Transport via Partial Galois Connections and Equivalences

Kevin Kappelmann

October 13, 2023

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

1.1 Binary Relations

1.1.1 Basic Functions

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

Summary Basic functions on binary relations.

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

bundle *rel-comp-syntax* **begin notation** *rel-comp* (**infixl** ∘∘ *55*) **end**
bundle *no-rel-comp-syntax* **begin no-notation** *rel-comp* (**infixl** ∘∘ *55*) **end**
unbundle *rel-comp-syntax*

lemma *rel-compI* [*intro*]:
  assumes *R x y*
  and *S y z*
  shows $(R \circ\circ S)\ x\ z$
  using *assms* **unfolding** *rel-comp-def* **by** *blast*

lemma *rel-compE* [*elim*]:
  assumes $(R \circ\circ S)\ x\ y$
  obtains *z* **where** *R x z S z y*
  using *assms* **unfolding** *rel-comp-def* **by** *blast*

lemma *rel-comp-assoc*: $R \circ\circ (S \circ\circ T) = (R \circ\circ S) \circ\circ T$
by (*intro ext*) *blast*

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

bundle *rel-inv-syntax* **begin notation** *rel-inv* ((-$^{-1}$) [*1000*]) **end**
bundle *no-rel-inv-syntax* **begin no-notation** *rel-inv* ((-$^{-1}$) [*1000*]) **end**
unbundle *rel-inv-syntax*

lemma *rel-invI* [*intro*]:
assumes $R\ x\ y$
shows $R^{-1}\ y\ x$
using *assms* **unfolding** *rel-inv-def* **.**

lemma *rel-invD* [*dest*]:
assumes $R^{-1}\ x\ y$
shows $R\ y\ x$
using *assms* **unfolding** *rel-inv-def* **.**

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

lemma *rel-inv-comp-eq* [*simp*]: $(R \circ\circ S)^{-1} = S^{-1} \circ\circ R^{-1}$
by (*intro ext*) *blast*

lemma *rel-inv-inv-eq-self* [*simp*]: $R^{-1-1} = R$
by *blast*

lemma *rel-inv-eq-iff-eq* [*iff*]: $R^{-1} = S^{-1} \longleftrightarrow R = S$
by (*blast dest*: *fun-cong*)

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

lemma *in-domI* [*intro*]:
assumes $R\ x\ y$
shows *in-dom* $R\ x$
using *assms* **unfolding** *in-dom-def* **by** *blast*

lemma *in-domE* [*elim*]:
assumes *in-dom* $R\ x$
obtains y **where** $R\ x\ y$
using *assms* **unfolding** *in-dom-def* **by** *blast*

lemma *in-dom-if-in-dom-rel-comp*:
assumes *in-dom* $(R \circ\circ S)\ x$
shows *in-dom* $R\ x$
using *assms* **by** *blast*

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

lemma *in-codomI* [*intro*]:
  assumes $R\ x\ y$
  shows *in-codom* $R\ y$
  using *assms* **unfolding** *in-codom-def* **by** *blast*

lemma *in-codomE* [*elim*]:
  assumes *in-codom* $R\ y$
  obtains x **where** $R\ x\ y$
  using *assms* **unfolding** *in-codom-def* **by** *blast*

lemma *in-codom-if-in-codom-rel-comp*:
  assumes *in-codom* $(R \circ\circ S)\ y$
  shows *in-codom* $S\ y$
  using *assms* **by** *blast*

lemma *in-codom-rel-inv-eq-in-dom* [*simp*]: *in-codom* (R^{-1}) = *in-dom* R
  by (*intro ext*) *blast*

lemma *in-dom-rel-inv-eq-in-codom* [*simp*]: *in-dom* (R^{-1}) = *in-codom* R
  by (*intro ext*) *blast*

definition *in-field* $R\ x \equiv$ *in-dom* $R\ x \lor$ *in-codom* $R\ x$

lemma *in-field-if-in-dom*:
  assumes *in-dom* $R\ x$
  shows *in-field* $R\ x$
  unfolding *in-field-def* **using** *assms* **by** *blast*

lemma *in-field-if-in-codom*:
  assumes *in-codom* $R\ x$
  shows *in-field* $R\ x$
  unfolding *in-field-def* **using** *assms* **by** *blast*

lemma *in-fieldE* [*elim*]:
  assumes *in-field* $R\ x$
  obtains (*in-dom*) x' **where** $R\ x\ x'$ | (*in-codom*) x' **where** $R\ x'\ x$
  using *assms* **unfolding** *in-field-def* **by** *blast*

lemma *in-fieldE'*:
  assumes *in-field* $R\ x$
  obtains (*in-dom*) *in-dom* $R\ x$ | (*in-codom*) *in-codom* $R\ x$
  using *assms* **by** *blast*

lemma *in-fieldI* [*intro*]:
  assumes $R\ x\ y$
  shows *in-field* $R\ x$ *in-field* $R\ y$
  using *assms* **by** (*auto intro*: *in-field-if-in-dom in-field-if-in-codom*)

lemma *in-field-iff-in-dom-or-in-codom*:
  in-field L x $\longleftrightarrow$ *in-dom* L x $\vee$ *in-codom* L x
  by *blast*

lemma *in-field-rel-inv-eq* [*simp*]: *in-field* R^{-1} = *in-field* R
  by (*intro ext*) *auto*

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
  using *assms* **by** *blast*

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
  using *assms* **by** (*intro ext*) (*auto elim*: *in-fieldE'*)

definition *rel-if* B R x y $\equiv$ B $\longrightarrow$ R x y

bundle *rel-if-syntax* **begin notation** (**output**) *rel-if* (**infixl** $\longrightarrow$ *50*) **end**
bundle *no-rel-if-syntax* **begin no-notation** (**output**) *rel-if* (**infixl** $\longrightarrow$ *50*) **end**
unbundle *rel-if-syntax*

lemma *rel-if-eq-rel-if-pred* [*simp*]:
  assumes B
  shows (*rel-if* B R) = R
  unfolding *rel-if-def* **using** *assms* **by** *blast*

lemma *rel-if-eq-top-if-not-pred* [*simp*]:
  assumes $\neg B$
  shows (*rel-if* B R) = (λ- -. *True*)
  unfolding *rel-if-def* **using** *assms* **by** *blast*

lemma *rel-if-if-impI* [*intro*]:
  assumes $B \Longrightarrow R$ x y
  shows (*rel-if* B R) x y
  unfolding *rel-if-def* **using** *assms* **by** *blast*

lemma *rel-ifE* [*elim*]:
  assumes (*rel-if* B R) x y
  obtains $\neg B$ | B R x y
  using *assms* **unfolding** *rel-if-def* **by** *blast*

lemma *rel-ifD*:
  assumes (*rel-if* B R) x y
  and B
  shows R x y
  using *assms* **by** *blast*

consts *restrict-left* :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow 'c \Rightarrow 'a \Rightarrow 'b \Rightarrow bool$

definition *restrict-right* $R\ P \equiv (restrict\text{-}left\ R^{-1}\ P)^{-1}$

overloading
restrict-left-pred $\equiv$ *restrict-left* :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow ('a \Rightarrow bool) \Rightarrow 'a \Rightarrow 'b \Rightarrow bool$
begin
definition *restrict-left-pred* $R\ P\ x\ y \equiv P\ x \wedge R\ x\ y$
end

bundle *restrict-syntax*
begin
notation *restrict-left* ((-)↾(-) [*1000*])
notation *restrict-right* ((-)↿(-) [*1000*])
end
bundle *no-restrict-syntax*
begin
no-notation *restrict-left* ((-)↾(-) [*1000*])
no-notation *restrict-right* ((-)↿(-) [*1000*])
end
unbundle *restrict-syntax*

lemma *restrict-leftI* [*intro*]:
assumes $R\ x\ y$
and $P\ x$
shows $R{\restriction_P}\ x\ y$
using *assms* **unfolding** *restrict-left-pred-def* **by** *blast*

lemma *restrict-leftE* [*elim*]:
assumes $R{\restriction_P}\ x\ y$
obtains $P\ x\ R\ x\ y$
using *assms* **unfolding** *restrict-left-pred-def* **by** *blast*

lemma *restrict-right-eq*: $R{\upharpoonleft_P} = ((R^{-1}){\restriction_P})^{-1}$
unfolding *restrict-right-def* **..**

lemma *rel-inv-restrict-right-rel-inv-eq-restrict-left* [*simp*]: $((R^{-1}){\upharpoonleft_P})^{-1} = R{\restriction_P}$
by (*simp add*: *restrict-right-eq*)

lemma *restrict-right-iff-restrict-left*: $R{\upharpoonleft_P}\ x\ y = (R^{-1}){\restriction_P}\ y\ x$
unfolding *restrict-right-eq* **by** *simp*

lemma *restrict-rightI* [*intro*]:
assumes $R\ x\ y$
and $P\ y$
shows $R{\upharpoonleft_P}\ x\ y$
using *assms* **by** (*auto iff*: *restrict-right-iff-restrict-left*)

lemma *restrict-rightE* [*elim*]:
assumes $R\restriction_P\ x\ y$
obtains $P\ y\ R\ x\ y$
using *assms* **by** (*auto iff*: *restrict-right-iff-restrict-left*)

lemma *rel-inv-restrict-left-inv-restrict-left-eq*:
fixes $R :: 'a \Rightarrow 'b \Rightarrow bool$ **and** $P :: 'a \Rightarrow bool$ **and** $Q :: 'b \Rightarrow bool$
shows $(((R\restriction_P)^{-1})\restriction_Q)^{-1} = (((R^{-1})\restriction_Q)^{-1})\restriction_P$
by (*intro ext iffI restrict-leftI rel-invI*) *auto*

lemma *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 $R\restriction_P\restriction_Q = R\restriction_Q\restriction_P$
unfolding *restrict-right-eq*
by (*fact rel-inv-restrict-left-inv-restrict-left-eq*)

lemma *in-dom-restrict-leftI* [*intro*]:
assumes $R\ x\ y$
and $P\ x$
shows *in-dom* $R\restriction_P\ x$
using *assms* **by** *blast*

lemma *in-dom-restrict-left-if-in-dom*:
assumes *in-dom* $R\ x$
and $P\ x$
shows *in-dom* $R\restriction_P\ x$
using *assms* **by** *blast*

lemma *in-dom-restrict-leftE* [*elim*]:
assumes *in-dom* $R\restriction_P\ x$
obtains y **where** $P\ x\ R\ x\ y$
using *assms* **by** *blast*

lemma *in-codom-restrict-leftI* [*intro*]:
assumes $R\ x\ y$
and $P\ x$
shows *in-codom* $R\restriction_P\ y$
using *assms* **by** *blast*

lemma *in-codom-restrict-leftE* [*elim*]:
assumes *in-codom* $R\restriction_P\ y$
obtains x **where** $P\ x\ R\ x\ y$
using *assms* **by** *blast*

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* $f\ g\ R\ x\ y = R\ (f\ x)\ (g\ y)$
unfolding *rel-bimap-def* **by** *simp*

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*
  unfolding *rel-map-def* **by** *simp*

lemma *rel-map-eq* [*simp*]: *rel-map f R x y* $=$ *R* (*f x*) (*f y*)
  by (*simp only*: *rel-bimap-self-eq-rel-map*[*symmetric*] *rel-bimap-eq*)

end

1.1.2 Order

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

lemma *le-relI* [*intro*]:
  assumes $\bigwedge x\ y.\ R\ x\ y \Longrightarrow S\ x\ y$
  shows $R \leq S$
  using *assms* **by** (*rule predicate2I*)

lemma *le-relD* [*dest*]:
  assumes $R \leq S$
  and $R\ x\ y$
  shows $S\ x\ y$
  using *assms* **by** (*rule predicate2D*)

lemma *le-relE*:
  assumes $R \leq S$
  and $R\ x\ y$
  obtains $S\ x\ y$
  using *assms* **by** *blast*

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

lemma *restrict-left-top-eq* [*simp*]: $(R :: {'a} \Rightarrow \text{-})\restriction_{(\top\ ::\ {'a} \Rightarrow \mathit{bool})} = R$
  by (*intro ext*) *auto*

end

1.1.3 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 $(R \sqcap S)\ x\ y$
using *assms* **by** (*rule inf2I*)

lemma *rel-infE* [*elim*]:
assumes $(R \sqcap S)\ x\ y$
obtains $R\ x\ y$ $S\ x\ y$
using *assms* **by** (*rule inf2E*)

lemma *rel-infD*:
assumes $(R \sqcap S)\ x\ y$
shows $R\ x\ y$ **and** $S\ x\ y$
using *assms* **by** *auto*

lemma *in-dom-rel-infI* [*intro*]:
assumes $R\ x\ y$
and $S\ x\ y$
shows *in-dom* $(R \sqcap S)\ x$
using *assms* **by** *blast*

lemma *in-dom-rel-infE* [*elim*]:
assumes *in-dom* $(R \sqcap S)\ x$
obtains y **where** $R\ x\ y$ $S\ x\ y$
using *assms* **by** *blast*

lemma *in-codom-rel-infI* [*intro*]:
assumes $R\ x\ y$
and $S\ x\ y$
shows *in-codom* $(R \sqcap S)\ y$
using *assms* **by** *blast*

lemma *in-codom-rel-infE* [*elim*]:
assumes *in-codom* $(R \sqcap S)\ y$
obtains x **where** $R\ x\ y$ $S\ x\ y$
using *assms* **by** *blast*

lemma *in-field-eq-in-dom-sup-in-codom*: *in-field* $L =$ (*in-dom* $L \sqcup$ *in-codom* L)
by (*intro ext*) (*simp add*: *in-field-iff-in-dom-or-in-codom*)

lemma *in-dom-restrict-left-eq* [*simp*]: *in-dom* $R{\restriction}_P =$ (*in-dom* $R \sqcap P$)
by (*intro ext*) *auto*

lemma *in-codom-restrict-left-eq* [*simp*]: *in-codom* $R{\restriction}_P = (\textit{in-codom}\ R \sqcap P)$
  by (*intro ext*) *auto*

lemma *restrict-left-restrict-left-eq* [*simp*]:
  fixes $R :: {'a} \Rightarrow$ - **and** $P\ Q :: {'a} \Rightarrow \textit{bool}$
  shows $R{\restriction}_P{\restriction}_Q = R{\restriction}_P \sqcap R{\restriction}_Q$
  by (*intro ext iffI restrict-leftI*) *auto*

lemma *restrict-left-restrict-right-eq* [*simp*]:
  fixes $R :: {'a} \Rightarrow {'b} \Rightarrow \textit{bool}$ **and** $P :: {'a} \Rightarrow \textit{bool}$ **and** $Q :: {'b} \Rightarrow \textit{bool}$
  shows $R{\restriction}_P{\upharpoonright}_Q = R{\restriction}_P \sqcap R{\upharpoonright}_Q$
  by (*intro ext iffI restrict-leftI restrict-rightI*) *auto*

lemma *restrict-right-restrict-left-eq* [*simp*]:
  fixes $R :: {'a} \Rightarrow {'b} \Rightarrow \textit{bool}$ **and** $P :: {'b} \Rightarrow \textit{bool}$ **and** $Q :: {'a} \Rightarrow \textit{bool}$
  shows $R{\upharpoonright}_P{\restriction}_Q = R{\upharpoonright}_P \sqcap R{\restriction}_Q$
  by (*intro ext iffI restrict-leftI restrict-rightI*) *auto*

lemma *restrict-right-restrict-right-eq* [*simp*]:
  fixes $R :: {'a} \Rightarrow {'b} \Rightarrow \textit{bool}$ **and** $P\ Q :: {'b} \Rightarrow \textit{bool}$
  shows $R{\upharpoonright}_P{\upharpoonright}_Q = R{\upharpoonright}_P \sqcap R{\upharpoonright}_Q$
  by (*intro ext iffI*) *auto*

lemma *restrict-left-sup-eq* [*simp*]: $(R :: {'a} \Rightarrow -){\restriction}_{((P :: {'a} \Rightarrow \textit{bool}) \sqcup Q)} = R{\restriction}_P \sqcup R{\restriction}_Q$
  by (*intro antisym le-relI*) (*auto elim*!: *restrict-leftE*)

lemma *restrict-left-inf-eq* [*simp*]: $(R :: {'a} \Rightarrow -){\restriction}_{((P :: {'a} \Rightarrow \textit{bool}) \sqcap Q)} = R{\restriction}_P \sqcap R{\restriction}_Q$
  by (*intro antisym le-relI*) (*auto elim*!: *restrict-leftE*)

lemma *inf-rel-bimap-and-eq-restrict-left-restrict-right*:
  $R \sqcap (\textit{rel-bimap}\ P\ Q\ (\wedge)) = R{\restriction}_P{\upharpoonright}_Q$
  by (*intro ext*) *auto*

end

1.2 Functions

1.2.1 Basic Functions

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

definition $\textit{id}\ x \equiv x$

lemma *id-eq-self* [*simp*]: *id* $x = x$
  unfolding *id-def* **..**

definition *comp* $f\ g\ x \equiv f\ (g\ x)$

bundle *comp-syntax* **begin notation** *comp* (**infixl** $\circ$ *55*) **end**
bundle *no-comp-syntax* **begin no-notation** *comp* (**infixl** $\circ$ *55*) **end**
unbundle *comp-syntax*

lemma *comp-eq* [*simp*]: $(f \circ g)\ x = f\ (g\ x)$
  unfolding *comp-def* **..**

lemma *id-comp-eq* [*simp*]: *id* $\circ f = f$
  by (*rule ext*) *simp*

lemma *comp-id-eq* [*simp*]: $f \circ$ *id* $= f$
  by (*rule ext*) *simp*

definition *dep-fun-map* $f\ g\ h\ x \equiv g\ x\ (f\ x)\ (h\ (f\ x))$

abbreviation *fun-map* $f\ g\ h \equiv$ *dep-fun-map* $f\ (\lambda\text{- -}.\ g)\ h$

bundle *dep-fun-map-syntax* **begin**
syntax
  -fun-map :: $('a \Rightarrow 'b) \Rightarrow ('c \Rightarrow 'd) \Rightarrow ('b \Rightarrow 'c) \Rightarrow$
    $('a \Rightarrow 'd)$ ((-) $\rightarrow$ (-) [*41*, *40*] *40*)
  -dep-fun-map :: *idt* $\Rightarrow ('a \Rightarrow 'b) \Rightarrow ('c \Rightarrow 'd) \Rightarrow ('b \Rightarrow 'c) \Rightarrow$
    $('a \Rightarrow 'd)$ ([-/ : / -] $\rightarrow$ (-) [*41*, *41*, *40*] *40*)
end
bundle *no-dep-fun-map-syntax* **begin**
no-syntax
  -fun-map :: $('a \Rightarrow 'b) \Rightarrow ('c \Rightarrow 'd) \Rightarrow ('b \Rightarrow 'c) \Rightarrow$
    $('a \Rightarrow 'd)$ ((-) $\rightarrow$ (-) [*41*, *40*] *40*)
  -dep-fun-map :: *idt* $\Rightarrow ('a \Rightarrow 'b) \Rightarrow ('c \Rightarrow 'd) \Rightarrow ('b \Rightarrow 'c) \Rightarrow$
    $('a \Rightarrow 'd)$ ([-/ : / -] $\rightarrow$ (-) [*41*, *41*, *40*] *40*)
end
unbundle *dep-fun-map-syntax*
translations
  $f \rightarrow g \rightleftharpoons$ *CONST fun-map* $f\ g$
  $[x : f] \rightarrow g \rightleftharpoons$ *CONST dep-fun-map* $f\ (\lambda x.\ g)$

lemma *dep-fun-map-eq* [*simp*]: $([x : f] \rightarrow g\ x)\ h\ x = g\ x\ (f\ x)\ (h\ (f\ x))$
  unfolding *dep-fun-map-def* **..**

lemma *fun-map-eq-comp* [*simp*]: $(f \rightarrow g)\ h = g \circ h \circ f$
  by *fastforce*

lemma *fun-map-eq* [*simp*]: $(f \rightarrow g)\ h\ x = g\ (h\ (f\ x))$

unfolding *fun-map-eq-comp* **by** *simp*

lemma *fun-map-id-eq-comp* [*simp*]: *fun-map id* = (∘)
by (*intro ext*) *simp*

lemma *fun-map-id-eq-comp′* [*simp*]: (*f* → *id*) *h* = *h* ∘ *f*
by (*intro ext*) *simp*

end

1.2.2 Lattice Syntax

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

bundle *lattice-syntax* — copied from theory Main
begin
notation
bot (⊥)
and *top* (⊤)
and *inf* (**infixl** ⊓ *70*)
and *sup* (**infixl** ⊔ *65*)
end
bundle *no-lattice-syntax*
begin
no-notation
bot (⊥)
and *top* (⊤)
and *inf* (**infixl** ⊓ *70*)
and *sup* (**infixl** ⊔ *65*)
end

unbundle *lattice-syntax*

end

1.2.3 Lattice

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

lemma *inf-predI* [*intro*]:

assumes $P\ x$
and $Q\ x$
shows $(P \sqcap Q)\ x$
using *assms* **by** (*intro inf1I*)

lemma *inf-predE* [*elim*]:
assumes $(P \sqcap Q)\ x$
obtains $P\ x\ Q\ x$
using *assms* **by** (*rule inf1E*)

lemma *inf-predD*:
assumes $(P \sqcap Q)\ x$
shows $P\ x$ **and** $Q\ x$
using *assms* **by** *auto*

end

1.2.4 Relators

theory *Function-Relators*
imports
Binary-Relation-Functions
Functions-Base
Predicates-Lattice
begin

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

definition *Dep-Fun-Rel-rel* $R\ S\ f\ g \equiv \forall x\ y.\ R\ x\ y \longrightarrow S\ x\ y\ (f\ x)\ (g\ y)$

abbreviation *Fun-Rel-rel* $R\ S \equiv$ *Dep-Fun-Rel-rel* $R\ (\lambda\text{- -}.\ S)$

definition *Dep-Fun-Rel-pred* $P\ R\ f\ g \equiv \forall x.\ P\ x \longrightarrow R\ x\ (f\ x)\ (g\ x)$

abbreviation *Fun-Rel-pred* $P\ R \equiv$ *Dep-Fun-Rel-pred* $P\ (\lambda\text{-}.\ R)$

bundle *Dep-Fun-Rel-syntax* **begin**
syntax
-Fun-Rel-rel :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow ('c \Rightarrow 'd \Rightarrow bool) \Rightarrow ('a \Rightarrow 'c) \Rightarrow$
$('b \Rightarrow 'd) \Rightarrow bool$ ((-) $\Rrightarrow$ (-) [*41*, *40*] *40*)
-Dep-Fun-Rel-rel :: $idt \Rightarrow idt \Rightarrow ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow ('c \Rightarrow 'd \Rightarrow bool) \Rightarrow$
$('a \Rightarrow 'c) \Rightarrow ('b \Rightarrow 'd) \Rightarrow bool$ ([-/ -/ ::/ -] $\Rrightarrow$ (-) [*41*, *41*, *41*, *40*] *40*)
-Dep-Fun-Rel-rel-if :: $idt \Rightarrow idt \Rightarrow ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool \Rightarrow ('c \Rightarrow 'd \Rightarrow bool)$
$\Rightarrow$
$('a \Rightarrow 'c) \Rightarrow ('b \Rightarrow 'd) \Rightarrow bool$ ([-/ -/ ::/ -/ |/ -] $\Rrightarrow$ (-) [*41*, *41*, *41*, *41*, *40*] *40*)
-Fun-Rel-pred :: $('a \Rightarrow bool) \Rightarrow ('b \Rightarrow 'c \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow$
$('a \Rightarrow 'c) \Rightarrow bool$ ([-] $\Rrightarrow$ (-) [*41*, *40*] *40*)
-Dep-Fun-Rel-pred :: $idt \Rightarrow ('a \Rightarrow bool) \Rightarrow ('b \Rightarrow 'c \Rightarrow bool) \Rightarrow$

$('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'c) \Rightarrow bool$ ([-/ ::/ -] $\Rrightarrow$ (-) [41, 41, 40] 40)
  -*Dep-Fun-Rel-pred-if* :: $idt \Rightarrow ('a \Rightarrow bool) \Rightarrow bool \Rightarrow ('b \Rightarrow 'c \Rightarrow bool) \Rightarrow$
    $('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'c) \Rightarrow bool$ ([-/ ::/ -/ |/ -] $\Rrightarrow$ (-) [41, 41, 41, 40] 40)
end
bundle *no-Dep-Fun-Rel-syntax* **begin**
no-syntax
  -*Fun-Rel-rel* :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow ('c \Rightarrow 'd \Rightarrow bool) \Rightarrow ('a \Rightarrow 'c) \Rightarrow$
    $('b \Rightarrow 'd) \Rightarrow bool$ ((-) $\Rrightarrow$ (-) [41, 40] 40)
  -*Dep-Fun-Rel-rel* :: $idt \Rightarrow idt \Rightarrow ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow ('c \Rightarrow 'd \Rightarrow bool) \Rightarrow$
    $('a \Rightarrow 'c) \Rightarrow ('b \Rightarrow 'd) \Rightarrow bool$ ([-/ -/ ::/ -] $\Rrightarrow$ (-) [41, 41, 41, 40] 40)
  -*Dep-Fun-Rel-rel-if* :: $idt \Rightarrow idt \Rightarrow ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool \Rightarrow ('c \Rightarrow 'd \Rightarrow bool)$
$\Rightarrow$
    $('a \Rightarrow 'c) \Rightarrow ('b \Rightarrow 'd) \Rightarrow bool$ ([-/ -/ ::/ -/ |/ -] $\Rrightarrow$ (-) [41, 41, 41, 41, 40] 40)
  -*Fun-Rel-pred* :: $('a \Rightarrow bool) \Rightarrow ('b \Rightarrow 'c \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow$
    $('a \Rightarrow 'c) \Rightarrow bool$ ([-] $\Rrightarrow$ (-) [41, 40] 40)
  -*Dep-Fun-Rel-pred* :: $idt \Rightarrow ('a \Rightarrow bool) \Rightarrow ('b \Rightarrow 'c \Rightarrow bool) \Rightarrow$
    $('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'c) \Rightarrow bool$ ([-/ ::/ -] $\Rrightarrow$ (-) [41, 41, 40] 40)
  -*Dep-Fun-Rel-pred-if* :: $idt \Rightarrow ('a \Rightarrow bool) \Rightarrow bool \Rightarrow ('b \Rightarrow 'c \Rightarrow bool) \Rightarrow$
    $('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'c) \Rightarrow bool$ ([-/ ::/ -/ |/ -] $\Rrightarrow$ (-) [41, 41, 41, 40] 40)
end
unbundle *Dep-Fun-Rel-syntax*
translations
  $R \Rrightarrow S \rightleftharpoons$ *CONST Fun-Rel-rel R S*
  $[x\ y :: R] \Rrightarrow S \rightleftharpoons$ *CONST Dep-Fun-Rel-rel R* $(\lambda x\ y.\ S)$
  $[x\ y :: R \mid B] \Rrightarrow S \rightleftharpoons$ *CONST Dep-Fun-Rel-rel R* $(\lambda x\ y.$ *CONST rel-if B S*$)$
  $[P] \Rrightarrow R \rightleftharpoons$ *CONST Fun-Rel-pred P R*
  $[x :: P] \Rrightarrow R \rightleftharpoons$ *CONST Dep-Fun-Rel-pred P* $(\lambda x.\ R)$
  $[x :: P \mid B] \Rrightarrow R \rightleftharpoons$ *CONST Dep-Fun-Rel-pred P* $(\lambda x.$ *CONST rel-if B R*$)$

lemma *Dep-Fun-Rel-relI* [*intro*]:
  assumes $\bigwedge x\ y.\ R\ x\ y \Longrightarrow S\ x\ y\ (f\ x)\ (g\ y)$
  shows $([x\ y :: R] \Rrightarrow S\ x\ y)\ f\ g$
  unfolding *Dep-Fun-Rel-rel-def* **using** *assms* **by** *blast*

lemma *Dep-Fun-Rel-relD*:
  assumes $([x\ y :: R] \Rrightarrow S\ x\ y)\ f\ g$
  and $R\ x\ y$
  shows $S\ x\ y\ (f\ x)\ (g\ y)$
  using *assms* **unfolding** *Dep-Fun-Rel-rel-def* **by** *blast*

lemma *Dep-Fun-Rel-relE* [*elim*]:
  assumes $([x\ y :: R] \Rrightarrow S\ x\ y)\ f\ g$
  and $R\ x\ y$
  obtains $S\ x\ y\ (f\ x)\ (g\ y)$
  using *assms* **unfolding** *Dep-Fun-Rel-rel-def* **by** *blast*

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

unfolding *Dep-Fun-Rel-pred-def* **using** *assms* **by** *blast*

lemma *Dep-Fun-Rel-predD*:
assumes $([x :: P] \Rrightarrow R\ x)\ f\ g$
and $P\ x$
shows $R\ x\ (f\ x)\ (g\ x)$
using *assms* **unfolding** *Dep-Fun-Rel-pred-def* **by** *blast*

lemma *Dep-Fun-Rel-predE* [*elim*]:
assumes $([x :: P] \Rrightarrow R\ x)\ f\ g$
and $P\ x$
obtains $R\ x\ (f\ x)\ (g\ x)$
using *assms* **unfolding** *Dep-Fun-Rel-pred-def* **by** *blast*

lemma *rel-inv-Dep-Fun-Rel-rel-eq* [*simp*]:
$([x\ y :: R] \Rrightarrow S\ x\ y)^{-1} = ([y\ x :: R^{-1}] \Rrightarrow (S\ x\ y)^{-1})$
by (*intro ext*) *auto*

lemma *rel-inv-Dep-Fun-Rel-pred-eq* [*simp*]:
$([x :: P] \Rrightarrow R\ x)^{-1} = ([x :: P] \Rrightarrow (R\ x)^{-1})$
by (*intro ext*) *auto*

lemma *Dep-Fun-Rel-pred-eq-Dep-Fun-Rel-rel*:
$([x :: P] \Rrightarrow R\ x) = ([x\ \text{-} :: (((\sqcap)\ P) \circ (=))] \Rrightarrow R\ x)$
by (*intro ext*) (*auto intro*!: *Dep-Fun-Rel-predI Dep-Fun-Rel-relI*)

lemma *Fun-Rel-eq-eq-eq* [*simp*]: $((=) \Rrightarrow (=)) = (=)$
by (*intro ext*) *auto*

Composition **lemma** *Dep-Fun-Rel-rel-compI*:
assumes *Dep-Fun-Rel1*: $([x\ y :: R] \Rrightarrow S\ x\ y)\ f\ g$
and *Dep-Fun-Rel2*: $\bigwedge x\ y.\ R\ x\ y \Longrightarrow ([x'\ y' :: T\ x\ y] \Rrightarrow U\ x\ y\ x'\ y')\ f'\ g'$
and *le*: $\bigwedge x\ y.\ R\ x\ y \Longrightarrow S\ x\ y\ (f\ x)\ (g\ y) \Longrightarrow T\ x\ y\ (f\ x)\ (g\ y)$
shows $([x\ y :: R] \Rrightarrow U\ x\ y\ (f\ x)\ (g\ y))\ (f' \circ f)\ (g' \circ g)$
using *assms* **by** (*intro Dep-Fun-Rel-relI*) (*auto 6 0*)

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 $([x\ y :: R] \Rrightarrow T\ x\ y\ (f\ x)\ (g\ y))\ (f' \circ f)\ (g' \circ g)$
using *assms* **by** (*intro Dep-Fun-Rel-rel-compI*)

lemma *Dep-Fun-Rel-pred-comp-Dep-Fun-Rel-rel-compI*:
assumes *Dep-Fun-Rel1*: $([x :: P] \Rrightarrow R\ x)\ f\ g$
and *Dep-Fun-Rel2*: $\bigwedge x.\ P\ x \Longrightarrow ([x'\ y' :: S\ x] \Rrightarrow T\ x\ x'\ y')\ f'\ g'$
and *le*: $\bigwedge x.\ P\ x \Longrightarrow R\ x\ (f\ x)\ (g\ x) \Longrightarrow S\ x\ (f\ x)\ (g\ x)$
shows $([x :: P] \Rrightarrow T\ x\ (f\ x)\ (g\ x))\ (f' \circ f)\ (g' \circ g)$
using *assms* **by** (*intro Dep-Fun-Rel-predI*) (*auto 6 0*)

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 $([x :: P] \Rrightarrow S\ x\ (f\ x)\ (g\ x))\ (f' \circ f)\ (g' \circ g)$
using *assms* **by** (*intro Dep-Fun-Rel-pred-comp-Dep-Fun-Rel-rel-compI*)

Restrictions **lemma** *restrict-left-Dep-Fun-Rel-rel-restrict-left-eq*:
fixes $R :: {'a1} \Rightarrow {'a2} \Rightarrow bool$
and $S :: {'a1} \Rightarrow {'a2} \Rightarrow {'b1} \Rightarrow {'b2} \Rightarrow bool$
and $P :: {'a1} \Rightarrow {'a2} \Rightarrow {'b1} \Rightarrow bool$
assumes $\bigwedge f\ x\ y.\ Q\ f \Longrightarrow R\ x\ y \Longrightarrow P\ x\ y\ (f\ x)$
shows $([x\ y :: R] \Rrightarrow (S\ x\ y)\restriction_{P\ x\ y})\restriction_Q = ([x\ y :: R] \Rrightarrow S\ x\ y)\restriction_Q$
using *assms* **by** (*intro ext iffI restrict-leftI Dep-Fun-Rel-relI*)
(*auto dest*!: *Dep-Fun-Rel-relD*)

lemma *restrict-right-Dep-Fun-Rel-rel-restrict-right-eq*:
fixes $R :: {'a1} \Rightarrow {'a2} \Rightarrow bool$
and $S :: {'a1} \Rightarrow {'a2} \Rightarrow {'b1} \Rightarrow {'b2} \Rightarrow bool$
and $P :: {'a1} \Rightarrow {'a2} \Rightarrow {'b2} \Rightarrow bool$
assumes $\bigwedge f\ x\ y.\ Q\ f \Longrightarrow R\ x\ y \Longrightarrow P\ x\ y\ (f\ y)$
shows $(([x\ y :: R] \Rrightarrow (S\ x\ y)\upharpoonright_{P\ x\ y})\upharpoonright_Q) = (([x\ y :: R] \Rrightarrow S\ x\ y)\upharpoonright_Q)$
unfolding *restrict-right-eq*
using *assms restrict-left-Dep-Fun-Rel-rel-restrict-left-eq*[**where** $?R=R^{-1}$
and $?S=\lambda y\ x.\ (S\ x\ y)^{-1}$]
by *simp*

end

1.2.5 Orders

theory *Predicates-Order*
imports
HOL.Orderings
begin

lemma *le-predI* [*intro*]:
assumes $\bigwedge x.\ P\ x \Longrightarrow Q\ x$
shows $P \le Q$
using *assms* **by** (*rule predicate1I*)

lemma *le-predD* [*dest*]:
assumes $P \le Q$
and $P\ x$
shows $Q\ x$
using *assms* **by** (*rule predicate1D*)

lemma *le-predE*:
assumes $P \le Q$

and $P\ x$
obtains $Q\ x$
using *assms* **by** *blast*

end

1.3 Predicates

theory *Predicates*
imports
Functions-Base
Predicates-Order
Predicates-Lattice
begin

Summary Basic concepts on predicates.

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

lemma *pred-map-eq* [*simp*]: *pred-map* $f\ P\ x = P\ (f\ x)$
unfolding *pred-map-def* **by** *simp*

lemma *comp-eq-pred-map* [*simp*]: $P \circ f =$ *pred-map* $f\ P$
by (*intro ext*) *simp*

end

Monotonicity

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

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

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

definition *dep-mono-wrt-rel* $R\ S\ f \equiv ([x\ y :: R] \Rrightarrow S\ x\ y)\ f\ f$

abbreviation *mono-wrt-rel* $R\ S \equiv$ *dep-mono-wrt-rel* $R\ (\lambda\text{- -}.\ S)$

definition *dep-mono-wrt-pred* $P\ Q\ f \equiv ([x :: P] \Rrightarrow (\lambda\text{-}.\ Q\ x))\ f\ f$

abbreviation *mono-wrt-pred P Q* ≡ *dep-mono-wrt-pred P* (λ-. *Q*)

bundle *dep-mono-wrt-syntax* **begin**
syntax
  -mono-wrt-rel :: (′*a* ⇒ ′*a* ⇒ *bool*) ⇒ (′*b* ⇒ ′*b* ⇒ *bool*) ⇒ (′*a* ⇒ ′*b*) ⇒
    bool ((-) ⇛$_m$ (-) [*41*, *40*] *40*)
  -dep-mono-wrt-rel :: *idt* ⇒ *idt* ⇒ (′*a* ⇒ ′*a* ⇒ *bool*) ⇒ (′*b* ⇒ ′*b* ⇒ *bool*) ⇒
    (′*a* ⇒ ′*b*) ⇒ *bool* ([-/ -/ ::/ -] ⇛$_m$ (-) [*41*, *41*, *41*, *40*] *40*)
  -dep-mono-wrt-rel-if :: *idt* ⇒ *idt* ⇒ (′*a* ⇒ ′*a* ⇒ *bool*) ⇒ *bool* ⇒ (′*b* ⇒ ′*b* ⇒ *bool*)
⇒
    (′*a* ⇒ ′*b*) ⇒ *bool* ([-/ -/ ::/ -/ |/ -] ⇛$_m$ (-) [*41*, *41*, *41*, *41*, *40*] *40*)
  -mono-wrt-pred :: (′*a* ⇒ *bool*) ⇒ (′*b* ⇒ ′*b* ⇒ *bool*) ⇒ (′*a* ⇒ ′*b*) ⇒
    bool ([-] ⇛$_m$ (-) [*41*, *40*] *40*)
  -dep-mono-wrt-pred :: *idt* ⇒ (′*a* ⇒ *bool*) ⇒ (′*b* ⇒ ′*b* ⇒ *bool*) ⇒
    (′*a* ⇒ ′*b*) ⇒ *bool* ([-/ ::/ -] ⇛$_m$ (-) [*41*, *41*, *40*] *40*)

end
bundle *no-dep-mono-wrt-syntax* **begin**
no-syntax
  -mono-wrt-rel :: (′*a* ⇒ ′*a* ⇒ *bool*) ⇒ (′*b* ⇒ ′*b* ⇒ *bool*) ⇒ (′*a* ⇒ ′*b*) ⇒
    bool ((-) ⇛$_m$ (-) [*41*, *40*] *40*)
  -dep-mono-wrt-rel :: *idt* ⇒ *idt* ⇒ (′*a* ⇒ ′*a* ⇒ *bool*) ⇒ (′*b* ⇒ ′*b* ⇒ *bool*) ⇒
    (′*a* ⇒ ′*b*) ⇒ *bool* ([-/ -/ ::/ -] ⇛$_m$ (-) [*41*, *41*, *41*, *40*] *40*)
  -dep-mono-wrt-rel-if :: *idt* ⇒ *idt* ⇒ (′*a* ⇒ ′*a* ⇒ *bool*) ⇒ *bool* ⇒ (′*b* ⇒ ′*b* ⇒ *bool*)
⇒
    (′*a* ⇒ ′*b*) ⇒ *bool* ([-/ -/ ::/ -/ |/ -] ⇛$_m$ (-) [*41*, *41*, *41*, *41*, *40*] *40*)
  -mono-wrt-pred :: (′*a* ⇒ *bool*) ⇒ (′*b* ⇒ ′*b* ⇒ *bool*) ⇒ (′*a* ⇒ ′*b*) ⇒
    bool ([-] ⇛$_m$ (-) [*41*, *40*] *40*)
  -dep-mono-wrt-pred :: *idt* ⇒ (′*a* ⇒ *bool*) ⇒ (′*b* ⇒ ′*b* ⇒ *bool*) ⇒
    (′*a* ⇒ ′*b*) ⇒ *bool* ([-/ ::/ -] ⇛$_m$ (-) [*41*, *41*, *40*] *40*)

end
unbundle *dep-mono-wrt-syntax*
translations
  R ⇛$_m$ *S* ⇌ *CONST mono-wrt-rel R S*
  [*x y* :: *R*] ⇛$_m$ *S* ⇌ *CONST dep-mono-wrt-rel R* (λ*x y*. *S*)
  [*x y* :: *R* | *B*] ⇛$_m$ *S* ⇌ *CONST dep-mono-wrt-rel R* (λ*x y*. *CONST rel-if B S*)
  [*P*] ⇛$_m$ *Q* ⇌ *CONST mono-wrt-pred P Q*
  [*x* :: *P*] ⇛$_m$ *Q* ⇌ *CONST dep-mono-wrt-pred P* (λ*x*. *Q*)

lemma *dep-mono-wrt-relI* [*intro*]:
  assumes ⋀*x y*. *R x y* ⟹ *S x y* (*f x*) (*f y*)
  shows ([*x y* :: *R*] ⇛$_m$ *S x y*) *f*
  using *assms* **unfolding** *dep-mono-wrt-rel-def* **by** *blast*

lemma *dep-mono-wrt-relE* [*elim*]:
  assumes $([x\ y :: R] \Rrightarrow_m S\ x\ y)\ f$
  and $R\ x\ y$
  obtains $S\ x\ y\ (f\ x)\ (f\ y)$
  using *assms* **unfolding** *dep-mono-wrt-rel-def* **by** *blast*

lemma *dep-mono-wrt-relD*:
  assumes $([x\ y :: R] \Rrightarrow_m S\ x\ y)\ f$
  and $R\ x\ y$
  shows $S\ x\ y\ (f\ x)\ (f\ y)$
  using *assms* **by** *blast*

lemma *dep-mono-wrt-predI* [*intro*]:
  assumes $\bigwedge x.\ P\ x \Longrightarrow Q\ x\ (f\ x)$
  shows $([x :: P] \Rrightarrow_m Q\ x)\ f$
  using *assms* **unfolding** *dep-mono-wrt-pred-def* **by** *blast*

lemma *dep-mono-wrt-predE* [*elim*]:
  assumes $([x :: P] \Rrightarrow_m Q\ x)\ f$
  and $P\ x$
  obtains $Q\ x\ (f\ x)$
  using *assms* **unfolding** *dep-mono-wrt-pred-def* **by** *blast*

lemma *dep-mono-wrt-predD*:
  assumes $([x :: P] \Rrightarrow_m Q\ x)\ f$
  and $P\ x$
  shows $Q\ x\ (f\ x)$
  using *assms* **by** *blast*

lemma *dep-mono-wrt-rel-if-Dep-Fun-Rel-rel-self*:
  assumes $([x\ y :: R] \Rrightarrow S\ x\ y)\ f\ f$
  shows $([x\ y :: R] \Rrightarrow_m S\ x\ y)\ f$
  using *assms* **by** *blast*

lemma *dep-mono-wrt-pred-if-Dep-Fun-Rel-pred-self*:
  assumes $([x :: P] \Rrightarrow (\lambda\text{-}.\ Q\ x))\ f\ f$
  shows $([x :: P] \Rrightarrow_m Q\ x)\ f$
  using *assms* **by** *blast*

lemma *Dep-Fun-Rel-rel-self-if-dep-mono-wrt-rel*:
  assumes $([x\ y :: R] \Rrightarrow_m S\ x\ y)\ f$
  shows $([x\ y :: R] \Rrightarrow S\ x\ y)\ f\ f$
  using *assms* **by** *blast*

lemma *Dep-Fun-Rel-pred-self-if-dep-mono-wrt-pred*:
  assumes $([x :: P] \Rrightarrow_m Q\ x)\ f$
  shows $([x :: P] \Rrightarrow (\lambda\text{-}.\ Q\ x))\ f\ f$
  using *assms* **by** *blast*

corollary *Dep-Fun-Rel-rel-self-iff-dep-mono-wrt-rel*:
$([x\ y :: R] \Rrightarrow S\ x\ y)\ f\ f \longleftrightarrow ([x\ y :: R] \Rrightarrow_m S\ x\ y)\ f$
using *dep-mono-wrt-rel-if-Dep-Fun-Rel-rel-self*
Dep-Fun-Rel-rel-self-if-dep-mono-wrt-rel **by** *blast*

corollary *Dep-Fun-Rel-pred-self-iff-dep-mono-wrt-pred*:
$([x :: P] \Rrightarrow (\lambda\text{-}.\ Q\ x))\ f\ f \longleftrightarrow ([x :: P] \Rrightarrow_m Q\ x)\ f$
using *dep-mono-wrt-pred-if-Dep-Fun-Rel-pred-self*
Dep-Fun-Rel-pred-self-if-dep-mono-wrt-pred **by** *blast*

lemma *dep-mono-wrt-rel-inv-eq* [*simp*]:
$([y\ x :: R^{-1}] \Rrightarrow_m (S\ x\ y)^{-1}) = ([x\ y :: R] \Rrightarrow_m S\ x\ y)$
by (*intro ext*) *auto*

lemma *in-dom-if-rel-if-dep-mono-wrt-rel*:
assumes $([x\ y :: R] \Rrightarrow_m S\ x\ y)\ f$
and $R\ x\ y$
shows *in-dom* $(S\ x\ y)\ (f\ x)$
using *assms* **by** (*intro in-domI*) *blast*

corollary *in-dom-if-in-dom-if-mono-wrt-rel*:
assumes $(R \Rrightarrow_m S)\ f$
shows $([\mathit{in\text{-}dom}\ R] \Rrightarrow_m \mathit{in\text{-}dom}\ S)\ f$
using *assms in-dom-if-rel-if-dep-mono-wrt-rel* **by** *fast*

lemma *in-codom-if-rel-if-dep-mono-wrt-rel*:
assumes $([x\ y :: R] \Rrightarrow_m S\ x\ y)\ f$
and $R\ x\ y$
shows *in-codom* $(S\ x\ y)\ (f\ y)$
using *assms* **by** (*intro in-codomI*) *blast*

corollary *in-codom-if-in-codom-if-mono-wrt-rel*:
assumes $(R \Rrightarrow_m S)\ f$
shows $([\mathit{in\text{-}codom}\ R] \Rrightarrow_m \mathit{in\text{-}codom}\ S)\ f$
using *assms in-dom-if-rel-if-dep-mono-wrt-rel* **by** *fast*

corollary *in-field-if-in-field-if-mono-wrt-rel*:
assumes $(R \Rrightarrow_m S)\ f$
shows $([\mathit{in\text{-}field}\ R] \Rrightarrow_m \mathit{in\text{-}field}\ S)\ f$
using *assms* **by** (*intro dep-mono-wrt-predI*) *blast*

lemma *le-rel-map-if-mono-wrt-rel*:
assumes $(R \Rrightarrow_m S)\ f$
shows $R \le \mathit{rel\text{-}map}\ f\ S$
using *assms* **by** (*intro le-relI*) *auto*

lemma *le-pred-map-if-mono-wrt-pred*:
assumes $([P] \Rrightarrow_m Q)\ f$

shows $P \leq$ *pred-map f Q*
using *assms* **by** (*intro le-predI*) *auto*

lemma *mono-wrt-rel-if-le-rel-map*:
assumes $R \leq$ *rel-map f S*
shows $(R \Rrightarrow_m S)$ *f*
using *assms* **by** (*intro dep-mono-wrt-relI*) *auto*

lemma *mono-wrt-pred-if-le-pred-map*:
assumes $P \leq$ *pred-map f Q*
shows $([P] \Rrightarrow_m Q)$ *f*
using *assms* **by** (*intro dep-mono-wrt-predI*) *auto*

corollary *mono-wrt-rel-iff-le-rel-map*: $(R \Rrightarrow_m S)$ *f* $\longleftrightarrow R \leq$ *rel-map f S*
using *mono-wrt-rel-if-le-rel-map le-rel-map-if-mono-wrt-rel* **by** *auto*

corollary *mono-wrt-pred-iff-le-pred-map*: $([P] \Rrightarrow_m Q)$ *f* $\longleftrightarrow P \leq$ *pred-map f Q*
using *mono-wrt-pred-if-le-pred-map le-pred-map-if-mono-wrt-pred* **by** *auto*

definition *mono* $\equiv ((\leq) \Rrightarrow_m (\leq))$

definition *antimono* $\equiv ((\leq) \Rrightarrow_m (\geq))$

lemma *monoI* [*intro*]:
assumes $\bigwedge x\ y.\ x \leq y \Longrightarrow f\ x \leq f\ y$
shows *mono f*
unfolding *mono-def* **using** *assms* **by** *blast*

lemma *monoE* [*elim*]:
assumes *mono f*
and $x \leq y$
obtains $f\ x \leq f\ y$
using *assms* **unfolding** *mono-def* **by** *blast*

lemma *monoD*:
assumes *mono f*
and $x \leq y$
shows $f\ x \leq f\ y$
using *assms* **by** *blast*

lemma *antimonoI* [*intro*]:
assumes $\bigwedge x\ y.\ x \leq y \Longrightarrow f\ y \leq f\ x$
shows *antimono f*
unfolding *antimono-def* **using** *assms* **by** *blast*

lemma *antimonoE* [*elim*]:
assumes *antimono f*
and $x \leq y$
obtains $f\ y \leq f\ x$

using *assms* **unfolding** *antimono-def* **by** *blast*

lemma *antimonoD*:
assumes *antimono* f
and $x \leq y$
shows $f\ y \leq f\ x$
using *assms* **by** *blast*

lemma *antimono-Dep-Fun-Rel-rel-left*: *antimono* $(\lambda R.\ [x\ y :: R] \Rrightarrow S\ x\ y)$
by (*intro antimonoI*) *auto*

lemma *antimono-Dep-Fun-Rel-pred-left*: *antimono* $(\lambda P.\ [x :: P] \Rrightarrow Q\ x)$
by (*intro antimonoI*) *auto*

lemma *antimono-dep-mono-wrt-rel-left*: *antimono* $(\lambda R.\ [x\ y :: R] \Rrightarrow_m S\ x\ y)$
by (*intro antimonoI*) *auto*

lemma *antimono-dep-mono-wrt-pred-left*: *antimono* $(\lambda P.\ [x :: P] \Rrightarrow_m Q\ x)$
by (*intro antimonoI*) *auto*

lemma *Dep-Fun-Rel-rel-if-le-left-if-Dep-Fun-Rel-rel*:
assumes $([x\ y :: R] \Rrightarrow S\ x\ y)\ f\ g$
and $T \leq R$
shows $([x\ y :: T] \Rrightarrow S\ x\ y)\ f\ g$
using *assms* **by** *blast*

lemma *Dep-Fun-Rel-pred-if-le-left-if-Dep-Fun-Rel-pred*:
assumes $([x :: P] \Rrightarrow Q\ x)\ f\ g$
and $T \leq P$
shows $([x :: T] \Rrightarrow Q\ x)\ f\ g$
using *assms* **by** *blast*

lemma *dep-mono-wrt-rel-if-le-left-if-dep-mono-wrt-rel*:
assumes $([x\ y :: R] \Rrightarrow_m S\ x\ y)\ f$
and $T \leq R$
shows $([x\ y :: T] \Rrightarrow_m S\ x\ y)\ f$
using *assms* **by** *blast*

lemma *dep-mono-wrt-pred-if-le-left-if-dep-mono-wrt-pred*:
assumes $([x :: P] \Rrightarrow_m Q\ x)\ f$
and $T \leq P$
shows $([x :: T] \Rrightarrow_m Q\ x)\ f$
using *assms* **by** *blast*

lemma *mono-Dep-Fun-Rel-rel-right*: *mono* $(\lambda S.\ [x\ y :: R] \Rrightarrow S\ x\ y)$
by (*intro monoI*) *blast*

lemma *mono-Dep-Fun-Rel-pred-right*: *mono* $(\lambda Q.\ [x :: P] \Rrightarrow Q\ x)$
by (*intro monoI*) *blast*

lemma *mono-dep-mono-wrt-rel-right*: $mono\ (\lambda S.\ [x\ y :: R] \Rrightarrow_m S\ x\ y)$
  by (*intro monoI*) *blast*

lemma *mono-dep-mono-wrt-pred-right*: $mono\ (\lambda Q.\ [x :: P] \Rrightarrow_m Q\ x)$
  by (*intro monoI*) *blast*

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 $([x\ y :: R] \Rrightarrow T\ x\ y)\ f\ g$
  using *assms* **by** (*intro Dep-Fun-Rel-relI*) *blast*

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 $([x :: P] \Rrightarrow T\ x)\ f\ g$
  using *assms* **by** *blast*

lemma *dep-mono-wrt-rel-if-le-right-if-dep-mono-wrt-rel*:
  assumes $([x\ y :: R] \Rrightarrow_m S\ x\ y)\ f$
  and $\bigwedge x\ y.\ R\ x\ y \Longrightarrow S\ x\ y\ (f\ x)\ (f\ y) \Longrightarrow T\ x\ y\ (f\ x)\ (f\ y)$
  shows $([x\ y :: R] \Rrightarrow_m T\ x\ y)\ f$
  using *assms* **by** (*intro dep-mono-wrt-relI*) *blast*

lemma *dep-mono-wrt-pred-if-le-right-if-dep-mono-wrt-pred*:
  assumes $([x :: P] \Rrightarrow_m Q\ x)\ f$
  and $\bigwedge x.\ P\ x \Longrightarrow Q\ x\ (f\ x) \Longrightarrow T\ x\ (f\ x)$
  shows $([x :: P] \Rrightarrow_m T\ x)\ f$
  using *assms* **by** *blast*

Composition **lemma** *dep-mono-wrt-rel-compI*:
  assumes $([x\ y :: R] \Rrightarrow_m S\ x\ y)\ f$
  and $\bigwedge x\ y.\ R\ x\ y \Longrightarrow ([x'\ y' :: T\ x\ y] \Rrightarrow_m U\ x\ y\ x'\ y')\ f'$
  and $\bigwedge x\ y.\ R\ x\ y \Longrightarrow S\ x\ y\ (f\ x)\ (f\ y) \Longrightarrow T\ x\ y\ (f\ x)\ (f\ y)$
  shows $([x\ y :: R] \Rrightarrow_m U\ x\ y\ (f\ x)\ (f\ y))\ (f' \circ f)$
  using *assms* **by** (*intro dep-mono-wrt-relI*) (*auto 6 0*)

corollary *dep-mono-wrt-rel-compI'*:
  assumes $([x\ y :: R] \Rrightarrow_m S\ x\ y)\ f$
  and $\bigwedge x\ y.\ R\ x\ y \Longrightarrow ([x'\ y' :: S\ x\ y] \Rrightarrow_m T\ x\ y\ x'\ y')\ f'$
  shows $([x\ y :: R] \Rrightarrow_m T\ x\ y\ (f\ x)\ (f\ y))\ (f' \circ f)$
  using *assms* **by** (*intro dep-mono-wrt-rel-compI*)

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

using *assms* **by** (*intro dep-mono-wrt-predI*) (*auto 6 0*)

lemma *dep-mono-wrt-pred-comp-dep-mono-wrt-pred-compI*:
assumes $([x :: P] \Rrightarrow_m Q\ x)\ f$
and $\bigwedge x.\ P\ x \Longrightarrow ([x' :: R\ x] \Rrightarrow_m S\ x\ x')\ f'$
and $\bigwedge x.\ P\ x \Longrightarrow Q\ x\ (f\ x) \Longrightarrow R\ x\ (f\ x)$
shows $([x :: P] \Rrightarrow_m S\ x\ (f\ x))\ (f' \circ f)$
using *assms* **by** (*intro dep-mono-wrt-predI*) (*auto 6 0*)

corollary *dep-mono-wrt-pred-comp-dep-mono-wrt-pred-compI'*:
assumes $([x :: P] \Rrightarrow_m Q\ x)\ f$
and $\bigwedge x.\ P\ x \Longrightarrow ([x' :: Q\ x] \Rrightarrow_m S\ x\ x')\ f'$
shows $([x :: P] \Rrightarrow_m S\ x\ (f\ x))\ (f' \circ f)$
using *assms* **by** (*intro dep-mono-wrt-pred-comp-dep-mono-wrt-pred-compI*)

Instantiations

lemma *mono-wrt-rel-self-id*: $(R \Rrightarrow_m R)$ *id* **by** *auto*
lemma *mono-wrt-pred-self-id*: $([P] \Rrightarrow_m P)$ *id* **by** *auto*

lemma *mono-in-dom*: *mono in-dom* **by** (*intro monoI*) *fast*
lemma *mono-in-codom*: *mono in-codom* **by** (*intro monoI*) *fast*
lemma *mono-in-field*: *mono in-field* **by** (*intro monoI*) *fast*
lemma *mono-rel-comp1*: *mono* $(\circ\circ)$ **by** (*intro monoI*) *fast*
lemma *mono-rel-comp2*: *mono* $((\circ\circ)\ x)$ **by** (*intro monoI*) *fast*

end

Reflexive

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

consts *reflexive-on* :: $'a \Rightarrow ('b \Rightarrow 'b \Rightarrow bool) \Rightarrow bool$

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

lemma *reflexive-onI* [*intro*]:
assumes $\bigwedge x.\ P\ x \Longrightarrow R\ x\ x$
shows *reflexive-on* $P\ R$
using *assms* **unfolding** *reflexive-on-pred-def* **by** *blast*

lemma *reflexive-onD* [*dest*]:
assumes *reflexive-on* $P\ R$

and $P\ x$
shows $R\ x\ x$
using *assms* **unfolding** *reflexive-on-pred-def* **by** *blast*

lemma *le-in-dom-if-reflexive-on*:
assumes *reflexive-on* $P\ R$
shows $P \leq$ *in-dom* R
using *assms* **by** *blast*

lemma *le-in-codom-if-reflexive-on*:
assumes *reflexive-on* $P\ R$
shows $P \leq$ *in-codom* R
using *assms* **by** *blast*

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
using *assms* **by** *blast*

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

lemma *antimono-reflexive-on* [*iff*]:
antimono (λ(P :: $'a \Rightarrow$ *bool*). *reflexive-on* P (R :: $'a \Rightarrow$ -))
by (*intro antimonoI*) *auto*

lemma *reflexive-on-if-le-pred-if-reflexive-on*:
fixes $P\ P'$ :: $'a \Rightarrow$ *bool* **and** R :: $'a \Rightarrow$ -
assumes *reflexive-on* $P\ R$
and $P' \leq P$
shows *reflexive-on* $P'\ R$
using *assms* **by** *blast*

lemma *reflexive-on-sup-eq* [*simp*]:
(*reflexive-on* :: ($'a \Rightarrow$ *bool*) $\Rightarrow$ ($'a \Rightarrow$ -) $\Rightarrow$ -) ((P :: $'a \Rightarrow$ *bool*) $\sqcup$ Q)
$=$ *reflexive-on* P $\sqcap$ *reflexive-on* Q
by (*intro ext iffI reflexive-onI*)
(*auto intro*: *reflexive-on-if-le-pred-if-reflexive-on*)

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

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

lemma *reflexive-eq-reflexive-on*:
reflexive (R :: $'a \Rightarrow$ -) $=$ *reflexive-on* ($\top$ :: $'a \Rightarrow$ *bool*) R
unfolding *reflexive-def* **..**

lemma *reflexiveI* [*intro*]:
  assumes $\bigwedge x.\ R\ x\ x$
  shows *reflexive R*
  unfolding *reflexive-eq-reflexive-on* **using** *assms* **by** (*intro reflexive-onI*)

lemma *reflexiveD*:
  assumes *reflexive R*
  shows $R\ x\ x$
  using *assms* **unfolding** *reflexive-eq-reflexive-on* **by** (*blast intro*: *top1I*)

lemma *reflexive-on-if-reflexive*:
  fixes $P :: 'a \Rightarrow bool$ **and** $R :: 'a \Rightarrow$ -
  assumes *reflexive R*
  shows *reflexive-on P R*
  using *assms* **by** (*intro reflexive-onI*) (*blast dest*: *reflexiveD*)

lemma *reflexive-rel-inv-iff-reflexive* [*iff*]:
  reflexive $R^{-1} \longleftrightarrow$ *reflexive R*
  by (*blast dest*: *reflexiveD*)

lemma *reflexive-iff-eq-le*: *reflexive* $R \longleftrightarrow ((=) \le R)$
  unfolding *reflexive-eq-reflexive-on reflexive-on-iff-eq-restrict-left-le*
  by *simp*

Instantiations

lemma *reflexive-eq*: *reflexive* $(=)$
  by (*rule reflexiveI*) (*rule refl*)

lemma *reflexive-top*: *reflexive* $\top$
  by (*rule reflexiveI*) *auto*

end

Symmetric

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

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

overloading
  symmetric-on-pred $\equiv$ *symmetric-on* $:: ('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
begin
  definition *symmetric-on-pred* $P\ R \equiv \forall x\ y.\ P\ x \land P\ y \land R\ x\ y \longrightarrow R\ y\ x$
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*
unfolding *symmetric-on-pred-def* **using** *assms* **by** *blast*

lemma *symmetric-onD*:
assumes *symmetric-on P R*
and $P\ x\ P\ y$
and $R\ x\ y$
shows $R\ y\ x$
using *assms* **unfolding** *symmetric-on-pred-def* **by** *blast*

lemma *symmetric-on-rel-inv-iff-symmetric-on* [*iff*]:
symmetric-on $P\ R^{-1} \longleftrightarrow$ *symmetric-on* $(P :: 'a \Rightarrow bool)\ (R :: 'a \Rightarrow -)$
by (*blast dest*: *symmetric-onD*)

lemma *antimono-symmetric-on* [*iff*]:
antimono $(\lambda(P :: 'a \Rightarrow bool).$ *symmetric-on* $P\ (R :: 'a \Rightarrow -))$
by (*intro antimonoI*) (*auto dest*: *symmetric-onD*)

lemma *symmetric-on-if-le-pred-if-symmetric-on*:
fixes $P\ P' :: 'a \Rightarrow bool$ **and** $R :: 'a \Rightarrow -$
assumes *symmetric-on P R*
and $P' \le P$
shows *symmetric-on* $P'\ R$
using *assms* **by** (*blast dest*: *symmetric-onD*)

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

lemma *symmetric-eq-symmetric-on*:
symmetric $(R :: 'a \Rightarrow -) =$ *symmetric-on* $(\top :: 'a \Rightarrow bool)\ R$
unfolding *symmetric-def* **..**

lemma *symmetricI* [*intro*]:
assumes $\bigwedge x\ y.\ R\ x\ y \Longrightarrow R\ y\ x$
shows *symmetric R*
unfolding *symmetric-eq-symmetric-on* **using** *assms* **by** (*intro symmetric-onI*)

lemma *symmetricD*:
assumes *symmetric R*
and $R\ x\ y$
shows $R\ y\ x$
using *assms* **unfolding** *symmetric-eq-symmetric-on* **by** (*auto dest*: *symmetric-onD*)

lemma *symmetric-on-if-symmetric*:
fixes $P :: 'a \Rightarrow bool$ **and** $R :: 'a \Rightarrow -$
assumes *symmetric R*
shows *symmetric-on P R*
using *assms* **by** (*intro symmetric-onI*) (*blast dest*: *symmetricD*)

lemma *symmetric-rel-inv-iff-symmetric* [*iff*]: *symmetric* R^{-1} $\longleftrightarrow$ *symmetric R*
  by (*blast dest*: *symmetricD*)

lemma *rel-inv-eq-self-if-symmetric* [*simp*]:
  assumes *symmetric R*
  shows $R^{-1} = R$
  using *assms* **by** (*blast dest*: *symmetricD*)

lemma *rel-iff-rel-if-symmetric*:
  assumes *symmetric R*
  shows $R\ x\ y \longleftrightarrow R\ y\ x$
  using *assms* **by** (*blast dest*: *symmetricD*)

lemma *symmetric-if-rel-inv-eq-self*:
  assumes $R^{-1} = R$
  shows *symmetric R*
  by (*intro symmetricI*, *subst assms*[*symmetric*]) *simp*

lemma *symmetric-iff-rel-inv-eq-self*: *symmetric R* $\longleftrightarrow$ $R^{-1} = R$
  using *rel-inv-eq-self-if-symmetric symmetric-if-rel-inv-eq-self* **by** *blast*

lemma *symmetric-if-symmetric-on-in-field*:
  assumes *symmetric-on* (*in-field R*) *R*
  shows *symmetric R*
  using *assms* **by** (*intro symmetricI*) (*blast dest*: *symmetric-onD*)

corollary *symmetric-on-in-field-iff-symmetric* [*simp*]:
  symmetric-on (*in-field R*) *R* $\longleftrightarrow$ *symmetric R*
  using *symmetric-if-symmetric-on-in-field symmetric-on-if-symmetric*
  by *blast*

Instantiations

lemma *symmetric-eq* [*iff*]: *symmetric* (=)
  by (*rule symmetricI*) (*rule sym*)

lemma *symmetric-top*: *symmetric* $\top$
  by (*rule symmetricI*) *auto*

end

Transitive

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

consts *transitive-on* :: $'a \Rightarrow ('b \Rightarrow 'b \Rightarrow bool) \Rightarrow bool$

overloading
  transitive-on-pred ≡ *transitive-on* :: $('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
begin
  definition *transitive-on-pred* $P\ R \equiv \forall x\ y\ z.\ P\ x \wedge P\ y \wedge P\ z \wedge R\ x\ y \wedge R\ y\ z \longrightarrow R\ x\ z$
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* $P\ R$
  unfolding *transitive-on-pred-def* **using** *assms* **by** *blast*

lemma *transitive-onD*:
  assumes *transitive-on* $P\ R$
  and $P\ x\ P\ y\ P\ z$
  and $R\ x\ y\ R\ y\ z$
  shows $R\ x\ z$
  using *assms* **unfolding** *transitive-on-pred-def* **by** *blast*

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 (*rule transitive-onI*)
  fix $x\ y\ z$ **assume** $R\ x\ y\ R\ y\ z$
  then have $(R \circ\circ R)\ x\ z$ **by** (*intro rel-compI*)
  moreover assume $P\ x\ P\ y\ P\ z$
  ultimately show $R\ x\ z$ **by** (*simp only*: *assms*)
qed

lemma *transitive-on-rel-inv-iff-transitive-on* [*iff*]:
  transitive-on $P\ R^{-1} \longleftrightarrow$ *transitive-on* $(P :: 'a \Rightarrow bool)\ (R :: 'a \Rightarrow -)$
  by (*auto intro*!: *transitive-onI dest*: *transitive-onD*)

lemma *antimono-transitive-on* [*iff*]:
  antimono $(\lambda(P :: 'a \Rightarrow bool).$ *transitive-on* $P\ (R :: 'a \Rightarrow -))$
  by (*intro antimonoI*) (*auto dest*: *transitive-onD*)

lemma *transitive-on-if-le-pred-if-transitive-on*:
  fixes $P\ P' :: 'a \Rightarrow bool$ **and** $R :: 'a \Rightarrow -$
  assumes *transitive-on* $P\ R$
  and $P' \leq P$
  shows *transitive-on* $P'\ R$
  using *assms* **by** (*auto dest*: *transitive-onD*)

definition *transitive* $(R :: 'a \Rightarrow -) \equiv$ *transitive-on* $(\top :: 'a \Rightarrow bool)\ R$

lemma *transitive-eq-transitive-on*:
  transitive $(R :: 'a \Rightarrow -) =$ *transitive-on* $(\top :: 'a \Rightarrow bool)\ R$

unfolding *transitive-def* **..**

lemma *transitiveI* [*intro*]:
assumes $\bigwedge x\ y\ z.\ R\ x\ y \Longrightarrow R\ y\ z \Longrightarrow R\ x\ z$
shows *transitive* R
unfolding *transitive-eq-transitive-on* **using** *assms* **by** (*intro transitive-onI*)

lemma *transitiveD* [*dest*]:
assumes *transitive* R
and $R\ x\ y\ R\ y\ z$
shows $R\ x\ z$
using *assms* **unfolding** *transitive-eq-transitive-on*
by (*auto dest*: *transitive-onD*)

lemma *transitive-on-if-transitive*:
fixes $P :: 'a \Rightarrow bool$ **and** $R :: 'a \Rightarrow$ -
assumes *transitive* R
shows *transitive-on* $P\ R$
using *assms* **by** (*intro transitive-onI*) *blast*

lemma *transitive-if-rel-comp-le-self*:
assumes $R \circ\circ R \leq R$
shows *transitive* R
using *assms* **unfolding** *transitive-eq-transitive-on*
by (*intro transitive-on-if-rel-comp-self-imp*) *blast*

lemma *rel-comp-le-self-if-transitive*:
assumes *transitive* R
shows $R \circ\circ R \leq R$
using *assms* **by** *blast*

corollary *transitive-iff-rel-comp-le-self*: *transitive* $R \longleftrightarrow R \circ\circ R \leq R$
using *transitive-if-rel-comp-le-self rel-comp-le-self-if-transitive* **by** *blast*

lemma *transitive-if-transitive-on-in-field*:
assumes *transitive-on* (*in-field* R) R
shows *transitive* R
using *assms* **by** (*intro transitiveI*) (*blast dest*: *transitive-onD*)

corollary *transitive-on-in-field-iff-transitive* [*simp*]:
transitive-on (*in-field* R) $R \longleftrightarrow$ *transitive* R
using *transitive-if-transitive-on-in-field transitive-on-if-transitive*
by *blast*

lemma *transitive-rel-inv-iff-transitive* [*iff*]:
transitive $R^{-1} \longleftrightarrow$ *transitive* R
by (*auto intro*!: *transitiveI*)

Instantiations **lemma** *transitive-eq*: *transitive* $(=)$

by (*rule transitiveI*) (*rule trans*)

lemma *transitive-top*: *transitive* $\top$
by (*rule transitiveI*) *auto*

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* $S\ x$
using *assms* **by** (*blast intro*: *in-dom-if-in-dom-rel-comp*)

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* $R\ z$
using *assms* **by** (*blast intro*: *in-codom-if-in-codom-rel-comp*)

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 −
from *le* **have** $(R1 \circ\circ R2)^{-1} \leq R3^{-1}$ **by** *blast*
with *symms* **show** *?thesis* **by** *simp*
qed

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 −
from *le* **have** $R3^{-1} \leq (R1 \circ\circ R2)^{-1}$ **by** *blast*
with *symms* **show** *?thesis* **by** *simp*
qed

corollary *rel-comp-le-rel-comp-if-rel-comp-le-rel-comp-if-symmetric*:
assumes *symmetric* $R1$ *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 $-$
from *assms* **have** $(R2 \circ\circ R1) \leq (R3 \circ\circ R4)^{-1}$
by (*intro rel-comp-le-rel-inv-if-rel-comp-le-if-symmetric*)
with *assms* **show** *?thesis* **by** *simp*
qed

corollary *rel-comp-le-rel-comp-iff-if-symmetric*:
assumes *symmetric R1 symmetric R2 symmetric R3 symmetric R4*
shows $(R1 \circ\circ R2) \leq (R3 \circ\circ R4) \longleftrightarrow (R2 \circ\circ R1) \leq (R4 \circ\circ R3)$
using *assms*
by (*blast intro*: *rel-comp-le-rel-comp-if-rel-comp-le-rel-comp-if-symmetric*)

corollary *eq-if-le-rel-if-symmetric*:
assumes *symmetric R symmetric S*
and $(R \circ\circ S) \leq (S \circ\circ R)$
shows $(R \circ\circ S) = (S \circ\circ R)$
using *assms rel-comp-le-rel-comp-iff-if-symmetric*[*of R S*]
by (*intro antisym*) *auto*

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 (*rule le-relI*)
fix *x1 x2* **assume**$(R \circ\circ S)$ *x1 x2*
then obtain *x3* **where** *R x1 x3 S x3 x2* **by** *blast*
then have *S x1 x3* **using** *le-rel* **by** *blast*
with ‹*S x3 x2*› **have** *S x1 x2* **using** *trans* **by** *blast*
with *refl-on* **have** *R x2 x2* **by** *blast*
then show $(S \circ\circ R)$ *x1 x2* **using** ‹*S x1 x2*› **by** *blast*
qed

end

Antisymmetric

theory *Binary-Relations-Antisymmetric*
imports
Binary-Relation-Functions
HOL-Syntax-Bundles-Lattices
begin

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

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* $\equiv \forall x\ y.\ P\ x \wedge P\ y \wedge R\ x\ y \wedge R\ y\ x \longrightarrow x = y$
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*
unfolding *antisymmetric-on-pred-def* **using** *assms* **by** *blast*

lemma *antisymmetric-onD*:
assumes *antisymmetric-on P R*
and *P x P y*
and *R x y R y x*
shows $x = y$
using *assms* **unfolding** *antisymmetric-on-pred-def* **by** *blast*

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

lemma *antisymmetric-eq-antisymmetric-on*:
antisymmetric (R :: $'a \Rightarrow$ -) = *antisymmetric-on* ($\top$ :: $'a \Rightarrow bool$) R
unfolding *antisymmetric-def* **..**

lemma *antisymmetricI* [*intro*]:
assumes $\bigwedge x\ y.\ R\ x\ y \Longrightarrow R\ y\ x \Longrightarrow x = y$
shows *antisymmetric R*
unfolding *antisymmetric-eq-antisymmetric-on* **using** *assms*
by (*intro antisymmetric-onI*)

lemma *antisymmetricD*:
assumes *antisymmetric R*
and *R x y R y x*
shows $x = y$
using *assms* **unfolding** *antisymmetric-eq-antisymmetric-on*
by (*auto dest*: *antisymmetric-onD*)

lemma *antisymmetric-on-if-antisymmetric*:
fixes P :: $'a \Rightarrow bool$ **and** R :: $'a \Rightarrow$ -
assumes *antisymmetric R*
shows *antisymmetric-on P R*
using *assms* **by** (*intro antisymmetric-onI*) (*blast dest*: *antisymmetricD*)

lemma *antisymmetric-if-antisymmetric-on-in-field*:
assumes *antisymmetric-on* (*in-field R*) *R*
shows *antisymmetric R*
using *assms* **by** (*intro antisymmetricI*) (*blast dest*: *antisymmetric-onD*)

corollary *antisymmetric-on-in-field-iff-antisymmetric* [*simp*]:
antisymmetric-on (*in-field* R) $R \longleftrightarrow$ *antisymmetric* R
using *antisymmetric-if-antisymmetric-on-in-field antisymmetric-on-if-antisymmetric*
by *blast*

end

Injective

theory *Binary-Relations-Injective*
imports
Binary-Relation-Functions
HOL-Syntax-Bundles-Lattices
ML-Unification.ML-Unification-HOL-Setup
begin

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

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* $P\ R \equiv \forall x\ x'\ y.\ P\ x \land P\ x' \land R\ x\ y \land R\ x'\ y \longrightarrow x = x'$
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* $P\ R$
unfolding *rel-injective-on-pred-def* **using** *assms* **by** *blast*

lemma *rel-injective-onD*:
assumes *rel-injective-on* $P\ R$
and $P\ x\ P\ x'$
and $R\ x\ y\ R\ x'\ y$
shows $x = x'$
using *assms* **unfolding** *rel-injective-on-pred-def* **by** *blast*

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

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* $P\ R \equiv \forall x\ x'\ y.\ P\ y \land R\ x\ y \land R\ x'\ y \longrightarrow x = x'$
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* $P\ R$

unfolding *rel-injective-at-pred-def* **using** *assms* **by** *blast*

lemma *rel-injective-atD*:
assumes *rel-injective-at* P R
and $P\ y$
and $R\ x\ y$ $R\ x'\ y$
shows $x = x'$
using *assms* **unfolding** *rel-injective-at-pred-def* **by** *blast*

definition *rel-injective* $(R :: 'a \Rightarrow -) \equiv$ *rel-injective-on* $(\top :: 'a \Rightarrow bool)\ R$

lemma *rel-injective-eq-rel-injective-on*:
rel-injective $(R :: 'a \Rightarrow -) =$ *rel-injective-on* $(\top :: 'a \Rightarrow bool)\ R$
unfolding *rel-injective-def* **..**

lemma *rel-injectiveI* [*intro*]:
assumes $\bigwedge x\ x'\ y.\ R\ x\ y \Longrightarrow R\ x'\ y \Longrightarrow x = x'$
shows *rel-injective* R
unfolding *rel-injective-eq-rel-injective-on* **using** *assms* **by** *blast*

lemma *rel-injectiveD*:
assumes *rel-injective* R
and $R\ x\ y$ $R\ x'\ y$
shows $x = x'$
using *assms* **unfolding** *rel-injective-eq-rel-injective-on*
by (*auto dest*: *rel-injective-onD*)

lemma *rel-injective-eq-rel-injective-at*:
rel-injective $(R :: 'a \Rightarrow 'b \Rightarrow bool) =$ *rel-injective-at* $(\top :: 'b \Rightarrow bool)\ R$
by (*intro iffI rel-injectiveI*) (*auto dest*: *rel-injective-atD rel-injectiveD*)

lemma *rel-injective-on-if-rel-injective*:
fixes $P :: 'a \Rightarrow bool$ **and** $R :: 'a \Rightarrow -$
assumes *rel-injective* R
shows *rel-injective-on* P R
using *assms* **by** (*blast dest*: *rel-injectiveD*)

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
using *assms* **by** (*blast dest*: *rel-injectiveD*)

lemma *rel-injective-if-rel-injective-on-in-dom*:
assumes *rel-injective-on* (*in-dom* R) R
shows *rel-injective* R
using *assms* **by** (*blast dest*: *rel-injective-onD*)

lemma *rel-injective-if-rel-injective-at-in-codom*:
  assumes *rel-injective-at* (*in-codom R*) *R*
  shows *rel-injective R*
  using *assms* **by** (*blast dest*: *rel-injective-atD*)

corollary *rel-injective-on-in-dom-iff-rel-injective* [*simp*]:
  rel-injective-on (*in-dom R*) *R* $\longleftrightarrow$ *rel-injective R*
  using *rel-injective-if-rel-injective-on-in-dom rel-injective-on-if-rel-injective*
  by *blast*

corollary *rel-injective-at-in-codom-iff-rel-injective* [*iff*]:
  rel-injective-at (*in-codom R*) *R* $\longleftrightarrow$ *rel-injective R*
  using *rel-injective-if-rel-injective-at-in-codom rel-injective-at-if-rel-injective*
  by *blast*

end

Irreflexive

theory *Binary-Relations-Irreflexive*
  imports
    Binary-Relation-Functions
    HOL-Syntax-Bundles-Lattices
begin

consts *irreflexive-on* :: $'a \Rightarrow ('b \Rightarrow 'b \Rightarrow bool) \Rightarrow bool$

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

lemma *irreflexive-onI* [*intro*]:
  assumes $\bigwedge x.\ P\ x \Longrightarrow \neg(R\ x\ x)$
  shows *irreflexive-on P R*
  using *assms* **unfolding** *irreflexive-on-pred-def* **by** *blast*

lemma *irreflexive-onD* [*dest*]:
  assumes *irreflexive-on P R*
  and $P\ x$
  shows $\neg(R\ x\ x)$
  using *assms* **unfolding** *irreflexive-on-pred-def* **by** *blast*

definition *irreflexive* $(R :: 'a \Rightarrow -) \equiv$ *irreflexive-on* $(\top :: 'a \Rightarrow bool)\ R$

lemma *irreflexive-eq-irreflexive-on*:
  irreflexive $(R :: 'a \Rightarrow -) =$ *irreflexive-on* $(\top :: 'a \Rightarrow bool)\ R$

unfolding *irreflexive-def* **..**

lemma *irreflexiveI* [*intro*]:
assumes $\bigwedge x.\ \neg(R\ x\ x)$
shows *irreflexive R*
unfolding *irreflexive-eq-irreflexive-on* **using** *assms* **by** (*intro irreflexive-onI*)

lemma *irreflexiveD*:
assumes *irreflexive R*
shows $\neg(R\ x\ x)$
using *assms* **unfolding** *irreflexive-eq-irreflexive-on* **by** *auto*

lemma *irreflexive-on-if-irreflexive*:
fixes $P :: {}'a \Rightarrow bool$ **and** $R :: {}'a \Rightarrow$ -
assumes *irreflexive R*
shows *irreflexive-on P R*
using *assms* **by** (*intro irreflexive-onI*) (*blast dest*: *irreflexiveD*)

end

Left Total

theory *Binary-Relations-Left-Total*
imports
Binary-Relation-Functions
HOL-Syntax-Bundles-Lattices
begin

consts *left-total-on* $:: {}'a \Rightarrow ({}'b \Rightarrow {}'c \Rightarrow bool) \Rightarrow bool$

overloading
left-total-on-pred $\equiv$ *left-total-on* $:: ({}'a \Rightarrow bool) \Rightarrow ({}'a \Rightarrow {}'b \Rightarrow bool) \Rightarrow bool$
begin
definition *left-total-on-pred P R* $\equiv \forall x.\ P\ x \longrightarrow$ *in-dom R x*
end

lemma *left-total-onI* [*intro*]:
assumes $\bigwedge x.\ P\ x \Longrightarrow$ *in-dom R x*
shows *left-total-on P R*
unfolding *left-total-on-pred-def* **using** *assms* **by** *blast*

lemma *left-total-onE* [*elim*]:
assumes *left-total-on P R*
and *P x*
obtains *y* **where** *R x y*
using *assms* **unfolding** *left-total-on-pred-def* **by** *blast*

lemma *in-dom-if-left-total-on*:

assumes *left-total-on P R*
and *P x*
shows *in-dom R x*
using *assms* **by** *blast*

definition *left-total* $(R :: 'a \Rightarrow \text{-}) \equiv$ *left-total-on* $(\top :: 'a \Rightarrow bool)$ *R*

lemma *left-total-eq-left-total-on*:
left-total $(R :: 'a \Rightarrow \text{-}) =$ *left-total-on* $(\top :: 'a \Rightarrow bool)$ *R*
unfolding *left-total-def* **..**

lemma *left-totalI* [*intro*]:
assumes $\bigwedge x$. *in-dom R x*
shows *left-total R*
unfolding *left-total-eq-left-total-on* **using** *assms* **by** (*intro left-total-onI*)

lemma *left-totalE*:
assumes *left-total R*
obtains *y* **where** *R x y*
using *assms* **unfolding** *left-total-eq-left-total-on* **by** (*blast intro*: *top1I*)

lemma *in-dom-if-left-total*:
assumes *left-total R*
shows *in-dom R x*
using *assms* **by** (*blast elim*: *left-totalE*)

lemma *left-total-on-if-left-total*:
fixes $P :: 'a \Rightarrow bool$ **and** $R :: 'a \Rightarrow \text{-}$
assumes *left-total R*
shows *left-total-on P R*
using *assms* **by** (*intro left-total-onI*) (*blast dest*: *in-dom-if-left-total*)

end

Right Unique

theory *Binary-Relations-Right-Unique*
imports
Binary-Relations-Injective
HOL-Syntax-Bundles-Lattices
begin

consts *right-unique-on* $:: 'a \Rightarrow ('b \Rightarrow 'c \Rightarrow bool) \Rightarrow bool$

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

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* $P\ R$
  using *assms* **unfolding** *right-unique-on-pred-def* **by** *blast*

lemma *right-unique-onD*:
  assumes *right-unique-on* $P\ R$
  and $P\ x$
  and $R\ x\ y\ R\ x\ y'$
  shows $y = y'$
  using *assms* **unfolding** *right-unique-on-pred-def* **by** *blast*

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

overloading
  right-unique-at-pred $\equiv$ *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* $P\ R$
  using *assms* **unfolding** *right-unique-at-pred-def* **by** *blast*

lemma *right-unique-atD*:
  assumes *right-unique-at* $P\ R$
  and $P\ y$
  and $P\ y'$
  and $R\ x\ y\ R\ x\ y'$
  shows $y = y'$
  using *assms* **unfolding** *right-unique-at-pred-def* **by** *blast*

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
  by (*blast dest*: *right-unique-atD rel-injective-onD*)

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
  by (*blast dest*: *right-unique-atD rel-injective-onD*)

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
by (*blast dest*: *right-unique-onD rel-injective-atD*)

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
by (*blast dest*: *right-unique-onD rel-injective-atD*)

definition *right-unique* $(R :: 'a \Rightarrow \text{-}) \equiv$ *right-unique-on* $(\top :: 'a \Rightarrow bool)$ R

lemma *right-unique-eq-right-unique-on*:
right-unique $(R :: 'a \Rightarrow \text{-}) =$ *right-unique-on* $(\top :: 'a \Rightarrow bool)$ R
unfolding *right-unique-def* **..**

lemma *right-uniqueI* [*intro*]:
assumes $\bigwedge x\ y\ y'.\ R\ x\ y \Longrightarrow R\ x\ y' \Longrightarrow y = y'$
shows *right-unique* R
unfolding *right-unique-eq-right-unique-on* **using** *assms* **by** *blast*

lemma *right-uniqueD*:
assumes *right-unique* R
and $R\ x\ y$ $R\ x\ y'$
shows $y = y'$
using *assms* **unfolding** *right-unique-eq-right-unique-on*
by (*auto dest*: *right-unique-onD*)

lemma *right-unique-eq-right-unique-at*:
right-unique $(R :: 'a \Rightarrow 'b \Rightarrow bool) =$ *right-unique-at* $(\top :: 'b \Rightarrow bool)$ R
by (*intro iffI right-uniqueI*) (*auto dest*: *right-unique-atD right-uniqueD*)

lemma *right-unique-on-if-right-unique*:
fixes $P :: 'a \Rightarrow bool$ **and** $R :: 'a \Rightarrow \text{-}$
assumes *right-unique* R
shows *right-unique-on* P R
using *assms* **by** (*blast dest*: *right-uniqueD*)

lemma *right-unique-at-if-right-unique*:
fixes $P :: 'a \Rightarrow bool$ **and** $R :: 'b \Rightarrow 'a \Rightarrow bool$
assumes *right-unique* R
shows *right-unique-at* P R
using *assms* **by** (*blast dest*: *right-uniqueD*)

lemma *right-unique-if-right-unique-on-in-dom*:
assumes *right-unique-on* (*in-dom* R) R
shows *right-unique* R
using *assms* **by** (*blast dest*: *right-unique-onD*)

lemma *right-unique-if-right-unique-at-in-codom*:

assumes *right-unique-at* (*in-codom* R) R
shows *right-unique* R
using *assms* **by** (*blast dest*: *right-unique-atD*)

corollary *right-unique-on-in-dom-iff-right-unique* [*iff*]:
right-unique-on (*in-dom* R) $R \longleftrightarrow$ *right-unique* R
using *right-unique-if-right-unique-on-in-dom right-unique-on-if-right-unique*
by *blast*

corollary *right-unique-at-in-codom-iff-right-unique* [*iff*]:
right-unique-at (*in-codom* R) $R \longleftrightarrow$ *right-unique* R
using *right-unique-if-right-unique-at-in-codom right-unique-at-if-right-unique*
by *blast*

lemma *right-unique-rel-inv-iff-rel-injective* [*iff*]:
right-unique $R^{-1} \longleftrightarrow$ *rel-injective* R
by (*blast dest*: *right-uniqueD rel-injectiveD*)

lemma *rel-injective-rel-inv-iff-right-unique* [*iff*]:
rel-injective $R^{-1} \longleftrightarrow$ *right-unique* R
by (*blast dest*: *right-uniqueD rel-injectiveD*)

Instantiatiats **lemma** *right-unique-eq*: *right-unique* $(=)$
by (*rule right-uniqueI*) *blast*

end

Surjective

theory *Binary-Relations-Surjective*
imports
Binary-Relations-Left-Total
HOL-Syntax-Bundles-Lattices
begin

consts *rel-surjective-at* :: $'a \Rightarrow ('b \Rightarrow 'c \Rightarrow bool) \Rightarrow bool$

overloading
rel-surjective-at-pred $\equiv$ *rel-surjective-at* :: $('a \Rightarrow bool) \Rightarrow ('b \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
begin
definition *rel-surjective-at-pred* P $R \equiv \forall y.\ P\ y \longrightarrow$ *in-codom* R y
end

lemma *rel-surjective-atI* [*intro*]:
assumes $\bigwedge y.\ P\ y \Longrightarrow$ *in-codom* R y
shows *rel-surjective-at* P R
unfolding *rel-surjective-at-pred-def* **using** *assms* **by** *blast*

lemma *rel-surjective-atE* [*elim*]:
assumes *rel-surjective-at P R*
and *P y*
obtains *x* **where** *R x y*
using *assms* **unfolding** *rel-surjective-at-pred-def* **by** *blast*

lemma *in-codom-if-rel-surjective-at-on*:
assumes *rel-surjective-at P R*
and *P y*
shows *in-codom R y*
using *assms* **by** *blast*

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*
by *fast*

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*
by *fast*

definition *rel-surjective* (R :: - $\Rightarrow$ $'a \Rightarrow$ -) $\equiv$ *rel-surjective-at* ($\top$ :: $'a \Rightarrow$ *bool*) R

lemma *rel-surjective-eq-rel-surjective-at*:
rel-surjective (R :: - $\Rightarrow$ $'a \Rightarrow$ -) = *rel-surjective-at* ($\top$ :: $'a \Rightarrow$ *bool*) R
unfolding *rel-surjective-def* **..**

lemma *rel-surjectiveI*:
assumes $\bigwedge y$. *in-codom R y*
shows *rel-surjective R*
unfolding *rel-surjective-eq-rel-surjective-at* **using** *assms* **by** (*intro rel-surjective-atI*)

lemma *rel-surjectiveE*:
assumes *rel-surjective R*
obtains *x* **where** *R x y*
using *assms* **unfolding** *rel-surjective-eq-rel-surjective-at*
by (*blast intro*: *top1I*)

lemma *in-codom-if-rel-surjective-at*:
assumes *rel-surjective R*
shows *in-codom R y*
using *assms* **by** (*blast elim*: *rel-surjectiveE*)

lemma *rel-surjective-rel-inv-iff-left-total* [*iff*]: *rel-surjective* R^{-1} $\longleftrightarrow$ *left-total R*
unfolding *rel-surjective-eq-rel-surjective-at left-total-eq-left-total-on*
by *simp*

lemma *left-total-rel-inv-iff-rel-surjective* [*iff*]: *left-total* R^{-1} $\longleftrightarrow$ *rel-surjective R*
unfolding *rel-surjective-eq-rel-surjective-at left-total-eq-left-total-on*

by *simp*

lemma *rel-surjective-at-if-surjective*:
fixes P :: $'a \Rightarrow$ *bool* **and** R :: - $\Rightarrow$ $'a \Rightarrow$ -
assumes *rel-surjective* R
shows *rel-surjective-at* P R
using *assms* **by** (*intro rel-surjective-atI*) (*blast dest*: *in-codom-if-rel-surjective-at*)

end

1.3.1 Basic Properties

theory *Binary-Relation-Properties*
imports
Binary-Relations-Antisymmetric
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
begin

end

1.3.2 Preorders

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

definition *preorder-on* P $R \equiv$ *reflexive-on* P $R \wedge$ *transitive-on* P R

lemma *preorder-onI* [*intro*]:
assumes *reflexive-on* P R
and *transitive-on* P R
shows *preorder-on* P R
unfolding *preorder-on-def* **using** *assms* **by** *blast*

lemma *preorder-onE* [*elim*]:
assumes *preorder-on* P R
obtains *reflexive-on* P R *transitive-on* P R
using *assms* **unfolding** *preorder-on-def* **by** *blast*

lemma *reflexive-on-if-preorder-on*:
  assumes *preorder-on P R*
  shows *reflexive-on P R*
  using *assms* **by** (*elim preorder-onE*)

lemma *transitive-on-if-preorder-on*:
  assumes *preorder-on P R*
  shows *transitive-on P R*
  using *assms* **by** (*elim preorder-onE*)

lemma *transitive-if-preorder-on-in-field*:
  assumes *preorder-on* (*in-field R*) *R*
  shows *transitive R*
  using *assms* **by** (*elim preorder-onE*) (*rule transitive-if-transitive-on-in-field*)

corollary *preorder-on-in-fieldE* [*elim*]:
  assumes *preorder-on* (*in-field R*) *R*
  obtains *reflexive-on* (*in-field R*) *R transitive R*
  using *assms*
  by (*blast dest*: *reflexive-on-if-preorder-on transitive-if-preorder-on-in-field*)

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

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 $R\ y\ x$
  using *assms* **by** *blast*

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 $R\ x\ y$
  using *assms* **by** *blast*

definition $preorder\ (R :: 'a \Rightarrow \text{-}) \equiv preorder\text{-}on\ (\top :: 'a \Rightarrow bool)\ R$

lemma *preorder-eq-preorder-on*:
  $preorder\ (R :: 'a \Rightarrow \text{-}) = preorder\text{-}on\ (\top :: 'a \Rightarrow bool)\ R$
  unfolding *preorder-def* **..**

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

unfolding *preorder-eq-preorder-on* **using** *assms*
by (*intro preorder-onI reflexive-on-if-reflexive transitive-on-if-transitive*)

lemma *preorderE* [*elim*]:
assumes *preorder R*
obtains *reflexive R transitive R*
using *assms* **unfolding** *preorder-eq-preorder-on* **by** (*elim preorder-onE*)
(*simp only*: *reflexive-eq-reflexive-on transitive-eq-transitive-on*)

lemma *preorder-on-if-preorder*:
fixes $P :: 'a \Rightarrow bool$ **and** $R :: 'a \Rightarrow$ -
assumes *preorder R*
shows *preorder-on P R*
using *assms* **by** (*elim preorderE*)
(*intro preorder-onI reflexive-on-if-reflexive transitive-on-if-transitive*)

Instantiations **lemma** *preorder-eq*: *preorder* (=)
using *reflexive-eq transitive-eq* **by** (*rule preorderI*)

end

1.3.3 Partial Equivalence Relations

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

definition *partial-equivalence-rel-on P R* $\equiv$ *transitive-on P R* $\wedge$ *symmetric-on P R*

lemma *partial-equivalence-rel-onI* [*intro*]:
assumes *transitive-on P R*
and *symmetric-on P R*
shows *partial-equivalence-rel-on P R*
unfolding *partial-equivalence-rel-on-def* **using** *assms* **by** *blast*

lemma *partial-equivalence-rel-onE* [*elim*]:
assumes *partial-equivalence-rel-on P R*
obtains *transitive-on P R symmetric-on P R*
using *assms* **unfolding** *partial-equivalence-rel-on-def* **by** *blast*

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*

using *assms* **by** (*blast dest*: *symmetric-onD transitive-onD*)

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 $R\ y\ y$
using *assms* **by** (*blast dest*: *symmetric-onD transitive-onD*)

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 :: $'a \Rightarrow$ -)
by *blast*

definition *partial-equivalence-rel* (R :: $'a \Rightarrow$ -) $\equiv$ *partial-equivalence-rel-on* ($\top$ :: $'a \Rightarrow bool$) R

lemma *partial-equivalence-rel-eq-partial-equivalence-rel-on*:
partial-equivalence-rel (R :: $'a \Rightarrow$ -) = *partial-equivalence-rel-on* ($\top$ :: $'a \Rightarrow bool$) R
unfolding *partial-equivalence-rel-def* **..**

lemma *partial-equivalence-relI* [*intro*]:
assumes *transitive* R
and *symmetric* R
shows *partial-equivalence-rel* R
unfolding *partial-equivalence-rel-eq-partial-equivalence-rel-on* **using** *assms*
by (*intro partial-equivalence-rel-onI transitive-on-if-transitive symmetric-on-if-symmetric*)

lemma *reflexive-on-in-field-if-partial-equivalence-rel*:
assumes *partial-equivalence-rel* R
shows *reflexive-on* (*in-field* R) R
using *assms* **unfolding** *partial-equivalence-rel-eq-partial-equivalence-rel-on*
by (*intro reflexive-onI*) (*blast*
intro: *top1I partial-equivalence-rel-on-rel-self-if-rel-dom*
partial-equivalence-rel-on-rel-self-if-rel-codom)

lemma *partial-equivalence-relE* [*elim*]:
assumes *partial-equivalence-rel* R
obtains *preorder-on* (*in-field* R) R *symmetric* R
using *assms* **unfolding** *partial-equivalence-rel-eq-partial-equivalence-rel-on*
by (*elim partial-equivalence-rel-onE*)
(*auto intro*: *reflexive-on-in-field-if-partial-equivalence-rel*
simp flip: *transitive-eq-transitive-on symmetric-eq-symmetric-on*)

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

using *assms* **by** (*elim partial-equivalence-relE preorder-on-in-fieldE*)
(*intro partial-equivalence-rel-onI transitive-on-if-transitive symmetric-on-if-symmetric*)

lemma *partial-equivalence-rel-rel-inv-iff-partial-equivalence-rel* [*iff*]:
partial-equivalence-rel $R^{-1} \longleftrightarrow$ *partial-equivalence-rel R*
unfolding *partial-equivalence-rel-eq-partial-equivalence-rel-on* **by** *blast*

corollary *in-codom-eq-in-dom-if-partial-equivalence-rel*:
assumes *partial-equivalence-rel R*
shows *in-codom R* = *in-dom R*
using *assms reflexive-on-in-field-if-partial-equivalence-rel in-codom-eq-in-dom-if-reflexive-on-in-field*
by *auto*

lemma *partial-equivalence-rel-rel-comp-self-eq-self*:
assumes *partial-equivalence-rel R*
shows $(R \circ\circ R) = R$
using *assms* **by** (*intro ext*) (*blast dest*: *symmetricD*)

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*
using *assms* **by** (*intro partial-equivalence-relI*)
(*auto intro*: *transitive-if-transitive-on-in-field symmetric-if-symmetric-on-in-field*)

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*
using *partial-equivalence-rel-if-partial-equivalence-rel-on-in-field partial-equivalence-rel-on-if-partial-equivalence-rel*
by *blast*

Instantiations **lemma** *partial-equivalence-rel-eq*: *partial-equivalence-rel* (=)
using *transitive-eq symmetric-eq* **by** (*rule partial-equivalence-relI*)

lemma *partial-equivalence-rel-top*: *partial-equivalence-rel* $\top$
using *transitive-top symmetric-top* **by** (*rule partial-equivalence-relI*)

end

1.3.4 Equivalences

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

definition *equivalence-rel-on P R* $\equiv$

partial-equivalence-rel-on P R $\wedge$ *reflexive-on P R*

lemma *equivalence-rel-onI* [*intro*]:
assumes *partial-equivalence-rel-on P R*
and *reflexive-on P R*
shows *equivalence-rel-on P R*
unfolding *equivalence-rel-on-def* **using** *assms* **by** *blast*

lemma *equivalence-rel-onE* [*elim*]:
assumes *equivalence-rel-on P R*
obtains *partial-equivalence-rel-on P R reflexive-on P R*
using *assms* **unfolding** *equivalence-rel-on-def* **by** *blast*

lemma *equivalence-rel-on-in-field-if-partial-equivalence-rel*:
assumes *partial-equivalence-rel R*
shows *equivalence-rel-on* (*in-field R*) *R*
using *assms*
by (*intro equivalence-rel-onI reflexive-on-in-field-if-partial-equivalence-rel*) *auto*

corollary *partial-equivalence-rel-iff-equivalence-rel-on-in-field*:
partial-equivalence-rel R $\longleftrightarrow$ *equivalence-rel-on* (*in-field R*) *R*
using *equivalence-rel-on-in-field-if-partial-equivalence-rel* **by** *auto*

definition *equivalence-rel* (R :: $'a \Rightarrow$ -) $\equiv$ *equivalence-rel-on* ($\top$ :: $'a \Rightarrow$ *bool*) R

lemma *equivalence-rel-eq-equivalence-rel-on*:
equivalence-rel (R :: $'a \Rightarrow$ -) = *equivalence-rel-on* ($\top$ :: $'a \Rightarrow$ *bool*) R
unfolding *equivalence-rel-def* **..**

lemma *equivalence-relI* [*intro*]:
assumes *partial-equivalence-rel R*
and *reflexive R*
shows *equivalence-rel R*
unfolding *equivalence-rel-eq-equivalence-rel-on* **using** *assms*
by (*intro equivalence-rel-onI partial-equivalence-rel-on-if-partial-equivalence-rel reflexive-on-if-reflexive*)

lemma *equivalence-relE* [*elim*]:
assumes *equivalence-rel R*
obtains *partial-equivalence-rel R reflexive R*
using *assms* **unfolding** *equivalence-rel-eq-equivalence-rel-on*
by (*elim equivalence-rel-onE*)
(*simp only*: *partial-equivalence-rel-eq-partial-equivalence-rel-on reflexive-eq-reflexive-on*)

lemma *equivalence-rel-on-if-equivalence*:
fixes P :: $'a \Rightarrow$ *bool* **and** R :: $'a \Rightarrow$ -
assumes *equivalence-rel R*

shows *equivalence-rel-on P R*
using *assms* **by** (*elim equivalence-relE*)
(*intro equivalence-rel-onI partial-equivalence-rel-on-if-partial-equivalence-rel reflexive-on-if-reflexive*)

