$
shows $R^{\oplus}\ x\ (f\ y)$
⟨*proof*⟩

lemma *mono-wrt-rel-Refl-Rel-Refl-Rel-if-mono-wrt-rel* [*intro*]:
assumes $(R \Rightarrow S)\ f$
shows $(R^{\oplus} \Rightarrow S^{\oplus})\ f$
⟨*proof*⟩

context *galois*
begin

interpretation *gR* : *galois* $(\leq_L)^{\oplus}\ (\leq_R)^{\oplus}\ l\ r$ ⟨*proof*⟩

lemma *Galois-Refl-RelI*:
assumes $((\leq_R) \Rightarrow (\leq_L))\ r$
and *in-field* $(\leq_L)^{\oplus}\ x$
and *in-field* $(\leq_R)^{\oplus}\ y$
and *in-codom* $(\leq_R)\ y \Longrightarrow x\ {}_L\lessapprox y$
shows (*galois-rel.Galois* $((\leq_L)^{\oplus})\ ((\leq_R)^{\oplus})\ r)\ x\ y$
⟨*proof*⟩

lemma *half-galois-prop-left-Refl-Rel-left-rightI*:
assumes $((\leq_L) \Rightarrow (\leq_R))\ l$
and $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$
shows $((\leq_L)^{\oplus}\ {}_h\trianglelefteq\ (\leq_R)^{\oplus})\ l\ r$
⟨*proof*⟩

interpretation *flip-inv* : *galois* $(\geq_R)\ (\geq_L)\ r\ l$
rewrites $((\geq_R) \Rightarrow (\geq_L)) \equiv ((\leq_R) \Rightarrow (\leq_L))$
and $\bigwedge R.\ (R^{-1})^{\oplus} \equiv (R^{\oplus})^{-1}$
and $\bigwedge R\ S\ f\ g.\ (R^{-1}\ {}_h\trianglelefteq\ S^{-1})\ f\ g \equiv (S\ \trianglelefteq_h\ R)\ g\ f$
⟨*proof*⟩

lemma *half-galois-prop-right-Refl-Rel-right-leftI*:
assumes $((\leq_R) \Rightarrow (\leq_L))\ r$
and $((\leq_L)\ \trianglelefteq_h\ (\leq_R))\ l\ r$
shows $((\leq_L)^{\oplus}\ \trianglelefteq_h\ (\leq_R)^{\oplus})\ l\ r$
⟨*proof*⟩

corollary *galois-prop-Refl-Rel-left-rightI*:
assumes $((\leq_L) \dashv (\leq_R))\ l\ r$
shows $((\leq_L)^{\oplus} \trianglelefteq (\leq_R)^{\oplus})\ l\ r$

⟨*proof*⟩

lemma *galois-connection-Refl-Rel-left-rightI*:
assumes $((\leq_L) \dashv (\leq_R))$ *l r*
shows $((\leq_L)^{\oplus} \dashv (\leq_R)^{\oplus})$ *l r*
⟨*proof*⟩

lemma *galois-equivalence-Refl-RelI*:
assumes $((\leq_L) \equiv_G (\leq_R))$ *l r*
shows $((\leq_L)^{\oplus} \equiv_G (\leq_R)^{\oplus})$ *l r*
⟨*proof*⟩

end

context *order-functors*
begin

lemma *inflationary-on-in-field-Refl-Rel-left*:
assumes $((\leq_L) \Rightarrow (\leq_R))$ *l*
and $((\leq_R) \Rightarrow (\leq_L))$ *r*
and *inflationary-on* (*in-dom* $(\leq_L)$) $(\leq_L)$ η
shows *inflationary-on* (*in-field* $(\leq_L)^{\oplus}$) $(\leq_L)^{\oplus}$ η
⟨*proof*⟩

lemma *inflationary-on-in-field-Refl-Rel-left'*:
assumes $((\leq_L) \Rightarrow (\leq_R))$ *l*
and $((\leq_R) \Rightarrow (\leq_L))$ *r*
and *inflationary-on* (*in-codom* $(\leq_L)$) $(\leq_L)$ η
shows *inflationary-on* (*in-field* $(\leq_L)^{\oplus}$) $(\leq_L)^{\oplus}$ η
⟨*proof*⟩

interpretation *inv* : *galois* $(\geq_L)$ $(\geq_R)$ *l r*
rewrites $((\geq_L) \Rightarrow (\geq_R)) \equiv ((\leq_L) \Rightarrow (\leq_R))$
and $((\geq_R) \Rightarrow (\geq_L)) \equiv ((\leq_R) \Rightarrow (\leq_L))$
and $\bigwedge R.\ (R^{-1})^{\oplus} \equiv (R^{\oplus})^{-1}$
and $\bigwedge R.$ *in-dom* $R^{-1} \equiv$ *in-codom* R
and $\bigwedge R.$ *in-codom* $R^{-1} \equiv$ *in-dom* R
and $\bigwedge R.$ *in-field* $R^{-1} \equiv$ *in-field* R
and $\bigwedge(P :: 'c \Rightarrow bool)\ (R :: 'd \Rightarrow 'c \Rightarrow bool).$
(*inflationary-on* P $R^{-1} :: ('c \Rightarrow 'd) \Rightarrow bool) \equiv$ *deflationary-on* P R
⟨*proof*⟩

lemma *deflationary-on-in-field-Refl-Rel-leftI*:
assumes $((\leq_L) \Rightarrow (\leq_R))$ *l*
and $((\leq_R) \Rightarrow (\leq_L))$ *r*
and *deflationary-on* (*in-dom* $(\leq_L)$) $(\leq_L)$ η
shows *deflationary-on* (*in-field* $(\leq_L)^{\oplus}$) $(\leq_L)^{\oplus}$ η
⟨*proof*⟩

lemma *deflationary-on-in-field-Refl-RelI-left'*:
assumes $((\leq_L) \Rightarrow (\leq_R))\ l$
and $((\leq_R) \Rightarrow (\leq_L))\ r$
and *deflationary-on* (*in-codom* $(\leq_L)$) $(\leq_L)\ \eta$
shows *deflationary-on* (*in-field* $(\leq_L)^{\oplus}$) $(\leq_L)^{\oplus}\ \eta$
⟨*proof*⟩

lemma *rel-equivalence-on-in-field-Refl-Rel-leftI*:
assumes $((\leq_L) \Rightarrow (\leq_R))\ l$
and $((\leq_R) \Rightarrow (\leq_L))\ r$
and *rel-equivalence-on* (*in-dom* $(\leq_L)$) $(\leq_L)\ \eta$
shows *rel-equivalence-on* (*in-field* $(\leq_L)^{\oplus}$) $(\leq_L)^{\oplus}\ \eta$
⟨*proof*⟩

lemma *rel-equivalence-on-in-field-Refl-Rel-leftI'*:
assumes $((\leq_L) \Rightarrow (\leq_R))\ l$
and $((\leq_R) \Rightarrow (\leq_L))\ r$
and *rel-equivalence-on* (*in-codom* $(\leq_L)$) $(\leq_L)\ \eta$
shows *rel-equivalence-on* (*in-field* $(\leq_L)^{\oplus}$) $(\leq_L)^{\oplus}\ \eta$
⟨*proof*⟩

interpretation *oR* : *order-functors* $(\leq_L)^{\oplus}$ $(\leq_R)^{\oplus}\ l\ r$ ⟨*proof*⟩

lemma *order-equivalence-Refl-RelI*:
assumes $((\leq_L) \equiv_o (\leq_R))\ l\ r$
shows $((\leq_L)^{\oplus} \equiv_o (\leq_R)^{\oplus})\ l\ r$
⟨*proof*⟩

end

end

2.14 Monotone Function Relator

theory *Monotone-Function-Relator*
imports
Reflexive-Relator
begin

abbreviation *Mono-Dep-Fun-Rel* $(R :: 'a \Rightarrow 'a \Rightarrow bool)$ $(S :: 'a \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b \Rightarrow bool) \equiv$
$((x\ y :: R) \Rrightarrow S\ x\ y)^{\oplus}$
abbreviation *Mono-Fun-Rel* $R\ S \equiv$ *Mono-Dep-Fun-Rel* R $(\lambda\text{-}\ \text{-}.\ S)$

open-bundle *Mono-Dep-Fun-Rel-syntax*
begin
notation *Mono-Fun-Rel* (**infixr** ‹⇛⊕› *40*)
syntax

-Mono-Dep-Fun-Rel-rel :: *idt* ⇒ *idt* ⇒ (*'a* ⇒ *'b* ⇒ *bool*) ⇒ (*'c* ⇒ *'d* ⇒ *bool*) ⇒
(*'a* ⇒ *'c*) ⇒ (*'b* ⇒ *'d*) ⇒ *bool* (‹*'*(*-/ -/ ::/ -*') ⇛⊕ (*-*)› [*41*, *41*, *41*, *40*] *40*)
-Mono-Dep-Fun-Rel-rel-if :: *idt* ⇒ *idt* ⇒ (*'a* ⇒ *'b* ⇒ *bool*) ⇒ *bool* ⇒ (*'c* ⇒ *'d*
⇒ *bool*) ⇒
(*'a* ⇒ *'c*) ⇒ (*'b* ⇒ *'d*) ⇒ *bool* (‹*'*(*-/ -/ ::/ -/ |/ -*') ⇛⊕ (*-*)› [*41*, *41*, *41*, *41*,
40] *40*)
end
syntax-consts
-Mono-Dep-Fun-Rel-rel -Mono-Dep-Fun-Rel-rel-if ⇌ *Mono-Dep-Fun-Rel*
translations
(*x y* :: *R*) ⇛⊕ *S* ⇌ *CONST Mono-Dep-Fun-Rel R* (λ*x y*. *S*)
(*x y* :: *R* | *B*) ⇛⊕ *S* ⇌ *CONST Mono-Dep-Fun-Rel R* (λ*x y*. *CONST rel-if B S*)

locale *Dep-Fun-Rel-orders* =
fixes *L* :: *'a* ⇒ *'b* ⇒ *bool*
and *R* :: *'a* ⇒ *'b* ⇒ *'c* ⇒ *'d* ⇒ *bool*
begin

sublocale *o* : *orders L R a b* **for** *a b* ⟨*proof*⟩

notation *L* (**infix** ‹≤$_L$› *50*)
notation *o.ge-left* (**infix** ‹≥$_L$› *50*)

notation *R* (‹(≤$_R$ (*-*) (*-*))› *50*)
abbreviation *right-infix c a b d* ≡ (≤$_R$ *a b*) *c d*
notation *right-infix* (‹(*-*) ≤$_R$ (*-*) (*-*) (*-*)› [*51*,*51*,*51*,*51*] *50*)

notation *o.ge-right* (‹(≥$_R$ (*-*) (*-*))› *50*)

abbreviation (*input*) *ge-right-infix d a b c* ≡ (≥$_R$ *a b*) *d c*
notation *ge-right-infix* (‹(*-*) ≥$_R$ (*-*) (*-*) (*-*)› [*51*,*51*,*51*,*51*] *50*)

abbreviation (*input*) *DFR* ≡ ((*a b* :: *L*) ⇛ *R a b*)

end

locale *hom-Dep-Fun-Rel-orders* = *Dep-Fun-Rel-orders L R*
for *L* :: *'a* ⇒ *'a* ⇒ *bool*
and *R* :: *'a* ⇒ *'a* ⇒ *'b* ⇒ *'b* ⇒ *bool*
begin

sublocale *ho* : *hom-orders L R a b* **for** *a b* ⟨*proof*⟩

lemma *Mono-Dep-Fun-Refl-Rel-right-eq-Mono-Dep-Fun-if-le-if-reflexive-onI*:
assumes *refl-L*: *reflexive-on* (*in-field* (≤$_L$)) (≤$_L$)
and ⋀*x1 x2*. *x1* ≤$_L$ *x2* ⟹ (≤$_R$ *x2 x2*) ≤ (≤$_R$ *x1 x2*)
and ⋀*x1 x2*. *x1* ≤$_L$ *x2* ⟹ (≤$_R$ *x1 x1*) ≤ (≤$_R$ *x1 x2*)
shows ((*x y* :: (≤$_L$)) ⇛⊕ (≤$_R$ *x y*)$^{\oplus}$) = ((*x y* :: (≤$_L$)) ⇛⊕ (≤$_R$ *x y*))
⟨*proof*⟩

lemma *Mono-Dep-Fun-Refl-Rel-right-eq-Mono-Dep-Fun-if-mono-if-reflexive-onI*:
assumes *reflexive-on* (*in-field* ($\leq_L$)) ($\leq_L$)
and ((*x1 x2* :: ($\geq_L$)) $\Rightarrow$ (*x3 x4* :: ($\leq_L$) | *x1* $\leq_L$ *x3*) $\Rrightarrow$ ($\leq$)) *R*
shows ((*x y* :: ($\leq_L$)) $\Rrightarrow\oplus$ ($\leq_R$ *x y*)$^{\oplus}$) = ((*x y* :: ($\leq_L$)) $\Rrightarrow\oplus$ ($\leq_R$ *x y*))
⟨*proof*⟩

end

context *hom-orders*
begin

sublocale *fro* : *hom-Dep-Fun-Rel-orders L* λ- -. *R* ⟨*proof*⟩

corollary *Mono-Fun-Rel-Refl-Rel-right-eq-Mono-Fun-RelI*:
assumes *reflexive-on* (*in-field* ($\leq_L$)) ($\leq_L$)
shows (($\leq_L$) $\Rrightarrow\oplus$ ($\leq_R$)$^{\oplus}$) = (($\leq_L$) $\Rrightarrow\oplus$ ($\leq_R$))
⟨*proof*⟩

end

end

2.15 Transport For Functions

2.15.1 Basic Setup

theory *Transport-Functions-Base*
imports
Monotone-Function-Relator
Transport-Base
begin

Summary Basic setup for closure proofs. We introduce locales for the syntax, the dependent relator, the non-dependent relator, the monotone dependent relator, and the monotone non-dependent relator.

definition *flip2 f x1 x2 x3 x4* ≡ *f x2 x1 x4 x3*

lemma *flip2-eq*: *flip2 f x1 x2 x3 x4* = *f x2 x1 x4 x3*
⟨*proof*⟩

lemma *flip2-eq-rel-inv* [*simp*]: *flip2 R x y* = (*R y x*)$^{-1}$
⟨*proof*⟩

lemma *flip2-flip2-eq-self* [*simp*]: *flip2* (*flip2 f*) = *f*
⟨*proof*⟩

lemma *flip2-eq-flip2-iff-eq* [*iff*]: *flip2 f* = *flip2 g* ⟷ *f* = *g*
⟨*proof*⟩

Dependent Function Relator **locale** *transport-Dep-Fun-Rel-syntax* =
t1 : *transport L1 R1 l1 r1* +
dfro1 : *hom-Dep-Fun-Rel-orders L1 L2* +
dfro2 : *hom-Dep-Fun-Rel-orders R1 R2*
for *L1* :: *'a1* ⇒ *'a1* ⇒ *bool*
and *R1* :: *'a2* ⇒ *'a2* ⇒ *bool*
and *l1* :: *'a1* ⇒ *'a2*
and *r1* :: *'a2* ⇒ *'a1*
and *L2* :: *'a1* ⇒ *'a1* ⇒ *'b1* ⇒ *'b1* ⇒ *bool*
and *R2* :: *'a2* ⇒ *'a2* ⇒ *'b2* ⇒ *'b2* ⇒ *bool*
and *l2* :: *'a2* ⇒ *'a1* ⇒ *'b1* ⇒ *'b2*
and *r2* :: *'a1* ⇒ *'a2* ⇒ *'b2* ⇒ *'b1*
begin

notation *L1* (**infix** ‹$\leq_{L1}$› *50*)
notation *R1* (**infix** ‹$\leq_{R1}$› *50*)

notation *t1.ge-left* (**infix** ‹$\geq_{L1}$› *50*)
notation *t1.ge-right* (**infix** ‹$\geq_{R1}$› *50*)

notation *t1.left-Galois* (**infix** ‹$_{L1}\lessapprox$› *50*)
notation *t1.ge-Galois-left* (**infix** ‹$\gtrapprox_{L1}$› *50*)
notation *t1.right-Galois* (**infix** ‹$_{R1}\lessapprox$› *50*)
notation *t1.ge-Galois-right* (**infix** ‹$\gtrapprox_{R1}$› *50*)
notation *t1.right-ge-Galois* (**infix** ‹$_{R1}\gtrapprox$› *50*)
notation *t1.Galois-right* (**infix** ‹$\lessapprox_{R1}$› *50*)
notation *t1.left-ge-Galois* (**infix** ‹$_{L1}\gtrapprox$› *50*)
notation *t1.Galois-left* (**infix** ‹$\lessapprox_{L1}$› *50*)

notation *t1.unit* (‹η_1›)
notation *t1.counit* (‹ε_1›)

notation *L2* (‹($\leq_{L2}$ (-) (-))› *50*)
notation *R2* (‹($\leq_{R2}$ (-) (-))› *50*)

notation *dfro1.right-infix* (‹(-) $\leq_{L2}$ (-) (-) (-)› [*51*,*51*,*51*,*51*] *50*)
notation *dfro2.right-infix* (‹(-) $\leq_{R2}$ (-) (-) (-)› [*51*,*51*,*51*,*51*] *50*)

notation *dfro1.o.ge-right* (‹($\geq_{L2}$ (-) (-))› *50*)
notation *dfro2.o.ge-right* (‹($\geq_{R2}$ (-) (-))› *50*)

notation *dfro1.ge-right-infix* (‹(-) $\geq_{L2}$ (-) (-) (-)› [*51*,*51*,*51*,*51*] *50*)
notation *dfro2.ge-right-infix* (‹(-) $\geq_{R2}$ (-) (-) (-)› [*51*,*51*,*51*,*51*] *50*)

notation *l2* (‹$l2_{(-)\ (-)}$›)

notation *r2* ($\langle r2_{(-)\ (-)}\rangle$)

sublocale *t2* : *transport* ($\leq_{L2\ x\ (r1\ x')}$) ($\leq_{R2\ (l1\ x)\ x'}$) $l2_{x'\ x}$ $r2_{x\ x'}$ **for** *x x'*
⟨*proof*⟩

notation *t2.left-Galois* ($\langle(_{L2\ (-)\ (-)}\lessapprox)\rangle$ *50*)
notation *t2.right-Galois* ($\langle(_{R2\ (-)\ (-)}\lessapprox)\rangle$ *50*)

abbreviation *left2-Galois-infix y x x' y'* ≡ $(_{L2\ x\ x'}\lessapprox)$ *y y'*
notation *left2-Galois-infix* ($\langle(-)\ _{L2\ (-)\ (-)}\lessapprox\ (-)\rangle$ [*51*,*51*,*51*,*51*] *50*)
abbreviation *right2-Galois-infix y' x x' y* ≡ $(_{R2\ x\ x'}\lessapprox)$ *y' y*
notation *right2-Galois-infix* ($\langle(-)\ _{R2\ (-)\ (-)}\lessapprox\ (-)\rangle$ [*51*,*51*,*51*,*51*] *50*)

notation *t2.ge-Galois-left* ($\langle(\gtrapprox_{L2\ (-)\ (-)})\rangle$ *50*)
notation *t2.ge-Galois-right* ($\langle(\gtrapprox_{R2\ (-)\ (-)})\rangle$ *50*)

abbreviation (*input*) *ge-Galois-left-left2-infix y' x x' y* ≡ $(\gtrapprox_{L2\ x\ x'})$ *y' y*
notation *ge-Galois-left-left2-infix* ($\langle(-)\ \gtrapprox_{L2\ (-)\ (-)}\ (-)\rangle$ [*51*,*51*,*51*,*51*] *50*)
abbreviation (*input*) *ge-Galois-left-right2-infix y x x' y'* ≡ $(\gtrapprox_{R2\ x\ x'})$ *y y'*
notation *ge-Galois-left-right2-infix* ($\langle(-)\ \gtrapprox_{R2\ (-)\ (-)}\ (-)\rangle$ [*51*,*51*,*51*,*51*] *50*)

notation *t2.right-ge-Galois* ($\langle(_{R2\ (-)\ (-)}\gtrapprox)\rangle$ *50*)
notation *t2.left-ge-Galois* ($\langle(_{L2\ (-)\ (-)}\gtrapprox)\rangle$ *50*)

abbreviation *left2-ge-Galois-left-infix y x x' y'* ≡ $(_{L2\ x\ x'}\gtrapprox)$ *y y'*
notation *left2-ge-Galois-left-infix* ($\langle(-)\ _{L2\ (-)\ (-)}\gtrapprox\ (-)\rangle$ [*51*,*51*,*51*,*51*] *50*)
abbreviation *right2-ge-Galois-left-infix y' x x' y* ≡ $(_{R2\ x\ x'}\gtrapprox)$ *y' y*
notation *right2-ge-Galois-left-infix* ($\langle(-)\ _{R2\ (-)\ (-)}\gtrapprox\ (-)\rangle$ [*51*,*51*,*51*,*51*] *50*)

notation *t2.Galois-right* ($\langle(\lessapprox_{R2\ (-)\ (-)})\rangle$ *50*)
notation *t2.Galois-left* ($\langle(\lessapprox_{L2\ (-)\ (-)})\rangle$ *50*)

abbreviation (*input*) *Galois-left2-infix y' x x' y* ≡ $(\lessapprox_{L2\ x\ x'})$ *y' y*
notation *Galois-left2-infix* ($\langle(-)\ \lessapprox_{L2\ (-)\ (-)}\ (-)\rangle$ [*51*,*51*,*51*,*51*] *50*)
abbreviation (*input*) *Galois-right2-infix y x x' y'* ≡ $(\lessapprox_{R2\ x\ x'})$ *y y'*
notation *Galois-right2-infix* ($\langle(-)\ \lessapprox_{R2\ (-)\ (-)}\ (-)\rangle$ [*51*,*51*,*51*,*51*] *50*)

abbreviation *t2-unit x x'* ≡ *t2.unit x' x*
notation *t2-unit* ($\langle\eta_{2\ (-)\ (-)}\rangle$)
abbreviation *t2-counit x x'* ≡ *t2.counit x' x*
notation *t2-counit* ($\langle\varepsilon_{2\ (-)\ (-)}\rangle$)

end

locale *transport-Dep-Fun-Rel* =
transport-Dep-Fun-Rel-syntax L1 R1 l1 r1 L2 R2 l2 r2
for *L1* :: *'a1* ⇒ *'a1* ⇒ *bool*

and $R1 :: 'a2 \Rightarrow 'a2 \Rightarrow bool$
and $l1 :: 'a1 \Rightarrow 'a2$
and $r1 :: 'a2 \Rightarrow 'a1$
and $L2 :: 'a1 \Rightarrow 'a1 \Rightarrow 'b1 \Rightarrow 'b1 \Rightarrow bool$
and $R2 :: 'a2 \Rightarrow 'a2 \Rightarrow 'b2 \Rightarrow 'b2 \Rightarrow bool$
and $l2 :: 'a2 \Rightarrow 'a1 \Rightarrow 'b1 \Rightarrow 'b2$
and $r2 :: 'a1 \Rightarrow 'a2 \Rightarrow 'b2 \Rightarrow 'b1$
begin

definition $L \equiv (x1\ x2 :: (\leq_{L1})) \Rrightarrow (\leq_{L2\ x1\ x2})$

lemma *left-rel-eq-Dep-Fun-Rel*: $L = ((x1\ x2 :: (\leq_{L1})) \Rrightarrow (\leq_{L2\ x1\ x2}))$
⟨*proof*⟩

definition $l \equiv ((x' : r1) \rightsquigarrow l2\ x')$

lemma *left-eq-dep-fun-map*: $l = ((x' : r1) \rightsquigarrow l2\ x')$
⟨*proof*⟩

lemma *left-eq* [*simp*]: $l\ f\ x' = l2_{x'\ (r1\ x')}\ (f\ (r1\ x'))$
⟨*proof*⟩

context
begin

interpretation *flip* : *transport-Dep-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2* ⟨*proof*⟩

abbreviation $R \equiv flip.L$
abbreviation $r \equiv flip.l$

lemma *right-rel-eq-Dep-Fun-Rel*: $R = ((x1'\ x2' :: (\leq_{R1})) \Rrightarrow (\leq_{R2\ x1'\ x2'}))$
⟨*proof*⟩

lemma *right-eq-dep-fun-map*: $r = ((x : l1) \rightsquigarrow r2\ x)$
⟨*proof*⟩

end

lemma *right-eq* [*simp*]: $r\ g\ x = r2_{x\ (l1\ x)}\ (g\ (l1\ x))$
⟨*proof*⟩

lemmas *transport-defs* = *left-rel-eq-Dep-Fun-Rel left-eq-dep-fun-map*
right-rel-eq-Dep-Fun-Rel right-eq-dep-fun-map

sublocale *transport L R l r* ⟨*proof*⟩

notation L (**infix** $\langle\leq_L\rangle$ *50*)
notation R (**infix** $\langle\leq_R\rangle$ *50*)

lemma *left-relI* [*intro*]:
assumes $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow f\ x1 \leq_{L2\ x1\ x2} f'\ x2$
shows $f \leq_L f'$
⟨*proof*⟩

lemma *left-relE* [*elim*]:
assumes $f \leq_L f'$
and $x1 \leq_{L1} x2$
obtains $f\ x1 \leq_{L2\ x1\ x2} f'\ x2$
⟨*proof*⟩

interpretation *flip-inv* :
transport-Dep-Fun-Rel $(\geq_{R1})$ $(\geq_{L1})$ *r1 l1 flip2 R2 flip2 L2 r2 l2* ⟨*proof*⟩

lemma *flip-inv-right-eq-ge-left*: *flip-inv.R* $= (\geq_L)$
⟨*proof*⟩

interpretation *flip* : *transport-Dep-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2* ⟨*proof*⟩

lemma *flip-inv-left-eq-ge-right*: *flip-inv.L* $\equiv (\geq_R)$
⟨*proof*⟩

Useful Rewritings for Dependent Relation **lemma** *left-rel2-unit-eqs-left-rel2I*:
assumes $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow (\leq_{L2\ (\eta_1\ x)\ x}) \leq (\leq_{L2\ x\ x})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow (\leq_{L2\ x\ (\eta_1\ x)}) \leq (\leq_{L2\ x\ x})$
and $x \leq_{L1} x$
and $x \equiv_{L1} \eta_1\ x$
shows $(\leq_{L2\ (\eta_1\ x)\ x}) = (\leq_{L2\ x\ x})$
and $(\leq_{L2\ x\ (\eta_1\ x)}) = (\leq_{L2\ x\ x})$
⟨*proof*⟩

lemma *left2-eq-if-bi-related-if-monoI*:
assumes *mono-L2*: $((x1\ x2 :: (\geq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rrightarrow (\leq))$ *L2*
and $x1 \leq_{L1} x2$
and $x1 \equiv_{L1} x3$
and $x2 \equiv_{L1} x4$
and *trans-L1*: *transitive* $(\leq_{L1})$
shows $(\leq_{L2\ x1\ x2}) = (\leq_{L2\ x3\ x4})$
⟨*proof*⟩

end

Function Relator **locale** *transport-Fun-Rel-syntax* =
tdfrs : *transport-Dep-Fun-Rel-syntax L1 R1 l1 r1* λ- -. *L2* λ- -. *R2*

```
    λ- -. l2 λ- -. r2
  for L1 :: 'a1 ⇒ 'a1 ⇒ bool
  and R1 :: 'a2 ⇒ 'a2 ⇒ bool
  and l1 :: 'a1 ⇒ 'a2
  and r1 :: 'a2 ⇒ 'a1
  and L2 :: 'b1 ⇒ 'b1 ⇒ bool
  and R2 :: 'b2 ⇒ 'b2 ⇒ bool
  and l2 :: 'b1 ⇒ 'b2
  and r2 :: 'b2 ⇒ 'b1
begin

notation L1 (infix ‹≤L1› 50)
notation R1 (infix ‹≤R1› 50)

notation tdfrs.t1.ge-left (infix ‹≥L1› 50)
notation tdfrs.t1.ge-right (infix ‹≥R1› 50)

notation tdfrs.t1.left-Galois (infix ‹L1≲› 50)
notation tdfrs.t1.ge-Galois-left (infix ‹≳L1› 50)
notation tdfrs.t1.right-Galois (infix ‹R1≲› 50)
notation tdfrs.t1.ge-Galois-right (infix ‹≳R1› 50)
notation tdfrs.t1.right-ge-Galois (infix ‹R1≳› 50)
notation tdfrs.t1.Galois-right (infix ‹≲R1› 50)
notation tdfrs.t1.left-ge-Galois (infix ‹L1≳› 50)
notation tdfrs.t1.Galois-left (infix ‹≲L1› 50)

notation tdfrs.t1.unit (‹η1›)
notation tdfrs.t1.counit (‹ε1›)

notation L2 (infix ‹≤L2› 50)
notation R2 (infix ‹≤R2› 50)

notation tdfrs.t2.ge-left (infix ‹≥L2› 50)
notation tdfrs.t2.ge-right (infix ‹≥R2› 50)

notation tdfrs.t2.left-Galois (infix ‹L2≲› 50)
notation tdfrs.t2.ge-Galois-left (infix ‹≳L2› 50)
notation tdfrs.t2.right-Galois (infix ‹R2≲› 50)
notation tdfrs.t2.ge-Galois-right (infix ‹≳R2› 50)
notation tdfrs.t2.right-ge-Galois (infix ‹R2≳› 50)
notation tdfrs.t2.Galois-right (infix ‹≲R2› 50)
notation tdfrs.t2.left-ge-Galois (infix ‹L2≳› 50)
notation tdfrs.t2.Galois-left (infix ‹≲L2› 50)

notation tdfrs.t2.unit (‹η2›)
notation tdfrs.t2.counit (‹ε2›)

end
```

locale *transport-Fun-Rel* =
  transport-Fun-Rel-syntax L1 R1 l1 r1 L2 R2 l2 r2 +
  tdfr : *transport-Dep-Fun-Rel L1 R1 l1 r1* λ- -. *L2* λ- -. *R2*
    λ- -. *l2* λ- -. *r2*
  for *L1* :: *'a1* ⇒ *'a1* ⇒ *bool*
  and *R1* :: *'a2* ⇒ *'a2* ⇒ *bool*
  and *l1* :: *'a1* ⇒ *'a2*
  and *r1* :: *'a2* ⇒ *'a1*
  and *L2* :: *'b1* ⇒ *'b1* ⇒ *bool*
  and *R2* :: *'b2* ⇒ *'b2* ⇒ *bool*
  and *l2* :: *'b1* ⇒ *'b2*
  and *r2* :: *'b2* ⇒ *'b1*
begin

notation *tdfr.L* (‹*L*›)
notation *tdfr.R* (‹*R*›)

abbreviation *l* ≡ *tdfr.l*
abbreviation *r* ≡ *tdfr.r*

notation *tdfr.L* (**infix** ‹$\leq_L$› *50*)
notation *tdfr.R* (**infix** ‹$\leq_R$› *50*)

notation *tdfr.ge-left* (**infix** ‹$\geq_L$› *50*)
notation *tdfr.ge-right* (**infix** ‹$\geq_R$› *50*)

notation *tdfr.left-Galois* (**infix** ‹${}_L\lessapprox$› *50*)
notation *tdfr.ge-Galois-left* (**infix** ‹$\gtrapprox_L$› *50*)
notation *tdfr.right-Galois* (**infix** ‹${}_R\lessapprox$› *50*)
notation *tdfr.ge-Galois-right* (**infix** ‹$\gtrapprox_R$› *50*)
notation *tdfr.right-ge-Galois* (**infix** ‹${}_R\gtrapprox$› *50*)
notation *tdfr.Galois-right* (**infix** ‹$\lessapprox_R$› *50*)
notation *tdfr.left-ge-Galois* (**infix** ‹${}_L\gtrapprox$› *50*)
notation *tdfr.Galois-left* (**infix** ‹$\lessapprox_L$› *50*)

notation *tdfr.unit* (‹*η*›)
notation *tdfr.counit* (‹*ε*›)

lemma *left-rel-eq-Fun-Rel*: $(\leq_L) = ((\leq_{L1}) \Rrightarrow (\leq_{L2}))$
  ⟨*proof*⟩

lemma *left-eq-fun-map*: *l* = (*r1* ⇝ *l2*)
  ⟨*proof*⟩

interpretation *flip* : *transport-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2* ⟨*proof*⟩

lemma *right-rel-eq-Fun-Rel*: $(\leq_R) = ((\leq_{R1}) \Rrightarrow (\leq_{R2}))$
  ⟨*proof*⟩

lemma *right-eq-fun-map*: $r = (l1 \rightsquigarrow r2)$
⟨*proof*⟩

lemmas *transport-defs* = *left-rel-eq-Fun-Rel right-rel-eq-Fun-Rel*
left-eq-fun-map right-eq-fun-map

end

Monotone Dependent Function Relator **locale** *transport-Mono-Dep-Fun-Rel* =
transport-Dep-Fun-Rel-syntax L1 R1 l1 r1 L2 R2 l2 r2
+ *tdfr* : *transport-Dep-Fun-Rel L1 R1 l1 r1 L2 R2 l2 r2*
for $L1 :: 'a1 \Rightarrow 'a1 \Rightarrow bool$
and $R1 :: 'a2 \Rightarrow 'a2 \Rightarrow bool$
and $l1 :: 'a1 \Rightarrow 'a2$
and $r1 :: 'a2 \Rightarrow 'a1$
and $L2 :: 'a1 \Rightarrow 'a1 \Rightarrow 'b1 \Rightarrow 'b1 \Rightarrow bool$
and $R2 :: 'a2 \Rightarrow 'a2 \Rightarrow 'b2 \Rightarrow 'b2 \Rightarrow bool$
and $l2 :: 'a2 \Rightarrow 'a1 \Rightarrow 'b1 \Rightarrow 'b2$
and $r2 :: 'a1 \Rightarrow 'a2 \Rightarrow 'b2 \Rightarrow 'b1$
begin

definition $L \equiv tdfr.L^{\oplus}$

lemma *left-rel-eq-tdfr-left-Refl-Rel*: $L = tdfr.L^{\oplus}$
⟨*proof*⟩

lemma *left-rel-eq-Mono-Dep-Fun-Rel*: $L = ((x1\ x2 :: (\leq_{L1})) \Rrightarrow\oplus (\leq_{L2\ x1\ x2}))$
⟨*proof*⟩

lemma *left-rel-eq-tdfr-left-rel-if-reflexive-on*:
assumes *reflexive-on* (*in-field tdfr.L*) *tdfr.L*
shows $L = tdfr.L$
⟨*proof*⟩

abbreviation $l \equiv tdfr.l$

lemma *left-eq-tdfr-left*: $l = tdfr.l$ ⟨*proof*⟩

interpretation *flip* : *transport-Mono-Dep-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2* ⟨*proof*⟩

abbreviation $R \equiv flip.L$

lemma *right-rel-eq-tdfr-right-Refl-Rel*: $R = tdfr.R^{\oplus}$
⟨*proof*⟩

lemma *right-rel-eq-Mono-Dep-Fun-Rel*: $R = ((y1\ y2 :: (\leq_{R1})) \Rrightarrow\oplus (\leq_{R2\ y1\ y2}))$
⟨*proof*⟩

lemma *right-rel-eq-tdfr-right-rel-if-reflexive-on*:
assumes *reflexive-on* (*in-field tdfr.R*) *tdfr.R*
shows *R* = *tdfr.R*
⟨*proof*⟩

abbreviation *r* ≡ *tdfr.r*

lemma *right-eq-tdfr-right*: *r* = *tdfr.r* ⟨*proof*⟩

lemmas *transport-defs* = *left-rel-eq-tdfr-left-Refl-Rel*
right-rel-eq-tdfr-right-Refl-Rel

sublocale *transport L R l r* ⟨*proof*⟩

notation *L* (**infix** ‹≤$_L$› *50*)
notation *R* (**infix** ‹≤$_R$› *50*)

end

Monotone Function Relator **locale** *transport-Mono-Fun-Rel* =
transport-Fun-Rel-syntax L1 R1 l1 r1 L2 R2 l2 r2 +
tfr : *transport-Fun-Rel L1 R1 l1 r1 L2 R2 l2 r2* +
tpdfr : *transport-Mono-Dep-Fun-Rel L1 R1 l1 r1* λ- -. *L2* λ- -. *R2*
λ- -. *l2* λ- -. *r2*
for *L1* :: *'a1* ⇒ *'a1* ⇒ *bool*
and *R1* :: *'a2* ⇒ *'a2* ⇒ *bool*
and *l1* :: *'a1* ⇒ *'a2*
and *r1* :: *'a2* ⇒ *'a1*
and *L2* :: *'b1* ⇒ *'b1* ⇒ *bool*
and *R2* :: *'b2* ⇒ *'b2* ⇒ *bool*
and *l2* :: *'b1* ⇒ *'b2*
and *r2* :: *'b2* ⇒ *'b1*
begin

notation *tpdfr.L* (‹*L*›)
notation *tpdfr.R* (‹*R*›)

abbreviation *l* ≡ *tpdfr.l*
abbreviation *r* ≡ *tpdfr.r*

notation *tpdfr.L* (**infix** ‹≤$_L$› *50*)
notation *tpdfr.R* (**infix** ‹≤$_R$› *50*)

notation *tpdfr.ge-left* (**infix** ‹≥$_L$› *50*)
notation *tpdfr.ge-right* (**infix** ‹≥$_R$› *50*)

notation *tpdfr.left-Galois* (**infix** $\langle_L \lessapprox \rangle$ *50*)
notation *tpdfr.ge-Galois-left* (**infix** $\langle \gtrapprox_L \rangle$ *50*)
notation *tpdfr.right-Galois* (**infix** $\langle_R \lessapprox \rangle$ *50*)
notation *tpdfr.ge-Galois-right* (**infix** $\langle \gtrapprox_R \rangle$ *50*)
notation *tpdfr.right-ge-Galois* (**infix** $\langle_R \gtrapprox \rangle$ *50*)
notation *tpdfr.Galois-right* (**infix** $\langle \lessapprox_R \rangle$ *50*)
notation *tpdfr.left-ge-Galois* (**infix** $\langle_L \gtrapprox \rangle$ *50*)
notation *tpdfr.Galois-left* (**infix** $\langle \lessapprox_L \rangle$ *50*)

notation *tpdfr.unit* ($\langle\eta\rangle$)
notation *tpdfr.counit* ($\langle\varepsilon\rangle$)

lemma *left-rel-eq-Mono-Fun-Rel*: $(\leq_L) = ((\leq_{L1}) \Rrightarrow\oplus (\leq_{L2}))$
⟨*proof*⟩

lemma *left-eq-fun-map*: $l = (r1 \rightsquigarrow l2)$
⟨*proof*⟩

interpretation *flip* : *transport-Mono-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2* ⟨*proof*⟩

lemma *right-rel-eq-Mono-Fun-Rel*: $(\leq_R) = ((\leq_{R1}) \Rrightarrow\oplus (\leq_{R2}))$
⟨*proof*⟩

lemma *right-eq-fun-map*: $r = (l1 \rightsquigarrow r2)$
⟨*proof*⟩

lemmas *transport-defs* = *tpdfr.transport-defs*

end

end

2.15.2 Monotonicity

theory *Transport-Functions-Monotone*
imports
Transport-Functions-Base
begin

Dependent Function Relator **context** *transport-Dep-Fun-Rel*
begin

interpretation *flip* : *transport-Dep-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2* ⟨*proof*⟩

lemma *mono-wrt-rel-leftI*:
assumes *mono-r1*: $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*
and *mono-l2*: $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$((\leq_{L2}\ (r1\ x1')\ (r1\ x2')) \Rightarrow (\leq_{R2}\ (\varepsilon_1\ x1')\ x2')) (l2\ x2'\ (r1\ x1'))$

and *R2-le1*: $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and *R2-l2-le1*: $\bigwedge x1'\ x2'\ y.\ x1' \leq_{R1} x2' \Longrightarrow$ *in-dom* $(\leq_{L2\ (r1\ x1')\ (r1\ x2')})\ y \Longrightarrow (\leq_{R2\ x1'\ x2'})\ (l2_{x2'\ (r1\ x1')}\ y) \leq (\leq_{R2\ x1'\ x2'})\ (l2_{x1'\ (r1\ x1')}\ y)$
and *ge-R2-l2-le2*: $\bigwedge x1'\ x2'\ y.\ x1' \leq_{R1} x2' \Longrightarrow$ *in-codom* $(\leq_{L2\ (r1\ x1')\ (r1\ x2')})\ y \Longrightarrow (\geq_{R2\ x1'\ x2'})\ (l2_{x2'\ (r1\ x1')}\ y) \leq (\geq_{R2\ x1'\ x2'})\ (l2_{x2'\ (r1\ x2')}\ y)$
shows $((\leq_L) \Rightarrow (\leq_R))\ l$
⟨*proof*⟩

lemma *mono-wrt-rel-left-in-dom-mono-left-assm*:
assumes (*in-dom* $(\leq_{L2\ (r1\ x1')\ (r1\ x2')}) \Rrightarrow (\leq_{R2\ x1'\ x2'}))\ (l2_{x1'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x1')})$
and *transitive* $(\leq_{R2\ x1'\ x2'})$
and $x1' \leq_{R1} x2'$
and *in-dom* $(\leq_{L2\ (r1\ x1')\ (r1\ x2')})\ y$
shows $(\leq_{R2\ x1'\ x2'})\ (l2_{x2'\ (r1\ x1')}\ y) \leq (\leq_{R2\ x1'\ x2'})\ (l2_{x1'\ (r1\ x1')}\ y)$
⟨*proof*⟩

lemma *mono-wrt-rel-left-in-codom-mono-left-assm*:
assumes (*in-codom* $(\leq_{L2\ (r1\ x1')\ (r1\ x2')}) \Rrightarrow (\leq_{R2\ x1'\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x2')})$
and *transitive* $(\leq_{R2\ x1'\ x2'})$
and $x1' \leq_{R1} x2'$
and *in-codom* $(\leq_{L2\ (r1\ x1')\ (r1\ x2')})\ y$
shows $(\geq_{R2\ x1'\ x2'})\ (l2_{x2'\ (r1\ x1')}\ y) \leq (\geq_{R2\ x1'\ x2'})\ (l2_{x2'\ (r1\ x2')}\ y)$
⟨*proof*⟩

lemma *mono-wrt-rel-left-if-transitiveI*:
assumes $((\leq_{R1}) \Rightarrow (\leq_{L1}))\ r1$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow ((\leq_{L2\ (r1\ x1')\ (r1\ x2')}) \Rightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}))\ (l2_{x2'\ (r1\ x1')})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\textit{in-dom}\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')}) \Rrightarrow (\leq_{R2\ x1'\ x2'}))\ (l2_{x1'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x1')})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\textit{in-codom}\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')}) \Rrightarrow (\leq_{R2\ x1'\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x2')})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$ *transitive* $(\leq_{R2\ x1'\ x2'})$
shows $((\leq_L) \Rightarrow (\leq_R))\ l$
⟨*proof*⟩

lemma *mono-wrt-rel-left2-if-mono-wrt-rel-left2-if-left-GaloisI*:
assumes $((\leq_{R1}) \Rightarrow (\leq_{L1}))\ r1$
and $\bigwedge x\ x'.\ x\ _{L1}\!\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \Rightarrow (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})$
shows $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow ((\leq_{L2\ (r1\ x1')\ (r1\ x2')}) \Rightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}))\ (l2_{x2'\ (r1\ x1')})$
⟨*proof*⟩

interpretation *flip-inv* :
transport-Dep-Fun-Rel $(\geq_{R1})$ $(\geq_{L1})$ *r1 l1 flip2 R2 flip2 L2 r2 l2*
rewrites *flip-inv.R* $\equiv (\geq_L)$ **and** *flip-inv.L* $\equiv (\geq_R)$
and *flip-inv.t1.counit* $\equiv \eta_1$
and $\bigwedge R\ x\ y.$ *(flip2 R x y)*$^{-1} \equiv R\ y\ x$
and $\bigwedge R\ x1\ x2.$ *in-dom (flip2 R x1 x2)* $\equiv$ *in-codom (R x2 x1)*
and $\bigwedge R\ x1\ x2.$ *in-codom (flip2 R x1 x2)* $\equiv$ *in-dom (R x2 x1)*
and $\bigwedge R\ S.\ (R^{-1} \Rightarrow S^{-1}) \equiv (R \Rightarrow S)$
and $\bigwedge x1\ x2\ x1'\ x2'.$ *(flip2 R2 x1' x2'* $\Rightarrow$ *flip2 L2 x1 x2)* $\equiv$
$((\leq_{R2\ x2'\ x1'}) \Rightarrow (\leq_{L2\ x2\ x1}))$
and $\bigwedge x1\ x2\ x3\ x4.$ *flip2 L2 x1 x2* $\leq$ *flip2 L2 x3 x4* $\equiv (\leq_{L2\ x2\ x1}) \leq (\leq_{L2\ x4\ x3})$
and $\bigwedge x1'\ x2'\ y1\ y2.$
flip-inv.dfro2.right-infix y1 x1' x2' $\leq$ *flip-inv.dfro2.right-infix y2 x1' x2'* $\equiv$
$(\geq_{L2\ x2'\ x1'})\ y1 \leq (\geq_{L2\ x2'\ x1'})\ y2$
and $\bigwedge(P :: 'f \Rightarrow bool)\ x1\ x2.\ (P \Rrightarrow$ *flip2 L2 x1 x2)* $\equiv (P \Rrightarrow (\geq_{L2\ x2\ x1}))$
and $\bigwedge(P :: 'f \Rightarrow bool)\ (R :: 'g \Rightarrow 'g \Rightarrow bool).\ (P \Rrightarrow R^{-1}) \equiv (P \Rrightarrow R)^{-1}$
and $\bigwedge x1\ x2.$ *transitive (flip2 L2 x1 x2)* $\equiv$ *transitive* $(\leq_{L2\ x2\ x1})$
⟨*proof*⟩

lemma *mono-wrt-rel-rightI*:
assumes $((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1*
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow ((\leq_{R2\ (l1\ x1)\ (l1\ x2)}) \Rightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}))\ (r2_{x1\ (l1\ x2)})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2\ y'.\ x1 \leq_{L1} x2 \Longrightarrow$ *in-codom* $(\leq_{R2\ (l1\ x1)\ (l1\ x2)})\ y' \Longrightarrow$
$(\geq_{L2\ x1\ x2})\ (r2_{x1\ (l1\ x2)}\ y') \leq (\geq_{L2\ x1\ x2})\ (r2_{x2\ (l1\ x2)}\ y')$
and $\bigwedge x1\ x2\ y'.\ x1 \leq_{L1} x2 \Longrightarrow$ *in-dom* $(\leq_{R2\ (l1\ x1)\ (l1\ x2)})\ y' \Longrightarrow$
$(\leq_{L2\ x1\ x2})\ (r2_{x1\ (l1\ x2)}\ y') \leq (\leq_{L2\ x1\ x2})\ (r2_{x1\ (l1\ x1)}\ y')$
shows $((\leq_R) \Rightarrow (\leq_L))\ r$
⟨*proof*⟩

lemma *mono-wrt-rel-right-if-transitiveI*:
assumes $((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1*
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow ((\leq_{R2\ (l1\ x1)\ (l1\ x2)}) \Rightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}))\ (r2_{x1\ (l1\ x2)})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *(in-codom* $(\leq_{R2\ (l1\ x1)\ (l1\ x2)}) \Rrightarrow (\leq_{L2\ x1\ x2}))$
$(r2_{x1\ (l1\ x2)})\ (r2_{x2\ (l1\ x2)})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *(in-dom* $(\leq_{R2\ (l1\ x1)\ (l1\ x2)}) \Rrightarrow (\leq_{L2\ x1\ x2}))$
$(r2_{x1\ (l1\ x1)})\ (r2_{x1\ (l1\ x2)})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
shows $((\leq_R) \Rightarrow (\leq_L))\ r$
⟨*proof*⟩

lemma *mono-wrt-rel-right2-if-mono-wrt-rel-right2-if-left-GaloisI*:
assumes *assms1*: $((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1* $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*
and *mono-r2*: $\bigwedge x\ x'.\ x\ {}_{L1}\!\lessapprox x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rightarrow (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
shows $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow ((\leq_{R2\ (l1\ x1)\ (l1\ x2)}) \Rightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}))$

