Transport via Partial Galois Connections and Equivalences

Kevin Kappelmann

April 29, 2024

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

theory *Adhoc-Overloading*
imports *Pure*
keywords *adhoc-overloading no-adhoc-overloading* :: *thy-decl*
begin

ML‹
signature ADHOC-OVERLOADING =

```
sig
  val is-overloaded: Proof.context -> string -> bool
  val generic-add-overloaded: string -> Context.generic -> Context.generic
  val generic-remove-overloaded: string -> Context.generic -> Context.generic
  val generic-add-variant: string -> term -> Context.generic -> Context.generic
  (*If the list of variants is empty at the end of generic-remove-variant, then
  generic-remove-overloaded is called implicitly.*)
  val generic-remove-variant: string -> term -> Context.generic -> Context.generic
  val show-variants: bool Config.T
end

structure Adhoc-Overloading: ADHOC-OVERLOADING =
struct

val show-variants = Attrib.setup-config-bool binding‹show-variants› (K false);


(* errors *)

fun err-duplicate-variant oconst =
  error (Duplicate variant of " ^ quote oconst);

fun err-not-a-variant oconst =
  error (Not a variant of " ^ quote oconst);

fun err-not-overloaded oconst =
  error (Constant " ^ quote oconst ^ " is not declared as overloaded);

fun err-unresolved-overloading ctxt0 (c, T) t instances =
  let
    val ctxt = Config.put show-variants true ctxt0
    val const-space = Proof-Context.const-space ctxt
    val prt-const =
      Pretty.block [Name-Space.pretty ctxt const-space c, Pretty.str "::", Pretty.brk 1,
        Pretty.quote (Syntax.pretty-typ ctxt T)]
  in
    error (Pretty.string-of (Pretty.chunks [
      Pretty.block [
        Pretty.str "Unresolved adhoc overloading of constant", Pretty.brk 1,
        prt-const, Pretty.brk 1, Pretty.str "in term", Pretty.brk 1,
        Pretty.block [Pretty.quote (Syntax.pretty-term ctxt t)]],
      Pretty.block (
        (if null instances then [Pretty.str "no instances"]
        else Pretty.fbreaks (
          Pretty.str "multiple instances:" ::
          map (Pretty.mark Markup.item o Syntax.pretty-term ctxt) instances)))]))
  end;
```

```
(* generic data *)

fun variants-eq ((v1, T1), (v2, T2)) =
  Term.aconv-untyped (v1, v2) andalso T1 = T2;

structure Overload-Data = Generic-Data
(
  type T =
    {variants : (term * typ) list Symtab.table,
     oconsts : string Termtab.table};
  val empty = {variants = Symtab.empty, oconsts = Termtab.empty};

  fun merge
    ({variants = vtab1, oconsts = otab1},
     {variants = vtab2, oconsts = otab2}) : T =
    let
      fun merge-oconsts - (oconst1, oconst2) =
        if oconst1 = oconst2 then oconst1
        else err-duplicate-variant oconst1;
    in
      {variants = Symtab.merge-list variants-eq (vtab1, vtab2),
       oconsts = Termtab.join merge-oconsts (otab1, otab2)}
    end;
);

fun map-tables f g =
  Overload-Data.map (fn {variants = vtab, oconsts = otab} =>
    {variants = f vtab, oconsts = g otab});

val is-overloaded = Symtab.defined o #variants o Overload-Data.get o Context.Proof;
val get-variants = Symtab.lookup o #variants o Overload-Data.get o Context.Proof;
val get-overloaded = Termtab.lookup o #oconsts o Overload-Data.get o Context.Proof;

fun generic-add-overloaded oconst context =
  if is-overloaded (Context.proof-of context) oconst then context
  else map-tables (Symtab.update (oconst, [])) I context;

fun generic-remove-overloaded oconst context =
  let
    fun remove-oconst-and-variants context oconst =
      let
        val remove-variants =
          (case get-variants (Context.proof-of context) oconst of
            NONE => I
          | SOME vs => fold (Termtab.remove (op =) o rpair oconst o fst) vs);
      in map-tables (Symtab.delete-safe oconst) remove-variants context end;
  in
   if is-overloaded (Context.proof-of context) oconst then remove-oconst-and-variants
context oconst
```

```
    else err-not-overloaded oconst
  end;

local
  fun generic-variant add oconst t context =
    let
      val ctxt = Context.proof-of context;
      val - = if is-overloaded ctxt oconst then () else err-not-overloaded oconst;
      val T = t |> fastype-of;
      val t' = Term.map-types (K dummyT) t;
    in
      if add then
        let
          val - =
            (case get-overloaded ctxt t' of
              NONE => ()
            | SOME oconst' => err-duplicate-variant oconst');
        in
        map-tables (Symtab.cons-list (oconst, (t', T))) (Termtab.update (t', oconst))
context
        end
      else
        let
          val - =
            if member variants-eq (the (get-variants ctxt oconst)) (t', T) then ()
            else err-not-a-variant oconst;
        in
          map-tables (Symtab.map-entry oconst (remove1 variants-eq (t', T)))
            (Termtab.delete-safe t') context
          |> (fn context =>
            (case get-variants (Context.proof-of context) oconst of
              SOME [] => generic-remove-overloaded oconst context
            | - => context))
        end
    end;
in
  val generic-add-variant = generic-variant true;
  val generic-remove-variant = generic-variant false;
end;


(* check / uncheck *)

fun unifiable-with thy T1 T2 =
  let
    val maxidx1 = Term.maxidx-of-typ T1;
    val T2' = Logic.incr-tvar (maxidx1 + 1) T2;
    val maxidx2 = Term.maxidx-typ T2' maxidx1;
  in can (Sign.typ-unify thy (T1, T2')) (Vartab.empty, maxidx2) end;
```

```
fun get-candidates ctxt (c, T) =
  get-variants ctxt c
  |> Option.map (map-filter (fn (t, T') =>
    if unifiable-with (Proof-Context.theory-of ctxt) T T'
    (*keep the type constraint for the type−inference check phase*)
    then SOME (Type.constraint T t)
    else NONE));

fun insert-variants ctxt t (oconst as Const (c, T)) =
      (case get-candidates ctxt (c, T) of
        SOME [] => err-unresolved-overloading ctxt (c, T) t []
      | SOME [variant] => variant
      | - => oconst)
  | insert-variants - - oconst = oconst;

fun insert-overloaded ctxt =
  let
    fun proc t =
      Term.map-types (K dummyT) t
      |> get-overloaded ctxt
      |> Option.map (Const o rpair (Term.type-of t));
  in
    Pattern.rewrite-term-top (Proof-Context.theory-of ctxt) [] [proc]
  end;

fun check ctxt =
  map (fn t => Term.map-aterms (insert-variants ctxt t) t);

fun uncheck ctxt ts =
  if Config.get ctxt show-variants orelse exists (is-none o try Term.type-of) ts then
ts
  else map (insert-overloaded ctxt) ts;

fun reject-unresolved ctxt =
  let
    val the-candidates = the o get-candidates ctxt;
    fun check-unresolved t =
      (case filter (is-overloaded ctxt o fst) (Term.add-consts t []) of
        [] => t
      | (cT :: -) => err-unresolved-overloading ctxt cT t (the-candidates cT));
  in map check-unresolved end;


(* setup *)

val - = Context.>>
  (Syntax-Phases.term-check 0 adhoc-overloading check
  #> Syntax-Phases.term-check 1 adhoc-overloading-unresolved-check reject-unresolved
```

```
  #> Syntax-Phases.term-uncheck 0 adhoc-overloading uncheck);


(* commands *)

fun generic-adhoc-overloading-cmd add =
  if add then
    fold (fn (oconst, ts) =>
      generic-add-overloaded oconst
      #> fold (generic-add-variant oconst) ts)
  else
    fold (fn (oconst, ts) =>
      fold (generic-remove-variant oconst) ts);

fun adhoc-overloading-cmd' add args phi =
  let val args' = args
    |> map (apsnd (map-filter (fn t =>
        let val t' = Morphism.term phi t;
        in if Term.aconv-untyped (t, t') then SOME t' else NONE end)));
  in generic-adhoc-overloading-cmd add args' end;

fun adhoc-overloading-cmd add raw-args lthy =
  let
    fun const-name ctxt =
      fst o dest-Const o Proof-Context.read-const {proper = false, strict = false}
ctxt;  (* FIXME {proper = true, strict = true} (!?) *)
    fun read-term ctxt = singleton (Variable.polymorphic ctxt) o Syntax.read-term
ctxt;
    val args =
      raw-args
      |> map (apfst (const-name lthy))
      |> map (apsnd (map (read-term lthy)));
  in
   Local-Theory.declaration {syntax = true, pervasive = false, pos = Position.thread-data
()}
      (adhoc-overloading-cmd' add args) lthy
  end;

val - =
  Outer-Syntax.local-theory command-keyword ‹adhoc-overloading›
    add adhoc overloading for constants / fixed variables
   (Parse.and-list1 (Parse.const -- Scan.repeat Parse.term) >> adhoc-overloading-cmd
true);

val - =
  Outer-Syntax.local-theory command-keyword ‹no-adhoc-overloading›
    delete adhoc overloading for constants / fixed variables
   (Parse.and-list1 (Parse.const -- Scan.repeat Parse.term) >> adhoc-overloading-cmd
false);
```