Instantiations **lemma** *equivalence-eq*: *equivalence-rel* (=)
using *partial-equivalence-rel-eq reflexive-eq* **by** (*rule equivalence-relI*)

lemma *equivalence-top*: *equivalence-rel* ⊤
using *partial-equivalence-rel-top reflexive-top* **by** (*rule equivalence-relI*)

end

1.3.5 Partial Orders

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

definition *partial-order-on P R* ≡ *preorder-on P R* ∧ *antisymmetric-on P R*

lemma *partial-order-onI* [*intro*]:
assumes *preorder-on P R*
and *antisymmetric-on P R*
shows *partial-order-on P R*
unfolding *partial-order-on-def* **using** *assms* **by** *blast*

lemma *partial-order-onE* [*elim*]:
assumes *partial-order-on P R*
obtains *preorder-on P R antisymmetric-on P R*
using *assms* **unfolding** *partial-order-on-def* **by** *blast*

lemma *transitive-if-partial-order-on-in-field*:
assumes *partial-order-on* (*in-field R*) *R*
shows *transitive R*
using *assms* **by** (*elim partial-order-onE*) (*rule transitive-if-preorder-on-in-field*)

lemma *antisymmetric-if-partial-order-on-in-field*:
assumes *partial-order-on* (*in-field R*) *R*
shows *antisymmetric R*
using *assms* **by** (*elim partial-order-onE*)
(*rule antisymmetric-if-antisymmetric-on-in-field*)

definition *partial-order* (R :: $'a \Rightarrow$ -) ≡ *partial-order-on* (⊤ :: $'a \Rightarrow$ *bool*) *R*

lemma *partial-order-eq-partial-order-on*:
partial-order (R :: $'a \Rightarrow$ -) = *partial-order-on* (⊤ :: $'a \Rightarrow$ *bool*) *R*

unfolding *partial-order-def* **..**

lemma *partial-orderI* [*intro*]:
assumes *preorder R*
and *antisymmetric R*
shows *partial-order R*
unfolding *partial-order-eq-partial-order-on* **using** *assms*
by (*intro partial-order-onI preorder-on-if-preorder antisymmetric-on-if-antisymmetric*)

lemma *partial-orderE* [*elim*]:
assumes *partial-order R*
obtains *preorder R antisymmetric R*
using *assms* **unfolding** *partial-order-eq-partial-order-on*
by (*elim partial-order-onE*)
(*simp only*: *preorder-eq-preorder-on antisymmetric-eq-antisymmetric-on*)

lemma *partial-order-on-if-partial-order*:
fixes *P* :: *'a* ⇒ *bool* **and** *R* :: *'a* ⇒ -
assumes *partial-order R*
shows *partial-order-on P R*
using *assms* **by** (*elim partial-orderE*)
(*intro partial-order-onI preorder-on-if-preorder antisymmetric-on-if-antisymmetric*)

end

1.3.6 Restricted Equality

theory *Restricted-Equality*
imports
Binary-Relations-Order-Base
Binary-Relation-Functions
Equivalence-Relations
Partial-Orders
begin

Summary Introduces the concept of 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.

consts *eq-restrict* :: *'a* ⇒ *'b* ⇒ *'b* ⇒ *bool*

bundle *eq-restrict-syntax*
begin
syntax
-eq-restrict :: *'a* ⇒ (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *'a* ⇒ *bool* ((-) =(_) (-) [*51*,*51*,*51*] *50*)
notation *eq-restrict* ('(=(_)'))
end
bundle *no-eq-restrict-syntax*
begin

no-syntax
  -eq-restrict :: $'a \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a \Rightarrow bool$ ((-) =(_) (-) [*51*,*51*,*51*] *50*)
no-notation *eq-restrict* ('(=(_)'))
end
unbundle *eq-restrict-syntax*

translations
  $x =_P y \rightleftharpoons$ *CONST eq-restrict P x y*

overloading
  eq-restrict-pred ≡ *eq-restrict* :: $('a \Rightarrow bool) \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$
begin
  definition *eq-restrict-pred* $(P :: 'a \Rightarrow bool) \equiv ((=) :: 'a \Rightarrow -)\restriction_P$
end

lemma *eq-restrict-eq-eq-restrict-left*: $((=_{P :: 'a \Rightarrow bool}) :: 'a \Rightarrow -) = (=)\restriction_P$
  unfolding *eq-restrict-pred-def* **by** *simp*

lemma *eq-restrictI* [*intro*]:
  assumes $x = y$
  and $P\ x$
  shows $x =_P y$
  unfolding *eq-restrict-eq-eq-restrict-left* **using** *assms* **by** *auto*

lemma *eq-restrictE* [*elim*]:
  assumes $x =_P y$
  obtains $P\ x\ y = x$
  using *assms* **unfolding** *eq-restrict-eq-eq-restrict-left* **by** *auto*

lemma *eq-restrict-iff*: $x =_P y \longleftrightarrow y = x \wedge P\ x$ **by** *auto*

lemma *eq-restrict-le-eq*: $((=_{P :: 'a \Rightarrow bool}) :: 'a \Rightarrow -) \leq (=)$
  by (*intro le-relI*) *auto*

lemma *eq-restrict-top-eq-eq* [*simp*]: $(=_{\top :: 'a \Rightarrow bool}) = ((=) :: 'a \Rightarrow -)$
  unfolding *eq-restrict-eq-eq-restrict-left* **by** *simp*

lemma *in-dom-eq-restrict-eq* [*simp*]: *in-dom* $(=_P) = P$ **by** *auto*
lemma *in-codom-eq-restrict-eq* [*simp*]: *in-codom* $(=_P) = P$ **by** *auto*
lemma *in-field-eq-restrict-eq* [*simp*]: *in-field* $(=_P) = P$ **by** *auto*

Order Properties **context**
  fixes $P :: 'a \Rightarrow bool$
begin

context
begin
lemma *reflexive-on-eq-restrict*: *reflexive-on* $P\ ((=_P) :: 'a \Rightarrow -)$ **by** *auto*
lemma *transitive-eq-restrict*: *transitive* $((=_P) :: 'a \Rightarrow -)$ **by** *auto*

lemma *symmetric-eq-restrict*: *symmetric* $((=_P) :: {'a} \Rightarrow \text{-})$ **by** *auto*
lemma *antisymmetric-eq-restrict*: *antisymmetric* $((=_P) :: {'a} \Rightarrow \text{-})$ **by** *auto*
end

context
begin
lemma *preorder-on-eq-restrict*: *preorder-on* P $((=_P) :: {'a} \Rightarrow \text{-})$
  using *reflexive-on-eq-restrict transitive-eq-restrict* **by** *auto*
lemma *partial-equivalence-rel-eq-restrict*: *partial-equivalence-rel* $((=_P) :: {'a} \Rightarrow \text{-})$
  using *symmetric-eq-restrict transitive-eq-restrict* **by** *auto*
end

lemma *partial-order-on-eq-restrict*: *partial-order-on* P $((=_P) :: {'a} \Rightarrow \text{-})$
  using *preorder-on-eq-restrict antisymmetric-eq-restrict* **by** *auto*
lemma *equivalence-rel-on-eq-restrict*: *equivalence-rel-on* P $((=_P) :: {'a} \Rightarrow \text{-})$
  using *partial-equivalence-rel-eq-restrict reflexive-on-eq-restrict* **by** *blast*
end

end

theory *LBinary-Relations*
  imports
    Binary-Relation-Functions
    Binary-Relations-Lattice
    Binary-Relations-Order
    Binary-Relation-Properties
    Restricted-Equality
begin

Summary Basic concepts on binary relations.

end

Injective

theory *Functions-Injective*
  imports
    Functions-Base
    HOL-Syntax-Bundles-Lattices
begin

consts *injective-on* $:: {'a} \Rightarrow ({'b} \Rightarrow {'c}) \Rightarrow bool$

overloading
  injective-on-pred $\equiv$ *injective-on* $:: ({'a} \Rightarrow bool) \Rightarrow ({'a} \Rightarrow {'b}) \Rightarrow bool$
begin
  definition *injective-on-pred* $P\ f \equiv \forall x\ x'.\ P\ x \longrightarrow P\ x' \longrightarrow f\ x = f\ x' \longrightarrow x = x'$
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* $P\ f$
  unfolding *injective-on-pred-def* **using** *assms* **by** *blast*

lemma *injective-onD*:
  assumes *injective-on* $P\ f$
  and $P\ x\ P\ x'$
  and $f\ x = f\ x'$
  shows $x = x'$
  using *assms* **unfolding** *injective-on-pred-def* **by** *blast*

definition *injective* $(f :: 'a \Rightarrow \text{-}) \equiv$ *injective-on* $(\top :: 'a \Rightarrow bool)\ f$

lemma *injective-eq-injective-on*:
  injective $(f :: 'a \Rightarrow \text{-}) =$ *injective-on* $(\top :: 'a \Rightarrow bool)\ f$
  unfolding *injective-def* **..**

lemma *injectiveI* [*intro*]:
  assumes $\bigwedge x\ x'.\ f\ x = f\ x' \Longrightarrow x = x'$
  shows *injective* f
  unfolding *injective-eq-injective-on* **using** *assms* **by** (*intro injective-onI*)

lemma *injectiveD*:
  assumes *injective* f
  and $f\ x = f\ x'$
  shows $x = x'$
  using *assms* **unfolding** *injective-eq-injective-on* **by** (*auto dest*: *injective-onD*)

lemma *injective-on-if-injective*:
  fixes $P :: 'a \Rightarrow bool$ **and** $f :: 'a \Rightarrow \text{-}$
  assumes *injective* f
  shows *injective-on* $P\ f$
  using *assms* **by** (*intro injective-onI*) (*blast dest*: *injectiveD*)

Instantiations **lemma** *injective-id*: *injective id* **by** *auto*

end

Inverse

theory *Functions-Inverse*
  imports
    Functions-Injective
begin

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

overloading
  inverse-on-pred $\equiv$ *inverse-on* :: $('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow 'a) \Rightarrow bool$
begin
  definition *inverse-on-pred* $P\ f\ g \equiv \forall x.\ P\ x \longrightarrow g\ (f\ x) = x$
end

lemma *inverse-onI* [*intro*]:
  assumes $\bigwedge x.\ P\ x \Longrightarrow g\ (f\ x) = x$
  shows *inverse-on* $P\ f\ g$
  unfolding *inverse-on-pred-def* **using** *assms* **by** *blast*

lemma *inverse-onD*:
  assumes *inverse-on* $P\ f\ g$
  and $P\ x$
  shows $g\ (f\ x) = x$
  using *assms* **unfolding** *inverse-on-pred-def* **by** *blast*

lemma *injective-on-if-inverse-on*:
  assumes *inv*: *inverse-on* $(P :: 'a \Rightarrow bool)\ (f :: 'a \Rightarrow \text{-})\ g$
  shows *injective-on* $P\ f$
proof (*rule injective-onI*)
  fix $x\ x'$
  assume *Px*: $P\ x$ **and** *Px'*: $P\ x'$ **and** *f-x-eq-f-x'*: $f\ x = f\ x'$
  from *inv* **have** $x = g\ (f\ x)$ **using** *Px* **by** (*intro inverse-onD*[*symmetric*])
  also have ... $= g\ (f\ x')$ **by** (*simp only*: *f-x-eq-f-x'*)
  also have ... $= x'$ **using** *inv Px'* **by** (*intro inverse-onD*)
  finally show $x = x'$ **.**
qed

definition *inverse* $(f :: 'a \Rightarrow \text{-}) \equiv$ *inverse-on* $(\top :: 'a \Rightarrow bool)\ f$

lemma *inverse-eq-inverse-on*:
  inverse $(f :: 'a \Rightarrow \text{-}) =$ *inverse-on* $(\top :: 'a \Rightarrow bool)\ f$
  unfolding *inverse-def* **..**

lemma *inverseI* [*intro*]:
  assumes $\bigwedge x.\ g\ (f\ x) = x$
  shows *inverse* $f\ g$
  unfolding *inverse-eq-inverse-on* **using** *assms* **by** (*intro inverse-onI*)

lemma *inverseD*:
  assumes *inverse* $f\ g$
  shows $g\ (f\ x) = x$
  using *assms* **unfolding** *inverse-eq-inverse-on* **by** (*auto dest*: *inverse-onD*)

lemma *inverse-on-if-inverse*:
  fixes $P :: 'a \Rightarrow bool$ **and** $f :: 'a \Rightarrow 'b$
  assumes *inverse* $f\ g$

shows *inverse-on P f g*
using *assms* **by** (*intro inverse-onI*) (*blast dest*: *inverseD*)

end

Bijection

theory *Functions-Bijection*
imports
Functions-Inverse
Functions-Monotone
begin

consts *bijection-on* :: $'a \Rightarrow 'b \Rightarrow ('c \Rightarrow 'd) \Rightarrow ('d \Rightarrow 'c) \Rightarrow bool$

overloading
bijection-on-pred $\equiv$ *bijection-on* :: $('a \Rightarrow bool) \Rightarrow ('b \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow 'a) \Rightarrow bool$
begin
definition *bijection-on-pred* $P\ P'\ f\ g \equiv$
$([P] \Rrightarrow_m P')\ f \wedge$
$([P'] \Rrightarrow_m P)\ g \wedge$
inverse-on $P\ f\ g \wedge$
inverse-on $P'\ g\ f$
end

lemma *bijection-onI* [*intro*]:
assumes $([P] \Rrightarrow_m P')\ f$
and $([P'] \Rrightarrow_m P)\ g$
and *inverse-on* $P\ f\ g$
and *inverse-on* $P'\ g\ f$
shows *bijection-on* $P\ P'\ f\ g$
using *assms* **unfolding** *bijection-on-pred-def* **by** *blast*

lemma *bijection-onE*:
assumes *bijection-on* $P\ P'\ f\ g$
obtains $([P] \Rrightarrow_m P')\ f$ $([P'] \Rrightarrow_m P)\ g$
inverse-on $P\ f\ g$ *inverse-on* $P'\ g\ f$
using *assms* **unfolding** *bijection-on-pred-def* **by** *blast*

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

lemma *mono-wrt-pred-if-bijection-on-left*:
assumes *bijection-on* $P\ P'\ f\ g$

shows $([P] \Rrightarrow_m P')\ f$
using *assms* **by** (*elim bijection-onE*)

lemma *mono-wrt-pred-if-bijection-on-right*:
assumes *bijection-on* $P\ P'\ f\ g$
shows $([P'] \Rrightarrow_m P)\ g$
using *assms* **by** (*elim bijection-onE*)

lemma *bijection-on-pred-right*:
assumes *bijection-on* $P\ P'\ f\ g$
and $P\ x$
shows $P'\ (f\ x)$
using *assms* **by** (*blast elim*: *bijection-onE*)

lemma *bijection-on-pred-left*:
assumes *bijection-on* $P\ P'\ f\ g$
and $P'\ y$
shows $P\ (g\ y)$
using *assms* **by** (*blast elim*: *bijection-onE*)

lemma *inverse-on-if-bijection-on-left-right*:
assumes *bijection-on* $P\ P'\ f\ g$
shows *inverse-on* $P\ f\ g$
using *assms* **by** (*elim bijection-onE*)

lemma *inverse-on-if-bijection-on-right-left*:
assumes *bijection-on* $P\ P'\ f\ g$
shows *inverse-on* $P'\ g\ f$
using *assms* **by** (*elim bijection-onE*)

lemma *bijection-on-left-right-eq-self*:
assumes *bijection-on* $P\ P'\ f\ g$
and $P\ x$
shows $g\ (f\ x) = x$
using *assms inverse-on-if-bijection-on-left-right*
by (*intro inverse-onD*)

lemma *bijection-on-right-left-eq-self'*:
assumes *bijection-on* $P\ P'\ f\ g$
and $P'\ y$
shows $f\ (g\ y) = y$
using *assms inverse-on-if-bijection-on-right-left* **by** (*intro inverse-onD*)

lemma *bijection-on-right-left-if-bijection-on-left-right*:
assumes *bijection-on* $P\ P'\ f\ g$
shows *bijection-on* $P'\ P\ g\ f$
using *assms* **by** (*auto elim*: *bijection-onE*)

lemma *injective-on-if-bijection-on-left*:

assumes *bijection-on* P P' f g
shows *injective-on* P f
using *assms*
by (*intro injective-on-if-inverse-on inverse-on-if-bijection-on-left-right*)

lemma *injective-on-if-bijection-on-right*:
assumes *bijection-on* P P' f g
shows *injective-on* P' g
by (*intro injective-on-if-inverse-on*)
(*fact inverse-on-if-bijection-on-right-left*[*OF assms*])

end

definition *bijection* $(f :: 'a \Rightarrow 'b) \equiv$ *bijection-on* $(\top :: 'a \Rightarrow bool)$ $(\top :: 'b \Rightarrow bool)$ f

lemma *bijection-eq-bijection-on*:
bijection $(f :: 'a \Rightarrow 'b) =$ *bijection-on* $(\top :: 'a \Rightarrow bool)$ $(\top :: 'b \Rightarrow bool)$ f
unfolding *bijection-def* **..**

lemma *bijectionI* [*intro*]:
assumes *inverse* f g
and *inverse* g f
shows *bijection* f g
unfolding *bijection-eq-bijection-on* **using** *assms*
by (*intro bijection-onI inverse-on-if-inverse dep-mono-wrt-predI*) *simp-all*

lemma *bijectionE* [*elim*]:
assumes *bijection* f g
obtains *inverse* f g *inverse* g f
using *assms* **unfolding** *bijection-eq-bijection-on inverse-eq-inverse-on*
by (*blast elim*: *bijection-onE*)

lemma *inverse-if-bijection-left-right*:
assumes *bijection* f g
shows *inverse* f g
using *assms* **by** (*elim bijectionE*)

lemma *inverse-if-bijection-right-left*:
assumes *bijection* f g
shows *inverse* g f
using *assms* **by** (*elim bijectionE*)

lemma *bijection-right-left-if-bijection-left-right*:
assumes *bijection* f g
shows *bijection* g f
using *assms* **by** *auto*

Instantiations **lemma** *bijection-on-self-id*:
  fixes $P :: 'a \Rightarrow bool$
  shows *bijection-on* P P $(id :: 'a \Rightarrow \text{-})$ *id*
  by (*intro bijection-onI inverse-onI dep-mono-wrt-predI*) *simp-all*

end

Surjective

theory *Functions-Surjective*
  imports
    HOL-Syntax-Bundles-Lattices
begin

consts *surjective-at* $:: 'a \Rightarrow ('b \Rightarrow 'c) \Rightarrow bool$

overloading
  surjective-at-pred $\equiv$ *surjective-at* $:: ('a \Rightarrow bool) \Rightarrow ('b \Rightarrow 'a) \Rightarrow bool$
begin
  definition *surjective-at-pred* $P\ f \equiv \forall\, y.\ P\ y \longrightarrow (\exists\, x.\ y = f\ x)$
end

lemma *surjective-atI* [*intro*]:
  assumes $\bigwedge y.\ P\ y \Longrightarrow \exists\, x.\ y = f\ x$
  shows *surjective-at* $P\ f$
  unfolding *surjective-at-pred-def* **using** *assms* **by** *blast*

lemma *surjective-atE* [*elim*]:
  assumes *surjective-at* $P\ f$
  and $P\ y$
  obtains x **where** $y = f\ x$
  using *assms* **unfolding** *surjective-at-pred-def* **by** *blast*

definition *surjective* $(f :: \text{-} \Rightarrow 'a) \equiv$ *surjective-at* $(\top :: 'a \Rightarrow bool)\ f$

lemma *surjective-eq-surjective-at*:
  surjective $(f :: \text{-} \Rightarrow 'a) =$ *surjective-at* $(\top :: 'a \Rightarrow bool)\ f$
  unfolding *surjective-def* **..**

lemma *surjectiveI* [*intro*]:
  assumes $\bigwedge y.\ \exists\, x.\ y = f\ x$
  shows *surjective* f
  unfolding *surjective-eq-surjective-at* **using** *assms* **by** (*intro surjective-atI*)

lemma *surjectiveE*:
  assumes *surjective* f
  obtains x **where** $y = f\ x$
  using *assms* **unfolding** *surjective-eq-surjective-at* **by** (*blast intro*: *top1I*)

lemma *surjective-at-if-surjective*:
  fixes $P :: 'a \Rightarrow bool$ **and** $f :: \text{-} \Rightarrow 'a$
  assumes *surjective f*
  shows *surjective-at P f*
  using *assms* **by** (*intro surjective-atI*) (*blast elim*: *surjectiveE*)

end

1.3.7 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
    Functions-Base
    Function-Properties
    Function-Relators
begin

Summary Basic concepts on functions.

end

1.3.8 Functions On Orders

Basics

theory *Order-Functions-Base*
  imports
    Functions-Monotone
    Restricted-Equality
begin

Bi-Relation **definition** *bi-related* $R\ x\ y \equiv R\ x\ y \wedge R\ y\ x$

bundle *bi-related-syntax* **begin**

syntax
  -bi-related :: *'a* ⇒ (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *'a* ⇒ *bool* ((-) $\equiv_{(-)}$ (-) [*51*,*51*,*51*] *50*)
notation *bi-related* ('(≡(_)'))
end
bundle *no-bi-related-syntax* **begin**
no-syntax
  -bi-related :: *'a* ⇒ (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *'a* ⇒ *bool* ((-) $\equiv_{(-)}$ (-) [*51*,*51*,*51*] *50*)
no-notation *bi-related* ('(≡(_)'))
end
unbundle *bi-related-syntax*
translations
  $x \equiv_R y$ ⇌ *CONST bi-related R x y*

lemma *bi-relatedI* [*intro*]:
  assumes *R x y*
  and *R y x*
  shows $x \equiv_R y$
  unfolding *bi-related-def* **using** *assms* **by** *blast*

lemma *bi-relatedE* [*elim*]:
  assumes $x \equiv_R y$
  obtains *R x y R y x*
  using *assms* **unfolding** *bi-related-def* **by** *blast*

lemma *symmetric-bi-related* [*iff*]: *symmetric* ($\equiv_R$)
  by (*intro symmetricI*) *blast*

lemma *reflexive-bi-related-if-reflexive* [*intro*]:
  assumes *reflexive R*
  shows *reflexive* ($\equiv_R$)
  using *assms* **by** (*intro reflexiveI*) (*blast dest*: *reflexiveD*)

lemma *transitive-bi-related-if-transitive* [*intro*]:
  assumes *transitive R*
  shows *transitive* ($\equiv_R$)
  using *assms* **by** (*intro transitiveI bi-relatedI*) *auto*

lemma *mono-bi-related* [*iff*]: *mono bi-related*
  by (*intro monoI*) *blast*

lemma *bi-related-if-le-rel-if-bi-related*:
  assumes $x \equiv_R y$
  and $R \leq S$
  shows $x \equiv_S y$
  using *assms* **by** *blast*

lemma *eq-if-bi-related-if-antisymmetric-on*:
  assumes *antisymmetric-on P R*
  and $x \equiv_R y$

and $P\ x\ P\ y$
shows $x = y$
using *assms* **by** (*blast dest*: *antisymmetric-onD*)

lemma *eq-if-bi-related-if-in-field-le-if-antisymmetric-on*:
assumes *antisymmetric-on* $P\ R$
and *in-field* $R \le P$
and $x \equiv_R y$
shows $x = y$
using *assms* **by** (*intro eq-if-bi-related-if-antisymmetric-on*) *blast*+

lemma *bi-related-le-eq-if-antisymmetric-on-in-field*:
assumes *antisymmetric-on* (*in-field* R) R
shows $(\equiv_R) \le (=)$
using *assms*
by (*intro le-relI eq-if-bi-related-if-in-field-le-if-antisymmetric-on*) *blast*+

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 $x \equiv_R y$
using *assms* **by** *blast*

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 $x \equiv_R y$
using *assms* **by** *blast*

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 $x = y$
using *assms* **by** (*blast intro*: *eq-if-bi-related-if-antisymmetric-on*
bi-related-if-all-rel-iff-if-reflexive-on)

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 $x = y$
using *assms* **by** (*blast intro*: *eq-if-bi-related-if-antisymmetric-on*
bi-related-if-all-rel-iff-if-reflexive-on')

Inflationary **consts** *inflationary-on* :: $'a \Rightarrow ('b \Rightarrow 'b \Rightarrow bool) \Rightarrow ('b \Rightarrow 'b) \Rightarrow bool$

overloading
  inflationary-on-pred ≡ *inflationary-on* ::
    $('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a) \Rightarrow bool$
begin

Often also called "extensive".

  definition *inflationary-on-pred* P $(R :: 'a \Rightarrow 'a \Rightarrow \text{-})$ $f \equiv \forall x.\ P\ x \longrightarrow R\ x\ (f\ x)$
end

lemma *inflationary-onI* [*intro*]:
  assumes $\bigwedge x.\ P\ x \Longrightarrow R\ x\ (f\ x)$
  shows *inflationary-on* $P\ R\ f$
  unfolding *inflationary-on-pred-def* **using** *assms* **by** *blast*

lemma *inflationary-onD* [*dest*]:
  assumes *inflationary-on* $P\ R\ f$
  and $P\ x$
  shows $R\ x\ (f\ x)$
  using *assms* **unfolding** *inflationary-on-pred-def* **by** *blast*

lemma *inflationary-on-eq-dep-mono-wrt-pred*: *inflationary-on* = *dep-mono-wrt-pred*
  by *blast*

lemma *antimono-inflationary-on-pred* [*iff*]:
  antimono $(\lambda(P :: 'a \Rightarrow bool).$ *inflationary-on* $P\ (R :: 'a \Rightarrow \text{-}))$
  by (*intro antimonoI*) *auto*

lemma *inflationary-on-if-le-pred-if-inflationary-on*:
  fixes $P\ P' :: 'a \Rightarrow bool$ **and** $R :: 'a \Rightarrow \text{-}$
  assumes *inflationary-on* $P\ R\ f$
  and $P' \le P$
  shows *inflationary-on* $P'\ R\ f$
  using *assms* **by** *blast*

lemma *mono-inflationary-on-rel* [*iff*]:
  mono $(\lambda(R :: 'a \Rightarrow \text{-}).$ *inflationary-on* $(P :: 'a \Rightarrow bool)\ R)$
  by (*intro monoI*) *auto*

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* $P\ R'\ f$
  using *assms* **by** *blast*

lemma *le-in-dom-if-inflationary-on*:
  assumes *inflationary-on* $P\ R\ f$
  shows $P \le$ *in-dom* R
  using *assms* **by** *blast*

lemma *inflationary-on-sup-eq* [*simp*]:
$(\textit{inflationary-on} :: ('a \Rightarrow \textit{bool}) \Rightarrow ('a \Rightarrow \text{-}) \Rightarrow \text{-})\ ((P :: 'a \Rightarrow \textit{bool}) \sqcup Q)$
$= \textit{inflationary-on}\ P \sqcap \textit{inflationary-on}\ Q$
by (*intro ext iffI inflationary-onI*)
(*auto intro*: *inflationary-on-if-le-pred-if-inflationary-on*)

definition *inflationary* $(R :: 'a \Rightarrow \text{-})\ f \equiv \textit{inflationary-on}\ (\top :: 'a \Rightarrow \textit{bool})\ R\ f$

lemma *inflationary-eq-inflationary-on*:
$\textit{inflationary}\ (R :: 'a \Rightarrow \text{-})\ f = \textit{inflationary-on}\ (\top :: 'a \Rightarrow \textit{bool})\ R\ f$
unfolding *inflationary-def* **..**

lemma *inflationaryI* [*intro*]:
assumes $\bigwedge x.\ R\ x\ (f\ x)$
shows *inflationary R f*
unfolding *inflationary-eq-inflationary-on* **using** *assms*
by (*intro inflationary-onI*)

lemma *inflationaryD*:
assumes *inflationary R f*
shows $R\ x\ (f\ x)$
using *assms* **unfolding** *inflationary-eq-inflationary-on* **by** *auto*

lemma *inflationary-on-if-inflationary*:
fixes $P :: 'a \Rightarrow \textit{bool}$ **and** $R :: 'a \Rightarrow \text{-}$
assumes *inflationary R f*
shows *inflationary-on P R f*
using *assms* **by** (*intro inflationary-onI*) (*blast dest*: *inflationaryD*)

lemma *inflationary-eq-dep-mono-wrt-pred*: $\textit{inflationary} = \textit{dep-mono-wrt-pred}\ \top$
by (*intro ext*) (*fastforce dest*: *inflationaryD*)

Deflationary **definition** *deflationary-on* $P\ R \equiv \textit{inflationary-on}\ P\ R^{-1}$

lemma *deflationary-on-eq-inflationary-on-rel-inv*:
$\textit{deflationary-on}\ P\ R = \textit{inflationary-on}\ P\ R^{-1}$
unfolding *deflationary-on-def* **..**

declare *deflationary-on-eq-inflationary-on-rel-inv*[*symmetric*, *simp*]

corollary *deflationary-on-rel-inv-eq-inflationary-on* [*simp*]:
$\textit{deflationary-on}\ P\ R^{-1} = \textit{inflationary-on}\ P\ R$
unfolding *deflationary-on-eq-inflationary-on-rel-inv* **by** *simp*

lemma *deflationary-onI* [*intro*]:
assumes $\bigwedge x.\ P\ x \Longrightarrow R\ (f\ x)\ x$
shows *deflationary-on P R f*

unfolding *deflationary-on-eq-inflationary-on-rel-inv* **using** *assms*
by (*intro inflationary-onI rel-invI*)

lemma *deflationary-onD* [*dest*]:
assumes *deflationary-on P R f*
and *P x*
shows $R\ (f\ x)\ x$
using *assms* **unfolding** *deflationary-on-eq-inflationary-on-rel-inv* **by** *blast*

lemma *deflationary-on-eq-dep-mono-wrt-pred-rel-inv*:
deflationary-on P R = $([x :: P] \Rrightarrow_m R^{-1}\ x)$
by *blast*

lemma *antimono-deflationary-on-pred* [*iff*]:
antimono (λ(*P* :: *'a* ⇒ *bool*). *deflationary-on P* (*R* :: *'a* ⇒ -))
by (*intro antimonoI*) *auto*

lemma *deflationary-on-if-le-pred-if-deflationary-on*:
fixes *P P'* :: *'a* ⇒ *bool* **and** *R* :: *'a* ⇒ -
assumes *deflationary-on P R f*
and $P' \leq P$
shows *deflationary-on P' R f*
using *assms* **by** *blast*

lemma *mono-deflationary-on-rel* [*iff*]:
mono (λ(*R* :: *'a* ⇒ -). *deflationary-on* (*P* :: *'a* ⇒ *bool*) *R*)
by (*intro monoI*) *auto*

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*
using *assms* **by** *auto*

lemma *le-in-dom-if-deflationary-on*:
assumes *deflationary-on P R f*
shows *P* ≤ *in-codom R*
using *assms* **by** *blast*

lemma *deflationary-on-sup-eq* [*simp*]:
(*deflationary-on* :: (*'a* ⇒ *bool*) ⇒ (*'a* ⇒ -) ⇒ -) ((*P* :: *'a* ⇒ *bool*) ⊔ *Q*)
= *deflationary-on P* ⊓ *deflationary-on Q*
unfolding *deflationary-on-eq-inflationary-on-rel-inv* **by** *auto*

definition *deflationary R* (*f* :: *'a* ⇒ -) ≡ *deflationary-on* (⊤ :: *'a* ⇒ *bool*) *R f*

lemma *deflationary-eq-deflationary-on*:
deflationary R (*f* :: *'a* ⇒ -) = *deflationary-on* (⊤ :: *'a* ⇒ *bool*) *R f*
unfolding *deflationary-def* **..**

lemma *deflationaryI* [*intro*]:
assumes $\bigwedge x.\ R\ (f\ x)\ x$
shows *deflationary R f*
unfolding *deflationary-eq-deflationary-on* **using** *assms* **by** (*intro deflationary-onI*)

lemma *deflationaryD*:
assumes *deflationary R f*
shows $R\ (f\ x)\ x$
using *assms* **unfolding** *deflationary-eq-deflationary-on* **by** *auto*

lemma *deflationary-on-if-deflationary*:
fixes $P :: 'a \Rightarrow bool$ **and** $f :: 'a \Rightarrow$ -
assumes *deflationary R f*
shows *deflationary-on P R f*
using *assms* **by** (*intro deflationary-onI*) (*blast dest*: *deflationaryD*)

lemma *deflationary-eq-dep-mono-wrt-pred-rel-inv*:
deflationary $R =$ *dep-mono-wrt-pred* $\top\ R^{-1}$
by (*intro ext*) (*fastforce dest*: *deflationaryD*)

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*
unfolding *rel-equivalence-on-def* **..**

lemma *rel-equivalence-onI* [*intro*]:
assumes *inflationary-on P R f*
and *deflationary-on P R f*
shows *rel-equivalence-on P R f*
unfolding *rel-equivalence-on-eq* **using** *assms* **by** *auto*

lemma *rel-equivalence-onE* [*elim*]:
assumes *rel-equivalence-on P R f*
obtains *inflationary-on P R f deflationary-on P R f*
using *assms* **unfolding** *rel-equivalence-on-eq* **by** *auto*

lemma *rel-equivalence-on-eq-dep-mono-wrt-pred-inf*:
rel-equivalence-on P R $=$ *dep-mono-wrt-pred P* $(R \sqcap R^{-1})$
by (*intro ext*) *fastforce*

lemma *bi-related-if-rel-equivalence-on*:
assumes *rel-equivalence-on P R f*
and $P\ x$
shows $x \equiv_R f\ x$
using *assms* **by** (*intro bi-relatedI*) *auto*

lemma *rel-equivalence-on-if-all-bi-related*:
assumes $\bigwedge x.\ P\ x \Longrightarrow x \equiv_R f\ x$
shows *rel-equivalence-on* $P\ R\ f$
using *assms* **by** *auto*

corollary *rel-equivalence-on-iff-all-bi-related*:
rel-equivalence-on $P\ R\ f \longleftrightarrow (\forall x.\ P\ x \longrightarrow x \equiv_R f\ x)$
using *rel-equivalence-on-if-all-bi-related bi-related-if-rel-equivalence-on*
by *blast*

lemma *rel-equivalence-onD* [*dest*]:
assumes *rel-equivalence-on* $P\ R\ f$
and $P\ x$
shows $R\ x\ (f\ x)\ R\ (f\ x)\ x$
using *assms* **by** (*auto dest*: *bi-related-if-rel-equivalence-on*)

lemma *rel-equivalence-on-rel-inv-eq-rel-equivalence-on* [*simp*]:
rel-equivalence-on $P\ R^{-1}$ = *rel-equivalence-on* $P\ R$
by (*intro ext*) *fastforce*

lemma *antimono-rel-equivalence-on-pred* [*iff*]:
antimono ($\lambda(P :: 'a \Rightarrow bool)$. *rel-equivalence-on* P ($R :: 'a \Rightarrow$ -))
by (*intro antimonoI*) *blast*

lemma *rel-equivalence-on-if-le-pred-if-rel-equivalence-on*:
fixes $P\ P' :: 'a \Rightarrow bool$ **and** $R :: 'a \Rightarrow$ -
assumes *rel-equivalence-on* $P\ R\ f$
and $P' \le P$
shows *rel-equivalence-on* $P'\ R\ f$
using *assms* **by** *blast*

lemma *rel-equivalence-on-sup-eq* [*simp*]:
(*rel-equivalence-on* :: $('a \Rightarrow bool) \Rightarrow ('a \Rightarrow -) \Rightarrow -)\ ((P :: 'a \Rightarrow bool) \sqcup Q)$
= *rel-equivalence-on* $P \sqcap$ *rel-equivalence-on* Q
unfolding *rel-equivalence-on-eq* **by** (*simp add*: *inf-aci*)

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
using *assms* **by** (*intro ext*) *blast*

lemma *reflexive-on-if-transitive-on-if-mon-wrt-pred-if-rel-equivalence-on*:
assumes *rel-equivalence-on* $P\ R\ f$
and $([P] \Rrightarrow_m P)\ f$
and *transitive-on* $P\ R$
shows *reflexive-on* $P\ R$
using *assms* **by** (*blast dest*: *transitive-onD*)

lemma *inflationary-on-eq-rel-equivalence-on-if-symmetric*:

assumes *symmetric R*
shows *inflationary-on P R* = *rel-equivalence-on P R*
using *assms*
by (*simp add*: *rel-equivalence-on-eq deflationary-on-eq-inflationary-on-rel-inv*)

lemma *deflationary-on-eq-rel-equivalence-on-if-symmetric*:
assumes *symmetric R*
shows *deflationary-on P R* = *rel-equivalence-on P R*
using *assms*
by (*simp add*: *deflationary-on-eq-inflationary-on-rel-inv rel-equivalence-on-eq*)

definition *rel-equivalence* (R :: $'a \Rightarrow$ -) $f \equiv$ *rel-equivalence-on* ($\top$ :: $'a \Rightarrow$ *bool*) R f

lemma *rel-equivalence-eq-rel-equivalence-on*:
rel-equivalence (R :: $'a \Rightarrow$ -) f = *rel-equivalence-on* ($\top$ :: $'a \Rightarrow$ *bool*) R f
unfolding *rel-equivalence-def* **..**

lemma *rel-equivalenceI* [*intro*]:
assumes *inflationary R f*
and *deflationary R f*
shows *rel-equivalence R f*
unfolding *rel-equivalence-eq-rel-equivalence-on* **using** *assms*
by (*intro rel-equivalence-onI*)
(*auto dest*: *inflationary-on-if-inflationary deflationary-on-if-deflationary*)

lemma *rel-equivalenceE* [*elim*]:
assumes *rel-equivalence R f*
obtains *inflationary R f deflationary R f*
using *assms* **unfolding** *rel-equivalence-eq-rel-equivalence-on*
by (*elim rel-equivalence-onE*)
(*simp only*: *inflationary-eq-inflationary-on deflationary-eq-deflationary-on*)

lemma *inflationary-if-rel-equivalence*:
assumes *rel-equivalence R f*
shows *inflationary R f*
using *assms* **by** (*elim rel-equivalenceE*)

lemma *deflationary-if-rel-equivalence*:
assumes *rel-equivalence R f*
shows *deflationary R f*
using *assms* **by** (*elim rel-equivalenceE*)

lemma *rel-equivalence-on-if-rel-equivalence*:
fixes P :: $'a \Rightarrow$ *bool* **and** R :: $'a \Rightarrow$ -
assumes *rel-equivalence R f*
shows *rel-equivalence-on P R f*
using *assms* **by** (*intro rel-equivalence-onI*)

(*auto dest*: *inflationary-on-if-inflationary deflationary-on-if-deflationary*)

lemma *bi-related-if-rel-equivalence*:
assumes *rel-equivalence R f*
shows $x \equiv_R f\, x$
using *assms* **by** (*intro bi-relatedI*) (*auto dest*: *inflationaryD deflationaryD*)

lemma *rel-equivalence-if-all-bi-related*:
assumes $\bigwedge x.\ x \equiv_R f\, x$
shows *rel-equivalence R f*
using *assms* **by** *auto*

lemma *rel-equivalenceD*:
assumes *rel-equivalence R f*
shows *R x* (*f x*) *R* (*f x*) *x*
using *assms* **by** (*auto dest*: *bi-related-if-rel-equivalence*)

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*
using *assms* **by** (*intro reflexive-onI*) *blast*

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*
using *assms reflexive-on-in-field-if-transitive-if-rel-equivalence-on*
using *assms* **by** *blast*

end

1.3.9 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** $\leq_L$ *50*)
notation R (**infix** $\leq_R$ *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** $\geq_L$ *50*)

abbreviation (*input*) *ge-right* $\equiv (\leq_R)^{-1}$
notation *ge-right* (**infix** $\geq_R$ *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 $((\leq_L) \Rrightarrow_m (\leq_R))\ l$
and *reflexive-on* $P\ (\leq_L)$
and $P\ x$
shows $l\ x \leq_R l\ x$
using *assms* **by** *blast*

lemma *left-right-rel-left-self-if-reflexive-on-in-dom-right-if-mono-left*:
assumes $((\leq_L) \Rrightarrow_m (\leq_R))\ l$
and *reflexive-on* (*in-dom* $(\leq_R)$) $(\leq_R)$
and *in-dom* $(\leq_L)\ x$
shows $l\ x \leq_R l\ x$
using *assms* **by** *blast*

lemma *left-right-rel-left-self-if-reflexive-on-in-codom-right-if-mono-left*:
assumes $((\leq_L) \Rrightarrow_m (\leq_R))\ l$
and *reflexive-on* (*in-codom* $(\leq_R)$) $(\leq_R)$
and *in-codom* $(\leq_L)\ x$
shows $l\ x \leq_R l\ x$
using *assms* **by** *blast*

lemma *left-right-rel-left-self-if-reflexive-on-in-field-right-if-mono-left*:
assumes $((\leq_L) \Rrightarrow_m (\leq_R))\ l$

and *reflexive-on* (*in-field* $(\leq_R)$) $(\leq_R)$
and *in-field* $(\leq_L)$ x
shows $l\ x \leq_R l\ x$
using *assms* **by** *blast*

lemma *mono-wrt-rel-left-if-reflexive-on-if-le-eq-if-mono-wrt-in-field*:
assumes ([*in-field* $(\leq_L)$] $\Rrightarrow_m$ P) l
and $(\leq_L) \leq (=)$
and *reflexive-on* P $(\leq_R)$
shows $((\leq_L) \Rrightarrow_m (\leq_R))$ l
using *assms* **by** (*intro dep-mono-wrt-relI*) *auto*

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*: $\eta = r \circ l$ **unfolding** *unit-def* **by** *simp*

lemma *unit-eq* [*simp*]: $\eta\ x = r\ (l\ x)$ **by** (*simp add*: *unit-eq-comp*)

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 **.**

definition *counit* $\equiv$ *flip.unit*

notation *counit* (ε)

lemma *counit-eq-comp*: $\varepsilon = l \circ r$ **unfolding** *counit-def flip.unit-def* **by** *simp*

lemma *counit-eq* [*simp*]: $\varepsilon\ x = l\ (r\ x)$ **by** (*simp add*: *counit-eq-comp*)

end

context
begin

interpretation *flip* : *order-functors* R L r l **.**

lemma *flip-counit-eq-unit*: *flip.counit* $= \eta$
  by (*intro ext*) *simp*

lemma *flip-unit-eq-counit*: *flip.unit* $= \varepsilon$
  by (*intro ext*) *simp*

lemma *inflationary-on-unit-if-left-rel-right-if-left-right-relI*:
  assumes $((\le_L) \Rrightarrow_m (\le_R))\ l$
  and *reflexive-on* $P\ (\le_L)$
  and $\bigwedge x\ y.\ P\ x \Longrightarrow l\ x \le_R y \Longrightarrow x \le_L r\ y$
  shows *inflationary-on* $P\ (\le_L)\ \eta$
  using *assms* **by** (*intro inflationary-onI*) *auto*

lemma *deflationary-on-unit-if-right-left-rel-if-right-rel-leftI*:
  assumes $((\le_L) \Rrightarrow_m (\le_R))\ l$
  and *reflexive-on* $P\ (\le_L)$
  and $\bigwedge x\ y.\ P\ x \Longrightarrow y \le_R l\ x \Longrightarrow r\ y \le_L x$
  shows *deflationary-on* $P\ (\le_L)\ \eta$
  using *assms* **by** (*intro deflationary-onI*) *auto*

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* $P\ (\le_L)\ \eta \longleftrightarrow$ *inflationary-on* $P\ (\le_L)\ \eta$
  using *assms* **by** (*intro iffI rel-equivalence-onI inflationary-onI deflationary-onI*)
  (*auto dest*!: *inverse-onD*)

lemma *reflexive-on-left-if-inflationary-on-unit-if-inverse-on*:
  assumes *inverse-on* $P\ l\ r$
  and *inflationary-on* $P\ (\le_L)\ \eta$
  shows *reflexive-on* $P\ (\le_L)$
  using *assms* **by** (*intro reflexive-onI*) (*auto dest*!: *inverse-onD*)

lemma *rel-equivalence-on-unit-if-reflexive-on-if-inverse-on*:
  assumes *inverse-on* $P\ l\ r$
  and *reflexive-on* $P\ (\le_L)$
  shows *rel-equivalence-on* $P\ (\le_L)\ \eta$
  using *assms* **by** (*intro rel-equivalence-onI inflationary-onI deflationary-onI*)
  (*auto dest*!: *inverse-onD*)

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* $P\ (\le_L)\ \eta \longleftrightarrow$ *reflexive-on* $P\ (\le_L)$
  using *assms reflexive-on-left-if-inflationary-on-unit-if-inverse-on*
    rel-equivalence-on-unit-if-reflexive-on-if-inverse-on
  by (*intro iffI*) *auto*

end

Here is an example of a free theorem.

notepad
begin
  interpret *flip* : *order-functors* $R\ L\ r\ l$
    rewrites *flip.unit* $\equiv \varepsilon$ **by** (*simp only*: *flip-unit-eq-counit*)
  have $[\![((\le_R) \Rrightarrow_m (\le_L))\ r$; *reflexive-on* $P\ (\le_R)$; $\bigwedge x\ y.\ [\![P\ x;\ r\ x \le_L y]\!] \Longrightarrow x \le_R l\ y]\!]$
    $\Longrightarrow$ *inflationary-on* $P\ (\le_R)\ \varepsilon$ **for** P
    by (*fact flip.inflationary-on-unit-if-left-rel-right-if-left-right-relI*)
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 $(\le_L)$ and $(\le_R)$ consists of two monotone functions (i.e. order functors) l and r such that $(x \le_L r\ y) = (l\ x \le_R y)$. We call this the *Galois property*. l is called the *left adjoint* and r the *right adjoint*. We call $(\le_L)$ the *left relation* and $(\le_R)$ 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* $(\le_R)\ y \wedge x \le_L r\ y$

abbreviation *left-Galois* $\equiv$ *Galois*
notation *left-Galois* (**infix** ${}_L\lessapprox$ *50*)

abbreviation (*input*) *ge-Galois-left* $\equiv ({}_L\lessapprox)^{-1}$
notation *ge-Galois-left* (**infix** $\gtrapprox_L$ *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* $(\le_R)\ y$
  and $x \le_L r\ y$
  shows $x\ {}_L\lessapprox\ y$
  unfolding *Galois-def* **using** *assms* **by** *blast*

lemma *left-GaloisE* [*elim*]:
  assumes $x\ {}_L\lessapprox\ y$
  obtains *in-codom* $(\le_R)\ y\ x \le_L r\ y$
  using *assms* **unfolding** *Galois-def* **by** *blast*

corollary *in-dom-left-if-left-Galois*:
  assumes $x\ {}_L\lessapprox\ y$
  shows *in-dom* $(\le_L)\ x$
  using *assms* **by** *blast*

corollary *left-Galois-iff-in-codom-and-left-rel-right*:
$x \ {}_{L}\lessapprox \ y \longleftrightarrow$ *in-codom* $(\le_R)$ $y \wedge x \le_L r\ y$
by *blast*

lemma *left-Galois-restrict-left-eq-left-Galois-left-restrict-left*:
$({}_{L}\lessapprox)\restriction_{P\ ::\ 'a \Rightarrow bool} =$ *galois-rel.Galois* $(\le_L)\restriction_P$ $(\le_R)$ r
by (*intro ext iffI galois-rel.left-GaloisI restrict-leftI*)
(*auto elim*: *galois-rel.left-GaloisE*)

lemma *left-Galois-restrict-right-eq-left-Galois-right-restrict-right*:
$({}_{L}\lessapprox)\upharpoonright_{P\ ::\ 'd \Rightarrow bool} =$ *galois-rel.Galois* $(\le_L)$ $(\le_R)\upharpoonright_P$ r
by (*intro ext iffI galois-rel.left-GaloisI restrict-rightI*)
(*auto elim*!: *galois-rel.left-GaloisE restrict-rightE*)

end

end

Equivalences

theory *Order-Equivalences*
imports
Order-Functors-Base
Partial-Equivalence-Relations
Preorders
begin

context *order-functors*
begin

definition *order-equivalence* $\equiv$
$((\le_L) \Rrightarrow_m (\le_R))$ $l \wedge$
$((\le_R) \Rrightarrow_m (\le_L))$ $r \wedge$
rel-equivalence-on (*in-field* $(\le_L)$) $(\le_L)$ $\eta \wedge$
rel-equivalence-on (*in-field* $(\le_R)$) $(\le_R)$ ε

notation *order-functors.order-equivalence* (**infix** $\equiv_o$ *50*)

lemma *order-equivalenceI* [*intro*]:
assumes $((\le_L) \Rrightarrow_m (\le_R))$ l
and $((\le_R) \Rrightarrow_m (\le_L))$ r
and *rel-equivalence-on* (*in-field* $(\le_L)$) $(\le_L)$ η
and *rel-equivalence-on* (*in-field* $(\le_R)$) $(\le_R)$ ε
shows $((\le_L) \equiv_o (\le_R))$ $l\ r$
unfolding *order-equivalence-def* **using** *assms* **by** *blast*

lemma *order-equivalenceE* [*elim*]:
assumes $((\le_L) \equiv_o (\le_R))$ $l\ r$

obtains $((\leq_L) \Rrightarrow_m (\leq_R))$ *l* $((\leq_R) \Rrightarrow_m (\leq_L))$ *r*
rel-equivalence-on (*in-field* $(\leq_L)$) $(\leq_L)$ η
rel-equivalence-on (*in-field* $(\leq_R)$) $(\leq_R)$ ε
using *assms* **unfolding** *order-equivalence-def* **by** *blast*

interpretation *of* : *order-functors S T f g* **for** *S T f g* **.**

lemma *rel-inv-order-equivalence-eq-order-equivalence* [*simp*]:
$((\leq_R) \equiv_o (\leq_L))^{-1} = ((\leq_L) \equiv_o (\leq_R))$
by (*intro ext*)
(*auto intro*!: *of.order-equivalenceI simp*: *of.flip-unit-eq-counit*)

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*
by (*simp flip*: *rel-inv-order-equivalence-eq-order-equivalence*)

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*]:
$((\geq_L) \equiv_o (\geq_R)) = ((\leq_L) \equiv_o (\leq_R))$
by (*intro ext*) (*auto intro*!: *of.order-equivalenceI*)

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* $(\leq_L)$
using *assms* **by** (*elim order-equivalenceE*)
(*rule in-codom-eq-in-dom-if-rel-equivalence-on-in-field*)

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* $(\leq_L)$) $(\leq_L)$
using *assms* **by** (*elim order-equivalenceE*)
(*rule preorder-on-in-field-if-transitive-if-rel-equivalence-on*)

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 −
from *assms* **obtain** $cy\ cy'$ **where** $(cy :: 'b) \neq cy'$ **by** *blast*
let $?cx = undefined :: 'a$
let $?L = \lambda x\ x'.\ ?cx = x \wedge x = x'$
and $?R = \lambda y\ y'.\ (y = cy \vee y = cy') \wedge (y' = cy \vee y' = cy') \wedge (y \neq cy' \vee y' \neq cy')$

and *?l* = $\lambda(a :: 'a)$. *cy*
and *?r* = $\lambda(b :: 'b)$. *?cx*
have $(?L \equiv_o ?R)$ *?l ?r* **using** ‹$cy \neq cy'$›
by (*intro of.order-equivalenceI*) (*auto 0 4*)
moreover have *partial-equivalence-rel ?L* **by** *blast*
moreover have
¬(*transitive-on* (*in-field ?R*) *?R*) **and** ¬(*reflexive-on* (*in-field ?R*) *?R*)
using ‹$cy \neq cy'$› **by** *auto*
ultimately show *?thesis* **by** *blast*
qed

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* **.**

interpretation *gr-flip-inv* : *galois-rel* $(\geq_R)$ $(\geq_L)$ *l* **.**

abbreviation *right-ge-Galois* ≡ *gr-flip-inv.Galois*
notation *right-ge-Galois* (**infix** $_R\gtrapprox$ *50*)

abbreviation (*input*) *Galois-right* ≡ *gr-flip-inv.ge-Galois-left*
notation *Galois-right* (**infix** $\lessapprox_R$ *50*)

lemma *Galois-rightI* [*intro*]:
assumes *in-dom* $(\leq_L)$ *x*
and $l\ x \leq_R y$
shows $x \lessapprox_R y$
using *assms* **by** *blast*

lemma *Galois-rightE* [*elim*]:
assumes $x \lessapprox_R y$
obtains *in-dom* $(\le_L)$ x $l\ x \le_R y$
using *assms* **by** *blast*

corollary *Galois-right-iff-in-dom-and-left-right-rel*:
$x \lessapprox_R y \longleftrightarrow$ *in-dom* $(\le_L)$ $x \wedge l\ x \le_R y$
by *blast*

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* $\equiv \forall\, x\ y.\ x\ {}_L\!\lessapprox\ y \longrightarrow l\ x \le_R y$

notation *galois-prop.half-galois-prop-left* (**infix** ${}_h\trianglelefteq$ *50*)

lemma *half-galois-prop-leftI* [*intro*]:
assumes $\bigwedge x\ y.\ x\ {}_L\!\lessapprox\ y \Longrightarrow l\ x \le_R y$
shows $((\le_L)\ {}_h\trianglelefteq\ (\le_R))\ l\ r$
unfolding *half-galois-prop-left-def* **using** *assms* **by** *blast*

lemma *half-galois-prop-leftD* [*dest*]:
assumes $((\le_L)\ {}_h\trianglelefteq\ (\le_R))\ l\ r$
and $x\ {}_L\!\lessapprox\ y$
shows $l\ x \le_R y$
using *assms* **unfolding** *half-galois-prop-left-def* **by** *blast*

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* $(\ge_R)$ $(\ge_L)$ r l **.**

definition *half-galois-prop-right* $\equiv$ *flip-inv.half-galois-prop-left*

notation *galois-prop.half-galois-prop-right* (**infix** $\trianglelefteq_h$ *50*)

lemma *half-galois-prop-rightI* [*intro*]:
assumes $\bigwedge x\ y.\ x \lessapprox_R y \Longrightarrow x \le_L r\ y$
shows $((\le_L) \trianglelefteq_h (\le_R))\ l\ r$
unfolding *half-galois-prop-right-def* **using** *assms* **by** *blast*

lemma *half-galois-prop-rightD* [*dest*]:
assumes $((\le_L) \trianglelefteq_h (\le_R))\ l\ r$

and $x \lessapprox_R y$
shows $x \leq_L r\ y$
using *assms* **unfolding** *half-galois-prop-right-def* **by** *blast*

interpretation g : *galois-prop* $S\ T\ f\ g$ **for** $S\ T\ f\ g$ **.**

lemma *rel-inv-half-galois-prop-right-eq-half-galois-prop-left-rel-inv* [*simp*]:
$((\leq_R) \trianglelefteq_h (\leq_L))^{-1} = ((\geq_L)\ {}_h\trianglelefteq (\geq_R))$
by (*intro ext*) *blast*

corollary *half-galois-prop-left-rel-inv-iff-half-galois-prop-right* [*iff*]:
$((\geq_L)\ {}_h\trianglelefteq (\geq_R))\ f\ g \longleftrightarrow ((\leq_R) \trianglelefteq_h (\leq_L))\ g\ f$
by (*simp flip*: *rel-inv-half-galois-prop-right-eq-half-galois-prop-left-rel-inv*)

lemma *rel-inv-half-galois-prop-left-eq-half-galois-prop-right-rel-inv* [*simp*]:
$((\leq_R)\ {}_h\trianglelefteq (\leq_L))^{-1} = ((\geq_L) \trianglelefteq_h (\geq_R))$
by (*intro ext*) *blast*

corollary *half-galois-prop-right-rel-inv-iff-half-galois-prop-left* [*iff*]:
$((\geq_L) \trianglelefteq_h (\geq_R))\ f\ g \longleftrightarrow ((\leq_R)\ {}_h\trianglelefteq (\leq_L))\ g\ f$
by (*simp flip*: *rel-inv-half-galois-prop-left-eq-half-galois-prop-right-rel-inv*)

end

context *galois*
begin

sublocale *galois-prop* $L\ R\ l\ r$ **.**

interpretation *flip* : *galois* $R\ L\ r\ l$ **.**

abbreviation *right-Galois* $\equiv$ *flip.Galois*
notation *right-Galois* (**infix** ${}_R\lessapprox$ *50*)

abbreviation (*input*) *ge-Galois-right* $\equiv$ *flip.ge-Galois-left*
notation *ge-Galois-right* (**infix** $\gtrapprox_R$ *50*)

abbreviation *left-ge-Galois* $\equiv$ *flip.right-ge-Galois*
notation *left-ge-Galois* (**infix** ${}_L\gtrapprox$ *50*)

abbreviation (*input*) *Galois-left* $\equiv$ *flip.Galois-right*
notation *Galois-left* (**infix** $\lessapprox_L$ *50*)

context
begin

interpretation *flip-inv* : *galois* $(\geq_R)\ (\geq_L)\ r\ l$ **.**

lemma *rel-unit-if-left-rel-if-mono-wrt-relI*:

assumes $((\leq_L) \Rrightarrow_m (\leq_R))$ *l*
and $x \lessapprox_R l\ x' \Longrightarrow x \leq_L \eta\ x'$
and $x \leq_L x'$
shows $x \leq_L \eta\ x'$
using *assms* **by** *auto*

corollary *rel-unit-if-left-rel-if-half-galois-prop-right-if-mono-wrt-rel*:
assumes $((\leq_L) \Rrightarrow_m (\leq_R))$ *l*
and $((\leq_L) \trianglelefteq_h (\leq_R))$ *l r*
and $x \leq_L x'$
shows $x \leq_L \eta\ x'$
using *assms* **by** (*auto intro*: *rel-unit-if-left-rel-if-mono-wrt-relI*)

corollary *rel-unit-if-reflexive-on-if-half-galois-prop-right-if-mono-wrt-rel*:
assumes $((\leq_L) \Rrightarrow_m (\leq_R))$ *l*
and $((\leq_L) \trianglelefteq_h (\leq_R))$ *l r*
and *reflexive-on* P $(\leq_L)$
and $P\ x$
shows $x \leq_L \eta\ x$
using *assms* **by** (*blast intro*: *rel-unit-if-left-rel-if-half-galois-prop-right-if-mono-wrt-rel*)

corollary *inflationary-on-unit-if-reflexive-on-if-half-galois-prop-rightI*:
fixes $P :: 'a \Rightarrow bool$
assumes $((\leq_L) \Rrightarrow_m (\leq_R))$ *l*
and $((\leq_L) \trianglelefteq_h (\leq_R))$ *l r*
and *reflexive-on* P $(\leq_L)$
shows *inflationary-on* P $(\leq_L)$ η
using *assms* **by** (*intro inflationary-onI*)
(*fastforce intro*: *rel-unit-if-reflexive-on-if-half-galois-prop-right-if-mono-wrt-rel*)

interpretation *flip* : *galois-prop* $R\ L\ r\ l$ **.**

lemma *right-rel-if-Galois-left-right-if-deflationary-onI*:
assumes $((\leq_R) \Rrightarrow_m (\leq_L))$ *r*
and $((\leq_R) \trianglelefteq_h (\leq_L))$ *r l*
and *deflationary-on* P $(\leq_R)$ ε
and *transitive* $(\leq_R)$
and $y \lessapprox_L r\ y'$
and $P\ y'$
shows $y \leq_R y'$
using *assms* **by** *force*

lemma *half-galois-prop-left-left-right-if-transitive-if-deflationary-on-if-mono-wrt-rel*:
assumes $((\leq_L) \Rrightarrow_m (\leq_R))$ *l*
and *deflationary-on* (*in-codom* $(\leq_R)$) $(\leq_R)$ ε
and *transitive* $(\leq_R)$
shows $((\leq_L)\ {}_h\!\trianglelefteq (\leq_R))$ *l r*
using *assms* **by** (*intro half-galois-prop-leftI*) *fastforce*

end

interpretation *flip-inv* : *galois* $(\geq_R)$ $(\geq_L)$ r l
rewrites *flip-inv.unit* $\equiv \varepsilon$ **and** *flip-inv.counit* $\equiv \eta$
and $\bigwedge R\ S.\ (R^{-1} \Rrightarrow_m S^{-1}) \equiv (R \Rrightarrow_m S)$
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 $\bigwedge P\ R.$ *inflationary-on* $P\ R^{-1} \equiv$ *deflationary-on* $P\ R$
and $\bigwedge P\ R.$ *deflationary-on* $P\ R^{-1} \equiv$ *inflationary-on* $P\ R$
and $\bigwedge(P :: {'b} \Rightarrow bool).$ *reflexive-on* $P\ (\geq_R) \equiv$ *reflexive-on* $P\ (\leq_R)$
and $\bigwedge R.$ *transitive* $R^{-1} \equiv$ *transitive* R
and $\bigwedge R.$ *in-codom* $R^{-1} \equiv$ *in-dom* R
by (*simp-all add*: *flip-unit-eq-counit flip-counit-eq-unit*
galois-prop.half-galois-prop-left-rel-inv-iff-half-galois-prop-right
galois-prop.half-galois-prop-right-rel-inv-iff-half-galois-prop-left)

corollary *counit-rel-if-right-rel-if-mono-wrt-relI*:
assumes $((\leq_R) \Rrightarrow_m (\leq_L))\ r$
and $r\ y\ {}_L\lessapprox\ y' \Longrightarrow \varepsilon\ y \leq_R y'$
and $y \leq_R y'$
shows $\varepsilon\ y \leq_R y'$
using *assms*
by (*fact flip-inv.rel-unit-if-left-rel-if-mono-wrt-relI*
[*simplified rel-inv-iff-rel*])

corollary *counit-rel-if-right-rel-if-half-galois-prop-left-if-mono-wrt-rel*:
assumes $((\leq_R) \Rrightarrow_m (\leq_L))\ r$
and $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$
and $y \leq_R y'$
shows $\varepsilon\ y \leq_R y'$
using *assms*
by (*fact flip-inv.rel-unit-if-left-rel-if-half-galois-prop-right-if-mono-wrt-rel*
[*simplified rel-inv-iff-rel*])

corollary *counit-rel-if-reflexive-on-if-half-galois-prop-left-if-mono-wrt-rel*:
assumes $((\leq_R) \Rrightarrow_m (\leq_L))\ r$
and $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$
and *reflexive-on* $P\ (\leq_R)$
and $P\ y$
shows $\varepsilon\ y \leq_R y$
using *assms*
by (*fact flip-inv.rel-unit-if-reflexive-on-if-half-galois-prop-right-if-mono-wrt-rel*
[*simplified rel-inv-iff-rel*])

corollary *deflationary-on-counit-if-reflexive-on-if-half-galois-prop-leftI*:
fixes $P :: {'b} \Rightarrow bool$
assumes $((\leq_R) \Rrightarrow_m (\leq_L))\ r$
and $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$

and *reflexive-on* P $(\leq_R)$
shows *deflationary-on* P $(\leq_R)$ ε
using *assms*
by (*fact flip-inv.inflationary-on-unit-if-reflexive-on-if-half-galois-prop-rightI*)

corollary *left-rel-if-left-right-Galois-if-inflationary-onI*:
assumes $((\leq_L) \Rrightarrow_m (\leq_R))$ l
and $((\leq_R)\ {}_h\trianglelefteq\ (\leq_L))$ r l
and *inflationary-on* P $(\leq_L)$ η
and *transitive* $(\leq_L)$
and $l\ x\ {}_R\lessapprox\ x'$
and $P\ x$
shows $x \leq_L x'$
using *assms* **by** (*intro flip-inv.right-rel-if-Galois-left-right-if-deflationary-onI*
[*simplified rel-inv-iff-rel*])

corollary *half-galois-prop-right-left-right-if-transitive-if-inflationary-on-if-mono-wrt-rel*:
assumes $((\leq_R) \Rrightarrow_m (\leq_L))$ r
and *inflationary-on* (*in-dom* $(\leq_L)$) $(\leq_L)$ η
and *transitive* $(\leq_L)$
shows $((\leq_L) \trianglelefteq_h (\leq_R))$ l r
using *assms*
by (*fact flip-inv.half-galois-prop-left-left-right-if-transitive-if-deflationary-on-if-mono-wrt-rel*)

end

context *order-functors*
begin

interpretation g : *galois* $L\ R\ l\ r$ **.**
interpretation *flip-g* : *galois* $R\ L\ r\ l$
rewrites *flip-g.unit* $\equiv \varepsilon$ **and** *flip-g.counit* $\equiv \eta$
by (*simp-all only*: *flip-unit-eq-counit flip-counit-eq-unit*)

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 $x \leq_L x'$
using *assms* **by** (*auto intro*!:
flip-g.right-rel-if-Galois-left-right-if-deflationary-onI
g.half-galois-prop-right-left-right-if-transitive-if-inflationary-on-if-mono-wrt-rel
elim!: *rel-equivalence-onE*
intro: *inflationary-on-if-le-pred-if-inflationary-on*
in-field-if-in-dom in-field-if-in-codom)

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** $\trianglelefteq$ *50*)

lemma *galois-propI* [*intro*]:
  assumes $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$
  and $((\leq_L) \trianglelefteq_h (\leq_R))\ l\ r$
  shows $((\leq_L) \trianglelefteq (\leq_R))\ l\ r$
  unfolding *galois-prop-def* **using** *assms* **by** *auto*

lemma *galois-propI'*:
  assumes $\bigwedge x\ y.\ \textit{in-dom}\ (\leq_L)\ x \Longrightarrow \textit{in-codom}\ (\leq_R)\ y \Longrightarrow x \leq_L r\ y \longleftrightarrow l\ x \leq_R y$
  shows $((\leq_L) \trianglelefteq (\leq_R))\ l\ r$
  using *assms* **by** *blast*

lemma *galois-propE* [*elim*]:
  assumes $((\leq_L) \trianglelefteq (\leq_R))\ l\ r$
  obtains $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r\ ((\leq_L) \trianglelefteq_h (\leq_R))\ l\ r$
  using *assms* **unfolding** *galois-prop-def* **by** *auto*

interpretation *g* : *galois-prop* $S\ T\ f\ g$ **for** $S\ T\ f\ g$**.**

lemma *galois-prop-eq-half-galois-prop-left-rel-inf-half-galois-prop-right*:
  $((\leq_L) \trianglelefteq (\leq_R)) = ((\leq_L)\ {}_h\trianglelefteq\ (\leq_R)) \sqcap ((\leq_L) \trianglelefteq_h (\leq_R))$
  by (*intro ext*) *auto*

lemma *galois-prop-left-rel-right-iff-left-right-rel*:
  assumes $((\leq_L) \trianglelefteq (\leq_R))\ l\ r$
  and $\textit{in-dom}\ (\leq_L)\ x\ \textit{in-codom}\ (\leq_R)\ y$
  shows $x \leq_L r\ y \longleftrightarrow l\ x \leq_R y$
  using *assms* **by** *blast*

lemma *rel-inv-galois-prop-eq-galois-prop-rel-inv* [*simp*]:
  $((\leq_R) \trianglelefteq (\leq_L))^{-1} = ((\geq_L) \trianglelefteq (\geq_R))$

by (*intro ext*) *blast*

corollary *galois-prop-rel-inv-iff-galois-prop* [*iff*]:
$((\geq_L) \trianglelefteq (\geq_R))\ f\ g \longleftrightarrow ((\leq_R) \trianglelefteq (\leq_L))\ g\ f$
by *auto*

end

context *galois*
begin

lemma *galois-prop-left-right-if-transitive-if-deflationary-on-if-inflationary-on-if-mono-wrt-rel*:
assumes $((\leq_L) \Rrightarrow_m (\leq_R))\ l$ **and** $((\leq_R) \Rrightarrow_m (\leq_L))\ r$
and *inflationary-on* (*in-dom* $(\leq_L)$) $(\leq_L)\ \eta$
and *deflationary-on* (*in-codom* $(\leq_R)$) $(\leq_R)\ \varepsilon$
and *transitive* $(\leq_L)$ *transitive* $(\leq_R)$
shows $((\leq_L) \trianglelefteq (\leq_R))\ l\ r$
using *assms*
by (*intro galois-propI*
half-galois-prop-left-left-right-if-transitive-if-deflationary-on-if-mono-wrt-rel
half-galois-prop-right-left-right-if-transitive-if-inflationary-on-if-mono-wrt-rel)

end

end

1.4.5 Galois Connections

theory *Galois-Connections*
imports
Galois-Property
begin

context *galois*
begin

definition *galois-connection* $\equiv$
$((\leq_L) \Rrightarrow_m (\leq_R))\ l \wedge ((\leq_R) \Rrightarrow_m (\leq_L))\ r \wedge ((\leq_L) \trianglelefteq (\leq_R))\ l\ r$

notation *galois.galois-connection* (**infix** $\dashv$ *50*)

lemma *galois-connectionI* [*intro*]:
assumes $((\leq_L) \Rrightarrow_m (\leq_R))\ l$ **and** $((\leq_R) \Rrightarrow_m (\leq_L))\ r$
and $((\leq_L) \trianglelefteq (\leq_R))\ l\ r$
shows $((\leq_L) \dashv (\leq_R))\ l\ r$
unfolding *galois-connection-def* **using** *assms* **by** *blast*

lemma *galois-connectionE* [*elim*]:

assumes $((\leq_L) \dashv (\leq_R))\ l\ r$
obtains $((\leq_L) \Rrightarrow_m (\leq_R))\ l\ ((\leq_R) \Rrightarrow_m (\leq_L))\ r\ ((\leq_L) \trianglelefteq (\leq_R))\ l\ r$
using *assms* **unfolding** *galois-connection-def* **by** *blast*

context
begin

interpretation g : *galois* $S\ T\ f\ g$ **for** $S\ T\ f\ g$**.**

lemma *rel-inv-galois-connection-eq-galois-connection-rel-inv* [*simp*]:
$((\leq_R) \dashv (\leq_L))^{-1} = ((\geq_L) \dashv (\geq_R))$
by (*intro ext*) *blast*

corollary *galois-connection-rel-inv-iff-galois-connection* [*iff*]:
$((\geq_L) \dashv (\geq_R))\ l\ r \longleftrightarrow ((\leq_R) \dashv (\leq_L))\ r\ l$
by (*simp flip*: *rel-inv-galois-connection-eq-galois-connection-rel-inv*)

lemma *rel-unit-if-left-rel-if-galois-connection*:
assumes $((\leq_L) \dashv (\leq_R))\ l\ r$
and $x \leq_L x'$
shows $x \leq_L \eta\ x'$
using *assms*
by (*blast intro*: *rel-unit-if-left-rel-if-half-galois-prop-right-if-mono-wrt-rel*)

end

lemma *counit-rel-if-right-rel-if-galois-connection*:
assumes $((\leq_L) \dashv (\leq_R))\ l\ r$
and $y \leq_R y'$
shows $\varepsilon\ y \leq_R y'$
using *assms*
by (*blast intro*: *counit-rel-if-right-rel-if-half-galois-prop-left-if-mono-wrt-rel*)

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 $x \leq_L \eta\ x$
using *assms*
by (*blast intro*: *rel-unit-if-reflexive-on-if-half-galois-prop-right-if-mono-wrt-rel*)

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 $\varepsilon\ y \leq_R y$
using *assms*
by (*blast intro*: *counit-rel-if-reflexive-on-if-half-galois-prop-left-if-mono-wrt-rel*)

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* $P\ (\leq_L)\ \eta$
  using *assms*
  by (*blast intro*: *inflationary-on-unit-if-reflexive-on-if-half-galois-prop-rightI*)

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* $P\ (\leq_R)\ \varepsilon$
  using *assms*
  by (*blast intro*: *deflationary-on-counit-if-reflexive-on-if-half-galois-prop-leftI*)

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** $\equiv_G$ *50*)

lemma *galois-equivalenceI* [*intro*]:

assumes $((\leq_L) \dashv (\leq_R))\ l\ r$
and $((\leq_R) \trianglelefteq (\leq_L))\ r\ l$
shows $((\leq_L) \equiv_G (\leq_R))\ l\ r$
unfolding *galois-equivalence-def* **using** *assms* **by** *blast*

lemma *galois-equivalenceE* [*elim*]:
assumes $((\leq_L) \equiv_G (\leq_R))\ l\ r$
obtains $((\leq_L) \dashv (\leq_R))\ l\ r$ $((\leq_R) \dashv (\leq_L))\ r\ l$
using *assms* **unfolding** *galois-equivalence-def*
by (*blast intro*: *galois.galois-connectionI*)

context
begin

interpretation *g* : *galois S T f g* **for** *S T f g*.

lemma *galois-equivalence-eq-galois-connection-rel-inf-galois-prop*:
$((\leq_L) \equiv_G (\leq_R)) = ((\leq_L) \dashv (\leq_R)) \sqcap ((\geq_L) \trianglelefteq (\geq_R))$
by (*intro ext*) *auto*

lemma *rel-inv-galois-equivalence-eq-galois-equivalence* [*simp*]:
$((\leq_R) \equiv_G (\leq_L))^{-1} = ((\leq_L) \equiv_G (\leq_R))$
by (*intro ext*) *auto*

corollary *galois-equivalence-right-left-iff-galois-equivalence-left-right*:
$((\leq_R) \equiv_G (\leq_L))\ r\ l \longleftrightarrow ((\leq_L) \equiv_G (\leq_R))\ l\ r$
by *auto*

lemma *galois-equivalence-rel-inv-eq-galois-equivalence* [*simp*]:
$((\geq_L) \equiv_G (\geq_R)) = ((\leq_L) \equiv_G (\leq_R))$
by (*intro ext*) *auto*

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* $P\ (\leq_L)\ \eta$
using *assms* **by** (*intro inflationary-on-unit-if-reflexive-on-if-galois-connection*)
(*elim galois-equivalenceE*)

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 −
interpret *flip* : *galois R L r l* .

show *?thesis* **using** *assms*
by (*auto intro*: *flip.deflationary-on-counit-if-reflexive-on-if-galois-connection simp only*: *flip.flip-unit-eq-counit*)
qed

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* $P\ (\leq_L)\ \eta$
using *assms* **by** (*intro rel-equivalence-onI inflationary-on-unit-if-reflexive-on-if-galois-equivalence deflationary-on-unit-if-reflexive-on-if-galois-equivalence*)

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 −
from *assms* **obtain** $cy\ cy'$ **where** $(cy :: 'b) \neq cy'$ **by** *blast*
let $?cx = undefined :: 'a$
let $?L = \lambda x\ x'.\ ?cx = x \wedge x = x'$
and $?R = \lambda y\ y'.\ (y = cy \vee y = cy') \wedge (y' = cy \vee y' = cy') \wedge (y \neq cy' \vee y' \neq cy')$
and $?l = \lambda(a :: 'a).\ cy$
and $?r = \lambda(b :: 'b).\ ?cx$
interpret *g* : *galois ?L ?R ?l ?r* **.**
interpret *flip-g* : *galois ?R ?L ?r ?l* **.**
have $(?L \equiv_G ?R)\ ?l\ ?r$ **using** $\langle cy \neq cy' \rangle$ **by** *blast*
moreover have *partial-equivalence-rel ?L* **by** *blast*
moreover have
$\neg($*transitive-on* (*in-field ?R*) $?R)$ **and** $\neg($*reflexive-on* (*in-field ?R*) $?R)$
using $\langle cy \neq cy' \rangle$ **by** *auto*
ultimately show *?thesis* **by** *blast*
qed

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 $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$
using *assms*
by (*intro half-galois-prop-left-left-right-if-transitive-if-deflationary-on-if-mono-wrt-rel*)
(*auto elim*!: *order-equivalenceE*
intro: *deflationary-on-if-le-pred-if-deflationary-on in-field-if-in-codom*
intro!: *le-predI*)

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 $((\leq_L) \trianglelefteq_h (\leq_R))\ l\ r$
using *assms*
by (*intro half-galois-prop-right-left-right-if-transitive-if-inflationary-on-if-mono-wrt-rel*)
(*auto elim*!: *order-equivalenceE*
intro: *inflationary-on-if-le-pred-if-inflationary-on in-field-if-in-dom*
intro!: *le-predI*
simp only: *flip-counit-eq-unit*)

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 $((\leq_L) \trianglelefteq (\leq_R))\ l\ r$
using *assms*
half-galois-prop-left-left-right-if-transitive-if-order-equivalence
half-galois-prop-right-left-right-if-transitive-if-order-equivalence
by *blast*

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 $((\leq_L) \dashv (\leq_R))\ l\ r$
using *assms galois-prop-left-right-if-transitive-if-order-equivalence*
by (*intro galois-connectionI*) *auto*

interpretation *flip* : *galois R L r l*
rewrites *flip.unit* $\equiv \varepsilon$
by (*simp only*: *flip-unit-eq-counit*)

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 $((\leq_L) \equiv_G (\leq_R))\ l\ r$
using *assms galois-connection-left-right-if-transitive-if-order-equivalence*
flip.galois-prop-left-right-if-transitive-if-order-equivalence
by (*intro galois-equivalenceI*)
(*auto simp only*: *order-equivalence-right-left-iff-order-equivalence-left-right*)

lemma *order-equivalence-if-reflexive-on-in-field-if-galois-equivalence*:

assumes $((\le_L) \equiv_G (\le_R))$ *l r*
and *reflexive-on* (*in-field* $(\le_L)$) $(\le_L)$ *reflexive-on* (*in-field* $(\le_R)$) $(\le_R)$
shows $((\le_L) \equiv_o (\le_R))$ *l r*
using *assms rel-equivalence-on-unit-if-reflexive-on-if-galois-equivalence*
flip.rel-equivalence-on-unit-if-reflexive-on-if-galois-equivalence
by (*intro order-equivalenceI*)
(*auto simp only*: *galois-equivalence-right-left-iff-galois-equivalence-left-right*)

corollary *galois-equivalence-eq-order-equivalence-if-preorder-on-in-field*:
assumes *preorder-on* (*in-field* $(\le_L)$) $(\le_L)$ *preorder-on* (*in-field* $(\le_R)$) $(\le_R)$
shows $((\le_L) \equiv_G (\le_R)) = ((\le_L) \equiv_o (\le_R))$
using *assms*
galois.order-equivalence-if-reflexive-on-in-field-if-galois-equivalence
galois.galois-equivalence-left-right-if-transitive-if-order-equivalence
by (*elim preorder-on-in-fieldE*, *intro ext*) *blast*

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* $(\ge_R)$ $(\ge_L)$ *l* **.**

lemma *left-Galois-if-Galois-right-if-half-galois-prop-right*:
assumes $((\le_L) \trianglelefteq_h (\le_R))$ *l r*
and $x \lessapprox_R y$
shows $x \; {}_L\!\lessapprox \; y$
using *assms* **by** (*intro left-GaloisI*) *auto*

lemma *Galois-right-if-left-Galois-if-half-galois-prop-left*:
assumes $((\le_L) \; {}_h\!\trianglelefteq \; (\le_R))$ *l r*
and $x \; {}_L\!\lessapprox \; y$
shows $x \lessapprox_R y$
using *assms* **by** *blast*

corollary *Galois-right-iff-left-Galois-if-galois-prop* [*iff*]:
assumes $((\le_L) \trianglelefteq (\le_R))$ *l r*
shows $x \lessapprox_R y \longleftrightarrow x \; {}_L\!\lessapprox \; y$

using *assms*
left-Galois-if-Galois-right-if-half-galois-prop-right
Galois-right-if-left-Galois-if-half-galois-prop-left
by *blast*

lemma *rel-inv-Galois-eq-flip-Galois-rel-inv-if-galois-prop* [*simp*]:
assumes $((\leq_L) \trianglelefteq (\leq_R))\ l\ r$
shows $(\gtrapprox_L) = (_R\gtrapprox)$
using *assms* **by** *blast*

corollary *flip-Galois-rel-inv-iff-Galois-if-galois-prop* [*iff*]:
assumes $((\leq_L) \trianglelefteq (\leq_R))\ l\ r$
shows $y\ _R\gtrapprox\ x \longleftrightarrow x\ _L\lessapprox\ y$
using *assms* **by** *blast*

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 *flip-inv.left-Galois*$^{-1}$ = *flip-inv.left-Galois*$^{-1}$
using *assms* **by** (*subst rel-inv-eq-iff-eq*[*symmetric*]) *simp*

end

context *galois*
begin

interpretation *flip-inv* : *galois* $(\geq_R)$ $(\geq_L)$ $r\ l$ **.**

context
begin

interpretation *flip* : *galois* $R\ L\ r\ l$ **.**

lemma *left-Galois-left-if-left-relI*:
assumes $((\leq_L) \Rrightarrow_m (\leq_R))\ l$
and $((\leq_L) \trianglelefteq_h (\leq_R))\ l\ r$
and $x \leq_L x'$
shows $x\ _L\lessapprox\ l\ x'$
using *assms*
by (*intro left-Galois-if-Galois-right-if-half-galois-prop-right*) *auto*

corollary *left-Galois-left-if-reflexive-on-if-half-galois-prop-rightI*:
assumes $((\leq_L) \Rrightarrow_m (\leq_R))\ l$
and $((\leq_L) \trianglelefteq_h (\leq_R))\ l\ r$
and *reflexive-on* P $(\leq_L)$
and $P\ x$
shows $x\ _L\lessapprox\ l\ x$
using *assms* **by** (*intro left-Galois-left-if-left-relI*) *auto*

lemma *left-Galois-left-if-in-codom-if-inflationary-onI*:

assumes $((\le_L) \Rrightarrow_m (\le_R))$ *l*
and *inflationary-on* P $(\le_L)$ η
and *in-codom* $(\le_L)$ x
and P x
shows x ${}_L\lessapprox$ l x
using *assms* **by** (*intro left-GaloisI*) (*auto elim*!: *in-codomE*)

lemma *left-Galois-left-if-in-codom-if-inflationary-on-in-codomI*:
assumes $((\le_L) \Rrightarrow_m (\le_R))$ *l*
and *inflationary-on* (*in-codom* $(\le_L)$) $(\le_L)$ η
and *in-codom* $(\le_L)$ x
shows x ${}_L\lessapprox$ l x
using *assms* **by** (*auto intro*!: *left-Galois-left-if-in-codom-if-inflationary-onI*)

lemma *left-Galois-left-if-left-rel-if-inflationary-on-in-fieldI*:
assumes $((\le_L) \Rrightarrow_m (\le_R))$ *l*
and *inflationary-on* (*in-field* $(\le_L)$) $(\le_L)$ η
and $x \le_L x$
shows x ${}_L\lessapprox$ l x
using *assms* **by** (*auto intro*!: *left-Galois-left-if-in-codom-if-inflationary-onI*)

lemma *right-left-Galois-if-right-relI*:
assumes $((\le_R) \Rrightarrow_m (\le_L))$ *r*
and $y \le_R y'$
shows r y ${}_L\lessapprox$ y'
using *assms* **by** (*intro left-GaloisI*) *auto*

corollary *right-left-Galois-if-reflexive-onI*:
assumes $((\le_R) \Rrightarrow_m (\le_L))$ *r*
and *reflexive-on* P $(\le_R)$
and P y
shows r y ${}_L\lessapprox$ y
using *assms* **by** (*intro right-left-Galois-if-right-relI*) *auto*

lemma *left-Galois-if-right-rel-if-left-GaloisI*:
assumes $((\le_R) \Rrightarrow_m (\le_L))$ *r*
and *transitive* $(\le_L)$
and x ${}_L\lessapprox$ y
and $y \le_R z$
shows x ${}_L\lessapprox$ z
using *assms* **by** (*intro left-GaloisI*) *auto*

lemma *left-Galois-if-left-Galois-if-left-relI*:
assumes *transitive* $(\le_L)$
and $x \le_L y$
and y ${}_L\lessapprox$ z
shows x ${}_L\lessapprox$ z
using *assms* **by** (*intro left-GaloisI*) *auto*

lemma *left-rel-if-right-Galois-if-left-GaloisI*:
  assumes $((\le_R)\ {}_h\trianglelefteq\ (\le_L))\ r\ l$
  and *transitive* $(\le_L)$
  and $x\ {}_L\lessapprox\ y$
  and $y\ {}_R\lessapprox\ z$
  shows $x \le_L z$
  using *assms* **by** *auto*

lemma *Dep-Fun-Rel-left-Galois-right-Galois-if-mono-wrt-rel* [*intro*]:
  assumes $((\le_L) \Rrightarrow_m (\le_R))\ l$
  shows $(({}_L\lessapprox) \Rrightarrow ({}_R\lessapprox))\ l\ r$
  using *assms* **by** *auto*

lemma *left-ge-Galois-eq-left-Galois-if-in-codom-eq-in-dom-if-symmetric*:
  assumes *symmetric* $(\le_L)$
  and *in-codom* $(\le_R)$ = *in-dom* $(\le_R)$
  shows $({}_L\gtrapprox) = ({}_L\lessapprox)$ — Note that *flip.right-ge-Galois* = *flip.right-ge-Galois*
  using *assms* **by** (*intro ext iffI*)
  (*auto elim*!: *galois-rel.left-GaloisE intro*!: *galois-rel.left-GaloisI*)

end

interpretation *flip* : *galois* $R\ L\ r\ l$ **.**

lemma *ge-Galois-right-eq-left-Galois-if-symmetric-if-in-codom-eq-in-dom-if-galois-prop*:
  assumes $((\le_L) \trianglelefteq (\le_R))\ l\ r$
  and *in-codom* $(\le_L)$ = *in-dom* $(\le_L)$
  and *symmetric* $(\le_R)$
  shows $(\gtrapprox_R) = ({}_L\lessapprox)$ — Note that *flip.left-Galois*$^{-1}$ = *flip.left-Galois*$^{-1}$
  using *assms*
  by (*simp only*: *inv-flip-Galois-rel-inv-eq-Galois-if-galois-prop*
    flip: *flip.left-ge-Galois-eq-left-Galois-if-in-codom-eq-in-dom-if-symmetric*)

interpretation *gp* : *galois-prop* $({}_L\lessapprox)\ ({}_R\lessapprox)\ l\ l$ **.**

lemma *half-galois-prop-left-left-Galois-right-Galois-if-half-galois-prop-leftI* [*intro*]:
  assumes $((\le_L)\ {}_h\trianglelefteq\ (\le_R))\ l\ r$
  shows $(({}_L\lessapprox)\ {}_h\trianglelefteq\ ({}_R\lessapprox))\ l\ l$
  using *assms* **by** *fast*

lemma *half-galois-prop-right-left-Galois-right-Galois-if-half-galois-prop-rightI* [*intro*]:
  assumes $((\le_L) \trianglelefteq_h (\le_R))\ l\ r$
  shows $(({}_L\lessapprox) \trianglelefteq_h ({}_R\lessapprox))\ l\ l$
  using *assms* **by** *fast*

corollary *galois-prop-left-Galois-right-Galois-if-galois-prop* [*intro*]:
  assumes $((\le_L) \trianglelefteq (\le_R))\ l\ r$
  shows $(({}_L\lessapprox) \trianglelefteq ({}_R\lessapprox))\ l\ l$
  using *assms* **by** *blast*

end

end

theory *Galois*
imports
Galois-Equivalences
Galois-Relator
begin

Summary We define the concept of (partial) Galois connections, Galois equivalences, and the Galois relator. For details refer to [2].

end

Closure Operators

theory *Closure-Operators*
imports
Order-Functions-Base
begin

definition *idempotent-on* $P\ R\ f \equiv$ *rel-equivalence-on* P (*rel-map* $f\ R$) f

lemma *idempotent-onI* [*intro*]:
assumes $\bigwedge x.\ P\ x \Longrightarrow f\ x \equiv_R f\ (f\ x)$
shows *idempotent-on* $P\ R\ f$
unfolding *idempotent-on-def* **using** *assms* **by** *fastforce*

lemma *idempotent-onE* [*elim*]:
assumes *idempotent-on* $P\ R\ f$
and $P\ x$
obtains $R\ (f\ (f\ x))\ (f\ x)\ R\ (f\ x)\ (f\ (f\ x))$
using *assms* **unfolding** *idempotent-on-def* **by** *fastforce*

lemma *rel-equivalence-on-rel-map-iff-idempotent-on* [*iff*]:
rel-equivalence-on P (*rel-map* $f\ R$) $f \longleftrightarrow$ *idempotent-on* $P\ R\ f$
unfolding *idempotent-on-def* **by** *simp*

lemma *bi-related-if-idempotent-onD*:
assumes *idempotent-on* $P\ R\ f$
and $P\ x$
shows $f\ x \equiv_R f\ (f\ x)$
using *assms* **by** *blast*

definition *idempotent* $(R :: 'a \Rightarrow \text{-})\ f \equiv$ *idempotent-on* $(\top :: 'a \Rightarrow bool)\ R\ f$

lemma *idempotent-eq-idempotent-on*:

idempotent $(R :: 'a \Rightarrow \text{-})\ f = $ *idempotent-on* $(\top :: 'a \Rightarrow bool)\ R\ f$
unfolding *idempotent-def* **..**

lemma *idempotentI* [*intro*]:
assumes $\bigwedge x.\ R\ (f\ (f\ x))\ (f\ x)$
and $\bigwedge x.\ R\ (f\ x)\ (f\ (f\ x))$
shows *idempotent* $R\ f$
unfolding *idempotent-eq-idempotent-on* **using** *assms* **by** *blast*

lemma *idempotentE* [*elim*]:
assumes *idempotent* $R\ f$
obtains $R\ (f\ (f\ x))\ (f\ x)\ R\ (f\ x)\ (f\ (f\ x))$
using *assms* **unfolding** *idempotent-eq-idempotent-on* **by** (*blast intro*: *top1I*)

lemma *idempotent-on-if-idempotent*:
fixes $P :: 'a \Rightarrow bool$ **and** $R :: 'a \Rightarrow \text{-}$
assumes *idempotent* $R\ f$
shows *idempotent-on* $P\ R\ f$
using *assms* **by** (*intro idempotent-onI*) *auto*

definition *closure-operator* $R\ f \equiv$
$(R \Rrightarrow_m R)\ f \wedge$ *inflationary-on* (*in-field* R) $R\ f \wedge$ *idempotent-on* (*in-field* R) $R\ f$

lemma *closure-operatorI* [*intro*]:
assumes $(R \Rrightarrow_m R)\ f$
and *inflationary-on* (*in-field* R) $R\ f$
and *idempotent-on* (*in-field* R) $R\ f$
shows *closure-operator* $R\ f$
unfolding *closure-operator-def* **using** *assms* **by** *blast*

lemma *closure-operatorE* [*elim*]:
assumes *closure-operator* $R\ f$
obtains $(R \Rrightarrow_m R)\ f$ *inflationary-on* (*in-field* R) $R\ f$
idempotent-on (*in-field* R) $R\ f$
using *assms* **unfolding** *closure-operator-def* **by** *blast*

lemma *mono-wrt-rel-if-closure-operator*:
assumes *closure-operator* $R\ f$
shows $(R \Rrightarrow_m R)\ f$
using *assms* **by** (*elim closure-operatorE*)

lemma *inflationary-on-in-field-if-closure-operator*:
assumes *closure-operator* $R\ f$
shows *inflationary-on* (*in-field* R) $R\ f$
using *assms* **by** (*elim closure-operatorE*)

lemma *idempotent-on-in-field-if-closure-operator*:
assumes *closure-operator* $R\ f$
shows *idempotent-on* (*in-field* R) $R\ f$

using *assms* **by** (*elim closure-operatorE*)

end

theory *Order-Functions*
imports
Order-Functions-Base
Closure-Operators
begin

Summary Basic functions on orders.

end

theory *Order-Functors*
imports
Order-Functors-Base
Order-Equivalences
begin

Summary Functors between orders aka. order-homomorphisms aka. monotone functions.

end

1.5 Orders

theory *Orders*
imports
Equivalence-Relations
Order-Functions
Order-Functors
Partial-Equivalence-Relations
Partial-Orders
Preorders
begin

Summary Basic order-theoretic concepts.

end

1.6 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 generalised relators, e.g. *injective-on* and *dep-mono-wrt-pred*.
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
begin

definition *mem-of* $A\ x \equiv x \in A$
lemma *mem-of-eq* [*simp*]: *mem-of* $\equiv \lambda A\ x.\ x \in A$ **unfolding** *mem-of-def* **by** *simp*
lemma *mem-of-iff* [*iff*]: *mem-of* $A\ x \longleftrightarrow x \in A$ **by** *simp*

end

1.7 Relation Syntax

theory *HOL-Syntax-Bundles-Relations*
imports *HOL.Relation*
begin

bundle *HOL-relation-syntax*
begin
notation *relcomp* (**infixr** *O 75*)

notation *relcompp* (**infixr** *OO 75*)
notation *converse* ((-$^{-1}$) [*1000*] *999*)
notation *conversep* ((-$^{-1-1}$) [*1000*] *1000*)
notation (*ASCII*)
converse ((-$\widehat{\ }$−*1*) [*1000*] *999*) **and**
conversep ((-$\widehat{\ }$−−*1*) [*1000*] *1000*)
end
bundle *no-HOL-relation-syntax*
begin
no-notation *relcomp* (**infixr** *O 75*)
no-notation *relcompp* (**infixr** *OO 75*)
no-notation *converse* ((-$^{-1}$) [*1000*] *999*)
no-notation *conversep* ((-$^{-1-1}$) [*1000*] *1000*)
no-notation (*ASCII*)
converse ((-$\widehat{\ }$−*1*) [*1000*] *999*) **and**
conversep ((-$\widehat{\ }$−−*1*) [*1000*] *1000*)
end

end

1.7.1 Alignment With Definitions from HOL.Main

theory *HOL-Alignment-Binary-Relations*
imports
Main
HOL-Mem-Of
HOL-Syntax-Bundles-Relations
LBinary-Relations
begin

unbundle *no-HOL-relation-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* ⇒ -) ⇒ - ≡
antisymmetric-on (*mem-of S*)
end

lemma *antisymmetric-on-set-eq-antisymmetric-on-pred* [*simp*]:
(*antisymmetric-on* (*S* :: *′a set*) :: (*′a* ⇒ -) ⇒ *bool*) =
antisymmetric-on (*mem-of S*)
unfolding *antisymmetric-on-set-def* **by** *simp*

lemma *antisymmetric-on-set-iff-antisymmetric-on-pred* [*iff*]:
antisymmetric-on (*S* :: *'a set*) (*R* :: *'a* ⇒ *-*) ⟷ *antisymmetric-on* (*mem-of S*) *R*
by *simp*

lemma *antisymp-eq-antisymmetric* [*HOL-bin-rel-alignment*]:
antisymp = *antisymmetric*
by (*intro ext*) (*auto intro*: *antisympI dest*: *antisymmetricD antisympD*)

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* ⇒ *-*) ⇒ *-* ≡
rel-injective-on (*mem-of S*)
definition *rel-injective-at-set* (*S* :: *'a set*) :: (*'b* ⇒ *'a* ⇒ *-*) ⇒ *-* ≡
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* ⇒ *-*) ⇒ *bool*) =
rel-injective-on (*mem-of S*)
unfolding *rel-injective-on-set-def* **by** *simp*

lemma *rel-injective-on-set-iff-rel-injective-on-pred* [*iff*]:
rel-injective-on (*S* :: *'a set*) (*R* :: *'a* ⇒ *-*) ⟷ *rel-injective-on* (*mem-of S*) *R*
by *simp*

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*)
unfolding *rel-injective-at-set-def* **by** *simp*

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*
by *simp*

lemma *left-unique-eq-rel-injective* [*HOL-bin-rel-alignment*]:
left-unique = *rel-injective*
by (*intro ext*) (*blast intro*: *left-uniqueI dest*: *rel-injectiveD left-uniqueD*)

Irreflexive

overloading
irreflexive-on-set ≡ *irreflexive-on* :: *'a set* ⇒ (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*
begin
definition *irreflexive-on-set* (*S* :: *'a set*) :: (*'a* ⇒ *-*) ⇒ *-* ≡
irreflexive-on (*mem-of S*)
end

lemma *irreflexive-on-set-eq-irreflexive-on-pred* [*simp*]:
  (*irreflexive-on* (*S* :: ′*a set*) :: (′*a* ⇒ -) ⇒ *bool*) =
    irreflexive-on (*mem-of S*)
  unfolding *irreflexive-on-set-def* **by** *simp*

lemma *irreflexive-on-set-iff-irreflexive-on-pred* [*iff*]:
  irreflexive-on (*S* :: ′*a set*) (*R* :: ′*a* ⇒ -) ⟷
    irreflexive-on (*mem-of S*) *R*
  by *simp*

lemma *irreflp-on-eq-irreflexive-on* [*HOL-bin-rel-alignment*]:
  irreflp-on = *irreflexive-on*
  by (*intro ext*) (*blast intro*: *irreflp-onI dest*: *irreflp-onD*)

lemma *irreflp-eq-irreflexive* [*HOL-bin-rel-alignment*]: *irreflp* = *irreflexive*
  by (*intro ext*) (*blast intro*: *irreflpI dest*: *irreflexiveD irreflpD*)

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* ⇒ -) ⇒ - ≡
    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* ⇒ -) ⇒ *bool*) =
    left-total-on (*mem-of S*)
  unfolding *left-total-on-set-def* **by** *simp*

lemma *left-total-on-set-iff-left-total-on-pred* [*iff*]:
  left-total-on (*S* :: ′*a set*) (*R* :: ′*a* ⇒ -) ⟷ *left-total-on* (*mem-of S*) *R*
  by *simp*

lemma *Transfer-left-total-eq-left-total* [*HOL-bin-rel-alignment*]:
  Transfer.left-total = *Binary-Relations-Left-Total.left-total*
  by (*intro ext*) (*fast intro*: *Transfer.left-totalI*
    elim: *Transfer.left-totalE Binary-Relations-Left-Total.left-totalE*)

Reflexive **overloading**
  reflexive-on-set ≡ *reflexive-on* :: ′*a set* ⇒ (′*a* ⇒ ′*a* ⇒ *bool*) ⇒ *bool*
begin
  definition *reflexive-on-set* (*S* :: ′*a set*) :: (′*a* ⇒ -) ⇒ - ≡
    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*)
  unfolding *reflexive-on-set-def* **by** *simp*

lemma *reflexive-on-set-iff-reflexive-on-pred* [*iff*]:
reflexive-on (*S* :: '*a set*) (*R* :: '*a* ⇒ '*a* ⇒ *bool*) ⟷
reflexive-on (*mem-of S*) *R*
by *simp*

lemma *reflp-on-eq-reflexive-on* [*HOL-bin-rel-alignment*]:
reflp-on = *reflexive-on*
by (*intro ext*) (*blast intro*: *reflp-onI dest*: *reflp-onD*)

lemma *reflp-eq-reflexive* [*HOL-bin-rel-alignment*]: *reflp* = *reflexive*
by (*intro ext*) (*blast intro*: *reflpI dest*: *reflexiveD reflpD*)

Right-Unique overloading
right-unique-on-set ≡ *right-unique-on* :: '*a set* ⇒ ('*a* ⇒ '*b* ⇒ *bool*) ⇒ *bool*
right-unique-at-set ≡ *right-unique-at* :: '*a set* ⇒ ('*b* ⇒ '*a* ⇒ *bool*) ⇒ *bool*
begin
definition *right-unique-on-set* (*S* :: '*a set*) :: ('*a* ⇒ -) ⇒ - ≡
right-unique-on (*mem-of S*)
definition *right-unique-at-set* (*S* :: '*a set*) :: ('*b* ⇒ '*a* ⇒ -) ⇒ - ≡
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* ⇒ -) ⇒ *bool*) =
right-unique-on (*mem-of S*)
unfolding *right-unique-on-set-def* **by** *simp*

lemma *right-unique-on-set-iff-right-unique-on-pred* [*iff*]:
right-unique-on (*S* :: '*a set*) (*R* :: '*a* ⇒ -) ⟷ *right-unique-on* (*mem-of S*) *R*
by *simp*

lemma *right-unique-at-set-eq-right-unique-at-pred* [*simp*]:
(*right-unique-at* (*S* :: '*a set*) :: ('*b* ⇒ '*a* ⇒ *bool*) ⇒ *bool*) =
right-unique-at (*mem-of S*)
unfolding *right-unique-at-set-def* **by** *simp*

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*
by *simp*

lemma *Transfer-right-unique-eq-right-unique* [*HOL-bin-rel-alignment*]:
Transfer.right-unique = *Binary-Relations-Right-Unique.right-unique*
by (*intro ext*) (*blast intro*: *Transfer.right-uniqueI*
dest: *Transfer.right-uniqueD Binary-Relations-Right-Unique.right-uniqueD*)

Surjective overloading
rel-surjective-at-set ≡ *rel-surjective-at* :: '*a set* ⇒ ('*b* ⇒ '*a* ⇒ *bool*) ⇒ *bool*

begin
  definition *rel-surjective-at-set* $(S :: 'a\ set) :: ('b \Rightarrow 'a \Rightarrow \text{-}) \Rightarrow \text{-} \equiv$
    rel-surjective-at (*mem-of S*)
end

lemma *rel-surjective-at-set-eq-rel-surjective-at-pred* [*simp*]:
  (*rel-surjective-at* $(S :: 'a\ set) :: ('b \Rightarrow 'a \Rightarrow \text{-}) \Rightarrow bool) =$
    rel-surjective-at (*mem-of S*)
  unfolding *rel-surjective-at-set-def* **by** *simp*

lemma *rel-surjective-at-set-iff-rel-surjective-at-pred* [*iff*]:
  rel-surjective-at $(S :: 'a\ set)$ $(R :: 'b \Rightarrow 'a \Rightarrow \text{-}) \longleftrightarrow$ *rel-surjective-at* (*mem-of S*)
R
  by *simp*

lemma *Transfer-right-total-eq-rel-surjective* [*HOL-bin-rel-alignment*]:
  Transfer.right-total = *rel-surjective*
  by (*intro ext*) (*fast intro*: *Transfer.right-totalI rel-surjectiveI*
    elim: *Transfer.right-totalE rel-surjectiveE*)