$(r2_{x1}\ (l1\ x2))$
⟨*proof*⟩

end

Function Relator **context** *transport-Fun-Rel*
begin

lemma *mono-wrt-rel-leftI*:
assumes $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*
and $((\leq_{L2}) \Rightarrow (\leq_{R2}))$ *l2*
shows $((\leq_{L}) \Rightarrow (\leq_{R}))$ *l*
⟨*proof*⟩

end

Monotone Dependent Function Relator **context** *transport-Mono-Dep-Fun-Rel*
begin

lemmas *mono-wrt-rel-leftI* = *mono-wrt-rel-Refl-Rel-Refl-Rel-if-mono-wrt-rel*
[*of tdfr.L tdfr.R l*, *folded transport-defs*]

end

Monotone Function Relator **context** *transport-Mono-Fun-Rel*
begin

lemmas *mono-wrt-rel-leftI* = *tpdfr.mono-wrt-rel-leftI*[*OF tfr.mono-wrt-rel-leftI*]

end

end

2.15.3 Galois Property

theory *Transport-Functions-Galois-Property*
imports
Transport-Functions-Monotone
begin

Dependent Function Relator **context** *transport-Dep-Fun-Rel*
begin

context
begin

interpretation *flip* : *transport-Dep-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2* ⟨*proof*⟩

lemma *left-right-rel-if-left-rel-rightI*:
assumes *mono-r1*: $((\leq_{R1}) \Rightarrow (\leq_{L1}))\ r1$
and *half-galois-prop-left1*: $((\leq_{L1})\ {}_h\trianglelefteq\ (\leq_{R1}))\ l1\ r1$
and *refl-R1*: *reflexive-on* $(\textit{in-dom}\ (\leq_{R1}))\ (\leq_{R1})$
and *half-galois-prop-left2*: $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$
$((\leq_{L2}\ (r1\ x')\ (r1\ x'))\ {}_h\trianglelefteq\ (\leq_{R2}\ (\varepsilon_1\ x')\ x'))\ (l2\ x'\ (r1\ x'))\ (r2_{(r1\ x')}\ x')$
and *R2-le1*: $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow (\leq_{R2}\ (\varepsilon_1\ x')\ x') \leq (\leq_{R2\ x'\ x'})$
and *R2-le2*: $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$
and *ge-L2-r2-le2*: $\bigwedge x'\ y'.\ x' \leq_{R1} x' \Longrightarrow \textit{in-codom}\ (\leq_{R2\ (\varepsilon_1\ x')\ x'})\ y' \Longrightarrow$
$(\geq_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')}\ y') \leq (\geq_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ x'}\ y')$
and *trans-R2*: $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow \textit{transitive}\ (\leq_{R2\ x1'\ x2'})$
and $g \leq_R g$
and $f \leq_L r\ g$
shows $l\ f \leq_R g$
⟨*proof*⟩

lemma *left-right-rel-if-left-rel-right-ge-left2-assmI*:
assumes *mono-r1*: $((\leq_{R1}) \Rightarrow (\leq_{L1}))\ r1$
and $((\leq_{L1})\ {}_h\trianglelefteq\ (\leq_{R1}))\ l1\ r1$
and $((\textit{in-codom}\ (\leq_{R2\ (\varepsilon_1\ x')\ x'})) \Rrightarrow (\leq_{L2\ (r1\ x')\ (r1\ x')}))$
$(r2_{(r1\ x')\ (\varepsilon_1\ x')})\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow \textit{transitive}\ (\leq_{L2\ x1\ x2})$
and $x' \leq_{R1} x'$
and $\textit{in-codom}\ (\leq_{R2\ (\varepsilon_1\ x')\ x'})\ y'$
shows $(\geq_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')}\ y') \leq (\geq_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ x'}\ y')$
⟨*proof*⟩

interpretation *flip-inv* :
transport-Dep-Fun-Rel $(\geq_{R1})\ (\geq_{L1})\ r1\ l1\ flip2\ R2\ flip2\ L2\ r2\ l2$
rewrites *flip-inv.L* $\equiv (\geq_R)$ **and** *flip-inv.R* $\equiv (\geq_L)$
and *flip-inv.t1.counit* $\equiv \eta_1$
and $\bigwedge R\ x\ y.\ (flip2\ R\ x\ y)^{-1} \equiv R\ y\ x$
and $\bigwedge R.\ \textit{in-dom}\ R^{-1} \equiv \textit{in-codom}\ R$
and $\bigwedge R\ x1\ x2.\ \textit{in-codom}\ (flip2\ R\ x1\ x2) \equiv \textit{in-dom}\ (R\ x2\ x1)$
and $\bigwedge R\ S.\ (R^{-1} \Rightarrow S^{-1}) \equiv (R \Rightarrow S)$
and $\bigwedge R\ S\ x1\ x2\ x1'\ x2'.\ (flip2\ R\ x1\ x2\ {}_h\trianglelefteq\ flip2\ S\ x1'\ x2') \equiv (S\ x2'\ x1' \trianglelefteq_h R\ x2\ x1)^{-1}$
and $\bigwedge R\ S.\ (R^{-1}\ {}_h\trianglelefteq\ S^{-1}) \equiv (S \trianglelefteq_h R)^{-1}$
and $\bigwedge x1\ x2\ x3\ x4.\ flip2\ L2\ x1\ x2 \leq flip2\ L2\ x3\ x4 \equiv (\leq_{L2\ x2\ x1}) \leq (\leq_{L2\ x4\ x3})$
and $\bigwedge (R :: 'z \Rightarrow 'z \Rightarrow bool)\ (P :: 'z \Rightarrow bool).\ \textit{reflexive-on}\ P\ R^{-1} \equiv \textit{reflexive-on}\ P\ R$
and $\bigwedge R\ x1\ x2.\ \textit{transitive}\ (flip2\ R\ x1\ x2 :: 'z \Rightarrow 'z \Rightarrow bool) \equiv \textit{transitive}\ (R\ x2\ x1)$
and $\bigwedge x\ x.\ ((\textit{in-dom}\ (\leq_{L2\ x'\ (\eta_1\ x')})) \Rrightarrow flip2\ R2\ (l1\ x')\ (l1\ x'))$
$\equiv ((\textit{in-dom}\ (\leq_{L2\ x'\ (\eta_1\ x')})) \Rrightarrow (\leq_{R2\ (l1\ x')\ (l1\ x')}))^{-1}$

⟨*proof*⟩

lemma *left-rel-right-if-left-right-relI*:
assumes $((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1*
and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-codom* $(\leq_{L1})$) $(\leq_{L1})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow ((\leq_{L2\ x\ (\eta_1\ x)}) \trianglelefteq_h (\leq_{R2\ (l1\ x)\ (l1\ x)})) (l2_{(l1\ x)\ x}) (r2_{x\ (l1\ x)})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow (\leq_{L2\ x\ (\eta_1\ x)}) \leq (\leq_{L2\ x\ x})$
and $\bigwedge x\ y.\ x \leq_{L1} x \Longrightarrow$ *in-dom* $(\leq_{L2\ x\ (\eta_1\ x)})\ y \Longrightarrow$
$(\leq_{R2\ (l1\ x)\ (l1\ x)}) (l2_{(l1\ x)\ (\eta_1\ x)\ y}) \leq (\leq_{R2\ (l1\ x)\ (l1\ x)}) (l2_{(l1\ x)\ x\ y})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $f \leq_L f$
and $l\ f \leq_R g$
shows $f \leq_L r\ g$
⟨*proof*⟩

lemma *left-rel-right-if-left-right-rel-le-right2-assmI*:
assumes $((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1*
and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))^{-1}$ *r1 l1*
and $((\textit{in-dom}\ (\leq_{L2\ x\ (\eta_1\ x)})) \Rrightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)})) (l2_{(l1\ x)\ x}) (l2_{(l1\ x)\ (\eta_1\ x)})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$ *transitive* $(\leq_{R2\ x1'\ x2'})$
and $x \leq_{L1} x$
and *in-dom* $(\leq_{L2\ x\ (\eta_1\ x)})\ y$
shows $(\leq_{R2\ (l1\ x)\ (l1\ x)}) (l2_{(l1\ x)\ (\eta_1\ x)\ y}) \leq (\leq_{R2\ (l1\ x)\ (l1\ x)}) (l2_{(l1\ x)\ x\ y})$
⟨*proof*⟩

end

lemma *left-rel-right-iff-left-right-relI*:
assumes $((\leq_{L1}) \dashv (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-codom* $(\leq_{L1})$) $(\leq_{L1})$
and *reflexive-on* (*in-dom* $(\leq_{R1})$) $(\leq_{R1})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$
$((\leq_{L2\ (r1\ x')\ (r1\ x')})\ {}_h\trianglelefteq\ (\leq_{R2\ (\varepsilon_1\ x')\ x'})) (l2_{\ x'\ (r1\ x')}) (r2_{(r1\ x')\ x'})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow ((\leq_{L2\ x\ (\eta_1\ x)}) \trianglelefteq_h (\leq_{R2\ (l1\ x)\ (l1\ x)})) (l2_{(l1\ x)\ x}) (r2_{x\ (l1\ x)})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow (\leq_{L2\ x\ (\eta_1\ x)}) \leq (\leq_{L2\ x\ x})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x')\ x'}) \leq (\leq_{R2\ x'\ x'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x\ y.\ x \leq_{L1} x \Longrightarrow$ *in-dom* $(\leq_{L2\ x\ (\eta_1\ x)})\ y \Longrightarrow$
$(\leq_{R2\ (l1\ x)\ (l1\ x)}) (l2_{(l1\ x)\ (\eta_1\ x)\ y}) \leq (\leq_{R2\ (l1\ x)\ (l1\ x)}) (l2_{(l1\ x)\ x\ y})$
and $\bigwedge x'\ y'.\ x' \leq_{R1} x' \Longrightarrow$ *in-codom* $(\leq_{R2\ (\varepsilon_1\ x')\ x'})\ y' \Longrightarrow$
$(\geq_{L2\ (r1\ x')\ (r1\ x')}) (r2_{(r1\ x')\ (\varepsilon_1\ x')\ y'}) \leq (\geq_{L2\ (r1\ x')\ (r1\ x')}) (r2_{(r1\ x')\ x'\ y'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$ *transitive* $(\leq_{R2\ x1'\ x2'})$

and $f \leq_L f$
and $g \leq_R g$
shows $f \leq_L r\ g \longleftrightarrow l\ f \leq_R g$
⟨*proof*⟩

lemma *half-galois-prop-left2-if-half-galois-prop-left2-if-left-GaloisI*:
assumes $((\leq_{R1}) \Rightarrow (\leq_{L1}))\ r1$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')})\ {}_h\trianglelefteq\ (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $x' \leq_{R1} x'$
shows $((\leq_{L2\ (r1\ x')\ (r1\ x')})\ {}_h\trianglelefteq\ (\leq_{R2\ (\varepsilon_1\ x')\ x'}))\ (l2_{\ x'\ (r1\ x')})\ (r2_{(r1\ x')\ x'})$
⟨*proof*⟩

lemma *half-galois-prop-right2-if-half-galois-prop-right2-if-left-GaloisI*:
assumes $((\leq_{L1}) \Rightarrow (\leq_{R1}))\ l1$
and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))\ l1\ r1$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')})\ \trianglelefteq_h\ (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $x \leq_{L1} x$
shows $((\leq_{L2\ x\ (\eta_1\ x)})\ \trianglelefteq_h\ (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (r2_{x\ (l1\ x)})$
⟨*proof*⟩

lemma *left-rel-right-iff-left-right-relI'*:
assumes $((\leq_{L1}) \dashv (\leq_{R1}))\ l1\ r1$
and *reflexive-on* $(in\text{-}codom\ (\leq_{L1}))\ (\leq_{L1})$
and *reflexive-on* $(in\text{-}dom\ (\leq_{R1}))\ (\leq_{R1})$
and *galois-prop2*: $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow$
$((\leq_{L2\ x\ (r1\ x')})\ \trianglelefteq\ (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow (\leq_{L2\ x\ (\eta_1\ x)}) \leq (\leq_{L2\ x\ x})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x')\ x'}) \leq (\leq_{R2\ x'\ x'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$
$((in\text{-}dom\ (\leq_{L2\ x\ (\eta_1\ x)}))) \Rrightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (l2_{(l1\ x)\ (\eta_1\ x)})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$
$((in\text{-}codom\ (\leq_{R2\ (\varepsilon_1\ x')\ x'})) \Rrightarrow (\leq_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow transitive\ (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow transitive\ (\leq_{R2\ x1'\ x2'})$
and $f \leq_L f$
and $g \leq_R g$
shows $f \leq_L r\ g \longleftrightarrow l\ f \leq_R g$
⟨*proof*⟩

lemma *left-rel-right-iff-left-right-rel-if-galois-prop-le-assms-leftI*:
assumes *galois-conn1*: $((\leq_{L1}) \dashv (\leq_{R1}))\ l1\ r1$
and *refl-L1*: *reflexive-on* $(in\text{-}field\ (\leq_{L1}))\ (\leq_{L1})$
and *antimono-L2*:
$((x1\ x2 :: (\leq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid (x2 \leq_{L1} x3 \wedge x4 \leq_{L1} \eta_1\ x3)) \Rrightarrow (\geq))$
L2
shows $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$

and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
⟨*proof*⟩

lemma *left-rel-right-iff-left-right-rel-if-galois-prop-le-assms-rightI*:
assumes *galois-conn1*: $((\leq_{L1}) \dashv (\leq_{R1}))\ l1\ r1$
and *refl-R1*: *reflexive-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and *mono-R2*:
$((x1'\ x2' :: (\leq_{R1}) \mid \varepsilon_1\ x2' \leq_{R1} x1') \Rightarrow (x3'\ x4' :: (\leq_{R1}) \mid x2' \leq_{R1} x3') \Rrightarrow (\leq))\ R2$
shows $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$
⟨*proof*⟩

corollary *left-rel-right-iff-left-right-rel-if-monoI*:
assumes $((\leq_{L1}) \dashv (\leq_{R1}))\ l1\ r1$
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and *reflexive-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \trianglelefteq (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid (x2 \leq_{L1} x3 \wedge x4 \leq_{L1} \eta_1\ x3)) \Rrightarrow (\geq))\ L2$
and $((x1'\ x2' :: (\leq_{R1}) \mid \varepsilon_1\ x2' \leq_{R1} x1') \Rightarrow (x3'\ x4' :: (\leq_{R1}) \mid x2' \leq_{R1} x3') \Rrightarrow (\leq))\ R2$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$
$((in\text{-}dom\ (\leq_{L2\ x\ (\eta_1\ x)})) \Rrightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (l2_{(l1\ x)\ (\eta_1\ x)})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$
$((in\text{-}codom\ (\leq_{R2\ (\varepsilon_1\ x')\ x'})) \Rrightarrow (\leq_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow transitive\ (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow transitive\ (\leq_{R2\ x1'\ x2'})$
and $f \leq_L f$
and $g \leq_R g$
shows $f \leq_L r\ g \longleftrightarrow l\ f \leq_R g$
⟨*proof*⟩

end

Function Relator **context** *transport-Fun-Rel*
begin

corollary *left-right-rel-if-left-rel-rightI*:
assumes $((\leq_{R1}) \Rightarrow (\leq_{L1}))\ r1$
and $((\leq_{L1})\ {}_h\trianglelefteq (\leq_{R1}))\ l1\ r1$
and *reflexive-on* (*in-dom* $(\leq_{R1})$) $(\leq_{R1})$
and $((\leq_{L2})\ {}_h\trianglelefteq (\leq_{R2}))\ l2\ r2$
and *transitive* $(\leq_{R2})$
and $g \leq_R g$
and $f \leq_L r\ g$
shows $l\ f \leq_R g$
⟨*proof*⟩

corollary *left-rel-right-if-left-right-relI*:
assumes $((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1*
and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-codom* $(\leq_{L1})$) $(\leq_{L1})$
and $((\leq_{L2}) \trianglelefteq_h (\leq_{R2}))$ *l2 r2*
and *transitive* $(\leq_{L2})$
and $f \leq_L f$
and $l\ f \leq_R g$
shows $f \leq_L r\ g$
⟨*proof*⟩

corollary *left-rel-right-iff-left-right-relI*:
assumes $((\leq_{L1}) \dashv (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-codom* $(\leq_{L1})$) $(\leq_{L1})$
and *reflexive-on* (*in-dom* $(\leq_{R1})$) $(\leq_{R1})$
and $((\leq_{L2}) \trianglelefteq (\leq_{R2}))$ *l2 r2*
and *transitive* $(\leq_{L2})$
and *transitive* $(\leq_{R2})$
and $f \leq_L f$
and $g \leq_R g$
shows $f \leq_L r\ g \longleftrightarrow l\ f \leq_R g$
⟨*proof*⟩

end

Monotone Dependent Function Relator **context** *transport-Mono-Dep-Fun-Rel* **begin**

lemma *half-galois-prop-left-left-rightI*:
assumes $(tdfr.L \Rightarrow tdfr.R)\ l$
and $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*
and $((\leq_{L1})\ {}_h\trianglelefteq\ (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-dom* $(\leq_{R1})$) $(\leq_{R1})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$
$((\leq_{L2}\ (r1\ x')\ (r1\ x'))\ {}_h\trianglelefteq\ (\leq_{R2}\ (\varepsilon_1\ x')\ x'))\ (l2\ x'\ (r1\ x'))\ (r2_{(r1\ x')}\ x')$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow (\leq_{R2}\ (\varepsilon_1\ x')\ x') \leq (\leq_{R2}\ x'\ x')$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2}\ x1'\ x1') \leq (\leq_{R2}\ x1'\ x2')$
and $\bigwedge x'\ y'.\ x' \leq_{R1} x' \Longrightarrow$ *in-codom* $(\leq_{R2}\ (\varepsilon_1\ x')\ x')\ y' \Longrightarrow$
$(\geq_{L2}\ (r1\ x')\ (r1\ x'))\ (r2_{(r1\ x')}\ (\varepsilon_1\ x')\ y') \leq (\geq_{L2}\ (r1\ x')\ (r1\ x'))\ (r2_{(r1\ x')}\ x'\ y')$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$ *transitive* $(\leq_{R2\ x1'\ x2'})$
shows $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$
⟨*proof*⟩

lemma *half-galois-prop-right-left-rightI*:
assumes $(tdfr.R \Rightarrow tdfr.L)\ r$
and $((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1*

and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-codom* $(\leq_{L1})$) $(\leq_{L1})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow ((\leq_{L2\ x\ (\eta_1\ x)}) \trianglelefteq_h (\leq_{R2\ (l1\ x)\ (l1\ x)})) (l2_{(l1\ x)\ x}) (r2_{x\ (l1\ x)})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow (\leq_{L2\ x\ (\eta_1\ x)}) \leq (\leq_{L2\ x\ x})$
and $\bigwedge x\ y.\ x \leq_{L1} x \Longrightarrow$ *in-dom* $(\leq_{L2\ x\ (\eta_1\ x)})\ y \Longrightarrow$
$(\leq_{R2\ (l1\ x)\ (l1\ x)}) (l2_{(l1\ x)\ (\eta_1\ x)}\ y) \leq (\leq_{R2\ (l1\ x)\ (l1\ x)}) (l2_{(l1\ x)\ x}\ y)$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
shows $((\leq_L) \trianglelefteq_h (\leq_R))\ l\ r$
⟨*proof*⟩

corollary *galois-prop-left-rightI*:
assumes $(tdfr.L \Rightarrow tdfr.R)\ l$ **and** $(tdfr.R \Rightarrow tdfr.L)\ r$
and $((\leq_{L1}) \dashv (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-codom* $(\leq_{L1})$) $(\leq_{L1})$
and *reflexive-on* (*in-dom* $(\leq_{R1})$) $(\leq_{R1})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$
$((\leq_{L2\ (r1\ x')\ (r1\ x')}) \ {}_h\trianglelefteq (\leq_{R2\ (\varepsilon_1\ x')\ x'})) (l2_{\ x'\ (r1\ x')}) (r2_{(r1\ x')\ x'})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow ((\leq_{L2\ x\ (\eta_1\ x)}) \trianglelefteq_h (\leq_{R2\ (l1\ x)\ (l1\ x)})) (l2_{(l1\ x)\ x}) (r2_{x\ (l1\ x)})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow (\leq_{L2\ x\ (\eta_1\ x)}) \leq (\leq_{L2\ x\ x})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x')\ x'}) \leq (\leq_{R2\ x'\ x'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x\ y.\ x \leq_{L1} x \Longrightarrow$ *in-dom* $(\leq_{L2\ x\ (\eta_1\ x)})\ y \Longrightarrow$
$(\leq_{R2\ (l1\ x)\ (l1\ x)}) (l2_{(l1\ x)\ (\eta_1\ x)}\ y) \leq (\leq_{R2\ (l1\ x)\ (l1\ x)}) (l2_{(l1\ x)\ x}\ y)$
and $\bigwedge x'\ y'.\ x' \leq_{R1} x' \Longrightarrow$ *in-codom* $(\leq_{R2\ (\varepsilon_1\ x')\ x'})\ y' \Longrightarrow$
$(\geq_{L2\ (r1\ x')\ (r1\ x')}) (r2_{(r1\ x')\ (\varepsilon_1\ x')}\ y') \leq (\geq_{L2\ (r1\ x')\ (r1\ x')}) (r2_{(r1\ x')\ x'}\ y')$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$ *transitive* $(\leq_{R2\ x1'\ x2'})$
shows $((\leq_L) \trianglelefteq (\leq_R))\ l\ r$
⟨*proof*⟩

corollary *galois-prop-left-rightI'*:
assumes $(tdfr.L \Rightarrow tdfr.R)\ l$ **and** $(tdfr.R \Rightarrow tdfr.L)\ r$
and $((\leq_{L1}) \dashv (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-codom* $(\leq_{L1})$) $(\leq_{L1})$
and *reflexive-on* (*in-dom* $(\leq_{R1})$) $(\leq_{R1})$
and *galois-prop2*: $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow$
$((\leq_{L2\ x\ (r1\ x')}) \trianglelefteq (\leq_{R2\ (l1\ x)\ x'})) (l2_{x'\ x}) (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow (\leq_{L2\ x\ (\eta_1\ x)}) \leq (\leq_{L2\ x\ x})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x')\ x'}) \leq (\leq_{R2\ x'\ x'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$
$((in\text{-}dom\ (\leq_{L2\ x\ (\eta_1\ x)})) \Rrightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)})) (l2_{(l1\ x)\ x}) (l2_{(l1\ x)\ (\eta_1\ x)})$

and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$
$((\textit{in-codom}\ (\leq_{R2\ (\varepsilon_1\ x')\ x'})) \Rrightarrow (\leq_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow \textit{transitive}\ (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow \textit{transitive}\ (\leq_{R2\ x1'\ x2'})$
shows $((\leq_L) \trianglelefteq (\leq_R))\ l\ r$
⟨*proof*⟩

corollary *galois-prop-left-right-if-mono-if-galois-propI*:
assumes $(\textit{tdfr.L} \Rightarrow \textit{tdfr.R})\ l$ **and** $(\textit{tdfr.R} \Rightarrow \textit{tdfr.L})\ r$
and $((\leq_{L1}) \dashv (\leq_{R1}))\ l1\ r1$
and $\textit{reflexive-on}\ (\textit{in-field}\ (\leq_{L1}))\ (\leq_{L1})$
and $\textit{reflexive-on}\ (\textit{in-field}\ (\leq_{R1}))\ (\leq_{R1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \trianglelefteq (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid (x2 \leq_{L1} x3 \wedge x4 \leq_{L1} \eta_1\ x3)) \Rrightarrow (\geq))\ L2$
and $((x1'\ x2' :: (\leq_{R1}) \mid \varepsilon_1\ x2' \leq_{R1} x1') \Rightarrow (x3'\ x4' :: (\leq_{R1}) \mid x2' \leq_{R1} x3') \Rrightarrow (\leq))\ R2$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$
$((\textit{in-dom}\ (\leq_{L2\ x\ (\eta_1\ x)})) \Rrightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (l2_{(l1\ x)\ (\eta_1\ x)})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$
$((\textit{in-codom}\ (\leq_{R2\ (\varepsilon_1\ x')\ x'})) \Rrightarrow (\leq_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow \textit{transitive}\ (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow \textit{transitive}\ (\leq_{R2\ x1'\ x2'})$
shows $((\leq_L) \trianglelefteq (\leq_R))\ l\ r$
⟨*proof*⟩

Note that we could further rewrite $[\![(\textit{tdfr.L} \Rightarrow \textit{tdfr.R})\ l;\ (\textit{tdfr.R} \Rightarrow \textit{tdfr.L})\ r;\ \textit{t1.galois-connection};\ \textit{reflexive-on}\ (\textit{in-field}\ (\leq_{L1}))\ (\leq_{L1});\ \textit{reflexive-on}\ (\textit{in-field}\ (\leq_{R1}))\ (\leq_{R1});\ \bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow \textit{t2.galois-prop}\ x\ x'\ l2_{x'\ x}\ r2_{x\ x'};\ ((x1\ x2 :: (\leq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1})) \Rrightarrow (x2 \leq_{L1} x3 \wedge x4 \leq_{L1} \eta_1\ x3) \longrightarrow (\lambda x\ y.\ y \leq x))\ L2;\ ((x1'\ x2' :: (\leq_{R1})) \Rightarrow \varepsilon_1\ x2' \leq_{R1} x1' \longrightarrow ((x3'\ x4' :: (\leq_{R1})) \Rrightarrow x2' \leq_{R1} x3' \longrightarrow (\leq)))\ R2;\ \bigwedge x.\ x \leq_{L1} x \Longrightarrow (\textit{in-dom}\ (\leq_{L2\ x\ \eta_1\ x}) \Rrightarrow \leq_{R2\ l1\ x\ l1\ x})\ l2_{l1\ x\ x}\ l2_{l1\ x\ \eta_1\ x};\ \bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow (\textit{in-codom}\ (\leq_{R2\ \varepsilon_1\ x'\ x'}) \Rrightarrow \leq_{L2\ r1\ x'\ r1\ x'})\ r2_{r1\ x'\ \varepsilon_1\ x'}\ r2_{r1\ x'\ x'};\ \bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow \textit{transitive}\ (\leq_{L2\ x1\ x2});\ \bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow \textit{transitive}\ (\leq_{R2\ x1'\ x2'})]\!] \Longrightarrow \textit{galois-prop}\ l\ r$, as we will do later for Galois connections, by applying $[\![((\leq_{R1}) \Rightarrow (\leq_{L1}))\ r1;\ \bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow ((\leq_{L2\ r1\ x1'\ r1\ x2'}) \Rrightarrow \leq_{R2\ \varepsilon_1\ x1'\ x2'})\ l2_{x2'\ r1\ x1'};\ \bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ \varepsilon_1\ x1'\ x2'}) \leq (\leq_{R2\ x1'\ x2'});\ \bigwedge x1'\ x2'\ y.\ [\![x1' \leq_{R1} x2';\ \textit{in-dom}\ (\leq_{L2\ r1\ x1'\ r1\ x2'})\ y]\!] \Longrightarrow \textit{dfro2.right-infix}\ (l2_{x2'\ r1\ x1'}\ y)\ x1'\ x2' \leq \textit{dfro2.right-infix}\ (l2_{x1'\ r1\ x1'}\ y)\ x1'\ x2';\ \bigwedge x1'\ x2'\ y.\ [\![x1' \leq_{R1} x2';\ \textit{in-codom}\ (\leq_{L2\ r1\ x1'\ r1\ x2'})\ y]\!] \Longrightarrow (\leq_{R2\ x1'\ x2'})^{-1}\ (l2_{x2'\ r1\ x1'}\ y) \leq (\leq_{R2\ x1'\ x2'})^{-1}\ (l2_{x2'\ r1\ x2'}\ y)]\!] \Longrightarrow (\textit{tdfr.L} \Rightarrow \textit{tdfr.R})\ l$ and $[\![((\leq_{L1}) \Rightarrow (\leq_{R1}))\ l1;\ \bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow ((\leq_{R2\ l1\ x1\ l1\ x2}) \Rightarrow \leq_{L2\ x1\ \eta_1\ x2})\ r2_{x1\ l1\ x2};\ \bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ \eta_1\ x2}) \leq (\leq_{L2\ x1\ x2});\ \bigwedge x1\ x2\ y'.\ [\![x1 \leq_{L1} x2;\ \textit{in-codom}\ (\leq_{R2\ l1\ x1\ l1\ x2})\ y']\!] \Longrightarrow (\leq_{L2\ x1\ x2})^{-1}\ (r2_{x1\ l1\ x2}\ y') \leq (\leq_{L2\ x1\ x2})^{-1}\ (r2_{x2\ l1\ x2}$

y'); $\bigwedge x1\ x2\ y'.\ [\![x1 \leq_{L1} x2;\ in\text{-}dom\ (\leq_{R2\ l1\ x1\ l1\ x2})\ y']\!] \Longrightarrow dfro1.right\text{-}infix\ (r2_{x1\ l1\ x2}\ y')\ x1\ x2 \leq dfro1.right\text{-}infix\ (r2_{x1\ l1\ x1}\ y')\ x1\ x2]\!] \Longrightarrow (tdfr.R \Rightarrow tdfr.L)\ r$ to the first premises. However, this is not really helpful here. Moreover, the resulting theorem will not result in a useful lemma for the flipped instance of *transport-Dep-Fun-Rel* since $[\![((\leq_{R1}) \Rightarrow (\leq_{L1}))\ r1;\ \bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow ((\leq_{L2\ r1\ x1'\ r1\ x2'}) \Rightarrow \leq_{R2\ \varepsilon_1\ x1'\ x2'})\ l2_{x2'\ r1\ x1'};\ \bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ \varepsilon_1\ x1'\ x2'}) \leq (\leq_{R2\ x1'\ x2'});\ \bigwedge x1'\ x2'\ y.\ [\![x1' \leq_{R1} x2';\ in\text{-}dom\ (\leq_{L2\ r1\ x1'\ r1\ x2'})\ y]\!] \Longrightarrow dfro2.right\text{-}infix\ (l2_{x2'\ r1\ x1'}\ y)\ x1'\ x2' \leq dfro2.right\text{-}infix\ (l2_{x1'\ r1\ x1'}\ y)\ x1'\ x2';\ \bigwedge x1'\ x2'\ y.\ [\![x1' \leq_{R1} x2';\ in\text{-}codom\ (\leq_{L2\ r1\ x1'\ r1\ x2'})\ y]\!] \Longrightarrow (\leq_{R2\ x1'\ x2'})^{-1}\ (l2_{x2'\ r1\ x1'}\ y) \leq (\leq_{R2\ x1'\ x2'})^{-1}\ (l2_{x2'\ r1\ x2'}\ y)]\!] \Longrightarrow (tdfr.L \Rightarrow tdfr.R)\ l$ and $[\![((\leq_{L1}) \Rightarrow (\leq_{R1}))\ l1;\ \bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow ((\leq_{R2\ l1\ x1\ l1\ x2}) \Rightarrow \leq_{L2\ x1\ \eta_1\ x2})\ r2_{x1\ l1\ x2};\ \bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ \eta_1\ x2}) \leq (\leq_{L2\ x1\ x2});\ \bigwedge x1\ x2\ y'.\ [\![x1 \leq_{L1} x2;\ in\text{-}codom\ (\leq_{R2\ l1\ x1\ l1\ x2})\ y']\!] \Longrightarrow (\leq_{L2\ x1\ x2})^{-1}\ (r2_{x1\ l1\ x2}\ y') \leq (\leq_{L2\ x1\ x2})^{-1}\ (r2_{x2\ l1\ x2}\ y');\ \bigwedge x1\ x2\ y'.\ [\![x1 \leq_{L1} x2;\ in\text{-}dom\ (\leq_{R2\ l1\ x1\ l1\ x2})\ y']\!] \Longrightarrow dfro1.right\text{-}infix\ (r2_{x1\ l1\ x2}\ y')\ x1\ x2 \leq dfro1.right\text{-}infix\ (r2_{x1\ l1\ x1}\ y')\ x1\ x2]\!] \Longrightarrow (tdfr.R \Rightarrow tdfr.L)\ r$ are not flipped dual but only flipped-inversed dual.

end

Monotone Function Relator **context** *transport-Mono-Fun-Rel*
begin

lemma *half-galois-prop-left-left-rightI*:
assumes $((\leq_{R1}) \Rightarrow (\leq_{L1}))\ r1$
and $((\leq_{L1})\ {}_h\trianglelefteq\ (\leq_{R1}))\ l1\ r1$
and *reflexive-on* $(in\text{-}dom\ (\leq_{R1}))\ (\leq_{R1})$
and $((\leq_{L2}) \Rightarrow (\leq_{R2}))\ l2$
and $((\leq_{L2})\ {}_h\trianglelefteq\ (\leq_{R2}))\ l2\ r2$
and *transitive* $(\leq_{R2})$
shows $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$
⟨*proof*⟩

interpretation *flip* : *transport-Mono-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2* ⟨*proof*⟩

lemma *half-galois-prop-right-left-rightI*:
assumes $((\leq_{L1}) \Rightarrow (\leq_{R1}))\ l1$
and $((\leq_{L1}) \trianglelefteq_h\ (\leq_{R1}))\ l1\ r1$
and *reflexive-on* $(in\text{-}codom\ (\leq_{L1}))\ (\leq_{L1})$
and $((\leq_{R2}) \Rightarrow (\leq_{L2}))\ r2$
and $((\leq_{L2}) \trianglelefteq_h\ (\leq_{R2}))\ l2\ r2$
and *transitive* $(\leq_{L2})$
shows $((\leq_L) \trianglelefteq_h\ (\leq_R))\ l\ r$
⟨*proof*⟩

corollary *galois-prop-left-rightI*:
assumes $((\leq_{L1}) \dashv (\leq_{R1}))\ l1\ r1$
and *reflexive-on* $(in\text{-}codom\ (\leq_{L1}))\ (\leq_{L1})$

and *reflexive-on* (*in-dom* ($\leq_{R1}$)) ($\leq_{R1}$)
and (($\leq_{L2}$) $\dashv$ ($\leq_{R2}$)) *l2 r2*
and *transitive* ($\leq_{L2}$)
and *transitive* ($\leq_{R2}$)
shows (($\leq_L$) $\trianglelefteq$ ($\leq_R$)) *l r*
⟨*proof*⟩

end

end

2.15.4 Galois Connection

theory *Transport-Functions-Galois-Connection*
imports
Transport-Functions-Galois-Property
Transport-Functions-Monotone
begin

Dependent Function Relator **context** *transport-Dep-Fun-Rel*
begin

Lemmas for Monotone Function Relator **lemma** *galois-connection-left-right-if-galois-connection-m*
assumes *galois-conn1*: (($\leq_{L1}$) $\dashv$ ($\leq_{R1}$)) *l1 r1*
and *refl-R1*: *reflexive-on* (*in-codom* ($\leq_{R1}$)) ($\leq_{R1}$)
and *R2-le1*: $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and *mono-l2-2*: $((x' :: \textit{in-codom}\ (\leq_{R1})) \Rightarrow (x1\ x2 :: (\leq_{L1}) \mid x2\ _{L1}\lessapprox x') \Rightarrow$
(*in-field* ($\leq_{L2\ x1\ (r1\ x')}$)) $\Rrightarrow$ ($\leq_{R2\ (l1\ x1)\ x'}$)) *l2*
shows $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
((*in-codom* ($\leq_{L2\ (r1\ x1')\ (r1\ x2')}$)) $\Rrightarrow$ ($\leq_{R2\ x1'\ x2'}$)) ($l2_{x2'\ (r1\ x1')}$) ($l2_{x2'\ (r1\ x2')}$)
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$
((*in-dom* ($\leq_{L2\ x\ (\eta_1\ x)}$)) $\Rrightarrow$ ($\leq_{R2\ (l1\ x)\ (l1\ x)}$)) ($l2_{(l1\ x)\ x}$) ($l2_{(l1\ x)\ (\eta_1\ x)}$)
⟨*proof*⟩

lemma *galois-connection-left-right-if-galois-connection-mono-assms-leftI*:
assumes *galois-conn1*: (($\leq_{L1}$) $\dashv$ ($\leq_{R1}$)) *l1 r1*
and *refl-R1*: *reflexive-on* (*in-field* ($\leq_{R1}$)) ($\leq_{R1}$)
and *R2-le1*: $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and *mono-l2*: $((x1'\ x2' :: (\leq_{R1})) \Rightarrow (x1\ x2 :: (\leq_{L1}) \mid x2\ _{L1}\lessapprox x1') \Rrightarrow$
in-field ($\leq_{L2\ x1\ (r1\ x2')}$) $\Rrightarrow$ ($\leq_{R2\ (l1\ x1)\ x2'}$)) *l2*
shows $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
((*in-dom* ($\leq_{L2\ (r1\ x1')\ (r1\ x2')}$)) $\Rrightarrow$ ($\leq_{R2\ x1'\ x2'}$)) ($l2_{x1'\ (r1\ x1')}$) ($l2_{x2'\ (r1\ x1')}$)
and (($x' :: \textit{in-codom}\ (\leq_{R1})$) $\Rightarrow$ ($x1\ x2 :: (\leq_{L1}) \mid x2\ _{L1}\lessapprox x'$) $\Rightarrow$
(*in-field* ($\leq_{L2\ x1\ (r1\ x')}$)) $\Rrightarrow$ ($\leq_{R2\ (l1\ x1)\ x'}$)) *l2*
⟨*proof*⟩

In theory, the following lemmas can be obtained by taking the flipped,

inverse interpretation of the locale; however, rewriting the assumptions is more involved than simply copying and adapting above proofs.

lemma *galois-connection-left-right-if-galois-connection-mono-2-assms-rightI*:
assumes *galois-conn1*: $((\leq_{L1}) \dashv (\leq_{R1}))$ *l1 r1*
and *refl-L1*: *reflexive-on* (*in-dom* $(\leq_{L1})$) $(\leq_{L1})$
and *L2-le2*: $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and *mono-r2-2*: $((x : \textit{in-dom}\ (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x\ {}_{L1}\lessapprox x1') \Rightarrow$
$(\textit{in-field}\ (\leq_{R2\ (l1\ x)\ x2'})) \Rrightarrow (\leq_{L2\ x\ (r1\ x2')}))\ r2$
shows $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$
$((\textit{in-dom}\ (\leq_{R2\ (l1\ x1)\ (l1\ x2)})) \Rrightarrow (\leq_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x1)})\ (r2_{x1\ (l1\ x2)})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$
$((\textit{in-codom}\ (\leq_{R2\ (\varepsilon_1\ x')\ x'})) \Rrightarrow (\leq_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})\ (r2_{(r1\ x')\ x'})$
⟨*proof*⟩

lemma *galois-connection-left-right-if-galois-connection-mono-assms-rightI*:
assumes *galois-conn1*: $((\leq_{L1}) \dashv (\leq_{R1}))$ *l1 r1*
and *refl-L1*: *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and *L2-le2*: $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and *mono-r2*: $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox x1') \Rrightarrow$
$\textit{in-field}\ (\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
shows $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$
$((\textit{in-codom}\ (\leq_{R2\ (l1\ x1)\ (l1\ x2)})) \Rrightarrow (\leq_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x2)})\ (r2_{x2\ (l1\ x2)})$
and $((x : \textit{in-dom}\ (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x\ {}_{L1}\lessapprox x1') \Rightarrow$
$(\textit{in-field}\ (\leq_{R2\ (l1\ x)\ x2'})) \Rrightarrow (\leq_{L2\ x\ (r1\ x2')}))\ r2$
⟨*proof*⟩

end

Monotone Dependent Function Relator **context** *transport-Mono-Dep-Fun-Rel* **begin**

interpretation *flip* : *transport-Mono-Dep-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2* ⟨*proof*⟩

lemma *galois-connection-left-rightI*:
assumes $(\textit{tdfr.L} \Rightarrow \textit{tdfr.R})\ l$ **and** $(\textit{tdfr.R} \Rightarrow \textit{tdfr.L})\ r$
and $((\leq_{L1}) \dashv (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-codom* $(\leq_{L1})$) $(\leq_{L1})$
and *reflexive-on* (*in-dom* $(\leq_{R1})$) $(\leq_{R1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \trianglelefteq (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow (\leq_{L2\ x\ (\eta_1\ x)}) \leq (\leq_{L2\ x\ x})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x')\ x'}) \leq (\leq_{R2\ x'\ x'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$
$((\textit{in-dom}\ (\leq_{L2\ x\ (\eta_1\ x)})) \Rrightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (l2_{(l1\ x)\ (\eta_1\ x)})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$