end;
›

end

Chapter 1

HOL-Basics

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

end

1.0.1 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

1.0.2 Lattice

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

lemma *inf-predI* [*intro*]:
assumes $P\ x$
and $Q\ x$
shows $(P \sqcap Q)\ x$
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

1.0.3 Bounded Quantifiers

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

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

bundle *ball-syntax*
begin
syntax
-ball :: ‹[*idts*, $'a$, *bool*] ⇒ *bool*› ((*2*∀ - : -./ -) *10*)
-ball2 :: ‹[*idts*, $'a$, *bool*] ⇒ *bool*›
notation *ball* (∀(_))
end
bundle *no-ball-syntax*
begin
no-syntax

-ball :: ‹[*idts*, *'a*, *bool*] ⇒ *bool*› ((*2*∀ - : -./ -) *10*)
-ball2 :: ‹[*idts*, *'a*, *bool*] ⇒ *bool*›
no-notation *ball* (∀(_))
end
unbundle *ball-syntax*
translations
∀ *x xs* : *P*. *Q* ⇀ *CONST ball P* (λ*x*. *-ball2 xs P Q*)
-ball2 x P Q ⇀ ∀ *x* : *P*. *Q*
∀ *x* : *P*. *Q* ⇌ *CONST ball P* (λ*x*. *Q*)

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

bundle *bex-syntax*
begin
syntax
-bex :: ‹[*idts*, *'a*, *bool*] ⇒ *bool*› ((*2*∃ - : -./ -) *10*)
-bex2 :: ‹[*idts*, *'a*, *bool*] ⇒ *bool*›
notation *bex* (∃(_))
end
bundle *no-bex-syntax*
begin
no-syntax
-bex :: ‹[*idts*, *'a*, *bool*] ⇒ *bool*› ((*2*∃ - : -./ -) *10*)
-bex2 :: ‹[*idts*, *'a*, *bool*] ⇒ *bool*›
no-notation *bex* (∃(_))
end
unbundle *bex-syntax*
translations
∃ *x xs* : *P*. *Q* ⇀ *CONST bex P* (λ*x*. *-bex2 xs P Q*)
-bex2 x P Q ⇀ ∃ *x* : *P*. *Q*
∃ *x* : *P*. *Q* ⇌ *CONST bex P* (λ*x*. *Q*)

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

bundle *bex1-syntax*
begin
syntax
-bex1 :: ‹[*idts*, *'a*, *bool*] ⇒ *bool*› ((*2*∃!- : -./ -) *10*)
-bex12 :: ‹[*idts*, *'a*, *bool*] ⇒ *bool*›
notation *bex1* (∃!(_))
end
bundle *no-bex1-syntax*
begin
no-syntax
-bex1 :: ‹[*idts*, *'a*, *bool*] ⇒ *bool*› ((*2*∃!- : -./ -) *10*)
-bex12 :: ‹[*idts*, *'a*, *bool*] ⇒ *bool*›
no-notation *bex1* (∃!(_))
end
unbundle *bex1-syntax*

translations
$\exists!x\ xs : P.\ Q \rightharpoonup CONST\ bex1\ P\ (\lambda x.\ \text{-}bex12\ xs\ P\ Q)$
$\text{-}bex12\ x\ P\ Q \rightharpoonup \exists!x : P.\ Q$
$\exists!x : P.\ Q \rightleftharpoons CONST\ bex1\ P\ (\lambda x.\ Q)$

bundle *bounded-quantifier-syntax* **begin unbundle** *ball-syntax bex-syntax bex1-syntax* **end**
bundle *no-bounded-quantifier-syntax* **begin unbundle** *no-ball-syntax no-bex-syntax no-bex1-syntax* **end**

definition *ball-pred* $P\ Q \equiv \forall x.\ P\ x \longrightarrow Q\ x$
adhoc-overloading *ball ball-pred*

definition *bex-pred* $P\ Q \equiv \exists x.\ P\ x \wedge Q\ x$
adhoc-overloading *bex bex-pred*