Symmetric **overloading**
  symmetric-on-set $\equiv$ *symmetric-on* $:: 'a\ set \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
begin
  definition *symmetric-on-set* $(S :: 'a\ set) :: ('a \Rightarrow \text{-}) \Rightarrow \text{-} \equiv$
    symmetric-on (*mem-of S*)
end

lemma *symmetric-on-set-eq-symmetric-on-pred* [*simp*]:
  (*symmetric-on* $(S :: 'a\ set) :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool) =$
    symmetric-on (*mem-of S*)
  unfolding *symmetric-on-set-def* **by** *simp*

lemma *symmetric-on-set-iff-symmetric-on-pred* [*iff*]:
  symmetric-on $(S :: 'a\ set)$ $(R :: 'a \Rightarrow 'a \Rightarrow bool) \longleftrightarrow$
    symmetric-on (*mem-of S*) *R*
  by *simp*

lemma *symp-eq-symmetric* [*HOL-bin-rel-alignment*]: *symp* = *symmetric*
  by (*intro ext*) (*blast intro*: *sympI dest*: *symmetricD sympD*)

Transitive **overloading**
  transitive-on-set $\equiv$ *transitive-on* $:: 'a\ set \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
begin
  definition *transitive-on-set* $(S :: 'a\ set) :: ('a \Rightarrow \text{-}) \Rightarrow \text{-} \equiv$
    transitive-on (*mem-of S*)
end

lemma *transitive-on-set-eq-transitive-on-pred* [*simp*]:
  (*transitive-on* $(S :: 'a\ set) :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool) =$

$transitive\text{-}on\ (mem\text{-}of\ S)$
unfolding *transitive-on-set-def* **by** *simp*

lemma *transitive-on-set-iff-transitive-on-pred* [*iff*]:
$transitive\text{-}on\ (S :: 'a\ set)\ (R :: 'a \Rightarrow 'a \Rightarrow bool) \longleftrightarrow$
$transitive\text{-}on\ (mem\text{-}of\ S)\ R$
by *simp*

lemma *transp-eq-transitive* [*HOL-bin-rel-alignment*]: $transp = transitive$
by (*intro ext*) (*blast intro*: *transpI dest*: *transpD*)

Functions **lemma** *relcompp-eq-rel-comp* [*HOL-bin-rel-alignment*]: $relcompp = rel\text{-}comp$
by (*intro ext*) *auto*

lemma *conversep-eq-rel-inv* [*HOL-bin-rel-alignment*]: $conversep = rel\text{-}inv$
by (*intro ext*) *auto*

lemma *Domainp-eq-in-dom* [*HOL-bin-rel-alignment*]: $Domainp = in\text{-}dom$
by (*intro ext*) *auto*

lemma *Rangep-eq-in-codom* [*HOL-bin-rel-alignment*]: $Rangep = in\text{-}codom$
by (*intro ext*) *auto*

overloading
$restrict\text{-}left\text{-}set \equiv restrict\text{-}left :: ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow ('a\ set) \Rightarrow 'a \Rightarrow 'b \Rightarrow bool$
begin
definition $restrict\text{-}left\text{-}set\ (R :: 'a \Rightarrow \text{-})\ (S :: 'a\ set) \equiv R{\restriction}_{mem\text{-}of\ S}$
end

lemma *restrict-left-set-eq-restrict-left-pred* [*simp*]:
$(R{\restriction}_{S :: 'a\ set} :: 'a \Rightarrow \text{-}) = R{\restriction}_{mem\text{-}of\ S}$
unfolding *restrict-left-set-def* **by** *simp*

lemma *restrict-left-set-iff-restrict-left-pred* [*iff*]:
$(R{\restriction}_{S :: 'a\ set} :: 'a \Rightarrow \text{-})\ x\ y \longleftrightarrow R{\restriction}_{mem\text{-}of\ S}\ x\ y$
by *simp*

Restricted Equality **lemma** *eq-onp-eq-eq-restrict* [*HOL-bin-rel-alignment*]: $eq\text{-}onp = eq\text{-}restrict$
unfolding *eq-onp-def* **by** (*intro ext*) *auto*

overloading
$eq\text{-}restrict\text{-}set \equiv eq\text{-}restrict :: 'a\ set \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$
begin
definition $eq\text{-}restrict\text{-}set\ (S :: 'a\ set) \equiv ((=_{mem\text{-}of\ S}) :: 'a \Rightarrow \text{-})$
end

lemma *eq-restrict-set-eq-eq-restrict-pred* [*simp*]:

$((=_{S\ ::\ 'a\ set})\ ::\ 'a \Rightarrow \text{-}) = (=_{mem\text{-}of\ S})$
unfolding *eq-restrict-set-def* **by** *simp*

lemma *eq-restrict-set-iff-eq-restrict-pred* [*iff*]:
$(x\ ::\ 'a) =_{(S\ ::\ 'a\ set)} y \longleftrightarrow x =_{mem\text{-}of\ S} y$
by *simp*

end

1.7.2 Function Syntax

theory *HOL-Syntax-Bundles-Functions*
imports *HOL.Fun*
begin

bundle *HOL-function-syntax*
begin
notation *comp* (**infixl** ∘ *55*)
end
bundle *no-HOL-function-syntax*
begin
no-notation *comp* (**infixl** ∘ *55*)
end

end

1.7.3 Alignment With 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 **overloading**
bijection-on-set ≡ *bijection-on* $::\ 'a\ set \Rightarrow 'b\ set \Rightarrow ('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow 'a) \Rightarrow bool$
begin
definition *bijection-on-set* $(S\ ::\ 'a\ set)\ (S'\ ::\ 'b\ set)\ ::\ ('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow 'a) \Rightarrow bool \equiv$
bijection-on (*mem-of* S) (*mem-of* S')

end

lemma *bijection-on-set-eq-bijection-on-pred* [*simp*]:
  (*bijection-on* (*S* :: *'a set*) (*S'* :: *'b set*) :: (*'a* ⇒ *'b*) ⇒ *-*) =
    bijection-on (*mem-of S*) (*mem-of S'*)
  unfolding *bijection-on-set-def* **by** *simp*

lemma *bijection-on-set-iff-bijection-on-pred* [*iff*]:
  bijection-on (*S* :: *'a set*) (*S'* :: *'b set*) (*f* :: *'a* ⇒ *'b*) *g* ⟷
    bijection-on (*mem-of S*) (*mem-of S'*) *f g*
  by *simp*

lemma *bij-betw-bijection-onE*:
  assumes *bij-betw f S S'*
  obtains *g* **where** *bijection-on S S' f g*
proof
  let *?g* = *the-inv-into S f*
  from *assms bij-betw-the-inv-into* **have** *bij-betw ?g S' S* **by** *blast*
  with *assms* **show** *bijection-on S S' f ?g*
    by (*auto intro*!: *bijection-onI*
      dest: *bij-betw-apply bij-betw-imp-inj-on the-inv-into-f-f*
      simp: *f-the-inv-into-f-bij-betw*)
qed

lemma *bij-betw-if-bijection-on*:
  assumes *bijection-on S S' f g*
  shows *bij-betw f S S'*
  using *assms* **by** (*intro bij-betw-byWitness*[**where** *?f'*=*g*])
  (*auto elim*: *bijection-onE dest*: *inverse-onD*)

corollary *bij-betw-iff-ex-bijection-on* [*HOL-fun-alignment*]:
  bij-betw f S S' ⟷ (∃ *g*. *bijection-on S S' f g*)
  by (*intro iffI*)
  (*auto elim*!: *bij-betw-bijection-onE intro*: *bij-betw-if-bijection-on*)

Injective **overloading**
  injective-on-set ≡ *injective-on* :: *'a set* ⇒ (*'a* ⇒ *'b*) ⇒ *bool*
begin
  definition *injective-on-set* (*S* :: *'a set*) :: (*'a* ⇒ *'b*) ⇒ *bool* ≡
    injective-on (*mem-of S*)
end

lemma *injective-on-set-eq-injective-on-pred* [*simp*]:
  (*injective-on* (*S* :: *'a set*) :: (*'a* ⇒ *'b*) ⇒ *-*) = *injective-on* (*mem-of S*)
  unfolding *injective-on-set-def* **by** *simp*

lemma *injective-on-set-iff-injective-on-pred* [*iff*]:
  injective-on (*S* :: *'a set*) (*f* :: *'a* ⇒ *'b*) ⟷ *injective-on* (*mem-of S*) *f*
  by *simp*

lemma *inj-on-iff-injective-on* [*HOL-fun-alignment*]: *inj-on f P* $\longleftrightarrow$ *injective-on P f*
  by (*auto intro*: *inj-onI dest*: *inj-onD injective-onD*)

lemma *inj-eq-injective* [*HOL-fun-alignment*]: *inj* = *injective*
  by (*auto intro*: *injI dest*: *injD injectiveD*)

Inverse **overloading**
  inverse-on-set $\equiv$ *inverse-on* :: *'a set* $\Rightarrow$ (*'a* $\Rightarrow$ *'b*) $\Rightarrow$ (*'b* $\Rightarrow$ *'a*) $\Rightarrow$ *bool*
begin
  definition *inverse-on-set* (*S* :: *'a set*) :: (*'a* $\Rightarrow$ *'b*) $\Rightarrow$ - $\equiv$
    inverse-on (*mem-of S*)
end

lemma *inverse-on-set-eq-inverse-on-pred* [*simp*]:
  (*inverse-on* (*S* :: *'a set*) :: (*'a* $\Rightarrow$ *'b*) $\Rightarrow$ -) = *inverse-on* (*mem-of S*)
  unfolding *inverse-on-set-def* **by** *simp*

lemma *inverse-on-set-iff-inverse-on-pred* [*iff*]:
  inverse-on (*S* :: *'a set*) (*f* :: *'a* $\Rightarrow$ *'b*) *g* $\longleftrightarrow$ *inverse-on* (*mem-of S*) *f g*
  by *simp*

Monotone **lemma** *monotone-on-eq-mono-wrt-rel-restrict-left-right* [*HOL-fun-alignment*]:
  monotone-on S R = *mono-wrt-rel* ($R\restriction_{S}\upharpoonright_{S}$)
  unfolding *restrict-right-eq*
  by (*intro ext*) (*blast intro*: *monotone-onI dest*: *monotone-onD*)

lemma *monotone-eq-mono-wrt-rel* [*HOL-fun-alignment*]: *monotone* = *mono-wrt-rel*
  by (*intro ext*) (*auto intro*: *monotoneI dest*: *monotoneD*)

lemma *pred-fun-eq-mono-wrt-pred* [*HOL-fun-alignment*]: *pred-fun* = *mono-wrt-pred*
  by (*intro ext*) *auto*

lemma *Fun-mono-eq-mono* [*HOL-fun-alignment*]: *Fun.mono* = *mono*
  by (*intro ext*) (*auto intro*: *Fun.mono-onI dest*: *Fun.monoD*)

lemma *Fun-antimono-eq-antimono* [*HOL-fun-alignment*]: *Fun.antimono* = *antimono*
  by (*intro ext*) (*auto intro*: *monotoneI dest*: *monotoneD*)

Surjective **overloading**
  surjective-at-set $\equiv$ *surjective-at* :: *'a set* $\Rightarrow$ (*'b* $\Rightarrow$ *'a*) $\Rightarrow$ *bool*
begin
  definition *surjective-at-set* (*S* :: *'a set*) :: (*'b* $\Rightarrow$ *'a*) $\Rightarrow$ *bool* $\equiv$
    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*)
unfolding *surjective-at-set-def* **by** *simp*

lemma *surjective-at-set-iff-surjective-at-pred* [*iff*]:
surjective-at (*S* :: *'a set*) (*f* :: *'b* ⇒ *'a*) ⟷ *surjective-at* (*mem-of S*) *f*
by *simp*

lemma *surj-eq-surjective* [*HOL-fun-alignment*]: *surj* = *surjective*
by (*intro ext*) (*fast intro*: *surjI dest*: *surjD elim*: *surjectiveE*)

Functions **lemma** *Fun-id-eq-id* [*HOL-fun-alignment*]: *Fun.id* = *Functions-Base.id*
by (*intro ext*) *simp*

lemma *Fun-comp-eq-comp* [*HOL-fun-alignment*]: *Fun.comp* = *Functions-Base.comp*
by (*intro ext*) *simp*

lemma *map-fun-eq-fun-map* [*HOL-fun-alignment*]: *map-fun* = *fun-map*
by (*intro ext*) *simp*

Relators **lemma** *rel-fun-eq-Fun-Rel-rel* [*HOL-fun-alignment*]: *rel-fun* = *Fun-Rel-rel*
by (*intro ext*) (*auto dest*: *rel-funD*)

end

1.8 Order Syntax

theory *HOL-Syntax-Bundles-Orders*
imports *HOL.Orderings*
begin

bundle *HOL-order-syntax*
begin
notation
less-eq ('(≤')) **and**
less-eq ((-/ ≤ -) [*51*, *51*] *50*) **and**
less ('(<')) **and**
less ((-/ < -) [*51*, *51*] *50*)
notation (*input*) *greater-eq* (**infix** ≥ *50*)
notation (*input*) *greater* (**infix** > *50*)
notation (*ASCII*)
less-eq ('(<=')) **and**
less-eq ((-/ <= -) [*51*, *51*] *50*)
notation (*input*) *greater-eq* (**infix** >= *50*)
end
bundle *no-HOL-order-syntax*
begin

no-notation
  less-eq ('(≤')) **and**
  less-eq ((-/ ≤ -) [*51*, *51*] *50*) **and**
  less ('(<')) **and**
  less ((-/ < -) [*51*, *51*] *50*)
no-notation (*input*) *greater-eq* (**infix** ≥ *50*)
no-notation (*input*) *greater* (**infix** > *50*)
no-notation (*ASCII*)
  less-eq ('(<=')) **and**
  less-eq ((-/ <= -) [*51*, *51*] *50*)
no-notation (*input*) *greater-eq* (**infix** >= *50*)
end

end

1.8.1 Alignment With 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

Bi-Related **lemma** (**in** *preorder-equiv*) *equiv-eq-bi-related* [*HOL-order-alignment*]:
  equiv = *bi-related* (≤)
  by (*intro ext*) (*auto intro*: *equiv-antisym dest*: *equivD1 equivD2*)

Inflationary **overloading**
  inflationary-on-set ≡ *inflationary-on* :: '*a set* ⇒ ('*a* ⇒ '*a* ⇒ *bool*) ⇒ ('*a* ⇒ '*a*) ⇒ *bool*
begin
  definition *inflationary-on-set* (*S* :: '*a set*) :: ('*a* ⇒ -) ⇒ - ≡
    inflationary-on (*mem-of S*)
end

lemma *inflationary-on-set-eq-inflationary-on-pred* [*simp*]:
  (*inflationary-on* (*S* :: '*a set*) :: ('*a* ⇒ -) ⇒ -) = *inflationary-on* (*mem-of S*)
  unfolding *inflationary-on-set-def* **by** *simp*

lemma *inflationary-on-set-iff-inflationary-on-pred* [*iff*]:
  inflationary-on (*S* :: '*a set*) (*R* :: '*a* ⇒ -) *f* ⟷ *inflationary-on* (*mem-of S*) *R f*
  by *simp*

Terms like *deflationary-on*, *rel-equivalence-on*, and *idempotent-on* are automatically overloaded. One can get similar correspondence lemmas by unfolding the corresponding definitional theorems, e.g. *deflationary-on ?P ?R* = *inflationary-on ?P* $?R^{-1}$.

Properties

Equivalence Relations **lemma** *equiv-eq-equivalence-rel* [*HOL-order-alignment*]:
equivp = *equivalence-rel*
by (*intro ext*) (*fastforce intro*!: *equivpI*
simp: *HOL-bin-rel-alignment reflexive-eq-reflexive-on elim*!: *equivpE*)

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*
using *assms* **by** (*fastforce intro*!: *part-equivpI*
simp: *HOL-bin-rel-alignment elim*!: *part-equivpE*)

Partial Orders **lemma** (**in** *order*) *partial-order* [*HOL-order-alignment*]: *partial-order* (≤)
using *order-refl order-trans order-antisym* **by** *blast*

Preorders **lemma** (**in** *partial-preordering*) *preorder* [*HOL-order-alignment*]: *preorder* (≤)
using *refl trans* **by** *blast*

lemma *partial-preordering-eq* [*HOL-order-alignment*]:
partial-preordering = *Preorders.preorder*
by (*intro ext*) (*auto intro*: *partial-preordering.intro*
dest: *partial-preordering.trans partial-preordering.refl reflexiveD*)

end

1.9 HOL Alignments

theory *HOL-Alignments*
imports
HOL-Alignment-Binary-Relations
HOL-Alignment-Functions
HOL-Alignment-Orders
begin

Summary Alignment of concepts with HOL counterparts

end

1.9.1 Alignment With 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) (.=)
  by blast

lemma transitive-on-carrier [HOL-Algebra-order-alignment]:
  transitive-on (carrier S) (.=)
  using trans by blast

lemma preorder-on-carrier [HOL-Algebra-order-alignment]:
  preorder-on (carrier S) (.=)
  using reflexive-on-carrier transitive-on-carrier by blast

lemma symmetric-on-carrier [HOL-Algebra-order-alignment]:
  symmetric-on (carrier S) (.=)
  using sym by blast

lemma partial-equivalence-rel-on-carrier [HOL-Algebra-order-alignment]:
  partial-equivalence-rel-on (carrier S) (.=)
  using transitive-on-carrier symmetric-on-carrier by blast

lemma equivalence-rel-on-carrier [HOL-Algebra-order-alignment]:
  equivalence-rel-on (carrier S) (.=)
  using reflexive-on-carrier partial-equivalence-rel-on-carrier by blast

end

lemma equivalence-iff-equivalence-rel-on-carrier [HOL-Algebra-order-alignment]:
  equivalence S ⟷ equivalence-rel-on (carrier S) (.=_S)
  using equivalence.equivalence-rel-on-carrier
  by (blast dest: intro!: equivalence.intro dest: symmetric-onD transitive-onD)

context partial-order
begin

lemma reflexive-on-carrier [HOL-Algebra-order-alignment]:
  reflexive-on (carrier L) (⊑)
  by blast
```

lemma *transitive-on-carrier* [*HOL-Algebra-order-alignment*]:
transitive-on (*carrier* L) ($\sqsubseteq$)
using *le-trans* **by** *blast*

lemma *preorder-on-carrier* [*HOL-Algebra-order-alignment*]:
preorder-on (*carrier* L) ($\sqsubseteq$)
using *reflexive-on-carrier transitive-on-carrier* **by** *blast*

lemma *antisymmetric-on-carrier* [*HOL-Algebra-order-alignment*]:
antisymmetric-on (*carrier* L) ($\sqsubseteq$)
by *blast*

lemma *partial-order-on-carrier* [*HOL-Algebra-order-alignment*]:
partial-order-on (*carrier* L) ($\sqsubseteq$)
using *preorder-on-carrier antisymmetric-on-carrier* **by** *blast*

end

end

1.9.2 Alignment With 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 $L \equiv (\sqsubseteq_{\mathcal{X}})\restriction_{carrier\ \mathcal{X}}\upharpoonleft_{carrier\ \mathcal{X}}$ **and** $R \equiv (\sqsubseteq_{\mathcal{Y}})\restriction_{carrier\ \mathcal{Y}}\upharpoonleft_{carrier\ \mathcal{Y}}$
and $l \equiv \pi^*$ **and** $r \equiv \pi_*$
notes *defs*[*simp*] = *L-def R-def l-def r-def* **and** *restrict-right-eq*[*simp*]
and *restrict-leftI*[*intro*!] *restrict-leftE*[*elim*!]
begin

interpretation *galois* $L\ R\ l\ r$ **.**

lemma *mono-wrt-rel-lower* [*HOL-Algebra-galois-alignment*]: $(L \Rrightarrow_m R)\ l$
using *lower-closed upper-closed* **by** (*fastforce intro*: *use-iso2*[*OF lower-iso*])

lemma *mono-wrt-rel-upper* [*HOL-Algebra-galois-alignment*]: $(R \Rrightarrow_m L)$ r
  using *lower-closed upper-closed* **by** (*fastforce intro*: *use-iso2*[*OF upper-iso*])

lemma *half-galois-prop-left* [*HOL-Algebra-galois-alignment*]: $(L\ {}_h{\trianglelefteq}\ R)$ l r
  using *galois-property lower-closed* **by** *fastforce*

lemma *half-galois-prop-right* [*HOL-Algebra-galois-alignment*]: $(L \trianglelefteq_h R)$ l r
  using *galois-property upper-closed* **by** *fastforce*

lemma *galois-prop* [*HOL-Algebra-galois-alignment*]: $(L \trianglelefteq R)$ l r
  using *half-galois-prop-left half-galois-prop-right* **by** *blast*

lemma *galois-connection* [*HOL-Algebra-galois-alignment*]: $(L \dashv R)$ l r
  using *mono-wrt-rel-lower mono-wrt-rel-upper galois-prop* **by** *blast*

end
end

context *galois-bijection*
begin

context
  fixes L R l r
  defines $L \equiv (\sqsubseteq_{\mathcal{X}})\restriction_{carrier}\ \mathcal{X}\upharpoonleft_{carrier}\ \mathcal{X}$ **and** $R \equiv (\sqsubseteq_{\mathcal{Y}})\restriction_{carrier}\ \mathcal{Y}\upharpoonleft_{carrier}\ \mathcal{Y}$
    and $l \equiv \pi^*$ **and** $r \equiv \pi_*$
  notes *defs*[*simp*] = *L-def R-def l-def r-def* **and** *restrict-right-eq*[*simp*]
    and *restrict-leftI*[*intro*!] *restrict-leftE*[*elim*!] *in-codom-restrict-leftE*[*elim*!]
begin

interpretation *galois* R L r l **.**

lemma *half-galois-prop-left-right-left* [*HOL-Algebra-galois-alignment*]:
  $(R\ {}_h{\trianglelefteq}\ L)$ r l
  using *gal-bij-conn.right lower-inv-eq upper-closed upper-inv-eq*
  by (*intro half-galois-prop-leftI*; *elim left-GaloisE*) (*auto*; *metis*)

lemma *half-galois-prop-right-right-left* [*HOL-Algebra-galois-alignment*]:
  $(R \trianglelefteq_h L)$ r l
  using *gal-bij-conn.left lower-closed lower-inv-eq upper-inv-eq*
  by (*intro half-galois-prop-rightI*; *elim Galois-rightE*) (*auto*; *metis*)

lemma *prop-right-right-left* [*HOL-Algebra-galois-alignment*]: $(R \trianglelefteq L)$ r l
  using *half-galois-prop-left-right-left half-galois-prop-right-right-left* **by** *blast*

lemma *galois-equivalence* [*HOL-Algebra-galois-alignment*]: $(L \equiv_G R)$ l r
  using *gal-bij-conn.galois-connection prop-right-right-left*
  by (*intro galois.galois-equivalenceI*) *auto*

end

end

end

1.10 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

1.11 HOL Syntax Bundles

1.11.1 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
bundle *no-HOL-ascii-syntax*
begin
no-notation (*ASCII*)
  Not (~ - [*40*] *40*) **and**
  conj (**infixr** & *35*) **and**
  disj (**infixr** | *30*) **and**
  implies (**infixr** −−> *25*) **and**
  not-equal (**infixl** ~= *50*)
no-syntax *-Let* :: [*letbinds*, *'a*] ⇒ *'a* ((*let* (-)/ *in* (-)) *10*)
end

end

1.11.2 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* (− - [*81*] *80*)
notation *Groups.times* (**infixl** ∗ *70*)
notation *abs* (|-|)
end
bundle *no-HOL-groups-syntax*
begin
no-notation *Groups.zero* (*0*)
no-notation *Groups.one* (*1*)
no-notation *Groups.plus* (**infixl** + *65*)
no-notation *Groups.minus* (**infixl** − *65*)
no-notation *Groups.uminus* (− - [*81*] *80*)
no-notation *Groups.times* (**infixl** ∗ *70*)
no-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
    HOL-Syntax-Bundles-Relations
begin

Summary Bundles to enable and disable syntax from HOL.

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* ≡
  $((\le_L) \dashv (\le_R))\ l\ r$
  $\wedge$ *preorder-on* (*in-field* $(\le_L)$) $(\le_L)$
  $\wedge$ *preorder-on* (*in-field* $(\le_R)$) $(\le_R)$

notation *transport.preorder-galois-connection* (**infix** $\dashv_{pre}$ *50*)

lemma *preorder-galois-connectionI* [*intro*]:
  assumes $((\le_L) \dashv (\le_R))\ l\ r$
  and *preorder-on* (*in-field* $(\le_L)$) $(\le_L)$
  and *preorder-on* (*in-field* $(\le_R)$) $(\le_R)$
  shows $((\le_L) \dashv_{pre} (\le_R))\ l\ r$
  unfolding *preorder-galois-connection-def* **using** *assms* **by** *blast*

lemma *preorder-galois-connectionE* [*elim*]:

assumes $((\le_L) \dashv_{pre} (\le_R))\ l\ r$
obtains $((\le_L) \dashv (\le_R))\ l\ r$ *preorder-on* (*in-field* $(\le_L)$) $(\le_L)$
preorder-on (*in-field* $(\le_R)$) $(\le_R)$
using *assms* **unfolding** *preorder-galois-connection-def* **by** *blast*

context
begin

interpretation t : *transport* $S\ T\ f\ g$ **for** $S\ T\ f\ g$ **.**

lemma *rel-inv-preorder-galois-connection-eq-preorder-galois-connection-rel-inv* [*simp*]:
$((\le_R) \dashv_{pre} (\le_L))^{-1} = ((\ge_L) \dashv_{pre} (\ge_R))$
by (*intro ext*) (*auto intro*!: *t.preorder-galois-connectionI*)

end

corollary *preorder-galois-connection-rel-inv-iff-preorder-galois-connection* [*iff*]:
$((\ge_L) \dashv_{pre} (\ge_R))\ l\ r \longleftrightarrow ((\le_R) \dashv_{pre} (\le_L))\ r\ l$
by (*simp flip*:
rel-inv-preorder-galois-connection-eq-preorder-galois-connection-rel-inv)

definition *partial-equivalence-rel-galois-connection* $\equiv$
$((\le_L) \dashv (\le_R))\ l\ r$
$\wedge$ *partial-equivalence-rel* $(\le_L)$
$\wedge$ *partial-equivalence-rel* $(\le_R)$

notation *transport.partial-equivalence-rel-galois-connection* (**infix** $\dashv_{PER}$ *50*)

lemma *partial-equivalence-rel-galois-connectionI* [*intro*]:
assumes $((\le_L) \dashv (\le_R))\ l\ r$
and *partial-equivalence-rel-on* (*in-field* $(\le_L)$) $(\le_L)$
and *partial-equivalence-rel-on* (*in-field* $(\le_R)$) $(\le_R)$
shows $((\le_L) \dashv_{PER} (\le_R))\ l\ r$
unfolding *partial-equivalence-rel-galois-connection-def* **using** *assms* **by** *blast*

lemma *partial-equivalence-rel-galois-connectionE* [*elim*]:
assumes $((\le_L) \dashv_{PER} (\le_R))\ l\ r$
obtains $((\le_L) \dashv_{pre} (\le_R))\ l\ r$ *symmetric* $(\le_L)$ *symmetric* $(\le_R)$
using *assms* **unfolding** *partial-equivalence-rel-galois-connection-def* **by** *blast*

context
begin

interpretation t : *transport* $S\ T\ f\ g$ **for** $S\ T\ f\ g$ **.**

lemma *rel-inv-partial-equivalence-rel-galois-connection-eq-partial-equivalence-rel-galois-connection-rel-inv*
[*simp*]: $((\le_R) \dashv_{PER} (\le_L))^{-1} = ((\ge_L) \dashv_{PER} (\ge_R))$
by (*intro ext*) *blast*

end

corollary *partial-equivalence-rel-galois-connection-rel-inv-iff-partial-equivalence-rel-galois-connection*
[*iff*]: $((\geq_L)\ \dashv_{PER}\ (\geq_R))\ l\ r \longleftrightarrow ((\leq_R)\ \dashv_{PER}\ (\leq_L))\ r\ l$
by (*simp flip*:
rel-inv-partial-equivalence-rel-galois-connection-eq-partial-equivalence-rel-galois-connection-rel-inv)

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 $((_L\lessapprox) \circ\circ (\gtrapprox_L)) = (\leq_L)$
proof (*intro ext iffI*)
fix $x\ x'$ **assume** $((_L\lessapprox) \circ\circ (\gtrapprox_L))\ x\ x'$
then obtain y **where** $x \leq_L r\ y\ r\ y \geq_L x'$ **by** *blast*
with *assms* **show** $x \leq_L x'$ **by** (*blast dest*: *symmetricD*)
next
fix $x\ x'$ **assume** $x \leq_L x'$
with *assms* **have** $x\ {}_L\lessapprox\ l\ x'\ x'\ {}_L\lessapprox\ l\ x'$
by (*blast intro*: *left-Galois-left-if-left-relI*)+
with *assms* **show** $((_L\lessapprox) \circ\circ (\gtrapprox_L))\ x\ x'$ **by** *auto*
qed

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** $\equiv_{pre}$ *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 $((\leq_L) \equiv_{pre} (\leq_R))\ l\ r$
unfolding *preorder-equivalence-def* **using** *assms* **by** *blast*

lemma *preorder-equivalence-if-order-equivalenceI*:
assumes $((\leq_L) \equiv_o (\leq_R))\ l\ r$
and *transitive* $(\leq_L)$
and *transitive* $(\leq_R)$
shows $((\leq_L) \equiv_{pre} (\leq_R))\ l\ r$
unfolding *preorder-equivalence-def* **using** *assms*
by (*blast intro*: *reflexive-on-in-field-if-transitive-if-rel-equivalence-on*
dest: *galois-equivalence-left-right-if-transitive-if-order-equivalence*)

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* $(\leq_R)$) $(\leq_R)$
using *assms* **unfolding** *preorder-equivalence-def* **by** *blast*

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* $(\leq_R)$) $(\leq_R)$
using *assms* **by** (*blast intro*:
order-equivalence-if-reflexive-on-in-field-if-galois-equivalence)

context
begin

interpretation t : *transport* S T f g **for** S T f g **.**

lemma *rel-inv-preorder-equivalence-eq-preorder-equivalence* [*simp*]:
$((\leq_R) \equiv_{pre} (\leq_L))^{-1} = ((\leq_L) \equiv_{pre} (\leq_R))$
by (*intro ext*) *blast*

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
by (*simp flip*: *rel-inv-preorder-equivalence-eq-preorder-equivalence*)

lemma *preorder-equivalence-rel-inv-eq-preorder-equivalence* [*simp*]:
$((\geq_L) \equiv_{pre} (\geq_R)) = ((\leq_L) \equiv_{pre} (\leq_R))$
by (*intro ext iffI*)
(*auto intro*!: *transport.preorder-equivalence-if-galois-equivalenceI*
elim!: *transport.preorder-equivalence-galois-equivalenceE*)

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** $\equiv_{PER}$ *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
unfolding *partial-equivalence-rel-equivalence-def* **using** *assms* **by** *blast*

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*
unfolding *partial-equivalence-rel-equivalence-def* **using** *assms*
by (*blast dest*: *galois-equivalence-left-right-if-transitive-if-order-equivalence*)

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* $(\leq_R)$
using *assms* **unfolding** *partial-equivalence-rel-equivalence-def* **by** *blast*

context
begin

interpretation *t* : *transport S T f g* **for** *S T f g* **.**

lemma *rel-inv-partial-equivalence-rel-equivalence-eq-partial-equivalence-rel-equivalence*
[*simp*]:
$((\leq_R) \equiv_{PER} (\leq_L))^{-1} = ((\leq_L) \equiv_{PER} (\leq_R))$
by (*intro ext*) *blast*

end

corollary *partial-equivalence-rel-equivalence-right-left-iff-partial-equivalence-rel-equivalence-left-right*:
$((\leq_R) \equiv_{PER} (\leq_L))$ *r l* $\longleftrightarrow$ $((\leq_L) \equiv_{PER} (\leq_R))$ *l r*
by (*simp flip*:
rel-inv-partial-equivalence-rel-equivalence-eq-partial-equivalence-rel-equivalence)

lemma *partial-equivalence-rel-equivalence-rel-inv-eq-partial-equivalence-rel-equivalence*
[*simp*]: $((\geq_L) \equiv_{PER} (\geq_R)) = ((\leq_L) \equiv_{PER} (\leq_R))$
by (*intro ext iffI*)
(*auto intro*!: *transport.partial-equivalence-rel-equivalence-if-galois-equivalenceI*
elim!: *transport.partial-equivalence-rel-equivalenceE*
transport.preorder-equivalence-galois-equivalenceE
preorder-on-in-fieldE)

end

end

2.2 Transport using Bijections

theory *Transport-Bijections*
imports
Functions-Bijection
Transport-Base
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*: $(L \Rrightarrow_m R)\ l$
  and *mono-wrt-rel-right*: $(R \Rrightarrow_m L)\ r$
  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* $L\ R\ l\ r$ **.**
interpretation *g-flip-inv* : *galois* $(\geq_R)\ (\geq_L)\ r\ l$ **.**

lemma *bijection-on-in-field*: *bijection-on* (*in-field* $(\leq_L)$) (*in-field* $(\leq_R)$) $l\ r$
  using *mono-wrt-rel-left mono-wrt-rel-right inverse-left-right inverse-right-left*
  by (*intro bijection-onI in-field-if-in-field-if-mono-wrt-rel*)
  auto

lemma *half-galois-prop-left*: $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$
  using *mono-wrt-rel-left inverse-right-left*
  by (*intro half-galois-prop-leftI*)
  (*auto dest*!: *in-field-if-in-codom inverse-onD*)

lemma *half-galois-prop-right*: $((\leq_L)\ \trianglelefteq_h\ (\leq_R))\ l\ r$
  using *mono-wrt-rel-right inverse-left-right*
  by (*intro half-galois-prop-rightI*)
  (*force dest*: *in-field-if-in-dom inverse-onD*)

lemma *galois-prop*: $((\leq_L) \trianglelefteq (\leq_R))\ l\ r$
  using *half-galois-prop-left half-galois-prop-right*
  by (*intro galois-propI*)

lemma *galois-connection*: $((\leq_L) \dashv (\leq_R))\ l\ r$
  using *mono-wrt-rel-left mono-wrt-rel-right galois-prop*
  by (*intro galois-connectionI*)

lemma *rel-equivalence-on-unitI*:
  assumes *reflexive-on* (*in-field* $(\leq_L)$) $(\leq_L)$
  shows *rel-equivalence-on* (*in-field* $(\leq_L)$) $(\leq_L)\ \eta$
  using *assms inverse-left-right*
  by (*subst rel-equivalence-on-unit-iff-reflexive-on-if-inverse-on*)

interpretation *flip* : *transport-bijection* $R\ L\ r\ l$
  rewrites *order-functors.unit* $r\ l \equiv \varepsilon$
  using *mono-wrt-rel-left mono-wrt-rel-right inverse-left-right inverse-right-left*
  by *unfold-locales* (*simp-all only*: *flip-unit-eq-counit*)

lemma *galois-equivalence*: $((\leq_L) \equiv_G (\leq_R))\ l\ r$

using *galois-connection flip.galois-prop* **by** (*intro galois-equivalenceI*)

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*
  using *assms mono-wrt-rel-left mono-wrt-rel-right rel-equivalence-on-unitI*
    rel-equivalence-on-counitI
  by (*intro order-equivalenceI*)

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*
  using *assms* **by** (*intro preorder-equivalence-if-galois-equivalenceI*
    galois-equivalence)
  simp-all

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*
  using *assms* **by** (*intro partial-equivalence-rel-equivalence-if-galois-equivalenceI*
    galois-equivalence)
  simp-all

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*) $L \equiv$ *True*
  and *reflexive-on* (*in-field R*) $R \equiv$ *True*
  and $\bigwedge P.\ (True \Longrightarrow P) \equiv Trueprop\ P$
  using *transport-bijection reflexive-on-in-field-left reflexive-on-in-field-right*
  by *auto*

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 :: $'a \Rightarrow 'a \Rightarrow$ *bool*
  and R :: $'b \Rightarrow 'b \Rightarrow$ *bool*
  and l :: $'a \Rightarrow 'b$
  and r :: $'b \Rightarrow 'a$
  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) $L \equiv$ *True*
  and *preorder-on* (*in-field* R) $R \equiv$ *True*
  and $\bigwedge P.$ (*True* $\Longrightarrow P) \equiv$ *Trueprop* P
  using *transport-bijection*
  by (*intro transport-reflexive-on-in-field-bijection.intro*)
  (*insert preorder-on-in-field-left preorder-on-in-field-right*, *auto*)

lemmas *preorder-equivalence* = *preorder-equivalenceI*

end

locale *transport-partial-equivalence-rel-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 *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* $L \equiv$ *True*
  and *partial-equivalence-rel* $R \equiv$ *True*
  and $\bigwedge P.$ (*True* $\Longrightarrow P) \equiv$ *Trueprop* P
  using *transport-bijection*
  by (*intro transport-preorder-on-in-field-bijection.intro*)
  (*insert partial-equivalence-rel-left partial-equivalence-rel-right*, *auto*)

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 **.**

sublocale *tper-bij?* : *transport-partial-equivalence-rel-bijection* $(=_P)$ $(=_Q)$ l r
using *bijection-on-in-field partial-equivalence-rel-eq-restrict*
eq-restrict-le-eq
by *unfold-locales*
(*auto elim*: *bijection-onE intro*!:
mono-wrt-rel-left-if-reflexive-on-if-le-eq-if-mono-wrt-in-field
[*of in-field* $(=_Q)$]
flip-of.mono-wrt-rel-left-if-reflexive-on-if-le-eq-if-mono-wrt-in-field
[*of in-field* $(=_P)$])

lemma *left-Galois-eq-Galois-eq-eq-restrict*: $({}_L\lessapprox) = ($*galois-rel.Galois* $(=)$ $(=)$ $r)\restriction_P\upharpoonleft_Q$
by (*subst galois-rel.left-Galois-restrict-left-eq-left-Galois-left-restrict-left*
galois-rel.left-Galois-restrict-right-eq-left-Galois-right-restrict-right
restrict-right-eq rel-inv-eq-self-if-symmetric)+
(*auto simp*: *eq-restrict-eq-eq-restrict-left*)

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 $(=_{\top\ ::\ 'b \Rightarrow bool}) = ((=)$ :: $'b \Rightarrow$ -)
using *bijection-on-in-field* **by** *unfold-locales simp-all*

end

end

2.3 Compositions With Agreeing Relations

2.3.1 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* :: $'a \Rightarrow 'a \Rightarrow bool$
  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 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** $\le_{L1}$ *50*)
notation *R1* (**infix** $\le_{R1}$ *50*)
notation *L2* (**infix** $\le_{L2}$ *50*)
notation *R2* (**infix** $\le_{R2}$ *50*)

notation *g1.ge-left* (**infix** $\ge_{L1}$ *50*)
notation *g1.ge-right* (**infix** $\ge_{R1}$ *50*)
notation *g2.ge-left* (**infix** $\ge_{L2}$ *50*)
notation *g2.ge-right* (**infix** $\ge_{R2}$ *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*: $l = l2 \circ l1$
  unfolding *l-def* **..**

lemma *left-eq* [*simp*]: $l\ x = l2\ (l1\ x)$
  unfolding *left-eq-comp* **by** *simp*

context
begin

interpretation *flip* : *transport-comp-agree R2 L2 r2 l2 R1 L1 r1 l1* **.**

abbreviation (*input*) $R \equiv$ *flip.L*
abbreviation $r \equiv$ *flip.l*

lemma *right-eq-comp*: $r = r1 \circ r2$
  unfolding *flip.l-def* **..**

lemma *right-eq* [*simp*]: $r\ z = r1\ (r2\ z)$
  unfolding *right-eq-comp* **by** *simp*

lemmas *transport-defs* = *left-eq-comp right-eq-comp*

end

sublocale *transport L R l r* **.**

notation *L* (**infix** $\le_L$ *50*)
notation *R* (**infix** $\le_R$ *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

2.3.2 Monotonicity

theory *Transport-Compositions-Agree-Monotone*
  imports
    Transport-Compositions-Agree-Base
begin

context *transport-comp-agree*
begin

lemma *mono-wrt-rel-leftI*:
  assumes $((\leq_{L1}) \Rrightarrow_m (\leq_{R1}))$ *l1* $((\leq_{L2}) \Rrightarrow_m (\leq_{R2}))$ *l2*
  and $\bigwedge x\ y.\ x \leq_{L1} y \Longrightarrow l1\ x \leq_{R1} l1\ y \Longrightarrow l1\ x \leq_{L2} l1\ y$
  shows $((\leq_L) \Rrightarrow_m (\leq_R))$ *l*
  unfolding *left-eq-comp* **using** *assms* **by** (*rule dep-mono-wrt-rel-compI*)

end

context *transport-comp-same*
begin

lemma *mono-wrt-rel-leftI*:
  assumes $((\leq_{L1}) \Rrightarrow_m (\leq_{R1}))$ *l1* $((\leq_{R1}) \Rrightarrow_m (\leq_{R2}))$ *l2*
  shows $((\leq_L) \Rrightarrow_m (\leq_R))$ *l*
  using *assms* **by** (*rule mono-wrt-rel-leftI*) *auto*

end

end

2.3.3 Galois Property

theory *Transport-Compositions-Agree-Galois-Property*
 imports
  Transport-Compositions-Agree-Base
begin

context *transport-comp-agree*
begin

lemma *galois-propI*:
 assumes *galois1*: $((\le_{L1}) \trianglelefteq (\le_{R1}))$ *l1 r1*
 and *galois2*: $((\le_{L2}) \trianglelefteq (\le_{R2}))$ *l2 r2*
 and *mono-l1*: $([\textit{in-dom } (\le_{L1})] \Rrightarrow_m \textit{in-dom } (\le_{L2}))\ l1$
 and *mono-r2*: $([\textit{in-codom } (\le_{R2})] \Rrightarrow_m \textit{in-codom } (\le_{R1}))\ r2$
 and *agree*: $([\textit{in-dom } (\le_{L1})] \Rrightarrow [\textit{in-codom } (\le_{R2})] \Rrightarrow (\longleftrightarrow))$
  $(\textit{rel-bimap } l1\ r2\ (\le_{R1}))\ (\textit{rel-bimap } l1\ r2\ (\le_{L2}))$
 shows $((\le_L) \trianglelefteq (\le_R))\ l\ r$
proof (*rule galois-prop.galois-propI'*)
 fix *x y* **assume** *in-dom* $(\le_L)$ *x in-codom* $(\le_R)$ *y*
 with *mono-r2 mono-l1* **have** *in-dom* $(\le_{L2})$ (*l1 x*) *in-codom* $(\le_{R1})$ (*r2 y*) **by** *auto*
 have $x \le_L r\ y \longleftrightarrow x \le_{L1} r1\ (r2\ y)$ **by** *simp*
 also from *galois1* ‹*in-dom* $(\le_{L1})$ *x*› ‹*in-codom* $(\le_{R1})$ (*r2 y*)›
  have $\ldots \longleftrightarrow l1\ x \le_{R1} r2\ y$
  by (*rule g1.galois-prop-left-rel-right-iff-left-right-rel*)
 also from *agree* ‹*in-dom* $(\le_{L1})$ *x*› ‹*in-codom* $(\le_{R2})$ *y*›
  have $\ldots \longleftrightarrow l1\ x \le_{L2} r2\ y$ **by** *fastforce*
 also from *galois2* ‹*in-dom* $(\le_{L2})$ (*l1 x*)› ‹*in-codom* $(\le_{R2})$ *y*›
  have $\ldots \longleftrightarrow l\ x \le_{R2} y$
  unfolding *l-def*
  by (*simp add*: *g2.galois-prop-left-rel-right-iff-left-right-rel*)
 finally show $x \le_L r\ y \longleftrightarrow l\ x \le_R y$ **.**
qed

end

context *transport-comp-same*
begin

corollary *galois-propI*:
 assumes $((\le_{L1}) \trianglelefteq (\le_{R1}))$ *l1 r1*
 and $((\le_{R1}) \trianglelefteq (\le_{R2}))$ *l2 r2*
 and $([\textit{in-dom } (\le_{L1})] \Rrightarrow_m \textit{in-dom } (\le_{R1}))\ l1$
 and $([\textit{in-codom } (\le_{R2})] \Rrightarrow_m \textit{in-codom } (\le_{R1}))\ r2$
 shows $((\le_L) \trianglelefteq (\le_R))\ l\ r$
 using *assms* **by** (*rule galois-propI*) *auto*

end

end

2.3.4 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* **.**

lemma *galois-connectionI*:
  assumes *galois*: $((\le_{L1}) \dashv (\le_{R1}))\ l1\ r1\ ((\le_{L2}) \dashv (\le_{R2}))\ l2\ r2$
  and *mono-L1-L2-l1*: $\bigwedge x\ y.\ x \le_{L1} y \Longrightarrow l1\ x \le_{R1} l1\ y \Longrightarrow l1\ x \le_{L2} l1\ y$
  and *mono-R2-R1-r2*: $\bigwedge x\ y.\ x \le_{R2} y \Longrightarrow r2\ x \le_{L2} r2\ y \Longrightarrow r2\ x \le_{R1} r2\ y$
  and $([in\text{-}dom\ (\le_{L1})] \Rrightarrow [in\text{-}codom\ (\le_{R2})] \Rrightarrow (\longleftrightarrow))$
    $(rel\text{-}bimap\ l1\ r2\ (\le_{R1}))\ (rel\text{-}bimap\ l1\ r2\ (\le_{L2}))$
  shows $((\le_L) \dashv (\le_R))\ l\ r$
proof −
  from *galois mono-L1-L2-l1* **have** $([in\text{-}dom\ (\le_{L1})] \Rrightarrow_m in\text{-}dom\ (\le_{L2}))\ l1$
    by (*intro dep-mono-wrt-predI*) (*blast elim*!: *in-domE g1.galois-connectionE*)
  moreover from *galois mono-R2-R1-r2*
    have $([in\text{-}codom\ (\le_{R2})] \Rrightarrow_m in\text{-}codom\ (\le_{R1}))\ r2$
    by (*intro dep-mono-wrt-predI*) (*blast elim*!: *in-codomE g2.galois-connectionE*)
  ultimately show *?thesis* **using** *assms*
    by (*intro galois-connectionI galois-propI mono-wrt-rel-leftI*
      flip.mono-wrt-rel-leftI)
    auto
qed

lemma *galois-connectionI'*:
  assumes $((\le_{L1}) \dashv (\le_{R1}))\ l1\ r1\ ((\le_{L2}) \dashv (\le_{R2}))\ l2\ r2$
  and $((\le_{L1}) \Rrightarrow_m (\le_{L2}))\ l1\ ((\le_{R2}) \Rrightarrow_m (\le_{R1}))\ r2$
  and $([in\text{-}dom\ (\le_{L1})] \Rrightarrow [in\text{-}codom\ (\le_{R2})] \Rrightarrow (\longleftrightarrow))$
    $(rel\text{-}bimap\ l1\ r2\ (\le_{R1}))\ (rel\text{-}bimap\ l1\ r2\ (\le_{L2}))$
  shows $((\le_L) \dashv (\le_R))\ l\ r$
  using *assms* **by** (*intro galois-connectionI*) *auto*

end

context *transport-comp-same*

begin

corollary *galois-connectionI*:
  assumes $((\le_{L1}) \dashv (\le_{R1}))$ *l1 r1* $((\le_{R1}) \dashv (\le_{R2}))$ *l2 r2*
  shows $((\le_{L}) \dashv (\le_{R}))$ *l r*
  using *assms* **by** (*rule galois-connectionI*) *auto*

end

end

2.3.5 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* **.**

lemma *galois-equivalenceI*:
  assumes *galois*: $((\le_{L1}) \equiv_G (\le_{R1}))$ *l1 r1* $((\le_{L2}) \equiv_G (\le_{R2}))$ *l2 r2*
  and *mono-L1-L2-l1*: $\bigwedge x\ y.\ x \le_{L1} y \Longrightarrow l1\ x \le_{R1} l1\ y \Longrightarrow l1\ x \le_{L2} l1\ y$
  and *mono-R2-R1-r2*: $\bigwedge x\ y.\ x \le_{R2} y \Longrightarrow r2\ x \le_{L2} r2\ y \Longrightarrow r2\ x \le_{R1} r2\ y$
  and ([*in-dom* $(\le_{L1})$] $\Rrightarrow$ [*in-codom* $(\le_{R2})$] $\Rrightarrow$ $(\longleftrightarrow)$)
    (*rel-bimap l1 r2* $(\le_{R1})$) (*rel-bimap l1 r2* $(\le_{L2})$)
  and *mono-iff2*: ([*in-dom* $(\le_{R2})$] $\Rrightarrow$ [*in-codom* $(\le_{L1})$] $\Rrightarrow$ $(\longleftrightarrow)$)
    (*rel-bimap r2 l1* $(\le_{R1})$) (*rel-bimap r2 l1* $(\le_{L2})$)
  shows $((\le_{L}) \equiv_G (\le_{R}))$ *l r*
proof −
  from *galois mono-L1-L2-l1* **have** ([*in-codom* $(\le_{L1})$] $\Rrightarrow_m$ *in-codom* $(\le_{L2})$) *l1*
    by (*intro dep-mono-wrt-predI*) *blast*
  moreover from *galois mono-R2-R1-r2* **have** ([*in-dom* $(\le_{R2})$] $\Rrightarrow_m$ *in-dom* $(\le_{R1})$)
r2
    by (*intro dep-mono-wrt-predI*) *blast*
  moreover from *mono-iff2* **have** ([*in-dom* $(\le_{R2})$] $\Rrightarrow$ [*in-codom* $(\le_{L1})$] $\Rrightarrow$ $(\longleftrightarrow)$)
    (*rel-bimap r2 l1* $(\le_{L2})$) (*rel-bimap r2 l1* $(\le_{R1})$) **by** *blast*
  ultimately show *?thesis* **using** *assms*
    by (*intro galois-equivalenceI galois-connectionI flip.galois-propI*) *auto*
qed

lemma *galois-equivalenceI'*:
  assumes $((\le_{L1}) \equiv_G (\le_{R1}))$ *l1 r1* $((\le_{L2}) \equiv_G (\le_{R2}))$ *l2 r2*
  and $((\le_{L1}) \Rrightarrow_m (\le_{L2}))$ *l1* $((\le_{R2}) \Rrightarrow_m (\le_{R1}))$ *r2*
  and ([*in-dom* $(\le_{L1})$] $\Rrightarrow$ [*in-codom* $(\le_{R2})$] $\Rrightarrow$ $(\longleftrightarrow)$)

$(rel\text{-}bimap\ l1\ r2\ (\le_{R1}))\ (rel\text{-}bimap\ l1\ r2\ (\le_{L2}))$
and $([in\text{-}dom\ (\le_{R2})] \Rrightarrow [in\text{-}codom\ (\le_{L1})] \Rrightarrow (\longleftrightarrow))$
$(rel\text{-}bimap\ r2\ l1\ (\le_{R1}))\ (rel\text{-}bimap\ r2\ l1\ (\le_{L2}))$
shows $((\le_{L}) \equiv_{G} (\le_{R}))\ l\ r$
using *assms* **by** (*intro galois-equivalenceI*) *auto*

end

context *transport-comp-same*
begin

lemma *galois-equivalenceI*:
assumes $((\le_{L1}) \equiv_{G} (\le_{R1}))\ l1\ r1\ ((\le_{R1}) \equiv_{G} (\le_{R2}))\ l2\ r2$
shows $((\le_{L}) \equiv_{G} (\le_{R}))\ l\ r$
using *assms* **by** (*rule galois-equivalenceI*) *auto*

end

end

2.3.6 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*: $([in\text{-}codom\ (\le_{R2})] \Rrightarrow_{m} in\text{-}codom\ (\le_{R1}))\ r2$
and *r2-L2-self-if-in-codom*: $\bigwedge z.\ in\text{-}codom\ (\le_{R2})\ z \Longrightarrow r2\ z \le_{L2} r2\ z$
shows $({}_{L}\lessapprox) \le (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$
proof (*rule le-relI*)
fix $x\ z$ **assume** $x\ {}_{L}\lessapprox\ z$
then have $x \le_{L1} r\ z\ in\text{-}codom\ (\le_{R})\ z$ **by** *auto*
with ‹$x \le_{L1} r\ z$› *in-codom-mono-r2* **have** $x\ {}_{L1}\lessapprox\ r2\ z$ **by** *auto*
moreover from ‹$in\text{-}codom\ (\le_{R2})\ z$› *r2-L2-self-if-in-codom* **have** $r2\ z\ {}_{L2}\lessapprox\ z$
by (*intro g2.left-GaloisI*) *auto*
ultimately show $(({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))\ x\ z$ **by** *blast*
qed

lemma *comp-left-Galois-le-left-GaloisI*:
assumes *mono-r1*: $((\le_{R1}) \Rrightarrow_{m} (\le_{L1}))\ r1$
and *trans-L1*: *transitive* $(\le_{L1})$
and *R1-r2-if-in-codom*: $\bigwedge y\ z.\ in\text{-}codom\ (\le_{R2})\ z \Longrightarrow y \le_{L2} r2\ z \Longrightarrow y \le_{R1} r2\ z$
shows $(({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox)) \le ({}_{L}\lessapprox)$

proof (*rule le-relI*)
  fix x z **assume** $(({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))\ x\ z$
  then obtain y **where** $x\ {}_{L1}\lessapprox\ y$ $y\ {}_{L2}\lessapprox\ z$ **by** *blast*
  then have $x \le_{L1} r1\ y$ $y \le_{L2} r2\ z$ *in-codom* $(\le_R)\ z$ **by** *auto*
  with *R1-r2-if-in-codom* **have** $y \le_{R1} r2\ z$ **by** *blast*
  with *mono-r1* **have** $r1\ y \le_{L1} r\ z$ **by** *auto*
  with ‹$x \le_{L1} r1\ y$› ‹*in-codom* $(\le_R)\ z$› **show** $x\ {}_{L}\lessapprox\ z$ **using** *trans-L1* **by** *blast*
qed

corollary *left-Galois-eq-comp-left-GaloisI*:
  assumes $((\le_{R1}) \Rrightarrow_m (\le_{L1}))\ r1$
  and *transitive* $(\le_{L1})$
  and $\bigwedge z.$ *in-codom* $(\le_{R2})\ z \Longrightarrow r2\ z \le_{L2} r2\ z$
  and $\bigwedge y\ z.$ *in-codom* $(\le_{R2})\ z \Longrightarrow y \le_{L2} r2\ z \Longrightarrow y \le_{R1} r2\ z$
  shows $({}_{L}\lessapprox) = (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$
  using *assms*
  by (*intro antisym left-Galois-le-comp-left-GaloisI comp-left-Galois-le-left-GaloisI*
    dep-mono-wrt-predI)
  fastforce

corollary *left-Galois-eq-comp-left-GaloisI'*:
  assumes $((\le_{R1}) \Rrightarrow_m (\le_{L1}))\ r1$
  and *transitive* $(\le_{L1})$
  and $((\le_{R2}) \Rrightarrow_m (\le_{L2}))\ r2$
  and *reflexive-on* (*in-codom* $(\le_{R2})$) $(\le_{R2})$
  and $\bigwedge y\ z.$ *in-codom* $(\le_{R2})\ z \Longrightarrow y \le_{L2} r2\ z \Longrightarrow y \le_{R1} r2\ z$
  shows $({}_{L}\lessapprox) = (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$
  using *assms* **by** (*intro left-Galois-eq-comp-left-GaloisI*) *auto*

corollary *left-Galois-eq-comp-left-GaloisI''*:
  assumes $((\le_{R1}) \Rrightarrow_m (\le_{L1}))\ r1$
  and *transitive* $(\le_{L1})$
  and $((\le_{R2}) \Rrightarrow_m (\le_{L2}))\ r2$
  and *reflexive-on* (*in-codom* $(\le_{L2})$) $(\le_{L2})$
  and $\bigwedge y\ z.$ *in-codom* $(\le_{R2})\ z \Longrightarrow y \le_{L2} r2\ z \Longrightarrow y \le_{R1} r2\ z$
  shows $({}_{L}\lessapprox) = (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$
  using *assms* **by** (*intro left-Galois-eq-comp-left-GaloisI*) (*auto 0 4*)

end

context *transport-comp-same*
begin

lemma *left-Galois-eq-comp-left-GaloisI*:
  assumes $((\le_{R1}) \Rrightarrow_m (\le_{L1}))\ r1$
  and *transitive* $(\le_{L1})$
  and $((\le_{R2}) \Rrightarrow_m (\le_{R1}))\ r2$
  and *reflexive-on* (*in-codom* $(\le_{R2})$) $(\le_{R2})$
  shows $({}_{L}\lessapprox) = (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$

using *assms* **by** (*intro left-Galois-eq-comp-left-GaloisI'*) *auto*

lemma *left-Galois-eq-comp-left-GaloisI'*:
assumes $((\le_{R1}) \Rrightarrow_m (\le_{L1}))$ *r1*
and *transitive* $(\le_{L1})$
and *reflexive-on* (*in-codom* $(\le_{R1})$) $(\le_{R1})$
and $((\le_{R2}) \Rrightarrow_m (\le_{R1}))$ *r2*
shows $({}_{L}\lessapprox) = (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$
using *assms* **by** (*intro left-Galois-eq-comp-left-GaloisI''*) *auto*

end

end

2.3.7 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*: $([P] \Rrightarrow_m P')$ *l1*
and *mono-r1*: $((\le_{R1}) \Rrightarrow_m (\le_{L1}))$ *r1*
and *inflationary-unit1*: *inflationary-on* P $(\le_{L1})$ η_1
and *trans-L1*: *transitive* $(\le_{L1})$
and *inflationary-unit2*: *inflationary-on* P' $(\le_{L2})$ η_2
and *L2-le-R1*: $\bigwedge x.\ P\ x \Longrightarrow l1\ x \le_{L2} r2\ (l\ x) \Longrightarrow l1\ x \le_{R1} r2\ (l\ x)$
shows *inflationary-on* P $(\le_L)$ η
proof (*rule inflationary-onI*)
fix x **assume** $P\ x$
with *mono-l1* **have** $P'\ (l1\ x)$ **by** *blast*
with *inflationary-unit2* **have** $l1\ x \le_{L2} r2\ (l\ x)$ **by** *auto*
with *L2-le-R1* ‹$P\ x$› **have** $l1\ x \le_{R1} r2\ (l\ x)$ **by** *blast*
with *mono-r1* **have** $\eta_1\ x \le_{L1} \eta\ x$ **by** *auto*
moreover from *inflationary-unit1* ‹$P\ x$› **have** $x \le_{L1} \eta_1\ x$ **by** *auto*
ultimately show $x \le_L \eta\ x$ **using** *trans-L1* **by** *blast*
qed

corollary *inflationary-on-in-field-unitI*:
assumes $((\le_{L1}) \Rrightarrow_m (\le_{L2}))$ *l1*
and $((\le_{R1}) \Rrightarrow_m (\le_{L1}))$ *r1*
and *inflationary-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$ η_1
and *transitive* $(\le_{L1})$

and *inflationary-on* (*in-field* $(\le_{L2})$) $(\le_{L2})$ η_2
and $\bigwedge x.$ *in-field* $(\le_{L1})$ $x \Longrightarrow l1\ x \le_{L2} r2\ (l\ x) \Longrightarrow l1\ x \le_{R1} r2\ (l\ x)$
shows *inflationary-on* (*in-field* $(\le_L)$) $(\le_L)$ η
using *assms* **by** (*intro inflationary-on-unitI dep-mono-wrt-predI*) *auto*

Deflationary **context**
begin

interpretation *inv* :
transport-comp-agree $(\ge_{L1})$ $(\ge_{R1})$ *l1 r1* $(\ge_{L2})$ $(\ge_{R2})$ *l2 r2*
rewrites $\bigwedge R\ S.\ (R^{-1} \Rrightarrow_m S^{-1}) \equiv (R \Rrightarrow_m S)$
and $\bigwedge R.$ *inflationary-on* $P\ R^{-1} \equiv$ *deflationary-on* $P\ R$
and $\bigwedge R.$ *transitive* $R^{-1} \equiv$ *transitive* R
and $\bigwedge R.$ *in-field* $R^{-1} \equiv$ *in-field* R
by *simp-all*

lemma *deflationary-on-in-field-unitI*:
assumes $((\le_{L1}) \Rrightarrow_m (\le_{L2}))$ *l1*
and $((\le_{R1}) \Rrightarrow_m (\le_{L1}))$ *r1*
and *deflationary-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$ η_1
and *transitive* $(\le_{L1})$
and *deflationary-on* (*in-field* $(\le_{L2})$) $(\le_{L2})$ η_2
and $\bigwedge x.$ *in-field* $(\le_{L1})$ $x \Longrightarrow r2\ (l\ x) \le_{L2} l1\ x \Longrightarrow r2\ (l\ x) \le_{R1} l1\ x$
shows *deflationary-on* (*in-field* $(\le_L)$) $(\le_L)$ η
using *assms* **by** (*intro inv.inflationary-on-in-field-unitI*[*simplified rel-inv-iff-rel*])
auto

end

Relational Equivalence

corollary *rel-equivalence-on-in-field-unitI*:
assumes $((\le_{L1}) \Rrightarrow_m (\le_{L2}))$ *l1*
and $((\le_{R1}) \Rrightarrow_m (\le_{L1}))$ *r1*
and *rel-equivalence-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$ η_1
and *transitive* $(\le_{L1})$
and *rel-equivalence-on* (*in-field* $(\le_{L2})$) $(\le_{L2})$ η_2
and $\bigwedge x.$ *in-field* $(\le_{L1})$ $x \Longrightarrow l1\ x \le_{L2} r2\ (l\ x) \Longrightarrow l1\ x \le_{R1} r2\ (l\ x)$
and $\bigwedge x.$ *in-field* $(\le_{L1})$ $x \Longrightarrow r2\ (l\ x) \le_{L2} l1\ x \Longrightarrow r2\ (l\ x) \le_{R1} l1\ x$
shows *rel-equivalence-on* (*in-field* $(\le_L)$) $(\le_L)$ η
using *assms* **by** (*intro rel-equivalence-onI*
inflationary-on-in-field-unitI deflationary-on-in-field-unitI)
auto

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* $\equiv \varepsilon$
  by (*simp-all only*: *g1.flip-unit-eq-counit g2.flip-unit-eq-counit flip-unit-eq-counit*)

lemma *order-equivalenceI*:
  assumes $((\le_{L1}) \equiv_o (\le_{R1}))$ *l1 r1*
  and *transitive* $(\le_{L1})$
  and $((\le_{L2}) \equiv_o (\le_{R2}))$ *l2 r2*
  and *transitive* $(\le_{R2})$
  and $\bigwedge x\ y.\ x \le_{L1} y \Longrightarrow l1\ x \le_{R1} l1\ y \Longrightarrow l1\ x \le_{L2} l1\ y$
  and $\bigwedge x\ y.\ x \le_{R2} y \Longrightarrow r2\ x \le_{L2} r2\ y \Longrightarrow r2\ x \le_{R1} r2\ y$
  and $\bigwedge x.$ *in-field* $(\le_{L1})\ x \Longrightarrow l1\ x \le_{L2} r2\ (l\ x) \Longrightarrow l1\ x \le_{R1} r2\ (l\ x)$
  and $\bigwedge x.$ *in-field* $(\le_{L1})\ x \Longrightarrow r2\ (l\ x) \le_{L2} l1\ x \Longrightarrow r2\ (l\ x) \le_{R1} l1\ x$
  and $\bigwedge x.$ *in-field* $(\le_{R2})\ x \Longrightarrow r2\ x \le_{R1} l1\ (r\ x) \Longrightarrow r2\ x \le_{L2} l1\ (r\ x)$
  and $\bigwedge x.$ *in-field* $(\le_{R2})\ x \Longrightarrow l1\ (r\ x) \le_{R1} r2\ x \Longrightarrow l1\ (r\ x) \le_{L2} r2\ x$
  shows $((\le_L) \equiv_o (\le_R))$ *l r*
  using *assms* **by** (*intro order-equivalenceI rel-equivalence-on-in-field-unitI*
    flip.rel-equivalence-on-in-field-unitI
    mono-wrt-rel-leftI flip.mono-wrt-rel-leftI dep-mono-wrt-relI)
  (*auto elim*!: *g1.order-equivalenceE g2.order-equivalenceE*)

end

context *transport-comp-same*
begin

lemma *order-equivalenceI*:
  assumes $((\le_{L1}) \equiv_o (\le_{R1}))$ *l1 r1*
  and *transitive* $(\le_{L1})$
  and $((\le_{R1}) \equiv_o (\le_{R2}))$ *l2 r2*
  and *transitive* $(\le_{R2})$
  shows $((\le_L) \equiv_o (\le_R))$ *l r*
  using *assms* **by** (*rule order-equivalenceI*) *auto*

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

2.4 Generic Compositions

2.4.1 Basic Setup

theory *Transport-Compositions-Generic-Base*
  imports
    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** $\le_{L1}$ *50*)
notation *R1* (**infix** $\le_{R1}$ *50*)
notation *L2* (**infix** $\le_{L2}$ *50*)
notation *R2* (**infix** $\le_{R2}$ *50*)

notation *t1.ge-left* (**infix** $\ge_{L1}$ *50*)
notation *t1.ge-right* (**infix** $\ge_{R1}$ *50*)
notation *t2.ge-left* (**infix** $\ge_{L2}$ *50*)
notation *t2.ge-right* (**infix** $\ge_{R2}$ *50*)

notation *t1.left-Galois* (**infix** ${}_{L1}\lessapprox$ *50*)
notation *t1.right-Galois* (**infix** ${}_{R1}\lessapprox$ *50*)
notation *t2.left-Galois* (**infix** ${}_{L2}\lessapprox$ *50*)

notation *t2.right-Galois* (**infix** $_{R2}\lessapprox$ *50*)

notation *t1.ge-Galois-left* (**infix** $\gtrapprox_{L1}$ *50*)
notation *t1.ge-Galois-right* (**infix** $\gtrapprox_{R1}$ *50*)
notation *t2.ge-Galois-left* (**infix** $\gtrapprox_{L2}$ *50*)
notation *t2.ge-Galois-right* (**infix** $\gtrapprox_{R2}$ *50*)

notation *t1.right-ge-Galois* (**infix** $_{R1}\gtrapprox$ *50*)
notation *t1.Galois-right* (**infix** $\lessapprox_{R1}$ *50*)
notation *t2.right-ge-Galois* (**infix** $_{R2}\gtrapprox$ *50*)
notation *t2.Galois-right* (**infix** $\lessapprox_{R2}$ *50*)

notation *t1.left-ge-Galois* (**infix** $_{L1}\gtrapprox$ *50*)
notation *t1.Galois-left* (**infix** $\lessapprox_{L1}$ *50*)
notation *t2.left-ge-Galois* (**infix** $_{L2}\gtrapprox$ *50*)
notation *t2.Galois-left* (**infix** $\lessapprox_{L2}$ *50*)

notation *t1.unit* (η_1)
notation *t1.counit* (ε_1)
notation *t2.unit* (η_2)
notation *t2.counit* (ε_2)

definition $L \equiv ({}_{L1}\lessapprox) \circ\circ (\leq_{L2}) \circ\circ ({}_{R1}\lessapprox)$

lemma *left-rel-eq-comp*: $L = ({}_{L1}\lessapprox) \circ\circ (\leq_{L2}) \circ\circ ({}_{R1}\lessapprox)$
  unfolding *L-def* **..**

definition $l \equiv l2 \circ l1$

lemma *left-eq-comp*: $l = l2 \circ l1$
  unfolding *l-def* **..**

lemma *left-eq* [*simp*]: $l\ x = l2\ (l1\ x)$
  unfolding *left-eq-comp* **by** *simp*

context
begin

interpretation *flip* : *transport-comp R2 L2 r2 l2 R1 L1 r1 l1* **.**

abbreviation $R \equiv$ *flip.L*
abbreviation $r \equiv$ *flip.l*

lemma *right-rel-eq-comp*: $R = ({}_{R2}\lessapprox) \circ\circ (\leq_{R1}) \circ\circ ({}_{L2}\lessapprox)$
  unfolding *flip.L-def* **..**

lemma *right-eq-comp*: $r = r1 \circ r2$
  unfolding *flip.l-def* **..**

lemma *right-eq* [*simp*]: $r\ z = r1\ (r2\ z)$
  unfolding *right-eq-comp* **by** *simp*

lemmas *transport-defs* = *left-rel-eq-comp left-eq-comp right-rel-eq-comp right-eq-comp*

end

sublocale *transport L R l r* **.**

notation *L* (**infix** $\le_L$ *50*)
notation *R* (**infix** $\le_R$ *50*)

lemma *left-relI* [*intro*]:
  assumes $x\ {}_{L1}\lessapprox\ y$
  and $y \le_{L2} y'$
  and $y'\ {}_{R1}\lessapprox\ x'$
  shows $x \le_L x'$
  unfolding *left-rel-eq-comp* **using** *assms* **by** *blast*

lemma *left-relE* [*elim*]:
  assumes $x \le_L x'$
  obtains $y\ y'$ **where** $x\ {}_{L1}\lessapprox\ y\ \ y \le_{L2} y'\ \ y'\ {}_{R1}\lessapprox\ x'$
  using *assms* **unfolding** *left-rel-eq-comp* **by** *blast*

context
begin

interpretation *flip* : *transport-comp R2 L2 r2 l2 R1 L1 r1 l1* **.**
interpretation *inv* : *transport-comp* $(\ge_{L1})\ (\ge_{R1})$ *l1 r1* $(\ge_{L2})\ (\ge_{R2})$ *l2 r2* **.**

lemma *ge-left-rel-eq-left-rel-inv-if-galois-prop* [*simp*]:
  assumes $((\le_{L1}) \trianglelefteq (\le_{R1}))$ *l1 r1* $((\le_{R1}) \trianglelefteq (\le_{L1}))$ *r1 l1*
  shows $(\ge_L)$ = *transport-comp.L* $(\ge_{L1})\ (\ge_{R1})$ *l1 r1* $(\ge_{L2})$
  using *assms* **unfolding** *left-rel-eq-comp inv.left-rel-eq-comp*
  by (*simp add*: *rel-comp-assoc*)

corollary *left-rel-inv-iff-left-rel-if-galois-prop* [*iff*]:
  assumes $((\le_{L1}) \trianglelefteq (\le_{R1}))$ *l1 r1* $((\le_{R1}) \trianglelefteq (\le_{L1}))$ *r1 l1*
  shows (*transport-comp.L* $(\ge_{L1})\ (\ge_{R1})$ *l1 r1* $(\ge_{L2})$) $x\ x' \longleftrightarrow x' \le_L x$
  using *assms* **by** (*simp flip*: *ge-left-rel-eq-left-rel-inv-if-galois-prop*)

Simplification of Relations

lemma *left-rel-le-left-rel1I*:
  assumes $((\le_{L1}) \trianglelefteq_h (\le_{R1}))$ *l1 r1*
  and $((\le_{R1})\ {}_h\trianglelefteq\ (\le_{L1}))$ *r1 l1*
  and *trans-L1*: *transitive* $(\le_{L1})$
  and *mono-l1*: $((\le_L) \Rrightarrow_m ((\le_{R1}) \circ\circ (\le_{R1})))$ *l1*

shows $(\leq_{L}) \leq (\leq_{L1})$
proof (*rule le-relI*)
fix $x\ x'$ **assume** $x \leq_{L} x'$
with *mono-l1* **obtain** y **where** $l1\ x \leq_{R1} y\ y \leq_{R1} l1\ x'$ **by** *blast*
with ‹$((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))\ l1\ r1$› ‹$x \leq_{L} x'$› **have** $x \leq_{L1} r1\ y$ **by** *blast*
moreover from ‹$((\leq_{R1})\ {}_h\trianglelefteq (\leq_{L1}))\ r1\ l1$› ‹$y \leq_{R1} l1\ x'$› ‹$x \leq_{L} x'$›
have ... $\leq_{L1} x'$ **by** *blast*
ultimately show $x \leq_{L1} x'$ **using** *trans-L1* **by** *blast*
qed

lemma *left-rel1-le-left-relI*:
assumes $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))\ l1\ r1$
and *mono-l1*: $((\leq_{L1}) \Rrightarrow_m ((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})))\ l1$
shows $(\leq_{L1}) \leq (\leq_{L})$
proof (*rule le-relI*)
fix $x\ x'$ **assume** $x \leq_{L1} x'$
with *mono-l1* **obtain** $y\ y'$ **where**
$l1\ x \leq_{R1} y\ y \leq_{L2} y'\ y' \leq_{R1} l1\ x'$ **by** *blast*
with ‹$((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))\ l1\ r1$› ‹$x \leq_{L1} x'$› **have** $x\ {}_{L1}\lessapprox\ y$ **by** *blast*
moreover note ‹$y \leq_{L2} y'$›
moreover from ‹$y' \leq_{R1} l1\ x'$› ‹$x \leq_{L1} x'$› **have** $y'\ {}_{R1}\lessapprox\ x'$ **by** *blast*
ultimately show $x \leq_{L} x'$ **by** *blast*
qed

corollary *left-rel-eq-left-rel1I*:
assumes $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))\ l1\ r1$
and $((\leq_{R1})\ {}_h\trianglelefteq (\leq_{L1}))\ r1\ l1$
and *transitive* $(\leq_{L1})$
and $((\leq_{L}) \Rrightarrow_m ((\leq_{R1}) \circ\circ (\leq_{R1})))\ l1$
and $((\leq_{L1}) \Rrightarrow_m ((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})))\ l1$
shows $(\leq_{L}) = (\leq_{L1})$
using *assms* **by** (*intro antisym left-rel-le-left-rel1I left-rel1-le-left-relI*)

Note that we may not necessarily have *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*:
$\exists (L1 :: bool \Rightarrow -)\ (R1 :: bool \Rightarrow -)\ l1\ r1$
$(L2 :: bool \Rightarrow -)\ (R2 :: bool \Rightarrow -)\ l2\ r2.$
$(L1 \equiv_o R1)\ l1\ r1$
$\wedge$ *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 (*intro exI conjI*)
let $?L1 = (=) :: bool \Rightarrow -$ **let** $?R1 = ?L1$ **let** $?l1 = id$ **let** $?r1 = ?l1$
let $?L2 = \top :: bool \Rightarrow bool \Rightarrow bool$ **let** $?R2 = ?L2$ **let** $?l2 = id$ **let** $?r2 = ?l2$
interpret *tc* : *transport-comp* $?L1\ ?R1\ ?l1\ ?r1\ ?L2\ ?R2\ ?l2\ ?r2$ **.**
show $(?L1 \equiv_o ?R1)\ ?l1\ ?r1$ **by** *fastforce*

show *equivalence-rel ?L1 equivalence-rel ?R1* **by** (*fact equivalence-eq*)+
show $(?L2 \equiv_o ?R2)$ *?l2 ?r2* **by** *fastforce*
show *equivalence-rel ?L2 equivalence-rel ?R2* **by** (*fact equivalence-top*)+
show $tc.L \neq ?L1$
proof −
  have $\neg(?L1\ False\ True)$ **by** *blast*
  moreover have *tc.L False True* **by** (*intro tc.left-relI*) *auto*
  ultimately show *?thesis* **by** *auto*
qed
qed

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* **.**

lemma *right-rel-if-left-relI*:
assumes $x \le_L x'$
and *l1-R1-if-L1-r1*: $\bigwedge y.\ in\text{-}codom\ (\le_{R1})\ y \Longrightarrow x \le_{L1} r1\ y \Longrightarrow l1\ x \le_{R1} y$
and *t-R1-if-l1-R1*: $\bigwedge y.\ l1\ x \le_{R1} y \Longrightarrow t\ y \le_{R1} y$
and *R2-l2-if-t-L2-if-l1-R1*:
$\bigwedge y\ y'.\ l1\ x \le_{R1} y \Longrightarrow t\ y \le_{L2} y' \Longrightarrow z \le_{R2} l2\ y'$
and *R1-b-if-R1-l1-if-R1-l1*:
$\bigwedge y\ y'.\ y \le_{R1} l1\ x' \Longrightarrow y' \le_{R1} l1\ x' \Longrightarrow y' \le_{R1} b\ y$
and *b-L2-r2-if-in-codom-L2-b-if-R1-l1*:
$\bigwedge y.\ y \le_{R1} l1\ x' \Longrightarrow in\text{-}codom\ (\le_{L2})\ (b\ y) \Longrightarrow b\ y \le_{L2} r2\ z'$
and *in-codom-R2-if-in-codom-L2-b-if-R1-l1*:
$\bigwedge y.\ y \le_{R1} l1\ x' \Longrightarrow in\text{-}codom\ (\le_{L2})\ (b\ y) \Longrightarrow in\text{-}codom\ (\le_{R2})\ z'$
and *rel-comp-le*: $(\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1}) \le (\le_{L2}) \circ\circ (\le_{R1})$
and *in-codom-rel-comp-le*: $in\text{-}codom\ ((\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1})) \le in\text{-}codom\ (\le_{L2})$
shows $z \le_R z'$
proof −
from $\langle x \le_L x' \rangle$ **obtain** $yl\ yl'$ **where** $l1\ x \le_{R1} yl\ \ yl \le_{L2} yl'\ \ yl' \le_{R1} l1\ x'$
  using *l1-R1-if-L1-r1* **by** *blast*
moreover then have $t\ yl \le_{R1} yl$ **by** (*intro t-R1-if-l1-R1*)
ultimately have $((\le_{L2}) \circ\circ (\le_{R1}))\ (t\ yl)\ (l1\ x')$ **using** *rel-comp-le* **by** *blast*
then obtain y **where** $t\ yl \le_{L2} y\ \ y \le_{R1} l1\ x'$ **by** *blast*
show $z \le_R z'$
proof (*rule flip.left-relI*)
  from $\langle t\ yl \le_{L2} y \rangle\ \langle l1\ x \le_{R1} yl \rangle$ **show** $z\ {}_{R2}\lessapprox\ y$
    by (*auto intro*: *R2-l2-if-t-L2-if-l1-R1*)
  from $\langle yl' \le_{R1} l1\ x' \rangle\ \langle y \le_{R1} l1\ x' \rangle$ **show** $y \le_{R1} b\ yl'$
    by (*rule R1-b-if-R1-l1-if-R1-l1*)
  show $b\ yl'\ {}_{L2}\lessapprox\ z'$
  proof (*rule t2.left-GaloisI*)

from ‹$yl' \le_{R1}$ *l1* x'› **have** $yl' \le_{R1}$ *b* yl'
by (*intro R1-b-if-R1-l1-if-R1-l1*)
with ‹*l1* $x \le_{R1}$ *yl*› ‹$yl \le_{L2}$ yl'› *in-codom-rel-comp-le*
have *in-codom* $(\le_{L2})$ (*b* yl') **by** *blast*
with ‹$yl' \le_{R1}$ *l1* x'› **show** *b* $yl' \le_{L2}$ *r2* z' *in-codom* $(\le_{R2})$ z'
by (*auto intro*: *b-L2-r2-if-in-codom-L2-b-if-R1-l1*
in-codom-R2-if-in-codom-L2-b-if-R1-l1)
qed
qed
qed

lemma *right-rel-if-left-relI*$'$:
assumes $x \le_L x'$
and *l1-R1-if-L1-r1*: $\bigwedge y.$ *in-codom* $(\le_{R1})$ $y \Longrightarrow x \le_{L1}$ *r1* $y \Longrightarrow$ *l1* $x \le_{R1} y$
and *R1-b-if-R1-l1*: $\bigwedge y.$ $y \le_{R1}$ *l1* $x' \Longrightarrow y \le_{R1}$ *b* y
and *L2-r2-if-L2-b-if-R1-l1*:
$\bigwedge y\ y'.$ $y \le_{R1}$ *l1* $x' \Longrightarrow y' \le_{L2}$ *b* $y \Longrightarrow y' \le_{L2}$ *r2* z'
and *in-codom-R2-if-L2-b-if-R1-l1*:
$\bigwedge y\ y'.$ $y \le_{R1}$ *l1* $x' \Longrightarrow y' \le_{L2}$ *b* $y \Longrightarrow$ *in-codom* $(\le_{R2})$ z'
and *t-R1-if-R1-l1-if-l1-R1*:
$\bigwedge y\ y'\ y''.$ *l1* $x \le_{R1} y \Longrightarrow$ *l1* $x \le_{R1} y' \Longrightarrow t\ y \le_{R1} y'$
and *R2-l2-t-if-in-dom-L2-t-if-l1-R1*:
$\bigwedge y\ y'.$ *l1* $x \le_{R1} y \Longrightarrow$ *in-dom* $(\le_{L2})$ $(t\ y) \Longrightarrow z \le_{R2}$ *l2* $(t\ y)$
and *in-codom-L2-t-if-in-dom-L2-t-if-l1-R1*:
$\bigwedge y\ y'.$ *l1* $x \le_{R1} y \Longrightarrow$ *in-dom* $(\le_{L2})$ $(t\ y) \Longrightarrow$ *in-codom* $(\le_{L2})$ $(t\ y)$
and *rel-comp-le*: $((\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1})) \le ((\le_{R1}) \circ\circ (\le_{L2}))$
and *in-dom-rel-comp-le*: *in-dom* $((\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1})) \le$ *in-dom* $(\le_{L2})$
shows $z \le_R z'$
proof −
from ‹$x \le_L x'$› **obtain** *yl* yl' **where** *l1* $x \le_{R1}$ *yl* $yl \le_{L2} yl'$ $yl' \le_{R1}$ *l1* x'
using *l1-R1-if-L1-r1* **by** *blast*
moreover then have $yl' \le_{R1}$ *b* yl' **by** (*intro R1-b-if-R1-l1*)
ultimately have $((\le_{R1}) \circ\circ (\le_{L2}))$ (*l1* x) (*b* yl') **using** *rel-comp-le* **by** *blast*
then obtain y **where** *l1* $x \le_{R1} y$ $y \le_{L2}$ *b* yl' **by** *blast*
show $z \le_R z'$
proof (*rule flip.left-relI*)
from ‹$yl' \le_{R1}$ *l1* x'› ‹$y \le_{L2}$ *b* yl'›
have *in-codom* $(\le_{R2})$ z' $y \le_{L2}$ *r2* z'
by (*auto intro*: *in-codom-R2-if-L2-b-if-R1-l1 L2-r2-if-L2-b-if-R1-l1*)
then show $y\ {}_{L2}\lessapprox z'$ **by** *blast*
from ‹*l1* $x \le_{R1}$ *yl*› ‹*l1* $x \le_{R1} y$› **show** *t yl* $\le_{R1} y$ **by** (*rule t-R1-if-R1-l1-if-l1-R1*)
show $z\ {}_{R2}\lessapprox$ *t yl*
proof (*rule flip.t1.left-GaloisI*)
from ‹*l1* $x \le_{R1}$ *yl*› **have** *t yl* $\le_{R1}$ *yl* **by** (*intro t-R1-if-R1-l1-if-l1-R1*)
with ‹$yl \le_{L2} yl'$› ‹$yl' \le_{R1}$ *l1* x'› *in-dom-rel-comp-le* **have** *in-dom* $(\le_{L2})$ (*t*
yl)
by *blast*
with ‹*l1* $x \le_{R1}$ *yl*›
show $z \le_{R2}$ *l2* (*t yl*) *in-codom* $(\le_{L2})$ (*t yl*) **by** (*auto intro*:

R2-l2-t-if-in-dom-L2-t-if-l1-R1 in-codom-L2-t-if-in-dom-L2-t-if-l1-R1)
qed
qed
qed

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 $((\le_{L1})\ {}_h\trianglelefteq\ (\le_{R1}))$ *l1 r1*
and *in-dom* $((\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1})) \le$ *in-dom* $(\le_{L2})$
shows ([*in-dom* $(\le_L)$] $\Rrightarrow_m$ *in-dom* $(\le_{L2})$) *l1*
using *assms* **by** (*intro dep-mono-wrt-predI*) *blast*

lemma *mono-in-codom-left-rel-left1-if-in-codom-rel-comp-le*:
assumes $((\le_{L1})\ {}_h\trianglelefteq\ (\le_{R1}))$ *l1 r1*
and *in-codom* $((\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1})) \le$ *in-codom* $(\le_{L2})$
shows ([*in-codom* $(\le_L)$] $\Rrightarrow_m$ *in-codom* $(\le_{L2})$) *l1*
using *assms* **by** (*intro dep-mono-wrt-predI*) *blast*

Simplification of Compatibility Conditions

Most results will depend on certain compatibility conditions between $(\le_{R1})$ and $(\le_{L2})$. 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*: *in-dom* $(R \circ\circ S \circ\circ R) \le P$
and *rel-comp-le*: $(S \circ\circ R \circ\circ S) \le (S \circ\circ R)$
shows $(R \circ\circ S \circ\circ R) \le (S \circ\circ R)$
proof (*intro le-relI*)
fix *x y* **assume** $(R \circ\circ S \circ\circ R)\ x\ y$
moreover with *in-dom-le refl-S* **have** *S x x* **by** *blast*
ultimately have $((S \circ\circ R \circ\circ S) \circ\circ R)\ x\ y$ **by** *blast*
with *rel-comp-le* **have** $(S \circ\circ R \circ\circ R)\ x\ y$ **by** *blast*
with *trans-R* **show** $(S \circ\circ R)\ x\ y$ **by** *blast*
qed

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* $(R \circ\circ S \circ\circ R) \le P$
and *rel-comp-le*: $(S \circ\circ R \circ\circ S) \le (R \circ\circ S)$
shows $(R \circ\circ S \circ\circ R) \le (R \circ\circ S)$
proof (*intro le-relI*)

fix x y **assume** $(R \circ\circ S \circ\circ R)\ x\ y$
moreover with *in-codom-le refl-S* **have** $S\ y\ y$ **by** *blast*
ultimately have $(R \circ\circ (S \circ\circ R \circ\circ S))\ x\ y$ **by** *blast*
with *rel-comp-le* **have** $(R \circ\circ R \circ\circ S)\ x\ y$ **by** *blast*
with *trans-R* **show** $(R \circ\circ S)\ x\ y$ **by** *blast*
qed

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 −
from *trans-R* **have** *R-R-le*: $(R \circ\circ R) \leq R$ **by** (*intro rel-comp-le-self-if-transitive*)
have $(R \circ\circ S \circ\circ R) \leq (S \circ\circ R \circ\circ R)$
using *monoD*[*OF mono-rel-comp1 R-S-le*] **by** *blast*
also have ... $\leq (S \circ\circ R)$
using *monoD*[*OF mono-rel-comp2 R-R-le*] **by** (*auto simp flip*: *rel-comp-assoc*)
finally show *?thesis* **.**
qed

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 −
from *trans-R* **have** *R-R-le*: $(R \circ\circ R) \leq R$ **by** (*intro rel-comp-le-self-if-transitive*)
have $(R \circ\circ S \circ\circ R) \leq (R \circ\circ R \circ\circ S)$
using *monoD*[*OF mono-rel-comp2 S-R-le*] **by** (*auto simp flip*: *rel-comp-assoc*)
also have ... $\leq (R \circ\circ S)$ **using** *monoD*[*OF mono-rel-comp1 R-R-le*] **by** *blast*
finally show *?thesis* **.**
qed

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 (*intro ext iffI*)
fix x y **assume** $(R \circ\circ S)\ x\ y$
then obtain z **where** $R\ x\ z$ $S\ z\ y$ **by** *blast*
with *R-le* **have** $(S \sqcup (=))\ x\ z$ **by** *blast*
with ‹$S\ z\ y$› *trans-S* **have** $S\ x\ y$ **by** *auto*
moreover from ‹$S\ z\ y$› *refl-R* **have** $R\ y\ y$ **by** *blast*
ultimately show $(S \circ\circ R)\ x\ y$ **by** *blast*
next
fix x y **assume** $(S \circ\circ R)\ x\ y$
then obtain z **where** $S\ x\ z$ $R\ z\ y$ **by** *blast*
with *R-le* **have** $(S \sqcup (=))\ z\ y$ **by** *blast*
with ‹$S\ x\ z$› *trans-S* **have** $S\ x\ y$ **by** *auto*

moreover from ‹$S\ x\ y$› *refl-R* **have** $R\ x\ x$ **by** *blast*
ultimately show $(R \circ\circ S)\ x\ y$ **by** *blast*
qed

lemma *rel-comp-eq-rel-comp-if-in-field-le-if-le-eq*:
assumes *le-eq*: $R \leq (=)$
and *in-field-le*: *in-field* $S \leq$ *in-field* R
shows $(R \circ\circ S) = (S \circ\circ R)$
proof (*intro ext iffI*)
fix $x\ y$ **assume** $(R \circ\circ S)\ x\ y$
then obtain z **where** $R\ x\ z\ S\ z\ y$ **by** *blast*
with *le-eq* **have** $S\ x\ y$ **by** *blast*
with *assms* **show** $(S \circ\circ R)\ x\ y$ **by** *blast*
next
fix $x\ y$ **assume** $(S \circ\circ R)\ x\ y$
then obtain z **where** $S\ x\ z\ R\ z\ y$ **by** *blast*
with *le-eq* **have** $S\ x\ y$ **by** *blast*
with *assms* **show** $(R \circ\circ S)\ x\ y$ **by** *blast*
qed

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 $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq ((\leq_{L2}) \circ\circ (\leq_{R1}))$
using *assms* **by** (*intro rel-comp-comp-le-rel-comp-if-rel-comp-comp-if-in-codom-leI*)
auto

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 $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq ((\leq_{R1}) \circ\circ (\leq_{L2}))$
using *assms* **by** (*intro rel-comp-comp-le-rel-comp-if-rel-comp-comp-if-in-dom-leI*)
auto

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* $(\leq_{L2})$
using *monoD*[*OF mono-in-dom assms*] **by** (*auto intro*: *in-dom-if-in-dom-rel-comp*)

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* $(\leq_{L2})$

using *monoD*[*OF mono-in-codom assms*]
by (*auto intro*: *in-codom-if-in-codom-rel-comp*)

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* $\equiv$
$(\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1}) \le (\le_{R1}) \circ\circ (\le_{L2})$
$\wedge$ *in-dom* $((\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1})) \le$ *in-dom* $(\le_{L2})$
$\wedge$ $((\le_{L2}) \circ\circ (\le_{R1}) \circ\circ (\le_{L2})) \le ((\le_{L2}) \circ\circ (\le_{R1}))$
$\wedge$ *in-dom* $((\le_{L2}) \circ\circ (\le_{R1}) \circ\circ (\le_{L2})) \le$ *in-dom* $(\le_{R1})$

lemma *middle-compatible-domI* [*intro*]:
assumes $(\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1}) \le (\le_{R1}) \circ\circ (\le_{L2})$
and *in-dom* $((\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1})) \le$ *in-dom* $(\le_{L2})$
and $((\le_{L2}) \circ\circ (\le_{R1}) \circ\circ (\le_{L2})) \le ((\le_{L2}) \circ\circ (\le_{R1}))$
and *in-dom* $((\le_{L2}) \circ\circ (\le_{R1}) \circ\circ (\le_{L2})) \le$ *in-dom* $(\le_{R1})$
shows *middle-compatible-dom*
unfolding *middle-compatible-dom-def* **using** *assms* **by** *blast*

lemma *middle-compatible-domE* [*elim*]:
assumes *middle-compatible-dom*
obtains $(\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1}) \le (\le_{R1}) \circ\circ (\le_{L2})$
and *in-dom* $((\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1})) \le$ *in-dom* $(\le_{L2})$
and $((\le_{L2}) \circ\circ (\le_{R1}) \circ\circ (\le_{L2})) \le ((\le_{L2}) \circ\circ (\le_{R1}))$
and *in-dom* $((\le_{L2}) \circ\circ (\le_{R1}) \circ\circ (\le_{L2})) \le$ *in-dom* $(\le_{R1})$
using *assms* **unfolding** *middle-compatible-dom-def* **by** *blast*

definition *middle-compatible-codom* $\equiv$
$((\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1})) \le ((\le_{L2}) \circ\circ (\le_{R1}))$
$\wedge$ *in-codom* $((\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1})) \le$ *in-codom* $(\le_{L2})$
$\wedge$ $(\le_{L2}) \circ\circ (\le_{R1}) \circ\circ (\le_{L2}) \le (\le_{R1}) \circ\circ (\le_{L2})$
$\wedge$ *in-codom* $((\le_{L2}) \circ\circ (\le_{R1}) \circ\circ (\le_{L2})) \le$ *in-codom* $(\le_{R1})$

lemma *middle-compatible-codomI* [*intro*]:
assumes $((\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1})) \le ((\le_{L2}) \circ\circ (\le_{R1}))$
and *in-codom* $((\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1})) \le$ *in-codom* $(\le_{L2})$
and $(\le_{L2}) \circ\circ (\le_{R1}) \circ\circ (\le_{L2}) \le (\le_{R1}) \circ\circ (\le_{L2})$
and *in-codom* $((\le_{L2}) \circ\circ (\le_{R1}) \circ\circ (\le_{L2})) \le$ *in-codom* $(\le_{R1})$
shows *middle-compatible-codom*
unfolding *middle-compatible-codom-def* **using** *assms* **by** *blast*

lemma *middle-compatible-codomE* [*elim*]:
assumes *middle-compatible-codom*
obtains $((\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1})) \le ((\le_{L2}) \circ\circ (\le_{R1}))$
and *in-codom* $((\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1})) \le$ *in-codom* $(\le_{L2})$
and $(\le_{L2}) \circ\circ (\le_{R1}) \circ\circ (\le_{L2}) \le (\le_{R1}) \circ\circ (\le_{L2})$

and *in-codom* $((\le_{L2}) \circ\circ (\le_{R1}) \circ\circ (\le_{L2})) \le$ *in-codom* $(\le_{R1})$
using *assms* **unfolding** *middle-compatible-codom-def* **by** *blast*

context
begin

interpretation *flip* : *transport-comp R2 L2 r2 l2 R1 L1 r1 l1* **.**

lemma *rel-comp-comp-le-assms-if-in-codom-rel-comp-comp-leI*:
assumes *preorder-on* (*in-field* $(\le_{R1})$) $(\le_{R1})$
and *preorder-on* (*in-field* $(\le_{L2})$) $(\le_{L2})$
and *middle-compatible-codom*
shows *middle-compatible-dom*
using *assms* **by** (*intro middle-compatible-domI*)
(*auto intro*!:
left2-right1-left2-le-left2-right1-if-right1-left2-right1-le-left2-right1
flip.left2-right1-left2-le-left2-right1-if-right1-left2-right1-le-left2-right1
in-dom-right1-left2-right1-le-if-right1-left2-right1-le
flip.in-dom-right1-left2-right1-le-if-right1-left2-right1-le
intro: *reflexive-on-if-le-pred-if-reflexive-on in-field-if-in-codom*)

lemma *rel-comp-comp-le-assms-if-in-dom-rel-comp-comp-leI*:
assumes *preorder-on* (*in-field* $(\le_{R1})$) $(\le_{R1})$
and *preorder-on* (*in-field* $(\le_{L2})$) $(\le_{L2})$
and *middle-compatible-dom*
shows *middle-compatible-codom*
using *assms* **by** (*intro middle-compatible-codomI*)
(*auto intro*!:
left2-right1-left2-le-right1-left2-if-right1-left2-right1-le-right1-left2I
flip.left2-right1-left2-le-right1-left2-if-right1-left2-right1-le-right1-left2I
in-codom-right1-left2-right1-le-if-right1-left2-right1-le
flip.in-codom-right1-left2-right1-le-if-right1-left2-right1-le
intro: *reflexive-on-if-le-pred-if-reflexive-on in-field-if-in-dom*)

lemma *middle-compatible-dom-iff-middle-compatible-codom-if-preorder-on*:
assumes *preorder-on* (*in-field* $(\le_{R1})$) $(\le_{R1})$
and *preorder-on* (*in-field* $(\le_{L2})$) $(\le_{L2})$
shows *middle-compatible-dom* $\longleftrightarrow$ *middle-compatible-codom*
using *assms* **by** (*intro iffI rel-comp-comp-le-assms-if-in-codom-rel-comp-comp-leI*)
(*auto intro*!: *rel-comp-comp-le-assms-if-in-dom-rel-comp-comp-leI*)

end

Finally we derive some sufficient assumptions for the compatibility conditions.

lemma *right1-left2-right1-le-assms-if-right1-left2-eqI*:
assumes *transitive* $(\le_{R1})$
and $((\le_{R1}) \circ\circ (\le_{L2})) = ((\le_{L2}) \circ\circ (\le_{R1}))$
shows $((\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1})) \le ((\le_{L2}) \circ\circ (\le_{R1}))$

and $((\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1})) \le ((\le_{R1}) \circ\circ (\le_{L2}))$
using *assms rel-comp-comp-le-rel-comp-if-rel-comp-le-if-transitive*[*of R1 L2*]
by *auto*

interpretation *flip* : *transport-comp R2 L2 r2 l2 R1 L1 r1 l1*
rewrites $((\le_{L2}) \circ\circ (\le_{R1})) = ((\le_{R1}) \circ\circ (\le_{L2})) \equiv ((\le_{R1}) \circ\circ (\le_{L2})) = ((\le_{L2}) \circ\circ (\le_{R1}))$
by (*simp only*: *eq-commute*)

lemma *middle-compatible-codom-if-rel-comp-eq-if-transitive*:
assumes *transitive* $(\le_{R1})$ *transitive* $(\le_{L2})$
and $((\le_{R1}) \circ\circ (\le_{L2})) = ((\le_{L2}) \circ\circ (\le_{R1}))$
shows *middle-compatible-codom*
using *assms* **by** (*intro middle-compatible-codomI*
in-codom-right1-left2-right1-le-if-right1-left2-right1-le
flip.in-codom-right1-left2-right1-le-if-right1-left2-right1-le
right1-left2-right1-le-assms-if-right1-left2-eqI
flip.right1-left2-right1-le-assms-if-right1-left2-eqI)
auto

lemma *middle-compatible-codom-if-right1-le-left2-eqI*:
assumes *preorder-on* (*in-field* $(\le_{R1})$) $(\le_{R1})$ *transitive* $(\le_{L2})$
and $(\le_{R1}) \le (\le_{L2}) \sqcup (=)$
and *in-field* $(\le_{L2}) \le$ *in-field* $(\le_{R1})$
shows *middle-compatible-codom*
using *assms* **by** (*intro middle-compatible-codomI*
in-codom-right1-left2-right1-le-if-right1-left2-right1-le
flip.in-codom-right1-left2-right1-le-if-right1-left2-right1-le
right1-left2-right1-le-assms-if-right1-left2-eqI
flip.right1-left2-right1-le-assms-if-right1-left2-eqI
rel-comp-eq-rel-comp-if-le-if-transitive-if-reflexive)
(*auto intro*: *reflexive-on-if-le-pred-if-reflexive-on*)

lemma *middle-compatible-codom-if-right1-le-eqI*:
assumes $(\le_{R1}) \le (=)$
and *transitive* $(\le_{L2})$
and *in-field* $(\le_{L2}) \le$ *in-field* $(\le_{R1})$
shows *middle-compatible-codom*
using *assms* **by** (*intro middle-compatible-codomI*
in-codom-right1-left2-right1-le-if-right1-left2-right1-le
flip.in-codom-right1-left2-right1-le-if-right1-left2-right1-le
right1-left2-right1-le-assms-if-right1-left2-eqI
flip.right1-left2-right1-le-assms-if-right1-left2-eqI
rel-comp-eq-rel-comp-if-in-field-le-if-le-eq)
auto

end

end

2.4.2 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* = ε_1 **and** *flip.t1.counit* $\equiv \eta_2$
by (*simp-all only*: *t1.flip-unit-eq-counit t2.flip-counit-eq-unit*)

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 $((\leq_{L2}) \Rrightarrow_m (\leq_{R2}))$ *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*: ([*in-codom* $(\leq_R)$] $\Rrightarrow_m$ *in-codom* $(\leq_{R1})$) *r2*
shows $((\leq_L)\ _h\trianglelefteq\ (\leq_R))$ *l r*
proof (*rule half-galois-prop-leftI*)
fix *x z* **assume** $x\ _L\lessapprox\ z$
then show $l\ x \leq_R z$
proof (*intro right-rel-if-left-relI*)
from ‹$x\ _L\lessapprox\ z$› **show** *in-codom* $(\leq_{R2})$ *z* **by** *blast*
fix *y* **assume** $y \leq_{R1}$ *l1* (*r z*)
moreover have *l1* (*r z*) $\leq_{R1}$ *r2 z*
proof −
from *mono-in-codom-r2* ‹$x\ _L\lessapprox\ z$› **have** *in-codom* $(\leq_{R1})$ (*r2 z*) **by** *blast*
with *deflationary-counit1* **show** *l1* (*r z*) $\leq_{R1}$ *r2 z* **by** *auto*
qed
ultimately show $y \leq_{R1}$ *r2 z* **using** *trans-R1* **by** *blast*
next
fix *y* **assume** *l1 x* $\leq_{L2}$ *y*
with ‹$((\leq_{L2}) \Rrightarrow_m (\leq_{R2}))$ *l2*› **show** *l x* $\leq_{R2}$ *l2 y* **by** *auto*
qed (*insert assms*, *auto*)
qed

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 $((\leq_{L2}) \Rrightarrow_m (\leq_{R2}))$ *l2*

and *refl-L2*: *reflexive-on* (*in-dom* ($\le_{L2}$)) ($\le_{L2}$)
and (($\le_{R1}$) ∘∘ ($\le_{L2}$) ∘∘ ($\le_{R1}$)) $\le$ (($\le_{R1}$) ∘∘ ($\le_{L2}$))
and *in-dom* (($\le_{R1}$) ∘∘ ($\le_{L2}$) ∘∘ ($\le_{R1}$)) $\le$ *in-dom* ($\le_{L2}$)
and *mono-in-codom-r2*: ([*in-codom* ($\le_R$)] $\Rrightarrow_m$ *in-codom* ($\le_{R1}$)) *r2*
shows (($\le_L$) ${}_h\trianglelefteq$ ($\le_R$)) *l r*
proof (*rule half-galois-prop-leftI*)
fix *x z* **assume** *x* ${}_L\lessapprox$ *z*
then show *l x* $\le_R$ *z*
proof (*intro right-rel-if-left-relI'*)
from ‹*x* ${}_L\lessapprox$ *z*› **show** *in-codom* ($\le_{R2}$) *z* **by** *blast*
fix *y* **assume** *y* $\le_{R1}$ *l1* (*r z*)
moreover have *l1* (*r z*) $\le_{R1}$ *r2 z*
proof −
from *mono-in-codom-r2* ‹*x* ${}_L\lessapprox$ *z*› **have** *in-codom* ($\le_{R1}$) (*r2 z*) **by** *blast*
with *deflationary-counit1* **show** *l1* (*r z*) $\le_{R1}$ *r2 z* **by** *auto*
qed
ultimately show *y* $\le_{R1}$ *r2 z* **using** *trans-R1* **by** *blast*
next
assume *in-dom* ($\le_{L2}$) (*l1 x*)
with *refl-L2* **have** *l1 x* $\le_{L2}$ *l1 x* **by** *blast*
with ‹(($\le_{L2}$) $\Rrightarrow_m$ ($\le_{R2}$)) *l2*› **show** *in-codom* ($\le_{L2}$) (*l1 x*) *l x* $\le_{R2}$ *l2* (*l1 x*)
by *auto*
qed (*insert assms*, *auto*)
qed

lemma *half-galois-prop-right-left-rightI*:
assumes (($\le_{R1}$) $\Rrightarrow_m$ ($\le_{L1}$)) *r1*
and (($\le_{L1}$) $\trianglelefteq_h$ ($\le_{R1}$)) *l1 r1*
and *inflationary-counit1*: *inflationary-on* (*in-codom* ($\le_{R1}$)) ($\le_{R1}$) ε_1
and (($\le_{R2}$) ${}_h\trianglelefteq$ ($\le_{L2}$)) *r2 l2*
and *inflationary-unit2*: *inflationary-on* (*in-dom* ($\le_{L2}$)) ($\le_{L2}$) η_2
and *trans-L2*: *transitive* ($\le_{L2}$)
and *mono-in-dom-l1*: ([*in-dom* ($\le_L$)] $\Rrightarrow_m$ *in-dom* ($\le_{L2}$)) *l1*
and (($\le_{L2}$) ∘∘ ($\le_{R1}$) ∘∘ ($\le_{L2}$)) $\le$ (($\le_{R1}$) ∘∘ ($\le_{L2}$))
and *in-codom* (($\le_{L2}$) ∘∘ ($\le_{R1}$) ∘∘ ($\le_{L2}$)) $\le$ *in-codom* ($\le_{R1}$)
shows (($\le_L$) $\trianglelefteq_h$ ($\le_R$)) *l r*
proof (*rule half-galois-prop-rightI*)
fix *x z* **assume** *x* $\lessapprox_R$ *z*
then show *x* $\le_L$ *r z*
proof (*intro flip.right-rel-if-left-relI*)
fix *y* **assume** *r2* (*l x*) $\le_{L2}$ *y*
moreover have *l1 x* $\le_{L2}$ *r2* (*l x*)
proof −
from *mono-in-dom-l1* ‹*x* $\lessapprox_R$ *z*› **have** *in-dom* ($\le_{L2}$) (*l1 x*) **by** *blast*
with *inflationary-unit2* **show** *l1 x* $\le_{L2}$ *r2* (*l x*) **by** *auto*
qed
ultimately show *l1 x* $\le_{L2}$ *y* **using** *trans-L2* **by** *blast*
fix *y* **assume** *l1 x* $\le_{R1}$ *y*
with ‹(($\le_{L1}$) $\trianglelefteq_h$ ($\le_{R1}$)) *l1 r1*› ‹*x* $\lessapprox_R$ *z*› **show** *x* $\le_{L1}$ *r1 y* **by** *blast*

next
assume *in-codom* $(\le_{R1})$ $(r2\ z)$
with *inflationary-counit1* **show** $r2\ z \le_{R1} l1\ (r\ z)$ **by** *auto*
from ‹$((\le_{R1}) \Rrightarrow_m (\le_{L1}))\ r1$› ‹*in-codom* $(\le_{R1})$ $(r2\ z)$› **show** *in-codom* $(\le_{L1})$ $(r\ z)$
by (*auto intro*: *in-codom-if-rel-if-dep-mono-wrt-rel*)
qed (*insert assms*, *auto elim*: *galois-rel.left-GaloisE*)
qed

lemma *half-galois-prop-right-left-rightI′*:
assumes $((\le_{R1}) \Rrightarrow_m (\le_{L1}))\ r1$
and *inflationary-unit1*: *inflationary-on* (*in-dom* $(\le_{L1})$) $(\le_{L1})$ η_1
and *inflationary-counit1*: $\bigwedge y\ z.\ y \le_{R1} r2\ z \Longrightarrow y \le_{R1} l1\ (r\ z)$
and *in-dom* $(\le_{R1}) \le$ *in-codom* $(\le_{R1})$
and $((\le_{R2})\ {}_h\trianglelefteq\ (\le_{L2}))\ r2\ l2$
and *inflationary-unit2*: *inflationary-on* (*in-dom* $(\le_{L2})$) $(\le_{L2})$ η_2
and *trans-L2*: *transitive* $(\le_{L2})$
and *mono-in-dom-l1*: ([*in-dom* $(\le_L)$] $\Rrightarrow_m$ *in-dom* $(\le_{L2})$) $l1$
and $((\le_{L2}) \circ\circ (\le_{R1}) \circ\circ (\le_{L2})) \le ((\le_{L2}) \circ\circ (\le_{R1}))$
and *in-dom* $((\le_{L2}) \circ\circ (\le_{R1}) \circ\circ (\le_{L2})) \le$ *in-dom* $(\le_{R1})$
shows $((\le_L) \trianglelefteq_h (\le_R))\ l\ r$
proof (*rule half-galois-prop-rightI*)
fix $x\ z$ **assume** $x \lessapprox_R z$
then show $x \le_L r\ z$
proof (*intro flip.right-rel-if-left-relI′*)
from ‹$x \lessapprox_R z$› *inflationary-unit1* **show** $x \le_{L1} r1\ (l1\ x)$
by (*fastforce elim*: *galois-rel.left-GaloisE*)
fix y **assume** $y \le_{R1} r2\ z$
with *inflationary-counit1* **show** $y \le_{R1} l1\ (r\ z)$ **by** *auto*
next
fix y
from *mono-in-dom-l1* ‹$x \lessapprox_R z$› **have** *in-dom* $(\le_{L2})$ $(l1\ x)$ **by** *blast*
with *inflationary-unit2* **have** $l1\ x \le_{L2} r2\ (l\ x)$ **by** *auto*
moreover assume $r2\ (l\ x) \le_{L2} y$
ultimately show $l1\ x \le_{L2} y$ **using** *trans-L2* **by** *blast*
qed (*insert assms*, *auto elim*: *galois-rel.left-GaloisE*)
qed

lemma *galois-prop-left-rightI*:
assumes $((\le_{R1}) \Rrightarrow_m (\le_{L1}))\ r1$
and $((\le_{L1}) \trianglelefteq (\le_{R1}))\ l1\ r1$
and *rel-equivalence-on* (*in-codom* $(\le_{R1})$) $(\le_{R1})$ ε_1
and *transitive* $(\le_{R1})$
and $((\le_{L2}) \Rrightarrow_m (\le_{R2}))\ l2$
and $((\le_{R2})\ {}_h\trianglelefteq\ (\le_{L2}))\ r2\ l2$
and *inflationary-on* (*in-dom* $(\le_{L2})$) $(\le_{L2})$ η_2
and *preorder-on* (*in-field* $(\le_{L2})$) $(\le_{L2})$
and *middle-compatible-codom*
shows $((\le_L) \trianglelefteq (\le_R))\ l\ r$

using *assms* **by** (*intro galois-propI*
half-galois-prop-left-left-rightI half-galois-prop-right-left-rightI
flip.mono-in-codom-left-rel-left1-if-in-codom-rel-comp-le
mono-in-dom-left-rel-left1-if-in-dom-rel-comp-le
in-dom-right1-left2-right1-le-if-right1-left2-right1-le)
(*auto elim*!: *preorder-on-in-fieldE*
intro: *reflexive-on-if-le-pred-if-reflexive-on in-field-if-in-codom*)

lemma *galois-prop-left-rightI'*:
assumes $((\le_{R1}) \Rrightarrow_m (\le_{L1}))$ *r1*
and $((\le_{L1})\ {}_h\trianglelefteq\ (\le_{R1}))$ *l1 r1*
and *inflationary-on* (*in-dom* $(\le_{L1})$) $(\le_{L1})$ η_1
and *rel-equiv-counit1*: *rel-equivalence-on* (*in-field* $(\le_{R1})$) $(\le_{R1})$ ε_1
and *trans-R1*: *transitive* $(\le_{R1})$
and $((\le_{L2}) \Rrightarrow_m (\le_{R2}))$ *l2*
and $((\le_{R2})\ {}_h\trianglelefteq\ (\le_{L2}))$ *r2 l2*
and *inflationary-on* (*in-dom* $(\le_{L2})$) $(\le_{L2})$ η_2
and *preorder-on* (*in-field* $(\le_{L2})$) $(\le_{L2})$
and *middle-compatible-dom*
shows $((\le_L) \trianglelefteq (\le_R))$ *l r*
proof (*rule galois-propI*)
show $((\le_L)\ {}_h\trianglelefteq\ (\le_R))$ *l r* **using** *assms*
by (*intro half-galois-prop-left-left-rightI'*
flip.mono-in-codom-left-rel-left1-if-in-codom-rel-comp-le
flip.in-codom-right1-left2-right1-le-if-right1-left2-right1-le)
(*auto elim*!: *rel-equivalence-onE preorder-on-in-fieldE*
intro: *deflationary-on-if-le-pred-if-deflationary-on*
reflexive-on-if-le-pred-if-reflexive-on
in-field-if-in-dom in-field-if-in-codom)
have $y \le_{R1}$ *l1* (*r1* (*r2 z*)) **if** $y \le_{R1}$ *r2 z* **for** *y z*
proof −
note ‹$y \le_{R1}$ *r2 z*›
moreover with *rel-equiv-counit1* **have** *r2 z* $\le_{R1}$ ε_1 (*r2 z*) **by** *auto*
ultimately show *?thesis* **using** *trans-R1* **by** *auto*
qed
moreover have *in-dom* $(\le_{R1}) \le$ *in-codom* $(\le_{R1})$
proof −
from *rel-equiv-counit1 trans-R1* **have** *reflexive-on* (*in-field* $(\le_{R1})$) $(\le_{R1})$
by (*intro reflexive-on-in-field-if-transitive-if-rel-equivalence-on*) *auto*
then show *?thesis* **by** (*simp only*: *in-codom-eq-in-dom-if-reflexive-on-in-field*)
qed
ultimately show $((\le_L) \trianglelefteq_h (\le_R))$ *l r* **using** *assms*
by (*intro half-galois-prop-right-left-rightI'*
mono-in-dom-left-rel-left1-if-in-dom-rel-comp-le)
auto
qed

end

end

2.4.3 Monotonicity