$((\mathit{in\text{-}codom}\ (\leq_{R2}\ (\varepsilon_1\ x')\ x')) \Rrightarrow (\leq_{L2}\ (r1\ x')\ (r1\ x'))) \ (r2\ (r1\ x')\ (\varepsilon_1\ x'))\ (r2\ (r1\ x')\ x')$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow \mathit{transitive}\ (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow \mathit{transitive}\ (\leq_{R2\ x1'\ x2'})$
shows $((\leq_L) \dashv (\leq_R))\ l\ r$
⟨*proof*⟩

lemma *galois-connection-left-rightI'*:
assumes $((\leq_{L1}) \dashv (\leq_{R1}))\ l1\ r1$
and $\mathit{reflexive\text{-}on}\ (\mathit{in\text{-}codom}\ (\leq_{L1}))\ (\leq_{L1})$
and $\mathit{reflexive\text{-}on}\ (\mathit{in\text{-}dom}\ (\leq_{R1}))\ (\leq_{R1})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$((\leq_{L2}\ (r1\ x1')\ (r1\ x2')) \Rightarrow (\leq_{R2}\ (\varepsilon_1\ x1')\ x2'))\ (l2_{x2'}\ (r1\ x1'))$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow ((\leq_{R2}\ (l1\ x1)\ (l1\ x2)) \Rightarrow (\leq_{L2\ x1}\ (\eta_1\ x2)))\ (r2_{x1}\ (l1\ x2))$
and $\bigwedge x\ x'.\ x\ {}_{L1}\!\lessapprox x' \Longrightarrow ((\leq_{L2\ x}\ (r1\ x')) \trianglelefteq (\leq_{R2}\ (l1\ x)\ x'))\ (l2_{x'}\ x)\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$((\mathit{in\text{-}dom}\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')})) \Rrightarrow (\leq_{R2\ x1'\ x2'}))\ (l2_{x1'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x1')})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$((\mathit{in\text{-}codom}\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')})) \Rrightarrow (\leq_{R2\ x1'\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x2')})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$
$((\mathit{in\text{-}dom}\ (\leq_{L2\ x\ (\eta_1\ x)})) \Rrightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (l2_{(l1\ x)\ (\eta_1\ x)})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$
$((\mathit{in\text{-}codom}\ (\leq_{R2\ (l1\ x1)\ (l1\ x2)})) \Rrightarrow (\leq_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x2)})\ (r2_{x2\ (l1\ x2)})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$
$((\mathit{in\text{-}dom}\ (\leq_{R2\ (l1\ x1)\ (l1\ x2)})) \Rrightarrow (\leq_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x1)})\ (r2_{x1\ (l1\ x2)})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$
$((\mathit{in\text{-}codom}\ (\leq_{R2\ (\varepsilon_1\ x')\ x'})) \Rrightarrow (\leq_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow \mathit{transitive}\ (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow \mathit{transitive}\ (\leq_{R2\ x1'\ x2'})$
shows $((\leq_L) \dashv (\leq_R))\ l\ r$
⟨*proof*⟩

lemma *galois-connection-left-right-if-galois-connectionI*:
assumes $((\leq_{L1}) \dashv (\leq_{R1}))\ l1\ r1$
and $\mathit{reflexive\text{-}on}\ (\mathit{in\text{-}codom}\ (\leq_{L1}))\ (\leq_{L1})$
and $\mathit{reflexive\text{-}on}\ (\mathit{in\text{-}dom}\ (\leq_{R1}))\ (\leq_{R1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\!\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \dashv (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$((\mathit{in\text{-}dom}\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')})) \Rrightarrow (\leq_{R2\ x1'\ x2'}))\ (l2_{x1'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x1')})$

and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$((\textit{in-codom}\ (\leq_{L2}\ (r1\ x1')\ (r1\ x2')))) \Rrightarrow (\leq_{R2\ x1'\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x2')})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$
$((\textit{in-dom}\ (\leq_{L2\ x\ (\eta_1\ x)}))) \Rrightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (l2_{(l1\ x)\ (\eta_1\ x)})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$
$((\textit{in-codom}\ (\leq_{R2\ (l1\ x1)\ (l1\ x2)}))) \Rrightarrow (\leq_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x2)})\ (r2_{x2\ (l1\ x2)})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$
$((\textit{in-dom}\ (\leq_{R2\ (l1\ x1)\ (l1\ x2)}))) \Rrightarrow (\leq_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x1)})\ (r2_{x1\ (l1\ x2)})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$
$((\textit{in-codom}\ (\leq_{R2\ (\varepsilon_1\ x')\ x'}))) \Rrightarrow (\leq_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow \textit{transitive}\ (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow \textit{transitive}\ (\leq_{R2\ x1'\ x2'})$
shows $((\leq_L) \dashv (\leq_R))\ l\ r$
⟨*proof*⟩

corollary *galois-connection-left-right-if-galois-connectionI'*:
assumes $((\leq_{L1}) \dashv (\leq_{R1}))\ l1\ r1$
and $\textit{reflexive-on}\ (\textit{in-field}\ (\leq_{L1}))\ (\leq_{L1})$
and $\textit{reflexive-on}\ (\textit{in-field}\ (\leq_{R1}))\ (\leq_{R1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow$
$((\leq_{L2\ x\ (r1\ x')}) \dashv (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$((\textit{in-dom}\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')})) \Rrightarrow (\leq_{R2\ x1'\ x2'}))\ (l2_{x1'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x1')})$
and $((x' : \textit{in-codom}\ (\leq_{R1})) \Rightarrow (x1\ x2 :: (\leq_{L1}) \mid x2\ {}_{L1}\lessapprox x') \Rightarrow$
$(\textit{in-field}\ (\leq_{L2\ x1\ (r1\ x')})) \Rrightarrow (\leq_{R2\ (l1\ x1)\ x'}))\ l2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$
$((\textit{in-codom}\ (\leq_{R2\ (l1\ x1)\ (l1\ x2)}))) \Rrightarrow (\leq_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x2)})\ (r2_{x2\ (l1\ x2)})$
and $((x : \textit{in-dom}\ (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x\ {}_{L1}\lessapprox x1') \Rightarrow$
$(\textit{in-field}\ (\leq_{R2\ (l1\ x)\ x2'})) \Rrightarrow (\leq_{L2\ x\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow \textit{transitive}\ (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow \textit{transitive}\ (\leq_{R2\ x1'\ x2'})$
shows $((\leq_L) \dashv (\leq_R))\ l\ r$
⟨*proof*⟩

corollary *galois-connection-left-right-if-mono-if-galois-connectionI*:
assumes $((\leq_{L1}) \dashv (\leq_{R1}))\ l1\ r1$
and $\textit{reflexive-on}\ (\textit{in-field}\ (\leq_{L1}))\ (\leq_{L1})$
and $\textit{reflexive-on}\ (\textit{in-field}\ (\leq_{R1}))\ (\leq_{R1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \dashv (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$

and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$
and $((x1'\ x2' :: (\leq_{R1})) \Rightarrow (x1\ x2 :: (\leq_{L1}) \mid x2\ {}_{L1}\lessapprox x1') \Rrightarrow$
in-field $(\leq_{L2\ x1\ (r1\ x2')}) \Rrightarrow (\leq_{R2\ (l1\ x1)\ x2'}))\ l2$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$ *transitive* $(\leq_{R2\ x1'\ x2'})$
shows $((\leq_L) \dashv (\leq_R))\ l\ r$
⟨*proof*⟩

corollary *galois-connection-left-right-if-mono-if-galois-connectionI'*:
assumes $((\leq_{L1}) \dashv (\leq_{R1}))\ l1\ r1$
and *reflexive-on* $(in\text{-}field\ (\leq_{L1}))\ (\leq_{L1})$
and *reflexive-on* $(in\text{-}field\ (\leq_{R1}))\ (\leq_{R1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \dashv (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $((\text{-}\ x2 :: (\leq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid (x2 \leq_{L1} x3 \wedge x4 \leq_{L1} \eta_1\ x3)) \Rrightarrow (\geq))$
L2
and $((x1'\ x2' :: (\leq_{R1}) \mid \varepsilon_1\ x2' \leq_{R1} x1') \Rightarrow (x3'\ \text{-} :: (\leq_{R1}) \mid x2' \leq_{R1} x3') \Rrightarrow$
$(\leq))\ R2$
and $((x1'\ x2' :: (\leq_{R1})) \Rightarrow (x1\ x2 :: (\leq_{L1}) \mid x2\ {}_{L1}\lessapprox x1') \Rrightarrow$
in-field $(\leq_{L2\ x1\ (r1\ x2')}) \Rrightarrow (\leq_{R2\ (l1\ x1)\ x2'}))\ l2$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$ *transitive* $(\leq_{R2\ x1'\ x2'})$
shows $((\leq_L) \dashv (\leq_R))\ l\ r$
⟨*proof*⟩

end

Monotone Function Relator **context** *transport-Mono-Fun-Rel* **begin**

interpretation *flip* : *transport-Mono-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2* ⟨*proof*⟩

lemma *galois-connection-left-rightI*:
assumes $((\leq_{L1}) \dashv (\leq_{R1}))\ l1\ r1$
and *reflexive-on* $(in\text{-}codom\ (\leq_{L1}))\ (\leq_{L1})$
and *reflexive-on* $(in\text{-}dom\ (\leq_{R1}))\ (\leq_{R1})$
and $((\leq_{L2}) \dashv (\leq_{R2}))\ l2\ r2$
and *transitive* $(\leq_{L2})$
and *transitive* $(\leq_{R2})$
shows $((\leq_L) \dashv (\leq_R))\ l\ r$
⟨*proof*⟩

end

end

2.15.5 Basic Order Properties

theory *Transport-Functions-Order-Base*
imports
Transport-Functions-Base
begin

Dependent Function Relator **context** *hom-Dep-Fun-Rel-orders*
begin

lemma *reflexive-on-in-domI*:
assumes *refl-L*: *reflexive-on* (*in-codom* ($\leq_L$)) ($\leq_L$)
and *R-le-R-if-L*: $\bigwedge$*x1 x2*. *x1* $\leq_L$ *x2* $\Longrightarrow$ ($\leq_R$ *x2 x2*) $\leq$ ($\leq_R$ *x1 x2*)
and *pequiv-R*: $\bigwedge$*x1 x2*. *x1* $\leq_L$ *x2* $\Longrightarrow$ *partial-equivalence-rel* ($\leq_R$ *x1 x2*)
shows *reflexive-on* (*in-dom DFR*) *DFR*
⟨*proof*⟩

lemma *reflexive-on-in-codomI*:
assumes *refl-L*: *reflexive-on* (*in-dom* ($\leq_L$)) ($\leq_L$)
and *R-le-R-if-L*: $\bigwedge$*x1 x2*. *x1* $\leq_L$ *x2* $\Longrightarrow$ ($\leq_R$ *x1 x1*) $\leq$ ($\leq_R$ *x1 x2*)
and *pequiv-R*: $\bigwedge$*x1 x2*. *x1* $\leq_L$ *x2* $\Longrightarrow$ *partial-equivalence-rel* ($\leq_R$ *x1 x2*)
shows *reflexive-on* (*in-codom DFR*) *DFR*
⟨*proof*⟩

corollary *reflexive-on-in-fieldI*:
assumes *reflexive-on* (*in-field* ($\leq_L$)) ($\leq_L$)
and $\bigwedge$*x1 x2*. *x1* $\leq_L$ *x2* $\Longrightarrow$ ($\leq_R$ *x2 x2*) $\leq$ ($\leq_R$ *x1 x2*)
and $\bigwedge$*x1 x2*. *x1* $\leq_L$ *x2* $\Longrightarrow$ ($\leq_R$ *x1 x1*) $\leq$ ($\leq_R$ *x1 x2*)
and $\bigwedge$*x1 x2*. *x1* $\leq_L$ *x2* $\Longrightarrow$ *partial-equivalence-rel* ($\leq_R$ *x1 x2*)
shows *reflexive-on* (*in-field DFR*) *DFR*
⟨*proof*⟩

lemma *transitiveI*:
assumes *refl-L*: *reflexive-on* (*in-dom* ($\leq_L$)) ($\leq_L$)
and *R-le-R-if-L*: $\bigwedge$*x1 x2*. *x1* $\leq_L$ *x2* $\Longrightarrow$ ($\leq_R$ *x1 x1*) $\leq$ ($\leq_R$ *x1 x2*)
and *trans*: $\bigwedge$*x1 x2*. *x1* $\leq_L$ *x2* $\Longrightarrow$ *transitive* ($\leq_R$ *x1 x2*)
shows *transitive DFR*
⟨*proof*⟩

lemma *transitiveI'*:
assumes *refl-L*: *reflexive-on* (*in-codom* ($\leq_L$)) ($\leq_L$)
and *R-le-R-if-L*: $\bigwedge$*x1 x2*. *x1* $\leq_L$ *x2* $\Longrightarrow$ ($\leq_R$ *x2 x2*) $\leq$ ($\leq_R$ *x1 x2*)
and *trans*: $\bigwedge$*x1 x2*. *x1* $\leq_L$ *x2* $\Longrightarrow$ *transitive* ($\leq_R$ *x1 x2*)
shows *transitive DFR*
⟨*proof*⟩

lemma *preorder-on-in-fieldI*:

assumes *reflexive-on* (*in-field* ($\leq_L$)) ($\leq_L$)
and $\bigwedge$*x1 x2*. *x1* $\leq_L$ *x2* $\Longrightarrow$ ($\leq_R$ *x2 x2*) $\leq$ ($\leq_R$ *x1 x2*)
and $\bigwedge$*x1 x2*. *x1* $\leq_L$ *x2* $\Longrightarrow$ ($\leq_R$ *x1 x1*) $\leq$ ($\leq_R$ *x1 x2*)
and *pequiv-R*: $\bigwedge$*x1 x2*. *x1* $\leq_L$ *x2* $\Longrightarrow$ *partial-equivalence-rel* ($\leq_R$ *x1 x2*)
shows *preorder-on* (*in-field DFR*) *DFR*
⟨*proof*⟩

lemma *symmetricI*:
assumes *sym-L*: *symmetric* ($\leq_L$)
and *R-le-R-if-L*: $\bigwedge$*x1 x2*. *x1* $\leq_L$ *x2* $\Longrightarrow$ ($\leq_R$ *x1 x2*) $\leq$ ($\leq_R$ *x2 x1*)
and *sym-R*: $\bigwedge$*x1 x2*. *x1* $\leq_L$ *x2* $\Longrightarrow$ *symmetric* ($\leq_R$ *x1 x2*)
shows *symmetric DFR*
⟨*proof*⟩

corollary *partial-equivalence-relI*:
assumes *reflexive-on* (*in-field* ($\leq_L$)) ($\leq_L$)
and *sym-L*: *symmetric* ($\leq_L$)
and *mono-R*: ((*x1 x2* :: ($\leq_L$)) $\Rightarrow$ (*x3 x4* :: ($\leq_L$) | *x1* $\leq_L$ *x3*) $\Rrightarrow$ ($\leq$)) *R*
and $\bigwedge$*x1 x2*. *x1* $\leq_L$ *x2* $\Longrightarrow$ *partial-equivalence-rel* ($\leq_R$ *x1 x2*)
shows *partial-equivalence-rel DFR*
⟨*proof*⟩

end

context *transport-Dep-Fun-Rel*
begin

lemmas *reflexive-on-in-field-leftI* = *dfro1.reflexive-on-in-fieldI*
[*folded left-rel-eq-Dep-Fun-Rel*]
lemmas *transitive-leftI* = *dfro1.transitiveI*[*folded left-rel-eq-Dep-Fun-Rel*]
lemmas *transitive-leftI'* = *dfro1.transitiveI'*[*folded left-rel-eq-Dep-Fun-Rel*]
lemmas *preorder-on-in-field-leftI* = *dfro1.preorder-on-in-fieldI*
[*folded left-rel-eq-Dep-Fun-Rel*]
lemmas *symmetric-leftI* = *dfro1.symmetricI*[*folded left-rel-eq-Dep-Fun-Rel*]
lemmas *partial-equivalence-rel-leftI* = *dfro1.partial-equivalence-relI*
[*folded left-rel-eq-Dep-Fun-Rel*]

Introduction Rules for Assumptions **lemma** *transitive-left2-if-transitive-left2-if-left-GaloisI*:
assumes (($\leq_{L1}$) $\Rightarrow$ ($\leq_{R1}$)) *l1*
and (($\leq_{L1}$) $\trianglelefteq_h$ ($\leq_{R1}$)) *l1 r1*
and *L2-eq*: ($\leq_{L2\ x1\ x2}$) = ($\leq_{L2\ x1\ (\eta_1\ x2)}$)
and $\bigwedge$*x x'*. *x* $_{L1}\lessapprox$ *x'* $\Longrightarrow$ *transitive* ($\leq_{L2\ x\ (r1\ x')}$)
and *x1* $\leq_{L1}$ *x2*
shows *transitive* ($\leq_{L2\ x1\ x2}$)
⟨*proof*⟩

lemma *symmetric-left2-if-symmetric-left2-if-left-GaloisI*:
assumes (($\leq_{L1}$) $\Rightarrow$ ($\leq_{R1}$)) *l1*
and (($\leq_{L1}$) $\trianglelefteq_h$ ($\leq_{R1}$)) *l1 r1*

and *L2-eq*: $(\leq_{L2\ x1\ x2}) = (\leq_{L2\ x1\ (\eta_1\ x2)})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow$ *symmetric* $(\leq_{L2\ x\ (r1\ x')})$
and $x1\ \leq_{L1}\ x2$
shows *symmetric* $(\leq_{L2\ x1\ x2})$
⟨*proof*⟩

lemma *partial-equivalence-rel-left2-if-partial-equivalence-rel-left2-if-left-GaloisI*:
assumes $((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1*
and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*
and *L2-eq*: $(\leq_{L2\ x1\ x2}) = (\leq_{L2\ x1\ (\eta_1\ x2)})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow$ *partial-equivalence-rel* $(\leq_{L2\ x\ (r1\ x')})$
and $x1\ \leq_{L1}\ x2$
shows *partial-equivalence-rel* $(\leq_{L2\ x1\ x2})$
⟨*proof*⟩

context
assumes *galois-prop*: $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
begin

interpretation *flip-inv* :
transport-Dep-Fun-Rel $(\geq_{R1})$ $(\geq_{L1})$ *r1 l1 flip2 R2 flip2 L2 r2 l2*
rewrites *flip-inv.t1.unit* $\equiv \varepsilon_1$
and $\bigwedge R\ x\ y.$ *(flip2 R x y)* $\equiv (R\ y\ x)^{-1}$
and $\bigwedge R\ S.\ R^{-1} = S^{-1} \equiv R = S$
and $\bigwedge R\ S.\ (R^{-1} \Rightarrow S^{-1}) \equiv (R \Rightarrow S)$
and $\bigwedge x\ x'.\ x'\ {}_{R1}\gtrapprox\ x \equiv x\ {}_{L1}\lessapprox\ x'$
and $((\geq_{R1}) \trianglelefteq_h (\geq_{L1}))$ *r1 l1* $\equiv$ *True*
and $\bigwedge(R :: 'z \Rightarrow 'z \Rightarrow bool).$ *transitive* $R^{-1} \equiv$ *transitive* R
and $\bigwedge(R :: 'z \Rightarrow 'z \Rightarrow bool).$ *symmetric* $R^{-1} \equiv$ *symmetric* R
and $\bigwedge(R :: 'z \Rightarrow 'z \Rightarrow bool).$ *partial-equivalence-rel* $R^{-1} \equiv$ *partial-equivalence-rel* R
and $\bigwedge P.$ $(True \Longrightarrow P) \equiv$ *Trueprop P*
and $\bigwedge P\ Q.$ $(True \Longrightarrow PROP\ P \Longrightarrow PROP\ Q) \equiv (PROP\ P \Longrightarrow True \Longrightarrow PROP\ Q)$
⟨*proof*⟩

lemma *transitive-right2-if-transitive-right2-if-left-GaloisI*:
assumes $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*
and $(\leq_{R2\ x1\ x2}) = (\leq_{R2\ (\varepsilon_1\ x1)\ x2})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow$ *transitive* $(\leq_{R2\ (l1\ x)\ x'})$
and $x1\ \leq_{R1}\ x2$
shows *transitive* $(\leq_{R2\ x1\ x2})$
⟨*proof*⟩

lemma *symmetric-right2-if-symmetric-right2-if-left-GaloisI*:
assumes $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*
and $(\leq_{R2\ x1\ x2}) = (\leq_{R2\ (\varepsilon_1\ x1)\ x2})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow$ *symmetric* $(\leq_{R2\ (l1\ x)\ x'})$

and $x1 \leq_{R1} x2$
shows *symmetric* $(\leq_{R2\ x1\ x2})$
⟨*proof*⟩

lemma *partial-equivalence-rel-right2-if-partial-equivalence-rel-right2-if-left-GaloisI*:
assumes $((\leq_{R1}) \Rightarrow (\leq_{L1}))\ r1$
and $(\leq_{R2\ x1\ x2}) = (\leq_{R2\ (\varepsilon_1\ x1)\ x2})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow$ *partial-equivalence-rel* $(\leq_{R2\ (l1\ x)\ x'})$
and $x1 \leq_{R1} x2$
shows *partial-equivalence-rel* $(\leq_{R2\ x1\ x2})$
⟨*proof*⟩

end

lemma *transitive-left2-if-preorder-equivalenceI*:
assumes *pre-equiv1*: $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))\ l1\ r1$
and $((x1\ x2 :: (\geq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rrightarrow (\leq))\ L2$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_{pre} (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $x1 \leq_{L1} x2$
shows *transitive* $(\leq_{L2\ x1\ x2})$
⟨*proof*⟩

lemma *symmetric-left2-if-partial-equivalence-rel-equivalenceI*:
assumes *PER-equiv1*: $((\leq_{L1}) \equiv_{PER} (\leq_{R1}))\ l1\ r1$
and $((x1\ x2 :: (\geq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rrightarrow (\leq))\ L2$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_{PER} (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $x1 \leq_{L1} x2$
shows *symmetric* $(\leq_{L2\ x1\ x2})$
⟨*proof*⟩

lemma *partial-equivalence-rel-left2-if-partial-equivalence-rel-equivalenceI*:
assumes *PER-equiv1*: $((\leq_{L1}) \equiv_{PER} (\leq_{R1}))\ l1\ r1$
and $((x1\ x2 :: (\geq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rrightarrow (\leq))\ L2$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_{PER} (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $x1 \leq_{L1} x2$
shows *partial-equivalence-rel* $(\leq_{L2\ x1\ x2})$
⟨*proof*⟩

interpretation *flip* : *transport-Dep-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2*
rewrites $flip.t1.counit \equiv \eta_1$ **and** $flip.t1.unit \equiv \varepsilon_1$
⟨*proof*⟩

lemma *transitive-right2-if-preorder-equivalenceI*:
assumes *pre-equiv1*: $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))\ l1\ r1$
and $((x1'\ x2' :: (\geq_{R1})) \Rightarrow (x3'\ x4' :: (\leq_{R1}) \mid x1' \leq_{R1} x3') \Rrightarrow (\leq))\ R2$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_{pre} (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $x1' \leq_{R1} x2'$
shows *transitive* $(\leq_{R2\ x1'\ x2'})$

⟨*proof*⟩

lemma *symmetric-right2-if-partial-equivalence-rel-equivalenceI*:
assumes *PER-equiv1*: $((\leq_{L1}) \equiv_{PER} (\leq_{R1}))$ *l1 r1*
and $((x1'\ x2' :: (\geq_{R1})) \Rightarrow (x3'\ x4' :: (\leq_{R1}) \mid x1' \leq_{R1} x3') \Rrightarrow (\leq))$ *R2*
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_{PER} (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $x1' \leq_{R1} x2'$
shows *symmetric* $(\leq_{R2\ x1'\ x2'})$
⟨*proof*⟩

lemma *partial-equivalence-rel-right2-if-partial-equivalence-rel-equivalenceI*:
assumes *PER-equiv1*: $((\leq_{L1}) \equiv_{PER} (\leq_{R1}))$ *l1 r1*
and $((x1'\ x2' :: (\geq_{R1})) \Rightarrow (x3'\ x4' :: (\leq_{R1}) \mid x1' \leq_{R1} x3') \Rrightarrow (\leq))$ *R2*
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_{PER} (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $x1' \leq_{R1} x2'$
shows *partial-equivalence-rel* $(\leq_{R2\ x1'\ x2'})$
⟨*proof*⟩

end

Function Relator **context** *transport-Fun-Rel*
begin

lemma *reflexive-on-in-field-leftI*:
assumes *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and *partial-equivalence-rel* $(\leq_{L2})$
shows *reflexive-on* (*in-field* $(\leq_{L})$) $(\leq_{L})$
⟨*proof*⟩

lemma *transitive-leftI*:
assumes *reflexive-on* (*in-dom* $(\leq_{L1})$) $(\leq_{L1})$
and *transitive* $(\leq_{L2})$
shows *transitive* $(\leq_{L})$
⟨*proof*⟩

lemma *transitive-leftI'*:
assumes *reflexive-on* (*in-codom* $(\leq_{L1})$) $(\leq_{L1})$
and *transitive* $(\leq_{L2})$
shows *transitive* $(\leq_{L})$
⟨*proof*⟩

lemma *preorder-on-in-field-leftI*:
assumes *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and *partial-equivalence-rel* $(\leq_{L2})$
shows *preorder-on* (*in-field* $(\leq_{L})$) $(\leq_{L})$
⟨*proof*⟩

lemma *symmetric-leftI*:
assumes *symmetric* $(\leq_{L1})$

and *symmetric* ($\leq_{L2}$)
shows *symmetric* ($\leq_L$)
⟨*proof*⟩

corollary *partial-equivalence-rel-leftI*:
assumes *reflexive-on* (*in-field* ($\leq_{L1}$)) ($\leq_{L1}$)
and *symmetric* ($\leq_{L1}$)
and *partial-equivalence-rel* ($\leq_{L2}$)
shows *partial-equivalence-rel* ($\leq_L$)
⟨*proof*⟩

end

Monotone Dependent Function Relator **context** *transport-Mono-Dep-Fun-Rel*
begin

lemmas *reflexive-on-in-field-leftI* = *Refl-Rel-reflexive-on-in-field*[*of tdfr.L*,
folded left-rel-eq-tdfr-left-Refl-Rel]
lemmas *transitive-leftI* = *Refl-Rel-transitiveI*
[*of tdfr.L*, *folded left-rel-eq-tdfr-left-Refl-Rel*]
lemmas *preorder-on-in-field-leftI* = *Refl-Rel-preorder-on-in-fieldI*[*of tdfr.L*,
folded left-rel-eq-tdfr-left-Refl-Rel]
lemmas *symmetric-leftI* = *Refl-Rel-symmetricI*[*of tdfr.L*,
OF tdfr.symmetric-leftI, *folded left-rel-eq-tdfr-left-Refl-Rel*]
lemmas *partial-equivalence-rel-leftI* = *Refl-Rel-partial-equivalence-relI*[*of tdfr.L*,
OF tdfr.partial-equivalence-rel-leftI, *folded left-rel-eq-tdfr-left-Refl-Rel*]

end

Monotone Function Relator **context** *transport-Mono-Fun-Rel*
begin

lemma *symmetric-leftI*:
assumes *symmetric* ($\leq_{L1}$)
and *symmetric* ($\leq_{L2}$)
shows *symmetric* ($\leq_L$)
⟨*proof*⟩

lemma *partial-equivalence-rel-leftI*:
assumes *reflexive-on* (*in-field* ($\leq_{L1}$)) ($\leq_{L1}$)
and *symmetric* ($\leq_{L1}$)
and *partial-equivalence-rel* ($\leq_{L2}$)
shows *partial-equivalence-rel* ($\leq_L$)
⟨*proof*⟩

end

end

2.15.6 Galois Equivalence

theory *Transport-Functions-Galois-Equivalence*
imports
Transport-Functions-Galois-Connection
Transport-Functions-Order-Base
begin

Dependent Function Relator **context** *transport-Dep-Fun-Rel*
begin

Lemmas for Monotone Function Relator **lemma** *flip-half-galois-prop-left2-if-half-galois-prop-left2-i*
assumes $((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1*
and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*
and *half-galois-prop-left2*: $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow$
$((\leq_{R2\ (l1\ x)\ x'})\ {}_h\trianglelefteq\ (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})\ (l2_{x'\ x})$
and $(\leq_{L2\ (\eta_1\ x)\ x}) = (\leq_{L2\ x\ x})$
and $(\leq_{L2\ x\ (\eta_1\ x)}) = (\leq_{L2\ x\ x})$
and $x \leq_{L1} x$
shows $((\leq_{R2\ (l1\ x)\ (l1\ x)})\ {}_h\trianglelefteq\ (\leq_{L2\ (\eta_1\ x)\ x}))\ (r2_{x\ (l1\ x)})\ (l2_{(l1\ x)\ x})$
⟨*proof*⟩

lemma *flip-half-galois-prop-right2-if-half-galois-prop-right2-if-GaloisI*:
assumes $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*
and *half-galois-prop-right2*: $\bigwedge x\ x'.\ x\ {}_{L1}\gtrapprox x' \Longrightarrow$
$((\leq_{R2\ (l1\ x)\ x'})\ \trianglelefteq_h\ (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})\ (l2_{x'\ x})$
and $(\leq_{R2\ (\varepsilon_1\ x')\ x'}) = (\leq_{R2\ x'\ x'})$
and $(\leq_{R2\ x'\ (\varepsilon_1\ x')}) = (\leq_{R2\ x'\ x'})$
and $x' \leq_{R1} x'$
shows $((\leq_{R2\ x'\ (\varepsilon_1\ x')})\ \trianglelefteq_h\ (\leq_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ x'})\ (l2_{x'\ (r1\ x')})$
⟨*proof*⟩

interpretation *flip* : *transport-Dep-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2*
rewrites *flip.t1.counit* $\equiv \eta_1$ **and** *flip.t1.unit* $\equiv \varepsilon_1$
⟨*proof*⟩

lemma *galois-equivalence-if-mono-if-galois-equivalence-mono-assms-leftI*:
assumes *galois-equiv1*: $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1*
and *preorder-L1*: *preorder-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and *mono-L2*: $((x1\ x2 :: (\geq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rrightarrow (\leq))\ L2$
shows $((x1\ x2 :: (\leq_{L1}) \mid \eta_1\ x2 \leq_{L1} x1) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid x2 \leq_{L1} x3) \Rrightarrow (\leq))\ L2$ (**is** *?goal1*)
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid (x2 \leq_{L1} x3 \wedge x4 \leq_{L1} \eta_1\ x3)) \Rrightarrow (\geq))\ L2$ (**is** *?goal2*)
⟨*proof*⟩

lemma *galois-equivalence-if-mono-if-galois-equivalence-Dep-Fun-Rel-pred-assm-leftI*:
assumes *galois-equiv1*: $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1*

and *refl-L1*: *reflexive-on* (*in-field* ($\leq_{L1}$)) ($\leq_{L1}$)
and *refl-R1*: *reflexive-on* (*in-field* ($\leq_{R1}$)) ($\leq_{R1}$)
and *mono-L2*: ((*x1 x2* :: ($\geq_{L1}$)) $\Rightarrow$ (*x3 x4* :: ($\leq_{L1}$) | *x1* $\leq_{L1}$ *x3*) $\Rrightarrow$ ($\leq$)) *L2*
and *mono-R2*: ((*x1'* *x2'* :: ($\geq_{R1}$)) $\Rightarrow$ (*x3'* *x4'* :: ($\leq_{R1}$) | *x1'* $\leq_{R1}$ *x3'*) $\Rrightarrow$ ($\leq$)) *R2*
and *mono-l2*: ((*x1'* *x2'* :: ($\leq_{R1}$)) $\Rightarrow$ (*x1 x2* :: ($\leq_{L1}$) | *x2* $_{L1}\lessapprox$ *x1'*) $\Rrightarrow$ *in-field* ($\leq_{L2\ x1\ (r1\ x2')}$) $\Rrightarrow$ ($\leq_{R2\ (l1\ x1)\ x2'}$)) *l2*
and $x \leq_{L1} x$
shows (*in-codom* ($\leq_{L2\ (\eta_1\ x)\ x}$) $\Rrightarrow$ ($\leq_{R2\ (l1\ x)\ (l1\ x)}$)) ($l2_{(l1\ x)\ (\eta_1\ x)}$) ($l2_{(l1\ x)\ x}$)
⟨*proof*⟩

lemma *galois-equivalence-if-mono-if-galois-equivalence-Dep-Fun-Rel-pred-assm-right*:
assumes *galois-equiv1*: (($\leq_{L1}$) $\equiv_G$ ($\leq_{R1}$)) *l1 r1*
and *refl-R1*: *reflexive-on* (*in-field* ($\leq_{R1}$)) ($\leq_{R1}$)
and *mono-L2*: ((*x1 x2* :: ($\geq_{L1}$)) $\Rightarrow$ (*x3 x4* :: ($\leq_{L1}$) | *x1* $\leq_{L1}$ *x3*) $\Rrightarrow$ ($\leq$)) *L2*
and *mono-R2*: ((*x1'* *x2'* :: ($\geq_{R1}$)) $\Rightarrow$ (*x3'* *x4'* :: ($\leq_{R1}$) | *x1'* $\leq_{R1}$ *x3'*) $\Rrightarrow$ ($\leq$)) *R2*
and *mono-r2*: ((*x1 x2* :: ($\leq_{L1}$)) $\Rightarrow$ (*x1'* *x2'* :: ($\leq_{R1}$) | *x2* $_{L1}\lessapprox$ *x1'*) $\Rrightarrow$ *in-field* ($\leq_{R2\ (l1\ x1)\ x2'}$) $\Rrightarrow$ ($\leq_{L2\ x1\ (r1\ x2')}$)) *r2*
and $x' \leq_{R1} x'$
shows (*in-dom* ($\leq_{R2\ x'\ (\varepsilon_1\ x')}$) $\Rrightarrow$ ($\leq_{L2\ (r1\ x')\ (r1\ x')}$)) ($r2_{(r1\ x')\ x'}$) ($r2_{(r1\ x')\ (\varepsilon_1\ x')}$)
⟨*proof*⟩

end

Monotone Dependent Function Relator **context** *transport-Mono-Dep-Fun-Rel*
begin

context
begin

interpretation *flip* : *transport-Mono-Dep-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2*
rewrites *flip.t1.counit* $\equiv \eta_1$ **and** *flip.t1.unit* $\equiv \varepsilon_1$
⟨*proof*⟩

lemma *galois-equivalence-if-galois-equivalenceI*:
assumes *galois-equiv1*: (($\leq_{L1}$) $\equiv_G$ ($\leq_{R1}$)) *l1 r1*
and *refl-L1*: *reflexive-on* (*in-field* ($\leq_{L1}$)) ($\leq_{L1}$)
and *reflexive-on* (*in-field* ($\leq_{R1}$)) ($\leq_{R1}$)
and *galois-equiv2*: $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow$ (($\leq_{L2\ x\ (r1\ x')}$) $\equiv_G$ ($\leq_{R2\ (l1\ x)\ x'}$)) ($l2_{x'\ x}$) ($r2_{x\ x'}$)
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow (\leq_{L2\ (\eta_1\ x)\ x}) \leq (\leq_{L2\ x\ x})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x2'\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$

and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow (\leq_{R2\ x'\ (\varepsilon_1\ x')}) \leq (\leq_{R2\ x'\ x'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$(\mathit{in\text{-}dom}\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')}) \Rrightarrow (\leq_{R2\ x1'\ x2'}))\ (l2_{x1'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x1')})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$(\mathit{in\text{-}codom}\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')}) \Rrightarrow (\leq_{R2\ x1'\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x2')})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$
$(\mathit{in\text{-}dom}\ (\leq_{L2\ x\ (\eta_1\ x)}) \Rrightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (l2_{(l1\ x)\ (\eta_1\ x)})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$
$(\mathit{in\text{-}codom}\ (\leq_{L2\ (\eta_1\ x)\ x}) \Rrightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ (\eta_1\ x)})\ (l2_{(l1\ x)\ x})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$
$(\mathit{in\text{-}codom}\ (\leq_{R2\ (l1\ x1)\ (l1\ x2)}) \Rrightarrow (\leq_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x2)})\ (r2_{x2\ (l1\ x2)})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$
$(\mathit{in\text{-}dom}\ (\leq_{R2\ (l1\ x1)\ (l1\ x2)}) \Rrightarrow (\leq_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x1)})\ (r2_{x1\ (l1\ x2)})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$
$(\mathit{in\text{-}codom}\ (\leq_{R2\ (\varepsilon_1\ x')\ x'}) \Rrightarrow (\leq_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$
$(\mathit{in\text{-}dom}\ (\leq_{R2\ x'\ (\varepsilon_1\ x')}) \Rrightarrow (\leq_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ x'})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow \mathit{transitive}\ (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow \mathit{transitive}\ (\leq_{R2\ x1'\ x2'})$
shows $((\leq_L) \equiv_G (\leq_R))\ l\ r$
⟨*proof*⟩

corollary *galois-equivalence-if-galois-equivalenceI'*:
assumes $((\leq_{L1}) \equiv_G (\leq_{R1}))\ l1\ r1$
and $\mathit{reflexive\text{-}on}\ (\mathit{in\text{-}field}\ (\leq_{L1}))\ (\leq_{L1})$
and $\mathit{reflexive\text{-}on}\ (\mathit{in\text{-}field}\ (\leq_{R1}))\ (\leq_{R1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\!\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_G (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow (\leq_{L2\ (\eta_1\ x)\ x}) \leq (\leq_{L2\ x\ x})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x2'\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow (\leq_{R2\ x'\ (\varepsilon_1\ x')}) \leq (\leq_{R2\ x'\ x'})$
and $((x1'\ x2' :: (\leq_{R1})) \Rightarrow (x1\ x2 :: (\leq_{L1}) \mid x2\ {}_{L1}\!\lessapprox x1') \Rrightarrow$
$\mathit{in\text{-}field}\ (\leq_{L2\ x1\ (r1\ x2')}) \Rrightarrow (\leq_{R2\ (l1\ x1)\ x2'}))\ l2$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$
$(\mathit{in\text{-}codom}\ (\leq_{L2\ (\eta_1\ x)\ x}) \Rrightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ (\eta_1\ x)})\ (l2_{(l1\ x)\ x})$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\!\lessapprox x1') \Rrightarrow$
$\mathit{in\text{-}field}\ (\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$
$(\mathit{in\text{-}dom}\ (\leq_{R2\ x'\ (\varepsilon_1\ x')}) \Rrightarrow (\leq_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ x'})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow \mathit{transitive}\ (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow \mathit{transitive}\ (\leq_{R2\ x1'\ x2'})$
shows $((\leq_L) \equiv_G (\leq_R))\ l\ r$

⟨*proof*⟩

corollary *galois-equivalence-if-mono-if-galois-equivalenceI*:
assumes $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and *reflexive-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and $\bigwedge x\ x'.\ x\ _{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_G (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $((x1\ x2 :: (\leq_{L1}) \mid \eta_1\ x2 \leq_{L1} x1) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid x2 \leq_{L1} x3) \Rrightarrow (\leq))$ *L2*
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid (x2 \leq_{L1} x3 \wedge x4 \leq_{L1} \eta_1\ x3)) \Rrightarrow (\geq))$ *L2*
and $((x1'\ x2' :: (\leq_{R1}) \mid \varepsilon_1\ x2' \leq_{R1} x1') \Rightarrow (x3'\ x4' :: (\leq_{R1}) \mid x2' \leq_{R1} x3') \Rrightarrow (\leq))$ *R2*
and $((x1'\ x2' :: (\leq_{R1})) \Rightarrow (x3'\ x4' :: (\leq_{R1}) \mid (x2' \leq_{R1} x3' \wedge x4' \leq_{R1} \varepsilon_1\ x3')) \Rrightarrow (\geq))$ *R2*
and $((x1'\ x2' :: (\leq_{R1})) \Rightarrow (x1\ x2 :: (\leq_{L1}) \mid x2\ _{L1}\lessapprox x1') \Rrightarrow$
in-field $(\leq_{L2\ x1\ (r1\ x2')}) \Rrightarrow (\leq_{R2\ (l1\ x1)\ x2'}))\ l2$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$
$(\textit{in-codom}\ (\leq_{L2\ (\eta_1\ x)\ x}) \Rrightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ (\eta_1\ x)})\ (l2_{(l1\ x)\ x})$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ _{L1}\lessapprox x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$
$(\textit{in-dom}\ (\leq_{R2\ x'\ (\varepsilon_1\ x')}) \Rrightarrow (\leq_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ x'})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow \textit{transitive}\ (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow \textit{transitive}\ (\leq_{R2\ x1'\ x2'})$
shows $((\leq_L) \equiv_G (\leq_R))\ l\ r$
⟨*proof*⟩

end

interpretation *flip* : *transport-Mono-Dep-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2*
rewrites *flip.t1.counit* $\equiv \eta_1$ **and** *flip.t1.unit* $\equiv \varepsilon_1$
⟨*proof*⟩

lemma *galois-equivalence-if-mono-if-preorder-equivalenceI*:
assumes $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and $\bigwedge x\ x'.\ x\ _{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_G (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $((x1\ x2 :: (\geq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rrightarrow (\leq))$ *L2*
and $((x1'\ x2' :: (\geq_{R1})) \Rightarrow (x3'\ x4' :: (\leq_{R1}) \mid x1' \leq_{R1} x3') \Rrightarrow (\leq))$ *R2*
and $((x1'\ x2' :: (\leq_{R1})) \Rightarrow (x1\ x2 :: (\leq_{L1}) \mid x2\ _{L1}\lessapprox x1') \Rrightarrow$
in-field $(\leq_{L2\ x1\ (r1\ x2')}) \Rrightarrow (\leq_{R2\ (l1\ x1)\ x2'}))\ l2$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ _{L1}\lessapprox x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow \textit{transitive}\ (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow \textit{transitive}\ (\leq_{R2\ x1'\ x2'})$
shows $((\leq_L) \equiv_G (\leq_R))\ l\ r$
⟨*proof*⟩

theorem *galois-equivalence-if-mono-if-preorder-equivalenceI'*:
assumes $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_{pre} (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $((x1\ x2 :: (\geq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rrightarrow (\leq))$ *L2*
and $((x1'\ x2' :: (\geq_{R1})) \Rightarrow (x3'\ x4' :: (\leq_{R1}) \mid x1' \leq_{R1} x3') \Rrightarrow (\leq))$ *R2*
and $((x1'\ x2' :: (\leq_{R1})) \Rightarrow (x1\ x2 :: (\leq_{L1}) \mid x2\ {}_{L1}\lessapprox x1') \Rrightarrow$
in-field $(\leq_{L2\ x1\ (r1\ x2')}) \Rrightarrow (\leq_{R2\ (l1\ x1)\ x2'}))$ *l2*
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))$ *r2*
shows $((\leq_L) \equiv_G (\leq_R))$ *l r*
⟨*proof*⟩

end

Monotone Function Relator **context** *transport-Mono-Fun-Rel*
begin

interpretation *flip* : *transport-Mono-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2* ⟨*proof*⟩

lemma *galois-equivalenceI*:
assumes $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and *reflexive-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and $((\leq_{L2}) \equiv_G (\leq_{R2}))$ *l2 r2*
and *transitive* $(\leq_{L2})$
and *transitive* $(\leq_{R2})$
shows $((\leq_L) \equiv_G (\leq_R))$ *l r*
⟨*proof*⟩

end

end

2.15.7 Simplification of Left and Right Relations

theory *Transport-Functions-Relation-Simplifications*
imports
Transport-Functions-Order-Base
Transport-Functions-Galois-Equivalence
begin

Dependent Function Relator **context** *transport-Dep-Fun-Rel*
begin

Due to *reflexive-on* (*in-field* (*transport-Dep-Fun-Rel.L ?L1.0 ?L2.0*)) (*transport-Dep-Fun-Rel.L ?L1.0 ?L2.0*) $\Longrightarrow$ *transport-Mono-Dep-Fun-Rel.L ?L1.0 ?L2.0 = transport-Dep-Fun-Rel.L ?L1.0 ?L2.0*, we can apply all re-

sults from *transport-Mono-Dep-Fun-Rel* to *transport-Dep-Fun-Rel* whenever $(\leq_L)$ and $(\leq_R)$ are reflexive.

lemma *reflexive-on-in-field-left-rel2-le-assmI*:
assumes *refl-L1*: *reflexive-on* (*in-dom* $(\leq_{L1})$) $(\leq_{L1})$
and *mono-L2*: $((x1 : \top) \Rightarrow (x2\ x3 :: (\leq_{L1}) \mid x1 \leq_{L1} x2) \Rightarrow (\leq))\ L2$
and $x1 \leq_{L1} x2$
shows $(\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
⟨*proof*⟩

lemma *reflexive-on-in-field-mono-assm-left2I*:
assumes *mono-L2*: $((x1\ x2 :: (\geq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rrightarrow (\leq))$ *L2*
and *refl-L1*: *reflexive-on* (*in-dom* $(\leq_{L1})$) $(\leq_{L1})$
shows $((x1 : \top) \Rightarrow (x2\ x3 :: (\leq_{L1}) \mid x1 \leq_{L1} x2) \Rightarrow (\leq))\ L2$
⟨*proof*⟩

lemma *reflexive-on-in-field-left-if-equivalencesI*:
assumes $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1*
and *preorder-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and $((x1\ x2 :: (\geq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rrightarrow (\leq))\ L2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *partial-equivalence-rel* $(\leq_{L2\ x1\ x2})$
shows *reflexive-on* (*in-field* $(\leq_L)$) $(\leq_L)$
⟨*proof*⟩

end

Monotone Dependent Function Relator **context** *transport-Mono-Dep-Fun-Rel* **begin**

lemma *left-rel-eq-tdfr-leftI*:
assumes *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *partial-equivalence-rel* $(\leq_{L2\ x1\ x2})$
shows $(\leq_L) =$ *tdfr.L*
⟨*proof*⟩

lemma *left-rel-eq-tdfr-leftI-if-equivalencesI*:
assumes $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1*
and *preorder-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and $((x1\ x2 :: (\geq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rrightarrow (\leq))\ L2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *partial-equivalence-rel* $(\leq_{L2\ x1\ x2})$
shows $(\leq_L) =$ *tdfr.L*
⟨*proof*⟩

end

Monotone Function Relator **context** *transport-Mono-Fun-Rel* **begin**

lemma *left-rel-eq-tfr-leftI*:
assumes *reflexive-on* (*in-field* ($\leq_{L1}$)) ($\leq_{L1}$)
and *partial-equivalence-rel* ($\leq_{L2}$)
shows ($\leq_L$) = *tfr.tdfr.L*
⟨*proof*⟩

end

end

2.15.8 Galois Relator

theory *Transport-Functions-Galois-Relator*
imports
Transport-Functions-Relation-Simplifications
begin