definition *bex1-pred* $P\ Q \equiv \exists!x.\ P\ x \wedge Q\ x$
adhoc-overloading *bex1 bex1-pred*

simproc-setup *defined-ball* $(\forall x : P.\ Q\ x \longrightarrow U\ x) = \langle$
K (*Quantifier1.rearrange-Ball* (*fn ctxt => unfold-tac ctxt @{thms ball-pred-def}*))
$\rangle$
simproc-setup *defined-bex* $(\exists x : P.\ Q\ x \wedge U\ x) = \langle$
K (*Quantifier1.rearrange-Bex* (*fn ctxt => unfold-tac ctxt @{thms bex-pred-def}*))
$\rangle$

lemma *ballI* [*intro*!]:
assumes $\bigwedge x.\ P\ x \Longrightarrow Q\ x$
shows $\forall x : P.\ Q\ x$
using *assms* **unfolding** *ball-pred-def* **by** *blast*

lemma *ballE* [*elim*]:
assumes $\forall x : P.\ Q\ x$
obtains $\bigwedge x.\ P\ x \Longrightarrow Q\ x$
using *assms* **unfolding** *ball-pred-def* **by** *blast*

lemma *ballE'*:
assumes $\forall x : P.\ Q\ x$
obtains $\neg(P\ x) \mid P\ x\ Q\ x$
using *assms* **by** *blast*

lemma *ballD*: $\forall x : P.\ Q\ x \Longrightarrow P\ x \Longrightarrow Q\ x$
by *blast*

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

lemma *ball-cong-simp* [*cong*]:
  $[\![P = P'; \bigwedge x.\ P'\ x =simp=>\ Q\ x \longleftrightarrow Q'\ x]\!] \Longrightarrow (\forall x : P.\ Q\ x) \longleftrightarrow (\forall x : P'.\ Q'\ x)$
  unfolding *simp-implies-def* **by** (*rule ball-cong*)

ML ‹
structure Simpdata =
struct
  open Simpdata
  val mksimps-pairs = [(***const-name*** ‹*ball-pred*›, @{*thms ballD*})] @ *mksimps-pairs*
end
open Simpdata
›
declaration ‹*fn - => Simplifier.map-ss* (*Simplifier.set-mksimps* (*mksimps mksimps-pairs*))›

lemma *atomize-ball*: $(\bigwedge x.\ P\ x \Longrightarrow Q\ x) \equiv$ *Trueprop* $(\forall x : P.\ Q\ x)$
  by (*simp only*: *ball-pred-def atomize-all atomize-imp*)

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

lemma *bexI* [*intro*!]:

  assumes $\exists x.\ Q\ x \wedge P\ x$
  shows $\exists x : P.\ Q\ x$
  using *assms* **unfolding** *bex-pred-def* **by** *blast*

lemma *bexE* [*elim*!]:
  assumes $\exists x : P.\ Q\ x$
  obtains *x* **where** *P x Q x*
  using *assms* **unfolding** *bex-pred-def* **by** *blast*

lemma *bexD*:
  assumes $\exists x : P.\ Q\ x$
  shows $\exists x.\ P\ x \wedge Q\ x$
  using *assms* **by** *blast*

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

lemma *bex-cong-simp* [*cong*]:
  $[\![P = P'; \bigwedge x.\ P'\ x =simp=>\ Q\ x \longleftrightarrow Q'\ x]\!] \Longrightarrow (\exists x : P.\ Q\ x) \longleftrightarrow (\exists x : P'.\ Q'\ x)$
  unfolding *simp-implies-def* **by** (*rule bex-cong*)

lemma *bex1I* [*intro*]:

assumes $\exists!x.\ Q\ x \wedge P\ x$
shows $\exists!x : P.\ Q\ x$
using *assms* **unfolding** *bex1-pred-def* **by** *blast*

lemma *bex1D* [*dest*!]:
assumes $\exists!x : P.\ Q\ x$
shows $\exists!x.\ P\ x \wedge Q\ x$
using *assms* **unfolding** *bex1-pred-def* **by** *blast*

lemma *bex1-cong*: $[\![P = P'; \bigwedge x.\ P\ x \Longrightarrow Q\ x \longleftrightarrow Q'\ x]\!] \Longrightarrow (\exists!x : P.\ Q\ x) \longleftrightarrow (\exists!x : P'.\ Q'\ x)$
by *blast*

lemma *bex1-cong-simp* [*cong*]:
$[\![P = P'; \bigwedge x.\ P\ x =simp=> Q\ x \longleftrightarrow Q'\ x]\!] \Longrightarrow (\exists!x : P.\ Q\ x) \longleftrightarrow (\exists!x : P'.\ Q'\ x)$
unfolding *simp-implies-def* **by** (*rule bex1-cong*)

lemma *ball-iff-ex-pred* [*iff*]: $(\forall x : P.\ Q) \longleftrightarrow ((\exists x.\ P\ x) \longrightarrow Q)$
by *auto*

lemma *bex-iff-ex-and* [*iff*]: $(\exists x : P.\ Q) \longleftrightarrow ((\exists x.\ P\ x) \wedge Q)$
by *blast*

lemma *ball-eq-imp-iff-imp* [*iff*]: $(\forall x : P.\ x = y \longrightarrow Q\ x) \longleftrightarrow (P\ y \longrightarrow Q\ y)$
by *blast*

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

lemma *bex-eq-iff-pred* [*iff*]: $(\exists x : P.\ x = y) \longleftrightarrow P\ y$
by *blast*

lemma *bex-eq-iff-pred'* [*iff*]: $(\exists x : P.\ y = x) \longleftrightarrow P\ y$
by *blast*

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

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

lemma *ball-and-iff-ball-and-ball*: $(\forall x : P.\ Q\ x \wedge U\ x) \longleftrightarrow (\forall x : P.\ Q\ x) \wedge (\forall x : P.\ U\ x)$
by *auto*

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

by *auto*

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

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

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

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

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

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

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

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

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

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

lemma *ball-top-eq-all* [*simp*]: $\forall_{\top} = All$ **by** *fastforce*

lemma *ball-top-eq-all-uhint* [*uhint*]:
assumes $P \equiv (\top :: 'a \Rightarrow bool)$
shows $\forall_P \equiv All$
using *assms* **by** *simp*

lemma *bex-top-eq-ex* [*simp*]: $\exists_{\top} = Ex$ **by** *fastforce*

lemma *bex-top-eq-ex-uhint* [*uhint*]:
assumes $P \equiv (\top :: 'a \Rightarrow bool)$
shows $\exists_P \equiv Ex$
using *assms* **by** *simp*

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

lemma *bex1-top-eq-ex1-uhint* [*uhint*]:
assumes $P \equiv (\top :: 'a \Rightarrow bool)$
shows $\exists!_P \equiv Ex1$
using *assms* **by** *simp*

end

1.1 Binary Relations

1.1.1 Basic Functions

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

Summary Basic functions on binary relations.