```
theory Transport-Compositions-Generic-Monotone
  imports
    Transport-Compositions-Generic-Base
begin

context transport-comp
begin

lemma mono-wrt-rel-leftI:
  assumes ((≤L1) h⊴ (≤R1)) l1 r1
  and ((≤L2) ⇛m (≤R2)) l2
  and inflationary-unit2: inflationary-on (in-codom (≤L2)) (≤L2) η2
  and ((≤R1) ∘∘ (≤L2) ∘∘ (≤R1)) ≤ ((≤L2) ∘∘ (≤R1))
  and in-codom ((≤R1) ∘∘ (≤L2) ∘∘ (≤R1)) ≤ in-codom (≤L2)
  shows ((≤L) ⇛m (≤R)) l
proof (rule dep-mono-wrt-relI)
  fix x x' assume x ≤L x'
  then show l x ≤R l x'
  proof (rule right-rel-if-left-relI)
    fix y' assume l1 x ≤L2 y'
    with ‹((≤L2) ⇛m (≤R2)) l2› show l x ≤R2 l2 y' by auto
  next
    assume in-codom (≤L2) (l1 x')
    with inflationary-unit2 show l1 x' ≤L2 r2 (l x') by auto
    from ‹in-codom (≤L2) (l1 x')› ‹((≤L2) ⇛m (≤R2)) l2›
      show in-codom (≤R2) (l x') by auto
  qed (insert assms, auto)
qed

lemma mono-wrt-rel-leftI':
  assumes ((≤L1) h⊴ (≤R1)) l1 r1
  and ((≤L2) ⇛m (≤R2)) l2
  and ((≤L2) ⊴h (≤R2)) l2 r2
  and refl-L2: reflexive-on (in-dom (≤L2)) (≤L2)
  and ((≤R1) ∘∘ (≤L2) ∘∘ (≤R1)) ≤ ((≤R1) ∘∘ (≤L2))
  and in-dom ((≤R1) ∘∘ (≤L2) ∘∘ (≤R1)) ≤ in-dom (≤L2)
  shows ((≤L) ⇛m (≤R)) l
proof (rule dep-mono-wrt-relI)
  fix x x' assume x ≤L x'
  then show l x ≤R l x'
  proof (rule right-rel-if-left-relI')
    fix y' assume y' ≤L2 l1 x'
    moreover with ‹((≤L2) ⇛m (≤R2)) l2› have l2 y' ≤R2 l x' by auto
    ultimately show in-codom (≤R2) (l x') y' ≤L2 r2 (l x')
```

using ‹$((\le_{L2}) \trianglelefteq_h (\le_{R2}))$ *l2 r2*› **by** *auto*
next
assume *in-dom* $(\le_{L2})$ *(l1 x)*
with *refl-L2* ‹$((\le_{L2}) \Rrightarrow_m (\le_{R2}))$ *l2*› **show** $l\ x \le_{R2} l2\ (l1\ x)$ **by** *auto*
qed (*insert assms*, *auto*)
qed

end

end

2.4.4 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* $\equiv \eta_2$
by (*simp-all only*: *t1.flip-unit-eq-counit t2.flip-counit-eq-unit*)

lemma *galois-connection-left-rightI*:
assumes $((\le_{R1}) \Rrightarrow_m (\le_{L1}))$ *r1*
and $((\le_{L1}) \trianglelefteq (\le_{R1}))$ *l1 r1*
and *rel-equivalence-on* (*in-codom* $(\le_{R1})$) $(\le_{R1})$ ε_1
and *transitive* $(\le_{R1})$
and $((\le_{L2}) \Rrightarrow_m (\le_{R2}))$ *l2*
and $((\le_{R2})\ {}_h\!\trianglelefteq (\le_{L2}))$ *r2 l2*
and *inflationary-on* (*in-field* $(\le_{L2})$) $(\le_{L2})$ η_2
and *preorder-on* (*in-field* $(\le_{L2})$) $(\le_{L2})$
and *middle-compatible-codom*
shows $((\le_L) \dashv (\le_R))$ *l r*
using *assms* **by** (*intro galois-connectionI galois-prop-left-rightI*
mono-wrt-rel-leftI flip.mono-wrt-rel-leftI)
(*auto intro*: *inflationary-on-if-le-pred-if-inflationary-on*
in-field-if-in-dom in-field-if-in-codom)

lemma *galois-connection-left-rightI'*:
assumes $((\le_{R1}) \Rrightarrow_m (\le_{L1}))$ *r1*
and $((\le_{L1})\ {}_h\!\trianglelefteq (\le_{R1}))$ *l1 r1*
and $((\le_{R1}) \trianglelefteq_h (\le_{L1}))$ *r1 l1*
and *inflationary-on* (*in-dom* $(\le_{L1})$) $(\le_{L1})$ η_1
and *rel-equivalence-on* (*in-field* $(\le_{R1})$) $(\le_{R1})$ ε_1

and *transitive* $(\leq_{R1})$
and $((\leq_{L2}) \Rrightarrow_m (\leq_{R2}))$ *l2*
and $((\leq_{L2}) \trianglelefteq_h (\leq_{R2}))$ *l2 r2*
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 $((\leq_L) \dashv (\leq_R))$ *l r*
using *assms* **by** (*intro galois-connectionI galois-prop-left-rightI′*
mono-wrt-rel-leftI′ flip.mono-wrt-rel-leftI′)
(*auto elim*!: *preorder-on-in-fieldE*
intro!: *reflexive-on-in-field-if-transitive-if-rel-equivalence-on*
intro: *reflexive-on-if-le-pred-if-reflexive-on in-field-if-in-dom*)

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*
using *assms* **by** (*intro galois-connection-left-rightI*)
(*auto elim*!: *galois.galois-connectionE*
intro!: *flip.t2.rel-equivalence-on-unit-if-reflexive-on-if-galois-equivalence*
t2.inflationary-on-unit-if-reflexive-on-if-galois-equivalence
intro: *rel-equivalence-on-if-le-pred-if-rel-equivalence-on*
in-field-if-in-codom)

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*
using *assms* **by** (*intro galois-connection-left-rightI′*)
(*auto elim*!: *rel-equivalence-onE*
intro!: *t1.half-galois-prop-left-left-right-if-transitive-if-deflationary-on-if-mono-wrt-rel*
flip.t1.half-galois-prop-left-left-right-if-transitive-if-deflationary-on-if-mono-wrt-rel
t2.half-galois-prop-right-left-right-if-transitive-if-inflationary-on-if-mono-wrt-rel
flip.t2.half-galois-prop-right-left-right-if-transitive-if-inflationary-on-if-mono-wrt-rel
preorder-on-in-field-if-transitive-if-rel-equivalence-on
rel-comp-comp-le-assms-if-in-codom-rel-comp-comp-leI
intro: *inflationary-on-if-le-pred-if-inflationary-on*
deflationary-on-if-le-pred-if-deflationary-on
in-field-if-in-dom in-field-if-in-codom)

end

end

2.4.5 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* = ε_1 **and** *flip.t1.counit* ≡ η_2 **and** *flip.t1.unit* ≡ ε_2
  by (*simp-all only*: *order-functors.flip-unit-eq-counit*)

lemma *galois-equivalenceI*:
  assumes $((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))$ *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 $((\leq_{L2}) \Rrightarrow_m (\leq_{R2}))$ *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*
  using *assms* **by** (*intro galois-equivalenceI galois-connection-left-rightI*
    flip.galois-prop-left-rightI)
  (*auto intro*!: *preorder-on-in-field-if-transitive-if-rel-equivalence-on*
    intro: *rel-equivalence-on-if-le-pred-if-rel-equivalence-on*
      inflationary-on-if-le-pred-if-inflationary-on
      in-field-if-in-dom in-field-if-in-codom)

lemma *galois-equivalenceI'*:
  assumes $((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))$ *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 $((\leq_{L2}) \Rrightarrow_m (\leq_{R2}))$ *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*
using *assms* **by** (*intro galois.galois-equivalenceI galois-connection-left-rightI'*
flip.galois-prop-left-rightI')
(*auto elim*!: *rel-equivalence-onE*
intro!: *preorder-on-in-field-if-transitive-if-rel-equivalence-on*
intro: *inflationary-on-if-le-pred-if-inflationary-on*
in-field-if-in-dom)

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*
using *assms* **by** (*intro galois-equivalenceI*)
(*auto intro*!: *t2.rel-equivalence-on-unit-if-reflexive-on-if-galois-equivalence*
flip.t2.rel-equivalence-on-unit-if-reflexive-on-if-galois-equivalence
intro: *reflexive-on-if-le-pred-if-reflexive-on*
in-field-if-in-dom in-field-if-in-codom)

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*
using *assms* **by** (*intro galois-equivalenceI'*)
(*auto elim*!: *rel-equivalence-onE*
intro!: *t1.half-galois-prop-left-left-right-if-transitive-if-deflationary-on-if-mono-wrt-rel*
flip.t1.half-galois-prop-left-left-right-if-transitive-if-deflationary-on-if-mono-wrt-rel
t2.half-galois-prop-right-left-right-if-transitive-if-inflationary-on-if-mono-wrt-rel
flip.t2.half-galois-prop-right-left-right-if-transitive-if-inflationary-on-if-mono-wrt-rel
rel-comp-comp-le-assms-if-in-codom-rel-comp-comp-leI
preorder-on-in-field-if-transitive-if-rel-equivalence-on
intro: *deflationary-on-if-le-pred-if-deflationary-on*
inflationary-on-if-le-pred-if-inflationary-on
in-field-if-in-dom in-field-if-in-codom)

end

end

2.4.6 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* $\equiv \varepsilon_1$
by (*simp only*: *t1.flip-unit-eq-counit*)

lemma *left-Galois-le-comp-left-GaloisI*:
assumes *mono-r1*: $((\le_{R1}) \Rrightarrow_m (\le_{L1}))$ *r1*
and *galois-prop1*: $((\le_{L1}) \trianglelefteq (\le_{R1}))$ *l1 r1*
and *preorder-R1*: *preorder-on* (*in-field* $(\le_{R1})$) $(\le_{R1})$
and *rel-comp-le*: $((\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1})) \le ((\le_{R1}) \circ\circ (\le_{L2}))$
and *mono-in-codom-r2*: ([*in-codom* $(\le_R)$] $\Rrightarrow_m$ *in-codom* $(\le_{R1})$) *r2*
shows $({}_{L}\lessapprox) \le (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$
proof (*rule le-relI*)
fix *x z* **assume** $x \;{}_{L}\lessapprox\; z$
then have *in-codom* $(\le_R)$ z $x \le_L r\ z$ **by** *auto*
with *galois-prop1* **obtain** $y\ y'$ **where** *in-dom* $(\le_{L1})$ x $l1\ x \le_{R1} y$ $y \le_{L2} y'$ y'
$\le_{R1} \varepsilon_1$ $(r2\ z)$
by (*auto elim*!: *left-relE*)
moreover have ε_1 $(r2\ z) \le_{R1} r2\ z$
proof −
from *mono-in-codom-r2* ‹*in-codom* $(\le_R)$ *z*› **have** *in-codom* $(\le_{R1})$ $(r2\ z)$ **by**
blast
with *mono-r1 galois-prop1 preorder-R1* **show** *?thesis* **by** (*blast intro*!:
t1.counit-rel-if-reflexive-on-if-half-galois-prop-left-if-mono-wrt-rel)
qed
ultimately have $y' \le_{R1} r2\ z$ **using** *preorder-R1* **by** *blast*
with ‹$l1\ x \le_{R1} y$› ‹$y \le_{L2} y'$› **have** $((\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1}))$ $(l1\ x)$ $(r2\ z)$
by *blast*
with *rel-comp-le* **obtain** y'' **where** $l1\ x \le_{R1} y''$ $y'' \le_{L2} r2\ z$ **by** *blast*
with *galois-prop1* ‹*in-dom* $(\le_{L1})$ *x*› **have** $x \;{}_{L1}\lessapprox\; y''$
by (*intro t1.left-Galois-if-Galois-right-if-half-galois-prop-right t1.left-GaloisI*)
auto
moreover from ‹*in-codom* $(\le_R)$ *z*› ‹$y'' \le_{L2} r2\ z$› **have** $y'' \;{}_{L2}\lessapprox\; z$
by (*intro t2.left-GaloisI*) *auto*
ultimately show $(({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$ $x\ z$ **by** *blast*
qed

lemma *comp-left-Galois-le-left-GaloisI*:
assumes *mono-r1*: $((\le_{R1}) \Rrightarrow_m (\le_{L1}))$ *r1*
and *half-galois-prop-left1*: $((\le_{L1})\ {}_{h}\trianglelefteq (\le_{R1}))$ *l1 r1*
and *half-galois-prop-right1*: $((\le_{R1}) \trianglelefteq_{h} (\le_{L1}))$ *r1 l1*
and *refl-R1*: *reflexive-on* (*in-codom* $(\le_{R1})$) $(\le_{R1})$
and *mono-l2*: $((\le_{L2}) \Rrightarrow_m (\le_{R2}))$ *l2*
and *refl-L2*: *reflexive-on* (*in-dom* $(\le_{L2})$) $(\le_{L2})$

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 (*intro le-relI left-GaloisI*)
fix x z **assume** $(({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))\ x\ z$
from ‹$(({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))\ x\ z$› **obtain** y **where** $x\ {}_{L1}\lessapprox\ y$ $y\ {}_{L2}\lessapprox\ z$ **by** *blast*
with *half-galois-prop-left1* **have** $l1\ x \leq_{R1} y$ $y \leq_{L2} r2\ z$ **by** *auto*
with *refl-R1 refl-L2* **have** $y \leq_{R1} y$ $y \leq_{L2} y$ **by** *auto*
show *in-codom* $(\leq_R)\ z$
proof (*intro in-codomI flip.left-relI*)
from *mono-l2* ‹$y \leq_{L2} y$› **show** $l2\ y\ {}_{R2}\lessapprox\ y$ **by** *blast*
show $y \leq_{R1} y$ $y\ {}_{L2}\lessapprox\ z$ **by** *fact+*
qed
show $x \leq_L r\ z$
proof (*intro left-relI*)
show $x\ {}_{L1}\lessapprox\ y$ $y \leq_{L2} r2\ z$ **by** *fact+*
show $r2\ z\ {}_{R1}\lessapprox\ r\ z$
proof (*intro flip.t2.left-GaloisI*)
from ‹$y \leq_{L2} y$› ‹$y \leq_{R1} y$› ‹$y \leq_{L2} r2\ z$› **have** $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2}))\ y$ $(r2\ z)$
by *blast*
with *in-codom-rel-comp-le* **have** *in-codom* $(\leq_{R1})\ (r2\ z)$ **by** *blast*
with *refl-R1* **have** $r2\ z \leq_{R1} r2\ z$ **by** *blast*
with *mono-r1* **show** *in-codom* $(\leq_{L1})\ (r\ z)$ **by** *auto*
with ‹$r2\ z \leq_{R1} r2\ z$› *half-galois-prop-right1 mono-r1*
show $r2\ z \leq_{R1} l1\ (r\ z)$ **by** (*auto intro*:
flip.t2.rel-unit-if-left-rel-if-half-galois-prop-right-if-mono-wrt-rel)
qed
qed
qed

corollary *left-Galois-eq-comp-left-GaloisI*:
assumes $((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))\ r1$
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 $((\leq_{L2}) \Rrightarrow_m (\leq_{R2}))\ l2$
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* $(\leq_R)$] $\Rrightarrow_m$ *in-codom* $(\leq_{R1})$) $r2$
and *in-codom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-codom* $(\leq_{R1})$
shows $({}_{L}\lessapprox) = (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$
using *assms*
by (*intro antisym left-Galois-le-comp-left-GaloisI comp-left-Galois-le-left-GaloisI*)
(*auto elim*!: *preorder-on-in-fieldE*
intro: *reflexive-on-if-le-pred-if-reflexive-on in-field-if-in-codom*)

corollary *left-Galois-eq-comp-left-GaloisI'*:
assumes $((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))\ r1$

and $((\le_{L1}) \trianglelefteq (\le_{R1}))$ *l1 r1*
and $((\le_{R1}) \trianglelefteq_h (\le_{L1}))$ *r1 l1*
and *preorder-on* (*in-field* $(\le_{R1})$) $(\le_{R1})$
and $((\le_{L2}) \Rrightarrow_m (\le_{R2}))$ *l2*
and $((\le_{R2})\ {}_h\!\trianglelefteq (\le_{L2}))$ *r2 l2*
and *reflexive-on* (*in-dom* $(\le_{L2})$) $(\le_{L2})$
and $((\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1})) \le ((\le_{R1}) \circ\circ (\le_{L2}))$
and *in-codom* $((\le_{L2}) \circ\circ (\le_{R1}) \circ\circ (\le_{L2})) \le$ *in-codom* $(\le_{R1})$
shows $({}_{L}\lessapprox) = (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$
using *assms* **by** (*intro left-Galois-eq-comp-left-GaloisI*
flip.mono-in-codom-left-rel-left1-if-in-codom-rel-comp-le)
auto

theorem *left-Galois-eq-comp-left-Galois-if-galois-connection-if-galois-equivalenceI'*:
assumes $((\le_{L1}) \equiv_G (\le_{R1}))$ *l1 r1*
and *preorder-on* (*in-field* $(\le_{R1})$) $(\le_{R1})$
and $((\le_{R2}) \dashv (\le_{L2}))$ *r2 l2*
and *reflexive-on* (*in-dom* $(\le_{L2})$) $(\le_{L2})$
and $((\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1})) \le ((\le_{R1}) \circ\circ (\le_{L2}))$
and *in-codom* $((\le_{L2}) \circ\circ (\le_{R1}) \circ\circ (\le_{L2})) \le$ *in-codom* $(\le_{R1})$
shows $({}_{L}\lessapprox) = (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$
using *assms* **by** (*intro left-Galois-eq-comp-left-GaloisI'*)
(*auto elim*!: *t1.galois-equivalenceE*)

corollary *left-Galois-eq-comp-left-Galois-if-galois-connection-if-galois-equivalenceI*:
assumes $((\le_{L1}) \equiv_G (\le_{R1}))$ *l1 r1*
and *preorder-on* (*in-field* $(\le_{R1})$) $(\le_{R1})$
and $((\le_{R2}) \dashv (\le_{L2}))$ *r2 l2*
and *reflexive-on* (*in-field* $(\le_{L2})$) $(\le_{L2})$
and *in-codom* $((\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1})) \le$ *in-codom* $(\le_{L2})$
and $((\le_{L2}) \circ\circ (\le_{R1}) \circ\circ (\le_{L2})) \le ((\le_{R1}) \circ\circ (\le_{L2}))$
and *in-codom* $((\le_{L2}) \circ\circ (\le_{R1}) \circ\circ (\le_{L2})) \le$ *in-codom* $(\le_{R1})$
shows $({}_{L}\lessapprox) = (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$
using *assms*
by (*intro left-Galois-eq-comp-left-Galois-if-galois-connection-if-galois-equivalenceI'*
flip.left2-right1-left2-le-left2-right1-if-right1-left2-right1-le-left2-right1)
auto

corollary *left-Galois-eq-comp-left-Galois-if-preorder-equivalenceI*:
assumes $((\le_{L1}) \equiv_{pre} (\le_{R1}))$ *l1 r1*
and $((\le_{R2}) \equiv_{pre} (\le_{L2}))$ *r2 l2*
and *in-codom* $((\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1})) \le$ *in-codom* $(\le_{L2})$
and $(\le_{L2}) \circ\circ (\le_{R1}) \circ\circ (\le_{L2}) \le (\le_{R1}) \circ\circ (\le_{L2})$
and *in-codom* $((\le_{L2}) \circ\circ (\le_{R1}) \circ\circ (\le_{L2})) \le$ *in-codom* $(\le_{R1})$
shows $({}_{L}\lessapprox) = (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$
using *assms* **by** (*intro*
left-Galois-eq-comp-left-Galois-if-galois-connection-if-galois-equivalenceI)
auto

end

end

2.4.7 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* **.**

Reflexivity

lemma *reflexive-on-in-dom-leftI*:
  assumes *galois-prop*: $((\le_{L1}) \trianglelefteq (\le_{R1}))$ *l1 r1*
  and *in-dom-L1-le*: *in-dom* $(\le_{L1}) \le$ *in-codom* $(\le_{L1})$
  and *refl-R1*: *reflexive-on* (*in-dom* $(\le_{R1})$) $(\le_{R1})$
  and *refl-L2*: *reflexive-on* (*in-dom* $(\le_{L2})$) $(\le_{L2})$
  and *mono-in-dom-l1*: ([*in-dom* $(\le_L)$] $\Rrightarrow_m$ *in-dom* $(\le_{L2})$) *l1*
  shows *reflexive-on* (*in-dom* $(\le_L)$) $(\le_L)$
proof (*rule reflexive-onI*)
  fix x **assume** *in-dom* $(\le_L)$ x
  then obtain x' **where** $x \le_L x'$ *in-dom* $(\le_{L1})$ x **by** *blast*
  show $x \le_L x$
  proof (*rule left-relI*)
    from *refl-R1* **have** *l1* $x \le_{R1}$ *l1* x
    proof (*rule reflexive-onD*)
      from ‹$x \le_L x'$› *galois-prop* **show** *in-dom* $(\le_{R1})$ (*l1* x) **by** *blast*
    qed
    then show x ${}_{L1}\lessapprox$ *l1* x
    proof (*intro t1.left-GaloisI*)
      from *galois-prop* ‹*in-dom* $(\le_{L1})$ x› ‹*l1* $x \le_{R1}$ *l1* x› **show** $x \le_{L1}$ *r1* (*l1* x)
by *blast*
    qed *blast*
    from *refl-L2* **show** *l1* $x \le_{L2}$ *l1* x
    proof (*rule reflexive-onD*)
      from *mono-in-dom-l1* ‹$x \le_L x'$› **show** *in-dom* $(\le_{L2})$ (*l1* x) **by** *blast*
    qed
    from ‹*l1* $x \le_{R1}$ *l1* x› **show** *l1* x ${}_{R1}\lessapprox$ x
    proof (*intro flip1.left-GaloisI*)
      from ‹*in-dom* $(\le_{L1})$ x› *in-dom-L1-le* **show** *in-codom* $(\le_{L1})$ x **by** *blast*
    qed
  qed
qed

lemma *reflexive-on-in-codom-leftI*:
  assumes *L1-r1-l1I*: $\bigwedge x.$ *in-dom* $(\le_{L1})$ $x \Longrightarrow l1\ x \le_{R1} l1\ x \Longrightarrow x \le_{L1} r1\ (l1\ x)$
  and *in-codom-L1-le*: *in-codom* $(\le_{L1}) \le$ *in-dom* $(\le_{L1})$
  and *refl-R1*: *reflexive-on* (*in-codom* $(\le_{R1})$) $(\le_{R1})$
  and *refl-L2*: *reflexive-on* (*in-codom* $(\le_{L2})$) $(\le_{L2})$
  and *mono-in-codom-l1*: ([*in-codom* $(\le_L)$] $\Rrightarrow_m$ *in-codom* $(\le_{L2})$) *l1*
  shows *reflexive-on* (*in-codom* $(\le_L)$) $(\le_L)$
proof (*rule reflexive-onI*)
  fix x **assume** *in-codom* $(\le_L)$ x
  then obtain x' **where** $x' \le_L x$ *in-codom* $(\le_{L1})$ x *in-codom* $(\le_{R1})$ $(l1\ x)$
    by *blast*
  show $x \le_L x$
  proof (*rule left-relI*)
    from *refl-R1* ‹*in-codom* $(\le_{R1})$ $(l1\ x)$› **have** $l1\ x \le_{R1} l1\ x$ **by** *blast*
    show $x\ {}_{L1}\lessapprox\ l1\ x$
    proof (*rule t1.left-GaloisI*)
      from *in-codom-L1-le* ‹*in-codom* $(\le_{L1})$ x› **have** *in-dom* $(\le_{L1})$ x **by** *blast*
      with ‹$l1\ x \le_{R1} l1\ x$› **show** $x \le_{L1} r1\ (l1\ x)$ **by** (*intro L1-r1-l1I*)
    qed *fact*
    from *refl-L2* **show** $l1\ x \le_{L2} l1\ x$
    proof (*rule reflexive-onD*)
      from *mono-in-codom-l1* ‹$x' \le_L x$› **show** *in-codom* $(\le_{L2})$ $(l1\ x)$ **by** *blast*
    qed
    show $l1\ x\ {}_{R1}\lessapprox\ x$ **by** (*rule flip1.left-GaloisI*) *fact+*
  qed
qed

corollary *reflexive-on-in-field-leftI*:
  assumes $((\le_{L1}) \trianglelefteq (\le_{R1}))$ *l1 r1*
  and *in-codom* $(\le_{L1})$ = *in-dom* $(\le_{L1})$
  and *reflexive-on* (*in-field* $(\le_{R1})$) $(\le_{R1})$
  and *reflexive-on* (*in-field* $(\le_{L2})$) $(\le_{L2})$
  and ([*in-field* $(\le_L)$] $\Rrightarrow_m$ *in-field* $(\le_{L2})$) *l1*
  shows *reflexive-on* (*in-field* $(\le_L)$) $(\le_L)$
proof −
  from *assms* **have** *reflexive-on* (*in-dom* $(\le_L)$) $(\le_L)$
    by (*intro reflexive-on-in-dom-leftI*)
    (*auto 0 4 intro*: *reflexive-on-if-le-pred-if-reflexive-on in-field-if-in-dom*)
  moreover from *assms* **have** *reflexive-on* (*in-codom* $(\le_L)$) $(\le_L)$
    by (*intro reflexive-on-in-codom-leftI*)
    (*auto 0 4 intro*: *reflexive-on-if-le-pred-if-reflexive-on in-field-if-in-codom*)
  ultimately show *?thesis* **by** (*auto iff*: *in-field-iff-in-dom-or-in-codom*)
qed

Transitivity

There are many similar proofs for transitivity. They slightly differ in their assumptions, particularly which of $(\le_{R1})$ and $(\le_{L2})$ 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 $((\le_{L1})\ {}_{h}\trianglelefteq\ (\le_{R1}))$ *l1 r1*
and *trans-L2*: *transitive* $(\le_{L2})$
and *R1-L2-R1-le*: $((\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1})) \le ((\le_{L2}) \circ\circ (\le_{R1}))$
shows *transitive* $(\le_{L})$
proof (*rule transitiveI*)
fix *x1 x2 x3* **assume** $x1 \le_{L} x2$ $x2 \le_{L} x3$
from ‹$x1 \le_{L} x2$› **obtain** *y1 y2* **where** $x1\ {}_{L1}\lessapprox\ y1$ $y1 \le_{L2} y2$ $y2 \le_{R1} l1\ x2$
by *blast*
from ‹$x2 \le_{L} x3$› ‹$((\le_{L1})\ {}_{h}\trianglelefteq\ (\le_{R1}))$ *l1 r1*› **obtain** *y3 y4* **where**
$l1\ x2 \le_{R1} y3$ $y3 \le_{L2} y4$ $y4 \le_{R1} l1\ x3$ *in-codom* $(\le_{L1})$ *x3* **by** *blast*
with *R1-L2-R1-le* **have** $((\le_{L2}) \circ\circ (\le_{R1}))\ (l1\ x2)\ (l1\ x3)$ **by** *blast*
then obtain *y* **where** $l1\ x2 \le_{L2} y$ $y \le_{R1} l1\ x3$ **by** *blast*
with ‹$y2 \le_{R1} l1\ x2$› *R1-L2-R1-le* **have** $((\le_{L2}) \circ\circ (\le_{R1}))\ y2\ (l1\ x3)$ **by** *blast*
then obtain y' **where** $y2 \le_{L2} y'$ $y' \le_{R1} l1\ x3$ **by** *blast*
with ‹$y1 \le_{L2} y2$› **have** $y1 \le_{L2} y'$ **using** *trans-L2* **by** *blast*
show $x1 \le_{L} x3$
proof (*rule left-relI*)
show $x1\ {}_{L1}\lessapprox\ y1$ $y1 \le_{L2} y'$ **by** *fact+*
show $y'\ {}_{R1}\lessapprox\ x3$ **by** (*rule flip1.left-GaloisI*) *fact+*
qed
qed

lemma *transitive-leftI'*:
assumes *galois-prop*: $((\le_{L1}) \trianglelefteq (\le_{R1}))$ *l1 r1*
and *trans-L2*: *transitive* $(\le_{L2})$
and *R1-L2-R1-le*: $((\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1})) \le ((\le_{R1}) \circ\circ (\le_{L2}))$
shows *transitive* $(\le_{L})$
proof (*rule transitiveI*)
fix *x1 x2 x3* **assume** $x1 \le_{L} x2$ $x2 \le_{L} x3$
from ‹$x1 \le_{L} x2$› *galois-prop* **obtain** *y1 y2* **where**
in-dom $(\le_{L1})$ *x1* $l1\ x1 \le_{R1} y1$ $y1 \le_{L2} y2$ $y2 \le_{R1} l1\ x2$ **by** *blast*
with *R1-L2-R1-le* **have** $((\le_{R1}) \circ\circ (\le_{L2}))\ (l1\ x1)\ (l1\ x2)$ **by** *blast*
then obtain *y* **where** $l1\ x1 \le_{R1} y$ $y \le_{L2} l1\ x2$ **by** *blast*
moreover from ‹$x2 \le_{L} x3$› *galois-prop* **obtain** *y3 y4* **where**
$l1\ x2 \le_{R1} y3$ $y3 \le_{L2} y4$ $y4\ {}_{R1}\lessapprox\ x3$ **by** *blast*
moreover note *R1-L2-R1-le*
ultimately have $((\le_{R1}) \circ\circ (\le_{L2}))\ (l1\ x1)\ y3$ **by** *blast*
then obtain y' **where** $l1\ x1 \le_{R1} y'$ $y' \le_{L2} y3$ **by** *blast*
with ‹$y3 \le_{L2} y4$› **have** $y' \le_{L2} y4$ **using** *trans-L2* **by** *blast*
show $x1 \le_{L} x3$
proof (*rule left-relI*)
from ‹*in-dom* $(\le_{L1})$ *x1*› ‹$l1\ x1 \le_{R1} y'$› *galois-prop* **show** $x1\ {}_{L1}\lessapprox\ y'$
by (*intro t1.left-Galois-if-Galois-right-if-half-galois-prop-right t1.left-GaloisI*)
auto
show $y' \le_{L2} y4$ **by** *fact*
from ‹$y' \le_{L2} y4$› ‹$y4\ {}_{R1}\lessapprox\ x3$› **show** $y4\ {}_{R1}\lessapprox\ x3$ **by** *blast*

```
  qed
qed
```

Preorders

```
lemma preorder-on-in-field-leftI:
  assumes ((≤L1) ⊴ (≤R1)) l1 r1
  and in-codom (≤L1) = in-dom (≤L1)
  and reflexive-on (in-field (≤R1)) (≤R1)
  and preorder-on (in-field (≤L2)) (≤L2)
  and mono-in-codom-l1: ([in-codom (≤L)] ⇛m in-codom (≤L2)) l1
  and R1-L2-R1-le: ((≤R1) ∘∘ (≤L2) ∘∘ (≤R1)) ≤ ((≤L2) ∘∘ (≤R1))
  shows preorder-on (in-field (≤L)) (≤L)
proof −
  have ([in-field (≤L)] ⇛m in-field (≤L2)) l1
  proof −
    from ‹((≤L1) ⊴ (≤R1)) l1 r1› R1-L2-R1-le
      have ([in-dom (≤L)] ⇛m in-dom (≤L2)) l1
      by (intro mono-in-dom-left-rel-left1-if-in-dom-rel-comp-le
        in-dom-right1-left2-right1-le-if-right1-left2-right1-le)
      auto
    with mono-in-codom-l1 show ?thesis by (intro dep-mono-wrt-predI) blast
  qed
  with assms show ?thesis by (intro preorder-onI)
    (auto intro: reflexive-on-in-field-leftI transitive-leftI)
qed