Dependent Function Relator **context** *transport-Dep-Fun-Rel*
begin

interpretation *flip* : *transport-Dep-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2*
rewrites *flip.t1.counit* ≡ η_1 ⟨*proof*⟩

lemma *Dep-Fun-Rel-left-Galois-if-left-GaloisI*:
assumes (($\leq_{L1}$) $_h\trianglelefteq$ ($\leq_{R1}$)) *l1 r1*
and *refl-L1*: *reflexive-on* (*in-dom* ($\leq_{L1}$)) ($\leq_{L1}$)
and *mono-r2*: $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rightarrow (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and *L2-le2*: $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
and *ge-L2-r2-le2*: $\bigwedge x\ x'\ y'.\ x\ {}_{L1}\gtrapprox x' \Longrightarrow$ *in-dom* $(\leq_{R2\ (l1\ x)\ x'})\ y' \Longrightarrow$
$(\geq_{L2\ x\ (r1\ x')})\ (r2_{x\ (l1\ x)}\ y') \leq (\geq_{L2\ x\ (r1\ x')})\ (r2_{x\ x'}\ y')$
and *trans-L2*: $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $g \leq_R g$
and $f\ {}_{L}\lessapprox g$
shows $((x\ x' :: ({}_{L1}\lessapprox)) \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
⟨*proof*⟩

lemma *left-rel-right-if-Dep-Fun-Rel-left-GaloisI*:
assumes *mono-l1*: (($\leq_{L1}$) ⇒ ($\leq_{R1}$)) *l1*
and *half-galois-prop-right1*: (($\leq_{L1}$) $\trianglelefteq_h$ ($\leq_{R1}$)) *l1 r1*
and *L2-unit-le2*: $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and *ge-L2-r2-le1*: $\bigwedge x1\ x2\ y'.\ x1 \leq_{L1} x2 \Longrightarrow$ *in-codom* $(\leq_{R2\ (l1\ x1)\ (l1\ x2)})\ y'$
$\Longrightarrow$
$(\geq_{L2\ x1\ x2})\ (r2_{x1\ (l1\ x2)}\ y') \leq (\geq_{L2\ x1\ x2})\ (r2_{x2\ (l1\ x2)}\ y')$
and *rel-f-g*: $((x\ x' :: ({}_{L1}\lessapprox)) \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
shows $f \leq_L r\ g$
⟨*proof*⟩

lemma *left-Galois-if-Dep-Fun-Rel-left-GaloisI*:
assumes $((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1*
and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2\ y'.\ x1 \leq_{L1} x2 \Longrightarrow$ *in-codom* $(\leq_{R2\ (l1\ x1)\ (l1\ x2)})\ y' \Longrightarrow$
$(\geq_{L2\ x1\ x2})\ (r2_{x1\ (l1\ x2)}\ y') \leq (\geq_{L2\ x1\ x2})\ (r2_{x2\ (l1\ x2)}\ y')$
and *in-codom* $(\leq_R)\ g$
and $((x\ x' :: (_{L1}\overset{\leqslant}{\approx})) \Rrightarrow (_{L2\ x\ x'}\overset{\leqslant}{\approx}))\ f\ g$
shows $f\ _{L}\overset{\leqslant}{\approx}\ g$
⟨*proof*⟩

lemma *left-right-rel-if-Dep-Fun-Rel-left-GaloisI*:
assumes *mono-r1*: $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*
and *half-galois-prop-left2*: $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$((\leq_{L2\ (r1\ x1')\ (r1\ x2')})\ {}_h\trianglelefteq\ (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (r2_{(r1\ x1')\ x2'})$
and *R2-le1*: $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and *R2-l2-le1*: $\bigwedge x1'\ x2'\ y.\ x1' \leq_{R1} x2' \Longrightarrow$ *in-dom* $(\leq_{L2\ (r1\ x1')\ (r1\ x2')})\ y$
$\Longrightarrow$
$(\leq_{R2\ x1'\ x2'})\ (l2_{x2'\ (r1\ x1')}\ y) \leq (\leq_{R2\ x1'\ x2'})\ (l2_{x1'\ (r1\ x1')}\ y)$
and *rel-f-g*: $((x\ x' :: (_{L1}\overset{\leqslant}{\approx})) \Rrightarrow (_{L2\ x\ x'}\overset{\leqslant}{\approx}))\ f\ g$
shows $l\ f \leq_R g$
⟨*proof*⟩

lemma *left-Galois-if-Dep-Fun-Rel-left-GaloisI'*:
assumes $((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1* **and** $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*
and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$((\leq_{L2\ (r1\ x1')\ (r1\ x2')})\ {}_h\trianglelefteq\ (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (r2_{(r1\ x1')\ x2'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1\ x2\ y'.\ x1 \leq_{L1} x2 \Longrightarrow$ *in-codom* $(\leq_{R2\ (l1\ x1)\ (l1\ x2)})\ y' \Longrightarrow$
$(\geq_{L2\ x1\ x2})\ (r2_{x1\ (l1\ x2)}\ y') \leq (\geq_{L2\ x1\ x2})\ (r2_{x2\ (l1\ x2)}\ y')$
and $\bigwedge x1'\ x2'\ y.\ x1' \leq_{R1} x2' \Longrightarrow$ *in-dom* $(\leq_{L2\ (r1\ x1')\ (r1\ x2')})\ y \Longrightarrow$
$(\leq_{R2\ x1'\ x2'})\ (l2_{x2'\ (r1\ x1')}\ y) \leq (\leq_{R2\ x1'\ x2'})\ (l2_{x1'\ (r1\ x1')}\ y)$
and $((x\ x' :: (_{L1}\overset{\leqslant}{\approx})) \Rrightarrow (_{L2\ x\ x'}\overset{\leqslant}{\approx}))\ f\ g$
shows $f\ _{L}\overset{\leqslant}{\approx}\ g$
⟨*proof*⟩

lemma *left-Galois-iff-Dep-Fun-Rel-left-GaloisI*:
assumes $((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1*
and $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-dom* $(\leq_{L1})$) $(\leq_{L1})$
and $\bigwedge x\ x'.\ x\ _{L1}\overset{\leqslant}{\approx}\ x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rightarrow (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$

and $\bigwedge x1\ x2\ y'.\ x1 \leq_{L1} x2 \Longrightarrow \textit{in-codom}\ (\leq_{R2}\ (l1\ x1)\ (l1\ x2))\ y' \Longrightarrow$
$(\geq_{L2}\ x1\ x2)\ (r2_{x1}\ (l1\ x2)\ y') \leq (\geq_{L2}\ x1\ x2)\ (r2_{x2}\ (l1\ x2)\ y')$
and $\bigwedge x\ x'\ y'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow \textit{in-dom}\ (\leq_{R2}\ (l1\ x)\ x')\ y' \Longrightarrow$
$(\geq_{L2}\ x\ (r1\ x'))\ (r2_{x}\ (l1\ x)\ y') \leq (\geq_{L2}\ x\ (r1\ x'))\ (r2_{x\ x'}\ y')$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow \textit{transitive}\ (\leq_{L2}\ x1\ x2)$
and $g \leq_R g$
shows $f\ {}_{L}\lessapprox\ g \longleftrightarrow ((x\ x' :: (_{L1}\lessapprox)) \Rrightarrow (_{L2\ x\ x'}\lessapprox))\ f\ g$
⟨*proof*⟩

lemma *left-Galois-iff-Dep-Fun-Rel-left-Galois-ge-left-rel2-assmI*:
assumes $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$
$((\textit{in-codom}\ (\leq_{R2}\ (l1\ x1)\ (l1\ x2))) \Rrightarrow (\leq_{L2}\ x1\ x2))\ (r2_{x1}\ (l1\ x2))\ (r2_{x2}\ (l1\ x2))$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow \textit{transitive}\ (\leq_{L2}\ x1\ x2)$
shows $\bigwedge x1\ x2\ y'.\ x1 \leq_{L1} x2 \Longrightarrow \textit{in-codom}\ (\leq_{R2}\ (l1\ x1)\ (l1\ x2))\ y' \Longrightarrow$
$(\geq_{L2}\ x1\ x2)\ (r2_{x1}\ (l1\ x2)\ y') \leq (\geq_{L2}\ x1\ x2)\ (r2_{x2}\ (l1\ x2)\ y')$
⟨*proof*⟩

lemma *left-Galois-iff-Dep-Fun-Rel-left-Galois-ge-left-rel2-assmI'*:
assumes $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow$
$((\textit{in-dom}\ (\leq_{R2}\ (l1\ x)\ x')) \Rrightarrow (\leq_{L2}\ x\ (r1\ x')))\ (r2_{x}\ (l1\ x))\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow \textit{transitive}\ (\leq_{L2}\ x1\ x2)$
shows $\bigwedge x\ x'\ y'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow \textit{in-dom}\ (\leq_{R2}\ (l1\ x)\ x')\ y' \Longrightarrow$
$(\geq_{L2}\ x\ (r1\ x'))\ (r2_{x}\ (l1\ x)\ y') \leq (\geq_{L2}\ x\ (r1\ x'))\ (r2_{x\ x'}\ y')$
⟨*proof*⟩

lemma *left-Galois-iff-Dep-Fun-Rel-left-Galois-mono-assm-in-codom-rightI*:
assumes *mono-l1*: $((\leq_{L1}) \Rightarrow (\leq_{R1}))\ l1$
and *half-galois-prop-right1*: $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))\ l1\ r1$
and *refl-L1*: *reflexive-on* $(\textit{in-codom}\ (\leq_{L1}))\ (\leq_{L1})$
and *L2-le-unit2*: $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2}\ x1\ (\eta_1\ x2)) \leq (\leq_{L2}\ x1\ x2)$
and *mono-r2*: $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1') \Rrightarrow$
$\textit{in-field}\ (\leq_{R2}\ (l1\ x1)\ x2') \Rrightarrow (\leq_{L2}\ x1\ (r1\ x2')))\ r2$
and $x1 \leq_{L1} x2$
shows $((\textit{in-codom}\ (\leq_{R2}\ (l1\ x1)\ (l1\ x2))) \Rrightarrow (\leq_{L2}\ x1\ x2))\ (r2_{x1}\ (l1\ x2))\ (r2_{x2}\ (l1\ x2))$
⟨*proof*⟩

lemma *left-Galois-iff-Dep-Fun-Rel-left-Galois-mono-assm-in-dom-rightI*:
assumes *mono-l1*: $((\leq_{L1}) \Rightarrow (\leq_{R1}))\ l1$
and *half-galois-prop-right1*: $((\leq_{L1}) \trianglelefteq (\leq_{R1}))\ l1\ r1$
and *refl-L1*: *reflexive-on* $(\textit{in-dom}\ (\leq_{L1}))\ (\leq_{L1})$
and *mono-r2*: $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1') \Rrightarrow$
$\textit{in-field}\ (\leq_{R2}\ (l1\ x1)\ x2') \Rrightarrow (\leq_{L2}\ x1\ (r1\ x2')))\ r2$
and $x\ {}_{L1}\lessapprox\ x'$
shows $((\textit{in-dom}\ (\leq_{R2}\ (l1\ x)\ x')) \Rrightarrow (\leq_{L2}\ x\ (r1\ x')))\ (r2_{x}\ (l1\ x))\ (r2_{x\ x'})$
⟨*proof*⟩

lemma *left-Galois-iff-Dep-Fun-Rel-left-Galois-if-monoI*:
assumes $((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1*
and $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rightarrow (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\gtrapprox x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $g \leq_R g$
shows $f\ {}_{L}\lessapprox\ g \longleftrightarrow ((x\ x' :: ({}_{L1}\gtrapprox)) \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
⟨*proof*⟩

lemma *left-Galois-iff-Dep-Fun-Rel-left-Galois-left-rel2-le-assmI*:
assumes *refl-L1*: *reflexive-on* (*in-dom* $(\leq_{L1})$) $(\leq_{L1})$
and *mono-L2*: $((x1 : \top) \Rightarrow (x2\ \text{-} :: (\leq_{L1}) \mid x1 \leq_{L1} x2) \Rightarrow (\leq))\ L2$
and $x1 \leq_{L1} x2$
shows $(\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
⟨*proof*⟩

lemma *left-Galois-iff-Dep-Fun-Rel-left-Galois-left-rel2-unit1-le-assmI*:
assumes *mono-l1*: $((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1*
and *half-galois-prop-right1*: $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*
and *refl-L1*: *reflexive-on* (*in-codom* $(\leq_{L1})$) $(\leq_{L1})$
and *antimono-L2*:
$((x1 : \top) \Rightarrow (x2\ x3 :: (\leq_{L1}) \mid (x1 \leq_{L1} x2 \wedge x3 \leq_{L1} \eta_1\ x2)) \Rightarrow (\geq))\ L2$
and $x1 \leq_{L1} x2$
shows $(\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
⟨*proof*⟩

lemma *left-Galois-iff-Dep-Fun-Rel-left-Galois-if-monoI'*:
assumes $((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1*
and $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rightarrow (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $((x1 : \top) \Rightarrow (x2\ \text{-} :: (\leq_{L1}) \mid x1 \leq_{L1} x2) \Rightarrow (\leq))\ L2$
and $((x1 : \top) \Rightarrow (x2\ x3 :: (\leq_{L1}) \mid (x1 \leq_{L1} x2 \wedge x3 \leq_{L1} \eta_1\ x2)) \Rightarrow (\geq))\ L2$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\gtrapprox x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $g \leq_R g$
shows $f\ {}_{L}\lessapprox\ g \longleftrightarrow ((x\ x' :: ({}_{L1}\gtrapprox)) \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
⟨*proof*⟩

corollary *left-Galois-iff-Dep-Fun-Rel-left-Galois-if-mono-if-galois-connectionI*:
assumes $((\leq_{L1}) \dashv (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$

and $\bigwedge x\ x'.\ x\ _{L1}\lessapprox\ x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rightarrow (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $((x1 : \top) \Rightarrow (x2 \text{ - :: } (\leq_{L1}) \mid x1 \leq_{L1} x2) \Rightarrow (\leq))\ L2$
and $((x1 : \top) \Rightarrow (x2\ x3 :: (\leq_{L1}) \mid (x1 \leq_{L1} x2 \wedge x3 \leq_{L1} \eta_1\ x2)) \Rightarrow (\geq))\ L2$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ _{L1}\lessapprox\ x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $g \leq_R g$
shows $f\ _L\lessapprox\ g \longleftrightarrow ((x\ x' :: (_{L1}\lessapprox)) \Rrightarrow (_{L2\ x\ x'}\lessapprox))\ f\ g$
⟨*proof*⟩

interpretation *flip-inv* : *galois* $(\geq_{R1})\ (\geq_{L1})\ r1\ l1$ ⟨*proof*⟩

lemma *left-Galois-iff-Dep-Fun-Rel-left-Galois-left-rel2-unit1-le-assm-if-galois-equivI*:
assumes *galois-equiv1*: $((\leq_{L1}) \equiv_G (\leq_{R1}))\ l1\ r1$
and *refl-L1*: *reflexive-on* $(in\text{-}field\ (\leq_{L1}))\ (\leq_{L1})$
and *mono-L2*: $((x1 : \top) \Rightarrow (x2 \text{ - :: } (\leq_{L1}) \mid x1 \leq_{L1} x2) \Rightarrow (\leq))\ L2$
and $x1 \leq_{L1} x2$
shows $(\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
⟨*proof*⟩

lemma *left-Galois-iff-Dep-Fun-Rel-left-Galois-if-galois-equivalenceI*:
assumes $((\leq_{L1}) \equiv_G (\leq_{R1}))\ l1\ r1$
and *reflexive-on* $(in\text{-}field\ (\leq_{L1}))\ (\leq_{L1})$
and $\bigwedge x\ x'.\ x\ _{L1}\lessapprox\ x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rightarrow (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $((x1 : \top) \Rightarrow (x2 \text{ - :: } (\leq_{L1}) \mid x1 \leq_{L1} x2) \Rightarrow (\leq))\ L2$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ _{L1}\lessapprox\ x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $g \leq_R g$
shows $f\ _L\lessapprox\ g \longleftrightarrow ((x\ x' :: (_{L1}\lessapprox)) \Rrightarrow (_{L2\ x\ x'}\lessapprox))\ f\ g$
⟨*proof*⟩

corollary *left-Galois-iff-Dep-Fun-Rel-left-Galois-if-galois-equivalenceI'*:
assumes $((\leq_{L1}) \equiv_G (\leq_{R1}))\ l1\ r1$
and *reflexive-on* $(in\text{-}field\ (\leq_{L1}))\ (\leq_{L1})$
and $\bigwedge x\ x'.\ x\ _{L1}\lessapprox\ x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rightarrow (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $((x1\ x2 :: (\geq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rrightarrow (\leq))\ L2$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ _{L1}\lessapprox\ x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $g \leq_R g$
shows $f\ _L\lessapprox\ g \longleftrightarrow ((x\ x' :: (_{L1}\lessapprox)) \Rrightarrow (_{L2\ x\ x'}\lessapprox))\ f\ g$
⟨*proof*⟩

corollary *left-Galois-iff-Dep-Fun-Rel-left-Galois-if-preorder-equivalenceI*:
assumes $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))\ l1\ r1$
and $\bigwedge x\ x'.\ x\ _{L1}\lessapprox\ x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rightarrow (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $((x1\ x2 :: (\geq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rrightarrow (\leq))\ L2$

and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lesssim\!\!\approx\ x1') \Rrightarrow$
$\quad$ *in-field* $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $g \leq_R g$
shows $f\ {}_L\lesssim\!\!\approx\ g \longleftrightarrow ((x\ x' :: ({}_{L1}\lesssim\!\!\approx)) \Rrightarrow ({}_{L2\ x\ x'}\lesssim\!\!\approx))\ f\ g$
⟨*proof*⟩

corollary *left-Galois-iff-Dep-Fun-Rel-left-Galois-if-preorder-equivalenceI'*:
assumes $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))\ l1\ r1$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lesssim\!\!\approx\ x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_{pre} (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $((x1\ x2 :: (\geq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rrightarrow (\leq))\ L2$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lesssim\!\!\approx\ x1') \Rrightarrow$
$\quad$ *in-field* $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $g \leq_R g$
shows $f\ {}_L\lesssim\!\!\approx\ g \longleftrightarrow ((x\ x' :: ({}_{L1}\lesssim\!\!\approx)) \Rrightarrow ({}_{L2\ x\ x'}\lesssim\!\!\approx))\ f\ g$
⟨*proof*⟩

Simplification of Restricted Function Relator **lemma** *Dep-Fun-Rel-left-Galois-restrict-left-right-eq*
assumes $((\leq_{L1}) \Rightarrow (\leq_{R1}))\ l1$ **and** $((\leq_{R1}) \Rightarrow (\leq_{L1}))\ r1$
and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))\ l1\ r1$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$\quad((\leq_{L2\ (r1\ x1')\ (r1\ x2')}) \ {}_h\trianglelefteq\ (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (r2_{(r1\ x1')\ x2'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'\ y.\ x1' \leq_{R1} x2' \Longrightarrow$ *in-dom* $(\leq_{L2\ (r1\ x1')\ (r1\ x2')})\ y \Longrightarrow$
$\quad(\leq_{R2\ x1'\ x2'})\ (l2_{x2'\ (r1\ x1')}\ y) \leq (\leq_{R2\ x1'\ x2'})\ (l2_{x1'\ (r1\ x1')}\ y)$
and $\bigwedge x1\ x2\ y'.\ x1 \leq_{L1} x2 \Longrightarrow$ *in-codom* $(\leq_{R2\ (l1\ x1)\ (l1\ x2)})\ y' \Longrightarrow$
$\quad(\geq_{L2\ x1\ x2})\ (r2_{x1\ (l1\ x2)}\ y') \leq (\geq_{L2\ x1\ x2})\ (r2_{x2\ (l1\ x2)}\ y')$
shows $((x\ x' :: ({}_{L1}\lesssim\!\!\approx)) \Rrightarrow ({}_{L2\ x\ x'}\lesssim\!\!\approx))\restriction_{in\text{-}dom\ (\leq_L)} \upharpoonright_{in\text{-}codom\ (\leq_R)}$
$\quad= ((x\ x' :: ({}_{L1}\lesssim\!\!\approx)) \Rrightarrow ({}_{L2\ x\ x'}\lesssim\!\!\approx))$
⟨*proof*⟩

lemma *Dep-Fun-Rel-left-Galois-restrict-left-right-eq-Dep-Fun-Rel-left-GaloisI'*:
assumes $((\leq_{L1}) \dashv (\leq_{R1}))\ l1\ r1$
and *reflexive-on* $(in\text{-}field\ (\leq_{L1}))\ (\leq_{L1})$
and *reflexive-on* $(in\text{-}field\ (\leq_{R1}))\ (\leq_{R1})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$\quad((\leq_{L2\ (r1\ x1')\ (r1\ x2')}) \ {}_h\trianglelefteq\ (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (r2_{(r1\ x1')\ x2'})$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid (x2 \leq_{L1} x3 \wedge x4 \leq_{L1} \eta_1\ x3)) \Rrightarrow (\geq))\ L2$
and $((x1'\ x2' :: (\leq_{R1}) \mid \varepsilon_1\ x2' \leq_{R1} x1') \Rightarrow (x3'\ x4' :: (\leq_{R1}) \mid x2' \leq_{R1} x3') \Rrightarrow (\leq))\ R2$
and $((x1'\ x2' :: (\leq_{R1})) \Rightarrow (x1\ x2 :: (\leq_{L1}) \mid x2\ {}_{L1}\lesssim\!\!\approx\ x1') \Rrightarrow$
$\quad$ *in-field* $(\leq_{L2\ x1\ (r1\ x2')}) \Rrightarrow (\leq_{R2\ (l1\ x1)\ x2'}))\ l2$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lesssim\!\!\approx\ x1') \Rrightarrow$
$\quad$ *in-field* $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$

and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$ *transitive* $(\leq_{R2\ x1'\ x2'})$
shows $((x\ x' :: ({}_{L1}\lessapprox)) \Rightarrow ({}_{L2\ x\ x'}\lessapprox))\restriction_{\textit{in-dom}\ (\leq_L)}\rceil_{\textit{in-codom}\ (\leq_R)}$
$= ((x\ x' :: ({}_{L1}\lessapprox)) \Rightarrow ({}_{L2\ x\ x'}\lessapprox))$
⟨*proof*⟩

Simplification of Restricted Function Relator for Nested Transports

lemma *Dep-Fun-Rel-left-Galois-restrict-left-right-restrict-left-right-eq*:
fixes $S :: 'a1 \Rightarrow 'a2 \Rightarrow 'b1 \Rightarrow 'b2 \Rightarrow bool$
assumes $((\leq_{L1})\ {}_h\trianglelefteq\ (\leq_{R1}))$ *l1 r1*
shows $((x\ x' :: ({}_{L1}\lessapprox)) \Rightarrow (S\ x\ x')\restriction_{\textit{in-dom}\ (\leq_{L2\ x\ (r1\ x')})}\rceil_{\textit{in-codom}\ (\leq_{R2\ (l1\ x)\ x'})}$
$\restriction_{\textit{in-dom}\ (\leq_L)}\rceil_{\textit{in-codom}\ (\leq_R)} =$
$((x\ x' :: ({}_{L1}\lessapprox)) \Rightarrow S\ x\ x')\restriction_{\textit{in-dom}\ (\leq_L)}\rceil_{\textit{in-codom}\ (\leq_R)}$ (**is** *?lhs = ?rhs*)
⟨*proof*⟩

end

Function Relator **context** *transport-Fun-Rel*
begin

corollary *Fun-Rel-left-Galois-if-left-GaloisI*:
assumes $((\leq_{L1})\ {}_h\trianglelefteq\ (\leq_{R1}))$ *l1 r1*
and *reflexive-on* $(\textit{in-dom}\ (\leq_{L1}))\ (\leq_{L1})$
and $((\leq_{R2}) \Rightarrow (\leq_{L2}))$ *r2*
and *transitive* $(\leq_{L2})$
and $g \leq_R g$
and $f\ {}_{L}\lessapprox\ g$
shows $(({}_{L1}\lessapprox) \Rrightarrow ({}_{L2}\lessapprox))\ f\ g$
⟨*proof*⟩

corollary *left-Galois-if-Fun-Rel-left-GaloisI*:
assumes $((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1*
and $((\leq_{L1})\ \trianglelefteq_h\ (\leq_{R1}))$ *l1 r1*
and *in-codom* $(\leq_R)\ g$
and $(({}_{L1}\lessapprox) \Rrightarrow ({}_{L2}\lessapprox))\ f\ g$
shows $f\ {}_{L}\lessapprox\ g$
⟨*proof*⟩

lemma *left-Galois-if-Fun-Rel-left-GaloisI'*:
assumes $((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1* **and** $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*
and $((\leq_{L1})\ \trianglelefteq_h\ (\leq_{R1}))$ *l1 r1*
and $((\leq_{L2})\ {}_h\trianglelefteq\ (\leq_{R2}))$ *l2 r2*
and $(({}_{L1}\lessapprox) \Rrightarrow ({}_{L2}\lessapprox))\ f\ g$
shows $f\ {}_{L}\lessapprox\ g$
⟨*proof*⟩

corollary *left-Galois-iff-Fun-Rel-left-GaloisI*:
assumes $((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1*

and $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-dom* $(\leq_{L1})$) $(\leq_{L1})$
and $((\leq_{R2}) \Rightarrow (\leq_{L2}))$ *r2*
and *transitive* $(\leq_{L2})$
and $g \leq_R g$
shows $f \ {}_L\lessapprox\ g \longleftrightarrow ((_{L1}\lessapprox) \Rrightarrow (_{L2}\lessapprox))\ f\ g$
⟨*proof*⟩

Simplification of Restricted Function Relator **lemma** *Fun-Rel-left-Galois-restrict-left-right-eq-Fun*
assumes $((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1* **and** $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*
and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*
and $((\leq_{L2}) \ {}_h\trianglelefteq (\leq_{R2}))$ *l2 r2*
shows $((_{L1}\lessapprox) \Rrightarrow (_{L2}\lessapprox))\restriction_{in\text{-}dom\ (\leq_L)}\restriction^{in\text{-}codom\ (\leq_R)} = ((_{L1}\lessapprox) \Rrightarrow (_{L2}\lessapprox))$
⟨*proof*⟩

Simplification of Restricted Function Relator for Nested Transports

lemma *Fun-Rel-left-Galois-restrict-left-right-restrict-left-right-eq*:
fixes $S :: 'b1 \Rightarrow 'b2 \Rightarrow bool$
assumes $((\leq_{L1}) \ {}_h\trianglelefteq (\leq_{R1}))$ *l1 r1*
shows $((_{L1}\lessapprox) \Rrightarrow S\restriction_{in\text{-}dom\ (\leq_{L2})}\restriction^{in\text{-}codom\ (\leq_{R2})})\restriction_{in\text{-}dom\ (\leq_L)}\restriction^{in\text{-}codom\ (\leq_R)}$
=
$((_{L1}\lessapprox) \Rrightarrow S)\restriction_{in\text{-}dom\ (\leq_L)}\restriction^{in\text{-}codom\ (\leq_R)}$
⟨*proof*⟩

end

Monotone Dependent Function Relator **context** *transport-Mono-Dep-Fun-Rel*
begin

lemma *Dep-Fun-Rel-left-Galois-if-left-GaloisI*:
assumes $((\leq_{L1}) \ {}_h\trianglelefteq (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-dom* $(\leq_{L1})$) $(\leq_{L1})$
and $\bigwedge x\ x'.\ x \ {}_{L1}\lessapprox\ x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rightarrow (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x\ x'\ y'.\ x \ {}_{L1}\lessapprox\ x' \Longrightarrow in\text{-}dom\ (\leq_{R2\ (l1\ x)\ x'})\ y' \Longrightarrow$
$(\geq_{L2\ x\ (r1\ x')})\ (r2_{x\ (l1\ x)}\ y') \leq (\geq_{L2\ x\ (r1\ x')})\ (r2_{x\ x'}\ y')$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow transitive\ (\leq_{L2\ x1\ x2})$
and $f \ {}_L\lessapprox\ g$
shows $((x\ x' :: (_{L1}\lessapprox)) \Rrightarrow (_{L2\ x\ x'}\lessapprox))\ f\ g$
⟨*proof*⟩

lemma *left-Galois-if-Dep-Fun-Rel-left-GaloisI*:
assumes $(tdfr.R \Rightarrow tdfr.L)\ r$
and $((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1*
and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2\ y'.\ x1 \leq_{L1} x2 \Longrightarrow in\text{-}codom\ (\leq_{R2\ (l1\ x1)\ (l1\ x2)})\ y' \Longrightarrow$
$(\geq_{L2\ x1\ x2})\ (r2_{x1\ (l1\ x2)}\ y') \leq (\geq_{L2\ x1\ x2})\ (r2_{x2\ (l1\ x2)}\ y')$

and *in-dom* $(\leq_L)$ f
and *in-codom* $(\leq_R)$ g
and $((x\ x' :: (_{L1}\lessapprox)) \Rrightarrow (_{L2\ x\ x'}\lessapprox))\ f\ g$
shows $f\ _L\lessapprox\ g$
⟨*proof*⟩

lemma *left-Galois-iff-Dep-Fun-Rel-left-GaloisI*:
assumes $(tdfr.R \Rightarrow tdfr.L)\ r$
and $((\leq_{L1}) \Rightarrow (\leq_{R1}))\ l1$
and $((\leq_{L1}) \trianglelefteq (\leq_{R1}))\ l1\ r1$
and *reflexive-on* $(in\text{-}field\ (\leq_{L1}))\ (\leq_{L1})$
and $\bigwedge x\ x'.\ x\ _{L1}\lessapprox\ x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rightarrow (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ _{L1}\lessapprox\ x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and *in-dom* $(\leq_L)\ f$
and *in-codom* $(\leq_R)\ g$
shows $f\ _L\lessapprox\ g \longleftrightarrow ((x\ x' :: (_{L1}\lessapprox)) \Rrightarrow (_{L2\ x\ x'}\lessapprox))\ f\ g$
⟨*proof*⟩

lemma *left-Galois-iff-Dep-Fun-Rel-left-Galois-if-mono-if-galois-connectionI*:
assumes *galois-conn1*: $((\leq_{L1}) \dashv (\leq_{R1}))\ l1\ r1$
and *refl-L1*: *reflexive-on* $(in\text{-}field\ (\leq_{L1}))\ (\leq_{L1})$
and $\bigwedge x\ x'.\ x\ _{L1}\lessapprox\ x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rightarrow (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
and *L2-le-unit2*: $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and *mono-r2*: $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ _{L1}\lessapprox\ x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and *trans-L2*: $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and *in-dom* $(\leq_L)\ f$
and *in-codom* $(\leq_R)\ g$
shows $f\ _L\lessapprox\ g \longleftrightarrow ((x\ x' :: (_{L1}\lessapprox)) \Rrightarrow (_{L2\ x\ x'}\lessapprox))\ f\ g$ (**is** *?lhs* $\longleftrightarrow$ *?rhs*)
⟨*proof*⟩

corollary *left-Galois-iff-Dep-Fun-Rel-left-Galois-if-mono-if-galois-connectionI'*:
assumes $((\leq_{L1}) \dashv (\leq_{R1}))\ l1\ r1$
and *reflexive-on* $(in\text{-}field\ (\leq_{L1}))\ (\leq_{L1})$
and $\bigwedge x\ x'.\ x\ _{L1}\lessapprox\ x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rightarrow (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $((x1 : \top) \Rightarrow (x2\ \text{-} :: (\leq_{L1}) \mid x1 \leq_{L1} x2) \Rightarrow (\leq))\ L2$
and $((x1 : \top) \Rightarrow (x2\ x3 :: (\leq_{L1}) \mid (x1 \leq_{L1} x2 \wedge x3 \leq_{L1} \eta_1\ x2)) \Rightarrow (\geq))\ L2$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ _{L1}\lessapprox\ x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and *in-dom* $(\leq_L)\ f$
and *in-codom* $(\leq_R)\ g$
shows $f\ _L\lessapprox\ g \longleftrightarrow ((x\ x' :: (_{L1}\lessapprox)) \Rrightarrow (_{L2\ x\ x'}\lessapprox))\ f\ g$ (**is** *?lhs* $\longleftrightarrow$ *?rhs*)

⟨*proof*⟩

corollary *left-Galois-eq-Dep-Fun-Rel-left-Galois-restrict-if-mono-if-galois-connectionI*:
assumes $((\leq_{L1}) \dashv (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rightarrow (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $((x1 : \top) \Rightarrow (x2 \text{ - :: } (\leq_{L1}) \mid x1 \leq_{L1} x2) \Rightarrow (\leq))\ L2$
and $((x1 : \top) \Rightarrow (x2\ x3 :: (\leq_{L1}) \mid (x1 \leq_{L1} x2 \wedge x3 \leq_{L1} \eta_1\ x2)) \Rightarrow (\geq))\ L2$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
shows $({}_{L}\lessapprox) = ((x\ x' :: ({}_{L1}\lessapprox)) \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\restriction_{\textit{in-dom}\ (\leq_L)}\restriction_{\textit{in-codom}\ (\leq_R)}$
⟨*proof*⟩

lemma *left-Galois-iff-Dep-Fun-Rel-left-Galois-if-galois-equivalenceI*:
assumes $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rightarrow (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $((x1\ x2 :: (\geq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rrightarrow (\leq))\ L2$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and *in-dom* $(\leq_L)\ f$
and *in-codom* $(\leq_R)\ g$
shows $f\ {}_{L}\lessapprox\ g \longleftrightarrow ((x\ x' :: ({}_{L1}\lessapprox)) \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
⟨*proof*⟩

theorem *left-Galois-eq-Dep-Fun-Rel-left-Galois-restrict-if-galois-equivalenceI*:
assumes $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rightarrow (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $((x1\ x2 :: (\geq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rrightarrow (\leq))\ L2$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
shows $({}_{L}\lessapprox) = ((x\ x' :: ({}_{L1}\lessapprox)) \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\restriction_{\textit{in-dom}\ (\leq_L)}\restriction_{\textit{in-codom}\ (\leq_R)}$
⟨*proof*⟩

corollary *left-Galois-iff-Dep-Fun-Rel-left-Galois-if-preorder-equivalenceI*:
assumes $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rightarrow (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $((x1\ x2 :: (\geq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rrightarrow (\leq))\ L2$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and *in-dom* $(\leq_L)\ f$
and *in-codom* $(\leq_R)\ g$

shows $f\ {}_{L}\lessapprox\ g \longleftrightarrow ((x\ x' :: (_{L1}\lessapprox)) \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
⟨*proof*⟩

corollary *left-Galois-eq-Dep-Fun-Rel-left-Galois-restrict-if-preorder-equivalenceI*:
assumes $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rightarrow (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $((x1\ x2 :: (\geq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rrightarrow (\leq))$ *L2*
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))$ *r2*
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
shows $({}_{L}\lessapprox) = ((x\ x' :: (_{L1}\lessapprox)) \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\restriction_{in\text{-}dom\ (\leq_L)}\restriction_{in\text{-}codom\ (\leq_R)}$
⟨*proof*⟩

corollary *left-Galois-iff-Dep-Fun-Rel-left-Galois-if-preorder-equivalenceI'*:
assumes $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_{pre} (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $((x1\ x2 :: (\geq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rrightarrow (\leq))$ *L2*
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))$ *r2*
and *in-dom* $(\leq_L)\ f$
and *in-codom* $(\leq_R)\ g$
shows $f\ {}_{L}\lessapprox\ g \longleftrightarrow ((x\ x' :: (_{L1}\lessapprox)) \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
⟨*proof*⟩

corollary *left-Galois-eq-Dep-Fun-Rel-left-Galois-restrict-if-preorder-equivalenceI'*:
assumes $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_{pre} (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $((x1\ x2 :: (\geq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rrightarrow (\leq))$ *L2*
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))$ *r2*
shows $({}_{L}\lessapprox) = ((x\ x' :: (_{L1}\lessapprox)) \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\restriction_{in\text{-}dom\ (\leq_L)}\restriction_{in\text{-}codom\ (\leq_R)}$
⟨*proof*⟩

Simplification of Restricted Function Relator **lemma** *Dep-Fun-Rel-left-Galois-restrict-left-right-eq*
assumes *reflexive-on* (*in-field tdfr.L*) *tdfr.L*
and *reflexive-on* (*in-field tdfr.R*) *tdfr.R*
and $((\leq_{L1}) \dashv (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and *reflexive-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$((\leq_{L2\ (r1\ x1')\ (r1\ x2')})\ {}_{h}\trianglelefteq\ (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (r2_{(r1\ x1')\ x2'})$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid (x2 \leq_{L1} x3 \wedge x4 \leq_{L1} \eta_1\ x3)) \Rrightarrow (\geq))$ *L2*
and $((x1'\ x2' :: (\leq_{R1}) \mid \varepsilon_1\ x2' \leq_{R1} x1') \Rightarrow (x3'\ x4' :: (\leq_{R1}) \mid x2' \leq_{R1} x3') \Rrightarrow (\leq))$ *R2*
and $((x1'\ x2' :: (\leq_{R1})) \Rightarrow (x1\ x2 :: (\leq_{L1}) \mid x2\ {}_{L1}\lessapprox\ x1') \Rrightarrow$
in-field $(\leq_{L2\ x1\ (r1\ x2')}) \Rrightarrow (\leq_{R2\ (l1\ x1)\ x2'}))$ *l2*

and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1') \Rrightarrow$
$\quad$ *in-field* $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$ *transitive* $(\leq_{R2\ x1'\ x2'})$
shows $((x\ x' :: ({}_{L1}\lessapprox)) \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\restriction_{\textit{in-dom}\ (\leq_L)}\rceil_{\textit{in-codom}\ (\leq_R)}$
$\quad = ((x\ x' :: ({}_{L1}\lessapprox)) \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))$
⟨*proof*⟩

interpretation *flip* : *transport-Dep-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2*
rewrites *flip.t1.unit* $\equiv \varepsilon_1$ ⟨*proof*⟩

lemma *Dep-Fun-Rel-left-Galois-restrict-left-right-eq-Dep-Fun-Rel-left-GaloisI*:
assumes $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$\quad((\leq_{L2\ (r1\ x1')\ (r1\ x2')})\ {}_h\trianglelefteq\ (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (r2_{(r1\ x1')\ x2'})$
and $((x1\ x2 :: (\geq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rrightarrow (\leq))\ L2$
and $((x1'\ x2' :: (\geq_{R1})) \Rightarrow (x3'\ x4' :: (\leq_{R1}) \mid x1' \leq_{R1} x3') \Rrightarrow (\leq))\ R2$
and $((x1'\ x2' :: (\leq_{R1})) \Rightarrow (x1\ x2 :: (\leq_{L1}) \mid x2\ {}_{L1}\lessapprox\ x1') \Rrightarrow$
$\quad$ *in-field* $(\leq_{L2\ x1\ (r1\ x2')}) \Rrightarrow (\leq_{R2\ (l1\ x1)\ x2'}))\ l2$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1') \Rrightarrow$
$\quad$ *in-field* $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and *PERS*: $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *partial-equivalence-rel* $(\leq_{L2\ x1\ x2})$
$\quad\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$ *partial-equivalence-rel* $(\leq_{R2\ x1'\ x2'})$
shows $((x\ x' :: ({}_{L1}\lessapprox)) \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\restriction_{\textit{in-dom}\ (\leq_L)}\rceil_{\textit{in-codom}\ (\leq_R)}$
$\quad = ((x\ x' :: ({}_{L1}\lessapprox)) \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))$
⟨*proof*⟩

Simplification of Restricted Function Relator for Nested Transports

lemma *Dep-Fun-Rel-left-Galois-restrict-left-right-restrict-left-right-eq*:
fixes $S :: 'a1 \Rightarrow 'a2 \Rightarrow 'b1 \Rightarrow 'b2 \Rightarrow bool$
assumes $((\leq_{L1})\ {}_h\trianglelefteq\ (\leq_{R1}))$ *l1 r1*
shows $((x\ x' :: ({}_{L1}\lessapprox)) \Rrightarrow (S\ x\ x')\restriction_{\textit{in-dom}\ (\leq_{L2\ x\ (r1\ x')})}\rceil_{\textit{in-codom}\ (\leq_{R2\ (l1\ x)\ x'})})$
$\quad\restriction_{\textit{in-dom}\ (\leq_L)}\rceil_{\textit{in-codom}\ (\leq_R)} =$
$\quad((x\ x' :: ({}_{L1}\lessapprox)) \Rrightarrow S\ x\ x')\restriction_{\textit{in-dom}\ (\leq_L)}\rceil_{\textit{in-codom}\ (\leq_R)}$
$\quad$(**is** *?lhs*$\restriction_{?DL}\rceil_{?CR} =$ *?rhs*$\restriction_{?DL}\rceil_{?CR}$)
⟨*proof*⟩

end

Monotone Function Relator **context** *transport-Mono-Fun-Rel* **begin**

corollary *Fun-Rel-left-Galois-if-left-GaloisI*:
assumes $((\leq_{L1})\ {}_h\trianglelefteq\ (\leq_{R1}))$ *l1 r1*
and *reflexive-on* $(\textit{in-dom}\ (\leq_{L1}))\ (\leq_{L1})$
and $((\leq_{R2}) \Rightarrow (\leq_{L2}))\ (r2)$
and *transitive* $(\leq_{L2})$

and $f\ {}_{L}\lessapprox g$
shows $((_{L1}\lessapprox) \Rrightarrow (_{L2}\lessapprox))\ f\ g$
⟨*proof*⟩

interpretation *flip* : *transport-Mono-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2* ⟨*proof*⟩

lemma *left-Galois-if-Fun-Rel-left-GaloisI*:
assumes $((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1*
and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*
and $((\leq_{R2}) \Rightarrow (\leq_{L2}))$ *r2*
and *in-dom* $(\leq_L)$ *f*
and *in-codom* $(\leq_R)$ *g*
and $((_{L1}\lessapprox) \Rrightarrow (_{L2}\lessapprox))\ f\ g$
shows $f\ {}_{L}\lessapprox g$
⟨*proof*⟩

corollary *left-Galois-iff-Fun-Rel-left-GaloisI*:
assumes $((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1*
and $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-dom* $(\leq_{L1})$) $(\leq_{L1})$
and $((\leq_{R2}) \Rightarrow (\leq_{L2}))$ *(r2)*
and *transitive* $(\leq_{L2})$
and *in-dom* $(\leq_L)$ *f*
and *in-codom* $(\leq_R)$ *g*
shows $f\ {}_{L}\lessapprox g \longleftrightarrow ((_{L1}\lessapprox) \Rrightarrow (_{L2}\lessapprox))\ f\ g$
⟨*proof*⟩

theorem *left-Galois-eq-Fun-Rel-left-Galois-restrictI*:
assumes $((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1*
and $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-dom* $(\leq_{L1})$) $(\leq_{L1})$
and $((\leq_{R2}) \Rightarrow (\leq_{L2}))$ *r2*
and *transitive* $(\leq_{L2})$
shows $(_{L}\lessapprox) = ((_{L1}\lessapprox) \Rrightarrow (_{L2}\lessapprox))\restriction_{\textit{in-dom}\ (\leq_L)}\restriction^{\textit{in-codom}\ (\leq_R)}$
⟨*proof*⟩

Simplification of Restricted Function Relator **lemma** *Fun-Rel-left-Galois-restrict-left-right-eq-Fun*
assumes *reflexive-on* (*in-field tfr.tdfr.L*) *tfr.tdfr.L*
and *reflexive-on* (*in-field tfr.tdfr.R*) *tfr.tdfr.R*
and $((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1* **and** $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*
and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*
and $((\leq_{L2})\ {}_h\trianglelefteq (\leq_{R2}))$ *l2 r2*
shows $((_{L1}\lessapprox) \Rrightarrow (_{L2}\lessapprox))\restriction_{\textit{in-dom}\ (\leq_L)}\restriction^{\textit{in-codom}\ (\leq_R)} = ((_{L1}\lessapprox) \Rrightarrow (_{L2}\lessapprox))$
⟨*proof*⟩

lemma *Fun-Rel-left-Galois-restrict-left-right-eq-Fun-Rel-left-GaloisI*:
assumes $((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1* **and** $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*
and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$

and *reflexive-on* (*in-field* ($\leq_{R1}$)) ($\leq_{R1}$)
and (($\leq_{L2}$) $\wr\trianglelefteq$ ($\leq_{R2}$)) *l2 r2*
and *partial-equivalence-rel* ($\leq_{L2}$)
and *partial-equivalence-rel* ($\leq_{R2}$)
shows (($_{L1}\lessapprox$) $\Rrightarrow$ ($_{L2}\lessapprox$))$\restriction_{in\text{-}dom\ (\leq_L)}{}^{\uparrow}{}_{in\text{-}codom\ (\leq_R)}$ = (($_{L1}\lessapprox$) $\Rrightarrow$ ($_{L2}\lessapprox$))
⟨*proof*⟩

Simplification of Restricted Function Relator for Nested Transports

lemma *Fun-Rel-left-Galois-restrict-left-right-restrict-left-right-eq*:
fixes S :: *'b1* ⇒ *'b2* ⇒ *bool*
assumes (($\leq_{L1}$) $\wr\trianglelefteq$ ($\leq_{R1}$)) *l1 r1*
shows (($_{L1}\lessapprox$) $\Rrightarrow$ $S\restriction_{in\text{-}dom\ (\leq_{L2})}{}^{\uparrow}{}_{in\text{-}codom\ (\leq_{R2})}\restriction_{in\text{-}dom\ (\leq_L)}{}^{\uparrow}{}_{in\text{-}codom\ (\leq_R)}$
=
(($_{L1}\lessapprox$) $\Rrightarrow$ S)$\restriction_{in\text{-}dom\ (\leq_L)}{}^{\uparrow}{}_{in\text{-}codom\ (\leq_R)}$
⟨*proof*⟩

end

end

2.15.9 Order Equivalence

theory *Transport-Functions-Order-Equivalence*
imports
Transport-Functions-Monotone
Transport-Functions-Galois-Equivalence
begin

Dependent Function Relator **context** *transport-Dep-Fun-Rel*
begin

Inflationary **lemma** *rel-unit-self-if-rel-selfI*:
assumes *inflationary-unit1*: *inflationary-on* (*in-codom* ($\leq_{L1}$)) ($\leq_{L1}$) η_1
and *refl-L1*: *reflexive-on* (*in-codom* ($\leq_{L1}$)) ($\leq_{L1}$)
and *trans-L1*: *transitive* ($\leq_{L1}$)
and *mono-l2*: $\bigwedge x.\ x \leq_{L1} x \Longrightarrow ((\leq_{L2\ x\ x}) \Rightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})$
and *mono-r2*: $\bigwedge x.\ x \leq_{L1} x \Longrightarrow ((\leq_{R2\ (l1\ x)\ (l1\ x)}) \Rightarrow (\leq_{L2\ x\ (\eta_1\ x)}))\ (r2_{x\ (l1\ x)})$
and *inflationary-unit2*: $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$
inflationary-on (*in-codom* ($\leq_{L2\ x\ x}$)) ($\leq_{L2\ x\ x}$) ($\eta_{2\ x\ (l1\ x)}$)
and *L2-le1*: $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and *L2-unit-le2*: $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and *ge-R2-l2-le2*: $\bigwedge x\ y.\ x \leq_{L1} x \Longrightarrow$ *in-codom* ($\leq_{L2\ x\ (\eta_1\ x)}$) $y \Longrightarrow$
($\geq_{R2\ (l1\ x)\ (l1\ x)}$) ($l2_{(l1\ x)\ x\ y}$) $\leq$ ($\geq_{R2\ (l1\ x)\ (l1\ x)}$) ($l2_{(l1\ x)\ (\eta_1\ x)\ y}$)
and *trans-L2*: $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* ($\leq_{L2\ x1\ x2}$)
and $f \leq_L f$
shows $f \leq_L \eta\ f$
⟨*proof*⟩