Domain, Codomain, and Field

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

lemma *in-domI* [*intro*]:
assumes $R\ x\ y$
shows *in-dom* $R\ x$
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-bottom-eq-bottom* [*simp*]: *in-dom* $\bot = \bot$ **by** *fastforce*
lemma *in-dom-top-eq-top* [*simp*]: *in-dom* $\top = \top$ **by** *fastforce*

lemma *in-dom-sup-eq-in-dom-sup-in-dom* [*simp*]: *in-dom* $(R \sqcup S) =$ *in-dom* $R \sqcup$ *in-dom* S **by** *fastforce*

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

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

lemma *in-codom-onI* [*intro*]:
assumes $R\ x\ y$
and $P\ x$

shows *in-codom-on P R y*
using *assms* **unfolding** *in-codom-on-pred-def* **by** *blast*

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

lemma *in-codom-on-pred-iff-bex-rel*: *in-codom-on P R y* $\longleftrightarrow$ $(\exists\, x : P.\ R\ x\ y)$ **by** *blast*

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

lemma *in-codom-on-eq-pred-eq* [*simp*]: *in-codom-on* $((=)\ P)\ R = R\ P$
by *auto*

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

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

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

lemma *in-codom-eq-in-codom-on-top*: *in-codom* = *in-codom-on* $\top$ **unfolding** *in-codom-def*
by *auto*

lemma *in-codom-eq-in-codom-on-top-uhint* [*uhint*]:
assumes $P \equiv (\top :: 'a \Rightarrow bool)$
shows *in-codom* $\equiv$ *in-codom-on P*
using *assms* **by** (*simp add*: *in-codom-eq-in-codom-on-top*)

lemma *in-codomI* [*intro*]:
assumes *R x y*
shows *in-codom R y*
using *assms* **by** (*urule in-codom-onI*) *simp*

lemma *in-codomE* [*elim*]:
assumes *in-codom R y*
obtains *x* **where** *R x y*
using *assms* **by** (*urule* (*e*) *in-codom-onE*)

definition *in-field R x* $\equiv$ *in-dom R x* $\vee$ *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 R x $\longleftrightarrow$ *in-dom R x* $\vee$ *in-codom R x*
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'*)

lemma *in-field-bottom-eq-bottom* [*simp*]: *in-field* $\bot$ = $\bot$ **by** *fastforce*
lemma *in-field-top-eq-top* [*simp*]: *in-field* $\top$ = $\top$ **by** *fastforce*

lemma *in-field-sup-eq-in-field-sup-in-field* [*simp*]: *in-field* (*R* $\sqcup$ *S*) = *in-field R* $\sqcup$ *in-field S*
by *fastforce*

Composition

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

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

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

adhoc-overloading *rel-comp rel-comp-rel*

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

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*

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-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*

Inverse

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

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*

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

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

lemma *rel-invD* [*dest*]:

assumes R^{-1} x y
shows R y x
using *assms* **unfolding** *rel-inv-rel-def* **.**

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

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

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

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

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*)

lemma *rel-inv-le-rel-inv-iff* [*iff*]: $R^{-1} \leq S^{-1} \longleftrightarrow R \leq S$
by (*auto intro*: *predicate2I dest*: *predicate2D*)

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

Restrictions

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

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*

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

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

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

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

consts *rel-restrict-left* :: $'a \Rightarrow 'b \Rightarrow 'a$
consts *rel-restrict-right* :: $'a \Rightarrow 'b \Rightarrow 'a$

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

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

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

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

lemma *rel-restrict-left-cong*:
assumes $\bigwedge x.\ P\ x \longleftrightarrow P'\ x$
and $\bigwedge x\ y.\ P'\ x \Longrightarrow R\ x\ y \longleftrightarrow R'\ x\ y$
shows $R{\restriction}_P = R'{\restriction}_{P'}$
using *assms* **by** (*intro ext iffI*) *blast+*

lemma *rel-restrict-left-top-eq* [*simp*]: $(R :: 'a \Rightarrow 'b \Rightarrow bool){\restriction}_{\top :: 'a \Rightarrow bool} = R$
by (*intro ext rel-restrict-leftI*) *auto*

lemma *rel-restrict-left-top-eq-uhint* [*uhint*]:
assumes $P \equiv (\top :: 'a \Rightarrow bool)$
shows $(R :: 'a \Rightarrow 'b \Rightarrow bool){\restriction}_P \equiv R$
using *assms* **by** *simp*

lemma *rel-restrict-left-bottom-eq* [*simp*]: $(R :: 'a \Rightarrow 'b \Rightarrow bool){\restriction}_{\bot :: 'a \Rightarrow bool} = \bot$
by (*intro ext rel-restrict-leftI*) *auto*

lemma *bottom-restrict-left-eq* [*simp*]: $\bot{\restriction}_{P :: 'a \Rightarrow bool} = \bot$
by *fastforce*

lemma *rel-restrict-left-le-self*: $(R :: 'a \Rightarrow 'b \Rightarrow bool){\restriction}_{(P :: 'a \Rightarrow bool)} \leq R$
by (*auto intro*: *predicate2I dest*: *predicate2D*)

lemma *le-rel-restrict-left-self-if-in-dom-le*:
assumes *in-dom* $R \leq P$
shows $R \leq (R :: 'a \Rightarrow 'b \Rightarrow bool){\restriction}_{(P :: 'a \Rightarrow bool)}$
using *assms* **by** (*auto intro*: *predicate2I dest*: *predicate2D predicate1D*)

corollary *rel-restrict-left-eq-self-if-in-dom-le* [*simp*]:
assumes *in-dom* $R \leq P$
shows $R{\restriction}_P = R$
using *assms rel-restrict-left-le-self le-rel-restrict-left-self-if-in-dom-le* **by** (*intro antisym*)

lemma *ex-rel-restrict-left-iff-in-codom-on* [*iff*]: $(\exists\, x.\ R{\restriction}_P\ x\ y) \longleftrightarrow (\textit{in-codom-on}\ P\ R\ y)$ **by** *blast*

lemma *rel-restrict-rightI* [*intro*]:
assumes $R\ x\ y$
and $P\ y$
shows $R{\restriction}_P\ x\ y$
using *assms* **unfolding** *rel-restrict-right-pred-def* **by** *blast*

lemma *rel-restrict-rightE* [*elim*]:
assumes $R{\restriction}_P\ x\ y$
obtains $P\ y\ R\ x\ y$
using *assms* **unfolding** *rel-restrict-right-pred-def* **by** *blast*

lemma *rel-restrict-right-cong*:
assumes $\bigwedge y.\ P\ y \longleftrightarrow P'\ y$
and $\bigwedge x\ y.\ P'\ y \Longrightarrow R\ x\ y \longleftrightarrow R'\ x\ y$
shows $R{\restriction}_P = R'{\restriction}_{P'}$
using *assms* **by** (*intro ext iffI*) *blast+*

lemma *rel-restrict-right-top-eq* [*simp*]: $(R :: 'a \Rightarrow 'b \Rightarrow bool){\restriction}_{\top :: 'b \Rightarrow bool} = R$
by (*intro ext rel-restrict-rightI*) *auto*