lemma preorder-on-in-field-leftI′:
  assumes ((≤L1) ⊴ (≤R1)) l1 r1
  and in-codom (≤L1) = in-dom (≤L1)
  and reflexive-on (in-field (≤R1)) (≤R1)
  and preorder-on (in-field (≤L2)) (≤L2)
  and mono-in-dom-l1: ([in-dom (≤L)] ⇛m in-dom (≤L2)) l1
  and R1-L2-R1-le: ((≤R1) ∘∘ (≤L2) ∘∘ (≤R1)) ≤ ((≤R1) ∘∘ (≤L2))
  shows preorder-on (in-field (≤L)) (≤L)
proof −
  have ([in-field (≤L)] ⇛m in-field (≤L2)) l1
  proof −
    from ‹((≤L1) ⊴ (≤R1)) l1 r1› R1-L2-R1-le
      have ([in-codom (≤L)] ⇛m in-codom (≤L2)) l1
      by (intro mono-in-codom-left-rel-left1-if-in-codom-rel-comp-le
        in-codom-right1-left2-right1-le-if-right1-left2-right1-le)
      auto
    with mono-in-dom-l1 show ?thesis by (intro dep-mono-wrt-predI) blast
  qed
  with assms show ?thesis by (intro preorder-onI)
    (auto intro: reflexive-on-in-field-leftI transitive-leftI′)
qed
```

Symmetry

lemma *symmetric-leftI*:
assumes $((\le_{L1}) \trianglelefteq (\le_{R1}))$ *l1 r1*
and *in-codom* $(\le_{L1})$ = *in-dom* $(\le_{L1})$
and *symmetric* $(\le_{R1})$
and *symmetric* $(\le_{L2})$
shows *symmetric* $(\le_{L})$
proof –
from *assms* **have** $(\gtrapprox_{R1}) = ({}_{L1}\lessapprox)$ **by** (*intro*
t1.ge-Galois-right-eq-left-Galois-if-symmetric-if-in-codom-eq-in-dom-if-galois-prop)
moreover then have $({}_{R1}\lessapprox) = (\gtrapprox_{L1})$
by (*subst rel-inv-eq-iff-eq*[*symmetric*]) *simp*
ultimately show *?thesis* **using** *assms* **unfolding** *left-rel-eq-comp*
by (*subst symmetric-iff-rel-inv-eq-self*) (*simp add*: *rel-comp-assoc*)
qed

lemma *partial-equivalence-rel-leftI*:
assumes $((\le_{L1}) \trianglelefteq (\le_{R1}))$ *l1 r1*
and *in-codom* $(\le_{L1})$ = *in-dom* $(\le_{L1})$
and *symmetric* $(\le_{R1})$
and *partial-equivalence-rel* $(\le_{L2})$
and $((\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1})) \le ((\le_{L2}) \circ\circ (\le_{R1}))$
shows *partial-equivalence-rel* $(\le_{L})$
using *assms* **by** (*intro partial-equivalence-relI transitive-leftI symmetric-leftI*)
auto

lemma *partial-equivalence-rel-leftI'*:
assumes $((\le_{L1}) \trianglelefteq (\le_{R1}))$ *l1 r1*
and *in-codom* $(\le_{L1})$ = *in-dom* $(\le_{L1})$
and *symmetric* $(\le_{R1})$
and *partial-equivalence-rel* $(\le_{L2})$
and $((\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1})) \le ((\le_{R1}) \circ\circ (\le_{L2}))$
shows *partial-equivalence-rel* $(\le_{L})$
using *assms* **by** (*intro partial-equivalence-relI transitive-leftI' symmetric-leftI*)
auto

end

end

2.4.8 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* **.**

Unit

Inflationary **lemma** *inflationary-on-in-dom-unitI*:
assumes $((\le_{R1}) \Rrightarrow_m (\le_{L1}))$ *r1*
and $((\le_{L1})\ {}_h\trianglelefteq\ (\le_{R1}))$ *l1 r1*
and *inflationary-unit1*: *inflationary-on* (*in-dom* $(\le_{L1})$) $(\le_{L1})$ η_1
and *inflationary-counit1*: *inflationary-on* (*in-codom* $(\le_{R1})$) $(\le_{R1})$ ε_1
and *refl-R1*: *reflexive-on* (*in-dom* $(\le_{R1})$) $(\le_{R1})$
and *inflationary-unit2*: *inflationary-on* (*in-dom* $(\le_{L2})$) $(\le_{L2})$ η_2
and *refl-L2*: *reflexive-on* (*in-dom* $(\le_{L2})$) $(\le_{L2})$
and *mono-in-dom-l1*: ([*in-dom* $(\le_L)$] $\Rrightarrow_m$ *in-dom* $(\le_{L2})$) *l1*
and *in-codom-rel-comp-le*: *in-codom* $((\le_{L2})\ \circ\circ\ (\le_{R1})\ \circ\circ\ (\le_{L2}))$ $\le$ *in-codom* $((\le_{R1}))$
shows *inflationary-on* (*in-dom* $(\le_L)$) $(\le_L)$ η
proof (*rule inflationary-onI*)
fix *x* **assume** *in-dom* $(\le_L)$ *x*
show $x \le_L \eta\ x$
proof (*rule left-relI*)
from ‹*in-dom* $(\le_L)$ *x*› ‹$((\le_{L1})\ {}_h\trianglelefteq\ (\le_{R1}))$ *l1 r1*› **have** *in-dom* $(\le_{R1})$ (*l1 x*) **by** *blast*
with *refl-R1* **have** *l1 x* $\le_{R1}$ *l1 x* **by** *blast*
moreover from ‹*in-dom* $(\le_L)$ *x*› **have** *in-dom* $(\le_{L1})$ *x* **by** *blast*
moreover note *inflationary-unit1*
ultimately show $x\ {}_{L1}\lessapprox$ *l1 x* **by** (*intro t1.left-GaloisI*) *auto*
from ‹*in-dom* $(\le_L)$ *x*› *mono-in-dom-l1* **have** *in-dom* $(\le_{L2})$ (*l1 x*) **by** *blast*
with *inflationary-unit2* **show** *l1 x* $\le_{L2}$ *r2* (*l x*) **by** *auto*
show *r2* (*l x*) ${}_{R1}\lessapprox$ η *x*
proof (*rule flip.t2.left-GaloisI*)
from *refl-L2* ‹*in-dom* $(\le_{L2})$ (*l1 x*)› **have** *l1 x* $\le_{L2}$ *l1 x* **by** *blast*
with *in-codom-rel-comp-le* ‹*l1 x* $\le_{R1}$ *l1 x*› ‹*l1 x* $\le_{L2}$ *r2* (*l x*)›
have *in-codom* $(\le_{R1})$ (*r2* (*l x*)) **by** *blast*
with ‹$((\le_{R1}) \Rrightarrow_m (\le_{L1}))$ *r1*› **show** *in-codom* $(\le_{L1})$ (η *x*)
by (*auto intro*: *in-codom-if-rel-if-dep-mono-wrt-rel*)
from ‹*in-codom* $(\le_{R1})$ (*r2* (*l x*))› *inflationary-counit1*
show *r2* (*l x*) $\le_{R1}$ *l1* (η *x*) **by** *auto*
qed
qed
qed

lemma *inflationary-on-in-codom-unitI*:
assumes $((\le_{R1}) \Rrightarrow_m (\le_{L1}))$ *r1*
and *inflationary-unit1*: *inflationary-on* (*in-codom* $(\le_{L1})$) $(\le_{L1})$ 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* ($\leq_L$)] $\Rrightarrow_m$ *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 (*rule inflationary-onI*)
fix *x* **assume** *in-codom* ($\leq_L$) *x*
show $x \leq_L \eta\ x$
proof (*rule left-relI*)
from ‹*in-codom* ($\leq_L$) *x*› **have** *in-codom* ($\leq_{L1}$) *x in-codom* ($\leq_{R1}$) (*l1 x*) **by** *blast+*
with *inflationary-unit1* **show** $x\ {}_{L1}\lessapprox\ l1\ x$ **by** (*intro t1.left-GaloisI*) *auto*
from *mono-in-codom-l1* ‹*in-codom* ($\leq_L$) *x*› **have** *in-codom* ($\leq_{L2}$) (*l1 x*) **by** *blast*
with *inflationary-unit2* **show** $l1\ x \leq_{L2} r2\ (l\ x)$ **by** *auto*
show $r2\ (l\ x)\ {}_{R1}\lessapprox\ \eta\ x$
proof (*rule flip.t2.left-GaloisI*)
from *refl-L2* ‹*in-codom* ($\leq_{L2}$) (*l1 x*)› **have** $l1\ x \leq_{L2} l1\ x$ **by** *blast*
moreover from *refl-R1* ‹*in-codom* ($\leq_{R1}$) (*l1 x*)› **have** $l1\ x \leq_{R1} l1\ x$ **by** *blast*
moreover note *in-codom-rel-comp-le* ‹$l1\ x \leq_{L2} r2\ (l\ x)$›
ultimately have *in-codom* ($\leq_{R1}$) (*r2* (*l x*)) **by** *blast*
with ‹(($\leq_{R1}$) $\Rrightarrow_m$ ($\leq_{L1}$)) *r1*› **show** *in-codom* ($\leq_{L1}$) (η *x*)
by (*auto intro*: *in-codom-if-rel-if-dep-mono-wrt-rel*)
from ‹*in-codom* ($\leq_{R1}$) (*r2* (*l x*))› *inflationary-counit1*
show $r2\ (l\ x) \leq_{R1} l1\ (\eta\ x)$ **by** *auto*
qed
qed
qed

corollary *inflationary-on-in-field-unitI*:
assumes (($\leq_{R1}$) $\Rrightarrow_m$ ($\leq_{L1}$)) *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* ($\leq_L$)] $\Rrightarrow_m$ *in-dom* ($\leq_{L2}$)) *l1*
and ([*in-codom* ($\leq_L$)] $\Rrightarrow_m$ *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 –
from *assms* **have** *inflationary-on* (*in-dom* ($\leq_L$)) ($\leq_L$) η
by (*intro inflationary-on-in-dom-unitI*)
(*auto intro*: *inflationary-on-if-le-pred-if-inflationary-on*
reflexive-on-if-le-pred-if-reflexive-on in-field-if-in-dom)

moreover from *assms* **have** *inflationary-on* (*in-codom* $(\le_{L})$) $(\le_{L})$ η
by (*intro inflationary-on-in-codom-unitI*)
(*auto intro*: *inflationary-on-if-le-pred-if-inflationary-on*
reflexive-on-if-le-pred-if-reflexive-on in-field-if-in-codom)
ultimately show *?thesis* **by** (*auto iff*: *in-field-iff-in-dom-or-in-codom*)
qed

Deflationary

lemma *deflationary-on-in-dom-unitI*:
assumes $((\le_{L1}) \Rrightarrow_m (\le_{R1}))$ *l1* $((\le_{R1}) \Rrightarrow_m (\le_{L1}))$ *r1*
and *refl-L1*: *reflexive-on* (*in-dom* $(\le_{L1})$) $(\le_{L1})$
and *in-dom-R1-le-in-codom-R1*: *in-dom* $(\le_{R1}) \le$ *in-codom* $(\le_{R1})$
and *deflationary-L2*: *deflationary-on* (*in-dom* $(\le_{L2})$) $(\le_{L2})$ η_2
and *refl-L2*: *reflexive-on* (*in-dom* $(\le_{L2})$) $(\le_{L2})$
and *mono-in-dom-l1*: ([*in-dom* $(\le_{L})$] $\Rrightarrow_m$ *in-dom* $(\le_{L2})$) *l1*
and *in-dom-rel-comp-le*: *in-dom* $((\le_{L2}) \circ\circ (\le_{R1}) \circ\circ (\le_{L2})) \le$ *in-dom* $((\le_{R1}))$
shows *deflationary-on* (*in-dom* $(\le_{L})$) $(\le_{L})$ η
proof (*rule deflationary-onI*)
fix x **assume** *in-dom* $(\le_{L})$ x
show $\eta\ x \le_{L} x$
proof (*rule left-relI*)
from *refl-L1* ‹*in-dom* $(\le_{L})$ x› **have** $x \le_{L1} x$ **by** *blast*
moreover with ‹$((\le_{L1}) \Rrightarrow_m (\le_{R1}))$ *l1*› **have** *l1* $x \le_{R1}$ *l1* x **by** *blast*
ultimately show *l1* x ${}_{R1}\lessapprox$ x **by** *auto*
from *mono-in-dom-l1* ‹*in-dom* $(\le_{L})$ x› **have** *in-dom* $(\le_{L2})$ (*l1* x) **by** *blast*
with *deflationary-L2* **show** *r2* $(l\ x) \le_{L2}$ *l1* x **by** *auto*
show $\eta\ x$ ${}_{L1}\lessapprox$ *r2* $(l\ x)$
proof (*rule t1.left-GaloisI*)
from *refl-L2* ‹*in-dom* $(\le_{L2})$ (*l1* x)› **have** *l1* $x \le_{L2}$ *l1* x **by** *blast*
with *in-dom-rel-comp-le* ‹*r2* $(l\ x) \le_{L2}$ *l1* x› ‹*l1* $x \le_{R1}$ *l1* x›
have *in-dom* $(\le_{R1})$ (*r2* $(l\ x)$) **by** *blast*
with ‹$((\le_{R1}) \Rrightarrow_m (\le_{L1}))$ *r1*› **have** *in-dom* $(\le_{L1})$ $(\eta\ x)$
by (*auto intro*: *in-dom-if-rel-if-dep-mono-wrt-rel*)
with *refl-L1* **show** $\eta\ x \le_{L1}$ *r1* (*r2* $(l\ x)$)
by (*auto intro*: *in-field-if-in-codom*)
from ‹*in-dom* $(\le_{R1})$ (*r2* $(l\ x)$)› *in-dom-R1-le-in-codom-R1*
show *in-codom* $(\le_{R1})$ (*r2* $(l\ x)$) **by** *blast*
qed
qed
qed

lemma *deflationary-on-in-codom-unitI*:
assumes $((\le_{L1}) \Rrightarrow_m (\le_{R1}))$ *l1* $((\le_{R1}) \Rrightarrow_m (\le_{L1}))$ *r1*
and *refl-L1*: *reflexive-on* (*in-codom* $(\le_{L1})$) $(\le_{L1})$
and *in-dom-R1-le-in-codom-R1*: *in-dom* $(\le_{R1}) \le$ *in-codom* $(\le_{R1})$
and *deflationary-L2*: *deflationary-on* (*in-codom* $(\le_{L2})$) $(\le_{L2})$ η_2
and *refl-L2*: *reflexive-on* (*in-codom* $(\le_{L2})$) $(\le_{L2})$
and *mono-in-codom-l1*: ([*in-codom* $(\le_{L})$] $\Rrightarrow_m$ *in-codom* $(\le_{L2})$) *l1*
and *in-dom-rel-comp-le*: *in-dom* $((\le_{L2}) \circ\circ (\le_{R1}) \circ\circ (\le_{L2})) \le$ *in-dom* $((\le_{R1}))$

shows *deflationary-on* (*in-codom* $(\leq_L)$) $(\leq_L)$ η
proof (*rule deflationary-onI*)
fix x **assume** *in-codom* $(\leq_L)$ x
show $\eta\ x \leq_L x$
proof (*rule left-relI*)
from *refl-L1* ‹*in-codom* $(\leq_L)$ x› **have** $x \leq_{L1} x$ **by** *blast*
moreover with ‹$((\leq_{L1}) \Rrightarrow_m (\leq_{R1}))$ *l1*› **have** *l1* $x \leq_{R1}$ *l1* x **by** *blast*
ultimately show *l1* x ${}_{R1}\lessapprox$ x **by** *auto*
from *mono-in-codom-l1* ‹*in-codom* $(\leq_L)$ x› **have** *in-codom* $(\leq_{L2})$ (*l1* x) **by** *blast*
with *deflationary-L2* **show** *r2* (*l* x) $\leq_{L2}$ *l1* x **by** *auto*
show $\eta\ x$ ${}_{L1}\lessapprox$ *r2* (*l* x)
proof (*rule t1.left-GaloisI*)
from *refl-L2* ‹*in-codom* $(\leq_{L2})$ (*l1* x)› **have** *l1* $x \leq_{L2}$ *l1* x **by** *blast*
with *in-dom-rel-comp-le* ‹*r2* (*l* x) $\leq_{L2}$ *l1* x› ‹*l1* $x \leq_{R1}$ *l1* x›
have *in-dom* $(\leq_{R1})$ (*r2* (*l* x)) **by** *blast*
with *in-dom-R1-le-in-codom-R1* **show** *in-codom* $(\leq_{R1})$ (*r2* (*l* x)) **by** *blast*
with ‹$((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))$ *r1*› **have** *in-codom* $(\leq_{L1})$ (η x)
by (*auto intro*: *in-codom-if-rel-if-dep-mono-wrt-rel*)
with *refl-L1* **show** $\eta\ x \leq_{L1}$ *r1* (*r2* (*l* x)) **by** *auto*
qed
qed
qed

corollary *deflationary-on-in-field-unitI*:
assumes $((\leq_{L1}) \Rrightarrow_m (\leq_{R1}))$ *l1* $((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))$ *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* $(\leq_L)$] $\Rrightarrow_m$ *in-dom* $(\leq_{L2})$) *l1*
and ([*in-codom* $(\leq_L)$] $\Rrightarrow_m$ *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 −
from *assms* **have** *deflationary-on* (*in-dom* $(\leq_L)$) $(\leq_L)$ η
by (*intro deflationary-on-in-dom-unitI*)
(*auto intro*: *deflationary-on-if-le-pred-if-deflationary-on*
reflexive-on-if-le-pred-if-reflexive-on in-field-if-in-dom)
moreover from *assms* **have** *deflationary-on* (*in-codom* $(\leq_L)$) $(\leq_L)$ η
by (*intro deflationary-on-in-codom-unitI*)
(*auto intro*: *deflationary-on-if-le-pred-if-deflationary-on*
reflexive-on-if-le-pred-if-reflexive-on in-field-if-in-codom)
ultimately show *?thesis* **by** (*auto iff*: *in-field-iff-in-dom-or-in-codom*)
qed

Relational Equivalence

corollary *rel-equivalence-on-in-field-unitI*:
assumes $((\leq_{L1}) \Rrightarrow_m (\leq_{R1}))$ *l1* $((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))$ *r1*

and $((\le_{L1})\ {}_h\trianglelefteq\ (\le_{R1}))$ *l1 r1*
and *inflationary-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$ η_1
and *inflationary-on* (*in-codom* $(\le_{R1})$) $(\le_{R1})$ ε_1
and *reflexive-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$
and *reflexive-on* (*in-field* $(\le_{R1})$) $(\le_{R1})$
and *rel-equivalence-on* (*in-field* $(\le_{L2})$) $(\le_{L2})$ η_2
and *reflexive-on* (*in-field* $(\le_{L2})$) $(\le_{L2})$
and ([*in-dom* $(\le_L)$] $\Rrightarrow_m$ *in-dom* $(\le_{L2})$) *l1*
and ([*in-codom* $(\le_L)$] $\Rrightarrow_m$ *in-codom* $(\le_{L2})$) *l1*
and *in-dom* $((\le_{L2}) \circ\circ (\le_{R1}) \circ\circ (\le_{L2}))$ $\le$ *in-dom* $((\le_{R1}))$
and *in-codom* $((\le_{L2}) \circ\circ (\le_{R1}) \circ\circ (\le_{L2}))$ $\le$ *in-codom* $((\le_{R1}))$
shows *rel-equivalence-on* (*in-field* $(\le_L)$) $(\le_L)$ η
using *assms* **by** (*intro rel-equivalence-onI*
inflationary-on-in-field-unitI deflationary-on-in-field-unitI)
(*auto simp only*: *in-codom-eq-in-dom-if-reflexive-on-in-field*)

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* $\equiv \varepsilon$
by (*simp-all only*: *order-functors.flip-counit-eq-unit*)

lemma *order-equivalenceI*:
assumes $((\le_{L1}) \Rrightarrow_m (\le_{R1}))$ *l1* $((\le_{R1}) \Rrightarrow_m (\le_{L1}))$ *r1*
and $((\le_{L1})\ {}_h\trianglelefteq\ (\le_{R1}))$ *l1 r1*
and *inflationary-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$ η_1
and *rel-equiv-counit1*: *rel-equivalence-on* (*in-field* $(\le_{R1})$) $(\le_{R1})$ ε_1
and *reflexive-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$
and *reflexive-on* (*in-field* $(\le_{R1})$) $(\le_{R1})$
and $((\le_{R2}) \Rrightarrow_m (\le_{L2}))$ *r2* $((\le_{L2}) \Rrightarrow_m (\le_{R2}))$ *l2*
and $((\le_{R2})\ {}_h\trianglelefteq\ (\le_{L2}))$ *r2 l2*
and *rel-equiv-unit2*: *rel-equivalence-on* (*in-field* $(\le_{L2})$) $(\le_{L2})$ η_2
and *inflationary-on* (*in-field* $(\le_{R2})$) $(\le_{R2})$ ε_2
and *reflexive-on* (*in-field* $(\le_{L2})$) $(\le_{L2})$
and *reflexive-on* (*in-field* $(\le_{R2})$) $(\le_{R2})$
and *middle-compatible*: *middle-compatible-codom*
shows $((\le_L) \equiv_o (\le_R))$ *l r*
proof (*rule order-equivalenceI*)

show $((\le_L) \Rrightarrow_m (\le_R))$ *l* **using** *rel-equiv-unit2* ‹$((\le_{L1})\ {}_h\trianglelefteq\ (\le_{R1}))$ *l1 r1*›
‹$((\le_{L2}) \Rrightarrow_m (\le_{R2}))$ *l2*› *middle-compatible*
by (*intro mono-wrt-rel-leftI*) *auto*
show $((\le_R) \Rrightarrow_m (\le_L))$ *r* **using** *rel-equiv-counit1* ‹$((\le_{R2})\ {}_h\trianglelefteq\ (\le_{L2}))$ *r2 l2*›
‹$((\le_{R1}) \Rrightarrow_m (\le_{L1}))$ *r1*› *middle-compatible*
by (*intro flip.mono-wrt-rel-leftI*)
(*auto intro*: *inflationary-on-if-le-pred-if-inflationary-on*
in-field-if-in-codom)
from *middle-compatible* **have** *in-dom-rel-comp-les*:
in-dom $((\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1})) \le$ *in-dom* $(\le_{L2})$
in-dom $((\le_{L2}) \circ\circ (\le_{R1}) \circ\circ (\le_{L2})) \le$ *in-dom* $((\le_{R1}))$
by (*auto intro*: *in-dom-right1-left2-right1-le-if-right1-left2-right1-le*
flip.in-dom-right1-left2-right1-le-if-right1-left2-right1-le)
moreover then have ([*in-dom* $(\le_L)$] $\Rrightarrow_m$ *in-dom* $(\le_{L2})$) *l1*
and ([*in-codom* $(\le_L)$] $\Rrightarrow_m$ *in-codom* $(\le_{L2})$) *l1*
using ‹$((\le_{L1})\ {}_h\trianglelefteq\ (\le_{R1}))$ *l1 r1*› *middle-compatible*
by (*auto intro*: *mono-in-dom-left-rel-left1-if-in-dom-rel-comp-le*
mono-in-codom-left-rel-left1-if-in-codom-rel-comp-le)
ultimately show *rel-equivalence-on* (*in-field* $(\le_L)$) $(\le_L)$ η
using *assms* **by** (*intro rel-equivalence-on-in-field-unitI*)
(*auto intro*: *inflationary-on-if-le-pred-if-inflationary-on*
intro!: *in-field-if-in-codom*)
note *in-dom-rel-comp-les*
moreover then have ([*in-dom* $(\le_R)$] $\Rrightarrow_m$ *in-dom* $(\le_{R1})$) *r2*
and ([*in-codom* $(\le_R)$] $\Rrightarrow_m$ *in-codom* $(\le_{R1})$) *r2*
using ‹$((\le_{R2})\ {}_h\trianglelefteq\ (\le_{L2}))$ *r2 l2*› *middle-compatible*
by (*auto intro*!: *flip.mono-in-dom-left-rel-left1-if-in-dom-rel-comp-le*
flip.mono-in-codom-left-rel-left1-if-in-codom-rel-comp-le)
ultimately show *rel-equivalence-on* (*in-field* $(\le_R)$) $(\le_R)$ ε
using *assms* **by** (*intro flip.rel-equivalence-on-in-field-unitI*)
(*auto intro*: *inflationary-on-if-le-pred-if-inflationary-on*
intro!: *in-field-if-in-codom*)
qed

corollary *order-equivalence-if-order-equivalenceI*:
assumes $((\le_{L1}) \equiv_o (\le_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$
and *transitive* $(\le_{R1})$
and $((\le_{L2}) \equiv_o (\le_{R2}))$ *l2 r2*
and *transitive* $(\le_{L2})$
and *reflexive-on* (*in-field* $(\le_{R2})$) $(\le_{R2})$
and *middle-compatible-codom*
shows $((\le_L) \equiv_o (\le_R))$ *l r*
using *assms* **by** (*intro order-equivalenceI*) (*auto*
elim!: *t1.order-equivalenceE t2.order-equivalenceE rel-equivalence-onE*
intro!: *reflexive-on-in-field-if-transitive-if-rel-equivalence-on*
t1.half-galois-prop-left-left-right-if-transitive-if-deflationary-on-if-mono-wrt-rel
flip.t1.half-galois-prop-left-left-right-if-transitive-if-deflationary-on-if-mono-wrt-rel
intro: *deflationary-on-if-le-pred-if-deflationary-on in-field-if-in-codom*)

corollary *order-equivalence-if-galois-equivalenceI*:
assumes $((\le_{L1}) \equiv_G (\le_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$
and *reflexive-on* (*in-field* $(\le_{R1})$) $(\le_{R1})$
and $((\le_{L2}) \equiv_G (\le_{R2}))$ *l2 r2*
and *reflexive-on* (*in-field* $(\le_{L2})$) $(\le_{L2})$
and *reflexive-on* (*in-field* $(\le_{R2})$) $(\le_{R2})$
and *middle-compatible-codom*
shows $((\le_L) \equiv_o (\le_R))$ *l r*
using *assms* **by** (*intro order-equivalenceI*)
(*auto elim*!: *t1.galois-equivalenceE t2.galois-equivalenceE*
intro!: *t1.inflationary-on-unit-if-reflexive-on-if-galois-equivalence*
flip.t1.inflationary-on-unit-if-reflexive-on-if-galois-equivalence
t2.rel-equivalence-on-unit-if-reflexive-on-if-galois-equivalence
flip.t2.rel-equivalence-on-unit-if-reflexive-on-if-galois-equivalence)

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* **.**

lemma *preorder-galois-connection-if-galois-equivalenceI*:
assumes $((\le_{L1}) \equiv_G (\le_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$
and *preorder-on* (*in-field* $(\le_{R1})$) $(\le_{R1})$
and $((\le_{L2}) \equiv_G (\le_{R2}))$ *l2 r2*
and *preorder-on* (*in-field* $(\le_{L2})$) $(\le_{L2})$
and *reflexive-on* (*in-field* $(\le_{R2})$) $(\le_{R2})$
and *middle-compatible-codom*
shows $((\le_L) \dashv_{pre} (\le_R))$ *l r*
using *assms* **by** (*intro preorder-galois-connectionI*)
(*auto elim*!: *t1.galois-equivalenceE t2.galois-equivalenceE*
intro!: *galois-connection-left-right-if-galois-equivalenceI*

preorder-on-in-field-leftI flip.preorder-on-in-field-leftI mono-in-codom-left-rel-left1-if-in-codom-rel-comp-le flip.mono-in-codom-left-rel-left1-if-in-codom-rel-comp-le in-codom-eq-in-dom-if-reflexive-on-in-field)

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*
using *assms* **by** (*intro preorder-galois-connection-if-galois-equivalenceI*)
auto

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 −
from *assms* **have** $((\leq_L) \dashv_{pre} (\leq_R))$ *l r*
by (*intro preorder-galois-connection-if-galois-equivalenceI*) *auto*
with *assms* **show** *?thesis* **by** (*intro preorder-equivalence-if-galois-equivalenceI*)
(*auto intro*!: *galois-equivalence-if-galois-equivalenceI preorder-galois-connection-if-galois-equivalenceI*)
qed

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*
using *assms* **by** (*intro preorder-equivalence-if-galois-equivalenceI*) *auto*

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*
using *assms* **by** (*intro partial-equivalence-rel-equivalence-if-galois-equivalenceI galois-equivalence-if-galois-equivalenceI partial-equivalence-rel-leftI flip.partial-equivalence-rel-leftI in-codom-eq-in-dom-if-partial-equivalence-rel*)
auto

Simplification of Galois relator **theorem** *left-Galois-eq-comp-left-GaloisI*:

assumes $((\le_{L1}) \equiv_{pre} (\le_{R1}))$ *l1 r1*
and $((\le_{R2}) \dashv_{pre} (\le_{L2}))$ *r2 l2*
and *middle-compatible-codom*
shows $({}_{L}\lessapprox) = (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$
using *assms* **by** (*intro left-Galois-eq-comp-left-Galois-if-galois-connection-if-galois-equivalenceI*)
auto

For theorems with weaker assumptions, see $[\![((\le_{R1}) \Rrightarrow_m (\le_{L1}))$ *r1*; *t1.galois-prop l1 r1*; *flip.t2.half-galois-prop-right*; *preorder-on* (*in-field* $(\le_{R1}))$ $(\le_{R1})$; $((\le_{L2}) \Rrightarrow_m (\le_{R2}))$ *l2*; *flip.t1.half-galois-prop-left*; *reflexive-on* (*in-dom* $(\le_{L2}))$ $(\le_{L2})$; $(\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1}) \le (\le_{R1}) \circ\circ (\le_{L2})$; *in-codom* $((\le_{L2}) \circ\circ (\le_{R1}) \circ\circ (\le_{L2})) \le$ *in-codom* $(\le_{R1})]\!] \Longrightarrow$ *flip.right-Galois* = *flip.t2.right-Galois* $\circ\circ$ *flip.t1.right-Galois*

$[\![$*t1.galois-equivalence*; *preorder-on* (*in-field* $(\le_{R1}))$ $(\le_{R1})$; *flip.t1.galois-connection*; *reflexive-on* (*in-field* $(\le_{L2}))$ $(\le_{L2})$; *in-codom* $((\le_{R1}) \circ\circ (\le_{L2}) \circ\circ (\le_{R1})) \le$ *in-codom* $(\le_{L2})$; $(\le_{L2}) \circ\circ (\le_{R1}) \circ\circ (\le_{L2}) \le (\le_{R1}) \circ\circ (\le_{L2})$; *in-codom* $((\le_{L2}) \circ\circ (\le_{R1}) \circ\circ (\le_{L2})) \le$ *in-codom* $(\le_{R1})]\!] \Longrightarrow$ *flip.right-Galois* = *flip.t2.right-Galois* $\circ\circ$ *flip.t1.right-Galois*.

Simplification of Compatibility Assumption See *Transport.Transport-Compositions-Gener*

end

end

2.5 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

2.6 Reflexive Relator

theory *Reflexive-Relator*
imports

Galois-Equivalences
Galois-Relator
begin

definition *Refl-Rel* $R\ x\ y \equiv R\ x\ x \wedge R\ y\ y \wedge R\ x\ y$

bundle *Refl-Rel-syntax* **begin notation** *Refl-Rel* ((-$^{\oplus}$) [*1000*]) **end**
bundle *no-Refl-Rel-syntax* **begin no-notation** *Refl-Rel* ((-$^{\oplus}$) [*1000*]) **end**
unbundle *Refl-Rel-syntax*

lemma *Refl-RelI* [*intro*]:
assumes $R\ x\ x$
and $R\ y\ y$
and $R\ x\ y$
shows $R^{\oplus}\ x\ y$
using *assms* **unfolding** *Refl-Rel-def* **by** *blast*

lemma *Refl-Rel-selfI* [*intro*]:
assumes $R\ x\ x$
shows $R^{\oplus}\ x\ x$
using *assms* **by** *blast*

lemma *Refl-RelE* [*elim*]:
assumes $R^{\oplus}\ x\ y$
obtains $R\ x\ x$ $R\ y\ y$ $R\ x\ y$
using *assms* **unfolding** *Refl-Rel-def* **by** *blast*

lemma *Refl-Rel-reflexive-on-in-field* [*iff*]:
reflexive-on (*in-field* $R^{\oplus}$) $R^{\oplus}$
by (*rule reflexive-onI*) *auto*

lemma *Refl-Rel-le-self* [*iff*]: $R^{\oplus} \leq R$ **by** *blast*

lemma *Refl-Rel-eq-self-if-reflexive-on* [*simp*]:
assumes *reflexive-on* (*in-field* R) R
shows $R^{\oplus} = R$
using *assms* **by** *blast*

lemma *reflexive-on-in-field-if-Refl-Rel-eq-self*:
assumes $R^{\oplus} = R$
shows *reflexive-on* (*in-field* R) R
by (*fact Refl-Rel-reflexive-on-in-field*[*of* R, *simplified assms*])

corollary *Refl-Rel-eq-self-iff-reflexive-on*:
$R^{\oplus} = R \longleftrightarrow$ *reflexive-on* (*in-field* R) R
using *Refl-Rel-eq-self-if-reflexive-on reflexive-on-in-field-if-Refl-Rel-eq-self*
by *blast*

lemma *Refl-Rel-Refl-Rel-eq* [*simp*]: $(R^{\oplus})^{\oplus} = R^{\oplus}$

by (*intro ext*) *auto*

lemma *rel-inv-Refl-Rel-eq* [*simp*]: $(R^{\oplus})^{-1} = (R^{-1})^{\oplus}$
by (*intro ext iffI Refl-RelI rel-invI*) *auto*

lemma *Refl-Rel-transitive-onI* [*intro*]:
assumes *transitive-on* $(P :: 'a \Rightarrow bool)$ $(R :: 'a \Rightarrow -)$
shows *transitive-on* P $R^{\oplus}$
using *assms* **by** (*intro transitive-onI*) (*blast dest*: *transitive-onD*)

corollary *Refl-Rel-transitiveI* [*intro*]:
assumes *transitive* R
shows *transitive* $R^{\oplus}$
using *assms* **by** *blast*

corollary *Refl-Rel-preorder-onI*:
assumes *transitive-on* P R
and $P \leq$ *in-field* $R^{\oplus}$
shows *preorder-on* P $R^{\oplus}$
using *assms* **by** (*intro preorder-onI*
reflexive-on-if-le-pred-if-reflexive-on[**where** *?P=in-field* $R^{\oplus}$ **and** *?P'=P*])
auto

corollary *Refl-Rel-preorder-on-in-fieldI* [*intro*]:
assumes *transitive* R
shows *preorder-on* (*in-field* $R^{\oplus}$) $R^{\oplus}$
using *assms* **by** (*intro Refl-Rel-preorder-onI*) *auto*

lemma *Refl-Rel-symmetric-onI* [*intro*]:
assumes *symmetric-on* $(P :: 'a \Rightarrow bool)$ $(R :: 'a \Rightarrow -)$
shows *symmetric-on* P $R^{\oplus}$
using *assms* **by** (*intro symmetric-onI*) (*auto dest*: *symmetric-onD*)

lemma *Refl-Rel-symmetricI* [*intro*]:
assumes *symmetric* R
shows *symmetric* $R^{\oplus}$
using *assms* **by** (*fold symmetric-on-in-field-iff-symmetric*)
(*blast intro*: *symmetric-on-if-le-pred-if-symmetric-on*)

lemma *Refl-Rel-partial-equivalence-rel-onI* [*intro*]:
assumes *partial-equivalence-rel-on* $(P :: 'a \Rightarrow bool)$ $(R :: 'a \Rightarrow -)$
shows *partial-equivalence-rel-on* P $R^{\oplus}$
using *assms* **by** (*intro partial-equivalence-rel-onI Refl-Rel-transitive-onI*
Refl-Rel-symmetric-onI) *auto*

lemma *Refl-Rel-partial-equivalence-relI* [*intro*]:
assumes *partial-equivalence-rel* R
shows *partial-equivalence-rel* $R^{\oplus}$
using *assms*

by (*intro partial-equivalence-relI Refl-Rel-transitiveI Refl-Rel-symmetricI*) *auto*

lemma *Refl-Rel-app-leftI*:
assumes $R\ (f\ x)\ y$
and *in-field* $S^{\oplus}\ x$
and *in-field* $R^{\oplus}\ y$
and $(S \Rrightarrow_m R)\ f$
shows $R^{\oplus}\ (f\ x)\ y$
proof (*rule Refl-RelI*)
from ‹*in-field* $R^{\oplus}\ y$› **show** $R\ y\ y$ **by** *blast*
from ‹*in-field* $S^{\oplus}\ x$› **have** $S\ x\ x$ **by** *blast*
with ‹$(S \Rrightarrow_m R)\ f$› **show** $R\ (f\ x)\ (f\ x)$ **by** *blast*
qed *fact*

corollary *Refl-Rel-app-rightI*:
assumes $R\ x\ (f\ y)$
and *in-field* $S^{\oplus}\ y$
and *in-field* $R^{\oplus}\ x$
and $(S \Rrightarrow_m R)\ f$
shows $R^{\oplus}\ x\ (f\ y)$
proof −
from *assms* **have** $(R^{-1})^{\oplus}\ (f\ y)\ x$ **by** (*intro Refl-Rel-app-leftI*[**where** $?S{=}S^{-1}$])
(*auto simp flip*: *rel-inv-Refl-Rel-eq*)
then show *?thesis* **by** *blast*
qed

lemma *mono-wrt-rel-Refl-Rel-Refl-Rel-if-mono-wrt-rel* [*intro*]:
assumes $(R \Rrightarrow_m S)\ f$
shows $(R^{\oplus} \Rrightarrow_m S^{\oplus})\ f$
using *assms* **by** (*intro dep-mono-wrt-relI*) *auto*

context *galois*
begin

interpretation *gR* : *galois* $(\le_L)^{\oplus}\ (\le_R)^{\oplus}\ l\ r$ **.**

lemma *Galois-Refl-RelI*:
assumes $((\le_R) \Rrightarrow_m (\le_L))\ r$
and *in-field* $(\le_L)^{\oplus}\ x$
and *in-field* $(\le_R)^{\oplus}\ y$
and *in-codom* $(\le_R)\ y \Longrightarrow x\ {}_{L}\lessapprox\ y$
shows (*galois-rel.Galois* $((\le_L)^{\oplus})\ ((\le_R)^{\oplus})\ r)\ x\ y$
using *assms* **by** (*intro gR.left-GaloisI in-codomI Refl-Rel-app-rightI*[**where** $?f{=}r$])
auto

lemma *half-galois-prop-left-Refl-Rel-left-rightI*:
assumes $((\le_L) \Rrightarrow_m (\le_R))\ l$
and $((\le_L)\ {}_h\trianglelefteq\ (\le_R))\ l\ r$
shows $((\le_L)^{\oplus}\ {}_h\trianglelefteq\ (\le_R)^{\oplus})\ l\ r$

using *assms* **by** (*intro gR.half-galois-prop-leftI Refl-RelI*)
(*auto elim*!: *in-codomE gR.left-GaloisE Refl-RelE*)

interpretation *flip-inv* : *galois* $(\geq_R)$ $(\geq_L)$ *r l*
rewrites $((\geq_R) \Rrightarrow_m (\geq_L)) \equiv ((\leq_R) \Rrightarrow_m (\leq_L))$
and $\bigwedge R.\ (R^{-1})^{\oplus} \equiv (R^{\oplus})^{-1}$
and $\bigwedge R\ S\ f\ g.\ (R^{-1}\ {}_h\trianglelefteq\ S^{-1})\ f\ g \equiv (S \trianglelefteq_h R)\ g\ f$
by (*simp-all add*: *galois-prop.half-galois-prop-left-rel-inv-iff-half-galois-prop-right*)

lemma *half-galois-prop-right-Refl-Rel-right-leftI*:
assumes $((\leq_R) \Rrightarrow_m (\leq_L))$ *r*
and $((\leq_L) \trianglelefteq_h (\leq_R))$ *l r*
shows $((\leq_L)^{\oplus} \trianglelefteq_h (\leq_R)^{\oplus})$ *l r*
using *assms* **by** (*fact flip-inv.half-galois-prop-left-Refl-Rel-left-rightI*)

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*
using *assms*
by (*intro gR.galois-propI half-galois-prop-left-Refl-Rel-left-rightI*
half-galois-prop-right-Refl-Rel-right-leftI) *auto*

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*
using *assms*
by (*intro gR.galois-connectionI galois-prop-Refl-Rel-left-rightI*) *auto*

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 −
interpret *flip* : *galois R L r l* **.**
show *?thesis* **using** *assms* **by** (*intro gR.galois-equivalenceI*
galois-connection-Refl-Rel-left-rightI flip.galois-prop-Refl-Rel-left-rightI)
auto
qed

end

context *order-functors*
begin

lemma *inflationary-on-in-field-Refl-Rel-left*:
assumes $((\leq_L) \Rrightarrow_m (\leq_R))$ *l*
and $((\leq_R) \Rrightarrow_m (\leq_L))$ *r*
and *inflationary-on* (*in-dom* $(\leq_L)$) $(\leq_L)$ η
shows *inflationary-on* (*in-field* $(\leq_L)^{\oplus}$) $(\leq_L)^{\oplus}$ η
using *assms*

by (*intro inflationary-onI Refl-RelI*) (*auto elim*!: *in-fieldE Refl-RelE*)

lemma *inflationary-on-in-field-Refl-Rel-left'*:
  assumes $((\leq_L) \Rrightarrow_m (\leq_R))$ l
  and $((\leq_R) \Rrightarrow_m (\leq_L))$ r
  and *inflationary-on* (*in-codom* $(\leq_L)$) $(\leq_L)$ η
  shows *inflationary-on* (*in-field* $(\leq_L)^\oplus$) $(\leq_L)^\oplus$ η
  using *assms*
  by (*intro inflationary-onI Refl-RelI*) (*auto elim*!: *in-fieldE Refl-RelE*)

interpretation *inv* : *galois* $(\geq_L)$ $(\geq_R)$ l r
  rewrites $((\geq_L) \Rrightarrow_m (\geq_R)) \equiv ((\leq_L) \Rrightarrow_m (\leq_R))$
  and $((\geq_R) \Rrightarrow_m (\geq_L)) \equiv ((\leq_R) \Rrightarrow_m (\leq_L))$
  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 $\bigwedge P\ R.$ *inflationary-on* P $R^{-1} \equiv$ *deflationary-on* P R
  by *simp-all*

lemma *deflationary-on-in-field-Refl-Rel-leftI*:
  assumes $((\leq_L) \Rrightarrow_m (\leq_R))$ l
  and $((\leq_R) \Rrightarrow_m (\leq_L))$ r
  and *deflationary-on* (*in-dom* $(\leq_L)$) $(\leq_L)$ η
  shows *deflationary-on* (*in-field* $(\leq_L)^\oplus$) $(\leq_L)^\oplus$ η
  using *assms* **by** (*fact inv.inflationary-on-in-field-Refl-Rel-left'*)

lemma *deflationary-on-in-field-Refl-RelI-left'*:
  assumes $((\leq_L) \Rrightarrow_m (\leq_R))$ l
  and $((\leq_R) \Rrightarrow_m (\leq_L))$ r
  and *deflationary-on* (*in-codom* $(\leq_L)$) $(\leq_L)$ η
  shows *deflationary-on* (*in-field* $(\leq_L)^\oplus$) $(\leq_L)^\oplus$ η
  using *assms* **by** (*fact inv.inflationary-on-in-field-Refl-Rel-left*)

lemma *rel-equivalence-on-in-field-Refl-Rel-leftI*:
  assumes $((\leq_L) \Rrightarrow_m (\leq_R))$ l
  and $((\leq_R) \Rrightarrow_m (\leq_L))$ r
  and *rel-equivalence-on* (*in-dom* $(\leq_L)$) $(\leq_L)$ η
  shows *rel-equivalence-on* (*in-field* $(\leq_L)^\oplus$) $(\leq_L)^\oplus$ η
  using *assms* **by** (*intro rel-equivalence-onI*
    inflationary-on-in-field-Refl-Rel-left
    deflationary-on-in-field-Refl-Rel-leftI)
  auto

lemma *rel-equivalence-on-in-field-Refl-Rel-leftI'*:
  assumes $((\leq_L) \Rrightarrow_m (\leq_R))$ l
  and $((\leq_R) \Rrightarrow_m (\leq_L))$ r
  and *rel-equivalence-on* (*in-codom* $(\leq_L)$) $(\leq_L)$ η
  shows *rel-equivalence-on* (*in-field* $(\leq_L)^\oplus$) $(\leq_L)^\oplus$ η

using *assms* **by** (*intro rel-equivalence-onI*
inflationary-on-in-field-Refl-Rel-left'
deflationary-on-in-field-Refl-RelI-left')
auto

interpretation *oR* : *order-functors* $(\leq_L)^{\oplus}$ $(\leq_R)^{\oplus}$ *l r* **.**

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 −
interpret *flip* : *galois R L r l*
rewrites *flip.unit* ≡ ε
by (*simp only*: *flip-unit-eq-counit*)
show *?thesis* **using** *assms* **by** (*intro oR.order-equivalenceI*
mono-wrt-rel-Refl-Rel-Refl-Rel-if-mono-wrt-rel
rel-equivalence-on-in-field-Refl-Rel-leftI
flip.rel-equivalence-on-in-field-Refl-Rel-leftI)
(*auto intro*: *rel-equivalence-on-if-le-pred-if-rel-equivalence-on*
in-field-if-in-dom)
qed

end

end

2.7 Monotone Function Relator

theory *Monotone-Function-Relator*
imports
Reflexive-Relator
begin

abbreviation *Mono-Dep-Fun-Rel R S* ≡ ([*x y* :: *R*] ⇛ *S x y*)$^{\oplus}$
abbreviation *Mono-Fun-Rel R S* ≡ *Mono-Dep-Fun-Rel R* (λ- -. *S*)

bundle *Mono-Dep-Fun-Rel-syntax* **begin**
syntax
-Mono-Fun-Rel-rel :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ (*'c* ⇒ *'d* ⇒ *bool*) ⇒ (*'a* ⇒ *'c*) ⇒
(*'b* ⇒ *'d*) ⇒ *bool* ((-) ⇛⊕ (-) [*41*, *40*] *40*)
-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
bundle *no-Mono-Dep-Fun-Rel-syntax* **begin**

no-syntax
-*Mono-Fun-Rel-rel* :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow ('c \Rightarrow 'd \Rightarrow bool) \Rightarrow ('a \Rightarrow 'c) \Rightarrow$
$('b \Rightarrow 'd) \Rightarrow bool$ ((-) $\Rrightarrow\oplus$ (-) [*41*, *40*] *40*)
-*Mono-Dep-Fun-Rel-rel* :: $idt \Rightarrow idt \Rightarrow ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow ('c \Rightarrow 'd \Rightarrow bool) \Rightarrow$
$('a \Rightarrow 'c) \Rightarrow ('b \Rightarrow 'd) \Rightarrow bool$ ([-/ -/ ::/ -] $\Rrightarrow\oplus$ (-) [*41*, *41*, *41*, *40*] *40*)
-*Mono-Dep-Fun-Rel-rel-if* :: $idt \Rightarrow idt \Rightarrow ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool \Rightarrow ('c \Rightarrow 'd$
$\Rightarrow bool) \Rightarrow$
$('a \Rightarrow 'c) \Rightarrow ('b \Rightarrow 'd) \Rightarrow bool$ ([-/ -/ ::/ -/ |/ -] $\Rrightarrow\oplus$ (-) [*41*, *41*, *41*, *41*, *40*]
40)
end
unbundle *Mono-Dep-Fun-Rel-syntax*

translations
$R \Rrightarrow\oplus S \rightleftharpoons$ *CONST Mono-Fun-Rel R S*
$[x\ y :: R] \Rrightarrow\oplus S \rightleftharpoons$ *CONST Mono-Dep-Fun-Rel R* $(\lambda x\ y.\ S)$
$[x\ y :: R \mid B] \Rrightarrow\oplus S \rightleftharpoons$ *CONST Mono-Dep-Fun-Rel R* $(\lambda x\ y.$ *CONST rel-if B S*$)$

locale *Dep-Fun-Rel-orders* =
fixes $L :: 'a \Rightarrow 'b \Rightarrow bool$
and $R :: 'a \Rightarrow 'b \Rightarrow 'c \Rightarrow 'd \Rightarrow bool$
begin

sublocale *o* : *orders L R a b* **for** *a b* **.**

notation *L* (**infix** $\le_L$ *50*)
notation *o.ge-left* (**infix** $\ge_L$ *50*)

notation *R* $((\le_{R\ (-)\ (-)})$ *50*)
abbreviation *right-infix c a b d* $\equiv (\le_{R\ a\ b})\ c\ d$
notation *right-infix* ((-) $\le_{R\ (-)\ (-)}$ (-) [*51*,*51*,*51*,*51*] *50*)

notation *o.ge-right* $((\ge_{R\ (-)\ (-)})$ *50*)

abbreviation (*input*) *ge-right-infix d a b c* $\equiv (\ge_{R\ a\ b})\ d\ c$
notation *ge-right-infix* ((-) $\ge_{R\ (-)\ (-)}$ (-) [*51*,*51*,*51*,*51*] *50*)

abbreviation (*input*) *DFR* $\equiv ([a\ b :: L] \Rrightarrow R\ a\ b)$

end

locale *hom-Dep-Fun-Rel-orders* = *Dep-Fun-Rel-orders L R*
for $L :: 'a \Rightarrow 'a \Rightarrow bool$
and $R :: 'a \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b \Rightarrow bool$
begin

sublocale *ho* : *hom-orders L R a b* **for** *a b* **.**

lemma *Mono-Dep-Fun-Refl-Rel-right-eq-Mono-Dep-Fun-if-le-if-reflexive-onI*:
assumes *refl-L*: *reflexive-on* (*in-field* $(\le_L)$) $(\le_L)$

and $\bigwedge x1\ x2.\ x1 \le_L x2 \Longrightarrow (\le_{R\ x2\ x2}) \le (\le_{R\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \le_L x2 \Longrightarrow (\le_{R\ x1\ x1}) \le (\le_{R\ x1\ x2})$
shows $([x\ y :: (\le_L)] \Rrightarrow\oplus (\le_{R\ x\ y})^{\oplus}) = ([x\ y :: (\le_L)] \Rrightarrow\oplus (\le_{R\ x\ y}))$
proof −
{
fix *f g x1 x2*
assume $([x\ y :: (\le_L)] \Rrightarrow (\le_{R\ x\ y}))\ f\ g\ x1 \le_L x1\ x1 \le_L x2$
with *assms* **have** $f\ x1 \le_{R\ x1\ x2} g\ x1\ f\ x2 \le_{R\ x1\ x2} g\ x2$ **by** *blast+*
}
with *refl-L* **show** *?thesis*
by (*intro ext iffI Refl-RelI Dep-Fun-Rel-relI*) (*auto elim*!: *Refl-RelE*)
qed

lemma *Mono-Dep-Fun-Refl-Rel-right-eq-Mono-Dep-Fun-if-mono-if-reflexive-onI*:
assumes *reflexive-on* (*in-field* $(\le_L)$) $(\le_L)$
and $([x1\ x2 :: (\ge_L)] \Rrightarrow_m [x3\ x4 :: (\le_L) \mid x1 \le_L x3] \Rrightarrow (\le))\ R$
shows $([x\ y :: (\le_L)] \Rrightarrow\oplus (\le_{R\ x\ y})^{\oplus}) = ([x\ y :: (\le_L)] \Rrightarrow\oplus (\le_{R\ x\ y}))$
using *assms*
by (*intro Mono-Dep-Fun-Refl-Rel-right-eq-Mono-Dep-Fun-if-le-if-reflexive-onI*)
auto

end

context *hom-orders*
begin

sublocale *fro* : *hom-Dep-Fun-Rel-orders L* λ- -. *R* **.**

corollary *Mono-Fun-Rel-Refl-Rel-right-eq-Mono-Fun-RelI*:
assumes *reflexive-on* (*in-field* $(\le_L)$) $(\le_L)$
shows $((\le_L) \Rrightarrow\oplus (\le_R)^{\oplus}) = ((\le_L) \Rrightarrow\oplus (\le_R))$
using *assms* **by** (*intro fro.Mono-Dep-Fun-Refl-Rel-right-eq-Mono-Dep-Fun-if-le-if-reflexive-onI*)
simp-all

end

end

2.8 Transport For Functions

2.8.1 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*
  unfolding *flip2-def* **by** *simp*

lemma *flip2-eq-rel-inv* [*simp*]: *flip2 R x y* = $(R\ y\ x)^{-1}$
  by (*intro ext*) (*simp only*: *flip2-eq rel-inv-iff-rel*)

lemma *flip2-flip2-eq-self* [*simp*]: *flip2* (*flip2 f*) = *f*
  by (*intro ext*) (*simp add*: *flip2-eq*)

lemma *flip2-eq-flip2-iff-eq* [*iff*]: *flip2 f* = *flip2 g* ⟷ *f* = *g*
  unfolding *flip2-def* **by** (*intro iffI ext*) (*auto dest*: *fun-cong*)

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** $\le_{L1}$ *50*)
notation *R1* (**infix** $\le_{R1}$ *50*)

notation *t1.ge-left* (**infix** $\ge_{L1}$ *50*)
notation *t1.ge-right* (**infix** $\ge_{R1}$ *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* $((\le_{L2\ (-)\ (-)})\ 50)$
notation *R2* $((\le_{R2\ (-)\ (-)})\ 50)$

notation *dfro1.right-infix* $((-)\ \le_{L2\ (-)\ (-)}\ (-)\ [51,51,51,51]\ 50)$
notation *dfro2.right-infix* $((-)\ \le_{R2\ (-)\ (-)}\ (-)\ [51,51,51,51]\ 50)$

notation *dfro1.o.ge-right* $((\ge_{L2\ (-)\ (-)})\ 50)$
notation *dfro2.o.ge-right* $((\ge_{R2\ (-)\ (-)})\ 50)$

notation *dfro1.ge-right-infix* $((-)\ \ge_{L2\ (-)\ (-)}\ (-)\ [51,51,51,51]\ 50)$
notation *dfro2.ge-right-infix* $((-)\ \ge_{R2\ (-)\ (-)}\ (-)\ [51,51,51,51]\ 50)$

notation *l2* $(l2_{(-)\ (-)})$
notation *r2* $(r2_{(-)\ (-)})$

sublocale *t2* : *transport* $(\le_{L2\ x\ (r1\ x')})$ $(\le_{R2\ (l1\ x)\ x'})$ $l2_{x'\ x}$ $r2_{x\ x'}$ **for** $x\ x'$ **.**

notation *t2.left-Galois* $((_{L2\ (-)\ (-)}\lessapprox)\ 50)$
notation *t2.right-Galois* $((_{R2\ (-)\ (-)}\lessapprox)\ 50)$

abbreviation *left2-Galois-infix* $y\ x\ x'\ y' \equiv (_{L2\ x\ x'}\lessapprox)\ y\ y'$
notation *left2-Galois-infix* $((-)\ _{L2\ (-)\ (-)}\lessapprox\ (-)\ [51,51,51,51]\ 50)$
abbreviation *right2-Galois-infix* $y'\ x\ x'\ y \equiv (_{R2\ x\ x'}\lessapprox)\ y'\ y$
notation *right2-Galois-infix* $((-)\ _{R2\ (-)\ (-)}\lessapprox\ (-)\ [51,51,51,51]\ 50)$

notation *t2.ge-Galois-left* $((\gtrapprox_{L2\ (-)\ (-)})\ 50)$
notation *t2.ge-Galois-right* $((\gtrapprox_{R2\ (-)\ (-)})\ 50)$

abbreviation (*input*) *ge-Galois-left-left2-infix* $y'\ x\ x'\ y \equiv (\gtrapprox_{L2\ x\ x'})\ y'\ y$
notation *ge-Galois-left-left2-infix* $((-)\ \gtrapprox_{L2\ (-)\ (-)}\ (-)\ [51,51,51,51]\ 50)$
abbreviation (*input*) *ge-Galois-left-right2-infix* $y\ x\ x'\ y' \equiv (\gtrapprox_{R2\ x\ x'})\ y\ y'$
notation *ge-Galois-left-right2-infix* $((-)\ \gtrapprox_{R2\ (-)\ (-)}\ (-)\ [51,51,51,51]\ 50)$

notation *t2.right-ge-Galois* $((_{R2\ (-)\ (-)}\gtrapprox)\ 50)$
notation *t2.left-ge-Galois* $((_{L2\ (-)\ (-)}\gtrapprox)\ 50)$

abbreviation *left2-ge-Galois-left-infix* $y\ x\ x'\ y' \equiv (_{L2\ x\ x'}\gtrapprox)\ y\ y'$
notation *left2-ge-Galois-left-infix* $((-)\ _{L2\ (-)\ (-)}\gtrapprox\ (-)\ [51,51,51,51]\ 50)$
abbreviation *right2-ge-Galois-left-infix* $y'\ x\ x'\ y \equiv (_{R2\ x\ x'}\gtrapprox)\ y'\ y$
notation *right2-ge-Galois-left-infix* $((-)\ _{R2\ (-)\ (-)}\gtrapprox\ (-)\ [51,51,51,51]\ 50)$

notation *t2.Galois-right* $((\lessapprox_{R2\ (-)\ (-)})\ 50)$
notation *t2.Galois-left* $((\lessapprox_{L2\ (-)\ (-)})\ 50)$

abbreviation (*input*) *Galois-left2-infix* $y'\ x\ x'\ y \equiv (\lessapprox_{L2\ x\ x'})\ y'\ y$
notation *Galois-left2-infix* $((-)\ \lessapprox_{L2\ (-)\ (-)}\ (-)\ [51,51,51,51]\ 50)$
abbreviation (*input*) *Galois-right2-infix* $y\ x\ x'\ y' \equiv (\lessapprox_{R2\ x\ x'})\ y\ y'$

notation *Galois-right2-infix* ((-) $\lessapprox_{R2\ (\text{-})\ (\text{-})}$ (-) [*51*,*51*,*51*,*51*] *50*)

abbreviation *t2-unit x x'* ≡ *t2.unit x' x*
notation *t2-unit* ($\eta_{2\ (\text{-})\ (\text{-})}$)
abbreviation *t2-counit x x'* ≡ *t2.counit x' x*
notation *t2-counit* ($\varepsilon_{2\ (\text{-})\ (\text{-})}$)

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* ⇒ *'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

definition *L* ≡ [*x1 x2* :: ($\le_{L1}$)] ⇛ ($\le_{L2\ x1\ x2}$)

lemma *left-rel-eq-Dep-Fun-Rel*: *L* = ([*x1 x2* :: ($\le_{L1}$)] ⇛ ($\le_{L2\ x1\ x2}$))
  unfolding *L-def* **..**

definition *l* ≡ ([*x'* : *r1*] → *l2 x'*)

lemma *left-eq-dep-fun-map*: *l* = ([*x'* : *r1*] → *l2 x'*)
  unfolding *l-def* **..**

lemma *left-eq* [*simp*]: *l f x'* = $l2_{x'\ (r1\ x')}$ (*f* (*r1 x'*))
  unfolding *left-eq-dep-fun-map* **by** *simp*

context
begin

interpretation *flip* : *transport-Dep-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2* **.**

abbreviation *R* ≡ *flip.L*
abbreviation *r* ≡ *flip.l*

lemma *right-rel-eq-Dep-Fun-Rel*: *R* = ([*x1' x2'* :: ($\le_{R1}$)] ⇛ ($\le_{R2\ x1'\ x2'}$))
  unfolding *flip.L-def* **..**

lemma *right-eq-dep-fun-map*: *r* = ([*x* : *l1*] → *r2 x*)
  unfolding *flip.l-def* **..**

end

lemma *right-eq* [*simp*]: $r\ g\ x = r2_{x\ (l1\ x)}\ (g\ (l1\ x))$
  unfolding *right-eq-dep-fun-map* **by** *simp*

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* **.**

notation *L* (**infix** $\le_L$ *50*)
notation *R* (**infix** $\le_R$ *50*)

lemma *left-relI* [*intro*]:
  assumes $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow f\ x1 \le_{L2\ x1\ x2} f'\ x2$
  shows $f \le_L f'$
  unfolding *left-rel-eq-Dep-Fun-Rel* **using** *assms* **by** *blast*

lemma *left-relE* [*elim*]:
  assumes $f \le_L f'$
  and $x1 \le_{L1} x2$
  obtains $f\ x1 \le_{L2\ x1\ x2} f'\ x2$
  using *assms* **unfolding** *left-rel-eq-Dep-Fun-Rel* **by** *blast*

interpretation *flip-inv* :
  transport-Dep-Fun-Rel $(\ge_{R1})$ $(\ge_{L1})$ *r1 l1 flip2 R2 flip2 L2 r2 l2* **.**

lemma *flip-inv-right-eq-ge-left*: *flip-inv.R* = $(\ge_L)$
  unfolding *left-rel-eq-Dep-Fun-Rel flip-inv.right-rel-eq-Dep-Fun-Rel*
  by (*simp only*: *rel-inv-Dep-Fun-Rel-rel-eq flip2-eq-rel-inv*[*symmetric*, *of L2*])

interpretation *flip* : *transport-Dep-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2* **.**

lemma *flip-inv-left-eq-ge-right*: *flip-inv.L* $\equiv$ $(\ge_R)$
  unfolding *flip.flip-inv-right-eq-ge-left* **.**

Useful Rewritings for Dependent Relation **lemma** *left-rel2-unit-eqs-left-rel2I*:
  assumes $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x2\ x2}) \le (\le_{L2\ x1\ x2})$
  and $\bigwedge x.\ x \le_{L1} x \Longrightarrow (\le_{L2\ (\eta_1\ x)\ x}) \le (\le_{L2\ x\ x})$
  and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ x1}) \le (\le_{L2\ x1\ x2})$
  and $\bigwedge x.\ x \le_{L1} x \Longrightarrow (\le_{L2\ x\ (\eta_1\ x)}) \le (\le_{L2\ x\ x})$
  and $x \le_{L1} x$
  and $x \equiv_{L1} \eta_1\ x$
  shows $(\le_{L2\ (\eta_1\ x)\ x}) = (\le_{L2\ x\ x})$
  and $(\le_{L2\ x\ (\eta_1\ x)}) = (\le_{L2\ x\ x})$
  using *assms* **by** (*auto intro*!: *antisym*)

lemma *left2-eq-if-bi-related-if-monoI*:

assumes *mono-L2*: $([x1\ x2 :: (\geq_{L1})] \Rrightarrow_m [x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3] \Rrightarrow (\leq))$ *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 (*intro antisym*)
from ‹$x1 \equiv_{L1} x3$› ‹$x2 \equiv_{L1} x4$› **have** $x3 \leq_{L1} x1$ $x2 \leq_{L1} x4$ **by** *auto*
with ‹$x1 \leq_{L1} x2$› *mono-L2* **show** $(\leq_{L2\ x1\ x2}) \leq (\leq_{L2\ x3\ x4})$ **by** *blast*
from ‹$x1 \equiv_{L1} x3$› ‹$x2 \equiv_{L1} x4$› **have** $x1 \leq_{L1} x3$ $x4 \leq_{L1} x2$ **by** *auto*
moreover from ‹$x3 \leq_{L1} x1$› ‹$x1 \leq_{L1} x2$› ‹$x2 \leq_{L1} x4$› **have** $x3 \leq_{L1} x4$
using *trans-L1* **by** *blast*
ultimately show $(\leq_{L2\ x3\ x4}) \leq (\leq_{L2\ x1\ x2})$ **using** *mono-L2* **by** *blast*
qed

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** $\leq_{L1}$ *50*)
notation *R1* (**infix** $\leq_{R1}$ *50*)

notation *tdfrs.t1.ge-left* (**infix** $\geq_{L1}$ *50*)
notation *tdfrs.t1.ge-right* (**infix** $\geq_{R1}$ *50*)

notation *tdfrs.t1.left-Galois* (**infix** $_{L1}\lessapprox$ *50*)
notation *tdfrs.t1.ge-Galois-left* (**infix** $\gtrapprox_{L1}$ *50*)
notation *tdfrs.t1.right-Galois* (**infix** $_{R1}\lessapprox$ *50*)
notation *tdfrs.t1.ge-Galois-right* (**infix** $\gtrapprox_{R1}$ *50*)
notation *tdfrs.t1.right-ge-Galois* (**infix** $_{R1}\gtrapprox$ *50*)
notation *tdfrs.t1.Galois-right* (**infix** $\lessapprox_{R1}$ *50*)
notation *tdfrs.t1.left-ge-Galois* (**infix** $_{L1}\gtrapprox$ *50*)
notation *tdfrs.t1.Galois-left* (**infix** $\lessapprox_{L1}$ *50*)

notation *tdfrs.t1.unit* (η_1)
notation *tdfrs.t1.counit* (ε_1)

notation *L2* (**infix** $\leq_{L2}$ *50*)
notation *R2* (**infix** $\leq_{R2}$ *50*)

notation *tdfrs.t2.ge-left* (**infix** $\geq_{L2}$ *50*)
notation *tdfrs.t2.ge-right* (**infix** $\geq_{R2}$ *50*)

notation *tdfrs.t2.left-Galois* (**infix** ${}_{L2}\lessapprox$ *50*)
notation *tdfrs.t2.ge-Galois-left* (**infix** $\gtrapprox_{L2}$ *50*)
notation *tdfrs.t2.right-Galois* (**infix** ${}_{R2}\lessapprox$ *50*)
notation *tdfrs.t2.ge-Galois-right* (**infix** $\gtrapprox_{R2}$ *50*)
notation *tdfrs.t2.right-ge-Galois* (**infix** ${}_{R2}\gtrapprox$ *50*)
notation *tdfrs.t2.Galois-right* (**infix** $\lessapprox_{R2}$ *50*)
notation *tdfrs.t2.left-ge-Galois* (**infix** ${}_{L2}\gtrapprox$ *50*)
notation *tdfrs.t2.Galois-left* (**infix** $\lessapprox_{L2}$ *50*)

notation *tdfrs.t2.unit* (η_2)
notation *tdfrs.t2.counit* (ε_2)

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*: $(\le_L) = ((\le_{L1}) \Rrightarrow (\le_{L2}))$
  unfolding *tdfr.left-rel-eq-Dep-Fun-Rel* **by** *simp*

lemma *left-eq-fun-map*: $l = (r1 \rightarrow l2)$
  by (*intro ext*) *simp*

interpretation *flip* : *transport-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2* **.**

lemma *right-rel-eq-Fun-Rel*: $(\le_R) = ((\le_{R1}) \Rrightarrow (\le_{R2}))$
  unfolding *flip.left-rel-eq-Fun-Rel* **..**

lemma *right-eq-fun-map*: $r = (l1 \rightarrow r2)$
  unfolding *flip.left-eq-fun-map* **..**

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* ⇒ *'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

definition $L \equiv tdfr.L^{\oplus}$

lemma *left-rel-eq-tdfr-left-Refl-Rel*: $L = tdfr.L^{\oplus}$
  unfolding *L-def* **..**

lemma *left-rel-eq-Mono-Dep-Fun-Rel*: $L = ([x1\ x2 :: (\le_{L1})] \Rrightarrow\oplus (\le_{L2\ x1\ x2}))$

unfolding *left-rel-eq-tdfr-left-Refl-Rel tdfr.left-rel-eq-Dep-Fun-Rel* **by** *simp*

lemma *left-rel-eq-tdfr-left-rel-if-reflexive-on*:
assumes *reflexive-on* (*in-field tdfr.L*) *tdfr.L*
shows $L = tdfr.L$
unfolding *left-rel-eq-tdfr-left-Refl-Rel* **using** *assms*
by (*rule Refl-Rel-eq-self-if-reflexive-on*)

abbreviation $l \equiv tdfr.l$

lemma *left-eq-tdfr-left*: $l = tdfr.l$ **..**

interpretation *flip* : *transport-Mono-Dep-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2* **.**

abbreviation $R \equiv flip.L$

lemma *right-rel-eq-tdfr-right-Refl-Rel*: $R = tdfr.R^{\oplus}$
unfolding *flip.left-rel-eq-tdfr-left-Refl-Rel* **..**

lemma *right-rel-eq-Mono-Dep-Fun-Rel*: $R = ([y1\ y2 :: (\leq_{R1})] \Rrightarrow\oplus (\leq_{R2\ y1\ y2}))$
unfolding *flip.left-rel-eq-Mono-Dep-Fun-Rel* **..**

lemma *right-rel-eq-tdfr-right-rel-if-reflexive-on*:
assumes *reflexive-on* (*in-field tdfr.R*) *tdfr.R*
shows $R = tdfr.R$
using *assms* **by** (*rule flip.left-rel-eq-tdfr-left-rel-if-reflexive-on*)

abbreviation $r \equiv tdfr.r$

lemma *right-eq-tdfr-right*: $r = tdfr.r$ **..**

lemmas *transport-defs* = *left-rel-eq-tdfr-left-Refl-Rel*
right-rel-eq-tdfr-right-Refl-Rel

sublocale *transport L R l r* **.**

notation *L* (**infix** $\leq_L$ *50*)
notation *R* (**infix** $\leq_R$ *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** $\leq_L$ *50*)
notation *tpdfr.R* (**infix** $\leq_R$ *50*)

notation *tpdfr.ge-left* (**infix** $\geq_L$ *50*)
notation *tpdfr.ge-right* (**infix** $\geq_R$ *50*)

notation *tpdfr.left-Galois* (**infix** $_L\lessapprox$ *50*)
notation *tpdfr.ge-Galois-left* (**infix** $\gtrapprox_L$ *50*)
notation *tpdfr.right-Galois* (**infix** $_R\lessapprox$ *50*)
notation *tpdfr.ge-Galois-right* (**infix** $\gtrapprox_R$ *50*)
notation *tpdfr.right-ge-Galois* (**infix** $_R\gtrapprox$ *50*)
notation *tpdfr.Galois-right* (**infix** $\lessapprox_R$ *50*)
notation *tpdfr.left-ge-Galois* (**infix** $_L\gtrapprox$ *50*)
notation *tpdfr.Galois-left* (**infix** $\lessapprox_L$ *50*)

notation *tpdfr.unit* (*η*)
notation *tpdfr.counit* (*ε*)

lemma *left-rel-eq-Mono-Fun-Rel*: $(\leq_L) = ((\leq_{L1}) \Rrightarrow\oplus (\leq_{L2}))$
unfolding *tpdfr.left-rel-eq-Mono-Dep-Fun-Rel* **by** *simp*

lemma *left-eq-fun-map*: *l* = (*r1* → *l2*)
unfolding *tfr.left-eq-fun-map* **..**

interpretation *flip* : *transport-Mono-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2* **.**

lemma *right-rel-eq-Mono-Fun-Rel*: $(\leq_R) = ((\leq_{R1}) \Rrightarrow\oplus (\leq_{R2}))$
unfolding *flip.left-rel-eq-Mono-Fun-Rel* **..**

lemma *right-eq-fun-map*: *r* = (*l1* → *r2*)
unfolding *flip.left-eq-fun-map* **..**

lemmas *transport-defs* = *tpdfr.transport-defs*

end

end

2.8.2 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* **.**

lemma *mono-wrt-rel-leftI*:
  assumes *mono-r1*: $((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))\ r1$
  and *mono-l2*: $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
    $((\leq_{L2\ (r1\ x1')\ (r1\ x2')}) \Rrightarrow_m (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}))\ (l2_{x2'\ (r1\ x1')})$
  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 $((\leq_L) \Rrightarrow_m (\leq_R))\ l$
proof (*intro dep-mono-wrt-relI flip.left-relI*)
  fix *f1 f2 x1′ x2′* **assume** [*iff*]: $x1' \leq_{R1} x2'$
  with *mono-r1* **have** $r1\ x1' \leq_{L1} r1\ x2'$ (**is** $?x1 \leq_{L1} ?x2$) **by** *blast*
  moreover assume $f1 \leq_L f2$
  ultimately have $f1\ ?x1 \leq_{L2\ ?x1\ ?x2} f2\ ?x2$ (**is** $?y1 \leq_{L2\ ?x1\ ?x2} ?y2$) **by** *blast*
  with *mono-l2* **have** $l2_{x2'\ ?x1}\ ?y1 \leq_{R2\ (\varepsilon_1\ x1')\ x2'} l2_{x2'\ ?x1}\ ?y2$ **by** *blast*
  with *R2-le1* **have** $l2_{x2'\ ?x1}\ ?y1 \leq_{R2\ x1'\ x2'} l2_{x2'\ ?x1}\ ?y2$ **by** *blast*
  with *R2-l2-le1* **have** $l2_{x1'\ ?x1}\ ?y1 \leq_{R2\ x1'\ x2'} l2_{x2'\ ?x1}\ ?y2$
    using ‹$?y1 \leq_{L2\ ?x1\ ?x2}$ -› **by** *blast*
  with *ge-R2-l2-le2* **have** $l2_{x1'\ ?x1}\ ?y1 \leq_{R2\ x1'\ x2'} l2_{x2'\ ?x2}\ ?y2$
    using ‹- $\leq_{L2\ ?x1\ ?x2}\ ?y2$› **by** *blast*
  then show $l\ f1\ x1' \leq_{R2\ x1'\ x2'} l\ f2\ x2'$ **by** *simp*
qed

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* $(\le_{L2\ (r1\ x1')\ (r1\ x2')})$ *y*
shows $(\le_{R2\ x1'\ x2'})\ (l2_{x2'\ (r1\ x1')}\ y) \le (\le_{R2\ x1'\ x2'})\ (l2_{x1'\ (r1\ x1')}\ y)$
using *assms* **by** *blast*

lemma *mono-wrt-rel-left-in-codom-mono-left-assm*:
assumes $([in\text{-}codom\ (\le_{L2\ (r1\ x1')\ (r1\ x2')})] \Rrightarrow (\le_{R2\ x1'\ x2'}))$
$(l2_{x2'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x2')})$
and *transitive* $(\le_{R2\ x1'\ x2'})$
and $x1' \le_{R1} x2'$
and *in-codom* $(\le_{L2\ (r1\ x1')\ (r1\ x2')})$ *y*
shows $(\ge_{R2\ x1'\ x2'})\ (l2_{x2'\ (r1\ x1')}\ y) \le (\ge_{R2\ x1'\ x2'})\ (l2_{x2'\ (r1\ x2')}\ y)$
using *assms* **by** *blast*

lemma *mono-wrt-rel-left-if-transitiveI*:
assumes $((\le_{R1}) \Rrightarrow_m (\le_{L1}))\ r1$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow$
$((\le_{L2\ (r1\ x1')\ (r1\ x2')}) \Rrightarrow_m (\le_{R2\ (\varepsilon_1\ x1')\ x2'}))\ (l2_{x2'\ (r1\ x1')})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ (\varepsilon_1\ x1')\ x2'}) \le (\le_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow$
$([in\text{-}dom\ (\le_{L2\ (r1\ x1')\ (r1\ x2')})] \Rrightarrow (\le_{R2\ x1'\ x2'}))\ (l2_{x1'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x1')})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow$
$([in\text{-}codom\ (\le_{L2\ (r1\ x1')\ (r1\ x2')})] \Rrightarrow (\le_{R2\ x1'\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x2')})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow$ *transitive* $(\le_{R2\ x1'\ x2'})$
shows $((\le_L) \Rrightarrow_m (\le_R))\ l$
using *assms* **by** (*intro mono-wrt-rel-leftI*
mono-wrt-rel-left-in-dom-mono-left-assm
mono-wrt-rel-left-in-codom-mono-left-assm)
auto

lemma *mono-wrt-rel-left2-if-mono-wrt-rel-left2-if-left-GaloisI*:
assumes $((\le_{R1}) \Rrightarrow_m (\le_{L1}))\ r1$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\le_{L2\ x\ (r1\ x')}) \Rrightarrow_m (\le_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})$
shows $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow$
$((\le_{L2\ (r1\ x1')\ (r1\ x2')}) \Rrightarrow_m (\le_{R2\ (\varepsilon_1\ x1')\ x2'}))\ (l2_{x2'\ (r1\ x1')})$
using *assms* **by** (*intro dep-mono-wrt-relI*) *fastforce*

interpretation *flip-inv* :
transport-Dep-Fun-Rel $(\ge_{R1})\ (\ge_{L1})$ *r1 l1 flip2 R2 flip2 L2 r2 l2*
rewrites *flip-inv.R* $\equiv (\ge_L)$ **and** *flip-inv.L* $\equiv (\ge_R)$
and *flip-inv.t1.counit* $\equiv \eta_1$
and $\bigwedge R\ x\ y.\ (flip2\ R\ x\ y)^{-1} \equiv R\ y\ x$
and $\bigwedge R\ x1\ x2.\ in\text{-}dom\ (flip2\ R\ x1\ x2) \equiv in\text{-}codom\ (R\ x2\ x1)$
and $\bigwedge R\ x1\ x2.\ in\text{-}codom\ (flip2\ R\ x1\ x2) \equiv in\text{-}dom\ (R\ x2\ x1)$
and $\bigwedge R\ S.\ (R^{-1} \Rrightarrow_m S^{-1}) \equiv (R \Rrightarrow_m S)$
and $\bigwedge x1\ x2\ x1'\ x2'.\ (flip2\ R2\ x1'\ x2' \Rrightarrow_m flip2\ L2\ x1\ x2) \equiv$
$((\le_{R2\ x2'\ x1'}) \Rrightarrow_m (\le_{L2\ x2\ x1}))$
and $\bigwedge x1\ x2\ x3\ x4.\ flip2\ L2\ x1\ x2 \le flip2\ L2\ x3\ x4 \equiv (\le_{L2\ x2\ x1}) \le (\le_{L2\ x4\ x3})$

and $\bigwedge x1' \; x2' \; y1 \; y2.$
flip-inv.dfro2.right-infix $y1 \; x1' \; x2' \le$ *flip-inv.dfro2.right-infix* $y2 \; x1' \; x2' \equiv$
$(\ge_{L2 \; x2' \; x1'}) \; y1 \le (\ge_{L2 \; x2' \; x1'}) \; y2$
and $\bigwedge P \; x1 \; x2.$ $([P] \Rrightarrow$ *flip2* $L2 \; x1 \; x2) \equiv ([P] \Rrightarrow (\ge_{L2 \; x2 \; x1}))$
and $\bigwedge P \; R.$ $([P] \Rrightarrow R^{-1}) \equiv ([P] \Rrightarrow R)^{-1}$
and $\bigwedge x1 \; x2.$ *transitive* (*flip2* $L2 \; x1 \; x2$) $\equiv$ *transitive* $(\le_{L2 \; x2 \; x1})$
by (*simp-all add*: *flip-inv-left-eq-ge-right flip-inv-right-eq-ge-left*
t1.flip-counit-eq-unit del: *rel-inv-iff-rel*)

lemma *mono-wrt-rel-rightI*:
assumes $((\le_{L1}) \Rrightarrow_m (\le_{R1})) \; l1$
and $\bigwedge x1 \; x2.$ $x1 \le_{L1} x2 \Longrightarrow ((\le_{R2 \; (l1 \; x1) \; (l1 \; x2)}) \Rrightarrow_m (\le_{L2 \; x1 \; (\eta_1 \; x2)})) \; (r2_{x1 \; (l1 \; x2)})$
and $\bigwedge x1 \; x2.$ $x1 \le_{L1} x2 \Longrightarrow (\le_{L2 \; x1 \; (\eta_1 \; x2)}) \le (\le_{L2 \; x1 \; x2})$
and $\bigwedge x1 \; x2 \; y'.$ $x1 \le_{L1} x2 \Longrightarrow$ *in-codom* $(\le_{R2 \; (l1 \; x1) \; (l1 \; x2)}) \; y' \Longrightarrow$
$(\ge_{L2 \; x1 \; x2}) \; (r2_{x1 \; (l1 \; x2)} \; y') \le (\ge_{L2 \; x1 \; x2}) \; (r2_{x2 \; (l1 \; x2)} \; y')$
and $\bigwedge x1 \; x2 \; y'.$ $x1 \le_{L1} x2 \Longrightarrow$ *in-dom* $(\le_{R2 \; (l1 \; x1) \; (l1 \; x2)}) \; y' \Longrightarrow$
$(\le_{L2 \; x1 \; x2}) \; (r2_{x1 \; (l1 \; x2)} \; y') \le (\le_{L2 \; x1 \; x2}) \; (r2_{x1 \; (l1 \; x1)} \; y')$
shows $((\le_R) \Rrightarrow_m (\le_L)) \; r$
using *assms* **by** (*intro flip-inv.mono-wrt-rel-leftI*[*simplified rel-inv-iff-rel*])

lemma *mono-wrt-rel-right-if-transitiveI*:
assumes $((\le_{L1}) \Rrightarrow_m (\le_{R1})) \; l1$
and $\bigwedge x1 \; x2.$ $x1 \le_{L1} x2 \Longrightarrow ((\le_{R2 \; (l1 \; x1) \; (l1 \; x2)}) \Rrightarrow_m (\le_{L2 \; x1 \; (\eta_1 \; x2)})) \; (r2_{x1 \; (l1 \; x2)})$
and $\bigwedge x1 \; x2.$ $x1 \le_{L1} x2 \Longrightarrow (\le_{L2 \; x1 \; (\eta_1 \; x2)}) \le (\le_{L2 \; x1 \; x2})$
and $\bigwedge x1 \; x2.$ $x1 \le_{L1} x2 \Longrightarrow$
([*in-codom* $(\le_{R2 \; (l1 \; x1) \; (l1 \; x2)})] \Rrightarrow (\le_{L2 \; x1 \; x2})) \; (r2_{x1 \; (l1 \; x2)}) \; (r2_{x2 \; (l1 \; x2)})$
and $\bigwedge x1 \; x2.$ $x1 \le_{L1} x2 \Longrightarrow$
([*in-dom* $(\le_{R2 \; (l1 \; x1) \; (l1 \; x2)})] \Rrightarrow (\le_{L2 \; x1 \; x2})) \; (r2_{x1 \; (l1 \; x1)}) \; (r2_{x1 \; (l1 \; x2)})$
and $\bigwedge x1 \; x2.$ $x1 \le_{L1} x2 \Longrightarrow$ *transitive* $(\le_{L2 \; x1 \; x2})$
shows $((\le_R) \Rrightarrow_m (\le_L)) \; r$
using *assms* **by** (*intro flip-inv.mono-wrt-rel-left-if-transitiveI*
[*simplified rel-inv-iff-rel*])

lemma *mono-wrt-rel-right2-if-mono-wrt-rel-right2-if-left-GaloisI*:
assumes *assms1*: $((\le_{L1}) \Rrightarrow_m (\le_{R1})) \; l1 \; ((\le_{L1}) \trianglelefteq_h (\le_{R1})) \; l1 \; r1$
and *mono-r2*: $\bigwedge x \; x'.$ $x \;_{L1}\lessapprox x' \Longrightarrow ((\le_{R2 \; (l1 \; x) \; x'}) \Rrightarrow_m (\le_{L2 \; x \; (r1 \; x')})) \; (r2_{x \; x'})$
shows $\bigwedge x1 \; x2.$ $x1 \le_{L1} x2 \Longrightarrow ((\le_{R2 \; (l1 \; x1) \; (l1 \; x2)}) \Rrightarrow_m (\le_{L2 \; x1 \; (\eta_1 \; x2)}))$
$(r2_{x1 \; (l1 \; x2)})$
proof −
show $((\le_{R2 \; (l1 \; x1) \; (l1 \; x2)}) \Rrightarrow_m (\le_{L2 \; x1 \; (\eta_1 \; x2)})) \; (r2_{x1 \; (l1 \; x2)})$ **if** $x1 \le_{L1} x2$
for *x1 x2*
proof −
from ‹$x1 \le_{L1} x2$› **have** $x1 \;_{L1}\lessapprox l1 \; x2$
using *assms1* **by** (*intro t1.left-Galois-left-if-left-relI*) *blast*
with *mono-r2* **show** *?thesis* **by** *auto*
qed
qed

end

Function Relator **context** *transport-Fun-Rel*
begin

lemma *mono-wrt-rel-leftI*:
assumes $((\le_{R1}) \Rrightarrow_m (\le_{L1}))$ *r1*
and $((\le_{L2}) \Rrightarrow_m (\le_{R2}))$ *l2*
shows $((\le_{L}) \Rrightarrow_m (\le_{R}))$ *l*
using *assms* **by** (*intro tdfr.mono-wrt-rel-leftI*) *simp-all*

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

2.8.3 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* **.**

lemma *left-right-rel-if-left-rel-rightI*:
assumes *mono-r1*: $((\le_{R1}) \Rrightarrow_m (\le_{L1}))$ *r1*

and *half-galois-prop-left1*: $((\leq_{L1})\ {}_h\trianglelefteq\ (\leq_{R1}))$ *l1 r1*
and *refl-R1*: *reflexive-on* (*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*: $\bigwedge x'\ y'.\ x' \leq_{R1} x' \Longrightarrow$ *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*: $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$ *transitive* $(\leq_{R2\ x1'\ x2'})$
and $g \leq_R g$
and $f \leq_L r\ g$
shows $l\ f \leq_R g$
proof (*rule flip.left-relI*)
fix $x1'\ x2'$
assume [*iff*]: $x1' \leq_{R1} x2'$
with *refl-R1* **have** [*iff*]: $x1' \leq_{R1} x1'$ **by** *auto*
with *mono-r1 half-galois-prop-left1* **have** [*iff*]: $\varepsilon_1\ x1' \leq_{R1} x1'$
by (*intro t1.counit-rel-if-right-rel-if-half-galois-prop-left-if-mono-wrt-rel*)
with *refl-R1* **have** $\varepsilon_1\ x1' \leq_{R1} \varepsilon_1\ x1'$ **by** *blast*
with ‹$g \leq_R g$› **have** $g\ (\varepsilon_1\ x1') \leq_{R2\ (\varepsilon_1\ x1')\ (\varepsilon_1\ x1')} g\ (\varepsilon_1\ x1')$ **by** *blast*
with *R2-le2* **have** $g\ (\varepsilon_1\ x1') \leq_{R2\ (\varepsilon_1\ x1')\ x1'} g\ (\varepsilon_1\ x1')$ **by** *blast*

let $?x1 = r1\ x1'$
from ‹$f \leq_L r\ g$› ‹$x1' \leq_{R1} x1'$› **have** $f\ ?x1 \leq_{L2\ ?x1\ ?x1} r\ g\ ?x1$ **using** *mono-r1* **by** *blast*
then have $f\ ?x1 \leq_{L2\ ?x1\ ?x1} r2_{\ ?x1\ (\varepsilon_1\ x1')}\ (g\ (\varepsilon_1\ x1'))$ **by** *simp*
with *ge-L2-r2-le2* **have** $f\ ?x1 \leq_{L2\ ?x1\ ?x1} r2_{\ ?x1\ x1'}\ (g\ (\varepsilon_1\ x1'))$
using ‹- $\leq_{R2\ (\varepsilon_1\ x1')\ x1'} g\ (\varepsilon_1\ x1')$› **by** *blast*
with *half-galois-prop-left2* **have** $l2_{\ x1'\ ?x1}\ (f\ ?x1) \leq_{R2\ (\varepsilon_1\ x1')\ x1'} g\ (\varepsilon_1\ x1')$
using ‹- $\leq_{R2\ (\varepsilon_1\ x1')\ x1'} g\ (\varepsilon_1\ x1')$› **by** *auto*
moreover from ‹$g \leq_R g$› ‹$\varepsilon_1\ x1' \leq_{R1} x1'$› **have** ... $\leq_{R2\ (\varepsilon_1\ x1')\ x1'} g\ x1'$ **by** *blast*
ultimately have $l2_{\ x1'\ ?x1}\ (f\ ?x1) \leq_{R2\ (\varepsilon_1\ x1')\ x1'} g\ x1'$ **using** *trans-R2* **by** *blast*
with *R2-le1 R2-le2* **have** $l2_{\ x1'\ ?x1}\ (f\ ?x1) \leq_{R2\ x1'\ x2'} g\ x1'$ **by** *blast*
moreover from ‹$g \leq_R g$› ‹$x1' \leq_{R1} x2'$› **have** ... $\leq_{R2\ x1'\ x2'} g\ x2'$ **by** *blast*
ultimately have $l2_{\ x1'\ ?x1}\ (f\ ?x1) \leq_{R2\ x1'\ x2'} g\ x2'$ **using** *trans-R2* **by** *blast*
then show $l\ f\ x1' \leq_{R2\ x1'\ x2'} g\ x2'$ **by** *simp*
qed

lemma *left-right-rel-if-left-rel-right-ge-left2-assmI*:
assumes *mono-r1*: $((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))$ *r1*
and $((\leq_{L1})\ {}_h\trianglelefteq\ (\leq_{R1}))$ *l1 r1*
and ([*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 $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$

and $x' \le_{R1} x'$
and *in-codom* $(\le_{R2\ (\varepsilon_1\ x')\ x'})\ y'$
shows $(\ge_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')}\ y') \le (\ge_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ x'}\ y')$
using *dep-mono-wrt-relD*[*OF mono-r1* ‹$x' \le_{R1} x'$›] *assms*(2−4,6)
by (*blast dest*!: *t1.half-galois-prop-leftD*)

interpretation *flip-inv* :
transport-Dep-Fun-Rel $(\ge_{R1})\ (\ge_{L1})$ *r1 l1 flip2 R2 flip2 L2 r2 l2*
rewrites *flip-inv.L* $\equiv (\ge_R)$ **and** *flip-inv.R* $\equiv (\ge_L)$
and *flip-inv.t1.counit* $\equiv \eta_1$
and $\bigwedge R\ x\ y.$ (*flip2* $R\ x\ y)^{-1} \equiv R\ y\ x$
and $\bigwedge R.$ *in-dom* $R^{-1} \equiv$ *in-codom* R
and $\bigwedge R\ x1\ x2.$ *in-codom* (*flip2* $R\ x1\ x2) \equiv$ *in-dom* $(R\ x2\ x1)$
and $\bigwedge R\ S.\ (R^{-1} \Rrightarrow_m S^{-1}) \equiv (R \Rrightarrow_m S)$
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* $L2\ x1\ x2 \le$ *flip2* $L2\ x3\ x4 \equiv (\le_{L2\ x2\ x1}) \le (\le_{L2\ x4\ x3})$
and $\bigwedge(R :: {}'z \Rightarrow \text{-})\ (P :: {}'z \Rightarrow bool).$ *reflexive-on* $P\ R^{-1} \equiv$ *reflexive-on* $P\ R$
and $\bigwedge R\ x1\ x2.$ *transitive* (*flip2* $R\ x1\ x2) \equiv$ *transitive* $(R\ x2\ x1)$
and $\bigwedge x\ x.$ ([*in-dom* $(\le_{L2\ x'\ \eta_1\ x'})] \Rrightarrow$ *flip2* $R2\ (l1\ x')\ (l1\ x'))$
$\equiv$ ([*in-dom* $(\le_{L2\ x'\ \eta_1\ x'})] \Rrightarrow (\le_{R2\ (l1\ x')\ (l1\ x')}))^{-1}$
by (*simp-all add*: *flip-inv-left-eq-ge-right flip-inv-right-eq-ge-left*
t1.flip-counit-eq-unit
galois-prop.rel-inv-half-galois-prop-right-eq-half-galois-prop-left-rel-inv)

lemma *left-rel-right-if-left-right-relI*:
assumes $((\le_{L1}) \Rrightarrow_m (\le_{R1}))\ l1$
and $((\le_{L1}) \trianglelefteq_h (\le_{R1}))\ l1\ r1$
and *reflexive-on* (*in-codom* $(\le_{L1}))\ (\le_{L1})$
and $\bigwedge x.\ x \le_{L1} x \Longrightarrow ((\le_{L2\ x\ (\eta_1\ x)}) \trianglelefteq_h (\le_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (r2_{x\ (l1\ x)})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x2\ x2}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x.\ x \le_{L1} x \Longrightarrow (\le_{L2\ x\ (\eta_1\ x)}) \le (\le_{L2\ x\ x})$
and $\bigwedge x\ y.\ x \le_{L1} x \Longrightarrow$ *in-dom* $(\le_{L2\ x\ (\eta_1\ x)})\ y \Longrightarrow$
$(\le_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ (\eta_1\ x)}\ y) \le (\le_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ x}\ y)$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow$ *transitive* $(\le_{L2\ x1\ x2})$
and $f \le_L f$
and $l\ f \le_R g$
shows $f \le_L r\ g$
using *assms*
by (*intro flip-inv.left-right-rel-if-left-rel-rightI*[*simplified rel-inv-iff-rel*])

lemma *left-rel-right-if-left-right-rel-le-right2-assmI*:
assumes $((\le_{L1}) \Rrightarrow_m (\le_{R1}))\ l1$
and $((\le_{L1}) \trianglelefteq_h (\le_{R1}))^{-1}\ r1\ l1$
and ([*in-dom* $(\le_{L2\ x\ (\eta_1\ x)})] \Rrightarrow (\le_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (l2_{(l1\ x)\ (\eta_1\ x)})$

and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow$ *transitive* $(\le_{R2\ x1'\ x2'})$
and $x \le_{L1} x$
and *in-dom* $(\le_{L2\ x\ (\eta_1\ x)})\ y$
shows $(\le_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ (\eta_1\ x)}\ y) \le (\le_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ x}\ y)$
using *assms* **by** (*intro flip-inv.left-right-rel-if-left-rel-right-ge-left2-assmI* [*simplified rel-inv-iff-rel*])
auto

end

lemma *left-rel-right-iff-left-right-relI*:
assumes $((\le_{L1}) \dashv (\le_{R1}))\ l1\ r1$
and *reflexive-on* (*in-codom* $(\le_{L1})) (\le_{L1})$
and *reflexive-on* (*in-dom* $(\le_{R1})) (\le_{R1})$
and $\bigwedge x'.\ x' \le_{R1} x' \Longrightarrow$
$((\le_{L2\ (r1\ x')\ (r1\ x')})\ {}_h\trianglelefteq\ (\le_{R2\ (\varepsilon_1\ x')\ x'}))\ (l2_{\ x'\ (r1\ x')})\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x.\ x \le_{L1} x \Longrightarrow ((\le_{L2\ x\ (\eta_1\ x)}) \trianglelefteq_h (\le_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (r2_{x\ (l1\ x)})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x2\ x2}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x.\ x \le_{L1} x \Longrightarrow (\le_{L2\ x\ (\eta_1\ x)}) \le (\le_{L2\ x\ x})$
and $\bigwedge x'.\ x' \le_{R1} x' \Longrightarrow (\le_{R2\ (\varepsilon_1\ x')\ x'}) \le (\le_{R2\ x'\ x'})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ x1'\ x1'}) \le (\le_{R2\ x1'\ x2'})$
and $\bigwedge x\ y.\ x \le_{L1} x \Longrightarrow$ *in-dom* $(\le_{L2\ x\ (\eta_1\ x)})\ y \Longrightarrow$
$(\le_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ (\eta_1\ x)}\ y) \le (\le_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ x}\ y)$
and $\bigwedge x'\ y'.\ x' \le_{R1} x' \Longrightarrow$ *in-codom* $(\le_{R2\ (\varepsilon_1\ x')\ x'})\ y' \Longrightarrow$
$(\ge_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')}\ y') \le (\ge_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ x'}\ y')$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow$ *transitive* $(\le_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow$ *transitive* $(\le_{R2\ x1'\ x2'})$
and $f \le_L f$
and $g \le_R g$
shows $f \le_L r\ g \longleftrightarrow l\ f \le_R g$
using *assms* **by** (*intro iffI left-right-rel-if-left-rel-rightI*)
(*auto intro*!: *left-rel-right-if-left-right-relI*)

lemma *half-galois-prop-left2-if-half-galois-prop-left2-if-left-GaloisI*:
assumes $((\le_{R1}) \Rrightarrow_m (\le_{L1}))\ r1$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\le_{L2\ x\ (r1\ x')})\ {}_h\trianglelefteq\ (\le_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $x' \le_{R1} x'$
shows $((\le_{L2\ (r1\ x')\ (r1\ x')})\ {}_h\trianglelefteq\ (\le_{R2\ (\varepsilon_1\ x')\ x'}))\ (l2_{\ x'\ (r1\ x')})\ (r2_{(r1\ x')\ x'})$
using *assms* **by** (*auto intro*: *t1.right-left-Galois-if-right-relI*)

lemma *half-galois-prop-right2-if-half-galois-prop-right2-if-left-GaloisI*:
assumes $((\le_{L1}) \Rrightarrow_m (\le_{R1}))\ l1$
and $((\le_{L1}) \trianglelefteq_h (\le_{R1}))\ l1\ r1$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\le_{L2\ x\ (r1\ x')}) \trianglelefteq_h (\le_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $x \le_{L1} x$
shows $((\le_{L2\ x\ (\eta_1\ x)}) \trianglelefteq_h (\le_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (r2_{x\ (l1\ x)})$

by (*auto intro*!: *assms t1.left-Galois-left-if-left-relI*)

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 *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 $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$
$([\mathit{in\text{-}dom}\ (\leq_{L2\ x\ (\eta_1\ x)})] \Rrightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (l2_{(l1\ x)\ (\eta_1\ x)})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$
$([\mathit{in\text{-}codom}\ (\leq_{R2\ (\varepsilon_1\ x')\ x'})] \Rrightarrow (\leq_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *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 −
from *galois-prop2* **have**
$((\leq_{L2\ x\ (r1\ x')})\ {}_h\trianglelefteq\ (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
$((\leq_{L2\ x\ (r1\ x')}) \trianglelefteq_h (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
if $x\ {}_{L1}\lessapprox\ x'$ **for** $x\ x'$
using ‹$x\ {}_{L1}\lessapprox\ x'$› **by** *blast*+
with *assms* **show** *?thesis*
by (*intro left-rel-right-iff-left-right-relI*
left-right-rel-if-left-rel-right-ge-left2-assmI
left-rel-right-if-left-right-rel-le-right2-assmI
half-galois-prop-left2-if-half-galois-prop-left2-if-left-GaloisI
half-galois-prop-right2-if-half-galois-prop-right2-if-left-GaloisI)
auto
qed

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-field* $(\leq_{L1})$) $(\leq_{L1})$
and *antimono-L2*:
$([x1\ x2 :: (\leq_{L1})] \Rrightarrow_m [x3\ x4 :: (\leq_{L1}) \mid (x2 \leq_{L1} x3 \wedge x4 \leq_{L1} \eta_1\ x3)] \Rrightarrow (\geq))$
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 −
fix *x1 x2* **assume** $x1 \leq_{L1} x2$
with *galois-conn1 refl-L1* **have** $x1 \leq_{L1} x1$ $x2 \leq_{L1} \eta_1\ x2$
by (*blast intro*:

t1.rel-unit-if-left-rel-if-half-galois-prop-right-if-mono-wrt-rel)+
moreover with *refl-L1* **have** $x2 \le_{L1} x2$ $\eta_1\ x2 \le_{L1} \eta_1\ x2$ **by** *auto*
moreover note *dep-mono-wrt-relD*[*OF antimono-L2* ‹$x1 \le_{L1} x2$›]
and *dep-mono-wrt-relD*[*OF antimono-L2* ‹$x1 \le_{L1} x1$›]
ultimately show $(\le_{L2\ x2\ x2}) \le (\le_{L2\ x1\ x2})$ $(\le_{L2\ x1\ (\eta_1\ x2)}) \le (\le_{L2\ x1\ x2})$
using ‹$x1 \le_{L1} x2$› **by** *auto*
qed

lemma *left-rel-right-iff-left-right-rel-if-galois-prop-le-assms-rightI*:
assumes *galois-conn1*: $((\le_{L1}) \dashv (\le_{R1}))\ l1\ r1$
and *refl-R1*: *reflexive-on* (*in-field* $(\le_{R1})$) $(\le_{R1})$
and *mono-R2*:
$([x1'\ x2' :: (\le_{R1}) \mid \varepsilon_1\ x2' \le_{R1} x1'] \Rrightarrow_m [x3'\ x4' :: (\le_{R1}) \mid x2' \le_{R1} x3'] \Rrightarrow (\le))\ R2$
shows $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ (\varepsilon_1\ x1')\ x2'}) \le (\le_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ x1'\ x1'}) \le (\le_{R2\ x1'\ x2'})$
proof −
fix $x1'\ x2'$ **assume** $x1' \le_{R1} x2'$
with *galois-conn1 refl-R1* **have** $x2' \le_{R1} x2'$ $\varepsilon_1\ x1' \le_{R1} x1'$
by (*blast intro*:
t1.counit-rel-if-right-rel-if-half-galois-prop-left-if-mono-wrt-rel)+
moreover with *refl-R1* **have** $x1' \le_{R1} x1'$ $\varepsilon_1\ x1' \le_{R1} \varepsilon_1\ x1'$ **by** *auto*
moreover note *dep-mono-wrt-relD*[*OF mono-R2* ‹$\varepsilon_1\ x1' \le_{R1} x1'$›]
and *dep-mono-wrt-relD*[*OF mono-R2* ‹$x1' \le_{R1} x1'$›]
ultimately show $(\le_{R2\ (\varepsilon_1\ x1')\ x2'}) \le (\le_{R2\ x1'\ x2'})$ $(\le_{R2\ x1'\ x1'}) \le (\le_{R2\ x1'\ x2'})$
using ‹$x1' \le_{R1} x2'$› **by** *auto*
qed

corollary *left-rel-right-iff-left-right-rel-if-monoI*:
assumes $((\le_{L1}) \dashv (\le_{R1}))\ l1\ r1$
and *reflexive-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$
and *reflexive-on* (*in-field* $(\le_{R1})$) $(\le_{R1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\le_{L2\ x\ (r1\ x')}) \trianglelefteq (\le_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x3\ x4 :: (\le_{L1}) \mid (x2 \le_{L1} x3 \wedge x4 \le_{L1} \eta_1\ x3)] \Rrightarrow (\ge))\ L2$
and $([x1'\ x2' :: (\le_{R1}) \mid \varepsilon_1\ x2' \le_{R1} x1'] \Rrightarrow_m [x3'\ x4' :: (\le_{R1}) \mid x2' \le_{R1} x3'] \Rrightarrow (\le))\ R2$
and $\bigwedge x.\ x \le_{L1} x \Longrightarrow$
$([\textit{in-dom}\ (\le_{L2\ x\ (\eta_1\ x)})] \Rrightarrow (\le_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (l2_{(l1\ x)\ (\eta_1\ x)})$
and $\bigwedge x'.\ x' \le_{R1} x' \Longrightarrow$
$([\textit{in-codom}\ (\le_{R2\ (\varepsilon_1\ x')\ x'})] \Rrightarrow (\le_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow$ *transitive* $(\le_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow$ *transitive* $(\le_{R2\ x1'\ x2'})$
and $f \le_L f$
and $g \le_R g$
shows $f \le_L r\ g \longleftrightarrow l\ f \le_R g$
using *assms* **by** (*intro left-rel-right-iff-left-right-relI'*
left-rel-right-iff-left-right-rel-if-galois-prop-le-assms-leftI

left-rel-right-iff-left-right-rel-if-galois-prop-le-assms-rightI)
(*auto intro*: *reflexive-on-if-le-pred-if-reflexive-on*
in-field-if-in-dom in-field-if-in-codom)

end

Function Relator **context** *transport-Fun-Rel*
begin

corollary *left-right-rel-if-left-rel-rightI*:
assumes $((\le_{R1}) \Rrightarrow_m (\le_{L1}))$ *r1*
and $((\le_{L1})\ {}_h\trianglelefteq\ (\le_{R1}))$ *l1 r1*
and *reflexive-on* (*in-dom* $(\le_{R1})$) $(\le_{R1})$
and $((\le_{L2})\ {}_h\trianglelefteq\ (\le_{R2}))$ *l2 r2*
and *transitive* $(\le_{R2})$
and $g \le_R g$
and $f \le_L r\ g$
shows $l\ f \le_R g$
using *assms* **by** (*intro tdfr.left-right-rel-if-left-rel-rightI*) *simp-all*

corollary *left-rel-right-if-left-right-relI*:
assumes $((\le_{L1}) \Rrightarrow_m (\le_{R1}))$ *l1*
and $((\le_{L1}) \trianglelefteq_h (\le_{R1}))$ *l1 r1*
and *reflexive-on* (*in-codom* $(\le_{L1})$) $(\le_{L1})$
and $((\le_{L2}) \trianglelefteq_h (\le_{R2}))$ *l2 r2*
and *transitive* $(\le_{L2})$
and $f \le_L f$
and $l\ f \le_R g$
shows $f \le_L r\ g$
using *assms* **by** (*intro tdfr.left-rel-right-if-left-right-relI*) *simp-all*

corollary *left-rel-right-iff-left-right-relI*:
assumes $((\le_{L1}) \dashv (\le_{R1}))$ *l1 r1*
and *reflexive-on* (*in-codom* $(\le_{L1})$) $(\le_{L1})$
and *reflexive-on* (*in-dom* $(\le_{R1})$) $(\le_{R1})$
and $((\le_{L2}) \trianglelefteq (\le_{R2}))$ *l2 r2*
and *transitive* $(\le_{L2})$
and *transitive* $(\le_{R2})$
and $f \le_L f$
and $g \le_R g$
shows $f \le_L r\ g \longleftrightarrow l\ f \le_R g$
using *assms* **by** (*intro tdfr.left-rel-right-iff-left-right-relI*) *auto*

end

Monotone Dependent Function Relator **context** *transport-Mono-Dep-Fun-Rel*
begin

lemma *half-galois-prop-left-left-rightI*:

assumes $(tdfr.L \Rrightarrow_m tdfr.R)\ l$
and $((\le_{R1}) \Rrightarrow_m (\le_{L1}))\ r1$
and $((\le_{L1})\ {}_h\trianglelefteq (\le_{R1}))\ l1\ r1$
and *reflexive-on* $(in\text{-}dom\ (\le_{R1}))\ (\le_{R1})$
and $\bigwedge x'.\ x' \le_{R1} x' \Longrightarrow$
$((\le_{L2\ (r1\ x')\ (r1\ x')})\ {}_h\trianglelefteq (\le_{R2\ (\varepsilon_1\ x')\ x'}))\ (l2_{\ x'\ (r1\ x')})\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x'.\ x' \le_{R1} x' \Longrightarrow (\le_{R2\ (\varepsilon_1\ x')\ x'}) \le (\le_{R2\ x'\ x'})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ x1'\ x1'}) \le (\le_{R2\ x1'\ x2'})$
and $\bigwedge x'\ y'.\ x' \le_{R1} x' \Longrightarrow in\text{-}codom\ (\le_{R2\ (\varepsilon_1\ x')\ x'})\ y' \Longrightarrow$
$(\ge_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')}\ y') \le (\ge_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ x'}$
$y')$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow transitive\ (\le_{R2\ x1'\ x2'})$
shows $((\le_L)\ {}_h\trianglelefteq (\le_R))\ l\ r$
unfolding *left-rel-eq-tdfr-left-Refl-Rel right-rel-eq-tdfr-right-Refl-Rel* **using** *assms*
by (*intro*
half-galois-prop-leftI[*unfolded left-rel-eq-tdfr-left-Refl-Rel right-rel-eq-tdfr-right-Refl-Rel*]
Refl-Rel-app-leftI[**where** *?f=l*]
tdfr.left-right-rel-if-left-rel-rightI)
(*auto elim*!: *galois-rel.left-GaloisE*)

lemma *half-galois-prop-right-left-rightI*:
assumes $(tdfr.R \Rrightarrow_m tdfr.L)\ r$
and $((\le_{L1}) \Rrightarrow_m (\le_{R1}))\ l1$
and $((\le_{L1}) \trianglelefteq_h (\le_{R1}))\ l1\ r1$
and *reflexive-on* $(in\text{-}codom\ (\le_{L1}))\ (\le_{L1})$
and $\bigwedge x.\ x \le_{L1} x \Longrightarrow ((\le_{L2\ x\ (\eta_1\ x)}) \trianglelefteq_h (\le_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (r2_{x\ (l1\ x)})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x2\ x2}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x.\ x \le_{L1} x \Longrightarrow (\le_{L2\ x\ (\eta_1\ x)}) \le (\le_{L2\ x\ x})$
and $\bigwedge x\ y.\ x \le_{L1} x \Longrightarrow in\text{-}dom\ (\le_{L2\ x\ (\eta_1\ x)})\ y \Longrightarrow$
$(\le_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ (\eta_1\ x)}\ y) \le (\le_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ x}\ y)$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow transitive\ (\le_{L2\ x1\ x2})$
shows $((\le_L) \trianglelefteq_h (\le_R))\ l\ r$
unfolding *left-rel-eq-tdfr-left-Refl-Rel right-rel-eq-tdfr-right-Refl-Rel* **using** *assms*
by (*intro*
half-galois-prop-rightI[*unfolded left-rel-eq-tdfr-left-Refl-Rel right-rel-eq-tdfr-right-Refl-Rel*]
Refl-Rel-app-rightI[**where** *?f=r*]
tdfr.left-rel-right-if-left-right-relI)
(*auto elim*!: *galois-rel.left-GaloisE in-codomE Refl-RelE intro*!: *in-fieldI*)

corollary *galois-prop-left-rightI*:
assumes $(tdfr.L \Rrightarrow_m tdfr.R)\ l$ **and** $(tdfr.R \Rrightarrow_m tdfr.L)\ r$
and $((\le_{L1}) \dashv (\le_{R1}))\ l1\ r1$
and *reflexive-on* $(in\text{-}codom\ (\le_{L1}))\ (\le_{L1})$
and *reflexive-on* $(in\text{-}dom\ (\le_{R1}))\ (\le_{R1})$
and $\bigwedge x'.\ x' \le_{R1} x' \Longrightarrow$
$((\le_{L2\ (r1\ x')\ (r1\ x')})\ {}_h\trianglelefteq (\le_{R2\ (\varepsilon_1\ x')\ x'}))\ (l2_{\ x'\ (r1\ x')})\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x.\ x \le_{L1} x \Longrightarrow ((\le_{L2\ x\ (\eta_1\ x)}) \trianglelefteq_h (\le_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (r2_{x\ (l1\ x)})$

and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x2\ x2}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x.\ x \le_{L1} x \Longrightarrow (\le_{L2\ x\ (\eta_1\ x)}) \le (\le_{L2\ x\ x})$
and $\bigwedge x'.\ x' \le_{R1} x' \Longrightarrow (\le_{R2\ (\varepsilon_1\ x')\ x'}) \le (\le_{R2\ x'\ x'})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ x1'\ x1'}) \le (\le_{R2\ x1'\ x2'})$
and $\bigwedge x\ y.\ x \le_{L1} x \Longrightarrow$ *in-dom* $(\le_{L2\ x\ (\eta_1\ x)})\ y \Longrightarrow$
$(\le_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ (\eta_1\ x)}\ y) \le (\le_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ x}\ y)$
and $\bigwedge x'\ y'.\ x' \le_{R1} x' \Longrightarrow$ *in-codom* $(\le_{R2\ (\varepsilon_1\ x')\ x'})\ y' \Longrightarrow$
$(\ge_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')}\ y') \le (\ge_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ x'}$
$y')$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow$ *transitive* $(\le_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow$ *transitive* $(\le_{R2\ x1'\ x2'})$
shows $((\le_L) \trianglelefteq (\le_R))\ l\ r$
using *assms* **by** (*intro galois-propI half-galois-prop-left-left-rightI*
half-galois-prop-right-left-rightI)
auto

corollary *galois-prop-left-rightI'*:
assumes $(tdfr.L \Rrightarrow_m tdfr.R)\ l$ **and** $(tdfr.R \Rrightarrow_m tdfr.L)\ r$
and $((\le_{L1}) \dashv (\le_{R1}))\ l1\ r1$
and *reflexive-on* (*in-codom* $(\le_{L1})$) $(\le_{L1})$
and *reflexive-on* (*in-dom* $(\le_{R1})$) $(\le_{R1})$
and *galois-prop2*: $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow$
$((\le_{L2\ x\ (r1\ x')}) \trianglelefteq (\le_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x2\ x2}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x.\ x \le_{L1} x \Longrightarrow (\le_{L2\ x\ (\eta_1\ x)}) \le (\le_{L2\ x\ x})$
and $\bigwedge x'.\ x' \le_{R1} x' \Longrightarrow (\le_{R2\ (\varepsilon_1\ x')\ x'}) \le (\le_{R2\ x'\ x'})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ x1'\ x1'}) \le (\le_{R2\ x1'\ x2'})$
and $\bigwedge x.\ x \le_{L1} x \Longrightarrow$
$([in\text{-}dom\ (\le_{L2\ x\ (\eta_1\ x)})] \Rrightarrow (\le_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (l2_{(l1\ x)\ (\eta_1\ x)})$
and $\bigwedge x'.\ x' \le_{R1} x' \Longrightarrow$
$([in\text{-}codom\ (\le_{R2\ (\varepsilon_1\ x')\ x'})] \Rrightarrow (\le_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow$ *transitive* $(\le_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow$ *transitive* $(\le_{R2\ x1'\ x2'})$
shows $((\le_L) \trianglelefteq (\le_R))\ l\ r$
proof –
from *galois-prop2* **have**
$((\le_{L2\ x\ (r1\ x')})\ {}_h\trianglelefteq\ (\le_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
$((\le_{L2\ x\ (r1\ x')}) \trianglelefteq_h (\le_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
if $x\ {}_{L1}\lessapprox x'$ **for** $x\ x'$
using ‹$x\ {}_{L1}\lessapprox x'$› **by** *blast+*
with *assms* **show** *?thesis* **by** (*intro galois-prop-left-rightI*
tdfr.left-right-rel-if-left-rel-right-ge-left2-assmI
tdfr.left-rel-right-if-left-right-rel-le-right2-assmI
tdfr.half-galois-prop-left2-if-half-galois-prop-left2-if-left-GaloisI
tdfr.half-galois-prop-right2-if-half-galois-prop-right2-if-left-GaloisI)
auto
qed

corollary *galois-prop-left-right-if-mono-if-galois-propI*:
  assumes $(tdfr.L \Rrightarrow_m tdfr.R)\ l$ **and** $(tdfr.R \Rrightarrow_m tdfr.L)\ r$
  and $((\le_{L1}) \dashv (\le_{R1}))\ l1\ r1$
  and *reflexive-on* $(in\text{-}field\ (\le_{L1}))\ (\le_{L1})$
  and *reflexive-on* $(in\text{-}field\ (\le_{R1}))\ (\le_{R1})$
  and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\le_{L2\ x\ (r1\ x')}) \trianglelefteq (\le_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
  and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x3\ x4 :: (\le_{L1}) \mid (x2 \le_{L1} x3 \wedge x4 \le_{L1} \eta_1\ x3)] \Rrightarrow (\ge))\ L2$
  and $([x1'\ x2' :: (\le_{R1}) \mid \varepsilon_1\ x2' \le_{R1} x1'] \Rrightarrow_m [x3'\ x4' :: (\le_{R1}) \mid x2' \le_{R1} x3'] \Rrightarrow (\le))\ R2$
  and $\bigwedge x.\ x \le_{L1} x \Longrightarrow$
    $([in\text{-}dom\ (\le_{L2\ x\ (\eta_1\ x)})] \Rrightarrow (\le_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (l2_{(l1\ x)\ (\eta_1\ x)})$
  and $\bigwedge x'.\ x' \le_{R1} x' \Longrightarrow$
    $([in\text{-}codom\ (\le_{R2\ (\varepsilon_1\ x')\ x'})] \Rrightarrow (\le_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})\ (r2_{(r1\ x')\ x'})$
  and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow$ *transitive* $(\le_{L2\ x1\ x2})$
  and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow$ *transitive* $(\le_{R2\ x1'\ x2'})$
  shows $((\le_L) \trianglelefteq (\le_R))\ l\ r$
  using *assms* **by** (*intro galois-prop-left-rightI'*
    tdfr.left-rel-right-iff-left-right-rel-if-galois-prop-le-assms-leftI
    tdfr.left-rel-right-iff-left-right-rel-if-galois-prop-le-assms-rightI)
  (*auto intro*: *reflexive-on-if-le-pred-if-reflexive-on*
    in-field-if-in-dom in-field-if-in-codom)

Note that we could further rewrite $[\![(tdfr.L \Rrightarrow_m tdfr.R)\ l;\ (tdfr.R \Rrightarrow_m tdfr.L)\ r;\ t1.galois\text{-}connection;\ reflexive\text{-}on\ (in\text{-}field\ (\le_{L1}))\ (\le_{L1});\ reflexive\text{-}on\ (in\text{-}field\ (\le_{R1}))\ (\le_{R1});\ \bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow t2.galois\text{-}prop\ x\ x'\ l2_{x'\ x}\ r2_{x\ x'};\ ([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x3\ x4 :: (\le_{L1})] \Rrightarrow (x2 \le_{L1} x3 \wedge x4 \le_{L1} \eta_1\ x3) \longrightarrow (\lambda x\ y.\ y \le x))\ L2;\ ([x1'\ x2' :: (\le_{R1})] \Rrightarrow_m \varepsilon_1\ x2' \le_{R1} x1' \longrightarrow ([x3'\ x4' :: (\le_{R1})] \Rrightarrow x2' \le_{R1} x3' \longrightarrow (\le)))\ R2;\ \bigwedge x.\ x \le_{L1} x \Longrightarrow ([in\text{-}dom\ (\le_{L2\ x\ \eta_1\ x})] \Rrightarrow \le_{R2\ l1\ x\ l1\ x})\ l2_{l1\ x\ x}\ l2_{l1\ x\ \eta_1\ x};\ \bigwedge x'.\ x' \le_{R1} x' \Longrightarrow ([in\text{-}codom\ (\le_{R2\ \varepsilon_1\ x'\ x'})] \Rrightarrow \le_{L2\ r1\ x'\ r1\ x'})\ r2_{r1\ x'\ \varepsilon_1\ x'}\ r2_{r1\ x'\ x'};\ \bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow transitive\ (\le_{L2\ x1\ x2});\ \bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow transitive\ (\le_{R2\ x1'\ x2'})]\!] \Longrightarrow galois\text{-}prop\ l\ r$, as we will do later for Galois connections, by applying $[\![((\le_{R1}) \Rrightarrow_m (\le_{L1}))\ r1;\ \bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{L2\ r1\ x1'\ r1\ x2'} \Rrightarrow_m \le_{R2\ \varepsilon_1\ x1'\ x2'})\ l2_{x2'\ r1\ x1'};\ \bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ \varepsilon_1\ x1'\ x2'}) \le (\le_{R2\ x1'\ x2'});\ \bigwedge x1'\ x2'\ y.\ [\![x1' \le_{R1} x2';\ in\text{-}dom\ (\le_{L2\ r1\ x1'\ r1\ x2'})\ y]\!] \Longrightarrow dfro2.right\text{-}infix\ (l2_{x2'\ r1\ x1'}\ y)\ x1'\ x2' \le dfro2.right\text{-}infix\ (l2_{x1'\ r1\ x1'}\ y)\ x1'\ x2';\ \bigwedge x1'\ x2'\ y.\ [\![x1' \le_{R1} x2';\ in\text{-}codom\ (\le_{L2\ r1\ x1'\ r1\ x2'})\ y]\!] \Longrightarrow (\le_{R2\ x1'\ x2'})^{-1}\ (l2_{x2'\ r1\ x1'}\ y) \le (\le_{R2\ x1'\ x2'})^{-1}\ (l2_{x2'\ r1\ x2'}\ y)]\!] \Longrightarrow (tdfr.L \Rrightarrow_m tdfr.R)\ l$ and $[\![((\le_{L1}) \Rrightarrow_m (\le_{R1}))\ l1;\ \bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{R2\ l1\ x1\ l1\ x2} \Rrightarrow_m \le_{L2\ x1\ \eta_1\ x2})\ r2_{x1\ l1\ x2};\ \bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ \eta_1\ x2}) \le (\le_{L2\ x1\ x2});\ \bigwedge x1\ x2\ y'.\ [\![x1 \le_{L1} x2;\ in\text{-}codom\ (\le_{R2\ l1\ x1\ l1\ x2})\ y']\!] \Longrightarrow (\le_{L2\ x1\ x2})^{-1}\ (r2_{x1\ l1\ x2}\ y') \le (\le_{L2\ x1\ x2})^{-1}\ (r2_{x2\ l1\ x2}\ y');\ \bigwedge x1\ x2\ y'.\ [\![x1 \le_{L1} x2;\ in\text{-}dom\ (\le_{R2\ l1\ x1\ l1\ x2})\ y']\!] \Longrightarrow dfro1.right\text{-}infix\ (r2_{x1\ l1\ x2}\ y')\ x1\ x2 \le dfro1.right\text{-}infix\ (r2_{x1\ l1\ x1}\ y')\ x1\ x2]\!] \Longrightarrow (tdfr.R$

$\Rrightarrow_m$ *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 $[\![((\le_{R1}) \Rrightarrow_m (\le_{L1}))\ r1;\ \bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{L2\ r1\ x1'\ r1\ x2'} \Rrightarrow_m \le_{R2\ \varepsilon_1\ x1'\ x2'})\ l2_{x2'\ r1\ x1'};\ \bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ \varepsilon_1\ x1'\ x2'}) \le (\le_{R2\ x1'\ x2'});\ \bigwedge x1'\ x2'\ y.\ [\![x1' \le_{R1} x2';\ in\text{-}dom\ (\le_{L2\ r1\ x1'\ r1\ x2'})\ y]\!] \Longrightarrow dfro2.right\text{-}infix\ (l2_{x2'\ r1\ x1'}\ y)\ x1'\ x2' \le dfro2.right\text{-}infix\ (l2_{x1'\ r1\ x1'}\ y)\ x1'\ x2';\ \bigwedge x1'\ x2'\ y.\ [\![x1' \le_{R1} x2';\ in\text{-}codom\ (\le_{L2\ r1\ x1'\ r1\ x2'})\ y]\!] \Longrightarrow (\le_{R2\ x1'\ x2'})^{-1}\ (l2_{x2'\ r1\ x1'}\ y) \le (\le_{R2\ x1'\ x2'})^{-1}\ (l2_{x2'\ r1\ x2'}\ y)]\!] \Longrightarrow (tdfr.L \Rrightarrow_m tdfr.R)\ l$ and $[\![((\le_{L1}) \Rrightarrow_m (\le_{R1}))\ l1;\ \bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{R2\ l1\ x1\ l1\ x2} \Rrightarrow_m \le_{L2\ x1\ \eta_1\ x2})\ r2_{x1\ l1\ x2};\ \bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ \eta_1\ x2}) \le (\le_{L2\ x1\ x2});\ \bigwedge x1\ x2\ y'.\ [\![x1 \le_{L1} x2;\ in\text{-}codom\ (\le_{R2\ l1\ x1\ l1\ x2})\ y']\!] \Longrightarrow (\le_{L2\ x1\ x2})^{-1}\ (r2_{x1\ l1\ x2}\ y') \le (\le_{L2\ x1\ x2})^{-1}\ (r2_{x2\ l1\ x2}\ y');\ \bigwedge x1\ x2\ y'.\ [\![x1 \le_{L1} x2;\ in\text{-}dom\ (\le_{R2\ l1\ x1\ l1\ x2})\ y']\!] \Longrightarrow dfro1.right\text{-}infix\ (r2_{x1\ l1\ x2}\ y')\ x1\ x2 \le dfro1.right\text{-}infix\ (r2_{x1\ l1\ x1}\ y')\ x1\ x2]\!] \Longrightarrow (tdfr.R \Rrightarrow_m tdfr.L)\ r$ 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 $((\le_{R1}) \Rrightarrow_m (\le_{L1}))\ r1$
  and $((\le_{L1})\ {}_h\!\trianglelefteq (\le_{R1}))\ l1\ r1$
  and *reflexive-on* $(in\text{-}dom\ (\le_{R1}))\ (\le_{R1})$
  and $((\le_{L2}) \Rrightarrow_m (\le_{R2}))\ l2$
  and $((\le_{L2})\ {}_h\!\trianglelefteq (\le_{R2}))\ l2\ r2$
  and *transitive* $(\le_{R2})$
  shows $((\le_L)\ {}_h\!\trianglelefteq (\le_R))\ l\ r$
  using *assms*
  by (*intro tpdfr.half-galois-prop-left-left-rightI tfr.mono-wrt-rel-leftI*)
  simp-all

interpretation *flip* : *transport-Mono-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2* **.**

lemma *half-galois-prop-right-left-rightI*:
  assumes $((\le_{L1}) \Rrightarrow_m (\le_{R1}))\ l1$
  and $((\le_{L1}) \trianglelefteq_h (\le_{R1}))\ l1\ r1$
  and *reflexive-on* $(in\text{-}codom\ (\le_{L1}))\ (\le_{L1})$
  and $((\le_{R2}) \Rrightarrow_m (\le_{L2}))\ r2$
  and $((\le_{L2}) \trianglelefteq_h (\le_{R2}))\ l2\ r2$
  and *transitive* $(\le_{L2})$
  shows $((\le_L) \trianglelefteq_h (\le_R))\ l\ r$
  using *assms*
  by (*intro tpdfr.half-galois-prop-right-left-rightI flip.tfr.mono-wrt-rel-leftI*)
  simp-all

corollary *galois-prop-left-rightI*:
assumes $((\le_{L1}) \dashv (\le_{R1}))$ *l1 r1*
and *reflexive-on* (*in-codom* $(\le_{L1})$) $(\le_{L1})$
and *reflexive-on* (*in-dom* $(\le_{R1})$) $(\le_{R1})$
and $((\le_{L2}) \dashv (\le_{R2}))$ *l2 r2*
and *transitive* $(\le_{L2})$
and *transitive* $(\le_{R2})$
shows $((\le_{L}) \trianglelefteq (\le_{R}))$ *l r*
using *assms* **by** (*intro tpdfr.galois-propI*
half-galois-prop-left-left-rightI half-galois-prop-right-left-rightI)
auto

end

end

2.8.4 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*: $((\le_{L1}) \dashv (\le_{R1}))$ *l1 r1*
and *refl-R1*: *reflexive-on* (*in-codom* $(\le_{R1})$) $(\le_{R1})$
and *R2-le1*: $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ (\varepsilon_1\ x1')\ x2'}) \le (\le_{R2\ x1'\ x2'})$
and *mono-l2-2*: $([x' :: \textit{in-codom}\ (\le_{R1})] \Rrightarrow_m [x1\ x2 :: (\le_{L1}) \mid x2\ {}_{L1}\lessapprox x'] \Rrightarrow_m$
$[\textit{in-field}\ (\le_{L2\ x1\ (r1\ x')})] \Rrightarrow (\le_{R2\ (l1\ x1)\ x'}))$ *l2*
shows $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow$
$([\textit{in-codom}\ (\le_{L2\ (r1\ x1')\ (r1\ x2')})] \Rrightarrow (\le_{R2\ x1'\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x2')})$
and $\bigwedge x.\ x \le_{L1} x \Longrightarrow$
$([\textit{in-dom}\ (\le_{L2\ x\ (\eta_1\ x)})] \Rrightarrow (\le_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (l2_{(l1\ x)\ (\eta_1\ x)})$
proof −
show $([\textit{in-codom}\ (\le_{L2\ (r1\ x1')\ (r1\ x2')})] \Rrightarrow (\le_{R2\ x1'\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x2')})$
if $x1' \le_{R1} x2'$ **for** $x1'\ x2'$
proof −
from *galois-conn1* ‹$x1' \le_{R1} x2'$› **have** $r1\ x1' \le_{L1} r1\ x2'$ $r1\ x2'\ {}_{L1}\lessapprox x2'$
using *refl-R1* **by** (*auto intro*: *t1.right-left-Galois-if-reflexive-onI*)
with *mono-l2-2* **show** *?thesis* **using** *R2-le1* ‹$x1' \le_{R1} x2'$› **by** *fastforce*
qed
show $([\textit{in-dom}\ (\le_{L2\ x\ (\eta_1\ x)})] \Rrightarrow (\le_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (l2_{(l1\ x)\ (\eta_1\ x)})$
if $x \le_{L1} x$ **for** x

proof −
  from *galois-conn1* ‹$x \le_{L1} x$› **have** $x \le_{L1} \eta_1\ x\ \eta_1\ x\ _{L1}\lessapprox\ l1\ x$
    by (*auto intro*!: *t1.right-left-Galois-if-right-relI*
      t1.rel-unit-if-left-rel-if-half-galois-prop-right-if-mono-wrt-rel
        [*unfolded t1.unit-eq*])
  with *mono-l2-2* **show** *?thesis* **by** *fastforce*
qed
qed

lemma *galois-connection-left-right-if-galois-connection-mono-assms-leftI*:
  assumes *galois-conn1*: $((\le_{L1}) \dashv (\le_{R1}))\ l1\ r1$
  and *refl-R1*: *reflexive-on* (*in-field* $(\le_{R1})$) $(\le_{R1})$
  and *R2-le1*: $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ (\varepsilon_1\ x1')\ x2'}) \le (\le_{R2\ x1'\ x2'})$
  and *mono-l2*: $([x1'\ x2' :: (\le_{R1})] \Rrightarrow_m [x1\ x2 :: (\le_{L1}) \mid x2\ _{L1}\lessapprox\ x1'] \Rrightarrow$
    $[\textit{in-field}\ (\le_{L2\ x1\ (r1\ x2')})] \Rrightarrow (\le_{R2\ (l1\ x1)\ x2'}))\ l2$
  shows $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow$
   $([\textit{in-dom}\ (\le_{L2\ (r1\ x1')\ (r1\ x2')})] \Rrightarrow (\le_{R2\ x1'\ x2'}))\ (l2_{x1'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x1')})$
  and $([x' :: \textit{in-codom}\ (\le_{R1})] \Rrightarrow_m [x1\ x2 :: (\le_{L1}) \mid x2\ _{L1}\lessapprox\ x'] \Rrightarrow_m$
    $[\textit{in-field}\ (\le_{L2\ x1\ (r1\ x')})] \Rrightarrow (\le_{R2\ (l1\ x1)\ x'}))\ l2$
proof −
 show $([\textit{in-dom}\ (\le_{L2\ (r1\ x1')\ (r1\ x2')})] \Rrightarrow (\le_{R2\ x1'\ x2'}))\ (l2_{x1'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x1')})$
  if $x1' \le_{R1} x2'$ **for** $x1'\ x2'$
 proof −
  from *galois-conn1* ‹$x1' \le_{R1} x2'$› **have** $r1\ x1' \le_{L1} r1\ x1'\ r1\ x1'\ _{L1}\lessapprox\ x1'$
    using *refl-R1* **by** *blast*+
  with *mono-l2* **show** *?thesis* **using** ‹$x1' \le_{R1} x2'$› *R2-le1* **by** (*auto 9 0*)
 qed
 from *mono-l2* **show** $([x' :: \textit{in-codom}\ (\le_{R1})] \Rrightarrow_m [x1\ x2 :: (\le_{L1}) \mid x2\ _{L1}\lessapprox\ x']$
$\Rrightarrow_m$
  $[\textit{in-field}\ (\le_{L2\ x1\ (r1\ x')})] \Rrightarrow (\le_{R2\ (l1\ x1)\ x'}))\ l2$ **using** *refl-R1* **by** *blast*
qed

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*: $((\le_{L1}) \dashv (\le_{R1}))\ l1\ r1$
  and *refl-L1*: *reflexive-on* (*in-dom* $(\le_{L1})$) $(\le_{L1})$
  and *L2-le2*: $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ (\eta_1\ x2)}) \le (\le_{L2\ x1\ x2})$
  and *mono-r2-2*: $([x :: \textit{in-dom}\ (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x\ _{L1}\lessapprox\ x1'] \Rrightarrow_m$
    $[\textit{in-field}\ (\le_{R2\ (l1\ x)\ x2'})] \Rrightarrow (\le_{L2\ x\ (r1\ x2')}))\ r2$
  shows $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow$
    $([\textit{in-dom}\ (\le_{R2\ (l1\ x1)\ (l1\ x2)})] \Rrightarrow (\le_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x1)})\ (r2_{x1\ (l1\ x2)})$
  and $\bigwedge x'.\ x' \le_{R1} x' \Longrightarrow$
   $([\textit{in-codom}\ (\le_{R2\ (\varepsilon_1\ x')\ x'})] \Rrightarrow (\le_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})\ (r2_{(r1\ x')\ x'})$
proof −
 show $([\textit{in-dom}\ (\le_{R2\ (l1\ x1)\ (l1\ x2)})] \Rrightarrow (\le_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x1)})\ (r2_{x1\ (l1\ x2)})$

if $x1 \le_{L1} x2$ **for** *x1 x2*
proof −
from *galois-conn1* ‹$x1 \le_{L1} x2$› **have** $x1\ {}_{L1}\lessapprox\ l1\ x1\ l1\ x1 \le_{R1} l1\ x2$
using *refl-L1* **by** (*auto intro*!: *t1.left-Galois-left-if-reflexive-on-if-half-galois-prop-rightI*)
with *mono-r2-2* **show** *?thesis* **using** *L2-le2* ‹$x1 \le_{L1} x2$› **by** (*auto 9 0*)
qed
show $([in\text{-}codom\ (\le_{R2\ (\varepsilon_1\ x')\ x'})] \Rrightarrow (\le_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})\ (r2_{(r1\ x')\ x'})$
if $x' \le_{R1} x'$ **for** x'
proof −
from *galois-conn1* ‹$x' \le_{R1} x'$› **have** $r1\ x'\ {}_{L1}\lessapprox\ \varepsilon_1\ x'\ \varepsilon_1\ x' \le_{R1} x'$
by (*auto intro*!: *t1.left-Galois-left-if-left-relI*
t1.counit-rel-if-right-rel-if-half-galois-prop-left-if-mono-wrt-rel
[*unfolded t1.counit-eq*])
with *mono-r2-2* **show** *?thesis* **by** *fastforce*
qed
qed

lemma *galois-connection-left-right-if-galois-connection-mono-assms-rightI*:
assumes *galois-conn1*: $((\le_{L1}) \dashv (\le_{R1}))\ l1\ r1$
and *refl-L1*: *reflexive-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$
and *L2-le2*: $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ (\eta_1\ x2)}) \le (\le_{L2\ x1\ x2})$
and *mono-r2*: $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[in\text{-}field\ (\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')}))\ r2$
shows $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow$
$([in\text{-}codom\ (\le_{R2\ (l1\ x1)\ (l1\ x2)})] \Rrightarrow (\le_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x2)})\ (r2_{x2\ (l1\ x2)})$
and $([x :: in\text{-}dom\ (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x\ {}_{L1}\lessapprox\ x1'] \Rrightarrow_m$
$[in\text{-}field\ (\le_{R2\ (l1\ x)\ x2'})] \Rrightarrow (\le_{L2\ x\ (r1\ x2')}))\ r2$
proof −
show $([in\text{-}codom\ (\le_{R2\ (l1\ x1)\ (l1\ x2)})] \Rrightarrow (\le_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x2)})\ (r2_{x2\ (l1\ x2)})$
if $x1 \le_{L1} x2$ **for** *x1 x2*
proof −
from *galois-conn1* ‹$x1 \le_{L1} x2$› **have** $x2\ {}_{L1}\lessapprox\ l1\ x2\ l1\ x2 \le_{R1} l1\ x2$
using *refl-L1* **by** (*blast intro*: *t1.left-Galois-left-if-reflexive-on-if-half-galois-prop-rightI*)+
with *mono-r2* **show** *?thesis* **using** ‹$x1 \le_{L1} x2$› *L2-le2* **by** *fastforce*
qed
from *mono-r2* **show** $([x :: in\text{-}dom\ (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x\ {}_{L1}\lessapprox\ x1']$
$\Rrightarrow_m$
$[in\text{-}field\ (\le_{R2\ (l1\ x)\ x2'})] \Rrightarrow (\le_{L2\ x\ (r1\ x2')}))\ r2$ **using** *refl-L1* **by** *blast*
qed

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* **.**

lemma *galois-connection-left-rightI*:

assumes $(tdfr.L \Rrightarrow_m tdfr.R)$ l **and** $(tdfr.R \Rrightarrow_m tdfr.L)$ r
and $((\le_{L1}) \dashv (\le_{R1}))$ $l1$ $r1$
and *reflexive-on* (*in-codom* $(\le_{L1})$) $(\le_{L1})$
and *reflexive-on* (*in-dom* $(\le_{R1})$) $(\le_{R1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\le_{L2\ x\ (r1\ x')}) \trianglelefteq (\le_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x2\ x2}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x.\ x \le_{L1} x \Longrightarrow (\le_{L2\ x\ (\eta_1\ x)}) \le (\le_{L2\ x\ x})$
and $\bigwedge x'.\ x' \le_{R1} x' \Longrightarrow (\le_{R2\ (\varepsilon_1\ x')\ x'}) \le (\le_{R2\ x'\ x'})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ x1'\ x1'}) \le (\le_{R2\ x1'\ x2'})$
and $\bigwedge x.\ x \le_{L1} x \Longrightarrow$
$([in\text{-}dom\ (\le_{L2\ x\ (\eta_1\ x)})] \Rrightarrow (\le_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (l2_{(l1\ x)\ (\eta_1\ x)})$
and $\bigwedge x'.\ x' \le_{R1} x' \Longrightarrow$
$([in\text{-}codom\ (\le_{R2\ (\varepsilon_1\ x')\ x'})] \Rrightarrow (\le_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow$ *transitive* $(\le_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow$ *transitive* $(\le_{R2\ x1'\ x2'})$
shows $((\le_L) \dashv (\le_R))$ l r
using *assms*
by (*intro galois-connectionI galois-prop-left-rightI′ mono-wrt-rel-leftI flip.mono-wrt-rel-leftI*)
auto

lemma *galois-connection-left-rightI′*:
assumes $((\le_{L1}) \dashv (\le_{R1}))$ $l1$ $r1$
and *reflexive-on* (*in-codom* $(\le_{L1})$) $(\le_{L1})$
and *reflexive-on* (*in-dom* $(\le_{R1})$) $(\le_{R1})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow$
$((\le_{L2\ (r1\ x1')\ (r1\ x2')}) \Rrightarrow_m (\le_{R2\ (\varepsilon_1\ x1')\ x2'}))\ (l2_{x2'\ (r1\ x1')})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow ((\le_{R2\ (l1\ x1)\ (l1\ x2)}) \Rrightarrow_m (\le_{L2\ x1\ (\eta_1\ x2)}))\ (r2_{x1\ (l1\ x2)})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\le_{L2\ x\ (r1\ x')}) \trianglelefteq (\le_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x2\ x2}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ (\eta_1\ x2)}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ (\varepsilon_1\ x1')\ x2'}) \le (\le_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ x1'\ x1'}) \le (\le_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow$
$([in\text{-}dom\ (\le_{L2\ (r1\ x1')\ (r1\ x2')})] \Rrightarrow (\le_{R2\ x1'\ x2'}))\ (l2_{x1'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x1')})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow$
$([in\text{-}codom\ (\le_{L2\ (r1\ x1')\ (r1\ x2')})] \Rrightarrow (\le_{R2\ x1'\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x2')})$
and $\bigwedge x.\ x \le_{L1} x \Longrightarrow$
$([in\text{-}dom\ (\le_{L2\ x\ (\eta_1\ x)})] \Rrightarrow (\le_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (l2_{(l1\ x)\ (\eta_1\ x)})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow$
$([in\text{-}codom\ (\le_{R2\ (l1\ x1)\ (l1\ x2)})] \Rrightarrow (\le_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x2)})\ (r2_{x2\ (l1\ x2)})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow$
$([in\text{-}dom\ (\le_{R2\ (l1\ x1)\ (l1\ x2)})] \Rrightarrow (\le_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x1)})\ (r2_{x1\ (l1\ x2)})$
and $\bigwedge x'.\ x' \le_{R1} x' \Longrightarrow$
$([in\text{-}codom\ (\le_{R2\ (\varepsilon_1\ x')\ x'})] \Rrightarrow (\le_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow$ *transitive* $(\le_{L2\ x1\ x2})$

and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow \textit{transitive}\ (\le_{R2\ x1'\ x2'})$
shows $((\le_L) \dashv (\le_R))\ l\ r$
using *assms*
by (*intro galois-connection-left-rightI tdfr.mono-wrt-rel-left-if-transitiveI tdfr.mono-wrt-rel-right-if-transitiveI*)
auto

lemma *galois-connection-left-right-if-galois-connectionI*:
assumes $((\le_{L1}) \dashv (\le_{R1}))\ l1\ r1$
and *reflexive-on* (*in-codom* $(\le_{L1})$) $(\le_{L1})$
and *reflexive-on* (*in-dom* $(\le_{R1})$) $(\le_{R1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\le_{L2\ x\ (r1\ x')}) \dashv (\le_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x2\ x2}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ (\eta_1\ x2)}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ (\varepsilon_1\ x1')\ x2'}) \le (\le_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ x1'\ x1'}) \le (\le_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow$
$([\textit{in-dom}\ (\le_{L2\ (r1\ x1')\ (r1\ x2')})] \Rrightarrow (\le_{R2\ x1'\ x2'}))\ (l2_{x1'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x1')})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow$
$([\textit{in-codom}\ (\le_{L2\ (r1\ x1')\ (r1\ x2')})] \Rrightarrow (\le_{R2\ x1'\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x2')})$
and $\bigwedge x.\ x \le_{L1} x \Longrightarrow$
$([\textit{in-dom}\ (\le_{L2\ x\ (\eta_1\ x)})] \Rrightarrow (\le_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (l2_{(l1\ x)\ (\eta_1\ x)})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow$
$([\textit{in-codom}\ (\le_{R2\ (l1\ x1)\ (l1\ x2)})] \Rrightarrow (\le_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x2)})\ (r2_{x2\ (l1\ x2)})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow$
$([\textit{in-dom}\ (\le_{R2\ (l1\ x1)\ (l1\ x2)})] \Rrightarrow (\le_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x1)})\ (r2_{x1\ (l1\ x2)})$
and $\bigwedge x'.\ x' \le_{R1} x' \Longrightarrow$
$([\textit{in-codom}\ (\le_{R2\ (\varepsilon_1\ x')\ x'})] \Rrightarrow (\le_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow \textit{transitive}\ (\le_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow \textit{transitive}\ (\le_{R2\ x1'\ x2'})$
shows $((\le_L) \dashv (\le_R))\ l\ r$
using *assms*
by (*intro galois-connection-left-rightI'*
tdfr.mono-wrt-rel-left2-if-mono-wrt-rel-left2-if-left-GaloisI
tdfr.mono-wrt-rel-right2-if-mono-wrt-rel-right2-if-left-GaloisI)
(*auto 7 0*)

corollary *galois-connection-left-right-if-galois-connectionI'*:
assumes $((\le_{L1}) \dashv (\le_{R1}))\ l1\ r1$
and *reflexive-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$
and *reflexive-on* (*in-field* $(\le_{R1})$) $(\le_{R1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow$
$((\le_{L2\ x\ (r1\ x')}) \dashv (\le_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x2\ x2}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ (\eta_1\ x2)}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ (\varepsilon_1\ x1')\ x2'}) \le (\le_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ x1'\ x1'}) \le (\le_{R2\ x1'\ x2'})$

and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow$
$([in\text{-}dom\ (\le_{L2\ (r1\ x1')\ (r1\ x2')})] \Rrightarrow (\le_{R2\ x1'\ x2'}))\ (l2_{x1'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x1')})$
and $([x' :: in\text{-}codom\ (\le_{R1})] \Rrightarrow_m [x1\ x2 :: (\le_{L1}) \mid x2\ {}_{L1}\lessapprox x'] \Rrightarrow_m$
$[in\text{-}field\ (\le_{L2\ x1\ (r1\ x')})] \Rrightarrow (\le_{R2\ (l1\ x1)\ x'}))\ l2$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow$
$([in\text{-}codom\ (\le_{R2\ (l1\ x1)\ (l1\ x2)})] \Rrightarrow (\le_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x2)})\ (r2_{x2\ (l1\ x2)})$
and $([x :: in\text{-}dom\ (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x\ {}_{L1}\lessapprox x1'] \Rrightarrow_m$
$[in\text{-}field\ (\le_{R2\ (l1\ x)\ x2'})] \Rrightarrow (\le_{L2\ x\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow transitive\ (\le_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow transitive\ (\le_{R2\ x1'\ x2'})$
shows $((\le_L) \dashv (\le_R))\ l\ r$
using *assms* **by** (*intro galois-connection-left-right-if-galois-connectionI*
tdfr.galois-connection-left-right-if-galois-connection-mono-2-assms-leftI
tdfr.galois-connection-left-right-if-galois-connection-mono-2-assms-rightI)
(*auto intro*: *reflexive-on-if-le-pred-if-reflexive-on in-field-if-in-dom in-field-if-in-codom*)

corollary *galois-connection-left-right-if-mono-if-galois-connectionI*:
assumes $((\le_{L1}) \dashv (\le_{R1}))\ l1\ r1$
and $reflexive\text{-}on\ (in\text{-}field\ (\le_{L1}))\ (\le_{L1})$
and $reflexive\text{-}on\ (in\text{-}field\ (\le_{R1}))\ (\le_{R1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\le_{L2\ x\ (r1\ x')}) \dashv (\le_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x2\ x2}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ (\eta_1\ x2)}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ (\varepsilon_1\ x1')\ x2'}) \le (\le_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ x1'\ x1'}) \le (\le_{R2\ x1'\ x2'})$
and $([x1'\ x2' :: (\le_{R1})] \Rrightarrow_m [x1\ x2 :: (\le_{L1}) \mid x2\ {}_{L1}\lessapprox x1'] \Rrightarrow$
$[in\text{-}field\ (\le_{L2\ x1\ (r1\ x2')})] \Rrightarrow (\le_{R2\ (l1\ x1)\ x2'}))\ l2$
and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x2\ {}_{L1}\lessapprox x1'] \Rrightarrow$
$[in\text{-}field\ (\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow transitive\ (\le_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow transitive\ (\le_{R2\ x1'\ x2'})$
shows $((\le_L) \dashv (\le_R))\ l\ r$
using *assms* **by** (*intro galois-connection-left-right-if-galois-connectionI'*
tdfr.galois-connection-left-right-if-galois-connection-mono-assms-leftI
tdfr.galois-connection-left-right-if-galois-connection-mono-assms-rightI)
auto

corollary *galois-connection-left-right-if-mono-if-galois-connectionI'*:
assumes $((\le_{L1}) \dashv (\le_{R1}))\ l1\ r1$
and $reflexive\text{-}on\ (in\text{-}field\ (\le_{L1}))\ (\le_{L1})$
and $reflexive\text{-}on\ (in\text{-}field\ (\le_{R1}))\ (\le_{R1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\le_{L2\ x\ (r1\ x')}) \dashv (\le_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $([\text{-}\ x2 :: (\le_{L1})] \Rrightarrow_m [x3\ x4 :: (\le_{L1}) \mid (x2 \le_{L1} x3 \wedge x4 \le_{L1} \eta_1\ x3)] \Rrightarrow (\ge))$
$L2$
and $([x1'\ x2' :: (\le_{R1}) \mid \varepsilon_1\ x2' \le_{R1} x1'] \Rrightarrow_m [x3'\ \text{-} :: (\le_{R1}) \mid x2' \le_{R1} x3'] \Rrightarrow$
$(\le))\ R2$
and $([x1'\ x2' :: (\le_{R1})] \Rrightarrow_m [x1\ x2 :: (\le_{L1}) \mid x2\ {}_{L1}\lessapprox x1'] \Rrightarrow$

$[in\text{-}field\ (\le_{L2\ x1\ (r1\ x2')})] \Rrightarrow (\le_{R2\ (l1\ x1)\ x2'})) l2$
and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x2\ {}_{L1}\lessapprox x1'] \Rrightarrow$
$[in\text{-}field\ (\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')})) r2$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow transitive\ (\le_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow transitive\ (\le_{R2\ x1'\ x2'})$
shows $((\le_L) \dashv (\le_R))\ l\ r$
using *assms* **by** (*intro galois-connection-left-right-if-mono-if-galois-connectionI*
tdfr.left-rel-right-iff-left-right-rel-if-galois-prop-le-assms-leftI
tdfr.left-rel-right-iff-left-right-rel-if-galois-prop-le-assms-rightI)
auto

end

Monotone Function Relator **context** *transport-Mono-Fun-Rel*
begin

interpretation *flip* : *transport-Mono-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2* **.**

lemma *galois-connection-left-rightI*:
assumes $((\le_{L1}) \dashv (\le_{R1}))\ l1\ r1$
and *reflexive-on* (*in-codom* $(\le_{L1})$) $(\le_{L1})$
and *reflexive-on* (*in-dom* $(\le_{R1})$) $(\le_{R1})$
and $((\le_{L2}) \dashv (\le_{R2}))\ l2\ r2$
and *transitive* $(\le_{L2})$
and *transitive* $(\le_{R2})$
shows $((\le_L) \dashv (\le_R))\ l\ r$
using *assms* **by** (*intro tpdfr.galois-connectionI galois-prop-left-rightI*
mono-wrt-rel-leftI flip.mono-wrt-rel-leftI)
auto

end

end

2.8.5 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* $(\le_L)$) $(\le_L)$
and *R-le-R-if-L*: $\bigwedge x1\ x2.\ x1 \le_L x2 \Longrightarrow (\le_{R\ x2\ x2}) \le (\le_{R\ x1\ x2})$
and *pequiv-R*: $\bigwedge x1\ x2.\ x1 \le_L x2 \Longrightarrow partial\text{-}equivalence\text{-}rel\ (\le_{R\ x1\ x2})$

shows *reflexive-on* (*in-dom DFR*) *DFR*
proof (*intro reflexive-onI Dep-Fun-Rel-relI*)
fix *f x1 x2*
assume *in-dom DFR f*
then obtain *g* **where** *DFR f g* **by** *auto*
moreover assume $x1 \le_L x2$
moreover with *refl-L* **have** $x2 \le_L x2$ **by** *blast*
ultimately have $f\ x1 \le_{R\ x1\ x2} g\ x2$ $f\ x2 \le_{R\ x1\ x2} g\ x2$
using *R-le-R-if-L* **by** *auto*
moreover with *pequiv-R* ‹$x1 \le_L x2$› **have** $g\ x2 \le_{R\ x1\ x2} f\ x2$
by (*blast dest*: *symmetricD*)
ultimately show $f\ x1 \le_{R\ x1\ x2} f\ x2$ **using** *pequiv-R* ‹$x1 \le_L x2$› **by** *blast*
qed

lemma *reflexive-on-in-codomI*:
assumes *refl-L*: *reflexive-on* (*in-dom* $(\le_L)$) $(\le_L)$
and *R-le-R-if-L*: $\bigwedge x1\ x2.\ x1 \le_L x2 \Longrightarrow (\le_{R\ x1\ x1}) \le (\le_{R\ x1\ x2})$
and *pequiv-R*: $\bigwedge x1\ x2.\ x1 \le_L x2 \Longrightarrow$ *partial-equivalence-rel* $(\le_{R\ x1\ x2})$
shows *reflexive-on* (*in-codom DFR*) *DFR*
proof (*intro reflexive-onI Dep-Fun-Rel-relI*)
fix *f x1 x2*
assume *in-codom DFR f*
then obtain *g* **where** *DFR g f* **by** *auto*
moreover assume $x1 \le_L x2$
moreover with *refl-L* **have** $x1 \le_L x1$ **by** *blast*
ultimately have $g\ x1 \le_{R\ x1\ x2} f\ x2$ $g\ x1 \le_{R\ x1\ x2} f\ x1$
using *R-le-R-if-L* **by** *auto*
moreover with *pequiv-R* ‹$x1 \le_L x2$› **have** $f\ x1 \le_{R\ x1\ x2} g\ x1$
by (*blast dest*: *symmetricD*)
ultimately show $f\ x1 \le_{R\ x1\ x2} f\ x2$ **using** *pequiv-R* ‹$x1 \le_L x2$› **by** *blast*
qed

corollary *reflexive-on-in-fieldI*:
assumes *reflexive-on* (*in-field* $(\le_L)$) $(\le_L)$
and $\bigwedge x1\ x2.\ x1 \le_L x2 \Longrightarrow (\le_{R\ x2\ x2}) \le (\le_{R\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \le_L x2 \Longrightarrow (\le_{R\ x1\ x1}) \le (\le_{R\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \le_L x2 \Longrightarrow$ *partial-equivalence-rel* $(\le_{R\ x1\ x2})$
shows *reflexive-on* (*in-field DFR*) *DFR*
proof −
from *assms* **have** *reflexive-on* (*in-dom DFR*) *DFR*
by (*intro reflexive-on-in-domI*)
(*auto intro*: *reflexive-on-if-le-pred-if-reflexive-on in-field-if-in-codom*)
moreover from *assms* **have** *reflexive-on* (*in-codom DFR*) *DFR*
by (*intro reflexive-on-in-codomI*)
(*auto intro*: *reflexive-on-if-le-pred-if-reflexive-on in-field-if-in-dom*)
ultimately show *?thesis* **by** (*auto iff*: *in-field-iff-in-dom-or-in-codom*)
qed

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 (*intro transitiveI Dep-Fun-Rel-relI*)
fix *f1 f2 f3 x1 x2* **assume** $x1 \leq_L x2$
with *refl-L* **have** $x1 \leq_L x1$ **by** *blast*
moreover assume *DFR f1 f2*
ultimately have $f1\ x1 \leq_{R\ x1\ x1} f2\ x1$ **by** *blast*
with *R-le-R-if-L* **have** $f1\ x1 \leq_{R\ x1\ x2} f2\ x1$ **using** ‹$x1 \leq_L x2$› **by** *blast*
assume *DFR f2 f3*
with ‹$x1 \leq_L x2$› **have** $f2\ x1 \leq_{R\ x1\ x2} f3\ x2$ **by** *blast*
with ‹$f1\ x1 \leq_{R\ x1\ x2} f2\ x1$› **show** $f1\ x1 \leq_{R\ x1\ x2} f3\ x2$
using *trans* ‹$x1 \leq_L x2$› **by** *blast*
qed

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 (*intro Binary-Relations-Transitive.transitiveI Dep-Fun-Rel-relI*)
fix *f1 f2 f3 x1 x2* **assume** *DFR f1 f2* $x1 \leq_L x2$
then have $f1\ x1 \leq_{R\ x1\ x2} f2\ x2$ **by** *blast*
from ‹$x1 \leq_L x2$› *refl-L* **have** $x2 \leq_L x2$ **by** *blast*
moreover assume *DFR f2 f3*
ultimately have $f2\ x2 \leq_{R\ x2\ x2} f3\ x2$ **by** *blast*
with *R-le-R-if-L* **have** $f2\ x2 \leq_{R\ x1\ x2} f3\ x2$ **using** ‹$x1 \leq_L x2$› **by** *blast*
with ‹$f1\ x1 \leq_{R\ x1\ x2} f2\ x2$› **show** $f1\ x1 \leq_{R\ x1\ x2} f3\ x2$
using *trans* ‹$x1 \leq_L x2$› **by** *blast*
qed

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*
using *assms* **by** (*intro preorder-onI reflexive-on-in-fieldI*)
(*auto intro*!: *transitiveI dest*: *pequiv-R elim*!: *partial-equivalence-relE*)

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 (*intro symmetricI Dep-Fun-Rel-relI*)
fix *f g x y* **assume** $x \leq_L y$
with *sym-L* **have** $y \leq_L x$ **by** (*rule symmetricD*)

moreover assume *DFR f g*
ultimately have $f\ y \le_{R\ y\ x} g\ x$ **by** *blast*
with *sym-R* ‹$y \le_L x$› **have** $g\ x \le_{R\ y\ x} f\ y$ **by** (*blast dest*: *symmetricD*)
with *R-le-R-if-L* ‹$y \le_L x$› **show** $g\ x \le_{R\ x\ y} f\ y$ **by** *blast*
qed

corollary *partial-equivalence-relI*:
assumes *reflexive-on* (*in-field* $(\le_L)$) $(\le_L)$
and *sym-L*: *symmetric* $(\le_L)$
and *mono-R*: $([x1\ x2 :: (\le_L)] \Rrightarrow_m [x3\ x4 :: (\le_L) \mid x1 \le_L x3] \Rrightarrow (\le))\ R$
and $\bigwedge x1\ x2.\ x1 \le_L x2 \Longrightarrow$ *partial-equivalence-rel* $(\le_{R\ x1\ x2})$
shows *partial-equivalence-rel DFR*
proof −
have $(\le_{R\ x1\ x2}) \le (\le_{R\ x2\ x1})$ **if** $x1 \le_L x2$ **for** *x1 x2*
proof −
from *sym-L* ‹$x1 \le_L x2$› **have** $x2 \le_L x1$ **by** (*rule symmetricD*)
with *mono-R* ‹$x1 \le_L x2$› **show** *?thesis* **by** *blast*
qed
with *assms* **show** *?thesis*
by (*intro partial-equivalence-relI transitiveI symmetricI*)
(*auto elim*: *partial-equivalence-relE*[*OF assms*(*4*)])
qed

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 $((\le_{L1}) \Rrightarrow_m (\le_{R1}))\ l1$
and $((\le_{L1}) \trianglelefteq_h (\le_{R1}))\ l1\ r1$
and *L2-eq*: $(\le_{L2\ x1\ x2}) = (\le_{L2\ x1\ (\eta_1\ x2)})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow$ *transitive* $(\le_{L2\ x\ (r1\ x')})$
and $x1 \le_{L1} x2$
shows *transitive* $(\le_{L2\ x1\ x2})$
by (*subst L2-eq*) (*auto intro*!: *assms t1.left-Galois-left-if-left-relI*)

lemma *symmetric-left2-if-symmetric-left2-if-left-GaloisI*:
assumes $((\le_{L1}) \Rrightarrow_m (\le_{R1}))\ l1$

and $((\le_{L1}) \trianglelefteq_h (\le_{R1}))$ *l1 r1*
and *L2-eq*: $(\le_{L2\ x1\ x2}) = (\le_{L2\ x1\ (\eta_1\ x2)})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow$ *symmetric* $(\le_{L2\ x\ (r1\ x')})$
and $x1 \le_{L1} x2$
shows *symmetric* $(\le_{L2\ x1\ x2})$
by (*subst L2-eq*) (*auto intro*!: *assms t1.left-Galois-left-if-left-relI*)

lemma *partial-equivalence-rel-left2-if-partial-equivalence-rel-left2-if-left-GaloisI*:
assumes $((\le_{L1}) \Rrightarrow_m (\le_{R1}))$ *l1*
and $((\le_{L1}) \trianglelefteq_h (\le_{R1}))$ *l1 r1*
and *L2-eq*: $(\le_{L2\ x1\ x2}) = (\le_{L2\ x1\ (\eta_1\ x2)})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow$ *partial-equivalence-rel* $(\le_{L2\ x\ (r1\ x')})$
and $x1 \le_{L1} x2$
shows *partial-equivalence-rel* $(\le_{L2\ x1\ x2})$
by (*subst L2-eq*) (*auto intro*!: *assms t1.left-Galois-left-if-left-relI*)

context
assumes *galois-prop*: $((\le_{L1}) \trianglelefteq (\le_{R1}))$ *l1 r1*
begin

interpretation *flip-inv* :
transport-Dep-Fun-Rel $(\ge_{R1})$ $(\ge_{L1})$ *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 $\bigwedge R\ S.\ (R^{-1} \Rrightarrow_m S^{-1}) \equiv (R \Rrightarrow_m S)$
and $\bigwedge x\ x'.\ x'\ {}_{R1}\gtrapprox\ x \equiv x\ {}_{L1}\lessapprox\ x'$
and $((\ge_{R1}) \trianglelefteq_h (\ge_{L1}))$ *r1 l1* $\equiv$ *True*
and $\bigwedge R.$ *transitive* $R^{-1} \equiv$ *transitive* R
and $\bigwedge R.$ *symmetric* $R^{-1} \equiv$ *symmetric* R
and $\bigwedge R.$ *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)
using *galois-prop*
by (*auto intro*!: *Eq-TrueI simp add*: *t1.flip-unit-eq-counit*
galois-prop.half-galois-prop-right-rel-inv-iff-half-galois-prop-left
simp del: *rel-inv-iff-rel*)

lemma *transitive-right2-if-transitive-right2-if-left-GaloisI*:
assumes $((\le_{R1}) \Rrightarrow_m (\le_{L1}))$ *r1*
and $(\le_{R2\ x1\ x2}) = (\le_{R2\ (\varepsilon_1\ x1)\ x2})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow$ *transitive* $(\le_{R2\ (l1\ x)\ x'})$
and $x1 \le_{R1} x2$
shows *transitive* $(\le_{R2\ x1\ x2})$
using *galois-prop assms*
by (*intro flip-inv.transitive-left2-if-transitive-left2-if-left-GaloisI*
[*simplified rel-inv-iff-rel*, *of x1*])

auto

lemma *symmetric-right2-if-symmetric-right2-if-left-GaloisI*:
assumes $((\le_{R1}) \Rrightarrow_m (\le_{L1}))$ *r1*
and $(\le_{R2\ x1\ x2}) = (\le_{R2\ (\varepsilon_1\ x1)\ x2})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow$ *symmetric* $(\le_{R2\ (l1\ x)\ x'})$
and $x1 \le_{R1} x2$
shows *symmetric* $(\le_{R2\ x1\ x2})$
using *galois-prop assms*
by (*intro flip-inv.symmetric-left2-if-symmetric-left2-if-left-GaloisI*
[*simplified rel-inv-iff-rel*, *of x1*])
auto

lemma *partial-equivalence-rel-right2-if-partial-equivalence-rel-right2-if-left-GaloisI*:
assumes $((\le_{R1}) \Rrightarrow_m (\le_{L1}))$ *r1*
and $(\le_{R2\ x1\ x2}) = (\le_{R2\ (\varepsilon_1\ x1)\ x2})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow$ *partial-equivalence-rel* $(\le_{R2\ (l1\ x)\ x'})$
and $x1 \le_{R1} x2$
shows *partial-equivalence-rel* $(\le_{R2\ x1\ x2})$
using *galois-prop assms*
by (*intro flip-inv.partial-equivalence-rel-left2-if-partial-equivalence-rel-left2-if-left-GaloisI*
[*simplified rel-inv-iff-rel*, *of x1*])
auto

end

lemma *transitive-left2-if-preorder-equivalenceI*:
assumes *pre-equiv1*: $((\le_{L1}) \equiv_{pre} (\le_{R1}))$ *l1 r1*
and $([x1\ x2 :: (\ge_{L1})] \Rrightarrow_m [x3\ x4 :: (\le_{L1}) \mid x1 \le_{L1} x3] \Rrightarrow (\le))$ *L2*
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\le_{L2\ x\ (r1\ x')}) \equiv_{pre} (\le_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $x1 \le_{L1} x2$
shows *transitive* $(\le_{L2\ x1\ x2})$
proof −
from ‹$x1 \le_{L1} x2$› *pre-equiv1* **have** $x2 \equiv_{L1} \eta_1\ x2$
by (*blast elim*: *t1.preorder-equivalence-order-equivalenceE*
intro: *bi-related-if-rel-equivalence-on*)
with *assms* **have** $(\le_{L2\ x1\ x2}) = (\le_{L2\ x1\ (\eta_1\ x2)})$
by (*intro left2-eq-if-bi-related-if-monoI*) *blast+*
with *assms* **show** *?thesis*
by (*intro transitive-left2-if-transitive-left2-if-left-GaloisI*[*of x1*]) *blast+*
qed

lemma *symmetric-left2-if-partial-equivalence-rel-equivalenceI*:
assumes *PER-equiv1*: $((\le_{L1}) \equiv_{PER} (\le_{R1}))$ *l1 r1*
and $([x1\ x2 :: (\ge_{L1})] \Rrightarrow_m [x3\ x4 :: (\le_{L1}) \mid x1 \le_{L1} x3] \Rrightarrow (\le))$ *L2*
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\le_{L2\ x\ (r1\ x')}) \equiv_{PER} (\le_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $x1 \le_{L1} x2$
shows *symmetric* $(\le_{L2\ x1\ x2})$

proof −
  from ‹$x1 \le_{L1} x2$› *PER-equiv1* **have** $x2 \equiv_{L1} \eta_1\ x2$
    by (*blast elim*: *t1.preorder-equivalence-order-equivalenceE*
      intro: *bi-related-if-rel-equivalence-on*)
  with *assms* **have** $(\le_{L2\ x1\ x2}) = (\le_{L2\ x1\ (\eta_1\ x2)})$
    by (*intro left2-eq-if-bi-related-if-monoI*) *blast+*
  with *assms* **show** *?thesis*
    by (*intro symmetric-left2-if-symmetric-left2-if-left-GaloisI*[*of x1*]) *blast+*
qed

lemma *partial-equivalence-rel-left2-if-partial-equivalence-rel-equivalenceI*:
  assumes *PER-equiv1*: $((\le_{L1}) \equiv_{PER} (\le_{R1}))\ l1\ r1$
  and $([x1\ x2 :: (\ge_{L1})] \Rrightarrow_m [x3\ x4 :: (\le_{L1}) \mid x1 \le_{L1} x3] \Rrightarrow (\le))\ L2$
  and $\bigwedge x\, x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\le_{L2\ x\ (r1\ x')}) \equiv_{PER} (\le_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
  and $x1 \le_{L1} x2$
  shows *partial-equivalence-rel* $(\le_{L2\ x1\ x2})$
proof −
  from ‹$x1 \le_{L1} x2$› *PER-equiv1* **have** $x2 \equiv_{L1} \eta_1\ x2$
    by (*blast elim*: *t1.preorder-equivalence-order-equivalenceE*
      intro: *bi-related-if-rel-equivalence-on*)
  with *assms* **have** $(\le_{L2\ x1\ x2}) = (\le_{L2\ x1\ (\eta_1\ x2)})$
    by (*intro left2-eq-if-bi-related-if-monoI*) *blast+*
  with *assms* **show** *?thesis*
   by (*intro partial-equivalence-rel-left2-if-partial-equivalence-rel-left2-if-left-GaloisI*[*of x1*])
    blast+
qed

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* $\equiv \varepsilon_1$
  by (*simp-all only*: *t1.flip-counit-eq-unit t1.flip-unit-eq-counit*)

lemma *transitive-right2-if-preorder-equivalenceI*:
  assumes *pre-equiv1*: $((\le_{L1}) \equiv_{pre} (\le_{R1}))\ l1\ r1$
  and $([x1'\ x2' :: (\ge_{R1})] \Rrightarrow_m [x3'\ x4' :: (\le_{R1}) \mid x1' \le_{R1} x3'] \Rrightarrow (\le))\ R2$
  and $\bigwedge x\, x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\le_{L2\ x\ (r1\ x')}) \equiv_{pre} (\le_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
  and $x1' \le_{R1} x2'$
  shows *transitive* $(\le_{R2\ x1'\ x2'})$
proof −
  from ‹$x1' \le_{R1} x2'$› *pre-equiv1* **have** $x1' \equiv_{R1} \varepsilon_1\ x1'$
    by (*blast elim*: *t1.preorder-equivalence-order-equivalenceE*
      intro: *bi-related-if-rel-equivalence-on*)
  with *assms* **have** $(\le_{R2\ x1'\ x2'}) = (\le_{R2\ (\varepsilon_1\ x1')\ x2'})$
    by (*intro flip.left2-eq-if-bi-related-if-monoI*) *blast+*
  with *assms* **show** *?thesis*
    by (*intro transitive-right2-if-transitive-right2-if-left-GaloisI*[*of x1'*]) *blast+*
qed

lemma *symmetric-right2-if-partial-equivalence-rel-equivalenceI*:
assumes *PER-equiv1*: $((\leq_{L1}) \equiv_{PER} (\leq_{R1}))$ *l1 r1*
and $([x1'\ x2' :: (\geq_{R1})] \Rrightarrow_m [x3'\ x4' :: (\leq_{R1}) \mid x1' \leq_{R1} x3'] \Rrightarrow (\leq))$ *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 –
from ‹$x1' \leq_{R1} x2'$› *PER-equiv1* **have** $x1' \equiv_{R1} \varepsilon_1\ x1'$
by (*blast elim*: *t1.preorder-equivalence-order-equivalenceE*
intro: *bi-related-if-rel-equivalence-on*)
with *assms* **have** $(\leq_{R2\ x1'\ x2'}) = (\leq_{R2\ (\varepsilon_1\ x1')\ x2'})$
by (*intro flip.left2-eq-if-bi-related-if-monoI*) *blast+*
with *assms* **show** *?thesis*
by (*intro symmetric-right2-if-symmetric-right2-if-left-GaloisI*[*of x1'*]) *blast+*
qed

lemma *partial-equivalence-rel-right2-if-partial-equivalence-rel-equivalenceI*:
assumes *PER-equiv1*: $((\leq_{L1}) \equiv_{PER} (\leq_{R1}))$ *l1 r1*
and $([x1'\ x2' :: (\geq_{R1})] \Rrightarrow_m [x3'\ x4' :: (\leq_{R1}) \mid x1' \leq_{R1} x3'] \Rrightarrow (\leq))$ *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 –
from ‹$x1' \leq_{R1} x2'$› *PER-equiv1* **have** $x1' \equiv_{R1} \varepsilon_1\ x1'$
by (*blast elim*: *t1.preorder-equivalence-order-equivalenceE*
intro: *bi-related-if-rel-equivalence-on*)
with *assms* **have** $(\leq_{R2\ x1'\ x2'}) = (\leq_{R2\ (\varepsilon_1\ x1')\ x2'})$
by (*intro flip.left2-eq-if-bi-related-if-monoI*) *blast+*
with *assms* **show** *?thesis*
by (*intro partial-equivalence-rel-right2-if-partial-equivalence-rel-right2-if-left-GaloisI*[*of x1'*])
blast+
qed

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* $(\leq_{L})$) $(\leq_{L})$
using *assms* **by** (*intro tdfr.reflexive-on-in-field-leftI*) *simp-all*

lemma *transitive-leftI*:
assumes *reflexive-on* (*in-dom* $(\leq_{L1})$) $(\leq_{L1})$
and *transitive* $(\leq_{L2})$

shows *transitive* $(\leq_L)$
using *assms* **by** (*intro tdfr.transitive-leftI*) *simp-all*

lemma *transitive-leftI'*:
assumes *reflexive-on* (*in-codom* $(\leq_{L1})$) $(\leq_{L1})$
and *transitive* $(\leq_{L2})$
shows *transitive* $(\leq_L)$
using *assms* **by** (*intro tdfr.transitive-leftI'*) *simp-all*

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* $(\leq_L)$) $(\leq_L)$
using *assms* **by** (*intro tdfr.preorder-on-in-field-leftI*) *simp-all*

lemma *symmetric-leftI*:
assumes *symmetric* $(\leq_{L1})$
and *symmetric* $(\leq_{L2})$
shows *symmetric* $(\leq_L)$
using *assms* **by** (*intro tdfr.symmetric-leftI*) *simp-all*

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* $(\leq_L)$
using *assms* **by** (*intro tdfr.partial-equivalence-rel-leftI*) *auto*

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* $(\leq_{L})$
using *assms* **by** (*intro tpdfr.symmetric-leftI*) *auto*

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* $(\leq_{L})$
using *assms* **by** (*intro tpdfr.partial-equivalence-rel-leftI*) *auto*

end

end

2.8.6 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 $((\leq_{L1}) \Rrightarrow_m (\leq_{R1}))$ *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 –
from *assms* **have** $x\ {}_{L1}\lessapprox\ l1\ x$ **by** (*intro t1.left-Galois-left-if-left-relI*) *auto*
with *half-galois-prop-left2*
have $((\leq_{R2\ (l1\ x)\ (l1\ x)})\ {}_h\trianglelefteq\ (\leq_{L2\ x\ (\eta_1\ x)}))\ (r2_{x\ (l1\ x)})\ (l2_{(l1\ x)\ x})$ **by** *auto*
with *assms* **show** *?thesis* **by** *simp*
qed

lemma *flip-half-galois-prop-right2-if-half-galois-prop-right2-if-GaloisI*:
assumes $((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))$ *r1*
and *half-galois-prop-right2*: $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow$

$((\le_{R2\ (l1\ x)\ x'}) \trianglelefteq_h (\le_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})\ (l2_{x'\ x})$
and $(\le_{R2\ (\varepsilon_1\ x')\ x'}) = (\le_{R2\ x'\ x'})$
and $(\le_{R2\ x'\ (\varepsilon_1\ x')}) = (\le_{R2\ x'\ x'})$
and $x' \le_{R1} x'$
shows $((\le_{R2\ x'\ (\varepsilon_1\ x')}) \trianglelefteq_h (\le_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ x'})\ (l2_{x'\ (r1\ x')})$
proof −
from *assms* **have** *r1 x'* $_{L1}\lessapprox$ *x'* **by** (*intro t1.right-left-Galois-if-right-relI*) *auto*
with *half-galois-prop-right2*
have $((\le_{R2\ (\varepsilon_1\ x')\ x'}) \trianglelefteq_h (\le_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ x'})\ (l2_{x'\ (r1\ x')})$ **by** *auto*
with *assms* **show** *?thesis* **by** *simp*
qed

interpretation *flip* : *transport-Dep-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2*
rewrites *flip.t1.counit* ≡ η_1 **and** *flip.t1.unit* ≡ ε_1
by (*simp-all only*: *t1.flip-counit-eq-unit t1.flip-unit-eq-counit*)

lemma *galois-equivalence-if-mono-if-galois-equivalence-mono-assms-leftI*:
assumes *galois-equiv1*: $((\le_{L1}) \equiv_G (\le_{R1}))$ *l1 r1*
and *preorder-L1*: *preorder-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$
and *mono-L2*: $([x1\ x2 :: (\ge_{L1})] \Rrightarrow_m [x3\ x4 :: (\le_{L1}) \mid x1 \le_{L1} x3] \Rrightarrow (\le))$ *L2*
shows $([x1\ x2 :: (\le_{L1}) \mid \eta_1\ x2 \le_{L1} x1] \Rrightarrow_m [x3\ x4 :: (\le_{L1}) \mid x2 \le_{L1} x3] \Rrightarrow (\le))$ *L2* (**is** *?goal1*)
and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x3\ x4 :: (\le_{L1}) \mid (x2 \le_{L1} x3 \land x4 \le_{L1} \eta_1\ x3)] \Rrightarrow (\ge))$ *L2* (**is** *?goal2*)
proof −
show *?goal1*
proof (*intro dep-mono-wrt-relI rel-if-if-impI Dep-Fun-Rel-relI*)
fix *x1 x2 x3 x4* **assume** $x1 \le_{L1} x2$
moreover with *galois-equiv1 preorder-L1* **have** $x2 \le_{L1} \eta_1\ x2$
by (*blast intro*: *t1.rel-unit-if-reflexive-on-if-galois-connection*)
moreover assume $\eta_1\ x2 \le_{L1} x1$
ultimately have $x2 \equiv_{L1} x1$ **using** *preorder-L1* **by** *blast*
moreover assume $x3 \le_{L1} x4$ $x2 \le_{L1} x3$
ultimately show $(\le_{L2\ x1\ x3}) \le (\le_{L2\ x2\ x4})$ **using** *preorder-L1 mono-L2* **by** *blast*
qed
show *?goal2*
proof (*intro dep-mono-wrt-relI rel-if-if-impI Dep-Fun-Rel-relI*)
fix *x1 x2 x3 x4* **presume** $x3 \le_{L1} x4$ $x4 \le_{L1} \eta_1\ x3$
moreover with *galois-equiv1 preorder-L1* **have** $\eta_1\ x3 \le_{L1} x3$
by (*blast intro*: *flip.t1.counit-rel-if-reflexive-on-if-galois-connection*)
ultimately have $x3 \equiv_{L1} x4$ **using** *preorder-L1* **by** *blast*
moreover presume $x1 \le_{L1} x2$ $x2 \le_{L1} x3$
ultimately show $(\le_{L2\ x2\ x4}) \le (\le_{L2\ x1\ x3})$ **using** *preorder-L1 mono-L2* **by** *fast*
qed *auto*
qed

lemma *galois-equivalence-if-mono-if-galois-equivalence-Dep-Fun-Rel-pred-assm-leftI*:
  assumes *galois-equiv1*: $((\le_{L1}) \equiv_G (\le_{R1}))$ *l1 r1*
  and *refl-L1*: *reflexive-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$
  and *refl-R1*: *reflexive-on* (*in-field* $(\le_{R1})$) $(\le_{R1})$
  and *mono-L2*: $([x1\ x2 :: (\ge_{L1})] \Rrightarrow_m [x3\ x4 :: (\le_{L1}) \mid x1 \le_{L1} x3] \Rrightarrow (\le))$ *L2*
  and *mono-R2*: $([x1'\ x2' :: (\ge_{R1})] \Rrightarrow_m [x3'\ x4' :: (\le_{R1}) \mid x1' \le_{R1} x3'] \Rrightarrow (\le))$
R2
  and *mono-l2*: $([x1'\ x2' :: (\le_{R1})] \Rrightarrow_m [x1\ x2 :: (\le_{L1}) \mid x2\ {}_{L1}\lessapprox x1'] \Rrightarrow$
    $[\textit{in-field}\ (\le_{L2\ x1\ (r1\ x2')})] \Rrightarrow (\le_{R2\ (l1\ x1)\ x2'}))$ *l2*
  and $x \le_{L1} x$
 shows $([\textit{in-codom}\ (\le_{L2\ (\eta_1\ x)\ x})] \Rrightarrow (\le_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ (\eta_1\ x)})\ (l2_{(l1\ x)\ x})$
proof (*intro Dep-Fun-Rel-predI*)
  fix y **assume** *in-codom* $(\le_{L2\ (\eta_1\ x)\ x})\ y$
  moreover from ‹$x \le_{L1} x$› *galois-equiv1 refl-L1* **have** $x \equiv_{L1} \eta_1\ x$
    by (*blast intro*: *bi-related-if-rel-equivalence-on*
      t1.rel-equivalence-on-unit-if-reflexive-on-if-galois-equivalence)
  moreover with *refl-L1* **have** $\eta_1\ x \le_{L1} \eta_1\ x$ **by** *blast*
  ultimately have *in-codom* $(\le_{L2\ (\eta_1\ x)\ (\eta_1\ x)})\ y$ **using** *mono-L2* **by** *blast*
  moreover from ‹$x \le_{L1} x$› *galois-equiv1*
    have $l1\ x \le_{R1} l1\ x$ $\eta_1\ x \le_{L1} x$ $x\ {}_{L1}\lessapprox l1\ x$
    by (*blast intro*: *t1.left-Galois-left-if-left-relI*
      flip.t1.counit-rel-if-right-rel-if-galois-connection)+
  moreover note
     Dep-Fun-Rel-relD[*OF dep-mono-wrt-relD*[*OF mono-l2* ‹$l1\ x \le_{R1} l1\ x$›] ‹$\eta_1\ x$
$\le_{L1} x$›]
  ultimately have $l2_{(l1\ x)\ (\eta_1\ x)}\ y \le_{R2\ (\varepsilon_1\ (l1\ x))\ (l1\ x)} l2_{(l1\ x)\ x}\ y$ **by** *auto*
  moreover note ‹$l1\ x \le_{R1} l1\ x$›
  moreover with *galois-equiv1 refl-R1* **have** $l1\ x \equiv_{R1} \varepsilon_1\ (l1\ x)$
    by (*blast intro*: *bi-related-if-rel-equivalence-on*
      flip.t1.rel-equivalence-on-unit-if-reflexive-on-if-galois-equivalence)
  ultimately show $l2_{(l1\ x)\ (\eta_1\ x)}\ y \le_{R2\ (l1\ x)\ (l1\ x)} l2_{(l1\ x)\ x}\ y$
    using *mono-R2* **by** *blast*
qed

lemma *galois-equivalence-if-mono-if-galois-equivalence-Dep-Fun-Rel-pred-assm-right*:
  assumes *galois-equiv1*: $((\le_{L1}) \equiv_G (\le_{R1}))$ *l1 r1*
  and *refl-R1*: *reflexive-on* (*in-field* $(\le_{R1})$) $(\le_{R1})$
  and *mono-L2*: $([x1\ x2 :: (\ge_{L1})] \Rrightarrow_m [x3\ x4 :: (\le_{L1}) \mid x1 \le_{L1} x3] \Rrightarrow (\le))$ *L2*
  and *mono-R2*: $([x1'\ x2' :: (\ge_{R1})] \Rrightarrow_m [x3'\ x4' :: (\le_{R1}) \mid x1' \le_{R1} x3'] \Rrightarrow (\le))$
R2
  and *mono-r2*: $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x2\ {}_{L1}\lessapprox x1'] \Rrightarrow$
    $[\textit{in-field}\ (\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')}))$ *r2*
  and $x' \le_{R1} x'$
 shows $([\textit{in-dom}\ (\le_{R2\ x'\ (\varepsilon_1\ x')})] \Rrightarrow (\le_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ x'})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})$
proof (*intro Dep-Fun-Rel-predI*)
  fix y **assume** *in-dom* $(\le_{R2\ x'\ (\varepsilon_1\ x')})\ y$
  moreover from ‹$x' \le_{R1} x'$› *galois-equiv1 refl-R1* **have** $x' \equiv_{R1} \varepsilon_1\ x'$

by (*blast intro*: *bi-related-if-rel-equivalence-on* *flip.t1.rel-equivalence-on-unit-if-reflexive-on-if-galois-equivalence*)
moreover with *refl-R1* **have** $\varepsilon_1\ x' \le_{R1} \varepsilon_1\ x'$ **by** *blast*
ultimately have *in-dom* $(\le_{R2\ (\varepsilon_1\ x')\ (\varepsilon_1\ x')})$ y **using** *mono-R2* **by** *blast*
moreover from ‹$x' \le_{R1} x'$› *galois-equiv1*
have $r1\ x' \le_{L1} r1\ x'$ $x' \le_{R1} \varepsilon_1\ x'$ $r1\ x'\ {}_{L1}\lessapprox x'$
by (*blast intro*: *t1.right-left-Galois-if-right-relI* *flip.t1.rel-unit-if-left-rel-if-galois-connection*)+
moreover note
Dep-Fun-Rel-relD[*OF dep-mono-wrt-relD*[*OF mono-r2* ‹$r1\ x' \le_{L1} r1\ x'$›] ‹$x' \le_{R1} \varepsilon_1\ x'$›]
ultimately have $r2_{(r1\ x')\ x'}\ y \le_{L2\ (r1\ x')\ (\eta_1\ (r1\ x'))}\ r2_{(r1\ x')\ (\varepsilon_1\ x')}\ y$ **by** *auto*
moreover note ‹$r1\ x' \le_{L1} r1\ x'$›
moreover with *galois-equiv1 refl-R1* **have** $r1\ x' \equiv_{L1} \eta_1\ (r1\ x')$
by (*blast intro*: *bi-related-if-rel-equivalence-on* *t1.rel-equivalence-on-unit-if-reflexive-on-if-galois-equivalence*)
ultimately show $r2_{(r1\ x')\ x'}\ y \le_{L2\ (r1\ x')\ (r1\ x')}\ r2_{(r1\ x')\ (\varepsilon_1\ x')}\ y$
using *mono-L2* **by** *blast*
qed

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* $\equiv \varepsilon_1$
by (*simp-all only*: *t1.flip-counit-eq-unit t1.flip-unit-eq-counit*)

lemma *galois-equivalence-if-galois-equivalenceI*:
assumes *galois-equiv1*: $((\le_{L1}) \equiv_G (\le_{R1}))\ l1\ r1$
and *refl-L1*: *reflexive-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$
and *reflexive-on* (*in-field* $(\le_{R1})$) $(\le_{R1})$
and *galois-equiv2*: $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow$
$((\le_{L2\ x\ (r1\ x')}) \equiv_G (\le_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x2\ x2}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x.\ x \le_{L1} x \Longrightarrow (\le_{L2\ (\eta_1\ x)\ x}) \le (\le_{L2\ x\ x})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ x1}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ (\eta_1\ x2)}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ x2'\ x2'}) \le (\le_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ (\varepsilon_1\ x1')\ x2'}) \le (\le_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ x1'\ x1'}) \le (\le_{R2\ x1'\ x2'})$
and $\bigwedge x'.\ x' \le_{R1} x' \Longrightarrow (\le_{R2\ x'\ (\varepsilon_1\ x')}) \le (\le_{R2\ x'\ x'})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow$
([*in-dom* $(\le_{L2\ (r1\ x1')\ (r1\ x2')})$] $\Rrightarrow (\le_{R2\ x1'\ x2'}))\ (l2_{x1'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x1')})$

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 $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$
$([in\text{-}dom\ (\leq_{L2\ x\ (\eta_1\ x)})] \Rrightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (l2_{(l1\ x)\ (\eta_1\ x)})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$
$([in\text{-}codom\ (\leq_{L2\ (\eta_1\ x)\ x})] \Rrightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ (\eta_1\ x)})\ (l2_{(l1\ x)\ x})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$
$([in\text{-}codom\ (\leq_{R2\ (l1\ x1)\ (l1\ x2)})] \Rrightarrow (\leq_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x2)})\ (r2_{x2\ (l1\ x2)})$
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 $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$
$([in\text{-}codom\ (\leq_{R2\ (\varepsilon_1\ x')\ x'})] \Rrightarrow (\leq_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$
$([in\text{-}dom\ (\leq_{R2\ x'\ (\varepsilon_1\ x')})] \Rrightarrow (\leq_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ x'})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow transitive\ (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow transitive\ (\leq_{R2\ x1'\ x2'})$
shows $((\leq_L) \equiv_G (\leq_R))\ l\ r$
proof −
from *galois-equiv2* **have**
$((\leq_{L2\ x\ (r1\ x')}) \dashv (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
$((\leq_{R2\ (l1\ x)\ x'})\ {}_h\trianglelefteq\ (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})\ (l2_{x'\ x})$
$((\leq_{R2\ (l1\ x)\ x'}) \trianglelefteq_h (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})\ (l2_{x'\ x})$
if $x\ {}_{L1}\lessapprox\ x'$ **for** $x\ x'$ **using** $\langle x\ {}_{L1}\lessapprox\ x'\rangle$
by (*blast elim*: *galois.galois-connectionE galois-prop.galois-propE*)+
moreover from *galois-equiv1 galois-equiv2* **have**
$\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$((\leq_{L2\ (r1\ x1')\ (r1\ x2')}) \Rrightarrow_m (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}))\ (l2_{x2'\ (r1\ x1')})$
by (*intro tdfr.mono-wrt-rel-left2-if-mono-wrt-rel-left2-if-left-GaloisI*) *auto*
moreover from *galois-equiv1 galois-equiv2* **have**
$\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow ((\leq_{R2\ (l1\ x1)\ (l1\ x2)}) \Rrightarrow_m (\leq_{L2\ x1\ (\eta_1\ x2)}))\ (r2_{x1\ (l1\ x2)})$