Deflationary interpretation *flip-inv* :
transport-Dep-Fun-Rel $(\geq_{R1})$ $(\geq_{L1})$ *r1 l1 flip2 R2 flip2 L2 r2 l2*
rewrites *flip-inv.L* $\equiv (\geq_R)$ **and** *flip-inv.R* $\equiv (\geq_L)$
and *flip-inv.unit* $\equiv \varepsilon$
and *flip-inv.t1.unit* $\equiv \varepsilon_1$
and $\bigwedge x\ y$. *flip-inv.t2-unit x y* $\equiv \varepsilon_{2\ y\ x}$
and $\bigwedge R\ x\ y$. $(\mathit{flip2}\ R\ x\ y)^{-1} \equiv R\ y\ x$
and $\bigwedge R$. *in-codom* $R^{-1} \equiv$ *in-dom* R
and $\bigwedge R\ x1\ x2$. *in-codom* $(\mathit{flip2}\ R\ x1\ x2) \equiv$ *in-dom* $(R\ x2\ x1)$
and $\bigwedge x1\ x2\ x1'\ x2'$. $(\mathit{flip2}\ R2\ x1'\ x2' \Rightarrow \mathit{flip2}\ L2\ x1\ x2) \equiv ((\leq_{R2\ x2'\ x1'}) \Rightarrow (\leq_{L2\ x2\ x1}))$
and $\bigwedge x1\ x2\ x1'\ x2'$. $(\mathit{flip2}\ L2\ x1\ x2 \Rightarrow \mathit{flip2}\ R2\ x1'\ x2') \equiv ((\leq_{L2\ x2\ x1}) \Rightarrow (\leq_{R2\ x2'\ x1'}))$
and $\bigwedge(R :: 'z \Rightarrow 'z \Rightarrow \mathit{bool})\ (P :: 'z \Rightarrow \mathit{bool})$.
$(\mathit{inflationary\text{-}on}\ P\ R^{-1} :: ('z \Rightarrow 'z) \Rightarrow \mathit{bool}) \equiv \mathit{deflationary\text{-}on}\ P\ R$
and $\bigwedge(P :: 'b2 \Rightarrow \mathit{bool})\ x$.
$(\mathit{inflationary\text{-}on}\ P\ (\mathit{flip2}\ R2\ x\ x) :: ('b2 \Rightarrow 'b2) \Rightarrow \mathit{bool}) \equiv \mathit{deflationary\text{-}on}\ P\ (\leq_{R2\ x\ x})$
and $\bigwedge x1\ x2\ x3\ x4$. $\mathit{flip2}\ R2\ x1\ x2 \leq \mathit{flip2}\ R2\ x3\ x4 \equiv (\leq_{R2\ x2\ x1}) \leq (\leq_{R2\ x4\ x3})$
and $\bigwedge(R :: 'z \Rightarrow 'z \Rightarrow \mathit{bool})\ (P :: 'z \Rightarrow \mathit{bool})$. $\mathit{reflexive\text{-}on}\ P\ R^{-1} \equiv \mathit{reflexive\text{-}on}\ P\ R$
and $\bigwedge(R :: 'z \Rightarrow 'z \Rightarrow \mathit{bool})$. $\mathit{transitive}\ R^{-1} \equiv \mathit{transitive}\ R$
and $\bigwedge x1'\ x2'$. $\mathit{transitive}\ (\mathit{flip2}\ R2\ x1'\ x2') \equiv \mathit{transitive}\ (\leq_{R2\ x2'\ x1'})$
⟨*proof*⟩

lemma *counit-rel-self-if-rel-selfI*:
assumes *deflationary-on* $(\mathit{in\text{-}dom}\ (\leq_{R1}))\ (\leq_{R1})\ \varepsilon_1$
and *reflexive-on* $(\mathit{in\text{-}dom}\ (\leq_{R1}))\ (\leq_{R1})$
and *transitive* $(\leq_{R1})$
and $\bigwedge x'$. $x' \leq_{R1} x' \Longrightarrow ((\leq_{L2\ (r1\ x')\ (r1\ x')}) \Rightarrow (\leq_{R2\ (\varepsilon_1\ x')\ x'}))\ (l2_{\ x'\ (r1\ x')})$
and $\bigwedge x'\ x'$. $x' \leq_{R1} x' \Longrightarrow ((\leq_{R2\ x'\ x'}) \Rightarrow (\leq_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x'$. $x' \leq_{R1} x' \Longrightarrow \mathit{deflationary\text{-}on}\ (\mathit{in\text{-}dom}\ (\leq_{R2\ x'\ x'}))\ (\leq_{R2\ x'\ x'})\ (\varepsilon_{2\ (r1\ x')\ x'})$
and $\bigwedge x1'\ x2'$. $x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'$. $x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x'\ y'$. $x' \leq_{R1} x' \Longrightarrow \mathit{in\text{-}dom}\ (\leq_{R2\ (\varepsilon_1\ x')\ x'})\ y' \Longrightarrow$
$(\leq_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ x'\ y'}) \leq (\leq_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')\ y'})$
and $\bigwedge x1'\ x2'$. $x1' \leq_{R1} x2' \Longrightarrow \mathit{transitive}\ (\leq_{R2\ x1'\ x2'})$
and $g \leq_R g$
shows $\varepsilon\ g \leq_R g$
⟨*proof*⟩

Relational Equivalence **lemma** *bi-related-unit-self-if-rel-self-aux*:
assumes *rel-equiv-unit1*: $\mathit{rel\text{-}equivalence\text{-}on}\ (\mathit{in\text{-}field}\ (\leq_{L1}))\ (\leq_{L1})\ \eta_1$
and *mono-r2*: $\bigwedge x$. $x \leq_{L1} x \Longrightarrow ((\leq_{R2\ (l1\ x)\ (l1\ x)}) \Rightarrow (\leq_{L2\ x\ x}))\ (r2_{x\ (l1\ x)})$
and *rel-equiv-unit2*: $\bigwedge x$. $x \leq_{L1} x \Longrightarrow$
$\mathit{rel\text{-}equivalence\text{-}on}\ (\mathit{in\text{-}field}\ (\leq_{L2\ x\ x}))\ (\leq_{L2\ x\ x})\ (\eta_{2\ x\ (l1\ x)})$

and *L2-le1*: $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and *L2-le2*: $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
and [*iff*]: $x \leq_{L1} x$
shows $((\leq_{R2\ (l1\ x)\ (l1\ x)}) \Rightarrow (\leq_{L2\ x\ (\eta_1\ x)}))\ (r2_{x\ (l1\ x)})$
and $((\leq_{R2\ (l1\ x)\ (l1\ x)}) \Rightarrow (\leq_{L2\ (\eta_1\ x)\ x}))\ (r2_{x\ (l1\ x)})$
and *deflationary-on* $(in\text{-}dom\ (\leq_{L2\ x\ x}))\ (\leq_{L2\ x\ x})\ \eta_{2\ x\ (l1\ x)}$
and *inflationary-on* $(in\text{-}codom\ (\leq_{L2\ x\ x}))\ (\leq_{L2\ x\ x})\ \eta_{2\ x\ (l1\ x)}$
⟨*proof*⟩

interpretation *flip* : *transport-Dep-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2*
rewrites $flip.counit \equiv \eta$ **and** $flip.t1.counit \equiv \eta_1$
and $\bigwedge x\ y.\ flip.t2\text{-}counit\ x\ y \equiv \eta_{2\ y\ x}$
⟨*proof*⟩

lemma *bi-related-unit-self-if-rel-selfI*:
assumes *rel-equiv-unit1*: *rel-equivalence-on* $(in\text{-}field\ (\leq_{L1}))\ (\leq_{L1})\ \eta_1$
and *trans-L1*: *transitive* $(\leq_{L1})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow ((\leq_{L2\ x\ x}) \Rightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow ((\leq_{R2\ (l1\ x)\ (l1\ x)}) \Rightarrow (\leq_{L2\ x\ x}))\ (r2_{x\ (l1\ x)})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$
rel-equivalence-on $(in\text{-}field\ (\leq_{L2\ x\ x}))\ (\leq_{L2\ x\ x})\ (\eta_{2\ x\ (l1\ x)})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ (\eta_1\ x1)\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x\ y.\ x \leq_{L1} x \Longrightarrow in\text{-}dom\ (\leq_{L2\ (\eta_1\ x)\ x})\ y \Longrightarrow$
$(\leq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ x\ y}) \leq (\leq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ (\eta_1\ x)\ y})$
and $\bigwedge x\ y.\ x \leq_{L1} x \Longrightarrow in\text{-}codom\ (\leq_{L2\ x\ (\eta_1\ x)})\ y \Longrightarrow$
$(\geq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ x\ y}) \leq (\geq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ (\eta_1\ x)\ y})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow transitive\ (\leq_{L2\ x1\ x2})$
and $f \leq_L f$
shows $f \equiv_L \eta\ f$
⟨*proof*⟩

Lemmas for Monotone Function Relator **lemma** *order-equivalence-if-order-equivalence-mono-assm*
assumes *order-equiv1*: $((\leq_{L1}) \equiv_o (\leq_{R1}))\ l1\ r1$
and *refl-R1*: *reflexive-on* $(in\text{-}field\ (\leq_{R1}))\ (\leq_{R1})$
and *R2-counit-le1*: $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and *mono-l2*: $((x1'\ x2' :: (\leq_{R1})) \Rightarrow (x1\ x2 :: (\leq_{L1}) \mid x2\ {}_{L1}\lessapprox\ x1') \Rrightarrow$
$in\text{-}field\ (\leq_{L2\ x1\ (r1\ x2')}) \Rrightarrow (\leq_{R2\ (l1\ x1)\ x2'}))\ l2$
and [*iff*]: $x1' \leq_{R1} x2'$
shows $((in\text{-}dom\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')})) \Rrightarrow (\leq_{R2\ x1'\ x2'}))\ (l2_{x1'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x1')})$
and $((in\text{-}codom\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')})) \Rrightarrow (\leq_{R2\ x1'\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x2')})$
⟨*proof*⟩

lemma *order-equivalence-if-order-equivalence-mono-assms-rightI*:

assumes *order-equiv1*: $((\leq_{L1}) \equiv_o (\leq_{R1}))$ *l1 r1*
and *refl-L1*: *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and *L2-unit-le2*: $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and *mono-r2*: $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and [*iff*]: $x1 \leq_{L1} x2$
shows $((\textit{in-codom}\ (\leq_{R2\ (l1\ x1)\ (l1\ x2)})) \Rrightarrow (\leq_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x2)})\ (r2_{x2\ (l1\ x2)})$
and $((\textit{in-dom}\ (\leq_{R2\ (l1\ x1)\ (l1\ x2)})) \Rrightarrow (\leq_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x1)})\ (r2_{x1\ (l1\ x2)})$
⟨*proof*⟩

lemma *l2-unit-bi-rel-selfI*:
assumes *pre-equiv1*: $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and *mono-L2*:
$((x1\ x2 :: (\leq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid (x2 \leq_{L1} x3 \wedge x4 \leq_{L1} \eta_1\ x3)) \Rrightarrow (\geq))$
L2
and *mono-R2*:
$((x1'\ x2' :: (\leq_{R1})) \Rightarrow (x3'\ x4' :: (\leq_{R1}) \mid (x2' \leq_{R1} x3' \wedge x4' \leq_{R1} \varepsilon_1\ x3')) \Rrightarrow$
$(\geq))\ R2$
and *mono-l2*: $((x1'\ x2' :: (\leq_{R1})) \Rightarrow (x1\ x2 :: (\leq_{L1}) \mid x2\ {}_{L1}\lessapprox x1') \Rrightarrow$
in-field $(\leq_{L2\ x1\ (r1\ x2')}) \Rrightarrow (\leq_{R2\ (l1\ x1)\ x2'}))\ l2$
and $x \leq_{L1} x$
and *in-field* $(\leq_{L2\ x\ x})\ y$
shows $l2_{(l1\ x)\ (\eta_1\ x)}\ y \equiv_{R2\ (l1\ x)\ (l1\ x)}\ l2_{(l1\ x)\ x}\ y$
⟨*proof*⟩

lemma *r2-counit-bi-rel-selfI*:
assumes *pre-equiv1*: $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and *mono-L2*:
$((x1\ x2 :: (\leq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid (x2 \leq_{L1} x3 \wedge x4 \leq_{L1} \eta_1\ x3)) \Rrightarrow (\geq))$
L2
and *mono-R2*:
$((x1'\ x2' :: (\leq_{R1})) \Rightarrow (x3'\ x4' :: (\leq_{R1}) \mid (x2' \leq_{R1} x3' \wedge x4' \leq_{R1} \varepsilon_1\ x3')) \Rrightarrow$
$(\geq))\ R2$
and *mono-r2*: $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $x' \leq_{R1} x'$
and *in-field* $(\leq_{R2\ x'\ x'})\ y'$
shows $r2_{(r1\ x')\ (\varepsilon_1\ x')}\ y' \equiv_{L2\ (r1\ x')\ (r1\ x')}\ r2_{(r1\ x')\ x'}\ y'$
⟨*proof*⟩

end

Function Relator **context** *transport-Fun-Rel*
begin

corollary *rel-unit-self-if-rel-selfI*:
assumes *inflationary-on* (*in-codom* $(\leq_{L1})$) $(\leq_{L1})\ \eta_1$
and *reflexive-on* (*in-codom* $(\leq_{L1})$) $(\leq_{L1})$

and *transitive* $(\leq_{L1})$
and $((\leq_{L2}) \Rightarrow (\leq_{R2}))$ *l2*
and $((\leq_{R2}) \Rightarrow (\leq_{L2}))$ *r2*
and *inflationary-on* (*in-codom* $(\leq_{L2})$) $(\leq_{L2})$ η_2
and *transitive* $(\leq_{L2})$
and $f \leq_L f$
shows $f \leq_L \eta\ f$
⟨*proof*⟩

corollary *counit-rel-self-if-rel-selfI*:
assumes *deflationary-on* (*in-dom* $(\leq_{R1})$) $(\leq_{R1})$ ε_1
and *reflexive-on* (*in-dom* $(\leq_{R1})$) $(\leq_{R1})$
and *transitive* $(\leq_{R1})$
and $((\leq_{L2}) \Rightarrow (\leq_{R2}))$ *l2*
and $((\leq_{R2}) \Rightarrow (\leq_{L2}))$ *r2*
and *deflationary-on* (*in-dom* $(\leq_{R2})$) $(\leq_{R2})$ ε_2
and *transitive* $(\leq_{R2})$
and $g \leq_R g$
shows $\varepsilon\ g \leq_R g$
⟨*proof*⟩

lemma *bi-related-unit-self-if-rel-selfI*:
assumes *rel-equivalence-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$ η_1
and *transitive* $(\leq_{L1})$
and $((\leq_{L2}) \Rightarrow (\leq_{R2}))$ *l2*
and $((\leq_{R2}) \Rightarrow (\leq_{L2}))$ *r2*
and *rel-equivalence-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$ η_2
and *transitive* $(\leq_{L2})$
and $f \leq_L f$
shows $f \equiv_L \eta\ f$
⟨*proof*⟩

end

Monotone Dependent Function Relator **context** *transport-Mono-Dep-Fun-Rel* **begin**

Inflationary **lemma** *inflationary-on-unitI*:
assumes $(tdfr.L \Rightarrow tdfr.R)\ l$ **and** $(tdfr.R \Rightarrow tdfr.L)\ r$
and *inflationary-on* (*in-codom* $(\leq_{L1})$) $(\leq_{L1})$ η_1
and *reflexive-on* (*in-codom* $(\leq_{L1})$) $(\leq_{L1})$
and *transitive* $(\leq_{L1})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow ((\leq_{L2}\ x\ x) \Rightarrow (\leq_{R2}\ (l1\ x)\ (l1\ x)))\ (l2_{(l1\ x)\ x})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow ((\leq_{R2}\ (l1\ x)\ (l1\ x)) \Rightarrow (\leq_{L2}\ x\ (\eta_1\ x)))\ (r2_{x\ (l1\ x)})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$ *inflationary-on* (*in-codom* $(\leq_{L2}\ x\ x)$) $(\leq_{L2}\ x\ x)$ $(\eta_{2\ x\ (l1\ x)})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2}\ x2\ x2) \leq (\leq_{L2}\ x1\ x2)$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2}\ x1\ (\eta_1\ x2)) \leq (\leq_{L2}\ x1\ x2)$
and $\bigwedge x\ y.\ x \leq_{L1} x \Longrightarrow$ *in-codom* $(\leq_{L2}\ x\ (\eta_1\ x))\ y \Longrightarrow$

$(\geq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ x\ y}) \leq (\geq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ (\eta_1\ x)\ y})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
shows *inflationary-on* (*in-field* $(\leq_L)$) $(\leq_L)\ \eta$
⟨*proof*⟩

Deflationary **lemma** *deflationary-on-counitI*:
assumes (*tdfr.L* ⇒ *tdfr.R*) *l* **and** (*tdfr.R* ⇒ *tdfr.L*) *r*
and *deflationary-on* (*in-dom* $(\leq_{R1})$) $(\leq_{R1})\ \varepsilon_1$
and *reflexive-on* (*in-dom* $(\leq_{R1})$) $(\leq_{R1})$
and *transitive* $(\leq_{R1})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow ((\leq_{L2\ (r1\ x')\ (r1\ x')}) \Rightarrow (\leq_{R2\ (\varepsilon_1\ x')\ x'}))\ (l2_{\ x'\ (r1\ x')})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$
$((\leq_{R2\ x'\ x'}) \Rightarrow (\leq_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$ *deflationary-on* (*in-dom* $(\leq_{R2\ x'\ x'})$) $(\leq_{R2\ x'\ x'})\ (\varepsilon_{2\ (r1\ x')\ x'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x'\ y'.\ x' \leq_{R1} x' \Longrightarrow$ *in-dom* $(\leq_{R2\ (\varepsilon_1\ x')\ x'})\ y' \Longrightarrow$
$(\leq_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ x'\ y'}) \leq (\leq_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')\ y'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$ *transitive* $(\leq_{R2\ x1'\ x2'})$
shows *deflationary-on* (*in-field* $(\leq_R)$) $(\leq_R)\ \varepsilon$
⟨*proof*⟩

Relational Equivalence **context**
begin

interpretation *flip* : *transport-Mono-Dep-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2*
rewrites *flip.counit* ≡ η **and** *flip.t1.counit* ≡ η_1
and $\bigwedge x\ y.$ *flip.t2-counit* $x\ y \equiv \eta_{2\ y\ x}$
⟨*proof*⟩

lemma *rel-equivalence-on-unitI*:
assumes (*tdfr.L* ⇒ *tdfr.R*) *l* **and** (*tdfr.R* ⇒ *tdfr.L*) *r*
and *rel-equiv-unit1*: *rel-equivalence-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})\ \eta_1$
and *trans-L1*: *transitive* $(\leq_{L1})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow ((\leq_{L2\ x\ x}) \Rightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow ((\leq_{R2\ (l1\ x)\ (l1\ x)}) \Rightarrow (\leq_{L2\ x\ x}))\ (r2_{x\ (l1\ x)})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$ *rel-equivalence-on* (*in-field* $(\leq_{L2\ x\ x})$) $(\leq_{L2\ x\ x})\ (\eta_{2\ x\ (l1\ x)})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ (\eta_1\ x1)\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x\ y.\ x \leq_{L1} x \Longrightarrow$ *in-dom* $(\leq_{L2\ (\eta_1\ x)\ x})\ y \Longrightarrow$
$(\leq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ x\ y}) \leq (\leq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ (\eta_1\ x)\ y})$
and $\bigwedge x\ y.\ x \leq_{L1} x \Longrightarrow$ *in-codom* $(\leq_{L2\ x\ (\eta_1\ x)})\ y \Longrightarrow$
$(\geq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ x\ y}) \leq (\geq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ (\eta_1\ x)\ y})$

and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
shows *rel-equivalence-on* (*in-field* $(\leq_L)$) $(\leq_L)$ η
⟨*proof*⟩

end

Order Equivalence **interpretation** *flip* : *transport-Mono-Dep-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2*
rewrites *flip.unit* $\equiv \varepsilon$ **and** *flip.t1.unit* $\equiv \varepsilon_1$
and *flip.counit* $\equiv \eta$ **and** *flip.t1.counit* $\equiv \eta_1$
and $\bigwedge x\ y.$ *flip.t2-unit* $x\ y \equiv \varepsilon_{2\ y\ x}$
⟨*proof*⟩

lemma *order-equivalenceI*:
assumes (*tdfr.L* ⇒ *tdfr.R*) *l* **and** (*tdfr.R* ⇒ *tdfr.L*) *r*
and *rel-equivalence-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$ η_1
and *rel-equivalence-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$ ε_1
and *transitive* $(\leq_{L1})$ **and** *transitive* $(\leq_{R1})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow ((\leq_{L2\ x\ x}) \Rightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow ((\leq_{L2\ (r1\ x')\ (r1\ x')}) \Rightarrow (\leq_{R2\ x'\ x'}))\ (l2_{x'\ (r1\ x')})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow ((\leq_{R2\ x'\ x'}) \Rightarrow (\leq_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow ((\leq_{R2\ (l1\ x)\ (l1\ x)}) \Rightarrow (\leq_{L2\ x\ x}))\ (r2_{x\ (l1\ x)})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$ *rel-equivalence-on* (*in-field* $(\leq_{L2\ x\ x})$) $(\leq_{L2\ x\ x})$ $(\eta_{2\ x\ (l1\ x)})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$
rel-equivalence-on (*in-field* $(\leq_{R2\ x'\ x'})$) $(\leq_{R2\ x'\ x'})$ $(\varepsilon_{2\ (r1\ x')\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ (\eta_1\ x1)\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x2'\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ (\varepsilon_1\ x2')}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x\ y.\ x \leq_{L1} x \Longrightarrow$ *in-dom* $(\leq_{L2\ (\eta_1\ x)\ x})\ y \Longrightarrow$
$(\leq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ x\ y}) \leq (\leq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ (\eta_1\ x)\ y})$
and $\bigwedge x\ y.\ x \leq_{L1} x \Longrightarrow$ *in-codom* $(\leq_{L2\ x\ (\eta_1\ x)})\ y \Longrightarrow$
$(\geq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ x\ y}) \leq (\geq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ (\eta_1\ x)\ y})$
and $\bigwedge x'\ y'.\ x' \leq_{R1} x' \Longrightarrow$ *in-dom* $(\leq_{R2\ (\varepsilon_1\ x')\ x'})\ y' \Longrightarrow$
$(\leq_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ x'\ y'}) \leq (\leq_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')\ y'})$
and $\bigwedge x'\ y'.\ x' \leq_{R1} x' \Longrightarrow$ *in-codom* $(\leq_{R2\ x'\ (\varepsilon_1\ x')})\ y' \Longrightarrow$
$(\geq_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ x'\ y'}) \leq (\geq_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')\ y'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{R1} x2 \Longrightarrow$ *transitive* $(\leq_{R2\ x1\ x2})$
shows $((\leq_L) \equiv_o (\leq_R))\ l\ r$

⟨*proof*⟩

lemma *order-equivalence-if-preorder-equivalenceI*:
assumes *pre-equiv1*: $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and *order-equiv2*: $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow$
$((\leq_{L2\ x\ (r1\ x')}) \equiv_o (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and *L2-les*: $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
$\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ (\eta_1\ x1)\ x2}) \leq (\leq_{L2\ x1\ x2})$
$\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
$\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and *R2-les*: $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x2'\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
$\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
$\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$
$\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ (\varepsilon_1\ x2')}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$((in\text{-}dom\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')}))) \Rrightarrow (\leq_{R2\ x1'\ x2'}))\ (l2_{x1'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x1')})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$((in\text{-}codom\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')}))) \Rrightarrow (\leq_{R2\ x1'\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x2')})$
and *l2-bi-rel*: $\bigwedge x\ y.\ x \leq_{L1} x \Longrightarrow in\text{-}field\ (\leq_{L2\ x\ x})\ y \Longrightarrow$
$l2_{(l1\ x)\ (\eta_1\ x)}\ y \equiv_{R2\ (l1\ x)\ (l1\ x)}\ l2_{(l1\ x)\ x}\ y$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$
$((in\text{-}codom\ (\leq_{R2\ (l1\ x1)\ (l1\ x2)}))) \Rrightarrow (\leq_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x2)})\ (r2_{x2\ (l1\ x2)})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$
$((in\text{-}dom\ (\leq_{R2\ (l1\ x1)\ (l1\ x2)}))) \Rrightarrow (\leq_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x1)})\ (r2_{x1\ (l1\ x2)})$
and *r2-bi-rel*: $\bigwedge x'\ y'.\ x' \leq_{R1} x' \Longrightarrow in\text{-}field\ (\leq_{R2\ x'\ x'})\ y' \Longrightarrow$
$r2_{(r1\ x')\ (\varepsilon_1\ x')}\ y' \equiv_{L2\ (r1\ x')\ (r1\ x')}\ r2_{(r1\ x')\ x'}\ y'$
and *trans-L2*: $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow transitive\ (\leq_{L2\ x1\ x2})$
and *trans-R2*: $\bigwedge x1\ x2.\ x1 \leq_{R1} x2 \Longrightarrow transitive\ (\leq_{R2\ x1\ x2})$
shows $((\leq_L) \equiv_o (\leq_R))\ l\ r$
⟨*proof*⟩

lemma *order-equivalence-if-preorder-equivalenceI'*:
assumes $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_o (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ (\eta_1\ x1)\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x2'\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ (\varepsilon_1\ x2')}) \leq (\leq_{R2\ x1'\ x2'})$
and $((x1'\ x2' :: (\leq_{R1})) \Rightarrow (x1\ x2 :: (\leq_{L1}) \mid x2\ {}_{L1}\lessapprox x1') \Rrightarrow$
$in\text{-}field\ (\leq_{L2\ x1\ (r1\ x2')}) \Rrightarrow (\leq_{R2\ (l1\ x1)\ x2'}))\ l2$
and $\bigwedge x\ y.\ x \leq_{L1} x \Longrightarrow in\text{-}field\ (\leq_{L2\ x\ x})\ y \Longrightarrow$
$l2_{(l1\ x)\ (\eta_1\ x)}\ y \equiv_{R2\ (l1\ x)\ (l1\ x)}\ l2_{(l1\ x)\ x}\ y$

and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x'\ y'.\ x' \leq_{R1} x' \Longrightarrow$ *in-field* $(\leq_{R2\ x'\ x'})\ y' \Longrightarrow$
$r2_{(r1\ x')\ (\varepsilon_1\ x')}\ y' \equiv_{L2\ (r1\ x')\ (r1\ x')}\ r2_{(r1\ x')\ x'}\ y'$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{R1} x2 \Longrightarrow$ *transitive* $(\leq_{R2\ x1\ x2})$
shows $((\leq_L) \equiv_o (\leq_R))\ l\ r$
⟨*proof*⟩

lemma *order-equivalence-if-mono-if-preorder-equivalenceI*:
assumes $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))\ l1\ r1$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_o (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $((x1\ x2 :: (\leq_{L1}) \mid \eta_1\ x2 \leq_{L1} x1) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid x2 \leq_{L1} x3) \Rrightarrow (\leq))$
L2
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid (x2 \leq_{L1} x3 \wedge x4 \leq_{L1} \eta_1\ x3)) \Rrightarrow$
$(\geq))\ L2$
and $((x1'\ x2' :: (\leq_{R1}) \mid \varepsilon_1\ x2' \leq_{R1} x1') \Rightarrow (x3'\ x4' :: (\leq_{R1}) \mid x2' \leq_{R1} x3') \Rrightarrow$
$(\leq))\ R2$
and $((x1'\ x2' :: (\leq_{R1})) \Rightarrow (x3'\ x4' :: (\leq_{R1}) \mid (x2' \leq_{R1} x3' \wedge x4' \leq_{R1} \varepsilon_1\ x3'))$
$\Rrightarrow (\geq))\ R2$
and $((x1'\ x2' :: (\leq_{R1})) \Rightarrow (x1\ x2 :: (\leq_{L1}) \mid x2\ {}_{L1}\lessapprox\ x1') \Rrightarrow$
in-field $(\leq_{L2\ x1\ (r1\ x2')}) \Rrightarrow (\leq_{R2\ (l1\ x1)\ x2'}))\ l2$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{R1} x2 \Longrightarrow$ *transitive* $(\leq_{R2\ x1\ x2})$
shows $((\leq_L) \equiv_o (\leq_R))\ l\ r$
⟨*proof*⟩

theorem *order-equivalence-if-mono-if-preorder-equivalenceI'*:
assumes $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))\ l1\ r1$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_{pre} (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $((x1\ x2 :: (\geq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rrightarrow (\leq))\ L2$
and $((x1'\ x2' :: (\geq_{R1})) \Rightarrow (x3'\ x4' :: (\leq_{R1}) \mid x1' \leq_{R1} x3') \Rrightarrow (\leq))\ R2$
and $((x1'\ x2' :: (\leq_{R1})) \Rightarrow (x1\ x2 :: (\leq_{L1}) \mid x2\ {}_{L1}\lessapprox\ x1') \Rrightarrow$
in-field $(\leq_{L2\ x1\ (r1\ x2')}) \Rrightarrow (\leq_{R2\ (l1\ x1)\ x2'}))\ l2$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
shows $((\leq_L) \equiv_o (\leq_R))\ l\ r$
⟨*proof*⟩

end

Monotone Function Relator **context** *transport-Mono-Fun-Rel* **begin**

interpretation *flip* : *transport-Mono-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2* ⟨*proof*⟩

lemma *inflationary-on-unitI*:
assumes $((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1*
and $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*
and *inflationary-on* (*in-codom* $(\leq_{L1})$) $(\leq_{L1})$ η_1
and *reflexive-on* (*in-codom* $(\leq_{L1})$) $(\leq_{L1})$
and *transitive* $(\leq_{L1})$
and $((\leq_{L2}) \Rightarrow (\leq_{R2}))$ *l2*
and $((\leq_{R2}) \Rightarrow (\leq_{L2}))$ *r2*
and *inflationary-on* (*in-codom* $(\leq_{L2})$) $(\leq_{L2})$ η_2
and *transitive* $(\leq_{L2})$
shows *inflationary-on* (*in-field* $(\leq_L)$) $(\leq_L)$ η
⟨*proof*⟩

lemma *deflationary-on-counitI*:
assumes $((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1*
and $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*
and *deflationary-on* (*in-dom* $(\leq_{R1})$) $(\leq_{R1})$ ε_1
and *reflexive-on* (*in-dom* $(\leq_{R1})$) $(\leq_{R1})$
and *transitive* $(\leq_{R1})$
and $((\leq_{L2}) \Rightarrow (\leq_{R2}))$ *l2*
and $((\leq_{R2}) \Rightarrow (\leq_{L2}))$ *r2*
and *deflationary-on* (*in-dom* $(\leq_{R2})$) $(\leq_{R2})$ ε_2
and *transitive* $(\leq_{R2})$
shows *deflationary-on* (*in-field* $(\leq_R)$) $(\leq_R)$ ε
⟨*proof*⟩

lemma *rel-equivalence-on-unitI*:
assumes $((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1*
and $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*
and *rel-equivalence-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$ η_1
and *transitive* $(\leq_{L1})$
and $((\leq_{L2}) \Rightarrow (\leq_{R2}))$ *l2*
and $((\leq_{R2}) \Rightarrow (\leq_{L2}))$ *r2*
and *rel-equivalence-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$ η_2
and *transitive* $(\leq_{L2})$
shows *rel-equivalence-on* (*in-field* $(\leq_L)$) $(\leq_L)$ η
⟨*proof*⟩

lemma *order-equivalenceI*:
assumes $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and $((\leq_{L2}) \equiv_{pre} (\leq_{R2}))$ *l2 r2*
shows $((\leq_L) \equiv_o (\leq_R))$ *l r*
⟨*proof*⟩

end

end

theory *Transport-Functions*
imports
Transport-Functions-Galois-Equivalence
Transport-Functions-Galois-Relator
Transport-Functions-Order-Base
Transport-Functions-Order-Equivalence
Transport-Functions-Relation-Simplifications
begin

Summary Composition under (dependent) (monotone) function relators. Refer to [2] for more details.

2.15.10 Summary of Main Results

More precise results can be found in the corresponding subtheories.

Monotone Dependent Function Relator **context** *transport-Mono-Dep-Fun-Rel*
begin

interpretation *flip* : *transport-Mono-Dep-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2*
rewrites *flip.t1.counit* ≡ η_1 **and** *flip.t1.unit* ≡ ε_1
⟨*proof*⟩

Closure of Order and Galois Concepts **theorem** *preorder-galois-connection-if-galois-connectionI*:
assumes $((\leq_{L1}) \dashv (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and *reflexive-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and $\bigwedge x\ x'.\ x\ _{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \dashv (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $((\text{-}\ x2 :: (\leq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid (x2 \leq_{L1} x3 \wedge x4 \leq_{L1} \eta_1\ x3)) \Rrightarrow (\geq))$ *L2*
and $((x1'\ x2' :: (\leq_{R1}) \mid \varepsilon_1\ x2' \leq_{R1} x1') \Rightarrow (x3'\ \text{-} :: (\leq_{R1}) \mid x2' \leq_{R1} x3') \Rrightarrow (\leq))$ *R2*
and $((x1'\ x2' :: (\leq_{R1})) \Rightarrow (x1\ x2 :: (\leq_{L1}) \mid x2\ _{L1}\lessapprox x1') \Rrightarrow$
in-field $(\leq_{L2\ x1\ (r1\ x2')}) \Rrightarrow (\leq_{R2\ (l1\ x1)\ x2'}))$ *l2*
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1}) \mid x2\ _{L1}\lessapprox x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))$ *r2*
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$ *transitive* $(\leq_{R2\ x1'\ x2'})$
shows $((\leq_L) \dashv_{pre} (\leq_R))\ l\ r$
⟨*proof*⟩

theorem *preorder-equivalenceI*:
assumes $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and $\bigwedge x\ x'.\ x\ _{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_{pre} (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $((x1\ \text{-} :: (\geq_{L1})) \Rightarrow (x3\ \text{-} :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rrightarrow (\leq))$ *L2*
and $((x1'\ \text{-} :: (\geq_{R1})) \Rightarrow (x3'\ \text{-} :: (\leq_{R1}) \mid x1' \leq_{R1} x3') \Rrightarrow (\leq))$ *R2*

and $((x1\,'\ x2\,' :: (\leq_{R1})) \Rightarrow (x1\ x2 :: (\leq_{L1}) \mid x2\ {}_{L1}\lessapprox\ x1\,') \Rrightarrow$
in-field $(\leq_{L2\ x1\ (r1\ x2\,')}) \Rrightarrow (\leq_{R2\ (l1\ x1)\ x2\,'}))\ l2$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1\,'\ x2\,' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1\,') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2\,'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2\,')}))\ r2$
shows $((\leq_L) \equiv_{pre} (\leq_R))\ l\ r$
⟨*proof*⟩

theorem *partial-equivalence-rel-equivalenceI*:
assumes $((\leq_{L1}) \equiv_{PER} (\leq_{R1}))\ l1\ r1$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_{PER} (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $((x1\ \text{-} :: (\geq_{L1})) \Rightarrow (x3\ \text{-} :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rrightarrow (\leq))\ L2$
and $((x1\,'\ \text{-} :: (\geq_{R1})) \Rightarrow (x3\,'\ \text{-} :: (\leq_{R1}) \mid x1\,' \leq_{R1} x3\,') \Rrightarrow (\leq))\ R2$
and $((x1\,'\ x2\,' :: (\leq_{R1})) \Rightarrow (x1\ x2 :: (\leq_{L1}) \mid x2\ {}_{L1}\lessapprox\ x1\,') \Rrightarrow$
in-field $(\leq_{L2\ x1\ (r1\ x2\,')}) \Rrightarrow (\leq_{R2\ (l1\ x1)\ x2\,'}))\ l2$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1\,'\ x2\,' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1\,') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2\,'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2\,')}))\ r2$
shows $((\leq_L) \equiv_{PER} (\leq_R))\ l\ r$
⟨*proof*⟩

Simplification of Left and Right Relations See ⟦*t1.galois-equivalence*; *preorder-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$; $((x1\ x2 :: (\leq_{L1})^{-1}) \Rightarrow (x3\ x4 :: (\leq_{L1})) \Rrightarrow x1 \leq_{L1} x3 \longrightarrow (\leq))\ L2$; $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *partial-equivalence-rel* $(\leq_{L2\ x1\ x2})$⟧ $\Longrightarrow$ *flip.R* = *flip.tdfr.R*.

Simplification of Galois relator See ⟦*t1.galois-connection*; *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$; $\bigwedge x\ x'.$ *flip.t1.right-Galois x x'* $\Longrightarrow ((\leq_{R2\ l1\ x\ x'}) \Rightarrow \leq_{L2\ x\ r1\ x'})\ r2_{x\ x'}$; $((x1 : \top) \Rightarrow (x2\ \text{-} :: (\leq_{L1})) \Rightarrow x1 \leq_{L1} x2 \longrightarrow (\leq))\ L2$; $((x1 : \top) \Rightarrow (x2\ x3 :: (\leq_{L1})) \Rightarrow (x1 \leq_{L1} x2 \wedge x3 \leq_{L1} \eta_1\ x2) \longrightarrow (\lambda x\ y.\ y \leq x))\ L2$; $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1\,'\ x2\,' :: (\leq_{R1})) \Rrightarrow$ *flip.t1.right-Galois x2 x1'* $\longrightarrow$ (*in-field* $(\leq_{R2\ l1\ x1\ x2\,'}) \Rrightarrow \leq_{L2\ x1\ r1\ x2\,'}))\ r2$; $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$⟧ $\Longrightarrow$ *flip.right-Galois* = (*Dep-Fun-Rel flip.t1.right-Galois t2.left-Galois*)$\restriction_{in\text{-}dom\ flip.R}\restriction_{in\text{-}codom\ flip.L}$

⟦*t1.preorder-equivalence*; $\bigwedge x\ x'.$ *flip.t1.right-Galois x x'* $\Longrightarrow ((\leq_{R2\ l1\ x\ x'}) \Rightarrow \leq_{L2\ x\ r1\ x'})\ r2_{x\ x'}$; $((x1\ x2 :: (\leq_{L1})^{-1}) \Rightarrow (x3\ x4 :: (\leq_{L1})) \Rrightarrow x1 \leq_{L1} x3 \longrightarrow (\leq))\ L2$; $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1\,'\ x2\,' :: (\leq_{R1})) \Rrightarrow$ *flip.t1.right-Galois x2 x1'* $\longrightarrow$ (*in-field* $(\leq_{R2\ l1\ x1\ x2\,'}) \Rrightarrow \leq_{L2\ x1\ r1\ x2\,'}))\ r2$; $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$⟧ $\Longrightarrow$ *flip.right-Galois* = (*Dep-Fun-Rel flip.t1.right-Galois t2.left-Galois*)$\restriction_{in\text{-}dom\ flip.R}\restriction_{in\text{-}codom\ flip.L}$

⟦*t1.preorder-equivalence*; $\bigwedge x\ x'.$ *flip.t1.right-Galois x x'* $\Longrightarrow$ *t2.preorder-equivalence x x'*; $((x1\ x2 :: (\leq_{L1})^{-1}) \Rightarrow (x3\ x4 :: (\leq_{L1})) \Rrightarrow x1 \leq_{L1} x3 \longrightarrow (\leq))\ L2$; $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1\,'\ x2\,' :: (\leq_{R1})) \Rrightarrow$ *flip.t1.right-Galois x2 x1'* $\longrightarrow$ (*in-field* $(\leq_{R2\ l1\ x1\ x2\,'}) \Rrightarrow \leq_{L2\ x1\ r1\ x2\,'}))\ r2$⟧ $\Longrightarrow$ *flip.right-Galois* = (*Dep-Fun-Rel flip.t1.right-Galois t2.left-Galois*)$\restriction_{in\text{-}dom\ flip.R}\restriction_{in\text{-}codom\ flip.L}$

⟦*t1.preorder-equivalence*; $\bigwedge x1\,'\ x2\,'.\ x1\,' \leq_{R1} x2\,' \Longrightarrow ((\leq_{L2\ r1\ x1\,'\ r1\ x2\,'})$

$_h\trianglelefteq\ (\leq_{R2\ \varepsilon_1\ x1'\ x2'}))\ l2_{x2'\ r1\ x1'\ r2_{r1\ x1'\ x2'}};\ ((x1\ x2 :: (\leq_{L1})^{-1}) \Rightarrow (x3\ x4$
$:: (\leq_{L1})) \Rrightarrow x1 \leq_{L1} x3 \longrightarrow (\leq))\ L2;\ ((x1'\ x2' :: (\leq_{R1})^{-1}) \Rightarrow (x3'\ x4' ::$
$(\leq_{R1})) \Rrightarrow x1' \leq_{R1} x3' \longrightarrow (\leq))\ R2;\ ((x1'\ x2' :: (\leq_{R1})) \Rightarrow (x1\ x2 :: (\leq_{L1}))$
$\Rrightarrow$ *flip.t1.right-Galois x2 x1'* $\longrightarrow$ (*in-field* $(\leq_{L2\ x1\ r1\ x2'}) \Rrightarrow \leq_{R2\ l1\ x1\ x2'}))$
$l2;\ ((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1'\ x2' :: (\leq_{R1})) \Rrightarrow$ *flip.t1.right-Galois x2 x1'* $\longrightarrow$
(*in-field* $(\leq_{R2\ l1\ x1\ x2'}) \Rrightarrow \leq_{L2\ x1\ r1\ x2'}))\ r2;\ \bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *par-*
tial-equivalence-rel $(\leq_{L2\ x1\ x2});\ \bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$ *partial-equivalence-rel*
$(\leq_{R2\ x1'\ x2'})]\!] \Longrightarrow$ (*Dep-Fun-Rel flip.t1.right-Galois t2.left-Galois*)$\restriction_{\textit{in-dom flip.R}}\restriction_{\textit{in-codom flip.L}}$
$=$ *Dep-Fun-Rel flip.t1.right-Galois t2.left-Galois*
t1.half-galois-prop-left $\Longrightarrow ((x\ x' ::$ *flip.t1.right-Galois*$) \Rrightarrow (?S\ x\ x')\restriction_{\textit{in-dom}\ (\leq_{L2\ x\ r1\ x'})}\restriction_{\textit{in-codom}\ (\leq}$
$=$ (*Dep-Fun-Rel flip.t1.right-Galois ?S*)$\restriction_{\textit{in-dom flip.R}}\restriction_{\textit{in-codom flip.L}}$

end

Monotone Function Relator **context** *transport-Mono-Fun-Rel*
begin

interpretation *flip* : *transport-Mono-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2* ⟨*proof*⟩

Closure of Order and Galois Concepts **lemma** *preorder-galois-connection-if-galois-connectionI*:
assumes $((\leq_{L1}) \dashv (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-codom* $(\leq_{L1})$) $(\leq_{L1})$ *reflexive-on* (*in-dom* $(\leq_{R1})$) $(\leq_{R1})$
and $((\leq_{L2}) \dashv (\leq_{R2}))$ *l2 r2*
and *transitive* $(\leq_{L2})$ *transitive* $(\leq_{R2})$
shows $((\leq_L) \dashv_{pre} (\leq_R))$ *l r*
⟨*proof*⟩

theorem *preorder-galois-connectionI*:
assumes $((\leq_{L1}) \dashv_{pre} (\leq_{R1}))$ *l1 r1*
and $((\leq_{L2}) \dashv_{pre} (\leq_{R2}))$ *l2 r2*
shows $((\leq_L) \dashv_{pre} (\leq_R))$ *l r*
⟨*proof*⟩

theorem *preorder-equivalence-if-galois-equivalenceI*:
assumes $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$ *reflexive-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and $((\leq_{L2}) \equiv_G (\leq_{R2}))$ *l2 r2*
and *transitive* $(\leq_{L2})$ *transitive* $(\leq_{R2})$
shows $((\leq_L) \equiv_{pre} (\leq_R))$ *l r*
⟨*proof*⟩

theorem *preorder-equivalenceI*:
assumes $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and $((\leq_{L2}) \equiv_{pre} (\leq_{R2}))$ *l2 r2*
shows $((\leq_L) \equiv_{pre} (\leq_R))$ *l r*
⟨*proof*⟩

theorem *partial-equivalence-rel-equivalenceI*:

assumes $((\leq_{L1}) \equiv_{PER} (\leq_{R1}))$ *l1 r1*
and $((\leq_{L2}) \equiv_{PER} (\leq_{R2}))$ *l2 r2*
shows $((\leq_L) \equiv_{PER} (\leq_R))$ *l r*
⟨*proof*⟩

Simplification of Left and Right Relations See ⟦*reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$; *partial-equivalence-rel* $(\leq_{L2})$⟧ ⟹ *flip.tpdfr.R* = *flip.tfr.tdfr.R*.

Simplification of Galois relator See ⟦$((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1*; *tdfrs.t1.galois-prop l1 r1*; *reflexive-on* (*in-dom* $(\leq_{L1})$) $(\leq_{L1})$; $((\leq_{R2}) \Rightarrow (\leq_{L2}))$ *r2*; *transitive* $(\leq_{L2})$⟧ ⟹ *flip.tpdfr.right-Galois* = (*flip.tdfrs.t1.right-Galois* ⇛ *flip.tdfrs.t2.right-Galois*)↾*in-dom flip.*
⟦$((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1*; $((\leq_{R1}) \Rightarrow (\leq_{L1}))$ *r1*; *tdfrs.t1.half-galois-prop-right*; *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$; *reflexive-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$; *tdfrs.t2.half-galois-prop-left*; *partial-equivalence-rel* $(\leq_{L2})$; *partial-equivalence-rel* $(\leq_{R2})$⟧ ⟹ (*flip.tdfrs.t1.right-Galois* ⇛ *flip.tdfrs.t2.right-Galois*)↾*in-dom flip.tpdfr.R*⤵*in-codom flip.tpdfr.L*
= (*flip.tdfrs.t1.right-Galois* ⇛ *flip.tdfrs.t2.right-Galois*)
tdfrs.t1.half-galois-prop-left ⟹ (*flip.tdfrs.t1.right-Galois* ⇛ *?S*↾*in-dom* $(\leq_{L2})$⤵*in-codom* $(\leq_{R2})$)↾*in-d*
= (*flip.tdfrs.t1.right-Galois* ⇛ *?S*)↾*in-dom flip.tpdfr.R*⤵*in-codom flip.tpdfr.L*.

end

Dependent Function Relator While a general transport of functions is only possible for the monotone function relator (see above), the locales *transport-Dep-Fun-Rel* and *transport-Fun-Rel* contain special cases to transport functions that are proven to be monotone using the standard function space.