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

lemma *rel-restrict-right-bottom-eq* [*simp*]: $(R :: 'a \Rightarrow 'b \Rightarrow bool){\restriction}_{\bot :: 'b \Rightarrow bool} = \bot$
by (*intro ext rel-restrict-rightI*) *auto*

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

lemma *rel-restrict-right-le-self*: $(R :: 'a \Rightarrow 'b \Rightarrow bool){\restriction}_{(P :: 'b \Rightarrow bool)} \leq R$
by (*auto intro*: *predicate2I dest*: *predicate2D*)

lemma *le-rel-restrict-right-self-if-in-codom-le*:
assumes *in-codom* $R \leq P$
shows $R \leq (R :: 'a \Rightarrow 'b \Rightarrow bool){\restriction}_{(P :: 'b \Rightarrow bool)}$
using *assms* **by** (*auto intro*: *predicate2I dest*: *predicate2D predicate1D*)

corollary *rel-restrict-right-eq-self-if-in-codom-le* [*simp*]:
assumes *in-codom* $R \leq P$
shows $R{\restriction}_P = R$
using *assms rel-restrict-right-le-self le-rel-restrict-right-self-if-in-codom-le*
by (*intro antisym*)

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

lemma *rel-restrict-right-eq*: $R{\restriction}_{P :: 'b \Rightarrow bool} = ((R^{-1}){\restriction}_P)^{-1}$
by *blast*

lemma *rel-inv-restrict-right-rel-inv-eq-restrict-left* [*simp*]: $((R^{-1}){\restriction}_{P :: 'a \Rightarrow bool})^{-1} = R{\restriction}_P$
by *blast*

lemma *rel-restrict-right-iff-restrict-left*: $R{\restriction}_{P :: 'b \Rightarrow bool}\ x\ y \longleftrightarrow (R^{-1}){\restriction}_P\ y\ x$
unfolding *rel-restrict-right-eq* **by** *simp*

end

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

lemma *rel-restrict-left-right-eq-restrict-right-left*:
fixes $R :: 'a \Rightarrow 'b \Rightarrow bool$ **and** $P :: 'a \Rightarrow bool$ **and** $Q :: 'b \Rightarrow bool$
shows $R\restriction_P\restriction_Q = R\restriction_Q\restriction_P$
unfolding *rel-restrict-right-eq*
by (*fact rel-inv-rel-restrict-left-inv-rel-restrict-left-eq*)

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

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

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

Mappers

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

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

end

1.2 Functions

1.2.1 Basic Functions

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

definition $id\ x \equiv x$

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

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

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*

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

lemma *comp-eq* [*simp*]: (*f* ∘ *g*) *x* = *f* (*g x*)
  unfolding *comp-fun-def* **..**

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

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

lemma *bottom-comp-eq* [*simp*]: ⊥ ∘ *f* = ⊥ **by** *auto*
lemma *top-comp-eq* [*simp*]: ⊤ ∘ *f* = ⊤ **by** *auto*

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

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

bundle *dep-fun-map-syntax* **begin**
syntax
  -fun-map :: ('*a* ⇒ '*b*) ⇒ ('*c* ⇒ '*d*) ⇒ ('*b* ⇒ '*c*) ⇒
    ('*a* ⇒ '*d*) ((-) ⇝ (-) [*41*, *40*] *40*)
  -dep-fun-map :: *idt* ⇒ ('*a* ⇒ '*b*) ⇒ ('*c* ⇒ '*d*) ⇒ ('*b* ⇒ '*c*) ⇒
    ('*a* ⇒ '*d*) ('(-/ : / -') ⇝ (-) [*41*, *41*, *40*] *40*)
end
bundle *no-dep-fun-map-syntax* **begin**
no-syntax
  -fun-map :: ('*a* ⇒ '*b*) ⇒ ('*c* ⇒ '*d*) ⇒ ('*b* ⇒ '*c*) ⇒
    ('*a* ⇒ '*d*) ((-) ⇝ (-) [*41*, *40*] *40*)
  -dep-fun-map :: *idt* ⇒ ('*a* ⇒ '*b*) ⇒ ('*c* ⇒ '*d*) ⇒ ('*b* ⇒ '*c*) ⇒
    ('*a* ⇒ '*d*) ('(-/ : / -') ⇝ (-) [*41*, *41*, *40*] *40*)
end
unbundle *dep-fun-map-syntax*
translations
  f ⇝ *g* ⇌ *CONST fun-map f g*
  (*x* : *f*) ⇝ *g* ⇌ *CONST dep-fun-map f* (λ*x*. *g*)

lemma *fun-map-eq-dep-fun-map*: (*f* ⇝ *g*) = ((- : *f*) ⇝ (λ-. *g*))
  unfolding *fun-map-def* **by** *simp*

lemma *fun-map-eq-dep-fun-map-uhint* [*uhint*]:
  assumes *f* ≡ *f'*
  and *g'* ≡ (λ- -. *g*)
  shows (*f* ⇝ *g*) = ((*x* : *f'*) ⇝ *g' x*)
  using *assms* **by** (*simp add*: *fun-map-eq-dep-fun-map*)

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

lemma *fun-map-eq-comp* [*simp*]: $(f \rightsquigarrow g)\ h = g \circ h \circ f$
by (*intro ext*) (*urule dep-fun-map-eq*)

lemma *fun-map-eq* [*simp*]: $(f \rightsquigarrow g)\ h\ x = g\ (h\ (f\ x))$
unfolding *fun-map-eq-comp* **by** *simp*

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

lemma *fun-map-id-eq-comp′* [*simp*]: $(f \rightsquigarrow id)\ h = h \circ f$
by (*intro ext*) *simp*