by (*intro tdfr.mono-wrt-rel-right2-if-mono-wrt-rel-right2-if-left-GaloisI*)
(*auto elim*!: *t1.galois-equivalenceE*)
moreover from *galois-equiv1 refl-L1* **have**
$\bigwedge x.\ x \leq_{L1} x \Longrightarrow x \equiv_{L1} \eta_1\ x$
$\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow x' \equiv_{R1} \varepsilon_1\ x'$
by (*blast intro*!: *bi-related-if-rel-equivalence-on*
t1.rel-equivalence-on-unit-if-reflexive-on-if-galois-equivalence
flip.t1.rel-equivalence-on-unit-if-reflexive-on-if-galois-equivalence)+
ultimately show *?thesis* **using** *assms*
by (*intro galois-equivalenceI*
galois-connection-left-right-if-galois-connectionI flip.galois-prop-left-rightI
tdfr.flip-half-galois-prop-left2-if-half-galois-prop-left2-if-left-GaloisI
tdfr.flip-half-galois-prop-right2-if-half-galois-prop-right2-if-GaloisI
tdfr.mono-wrt-rel-left-if-transitiveI tdfr.mono-wrt-rel-right-if-transitiveI
flip.tdfr.left-rel-right-if-left-right-rel-le-right2-assmI
flip.tdfr.left-right-rel-if-left-rel-right-ge-left2-assmI
tdfr.left-rel2-unit-eqs-left-rel2I

flip.tdfr.left-rel2-unit-eqs-left-rel2I)
(*auto elim*!: *t1.galois-equivalenceE*
intro: *reflexive-on-if-le-pred-if-reflexive-on in-field-if-in-dom*
in-field-if-in-codom)
qed

corollary *galois-equivalence-if-galois-equivalenceI'*:
assumes $((\le_{L1}) \equiv_G (\le_{R1}))\ l1\ r1$
and *reflexive-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$
and *reflexive-on* (*in-field* $(\le_{R1})$) $(\le_{R1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\le_{L2\ x\ (r1\ x')}) \equiv_G (\le_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x2\ x2}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x.\ x \le_{L1} x \Longrightarrow (\le_{L2\ (\eta_1\ x)\ x}) \le (\le_{L2\ x\ x})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ x1}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ (\eta_1\ x2)}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ x2'\ x2'}) \le (\le_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ (\varepsilon_1\ x1')\ x2'}) \le (\le_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ x1'\ x1'}) \le (\le_{R2\ x1'\ x2'})$
and $\bigwedge x'.\ x' \le_{R1} x' \Longrightarrow (\le_{R2\ x'\ (\varepsilon_1\ x')}) \le (\le_{R2\ x'\ x'})$
and $([x1'\ x2' :: (\le_{R1})] \Rrightarrow_m [x1\ x2 :: (\le_{L1})\ |\ x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[\textit{in-field}\ (\le_{L2\ x1\ (r1\ x2')})] \Rrightarrow (\le_{R2\ (l1\ x1)\ x2'}))\ l2$
and $\bigwedge x.\ x \le_{L1} x \Longrightarrow$
$([\textit{in-codom}\ (\le_{L2\ (\eta_1\ x)\ x})] \Rrightarrow (\le_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ (\eta_1\ x)})\ (l2_{(l1\ x)\ x})$
and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1})\ |\ x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[\textit{in-field}\ (\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x'.\ x' \le_{R1} x' \Longrightarrow$
$([\textit{in-dom}\ (\le_{R2\ x'\ (\varepsilon_1\ x')})] \Rrightarrow (\le_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ x'})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow \textit{transitive}\ (\le_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow \textit{transitive}\ (\le_{R2\ x1'\ x2'})$
shows $((\le_L) \equiv_G (\le_R))\ l\ r$
using *assms* **by** (*intro galois-equivalence-if-galois-equivalenceI*
tdfr.galois-connection-left-right-if-galois-connection-mono-assms-leftI
tdfr.galois-connection-left-right-if-galois-connection-mono-assms-rightI
tdfr.galois-connection-left-right-if-galois-connection-mono-2-assms-leftI
tdfr.galois-connection-left-right-if-galois-connection-mono-2-assms-rightI)
(*auto intro*: *reflexive-on-if-le-pred-if-reflexive-on in-field-if-in-dom*
in-field-if-in-codom)

corollary *galois-equivalence-if-mono-if-galois-equivalenceI*:
assumes $((\le_{L1}) \equiv_G (\le_{R1}))\ l1\ r1$
and *reflexive-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$
and *reflexive-on* (*in-field* $(\le_{R1})$) $(\le_{R1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\le_{L2\ x\ (r1\ x')}) \equiv_G (\le_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $([x1\ x2 :: (\le_{L1})\ |\ \eta_1\ x2 \le_{L1} x1] \Rrightarrow_m [x3\ x4 :: (\le_{L1})\ |\ x2 \le_{L1} x3] \Rrightarrow (\le))$
$L2$
and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x3\ x4 :: (\le_{L1})\ |\ (x2 \le_{L1} x3 \wedge x4 \le_{L1} \eta_1\ x3)] \Rrightarrow$
$(\ge))\ L2$

and $([x1'\ x2' :: (\le_{R1}) \mid \varepsilon_1\ x2' \le_{R1} x1'] \Rrightarrow_m [x3'\ x4' :: (\le_{R1}) \mid x2' \le_{R1} x3'] \Rrightarrow (\le))\ R2$
and $([x1'\ x2' :: (\le_{R1})] \Rrightarrow_m [x3'\ x4' :: (\le_{R1}) \mid (x2' \le_{R1} x3' \wedge x4' \le_{R1} \varepsilon_1\ x3')] \Rrightarrow (\ge))\ R2$
and $([x1'\ x2' :: (\le_{R1})] \Rrightarrow_m [x1\ x2 :: (\le_{L1}) \mid x2\ {}_{L1}\lessapprox x1'] \Rrightarrow [\textit{in-field}\ (\le_{L2\ x1\ (r1\ x2')})] \Rrightarrow (\le_{R2\ (l1\ x1)\ x2'}))\ l2$
and $\bigwedge x.\ x \le_{L1} x \Longrightarrow ([\textit{in-codom}\ (\le_{L2\ (\eta_1\ x)\ x})] \Rrightarrow (\le_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ (\eta_1\ x)})\ (l2_{(l1\ x)\ x})$
and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x2\ {}_{L1}\lessapprox x1'] \Rrightarrow [\textit{in-field}\ (\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x'.\ x' \le_{R1} x' \Longrightarrow ([\textit{in-dom}\ (\le_{R2\ x'\ (\varepsilon_1\ x')})] \Rrightarrow (\le_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ x'})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow \textit{transitive}\ (\le_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow \textit{transitive}\ (\le_{R2\ x1'\ x2'})$
shows $((\le_L) \equiv_G (\le_R))\ l\ r$
using *assms* **by** (*intro galois-equivalence-if-galois-equivalenceI'*
tdfr.left-rel-right-iff-left-right-rel-if-galois-prop-le-assms-leftI
flip.tdfr.left-rel-right-iff-left-right-rel-if-galois-prop-le-assms-leftI
tdfr.left-rel-right-iff-left-right-rel-if-galois-prop-le-assms-rightI
flip.tdfr.left-rel-right-iff-left-right-rel-if-galois-prop-le-assms-rightI)
auto

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* $\equiv \varepsilon_1$
by (*simp-all only*: *t1.flip-counit-eq-unit t1.flip-unit-eq-counit*)

lemma *galois-equivalence-if-mono-if-preorder-equivalenceI*:
assumes $((\le_{L1}) \equiv_{pre} (\le_{R1}))\ l1\ r1$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\le_{L2\ x\ (r1\ x')}) \equiv_G (\le_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $([x1\ x2 :: (\ge_{L1})] \Rrightarrow_m [x3\ x4 :: (\le_{L1}) \mid x1 \le_{L1} x3] \Rrightarrow (\le))\ L2$
and $([x1'\ x2' :: (\ge_{R1})] \Rrightarrow_m [x3'\ x4' :: (\le_{R1}) \mid x1' \le_{R1} x3'] \Rrightarrow (\le))\ R2$
and $([x1'\ x2' :: (\le_{R1})] \Rrightarrow_m [x1\ x2 :: (\le_{L1}) \mid x2\ {}_{L1}\lessapprox x1'] \Rrightarrow [\textit{in-field}\ (\le_{L2\ x1\ (r1\ x2')})] \Rrightarrow (\le_{R2\ (l1\ x1)\ x2'}))\ l2$
and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x2\ {}_{L1}\lessapprox x1'] \Rrightarrow [\textit{in-field}\ (\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow \textit{transitive}\ (\le_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow \textit{transitive}\ (\le_{R2\ x1'\ x2'})$
shows $((\le_L) \equiv_G (\le_R))\ l\ r$
using *assms* **by** (*intro galois-equivalence-if-mono-if-galois-equivalenceI*
tdfr.galois-equivalence-if-mono-if-galois-equivalence-mono-assms-leftI
flip.tdfr.galois-equivalence-if-mono-if-galois-equivalence-mono-assms-leftI
tdfr.galois-equivalence-if-mono-if-galois-equivalence-Dep-Fun-Rel-pred-assm-leftI
tdfr.galois-equivalence-if-mono-if-galois-equivalence-Dep-Fun-Rel-pred-assm-right)
auto

theorem *galois-equivalence-if-mono-if-preorder-equivalenceI'*:

assumes $((\le_{L1}) \equiv_{pre} (\le_{R1}))$ *l1 r1*
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\le_{L2\ x\ (r1\ x')}) \equiv_{pre} (\le_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $([x1\ x2 :: (\ge_{L1})] \Rrightarrow_m [x3\ x4 :: (\le_{L1}) \mid x1 \le_{L1} x3] \Rrightarrow (\le))\ L2$
and $([x1'\ x2' :: (\ge_{R1})] \Rrightarrow_m [x3'\ x4' :: (\le_{R1}) \mid x1' \le_{R1} x3'] \Rrightarrow (\le))\ R2$
and $([x1'\ x2' :: (\le_{R1})] \Rrightarrow_m [x1\ x2 :: (\le_{L1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[\textit{in-field}\ (\le_{L2\ x1\ (r1\ x2')})] \Rrightarrow (\le_{R2\ (l1\ x1)\ x2'}))\ l2$
and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[\textit{in-field}\ (\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')}))\ r2$
shows $((\le_L) \equiv_G (\le_R))\ l\ r$
using *assms* **by** (*intro galois-equivalence-if-mono-if-preorder-equivalenceI*
tdfr.transitive-left2-if-preorder-equivalenceI
tdfr.transitive-right2-if-preorder-equivalenceI)
auto

end

Monotone Function Relator **context** *transport-Mono-Fun-Rel*
begin

interpretation *flip* : *transport-Mono-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2* **.**

lemma *galois-equivalenceI*:
assumes $((\le_{L1}) \equiv_G (\le_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$
and *reflexive-on* (*in-field* $(\le_{R1})$) $(\le_{R1})$
and $((\le_{L2}) \equiv_G (\le_{R2}))$ *l2 r2*
and *transitive* $(\le_{L2})$
and *transitive* $(\le_{R2})$
shows $((\le_L) \equiv_G (\le_R))\ l\ r$
using *assms* **by** (*intro tpdfr.galois-equivalenceI*
galois-connection-left-rightI flip.galois-prop-left-rightI)
(*auto intro*: *reflexive-on-if-le-pred-if-reflexive-on*
in-field-if-in-dom in-field-if-in-codom)

end

end

2.8.7 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 results from *transport-Mono-Dep-Fun-Rel* to *transport-Dep-Fun-Rel* whenever $(\le_L)$ and $(\le_R)$ are reflexive.

lemma *reflexive-on-in-field-left-rel2-le-assmI*:
  assumes *refl-L1*: *reflexive-on* (*in-dom* $(\le_{L1})$) $(\le_{L1})$
  and *mono-L2*: $([x1 :: \top] \Rrightarrow_m [x2\ x3 :: (\le_{L1}) \mid x1 \le_{L1} x2] \Rrightarrow_m (\le))$ *L2*
  and $x1 \le_{L1} x2$
  shows $(\le_{L2\ x1\ x1}) \le (\le_{L2\ x1\ x2})$
proof −
  from *refl-L1* ‹$x1 \le_{L1} x2$› **have** $x1 \le_{L1} x1$ **by** *blast*
  with *dep-mono-wrt-relD*[*OF dep-mono-wrt-predD*[*OF mono-L2*] ‹$x1 \le_{L1} x2$›]
    show $(\le_{L2\ x1\ x1}) \le (\le_{L2\ x1\ x2})$ **by** *auto*
qed

lemma *reflexive-on-in-field-mono-assm-left2I*:
  assumes *mono-L2*: $([x1\ x2 :: (\ge_{L1})] \Rrightarrow_m [x3\ x4 :: (\le_{L1}) \mid x1 \le_{L1} x3] \Rrightarrow (\le))$ *L2*
  and *refl-L1*: *reflexive-on* (*in-dom* $(\le_{L1})$) $(\le_{L1})$
  shows $([x1 :: \top] \Rrightarrow_m [x2\ x3 :: (\le_{L1}) \mid x1 \le_{L1} x2] \Rrightarrow_m (\le))$ *L2*
proof (*intro dep-mono-wrt-predI dep-mono-wrt-relI rel-if-if-impI*)
  fix *x1 x2 x3* **assume** $x1 \le_{L1} x2$ $x2 \le_{L1} x3$
  with *refl-L1* **have** $x1 \ge_{L1} x1$ **by** *blast*
  from *Dep-Fun-Rel-relD*[*OF dep-mono-wrt-relD*[*OF mono-L2* ‹$x1 \ge_{L1} x1$›]
    ‹$x2 \le_{L1} x3$›] ‹$x1 \le_{L1} x2$›
    show $(\le_{L2\ x1\ x2}) \le (\le_{L2\ x1\ x3})$ **by** *blast*
qed

lemma *reflexive-on-in-field-left-if-equivalencesI*:
  assumes $((\le_{L1}) \equiv_G (\le_{R1}))$ *l1 r1*
  and *preorder-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$
  and $([x1\ x2 :: (\ge_{L1})] \Rrightarrow_m [x3\ x4 :: (\le_{L1}) \mid x1 \le_{L1} x3] \Rrightarrow (\le))$ *L2*
  and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow$ *partial-equivalence-rel* $(\le_{L2\ x1\ x2})$
  shows *reflexive-on* (*in-field* $(\le_L)$) $(\le_L)$
  using *assms*
  by (*intro reflexive-on-in-field-leftI*
    left-rel-right-iff-left-right-rel-if-galois-prop-le-assms-leftI
    galois-equivalence-if-mono-if-galois-equivalence-mono-assms-leftI
    reflexive-on-in-field-left-rel2-le-assmI
    reflexive-on-in-field-mono-assm-left2I)
  (*auto intro*: *reflexive-on-if-le-pred-if-reflexive-on in-field-if-in-dom*)

end

Monotone Dependent Function Relator **context** *transport-Mono-Dep-Fun-Rel*
begin

lemma *left-rel-eq-tdfr-leftI*:
assumes *reflexive-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x2\ x2}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ x1}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow$ *partial-equivalence-rel* $(\le_{L2\ x1\ x2})$
shows $(\le_L) =$ *tdfr.L*
using *assms* **by** (*intro left-rel-eq-tdfr-left-rel-if-reflexive-on tdfr.reflexive-on-in-field-leftI*)
auto

lemma *left-rel-eq-tdfr-leftI-if-equivalencesI*:
assumes $((\le_{L1}) \equiv_G (\le_{R1}))$ *l1 r1*
and *preorder-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$
and $([x1\ x2 :: (\ge_{L1})] \Rrightarrow_m [x3\ x4 :: (\le_{L1}) \mid x1 \le_{L1} x3] \Rrightarrow (\le))$ *L2*
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow$ *partial-equivalence-rel* $(\le_{L2\ x1\ x2})$
shows $(\le_L) =$ *tdfr.L*
using *assms* **by** (*intro left-rel-eq-tdfr-left-rel-if-reflexive-on tdfr.reflexive-on-in-field-left-if-equivalencesI*)
auto

end

Monotone Function Relator **context** *transport-Mono-Fun-Rel*
begin

lemma *left-rel-eq-tfr-leftI*:
assumes *reflexive-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$
and *partial-equivalence-rel* $(\le_{L2})$
shows $(\le_L) =$ *tfr.tdfr.L*
using *assms* **by** (*intro tpdfr.left-rel-eq-tdfr-leftI*) *auto*

end

end

2.8.8 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* $\equiv \eta_1$ **by** (*simp only*: *t1.flip-counit-eq-unit*)

lemma *Dep-Fun-Rel-left-Galois-if-left-GaloisI*:
assumes $((\le_{L1})\ {}_{h}\trianglelefteq\ (\le_{R1}))\ l1\ r1$
and *refl-L1*: *reflexive-on* $(\textit{in-dom}\ (\le_{L1}))\ (\le_{L1})$
and *mono-r2*: $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\le_{R2\ (l1\ x)\ x'}) \Rrightarrow_m (\le_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and *L2-le2*: $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ x1}) \le (\le_{L2\ x1\ x2})$
and *ge-L2-r2-le2*: $\bigwedge x\ x'\ y'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow \textit{in-dom}\ (\le_{R2\ (l1\ x)\ x'})\ y' \Longrightarrow$
$(\ge_{L2\ x\ (r1\ x')})\ (r2_{x\ (l1\ x)}\ y') \le (\ge_{L2\ x\ (r1\ x')})\ (r2_{x\ x'}\ y')$
and *trans-L2*: $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow \textit{transitive}\ (\le_{L2\ x1\ x2})$
and $g \le_R g$
and $f\ {}_{L}\lessapprox\ g$
shows $([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
proof (*intro Dep-Fun-Rel-relI*)
fix $x\ x'$ **assume** $x\ {}_{L1}\lessapprox\ x'$
show $f\ x\ {}_{L2\ x\ x'}\lessapprox\ g\ x'$
proof (*intro t2.left-GaloisI*)
from ‹$x\ {}_{L1}\lessapprox\ x'$› ‹$((\le_{L1})\ {}_{h}\trianglelefteq\ (\le_{R1}))\ l1\ r1$› **have** $x \le_{L1} r1\ x'$ $l1\ x \le_{R1} x'$ **by** *auto*
with ‹$g \le_R g$› **have** $g\ (l1\ x) \le_{R2\ (l1\ x)\ x'} g\ x'$ **by** *blast*
then show *in-codom* $(\le_{R2\ (l1\ x)\ x'})\ (g\ x')$ **by** *blast*

from ‹$f\ {}_{L}\lessapprox\ g$› **have** $f \le_L r\ g$ **by** *blast*
moreover from *refl-L1* ‹$x\ {}_{L1}\lessapprox\ x'$› **have** $x \le_{L1} x$ **by** *blast*
ultimately have $f\ x \le_{L2\ x\ x} (r\ g)\ x$ **by** *blast*
with *L2-le2* ‹$x \le_{L1} r1\ x'$› **have** $f\ x \le_{L2\ x\ (r1\ x')} (r\ g)\ x$ **by** *blast*
then have $f\ x \le_{L2\ x\ (r1\ x')} r2_{x\ (l1\ x)}\ (g\ (l1\ x))$ **by** *simp*
with *ge-L2-r2-le2* **have** $f\ x \le_{L2\ x\ (r1\ x')} r2_{x\ x'}\ (g\ (l1\ x))$
using ‹$x\ {}_{L1}\lessapprox\ x'$› ‹$g\ (l1\ x) \le_{R2\ (l1\ x)\ x'}$ -› **by** *blast*
moreover have ... $\le_{L2\ x\ (r1\ x')} r2_{x\ x'}\ (g\ x')$
using *mono-r2* ‹$x\ {}_{L1}\lessapprox\ x'$› ‹$g\ (l1\ x) \le_{R2\ (l1\ x)\ x'} g\ x'$› **by** *blast*
ultimately show $f\ x \le_{L2\ x\ (r1\ x')} r2_{x\ x'}\ (g\ x')$
using *trans-L2* ‹$x\ {}_{L1}\lessapprox\ x'$› **by** *blast*
qed
qed

lemma *left-rel-right-if-Dep-Fun-Rel-left-GaloisI*:
assumes *mono-l1*: $((\le_{L1}) \Rrightarrow_m (\le_{R1}))\ l1$
and *half-galois-prop-right1*: $((\le_{L1}) \trianglelefteq_h (\le_{R1}))\ l1\ r1$
and *L2-unit-le2*: $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ (\eta_1\ x2)}) \le (\le_{L2\ x1\ x2})$
and *ge-L2-r2-le1*: $\bigwedge x1\ x2\ y'.\ x1 \le_{L1} x2 \Longrightarrow \textit{in-codom}\ (\le_{R2\ (l1\ x1)\ (l1\ x2)})\ y'$
$\Longrightarrow$
$(\ge_{L2\ x1\ x2})\ (r2_{x1\ (l1\ x2)}\ y') \le (\ge_{L2\ x1\ x2})\ (r2_{x2\ (l1\ x2)}\ y')$
and *rel-f-g*: $([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
shows $f \le_L r\ g$
proof (*intro left-relI*)
fix $x1\ x2$ **assume** $x1 \le_{L1} x2$
with *mono-l1 half-galois-prop-right1* **have** $x1\ {}_{L1}\lessapprox\ l1\ x2$
by (*intro t1.left-Galois-left-if-left-relI*) *auto*

with *rel-f-g* **have** $f\ x1\ {}_{L2\ x1\ (l1\ x2)}\lessapprox g\ (l1\ x2)$ **by** *blast*
then have *in-codom* $(\le_{R2\ (l1\ x1)\ (l1\ x2)})\ (g\ (l1\ x2))$
$f\ x1 \le_{L2\ x1\ (\eta_1\ x2)} r2_{x1\ (l1\ x2)}\ (g\ (l1\ x2))$ **by** *auto*
with *L2-unit-le2* ‹$x1 \le_{L1} x2$› **have** $f\ x1 \le_{L2\ x1\ x2} r2_{x1\ (l1\ x2)}\ (g\ (l1\ x2))$ **by** *blast*
with *ge-L2-r2-le1* ‹$x1 \le_{L1} x2$› ‹*in-codom* $(\le_{R2\ (l1\ x1)\ (l1\ x2)})\ (g\ (l1\ x2))$›
have $f\ x1 \le_{L2\ x1\ x2} r2_{x2\ (l1\ x2)}\ (g\ (l1\ x2))$ **by** *blast*
then show $f\ x1 \le_{L2\ x1\ x2} r\ g\ x2$ **by** *simp*
qed

lemma *left-Galois-if-Dep-Fun-Rel-left-GaloisI*:
assumes $((\le_{L1}) \Rrightarrow_m (\le_{R1}))\ l1$
and $((\le_{L1}) \trianglelefteq_h (\le_{R1}))\ l1\ r1$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ (\eta_1\ x2)}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x1\ x2\ y'.\ x1 \le_{L1} x2 \Longrightarrow$ *in-codom* $(\le_{R2\ (l1\ x1)\ (l1\ x2)})\ y' \Longrightarrow$
$(\ge_{L2\ x1\ x2})\ (r2_{x1\ (l1\ x2)}\ y') \le (\ge_{L2\ x1\ x2})\ (r2_{x2\ (l1\ x2)}\ y')$
and *in-codom* $(\le_R)\ g$
and $([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
shows $f\ {}_L\lessapprox g$
using *assms* **by** (*intro left-GaloisI left-rel-right-if-Dep-Fun-Rel-left-GaloisI*) *auto*

lemma *left-right-rel-if-Dep-Fun-Rel-left-GaloisI*:
assumes *mono-r1*: $((\le_{R1}) \Rrightarrow_m (\le_{L1}))\ r1$
and *half-galois-prop-left2*: $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow$
$((\le_{L2\ (r1\ x1')\ (r1\ x2')})\ {}_h\trianglelefteq (\le_{R2\ (\varepsilon_1\ x1')\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (r2_{(r1\ x1')\ x2'})$
and *R2-le1*: $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ (\varepsilon_1\ x1')\ x2'}) \le (\le_{R2\ x1'\ x2'})$
and *R2-l2-le1*: $\bigwedge x1'\ x2'\ y.\ x1' \le_{R1} x2' \Longrightarrow$ *in-dom* $(\le_{L2\ (r1\ x1')\ (r1\ x2')})\ y \Longrightarrow$
$(\le_{R2\ x1'\ x2'})\ (l2_{x2'\ (r1\ x1')}\ y) \le (\le_{R2\ x1'\ x2'})\ (l2_{x1'\ (r1\ x1')}\ y)$
and *rel-f-g*: $([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
shows $l\ f \le_R g$
proof (*rule flip.left-relI*)
fix $x1'\ x2'$ **assume** $x1' \le_{R1} x2'$
with *mono-r1* **have** $r1\ x1'\ {}_{L1}\lessapprox x2'$ **by** *blast*
with *rel-f-g* **have** $f\ (r1\ x1')\ {}_{L2\ (r1\ x1')\ x2'}\lessapprox g\ x2'$ **by** *blast*
with *half-galois-prop-left2*[*OF* ‹$x1' \le_{R1} x2'$›]
have $l2_{x2'\ (r1\ x1')}\ (f\ (r1\ x1')) \le_{R2\ (\varepsilon_1\ x1')\ x2'} g\ x2'$ **by** *auto*
with *R2-le1* ‹$x1' \le_{R1} x2'$› **have** $l2_{x2'\ (r1\ x1')}\ (f\ (r1\ x1')) \le_{R2\ x1'\ x2'} g\ x2'$
by *blast*
with *R2-l2-le1* ‹$x1' \le_{R1} x2'$› ‹$f\ (r1\ x1')\ {}_{L2\ (r1\ x1')\ x2'}\lessapprox$ -›
have $l2_{x1'\ (r1\ x1')}\ (f\ (r1\ x1')) \le_{R2\ x1'\ x2'} g\ x2'$ **by** *blast*
then show $l\ f\ x1' \le_{R2\ x1'\ x2'} g\ x2'$ **by** *simp*
qed

lemma *left-Galois-if-Dep-Fun-Rel-left-GaloisI'*:
assumes $((\le_{L1}) \Rrightarrow_m (\le_{R1}))\ l1$ **and** $((\le_{R1}) \Rrightarrow_m (\le_{L1}))\ r1$

and $((\le_{L1}) \trianglelefteq_h (\le_{R1}))$ *l1 r1*
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow$
$((\le_{L2\ (r1\ x1')\ (r1\ x2')})\ {}_h\trianglelefteq\ (\le_{R2\ (\varepsilon_1\ x1')\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (r2_{(r1\ x1')\ x2'})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ (\eta_1\ x2)}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ (\varepsilon_1\ x1')\ x2'}) \le (\le_{R2\ x1'\ x2'})$
and $\bigwedge x1\ x2\ y'.\ x1 \le_{L1} x2 \Longrightarrow \textit{in-codom}\ (\le_{R2\ (l1\ x1)\ (l1\ x2)})\ y' \Longrightarrow$
$(\ge_{L2\ x1\ x2})\ (r2_{x1\ (l1\ x2)}\ y') \le (\ge_{L2\ x1\ x2})\ (r2_{x2\ (l1\ x2)}\ y')$
and $\bigwedge x1'\ x2'\ y.\ x1' \le_{R1} x2' \Longrightarrow \textit{in-dom}\ (\le_{L2\ (r1\ x1')\ (r1\ x2')})\ y \Longrightarrow$
$(\le_{R2\ x1'\ x2'})\ (l2_{x2'\ (r1\ x1')}\ y) \le (\le_{R2\ x1'\ x2'})\ (l2_{x1'\ (r1\ x1')}\ y)$
and $([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
shows $f\ {}_{L}\lessapprox\ g$
using *assms* **by** (*intro left-Galois-if-Dep-Fun-Rel-left-GaloisI in-codomI*[**where** *?x=l f*])
(*auto intro*!: *left-right-rel-if-Dep-Fun-Rel-left-GaloisI*)

lemma *left-Galois-iff-Dep-Fun-Rel-left-GaloisI*:
assumes $((\le_{L1}) \Rrightarrow_m (\le_{R1}))$ *l1*
and $((\le_{L1}) \trianglelefteq (\le_{R1}))$ *l1 r1*
and *reflexive-on* (*in-dom* $(\le_{L1})$) $(\le_{L1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\le_{R2\ (l1\ x)\ x'}) \Rrightarrow_m (\le_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ x1}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ (\eta_1\ x2)}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x1\ x2\ y'.\ x1 \le_{L1} x2 \Longrightarrow \textit{in-codom}\ (\le_{R2\ (l1\ x1)\ (l1\ x2)})\ y' \Longrightarrow$
$(\ge_{L2\ x1\ x2})\ (r2_{x1\ (l1\ x2)}\ y') \le (\ge_{L2\ x1\ x2})\ (r2_{x2\ (l1\ x2)}\ y')$
and $\bigwedge x\ x'\ y'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow \textit{in-dom}\ (\le_{R2\ (l1\ x)\ x'})\ y' \Longrightarrow$
$(\ge_{L2\ x\ (r1\ x')})\ (r2_{x\ (l1\ x)}\ y') \le (\ge_{L2\ x\ (r1\ x')})\ (r2_{x\ x'}\ y')$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow \textit{transitive}\ (\le_{L2\ x1\ x2})$
and $g \le_R g$
shows $f\ {}_{L}\lessapprox\ g \longleftrightarrow ([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
using *assms* **by** (*intro iffI*)
(*auto intro*!: *Dep-Fun-Rel-left-Galois-if-left-GaloisI left-Galois-if-Dep-Fun-Rel-left-GaloisI*)

lemma *left-Galois-iff-Dep-Fun-Rel-left-Galois-ge-left-rel2-assmI*:
assumes $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow$
$([\textit{in-codom}\ (\le_{R2\ (l1\ x1)\ (l1\ x2)})] \Rrightarrow (\le_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x2)})\ (r2_{x2\ (l1\ x2)})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow \textit{transitive}\ (\le_{L2\ x1\ x2})$
shows $\bigwedge x1\ x2\ y'.\ x1 \le_{L1} x2 \Longrightarrow \textit{in-codom}\ (\le_{R2\ (l1\ x1)\ (l1\ x2)})\ y' \Longrightarrow$
$(\ge_{L2\ x1\ x2})\ (r2_{x1\ (l1\ x2)}\ y') \le (\ge_{L2\ x1\ x2})\ (r2_{x2\ (l1\ x2)}\ y')$
using *assms* **by** *blast*

lemma *left-Galois-iff-Dep-Fun-Rel-left-Galois-ge-left-rel2-assmI'*:
assumes $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow$
$([\textit{in-dom}\ (\le_{R2\ (l1\ x)\ x'})] \Rrightarrow (\le_{L2\ x\ (r1\ x')}))\ (r2_{x\ (l1\ x)})\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow \textit{transitive}\ (\le_{L2\ x1\ x2})$
shows $\bigwedge x\ x'\ y'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow \textit{in-dom}\ (\le_{R2\ (l1\ x)\ x'})\ y' \Longrightarrow$
$(\ge_{L2\ x\ (r1\ x')})\ (r2_{x\ (l1\ x)}\ y') \le (\ge_{L2\ x\ (r1\ x')})\ (r2_{x\ x'}\ y')$

using *assms* **by** *blast*

lemma *left-Galois-iff-Dep-Fun-Rel-left-Galois-mono-assm-in-codom-rightI*:
assumes *mono-l1*: $((\le_{L1}) \Rrightarrow_m (\le_{R1}))$ *l1*
and *half-galois-prop-right1*: $((\le_{L1}) \trianglelefteq_h (\le_{R1}))$ *l1 r1*
and *refl-L1*: *reflexive-on* (*in-codom* $(\le_{L1})$) $(\le_{L1})$
and *L2-le-unit2*: $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ (\eta_1\ x2)}) \le (\le_{L2\ x1\ x2})$
and *mono-r2*: $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x2\ {}_{L1}\lessapprox x1'] \Rrightarrow$
$[\textit{in-field}\ (\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')}))$ *r2*
and $x1 \le_{L1} x2$
shows $([\textit{in-codom}\ (\le_{R2\ (l1\ x1)\ (l1\ x2)})] \Rrightarrow (\le_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x2)})\ (r2_{x2\ (l1\ x2)})$
proof (*intro Dep-Fun-Rel-predI*)
from *mono-l1 half-galois-prop-right1 refl-L1* ‹$x1 \le_{L1} x2$›
have $l1\ x2 \le_{R1} l1\ x2\ x2\ {}_{L1}\lessapprox l1\ x2$
by (*blast intro*: *t1.left-Galois-left-if-left-relI*)+
fix y' **assume** *in-codom* $(\le_{R2\ (l1\ x1)\ (l1\ x2)})\ y'$
with *Dep-Fun-Rel-relD*[*OF*
dep-mono-wrt-relD[*OF mono-r2* ‹$x1 \le_{L1} x2$›] ‹$l1\ x2 \le_{R1} l1\ x2$›]
have $r2_{x1\ (l1\ x2)}\ y' \le_{L2\ x1\ (\eta_1\ x2)} r2_{x2\ (l1\ x2)}\ y'$
using ‹$x2\ {}_{L1}\lessapprox l1\ x2$› **by** (*auto dest*: *in-field-if-in-codom*)
with *L2-le-unit2* ‹$x1 \le_{L1} x2$› **show** $r2_{x1\ (l1\ x2)}\ y' \le_{L2\ x1\ x2} r2_{x2\ (l1\ x2)}\ y'$
by *blast*
qed

lemma *left-Galois-iff-Dep-Fun-Rel-left-Galois-mono-assm-in-dom-rightI*:
assumes *mono-l1*: $((\le_{L1}) \Rrightarrow_m (\le_{R1}))$ *l1*
and *half-galois-prop-right1*: $((\le_{L1}) \trianglelefteq (\le_{R1}))$ *l1 r1*
and *refl-L1*: *reflexive-on* (*in-dom* $(\le_{L1})$) $(\le_{L1})$
and *mono-r2*: $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x2\ {}_{L1}\lessapprox x1'] \Rrightarrow$
$[\textit{in-field}\ (\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')}))$ *r2*
and $x\ {}_{L1}\lessapprox x'$
shows $([\textit{in-dom}\ (\le_{R2\ (l1\ x)\ x'})] \Rrightarrow (\le_{L2\ x\ (r1\ x')}))\ (r2_{x\ (l1\ x)})\ (r2_{x\ x'})$
proof −
from *mono-l1 half-galois-prop-right1 refl-L1* ‹$x\ {}_{L1}\lessapprox x'$›
have $x \le_{L1} x\ l1\ x \le_{R1} x'\ x\ {}_{L1}\lessapprox l1\ x$
by (*auto intro*!: *t1.half-galois-prop-leftD t1.left-Galois-left-if-left-relI*)
with *Dep-Fun-Rel-relD*[*OF dep-mono-wrt-relD*[*OF mono-r2* ‹$x \le_{L1} x$›] ‹$l1\ x$
$\le_{R1} x'$›]
show *?thesis* **by** *blast*
qed

lemma *left-Galois-iff-Dep-Fun-Rel-left-Galois-if-monoI*:
assumes $((\le_{L1}) \Rrightarrow_m (\le_{R1}))$ *l1*
and $((\le_{L1}) \trianglelefteq (\le_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\le_{R2\ (l1\ x)\ x'}) \Rrightarrow_m (\le_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ x1}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ (\eta_1\ x2)}) \le (\le_{L2\ x1\ x2})$

and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[\textit{in-field}\ (\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow \textit{transitive}\ (\le_{L2\ x1\ x2})$
and $g \le_R g$
shows $f\ {}_{L}\lessapprox\ g \longleftrightarrow ([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
using *assms* **by** (*intro left-Galois-iff-Dep-Fun-Rel-left-GaloisI*
left-Galois-iff-Dep-Fun-Rel-left-Galois-ge-left-rel2-assmI
left-Galois-iff-Dep-Fun-Rel-left-Galois-ge-left-rel2-assmI'
left-Galois-iff-Dep-Fun-Rel-left-Galois-mono-assm-in-dom-rightI
left-Galois-iff-Dep-Fun-Rel-left-Galois-mono-assm-in-codom-rightI)
(*auto intro*: *reflexive-on-if-le-pred-if-reflexive-on in-field-if-in-dom*
in-field-if-in-codom)

lemma *left-Galois-iff-Dep-Fun-Rel-left-Galois-left-rel2-le-assmI*:
assumes *refl-L1*: *reflexive-on* (*in-dom* $(\le_{L1})) (\le_{L1})$
and *mono-L2*: $([x1 :: \top] \Rrightarrow_m [x2\ \text{-} :: (\le_{L1}) \mid x1 \le_{L1} x2] \Rrightarrow_m (\le))\ L2$
and $x1 \le_{L1} x2$
shows $(\le_{L2\ x1\ x1}) \le (\le_{L2\ x1\ x2})$
proof −
from *refl-L1* ‹$x1 \le_{L1} x2$› **have** $x1 \le_{L1} x1$ **by** *blast*
with *dep-mono-wrt-relD*[*OF dep-mono-wrt-predD*[*OF mono-L2*] ‹$x1 \le_{L1} x2$›]
show $(\le_{L2\ x1\ x1}) \le (\le_{L2\ x1\ x2})$ **by** *auto*
qed

lemma *left-Galois-iff-Dep-Fun-Rel-left-Galois-left-rel2-unit1-le-assmI*:
assumes *mono-l1*: $((\le_{L1}) \Rrightarrow_m (\le_{R1}))\ l1$
and *half-galois-prop-right1*: $((\le_{L1}) \trianglelefteq_h (\le_{R1}))\ l1\ r1$
and *refl-L1*: *reflexive-on* (*in-codom* $(\le_{L1})) (\le_{L1})$
and *antimono-L2*:
$([x1 :: \top] \Rrightarrow_m [x2\ x3 :: (\le_{L1}) \mid (x1 \le_{L1} x2 \wedge x3 \le_{L1} \eta_1\ x2)] \Rrightarrow_m (\ge))\ L2$
and $x1 \le_{L1} x2$
shows $(\le_{L2\ x1\ (\eta_1\ x2)}) \le (\le_{L2\ x1\ x2})$
proof −
from *mono-l1 half-galois-prop-right1 refl-L1* ‹$x1 \le_{L1} x2$› **have** $x2 \le_{L1} \eta_1\ x2$
by (*blast intro*: *t1.rel-unit-if-reflexive-on-if-half-galois-prop-right-if-mono-wrt-rel*)
with *refl-L1* **have** $\eta_1\ x2 \le_{L1} \eta_1\ x2$ **by** *blast*
with *dep-mono-wrt-relD*[*OF dep-mono-wrt-predD*[*OF antimono-L2*] ‹$x2 \le_{L1} \eta_1$
$x2$›]
show $(\le_{L2\ x1\ (\eta_1\ x2)}) \le (\le_{L2\ x1\ x2})$ **using** ‹$x1 \le_{L1} x2$› **by** *auto*
qed

lemma *left-Galois-iff-Dep-Fun-Rel-left-Galois-if-monoI'*:
assumes $((\le_{L1}) \Rrightarrow_m (\le_{R1}))\ l1$
and $((\le_{L1}) \trianglelefteq (\le_{R1}))\ l1\ r1$
and *reflexive-on* (*in-field* $(\le_{L1})) (\le_{L1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\le_{R2\ (l1\ x)\ x'}) \Rrightarrow_m (\le_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $([x1 :: \top] \Rrightarrow_m [x2\ \text{-} :: (\le_{L1}) \mid x1 \le_{L1} x2] \Rrightarrow_m (\le))\ L2$
and $([x1 :: \top] \Rrightarrow_m [x2\ x3 :: (\le_{L1}) \mid (x1 \le_{L1} x2 \wedge x3 \le_{L1} \eta_1\ x2)] \Rrightarrow_m (\ge))\ L2$
and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$

$[in\text{-}field\ (\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')})) r2$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow transitive\ (\le_{L2\ x1\ x2})$
and $g \le_R g$
shows $f\ {}_{L}\lessapprox\ g \longleftrightarrow ([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
using *assms* **by** (*intro left-Galois-iff-Dep-Fun-Rel-left-Galois-if-monoI left-Galois-iff-Dep-Fun-Rel-left-Galois-left-rel2-unit1-le-assmI left-Galois-iff-Dep-Fun-Rel-left-Galois-left-rel2-le-assmI*)
(*auto intro*: *reflexive-on-if-le-pred-if-reflexive-on in-field-if-in-codom in-field-if-in-dom*)

corollary *left-Galois-iff-Dep-Fun-Rel-left-Galois-if-mono-if-galois-connectionI*:
assumes $((\le_{L1}) \dashv (\le_{R1}))\ l1\ r1$
and $reflexive\text{-}on\ (in\text{-}field\ (\le_{L1}))\ (\le_{L1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\le_{R2\ (l1\ x)\ x'}) \Rrightarrow_m (\le_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $([x1 :: \top] \Rrightarrow_m [x2\ \text{-} :: (\le_{L1}) \mid x1 \le_{L1} x2] \Rrightarrow_m (\le))\ L2$
and $([x1 :: \top] \Rrightarrow_m [x2\ x3 :: (\le_{L1}) \mid (x1 \le_{L1} x2 \wedge x3 \le_{L1} \eta_1\ x2)] \Rrightarrow_m (\ge))\ L2$
and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[in\text{-}field\ (\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow transitive\ (\le_{L2\ x1\ x2})$
and $g \le_R g$
shows $f\ {}_{L}\lessapprox\ g \longleftrightarrow ([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
using *assms* **by** (*intro left-Galois-iff-Dep-Fun-Rel-left-Galois-if-monoI'*) *auto*

interpretation *flip-inv* : $galois\ (\ge_{R1})\ (\ge_{L1})\ r1\ l1$ **.**

lemma *left-Galois-iff-Dep-Fun-Rel-left-Galois-left-rel2-unit1-le-assm-if-galois-equivI*:
assumes *galois-equiv1*: $((\le_{L1}) \equiv_G (\le_{R1}))\ l1\ r1$
and *refl-L1*: $reflexive\text{-}on\ (in\text{-}field\ (\le_{L1}))\ (\le_{L1})$
and *mono-L2*: $([x1 :: \top] \Rrightarrow_m [x2\ \text{-} :: (\le_{L1}) \mid x1 \le_{L1} x2] \Rrightarrow_m (\le))\ L2$
and $x1 \le_{L1} x2$
shows $(\le_{L2\ x1\ (\eta_1\ x2)}) \le (\le_{L2\ x1\ x2})$
proof −
from *refl-L1* ‹$x1 \le_{L1} x2$› **have** $x1 \le_{L1} x1$ **by** *blast*
from *galois-equiv1 refl-L1* ‹$x1 \le_{L1} x2$› **have** $\eta_1\ x2 \le_{L1} x2$ **by** (*intro flip.t1.counit-rel-if-reflexive-on-if-half-galois-prop-left-if-mono-wrt-rel*)
blast+
have $x1 \le_{L1} \eta_1\ x2$ **by** (*rule t1.rel-unit-if-left-rel-if-mono-wrt-relI*)
(*insert galois-equiv1 refl-L1* ‹$x1 \le_{L1} x2$›, *auto*)
with *dep-mono-wrt-relD*[*OF dep-mono-wrt-predD*[*OF mono-L2*] ‹$\eta_1\ x2 \le_{L1} x2$›]
show $(\le_{L2\ x1\ (\eta_1\ x2)}) \le (\le_{L2\ x1\ x2})$ **by** *auto*
qed

lemma *left-Galois-iff-Dep-Fun-Rel-left-Galois-if-galois-equivalenceI*:
assumes $((\le_{L1}) \equiv_G (\le_{R1}))\ l1\ r1$
and $reflexive\text{-}on\ (in\text{-}field\ (\le_{L1}))\ (\le_{L1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\le_{R2\ (l1\ x)\ x'}) \Rrightarrow_m (\le_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $([x1 :: \top] \Rrightarrow_m [x2\ \text{-} :: (\le_{L1}) \mid x1 \le_{L1} x2] \Rrightarrow_m (\le))\ L2$
and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$

$[in\text{-}field\ (\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')})) \ r2$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow transitive\ (\le_{L2\ x1\ x2})$
and $g \le_R g$
shows $f\ {}_{L}\lessapprox\ g \longleftrightarrow ([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
using *assms* **by** (*intro*
left-Galois-iff-Dep-Fun-Rel-left-Galois-if-monoI
left-Galois-iff-Dep-Fun-Rel-left-Galois-left-rel2-le-assmI
left-Galois-iff-Dep-Fun-Rel-left-Galois-left-rel2-unit1-le-assm-if-galois-equivI)
auto

corollary *left-Galois-iff-Dep-Fun-Rel-left-Galois-if-galois-equivalenceI'*:
assumes $((\le_{L1}) \equiv_G (\le_{R1}))\ l1\ r1$
and $reflexive\text{-}on\ (in\text{-}field\ (\le_{L1}))\ (\le_{L1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\le_{R2\ (l1\ x)\ x'}) \Rrightarrow_m (\le_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $([x1\ x2 :: (\ge_{L1})] \Rrightarrow_m [x3\ x4 :: (\le_{L1}) \mid x1 \le_{L1} x3] \Rrightarrow (\le))\ L2$
and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[in\text{-}field\ (\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow transitive\ (\le_{L2\ x1\ x2})$
and $g \le_R g$
shows $f\ {}_{L}\lessapprox\ g \longleftrightarrow ([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
using *assms* **by** (*intro left-Galois-iff-Dep-Fun-Rel-left-Galois-if-galois-equivalenceI*
reflexive-on-in-field-mono-assm-left2I)
auto

corollary *left-Galois-iff-Dep-Fun-Rel-left-Galois-if-preorder-equivalenceI*:
assumes $((\le_{L1}) \equiv_{pre} (\le_{R1}))\ l1\ r1$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\le_{R2\ (l1\ x)\ x'}) \Rrightarrow_m (\le_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $([x1\ x2 :: (\ge_{L1})] \Rrightarrow_m [x3\ x4 :: (\le_{L1}) \mid x1 \le_{L1} x3] \Rrightarrow (\le))\ L2$
and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[in\text{-}field\ (\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow transitive\ (\le_{L2\ x1\ x2})$
and $g \le_R g$
shows $f\ {}_{L}\lessapprox\ g \longleftrightarrow ([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
using *assms* **by** (*intro left-Galois-iff-Dep-Fun-Rel-left-Galois-if-galois-equivalenceI'*)
auto

corollary *left-Galois-iff-Dep-Fun-Rel-left-Galois-if-preorder-equivalenceI'*:
assumes $((\le_{L1}) \equiv_{pre} (\le_{R1}))\ l1\ r1$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\le_{L2\ x\ (r1\ x')}) \equiv_{pre} (\le_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $([x1\ x2 :: (\ge_{L1})] \Rrightarrow_m [x3\ x4 :: (\le_{L1}) \mid x1 \le_{L1} x3] \Rrightarrow (\le))\ L2$
and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[in\text{-}field\ (\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')}))\ r2$
and $g \le_R g$
shows $f\ {}_{L}\lessapprox\ g \longleftrightarrow ([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
using *assms* **by** (*intro left-Galois-iff-Dep-Fun-Rel-left-Galois-if-preorder-equivalenceI*
transitive-left2-if-preorder-equivalenceI)
(*auto 5 0*)

Simplification of Restricted Function Relator **lemma** *Dep-Fun-Rel-left-Galois-restrict-left-right-eq*
assumes $((\le_{L1}) \Rrightarrow_m (\le_{R1}))$ *l1* **and** $((\le_{R1}) \Rrightarrow_m (\le_{L1}))$ *r1*
and $((\le_{L1}) \trianglelefteq_h (\le_{R1}))$ *l1 r1*
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow$
$((\le_{L2\ (r1\ x1')\ (r1\ x2')})\ {}_h\trianglelefteq\ (\le_{R2\ (\varepsilon_1\ x1')\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (r2_{(r1\ x1')\ x2'})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ (\eta_1\ x2)}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ (\varepsilon_1\ x1')\ x2'}) \le (\le_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'\ y.\ x1' \le_{R1} x2' \Longrightarrow$ *in-dom* $(\le_{L2\ (r1\ x1')\ (r1\ x2')})\ y \Longrightarrow$
$(\le_{R2\ x1'\ x2'})\ (l2_{x2'\ (r1\ x1')}\ y) \le (\le_{R2\ x1'\ x2'})\ (l2_{x1'\ (r1\ x1')}\ y)$
and $\bigwedge x1\ x2\ y'.\ x1 \le_{L1} x2 \Longrightarrow$ *in-codom* $(\le_{R2\ (l1\ x1)\ (l1\ x2)})\ y' \Longrightarrow$
$(\ge_{L2\ x1\ x2})\ (r2_{x1\ (l1\ x2)}\ y') \le (\ge_{L2\ x1\ x2})\ (r2_{x2\ (l1\ x2)}\ y')$
shows $([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\restriction_{in\text{-}dom\ (\le_L)}\upharpoonright_{in\text{-}codom\ (\le_R)}$
$= ([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))$
using *assms* **by** (*intro ext iffI restrict-leftI restrict-rightI*
in-domI[**where** *?y=r -*] *left-rel-right-if-Dep-Fun-Rel-left-GaloisI*
in-codomI[**where** *?x=l -*] *left-right-rel-if-Dep-Fun-Rel-left-GaloisI*)
auto

lemma *Dep-Fun-Rel-left-Galois-restrict-left-right-eq-Dep-Fun-Rel-left-GaloisI'*:
assumes $((\le_{L1}) \dashv (\le_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$
and *reflexive-on* (*in-field* $(\le_{R1})$) $(\le_{R1})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow$
$((\le_{L2\ (r1\ x1')\ (r1\ x2')})\ {}_h\trianglelefteq\ (\le_{R2\ (\varepsilon_1\ x1')\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (r2_{(r1\ x1')\ x2'})$
and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x3\ x4 :: (\le_{L1}) \mid (x2 \le_{L1} x3 \wedge x4 \le_{L1} \eta_1\ x3)] \Rrightarrow (\ge))$ *L2*
and $([x1'\ x2' :: (\le_{R1}) \mid \varepsilon_1\ x2' \le_{R1} x1'] \Rrightarrow_m [x3'\ x4' :: (\le_{R1}) \mid x2' \le_{R1} x3'] \Rrightarrow (\le))$ *R2*
and $([x1'\ x2' :: (\le_{R1})] \Rrightarrow_m [x1\ x2 :: (\le_{L1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[$*in-field* $(\le_{L2\ x1\ (r1\ x2')})] \Rrightarrow (\le_{R2\ (l1\ x1)\ x2'}))$ *l2*
and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[$*in-field* $(\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')}))$ *r2*
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow$ *transitive* $(\le_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow$ *transitive* $(\le_{R2\ x1'\ x2'})$
shows $([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\restriction_{in\text{-}dom\ (\le_L)}\upharpoonright_{in\text{-}codom\ (\le_R)}$
$= ([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))$
using *assms* **by** (*intro*
Dep-Fun-Rel-left-Galois-restrict-left-right-eq-Dep-Fun-Rel-left-GaloisI
left-rel-right-iff-left-right-rel-if-galois-prop-le-assms-leftI
reflexive-on-in-field-mono-assm-left2I
left-rel-right-iff-left-right-rel-if-galois-prop-le-assms-rightI
mono-wrt-rel-left-in-dom-mono-left-assm
galois-connection-left-right-if-galois-connection-mono-assms-leftI
galois-connection-left-right-if-galois-connection-mono-assms-rightI
left-Galois-iff-Dep-Fun-Rel-left-Galois-ge-left-rel2-assmI)
auto

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 $((\le_{L1})\ {}_h\trianglelefteq\ (\le_{R1}))\ l1\ r1$
shows $([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow (S\ x\ x')\restriction_{in\text{-}dom\ (\le_{L2\ x\ (r1\ x')})}\upharpoonright_{in\text{-}codom\ (\le_{R2\ (l1\ x)\ x'})})$
$\restriction_{in\text{-}dom\ (\le_L)}\upharpoonright_{in\text{-}codom\ (\le_R)} =$
$([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow S\ x\ x')\restriction_{in\text{-}dom\ (\le_L)}\upharpoonright_{in\text{-}codom\ (\le_R)}$ (**is** *?lhs* = *?rhs*)
proof –
have *?lhs* =
$([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow (S\ x\ x')\upharpoonright_{in\text{-}codom\ (\le_{R2\ (l1\ x)\ x'})})$
$\restriction_{in\text{-}dom\ (\le_L)}\upharpoonright_{in\text{-}codom\ (\le_R)}$
by (*subst restrict-left-right-eq-restrict-right-left*,
subst restrict-left-Dep-Fun-Rel-rel-restrict-left-eq)
auto
also have ... = *?rhs*
using *assms* **by** (*subst restrict-left-right-eq-restrict-right-left*,
subst restrict-right-Dep-Fun-Rel-rel-restrict-right-eq)
(*auto elim*!: *in-codomE t1.left-GaloisE*
simp only: *restrict-left-right-eq-restrict-right-left*)
finally show *?thesis* **.**
qed

end

Function Relator

context *transport-Fun-Rel*
begin

corollary *Fun-Rel-left-Galois-if-left-GaloisI*:
assumes $((\le_{L1})\ {}_h\trianglelefteq\ (\le_{R1}))\ l1\ r1$
and *reflexive-on* $(in\text{-}dom\ (\le_{L1}))\ (\le_{L1})$
and $((\le_{R2}) \Rrightarrow_m (\le_{L2}))\ r2$
and *transitive* $(\le_{L2})$
and $g \le_R g$
and $f\ {}_L\lessapprox\ g$
shows $(({}_{L1}\lessapprox) \Rrightarrow ({}_{L2}\lessapprox))\ f\ g$
using *assms* **by** (*intro tdfr.Dep-Fun-Rel-left-Galois-if-left-GaloisI*) *simp-all*

corollary *left-Galois-if-Fun-Rel-left-GaloisI*:
assumes $((\le_{L1}) \Rrightarrow_m (\le_{R1}))\ l1$
and $((\le_{L1}) \trianglelefteq_h (\le_{R1}))\ l1\ r1$
and *in-codom* $(\le_R)\ g$
and $(({}_{L1}\lessapprox) \Rrightarrow ({}_{L2}\lessapprox))\ f\ g$
shows $f\ {}_L\lessapprox\ g$
using *assms* **by** (*intro tdfr.left-Galois-if-Dep-Fun-Rel-left-GaloisI*) *simp-all*

lemma *left-Galois-if-Fun-Rel-left-GaloisI'*:
assumes $((\le_{L1}) \Rrightarrow_m (\le_{R1}))\ l1$ **and** $((\le_{R1}) \Rrightarrow_m (\le_{L1}))\ r1$

and $((\le_{L1}) \trianglelefteq_h (\le_{R1}))$ *l1 r1*
and $((\le_{L2})\ {}_h\trianglelefteq (\le_{R2}))$ *l2 r2*
and $(({}_{L1}\lessapprox) \Rrightarrow ({}_{L2}\lessapprox))\ f\ g$
shows $f\ {}_{L}\lessapprox\ g$
using *assms* **by** (*intro tdfr.left-Galois-if-Dep-Fun-Rel-left-GaloisI′*) *simp-all*

corollary *left-Galois-iff-Fun-Rel-left-GaloisI*:
assumes $((\le_{L1}) \Rrightarrow_m (\le_{R1}))$ *l1*
and $((\le_{L1}) \trianglelefteq (\le_{R1}))$ *l1 r1*
and *reflexive-on* (*in-dom* $(\le_{L1})$) $(\le_{L1})$
and $((\le_{R2}) \Rrightarrow_m (\le_{L2}))$ *r2*
and *transitive* $(\le_{L2})$
and $g \le_R g$
shows $f\ {}_{L}\lessapprox\ g \longleftrightarrow (({}_{L1}\lessapprox) \Rrightarrow ({}_{L2}\lessapprox))\ f\ g$
using *assms* **by** (*intro tdfr.left-Galois-iff-Dep-Fun-Rel-left-GaloisI*) *simp-all*

Simplification of Restricted Function Relator **lemma** *Fun-Rel-left-Galois-restrict-left-right-eq-Fun*
assumes $((\le_{L1}) \Rrightarrow_m (\le_{R1}))$ *l1* **and** $((\le_{R1}) \Rrightarrow_m (\le_{L1}))$ *r1*
and $((\le_{L1}) \trianglelefteq_h (\le_{R1}))$ *l1 r1*
and $((\le_{L2})\ {}_h\trianglelefteq (\le_{R2}))$ *l2 r2*
shows $(({}_{L1}\lessapprox) \Rrightarrow ({}_{L2}\lessapprox))\restriction_{in\text{-}dom\ (\le_L)}\upharpoonright_{in\text{-}codom\ (\le_R)} = (({}_{L1}\lessapprox) \Rrightarrow ({}_{L2}\lessapprox))$
using *assms*
by (*intro tdfr.Dep-Fun-Rel-left-Galois-restrict-left-right-eq-Dep-Fun-Rel-left-GaloisI*)
simp-all

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 $((\le_{L1})\ {}_h\trianglelefteq (\le_{R1}))$ *l1 r1*
shows $(({}_{L1}\lessapprox) \Rrightarrow S\restriction_{in\text{-}dom\ (\le_{L2})}\upharpoonright_{in\text{-}codom\ (\le_{R2})})\restriction_{in\text{-}dom\ (\le_L)}\upharpoonright_{in\text{-}codom\ (\le_R)}$
$=$
$(({}_{L1}\lessapprox) \Rrightarrow S)\restriction_{in\text{-}dom\ (\le_L)}\upharpoonright_{in\text{-}codom\ (\le_R)}$
using *assms*
by (*intro tdfr.Dep-Fun-Rel-left-Galois-restrict-left-right-restrict-left-right-eq*)
simp-all

end

Monotone Dependent Function Relator **context** *transport-Mono-Dep-Fun-Rel*
begin

lemma *Dep-Fun-Rel-left-Galois-if-left-GaloisI*:
assumes $((\le_{L1})\ {}_h\trianglelefteq (\le_{R1}))$ *l1 r1*
and *reflexive-on* (*in-dom* $(\le_{L1})$) $(\le_{L1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\le_{R2\ (l1\ x)\ x'}) \Rrightarrow_m (\le_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ x1}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x\ x'\ y'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow in\text{-}dom\ (\le_{R2\ (l1\ x)\ x'})\ y' \Longrightarrow$
$(\ge_{L2\ x\ (r1\ x')})\ (r2_{x\ (l1\ x)}\ y') \le (\ge_{L2\ x\ (r1\ x')})\ (r2_{x\ x'}\ y')$

and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow$ *transitive* $(\le_{L2\ x1\ x2})$
and $f\ {}_{L}\lessapprox\ g$
shows $([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
using *assms* **unfolding** *left-rel-eq-tdfr-left-Refl-Rel right-rel-eq-tdfr-right-Refl-Rel*
by (*intro tdfr.Dep-Fun-Rel-left-Galois-if-left-GaloisI tdfr.left-GaloisI*)
(*auto elim*!: *galois-rel.left-GaloisE in-codomE*)

lemma *left-Galois-if-Dep-Fun-Rel-left-GaloisI*:
assumes $(tdfr.R \Rrightarrow_m tdfr.L)\ r$
and $((\le_{L1}) \Rrightarrow_m (\le_{R1}))\ l1$
and $((\le_{L1}) \trianglelefteq_h (\le_{R1}))\ l1\ r1$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ (\eta_1\ x2)}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x1\ x2\ y'.\ x1 \le_{L1} x2 \Longrightarrow$ *in-codom* $(\le_{R2\ (l1\ x1)\ (l1\ x2)})\ y' \Longrightarrow$
$(\ge_{L2\ x1\ x2})\ (r2_{x1\ (l1\ x2)}\ y') \le (\ge_{L2\ x1\ x2})\ (r2_{x2\ (l1\ x2)}\ y')$
and *in-dom* $(\le_L)\ f$
and *in-codom* $(\le_R)\ g$
and $([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
shows $f\ {}_{L}\lessapprox\ g$
using *assms* **unfolding** *left-rel-eq-tdfr-left-Refl-Rel right-rel-eq-tdfr-right-Refl-Rel*
by (*intro tdfr.Galois-Refl-RelI tdfr.left-Galois-if-Dep-Fun-Rel-left-GaloisI*)
(*auto simp*: *in-codom-eq-in-dom-if-reflexive-on-in-field*)

lemma *left-Galois-iff-Dep-Fun-Rel-left-GaloisI*:
assumes $(tdfr.R \Rrightarrow_m tdfr.L)\ r$
and $((\le_{L1}) \Rrightarrow_m (\le_{R1}))\ l1$
and $((\le_{L1}) \trianglelefteq (\le_{R1}))\ l1\ r1$
and *reflexive-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\le_{R2\ (l1\ x)\ x'}) \Rrightarrow_m (\le_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ x1}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ (\eta_1\ x2)}) \le (\le_{L2\ x1\ x2})$
and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1})\ |\ x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[\textit{in-field}\ (\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow$ *transitive* $(\le_{L2\ x1\ x2})$
and *in-dom* $(\le_L)\ f$
and *in-codom* $(\le_R)\ g$
shows $f\ {}_{L}\lessapprox\ g \longleftrightarrow ([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
using *assms* **by** (*intro iffI Dep-Fun-Rel-left-Galois-if-left-GaloisI*
tdfr.left-Galois-iff-Dep-Fun-Rel-left-Galois-ge-left-rel2-assmI'
tdfr.left-Galois-iff-Dep-Fun-Rel-left-Galois-mono-assm-in-dom-rightI)
(*auto intro*!: *left-Galois-if-Dep-Fun-Rel-left-GaloisI*
tdfr.left-Galois-iff-Dep-Fun-Rel-left-Galois-ge-left-rel2-assmI
tdfr.left-Galois-iff-Dep-Fun-Rel-left-Galois-mono-assm-in-codom-rightI
intro: *reflexive-on-if-le-pred-if-reflexive-on*
in-field-if-in-dom in-field-if-in-codom)

lemma *left-Galois-iff-Dep-Fun-Rel-left-Galois-if-mono-if-galois-connectionI*:
assumes *galois-conn1*: $((\le_{L1}) \dashv (\le_{R1}))\ l1\ r1$
and *refl-L1*: *reflexive-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$

and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\le_{R2\ (l1\ x)\ x'}) \Rrightarrow_m (\le_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ x1}) \le (\le_{L2\ x1\ x2})$
and *L2-le-unit2*: $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ (\eta_1\ x2)}) \le (\le_{L2\ x1\ x2})$
and *mono-r2*: $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[\textit{in-field}\ (\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')}))\ r2$
and *trans-L2*: $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow \textit{transitive}\ (\le_{L2\ x1\ x2})$
and *in-dom* $(\le_L)\ f$
and *in-codom* $(\le_R)\ g$
shows $f\ {}_{L}\lessapprox\ g \longleftrightarrow ([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$ (**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof –
have $(\le_{L2\ x1\ x2})\ (r2_{x1\ (l1\ x2)}\ y') \le (\le_{L2\ x1\ x2})\ (r2_{x1\ (l1\ x1)}\ y')$
if *hyps*: $x1 \le_{L1} x2$ *in-dom* $(\le_{R2\ (l1\ x1)\ (l1\ x2)})\ y'$ **for** $x1\ x2\ y'$
proof –
have $([\textit{in-dom}\ (\le_{R2\ (l1\ x1)\ (l1\ x2)})] \Rrightarrow (\le_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x1)})\ (r2_{x1\ (l1\ x2)})$
proof (*intro Dep-Fun-Rel-predI*)
from *galois-conn1 refl-L1* ‹$x1 \le_{L1} x2$›
have $x1 \le_{L1} x1\ \ l1\ x1 \le_{R1} l1\ x2\ \ x1\ {}_{L1}\lessapprox\ l1\ x1$
by (*blast intro*: *t1.left-Galois-left-if-left-relI*)+
fix y' **assume** *in-dom* $(\le_{R2\ (l1\ x1)\ (l1\ x2)})\ y'$
with *Dep-Fun-Rel-relD*[*OF dep-mono-wrt-relD*[*OF mono-r2* ‹$x1 \le_{L1} x1$›]
‹$l1\ x1 \le_{R1} l1\ x2$›]
have $r2_{x1\ (l1\ x1)}\ y' \le_{L2\ x1\ (\eta_1\ x2)} r2_{x1\ (l1\ x2)}\ y'$
using ‹$x1\ {}_{L1}\lessapprox\ l1\ x1$› **by** (*auto dest*: *in-field-if-in-dom*)
with *L2-le-unit2* ‹$x1 \le_{L1} x2$› **show** $r2_{x1\ (l1\ x1)}\ y' \le_{L2\ x1\ x2} r2_{x1\ (l1\ x2)}$
y'
by *blast*
qed
with *hyps* **show** *?thesis* **using** *trans-L2* **by** *blast*
qed
then show *?thesis* **using** *assms*
using *assms* **by** (*intro left-Galois-iff-Dep-Fun-Rel-left-GaloisI*
tdfr.mono-wrt-rel-rightI
tdfr.mono-wrt-rel-right2-if-mono-wrt-rel-right2-if-left-GaloisI
tdfr.left-Galois-iff-Dep-Fun-Rel-left-Galois-ge-left-rel2-assmI
tdfr.left-Galois-iff-Dep-Fun-Rel-left-Galois-mono-assm-in-codom-rightI)
(*auto intro*: *reflexive-on-if-le-pred-if-reflexive-on in-field-if-in-codom*)
qed

corollary *left-Galois-iff-Dep-Fun-Rel-left-Galois-if-mono-if-galois-connectionI'*:
assumes $((\le_{L1}) \dashv (\le_{R1}))\ l1\ r1$
and *reflexive-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\le_{R2\ (l1\ x)\ x'}) \Rrightarrow_m (\le_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $([x1 :: \top] \Rrightarrow_m [x2\ \text{-} :: (\le_{L1}) \mid x1 \le_{L1} x2] \Rrightarrow_m (\le))\ L2$
and $([x1 :: \top] \Rrightarrow_m [x2\ x3 :: (\le_{L1}) \mid (x1 \le_{L1} x2 \wedge x3 \le_{L1} \eta_1\ x2)] \Rrightarrow_m (\ge))\ L2$
and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[\textit{in-field}\ (\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow \textit{transitive}\ (\le_{L2\ x1\ x2})$

and *in-dom* $(\le_L)$ *f*
and *in-codom* $(\le_R)$ *g*
shows $f\ {}_{L}\lessapprox\ g \longleftrightarrow ([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$ (**is** *?lhs* $\longleftrightarrow$ *?rhs*)
using *assms* **by** (*intro*
left-Galois-iff-Dep-Fun-Rel-left-Galois-if-mono-if-galois-connectionI
tdfr.left-Galois-iff-Dep-Fun-Rel-left-Galois-left-rel2-unit1-le-assmI
tdfr.left-Galois-iff-Dep-Fun-Rel-left-Galois-left-rel2-le-assmI)
auto

corollary *left-Galois-eq-Dep-Fun-Rel-left-Galois-restrict-if-mono-if-galois-connectionI*:
assumes $((\le_{L1}) \dashv (\le_{R1}))\ l1\ r1$
and *reflexive-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\le_{R2\ (l1\ x)\ x'}) \Rrightarrow_m (\le_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $([x1 :: \top] \Rrightarrow_m [x2\ \text{-} :: (\le_{L1}) \mid x1 \le_{L1} x2] \Rrightarrow_m (\le))\ L2$
and $([x1 :: \top] \Rrightarrow_m [x2\ x3 :: (\le_{L1}) \mid (x1 \le_{L1} x2 \wedge x3 \le_{L1} \eta_1\ x2)] \Rrightarrow_m (\ge))\ L2$
and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[\textit{in-field}\ (\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow \textit{transitive}\ (\le_{L2\ x1\ x2})$
shows $({}_{L}\lessapprox) = ([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\restriction_{\textit{in-dom}\ (\le_L)}\upharpoonright_{\textit{in-codom}\ (\le_R)}$
using *assms* **by** (*intro ext iffI restrict-leftI restrict-rightI*
iffD1[*OF left-Galois-iff-Dep-Fun-Rel-left-Galois-if-mono-if-galois-connectionI'*])
(*auto intro*!:
iffD2[*OF left-Galois-iff-Dep-Fun-Rel-left-Galois-if-mono-if-galois-connectionI'*])

lemma *left-Galois-iff-Dep-Fun-Rel-left-Galois-if-galois-equivalenceI*:
assumes $((\le_{L1}) \equiv_G (\le_{R1}))\ l1\ r1$
and *reflexive-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\le_{R2\ (l1\ x)\ x'}) \Rrightarrow_m (\le_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $([x1\ x2 :: (\ge_{L1})] \Rrightarrow_m [x3\ x4 :: (\le_{L1}) \mid x1 \le_{L1} x3] \Rrightarrow (\le))\ L2$
and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[\textit{in-field}\ (\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow \textit{transitive}\ (\le_{L2\ x1\ x2})$
and *in-dom* $(\le_L)$ *f*
and *in-codom* $(\le_R)$ *g*
shows $f\ {}_{L}\lessapprox\ g \longleftrightarrow ([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
using *assms* **by** (*intro left-Galois-iff-Dep-Fun-Rel-left-Galois-if-mono-if-galois-connectionI*
tdfr.left-Galois-iff-Dep-Fun-Rel-left-Galois-left-rel2-le-assmI
tdfr.reflexive-on-in-field-mono-assm-left2I
tdfr.left-Galois-iff-Dep-Fun-Rel-left-Galois-left-rel2-unit1-le-assm-if-galois-equivI)
auto

theorem *left-Galois-eq-Dep-Fun-Rel-left-Galois-restrict-if-galois-equivalenceI*:
assumes $((\le_{L1}) \equiv_G (\le_{R1}))\ l1\ r1$
and *reflexive-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\le_{R2\ (l1\ x)\ x'}) \Rrightarrow_m (\le_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $([x1\ x2 :: (\ge_{L1})] \Rrightarrow_m [x3\ x4 :: (\le_{L1}) \mid x1 \le_{L1} x3] \Rrightarrow (\le))\ L2$
and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[\textit{in-field}\ (\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')}))\ r2$

and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow \mathit{transitive}\ (\le_{L2\ x1\ x2})$
shows $({}_{L}\lessapprox) = ([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\restriction_{\mathit{in\text{-}dom}\ (\le_L)}\upharpoonleft_{\mathit{in\text{-}codom}\ (\le_R)}$
using *assms* **by** (*intro ext iffI restrict-leftI restrict-rightI*
iffD1[*OF left-Galois-iff-Dep-Fun-Rel-left-Galois-if-galois-equivalenceI*])
(*auto intro*!: *iffD2*[*OF left-Galois-iff-Dep-Fun-Rel-left-Galois-if-galois-equivalenceI*])

corollary *left-Galois-iff-Dep-Fun-Rel-left-Galois-if-preorder-equivalenceI*:
assumes $((\le_{L1}) \equiv_{pre} (\le_{R1}))\ l1\ r1$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\le_{R2\ (l1\ x)\ x'}) \Rrightarrow_m (\le_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $([x1\ x2 :: (\ge_{L1})] \Rrightarrow_m [x3\ x4 :: (\le_{L1}) \mid x1 \le_{L1} x3] \Rrightarrow (\le))\ L2$
and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[\mathit{in\text{-}field}\ (\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow \mathit{transitive}\ (\le_{L2\ x1\ x2})$
and $\mathit{in\text{-}dom}\ (\le_L)\ f$
and $\mathit{in\text{-}codom}\ (\le_R)\ g$
shows $f\ {}_{L}\lessapprox\ g \longleftrightarrow ([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
using *assms* **by** (*intro left-Galois-iff-Dep-Fun-Rel-left-Galois-if-galois-equivalenceI*)
auto

corollary *left-Galois-eq-Dep-Fun-Rel-left-Galois-restrict-if-preorder-equivalenceI*:
assumes $((\le_{L1}) \equiv_{pre} (\le_{R1}))\ l1\ r1$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\le_{R2\ (l1\ x)\ x'}) \Rrightarrow_m (\le_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $([x1\ x2 :: (\ge_{L1})] \Rrightarrow_m [x3\ x4 :: (\le_{L1}) \mid x1 \le_{L1} x3] \Rrightarrow (\le))\ L2$
and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[\mathit{in\text{-}field}\ (\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow \mathit{transitive}\ (\le_{L2\ x1\ x2})$
shows $({}_{L}\lessapprox) = ([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\restriction_{\mathit{in\text{-}dom}\ (\le_L)}\upharpoonleft_{\mathit{in\text{-}codom}\ (\le_R)}$
using *assms* **by** (*intro ext iffI restrict-leftI restrict-rightI*
iffD1[*OF left-Galois-iff-Dep-Fun-Rel-left-Galois-if-preorder-equivalenceI*])
(*auto intro*!: *iffD2*[*OF left-Galois-iff-Dep-Fun-Rel-left-Galois-if-preorder-equivalenceI*])

corollary *left-Galois-iff-Dep-Fun-Rel-left-Galois-if-preorder-equivalenceI'*:
assumes $((\le_{L1}) \equiv_{pre} (\le_{R1}))\ l1\ r1$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\le_{L2\ x\ (r1\ x')}) \equiv_{pre} (\le_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $([x1\ x2 :: (\ge_{L1})] \Rrightarrow_m [x3\ x4 :: (\le_{L1}) \mid x1 \le_{L1} x3] \Rrightarrow (\le))\ L2$
and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[\mathit{in\text{-}field}\ (\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')}))\ r2$
and $\mathit{in\text{-}dom}\ (\le_L)\ f$
and $\mathit{in\text{-}codom}\ (\le_R)\ g$
shows $f\ {}_{L}\lessapprox\ g \longleftrightarrow ([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
using *assms* **by** (*intro left-Galois-iff-Dep-Fun-Rel-left-Galois-if-preorder-equivalenceI*
tdfr.transitive-left2-if-preorder-equivalenceI)
(*auto 5 0*)

corollary *left-Galois-eq-Dep-Fun-Rel-left-Galois-restrict-if-preorder-equivalenceI'*:
assumes $((\le_{L1}) \equiv_{pre} (\le_{R1}))\ l1\ r1$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\le_{L2\ x\ (r1\ x')}) \equiv_{pre} (\le_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$

and $(([x1\ x2 :: (\geq_{L1})] \Rrightarrow_m [x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3] \Rrightarrow (\leq))\ L2$
and $(([x1\ x2 :: (\leq_{L1})] \Rrightarrow_m [x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[in\text{-}field\ (\leq_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
shows $({}_{L}\lessapprox) = ([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\restriction_{in\text{-}dom\ (\leq_L)}\upharpoonleft_{in\text{-}codom\ (\leq_R)}$
using *assms* **by** (*intro ext iffI restrict-leftI restrict-rightI*
iffD1[*OF left-Galois-iff-Dep-Fun-Rel-left-Galois-if-preorder-equivalenceI'*])
(*auto intro*!: *iffD2*[*OF left-Galois-iff-Dep-Fun-Rel-left-Galois-if-preorder-equivalenceI'*])

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* $(in\text{-}field\ (\leq_{L1}))\ (\leq_{L1})$
and *reflexive-on* $(in\text{-}field\ (\leq_{R1}))\ (\leq_{R1})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$((\leq_{L2\ (r1\ x1')\ (r1\ x2')})\ {}_h\trianglelefteq\ (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (r2_{(r1\ x1')\ x2'})$
and $(([x1\ x2 :: (\leq_{L1})] \Rrightarrow_m [x3\ x4 :: (\leq_{L1}) \mid (x2 \leq_{L1} x3 \wedge x4 \leq_{L1} \eta_1\ x3)] \Rrightarrow (\geq))\ L2$
and $(([x1'\ x2' :: (\leq_{R1}) \mid \varepsilon_1\ x2' \leq_{R1} x1'] \Rrightarrow_m [x3'\ x4' :: (\leq_{R1}) \mid x2' \leq_{R1} x3'] \Rrightarrow (\leq))\ R2$
and $(([x1'\ x2' :: (\leq_{R1})] \Rrightarrow_m [x1\ x2 :: (\leq_{L1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[in\text{-}field\ (\leq_{L2\ x1\ (r1\ x2')})] \Rrightarrow (\leq_{R2\ (l1\ x1)\ x2'}))\ l2$
and $(([x1\ x2 :: (\leq_{L1})] \Rrightarrow_m [x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[in\text{-}field\ (\leq_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
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 $([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\restriction_{in\text{-}dom\ (\leq_L)}\upharpoonleft_{in\text{-}codom\ (\leq_R)}$
$= ([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))$
using *assms* **by** (*auto simp only*: *left-rel-eq-tdfr-left-rel-if-reflexive-on*
right-rel-eq-tdfr-right-rel-if-reflexive-on
intro!: *tdfr.Dep-Fun-Rel-left-Galois-restrict-left-right-eq-Dep-Fun-Rel-left-GaloisI'*)

interpretation *flip* : *transport-Dep-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2*
rewrites $flip.t1.unit \equiv \varepsilon_1$ **by** (*simp only*: *t1.flip-unit-eq-counit*)

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 $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$((\leq_{L2\ (r1\ x1')\ (r1\ x2')})\ {}_h\trianglelefteq\ (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (r2_{(r1\ x1')\ x2'})$
and $(([x1\ x2 :: (\geq_{L1})] \Rrightarrow_m [x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3] \Rrightarrow (\leq))\ L2$
and $(([x1'\ x2' :: (\geq_{R1})] \Rrightarrow_m [x3'\ x4' :: (\leq_{R1}) \mid x1' \leq_{R1} x3'] \Rrightarrow (\leq))\ R2$
and $(([x1'\ x2' :: (\leq_{R1})] \Rrightarrow_m [x1\ x2 :: (\leq_{L1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[in\text{-}field\ (\leq_{L2\ x1\ (r1\ x2')})] \Rrightarrow (\leq_{R2\ (l1\ x1)\ x2'}))\ l2$
and $(([x1\ x2 :: (\leq_{L1})] \Rrightarrow_m [x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[in\text{-}field\ (\leq_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and *PERS*: $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow partial\text{-}equivalence\text{-}rel\ (\leq_{L2\ x1\ x2})$
$\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow partial\text{-}equivalence\text{-}rel\ (\leq_{R2\ x1'\ x2'})$

shows $([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\restriction_{in\text{-}dom\ (\le_L)}\upharpoonright_{in\text{-}codom\ (\le_R)}$
$= ([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))$
using *assms* **by** (*intro*
Dep-Fun-Rel-left-Galois-restrict-left-right-eq-Dep-Fun-Rel-left-Galois-if-reflexive-onI
tdfr.reflexive-on-in-field-left-if-equivalencesI
flip.reflexive-on-in-field-left-if-equivalencesI
tdfr.galois-equivalence-if-mono-if-galois-equivalence-mono-assms-leftI
flip.galois-equivalence-if-mono-if-galois-equivalence-mono-assms-leftI)
(*auto dest*!: *PERS*)

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 $((\le_{L1})\ {}_h\trianglelefteq\ (\le_{R1}))\ l1\ r1$
shows $([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow (S\ x\ x')\restriction_{in\text{-}dom\ (\le_{L2\ x\ (r1\ x')})}\upharpoonright_{in\text{-}codom\ (\le_{R2\ (l1\ x)\ x'})})$
$\restriction_{in\text{-}dom\ (\le_L)}\upharpoonright_{in\text{-}codom\ (\le_R)} =$
$([x\ x' :: ({}_{L1}\lessapprox)] \Rrightarrow S\ x\ x')\restriction_{in\text{-}dom\ (\le_L)}\upharpoonright_{in\text{-}codom\ (\le_R)}$
(**is** $?lhs\restriction_{?DL}\upharpoonright_{?CR} = ?rhs\restriction_{?DL}\upharpoonright_{?CR}$)
proof (*intro ext*)
fix *f g*
have $?lhs\restriction_{?DL}\upharpoonright_{?CR}\ f\ g \longleftrightarrow ?lhs\ f\ g \wedge ?DL\ f \wedge ?CR\ g$ **by** *blast*
also have ... $\longleftrightarrow ?lhs\restriction_{in\text{-}dom\ tdfr.L}\upharpoonright_{in\text{-}codom\ tdfr.R}\ f\ g \wedge ?DL\ f \wedge ?CR\ g$
unfolding *left-rel-eq-tdfr-left-Refl-Rel right-rel-eq-tdfr-right-Refl-Rel*
by *blast*
also with *assms* **have** ... $\longleftrightarrow ?rhs\restriction_{in\text{-}dom\ tdfr.L}\upharpoonright_{in\text{-}codom\ tdfr.R}\ f\ g \wedge ?DL\ f \wedge$
$?CR\ g$
by (*simp only*:
tdfr.Dep-Fun-Rel-left-Galois-restrict-left-right-restrict-left-right-eq)
also have ... $\longleftrightarrow ?rhs\restriction_{?DL}\upharpoonright_{?CR}\ f\ g$
unfolding *left-rel-eq-tdfr-left-Refl-Rel right-rel-eq-tdfr-right-Refl-Rel*
by *blast*
finally show $?lhs\restriction_{?DL}\upharpoonright_{?CR}\ f\ g \longleftrightarrow ?rhs\restriction_{?DL}\upharpoonright_{?CR}\ f\ g$.
qed

end

Monotone Function Relator **context** *transport-Mono-Fun-Rel*
begin

corollary *Fun-Rel-left-Galois-if-left-GaloisI*:
assumes $((\le_{L1})\ {}_h\trianglelefteq\ (\le_{R1}))\ l1\ r1$
and *reflexive-on* $(in\text{-}dom\ (\le_{L1}))\ (\le_{L1})$
and $((\le_{R2}) \Rrightarrow_m (\le_{L2}))\ (r2)$
and *transitive* $(\le_{L2})$
and $f\ {}_L\lessapprox\ g$
shows $(({}_{L1}\lessapprox) \Rrightarrow ({}_{L2}\lessapprox))\ f\ g$
using *assms* **by** (*intro tpdfr.Dep-Fun-Rel-left-Galois-if-left-GaloisI*) *simp-all*

interpretation *flip* : *transport-Mono-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2* **.**

lemma *left-Galois-if-Fun-Rel-left-GaloisI*:
assumes $((\le_{L1}) \Rrightarrow_m (\le_{R1}))$ *l1*
and $((\le_{L1}) \trianglelefteq_h (\le_{R1}))$ *l1 r1*
and $((\le_{R2}) \Rrightarrow_m (\le_{L2}))$ *r2*
and *in-dom* $(\le_L)$ *f*
and *in-codom* $(\le_R)$ *g*
and $((_{L1}\lessapprox) \Rrightarrow (_{L2}\lessapprox))$ *f g*
shows *f* $_{L}\lessapprox$ *g*
using *assms*
by (*intro tpdfr.left-Galois-if-Dep-Fun-Rel-left-GaloisI flip.tfr.mono-wrt-rel-leftI*)
simp-all

corollary *left-Galois-iff-Fun-Rel-left-GaloisI*:
assumes $((\le_{L1}) \Rrightarrow_m (\le_{R1}))$ *l1*
and $((\le_{L1}) \trianglelefteq (\le_{R1}))$ *l1 r1*
and *reflexive-on* (*in-dom* $(\le_{L1})$) $(\le_{L1})$
and $((\le_{R2}) \Rrightarrow_m (\le_{L2}))$ (*r2*)
and *transitive* $(\le_{L2})$
and *in-dom* $(\le_L)$ *f*
and *in-codom* $(\le_R)$ *g*
shows *f* $_{L}\lessapprox$ *g* $\longleftrightarrow$ $((_{L1}\lessapprox) \Rrightarrow (_{L2}\lessapprox))$ *f g*
using *assms* **by** (*intro iffI Fun-Rel-left-Galois-if-left-GaloisI*)
(*auto intro*!: *left-Galois-if-Fun-Rel-left-GaloisI*)

theorem *left-Galois-eq-Fun-Rel-left-Galois-restrictI*:
assumes $((\le_{L1}) \Rrightarrow_m (\le_{R1}))$ *l1*
and $((\le_{L1}) \trianglelefteq (\le_{R1}))$ *l1 r1*
and *reflexive-on* (*in-dom* $(\le_{L1})$) $(\le_{L1})$
and $((\le_{R2}) \Rrightarrow_m (\le_{L2}))$ *r2*
and *transitive* $(\le_{L2})$
shows $(_{L}\lessapprox) = ((_{L1}\lessapprox) \Rrightarrow (_{L2}\lessapprox))\restriction_{in\text{-}dom\ (\le_L)}\upharpoonleft_{in\text{-}codom\ (\le_R)}$
using *assms* **by** (*intro ext iffI restrict-leftI restrict-rightI*
iffD1[*OF left-Galois-iff-Fun-Rel-left-GaloisI*])
(*auto elim*!: *tpdfr.left-GaloisE intro*!: *iffD2*[*OF left-Galois-iff-Fun-Rel-left-GaloisI*])

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 $((\le_{L1}) \Rrightarrow_m (\le_{R1}))$ *l1* **and** $((\le_{R1}) \Rrightarrow_m (\le_{L1}))$ *r1*
and $((\le_{L1}) \trianglelefteq_h (\le_{R1}))$ *l1 r1*
and $((\le_{L2})\ {_h}\trianglelefteq (\le_{R2}))$ *l2 r2*
shows $((_{L1}\lessapprox) \Rrightarrow (_{L2}\lessapprox))\restriction_{in\text{-}dom\ (\le_L)}\upharpoonleft_{in\text{-}codom\ (\le_R)} = ((_{L1}\lessapprox) \Rrightarrow (_{L2}\lessapprox))$
using *assms* **by** (*auto simp only*: *tpdfr.left-rel-eq-tdfr-left-rel-if-reflexive-on*
tpdfr.right-rel-eq-tdfr-right-rel-if-reflexive-on
intro!: *tfr.Fun-Rel-left-Galois-restrict-left-right-eq-Fun-Rel-left-GaloisI*)

lemma *Fun-Rel-left-Galois-restrict-left-right-eq-Fun-Rel-left-GaloisI*:

assumes $((\le_{L1}) \Rrightarrow_m (\le_{R1}))$ *l1* **and** $((\le_{R1}) \Rrightarrow_m (\le_{L1}))$ *r1*
and $((\le_{L1}) \trianglelefteq_h (\le_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$
and *reflexive-on* (*in-field* $(\le_{R1})$) $(\le_{R1})$
and $((\le_{L2})\ {}_h\trianglelefteq\ (\le_{R2}))$ *l2 r2*
and *partial-equivalence-rel* $(\le_{L2})$
and *partial-equivalence-rel* $(\le_{R2})$
shows $((_{L1}\lessapprox) \Rrightarrow (_{L2}\lessapprox))\restriction_{in\text{-}dom\ (\le_L)}\upharpoonright_{in\text{-}codom\ (\le_R)} = ((_{L1}\lessapprox) \Rrightarrow (_{L2}\lessapprox))$
using *assms* **by** (*intro*
Fun-Rel-left-Galois-restrict-left-right-eq-Fun-Rel-left-Galois-if-reflexive-onI
tfr.reflexive-on-in-field-leftI
flip.tfr.reflexive-on-in-field-leftI)
auto

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 $((\le_{L1})\ {}_h\trianglelefteq\ (\le_{R1}))$ *l1 r1*
shows $((_{L1}\lessapprox) \Rrightarrow S\restriction_{in\text{-}dom\ (\le_{L2})}\upharpoonright_{in\text{-}codom\ (\le_{R2})})\restriction_{in\text{-}dom\ (\le_L)}\upharpoonright_{in\text{-}codom\ (\le_R)}$
$=$
$((_{L1}\lessapprox) \Rrightarrow S)\restriction_{in\text{-}dom\ (\le_L)}\upharpoonright_{in\text{-}codom\ (\le_R)}$
using *assms*
by (*intro tpdfr.Dep-Fun-Rel-left-Galois-restrict-left-right-restrict-left-right-eq*)
simp-all

end

end

2.8.9 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* $(\le_{L1})$) $(\le_{L1})$ η_1
and *refl-L1*: *reflexive-on* (*in-codom* $(\le_{L1})$) $(\le_{L1})$
and *trans-L1*: *transitive* $(\le_{L1})$
and *mono-l2*: $\bigwedge x.\ x \le_{L1} x \Longrightarrow ((\le_{L2\ x\ x}) \Rrightarrow_m (\le_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})$
and *mono-r2*: $\bigwedge x.\ x \le_{L1} x \Longrightarrow ((\le_{R2\ (l1\ x)\ (l1\ x)}) \Rrightarrow_m (\le_{L2\ x\ (\eta_1\ x)}))\ (r2_{x\ (l1\ x)})$
and *inflationary-unit2*: $\bigwedge x.\ x \le_{L1} x \Longrightarrow$

inflationary-on (*in-codom* $(\le_{L2\ x\ x})$) $(\le_{L2\ x\ x})$ $(\eta_{2\ x\ (l1\ x)})$
and *L2-le1*: $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x2\ x2}) \le (\le_{L2\ x1\ x2})$
and *L2-unit-le2*: $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ (\eta_1\ x2)}) \le (\le_{L2\ x1\ x2})$
and *ge-R2-l2-le2*: $\bigwedge x\ y.\ x \le_{L1} x \Longrightarrow$ *in-codom* $(\le_{L2\ x\ (\eta_1\ x)})\ y \Longrightarrow$
$(\ge_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ x}\ y) \le (\ge_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ (\eta_1\ x)}\ y)$
and *trans-L2*: $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow$ *transitive* $(\le_{L2\ x1\ x2})$
and $f \le_L f$
shows $f \le_L \eta\ f$
proof (*intro left-relI*)
fix *x1 x2* **assume** [*iff*]: $x1 \le_{L1} x2$
moreover with *inflationary-unit1* **have** $x2 \le_{L1} \eta_1\ x2$ **by** *blast*
ultimately have $x1 \le_{L1} \eta_1\ x2$ **using** *trans-L1* **by** *blast*
with ‹$f \le_L f$› **have** $f\ x1 \le_{L2\ x1\ (\eta_1\ x2)} f\ (\eta_1\ x2)$ **by** *blast*
with *L2-unit-le2* **have** $f\ x1 \le_{L2\ x1\ x2} f\ (\eta_1\ x2)$ **by** *blast*
moreover have ... $\le_{L2\ x1\ x2} \eta\ f\ x2$
proof −
from *refl-L1* ‹$x2 \le_{L1} \eta_1\ x2$› **have** $\eta_1\ x2 \le_{L1} \eta_1\ x2$ **by** *blast*
with ‹$f \le_L f$› **have** $f\ (\eta_1\ x2) \le_{L2\ (\eta_1\ x2)\ (\eta_1\ x2)} f\ (\eta_1\ x2)$ **by** *blast*
with *L2-le1* **have** $f\ (\eta_1\ x2) \le_{L2\ x2\ (\eta_1\ x2)} f\ (\eta_1\ x2)$
using ‹$x2 \le_{L1} \eta_1\ x2$› **by** *blast*
moreover from *refl-L1* ‹$x1 \le_{L1} x2$› **have** [*iff*]: $x2 \le_{L1} x2$ **by** *blast*
ultimately have $f\ (\eta_1\ x2) \le_{L2\ x2\ x2} f\ (\eta_1\ x2)$ **using** *L2-unit-le2* **by** *blast*
with *inflationary-unit2* **have** $f\ (\eta_1\ x2) \le_{L2\ x2\ x2} \eta_{2\ x2\ (l1\ x2)}\ (f\ (\eta_1\ x2))$ **by** *blast*
moreover have ... $\le_{L2\ x2\ x2} \eta\ f\ x2$
proof −
from ‹$f\ (\eta_1\ x2) \le_{L2\ x2\ x2} f\ (\eta_1\ x2)$› *mono-l2*
have $l2_{(l1\ x2)\ x2}\ (f\ (\eta_1\ x2)) \le_{R2\ (l1\ x2)\ (l1\ x2)} l2_{(l1\ x2)\ x2}\ (f\ (\eta_1\ x2))$
by *blast*
with *ge-R2-l2-le2*
have $l2_{(l1\ x2)\ x2}\ (f\ (\eta_1\ x2)) \le_{R2\ (l1\ x2)\ (l1\ x2)} l2_{(l1\ x2)\ (\eta_1\ x2)}\ (f\ (\eta_1\ x2))$
using ‹$f\ (\eta_1\ x2) \le_{L2\ x2\ (\eta_1\ x2)} f\ (\eta_1\ x2)$› **by** *blast*
with *mono-r2* **have** $\eta_{2\ x2\ (l1\ x2)}\ (f\ (\eta_1\ x2)) \le_{L2\ x2\ (\eta_1\ x2)} \eta\ f\ x2$
by *auto*
with *L2-unit-le2* **show** *?thesis* **by** *blast*
qed
ultimately have $f\ (\eta_1\ x2) \le_{L2\ x2\ x2} \eta\ f\ x2$ **using** *trans-L2* **by** *blast*
with *L2-le1* **show** *?thesis* **by** *blast*
qed
ultimately show $f\ x1 \le_{L2\ x1\ x2} \eta\ f\ x2$ **using** *trans-L2* **by** *blast*
qed

Deflationary **interpretation** *flip-inv* :
transport-Dep-Fun-Rel $(\ge_{R1})$ $(\ge_{L1})$ *r1 l1 flip2 R2 flip2 L2 r2 l2*
rewrites *flip-inv.L* $\equiv (\ge_R)$ **and** *flip-inv.R* $\equiv (\ge_L)$
and *flip-inv.unit* $\equiv \varepsilon$
and *flip-inv.t1.unit* $\equiv \varepsilon_1$

and $\bigwedge x\ y.\ \mathit{flip\text{-}inv.t2\text{-}unit}\ x\ y \equiv \varepsilon_{2\ y\ x}$
and $\bigwedge R\ x\ y.\ (\mathit{flip2}\ R\ x\ y)^{-1} \equiv R\ y\ x$
and $\bigwedge R.\ \mathit{in\text{-}codom}\ R^{-1} \equiv \mathit{in\text{-}dom}\ R$
and $\bigwedge R\ x1\ x2.\ \mathit{in\text{-}codom}\ (\mathit{flip2}\ R\ x1\ x2) \equiv \mathit{in\text{-}dom}\ (R\ x2\ x1)$
and $\bigwedge x1\ x2\ x1'\ x2'.\ (\mathit{flip2}\ R2\ x1'\ x2' \Rrightarrow_m \mathit{flip2}\ L2\ x1\ x2) \equiv ((\le_{R2\ x2'\ x1'}) \Rrightarrow_m (\le_{L2\ x2\ x1}))$
and $\bigwedge x1\ x2\ x1'\ x2'.\ (\mathit{flip2}\ L2\ x1\ x2 \Rrightarrow_m \mathit{flip2}\ R2\ x1'\ x2') \equiv ((\le_{L2\ x2\ x1}) \Rrightarrow_m (\le_{R2\ x2'\ x1'}))$
and $\bigwedge P.\ \mathit{inflationary\text{-}on}\ P\ (\ge_{R1}) \equiv \mathit{deflationary\text{-}on}\ P\ (\le_{R1})$
and $\bigwedge P\ x.\ \mathit{inflationary\text{-}on}\ P\ (\mathit{flip2}\ R2\ x\ x) \equiv \mathit{deflationary\text{-}on}\ P\ (\le_{R2\ x\ x})$
and $\bigwedge x1\ x2\ x3\ x4.\ \mathit{flip2}\ R2\ x1\ x2 \le \mathit{flip2}\ R2\ x3\ x4 \equiv (\le_{R2\ x2\ x1}) \le (\le_{R2\ x4\ x3})$
and $\bigwedge(R :: {'z} \Rightarrow -)\ (P :: {'z} \Rightarrow \mathit{bool}).\ \mathit{reflexive\text{-}on}\ P\ R^{-1} \equiv \mathit{reflexive\text{-}on}\ P\ R$
and $\bigwedge R.\ \mathit{transitive}\ R^{-1} \equiv \mathit{transitive}\ R$
and $\bigwedge x1'\ x2'.\ \mathit{transitive}\ (\mathit{flip2}\ R2\ x1'\ x2') \equiv \mathit{transitive}\ (\le_{R2\ x2'\ x1'})$
by (*simp-all add*: *flip-inv-left-eq-ge-right flip-inv-right-eq-ge-left flip-unit-eq-counit t1.flip-unit-eq-counit t2.flip-unit-eq-counit galois-prop.rel-inv-half-galois-prop-right-eq-half-galois-prop-left-rel-inv*)

lemma *counit-rel-self-if-rel-selfI*:
assumes $\mathit{deflationary\text{-}on}\ (\mathit{in\text{-}dom}\ (\le_{R1}))\ (\le_{R1})\ \varepsilon_1$
and $\mathit{reflexive\text{-}on}\ (\mathit{in\text{-}dom}\ (\le_{R1}))\ (\le_{R1})$
and $\mathit{transitive}\ (\le_{R1})$
and $\bigwedge x'.\ x' \le_{R1} x' \Longrightarrow ((\le_{L2\ (r1\ x')\ (r1\ x')}) \Rrightarrow_m (\le_{R2\ (\varepsilon_1\ x')\ x'}))\ (l2_{\ x'\ (r1\ x')})$
and $\bigwedge x'\ x'.\ x' \le_{R1} x' \Longrightarrow ((\le_{R2\ x'\ x'}) \Rrightarrow_m (\le_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x'.\ x' \le_{R1} x' \Longrightarrow \mathit{deflationary\text{-}on}\ (\mathit{in\text{-}dom}\ (\le_{R2\ x'\ x'}))\ (\le_{R2\ x'\ x'})\ (\varepsilon_{2\ (r1\ x')\ x'})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ (\varepsilon_1\ x1')\ x2'}) \le (\le_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ x1'\ x1'}) \le (\le_{R2\ x1'\ x2'})$
and $\bigwedge x'\ y'.\ x' \le_{R1} x' \Longrightarrow \mathit{in\text{-}dom}\ (\le_{R2\ (\varepsilon_1\ x')\ x'})\ y' \Longrightarrow$
$(\le_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ x'}\ y') \le (\le_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')}\ y')$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow \mathit{transitive}\ (\le_{R2\ x1'\ x2'})$
and $g \le_R g$
shows $\varepsilon\ g \le_R g$
using *assms* **by** (*intro flip-inv.rel-unit-self-if-rel-selfI*[*simplified rel-inv-iff-rel*])

Relational Equivalence **lemma** *bi-related-unit-self-if-rel-self-aux*:
assumes *rel-equiv-unit1*: $\mathit{rel\text{-}equivalence\text{-}on}\ (\mathit{in\text{-}field}\ (\le_{L1}))\ (\le_{L1})\ \eta_1$
and *mono-r2*: $\bigwedge x.\ x \le_{L1} x \Longrightarrow ((\le_{R2\ (l1\ x)\ (l1\ x)}) \Rrightarrow_m (\le_{L2\ x\ x}))\ (r2_{x\ (l1\ x)})$
and *rel-equiv-unit2*: $\bigwedge x.\ x \le_{L1} x \Longrightarrow$
$\mathit{rel\text{-}equivalence\text{-}on}\ (\mathit{in\text{-}field}\ (\le_{L2\ x\ x}))\ (\le_{L2\ x\ x})\ (\eta_{2\ x\ (l1\ x)})$
and *L2-le1*: $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x2\ x2}) \le (\le_{L2\ x1\ x2})$
and *L2-le2*: $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ x1}) \le (\le_{L2\ x1\ x2})$
and [*iff*]: $x \le_{L1} x$
shows $((\le_{R2\ (l1\ x)\ (l1\ x)}) \Rrightarrow_m (\le_{L2\ x\ (\eta_1\ x)}))\ (r2_{x\ (l1\ x)})$
and $((\le_{R2\ (l1\ x)\ (l1\ x)}) \Rrightarrow_m (\le_{L2\ (\eta_1\ x)\ x}))\ (r2_{x\ (l1\ x)})$
and $\mathit{deflationary\text{-}on}\ (\mathit{in\text{-}dom}\ (\le_{L2\ x\ x}))\ (\le_{L2\ x\ x})\ \eta_{2\ x\ (l1\ x)}$
and $\mathit{inflationary\text{-}on}\ (\mathit{in\text{-}codom}\ (\le_{L2\ x\ x}))\ (\le_{L2\ x\ x})\ \eta_{2\ x\ (l1\ x)}$

proof –
  from *rel-equiv-unit1* **have** $x \equiv_{L1} \eta_1\ x$ **by** *blast*
  with *mono-r2* **show** $((\le_{R2\ (l1\ x)\ (l1\ x)}) \Rrightarrow_m (\le_{L2\ x\ (\eta_1\ x)}))\ (r2_{x\ (l1\ x)})$
    and $((\le_{R2\ (l1\ x)\ (l1\ x)}) \Rrightarrow_m (\le_{L2\ (\eta_1\ x)\ x}))\ (r2_{x\ (l1\ x)})$
    using *L2-le1 L2-le2* **by** *blast+*
qed (*insert rel-equiv-unit2*, *blast+*)

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-counit* $x\ y \equiv \eta_{2\ y\ x}$
  by (*simp-all add*: *order-functors.flip-counit-eq-unit*)

lemma *bi-related-unit-self-if-rel-selfI*:
  assumes *rel-equiv-unit1*: *rel-equivalence-on* (*in-field* $(\le_{L1})$) $(\le_{L1})\ \eta_1$
  and *trans-L1*: *transitive* $(\le_{L1})$
  and $\bigwedge x.\ x \le_{L1} x \Longrightarrow ((\le_{L2\ x\ x}) \Rrightarrow_m (\le_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})$
  and $\bigwedge x.\ x \le_{L1} x \Longrightarrow ((\le_{R2\ (l1\ x)\ (l1\ x)}) \Rrightarrow_m (\le_{L2\ x\ x}))\ (r2_{x\ (l1\ x)})$
  and $\bigwedge x.\ x \le_{L1} x \Longrightarrow$
    rel-equivalence-on (*in-field* $(\le_{L2\ x\ x})$) $(\le_{L2\ x\ x})\ (\eta_{2\ x\ (l1\ x)})$
  and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x2\ x2}) \le (\le_{L2\ x1\ x2})$
  and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ (\eta_1\ x1)\ x2}) \le (\le_{L2\ x1\ x2})$
  and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ x1}) \le (\le_{L2\ x1\ x2})$
  and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ (\eta_1\ x2)}) \le (\le_{L2\ x1\ x2})$
  and $\bigwedge x\ y.\ x \le_{L1} x \Longrightarrow$ *in-dom* $(\le_{L2\ (\eta_1\ x)\ x})\ y \Longrightarrow$
    $(\le_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ x}\ y) \le (\le_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ (\eta_1\ x)}\ y)$
  and $\bigwedge x\ y.\ x \le_{L1} x \Longrightarrow$ *in-codom* $(\le_{L2\ x\ (\eta_1\ x)})\ y \Longrightarrow$
    $(\ge_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ x}\ y) \le (\ge_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ (\eta_1\ x)}\ y)$
  and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow$ *transitive* $(\le_{L2\ x1\ x2})$
  and $f \le_L f$
  shows $f \equiv_L \eta\ f$
proof –
  from *rel-equiv-unit1 trans-L1* **have** *reflexive-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$
    by (*intro reflexive-on-in-field-if-transitive-if-rel-equivalence-on*)
  with *assms* **show** *?thesis*
    by (*intro bi-relatedI rel-unit-self-if-rel-selfI*
      flip.counit-rel-self-if-rel-selfI
      bi-related-unit-self-if-rel-self-aux)
    (*auto intro*: *inflationary-on-if-le-pred-if-inflationary-on*
      deflationary-on-if-le-pred-if-deflationary-on
      reflexive-on-if-le-pred-if-reflexive-on
      in-field-if-in-dom in-field-if-in-codom)
qed

Lemmas for Monotone Function Relator **lemma** *order-equivalence-if-order-equivalence-mono-assm*
  assumes *order-equiv1*: $((\le_{L1}) \equiv_o (\le_{R1}))\ l1\ r1$
  and *refl-R1*: *reflexive-on* (*in-field* $(\le_{R1})$) $(\le_{R1})$
  and *R2-counit-le1*: $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ (\varepsilon_1\ x1')\ x2'}) \le (\le_{R2\ x1'\ x2'})$

and *mono-l2*: $([x1' \; x2' :: (\le_{R1})] \Rrightarrow_m [x1 \; x2 :: (\le_{L1}) \mid x2 \; {}_{L1}\lessapprox \; x1'] \Rrightarrow$
$[in\text{-}field \; (\le_{L2 \; x1 \; (r1 \; x2')})] \Rrightarrow (\le_{R2 \; (l1 \; x1) \; x2'})) \; l2$
and [*iff*]: $x1' \le_{R1} x2'$
shows $([in\text{-}dom \; (\le_{L2 \; (r1 \; x1') \; (r1 \; x2')})] \Rrightarrow (\le_{R2 \; x1' \; x2'})) \; (l2_{x1' \; (r1 \; x1')}) \; (l2_{x2' \; (r1 \; x1')})$
and $([in\text{-}codom \; (\le_{L2 \; (r1 \; x1') \; (r1 \; x2')})] \Rrightarrow (\le_{R2 \; x1' \; x2'})) \; (l2_{x2' \; (r1 \; x1')}) \; (l2_{x2' \; (r1 \; x2')})$
proof −
from *refl-R1* **have** $x1' \le_{R1} x1'$ $x2' \le_{R1} x2'$ **by** *auto*
moreover with *order-equiv1*
have $r1 \; x1' \le_{L1} r1 \; x2'$ $r1 \; x1' \le_{L1} r1 \; x1'$ $r1 \; x2' \le_{L1} r1 \; x2'$ **by** *auto*
ultimately have $r1 \; x1' \; {}_{L1}\lessapprox \; x1'$ $r1 \; x2' \; {}_{L1}\lessapprox \; x2'$ **by** *blast+*
note *Dep-Fun-Rel-relD*[*OF dep-mono-wrt-relD*[*OF mono-l2* ‹$x1' \le_{R1} x2'$›]
‹$r1 \; x1' \le_{L1} r1 \; x1'$›]
with ‹$r1 \; x1' \; {}_{L1}\lessapprox \; x1'$› *R2-counit-le1*
show $([in\text{-}dom \; (\le_{L2 \; (r1 \; x1') \; (r1 \; x2')})] \Rrightarrow (\le_{R2 \; x1' \; x2'})) \; (l2_{x1' \; (r1 \; x1')}) \; (l2_{x2' \; (r1 \; x1')})$
by (*intro Dep-Fun-Rel-predI*) (*auto dest*!: *in-field-if-in-dom*)
note *Dep-Fun-Rel-relD*[*OF dep-mono-wrt-relD*[*OF mono-l2* ‹$x2' \le_{R1} x2'$›]
‹$r1 \; x1' \le_{L1} r1 \; x2'$›]
with ‹$r1 \; x2' \; {}_{L1}\lessapprox \; x2'$› *R2-counit-le1*
show $([in\text{-}codom \; (\le_{L2 \; (r1 \; x1') \; (r1 \; x2')})] \Rrightarrow (\le_{R2 \; x1' \; x2'})) \; (l2_{x2' \; (r1 \; x1')})$
$(l2_{x2' \; (r1 \; x2')})$
by (*intro Dep-Fun-Rel-predI*) (*auto dest*!: *in-field-if-in-codom*)
qed

lemma *order-equivalence-if-order-equivalence-mono-assms-rightI*:
assumes *order-equiv1*: $((\le_{L1}) \equiv_o (\le_{R1})) \; l1 \; r1$
and *refl-L1*: *reflexive-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$
and *L2-unit-le2*: $\bigwedge x1 \; x2. \; x1 \le_{L1} x2 \Longrightarrow (\le_{L2 \; x1 \; (\eta_1 \; x2)}) \le (\le_{L2 \; x1 \; x2})$
and *mono-r2*: $([x1 \; x2 :: (\le_{L1})] \Rrightarrow_m [x1' \; x2' :: (\le_{R1}) \mid x2 \; {}_{L1}\lessapprox \; x1'] \Rrightarrow$
$[in\text{-}field \; (\le_{R2 \; (l1 \; x1) \; x2'})] \Rrightarrow (\le_{L2 \; x1 \; (r1 \; x2')})) \; r2$
and [*iff*]: $x1 \le_{L1} x2$
shows $([in\text{-}codom \; (\le_{R2 \; (l1 \; x1) \; (l1 \; x2)})] \Rrightarrow (\le_{L2 \; x1 \; x2})) \; (r2_{x1 \; (l1 \; x2)}) \; (r2_{x2 \; (l1 \; x2)})$
and $([in\text{-}dom \; (\le_{R2 \; (l1 \; x1) \; (l1 \; x2)})] \Rrightarrow (\le_{L2 \; x1 \; x2})) \; (r2_{x1 \; (l1 \; x1)}) \; (r2_{x1 \; (l1 \; x2)})$
proof −
from *refl-L1* **have** $x1 \le_{L1} x1$ $x2 \le_{L1} x2$ **by** *auto*
moreover with *order-equiv1*
have $l1 \; x1 \le_{R1} l1 \; x2$ $l1 \; x1 \le_{R1} l1 \; x1$ $l1 \; x2 \le_{R1} l1 \; x2$ **by** *auto*
ultimately have $x1 \; {}_{L1}\lessapprox \; l1 \; x1$ $x2 \; {}_{L1}\lessapprox \; l1 \; x2$ **using** *order-equiv1*
by (*auto intro*!: *t1.left-Galois-left-if-in-codom-if-inflationary-onI*)
note *Dep-Fun-Rel-relD*[*OF dep-mono-wrt-relD*[*OF mono-r2* ‹$x1 \le_{L1} x2$›]
‹$l1 \; x2 \le_{R1} l1 \; x2$›]
with ‹$x2 \; {}_{L1}\lessapprox \; l1 \; x2$› *L2-unit-le2*
show $([in\text{-}codom \; (\le_{R2 \; (l1 \; x1) \; (l1 \; x2)})] \Rrightarrow (\le_{L2 \; x1 \; x2})) \; (r2_{x1 \; (l1 \; x2)}) \; (r2_{x2 \; (l1 \; x2)})$
by (*intro Dep-Fun-Rel-predI*) (*auto dest*!: *in-field-if-in-codom*)
note *Dep-Fun-Rel-relD*[*OF dep-mono-wrt-relD*[*OF mono-r2* ‹$x1 \le_{L1} x1$›]
‹$l1 \; x1 \le_{R1} l1 \; x2$›]
with ‹$x1 \; {}_{L1}\lessapprox \; l1 \; x1$› *L2-unit-le2*
show $([in\text{-}dom \; (\le_{R2 \; (l1 \; x1) \; (l1 \; x2)})] \Rrightarrow (\le_{L2 \; x1 \; x2})) \; (r2_{x1 \; (l1 \; x1)}) \; (r2_{x1 \; (l1 \; x2)})$

by (*intro Dep-Fun-Rel-predI*) (*auto dest*!: *in-field-if-in-dom*)
qed

lemma *l2-unit-bi-rel-selfI*:
assumes *pre-equiv1*: $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and *mono-L2*:
$([x1\ x2 :: (\leq_{L1})] \Rrightarrow_m [x3\ x4 :: (\leq_{L1}) \mid (x2 \leq_{L1} x3 \wedge x4 \leq_{L1} \eta_1\ x3)] \Rrightarrow (\geq))$ *L2*
and *mono-R2*:
$([x1'\ x2' :: (\leq_{R1})] \Rrightarrow_m [x3'\ x4' :: (\leq_{R1}) \mid (x2' \leq_{R1} x3' \wedge x4' \leq_{R1} \varepsilon_1\ x3')] \Rrightarrow (\geq))$ *R2*
and *mono-l2*: $([x1'\ x2' :: (\leq_{R1})] \Rrightarrow_m [x1\ x2 :: (\leq_{L1}) \mid x2\ {}_{L1}\lessapprox x1'] \Rrightarrow$
$[in\text{-}field\ (\leq_{L2\ x1\ (r1\ x2')})] \Rrightarrow (\leq_{R2\ (l1\ x1)\ x2'}))$ *l2*
and $x \leq_{L1} x$
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 (*rule bi-relatedI*)
note *t1.preorder-equivalence-order-equivalenceE*[*elim*!]
from ‹$x \leq_{L1} x$› *pre-equiv1* **have** $l1\ x \leq_{R1} l1\ x$ $x \leq_{L1} \eta_1\ x$ $\eta_1\ x \leq_{L1} x$ **by** *blast*+
with *pre-equiv1* **have** $x\ {}_{L1}\lessapprox l1\ x$ $\eta_1\ x\ {}_{L1}\lessapprox l1\ x$ **by** (*auto 4 3*)
from *pre-equiv1* ‹$x \leq_{L1} \eta_1\ x$› **have** $x \leq_{L1} \eta_1\ (\eta_1\ x)$ **by** *fastforce*
moreover note ‹*in-field* $(\leq_{L2\ x\ x})$ *y*›
Dep-Fun-Rel-relD[*OF dep-mono-wrt-relD*[*OF mono-L2* ‹$\eta_1\ x \leq_{L1} x$›] ‹$\eta_1\ x \leq_{L1} x$›]
Dep-Fun-Rel-relD[*OF dep-mono-wrt-relD*[*OF mono-L2* ‹$x \leq_{L1} x$›] ‹$\eta_1\ x \leq_{L1} x$›]
ultimately have *in-field* $(\leq_{L2\ (\eta_1\ x)\ (\eta_1\ x)})$ *y* *in-field* $(\leq_{L2\ x\ (\eta_1\ x)})$ *y*
using ‹$x \leq_{L1} \eta_1\ x$› **by** *blast*+
moreover note ‹$x\ {}_{L1}\lessapprox l1\ x$›
Dep-Fun-Rel-relD[*OF dep-mono-wrt-relD*[*OF mono-l2* ‹$l1\ x \leq_{R1} l1\ x$›] ‹$\eta_1\ x \leq_{L1} x$›]
ultimately have $l2_{(l1\ x)\ (\eta_1\ x)}\ y \leq_{R2\ (\varepsilon_1\ (l1\ x))\ (l1\ x)} l2_{(l1\ x)\ x}\ y$ **by** *auto*
moreover from *pre-equiv1* ‹$l1\ x \leq_{R1} l1\ x$›
have $\varepsilon_1\ (l1\ x) \leq_{R1} l1\ x$ $l1\ x \leq_{R1} \varepsilon_1\ (l1\ x)$ **by** *fastforce*+
moreover note *Dep-Fun-Rel-relD*[*OF dep-mono-wrt-relD*
[*OF mono-R2* ‹$l1\ x \leq_{R1} \varepsilon_1\ (l1\ x)$›] ‹$l1\ x \leq_{R1} l1\ x$›]
ultimately show $l2_{(l1\ x)\ (\eta_1\ x)}\ y \leq_{R2\ (l1\ x)\ (l1\ x)} l2_{(l1\ x)\ x}\ y$ **by** *blast*
note ‹$\eta_1\ x\ {}_{L1}\lessapprox l1\ x$› ‹*in-field* $(\leq_{L2\ x\ (\eta_1\ x)})$ *y*›
Dep-Fun-Rel-relD[*OF dep-mono-wrt-relD*[*OF mono-l2* ‹$l1\ x \leq_{R1} l1\ x$›] ‹$x \leq_{L1} \eta_1\ x$›]
then show $l2_{(l1\ x)\ x}\ y \leq_{R2\ (l1\ x)\ (l1\ x)} l2_{(l1\ x)\ (\eta_1\ x)}\ y$ **by** *auto*
qed

lemma *r2-counit-bi-rel-selfI*:
assumes *pre-equiv1*: $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and *mono-L2*:
$([x1\ x2 :: (\leq_{L1})] \Rrightarrow_m [x3\ x4 :: (\leq_{L1}) \mid (x2 \leq_{L1} x3 \wedge x4 \leq_{L1} \eta_1\ x3)] \Rrightarrow (\geq))$ *L2*

and *mono-R2*:
$([x1' \ x2' :: (\le_{R1})] \Rrightarrow_m [x3' \ x4' :: (\le_{R1}) \mid (x2' \le_{R1} x3' \wedge x4' \le_{R1} \varepsilon_1 \ x3')] \Rrightarrow (\ge))$ *R2*
and *mono-r2*: $([x1 \ x2 :: (\le_{L1})] \Rrightarrow_m [x1' \ x2' :: (\le_{R1}) \mid x2 \ {}_{L1}\lessapprox \ x1'] \Rrightarrow$
$[\textit{in-field} \ (\le_{R2 \ (l1 \ x1) \ x2'})] \Rrightarrow (\le_{L2 \ x1 \ (r1 \ x2')}))$ *r2*
and $x' \le_{R1} x'$
and *in-field* $(\le_{R2 \ x' \ x'}) \ y'$
shows $r2_{(r1 \ x') \ (\varepsilon_1 \ x')} \ y' \equiv_{L2 \ (r1 \ x') \ (r1 \ x')} r2_{(r1 \ x') \ x'} \ y'$
proof (*rule bi-relatedI*)
note *t1.preorder-equivalence-order-equivalenceE*[*elim*!]
from ‹$x' \le_{R1} x'$› *pre-equiv1* **have** $r1 \ x' \le_{L1} r1 \ x' \ x' \le_{R1} \varepsilon_1 \ x' \ \varepsilon_1 \ x' \le_{R1} x'$ **by** *blast+*
with *pre-equiv1* **have** $r1 \ x' \ {}_{L1}\lessapprox \ x' \ r1 \ x' \ {}_{L1}\lessapprox \ \varepsilon_1 \ x'$ **by** *auto*
from *pre-equiv1* ‹$x' \le_{R1} \varepsilon_1 \ x'$› **have** $x' \le_{R1} \varepsilon_1 \ (\varepsilon_1 \ x')$ **by** *fastforce*
moreover note ‹*in-field* $(\le_{R2 \ x' \ x'}) \ y'$›
Dep-Fun-Rel-relD[*OF dep-mono-wrt-relD*[*OF mono-R2* ‹$\varepsilon_1 \ x' \le_{R1} x'$›] ‹$\varepsilon_1 \ x' \le_{R1} x'$›]
Dep-Fun-Rel-relD[*OF dep-mono-wrt-relD*[*OF mono-R2* ‹$\varepsilon_1 \ x' \le_{R1} x'$›] ‹$x' \le_{R1} x'$›]
ultimately have *in-field* $(\le_{R2 \ (\varepsilon_1 \ x') \ (\varepsilon_1 \ x')}) \ y'$ *in-field* $(\le_{R2 \ (\varepsilon_1 \ x') \ x'}) \ y'$
using ‹$x' \le_{R1} \varepsilon_1 \ x'$› ‹$x' \le_{R1} x'$› **by** *blast+*
moreover note ‹$r1 \ x' \ {}_{L1}\lessapprox \ \varepsilon_1 \ x'$›
Dep-Fun-Rel-relD[*OF dep-mono-wrt-relD*[*OF mono-r2* ‹$r1 \ x' \le_{L1} r1 \ x'$›] ‹$\varepsilon_1 \ x' \le_{R1} x'$›]
ultimately show $r2_{(r1 \ x') \ (\varepsilon_1 \ x')} \ y' \le_{L2 \ (r1 \ x') \ (r1 \ x')} r2_{(r1 \ x') \ x'} \ y'$ **by** *auto*
note ‹$r1 \ x' \ {}_{L1}\lessapprox \ x'$› ‹*in-field* $(\le_{R2 \ (\varepsilon_1 \ x') \ (\varepsilon_1 \ x')}) \ y'$›
Dep-Fun-Rel-relD[*OF dep-mono-wrt-relD*[*OF mono-r2* ‹$r1 \ x' \le_{L1} r1 \ x'$›] ‹$x' \le_{R1} \varepsilon_1 \ x'$›]
then have $r2_{(r1 \ x') \ x'} \ y' \le_{L2 \ (r1 \ x') \ (\eta_1 \ (r1 \ x'))} r2_{(r1 \ x') \ (\varepsilon_1 \ x')} \ y'$ **by** *auto*
moreover from *pre-equiv1* ‹$r1 \ x' \le_{L1} r1 \ x'$›
have $\eta_1 \ (r1 \ x') \le_{L1} r1 \ x' \ r1 \ x' \le_{L1} \eta_1 \ (r1 \ x')$ **by** *fastforce+*
moreover note *Dep-Fun-Rel-relD*[*OF dep-mono-wrt-relD*
[*OF mono-L2* ‹$r1 \ x' \le_{L1} r1 \ x'$›] ‹$r1 \ x' \le_{L1} \eta_1 \ (r1 \ x')$›]
ultimately show $r2_{(r1 \ x') \ x'} \ y' \le_{L2 \ (r1 \ x') \ (r1 \ x')} r2_{(r1 \ x') \ (\varepsilon_1 \ x')} \ y'$
using *pre-equiv1* **by** *blast*
qed

end

Function Relator **context** *transport-Fun-Rel*
begin

corollary *rel-unit-self-if-rel-selfI*:
assumes *inflationary-on* (*in-codom* $(\le_{L1})) \ (\le_{L1}) \ \eta_1$
and *reflexive-on* (*in-codom* $(\le_{L1})) \ (\le_{L1})$
and *transitive* $(\le_{L1})$
and $((\le_{L2}) \Rrightarrow_m (\le_{R2}))$ *l2*
and $((\le_{R2}) \Rrightarrow_m (\le_{L2}))$ *r2*

and *inflationary-on* (*in-codom* ($\leq_{L2}$)) ($\leq_{L2}$) η_2
and *transitive* ($\leq_{L2}$)
and $f \leq_L f$
shows $f \leq_L \eta\ f$
using *assms* **by** (*intro tdfr.rel-unit-self-if-rel-selfI*) *simp-all*

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 (($\leq_{L2}$) $\Rrightarrow_m$ ($\leq_{R2}$)) *l2*
and (($\leq_{R2}$) $\Rrightarrow_m$ ($\leq_{L2}$)) *r2*
and *deflationary-on* (*in-dom* ($\leq_{R2}$)) ($\leq_{R2}$) ε_2
and *transitive* ($\leq_{R2}$)
and $g \leq_R g$
shows $\varepsilon\ g \leq_R g$
using *assms* **by** (*intro tdfr.counit-rel-self-if-rel-selfI*) *simp-all*

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 (($\leq_{L2}$) $\Rrightarrow_m$ ($\leq_{R2}$)) *l2*
and (($\leq_{R2}$) $\Rrightarrow_m$ ($\leq_{L2}$)) *r2*
and *rel-equivalence-on* (*in-field* ($\leq_{L2}$)) ($\leq_{L2}$) η_2
and *transitive* ($\leq_{L2}$)
and $f \leq_L f$
shows $f \equiv_L \eta\ f$
using *assms* **by** (*intro tdfr.bi-related-unit-self-if-rel-selfI*) *simp-all*

end

Monotone Dependent Function Relator **context** *transport-Mono-Dep-Fun-Rel*
begin

Inflationary **lemma** *inflationary-on-unitI*:
assumes (*tdfr.L* $\Rrightarrow_m$ *tdfr.R*) *l* **and** (*tdfr.R* $\Rrightarrow_m$ *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 $\bigwedge x.\ x \leq_{L1} x \Longrightarrow ((\leq_{L2\ x\ x}) \Rrightarrow_m (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow ((\leq_{R2\ (l1\ x)\ (l1\ x)}) \Rrightarrow_m (\leq_{L2\ x\ (\eta_1\ x)}))\ (r2_{x\ (l1\ x)})$
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* ($\leq_L$)) ($\leq_L$) η

unfolding *left-rel-eq-tdfr-left-Refl-Rel* **using** *assms*
by (*intro inflationary-onI Refl-RelI*)
(*auto intro*: *tdfr.rel-unit-self-if-rel-selfI*[*simplified unit-eq*] *elim*!: *Refl-RelE*)

Deflationary **lemma** *deflationary-on-counitI*:
assumes $(tdfr.L \Rrightarrow_m tdfr.R)\ l$ **and** $(tdfr.R \Rrightarrow_m tdfr.L)\ r$
and *deflationary-on* $(\textit{in-dom}\ (\le_{R1}))\ (\le_{R1})\ \varepsilon_1$
and *reflexive-on* $(\textit{in-dom}\ (\le_{R1}))\ (\le_{R1})$
and *transitive* $(\le_{R1})$
and $\bigwedge x'.\ x' \le_{R1} x' \Longrightarrow ((\le_{L2\ (r1\ x')\ (r1\ x')}) \Rrightarrow_m (\le_{R2\ (\varepsilon_1\ x')\ x'}))\ (l2_{\ x'\ (r1\ x')})$
and $\bigwedge x'.\ x' \le_{R1} x' \Longrightarrow$
$((\le_{R2\ x'\ x'}) \Rrightarrow_m (\le_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x'.\ x' \le_{R1} x' \Longrightarrow$ *deflationary-on* $(\textit{in-dom}\ (\le_{R2\ x'\ x'}))\ (\le_{R2\ x'\ x'})\ (\varepsilon_{2\ (r1\ x')\ x'})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ (\varepsilon_1\ x1')\ x2'}) \le (\le_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ x1'\ x1'}) \le (\le_{R2\ x1'\ x2'})$
and $\bigwedge x'\ y'.\ x' \le_{R1} x' \Longrightarrow \textit{in-dom}\ (\le_{R2\ (\varepsilon_1\ x')\ x'})\ y' \Longrightarrow$
$(\le_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ x'}\ y') \le (\le_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')}$
$y')$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow$ *transitive* $(\le_{R2\ x1'\ x2'})$
shows *deflationary-on* $(\textit{in-field}\ (\le_R))\ (\le_R)\ \varepsilon$
unfolding *right-rel-eq-tdfr-right-Refl-Rel* **using** *assms*
by (*intro deflationary-onI Refl-RelI*)
(*auto intro*: *tdfr.counit-rel-self-if-rel-selfI*[*simplified counit-eq*]
elim!: *Refl-RelE*)

Relational Equivalence **context**
begin

interpretation *flip* : *transport-Mono-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-counit* $x\ y \equiv \eta_{2\ y\ x}$
by (*simp-all add*: *order-functors.flip-counit-eq-unit*)

lemma *rel-equivalence-on-unitI*:
assumes $(tdfr.L \Rrightarrow_m tdfr.R)\ l$ **and** $(tdfr.R \Rrightarrow_m tdfr.L)\ r$
and *rel-equiv-unit1*: *rel-equivalence-on* $(\textit{in-field}\ (\le_{L1}))\ (\le_{L1})\ \eta_1$
and *trans-L1*: *transitive* $(\le_{L1})$
and $\bigwedge x.\ x \le_{L1} x \Longrightarrow ((\le_{L2\ x\ x}) \Rrightarrow_m (\le_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})$
and $\bigwedge x.\ x \le_{L1} x \Longrightarrow ((\le_{R2\ (l1\ x)\ (l1\ x)}) \Rrightarrow_m (\le_{L2\ x\ x}))\ (r2_{x\ (l1\ x)})$
and $\bigwedge x.\ x \le_{L1} x \Longrightarrow$ *rel-equivalence-on* $(\textit{in-field}\ (\le_{L2\ x\ x}))\ (\le_{L2\ x\ x})\ (\eta_{2\ x\ (l1\ x)})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x2\ x2}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ (\eta_1\ x1)\ x2}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ x1}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ (\eta_1\ x2)}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x\ y.\ x \le_{L1} x \Longrightarrow \textit{in-dom}\ (\le_{L2\ (\eta_1\ x)\ x})\ y \Longrightarrow$
$(\le_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ x}\ y) \le (\le_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ (\eta_1\ x)}\ y)$
and $\bigwedge x\ y.\ x \le_{L1} x \Longrightarrow \textit{in-codom}\ (\le_{L2\ x\ (\eta_1\ x)})\ y \Longrightarrow$

$(\geq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ x}\ y) \leq (\geq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ (\eta_1\ x)}\ y)$
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 –
from *rel-equiv-unit1 trans-L1* **have** *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
by (*intro reflexive-on-in-field-if-transitive-if-rel-equivalence-on*)
with *assms* **show** *?thesis*
by (*intro rel-equivalence-onI inflationary-on-unitI flip.deflationary-on-counitI*)
(*auto intro*!: *tdfr.bi-related-unit-self-if-rel-self-aux*
intro: *inflationary-on-if-le-pred-if-inflationary-on*
deflationary-on-if-le-pred-if-deflationary-on
reflexive-on-if-le-pred-if-reflexive-on
in-field-if-in-dom in-field-if-in-codom
elim!: *rel-equivalence-onE*
simp only:)
qed

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* $x\ y \equiv \varepsilon_{2\ y\ x}$
by (*simp-all add*: *order-functors.flip-counit-eq-unit*)

lemma *order-equivalenceI*:
assumes $(tdfr.L \Rrightarrow_m tdfr.R)\ l$ **and** $(tdfr.R \Rrightarrow_m tdfr.L)\ r$
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 $\bigwedge x.\ x \leq_{L1} x \Longrightarrow ((\leq_{L2\ x\ x}) \Rrightarrow_m (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow ((\leq_{L2\ (r1\ x')\ (r1\ x')}) \Rrightarrow_m (\leq_{R2\ x'\ x'}))\ (l2_{x'\ (r1\ x')})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow ((\leq_{R2\ x'\ x'}) \Rrightarrow_m (\leq_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow ((\leq_{R2\ (l1\ x)\ (l1\ x)}) \Rrightarrow_m (\leq_{L2\ x\ x}))\ (r2_{x\ (l1\ x)})$
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 $\bigwedge x\ y.\ x \le_{L1} x \Longrightarrow$ *in-dom* $(\le_{L2\ (\eta_1\ x)\ x})\ y \Longrightarrow$
$(\le_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ x}\ y) \le (\le_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ (\eta_1\ x)}\ y)$
and $\bigwedge x\ y.\ x \le_{L1} x \Longrightarrow$ *in-codom* $(\le_{L2\ x\ (\eta_1\ x)})\ y \Longrightarrow$
$(\ge_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ x}\ y) \le (\ge_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ (\eta_1\ x)}\ y)$
and $\bigwedge x'\ y'.\ x' \le_{R1} x' \Longrightarrow$ *in-dom* $(\le_{R2\ (\varepsilon_1\ x')\ x'})\ y' \Longrightarrow$
$(\le_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ x'}\ y') \le (\le_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')}\ y')$
and $\bigwedge x'\ y'.\ x' \le_{R1} x' \Longrightarrow$ *in-codom* $(\le_{R2\ x'\ (\varepsilon_1\ x')})\ y' \Longrightarrow$
$(\ge_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ x'}\ y') \le (\ge_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')}\ y')$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow$ *transitive* $(\le_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \le_{R1} x2 \Longrightarrow$ *transitive* $(\le_{R2\ x1\ x2})$
shows $((\le_L) \equiv_o (\le_R))\ l\ r$
using *assms*
by (*intro order-equivalenceI rel-equivalence-on-unitI flip.rel-equivalence-on-unitI mono-wrt-rel-leftI flip.mono-wrt-rel-leftI*)
auto

lemma *order-equivalence-if-preorder-equivalenceI*:
assumes *pre-equiv1*: $((\le_{L1}) \equiv_{pre} (\le_{R1}))\ l1\ r1$
and *order-equiv2*: $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow$
$((\le_{L2\ x\ (r1\ x')}) \equiv_o (\le_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and *L2-les*: $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x2\ x2}) \le (\le_{L2\ x1\ x2})$
$\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ (\eta_1\ x1)\ x2}) \le (\le_{L2\ x1\ x2})$
$\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ x1}) \le (\le_{L2\ x1\ x2})$
$\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ (\eta_1\ x2)}) \le (\le_{L2\ x1\ x2})$
and *R2-les*: $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ x2'\ x2'}) \le (\le_{R2\ x1'\ x2'})$
$\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ (\varepsilon_1\ x1')\ x2'}) \le (\le_{R2\ x1'\ x2'})$
$\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ x1'\ x1'}) \le (\le_{R2\ x1'\ x2'})$
$\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ x1'\ (\varepsilon_1\ x2')}) \le (\le_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow$
$([$*in-dom* $(\le_{L2\ (r1\ x1')\ (r1\ x2')})] \Rrightarrow (\le_{R2\ x1'\ x2'}))\ (l2_{x1'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x1')})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow$
$([$*in-codom* $(\le_{L2\ (r1\ x1')\ (r1\ x2')})] \Rrightarrow (\le_{R2\ x1'\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x2')})$
and *l2-bi-rel*: $\bigwedge x\ y.\ x \le_{L1} x \Longrightarrow$ *in-field* $(\le_{L2\ x\ x})\ y \Longrightarrow$
$l2_{(l1\ x)\ (\eta_1\ x)}\ y \equiv_{R2\ (l1\ x)\ (l1\ x)} l2_{(l1\ x)\ x}\ y$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow$
$([$*in-codom* $(\le_{R2\ (l1\ x1)\ (l1\ x2)})] \Rrightarrow (\le_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x2)})\ (r2_{x2\ (l1\ x2)})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow$
$([$*in-dom* $(\le_{R2\ (l1\ x1)\ (l1\ x2)})] \Rrightarrow (\le_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x1)})\ (r2_{x1\ (l1\ x2)})$
and *r2-bi-rel*: $\bigwedge x'\ y'.\ x' \le_{R1} x' \Longrightarrow$ *in-field* $(\le_{R2\ x'\ x'})\ y' \Longrightarrow$
$r2_{(r1\ x')\ (\varepsilon_1\ x')}\ y' \equiv_{L2\ (r1\ x')\ (r1\ x')} r2_{(r1\ x')\ x'}\ y'$
and *trans-L2*: $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow$ *transitive* $(\le_{L2\ x1\ x2})$
and *trans-R2*: $\bigwedge x1\ x2.\ x1 \le_{R1} x2 \Longrightarrow$ *transitive* $(\le_{R2\ x1\ x2})$
shows $((\le_L) \equiv_o (\le_R))\ l\ r$
proof –

from *pre-equiv1 L2-les* **have** *L2-unit-eq1*: $(\leq_{L2\ (\eta_1\ x)\ x}) = (\leq_{L2\ x\ x})$
and *L2-unit-eq2*: $(\leq_{L2\ x\ (\eta_1\ x)}) = (\leq_{L2\ x\ x})$
if $x \leq_{L1} x$ **for** x **using** ‹$x \leq_{L1} x$›
by (*auto elim*!: *t1.preorder-equivalence-order-equivalenceE*
intro!: *tdfr.left-rel2-unit-eqs-left-rel2I bi-related-if-rel-equivalence-on*
simp del: *t1.unit-eq*)
from *pre-equiv1 R2-les* **have** *R2-counit-eq1*: $(\leq_{R2\ (\varepsilon_1\ x')\ x'}) = (\leq_{R2\ x'\ x'})$
and *R2-counit-eq2*: $(\leq_{R2\ x'\ (\varepsilon_1\ x')}) = (\leq_{R2\ x'\ x'})$ (**is** *?goal2*)
if $x' \leq_{R1} x'$ **for** x' **using** ‹$x' \leq_{R1} x'$›
by (*auto elim*!: *t1.preorder-equivalence-order-equivalenceE*
intro!: *flip.tdfr.left-rel2-unit-eqs-left-rel2I bi-related-if-rel-equivalence-on*
simp del: *t1.counit-eq*)
from *order-equiv2* **have**
mono-l2: $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \Rrightarrow_m (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})$
and *mono-r2*: $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rrightarrow_m (\leq_{L2\ x\ (r1\ x')}))$
$(r2_{x\ x'})$
by *auto*
moreover have *rel-equivalence-on* (*in-field* $(\leq_{L2\ x\ x})$) $(\leq_{L2\ x\ x})$ $(\eta_{2\ x\ (l1\ x)})$ (**is**
?goal1)
and $((\leq_{L2\ x\ x}) \Rrightarrow_m (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})$ (**is** *?goal2*)
if [*iff*]: $x \leq_{L1} x$ **for** x
proof −
from *pre-equiv1* **have** $x\ {}_{L1}\lessapprox\ l1\ x$
by (*auto intro*!: *t1.left-GaloisI*
elim!: *t1.preorder-equivalence-order-equivalenceE t1.order-equivalenceE*)
with *order-equiv2* **have** $((\leq_{L2\ x\ x}) \equiv_o (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (r2_{x\ (l1\ x)})$
by (*auto simp flip*: *L2-unit-eq2*)
then show *?goal1 ?goal2* **by** (*auto elim*: *order-functors.order-equivalenceE*)
qed
moreover have
rel-equivalence-on (*in-field* $(\leq_{R2\ x'\ x'})$) $(\leq_{R2\ x'\ x'})$ $(\varepsilon_{2\ (r1\ x')\ x'})$ (**is** *?goal1*)
and $((\leq_{R2\ x'\ x'}) \Rrightarrow_m (\leq_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ x'})$ (**is** *?goal2*)
if [*iff*]: $x' \leq_{R1} x'$ **for** x'
proof −
from *pre-equiv1* **have** $r1\ x'\ {}_{L1}\lessapprox\ x'$ **by** *blast*
with *order-equiv2* **have** $((\leq_{L2\ (r1\ x')\ (r1\ x')}) \equiv_o (\leq_{R2\ x'\ x'}))\ (l2_{x'\ (r1\ x')})$
$(r2_{(r1\ x')\ x'})$
by (*auto simp flip*: *R2-counit-eq1*)
then show *?goal1 ?goal2* **by** (*auto elim*: *order-functors.order-equivalenceE*)
qed
moreover from *mono-l2 tdfr.mono-wrt-rel-left2-if-mono-wrt-rel-left2-if-left-GaloisI*
have $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow ((\leq_{L2\ (r1\ x1')\ (r1\ x2')}) \Rrightarrow_m (\leq_{R2\ x1'\ x2'}))$
$(l2_{x2'\ (r1\ x1')})$
using *pre-equiv1 R2-les*(*2*) **by** *blast*
moreover from *pre-equiv1* **have** $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))\ l1\ r1$
by (*intro t1.half-galois-prop-right-left-right-if-transitive-if-order-equivalence*)

(*auto elim*!: *t1.preorder-equivalence-order-equivalenceE*)
moreover with *mono-r2 tdfr.mono-wrt-rel-right2-if-mono-wrt-rel-right2-if-left-GaloisI*
have $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow ((\le_{R2\ (l1\ x1)\ (l1\ x2)}) \Rrightarrow_m (\le_{L2\ x1\ (\eta_1\ x2)}))$ $(r2_{x1\ (l1\ x2)})$
using *pre-equiv1* **by** *blast*
moreover with *L2-les*
have $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow ((\le_{R2\ (l1\ x1)\ (l1\ x2)}) \Rrightarrow_m (\le_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x2)})$
by *blast*
moreover have *in-dom* $(\le_{L2\ (\eta_1\ x)\ x})\ y \Longrightarrow$
$(\le_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ x}\ y) \le (\le_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ (\eta_1\ x)}\ y)$
(**is** - $\Longrightarrow$ *?goal1*)
and *in-codom* $(\le_{L2\ x\ (\eta_1\ x)})\ y \Longrightarrow$
$(\ge_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ x}\ y) \le (\ge_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ (\eta_1\ x)}\ y)$
(**is** - $\Longrightarrow$ *?goal2*)
if [*iff*]: $x \le_{L1} x$ **for** *x y*
proof −
presume *in-dom* $(\le_{L2\ (\eta_1\ x)\ x})\ y \lor$ *in-codom* $(\le_{L2\ x\ (\eta_1\ x)})\ y$
then have *in-field* $(\le_{L2\ x\ x})\ y$ **using** *L2-unit-eq1 L2-unit-eq2* **by** *auto*
with *l2-bi-rel* **have** $l2_{(l1\ x)\ (\eta_1\ x)}\ y \equiv_{R2\ (l1\ x)\ (l1\ x)} l2_{(l1\ x)\ x}\ y$ **by** *blast*
moreover from *pre-equiv1* **have** ‹$l1\ x \le_{R1} l1\ x$› **by** *blast*
ultimately show *?goal1 ?goal2* **using** *trans-R2* **by** *blast+*
qed *auto*
moreover have *in-dom* $(\le_{R2\ (\varepsilon_1\ x')\ x'})\ y' \Longrightarrow$
$(\le_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ x'}\ y') \le (\le_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')}$ $y')$
(**is** - $\Longrightarrow$ *?goal1*)
and *in-codom* $(\le_{R2\ x'\ (\varepsilon_1\ x')})\ y' \Longrightarrow$
$(\ge_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ x'}\ y') \le (\ge_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')}$ $y')$
(**is** - $\Longrightarrow$ *?goal2*)
if [*iff*]: $x' \le_{R1} x'$ **for** $x'\ y'$
proof −
presume *in-dom* $(\le_{R2\ (\varepsilon_1\ x')\ x'})\ y' \lor$ *in-codom* $(\le_{R2\ x'\ (\varepsilon_1\ x')})\ y'$
then have *in-field* $(\le_{R2\ x'\ x'})\ y'$ **using** *R2-counit-eq1 R2-counit-eq2* **by** *auto*
with *r2-bi-rel* **have** $r2_{(r1\ x')\ (\varepsilon_1\ x')}\ y' \equiv_{L2\ (r1\ x')\ (r1\ x')} r2_{(r1\ x')\ x'}\ y'$
by *blast*
moreover from *pre-equiv1* **have** ‹$r1\ x' \le_{L1} r1\ x'$› **by** *blast*
ultimately show *?goal1 ?goal2* **using** *trans-L2* **by** *blast+*
qed *auto*
ultimately show *?thesis* **using** *assms*
by (*intro order-equivalenceI*
tdfr.mono-wrt-rel-left-if-transitiveI
tdfr.mono-wrt-rel-left2-if-mono-wrt-rel-left2-if-left-GaloisI
tdfr.mono-wrt-rel-right-if-transitiveI
tdfr.mono-wrt-rel-right2-if-mono-wrt-rel-right2-if-left-GaloisI)
(*auto elim*!: *t1.preorder-equivalence-order-equivalenceE*)
qed

lemma *order-equivalence-if-preorder-equivalenceI'*:
assumes $((\le_{L1}) \equiv_{pre} (\le_{R1}))\ l1\ r1$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\le_{L2\ x\ (r1\ x')}) \equiv_o (\le_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x2\ x2}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ (\eta_1\ x1)\ x2}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ x1}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow (\le_{L2\ x1\ (\eta_1\ x2)}) \le (\le_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ x2'\ x2'}) \le (\le_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ (\varepsilon_1\ x1')\ x2'}) \le (\le_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ x1'\ x1'}) \le (\le_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow (\le_{R2\ x1'\ (\varepsilon_1\ x2')}) \le (\le_{R2\ x1'\ x2'})$
and $([x1'\ x2' :: (\le_{R1})] \Rrightarrow_m [x1\ x2 :: (\le_{L1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[\textit{in-field}\ (\le_{L2\ x1\ (r1\ x2')})] \Rrightarrow (\le_{R2\ (l1\ x1)\ x2'}))\ l2$
and $\bigwedge x\ y.\ x \le_{L1} x \Longrightarrow \textit{in-field}\ (\le_{L2\ x\ x})\ y \Longrightarrow$
$l2_{(l1\ x)\ (\eta_1\ x)}\ y \equiv_{R2\ (l1\ x)\ (l1\ x)}\ l2_{(l1\ x)\ x}\ y$
and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[\textit{in-field}\ (\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x'\ y'.\ x' \le_{R1} x' \Longrightarrow \textit{in-field}\ (\le_{R2\ x'\ x'})\ y' \Longrightarrow$
$r2_{(r1\ x')\ (\varepsilon_1\ x')}\ y' \equiv_{L2\ (r1\ x')\ (r1\ x')}\ r2_{(r1\ x')\ x'}\ y'$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow \textit{transitive}\ (\le_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \le_{R1} x2 \Longrightarrow \textit{transitive}\ (\le_{R2\ x1\ x2})$
shows $((\le_L) \equiv_o (\le_R))\ l\ r$
using *assms* **by** (*intro order-equivalence-if-preorder-equivalenceI*
tdfr.order-equivalence-if-order-equivalence-mono-assms-leftI
tdfr.order-equivalence-if-order-equivalence-mono-assms-rightI
reflexive-on-in-field-if-transitive-if-rel-equivalence-on)
(*auto elim*!: *t1.preorder-equivalence-order-equivalenceE*)

lemma *order-equivalence-if-mono-if-preorder-equivalenceI*:
assumes $((\le_{L1}) \equiv_{pre} (\le_{R1}))\ l1\ r1$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\le_{L2\ x\ (r1\ x')}) \equiv_o (\le_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $([x1\ x2 :: (\le_{L1}) \mid \eta_1\ x2 \le_{L1} x1] \Rrightarrow_m [x3\ x4 :: (\le_{L1}) \mid x2 \le_{L1} x3] \Rrightarrow (\le))$ $L2$
and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x3\ x4 :: (\le_{L1}) \mid (x2 \le_{L1} x3 \land x4 \le_{L1} \eta_1\ x3)] \Rrightarrow (\ge))$ $L2$
and $([x1'\ x2' :: (\le_{R1}) \mid \varepsilon_1\ x2' \le_{R1} x1'] \Rrightarrow_m [x3'\ x4' :: (\le_{R1}) \mid x2' \le_{R1} x3'] \Rrightarrow (\le))$ $R2$
and $([x1'\ x2' :: (\le_{R1})] \Rrightarrow_m [x3'\ x4' :: (\le_{R1}) \mid (x2' \le_{R1} x3' \land x4' \le_{R1} \varepsilon_1\ x3')] \Rrightarrow (\ge))$ $R2$
and $([x1'\ x2' :: (\le_{R1})] \Rrightarrow_m [x1\ x2 :: (\le_{L1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[\textit{in-field}\ (\le_{L2\ x1\ (r1\ x2')})] \Rrightarrow (\le_{R2\ (l1\ x1)\ x2'}))\ l2$
and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[\textit{in-field}\ (\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow \textit{transitive}\ (\le_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \le_{R1} x2 \Longrightarrow \textit{transitive}\ (\le_{R2\ x1\ x2})$
shows $((\le_L) \equiv_o (\le_R))\ l\ r$

using *assms* **by** (*intro order-equivalence-if-preorder-equivalenceI'*
tdfr.l2-unit-bi-rel-selfI tdfr.r2-counit-bi-rel-selfI
tdfr.left-rel-right-iff-left-right-rel-if-galois-prop-le-assms-leftI
flip.tdfr.left-rel-right-iff-left-right-rel-if-galois-prop-le-assms-leftI
tdfr.left-rel-right-iff-left-right-rel-if-galois-prop-le-assms-rightI
flip.tdfr.left-rel-right-iff-left-right-rel-if-galois-prop-le-assms-rightI
t1.galois-connection-left-right-if-transitive-if-order-equivalence
flip.t1.galois-connection-left-right-if-transitive-if-order-equivalence
reflexive-on-in-field-if-transitive-if-rel-equivalence-on)
(*auto elim*!: *t1.preorder-equivalence-order-equivalenceE*)

theorem *order-equivalence-if-mono-if-preorder-equivalenceI'*:
assumes $((\le_{L1}) \equiv_{pre} (\le_{R1}))\ l1\ r1$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\le_{L2\ x\ (r1\ x')}) \equiv_{pre} (\le_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $([x1\ x2 :: (\ge_{L1})] \Rrightarrow_m [x3\ x4 :: (\le_{L1}) \mid x1 \le_{L1} x3] \Rrightarrow (\le))\ L2$
and $([x1'\ x2' :: (\ge_{R1})] \Rrightarrow_m [x3'\ x4' :: (\le_{R1}) \mid x1' \le_{R1} x3'] \Rrightarrow (\le))\ R2$
and $([x1'\ x2' :: (\le_{R1})] \Rrightarrow_m [x1\ x2 :: (\le_{L1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[in\text{-}field\ (\le_{L2\ x1\ (r1\ x2')})] \Rrightarrow (\le_{R2\ (l1\ x1)\ x2'}))\ l2$
and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[in\text{-}field\ (\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')}))\ r2$
shows $((\le_L) \equiv_o (\le_R))\ l\ r$
using *assms* **by** (*intro order-equivalence-if-mono-if-preorder-equivalenceI*
tdfr.galois-equivalence-if-mono-if-galois-equivalence-mono-assms-leftI
flip.tdfr.galois-equivalence-if-mono-if-galois-equivalence-mono-assms-leftI
tdfr.transitive-left2-if-preorder-equivalenceI
tdfr.transitive-right2-if-preorder-equivalenceI
t1.preorder-on-in-field-left-if-transitive-if-order-equivalence
flip.t1.preorder-on-in-field-left-if-transitive-if-order-equivalence
t1.galois-equivalence-left-right-if-transitive-if-order-equivalence
flip.t1.galois-equivalence-left-right-if-transitive-if-order-equivalence)
(*auto elim*!: *t1.preorder-equivalence-order-equivalenceE*
t2.preorder-equivalence-order-equivalenceE)

end

Monotone Function Relator **context** *transport-Mono-Fun-Rel*
begin

interpretation *flip* : *transport-Mono-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2* **.**

lemma *inflationary-on-unitI*:
assumes $((\le_{L1}) \Rrightarrow_m (\le_{R1}))\ l1$
and $((\le_{R1}) \Rrightarrow_m (\le_{L1}))\ r1$
and $inflationary\text{-}on\ (in\text{-}codom\ (\le_{L1}))\ (\le_{L1})\ \eta_1$
and $reflexive\text{-}on\ (in\text{-}codom\ (\le_{L1}))\ (\le_{L1})$
and $transitive\ (\le_{L1})$
and $((\le_{L2}) \Rrightarrow_m (\le_{R2}))\ l2$
and $((\le_{R2}) \Rrightarrow_m (\le_{L2}))\ r2$
and $inflationary\text{-}on\ (in\text{-}codom\ (\le_{L2}))\ (\le_{L2})\ \eta_2$

and *transitive* $(\leq_{L2})$
shows *inflationary-on* $($*in-field* $(\leq_L))$ $(\leq_L)$ η
using *assms* **by** (*intro tpdfr.inflationary-on-unitI*
tfr.mono-wrt-rel-leftI flip.tfr.mono-wrt-rel-leftI)
simp-all

lemma *deflationary-on-counitI*:
assumes $((\leq_{L1}) \Rrightarrow_m (\leq_{R1}))$ *l1*
and $((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))$ *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 $((\leq_{L2}) \Rrightarrow_m (\leq_{R2}))$ *l2*
and $((\leq_{R2}) \Rrightarrow_m (\leq_{L2}))$ *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
using *assms* **by** (*intro tpdfr.deflationary-on-counitI*
tfr.mono-wrt-rel-leftI flip.tfr.mono-wrt-rel-leftI)
simp-all

lemma *rel-equivalence-on-unitI*:
assumes $((\leq_{L1}) \Rrightarrow_m (\leq_{R1}))$ *l1*
and $((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))$ *r1*
and *rel-equivalence-on* $($*in-field* $(\leq_{L1}))$ $(\leq_{L1})$ η_1
and *transitive* $(\leq_{L1})$
and $((\leq_{L2}) \Rrightarrow_m (\leq_{R2}))$ *l2*
and $((\leq_{R2}) \Rrightarrow_m (\leq_{L2}))$ *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
using *assms* **by** (*intro tpdfr.rel-equivalence-on-unitI*
tfr.mono-wrt-rel-leftI flip.tfr.mono-wrt-rel-leftI)
simp-all

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*
using *assms* **by** (*intro tpdfr.order-equivalenceI*
tfr.mono-wrt-rel-leftI flip.tfr.mono-wrt-rel-leftI)
(*auto elim*!: *tdfrs.t1.preorder-equivalence-order-equivalenceE*
tdfrs.t2.preorder-equivalence-order-equivalenceE)

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.

2.8.10 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* $\equiv \eta_1$ **and** *flip.t1.unit* $\equiv \varepsilon_1$
by (*simp-all only*: *t1.flip-counit-eq-unit t1.flip-unit-eq-counit*)

Closure of Order and Galois Concepts **theorem** *preorder-galois-connection-if-galois-connectionI*:
assumes $((\le_{L1}) \dashv (\le_{R1}))\ l1\ r1$
and *reflexive-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$
and *reflexive-on* (*in-field* $(\le_{R1})$) $(\le_{R1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\le_{L2\ x\ (r1\ x')}) \dashv (\le_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $([-\ x2 :: (\le_{L1})] \Rrightarrow_m [x3\ x4 :: (\le_{L1}) \mid (x2 \le_{L1} x3 \wedge x4 \le_{L1} \eta_1\ x3)] \Rrightarrow (\ge))$
$L2$
and $([x1'\ x2' :: (\le_{R1}) \mid \varepsilon_1\ x2' \le_{R1} x1'] \Rrightarrow_m [x3'\ - :: (\le_{R1}) \mid x2' \le_{R1} x3'] \Rrightarrow$
$(\le))\ R2$
and $([x1'\ x2' :: (\le_{R1})] \Rrightarrow_m [x1\ x2 :: (\le_{L1}) \mid x2\ {}_{L1}\lessapprox x1'] \Rrightarrow$
$[\textit{in-field}\ (\le_{L2\ x1\ (r1\ x2')})] \Rrightarrow (\le_{R2\ (l1\ x1)\ x2'}))\ l2$
and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x2\ {}_{L1}\lessapprox x1'] \Rrightarrow$
$[\textit{in-field}\ (\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow \textit{transitive}\ (\le_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow \textit{transitive}\ (\le_{R2\ x1'\ x2'})$
shows $((\le_L) \dashv_{pre} (\le_R))\ l\ r$
using *assms* **by** (*intro preorder-galois-connectionI*
galois-connection-left-right-if-mono-if-galois-connectionI'
preorder-on-in-field-leftI flip.preorder-on-in-field-leftI
tdfr.transitive-leftI' flip.tdfr.transitive-leftI
tdfr.left-rel-right-iff-left-right-rel-if-galois-prop-le-assms-leftI
tdfr.left-rel-right-iff-left-right-rel-if-galois-prop-le-assms-rightI)
(*auto intro*: *reflexive-on-if-le-pred-if-reflexive-on*
in-field-if-in-dom in-field-if-in-codom)

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 $([x1\ \text{-} :: (\geq_{L1})] \Rrightarrow_m [x3\ \text{-} :: (\leq_{L1}) \mid x1 \leq_{L1} x3] \Rrightarrow (\leq))\ L2$
and $([x1'\ \text{-} :: (\geq_{R1})] \Rrightarrow_m [x3'\ \text{-} :: (\leq_{R1}) \mid x1' \leq_{R1} x3'] \Rrightarrow (\leq))\ R2$
and $([x1'\ x2' :: (\leq_{R1})] \Rrightarrow_m [x1\ x2 :: (\leq_{L1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[\textit{in-field}\ (\leq_{L2\ x1\ (r1\ x2')})] \Rrightarrow (\leq_{R2\ (l1\ x1)\ x2'}))\ l2$
and $([x1\ x2 :: (\leq_{L1})] \Rrightarrow_m [x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[\textit{in-field}\ (\leq_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
shows $((\leq_L) \equiv_{pre} (\leq_R))\ l\ r$
using *assms* **by** (*intro preorder-equivalence-if-galois-equivalenceI*
galois-equivalence-if-mono-if-preorder-equivalenceI'
preorder-on-in-field-leftI flip.preorder-on-in-field-leftI
tdfr.transitive-leftI' flip.tdfr.transitive-leftI
tdfr.transitive-left2-if-preorder-equivalenceI
tdfr.transitive-right2-if-preorder-equivalenceI
tdfr.left-rel-right-iff-left-right-rel-if-galois-prop-le-assms-leftI
tdfr.left-rel-right-iff-left-right-rel-if-galois-prop-le-assms-rightI
tdfr.galois-equivalence-if-mono-if-galois-equivalence-mono-assms-leftI
flip.tdfr.galois-equivalence-if-mono-if-galois-equivalence-mono-assms-leftI)
(*auto intro*: *reflexive-on-if-le-pred-if-reflexive-on*
in-field-if-in-dom in-field-if-in-codom)

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 $([x1\ \text{-} :: (\geq_{L1})] \Rrightarrow_m [x3\ \text{-} :: (\leq_{L1}) \mid x1 \leq_{L1} x3] \Rrightarrow (\leq))\ L2$
and $([x1'\ \text{-} :: (\geq_{R1})] \Rrightarrow_m [x3'\ \text{-} :: (\leq_{R1}) \mid x1' \leq_{R1} x3'] \Rrightarrow (\leq))\ R2$
and $([x1'\ x2' :: (\leq_{R1})] \Rrightarrow_m [x1\ x2 :: (\leq_{L1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[\textit{in-field}\ (\leq_{L2\ x1\ (r1\ x2')})] \Rrightarrow (\leq_{R2\ (l1\ x1)\ x2'}))\ l2$
and $([x1\ x2 :: (\leq_{L1})] \Rrightarrow_m [x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1'] \Rrightarrow$
$[\textit{in-field}\ (\leq_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
shows $((\leq_L) \equiv_{PER} (\leq_R))\ l\ r$
using *assms* **by** (*intro partial-equivalence-rel-equivalence-if-galois-equivalenceI*
galois-equivalence-if-mono-if-preorder-equivalenceI'
tdfr.transitive-left2-if-preorder-equivalenceI
tdfr.transitive-right2-if-preorder-equivalenceI
partial-equivalence-rel-leftI flip.partial-equivalence-rel-leftI
tdfr.partial-equivalence-rel-left2-if-partial-equivalence-rel-equivalenceI
tdfr.partial-equivalence-rel-right2-if-partial-equivalence-rel-equivalenceI)
auto

Simplification of Left and Right Relations See $[\![$*t1.galois-equivalence*; *preorder-on* (*in-field* $(\leq_{L1})) (\leq_{L1})$; $([x1\ x2 :: (\leq_{L1})^{-1}] \Rrightarrow_m [x3\ x4 :: (\leq_{L1})] \Rrightarrow x1 \leq_{L1} x3 \longrightarrow (\leq))\ L2$; $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *partial-equivalence-rel* $(\leq_{L2\ x1\ x2})]\!] \Longrightarrow$ *flip.R* = *flip.tdfr.R*.

Simplification of Galois relator See $[\![\mathit{t1.galois\text{-}connection};\ \mathit{reflexive\text{-}on}\ (\mathit{in\text{-}field}\ (\le_{L1}))\ (\le_{L1});\ \bigwedge x\ x'.\ \mathit{flip.t1.right\text{-}Galois}\ x\ x' \Longrightarrow (\le_{R2\ l1\ x\ x'} \Rrightarrow_m \le_{L2\ x\ r1\ x'})\ r2_{x\ x'};\ ([x1 :: \top] \Rrightarrow_m [x2\ \text{-} :: (\le_{L1})] \Rrightarrow_m x1 \le_{L1} x2 \longrightarrow (\le))\ L2;\ ([x1 :: \top] \Rrightarrow_m [x2\ x3 :: (\le_{L1})] \Rrightarrow_m (x1 \le_{L1} x2 \wedge x3 \le_{L1} \eta_1\ x2) \longrightarrow (\lambda x\ y.\ y \le x))\ L2;\ ([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1})] \Rrightarrow \mathit{flip.t1.right\text{-}Galois}\ x2\ x1' \longrightarrow ([\mathit{in\text{-}field}\ (\le_{R2\ l1\ x1\ x2'})] \Rrightarrow \le_{L2\ x1\ r1\ x2'}))\ r2;\ \bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow \mathit{transitive}\ (\le_{L2\ x1\ x2})]\!] \Longrightarrow \mathit{flip.right\text{-}Galois} = (\mathit{Dep\text{-}Fun\text{-}Rel\text{-}rel}\ \mathit{flip.t1.right\text{-}Galois}\ \mathit{t2.left\text{-}Galois})\restriction_{\mathit{in\text{-}dom}\ \mathit{flip.R}}\upharpoonleft_{\mathit{in\text{-}codom}\ \mathit{flip.L}}$

$[\![\mathit{t1.preorder\text{-}equivalence};\ \bigwedge x\ x'.\ \mathit{flip.t1.right\text{-}Galois}\ x\ x' \Longrightarrow (\le_{R2\ l1\ x\ x'} \Rrightarrow_m \le_{L2\ x\ r1\ x'})\ r2_{x\ x'};\ ([x1\ x2 :: (\le_{L1})^{-1}] \Rrightarrow_m [x3\ x4 :: (\le_{L1})] \Rrightarrow x1 \le_{L1} x3 \longrightarrow (\le))\ L2;\ ([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1})] \Rrightarrow \mathit{flip.t1.right\text{-}Galois}\ x2\ x1' \longrightarrow ([\mathit{in\text{-}field}\ (\le_{R2\ l1\ x1\ x2'})] \Rrightarrow \le_{L2\ x1\ r1\ x2'}))\ r2;\ \bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow \mathit{transitive}\ (\le_{L2\ x1\ x2})]\!] \Longrightarrow \mathit{flip.right\text{-}Galois} = (\mathit{Dep\text{-}Fun\text{-}Rel\text{-}rel}\ \mathit{flip.t1.right\text{-}Galois}\ \mathit{t2.left\text{-}Galois})\restriction_{\mathit{in\text{-}dom}\ \mathit{flip.R}}\upharpoonleft_{\mathit{in\text{-}codom}\ \mathit{flip.L}}$

$[\![\mathit{t1.preorder\text{-}equivalence};\ \bigwedge x\ x'.\ \mathit{flip.t1.right\text{-}Galois}\ x\ x' \Longrightarrow \mathit{t2.preorder\text{-}equivalence}\ x\ x';\ ([x1\ x2 :: (\le_{L1})^{-1}] \Rrightarrow_m [x3\ x4 :: (\le_{L1})] \Rrightarrow x1 \le_{L1} x3 \longrightarrow (\le))\ L2;\ ([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1})] \Rrightarrow \mathit{flip.t1.right\text{-}Galois}\ x2\ x1' \longrightarrow ([\mathit{in\text{-}field}\ (\le_{R2\ l1\ x1\ x2'})] \Rrightarrow \le_{L2\ x1\ r1\ x2'}))\ r2]\!] \Longrightarrow \mathit{flip.right\text{-}Galois} = (\mathit{Dep\text{-}Fun\text{-}Rel\text{-}rel}\ \mathit{flip.t1.right\text{-}Galois}\ \mathit{t2.left\text{-}Galois})\restriction_{\mathit{in\text{-}dom}\ \mathit{flip.R}}\upharpoonleft_{\mathit{in\text{-}codom}\ \mathit{flip.L}}$

$[\![\mathit{t1.preorder\text{-}equivalence};\ \bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow ((\le_{L2\ r1\ x1'\ r1\ x2'})\ {}_h\trianglelefteq\ (\le_{R2\ \varepsilon_1\ x1'\ x2'}))\ l2_{x2'\ r1\ x1'}\ r2_{r1\ x1'\ x2'};\ ([x1\ x2 :: (\le_{L1})^{-1}] \Rrightarrow_m [x3\ x4 :: (\le_{L1})] \Rrightarrow x1 \le_{L1} x3 \longrightarrow (\le))\ L2;\ ([x1'\ x2' :: (\le_{R1})^{-1}] \Rrightarrow_m [x3'\ x4' :: (\le_{R1})] \Rrightarrow x1' \le_{R1} x3' \longrightarrow (\le))\ R2;\ ([x1'\ x2' :: (\le_{R1})] \Rrightarrow_m [x1\ x2 :: (\le_{L1})] \Rrightarrow \mathit{flip.t1.right\text{-}Galois}\ x2\ x1' \longrightarrow ([\mathit{in\text{-}field}\ (\le_{L2\ x1\ r1\ x2'})] \Rrightarrow \le_{R2\ l1\ x1\ x2'}))\ l2;\ ([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1})] \Rrightarrow \mathit{flip.t1.right\text{-}Galois}\ x2\ x1' \longrightarrow ([\mathit{in\text{-}field}\ (\le_{R2\ l1\ x1\ x2'})] \Rrightarrow \le_{L2\ x1\ r1\ x2'}))\ r2;\ \bigwedge x1\ x2.\ x1 \le_{L1} x2 \Longrightarrow \mathit{partial\text{-}equivalence\text{-}rel}\ (\le_{L2\ x1\ x2});\ \bigwedge x1'\ x2'.\ x1' \le_{R1} x2' \Longrightarrow \mathit{partial\text{-}equivalence\text{-}rel}\ (\le_{R2\ x1'\ x2'})]\!] \Longrightarrow (\mathit{Dep\text{-}Fun\text{-}Rel\text{-}rel}\ \mathit{flip.t1.right\text{-}Galois}\ \mathit{t2.left\text{-}Galois})\restriction_{\mathit{in\text{-}dom}\ \mathit{flip.R}}\upharpoonleft_{\mathit{in\text{-}codom}\ \mathit{flip.L}} = \mathit{Dep\text{-}Fun\text{-}Rel\text{-}rel}\ \mathit{flip.t1.right\text{-}Galois}\ \mathit{t2.left\text{-}Galois}$

$\mathit{t1.half\text{-}galois\text{-}prop\text{-}left} \Longrightarrow ([x\ x' :: \mathit{flip.t1.right\text{-}Galois}] \Rrightarrow (?S\ x\ x')\restriction_{\mathit{in\text{-}dom}\ (\le_{L2\ x\ r1\ x'})}\upharpoonleft_{\mathit{in\text{-}codom}\ (\le}$ $= (\mathit{Dep\text{-}Fun\text{-}Rel\text{-}rel}\ \mathit{flip.t1.right\text{-}Galois}\ ?S)\restriction_{\mathit{in\text{-}dom}\ \mathit{flip.R}}\upharpoonleft_{\mathit{in\text{-}codom}\ \mathit{flip.L}}$

end

Monotone Function Relator **context** *transport-Mono-Fun-Rel*
begin

interpretation *flip* : *transport-Mono-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2* **.**

Closure of Order and Galois Concepts **lemma** *preorder-galois-connection-if-galois-connectionI*:
assumes $((\le_{L1}) \dashv (\le_{R1}))\ l1\ r1$
and $\mathit{reflexive\text{-}on}\ (\mathit{in\text{-}codom}\ (\le_{L1}))\ (\le_{L1})\ \mathit{reflexive\text{-}on}\ (\mathit{in\text{-}dom}\ (\le_{R1}))\ (\le_{R1})$
and $((\le_{L2}) \dashv (\le_{R2}))\ l2\ r2$
and $\mathit{transitive}\ (\le_{L2})\ \mathit{transitive}\ (\le_{R2})$

shows $((\leq_L) \dashv_{pre} (\leq_R))$ *l r*
using *assms* **by** (*intro tpdfr.preorder-galois-connectionI*
galois-connection-left-rightI
tpdfr.preorder-on-in-field-leftI flip.tpdfr.preorder-on-in-field-leftI
tfr.transitive-leftI' flip.tfr.transitive-leftI)
auto

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*
using *assms* **by** (*intro preorder-galois-connection-if-galois-connectionI*)
(*auto intro*: *reflexive-on-if-le-pred-if-reflexive-on*
in-field-if-in-dom in-field-if-in-codom)

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*
using *assms* **by** (*intro tpdfr.preorder-equivalence-if-galois-equivalenceI*
galois-equivalenceI
tpdfr.preorder-on-in-field-leftI flip.tpdfr.preorder-on-in-field-leftI
tfr.transitive-leftI flip.tfr.transitive-leftI)
(*auto intro*: *reflexive-on-if-le-pred-if-reflexive-on in-field-if-in-dom*)

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*
using *assms* **by** (*intro preorder-equivalence-if-galois-equivalenceI*) *auto*

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*
using *assms* **by** (*intro tpdfr.partial-equivalence-rel-equivalence-if-galois-equivalenceI*
galois-equivalenceI
partial-equivalence-rel-leftI flip.partial-equivalence-rel-leftI)
auto

Simplification of Left and Right Relations See $\llbracket$*reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$; *partial-equivalence-rel* $(\leq_{L2})\rrbracket \Longrightarrow$ *flip.tpdfr.R* = *flip.tfr.tdfr.R*.

Simplification of Galois relator See $\llbracket((\leq_{L1}) \Rrightarrow_m (\leq_{R1}))$ *l1*; *tdfrs.t1.galois-prop l1 r1*; *reflexive-on* (*in-dom* $(\leq_{L1})$) $(\leq_{L1})$; $((\leq_{R2}) \Rrightarrow_m (\leq_{L2}))$ *r2*; *transitive* $(\leq_{L2})\rrbracket \Longrightarrow$ *flip.tpdfr.right-Galois* = (*flip.tdfrs.t1.right-Galois*

$\Rrightarrow$ *flip.tdfrs.t2.right-Galois*$)\restriction_{\textit{in-dom flip.tpdfr.R}}\upharpoonleft_{\textit{in-codom flip.tpdfr.L}}$

$[\![((\le_{L1}) \Rrightarrow_m (\le_{R1}))$ *l1*; $((\le_{R1}) \Rrightarrow_m (\le_{L1}))$ *r1*; *tdfrs.t1.half-galois-prop-right*; *reflexive-on* (*in-field* $(\le_{L1})$) $(\le_{L1})$; *reflexive-on* (*in-field* $(\le_{R1})$) $(\le_{R1})$; *tdfrs.t2.half-galois-prop-left*; *partial-equivalence-rel* $(\le_{L2})$; *partial-equivalence-rel* $(\le_{R2})]\!] \Longrightarrow$ (*flip.tdfrs.t1.right-Galois* $\Rrightarrow$ *flip.tdfrs.t2.right-Galois*$)\restriction_{\textit{in-dom flip.tpdfr.R}}\upharpoonleft_{\textit{in-codom flip.tpdfr.L}}$ = (*flip.tdfrs.t1.right-Galois* $\Rrightarrow$ *flip.tdfrs.t2.right-Galois*)

tdfrs.t1.half-galois-prop-left $\Longrightarrow$ (*flip.tdfrs.t1.right-Galois* $\Rrightarrow$ *?S*$\restriction_{\textit{in-dom} (\le_{L2})}\upharpoonleft_{\textit{in-codom} (\le_{R2})})\restriction_{\textit{in-d}}$
= (*flip.tdfrs.t1.right-Galois* $\Rrightarrow$ *?S*$)\restriction_{\textit{in-dom flip.tpdfr.R}}\upharpoonleft_{\textit{in-codom flip.tpdfr.L}}$.

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*; ($[x1\ x2 :: ?L1.0^{-1}] \Rrightarrow_m [x3\ x4 :: ?L1.0] \Rrightarrow ?L1.0\ x1\ x3 \longrightarrow (\le)$) *?L2.0*; $\bigwedge x1\ x2.$ *?L1.0 x1 x2* $\Longrightarrow$ *partial-equivalence-rel* (*?L2.0 x1 x2*)$]\!] \Longrightarrow$ *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* **.**
interpretation *flip* : *transport-Mono-Dep-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2* **.**

theorem *partial-equivalence-rel-equivalenceI*:
  assumes $((\le_{L1}) \equiv_{PER} (\le_{R1}))$ *l1 r1*
  and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\le_{L2\ x\ (r1\ x')}) \equiv_{PER} (\le_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
  and $([x1\ x2 :: (\ge_{L1})] \Rrightarrow_m [x3\ x4 :: (\le_{L1}) \mid x1 \le_{L1} x3] \Rrightarrow (\le))$ *L2*
  and $([x1'\ x2' :: (\ge_{R1})] \Rrightarrow_m [x3'\ x4' :: (\le_{R1}) \mid x1' \le_{R1} x3'] \Rrightarrow (\le))$ *R2*
  and $([x1'\ x2' :: (\le_{R1})] \Rrightarrow_m [x1\ x2 :: (\le_{L1}) \mid x2\ {}_{L1}\lessapprox x1'] \Rrightarrow$
    $[\textit{in-field}\ (\le_{L2\ x1\ (r1\ x2')})] \Rrightarrow (\le_{R2\ (l1\ x1)\ x2'}))$ *l2*
  and $([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x2\ {}_{L1}\lessapprox x1'] \Rrightarrow$
    $[\textit{in-field}\ (\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')}))$ *r2*
  shows $((\le_L) \equiv_{PER} (\le_R))$ *l r*
proof −
  from *assms* **have** $((\le_L) \equiv_{PER} (\le_R))$ = (*tpdfr.L* $\equiv_{PER}$ *tpdfr.R*)
    by (*subst tpdfr.left-rel-eq-tdfr-leftI-if-equivalencesI*
      flip.left-rel-eq-tdfr-leftI-if-equivalencesI,
     auto intro!: *partial-equivalence-rel-left2-if-partial-equivalence-rel-equivalenceI*
      partial-equivalence-rel-right2-if-partial-equivalence-rel-equivalenceI
     iff: *t1.galois-equivalence-right-left-iff-galois-equivalence-left-right*)+

with *assms* **show** *?thesis*
by (*auto intro*!: *tpdfr.partial-equivalence-rel-equivalenceI*)
qed

end

Function Relator **context** *transport-Fun-Rel*
begin

interpretation *tpfr* : *transport-Mono-Fun-Rel L1 R1 l1 r1 L2 R2 l2 r2* **.**
interpretation *flip-tpfr* : *transport-Mono-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2* **.**

theorem *partial-equivalence-rel-equivalenceI*:
assumes $((\le_{L1}) \equiv_{PER} (\le_{R1}))$ *l1 r1*
and $((\le_{L2}) \equiv_{PER} (\le_{R2}))$ *l2 r2*
shows $((\le_{L}) \equiv_{PER} (\le_{R}))$ *l r*
proof −
from *assms* **have** $((\le_{L}) \equiv_{PER} (\le_{R})) = (tpfr.tpdfr.L \equiv_{PER} tpfr.tpdfr.R)$
by (*subst tpfr.left-rel-eq-tfr-leftI flip-tpfr.left-rel-eq-tfr-leftI*; *auto*)+
with *assms* **show** *?thesis* **by** (*auto intro*!: *tpfr.partial-equivalence-rel-equivalenceI*)
qed

end

end

2.9 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*
by (*intro transport-bijection.intro*) *auto*

interpretation *transport L L id id* **.**

lemma *left-Galois-eq-left*: $({}_{L}\lessapprox) = (\le_{L})$
by (*intro ext iffI*) *auto*

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*
  using *reflexive-on-in-field* **by** *unfold-locales auto*

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*
  using *preorder-on-in-field* **by** *unfold-locales auto*

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*
  using *partial-equivalence-rel* **by** *unfold-locales auto*

end

interpretation *transport-eq-restrict-id* :
  transport-eq-restrict-bijection P P *id id* **for** $P :: 'a \Rightarrow bool$
  using *bijection-on-self-id* **by** (*unfold-locales*) *auto*

interpretation *transport-eq-id* : *transport-eq-bijection id id*
  using *bijection-on-self-id* **by** (*unfold-locales*) *auto*

end

theory *Transport*
  imports
    Transport-Bijections
    Transport-Compositions
    Transport-Functions
    Transport-Identity
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

2.10 Transport for Natural Functors

2.10.1 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*: $BNF\text{-}Def.Grp\ UNIV\ f = (=) \circ f$
by (*intro ext*) (*auto elim*: *GrpE intro*: *GrpI*)

lemma *eq-comp-rel-comp-eq-comp*: $(=) \circ f \circ\circ R = R \circ f$
by (*intro ext*) *auto*

lemma *Domain-Collect-case-prod-eq-Collect-in-dom*:
$Domain\ \{(x,\ y).\ R\ x\ y\} = \{x.\ in\text{-}dom\ R\ x\}$
by *blast*

lemma *ball-in-dom-iff-ball-ex*:
$(\forall x \in S.\ in\text{-}dom\ R\ x) \longleftrightarrow (\forall x \in S.\ \exists y.\ R\ x\ y)$
by *blast*

lemma *pair-mem-Collect-case-prod-iff*: $(x,\ y) \in \{(x,\ y).\ R\ x\ y\} \longleftrightarrow R\ x\ y$
by *blast*

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*: $Fmap\ id\ id\ id\ x = x$
by (*subst Fmap-id*, *subst id-eq-self*, *rule refl*)

lemma *Fmap-comp-eq-Fmap-Fmap*:
$Fmap\ (g1 \circ f1)\ (g2 \circ f2)\ (g3 \circ f3)\ x = Fmap\ g1\ g2\ g3\ (Fmap\ f1\ f2\ f3\ x)$
by (*fact fun-cong*[*OF Fmap-comp*, *simplified comp-eq*])

lemma *Fset1-Fmap-eq-image-Fset1*: $Fset1\ (Fmap\ f1\ f2\ f3\ x) = f1\ `\ Fset1\ x$
by (*fact fun-cong*[*OF Fset1-natural*, *simplified comp-eq*])

lemma *Fset2-Fmap-eq-image-Fset2*: $Fset2\ (Fmap\ f1\ f2\ f3\ x) = f2\ `\ Fset2\ x$
by (*fact fun-cong*[*OF Fset2-natural*, *simplified comp-eq*])

lemma *Fset3-Fmap-eq-image-Fset3*: $Fset3\ (Fmap\ f1\ f2\ f3\ x) = f3\ `\ Fset3\ x$
by (*fact fun-cong*[*OF Fset3-natural*, *simplified comp-eq*])

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* ⊆ {(*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*
apply (*subst Frel-def*)
apply (*intro exI conjI CollectI*)
apply (*fact assms*)+
done

lemma *FrelE*:
assumes *Frel R1 R2 R3 x y*
obtains *z* **where** *Fset1 z* ⊆ {(*x*, *y*). *R1 x y*} *Fset2 z* ⊆ {(*x*, *y*). *R2 x y*}
Fset3 z ⊆ {(*x*, *y*). *R3 x y*} *Fmap fst fst fst z* = *x Fmap snd snd snd z* = *y*
apply (*insert assms*)
apply (*subst* (*asm*) *Frel-def*)
apply (*elim exE CollectE conjE*)
apply *assumption*
done

lemma *Grp-UNIV-Fmap-eq-Frel-Grp*: *BNF-Def.Grp UNIV* (*Fmap f1 f2 f3*) =
Frel (*BNF-Def.Grp UNIV f1*) (*BNF-Def.Grp UNIV f2*) (*BNF-Def.Grp UNIV f3*)
apply (*intro ext iffI*)
apply (*rule FrelI*[**where**
?z=Fmap (*BNF-Def.convol id f1*) (*BNF-Def.convol id f2*) (*BNF-Def.convol id f3*) *-*])
apply (*subst Fset-Fmap-eqs*,
rule image-subsetI,
rule convol-mem-GrpI[*simplified Fun-id-eq-id*],
rule UNIV-I)+
apply (*unfold Fmap-comp-eq-Fmap-Fmap*[*symmetric*]
fst-convol[*simplified Fun-comp-eq-comp*]
snd-convol[*simplified Fun-comp-eq-comp*]
Fmap-id id-eq-self)
apply (*rule refl*)
apply (*subst* (*asm*) *Grp-UNIV-eq-eq-comp*)
apply (*subst* (*asm*) *comp-eq*)
apply *assumption*
apply (*erule FrelE*)
apply *hypsubst*
apply (*subst Grp-UNIV-eq-eq-comp*)
apply (*subst comp-eq*)
apply (*subst Fmap-comp-eq-Fmap-Fmap*[*symmetric*])
apply (*rule Fmap-cong*;
rule Collect-case-prod-Grp-eqD[*simplified Fun-comp-eq-comp*],
drule rev-subsetD,
assumption+)
done

lemma *Frel-Grp-UNIV-Fmap*:
Frel (*BNF-Def.Grp UNIV f1*) (*BNF-Def.Grp UNIV f2*) (*BNF-Def.Grp UNIV f3*)
x (*Fmap f1 f2 f3 x*)
apply (*subst Grp-UNIV-Fmap-eq-Frel-Grp*[*symmetric*])
apply (*subst Grp-UNIV-eq-eq-comp*)
apply (*subst comp-eq*)
apply (*rule refl*)
done

lemma *Frel-Grp-UNIV-iff-eq-Fmap*:
Frel (*BNF-Def.Grp UNIV f1*) (*BNF-Def.Grp UNIV f2*) (*BNF-Def.Grp UNIV f3*) *x y* $\longleftrightarrow$
(*y* = *Fmap f1 f2 f3 x*)
by (*subst eq-commute*[*of y*])
(*fact fun-cong*[*OF fun-cong*[*OF Grp-UNIV-Fmap-eq-Frel-Grp*],
simplified Grp-UNIV-eq-eq-comp comp-eq, *folded Grp-UNIV-eq-eq-comp*, *symmetric*])

lemma *Frel-eq*: *Frel* (=) (=) (=) = (=)
apply (*unfold BNF-Def.eq-alt*[*simplified Fun-id-eq-id*])
apply (*subst Grp-UNIV-Fmap-eq-Frel-Grp*[*symmetric*])
apply (*subst Fmap-id*)
apply (*fold BNF-Def.eq-alt*[*simplified Fun-id-eq-id*])
apply (*rule refl*)
done

corollary *Frel-eq-self*: *Frel* (=) (=) (=) *x x*
by (*fact iffD2*[*OF fun-cong*[*OF fun-cong*[*OF Frel-eq*]] *refl*])

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*
apply (*insert assms*(*1*))
apply (*erule FrelE*)
apply (*rule FrelI*)
apply (*rule subsetI*,
frule rev-subsetD,
assumption,
frule imageI[*of - Fset1 - fst*]
imageI[*of - Fset2 - fst*]
imageI[*of - Fset3 - fst*],
drule imageI[*of - Fset1 - snd*]
imageI[*of - Fset2 - snd*]
imageI[*of - Fset3 - snd*],
(*subst* (*asm*) *Fset-Fmap-eqs*[*symmetric*])+,

    intro CollectI case-prodI2,
    rule assms;
    hypsubst,
    unfold fst-conv snd-conv,
    (elim CollectE case-prodE Pair-inject, hypsubst)?,
    assumption)+
  apply *assumption+*
done

corollary *Frel-mono*:
  assumes $R1 \leq S1$ $R2 \leq S2$ $R3 \leq S3$
  shows $Frel\ R1\ R2\ R3 \leq Frel\ S1\ S2\ S3$
  apply (*intro le-relI*)
  apply (*rule Frel-mono-strong*)
    apply *assumption*
    apply (*insert assms*)
    apply (*drule le-relD*[*OF assms*(*1*)] *le-relD*[*OF assms*(*2*)] *le-relD*[*OF assms*(*3*)],
      assumption)+
  done

lemma *Frel-refl-strong*:
  assumes $\bigwedge x1.\ x1 \in Fset1\ x \Longrightarrow R1\ x1\ x1$
  and $\bigwedge x2.\ x2 \in Fset2\ x \Longrightarrow R2\ x2\ x2$
  and $\bigwedge x3.\ x3 \in Fset3\ x \Longrightarrow R3\ x3\ x3$
  shows *Frel R1 R2 R3 x x*
  by (*rule Frel-mono-strong*[*OF Frel-eq-self*[*of x*]];
    drule assms, *hypsubst*, *assumption*)

lemma *Frel-cong*:
  assumes $\bigwedge x1\ y1.\ x1 \in Fset1\ x \Longrightarrow y1 \in Fset1\ y \Longrightarrow R1\ x1\ y1 \longleftrightarrow R1'\ x1\ y1$
  and $\bigwedge x2\ y2.\ x2 \in Fset2\ x \Longrightarrow y2 \in Fset2\ y \Longrightarrow R2\ x2\ y2 \longleftrightarrow R2'\ x2\ y2$
  and $\bigwedge x3\ y3.\ x3 \in Fset3\ x \Longrightarrow y3 \in Fset3\ y \Longrightarrow R3\ x3\ y3 \longleftrightarrow R3'\ x3\ y3$
  shows $Frel\ R1\ R2\ R3\ x\ y = Frel\ R1'\ R2'\ R3'\ x\ y$
  by (*rule iffI*;
    rule Frel-mono-strong,
    assumption;
    rule iffD1[*OF assms*(*1*)] *iffD1*[*OF assms*(*2*)] *iffD1*[*OF assms*(*3*)]
      iffD2[*OF assms*(*1*)] *iffD2*[*OF assms*(*2*)] *iffD2*[*OF assms*(*3*)];
    assumption)

lemma *Frel-rel-inv-eq-rel-inv-Frel*: $Frel\ R1^{-1}\ R2^{-1}\ R3^{-1} = (Frel\ R1\ R2\ R3)^{-1}$
  by (*intro ext iffI*;
    unfold rel-inv-iff-rel,
    erule FrelE,
    hypsubst,
    rule FrelI[**where** *?z=Fmap prod.swap prod.swap prod.swap -*];
      ((*subst Fset-Fmap-eqs*,
      rule image-subsetI,
      drule rev-subsetD,