Moreover, in the special case of equivalences on partial equivalence relations, the standard function space is monotone - see ⟦*galois.galois-equivalence ?L1.0 ?R1.0 ?l1.0 ?r1.0*; *preorder-on* (*in-field ?L1.0*) *?L1.0*; $((x1\ x2 :: ?L1.0^{-1}) \Rightarrow (x3\ x4 :: ?L1.0) \Rrightarrow ?L1.0\ x1\ x3 \longrightarrow (\leq))$ *?L2.0*; $\bigwedge x1\ x2.$ *?L1.0 x1 x2* ⟹ *partial-equivalence-rel* (*?L2.0 x1 x2*)⟧ ⟹ *transport-Mono-Dep-Fun-Rel.L ?L1.0 ?L2.0* = *transport-Dep-Fun-Rel.L ?L1.0 ?L2.0* As such, we can derive general transport theorems from the monotone cases above.

context *transport-Dep-Fun-Rel*
begin

interpretation *tpdfr* : *transport-Mono-Dep-Fun-Rel L1 R1 l1 r1 L2 R2 l2 r2* ⟨*proof*⟩
interpretation *flip* : *transport-Mono-Dep-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2* ⟨*proof*⟩

theorem *partial-equivalence-rel-equivalenceI*:
assumes $((\leq_{L1}) \equiv_{PER} (\leq_{R1}))$ *l1 r1*
and $\bigwedge x\ x'.\ x\ {}_{L1}\!\lessapprox x' \Longrightarrow ((\leq_{L2}\ x\ (r1\ x')) \equiv_{PER} (\leq_{R2}\ (l1\ x)\ x'))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $((x1\ x2 :: (\geq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rrightarrow (\leq))$ *L2*
and $((x1'\ x2' :: (\geq_{R1})) \Rightarrow (x3'\ x4' :: (\leq_{R1}) \mid x1' \leq_{R1} x3') \Rrightarrow (\leq))$ *R2*

and $((x1\,'\ x2\,' :: (\leq_{R1})) \Rightarrow (x1\ x2 :: (\leq_{L1}) \mid x2\ {}_{L1}\lessapprox x1\,') \Rrightarrow$
in-field $(\leq_{L2\ x1\ (r1\ x2\,')}) \Rrightarrow (\leq_{R2\ (l1\ x1)\ x2\,'}))\ l2$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow (x1\,'\ x2\,' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox x1\,') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2\,'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2\,')}))\ r2$
shows $((\leq_L) \equiv_{PER} (\leq_R))\ l\ r$
⟨*proof*⟩

end

Function Relator **context** *transport-Fun-Rel*
begin

interpretation *tpfr* : *transport-Mono-Fun-Rel L1 R1 l1 r1 L2 R2 l2 r2* ⟨*proof*⟩
interpretation *flip-tpfr* : *transport-Mono-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2* ⟨*proof*⟩

theorem *partial-equivalence-rel-equivalenceI*:
assumes $((\leq_{L1}) \equiv_{PER} (\leq_{R1}))\ l1\ r1$
and $((\leq_{L2}) \equiv_{PER} (\leq_{R2}))\ l2\ r2$
shows $((\leq_L) \equiv_{PER} (\leq_R))\ l\ r$
⟨*proof*⟩

end

end

2.16 Transport using Identity

theory *Transport-Identity*
imports
Transport-Bijections
begin

Summary Setup for Transport using the identity transport function.

locale *transport-id* =
fixes $L :: 'a \Rightarrow 'a \Rightarrow bool$
begin

sublocale *tbij?* : *transport-bijection L L id id*
⟨*proof*⟩

interpretation *transport L L id id* ⟨*proof*⟩

lemma *left-Galois-eq-left*: $({}_L\lessapprox) = (\leq_L)$
⟨*proof*⟩

end

locale *transport-reflexive-on-in-field-id* =
  fixes $L :: 'a \Rightarrow 'a \Rightarrow bool$
  assumes *reflexive-on-in-field*: *reflexive-on* (*in-field* L) L
begin

sublocale *trefl-bij?* : *transport-reflexive-on-in-field-bijection* L L *id id*
  ⟨*proof*⟩

end

locale *transport-preorder-on-in-field-id* =
  fixes $L :: 'a \Rightarrow 'a \Rightarrow bool$
  assumes *preorder-on-in-field*: *preorder-on* (*in-field* L) L
begin

sublocale *tpre-bij?* : *transport-preorder-on-in-field-bijection* L L *id id*
  ⟨*proof*⟩

end

locale *transport-partial-equivalence-rel-id* =
  fixes $L :: 'a \Rightarrow 'a \Rightarrow bool$
  assumes *partial-equivalence-rel*: *partial-equivalence-rel* L
begin

sublocale *tper-bij?* : *transport-partial-equivalence-rel-bijection* L L *id id*
  ⟨*proof*⟩

end

interpretation *transport-eq-restrict-id* :
  transport-eq-restrict-bijection P P *id id* **for** $P :: 'a \Rightarrow bool$
  ⟨*proof*⟩

interpretation *transport-eq-id* : *transport-eq-bijection id id*
  ⟨*proof*⟩

end

theory *Transport-Black-Box*
  imports
    Transport-Bijections
    Transport-Compositions
    Transport-Functions
    Transport-Identity
begin

Summary The theory for black-box transports. For details, refer to [2].

end

2.17 White-Box Transport of (Restricted) Equality

theory *Transport-Equality*
imports
Restricted-Equality
Binary-Relations-Bi-Unique
begin

Summary Theorems for white-box transports of (restricted) equalities.

context
fixes $R :: 'a \Rightarrow 'b \Rightarrow bool$ **and** $P :: 'a \Rightarrow bool$ **and** $Q :: 'b \Rightarrow bool$
begin

lemma *Fun-Rel-imp-eq-restrict-if-right-unique-onI*:
assumes *runique*: *right-unique-on P R*
and *rel*: $(R \Rrightarrow (\longrightarrow))\ P\ Q$
shows $(R \Rrightarrow R \Rrightarrow (\longrightarrow))\ (=_P)\ (=_Q)$
⟨*proof*⟩

lemma *Fun-Rel-rev-imp-eq-restrict-if-rel-injective-atI*:
assumes *rinjective*: *rel-injective-at Q R*
and *rel*: $(R \Rrightarrow (\longleftarrow))\ P\ Q$
shows $(R \Rrightarrow R \Rrightarrow (\longleftarrow))\ (=_P)\ (=_Q)$
⟨*proof*⟩

corollary *Fun-Rel-iff-eq-restrict-if-bi-unique-onI*:
assumes *bi-unique*: *bi-unique-on P R*
and $(R \Rrightarrow (\longleftrightarrow))\ P\ Q$
shows $(R \Rrightarrow R \Rrightarrow (\longleftrightarrow))\ (=_P)\ (=_Q)$
⟨*proof*⟩

lemma *right-unique-on-if-Fun-Rel-imp-eq-restrict*:
assumes $(R \Rrightarrow R \Rrightarrow (\longrightarrow))\ (=_P)\ (=)$
shows *right-unique-on P R*
⟨*proof*⟩

lemma *Fun-Rel-imp-if-Fun-Rel-imp-eq-restrict*:
assumes $(R \Rrightarrow R \Rrightarrow (\longrightarrow))\ (=_P)\ ((S :: 'b \Rightarrow 'b \Rightarrow bool)\restriction_Q)$
shows $(R \Rrightarrow (\longrightarrow))\ P\ Q$
⟨*proof*⟩

corollary *Fun-Rel-imp-eq-restrict-iff-right-unique-on-and-Fun-Rel-imp*:
$(R \Rrightarrow R \Rrightarrow (\longrightarrow))\ (=_P)\ (=_Q) \longleftrightarrow (\textit{right-unique-on}\ P\ R \wedge (R \Rrightarrow (\longrightarrow))\ P\ Q)$
⟨*proof*⟩

lemma *rel-injective-at-if-Fun-Rel-rev-imp-eq-restrict*:
assumes $(R \Rrightarrow R \Rrightarrow (\longleftarrow))\ (=)\ (=_Q)$
shows *rel-injective-at* $Q\ R$
⟨*proof*⟩

lemma *Fun-Rel-rev-imp-if-Fun-Rel-rev-imp-eq-restrict*:
assumes $(R \Rrightarrow R \Rrightarrow (\longleftarrow))\ ((S :: 'a \Rightarrow 'a \Rightarrow \textit{bool})\restriction_P)\ (=_Q)$
shows $(R \Rrightarrow (\longleftarrow))\ P\ Q$
⟨*proof*⟩

corollary *Fun-Rel-rev-imp-eq-restrict-iff-rel-injective-at-and-Fun-Rel-rev-imp*:
$(R \Rrightarrow R \Rrightarrow (\longleftarrow))\ (=_P)\ (=_Q) \longleftrightarrow (\textit{rel-injective-at}\ Q\ R \wedge (R \Rrightarrow (\longleftarrow))\ P\ Q)$
⟨*proof*⟩

lemma *bi-unique-on-if-Fun-Rel-iff-eq-restrict*:
assumes $(R \Rrightarrow R \Rrightarrow (\longleftrightarrow))\ (=_P)\ (=_Q)$
shows *bi-unique-on* $P\ R$
⟨*proof*⟩

lemma *Fun-Rel-iff-if-Fun-Rel-iff-eq-restrict*:
assumes $(R \Rrightarrow R \Rrightarrow (\longleftrightarrow))\ (=_P)\ (=_Q)$
shows $(R \Rrightarrow (\longleftrightarrow))\ P\ Q$
⟨*proof*⟩

corollary *Fun-Rel-iff-eq-restrict-iff-bi-unique-on-and-Fun-Rel-iff*:
$(R \Rrightarrow R \Rrightarrow (\longleftrightarrow))\ (=_P)\ (=_Q) \longleftrightarrow (\textit{bi-unique-on}\ P\ R \wedge (R \Rrightarrow (\longleftrightarrow))\ P\ Q)$
⟨*proof*⟩

end

context
fixes $R :: 'a \Rightarrow 'b \Rightarrow \textit{bool}$
begin

corollary *Fun-Rel-imp-eq-if-right-unique*:
assumes *right-unique* R
shows $(R \Rrightarrow R \Rrightarrow (\longrightarrow))\ (=)\ (=)$
⟨*proof*⟩

corollary *Fun-Rel-rev-imp-eq-if-rel-injective*:
assumes *rel-injective* R
shows $(R \Rrightarrow R \Rrightarrow (\longleftarrow))\ (=)\ (=)$
⟨*proof*⟩

corollary *Fun-Rel-iff-eq-if-bi-unique*:

assumes *bi-unique R*
shows $(R \Rrightarrow R \Rrightarrow (\longleftrightarrow))\ (=)\ (=)$
⟨*proof*⟩

corollary *right-unique-if-Fun-Rel-imp-eq*:
assumes $(R \Rrightarrow R \Rrightarrow (\longrightarrow))\ (=)\ (=)$
shows *right-unique R*
⟨*proof*⟩

corollary *Fun-Rel-imp-eq-iff-right-unique*: $(R \Rrightarrow R \Rrightarrow (\longrightarrow))\ (=)\ (=) \longleftrightarrow$ *right-unique R*
⟨*proof*⟩

corollary *rel-injective-if-Fun-Rel-rev-imp-eq*:
assumes $(R \Rrightarrow R \Rrightarrow (\longleftarrow))\ (=)\ (=)$
shows *rel-injective R*
⟨*proof*⟩

corollary *Fun-Rel-rev-imp-eq-iff-rel-injective*: $(R \Rrightarrow R \Rrightarrow (\longleftarrow))\ (=)\ (=) \longleftrightarrow$ *rel-injective R*
⟨*proof*⟩

corollary *bi-unique-if-Fun-Rel-iff-eq*:
assumes $(R \Rrightarrow R \Rrightarrow (\longleftrightarrow))\ (=)\ (=)$
shows *bi-unique R*
⟨*proof*⟩

corollary *Fun-Rel-iff-eq-iff-bi-unique*: $(R \Rrightarrow R \Rrightarrow (\longleftrightarrow))\ (=)\ (=) \longleftrightarrow$ *bi-unique R*
⟨*proof*⟩

end

end

2.18 White-Box Transport of (Bounded) Universal Quantifier

theory *Transport-Universal-Quantifier*
imports
Bounded-Quantifiers
Binary-Relations-Bi-Total
Reverse-Implies
begin

Summary Theorems for white-box transports of (bounded) universal quantifiers.

context
  fixes $R :: 'a \Rightarrow 'b \Rightarrow bool$
  and $P :: 'a \Rightarrow bool$ **and** $Q :: 'b \Rightarrow bool$
begin

lemma *Fun-Rel-restricts-imp-ball-if-rel-surjective-atI*:
  assumes *rel-surjective-at* Q $R{\restriction}_P$
  shows $((R{\restriction}_P){\restriction}_Q \Rrightarrow (\longrightarrow)) \Rrightarrow (\longrightarrow)) \forall_P \forall_Q$
  ⟨*proof*⟩

lemma *Fun-Rel-restricts-rev-imp-ball-if-left-total-onI*:
  assumes *left-total-on* P $R{\restriction}_Q$
  shows $((R{\restriction}_P){\restriction}_Q \Rrightarrow (\longleftarrow)) \Rrightarrow (\longleftarrow)) \forall_P \forall_Q$
  ⟨*proof*⟩

lemma *Fun-Rel-restricts-iff-ball-if-left-total-on-if-rel-surjective-at*:
  assumes *rel-surjective-at* Q $R{\restriction}_P$
  and *left-total-on* P $R{\restriction}_Q$
  shows $((R{\restriction}_P){\restriction}_Q \Rrightarrow (\longleftrightarrow)) \Rrightarrow (\longleftrightarrow)) \forall_P \forall_Q$
  ⟨*proof*⟩

corollary *Fun-Rel-restricts-iff-ball-if-bi-total-on*:
  assumes *bi-total-on* P Q $R{\restriction}_P{\restriction}_Q$
  shows $((R{\restriction}_P){\restriction}_Q \Rrightarrow (\longleftrightarrow)) \Rrightarrow (\longleftrightarrow)) \forall_P \forall_Q$
  ⟨*proof*⟩

lemma *rel-surjective-at-restrict-left-if-Fun-Rel-imp-ball*:
  assumes $((R \Rrightarrow (\longrightarrow)) \Rrightarrow (\longrightarrow)) \forall_P \forall_Q$
  shows *rel-surjective-at* Q $R{\restriction}_P$
⟨*proof*⟩

lemma *Fun-Rel-Fun-Rel-if-le-left-if-Fun-Rel-Fun-Rel*:
  assumes $(((O :: 'a \Rightarrow 'b \Rightarrow bool) \Rrightarrow S) \Rrightarrow T)\ U\ V$
  and $O \leq O'$
  shows $((O' \Rrightarrow S) \Rrightarrow T)\ U\ V$
⟨*proof*⟩

corollary *Fun-Rel-imp-ball-iff-rel-surjective-at-restrict-left*:
  $((R \Rrightarrow (\longrightarrow)) \Rrightarrow (\longrightarrow)) \forall_P \forall_Q \longleftrightarrow$ *rel-surjective-at* Q $R{\restriction}_P$
  ⟨*proof*⟩

lemma *left-total-on-restrict-right-if-Fun-Rel-rev-imp-ball*:
  assumes $((R \Rrightarrow (\longleftarrow)) \Rrightarrow (\longleftarrow)) \forall_P \forall_Q$
  shows *left-total-on* P $R{\restriction}_Q$
⟨*proof*⟩

corollary *Fun-Rel-rev-imp-ball-iff-left-total-on-restrict-right*:
((*R* ⇛ (⟵)) ⇛ (⟵)) ∀ *P* ∀ *Q* ⟷ *left-total-on P R*⌉ *Q*
⟨*proof*⟩

lemma *bi-total-on-if-Fun-Rel-iff-ball*:
assumes ((*R* ⇛ (⟷)) ⇛ (⟷)) ∀ *P* ∀ *Q*
shows *bi-total-on P Q R*
⟨*proof*⟩

corollary *Fun-Rel-iff-ball-iff-bi-total-on-if-Fun-Rel-iff*:
assumes (*R* ⇛ (⟷)) *P Q*
shows ((*R* ⇛ (⟷)) ⇛ (⟷)) ∀ *P* ∀ *Q* ⟷ *bi-total-on P Q R*
⟨*proof*⟩

end

context
fixes *R* :: *'a* ⇒ *'b* ⇒ *bool*
begin

corollary *Fun-Rel-imp-all-if-rel-surjective*:
assumes *rel-surjective R*
shows ((*R* ⇛ (⟶)) ⇛ (⟶)) *All All*
⟨*proof*⟩

corollary *Fun-Rel-rev-imp-all-if-left-total*:
assumes *left-total R*
shows ((*R* ⇛ (⟵)) ⇛ (⟵)) *All All*
⟨*proof*⟩

corollary *Fun-Rel-iff-all-if-bi-total*:
assumes *bi-total R*
shows ((*R* ⇛ (⟷)) ⇛ (⟷)) *All All*
⟨*proof*⟩

corollary *rel-surjective-if-Fun-Rel-imp-all*:
assumes ((*R* ⇛ (⟶)) ⇛ (⟶)) *All All*
shows *rel-surjective R*
⟨*proof*⟩

corollary *Fun-Rel-imp-all-iff-rel-surjective*:
((*R* ⇛ (⟶)) ⇛ (⟶)) *All All* ⟷ *rel-surjective R*
⟨*proof*⟩

corollary *left-total-if-Fun-Rel-rev-imp-all*:
assumes ((*R* ⇛ (⟵)) ⇛ (⟵)) *All All*
shows *left-total R*
⟨*proof*⟩

corollary *Fun-Rel-rev-imp-all-iff-left-total*:
((*R* ⇛ (⟵)) ⇛ (⟵)) *All All* ⟷ *left-total R*
⟨*proof*⟩

corollary *bi-total-if-Fun-Rel-iff-all*:
assumes ((*R* ⇛ (⟷)) ⇛ (⟷)) *All All*
shows *bi-total R*
⟨*proof*⟩

corollary *Fun-Rel-iff-all-iff-bi-total*:
((*R* ⇛ (⟷)) ⇛ (⟷)) *All All* ⟷ *bi-total R*
⟨*proof*⟩

end

end

2.19 White-Box Transport of (Bounded) Existential Quantifier

theory *Transport-Existential-Quantifier*
imports
Transport-Universal-Quantifier
begin

Summary Theorems for white-box transports of (bounded) existential quantifiers.

context
fixes *R* :: *'a* ⇒ *'b* ⇒ *bool* **and** *P* :: *'a* ⇒ *bool* **and** *Q* :: *'b* ⇒ *bool*
begin

lemma *Fun-Rel-restricts-imp-bex-if-left-total-onI*:
assumes *left-total-on P R*↾*Q*
shows ((*R*↾*P*↾*Q* ⇛ (⟶)) ⇛ (⟶)) ∃ *P* ∃ *Q*
⟨*proof*⟩

lemma *Fun-Rel-restricts-rev-imp-bex-if-rel-surjective-at*:
assumes *rel-surjective-at Q R*↾*P*
shows ((*R*↾*P*↾*Q* ⇛ (⟵)) ⇛ (⟵)) ∃ *P* ∃ *Q*
⟨*proof*⟩

lemma *Fun-Rel-restricts-iff-bex-if-left-total-on-if-rel-surjective-at*:
assumes *rel-surjective-at Q R*↾*P*
and *left-total-on P R*↾*Q*

shows $((R\restriction_P\restriction_Q \Rrightarrow (\longleftrightarrow)) \Rrightarrow (\longleftrightarrow))\ \exists_P\ \exists_Q$
⟨*proof*⟩

corollary *Fun-Rel-restricts-iff-bex-if-bi-total-on*:
assumes *bi-total-on P Q* $R\restriction_P\restriction_Q$
shows $((R\restriction_P\restriction_Q \Rrightarrow (\longleftrightarrow)) \Rrightarrow (\longleftrightarrow))\ \exists_P\ \exists_Q$
⟨*proof*⟩

lemma *left-total-on-restrict-right-if-Fun-Rel-imp-bex*:
assumes $((R \Rrightarrow (\longrightarrow)) \Rrightarrow (\longrightarrow))\ \exists_P\ \exists_Q$
shows *left-total-on P* $R\restriction_Q$
⟨*proof*⟩

corollary *Fun-Rel-imp-all-on-iff-left-total-on-restrict-right*:
$((R \Rrightarrow (\longrightarrow)) \Rrightarrow (\longrightarrow))\ \exists_P\ \exists_Q \longleftrightarrow$ *left-total-on P* $R\restriction_Q$
⟨*proof*⟩

lemma *rel-surjective-at-restrict-left-if-Fun-Rel-rev-imp-bex*:
assumes $((R \Rrightarrow (\longleftarrow)) \Rrightarrow (\longleftarrow))\ \exists_P\ \exists_Q$
shows *rel-surjective-at Q* $R\restriction_P$
⟨*proof*⟩

corollary *Fun-Rel-rev-imp-bex-iff-rel-surjective-at-restrict-left*:
$((R \Rrightarrow (\longleftarrow)) \Rrightarrow (\longleftarrow))\ \exists_P\ \exists_Q \longleftrightarrow$ *rel-surjective-at Q* $R\restriction_P$
⟨*proof*⟩

lemma *bi-total-on-if-Fun-Rel-iff-bex*:
assumes $((R \Rrightarrow (\longleftrightarrow)) \Rrightarrow (\longleftrightarrow))\ \exists_P\ \exists_Q$
shows *bi-total-on P Q R*
⟨*proof*⟩

corollary *Fun-Rel-iff-bex-iff-bi-total-on-if-Fun-Rel-iff*:
assumes $(R \Rrightarrow (\longleftrightarrow))\ P\ Q$
shows $((R \Rrightarrow (\longleftrightarrow)) \Rrightarrow (\longleftrightarrow))\ \exists_P\ \exists_Q \longleftrightarrow$ *bi-total-on P Q R*
⟨*proof*⟩

end

context
fixes $R :: 'a \Rightarrow 'b \Rightarrow bool$
begin

corollary *Fun-Rel-imp-ex-if-left-total*:
assumes *left-total R*
shows $((R \Rrightarrow (\longrightarrow)) \Rrightarrow (\longrightarrow))$ *Ex Ex*
⟨*proof*⟩

corollary *Fun-Rel-rev-imp-ex-if-rel-surjective*:
assumes *rel-surjective R*

shows $((R \Rightarrow (\longleftarrow)) \Rightarrow (\longleftarrow))$ *Ex Ex*
⟨*proof*⟩

corollary *Fun-Rel-iff-ex-if-bi-total*:
assumes *bi-total R*
shows $((R \Rightarrow (\longleftrightarrow)) \Rightarrow (\longleftrightarrow))$ *Ex Ex*
⟨*proof*⟩

corollary *left-total-if-Fun-Rel-imp-ex*:
assumes $((R \Rightarrow (\longrightarrow)) \Rightarrow (\longrightarrow))$ *Ex Ex*
shows *left-total R*
⟨*proof*⟩

corollary *Fun-Rel-imp-ex-iff-left-total*:
$((R \Rightarrow (\longrightarrow)) \Rightarrow (\longrightarrow))$ *Ex Ex* $\longleftrightarrow$ *left-total R*
⟨*proof*⟩

corollary *rel-surjective-if-Fun-Rel-rev-imp-ex*:
assumes $((R \Rightarrow (\longleftarrow)) \Rightarrow (\longleftarrow))$ *Ex Ex*
shows *rel-surjective R*
⟨*proof*⟩

corollary *Fun-Rel-rev-imp-ex-iff-rel-surjective*:
$((R \Rightarrow (\longleftarrow)) \Rightarrow (\longleftarrow))$ *Ex Ex* $\longleftrightarrow$ *rel-surjective R*
⟨*proof*⟩

corollary *bi-total-if-Fun-Rel-iff-ex*:
assumes $((R \Rightarrow (\longleftrightarrow)) \Rightarrow (\longleftrightarrow))$ *Ex Ex*
shows *bi-total R*
⟨*proof*⟩

corollary *Fun-Rel-iff-ex-iff-bi-total*:
$((R \Rightarrow (\longleftrightarrow)) \Rightarrow (\longleftrightarrow))$ *Ex Ex* $\longleftrightarrow$ *bi-total R*
⟨*proof*⟩

end

end

2.20 Properties of Galois Relator for White-Box Transport Side Conditions

theory *Transport-Galois-Relator-Properties*
imports
Binary-Relations-Bi-Total
Binary-Relations-Bi-Unique

Galois-Connections
Galois-Relator
begin

Summary Properties of Galois relator arising as side conditions for white-box transport.

context *galois*
begin

Right-Uniqueness **context**
fixes $P :: 'a \Rightarrow bool$ **and** $Q :: 'b \Rightarrow bool$
begin

lemma *right-unique-at-left-Galois-if-right-unique-at-rightI*:
assumes *right-unique-at* Q $(\leq_R)$
and $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$
shows *right-unique-at* Q $({}_L\lessapprox)$
⟨*proof*⟩

lemma *right-unique-at-right-if-right-unique-at-left-GaloisI*:
assumes *right-unique-at* Q $({}_L\lessapprox)$
and $((\leq_R) \Rightarrow (\leq_L))\ r$
shows *right-unique-at* Q $(\leq_R)$
⟨*proof*⟩

corollary *right-unique-at-left-Galois-iff-right-unique-at-rightI*:
assumes $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$
and $((\leq_R) \Rightarrow (\leq_L))\ r$
shows *right-unique-at* Q $({}_L\lessapprox) \longleftrightarrow$ *right-unique-at* Q $(\leq_R)$
⟨*proof*⟩

corollary *right-unique-on-left-Galois-if-right-unique-at-rightI*:
assumes *right-unique-at* Q $(\leq_R)$
and $(({}_L\lessapprox) \Rrightarrow (\longrightarrow))\ P\ Q$
and $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$
shows *right-unique-on* P $({}_L\lessapprox)$
⟨*proof*⟩

corollary *right-unique-at-right-if-right-unique-on-left-GaloisI*:
assumes *right-unique-on* P $({}_L\lessapprox)$
and $(({}_L\lessapprox) \Rrightarrow (\longleftarrow))\ P\ Q$
and $((\leq_R) \Rightarrow (\leq_L))\ r$
shows *right-unique-at* Q $(\leq_R)$
⟨*proof*⟩

corollary *right-unique-on-left-Galois-iff-right-unique-at-rightI*:
assumes $(({}_L\lessapprox) \Rrightarrow (\longleftrightarrow))\ P\ Q$
and $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$

and $((\leq_R) \Rightarrow (\leq_L))\ r$
shows *right-unique-on* $P\ ({}_L\lessapprox)$ $\longleftrightarrow$ *right-unique-at* $Q\ (\leq_R)$
⟨*proof*⟩

end

corollary *right-unique-left-Galois-if-right-unique-rightI*:
assumes *right-unique* $(\leq_R)$
and $(({\leq_L})\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$
shows *right-unique* $({}_L\gtrapprox)$
⟨*proof*⟩

corollary *right-unique-right-if-right-unique-left-GaloisI*:
assumes *right-unique* $({}_L\lessapprox)$
and $((\leq_R) \Rightarrow (\leq_L))\ r$
shows *right-unique* $(\leq_R)$
⟨*proof*⟩

corollary *right-unique-left-Galois-iff-right-unique-rightI*:
assumes $(({\leq_L})\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$
and $((\leq_R) \Rightarrow (\leq_L))\ r$
shows *right-unique* $({}_L\lessapprox)$ $\longleftrightarrow$ *right-unique* $(\leq_R)$
⟨*proof*⟩

Injectivity **context**
fixes $P :: 'a \Rightarrow bool$ **and** $Q :: 'b \Rightarrow bool$
begin

lemma *injective-on-left-Galois-if-rel-injective-on-left*:
assumes *rel-injective-on* $P\ (\leq_L)$
shows *rel-injective-on* $P\ ({}_L\lessapprox)$
⟨*proof*⟩

lemma *rel-injective-on-left-if-injective-on-left-GaloisI*:
assumes *rel-injective-on* $P\ ({}_L\lessapprox)$
and $((\leq_L) \Rightarrow (\leq_R))\ l$
and $((\leq_L) \trianglelefteq_h\ (\leq_R))\ l\ r$
shows *rel-injective-on* $P\ (\leq_L)$
⟨*proof*⟩

corollary *injective-on-left-Galois-iff-rel-injective-on-leftI*:
assumes $((\leq_L) \Rightarrow (\leq_R))\ l$
and $((\leq_L) \trianglelefteq_h\ (\leq_R))\ l\ r$
shows *rel-injective-on* $P\ ({}_L\gtrapprox)$ $\longleftrightarrow$ *rel-injective-on* $P\ (\leq_L)$
⟨*proof*⟩

corollary *injective-at-left-Galois-if-rel-injective-on-leftI*:
assumes *rel-injective-on* $P\ (\leq_L)$
and $(({}_L\lessapprox) \Rrightarrow (\longleftarrow))\ P\ Q$

shows *rel-injective-at* Q $({}_L\lessapprox)$
⟨*proof*⟩

corollary *rel-injective-on-left-if-injective-at-left-GaloisI*:
assumes *rel-injective-at* Q $({}_L\lessapprox)$
and $(({}_L\lessapprox) \Rrightarrow (\longrightarrow))$ P Q
and $((\leq_L) \Rightarrow (\leq_R))$ l
and $((\leq_L) \trianglelefteq_h (\leq_R))$ l r
shows *rel-injective-on* P $(\leq_L)$
⟨*proof*⟩

corollary *injective-at-left-Galois-iff-rel-injective-on-leftI*:
assumes $(({}_L\lessapprox) \Rrightarrow (\longleftrightarrow))$ P Q
and $((\leq_L) \Rightarrow (\leq_R))$ l
and $((\leq_L) \trianglelefteq_h (\leq_R))$ l r
shows *rel-injective-at* Q $({}_L\lessapprox) \longleftrightarrow$ *rel-injective-on* P $(\leq_L)$
⟨*proof*⟩

end

corollary *injective-left-Galois-if-rel-injective-left*:
assumes *rel-injective* $(\leq_L)$
shows *rel-injective* $({}_L\lessapprox)$
⟨*proof*⟩

corollary *rel-injective-left-if-injective-left-GaloisI*:
assumes *rel-injective* $({}_L\lessapprox)$
and $((\leq_L) \Rightarrow (\leq_R))$ l
and $((\leq_L) \trianglelefteq_h (\leq_R))$ l r
shows *rel-injective* $(\leq_L)$
⟨*proof*⟩

corollary *rel-injective-left-Galois-iff-rel-injective-leftI*:
assumes $((\leq_L) \Rightarrow (\leq_R))$ l
and $((\leq_L) \trianglelefteq_h (\leq_R))$ l r
shows *rel-injective* $({}_L\lessapprox) \longleftrightarrow$ *rel-injective* $(\leq_L)$
⟨*proof*⟩

Bi-Uniqueness context

fixes $P :: 'a \Rightarrow bool$ **and** $Q :: 'b \Rightarrow bool$
begin

corollary *bi-unique-on-left-Galois-if-right-unique-at-right-if-rel-injective-on-leftI*:
assumes *rel-injective-on* P $(\leq_L)$
and *right-unique-at* Q $(\leq_R)$
and $(({}_L\lessapprox) \Rrightarrow (\longrightarrow))$ P Q
and $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))$ l r
shows *bi-unique-on* P $({}_L\lessapprox)$
⟨*proof*⟩

corollary *rel-injective-on-left-and-right-unique-at-right-if-bi-unique-on-left-GaloisI*:
assumes *bi-unique-on* P $({}_L\lessapprox)$
and $(({}_L\lessapprox) \Rrightarrow (\longleftarrow))$ P Q
and $((\leq_L) \Rightarrow (\leq_R))$ l
and $((\leq_R) \Rightarrow (\leq_L))$ r
and $((\leq_L) \trianglelefteq_h (\leq_R))$ l r
shows *rel-injective-on* P $(\leq_L) \wedge$ *right-unique-at* Q $(\leq_R)$
⟨*proof*⟩

corollary *bi-unique-on-left-Galois-iff-rel-injective-on-left-and-right-unique-at-rightI*:
assumes $(({}_L\lessapprox) \Rrightarrow (\longleftrightarrow))$ P Q
and $((\leq_L) \dashv (\leq_R))$ l r
shows *bi-unique-on* P $({}_L\lessapprox) \longleftrightarrow$ *rel-injective-on* P $(\leq_L) \wedge$ *right-unique-at* Q $(\leq_R)$
⟨*proof*⟩

end

corollary *bi-unique-left-Galois-if-right-unique-right-if-rel-injective-leftI*:
assumes *rel-injective* $(\leq_L)$
and *right-unique* $(\leq_R)$
and $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))$ l r
shows *bi-unique* $({}_L\lessapprox)$
⟨*proof*⟩

corollary *rel-injective-left-and-right-unique-right-if-bi-unique-left-GaloisI*:
assumes *bi-unique* $({}_L\lessapprox)$
and $((\leq_L) \Rightarrow (\leq_R))$ l
and $((\leq_R) \Rightarrow (\leq_L))$ r
and $((\leq_L) \trianglelefteq_h (\leq_R))$ l r
shows *rel-injective* $(\leq_L) \wedge$ *right-unique* $(\leq_R)$
⟨*proof*⟩

corollary *bi-unique-left-Galois-iff-rel-injective-left-and-right-unique-rightI*:
assumes $((\leq_L) \dashv (\leq_R))$ l r
shows *bi-unique* $({}_L\lessapprox) \longleftrightarrow$ *rel-injective* $(\leq_L) \wedge$ *right-unique* $(\leq_R)$
⟨*proof*⟩

Surjectivity **context**
fixes $Q :: {}'b \Rightarrow bool$
begin

lemma *surjective-at-left-Galois-if-rel-surjective-at-rightI*:
assumes *rel-surjective-at* Q $(\leq_R)$
and $((\leq_R) \Rightarrow (\leq_L))$ r
shows *rel-surjective-at* Q $({}_L\lessapprox)$
⟨*proof*⟩

lemma *rel-surjective-at-right-if-surjective-at-left-Galois*:
assumes *rel-surjective-at* Q $({}_{L}\lessapprox)$
shows *rel-surjective-at* Q $(\leq_R)$
⟨*proof*⟩

corollary *rel-surjective-at-right-iff-surjective-at-left-GaloisI*:
assumes $((\leq_R) \Rightarrow (\leq_L))$ r
shows *rel-surjective-at* Q $({}_{L}\lessapprox) \longleftrightarrow$ *rel-surjective-at* Q $(\leq_R)$
⟨*proof*⟩

end

corollary *surjective-left-Galois-if-rel-surjective-rightI*:
assumes *rel-surjective* $(\leq_R)$
and $((\leq_R) \Rightarrow (\leq_L))$ r
shows *rel-surjective* $({}_{L}\gtrapprox)$
⟨*proof*⟩

corollary *rel-surjective-right-if-surjective-left-Galois*:
assumes *rel-surjective* $({}_{L}\gtrapprox)$
shows *rel-surjective* $(\leq_R)$
⟨*proof*⟩

corollary *rel-surjective-right-iff-surjective-left-GaloisI*:
assumes $((\leq_R) \Rightarrow (\leq_L))$ r
shows *rel-surjective* $({}_{L}\gtrapprox) \longleftrightarrow$ *rel-surjective* $(\leq_R)$
⟨*proof*⟩

Left-Totality **context**
fixes $P :: 'a \Rightarrow bool$
begin

lemma *left-total-on-left-Galois-if-left-total-on-leftI*:
assumes *left-total-on* P $(\leq_L)$
and $((\leq_L) \Rightarrow (\leq_R))$ l
and $((\leq_L) \trianglelefteq_h (\leq_R))$ l r
shows *left-total-on* P $({}_{L}\lessapprox)$
⟨*proof*⟩

lemma *left-total-on-left-if-left-total-on-left-GaloisI*:
assumes *left-total-on* P $({}_{L}\gtrapprox)$
shows *left-total-on* P $(\leq_L)$
⟨*proof*⟩

corollary *left-total-on-left-Galois-iff-left-total-on-leftI*:
assumes $((\leq_L) \Rightarrow (\leq_R))$ l
and $((\leq_L) \trianglelefteq_h (\leq_R))$ l r
shows *left-total-on* P $({}_{L}\gtrapprox) \longleftrightarrow$ *left-total-on* P $(\leq_L)$
⟨*proof*⟩

end

corollary *left-total-left-Galois-if-left-total-leftI*:
assumes *left-total* $(\leq_L)$
and $((\leq_L) \Rightarrow (\leq_R))$ l
and $((\leq_L) \trianglelefteq_h (\leq_R))$ l r
shows *left-total* $({}_L\lessapprox)$
⟨*proof*⟩

corollary *left-total-left-if-left-total-left-Galois*:
assumes *left-total* $({}_L\lessapprox)$
shows *left-total* $(\leq_L)$
⟨*proof*⟩

corollary *left-total-left-Galois-iff-left-total-leftI*:
assumes $((\leq_L) \Rightarrow (\leq_R))$ l
and $((\leq_L) \trianglelefteq_h (\leq_R))$ l r
shows *left-total* $({}_L\lessapprox) \longleftrightarrow$ *left-total* $(\leq_L)$
⟨*proof*⟩

Bi-Totality **context**
fixes $P :: 'a \Rightarrow bool$ **and** $Q :: 'b \Rightarrow bool$
begin

lemma *bi-total-on-left-GaloisI*:
assumes *left-total-on* P $(\leq_L)$
and *rel-surjective-at* Q $(\leq_R)$
and $((\leq_L) \Rightarrow (\leq_R))$ l
and $((\leq_R) \Rightarrow (\leq_L))$ r
and $((\leq_L) \trianglelefteq_h (\leq_R))$ l r
shows *bi-total-on* P Q $({}_L\lessapprox)$
⟨*proof*⟩

lemma *left-total-on-left-rel-surjective-at-right-if-bi-total-on-left-GaloisE*:
assumes *bi-total-on* P Q $({}_L\lessapprox)$
obtains *left-total-on* P $(\leq_L)$ *rel-surjective-at* Q $(\leq_R)$
⟨*proof*⟩

corollary *bi-total-on-left-Galois-iff-left-total-on-left-and-rel-surjective-on-rightI*:
assumes $((\leq_L) \Rightarrow (\leq_R))$ l
and $((\leq_R) \Rightarrow (\leq_L))$ r
and $((\leq_L) \trianglelefteq_h (\leq_R))$ l r
shows *bi-total-on* P Q $({}_L\lessapprox) \longleftrightarrow$ *left-total-on* P $(\leq_L) \wedge$ *rel-surjective-at* Q $(\leq_R)$
⟨*proof*⟩

end

corollary *bi-total-left-GaloisI*:

assumes *left-total* $(\leq_L)$
and *rel-surjective* $(\leq_R)$
and $((\leq_L) \Rightarrow (\leq_R))\ l$
and $((\leq_R) \Rightarrow (\leq_L))\ r$
and $((\leq_L) \trianglelefteq_h (\leq_R))\ l\ r$
shows *bi-total* $({}_L\lesssim)$
⟨*proof*⟩

corollary *left-total-left-rel-surjective-right-if-bi-total-left-GaloisE*:
assumes *bi-total* $({}_L\lesssim)$
obtains *left-total* $(\leq_L)$ *rel-surjective* $(\leq_R)$
⟨*proof*⟩

corollary *bi-total-left-Galois-iff-left-total-left-and-rel-surjective-rightI*:
assumes $((\leq_L) \Rightarrow (\leq_R))\ l$
and $((\leq_R) \Rightarrow (\leq_L))\ r$
and $((\leq_L) \trianglelefteq_h (\leq_R))\ l\ r$
shows *bi-total* $({}_L\lesssim) \longleftrightarrow$ *left-total* $(\leq_L) \wedge$ *rel-surjective* $(\leq_R)$
⟨*proof*⟩

Function Relator **lemma** *Fun-Rel-left-Galois-left-Galois-imp-left-rightI*:
assumes *monol*: $((\leq_L) \Rightarrow (\leq_R))\ l$
and *half-gal*: $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$
and *perR*: *partial-equivalence-rel* $(\leq_R)$
shows $(({}_L\lesssim) \Rrightarrow ({}_L\lesssim) \Rrightarrow (\longrightarrow))\ (\leq_L)\ (\leq_R)$
⟨*proof*⟩

lemma *Fun-Rel-left-Galois-left-Galois-rev-imp-left-rightI*:
assumes *monor*: $((\leq_R) \Rightarrow (\leq_L))\ r$
and *perL*: *partial-equivalence-rel* $(\leq_L)$
shows $(({}_L\lesssim) \Rrightarrow ({}_L\lesssim) \Rrightarrow (\longleftarrow))\ (\leq_L)\ (\leq_R)$
⟨*proof*⟩

corollary *Fun-Rel-left-Galois-left-Galois-iff-left-rightI*:
assumes $((\leq_L) \Rightarrow (\leq_R))\ l$
and $((\leq_R) \Rightarrow (\leq_L))\ r$
and $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$
and *partial-equivalence-rel* $(\leq_L)$
and *partial-equivalence-rel* $(\leq_R)$
shows $(({}_L\lesssim) \Rrightarrow ({}_L\lesssim) \Rrightarrow (\longleftrightarrow))\ (\leq_L)\ (\leq_R)$
⟨*proof*⟩

end

end

theory *Transport-White-Box*
imports

Transport-Equality
Transport-Existential-Quantifier
Transport-Galois-Relator-Properties
Transport-Universal-Quantifier
begin

Summary Theorems for white-box transports.

context *galois*
begin

notepad
begin
print-statement *Fun-Rel-imp-eq-restrict-if-right-unique-onI*
[*of in-field* ($\leq_L$) ($_L\lessapprox$) *in-field* ($\leq_R$)]

⟨*proof*⟩
end

end

end

theory *Transport*
imports
Transport-Black-Box
Transport-White-Box
begin

Summary We formalise the theory for the Transport framework. The Transport framework allows us to transport terms along (partial) Galois connections (*galois.galois-connection*) and equivalences (*galois.galois-equivalence*). For details, refer to [2].

end

2.21 Transport for Dependent Function Relator with Non-Dependent Functions

theory *Transport-Rel-If*
imports
Transport-Black-Box
begin

Summary We introduce a special case of *transport-Dep-Fun-Rel*. The derived theorem is easier to apply and supported by the current prototype.

context

fixes $P :: 'a \Rightarrow bool$ **and** $R :: 'a \Rightarrow 'a \Rightarrow bool$
begin

lemma *reflexive-on-rel-if-if-reflexive-onI* [*intro*]:
assumes $B \Longrightarrow$ *reflexive-on* P R
shows *reflexive-on* P (*rel-if* B R)
⟨*proof*⟩

lemma *transitive-on-rel-if-if-transitive-onI* [*intro*]:
assumes $B \Longrightarrow$ *transitive-on* P R
shows *transitive-on* P (*rel-if* B R)
⟨*proof*⟩

lemma *preorder-on-rel-if-if-preorder-onI* [*intro*]:
assumes $B \Longrightarrow$ *preorder-on* P R
shows *preorder-on* P (*rel-if* B R)
⟨*proof*⟩

lemma *symmetric-on-rel-if-if-symmetric-onI* [*intro*]:
assumes $B \Longrightarrow$ *symmetric-on* P R
shows *symmetric-on* P (*rel-if* B R)
⟨*proof*⟩

lemma *partial-equivalence-rel-on-rel-if-if-partial-equivalence-rel-onI* [*intro*]:
assumes $B \Longrightarrow$ *partial-equivalence-rel-on* P R
shows *partial-equivalence-rel-on* P (*rel-if* B R)
⟨*proof*⟩

lemma *rel-if-dep-mono-wrt-rel-if-iff-if-dep-mono-wrt-relI*:
assumes $B \Longrightarrow B' \Longrightarrow ((x\ y :: R) \Rightarrow S\ x\ y)\ f$
and $B \longleftrightarrow B'$
shows $((x\ y ::$ *rel-if* $B\ R) \Rightarrow$ (*rel-if* B' $(S\ x\ y))) f$
⟨*proof*⟩

corollary *reflexive-rel-if-if-reflexiveI* [*intro*]:
assumes $B \Longrightarrow$ *reflexive* R
shows *reflexive* (*rel-if* B R)
⟨*proof*⟩

corollary *transitive-rel-if-if-transitiveI* [*intro*]:
assumes $B \Longrightarrow$ *transitive* R
shows *transitive* (*rel-if* B R)
⟨*proof*⟩

end

context
fixes $P :: 'a \Rightarrow bool$ **and** $R :: 'a \Rightarrow 'a \Rightarrow bool$
begin

corollary *preorder-rel-if-if-preorderI* [*intro*]:
assumes $B \Longrightarrow$ *preorder* R
shows *preorder* (*rel-if* B R)
⟨*proof*⟩

corollary *symmetric-rel-if-if-symmetricI* [*intro*]:
assumes $B \Longrightarrow$ *symmetric* R
shows *symmetric* (*rel-if* B R)
⟨*proof*⟩

corollary *partial-equivalence-rel-rel-if-if-partial-equivalence-relI* [*intro*]:
assumes $B \Longrightarrow$ *partial-equivalence-rel* R
shows *partial-equivalence-rel* (*rel-if* B R)
⟨*proof*⟩

end

context *galois-prop*
begin

interpretation *rel-if* : *galois-prop rel-if* B $(\leq_L)$ *rel-if* B' $(\leq_R)$ l r ⟨*proof*⟩
interpretation *flip-inv* : *galois-prop* $(\geq_R)$ $(\geq_L)$ r l ⟨*proof*⟩

lemma *rel-if-half-galois-prop-left-if-iff-if-half-galois-prop-leftI*:
assumes $B \Longrightarrow B' \Longrightarrow ((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$
and $B \longleftrightarrow B'$
shows $((\textit{rel-if}\ B\ (\leq_L))\ {}_h\trianglelefteq\ (\textit{rel-if}\ B'\ (\leq_R)))\ l\ r$
⟨*proof*⟩

lemma *rel-if-half-galois-prop-right-if-iff-if-half-galois-prop-rightI*:
assumes $B \Longrightarrow B' \Longrightarrow ((\leq_L)\ \trianglelefteq_h\ (\leq_R))\ l\ r$
and $B \longleftrightarrow B'$
shows $((\textit{rel-if}\ B\ (\leq_L))\ \trianglelefteq_h\ (\textit{rel-if}\ B'\ (\leq_R)))\ l\ r$
⟨*proof*⟩

lemma *rel-if-galois-prop-if-iff-if-galois-propI*:
assumes $B \Longrightarrow B' \Longrightarrow ((\leq_L)\ \trianglelefteq\ (\leq_R))\ l\ r$
and $B \longleftrightarrow B'$
shows $((\textit{rel-if}\ B\ (\leq_L))\ \trianglelefteq\ (\textit{rel-if}\ B'\ (\leq_R)))\ l\ r$
⟨*proof*⟩

end

context *galois*
begin

interpretation *rel-if* : *galois rel-if* B $(\leq_L)$ *rel-if* B' $(\leq_R)$ l r ⟨*proof*⟩

lemma *rel-if-galois-connection-if-iff-if-galois-connectionI*:
assumes $B \Longrightarrow B' \Longrightarrow ((\leq_L) \dashv (\leq_R))\ l\ r$
and $B \longleftrightarrow B'$
shows $((\textit{rel-if}\ B\ (\leq_L)) \dashv (\textit{rel-if}\ B'\ (\leq_R)))\ l\ r$
⟨*proof*⟩

lemma *rel-if-galois-equivalence-if-iff-if-galois-equivalenceI*:
assumes $B \Longrightarrow B' \Longrightarrow ((\leq_L) \equiv_G (\leq_R))\ l\ r$
and $B \longleftrightarrow B'$
shows $((\textit{rel-if}\ B\ (\leq_L)) \equiv_G (\textit{rel-if}\ B'\ (\leq_R)))\ l\ r$
⟨*proof*⟩

end

context *transport*
begin

interpretation *rel-if* : *transport rel-if B* $(\leq_L)$ *rel-if B'* $(\leq_R)$ *l r* ⟨*proof*⟩

lemma *rel-if-preorder-equivalence-if-iff-if-preorder-equivalenceI*:
assumes $B \Longrightarrow B' \Longrightarrow ((\leq_L) \equiv_{pre} (\leq_R))\ l\ r$
and $B \longleftrightarrow B'$
shows $((\textit{rel-if}\ B\ (\leq_L)) \equiv_{pre} (\textit{rel-if}\ B'\ (\leq_R)))\ l\ r$
⟨*proof*⟩

lemma *rel-if-partial-equivalence-rel-equivalence-if-iff-if-partial-equivalence-rel-equivalenceI*:
assumes $B \Longrightarrow B' \Longrightarrow ((\leq_L) \equiv_{PER} (\leq_R))\ l\ r$
and $B \longleftrightarrow B'$
shows $((\textit{rel-if}\ B\ (\leq_L)) \equiv_{PER} (\textit{rel-if}\ B'\ (\leq_R)))\ l\ r$
⟨*proof*⟩

end

locale *transport-Dep-Fun-Rel-no-dep-fun* =
transport-Dep-Fun-Rel-syntax L1 R1 l1 r1 L2 R2 λ- -. *l2* λ- -. *r2* +
tdfr : *transport-Dep-Fun-Rel L1 R1 l1 r1 L2 R2* λ- -. *l2* λ- -. *r2*
for *L1* :: *'a1* ⇒ *'a1* ⇒ *bool*
and *R1* :: *'a2* ⇒ *'a2* ⇒ *bool*
and *l1* :: *'a1* ⇒ *'a2*
and *r1* :: *'a2* ⇒ *'a1*
and *L2* :: *'a1* ⇒ *'a1* ⇒ *'b1* ⇒ *'b1* ⇒ *bool*
and *R2* :: *'a2* ⇒ *'a2* ⇒ *'b2* ⇒ *'b2* ⇒ *bool*
and *l2* :: *'b1* ⇒ *'b2*
and *r2* :: *'b2* ⇒ *'b1*
begin

notation *t2.unit* (‹η_2›)
notation *t2.counit* (‹ε_2›)

abbreviation $L \equiv tdfr.L$
abbreviation $R \equiv tdfr.R$

abbreviation $l \equiv tdfr.l$
abbreviation $r \equiv tdfr.r$

notation $tdfr.L$ (**infix** $\langle \leq_L \rangle$ *50*)
notation $tdfr.R$ (**infix** $\langle \leq_R \rangle$ *50*)

notation *tdfr.ge-left* (**infix** $\langle \geq_L \rangle$ *50*)
notation *tdfr.ge-right* (**infix** $\langle \geq_R \rangle$ *50*)

notation *tdfr.unit* ($\langle \eta \rangle$)
notation *tdfr.counit* ($\langle \varepsilon \rangle$)

theorem *partial-equivalence-rel-equivalenceI*:
assumes *per-equiv1*: $((\leq_{L1}) \equiv_{PER} (\leq_{R1}))$ *l1 r1*
and *per-equiv2*: $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_{PER} (\leq_{R2\ (l1\ x)\ x'}))$ *l2 r2*
and $((x1\ x2 :: (\geq_{L1})) \Rightarrow (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rrightarrow (\leq))$ *L2*
and $((x1'\ x2' :: (\geq_{R1})) \Rightarrow (x3'\ x4' :: (\leq_{R1}) \mid x1' \leq_{R1} x3') \Rrightarrow (\leq))$ *R2*
shows $((\leq_L) \equiv_{PER} (\leq_R))$ *l r*
⟨*proof*⟩

end

end

2.22 Transport via Equivalences on PERs (Prototype)

theory *Transport-Prototype*
imports
Transport-Rel-If
ML-Unification.ML-Unification-HOL-Setup
ML-Unification.Unify-Resolve-Tactics
keywords *trp-term* :: *thy-goal-defn*
begin

Summary We implement a simple Transport prototype. The prototype is restricted to work with equivalences on partial equivalence relations. It is also not forming the compositions of equivalences so far. The support for dependent function relators is restricted to the form described in ⟦*transport.partial-equivalence-rel-equivalence ?L1.0 ?R1.0 ?l1.0 ?r1.0*; $\bigwedge x\ x'$. *galois-rel.Galois ?L1.0 ?R1.0 ?r1.0 x x'* ⟹ *transport.partial-equivalence-rel-equivalence (?L2.0 x (?r1.0 x')) (?R2.0 (?l1.0 x) x') ?l2.0 ?r2.0*; $((x1\ x2 :: ?L1.0^{-1})$

$\Rightarrow (x3\ x4 :: ?L1.0) \Rrightarrow ?L1.0\ x1\ x3 \longrightarrow (\leq))\ ?L2.0; ((x1'\ x2' :: ?R1.0^{-1}) \Rightarrow$
$(x3'\ x4' :: ?R1.0) \Rrightarrow ?R1.0\ x1'\ x3' \longrightarrow (\leq))\ ?R2.0]\!] \Longrightarrow$ *transport.partial-equivalence-rel-equivalence*
(*transport-Dep-Fun-Rel.L ?L1.0 ?L2.0*) (*transport-Dep-Fun-Rel.L ?R1.0 ?R2.0*)
(*transport-Dep-Fun-Rel.l ?r1.0* (λ- -. *?l2.0*)) (*transport-Dep-Fun-Rel.l ?l1.0*
(λ- -. *?r2.0*)): The relations can be dependent, but the functions must be simple. This is not production ready, but a proof of concept.