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

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

lemma *has-inverse-on-pred-eq-in-codom-on*: *has-inverse-on* $P =$ *in-codom-on* $P \circ ((\circ)\ (=))$
unfolding *has-inverse-on-pred-def* **by** *auto*

lemma *has-inverse-onI* [*intro*]:
assumes $P\ x$
shows *has-inverse-on* $P\ f\ (f\ x)$
using *assms* **unfolding** *has-inverse-on-pred-def* **by** *auto*

lemma *has-inverse-on-if-pred-if-eq*:
assumes $y = f\ x$
and $P\ x$
shows *has-inverse-on* $P\ f\ y$
using *assms* **by** *auto*

lemma *has-inverse-onE* [*elim*]:
assumes *has-inverse-on* $P\ f\ y$
obtains x **where** $P\ x\ y = f\ x$
using *assms* **unfolding** *has-inverse-on-pred-def* **by** *auto*

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

lemma *has-inverse-on-bottom-eq* [*simp*]: *has-inverse-on* $\bot = \bot$
by (*urule in-codom-bottom-pred-eq-bottom*)

lemma *has-inverse-on-eq-pred-eq-eq-app* [*simp*]: *has-inverse-on* ((=) *P*) *f* = (=) (*f P*)
by (*urule in-codom-on-eq-pred-eq*)

lemma *has-inverse-on-sup-eq-has-inverse-on-sup-has-inverse-on* [*simp*]:
has-inverse-on (*P* ⊔ *Q*) = *has-inverse-on P* ⊔ *has-inverse-on Q*
supply *ext*[*simp*] **by** (*urule in-codom-on-sup-pred-eq-in-codom-on-sup-in-codom-on*)

end

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

lemma *has-inverse-eq-has-inverse-on-top*: *has-inverse* = *has-inverse-on* ⊤
unfolding *has-inverse-def* **by** *simp*

lemma *has-inverse-eq-has-inverse-on-top-uhint* [*uhint*]:
assumes *P* ≡ ⊤
shows *has-inverse* ≡ *has-inverse-on P*
using *assms* **by** (*simp add*: *has-inverse-eq-has-inverse-on-top*)

lemma *has-inverse-iff-has-inverse-on-top*: *has-inverse f y* ⟷ *has-inverse-on* ⊤ *f y*
by (*simp add*: *has-inverse-eq-has-inverse-on-top*)

lemma *has-inverseI* [*intro*]:
assumes *y* = *f x*
shows *has-inverse f y*
using *assms* **by** (*urule has-inverse-on-if-pred-if-eq*) *simp*

lemma *has-inverseE* [*elim*]:
assumes *has-inverse f y*
obtains *x* **where** *y* = *f x*
using *assms* **by** (*urule* (*e*) *has-inverse-onE*)

end

1.2.2 Relators

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

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

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

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

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

definition *Fun-Rel-rel* $(R :: 'a \Rightarrow 'b \Rightarrow bool)\ (S :: 'c \Rightarrow 'd \Rightarrow bool) \equiv$
  $((- - :: R) \Rrightarrow S) :: ('a \Rightarrow 'c) \Rightarrow ('b \Rightarrow 'd) \Rightarrow bool$
adhoc-overloading *Fun-Rel Fun-Rel-rel*

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

definition *Fun-Rel-pred* $(P :: 'a \Rightarrow bool)\ (R :: 'b \Rightarrow 'c \Rightarrow bool) \equiv$
  $((- : P) \Rrightarrow R) :: ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'c) \Rightarrow bool$
adhoc-overloading *Fun-Rel Fun-Rel-pred*

lemma *Fun-Rel-rel-eq-Dep-Fun-Rel-rel*:
(($R :: 'a \Rightarrow 'b \Rightarrow bool$) $\Rrightarrow$ ($S :: 'c \Rightarrow 'd \Rightarrow bool$)) = (($- - :: R$) $\Rrightarrow$ S)
unfolding *Fun-Rel-rel-def* **by** *simp*

lemma *Fun-Rel-rel-eq-Dep-Fun-Rel-rel-uhint* [*uhint*]:
assumes $R \equiv R'$
and $S' \equiv (\lambda(- :: 'a)\ (- :: 'b).\ S)$
shows (($R :: 'a \Rightarrow 'b \Rightarrow bool$) $\Rrightarrow$ ($S :: 'c \Rightarrow 'd \Rightarrow bool$)) = (($x\ y :: R'$) $\Rrightarrow$ $S'\ x\ y$)
using *assms* **by** (*simp add*: *Fun-Rel-rel-eq-Dep-Fun-Rel-rel*)

lemma *Fun-Rel-rel-iff-Dep-Fun-Rel-rel*:
(($R :: 'a \Rightarrow 'b \Rightarrow bool$) $\Rrightarrow$ ($S :: 'c \Rightarrow 'd \Rightarrow bool$)) $f\ g \longleftrightarrow$ (($- - :: R$) $\Rrightarrow$ S) $f\ g$
by (*simp add*: *Fun-Rel-rel-eq-Dep-Fun-Rel-rel*)

lemma *Fun-Rel-pred-eq-Dep-Fun-Rel-pred*:
(($P :: 'a \Rightarrow bool$) $\Rrightarrow$ ($R :: 'b \Rightarrow 'c \Rightarrow bool$)) = (($-$: P) $\Rrightarrow$ R)
unfolding *Fun-Rel-pred-def* **by** *simp*

lemma *Fun-Rel-pred-eq-Dep-Fun-Rel-pred-uhint* [*uhint*]:
assumes $P \equiv P'$
and $R' \equiv (\lambda(- :: 'a).\ R)$
shows (($P :: 'a \Rightarrow bool$) $\Rrightarrow$ ($R :: 'b \Rightarrow 'c \Rightarrow bool$)) = (($x$: P') $\Rrightarrow$ $R'\ x$)
using *assms* **by** (*simp add*: *Fun-Rel-pred-eq-Dep-Fun-Rel-pred*)

lemma *Fun-Rel-pred-iff-Dep-Fun-Rel-pred*:
(($P :: 'a \Rightarrow bool$) $\Rrightarrow$ ($R :: 'b \Rightarrow 'c \Rightarrow bool$)) $f\ g \longleftrightarrow$ (($-$: P) $\Rrightarrow$ R) ($f :: 'a \Rightarrow 'b$) ($g :: 'a \Rightarrow 'c$)
by (*simp add*: *Fun-Rel-pred-eq-Dep-Fun-Rel-pred*)

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

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

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

lemma *Fun-Rel-relE*:
assumes $((x\ y :: R) \Rrightarrow S\ x\ y)\ f\ g$
obtains $\bigwedge x\ y.\ R\ x\ y \Longrightarrow S\ x\ y\ (f\ x)\ (g\ y)$
using *assms* **by** (*urule* (*e*) *Dep-Fun-Rel-relE*)