```
    assumption,
    elim CollectE case-prodE,
    hypsubst,
    subst swap-simp,
    subst pair-mem-Collect-case-prod-iff,
    assumption) |
    (subst Fmap-comp-eq-Fmap-Fmap[symmetric],
      rule Fmap-cong;
      unfold comp-eq fst-swap snd-swap,
      rule refl)))
```

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
  unfolding fstOp-def sndOp-def by (intro ext) simp

lemma Frel-rel-comp-le-Frel-comp:
  Frel (R1 ∘∘ S1) (R2 ∘∘ S2) (R3 ∘∘ S3) ≤ (Frel R1 R2 R3 ∘∘ Frel S1 S2 S3)
  apply (rule le-relI)
  apply (erule FrelE)
  apply (rule rel-compI[where ?y=Fmap (snd ∘ BNF-Def.fstOp R1 S1)
    (snd ∘ BNF-Def.fstOp R2 S2) (snd ∘ BNF-Def.fstOp R3 S3) -])
  apply (rule FrelI[where ?z=Fmap (BNF-Def.fstOp R1 S1)
    (BNF-Def.fstOp R2 S2) (BNF-Def.fstOp R3 S3) -])
    apply (subst Fset-Fmap-eqs,
      intro image-subsetI,
      rule fstOp-in[unfolded relcompp-eq-rel-comp],
      drule rev-subsetD,
      assumption+)+
    apply (subst Fmap-comp-eq-Fmap-Fmap[symmetric])
    apply (fold ext[of fst fst ∘ -, OF fst-fstOp[unfolded Fun-comp-eq-comp]])
    apply hypsubst
    apply (rule refl)
    apply (subst Fmap-comp-eq-Fmap-Fmap[symmetric])
    apply (rule refl)
    apply (rule FrelI[where ?z=Fmap (BNF-Def.sndOp R1 S1)
      (BNF-Def.sndOp R2 S2) (BNF-Def.sndOp R3 S3) -])
    apply (subst Fset-Fmap-eqs,
      intro image-subsetI,
      rule sndOp-in[unfolded relcompp-eq-rel-comp],
      drule rev-subsetD,
      assumption+)+
    apply (subst Fmap-comp-eq-Fmap-Fmap[symmetric])
    apply (unfold fst-sndOp-eq-snd-fstOp)
```