The package provides a command **trp-term**, which sets up the required goals to prove a given term. See the examples in this directory for some use cases and refer to [2] for more details.

Theorem Setups **context** *transport*
begin

lemma *left-Galois-left-if-left-rel-if-partial-equivalence-rel-equivalence*:
assumes $((\leq_L) \equiv_{PER} (\leq_R))\ l\ r$
and $x \leq_L x'$
shows $x\ {}_L\lessapprox l\ x$
⟨*proof*⟩

definition *transport-per x y* $\equiv ((\leq_L) \equiv_{PER} (\leq_R))\ l\ r \wedge x\ {}_L\lessapprox y$

The choice of x' is arbitrary. All we need is *in-dom* $(\leq_L)$ *x*.

lemma *transport-per-start*:
assumes $((\leq_L) \equiv_{PER} (\leq_R))\ l\ r$
and $x \leq_L x'$
shows *transport-per x* $(l\ x)$
⟨*proof*⟩

lemma *left-Galois-if-transport-per*:
assumes *transport-per x y*
shows $x\ {}_L\lessapprox y$
⟨*proof*⟩

end

context *transport-Fun-Rel*
begin

Simplification of Galois relator for simple function relator.

corollary *left-Galois-eq-Fun-Rel-left-Galois*:
assumes $((\leq_{L1}) \equiv_{PER} (\leq_{R1}))$ *l1 r1*
and $((\leq_{L2}) \equiv_{PER} (\leq_{R2}))$ *l2 r2*
shows $({}_L\lessapprox) = (({}_{L1}\lessapprox) \Rrightarrow ({}_{L2}\lessapprox))$
⟨*proof*⟩

end

lemmas *related-Fun-Rel-combI* = *Fun-Rel-relD*[*rotated*]

lemma *related-Fun-Rel-lambdaI*:
assumes $\bigwedge x\ y.\ R\ x\ y \Longrightarrow S\ (f\ x)\ (g\ y)$
and $T = (R \Rightarrow S)$
shows $T\ f\ g$
⟨*proof*⟩

General ML setups ⟨*ML*⟩

Unification Setup ⟨*ML*⟩
declare [[*trp-uhint* **where** *hint-preprocessor* = ‹*Unification-Hints-Base.obj-logic-hint-preprocessor*
@{*thm atomize-eq*[*symmetric*]} (*Conv.rewr-conv* @{*thm eq-eq-True*})›]]
declare [[*trp-ucombine add* = ‹*Transport-Unification-Combine.eunif-data*
(*Transport-Unification-Hints.try-hints*
|> *Unification-Combinator.norm-unifier*
(*Unification-Util.inst-norm-term′*
Transport-Mixed-Unification.norms-first-higherp-decomp-comb-higher-unify)
|> *K*)
(*Transport-Unification-Combine.default-metadata Transport-Unification-Hints.binding*)›]]

Prototype ⟨*ML*⟩

declare
transport-Dep-Fun-Rel.transport-defs[*trp-def*]
transport-Fun-Rel.transport-defs[*trp-def*]

declare

transport-Fun-Rel.partial-equivalence-rel-equivalenceI[*rotated*, *per-intro*]
transport-eq-id.partial-equivalence-rel-equivalenceI[*per-intro*]
transport-eq-restrict-id.partial-equivalence-rel-equivalence[*per-intro*]

declare
transport-id.left-Galois-eq-left[*trp-relator-rewrite*]
transport-Fun-Rel.left-Galois-eq-Fun-Rel-left-Galois[*trp-relator-rewrite*]

end

2.23 Syntax Bundles for Transport

theory *Transport-Syntax*
imports
Transport-Base
begin

abbreviation *Galois-infix x L R r y* ≡ *galois-rel.Galois L R r x y*
abbreviation (*input*) *ge-Galois R r L* ≡ *galois-rel.ge-Galois-left L R r*
abbreviation (*input*) *ge-Galois-infix y R r L x* ≡ *ge-Galois R r L y x*

bundle *galois-rel-syntax*
begin
notation *galois-rel.Galois* (‹'((-)≲≈(-) (-))'›)
notation *Galois-infix* (‹(-) (-)≲≈(-) (-)) (-)› [*51*,*51*,*51*,*51*,*51*] *50*)
notation *ge-Galois* (‹'((-) (-)≳≈(-)')›)
notation *ge-Galois-infix* (‹(-) ((-) (-))≳≈(-) (-)› [*51*,*51*,*51*,*51*,*51*] *50*)
end

bundle *transport-syntax*
begin
notation *transport.preorder-equivalence* (**infix** ‹≡$_{pre}$› *50*)
notation *transport.partial-equivalence-rel-equivalence* (**infix** ‹≡$_{PER}$› *50*)
end

end

2.24 Example Transports for Dependent Function Relator

theory *Transport-Dep-Fun-Rel-Examples*
imports
Transport-Prototype
Transport-Syntax
HOL-Alignment-Binary-Relations
HOL−Library.IArray
begin

Summary Dependent function relator examples from [2]. Refer to the paper for more details.

context
includes *galois-rel-syntax* **and** *transport-syntax*
notes
transport.rel-if-partial-equivalence-rel-equivalence-if-iff-if-partial-equivalence-rel-equivalenceI
[*rotated*, *per-intro*]
transport-Dep-Fun-Rel-no-dep-fun.partial-equivalence-rel-equivalenceI
[*ML-Krattr* ‹*Drule.rearrange-prems* [*1*] #> *Drule.rearrange-prems* [*2*,*3*]›, *per-intro*]
begin

interpretation *transport L R l r* **for** *L R l r* ⟨*proof*⟩

abbreviation *Zpos* ≡ ((=$_{(\leq)(0\ ::\ int)}$) :: *int* ⇒ -)

lemma *Zpos-per* [*per-intro*]: (*Zpos* $\equiv_{PER}$ (=)) *nat int*
⟨*proof*⟩

lemma *sub-parametric* [*trp-in-dom*]:
((*i* - :: *Zpos*) ⇛ (*j* - :: *Zpos* | *j* ≤ *i*) ⇛ *Zpos*) (−) (−)
⟨*proof*⟩

trp-term *nat-sub* :: *nat* ⇒ *nat* ⇒ *nat* **where** *x* = (−) :: *int* ⇒ -
and *L* = (*i* - :: *Zpos*) ⇛ (*j* - :: *Zpos* | *j* ≤ *i*) ⇛ *Zpos*
and *R* = (*n* - :: (=)) ⇛ (*m* - :: (=)| *m* ≤ *n*) ⇛ (=)

⟨*proof*⟩

thm *nat-sub-app-eq*

Note: as of now, **trp-term** does not rewrite the Galois relator of dependent function relators.

thm *nat-sub-related′*

abbreviation *LRel* ≡ *list-all2*
abbreviation *IARel* ≡ *rel-iarray*

lemma [*per-intro*]:
assumes *partial-equivalence-rel R*
shows (*LRel R* $\equiv_{PER}$ *IARel R*) *IArray.IArray IArray.list-of*
⟨*proof*⟩

lemma [*trp-in-dom*]:
((*xs* - :: *LRel R*) ⇛ (*i* - :: (=) | *i* < *length xs*) ⇛ *R*) (!) (!)
⟨*proof*⟩

context
fixes *R* :: *′a* ⇒ *′a* ⇒ *bool* **assumes** [*per-intro*]: *partial-equivalence-rel R*
begin

interpretation *Rper* : *transport-partial-equivalence-rel-id R*
⟨*proof*⟩

declare *Rper.partial-equivalence-rel-equivalence* [*per-intro*]

trp-term *iarray-index* **where** *x* = (!) :: *′a list* ⇒ -
and *L* = ((*xs* - :: *LRel R*) ⇛ (*i* - :: (=) | *i* < *length xs*) ⇛ *R*)
and *R* = ((*xs* - :: *IARel R*) ⇛ (*i* - :: (=) | *i* < *IArray.length xs*) ⇛ *R*)
⟨*proof*⟩

end
end

end

2.25 Example Transports Between Lists and Sets

theory *Transport-Lists-Sets-Examples*
imports
Transport-Prototype
Transport-Syntax
HOL−Library.FSet
begin

Summary Introductory examples from [2]. Transports between lists and (finite) sets. Refer to the paper for more details.

context
includes *galois-rel-syntax* **and** *transport-syntax*
begin

Introductory examples from paper Left and right relations.

definition *LFSL xs xs′* ≡ *fset-of-list xs* = *fset-of-list xs′*
abbreviation (*input*) (*LFSR* :: *′a fset* ⇒ -) ≡ (=)
definition *LSL xs xs′* ≡ *set xs* = *set xs′*
abbreviation (*input*) (*LSR* :: *′a set* ⇒ -) ≡ (=$_{\mathit{finite}\ ::\ 'a\ \mathit{set}\ \Rightarrow\ \mathit{bool}}$)

interpretation *t* : *transport LSL R l r* **for** *LSL R l r* ⟨*proof*⟩

Proofs of equivalences.

lemma *list-fset-PER* [*per-intro*]: (*LFSL* ≡$_{PER}$ *LFSR*) *fset-of-list sorted-list-of-fset*
⟨*proof*⟩

lemma *list-set-PER* [*per-intro*]: (*LSL* ≡$_{PER}$ *LSR*) *set sorted-list-of-set*
⟨*proof*⟩

We can rewrite the Galois relators in the following theorems to the relator of the paper.

definition *LFS xs s* ≡ *fset-of-list xs* = *s*
definition *LS xs s* ≡ *set xs* = *s*

lemma *LFSL-Galois-eq-LFS*: ($_{LFSL}\overset{\leqslant}{\approx}_{LFSR\ \mathit{sorted\text{-}list\text{-}of\text{-}fset}}$) ≡ *LFS*
⟨*proof*⟩
lemma *LFSR-Galois-eq-inv-LFS*: ($_{LFSR}\overset{\leqslant}{\approx}_{LFSL\ \mathit{fset\text{-}of\text{-}list}}$) ≡ LFS^{-1}
⟨*proof*⟩
lemma *LSL-Galois-eq-LS*: ($_{LSL}\overset{\leqslant}{\approx}_{LSR\ \mathit{sorted\text{-}list\text{-}of\text{-}set}}$) ≡ *LS*
⟨*proof*⟩

declare *LFSL-Galois-eq-LFS*[*trp-relator-rewrite*, *trp-uhint*]
LFSR-Galois-eq-inv-LFS[*trp-relator-rewrite*, *trp-uhint*]
LSL-Galois-eq-LS[*trp-relator-rewrite*, *trp-uhint*]

definition *max-list xs* ≡ *foldr max xs* (*0* :: *nat*)

Proof of parametricity for *max-list*.

lemma *max-max-list-removeAll-eq-maxlist*:
assumes $x \in$ *set xs*
shows *max x* (*max-list* (*removeAll x xs*)) = *max-list xs*
⟨*proof*⟩

lemma *max-list-parametric* [*trp-in-dom*]: (*LSL* ⇛ (=)) *max-list max-list*
⟨*proof*⟩

lemma *LFSL-eq-LSL*: *LFSL* ≡ *LSL*
⟨*proof*⟩

lemma *max-list-parametricfin* [*trp-in-dom*]: (*LFSL* ⇛ (=)) *max-list max-list*
⟨*proof*⟩

Transport from lists to finite sets.

trp-term *max-fset* :: *nat fset* ⇒ *nat* **where** *x* = *max-list*
and *L* = (*LFSL* ⇛ (=))
⟨*proof*⟩

Use **print-theorems** to show all theorems. Here's the correctness theorem:

lemma (*LFS* ⇛ (=)) *max-list max-fset* ⟨*proof*⟩

lemma [*trp-in-dom*]: (*LFSR* ⇛ (=)) *max-fset max-fset* ⟨*proof*⟩

Transport from lists to sets.

trp-term *max-set* :: *nat set* ⇒ *nat* **where** *x* = *max-list*
⟨*proof*⟩

lemma (*LS* ⇛ (=)) *max-list max-set* ⟨*proof*⟩

The registration of symmetric equivalence rules is not done by default as of now, but that would not be a problem in principle.

lemma *list-fset-PER-sym* [*per-intro*]:
(*LFSR* $\equiv_{PER}$ *LFSL*) *sorted-list-of-fset fset-of-list*
⟨*proof*⟩

Transport from finite sets to lists.

trp-term *max-list′* :: *nat list* ⇒ *nat* **where** *x* = *max-fset*
⟨*proof*⟩

lemma (LFS^{-1} ⇛ (=)) *max-fset max-list′* ⟨*proof*⟩

Transporting higher-order functions.

lemma *map-parametric* [*trp-in-dom*]:
(((=) ⇛ (=)) ⇛ *LSL* ⇛ *LSL*) *map map*
⟨*proof*⟩

lemma [*trp-uhint*]: $P \equiv (=) \Longrightarrow P \equiv (=) \Rrightarrow (=)$ ⟨*proof*⟩
lemma [*trp-uhint*]: $P \equiv \top \Longrightarrow (=_{P\ ::\ 'a \Rightarrow bool}) \equiv ((=)\ ::\ 'a \Rightarrow \text{-})$ ⟨*proof*⟩

trp-term *map-set* :: $('a ::$ *linorder* $\Rightarrow\ 'b) \Rightarrow\ 'a$ *set* $\Rightarrow ('b ::$ *linorder*$)$ *set*
  where $x =$ *map* :: $('a ::$ *linorder* $\Rightarrow\ 'b) \Rightarrow\ 'a$ *list* $\Rightarrow ('b ::$ *linorder*$)$ *list*
  ⟨*proof*⟩

lemma $(((=) \Rrightarrow (=)) \Rrightarrow LS \Rrightarrow LS)$ *map map-set* ⟨*proof*⟩

lemma *filter-parametric* [*trp-in-dom*]:
  $(((=) \Rrightarrow (\longleftrightarrow)) \Rrightarrow LSL \Rrightarrow LSL)$ *filter filter*
  ⟨*proof*⟩

trp-term *filter-set* :: $('a ::$ *linorder* $\Rightarrow$ *bool*$) \Rightarrow\ 'a$ *set* $\Rightarrow\ 'a$ *set*
  where $x =$ *filter* :: $('a ::$ *linorder* $\Rightarrow$ *bool*$) \Rightarrow\ 'a$ *list* $\Rightarrow\ 'a$ *list*
  ⟨*proof*⟩

lemma $(((=) \Rrightarrow (=)) \Rrightarrow LS \Rrightarrow LS)$ *filter filter-set* ⟨*proof*⟩

lemma *append-parametric* [*trp-in-dom*]:
  $(LSL \Rrightarrow LSL \Rrightarrow LSL)$ *append append*
  ⟨*proof*⟩

trp-term *append-set* :: $('a ::$ *linorder*$)$ *set* $\Rightarrow\ 'a$ *set* $\Rightarrow\ 'a$ *set*
  where $x =$ *append* :: $('a ::$ *linorder*$)$ *list* $\Rightarrow\ 'a$ *list* $\Rightarrow\ 'a$ *list*
  ⟨*proof*⟩

lemma $(LS \Rrightarrow LS \Rrightarrow LS)$ *append append-set* ⟨*proof*⟩

The prototype also provides a simplified definition.

lemma *append-set s s'* $\equiv$ *set* (*sorted-list-of-set s*) $\cup$ *set* (*sorted-list-of-set s'*)
  ⟨*proof*⟩

lemma *finite s* $\Longrightarrow$ *finite s'* $\Longrightarrow$ *append-set s s'* $= s \cup s'$
  ⟨*proof*⟩

end

end

2.26 Transport for Partial Quotient Types

theory *Transport-Partial-Quotient-Types*
  imports
    HOL.Lifting

Transport
begin

Summary Every partial quotient type *Quotient*, as used by the Lifting package, is transportable.

context *transport*
begin

interpretation t : *transport* L $(=)$ l r ⟨*proof*⟩

lemma *Quotient-T-eq-Galois*:
  assumes *Quotient* $(\leq_L)$ l r T
  shows T = *t.Galois*
⟨*proof*⟩

lemma *Quotient-if-preorder-equivalence*:
  assumes $((\leq_L) \equiv_{pre} (=))$ l r
  shows *Quotient* $(\leq_L)$ l r *t.Galois*
⟨*proof*⟩

lemma *partial-equivalence-rel-equivalence-if-Quotient*:
  assumes *Quotient* $(\leq_L)$ l r T
  shows $((\leq_L) \equiv_{PER} (=))$ l r
⟨*proof*⟩

corollary *Quotient-iff-partial-equivalence-rel-equivalence*:
  Quotient $(\leq_L)$ l r *t.Galois* $\longleftrightarrow$ $((\leq_L) \equiv_{PER} (=))$ l r
  ⟨*proof*⟩

corollary *Quotient-T-eq-ge-Galois-right*:
  assumes *Quotient* $(\leq_L)$ l r T
  shows T = *t.ge-Galois-right*
  ⟨*proof*⟩

end

end

2.27 Transport for HOL Type Definitions

theory *Transport-Typedef-Base*
  imports
    HOL-Alignment-Binary-Relations
    Transport-Bijections
    HOL.Typedef
begin

context *type-definition*
begin

abbreviation (*input*) $L :: 'a \Rightarrow 'a \Rightarrow bool \equiv (=_A)$
abbreviation (*input*) $R :: 'b \Rightarrow 'b \Rightarrow bool \equiv (=)$

sublocale *transport?* :
transport-eq-restrict-bijection mem-of A $\top :: 'b \Rightarrow bool$ *Abs Rep*
rewrites $(=_{mem\text{-}of\ A}) \equiv L$
and $(=_{\top :: 'b \Rightarrow bool}) \equiv R$
and $(galois\text{-}rel.Galois\ (=)\ (=)\ Rep)\restriction_{mem\text{-}of\ A}\restriction\top :: 'b \Rightarrow bool \equiv$
$(galois\text{-}rel.Galois\ (=)\ (=)\ Rep)$
⟨*proof*⟩

interpretation *galois L R Abs Rep* ⟨*proof*⟩

lemma *Rep-left-Galois-self*: $Rep\ y\ {}_L\lessapprox y$
⟨*proof*⟩

definition $AR\ x\ y \equiv x = Rep\ y$

lemma *left-Galois-eq-AR*: *left-Galois* $= AR$
⟨*proof*⟩

end

end

theory *Transport-Typedef*
imports
HOL−Library.FSet
Transport-Typedef-Base
Transport-Prototype
Transport-Syntax
HOL-Alignment-Functions
begin

context
includes *galois-rel-syntax* **and** *transport-syntax*
begin

typedef $pint = \{i :: int.\ 0 \le i\}$ ⟨*proof*⟩

interpretation *typedef-pint* : *type-definition Rep-pint Abs-pint* $\{i :: int.\ 0 \le i\}$
⟨*proof*⟩

lemma [*trp-relator-rewrite*, *trp-uhint*]:
$(=_{Collect\ ((\le)\ (0 :: int))})\lessapprox(=)\ Rep\text{-}pint \equiv typedef\text{-}pint.AR$

⟨*proof*⟩

typedef *'a fset* = {*s* :: *'a set*. *finite s*} ⟨*proof*⟩

interpretation *typedef-fset* :
type-definition Rep-fset Abs-fset {*s* :: *'a set*. *finite s*}
⟨*proof*⟩

lemma [*trp-relator-rewrite*, *trp-uhint*]:
$(=_{\{s :: 'a\ set.\ finite\ s\}}) :: 'a\ set \Rightarrow \text{-} \overset{<}{\approx} (=)\ Rep\text{-}fset) \equiv typedef\text{-}fset.AR$
⟨*proof*⟩

lemma *eq-restrict-set-eq-eq-uhint* [*trp-uhint*]:
$(\bigwedge x.\ P\ x \equiv x \in A) \Longrightarrow ((=_{A :: 'a\ set}) :: 'a \Rightarrow \text{-}) \equiv (=_P)$
⟨*proof*⟩

declare
typedef-pint.partial-equivalence-rel-equivalence[*per-intro*]
typedef-fset.partial-equivalence-rel-equivalence[*per-intro*]

lemma *one-parametric* [*trp-in-dom*]: *typedef-pint.L 1 1* ⟨*proof*⟩

trp-term *pint-one* :: *pint* **where** *x* = *1* :: *int*
⟨*proof*⟩

lemma *add-parametric* [*trp-in-dom*]:
(*typedef-pint.L* ⇛ *typedef-pint.L* ⇛ *typedef-pint.L*) (+) (+)
⟨*proof*⟩

trp-term *pint-add* :: *pint* ⇒ *pint* ⇒ *pint*
where *x* = (+) :: *int* ⇒ -
⟨*proof*⟩

lemma *empty-parametric* [*trp-in-dom*]: *typedef-fset.L* {} {}
⟨*proof*⟩

trp-term *fempty* :: *'a fset* **where** *x* = {} :: *'a set*
⟨*proof*⟩

lemma *insert-parametric* [*trp-in-dom*]:
((=) ⇛ *typedef-fset.L* ⇛ *typedef-fset.L*) *insert insert*
⟨*proof*⟩

trp-term *finsert* :: *'a* ⇒ *'a fset* ⇒ *'a fset* **where** *x* = *insert*
and *L* = ((=) ⇛ *typedef-fset.L* ⇛ *typedef-fset.L*)
and *R* = ((=) ⇛ *typedef-fset.R* ⇛ *typedef-fset.R*)

⟨*proof*⟩

context
notes *refl*[*trp-related-intro*]
begin

trp-term *insert-add-int-fset-whitebox* :: *int fset*
where $x = insert\ (1 + (1 :: int))\ \{\}$!
⟨*proof*⟩

lemma *empty-parametric'* [*trp-related-intro*]: (*rel-set R*) {} {}
⟨*proof*⟩

lemma *insert-parametric'* [*trp-related-intro*]:
(*R* ⇛ *rel-set R* ⇛ *rel-set R*) *insert insert*
⟨*proof*⟩

context
assumes [*trp-uhint*]:

$L \equiv rel\text{-}set\ (L1 :: int \Rightarrow int \Rightarrow bool) \Longrightarrow R \equiv rel\text{-}set\ (R1 :: pint \Rightarrow pint \Rightarrow bool)$
$\Longrightarrow r \equiv image\ r1 \Longrightarrow S \equiv ({}_{L1}\lessapprox_{R1\ r1}) \Longrightarrow ({}_{L}\lessapprox_{R\ r}) \equiv rel\text{-}set\ S$
begin

trp-term *insert-add-pint-set-whitebox* :: *pint set*
where $x = insert\ (1 + (1 :: int))\ \{\}$!
⟨*proof*⟩

print-statement *insert-add-int-fset-whitebox-def insert-add-pint-set-whitebox-def*

end
end

lemma *image-parametric* [*trp-in-dom*]:
(((=) ⇛ (=)) ⇛ *typedef-fset.L* ⇛ *typedef-fset.L*) *image image*
⟨*proof*⟩

trp-term *fimage* :: ('*a* ⇒ '*b*) ⇒ '*a fset* ⇒ '*b fset* **where** *x* = *image*
⟨*proof*⟩

lemma *image-parametric'* [*trp-related-intro*]:
((*R* ⇛ *S*) ⇛ *rel-set R* ⇛ *rel-set S*) *image image*
⟨*proof*⟩

lemma *Galois-id-hint* [*trp-uhint*]:
(*L* :: *'a* ⇒ *'a* ⇒ *bool*) ≡ *R* ⟹ *r* ≡ *id* ⟹ *E* ≡ *L* ⟹ (*L*≲≈*R r*) ≡ *E*
⟨*proof*⟩

lemma *Freq* [*trp-uhint*]: *L* ≡ (=) ⇛ (=) ⟹ *L* ≡ (=)
⟨*proof*⟩

context
fixes *L1 R1 l1 r1 L R l r*
assumes *per1*: (*L1* ≡*PER R1*) *l1 r1*
defines *L* ≡ *rel-set L1* **and** *R* ≡ *rel-set R1*
and *l* ≡ *image l1* **and** *r* ≡ *image r1*
begin

interpretation *transport L R l r* ⟨*proof*⟩

context

assumes *transport-per-set*: ((≤*L*) ≡*PER* (≤*R*)) *l r*
and *compat*: *transport-comp.middle-compatible-codom R typedef-fset.L*
begin

trp-term *fempty-param* :: *'b fset*
where *x* = {} :: *'a set*
and *L* = *transport-comp.L ?L1 ?R1* (*?l1* :: *'a set* ⇒ *'b set*) *?r1 typedef-fset.L*
and *R* = *transport-comp.L typedef-fset.R typedef-fset.L ?r2 ?l2 ?R1*
⟨*proof*⟩

definition *set-succ* ≡ *image* ((+) (*1* :: *int*))

lemma *set-succ-parametric* [*trp-in-dom*]:
(*typedef-fset.L* ⇛ *typedef-fset.L*) *set-succ set-succ*
⟨*proof*⟩

trp-term *fset-succ* :: *int fset* ⇒ *int fset*
where *x* = *set-succ*
and *L* = *typedef-fset.L* ⇛ *typedef-fset.L*
and *R* = *typedef-fset.R* ⇛ *typedef-fset.R*
⟨*proof*⟩

trp-term *fset-succ'* :: *int fset* ⇒ *int fset*
where *x* = *set-succ*
and *L* = *typedef-fset.L* ⇛ *typedef-fset.L*
and *R* = *typedef-fset.R* ⇛ *typedef-fset.R*
unfold set-succ-def !

⟨*proof*⟩

lemma *pint-middle-compat*:

transport-comp.middle-compatible-codom (*rel-set* ((=) :: *pint* ⇒ -))
(=*Collect* (*finite* :: *pint set* ⇒ -))
⟨*proof*⟩

trp-term *pint-fset-succ* :: *pint fset* ⇒ *pint fset*
where x = *set-succ* :: *int set* ⇒ *int set*

⟨*proof*⟩

end
end
end

end

2.28 Transport for Natural Functors

2.28.1 Basic Setup

theory *Transport-Natural-Functors-Base*
imports
HOL.BNF-Def
HOL-Alignment-Functions
Transport-Base
begin

Summary Basic setup for closure proofs and simple lemmas.

In the following, we willingly use granular apply-style proofs since, in practice, these theorems have to be automatically generated whenever we declare a new natural functor.

Note that "HOL-Library" provides a command *bnf-axiomatization* which allows one to axiomatically declare a bounded natural functor. However, we only need a subset of these axioms - the boundedness of the functor is irrelevant for our purposes. For this reason - and the sake of completeness - we state all the required axioms explicitly below.

lemma *Grp-UNIV-eq-eq-comp*: *BNF-Def.Grp UNIV f* = (=) ∘ *f*
⟨*proof*⟩

lemma *eq-comp-rel-comp-eq-comp*: (=) ∘ *f* ∘∘ *R* = *R* ∘ *f*
⟨*proof*⟩

lemma *Domain-Collect-case-prod-eq-Collect-in-dom*:
Domain {(*x*, *y*). *R x y*} = {*x*. *in-dom R x*}
⟨*proof*⟩

lemma *ball-in-dom-iff-ball-ex*:
(∀ *x* ∈ *S*. *in-dom R x*) ⟷ (∀ *x* ∈ *S*. ∃ *y*. *R x y*)

⟨*proof*⟩

lemma *pair-mem-Collect-case-prod-iff*: $(x, y) \in \{(x, y).\ R\ x\ y\} \longleftrightarrow R\ x\ y$
⟨*proof*⟩

Natural Functor Axiomatisation **typedecl** $('d, 'a, 'b, 'c)\ F$

consts $Fmap :: ('a1 \Rightarrow 'a2) \Rightarrow ('b1 \Rightarrow 'b2) \Rightarrow ('c1 \Rightarrow 'c2) \Rightarrow$
$('d, 'a1, 'b1, 'c1)\ F \Rightarrow ('d, 'a2, 'b2, 'c2)\ F$
$Fset1 :: ('d, 'a, 'b, 'c)\ F \Rightarrow 'a\ set$
$Fset2 :: ('d, 'a, 'b, 'c)\ F \Rightarrow 'b\ set$
$Fset3 :: ('d, 'a, 'b, 'c)\ F \Rightarrow 'c\ set$

axiomatization
where *Fmap-id*: $Fmap\ id\ id\ id = id$
and *Fmap-comp*: $\bigwedge f1\ f2\ f3\ g1\ g2\ g3.$
$Fmap\ (g1 \circ f1)\ (g2 \circ f2)\ (g3 \circ f3) = Fmap\ g1\ g2\ g3 \circ Fmap\ f1\ f2\ f3$
and *Fmap-cong*: $\bigwedge f1\ f2\ f3\ g1\ g2\ g3\ x.$
$(\bigwedge x1.\ x1 \in Fset1\ x \Longrightarrow f1\ x1 = g1\ x1) \Longrightarrow$
$(\bigwedge x2.\ x2 \in Fset2\ x \Longrightarrow f2\ x2 = g2\ x2) \Longrightarrow$
$(\bigwedge x3.\ x3 \in Fset3\ x \Longrightarrow f3\ x3 = g3\ x3) \Longrightarrow$
$Fmap\ f1\ f2\ f3\ x = Fmap\ g1\ g2\ g3\ x$
and *Fset1-natural*: $\bigwedge f1\ f2\ f3.\ Fset1 \circ Fmap\ f1\ f2\ f3 = image\ f1 \circ Fset1$
and *Fset2-natural*: $\bigwedge f1\ f2\ f3.\ Fset2 \circ Fmap\ f1\ f2\ f3 = image\ f2 \circ Fset2$
and *Fset3-natural*: $\bigwedge f1\ f2\ f3.\ Fset3 \circ Fmap\ f1\ f2\ f3 = image\ f3 \circ Fset3$

lemma *Fmap-id-eq-self*: $Fmap\ id\ id\ id\ x = x$
⟨*proof*⟩

lemma *Fmap-comp-eq-Fmap-Fmap*:
$Fmap\ (g1 \circ f1)\ (g2 \circ f2)\ (g3 \circ f3)\ x = Fmap\ g1\ g2\ g3\ (Fmap\ f1\ f2\ f3\ x)$
⟨*proof*⟩

lemma *Fset1-Fmap-eq-image-Fset1*: $Fset1\ (Fmap\ f1\ f2\ f3\ x) = f1\ `\ Fset1\ x$
⟨*proof*⟩

lemma *Fset2-Fmap-eq-image-Fset2*: $Fset2\ (Fmap\ f1\ f2\ f3\ x) = f2\ `\ Fset2\ x$
⟨*proof*⟩

lemma *Fset3-Fmap-eq-image-Fset3*: $Fset3\ (Fmap\ f1\ f2\ f3\ x) = f3\ `\ Fset3\ x$
⟨*proof*⟩

lemmas *Fset-Fmap-eqs* = *Fset1-Fmap-eq-image-Fset1 Fset2-Fmap-eq-image-Fset2*
Fset3-Fmap-eq-image-Fset3

Relator **definition** $Frel :: ('a1 \Rightarrow 'a2 \Rightarrow bool) \Rightarrow ('b1 \Rightarrow 'b2 \Rightarrow bool) \Rightarrow ('c1$
$\Rightarrow 'c2 \Rightarrow bool) \Rightarrow$
$('d, 'a1, 'b1, 'c1)\ F \Rightarrow ('d, 'a2, 'b2, 'c2)\ F \Rightarrow bool$
where $Frel\ R1\ R2\ R3\ x\ y \equiv (\exists z.$

$z \in \{x.\ Fset1\ x \subseteq \{(x,\ y).\ R1\ x\ y\} \wedge Fset2\ x \subseteq \{(x,\ y).\ R2\ x\ y\}$
$\wedge\ Fset3\ x \subseteq \{(x,\ y).\ R3\ x\ y\}\}$
$\wedge\ Fmap\ fst\ fst\ fst\ z = x$
$\wedge\ Fmap\ snd\ snd\ snd\ z = y)$

lemma *FrelI*:
assumes $Fset1\ z \subseteq \{(x,\ y).\ R1\ x\ y\}$
and $Fset2\ z \subseteq \{(x,\ y).\ R2\ x\ y\}$
and $Fset3\ z \subseteq \{(x,\ y).\ R3\ x\ y\}$
and $Fmap\ fst\ fst\ fst\ z = x$
and $Fmap\ snd\ snd\ snd\ z = y$
shows $Frel\ R1\ R2\ R3\ x\ y$
⟨*proof*⟩

lemma *FrelE*:
assumes $Frel\ R1\ R2\ R3\ x\ y$
obtains z **where** $Fset1\ z \subseteq \{(x,\ y).\ R1\ x\ y\}\ Fset2\ z \subseteq \{(x,\ y).\ R2\ x\ y\}$
$Fset3\ z \subseteq \{(x,\ y).\ R3\ x\ y\}\ Fmap\ fst\ fst\ fst\ z = x\ Fmap\ snd\ snd\ snd\ z = y$
⟨*proof*⟩

lemma *Grp-UNIV-Fmap-eq-Frel-Grp*: $BNF\text{-}Def.Grp\ UNIV\ (Fmap\ f1\ f2\ f3) =$
$Frel\ (BNF\text{-}Def.Grp\ UNIV\ f1)\ (BNF\text{-}Def.Grp\ UNIV\ f2)\ (BNF\text{-}Def.Grp\ UNIV\ f3)$
⟨*proof*⟩

lemma *Frel-Grp-UNIV-Fmap*:
$Frel\ (BNF\text{-}Def.Grp\ UNIV\ f1)\ (BNF\text{-}Def.Grp\ UNIV\ f2)\ (BNF\text{-}Def.Grp\ UNIV\ f3)$
$x\ (Fmap\ f1\ f2\ f3\ x)$
⟨*proof*⟩

lemma *Frel-Grp-UNIV-iff-eq-Fmap*:
$Frel\ (BNF\text{-}Def.Grp\ UNIV\ f1)\ (BNF\text{-}Def.Grp\ UNIV\ f2)\ (BNF\text{-}Def.Grp\ UNIV\ f3)\ x\ y \longleftrightarrow$
$(y = Fmap\ f1\ f2\ f3\ x)$
⟨*proof*⟩

lemma *Frel-eq*: $Frel\ (=)\ (=)\ (=) = (=)$
⟨*proof*⟩

corollary *Frel-eq-self*: $Frel\ (=)\ (=)\ (=)\ x\ x$
⟨*proof*⟩

lemma *Frel-mono-strong*:
assumes $Frel\ R1\ R2\ R3\ x\ y$
and $\bigwedge x1\ y1.\ x1 \in Fset1\ x \Longrightarrow y1 \in Fset1\ y \Longrightarrow R1\ x1\ y1 \Longrightarrow S1\ x1\ y1$
and $\bigwedge x2\ y2.\ x2 \in Fset2\ x \Longrightarrow y2 \in Fset2\ y \Longrightarrow R2\ x2\ y2 \Longrightarrow S2\ x2\ y2$
and $\bigwedge x3\ y3.\ x3 \in Fset3\ x \Longrightarrow y3 \in Fset3\ y \Longrightarrow R3\ x3\ y3 \Longrightarrow S3\ x3\ y3$
shows $Frel\ S1\ S2\ S3\ x\ y$

⟨*proof*⟩

corollary *Frel-mono*:
assumes *R1* ≤ *S1* *R2* ≤ *S2* *R3* ≤ *S3*
shows *Frel R1 R2 R3* ≤ *Frel S1 S2 S3*
⟨*proof*⟩

lemma *Frel-refl-strong*:
assumes ⋀*x1*. *x1* ∈ *Fset1 x* ⟹ *R1 x1 x1*
and ⋀*x2*. *x2* ∈ *Fset2 x* ⟹ *R2 x2 x2*
and ⋀*x3*. *x3* ∈ *Fset3 x* ⟹ *R3 x3 x3*
shows *Frel R1 R2 R3 x x*
⟨*proof*⟩

lemma *Frel-cong*:
assumes ⋀*x1 y1*. *x1* ∈ *Fset1 x* ⟹ *y1* ∈ *Fset1 y* ⟹ *R1 x1 y1* ⟷ *R1′ x1 y1*
and ⋀*x2 y2*. *x2* ∈ *Fset2 x* ⟹ *y2* ∈ *Fset2 y* ⟹ *R2 x2 y2* ⟷ *R2′ x2 y2*
and ⋀*x3 y3*. *x3* ∈ *Fset3 x* ⟹ *y3* ∈ *Fset3 y* ⟹ *R3 x3 y3* ⟷ *R3′ x3 y3*
shows *Frel R1 R2 R3 x y* = *Frel R1′ R2′ R3′ x y*
⟨*proof*⟩

lemma *Frel-rel-inv-eq-rel-inv-Frel*: *Frel R1*$^{-1}$ *R2*$^{-1}$ *R3*$^{-1}$ = (*Frel R1 R2 R3*)$^{-1}$
⟨*proof*⟩

Given the former axioms, the following axiom - subdistributivity of the relator - is equivalent to the (F, Fmap) functor preserving weak pullbacks.