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* [*dest*]:
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*:
assumes $((x : P) \Rrightarrow R\ x)\ f\ g$
obtains $\bigwedge x.\ P\ x \Longrightarrow R\ x\ (f\ x)\ (g\ x)$
using *assms* **unfolding** *Dep-Fun-Rel-pred-def* **by** *blast*

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

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

lemma *Fun-Rel-predE*:
assumes $(P \Rrightarrow R)\ f\ g$
obtains $\bigwedge x.\ P\ x \Longrightarrow R\ (f\ x)\ (g\ x)$
using *assms* **by** (*urule* (*e*) *Dep-Fun-Rel-predE*)

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

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

lemma *Dep-Fun-Rel-pred-eq-Dep-Fun-Rel-rel*:
fixes $P :: 'a \Rightarrow bool$ **and** $R :: 'a \Rightarrow 'b \Rightarrow 'c \Rightarrow bool$
shows $((x : P) \Rrightarrow R\ x) = ((x\ (\text{-} :: 'a) :: (((\sqcap)\ P) \circ (=))) \Rrightarrow R\ x)$
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 $((x\ y :: R) \Rrightarrow S\ x\ y)\ f\ g$
and $\bigwedge x\ y.\ R\ x\ y \Longrightarrow ((x'\ y' :: T\ x\ y) \Rrightarrow U\ x\ y\ x'\ y')\ f'\ g'$
and $\bigwedge x\ y.\ R\ x\ y \Longrightarrow S\ x\ y\ (f\ x)\ (g\ y) \Longrightarrow T\ x\ y\ (f\ x)\ (g\ y)$
shows $((x\ y :: R) \Rrightarrow U\ x\ y\ (f\ x)\ (g\ y))\ (f' \circ f)\ (g' \circ g)$
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 $((x : P) \Rrightarrow R\ x)\ f\ g$
and $\bigwedge x.\ P\ x \Longrightarrow ((x'\ y' :: S\ x) \Rrightarrow T\ x\ x'\ y')\ f'\ g'$
and $\bigwedge x.\ P\ x \Longrightarrow R\ x\ (f\ x)\ (g\ x) \Longrightarrow S\ x\ (f\ x)\ (g\ x)$
shows $((x : P) \Rrightarrow T\ x\ (f\ x)\ (g\ x))\ (f' \circ f)\ (g' \circ g)$
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*:
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 rel-restrict-leftI Dep-Fun-Rel-relI*)
(*auto dest*!: *Dep-Fun-Rel-relD*)

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

1.2.3 Functions on Predicates

theory *Predicate-Functions*
imports
Functions-Base
begin

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

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

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

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

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

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

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

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

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

lemma *pred-ifE* [*elim*]:
assumes (*pred-if B P*) *x*

obtains $\neg B \mid B\ P\ x$
using *assms* **unfolding** *pred-if-pred-def* **by** *blast*

lemma *pred-ifD*:
assumes (*pred-if B P*) *x*
and *B*
shows *P x*
using *assms* **by** *blast*

end

1.2.4 Orders

theory *Predicates-Order*
imports
HOL.Orderings
begin

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

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

lemma *le-predE*:
assumes $P \leq Q$
and *P x*
obtains *Q x*
using *assms* **by** *blast*

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

end

Monotonicity

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

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

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

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

bundle *dep-mono-wrt-syntax* **begin**
syntax
-mono-wrt :: $'a \Rightarrow 'b \Rightarrow 'c \Rightarrow bool$ $((\text{-}) \Rrightarrow_m (\text{-})\ [41, 40]\ 40)$
-dep-mono-wrt-rel :: $idt \Rightarrow idt \Rightarrow 'a \Rightarrow 'b \Rightarrow 'c \Rightarrow bool$
$('(\text{-}/\ \text{-}/ ::/\ \text{-}') \Rrightarrow_m (\text{-})\ [41, 41, 41, 40]\ 40)$
-dep-mono-wrt-rel-if :: $idt \Rightarrow idt \Rightarrow 'a \Rightarrow bool \Rightarrow 'b \Rightarrow 'c \Rightarrow bool$
$('(\text{-}/\ \text{-}/ ::/\ \text{-}/ |/\ \text{-}') \Rrightarrow_m (\text{-})\ [41, 41, 41, 40, 40]\ 40)$
-dep-mono-wrt-pred :: $idt \Rightarrow 'a \Rightarrow 'b \Rightarrow 'c \Rightarrow bool$ $('(\text{-}/ :/\ \text{-}') \Rrightarrow_m (\text{-})\ [41, 41, 40]\ 40)$
-dep-mono-wrt-pred-if :: $idt \Rightarrow 'a \Rightarrow bool \Rightarrow 'b \Rightarrow 'c \Rightarrow bool$
$('(\text{-}/ :/\ \text{-}/ |/\ \text{-}') \Rrightarrow_m (\text{-})\ [41, 41, 40, 40]\ 40)$
end
bundle *no-dep-mono-wrt-syntax* **begin**
no-syntax
-mono-wrt :: $'a \Rightarrow 'b \Rightarrow 'c \Rightarrow bool$ $((\text{-}) \Rrightarrow_m (\text{-})\ [41, 40]\ 40)$
-dep-mono-wrt-rel :: $idt \Rightarrow idt \Rightarrow 'a \Rightarrow 'b \Rightarrow 'c \Rightarrow bool$
$('(\text{-}/\ \text{-}/ ::/\ \text{-}') \Rrightarrow_m (\text{-})\ [41, 41, 41, 40]\ 40)$
-dep-mono-wrt-rel-if :: $idt \Rightarrow idt \Rightarrow 'a \Rightarrow bool \Rightarrow 'b \Rightarrow 'c \Rightarrow bool$
$('(\text{-}/\ \text{-}/ ::/\ \text{-}/ |/\ \text{-}') \Rrightarrow_m (\text{-})\ [41, 41, 41, 40, 40]\ 40)$
-dep-mono-wrt-pred :: $idt \Rightarrow 'a \Rightarrow 'b \Rightarrow 'c \Rightarrow bool$ $('(\text{-}/ :/\ \text{-}') \Rrightarrow_m (\text{-})\ [41, 41, 40]\ 40)$
-dep-mono-wrt-pred-if :: $idt \Rightarrow 'a \Rightarrow bool \Rightarrow 'b \Rightarrow 'c \Rightarrow bool$
$('(\text{-}/ :/\ \text{-}/ |/\ \text{-}') \Rrightarrow_m (\text{-})\ [41, 41, 40, 40]\ 40)$
end
unbundle *dep-mono-wrt-syntax*
translations
$R \Rrightarrow_m S \rightleftharpoons CONST\ mono\text{-}wrt\ R\ S$
$(x\ y :: R) \Rrightarrow_m S \rightleftharpoons CONST\ dep\text{-}mono\text{-}wrt\ R\ (\lambda x\ y.\ S)$
$(x\ y :: R \mid B) \Rrightarrow_m S \rightleftharpoons CONST\ dep\text{-}mono\text{-}wrt\ R\ (\lambda x\ y.\ CONST\ rel\text{-}if\ B\ S)$
$(x : P) \Rrightarrow_m Q \rightleftharpoons CONST\ dep\text{-}mono\text{-}wrt\ P\ (\lambda x.\ Q)$
$(x : P \mid B) \Rrightarrow_m Q \rightleftharpoons CONST\ dep\text{-}mono\text{-}wrt\ P\ (\lambda x.\ CONST\ pred\text{-}if\ B\ Q)$