apply (*rule refl*)
apply (*subst Fmap-comp-eq-Fmap-Fmap*[*symmetric*])
apply (*fold ext*[*of snd snd* ∘ -, *OF snd-sndOp*[*unfolded Fun-comp-eq-comp*]])
apply *hypsubst*
apply (*rule refl*)
done

corollary *Frel-comp-eq-Frel-rel-comp*:
Frel R1 R2 R3 ∘∘ *Frel S1 S2 S3* = *Frel* (*R1* ∘∘ *S1*) (*R2* ∘∘ *S2*) (*R3* ∘∘ *S3*)
by (*rule antisym*; *rule Frel-comp-le-Frel-rel-comp Frel-rel-comp-le-Frel-comp*)

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*
apply (*rule iffI*)
apply (*fold comp-eq*[*of R1*] *comp-eq*[*of R2*] *comp-eq*[*of R3*])
apply (*drule rel-compI*[**where** *?R=Frel - - -* **and** *?S=Frel - - -*,
OF Frel-Grp-UNIV-Fmap])
apply (*unfold Grp-UNIV-eq-eq-comp*)
apply (*drule le-relD*[*OF Frel-comp-le-Frel-rel-comp*])
apply (*unfold eq-comp-rel-comp-eq-comp*)
apply *assumption*
apply (*fold eq-comp-rel-comp-eq-comp*[**where** *?R=R1*]
eq-comp-rel-comp-eq-comp[**where** *?R=R2*]
eq-comp-rel-comp-eq-comp[**where** *?R=R3*]
Grp-UNIV-eq-eq-comp)
apply (*drule le-relD*[*OF Frel-rel-comp-le-Frel-comp*])
apply (*erule rel-compE*)
apply (*subst* (*asm*) *Frel-Grp-UNIV-iff-eq-Fmap*)
apply *hypsubst*
apply *assumption*
done

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*
apply (*subst rel-inv-iff-rel*[*of Frel - - -*, *symmetric*])
apply (*subst Frel-rel-inv-eq-rel-inv-Frel*[*symmetric*])
apply (*subst Frel-Fmap-eq1*)
apply (*rule sym*)
apply (*subst rel-inv-iff-rel*[*of Frel - - -*, *symmetric*])
apply (*subst Frel-rel-inv-eq-rel-inv-Frel*[*symmetric*])
apply (*unfold rel-inv-iff-rel*)
apply (*rule refl*)
done

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* ($(=)\restriction_{P1}$) ($(=)\restriction_{P2}$) ($(=)\restriction_{P3}$) *x x*

lemma *Fpred-mono-strong*:
assumes *Fpred P1 P2 P3 x*
and $\bigwedge x1.\ x1 \in Fset1\ x \Longrightarrow P1\ x1 \Longrightarrow Q1\ x1$
and $\bigwedge x2.\ x2 \in Fset2\ x \Longrightarrow P2\ x2 \Longrightarrow Q2\ x2$
and $\bigwedge x3.\ x3 \in Fset3\ x \Longrightarrow P3\ x3 \Longrightarrow Q3\ x3$
shows *Fpred Q1 Q2 Q3 x*
apply (*insert assms*(*1*))
apply (*unfold Fpred-def*)
apply (*rule Frel-mono-strong*,
assumption;
erule restrict-leftE,
rule restrict-leftI,
assumption,
rule assms,
assumption+)
done

lemma *Fpred-top*: *Fpred* ⊤ ⊤ ⊤ *x*
apply (*subst Fpred-def*)
apply (*rule Frel-refl-strong*;
subst restrict-left-top-eq,
rule refl)
done

lemma *FpredI*:
assumes $\bigwedge x1.\ x1 \in Fset1\ x \Longrightarrow P1\ x1$
and $\bigwedge x2.\ x2 \in Fset2\ x \Longrightarrow P2\ x2$
and $\bigwedge x3.\ x3 \in Fset3\ x \Longrightarrow P3\ x3$
shows *Fpred P1 P2 P3 x*
using *assms* **by** (*rule Fpred-mono-strong*[*OF Fpred-top*])

lemma *FpredE*:
assumes *Fpred P1 P2 P3 x*
obtains $\bigwedge x1.\ x1 \in Fset1\ x \Longrightarrow P1\ x1$
$\bigwedge x2.\ x2 \in Fset2\ x \Longrightarrow P2\ x2$
$\bigwedge x3.\ x3 \in Fset3\ x \Longrightarrow P3\ x3$
by (*elim meta-impE*; (*assumption* |
insert assms,
subst (*asm*) *Fpred-def*,
erule FrelE,
hypsubst,
subst (*asm*) *Fset-Fmap-eqs*,
subst (*asm*) *Domain-fst*[*symmetric*],
drule rev-subsetD,
rule Domain-mono,
assumption,
unfold Domain-Collect-case-prod-eq-Collect-in-dom in-dom-restrict-left-eq,
elim CollectE inf1E,

assumption))

lemma *Fpred-eq-ball*: *Fpred P1 P2 P3* =
(λ*x*. *Ball* (*Fset1 x*) *P1* ∧ *Ball* (*Fset2 x*) *P2* ∧ *Ball* (*Fset3 x*) *P3*)
by (*intro ext iffI conjI ballI FpredI*; *elim FpredE conjE bspec*)

lemma *Fpred-Fmap-eq*:
Fpred P1 P2 P3 (*Fmap f1 f2 f3 x*) = *Fpred* (*P1* ∘ *f1*) (*P2* ∘ *f2*) (*P3* ∘ *f3*) *x*
by (*unfold Fpred-def Frel-Fmap-eqs*)
(*rule iffI*;
erule FrelE,
hypsubst,
unfold Frel-Fmap-eqs,
rule Frel-refl-strong;
rule restrict-leftI,
rule refl,
drule rev-subsetD,
assumption,
elim CollectE case-prodE restrict-leftE,
hypsubst,
unfold comp-eq fst-conv,
assumption)

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*
apply (*insert assms*)
apply (*elim in-domE FrelE*)
apply *hypsubst*
apply (*subst Fpred-Fmap-eq*)
apply (*rule FpredI*;
drule rev-subsetD,
assumption,
elim CollectE case-prodE,
hypsubst,
unfold comp-eq fst-conv,
rule in-domI,
assumption)
done

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*
apply (*insert assms*)
apply (*subst* (*asm*) *Fpred-eq-ball*)
apply (*elim conjE*)
apply (*subst* (*asm*) *ball-in-dom-iff-ball-ex*,
drule bchoice, — requires the axiom of choice.
erule exE)+

apply (*rule in-domI*[**where** *?x*=*x* **and** *?y*=*Fmap - - - x* **for** *x*])
apply (*subst Frel-Fmap-eq2*)
apply (*rule Frel-refl-strong*)
apply (*drule bspec*[*of Fset1 -*])
apply *assumption+*
apply (*drule bspec*[*of Fset2 -*])
apply *assumption+*
apply (*drule bspec*[*of Fset3 -*])
apply *assumption+*
done

lemma *in-dom-Frel-eq-Fpred-in-dom*:
in-dom (*Frel R1 R2 R3*) = *Fpred* (*in-dom R1*) (*in-dom R2*) (*in-dom R3*)
by (*intro ext iffI*; *rule Fpred-in-dom-if-in-dom-Frel in-dom-Frel-if-Fpred-in-dom*)

lemma *in-codom-Frel-eq-Fpred-in-codom*:
in-codom (*Frel R1 R2 R3*) = *Fpred* (*in-codom R1*) (*in-codom R2*) (*in-codom R3*)
apply (*subst in-dom-rel-inv-eq-in-codom*[*symmetric*])
apply (*subst Frel-rel-inv-eq-rel-inv-Frel*[*symmetric*])
apply (*subst in-dom-Frel-eq-Fpred-in-dom*)
apply (*subst in-dom-rel-inv-eq-in-codom*)+
apply (*rule refl*)
done

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*)
apply (*subst in-field-eq-in-dom-sup-in-codom*)
apply (*subst in-dom-Frel-eq-Fpred-in-dom*)
apply (*subst in-codom-Frel-eq-Fpred-in-codom*)
apply (*rule refl*)
done

lemma *Frel-restrict-left-Fpred-eq-Frel-restrict-left*:
fixes *R1* :: $'a1 \Rightarrow 'a2 \Rightarrow bool$
and *R2* :: $'b1 \Rightarrow 'b2 \Rightarrow bool$
and *R3* :: $'c1 \Rightarrow 'c2 \Rightarrow bool$
and *P1* :: $'a1 \Rightarrow bool$
and *P2* :: $'b1 \Rightarrow bool$
and *P3* :: $'c1 \Rightarrow bool$
shows $(Frel\ R1\ R2\ R3 :: ('d, 'a1, 'b1, 'c1)\ F \Rightarrow -)\restriction_{Fpred\ P1\ P2\ P3 :: ('d, 'a1, 'b1, 'c1)\ F \Rightarrow -}$
=
$Frel\ (R1\restriction_{P1})\ (R2\restriction_{P2})\ (R3\restriction_{P3})$
apply (*intro ext*)
apply (*rule iffI*)
apply (*erule restrict-leftE*)
apply (*elim FpredE*)
apply (*rule Frel-mono-strong*,

assumption;
rule restrict-leftI,
assumption+)
apply (*rule restrict-leftI*)
apply (*rule Frel-mono-strong*,
assumption;
erule restrict-leftE,
assumption)
apply (*drule in-domI*[*of Frel* ($R1\restriction_{P1}$) ($R2\restriction_{P2}$) ($R3\restriction_{P3}$)])
apply (*drule Fpred-in-dom-if-in-dom-Frel*)
apply (*rule Fpred-mono-strong*,
assumption;
unfold in-dom-restrict-left-eq inf-apply inf-bool-def;
rule conjunct2,
assumption)
done

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* ⇒ -)$\upharpoonleft_{Fpred\ P1\ P2\ P3\ ::\ ('d,\ 'a2,\ 'b2,\ 'c2)\ F\ \Rightarrow\ -}$ =
Frel ($R1\upharpoonleft_{P1}$) ($R2\upharpoonleft_{P2}$) ($R3\upharpoonleft_{P3}$)
apply (*subst restrict-right-eq*)
apply (*subst Frel-rel-inv-eq-rel-inv-Frel*[*symmetric*])
apply (*subst Frel-restrict-left-Fpred-eq-Frel-restrict-left*)
apply (*subst Frel-rel-inv-eq-rel-inv-Frel*[*symmetric*])
apply (*fold restrict-right-eq*)
apply (*rule refl*)
done

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* ⇒ *'a2*
and *L3* :: *'a3* ⇒ *'a3* ⇒ *bool*
and *R3* :: *'b3* ⇒ *'b3* ⇒ *bool*
and *l3* :: *'a3* ⇒ *'b3*

and *r3* :: *'b3* ⇒ *'a3*
begin

notation *L1* (**infix** $\leq_{L1}$ *50*)
notation *R1* (**infix** $\leq_{R1}$ *50*)
notation *L2* (**infix** $\leq_{L2}$ *50*)
notation *R2* (**infix** $\leq_{R2}$ *50*)
notation *L3* (**infix** $\leq_{L3}$ *50*)
notation *R3* (**infix** $\leq_{R3}$ *50*)

notation *t1.ge-left* (**infix** $\geq_{L1}$ *50*)
notation *t1.ge-right* (**infix** $\geq_{R1}$ *50*)
notation *t2.ge-left* (**infix** $\geq_{L2}$ *50*)
notation *t2.ge-right* (**infix** $\geq_{R2}$ *50*)
notation *t3.ge-left* (**infix** $\geq_{L3}$ *50*)
notation *t3.ge-right* (**infix** $\geq_{R3}$ *50*)

notation *t1.left-Galois* (**infix** ${}_{L1}\lessapprox$ *50*)
notation *t1.right-Galois* (**infix** ${}_{R1}\lessapprox$ *50*)
notation *t2.left-Galois* (**infix** ${}_{L2}\lessapprox$ *50*)
notation *t2.right-Galois* (**infix** ${}_{R2}\lessapprox$ *50*)
notation *t3.left-Galois* (**infix** ${}_{L3}\lessapprox$ *50*)
notation *t3.right-Galois* (**infix** ${}_{R3}\lessapprox$ *50*)

notation *t1.ge-Galois-left* (**infix** $\gtrapprox_{L1}$ *50*)
notation *t1.ge-Galois-right* (**infix** $\gtrapprox_{R1}$ *50*)
notation *t2.ge-Galois-left* (**infix** $\gtrapprox_{L2}$ *50*)
notation *t2.ge-Galois-right* (**infix** $\gtrapprox_{R2}$ *50*)
notation *t3.ge-Galois-left* (**infix** $\gtrapprox_{L3}$ *50*)
notation *t3.ge-Galois-right* (**infix** $\gtrapprox_{R3}$ *50*)

notation *t1.right-ge-Galois* (**infix** ${}_{R1}\gtrapprox$ *50*)
notation *t1.Galois-right* (**infix** $\lessapprox_{R1}$ *50*)
notation *t2.right-ge-Galois* (**infix** ${}_{R2}\gtrapprox$ *50*)
notation *t2.Galois-right* (**infix** $\lessapprox_{R2}$ *50*)
notation *t3.right-ge-Galois* (**infix** ${}_{R3}\gtrapprox$ *50*)
notation *t3.Galois-right* (**infix** $\lessapprox_{R3}$ *50*)

notation *t1.left-ge-Galois* (**infix** ${}_{L1}\gtrapprox$ *50*)
notation *t1.Galois-left* (**infix** $\lessapprox_{L1}$ *50*)
notation *t2.left-ge-Galois* (**infix** ${}_{L2}\gtrapprox$ *50*)
notation *t2.Galois-left* (**infix** $\lessapprox_{L2}$ *50*)
notation *t3.left-ge-Galois* (**infix** ${}_{L3}\gtrapprox$ *50*)
notation *t3.Galois-left* (**infix** $\lessapprox_{L3}$ *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* ($\leq_{L1}$) ($\leq_{L2}$) ($\leq_{L3}$)
  unfolding *L-def* **..**

definition $l \equiv$ *Fmap l1 l2 l3*

lemma *left-eq-Fmap*: $l =$ *Fmap l1 l2 l3*
  unfolding *l-def* **..**

context
begin

interpretation *flip* :
  transport-natural-functor R1 L1 r1 l1 R2 L2 r2 l2 R3 L3 r3 l3 **.**

abbreviation $R \equiv$ *flip.L*
abbreviation $r \equiv$ *flip.l*

lemma *right-rel-eq-Frel*: $R =$ *Frel* ($\leq_{R1}$) ($\leq_{R2}$) ($\leq_{R3}$)
  unfolding *flip.left-rel-eq-Frel* **..**

lemma *right-eq-Fmap*: $r =$ *Fmap r1 r2 r3*
  unfolding *flip.left-eq-Fmap* **..**

lemmas *transport-defs* = *left-rel-eq-Frel left-eq-Fmap*
  right-rel-eq-Frel right-eq-Fmap

end

sublocale *transport L R l r* **.**

notation *L* (**infix** $\leq_L$ *50*)
notation *R* (**infix** $\leq_R$ *50*)

lemma *unit-eq-Fmap*: $\eta =$ *Fmap* η_1 η_2 η_3
  unfolding *unit-eq-comp* **by** (*simp only*: *right-eq-Fmap left-eq-Fmap*
    flip: *Fmap-comp t1.unit-eq-comp t2.unit-eq-comp t3.unit-eq-comp*)

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* $\equiv \varepsilon_3$
  by (*simp-all only*: *order-functors.flip-counit-eq-unit*)

lemma *counit-eq-Fmap*: $\varepsilon = Fmap\ \varepsilon_1\ \varepsilon_2\ \varepsilon_3$
  by (*fact flip-inv.unit-eq-Fmap*)

lemma *flip-inv-right-eq-ge-left*: *flip-inv.R* $= (\geq_L)$
  unfolding *left-rel-eq-Frel flip-inv.right-rel-eq-Frel*
  by (*fact Frel-rel-inv-eq-rel-inv-Frel*)

interpretation *flip* :
  transport-natural-functor R1 L1 r1 l1 R2 L2 r2 l2 R3 L3 r3 l3 **.**

lemma *flip-inv-left-eq-ge-right*: *flip-inv.L* $\equiv (\geq_R)$
  unfolding *flip.flip-inv-right-eq-ge-left* **.**

lemma *mono-wrt-rel-leftI*:
  assumes $((\leq_{L1}) \Rrightarrow_m (\leq_{R1}))\ l1$
  and $((\leq_{L2}) \Rrightarrow_m (\leq_{R2}))\ l2$
  and $((\leq_{L3}) \Rrightarrow_m (\leq_{R3}))\ l3$
  shows $((\leq_L) \Rrightarrow_m (\leq_R))\ l$
  apply (*unfold left-rel-eq-Frel right-rel-eq-Frel left-eq-Fmap*)
  apply (*rule dep-mono-wrt-relI*)
  apply (*unfold Frel-Fmap-eqs*)
  apply (*fold rel-map-eq*)
  apply (*rule le-relD*[*OF Frel-mono*])
    apply (*subst mono-wrt-rel-iff-le-rel-map*[*symmetric*], *rule assms*)+
    apply *assumption*
  done

end

end

2.10.2 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 $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$
  apply (*rule half-galois-prop-leftI*)
  apply (*erule left-GaloisE*)

apply (*unfold left-rel-eq-Frel right-rel-eq-Frel left-eq-Fmap right-eq-Fmap*)
apply (*subst* (*asm*) *in-codom-Frel-eq-Fpred-in-codom*)
apply (*erule FpredE*)
apply (*unfold Frel-Fmap-eqs*)
apply (*rule Frel-mono-strong*,
assumption;
rule t1.half-galois-prop-leftD t2.half-galois-prop-leftD t3.half-galois-prop-leftD,
rule assms,
rule t1.left-GaloisI t2.left-GaloisI t3.left-GaloisI;
assumption)
done

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 $\bigwedge R\ S\ f\ g.\ (R^{-1}\ {}_h\trianglelefteq\ S^{-1})\ f\ g \equiv (S \trianglelefteq_h R)\ g\ f$
by (*simp-all only*: *flip-inv-left-eq-ge-right flip-inv-right-eq-ge-left*
galois-prop.half-galois-prop-left-rel-inv-iff-half-galois-prop-right)

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*
using *assms* **by** (*intro flip-inv.half-galois-prop-leftI*)

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*
using *assms* **by** (*elim galois-prop.galois-propE*)
(*intro galois-propI half-galois-prop-leftI half-galois-prop-rightI*)

interpretation *flip* :
transport-natural-functor R1 L1 r1 l1 R2 L2 r2 l2 R3 L3 r3 l3 **.**

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*
using *assms* **by** (*elim galois.galois-connectionE*) (*intro*
galois-connectionI galois-propI mono-wrt-rel-leftI flip.mono-wrt-rel-leftI)

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*
using *assms* **by** (*elim galois.galois-equivalenceE flip.t1.galois-connectionE flip.t2.galois-connectionE flip.t3.galois-connectionE*)
(*intro galois-equivalenceI galois-connectionI flip.galois-propI*)

end

end

2.10.3 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* $({}_{L1}\lessapprox)$ $({}_{L2}\lessapprox)$ $({}_{L3}\lessapprox)$
apply (*rule le-relI*)
apply (*erule left-GaloisE*)
apply (*unfold left-rel-eq-Frel right-rel-eq-Frel right-eq-Fmap*)
apply (*subst* (*asm*) *in-codom-Frel-eq-Fpred-in-codom*)
apply (*erule FpredE*)
apply (*subst* (*asm*) *Frel-Fmap-eq2*)
apply (*rule Frel-mono-strong*,
assumption;
rule t1.left-GaloisI t2.left-GaloisI t3.left-GaloisI;
assumption)
done

lemma *Frel-left-Galois-le-left-Galois*:
Frel $({}_{L1}\lessapprox)$ $({}_{L2}\lessapprox)$ $({}_{L3}\lessapprox) \leq ({}_{L}\lessapprox)$
apply (*rule le-relI*)
apply (*unfold t1.left-Galois-iff-in-codom-and-left-rel-right t2.left-Galois-iff-in-codom-and-left-rel-right t3.left-Galois-iff-in-codom-and-left-rel-right*)
apply (*fold*
restrict-right-eq[*of* $\lambda x\ y.\ x \leq_{L1}$ *r1 y in-codom* $(\leq_{R1})$,
unfolded restrict-left-pred-def rel-inv-iff-rel]
restrict-right-eq[*of* $\lambda x\ y.\ x \leq_{L2}$ *r2 y in-codom* $(\leq_{R2})$,
unfolded restrict-left-pred-def rel-inv-iff-rel]
restrict-right-eq[*of* $\lambda x\ y.\ x \leq_{L3}$ *r3 y in-codom* $(\leq_{R3})$,
unfolded restrict-left-pred-def rel-inv-iff-rel])
apply (*subst* (*asm*) *Frel-restrict-right-Fpred-eq-Frel-restrict-right*[*symmetric*])
apply (*erule restrict-rightE*)

apply (*subst* (*asm*) *in-codom-Frel-eq-Fpred-in-codom*[*symmetric*])
apply (*erule in-codomE*)
apply (*rule left-GaloisI*)
  apply (*rule in-codomI*)
  apply (*subst right-rel-eq-Frel*)
  apply *assumption*
  apply (*unfold left-rel-eq-Frel right-eq-Fmap Frel-Fmap-eq2*)
  apply *assumption*
done

corollary *left-Galois-eq-Frel-left-Galois*: $({}_{L}\lessapprox) = Frel\ ({}_{L1}\lessapprox)\ ({}_{L2}\lessapprox)\ ({}_{L3}\lessapprox)$
by (*intro antisym left-Galois-Frel-left-Galois Frel-left-Galois-le-left-Galois*)

end

end

2.10.4 Basic Order Properties

theory *Transport-Natural-Functors-Order-Base*
imports
Transport-Natural-Functors-Base
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*
apply (*subst reflexive-on-iff-eq-restrict-left-le*)
apply (*subst Frel-eq*[*symmetric*])
apply (*unfold R-def*)
apply (*subst in-field-Frel-eq-Fpred-in-in-field*)
apply (*subst restrict-left-sup-eq*)
apply (*subst Frel-restrict-left-Fpred-eq-Frel-restrict-left*)+
apply (*rule le-supI*;
rule Frel-mono;
subst reflexive-on-iff-eq-restrict-left-le[*symmetric*],
rule reflexive-on-if-le-pred-if-reflexive-on,
rule assms,
rule le-predI[*OF in-field-if-in-dom*]
le-predI[*OF in-field-if-in-codom*],
assumption)
done

lemma *transitive-FrelI*:
assumes *transitive R1*

and *transitive R2*
and *transitive R3*
shows *transitive* (*Frel R1 R2 R3*)
apply (*subst transitive-iff-rel-comp-le-self*)
apply (*subst Frel-comp-eq-Frel-rel-comp*)
apply (*rule Frel-mono*;
subst transitive-iff-rel-comp-le-self[*symmetric*],
rule assms)
done

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* ≡ *Frel R1 R2 R3*
shows *preorder-on* (*in-field R*) *R*
apply (*unfold R-def*)
apply (*insert assms*)
apply (*elim preorder-on-in-fieldE*)
apply (*rule preorder-onI*)
apply (*rule reflexive-on-in-field-FrelI*; *assumption*)
apply (*subst transitive-on-in-field-iff-transitive*)
apply (*rule transitive-FrelI*; *assumption*)
done

lemma *symmetric-FrelI*:
assumes *symmetric R1*
and *symmetric R2*
and *symmetric R3*
shows *symmetric* (*Frel R1 R2 R3*)
apply (*subst symmetric-iff-rel-inv-eq-self*)
apply (*subst Frel-rel-inv-eq-rel-inv-Frel*[*symmetric*])
apply (*subst rel-inv-eq-self-if-symmetric*, *fact*)+
apply (*rule refl*)
done

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*)
apply (*insert assms*)
apply (*elim partial-equivalence-relE preorder-on-in-fieldE*)
apply (*rule partial-equivalence-relI*;
rule transitive-FrelI symmetric-FrelI;
assumption)
done

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

2.10.5 Order Equivalence

theory *Transport-Natural-Functors-Order-Equivalence*
imports
Transport-Natural-Functors-Base
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*
defines *R* ≡ *Frel R1 R2 R3*
shows *inflationary-on* (*in-dom R*) *R* (*Fmap f1 f2 f3*)
apply (*unfold R-def*)
apply (*rule inflationary-onI*)
apply (*subst* (*asm*) *in-dom-Frel-eq-Fpred-in-dom*)
apply (*erule FpredE*)
apply (*subst Frel-Fmap-eq2*)
apply (*rule Frel-refl-strong*)
apply (*rule inflationary-onD*[**where** *?R=R1*] *inflationary-onD*[**where** *?R=R2*]
inflationary-onD[**where** *?R=R3*],
rule assms,
assumption+)+
done

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*

defines *R* ≡ *Frel R1 R2 R3*
shows *inflationary-on* (*in-codom R*) *R* (*Fmap f1 f2 f3*)
apply (*unfold R-def*)
apply (*rule inflationary-onI*)
apply (*subst* (*asm*) *in-codom-Frel-eq-Fpred-in-codom*)
apply (*erule FpredE*)
apply (*subst Frel-Fmap-eq2*)
apply (*rule Frel-refl-strong*)
apply (*rule inflationary-onD*[**where** *?R=R1*] *inflationary-onD*[**where** *?R=R2*]
inflationary-onD[**where** *?R=R3*],
rule assms,
assumption+)+
done

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*
defines *R* ≡ *Frel R1 R2 R3*
shows *inflationary-on* (*in-field R*) *R* (*Fmap f1 f2 f3*)
apply (*unfold R-def*)
apply (*subst in-field-eq-in-dom-sup-in-codom*)
apply (*subst inflationary-on-sup-eq*)
apply (*unfold inf-apply*)
apply (*subst inf-bool-def*)
apply (*rule conjI*;
rule inflationary-on-in-dom-FrelI inflationary-on-in-codom-FrelI;
rule inflationary-on-if-le-pred-if-inflationary-on,
rule assms,
rule le-predI,
rule in-field-if-in-dom in-field-if-in-codom,
assumption)
done

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*
defines *R* ≡ *Frel R1 R2 R3*
shows *deflationary-on* (*in-dom R*) *R* (*Fmap f1 f2 f3*)
apply (*unfold R-def*)
apply (*subst deflationary-on-eq-inflationary-on-rel-inv*)
apply (*subst in-codom-rel-inv-eq-in-dom*[*symmetric*])
apply (*unfold Frel-rel-inv-eq-rel-inv-Frel*[*symmetric*])
apply (*rule inflationary-on-in-codom-FrelI*;
subst deflationary-on-eq-inflationary-on-rel-inv[*symmetric*],
subst in-codom-rel-inv-eq-in-dom,
rule assms)
done

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*
  defines *R* ≡ *Frel R1 R2 R3*
  shows *deflationary-on* (*in-codom R*) *R* (*Fmap f1 f2 f3*)
  apply (*unfold R-def*)
  apply (*subst deflationary-on-eq-inflationary-on-rel-inv*)
  apply (*subst in-dom-rel-inv-eq-in-codom*[*symmetric*])
  apply (*unfold Frel-rel-inv-eq-rel-inv-Frel*[*symmetric*])
  apply (*rule inflationary-on-in-dom-FrelI*;
    subst deflationary-on-eq-inflationary-on-rel-inv[*symmetric*],
    subst in-dom-rel-inv-eq-in-codom,
    rule assms)
  done

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*
  defines *R* ≡ *Frel R1 R2 R3*
  shows *deflationary-on* (*in-field R*) *R* (*Fmap f1 f2 f3*)
  apply (*unfold R-def*)
  apply (*subst deflationary-on-eq-inflationary-on-rel-inv*)
  apply (*subst in-field-rel-inv-eq*[*symmetric*])
  apply (*unfold Frel-rel-inv-eq-rel-inv-Frel*[*symmetric*])
  apply (*rule inflationary-on-in-field-FrelI*;
    subst deflationary-on-eq-inflationary-on-rel-inv[*symmetric*],
    subst in-field-rel-inv-eq,
    rule assms)
  done

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*
  defines *R* ≡ *Frel R1 R2 R3*
  shows *rel-equivalence-on* (*in-field R*) *R* (*Fmap f1 f2 f3*)
  apply (*unfold R-def*)
  apply (*subst rel-equivalence-on-eq*)
  apply (*unfold inf-apply*)
  apply (*subst inf-bool-def*)
  apply (*insert assms*)
  apply (*elim rel-equivalence-onE*)
  apply (*rule conjI*;
    rule inflationary-on-in-field-FrelI deflationary-on-in-field-FrelI;
    assumption)
  done

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* ≡ ε **and** *flip.t1.unit* ≡ ε_1
  and *flip.t2.unit* ≡ ε_2 **and** *flip.t3.unit* ≡ ε_3
  by (*simp-all only*: *order-functors.flip-counit-eq-unit*)

lemma *order-equivalenceI*:
  assumes $((\le_{L1}) \equiv_o (\le_{R1}))$ *l1 r1*
  and $((\le_{L2}) \equiv_o (\le_{R2}))$ *l2 r2*
  and $((\le_{L3}) \equiv_o (\le_{R3}))$ *l3 r3*
  shows $((\le_{L}) \equiv_o (\le_{R}))$ *l r*
  apply (*insert assms*)
  apply (*elim order-functors.order-equivalenceE*)
  apply (*rule order-equivalenceI*;
    rule mono-wrt-rel-leftI
      flip.mono-wrt-rel-leftI
      rel-equivalence-on-in-field-unitI
      flip.rel-equivalence-on-in-field-unitI;
    assumption)
  done

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 **.**

theorem *preorder-equivalenceI*:
  assumes $((\le_{L1}) \equiv_{pre} (\le_{R1}))$ *l1 r1*
  and $((\le_{L2}) \equiv_{pre} (\le_{R2}))$ *l2 r2*
  and $((\le_{L3}) \equiv_{pre} (\le_{R3}))$ *l3 r3*
  shows $((\le_{L}) \equiv_{pre} (\le_{R}))$ *l r*
  apply (*insert assms*)
  apply (*elim transport.preorder-equivalence-galois-equivalenceE*)
  apply (*intro preorder-equivalence-if-galois-equivalenceI*
    galois-equivalenceI
    preorder-on-in-field-leftI flip.preorder-on-in-field-leftI)
  apply *assumption+*
  done

theorem *partial-equivalence-rel-equivalenceI*:
  assumes $((\le_{L1}) \equiv_{PER} (\le_{R1}))$ *l1 r1*
  and $((\le_{L2}) \equiv_{PER} (\le_{R2}))$ *l2 r2*
  and $((\le_{L3}) \equiv_{PER} (\le_{R3}))$ *l3 r3*
  shows $((\le_{L}) \equiv_{PER} (\le_{R}))$ *l r*
  apply (*insert assms*)
  apply (*elim transport.partial-equivalence-rel-equivalenceE*
    transport.preorder-equivalence-galois-equivalenceE
    preorder-on-in-fieldE)
  apply (*intro partial-equivalence-rel-equivalence-if-galois-equivalenceI*
    galois-equivalenceI
    partial-equivalence-rel-leftI flip.partial-equivalence-rel-leftI
    partial-equivalence-relI)
  apply *assumption+*
  done

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

2.11 Transport for Dependent Function Relator with Non-Dependent Functions

theory *Transport-Rel-If*

imports
Transport
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)
using *assms* **by** (*intro reflexive-onI*) *blast*

lemma *transitive-on-rel-if-if-transitive-onI* [*intro*]:
assumes $B \Longrightarrow$ *transitive-on* P R
shows *transitive-on* P (*rel-if* B R)
using *assms* **by** (*intro transitive-onI*) (*blast dest*: *transitive-onD*)

lemma *preorder-on-rel-if-if-preorder-onI* [*intro*]:
assumes $B \Longrightarrow$ *preorder-on* P R
shows *preorder-on* P (*rel-if* B R)
using *assms* **by** (*intro preorder-onI*) *auto*

lemma *symmetric-on-rel-if-if-symmetric-onI* [*intro*]:
assumes $B \Longrightarrow$ *symmetric-on* P R
shows *symmetric-on* P (*rel-if* B R)
using *assms* **by** (*intro symmetric-onI*) (*blast dest*: *symmetric-onD*)

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)
using *assms* **by** (*intro partial-equivalence-rel-onI*)
auto

lemma *rel-if-dep-mono-wrt-rel-if-iff-if-dep-mono-wrt-relI*:
assumes $B \Longrightarrow B' \Longrightarrow ([x\ y :: R] \Rrightarrow_m S\ x\ y)\ f$
and $B \longleftrightarrow B'$
shows $([x\ y :: (\textit{rel-if}\ B\ R)] \Rrightarrow_m (\textit{rel-if}\ B'\ (S\ x\ y)))\ f$
using *assms* **by** (*intro dep-mono-wrt-relI*) *auto*

end

corollary *reflexive-rel-if-if-reflexiveI* [*intro*]:
assumes $B \Longrightarrow$ *reflexive* R
shows *reflexive* (*rel-if* B R)
using *assms* **unfolding** *reflexive-eq-reflexive-on* **by** *blast*

corollary *transitive-rel-if-if-transitiveI* [*intro*]:
  assumes $B \Longrightarrow$ *transitive* R
  shows *transitive* (*rel-if* B R)
  using *assms* **unfolding** *transitive-eq-transitive-on* **by** *blast*

corollary *preorder-rel-if-if-preorderI* [*intro*]:
  assumes $B \Longrightarrow$ *preorder* R
  shows *preorder* (*rel-if* B R)
  using *assms* **unfolding** *preorder-eq-preorder-on* **by** *blast*

corollary *symmetric-rel-if-if-symmetricI* [*intro*]:
  assumes $B \Longrightarrow$ *symmetric* R
  shows *symmetric* (*rel-if* B R)
  using *assms* **unfolding** *symmetric-eq-symmetric-on* **by** *blast*

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)
  using *assms* **unfolding** *partial-equivalence-rel-eq-partial-equivalence-rel-on*
  by *blast*

context *galois-prop*
begin

interpretation *rel-if* : *galois-prop rel-if* B $(\le_L)$ *rel-if* B' $(\le_R)$ l r **.**
interpretation *flip-inv* : *galois-prop* $(\ge_R)$ $(\ge_L)$ r l **.**

lemma *rel-if-half-galois-prop-left-if-iff-if-half-galois-prop-leftI*:
  assumes $B \Longrightarrow B' \Longrightarrow ((\le_L)\ {}_h\trianglelefteq\ (\le_R))\ l\ r$
  and $B \longleftrightarrow B'$
  shows $((\mathit{rel\text{-}if}\ B\ (\le_L))\ {}_h\trianglelefteq\ (\mathit{rel\text{-}if}\ B'\ (\le_R)))\ l\ r$
  using *assms* **by** (*intro rel-if.half-galois-prop-leftI*) *auto*

lemma *rel-if-half-galois-prop-right-if-iff-if-half-galois-prop-rightI*:
  assumes $B \Longrightarrow B' \Longrightarrow ((\le_L)\ \trianglelefteq_h\ (\le_R))\ l\ r$
  and $B \longleftrightarrow B'$
  shows $((\mathit{rel\text{-}if}\ B\ (\le_L))\ \trianglelefteq_h\ (\mathit{rel\text{-}if}\ B'\ (\le_R)))\ l\ r$
  using *assms* **by** (*intro rel-if.half-galois-prop-rightI*) *fastforce*

lemma *rel-if-galois-prop-if-iff-if-galois-propI*:
  assumes $B \Longrightarrow B' \Longrightarrow ((\le_L)\ \trianglelefteq\ (\le_R))\ l\ r$
  and $B \longleftrightarrow B'$
  shows $((\mathit{rel\text{-}if}\ B\ (\le_L))\ \trianglelefteq\ (\mathit{rel\text{-}if}\ B'\ (\le_R)))\ l\ r$
  using *assms* **by** (*intro rel-if.galois-propI*
    rel-if-half-galois-prop-left-if-iff-if-half-galois-prop-leftI
    rel-if-half-galois-prop-right-if-iff-if-half-galois-prop-rightI)
  auto

end

context *galois*
begin

interpretation *rel-if* : *galois rel-if* B $(\leq_L)$ *rel-if* B' $(\leq_R)$ l r **.**

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 $((\textit{rel-if}\ B\ (\leq_L)) \dashv (\textit{rel-if}\ B'\ (\leq_R)))\ l\ r$
  using *assms* **by** (*intro rel-if.galois-connectionI*
    rel-if-dep-mono-wrt-rel-if-iff-if-dep-mono-wrt-relI
    rel-if-galois-prop-if-iff-if-galois-propI)
  auto

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 $((\textit{rel-if}\ B\ (\leq_L)) \equiv_G (\textit{rel-if}\ B'\ (\leq_R)))\ l\ r$
  using *assms* **by** (*intro rel-if.galois-equivalenceI*
    rel-if-galois-connection-if-iff-if-galois-connectionI
    galois-prop.rel-if-galois-prop-if-iff-if-galois-propI)
  (*auto elim*: *galois.galois-connectionE*)

end

context *transport*
begin

interpretation *rel-if* : *transport rel-if* B $(\leq_L)$ *rel-if* B' $(\leq_R)$ l r **.**

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 $((\textit{rel-if}\ B\ (\leq_L)) \equiv_{pre} (\textit{rel-if}\ B'\ (\leq_R)))\ l\ r$
  using *assms* **by** (*intro rel-if.preorder-equivalence-if-galois-equivalenceI*
    rel-if-galois-equivalence-if-iff-if-galois-equivalenceI
    preorder-on-rel-if-if-preorder-onI)
  blast+

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 $((\textit{rel-if}\ B\ (\leq_L)) \equiv_{PER} (\textit{rel-if}\ B'\ (\leq_R)))\ l\ r$
  using *assms* **by** (*intro*
    rel-if.partial-equivalence-rel-equivalence-if-galois-equivalenceI
    rel-if-galois-equivalence-if-iff-if-galois-equivalenceI)
  blast+

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* $\Rightarrow$ *'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* :: *'b1* $\Rightarrow$ *'b2*
and *r2* :: *'b2* $\Rightarrow$ *'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** $\leq_L$ *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.unit* (η)
notation *tdfr.counit* (ε)

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 $([x1\ x2 :: (\geq_{L1})] \Rrightarrow_m [x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3] \Rrightarrow (\leq))$ *L2*
and $([x1'\ x2' :: (\geq_{R1})] \Rrightarrow_m [x3'\ x4' :: (\leq_{R1}) \mid x1' \leq_{R1} x3'] \Rrightarrow (\leq))$ *R2*
shows $((\leq_L) \equiv_{PER} (\leq_R))$ *l r*
proof −
have *per2I*: $((\leq_{L2\ x1\ (r1\ x2')}) \equiv_{PER} (\leq_{R2\ (l1\ x1)\ x2'}))$ *l2 r2*
if *hyps*: $x1 \leq_{L1} x2$ $x2\ {}_{L1}\lessapprox\ x1'$ $x1' \leq_{R1} x2'$ **for** *x1 x2 x1' x2'*
proof −
from *hyps* **have** $x1\ {}_{L1}\lessapprox\ x2'$
using *per-equiv1 t1.left-Galois-if-left-Galois-if-left-relI*
t1.left-Galois-if-right-rel-if-left-GaloisI
by *fast*
with *per-equiv2* **show** *?thesis* **by** *blast*

qed
have $([x1'\ x2' :: (\le_{R1})] \Rrightarrow_m [x1\ x2 :: (\le_{L1}) \mid x2\ {}_{L1}\lessapprox x1'] \Rrightarrow$
$[in\text{-}field\ (\le_{L2\ x1\ (r1\ x2')})] \Rrightarrow (\le_{R2\ (l1\ x1)\ x2'}))\ (\lambda\text{- -. } l2)$
by (*intro dep-mono-wrt-relI Dep-Fun-Rel-relI Dep-Fun-Rel-predI rel-if-if-impI*)
(*auto 8 0 dest*!: *per2I*)
moreover have
$([x1\ x2 :: (\le_{L1})] \Rrightarrow_m [x1'\ x2' :: (\le_{R1}) \mid x2\ {}_{L1}\lessapprox x1'] \Rrightarrow$
$[in\text{-}field\ (\le_{R2\ (l1\ x1)\ x2'})] \Rrightarrow (\le_{L2\ x1\ (r1\ x2')}))\ (\lambda\text{- -. } r2)$
by (*intro dep-mono-wrt-relI Dep-Fun-Rel-relI Dep-Fun-Rel-predI rel-if-if-impI*)
(*auto 8 0 dest*!: *per2I*)
ultimately show *?thesis*
using *assms* **by** (*intro tdfr.partial-equivalence-rel-equivalenceI*) *auto*
qed

end

end

2.12 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 $[\![transport.partial\text{-}equivalence\text{-}rel\text{-}equivalence\ ?L1.0\ ?R1.0\ ?l1.0\ ?r1.0;\ \bigwedge x\ x'.\ galois\text{-}rel.Galois\ ?L1.0\ ?R1.0\ ?r1.0\ x\ x' \Longrightarrow transport.partial\text{-}equivalence\text{-}rel\text{-}equivalence\ (?L2.0\ x\ (?r1.0\ x'))\ (?R2.0\ (?l1.0\ x)\ x')\ ?l2.0\ ?r2.0;\ ([x1\ x2 :: ?L1.0^{-1}] \Rrightarrow_m [x3\ x4 :: ?L1.0] \Rrightarrow ?L1.0\ x1\ x3 \longrightarrow (\le))\ ?L2.0;\ ([x1'\ x2' :: ?R1.0^{-1}] \Rrightarrow_m [x3'\ x4' :: ?R1.0] \Rrightarrow ?R1.0\ x1'\ x3' \longrightarrow (\le))\ ?R2.0]\!] \Longrightarrow transport.partial\text{-}equivalence\text{-}rel\text{-}equivale$ $(transport\text{-}Dep\text{-}Fun\text{-}Rel.L\ ?L1.0\ ?L2.0)\ (transport\text{-}Dep\text{-}Fun\text{-}Rel.L\ ?R1.0\ ?R2.0)\ (transport\text{-}Dep\text{-}Fun\text{-}Rel.l\ ?r1.0\ (\lambda\text{- -. } ?l2.0))\ (transport\text{-}Dep\text{-}Fun\text{-}Rel.l\ ?l1.0\ (\lambda\text{- -. } ?r2.0))$: 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 $((\le_L) \equiv_{PER} (\le_R))\ l\ r$
  and $x \le_L x'$
  shows $x\ {}_{L}\lessapprox\ l\ x$
  using *assms* **by** (*intro left-Galois-left-if-left-rel-if-inflationary-on-in-fieldI*)
  (*blast elim*: *preorder-equivalence-order-equivalenceE*)+

definition *transport-per* $x\ y \equiv ((\le_L) \equiv_{PER} (\le_R))\ l\ r \wedge x\ {}_{L}\lessapprox\ y$

The choice of x' is arbitrary. All we need is *in-dom* $(\le_L)$ x.

lemma *transport-per-start*:
  assumes $((\le_L) \equiv_{PER} (\le_R))\ l\ r$
  and $x \le_L x'$
  shows *transport-per* x $(l\ x)$
  using *assms* **unfolding** *transport-per-def*
  by (*blast intro*: *left-Galois-left-if-left-rel-if-partial-equivalence-rel-equivalence*)

lemma *left-Galois-if-transport-per*:
  assumes *transport-per* $x\ y$
  shows $x\ {}_{L}\lessapprox\ y$
  using *assms* **unfolding** *transport-per-def* **by** *blast*

end

context *transport-Fun-Rel*
begin

Simplification of Galois relator for simple function relator.

corollary *left-Galois-eq-Fun-Rel-left-Galois*:
  assumes $((\le_{L1}) \equiv_{PER} (\le_{R1}))\ l1\ r1$
  and $((\le_{L2}) \equiv_{PER} (\le_{R2}))\ l2\ r2$
  shows $({}_{L}\lessapprox) = (({}_{L1}\lessapprox) \Rrightarrow ({}_{L2}\lessapprox))$
proof (*intro ext*)
  fix $f\ g$
  show $f\ {}_{L}\lessapprox\ g \longleftrightarrow (({}_{L1}\lessapprox) \Rrightarrow ({}_{L2}\lessapprox))\ f\ g$
  proof
    assume $f\ {}_{L}\lessapprox\ g$
    moreover have $g \le_R g$
    proof −
      from *assms* **have** *per*: $((\le_L) \equiv_{PER} (\le_R))\ l\ r$
        by (*intro partial-equivalence-rel-equivalenceI*) *auto*
      with ‹$f\ {}_{L}\lessapprox\ g$› **show** *?thesis* **by** *blast*
    qed
    ultimately show $(({}_{L1}\lessapprox) \Rrightarrow ({}_{L2}\lessapprox))\ f\ g$ **using** *assms*
      by (*intro Fun-Rel-left-Galois-if-left-GaloisI*)
      (*auto elim*!: *tdfrs.t1.partial-equivalence-rel-equivalenceE*

tdfrs.t1.preorder-equivalence-galois-equivalenceE
tdfrs.t1.galois-equivalenceE
intro: *reflexive-on-if-le-pred-if-reflexive-on in-field-if-in-dom*)
next
assume $((_{L1}\lessapprox) \Rrightarrow (_{L2}\lessapprox))\ f\ g$
with *assms* **have** $((_{L1}\lessapprox) \Rrightarrow (_{L2}\lessapprox))\restriction_{in\text{-}dom\ (\le_L)}\upharpoonleft_{in\text{-}codom\ (\le_R)}\ f\ g$
by (*subst Fun-Rel-left-Galois-restrict-left-right-eq-Fun-Rel-left-GaloisI*) *blast+*
with *assms* **show** $f\ _{L}\lessapprox\ g$
by (*intro left-Galois-if-Fun-Rel-left-GaloisI*) *blast+*
qed
qed

end

lemmas *related-Fun-Rel-combI* = *Dep-Fun-Rel-relD*[**where** *?S*=λ- -. *S* **for** *S*, *rotated*]
lemma *related-Fun-Rel-lambdaI*:
assumes $\bigwedge x\ y.\ R\ x\ y \Longrightarrow S\ (f\ x)\ (g\ y)$
and $T = (R \Rrightarrow S)$
shows $T\ f\ g$
using *assms* **by** *blast*

General ML setups **ML-file**‹*transport-util.ML*›

Unification Setup **ML**‹

```
@{functor-instance struct-name = Transport-Unification-Combine
  and functor-name = Unification-Combine
  and id = Transport-Util.transport-id}
›
```

local-setup ‹*Transport-Unification-Combine.setup-attribute NONE*›
ML‹

```
@{functor-instance struct-name = Transport-Mixed-Unification
  and functor-name = Mixed-Unification
  and id = Transport-Util.transport-id
  and more-args = ‹structure UC = Transport-Unification-Combine›}
›
```

ML‹

```
@{functor-instance struct-name = Transport-Unification-Hints
  and functor-name = Term-Index-Unification-Hints
  and id = Transport-Util.transport-id
  and more-args = ‹
    structure TI = Discrimination-Tree
    val init-args = {
      concl-unifier = SOME Higher-Order-Pattern-Unification.unify,
    normalisers = SOME Transport-Mixed-Unification.norms-first-higherp-first-comb-higher-unify,
    prems-unifier = SOME (Transport-Mixed-Unification.first-higherp-first-comb-higher-unify
      |> Unification-Combinator.norm-unifier Envir-Normalisation.beta-norm-term-unif),
      retrieval = SOME (Term-Index-Unification-Hints-Args.mk-sym-retrieval
```

TI.norm-term TI.unifiables),
hint-preprocessor = *SOME* (*K I*)
}›}
›
local-setup ‹*Transport-Unification-Hints.setup-attribute NONE*›
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*
(#*norm-term Transport-Mixed-Unification.norms-first-higherp-first-comb-higher-unify*)
|> *K*)
(*Transport-Unification-Combine.default-metadata Transport-Unification-Hints.binding*)›]]

Prototype **ML-file**‹*transport.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

2.13 Syntax Bundles for Transport

theory *Transport-Syntax*
imports
Transport
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* ('((-)$\lessapprox$($_{(-)\ (-)}$)'))

notation *Galois-infix* ((-) (-)$\lessapprox_{((-)\ (-))}$ (-) [*51*,*51*,*51*,*51*,*51*] *50*)
notation *ge-Galois* ('(($_{(-)\ (-)}$)$\gtrapprox$(-)'))
notation *ge-Galois-infix* ((-) ($_{(-)\ (-)}$)$\gtrapprox$(-) (-) [*51*,*51*,*51*,*51*,*51*] *50*)
end
bundle *no-galois-rel-syntax*
begin
no-notation *galois-rel.Galois* ('((-)$\lessapprox_{((-)\ (-))}$'))
no-notation *Galois-infix* ((-) (-)$\lessapprox_{((-)\ (-))}$ (-) [*51*,*51*,*51*,*51*,*51*] *50*)
no-notation *ge-Galois* ('(($_{(-)\ (-)}$)$\gtrapprox$(-)'))
no-notation *ge-Galois-infix* ((-) ($_{(-)\ (-)}$)$\gtrapprox$(-) (-) [*51*,*51*,*51*,*51*,*51*] *50*)
end

bundle *transport-syntax*
begin
notation *transport.preorder-equivalence* (**infix** $\equiv_{pre}$ *50*)
notation *transport.partial-equivalence-rel-equivalence* (**infix** $\equiv_{PER}$ *50*)
end
bundle *no-transport-syntax*
begin
no-notation *transport.preorder-equivalence* (**infix** $\equiv_{pre}$ *50*)
no-notation *transport.partial-equivalence-rel-equivalence* (**infix** $\equiv_{PER}$ *50*)
end

end

2.14 Example Transports for Dependent Function Relator

theory *Transport-Dep-Fun-Rel-Examples*
imports
Transport-Prototype
Transport-Syntax
HOL−Library.IArray
begin

Summary Dependent function relator examples from [2]. Refer to the paper for more details.

context
includes *galois-rel-syntax 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* ‹*Conversion-Util.move-prems-to-front-conv* [*1*] |> *Conversion-Util.thm-conv*›,
ML-Krattr ‹*Conversion-Util.move-prems-to-front-conv* [*2*,*3*] |> *Conversion-Util.thm-conv*›,

per-intro]
begin

interpretation *transport L R l r* **for** *L R l r* **.**

abbreviation *Zpos* ≡ $((=_{(\leq)(0\ ::\ int)})$ :: *int* ⇒ *-*)

lemma *Zpos-per* [*per-intro*]: (*Zpos* $\equiv_{PER}$ (=)) *nat int*
  by *fastforce*

lemma *sub-parametric* [*trp-in-dom*]:
  ([*i* - :: *Zpos*] ⇛ [*j* - :: *Zpos* | *j* ≤ *i*] ⇛ *Zpos*) (−) (−)
  by *fastforce*

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*] ⇛ (=)

  by (*trp-prover*) *fastforce*+

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 *transp-eq-transitive*: *transp* = *transitive*
  by (*auto intro*: *transpI dest*: *transpD*)
lemma *symp-eq-symmetric*: *symp* = *symmetric*
  by (*auto intro*: *sympI dest*: *sympD symmetricD*)

lemma [*per-intro*]:
  assumes *partial-equivalence-rel R*
  shows (*LRel R* $\equiv_{PER}$ *IARel R*) *IArray.IArray IArray.list-of*
  using *assms* **by** (*fastforce simp flip*: *transp-eq-transitive symp-eq-symmetric*
    intro: *list.rel-transp list.rel-symp iarray.rel-transp iarray.rel-symp*
    elim: *iarray.rel-cases*)+

lemma [*trp-in-dom*]:
  ([*xs* - :: *LRel R*] ⇛ [*i* - :: (=) | *i* < *length xs*] ⇛ *R*) (!) (!)
  by (*fastforce simp*: *list-all2-lengthD list-all2-nthD2*)

context
  fixes *R* :: ′*a* ⇒ - **assumes** [*per-intro*]: *partial-equivalence-rel R*
begin

interpretation *Rper* : *transport-partial-equivalence-rel-id R*
  by *unfold-locales per-prover*

declare *Rper.partial-equivalence-rel-equivalence* [*per-intro*]

trp-term *iarray-index* **where** $x = (!) :: 'a\ list \Rightarrow$ -
  and $L = ([xs\ \text{-} :: LRel\ R] \Rrightarrow [i\ \text{-} :: (=) \mid i < length\ xs] \Rrightarrow R)$
  and $R = ([xs\ \text{-} :: IARel\ R] \Rrightarrow [i\ \text{-} :: (=) \mid i < IArray.length\ xs] \Rrightarrow R)$
  by (*trp-prover*)

  (*fastforce simp*: *list-all2-lengthD elim*: *iarray.rel-cases*)+

end
end

end

2.15 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 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* ⇒ -) ≡ $(=_{finite\ ::\ 'a\ set\ \Rightarrow\ bool})$

interpretation *t* : *transport LSL R l r* **for** *LSL R l r* **.**

Proofs of equivalences.

lemma *list-fset-PER* [*per-intro*]: $(LFSL \equiv_{PER} LFSR)$ *fset-of-list sorted-list-of-fset*
  unfolding *LFSL-def* **by** *fastforce*

lemma *list-set-PER* [*per-intro*]: $(LSL \equiv_{PER} LSR)$ *set sorted-list-of-set*
  unfolding *LSL-def* **by** *fastforce*

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*: $({}_{LFSL}\lessapprox_{LFSR\ sorted\text{-}list\text{-}of\text{-}fset}) \equiv LFS$
  unfolding *LFS-def LFSL-def* **by** (*intro eq-reflection ext*) (*auto*)
lemma *LFSR-Galois-eq-inv-LFS*: $({}_{LFSR}\lessapprox_{LFSL\ fset\text{-}of\text{-}list}) \equiv LFS^{-1}$
  unfolding *LFS-def LFSL-def* **by** (*intro eq-reflection ext*) (*auto*)
lemma *LSL-Galois-eq-LS*: $({}_{LSL}\lessapprox_{LSR\ sorted\text{-}list\text{-}of\text{-}set}) \equiv LS$
  unfolding *LS-def LSL-def* **by** (*intro eq-reflection ext*) (*auto*)

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* ∈ *set xs*
  shows *max x* (*max-list* (*removeAll x xs*)) = *max-list xs*
  unfolding *max-list-def* **using** *assms* **by** (*induction xs*)
  (*simp-all*, (*metis max.left-idem removeAll-id max.left-commute*)+)

lemma *max-list-parametric* [*trp-in-dom*]: (*LSL* ⇛ (=)) *max-list max-list*
proof (*intro Dep-Fun-Rel-relI*)
  fix *xs xs′* :: *nat list* **assume** *LSL xs xs′*
  then have *finite* (*set xs*) *set xs* = *set xs′* **unfolding** *LSL-def* **by** *auto*
  then show *max-list xs* = *max-list xs′*
  proof (*induction set xs arbitrary*: *xs xs′ rule*: *finite-induct*)
    case (*insert x F*)
    then have *F* = *set* (*removeAll x xs*) **by** *auto*
    moreover from *insert* **have** ... = *set* (*removeAll x xs′*) **by** *auto*
    ultimately have *max-list* (*removeAll x xs*) = *max-list* (*removeAll x xs′*)
      (**is** *?lhs* = *?rhs*) **using** *insert* **by** *blast*
    then have *max x ?lhs* = *max x ?rhs* **by** *simp*
    then show *?case*
      using *insert max-max-list-removeAll-eq-maxlist insertI1* **by** *metis*
  qed *auto*
qed

lemma *LFSL-eq-LSL*: *LFSL* ≡ *LSL*
 unfolding *LFSL-def LSL-def* **by** (*intro eq-reflection ext*) (*auto simp*: *fset-of-list-elem*)

lemma *max-list-parametricfin* [*trp-in-dom*]: (*LFSL* ⇛ (=)) *max-list max-list*
  using *max-list-parametric* **by** (*simp only*: *LFSL-eq-LSL*)

Transport from lists to finite sets.

trp-term *max-fset* :: *nat fset* ⇒ *nat* **where** x = *max-list*
and L = (*LFSL* ⇛ (=))
by *trp-prover*

Use **print-theorems** to show all theorems. Here's the correctness theorem:

lemma (*LFS* ⇛ (=)) *max-list max-fset* **by** (*trp-hints-resolve max-fset-related′*)

lemma [*trp-in-dom*]: (*LFSR* ⇛ (=)) *max-fset max-fset* **by** *simp*

Transport from lists to sets.

trp-term *max-set* :: *nat set* ⇒ *nat* **where** x = *max-list*
by *trp-prover*

lemma (*LS* ⇛ (=)) *max-list max-set* **by** (*trp-hints-resolve max-set-related′*)

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*
by (*subst transport.partial-equivalence-rel-equivalence-right-left-iff-partial-equivalence-rel-equivalence-left-righ*
(*fact list-fset-PER*)

Transport from finite sets to lists.

trp-term *max-list′* :: *nat list* ⇒ *nat* **where** x = *max-fset*
by *trp-prover*

lemma (LFS^{-1} ⇛ (=)) *max-fset max-list′* **by** (*trp-hints-resolve max-list′-related′*)

Transporting higher-order functions.

lemma *map-parametric* [*trp-in-dom*]:
(((=) ⇛ (=)) ⇛ *LSL* ⇛ *LSL*) *map map*
unfolding *LSL-def* **by** (*intro Dep-Fun-Rel-relI*) *simp*

lemma [*trp-uhint*]: $P \equiv (=) \Longrightarrow P \equiv (=)$ ⇛ (=) **by** *simp*
lemma [*trp-uhint*]: $P \equiv \top \Longrightarrow (=_{P\ ::\ 'a\ \Rightarrow\ bool}) \equiv ((=) :: 'a \Rightarrow \text{-})$ **by** *simp*

trp-term *map-set* :: (′*a* :: *linorder* ⇒ ′*b*) ⇒ ′*a set* ⇒ (′*b* :: *linorder*) *set*
where x = *map* :: (′*a* :: *linorder* ⇒ ′*b*) ⇒ ′*a list* ⇒ (′*b* :: *linorder*) *list*
by *trp-prover*

lemma (((=) ⇛ (=)) ⇛ *LS* ⇛ *LS*) *map map-set* **by** (*trp-hints-resolve map-set-related′*)

lemma *filter-parametric* [*trp-in-dom*]:
(((=) ⇛ (⟷)) ⇛ *LSL* ⇛ *LSL*) *filter filter*
unfolding *LSL-def* **by** (*intro Dep-Fun-Rel-relI*) *simp*

trp-term *filter-set* :: $('a :: linorder \Rightarrow bool) \Rightarrow 'a\ set \Rightarrow 'a\ set$
  where x = *filter* :: $('a :: linorder \Rightarrow bool) \Rightarrow 'a\ list \Rightarrow 'a\ list$
  by *trp-prover*

lemma $(((=) \Rrightarrow (=)) \Rrightarrow LS \Rrightarrow LS)$ *filter filter-set* **by** (*trp-hints-resolve filter-set-related'*)

lemma *append-parametric* [*trp-in-dom*]:
  $(LSL \Rrightarrow LSL \Rrightarrow LSL)$ *append append*
  unfolding *LSL-def* **by** (*intro Dep-Fun-Rel-relI*) *simp*

trp-term *append-set* :: $('a :: linorder)\ set \Rightarrow 'a\ set \Rightarrow 'a\ set$
  where x = *append* :: $('a :: linorder)\ list \Rightarrow 'a\ list \Rightarrow 'a\ list$
  by *trp-prover*

lemma $(LS \Rrightarrow LS \Rrightarrow LS)$ *append append-set* **by** (*trp-hints-resolve append-set-related'*)

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')
  by (*fact append-set-app-eq*)

lemma *finite* $s \Longrightarrow$ *finite* $s' \Longrightarrow$ *append-set* $s\ s' = s \cup s'$
  by (*auto simp*: *append-set-app-eq*)

end

end

2.16 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* $L\ (=)\ l\ r$ **.**

lemma *Quotient-T-eq-Galois*:
  assumes *Quotient* $(\le_L)\ l\ r\ T$
  shows $T = t.Galois$
proof (*intro ext iffI*)
  fix $x\ y$ **assume** $T\ x\ y$

with *assms* **have** $x \le_L x$ $l\ x = y$ **using** *Quotient-cr-rel* **by** *auto*
with *assms* **have** $r\ (l\ x) \le_L x$ $r\ (l\ x) \le_L r\ y$
using *Quotient-rep-abs Quotient-rep-reflp* **by** *auto*
with *assms* **have** $x \le_L r\ y$ **using** *Quotient-part-equivp*
by (*blast elim*: *part-equivpE dest*: *transpD sympD*)
then show *t.Galois* $x\ y$ **by** *blast*
next
fix $x\ y$ **assume** *t.Galois* $x\ y$
with *assms* **show** $T\ x\ y$ **using** *Quotient-cr-rel Quotient-refl1 Quotient-symp*
by (*fastforce intro*: *Quotient-rel-abs2*[*symmetric*] *dest*: *sympD*)
qed

lemma *Quotient-if-preorder-equivalence*:
assumes $((\le_L) \equiv_{pre} (=))\ l\ r$
shows *Quotient* $(\le_L)\ l\ r$ *t.Galois*
proof (*rule QuotientI*)
from *assms* **show** *g2*: $l\ (r\ y) = y$ **for** y **by** *fastforce*
from *assms* **show** $r\ y \le_L r\ y$ **for** y **by** *blast*
show *g1*: $x \le_L x' \longleftrightarrow x \le_L x \wedge x' \le_L x' \wedge l\ x = l\ x'$
(**is** *?lhs* $\longleftrightarrow$ *?rhs*) **for** $x\ x'$
proof (*rule iffI*)
assume *?rhs*
with *assms* **have** $\eta\ x \le_L \eta\ x'$ **by** *fastforce*
moreover from ‹*?rhs*› *assms* **have** $x \le_L \eta\ x$ $\eta\ x' \le_L x'$
by (*blast elim*: *t.preorder-equivalence-order-equivalenceE*)+
moreover from *assms* **have** *transitive* $(\le_L)$ **by** *blast*
ultimately show $x \le_L x'$ **by** *blast*
next
assume *?lhs*
with *assms* **show** *?rhs* **by** *blast*
qed
from *assms* **show** *t.Galois* $= (\lambda x\ y.\ x \le_L x \wedge l\ x = y)$
by (*intro ext iffI*)
(*metis g1 g2 t.left-GaloisE*,
auto intro!: *t.left-Galois-left-if-left-rel-if-inflationary-on-in-fieldI*
elim!: *t.preorder-equivalence-order-equivalenceE*)
qed

lemma *partial-equivalence-rel-equivalence-if-Quotient*:
assumes *Quotient* $(\le_L)\ l\ r\ T$
shows $((\le_L) \equiv_{PER} (=))\ l\ r$
proof (*rule t.partial-equivalence-rel-equivalence-if-order-equivalenceI*)
from *Quotient-part-equivp*[*OF assms*] **show** *partial-equivalence-rel* $(\le_L)$
by (*blast elim*: *part-equivpE dest*: *transpD sympD*)
have $x \equiv_L r\ (l\ x)$ **if** *in-field* $(\le_L)\ x$ **for** x
proof −
from *assms* ‹*in-field* $(\le_L)\ x$› **have** $x \le_L x$
using *Quotient-refl1 Quotient-refl2* **by** *fastforce*
with *assms Quotient-rep-abs Quotient-symp* **show** *?thesis*

by (*fastforce dest*: *sympD*)
qed
with *assms* **show** $((\le_L) \equiv_o (=))\ l\ r$
using *Quotient-abs-rep Quotient-rel-abs Quotient-rep-reflp Quotient-abs-rep*[*symmetric*]
by (*intro t.order-equivalenceI dep-mono-wrt-relI rel-equivalence-onI inflationary-onI deflationary-onI*)
auto
qed *auto*

corollary *Quotient-iff-partial-equivalence-rel-equivalence*:
Quotient $(\le_L)\ l\ r\ t.Galois \longleftrightarrow ((\le_L) \equiv_{PER} (=))\ l\ r$
using *Quotient-if-preorder-equivalence partial-equivalence-rel-equivalence-if-Quotient*
by *blast*

corollary *Quotient-T-eq-ge-Galois-right*:
assumes *Quotient* $(\le_L)\ l\ r\ T$
shows $T = t.ge\text{-}Galois\text{-}right$
using *assms*
by (*subst t.ge-Galois-right-eq-left-Galois-if-symmetric-if-in-codom-eq-in-dom-if-galois-prop*)
(*blast dest*: *partial-equivalence-rel-equivalence-if-Quotient*
intro: *in-codom-eq-in-dom-if-reflexive-on-in-field Quotient-T-eq-Galois*)+

end

end

2.17 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 $(galois\text{-}rel.Galois\ (=)\ (=)\ Rep)\restriction_{mem\text{-}of\ A}\upharpoonright_{\top\ ::\ 'b \Rightarrow bool} \equiv$
$(galois\text{-}rel.Galois\ (=)\ (=)\ Rep)$

using *Abs-inverse Rep-inverse*
by (*intro transport-eq-restrict-bijection.intro bijection-onI*)
(*auto simp*: *restrict-right-eq*
intro!: *eq-reflection galois-rel.left-GaloisI Rep*
elim: *galois-rel.left-GaloisE*)

interpretation *galois L R Abs Rep* **.**

lemma *Rep-left-Galois-self*: *Rep y* ${}_{L}\lessapprox$ *y*
using *Rep* **by** (*intro left-GaloisI*) *auto*

definition *AR x y* ≡ *x* = *Rep y*

lemma *left-Galois-eq-AR*: *left-Galois* = *AR*
unfolding *AR-def*
by (*auto intro*!: *galois-rel.left-GaloisI Rep elim*: *galois-rel.left-GaloisE*)

end

end

theory *Transport-Typedef*
imports
HOL−Library.FSet
Transport-Typedef-Base
Transport-Prototype
Transport-Syntax
begin

context
includes *galois-rel-syntax transport-syntax*
begin

typedef *pint* = {*i* :: *int*. *0* ≤ *i*} **by** *auto*

interpretation *typedef-pint* : *type-definition Rep-pint Abs-pint* {*i* :: *int*. *0* ≤ *i*}
by (*fact type-definition-pint*)

lemma [*trp-relator-rewrite*, *trp-uhint*]:
$\left({}_{(=_{Collect\ ((\leq)\ (0\ ::\ int))})}\lessapprox_{(=)\ Rep\text{-}pint}\right) \equiv$ *typedef-pint.AR*
using *typedef-pint.left-Galois-eq-AR* **by** (*intro eq-reflection*) *simp*

typedef *'a fset* = {*s* :: *'a set*. *finite s*} **by** *auto*

interpretation *typedef-fset* :
type-definition Rep-fset Abs-fset {*s* :: *'a set*. *finite s*}
by (*fact type-definition-fset*)

lemma [*trp-relator-rewrite*, *trp-uhint*]:
$((=_{\{s\ ::\ 'a\ set.\ finite\ s\}}) :: 'a\ set \Rightarrow \text{-} \overset{\lessapprox}{\approx}_{(=)\ Rep\text{-}fset}) \equiv typedef\text{-}fset.AR$
using *typedef-fset.left-Galois-eq-AR* **by** (*intro eq-reflection*) *simp*

lemma *eq-restrict-set-eq-eq-uhint* [*trp-uhint*]:
$P \equiv \lambda x.\ x \in A \Longrightarrow ((=_{A\ ::\ 'a\ set}) :: 'a \Rightarrow \text{-}) \equiv (=_P)$
by *simp*

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* **by** *auto*

trp-term *pint-one* :: *pint* **where** $x = 1$ :: *int*
by *trp-prover*

lemma *add-parametric* [*trp-in-dom*]:
(*typedef-pint.L* ⇛ *typedef-pint.L* ⇛ *typedef-pint.L*) (+) (+)
by (*intro Dep-Fun-Rel-relI*) (*auto intro*!: *eq-restrictI elim*!: *eq-restrictE*)

trp-term *pint-add* :: *pint* ⇒ *pint* ⇒ *pint*
where $x = (+)$ :: *int* ⇒ -
by *trp-prover*

lemma *empty-parametric* [*trp-in-dom*]: *typedef-fset.L* {} {}
by *auto*

trp-term *fempty* :: *'a fset* **where** $x = \{\}$ :: *'a set*
by *trp-prover*

lemma *insert-parametric* [*trp-in-dom*]:
((=) ⇛ *typedef-fset.L* ⇛ *typedef-fset.L*) *insert insert*
by *auto*

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*)
by *trp-prover*

context
notes *refl*[*trp-related-intro*]
begin

trp-term *insert-add-int-fset-whitebox* :: *int fset*
where x = *insert* (*1* + (*1* :: *int*)) {} !
by *trp-whitebox-prover*

lemma *empty-parametric′* [*trp-related-intro*]: (*rel-set R*) {} {}
by (*intro Dep-Fun-Rel-relI rel-setI*) (*auto dest*: *rel-setD1 rel-setD2*)

lemma *insert-parametric′* [*trp-related-intro*]:
($R \Rrightarrow$ *rel-set* $R \Rrightarrow$ *rel-set* R) *insert insert*
by (*intro Dep-Fun-Rel-relI rel-setI*) (*auto dest*: *rel-setD1 rel-setD2*)

context
assumes [*trp-uhint*]:

$L \equiv$ *rel-set* ($L1$ :: *int* $\Rightarrow$ *int* $\Rightarrow$ *bool*) $\Longrightarrow R \equiv$ *rel-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-set S*
begin

trp-term *insert-add-pint-set-whitebox* :: *pint set*
where x = *insert* (*1* + (*1* :: *int*)) {} !
by *trp-whitebox-prover*

print-statement *insert-add-int-fset-whitebox-def insert-add-pint-set-whitebox-def*

end
end

lemma *image-parametric* [*trp-in-dom*]:
$(((=) \Rrightarrow (=)) \Rrightarrow$ *typedef-fset.L* $\Rrightarrow$ *typedef-fset.L*) *image image*
by (*intro Dep-Fun-Rel-relI*) *auto*

trp-term *fimage* :: $('a \Rightarrow 'b) \Rightarrow 'a$ *fset* $\Rightarrow 'b$ *fset* **where** x = *image*
by *trp-prover*

lemma *rel-fun-eq-Fun-Rel-rel*: *rel-fun* = *Fun-Rel-rel*
by (*intro ext iffI Dep-Fun-Rel-relI*) (*auto elim*: *rel-funE*)

lemma *image-parametric′* [*trp-related-intro*]:
(($R \Rrightarrow S$) $\Rrightarrow$ *rel-set* $R \Rrightarrow$ *rel-set* S) *image image*
using *transfer-raw*[*simplified rel-fun-eq-Fun-Rel-rel Transfer.Rel-def*]
by *simp*

lemma *Galois-id-hint* [*trp-uhint*]:
(L :: $'a \Rightarrow 'a \Rightarrow$ *bool*) $\equiv R \Longrightarrow r \equiv id \Longrightarrow E \equiv L \Longrightarrow ({}_{L}\lessapprox_{R\ r}) \equiv E$
by (*simp only*: *eq-reflection*[*OF transport-id.left-Galois-eq-left*])

lemma *Freq* [*trp-uhint*]: $L \equiv (=) \Rrightarrow (=) \Longrightarrow L \equiv (=)$
  by *auto*

context
  fixes *L1 R1 l1 r1 L R l r*
  assumes *per1*: $(L1 \equiv_{PER} R1)$ *l1 r1*
  defines $L \equiv$ *rel-set L1* **and** $R \equiv$ *rel-set R1*
  and $l \equiv$ *image l1* **and** $r \equiv$ *image r1*
begin

interpretation *transport L R l r* **.**

context

  assumes *transport-per-set*: $((\leq_L) \equiv_{PER} (\leq_R))$ *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* $\Rightarrow$ *'b set*) *?r1 typedef-fset.L*
  and $R =$ *transport-comp.L typedef-fset.R typedef-fset.L ?r2 ?l2 ?R1*
  apply (*rule transport-comp.partial-equivalence-rel-equivalenceI*)
    apply (*rule transport-per-set*)
    apply *per-prover*
    apply (*fact compat*)
  apply (*rule transport-comp.left-relI*[**where** *?y*={} **and** *?y'*={}])
  apply (*unfold L-def R-def l-def r-def*)
  apply (*auto intro*!: *galois-rel.left-GaloisI in-codomI empty-transfer*)
  done

definition *set-succ* $\equiv$ *image* ((+) (*1* :: *int*))

lemma *set-succ-parametric* [*trp-in-dom*]:
  (*typedef-fset.L* $\Rrightarrow$ *typedef-fset.L*) *set-succ set-succ*
  unfolding *set-succ-def* **by** *auto*

trp-term *fset-succ* :: *int fset* $\Rightarrow$ *int fset*
  where $x =$ *set-succ*
  and $L =$ *typedef-fset.L* $\Rrightarrow$ *typedef-fset.L*
  and $R =$ *typedef-fset.R* $\Rrightarrow$ *typedef-fset.R*
  by *trp-prover*

trp-term *fset-succ'* :: *int fset* $\Rightarrow$ *int fset*
  where $x =$ *set-succ*
  and $L =$ *typedef-fset.L* $\Rrightarrow$ *typedef-fset.L*
  and $R =$ *typedef-fset.R* $\Rrightarrow$ *typedef-fset.R*
  unfold set-succ-def !

using *refl*[*trp-related-intro*]
apply (*tactic* ‹*Transport.instantiate-skeleton-tac* @{*context*} *1*›)
apply (*tactic* ‹*Transport.transport-related-step-tac* @{*context*} *1*›)
apply (*tactic* ‹*Transport.transport-related-step-tac* @{*context*} *1*›)
apply (*tactic* ‹*Transport.transport-related-step-tac* @{*context*} *1*›)
apply (*tactic* ‹*Transport.transport-related-step-tac* @{*context*} *1*›)
apply (*tactic* ‹*Transport.transport-related-step-tac* @{*context*} *1*›)
apply (*tactic* ‹*Transport.transport-related-step-tac* @{*context*} *1*›)
apply *assumption*
apply *assumption*
prefer *3*
apply (*tactic* ‹*Transport.transport-related-step-tac* @{*context*} *1*›)
apply (*tactic* ‹*Transport.transport-related-step-tac* @{*context*} *1*›)
apply (*fold trp-def*)
apply (*trp-relator-rewrite*)+
apply (*unfold trp-def*)
apply (*trp-hints-resolve refl*)
done

lemma *pint-middle-compat*:
transport-comp.middle-compatible-codom (*rel-set* ((=) :: *pint* ⇒ -))
($=_{Collect\ (finite\ ::\ pint\ set\ \Rightarrow\ -)}$)
by (*intro transport-comp.middle-compatible-codom-if-right1-le-eqI*)
(*auto simp*: *rel-set-eq intro*!: *transitiveI*)

trp-term *pint-fset-succ* :: *pint fset* ⇒ *pint fset*
where *x* = *set-succ* :: *int set* ⇒ *int set*

oops

end
end
end

end

2.18 Transport Paper Guide

theory *Transport-Via-Partial-Galois-Connections-Equivalences-Paper*
imports
Transport
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: $\textit{galois-prop.galois-prop}\ ?L\ ?R\ ?l\ ?r \Longrightarrow (\textit{galois-rel.Galois}\ ?R^{-1}\ ?L^{-1}\ ?l)^{-1}\ ?x\ ?y = \textit{Galois-infix}\ ?x\ ?L\ ?R\ ?r\ ?y$
- 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: $[\![\textit{order-functors.order-equivalence}\ ?L\ ?R\ ?l\ ?r;\ \textit{transitive}\ ?L;\ \textit{transitive}\ ?R]\!] \Longrightarrow \textit{galois.galois-equivalence}\ ?L\ ?R\ ?l\ ?r$
  - Lemma 5: $[\![\textit{galois.galois-equivalence}\ ?L\ ?R\ ?l\ ?r;\ \textit{reflexive-on}\ (\textit{in-field}\ ?L)\ ?L;\ \textit{reflexive-on}\ (\textit{in-field}\ ?R)\ ?R]\!] \Longrightarrow \textit{order-functors.order-equivalence}\ ?L\ ?R\ ?l\ ?r$
- Section 4.1: Closure (Dependent) Function Relator can be found in `Functions`.
  - Monotone function relator *Transport.Monotone-Function-Relator*.
  - Setup of construction *Transport.Transport-Functions-Base*.
  - Theorem 1: see *Transport.Transport-Functions*
  - Theorem 2: see $[\![\textit{transport.preorder-equivalence}\ ?L1.0\ ?R1.0\ ?l1.0\ ?r1.0;\ \bigwedge x\ x'.\ \textit{Galois-infix}\ x\ ?L1.0\ ?R1.0\ ?r1.0\ x' \Longrightarrow \textit{transport.preorder-equivalence}\ (?L2.0\ x\ (?r1.0\ x'))\ (?R2.0\ (?l1.0\ x)\ x')\ (?l2.0\ x'\ x)\ (?r2.0\ x\ x');\ ([x1\ x2 :: ?L1.0^{-1}] \Rrightarrow_m [x3\ x4 :: ?L1.0] \Rrightarrow ?L1.0\ x1\ x3 \longrightarrow (\le))\ ?L2.0;\ ([x1\ x2 :: ?L1.0] \Rrightarrow_m [x1'\ x2' :: ?R1.0] \Rrightarrow \textit{Galois-infix}\ x2\ ?L1.0\ ?R1.0\ ?r1.0\ x1' \longrightarrow ([\textit{in-field}\ (?R2.0\ (?l1.0\ x1)\ x2')] \Rrightarrow ?L2.0\ x1\ (?r1.0\ x2')))\ ?r2.0;\ \textit{in-dom}\ (\textit{transport-Mono-Dep-Fun-Rel.L}\ ?L1.0\ ?L2.0)\ ?f;$

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-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*] $\Rrightarrow$ *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: $\llbracket$*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*$\rrbracket \Longrightarrow$ *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: $\llbracket$(*?L1.0* $\Rrightarrow_m$ *?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* $\Rrightarrow_m$ *?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*$\rrbracket \Longrightarrow$ *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* $\Rrightarrow$ *galois-rel.Galois ?L2.0 ?R2.0 ?r2.0*) *?f ?g*
- Theorem 7: $\llbracket$*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*; $\bigwedge$*x x′*. *Galois-infix x ?L1.0 ?R1.0 ?r1.0 x′* $\Longrightarrow$ *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*] $\Rrightarrow_m$ [*x3 x4* :: *?L1.0*] $\Rrightarrow$ (*?L1.0 x2 x3* $\wedge$ *?L1.0 x4* (*order-functors.unit ?l1.0 ?r1.0 x3*)) $\longrightarrow$ (λ*x y*. *y* $\le$ *x*)) *?L2.0*; ([*x1′ x2′* :: *?R1.0*] $\Rrightarrow_m$ *?R1.0* (*order-functors.counit* *?l1.0 ?r1.0 x2′*) *x1′* $\longrightarrow$ ([*x3′* - :: *?R1.0*] $\Rrightarrow$ *?R1.0 x2′ x3′* $\longrightarrow$ ($\le$))) *?R2.0*; ([*x1′ x2′* :: *?R1.0*] $\Rrightarrow_m$ [*x1 x2* :: *?L1.0*] $\Rrightarrow$ *Galois-infix x2 ?L1.0 ?R1.0 ?r1.0 x1′* $\longrightarrow$ ([*in-field* (*?L2.0 x1* (*?r1.0 x2′*))] $\Rrightarrow$ *?R2.0* (*?l1.0 x1*) *x2′*)) *?l2.0*; ([*x1 x2* :: *?L1.0*] $\Rrightarrow_m$ [*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*); $\bigwedge$*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′* $\Rrightarrow_m$ *?L2.0 x* (*?r1.0 x′*))

$(?r2.0\ x\ x')$; $([x1 :: \top] \Rrightarrow_m [x2\ \text{-} :: ?L1.0] \Rrightarrow_m ?L1.0\ x1\ x2 \longrightarrow (\le))\ ?L2.0$; $([x1 :: \top] \Rrightarrow_m [x2\ x3 :: ?L1.0] \Rrightarrow_m (?L1.0\ x1\ x2 \wedge ?L1.0\ x3\ (\mathit{order\text{-}functors.unit}\ ?l1.0\ ?r1.0\ x2)) \longrightarrow (\lambda x\ y.\ y \le x))\ ?L2.0$; $([x1\ x2 :: ?L1.0] \Rrightarrow_m [x1'\ x2' :: ?R1.0] \Rrightarrow \mathit{Galois\text{-}infix}\ x2\ ?L1.0\ ?R1.0\ ?r1.0\ x1' \longrightarrow ([\mathit{in\text{-}field}\ (?R2.0\ (?l1.0\ x1)\ x2')] \Rrightarrow ?L2.0\ x1\ (?r1.0\ x2')))\ ?r2.0$; $\bigwedge x1\ x2.\ ?L1.0\ x1\ x2 \Longrightarrow \mathit{transitive}\ (?L2.0\ x1\ x2)$; $\mathit{in\text{-}dom}\ (\mathit{transport\text{-}Mono\text{-}Dep\text{-}Fun\text{-}Rel.L}\ ?L1.0\ ?L2.0)\ ?f$; $\mathit{in\text{-}codom}\ (\mathit{transport\text{-}Mono\text{-}Dep\text{-}Fun\text{-}Rel.L}\ ?R1.0\ ?R2.0)\ ?g]\!]$ $\Longrightarrow \mathit{Galois\text{-}infix}\ ?f\ (\mathit{transport\text{-}Mono\text{-}Dep\text{-}Fun\text{-}Rel.L}\ ?L1.0\ ?L2.0)\ (\mathit{transport\text{-}Mono\text{-}Dep\text{-}Fun\text{-}Rel.L}\ ?R1.0\ ?R2.0)\ (\mathit{transport\text{-}Dep\text{-}Fun\text{-}Rel.l}\ ?l1.0\ ?r2.0)\ ?g = ([x\ x' :: \mathit{galois\text{-}rel.Galois}\ ?L1.0\ ?R1.0\ ?r1.0] \Rrightarrow \mathit{galois\text{-}rel.Galois}\ (?L2.0\ x\ (?r1.0\ x'))\ (?R2.0\ (?l1.0\ x)\ x')\ (?r2.0\ x\ x'))\ ?f\ ?g$

  - Lemma 8 $[\![\mathit{galois.galois\text{-}equivalence}\ ?L1.0\ ?R1.0\ ?l1.0\ ?r1.0$; $\mathit{Preorders.preorder\text{-}on}\ (\mathit{in\text{-}field}\ ?L1.0)\ ?L1.0$; $([x1\ x2 :: ?L1.0^{-1}] \Rrightarrow_m [x3\ x4 :: ?L1.0] \Rrightarrow ?L1.0\ x1\ x3 \longrightarrow (\le))\ ?L2.0$; $\bigwedge x1\ x2.\ ?L1.0\ x1\ x2 \Longrightarrow \mathit{partial\text{-}equivalence\text{-}rel}\ (?L2.0\ x1\ x2)]\!] \Longrightarrow \mathit{transport\text{-}Mono\text{-}Dep\text{-}Fun\text{-}Rel.L}\ ?L1.0\ ?L2.0 = \mathit{transport\text{-}Dep\text{-}Fun\text{-}Rel.L}\ ?L1.0\ ?L2.0$
  - Lemma 9: $[\![\mathit{reflexive\text{-}on}\ (\mathit{in\text{-}field}\ ?L1.0)\ ?L1.0$; $\mathit{partial\text{-}equivalence\text{-}rel}\ ?L2.0]\!] \Longrightarrow \mathit{transport\text{-}Mono\text{-}Dep\text{-}Fun\text{-}Rel.L}\ ?L1.0\ (\lambda\text{-}\ \text{-}.\ ?L2.0) = \mathit{transport\text{-}Dep\text{-}Fun\text{-}Rel.L}\ ?L1.0\ (\lambda\text{-}\ \text{-}.\ ?L2.0)$

- Section 4.2: Closure Natural Functors can be found in `Natural_Functors`.
  - Theorem 3: see *Transport.Transport-Natural-Functors*
  - Theorem 4: $\mathit{galois\text{-}rel.Galois}\ (\mathit{transport\text{-}natural\text{-}functor.L}\ ?L1.0\ ?L2.0\ ?L3.0)\ (\mathit{transport\text{-}natural\text{-}functor.L}\ ?R1.0\ ?R2.0\ ?R3.0)\ (\mathit{transport\text{-}natural\text{-}functor.l}\ ?r1.0\ ?r2.0\ ?r3.0) = \mathit{Frel}\ (\mathit{galois\text{-}rel.Galois}\ ?L1.0\ ?R1.0\ ?r1.0)\ (\mathit{galois\text{-}rel.Galois}\ ?L2.0\ ?R2.0\ ?r2.0)\ (\mathit{galois\text{-}rel.Galois}\ ?L3.0\ ?R3.0\ ?r3.0)$

- Section 4.3: Closure Compositions can be found in `Compositions`.
  - Setup for simple case in *Transport.Transport-Compositions-Agree-Base*
  - Setup for generic case in *Transport.Transport-Compositions-Generic-Base*
  - Theorem 5: $[\![\mathit{transport.preorder\text{-}equivalence}\ ?L1.0\ ?R1.0\ ?l1.0\ ?r1.0$; $\mathit{transport.preorder\text{-}equivalence}\ ?L2.0\ ?R2.0\ ?l2.0\ ?r2.0$; $\mathit{transport\text{-}comp.middle\text{-}compatible\text{-}codom}\ ?R1.0\ ?L2.0]\!] \Longrightarrow \mathit{transport.preorder\text{-}equivalence}\ (\mathit{transport\text{-}comp.L}\ ?L1.0\ ?R1.0\ ?l1.0\ ?r1.0\ ?L2.0)\ (\mathit{transport\text{-}comp.L}\ ?R2.0\ ?L2.0\ ?r2.0\ ?l2.0\ ?R1.0)\ (\mathit{transport\text{-}comp.l}\ ?l1.0\ ?l2.0)\ (\mathit{transport\text{-}comp.l}\ ?r2.0\ ?r1.0)$ 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*$]\!] \Longrightarrow$ *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*$]\!] \Longrightarrow$ *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* $\circ\circ$ *galois-rel.Galois ?L2.0 ?R2.0 ?r2.0*

  (*results from Appendix*)
- Theorem 9: see `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*$]\!] \Longrightarrow$ *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* $\Rrightarrow_m$ *?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* $\Rrightarrow_m$ *?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* $\circ\circ$ *?L2.0* $\circ\circ$ *?R1.0* $\le$ *?R1.0* $\circ\circ$ *?L2.0*; *in-codom* (*?L2.0* $\circ\circ$ *?R1.0* $\circ\circ$ *?L2.0*) $\le$ *in-codom ?R1.0*$]\!] \Longrightarrow$ *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* $\circ\circ$ *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] Kevin Kappelmann. Transport via Partial Galois Connections and Equivalences, 2023.