axiomatization
where *Frel-comp-le-Frel-rel-comp*: ⋀*R1 R2 R3 S1 S2 S3*.
Frel R1 R2 R3 ∘∘ *Frel S1 S2 S3* ≤ *Frel* (*R1* ∘∘ *S1*) (*R2* ∘∘ *S2*) (*R3* ∘∘ *S3*)

lemma *fst-sndOp-eq-snd-fstOp*: *fst* ∘ *BNF-Def.sndOp P Q* = *snd* ∘ *BNF-Def.fstOp P Q*
⟨*proof*⟩

lemma *Frel-rel-comp-le-Frel-comp*:
Frel (*R1* ∘∘ *S1*) (*R2* ∘∘ *S2*) (*R3* ∘∘ *S3*) ≤ (*Frel R1 R2 R3* ∘∘ *Frel S1 S2 S3*)
⟨*proof*⟩

corollary *Frel-comp-eq-Frel-rel-comp*:
Frel R1 R2 R3 ∘∘ *Frel S1 S2 S3* = *Frel* (*R1* ∘∘ *S1*) (*R2* ∘∘ *S2*) (*R3* ∘∘ *S3*)
⟨*proof*⟩

lemma *Frel-Fmap-eq1*: *Frel R1 R2 R3* (*Fmap f1 f2 f3 x*) *y* =
Frel (λ*x*. *R1* (*f1 x*)) (λ*x*. *R2* (*f2 x*)) (λ*x*. *R3* (*f3 x*)) *x y*
⟨*proof*⟩

lemma *Frel-Fmap-eq2*: *Frel R1 R2 R3 x* (*Fmap g1 g2 g3 y*) =
Frel (λ*x y*. *R1 x* (*g1 y*)) (λ*x y*. *R2 x* (*g2 y*)) (λ*x y*. *R3 x* (*g3 y*)) *x y*
⟨*proof*⟩

lemmas *Frel-Fmap-eqs* = *Frel-Fmap-eq1 Frel-Fmap-eq2*

Predicator **definition** *Fpred* :: ('*a* ⇒ *bool*) ⇒ ('*b* ⇒ *bool*) ⇒ ('*c* ⇒ *bool*) ⇒
('*d*, '*a*, '*b*, '*c*) *F* ⇒ *bool*
where *Fpred P1 P2 P3 x* ≡ *Frel* ((=)↾$_{P1}$) ((=)↾$_{P2}$) ((=)↾$_{P3}$) *x x*

lemma *Fpred-mono-strong*:
assumes *Fpred P1 P2 P3 x*
and ⋀*x1*. *x1* ∈ *Fset1 x* ⟹ *P1 x1* ⟹ *Q1 x1*
and ⋀*x2*. *x2* ∈ *Fset2 x* ⟹ *P2 x2* ⟹ *Q2 x2*
and ⋀*x3*. *x3* ∈ *Fset3 x* ⟹ *P3 x3* ⟹ *Q3 x3*
shows *Fpred Q1 Q2 Q3 x*
⟨*proof*⟩

lemma *Fpred-top*: *Fpred* ⊤ ⊤ ⊤ *x*
⟨*proof*⟩

lemma *FpredI*:
assumes ⋀*x1*. *x1* ∈ *Fset1 x* ⟹ *P1 x1*
and ⋀*x2*. *x2* ∈ *Fset2 x* ⟹ *P2 x2*
and ⋀*x3*. *x3* ∈ *Fset3 x* ⟹ *P3 x3*
shows *Fpred P1 P2 P3 x*
⟨*proof*⟩

lemma *FpredE*:
assumes *Fpred P1 P2 P3 x*
obtains ⋀*x1*. *x1* ∈ *Fset1 x* ⟹ *P1 x1*
⋀*x2*. *x2* ∈ *Fset2 x* ⟹ *P2 x2*
⋀*x3*. *x3* ∈ *Fset3 x* ⟹ *P3 x3*
⟨*proof*⟩

lemma *Fpred-eq-ball*: *Fpred P1 P2 P3* =
(λ*x*. *Ball* (*Fset1 x*) *P1* ∧ *Ball* (*Fset2 x*) *P2* ∧ *Ball* (*Fset3 x*) *P3*)
⟨*proof*⟩

lemma *Fpred-Fmap-eq*:
Fpred P1 P2 P3 (*Fmap f1 f2 f3 x*) = *Fpred* (*P1* ∘ *f1*) (*P2* ∘ *f2*) (*P3* ∘ *f3*) *x*
⟨*proof*⟩

lemma *Fpred-in-dom-if-in-dom-Frel*:
assumes *in-dom* (*Frel R1 R2 R3*) *x*
shows *Fpred* (*in-dom R1*) (*in-dom R2*) (*in-dom R3*) *x*
⟨*proof*⟩

lemma *in-dom-Frel-if-Fpred-in-dom*:
assumes *Fpred* (*in-dom R1*) (*in-dom R2*) (*in-dom R3*) *x*
shows *in-dom* (*Frel R1 R2 R3*) *x*
⟨*proof*⟩

lemma *in-dom-Frel-eq-Fpred-in-dom*:
in-dom (*Frel R1 R2 R3*) = *Fpred* (*in-dom R1*) (*in-dom R2*) (*in-dom R3*)
⟨*proof*⟩

lemma *in-codom-Frel-eq-Fpred-in-codom*:
in-codom (*Frel R1 R2 R3*) = *Fpred* (*in-codom R1*) (*in-codom R2*) (*in-codom R3*)
⟨*proof*⟩

lemma *in-field-Frel-eq-Fpred-in-in-field*:
in-field (*Frel R1 R2 R3*) =
Fpred (*in-dom R1*) (*in-dom R2*) (*in-dom R3*) ⊔
Fpred (*in-codom R1*) (*in-codom R2*) (*in-codom R3*)
⟨*proof*⟩

lemma *Frel-restrict-left-Fpred-eq-Frel-restrict-left*:
fixes *R1* :: *'a1* ⇒ *'a2* ⇒ *bool*
and *R2* :: *'b1* ⇒ *'b2* ⇒ *bool*
and *R3* :: *'c1* ⇒ *'c2* ⇒ *bool*
and *P1* :: *'a1* ⇒ *bool*
and *P2* :: *'b1* ⇒ *bool*
and *P3* :: *'c1* ⇒ *bool*
shows (*Frel R1 R2 R3* :: (*'d*, *'a1*, *'b1*, *'c1*) *F* ⇒ -)↾$_{Fpred\ P1\ P2\ P3}$ :: (*'d*, *'a1*, *'b1*, *'c1*) *F* ⇒ -
=
Frel (*R1*↾$_{P1}$) (*R2*↾$_{P2}$) (*R3*↾$_{P3}$)
⟨*proof*⟩

lemma *Frel-restrict-right-Fpred-eq-Frel-restrict-right*:
fixes *R1* :: *'a1* ⇒ *'a2* ⇒ *bool*
and *R2* :: *'b1* ⇒ *'b2* ⇒ *bool*
and *R3* :: *'c1* ⇒ *'c2* ⇒ *bool*
and *P1* :: *'a2* ⇒ *bool*
and *P2* :: *'b2* ⇒ *bool*
and *P3* :: *'c2* ⇒ *bool*
shows (*Frel R1 R2 R3* :: - ⇒ (*'d*, *'a2*, *'b2*, *'c2*) *F* ⇒ -)↿$_{Fpred\ P1\ P2\ P3}$ :: (*'d*, *'a2*, *'b2*, *'c2*) *F* ⇒ -
=
Frel (*R1*↿$_{P1}$) (*R2*↿$_{P2}$) (*R3*↿$_{P3}$)
⟨*proof*⟩

locale *transport-natural-functor* =
t1 : *transport L1 R1 l1 r1* + *t2* : *transport L2 R2 l2 r2* +
t3 : *transport L3 R3 l3 r3*
for *L1* :: *'a1* ⇒ *'a1* ⇒ *bool*
and *R1* :: *'b1* ⇒ *'b1* ⇒ *bool*
and *l1* :: *'a1* ⇒ *'b1*
and *r1* :: *'b1* ⇒ *'a1*
and *L2* :: *'a2* ⇒ *'a2* ⇒ *bool*
and *R2* :: *'b2* ⇒ *'b2* ⇒ *bool*
and *l2* :: *'a2* ⇒ *'b2*

and $r2 :: 'b2 \Rightarrow 'a2$
and $L3 :: 'a3 \Rightarrow 'a3 \Rightarrow bool$
and $R3 :: 'b3 \Rightarrow 'b3 \Rightarrow bool$
and $l3 :: 'a3 \Rightarrow 'b3$
and $r3 :: 'b3 \Rightarrow 'a3$
begin

notation $L1$ (**infix** $\langle \leq_{L1} \rangle$ 50)
notation $R1$ (**infix** $\langle \leq_{R1} \rangle$ 50)
notation $L2$ (**infix** $\langle \leq_{L2} \rangle$ 50)
notation $R2$ (**infix** $\langle \leq_{R2} \rangle$ 50)
notation $L3$ (**infix** $\langle \leq_{L3} \rangle$ 50)
notation $R3$ (**infix** $\langle \leq_{R3} \rangle$ 50)

notation $t1.ge\text{-}left$ (**infix** $\langle \geq_{L1} \rangle$ 50)
notation $t1.ge\text{-}right$ (**infix** $\langle \geq_{R1} \rangle$ 50)
notation $t2.ge\text{-}left$ (**infix** $\langle \geq_{L2} \rangle$ 50)
notation $t2.ge\text{-}right$ (**infix** $\langle \geq_{R2} \rangle$ 50)
notation $t3.ge\text{-}left$ (**infix** $\langle \geq_{L3} \rangle$ 50)
notation $t3.ge\text{-}right$ (**infix** $\langle \geq_{R3} \rangle$ 50)

notation $t1.left\text{-}Galois$ (**infix** $\langle {}_{L1}\lessapprox \rangle$ 50)
notation $t1.right\text{-}Galois$ (**infix** $\langle {}_{R1}\lessapprox \rangle$ 50)
notation $t2.left\text{-}Galois$ (**infix** $\langle {}_{L2}\lessapprox \rangle$ 50)
notation $t2.right\text{-}Galois$ (**infix** $\langle {}_{R2}\lessapprox \rangle$ 50)
notation $t3.left\text{-}Galois$ (**infix** $\langle {}_{L3}\lessapprox \rangle$ 50)
notation $t3.right\text{-}Galois$ (**infix** $\langle {}_{R3}\lessapprox \rangle$ 50)

notation $t1.ge\text{-}Galois\text{-}left$ (**infix** $\langle \gtrapprox_{L1} \rangle$ 50)
notation $t1.ge\text{-}Galois\text{-}right$ (**infix** $\langle \gtrapprox_{R1} \rangle$ 50)
notation $t2.ge\text{-}Galois\text{-}left$ (**infix** $\langle \gtrapprox_{L2} \rangle$ 50)
notation $t2.ge\text{-}Galois\text{-}right$ (**infix** $\langle \gtrapprox_{R2} \rangle$ 50)
notation $t3.ge\text{-}Galois\text{-}left$ (**infix** $\langle \gtrapprox_{L3} \rangle$ 50)
notation $t3.ge\text{-}Galois\text{-}right$ (**infix** $\langle \gtrapprox_{R3} \rangle$ 50)

notation $t1.right\text{-}ge\text{-}Galois$ (**infix** $\langle {}_{R1}\gtrapprox \rangle$ 50)
notation $t1.Galois\text{-}right$ (**infix** $\langle \lessapprox_{R1} \rangle$ 50)
notation $t2.right\text{-}ge\text{-}Galois$ (**infix** $\langle {}_{R2}\gtrapprox \rangle$ 50)
notation $t2.Galois\text{-}right$ (**infix** $\langle \lessapprox_{R2} \rangle$ 50)
notation $t3.right\text{-}ge\text{-}Galois$ (**infix** $\langle {}_{R3}\gtrapprox \rangle$ 50)
notation $t3.Galois\text{-}right$ (**infix** $\langle \lessapprox_{R3} \rangle$ 50)

notation $t1.left\text{-}ge\text{-}Galois$ (**infix** $\langle {}_{L1}\gtrapprox \rangle$ 50)
notation $t1.Galois\text{-}left$ (**infix** $\langle \lessapprox_{L1} \rangle$ 50)
notation $t2.left\text{-}ge\text{-}Galois$ (**infix** $\langle {}_{L2}\gtrapprox \rangle$ 50)
notation $t2.Galois\text{-}left$ (**infix** $\langle \lessapprox_{L2} \rangle$ 50)
notation $t3.left\text{-}ge\text{-}Galois$ (**infix** $\langle {}_{L3}\gtrapprox \rangle$ 50)
notation $t3.Galois\text{-}left$ (**infix** $\langle \lessapprox_{L3} \rangle$ 50)

notation *t1.unit* (‹η_1›)
notation *t1.counit* (‹ε_1›)
notation *t2.unit* (‹η_2›)
notation *t2.counit* (‹ε_2›)
notation *t3.unit* (‹η_3›)
notation *t3.counit* (‹ε_3›)

definition $L \equiv$ *Frel* ($\leq_{L1}$) ($\leq_{L2}$) ($\leq_{L3}$)

lemma *left-rel-eq-Frel*: $L =$ *Frel* ($\leq_{L1}$) ($\leq_{L2}$) ($\leq_{L3}$)
⟨*proof*⟩

definition $l \equiv$ *Fmap l1 l2 l3*

lemma *left-eq-Fmap*: $l =$ *Fmap l1 l2 l3*
⟨*proof*⟩

context
begin

interpretation *flip* :
transport-natural-functor R1 L1 r1 l1 R2 L2 r2 l2 R3 L3 r3 l3 ⟨*proof*⟩

abbreviation $R \equiv$ *flip.L*
abbreviation $r \equiv$ *flip.l*

lemma *right-rel-eq-Frel*: $R =$ *Frel* ($\leq_{R1}$) ($\leq_{R2}$) ($\leq_{R3}$)
⟨*proof*⟩

lemma *right-eq-Fmap*: $r =$ *Fmap r1 r2 r3*
⟨*proof*⟩

lemmas *transport-defs* = *left-rel-eq-Frel left-eq-Fmap*
right-rel-eq-Frel right-eq-Fmap

end

sublocale *transport L R l r* ⟨*proof*⟩

notation L (**infix** ‹$\leq_L$› *50*)
notation R (**infix** ‹$\leq_R$› *50*)

lemma *unit-eq-Fmap*: $\eta =$ *Fmap* η_1 η_2 η_3
⟨*proof*⟩

interpretation *flip-inv* : *transport-natural-functor* ($\geq_{R1}$) ($\geq_{L1}$) *r1 l1*
($\geq_{R2}$) ($\geq_{L2}$) *r2 l2* ($\geq_{R3}$) ($\geq_{L3}$) *r3 l3*
rewrites *flip-inv.unit* $\equiv \varepsilon$ **and** *flip-inv.t1.unit* $\equiv \varepsilon_1$

and *flip-inv.t2.unit* $\equiv \varepsilon_2$ **and** *flip-inv.t3.unit* $\equiv \varepsilon_3$
⟨*proof*⟩

lemma *counit-eq-Fmap*: $\varepsilon = Fmap\ \varepsilon_1\ \varepsilon_2\ \varepsilon_3$
⟨*proof*⟩

lemma *flip-inv-right-eq-ge-left*: *flip-inv.R* $= (\geq_L)$
⟨*proof*⟩

interpretation *flip* :
transport-natural-functor R1 L1 r1 l1 R2 L2 r2 l2 R3 L3 r3 l3 ⟨*proof*⟩

lemma *flip-inv-left-eq-ge-right*: *flip-inv.L* $\equiv (\geq_R)$
⟨*proof*⟩

lemma *mono-wrt-rel-leftI*:
assumes $((\leq_{L1}) \Rightarrow (\leq_{R1}))$ *l1*
and $((\leq_{L2}) \Rightarrow (\leq_{R2}))$ *l2*
and $((\leq_{L3}) \Rightarrow (\leq_{R3}))$ *l3*
shows $((\leq_L) \Rightarrow (\leq_R))$ *l*
⟨*proof*⟩

end

end

2.28.2 Galois Concepts

theory *Transport-Natural-Functors-Galois*
imports
Transport-Natural-Functors-Base
begin

context *transport-natural-functor*
begin

lemma *half-galois-prop-leftI*:
assumes $((\leq_{L1})\ {}_h\trianglelefteq\ (\leq_{R1}))$ *l1 r1*
and $((\leq_{L2})\ {}_h\trianglelefteq\ (\leq_{R2}))$ *l2 r2*
and $((\leq_{L3})\ {}_h\trianglelefteq\ (\leq_{R3}))$ *l3 r3*
shows $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))$ *l r*
⟨*proof*⟩

interpretation *flip-inv* : *transport-natural-functor* $(\geq_{R1})$ $(\geq_{L1})$ *r1 l1* $(\geq_{R2})$ $(\geq_{L2})$ *r2 l2* $(\geq_{R3})$ $(\geq_{L3})$ *r3 l3*
rewrites *flip-inv.R* $\equiv (\geq_L)$
and *flip-inv.L* $\equiv (\geq_R)$
and $\bigwedge R\ S\ f\ g.\ (R^{-1}\ {}_h\trianglelefteq\ S^{-1})\ f\ g \equiv (S\ \trianglelefteq_h\ R)\ g\ f$

⟨*proof*⟩

lemma *half-galois-prop-rightI*:
assumes $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*
and $((\leq_{L2}) \trianglelefteq_h (\leq_{R2}))$ *l2 r2*
and $((\leq_{L3}) \trianglelefteq_h (\leq_{R3}))$ *l3 r3*
shows $((\leq_L) \trianglelefteq_h (\leq_R))$ *l r*
⟨*proof*⟩

corollary *galois-propI*:
assumes $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
and $((\leq_{L2}) \trianglelefteq (\leq_{R2}))$ *l2 r2*
and $((\leq_{L3}) \trianglelefteq (\leq_{R3}))$ *l3 r3*
shows $((\leq_L) \trianglelefteq (\leq_R))$ *l r*
⟨*proof*⟩

interpretation *flip* :
transport-natural-functor R1 L1 r1 l1 R2 L2 r2 l2 R3 L3 r3 l3 ⟨*proof*⟩

corollary *galois-connectionI*:
assumes $((\leq_{L1}) \dashv (\leq_{R1}))$ *l1 r1*
and $((\leq_{L2}) \dashv (\leq_{R2}))$ *l2 r2*
and $((\leq_{L3}) \dashv (\leq_{R3}))$ *l3 r3*
shows $((\leq_L) \dashv (\leq_R))$ *l r*
⟨*proof*⟩

corollary *galois-equivalenceI*:
assumes $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1*
and $((\leq_{L2}) \equiv_G (\leq_{R2}))$ *l2 r2*
and $((\leq_{L3}) \equiv_G (\leq_{R3}))$ *l3 r3*
shows $((\leq_L) \equiv_G (\leq_R))$ *l r*
⟨*proof*⟩

end

end

2.28.3 Galois Relator

theory *Transport-Natural-Functors-Galois-Relator*
imports
Transport-Natural-Functors-Base
begin

context *transport-natural-functor*
begin

lemma *left-Galois-Frel-left-Galois*: $({}_L\lessapprox) \leq$ *Frel* $({}_{L1}\lessapprox)$ $({}_{L2}\lessapprox)$ $({}_{L3}\lessapprox)$

⟨*proof*⟩

lemma *Frel-left-Galois-le-left-Galois*:
Frel $(_{L1}\lessapprox)$ $(_{L2}\lessapprox)$ $(_{L3}\lessapprox) \leq (_{L}\lessapprox)$
⟨*proof*⟩

corollary *left-Galois-eq-Frel-left-Galois*: $(_{L}\lessapprox) = Frel\ (_{L1}\lessapprox)\ (_{L2}\lessapprox)\ (_{L3}\lessapprox)$
⟨*proof*⟩

end

end

2.28.4 Basic Order Properties

theory *Transport-Natural-Functors-Order-Base*
imports
Transport-Natural-Functors-Base
begin

context
fixes *R1* :: $'a \Rightarrow 'a \Rightarrow bool$ **and** *R2* :: $'b \Rightarrow 'b \Rightarrow bool$ **and** *R3* :: $'c \Rightarrow 'c \Rightarrow bool$
begin

lemma *reflexive-on-in-field-FrelI*:
assumes *reflexive-on* (*in-field R1*) *R1*
and *reflexive-on* (*in-field R2*) *R2*
and *reflexive-on* (*in-field R3*) *R3*
defines $R \equiv Frel\ R1\ R2\ R3$
shows *reflexive-on* (*in-field R*) *R*
⟨*proof*⟩

lemma *transitive-FrelI*:
assumes *transitive R1*
and *transitive R2*
and *transitive R3*
shows *transitive* (*Frel R1 R2 R3*)
⟨*proof*⟩

lemma *preorder-on-in-field-FrelI*:
assumes *preorder-on* (*in-field R1*) *R1*
and *preorder-on* (*in-field R2*) *R2*
and *preorder-on* (*in-field R3*) *R3*
defines $R \equiv Frel\ R1\ R2\ R3$
shows *preorder-on* (*in-field R*) *R*
⟨*proof*⟩

lemma *symmetric-FrelI*:

assumes *symmetric R1*
and *symmetric R2*
and *symmetric R3*
shows *symmetric* (*Frel R1 R2 R3*)
⟨*proof*⟩

lemma *partial-equivalence-rel-FrelI*:
assumes *partial-equivalence-rel R1*
and *partial-equivalence-rel R2*
and *partial-equivalence-rel R3*
shows *partial-equivalence-rel* (*Frel R1 R2 R3*)
⟨*proof*⟩

end

context *transport-natural-functor*
begin

lemmas *reflexive-on-in-field-leftI* = *reflexive-on-in-field-FrelI*
[*of L1 L2 L3*, *folded transport-defs*]

lemmas *transitive-leftI* = *transitive-FrelI*[*of L1 L2 L3*, *folded transport-defs*]

lemmas *preorder-on-in-field-leftI* = *preorder-on-in-field-FrelI*
[*of L1 L2 L3*, *folded transport-defs*]

lemmas *symmetricI* = *symmetric-FrelI*[*of L1 L2 L3*, *folded transport-defs*]

lemmas *partial-equivalence-rel-leftI* = *partial-equivalence-rel-FrelI*
[*of L1 L2 L3*, *folded transport-defs*]

end

end

2.28.5 Order Equivalence

theory *Transport-Natural-Functors-Order-Equivalence*
imports
Transport-Natural-Functors-Base
begin

context
fixes *R1* :: *'a* ⇒ *'a* ⇒ *bool* **and** *R2* :: *'b* ⇒ *'b* ⇒ *bool* **and** *R3* :: *'c* ⇒ *'c* ⇒ *bool*
and *f1* :: *'a* ⇒ *'a* **and** *f2* :: *'b* ⇒ *'b* **and** *f3* :: *'c* ⇒ *'c*
and *R* :: (*'d*, *'a*, *'b*, *'c*) *F* ⇒ (*'d*, *'a*, *'b*, *'c*) *F* ⇒ *bool*
and *f* :: (*'d*, *'a*, *'b*, *'c*) *F* ⇒ (*'d*, *'a*, *'b*, *'c*) *F*
defines *R* ≡ *Frel R1 R2 R3* **and** *f* ≡ *Fmap f1 f2 f3*

begin

lemma *inflationary-on-in-dom-FrelI*:
assumes *inflationary-on* (*in-dom R1*) *R1 f1*
and *inflationary-on* (*in-dom R2*) *R2 f2*
and *inflationary-on* (*in-dom R3*) *R3 f3*
shows *inflationary-on* (*in-dom R*) *R f*
⟨*proof*⟩

lemma *inflationary-on-in-codom-FrelI*:
assumes *inflationary-on* (*in-codom R1*) *R1 f1*
and *inflationary-on* (*in-codom R2*) *R2 f2*
and *inflationary-on* (*in-codom R3*) *R3 f3*
shows *inflationary-on* (*in-codom R*) *R f*
⟨*proof*⟩

end

context
fixes *R1* :: *'a* ⇒ *'a* ⇒ *bool* **and** *R2* :: *'b* ⇒ *'b* ⇒ *bool* **and** *R3* :: *'c* ⇒ *'c* ⇒ *bool*
and *f1* :: *'a* ⇒ *'a* **and** *f2* :: *'b* ⇒ *'b* **and** *f3* :: *'c* ⇒ *'c*
and *R* :: (*'d*, *'a*, *'b*, *'c*) *F* ⇒ (*'d*, *'a*, *'b*, *'c*) *F* ⇒ *bool*
and *f* :: (*'d*, *'a*, *'b*, *'c*) *F* ⇒ (*'d*, *'a*, *'b*, *'c*) *F*
defines *R* ≡ *Frel R1 R2 R3* **and** *f* ≡ *Fmap f1 f2 f3*
begin

lemma *inflationary-on-in-field-FrelI*:
assumes *inflationary-on* (*in-field R1*) *R1 f1*
and *inflationary-on* (*in-field R2*) *R2 f2*
and *inflationary-on* (*in-field R3*) *R3 f3*
shows *inflationary-on* (*in-field R*) *R f*
⟨*proof*⟩

lemma *deflationary-on-in-dom-FrelI*:
assumes *deflationary-on* (*in-dom R1*) *R1 f1*
and *deflationary-on* (*in-dom R2*) *R2 f2*
and *deflationary-on* (*in-dom R3*) *R3 f3*
shows *deflationary-on* (*in-dom R*) *R f*
⟨*proof*⟩

lemma *deflationary-on-in-codom-FrelI*:
assumes *deflationary-on* (*in-codom R1*) *R1 f1*
and *deflationary-on* (*in-codom R2*) *R2 f2*
and *deflationary-on* (*in-codom R3*) *R3 f3*
shows *deflationary-on* (*in-codom R*) *R f*
⟨*proof*⟩

end

context
fixes *R1* :: $'a \Rightarrow 'a \Rightarrow bool$ **and** *R2* :: $'b \Rightarrow 'b \Rightarrow bool$ **and** *R3* :: $'c \Rightarrow 'c \Rightarrow bool$
and *f1* :: $'a \Rightarrow 'a$ **and** *f2* :: $'b \Rightarrow 'b$ **and** *f3* :: $'c \Rightarrow 'c$
and *R* :: $('d, 'a, 'b, 'c)\ F \Rightarrow ('d, 'a, 'b, 'c)\ F \Rightarrow bool$
and *f* :: $('d, 'a, 'b, 'c)\ F \Rightarrow ('d, 'a, 'b, 'c)\ F$
defines $R \equiv$ *Frel R1 R2 R3* **and** $f \equiv$ *Fmap f1 f2 f3*
begin

lemma *deflationary-on-in-field-FrelI*:
assumes *deflationary-on* (*in-field R1*) *R1 f1*
and *deflationary-on* (*in-field R2*) *R2 f2*
and *deflationary-on* (*in-field R3*) *R3 f3*
shows *deflationary-on* (*in-field R*) *R f*
⟨*proof*⟩

lemma *rel-equivalence-on-in-field-FrelI*:
assumes *rel-equivalence-on* (*in-field R1*) *R1 f1*
and *rel-equivalence-on* (*in-field R2*) *R2 f2*
and *rel-equivalence-on* (*in-field R3*) *R3 f3*
shows *rel-equivalence-on* (*in-field R*) *R f*
⟨*proof*⟩

end

context *transport-natural-functor*
begin

lemmas *inflationary-on-in-field-unitI* = *inflationary-on-in-field-FrelI*
[*of L1* η_1 *L2* η_2 *L3* η_3, *folded transport-defs unit-eq-Fmap*]

lemmas *deflationary-on-in-field-unitI* = *deflationary-on-in-field-FrelI*
[*of L1* η_1 *L2* η_2 *L3* η_3, *folded transport-defs unit-eq-Fmap*]

lemmas *rel-equivalence-on-in-field-unitI* = *rel-equivalence-on-in-field-FrelI*
[*of L1* η_1 *L2* η_2 *L3* η_3, *folded transport-defs unit-eq-Fmap*]

interpretation *flip* :
transport-natural-functor R1 L1 r1 l1 R2 L2 r2 l2 R3 L3 r3 l3
rewrites *flip.unit* $\equiv \varepsilon$ **and** *flip.t1.unit* $\equiv \varepsilon_1$
and *flip.t2.unit* $\equiv \varepsilon_2$ **and** *flip.t3.unit* $\equiv \varepsilon_3$
⟨*proof*⟩

lemma *order-equivalenceI*:
assumes $((\leq_{L1}) \equiv_o (\leq_{R1}))$ *l1 r1*
and $((\leq_{L2}) \equiv_o (\leq_{R2}))$ *l2 r2*
and $((\leq_{L3}) \equiv_o (\leq_{R3}))$ *l3 r3*
shows $((\leq_L) \equiv_o (\leq_R))$ *l r*
⟨*proof*⟩

end

end

theory *Transport-Natural-Functors*
imports
Transport-Natural-Functors-Galois
Transport-Natural-Functors-Galois-Relator
Transport-Natural-Functors-Order-Base
Transport-Natural-Functors-Order-Equivalence
begin

Summary Summary of results for a fixed natural functor with 3 parameters. All apply-style proofs are written such that they also apply to functors with other arities. An automatic derivation of these results for all natural functors needs to be implemented in the BNF package. This is future work.

context *transport-natural-functor*
begin

interpretation *flip* :
transport-natural-functor R1 L1 r1 l1 R2 L2 r2 l2 R3 L3 r3 l3 ⟨*proof*⟩

theorem *preorder-equivalenceI*:
assumes $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and $((\leq_{L2}) \equiv_{pre} (\leq_{R2}))$ *l2 r2*
and $((\leq_{L3}) \equiv_{pre} (\leq_{R3}))$ *l3 r3*
shows $((\leq_{L}) \equiv_{pre} (\leq_{R}))$ *l r*
⟨*proof*⟩

theorem *partial-equivalence-rel-equivalenceI*:
assumes $((\leq_{L1}) \equiv_{PER} (\leq_{R1}))$ *l1 r1*
and $((\leq_{L2}) \equiv_{PER} (\leq_{R2}))$ *l2 r2*
and $((\leq_{L3}) \equiv_{PER} (\leq_{R3}))$ *l3 r3*
shows $((\leq_{L}) \equiv_{PER} (\leq_{R}))$ *l r*
⟨*proof*⟩

For the simplification of the Galois relator see *flip.right-Galois* = *Frel flip.t1.right-Galois flip.t2.right-Galois flip.t3.right-Galois*.

end

end

2.29 Transport Paper Guide

theory *Transport-Via-Partial-Galois-Connections-Equivalences-Paper*
imports

Transport-Black-Box
Transport-Natural-Functors
Transport-Partial-Quotient-Types
Transport-Prototype
Transport-Lists-Sets-Examples
Transport-Dep-Fun-Rel-Examples
Transport-Typedef-Base
begin

- Section 3.1: Order basics can be found in *Transport.Binary-Relation-Properties*, *Transport.Preorders*, *Transport.Partial-Equivalence-Relations*, *Transport.Equivalence-Relations*, and *Transport.Order-Functions-Base*. Theorem
- Section 3.2: Function relators and monotonicity can be found in *Transport.Function-Relators* and *Transport.Functions-Monotone*
- Section 3.3: Galois relator can be found in *Transport.Galois-Relator-Base*.
  - Lemma 1: *Transport.Transport-Partial-Quotient-Types* (*results from Appendix*)
  - Lemma 3: *galois-prop.galois-prop ?L ?R ?l ?r* $\Longrightarrow$ (*galois-rel.Galois ?R*$^{-1}$ *?L*$^{-1}$ *?l*)$^{-1}$ *?x ?y* = *Galois-infix ?x ?L ?R ?r ?y*
- Section 3.4: Partial Galois Connections and Equivalences can be found in *Transport.Half-Galois-Property*, *Transport.Galois-Property*, *Transport.Galois-Connections*, *Transport.Galois-Equivalences*, and *Transport.Order-Equivalences*.
  - Lemma 2: *Transport.Transport-Partial-Quotient-Types* (*results from Appendix*)
  - Lemma 4: ⟦*order-functors.order-equivalence ?L ?R ?l ?r*; *transitive ?L*; *transitive ?R*⟧ $\Longrightarrow$ *galois.galois-equivalence ?L ?R ?l ?r*
  - Lemma 5: ⟦*galois.galois-equivalence ?L ?R ?l ?r*; *reflexive-on* (*in-field ?L*) *?L*; *reflexive-on* (*in-field ?R*) *?R*⟧ $\Longrightarrow$ *order-functors.order-equivalence ?L ?R ?l ?r*
- Section 4.1: Closure (Dependent) Function Relator can be found in `Black_Box/Functions`.
  - Monotone function relator *Transport.Monotone-Function-Relator*.
  - Setup of construction *Transport.Transport-Functions-Base*.
  - Theorem 1: see *Transport.Transport-Functions*

– Theorem 2: see ⟦*transport.preorder-equivalence ?L1.0 ?R1.0 ?l1.0 ?r1.0*; ⋀*x x′. Galois-infix x ?L1.0 ?R1.0 ?r1.0 x′ ⟹ transport.preorder-equivalence* (*?L2.0 x* (*?r1.0 x′*)) (*?R2.0* (*?l1.0 x*) *x′*) (*?l2.0 x′ x*) (*?r2.0 x x′*); ((*x1 x2* :: *?L1.0*$^{-1}$) ⇒ (*x3 x4* :: *?L1.0*) ⇛ *?L1.0 x1 x3* ⟶ (≤)) *?L2.0*; ((*x1 x2* :: *?L1.0*) ⇒ (*x1′ x2′* :: *?R1.0*) ⇛ *Galois-infix x2 ?L1.0 ?R1.0 ?r1.0 x1′* ⟶ (*in-field* (*?R2.0* (*?l1.0 x1*) *x2′*) ⇛ *?L2.0 x1* (*?r1.0 x2′*))) *?r2.0*; *in-dom* (*transport-Mono-Dep-Fun-Rel.L ?L1.0 ?L2.0*) *?f*; *in-codom* (*transport-Mono-Dep-Fun-Rel.L ?R1.0 ?R2.0*) *?g*⟧ ⟹ *Galois-infix ?f* (*transport-Mono-Dep-Fun-Rel.L ?L1.0 ?L2.0*) (*transport-Mono-Dep-Fun-R ?R1.0 ?R2.0*) (*transport-Dep-Fun-Rel.l ?l1.0 ?r2.0*) *?g* = ((*x x′* :: *galois-rel.Galois ?L1.0 ?R1.0 ?r1.0*) ⇛ *galois-rel.Galois* (*?L2.0 x* (*?r1.0 x′*)) (*?R2.0* (*?l1.0 x*) *x′*) (*?r2.0 x x′*)) *?f ?g* (*results from Appendix*)

– Lemma 6: ⟦*galois.galois-connection ?L1.0 ?R1.0 ?l1.0 ?r1.0*; *reflexive-on* (*in-codom ?L1.0*) *?L1.0*; *reflexive-on* (*in-dom ?R1.0*) *?R1.0*; *galois.galois-connection ?L2.0 ?R2.0 ?l2.0 ?r2.0*; *transitive ?L2.0*; *transitive ?R2.0*⟧ ⟹ *galois.galois-connection* (*transport-Mono-Dep-Fun-Rel.L ?L1.0* (λ- -. *?L2.0*)) (*transport-Mono-Dep-Fun-Rel.L ?R1.0* (λ- -. *?R2.0*)) (*transport-Dep-Fun-Rel.l ?r1.0* (λ- -. *?l2.0*)) (*transport-Dep-Fun-Rel.l ?l1.0* (λ- -. *?r2.0*))

– Lemma 7: ⟦(*?L1.0* ⇒ *?R1.0*) *?l1.0*; *galois-prop.galois-prop ?L1.0 ?R1.0 ?l1.0 ?r1.0*; *reflexive-on* (*in-dom ?L1.0*) *?L1.0*; (*?R2.0* ⇒ *?L2.0*) *?r2.0*; *transitive ?L2.0*; *in-dom* (*transport-Mono-Dep-Fun-Rel.L ?L1.0* (λ- -. *?L2.0*)) *?f*; *in-codom* (*transport-Mono-Dep-Fun-Rel.L ?R1.0* (λ- -. *?R2.0*)) *?g*⟧ ⟹ *Galois-infix ?f* (*transport-Mono-Dep-Fun-Rel.L ?L1.0* (λ- -. *?L2.0*)) (*transport-Mono-Dep-Fun-Rel.L ?R1.0* (λ- -. *?R2.0*)) (*transport-Dep-Fun-Rel.l ?l1.0* (λ- -. *?r2.0*)) *?g* = (*galois-rel.Galois ?L1.0 ?R1.0 ?r1.0* ⇛ *galois-rel.Galois ?L2.0 ?R2.0 ?r2.0*) *?f ?g*

– Theorem 7: ⟦*galois.galois-connection ?L1.0 ?R1.0 ?l1.0 ?r1.0*; *reflexive-on* (*in-field ?L1.0*) *?L1.0*; *reflexive-on* (*in-field ?R1.0*) *?R1.0*; ⋀*x x′. Galois-infix x ?L1.0 ?R1.0 ?r1.0 x′ ⟹ galois.galois-connection* (*?L2.0 x* (*?r1.0 x′*)) (*?R2.0* (*?l1.0 x*) *x′*) (*?l2.0 x′ x*) (*?r2.0 x x′*); ((- *x2* :: *?L1.0*) ⇒ (*x3 x4* :: *?L1.0*) ⇛ (*?L1.0 x2 x3* ∧ *?L1.0 x4* (*order-functors.unit ?l1.0 ?r1.0 x3*)) ⟶ (λ*x y*. *y* ≤ *x*)) *?L2.0*; ((*x1′ x2′* :: *?R1.0*) ⇒ *?R1.0* (*order-functors.counit ?l1.0 ?r1.0 x2′*) *x1′* ⟶ ((*x3′* - :: *?R1.0*) ⇛ *?R1.0 x2′ x3′* ⟶ (≤))) *?R2.0*; ((*x1′ x2′* :: *?R1.0*) ⇒ (*x1 x2* :: *?L1.0*) ⇛ *Galois-infix x2 ?L1.0 ?R1.0 ?r1.0 x1′* ⟶ (*in-field* (*?L2.0 x1* (*?r1.0 x2′*)) ⇛ *?R2.0* (*?l1.0 x1*) *x2′*)) *?l2.0*; ((*x1 x2* :: *?L1.0*) ⇒ (*x1′ x2′* :: *?R1.0*) ⇛ *Galois-infix x2 ?L1.0 ?R1.0 ?r1.0 x1′* ⟶ (*in-field* (*?R2.0* (*?l1.0 x1*) *x2′*) ⇛ *?L2.0 x1* (*?r1.0 x2′*))) *?r2.0*; ⋀*x1 x2*.

?L1.0 x1 x2 $\Longrightarrow$ *transitive* (*?L2.0 x1 x2*); $\bigwedge$*x1′ x2′*. *?R1.0 x1′ x2′* $\Longrightarrow$ *transitive* (*?R2.0 x1′ x2′*)$\rrbracket$ $\Longrightarrow$ *galois.galois-connection* (*transport-Mono-Dep-Fun-Rel.L ?L1.0 ?L2.0*) (*transport-Mono-Dep-Fun-Rel.L ?R1.0 ?R2.0*) (*transport-Dep-Fun-Rel.l ?r1.0 ?l2.0*) (*transport-Dep-Fun-Rel.l ?l1.0 ?r2.0*)

- Theorem 8: $\llbracket$*galois.galois-connection ?L1.0 ?R1.0 ?l1.0 ?r1.0*; *reflexive-on* (*in-field ?L1.0*) *?L1.0*; $\bigwedge$*x x′*. *Galois-infix x ?L1.0 ?R1.0 ?r1.0 x′* $\Longrightarrow$ (*?R2.0* (*?l1.0 x*) *x′* $\Rightarrow$ *?L2.0 x* (*?r1.0 x′*)) (*?r2.0 x x′*); ((*x1* : $\top$) $\Rightarrow$ (*x2* - :: *?L1.0*) $\Rightarrow$ *?L1.0 x1 x2* $\longrightarrow$ ($\leq$)) *?L2.0*; ((*x1* : $\top$) $\Rightarrow$ (*x2 x3* :: *?L1.0*) $\Rightarrow$ (*?L1.0 x1 x2* $\wedge$ *?L1.0 x3* (*order-functors.unit ?l1.0 ?r1.0 x2*)) $\longrightarrow$ (λ*x y*. *y* $\leq$ *x*)) *?L2.0*; ((*x1 x2* :: *?L1.0*) $\Rightarrow$ (*x1′ x2′* :: *?R1.0*) $\Rrightarrow$ *Galois-infix x2 ?L1.0 ?R1.0 ?r1.0 x1′* $\longrightarrow$ (*in-field* (*?R2.0* (*?l1.0 x1*) *x2′*) $\Rrightarrow$ *?L2.0 x1* (*?r1.0 x2′*))) *?r2.0*; $\bigwedge$*x1 x2*. *?L1.0 x1 x2* $\Longrightarrow$ *transitive* (*?L2.0 x1 x2*); *in-dom* (*transport-Mono-Dep-Fun-Rel.L ?L1.0 ?L2.0*) *?f*; *in-codom* (*transport-Mono-Dep-Fun-Rel.L ?R1.0 ?R2.0*) *?g*$\rrbracket$ $\Longrightarrow$ *Galois-infix ?f* (*transport-Mono-Dep-Fun-Rel.L ?L1.0 ?L2.0*) (*transport-Mono-Dep-Fun-Rel.L ?R1.0 ?R2.0*) (*transport-Dep-Fun-Rel.l ?l1.0 ?r2.0*) *?g* = ((*x x′* :: *galois-rel.Galois ?L1.0 ?R1.0 ?r1.0*) $\Rrightarrow$ *galois-rel.Galois* (*?L2.0 x* (*?r1.0 x′*)) (*?R2.0* (*?l1.0 x*) *x′*) (*?r2.0 x x′*)) *?f ?g*
- Lemma 8 $\llbracket$*galois.galois-equivalence ?L1.0 ?R1.0 ?l1.0 ?r1.0*; *Pre-orders.preorder-on* (*in-field ?L1.0*) *?L1.0*; ((*x1 x2* :: *?L1.0*$^{-1}$) $\Rightarrow$ (*x3 x4* :: *?L1.0*) $\Rrightarrow$ *?L1.0 x1 x3* $\longrightarrow$ ($\leq$)) *?L2.0*; $\bigwedge$*x1 x2*. *?L1.0 x1 x2* $\Longrightarrow$ *partial-equivalence-rel* (*?L2.0 x1 x2*)$\rrbracket$ $\Longrightarrow$ *transport-Mono-Dep-Fun-Rel.L ?L1.0 ?L2.0* = *transport-Dep-Fun-Rel.L ?L1.0 ?L2.0*
- Lemma 9: $\llbracket$*reflexive-on* (*in-field ?L1.0*) *?L1.0*; *partial-equivalence-rel ?L2.0*$\rrbracket$ $\Longrightarrow$ *transport-Mono-Dep-Fun-Rel.L ?L1.0* (λ- -. *?L2.0*) = *transport-Dep-Fun-Rel.L ?L1.0* (λ- -. *?L2.0*)

- Section 4.2: Closure Natural Functors can be found in `Black_Box/Natural_Functors`.
  - Theorem 3: see *Transport.Transport-Natural-Functors*
  - Theorem 4: *galois-rel.Galois* (*transport-natural-functor.L ?L1.0 ?L2.0 ?L3.0*) (*transport-natural-functor.L ?R1.0 ?R2.0 ?R3.0*) (*transport-natural-functor.l ?r1.0 ?r2.0 ?r3.0*) = *Frel* (*galois-rel.Galois ?L1.0 ?R1.0 ?r1.0*) (*galois-rel.Galois ?L2.0 ?R2.0 ?r2.0*) (*galois-rel.Galois ?L3.0 ?R3.0 ?r3.0*)

- Section 4.3: Closure Compositions can be found in `Black_Box/Compositions`.
  - Setup for simple case in *Transport.Transport-Compositions-Agree-Base*

- Setup for generic case in *Transport.Transport-Compositions-Generic-Base*
- Theorem 5: ⟦*transport.preorder-equivalence ?L1.0 ?R1.0 ?l1.0 ?r1.0*; *transport.preorder-equivalence ?L2.0 ?R2.0 ?l2.0 ?r2.0*; *transport-comp.middle-compatible-codom ?R1.0 ?L2.0*⟧ ⟹ *transport.preorder-equivalence* (*transport-comp.L ?L1.0 ?R1.0 ?l1.0 ?r1.0 ?L2.0*) (*transport-comp.L ?R2.0 ?L2.0 ?r2.0 ?l2.0 ?R1.0*) (*transport-comp.l ?l1.0 ?l2.0*) (*transport-comp.l ?r2.0 ?r1.0*) and ⟦*transport.partial-equivalence-rel-equivalence ?L1.0 ?R1.0 ?l1.0 ?r1.0*; *transport.partial-equivalence-rel-equivalence ?L2.0 ?R2.0 ?l2.0 ?r2.0*; *transport-comp.middle-compatible-codom ?R1.0 ?L2.0*⟧ ⟹ *transport.partial-equivalence-rel-equivalence* (*transport-comp.L ?L1.0 ?R1.0 ?l1.0 ?r1.0 ?L2.0*) (*transport-comp.L ?R2.0 ?L2.0 ?r2.0 ?l2.0 ?R1.0*) (*transport-comp.l ?l1.0 ?l2.0*) (*transport-comp.l ?r2.0 ?r1.0*)
- Theorem 6: ⟦*transport.preorder-equivalence ?L1.0 ?R1.0 ?l1.0 ?r1.0*; *transport.preorder-galois-connection ?R2.0 ?L2.0 ?r2.0 ?l2.0*; *transport-comp.middle-compatible-codom ?R1.0 ?L2.0*⟧ ⟹ *galois-rel.Galois* (*transport-comp.L ?L1.0 ?R1.0 ?l1.0 ?r1.0 ?L2.0*) (*transport-comp.L ?R2.0 ?L2.0 ?r2.0 ?l2.0 ?R1.0*) (*transport-comp.l ?r2.0 ?r1.0*) = *galois-rel.Galois ?L1.0 ?R1.0 ?r1.0* ∘∘ *galois-rel.Galois ?L2.0 ?R2.0 ?r2.0*
  (*results from Appendix*)
- Theorem 9: see `Black_Box/Compositions/Agree`, results in *transport-comp-same.*
- Theorem 10: ⟦*galois.galois-equivalence ?L1.0 ?R1.0 ?l1.0 ?r1.0*; *Preorders.preorder-on* (*in-field ?R1.0*) *?R1.0*; *galois.galois-equivalence ?L2.0 ?R2.0 ?l2.0 ?r2.0*; *Preorders.preorder-on* (*in-field ?L2.0*) *?L2.0*; *transport-comp.middle-compatible-codom ?R1.0 ?L2.0*⟧ ⟹ *galois.galois-connection* (*transport-comp.L ?L1.0 ?R1.0 ?l1.0 ?r1.0 ?L2.0*) (*transport-comp.L ?R2.0 ?L2.0 ?r2.0 ?l2.0 ?R1.0*) (*transport-comp.l ?l1.0 ?l2.0*) (*transport-comp.l ?r2.0 ?r1.0*)
- Theorem 11: ⟦(*?R1.0* ⇒ *?L1.0*) *?r1.0*; *galois-prop.galois-prop ?L1.0 ?R1.0 ?l1.0 ?r1.0*; *galois-prop.half-galois-prop-right ?R1.0 ?L1.0 ?r1.0 ?l1.0*; *Preorders.preorder-on* (*in-field ?R1.0*) *?R1.0*; (*?L2.0* ⇒ *?R2.0*) *?l2.0*; *galois-prop.half-galois-prop-left ?R2.0 ?L2.0 ?r2.0 ?l2.0*; *reflexive-on* (*in-dom ?L2.0*) *?L2.0*; *?R1.0* ∘∘ *?L2.0* ∘∘ *?R1.0* ≤ *?R1.0* ∘∘ *?L2.0*; *in-codom* (*?L2.0* ∘∘ *?R1.0* ∘∘ *?L2.0*) ≤ *in-codom ?R1.0*⟧ ⟹ *galois-rel.Galois* (*transport-comp.L ?L1.0 ?R1.0 ?l1.0 ?r1.0 ?L2.0*) (*transport-comp.L ?R2.0 ?L2.0 ?r2.0 ?l2.0 ?R1.0*) (*transport-comp.l ?r2.0 ?r1.0*) = *galois-rel.Galois ?L1.0 ?R1.0 ?r1.0* ∘∘ *galois-rel.Galois ?L2.0 ?R2.0 ?r2.0*

- Section 5:

  - Implementation *Transport.Transport-Prototype*: Note: the command "trp" from the paper is called **trp-term** and the method "trprover" is called "trp_term_prover".
  - Example 1: *Transport.Transport-Lists-Sets-Examples*
  - Example 2: *Transport.Transport-Dep-Fun-Rel-Examples*
  - Example 3: https://github.com/kappelmann/Isabelle-Set/blob/fdf59444d9a53b5279080fb4d24893c9efa31160/Isabelle_Set/Integers/Integers_Transport.thy

- Proof: Partial Quotient Types are a special case: *Transport.Transport-Partial-Quotient-Types*
- Proof: Typedefs are a special case: *Transport.Transport-Typedef-Base*
- Proof: Set-Extensions are a special case: https://github.com/kappelmann/Isabelle-Set/blob/fdf59444d9a53b5279080fb4d24893c9efa31160/Isabelle_Set/Set_Extensions/Set_Extensions_Transport.thy
- Proof: Bijections as special case: *Transport.Transport-Bijections*

end

Bibliography

[1] Huffman, Brian and Kunčar, Ondřej. Lifting and Transfer: A modular design for quotients in Isabelle/HOL. In *Certified Programs and Proofs: Third International Conference, CPP 2013, Melbourne, VIC, Australia, December 11-13, 2013, Proceedings 3*, pages 131–146. Springer, 2013.

[2] K. Kappelmann. Transport via partial galois connections and equivalences. In C.-K. Hur, editor, *Programming Languages and Systems*, pages 225–245, Singapore, 2023. Springer Nature Singapore.