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

definition *mono-wrt-rel* $(R :: 'a \Rightarrow 'a \Rightarrow bool)\ (S :: 'b \Rightarrow 'b \Rightarrow bool) \equiv$
$((\text{-}\ \text{-} :: R) \Rrightarrow_m S) :: ('a \Rightarrow 'b) \Rightarrow bool$
adhoc-overloading *mono-wrt mono-wrt-rel*

definition *dep-mono-wrt-pred* $(P :: 'a \Rightarrow bool)$ $(Q :: 'a \Rightarrow 'b \Rightarrow bool)$ $(f :: 'a \Rightarrow 'b) \equiv$
$((x : P) \Rrightarrow (\lambda(\text{-} :: 'b).\ Q\ x))\ f\ f$
adhoc-overloading *dep-mono-wrt dep-mono-wrt-pred*

definition *mono-wrt-pred* $(P :: 'a \Rightarrow bool)$ $(Q :: 'b \Rightarrow bool) \equiv$
$(((\text{-} :: 'a) : P) \Rrightarrow_m Q) :: ('a \Rightarrow 'b) \Rightarrow bool$
adhoc-overloading *mono-wrt mono-wrt-pred*

lemma *mono-wrt-rel-eq-dep-mono-wrt-rel*:
$((R :: 'a \Rightarrow 'a \Rightarrow bool) \Rrightarrow_m (S :: 'b \Rightarrow 'b \Rightarrow bool)) = ((\text{-}\ \text{-} :: R) \Rrightarrow_m S)$
unfolding *mono-wrt-rel-def* **by** *simp*

lemma *mono-wrt-rel-eq-dep-mono-wrt-rel-uhint* [*uhint*]:
assumes $R \equiv R'$
and $S' \equiv (\lambda(\text{-} :: 'a)\ (\text{-} :: 'a).\ S)$
shows $((R :: 'a \Rightarrow 'a \Rightarrow bool) \Rrightarrow_m (S :: 'b \Rightarrow 'b \Rightarrow bool)) = ((x\ y :: R') \Rrightarrow_m S'\ x\ y)$
using *assms* **by** (*simp add*: *mono-wrt-rel-eq-dep-mono-wrt-rel*)

lemma *mono-wrt-rel-iff-dep-mono-wrt-rel*:
$((R :: 'a \Rightarrow 'a \Rightarrow bool) \Rrightarrow_m (S :: 'b \Rightarrow 'b \Rightarrow bool))\ f \longleftrightarrow$
dep-mono-wrt $R\ (\lambda(\text{-} :: 'a)\ (\text{-} :: 'a).\ S)\ (f :: 'a \Rightarrow 'b)$
by (*simp add*: *mono-wrt-rel-eq-dep-mono-wrt-rel*)

lemma *mono-wrt-pred-eq-dep-mono-wrt-pred*:
$((P :: 'a \Rightarrow bool) \Rrightarrow_m (Q :: 'b \Rightarrow bool)) = (((\text{-} :: 'a) : P) \Rrightarrow_m Q)$
unfolding *mono-wrt-pred-def* **by** *simp*

lemma *mono-wrt-pred-eq-dep-mono-wrt-pred-uhint* [*uhint*]:
assumes $P \equiv P'$
and $Q' \equiv (\lambda(\text{-} :: 'a).\ Q)$
shows $((P :: 'a \Rightarrow bool) \Rrightarrow_m (Q :: 'b \Rightarrow bool)) = (((x : P') \Rrightarrow_m Q'\ x) :: ('a \Rightarrow 'b) \Rightarrow bool)$
using *assms* **by** (*simp add*: *mono-wrt-pred-eq-dep-mono-wrt-pred*)

lemma *mono-wrt-pred-iff-dep-mono-wrt-pred*:
$((P :: 'a \Rightarrow bool) \Rrightarrow_m (Q :: 'b \Rightarrow bool))\ f \longleftrightarrow (((\text{-} :: 'a) : P) \Rrightarrow_m Q)\ (f :: 'a \Rightarrow 'b)$
by (*simp add*: *mono-wrt-pred-eq-dep-mono-wrt-pred*)

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

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

using *assms* **unfolding** *dep-mono-wrt-rel-def* **by** *blast*

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

lemma *mono-wrt-relI* [*intro*]:
assumes $\bigwedge x\ y.\ R\ x\ y \Longrightarrow S\ (f\ x)\ (f\ y)$
shows $(R \Rrightarrow_m S)\ f$
using *assms* **by** (*urule dep-mono-wrt-relI*)

lemma *mono-wrt-relE*:
assumes $(R \Rrightarrow_m S)\ f$
obtains $\bigwedge x\ y.\ R\ x\ y \Longrightarrow S\ (f\ x)\ (f\ y)$
using *assms* **by** (*urule* (*e*) *dep-mono-wrt-relE*)

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

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*:
assumes $((x : P) \Rrightarrow_m Q\ x)\ f$
obtains $\bigwedge x.\ P\ x \Longrightarrow Q\ x\ (f\ x)$
using *assms* **unfolding** *dep-mono-wrt-pred-def* **by** *blast*

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

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

lemma *mono-wrt-predE*:
assumes $(P \Rrightarrow_m Q)\ f$
obtains $\bigwedge x.\ P\ x \Longrightarrow Q\ (f\ x)$
using *assms* **by** (*urule* (*e*) *dep-mono-wrt-predE*)

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

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

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

lemma *Dep-Fun-Rel-rel-self-if-dep-mono-wrt-rel*:
assumes $((x\ y :: R) \Rightarrow_m S\ x\ y)\ f$
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 $((x : P) \Rightarrow_m Q\ x)\ f$
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*:
$((x\ y :: R) \Rrightarrow S\ x\ y)\ f\ f \longleftrightarrow ((x\ y :: R) \Rightarrow_m S\ x\ y)\ f$
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*:
$((x : P) \Rrightarrow (\lambda(\text{-} :: 'b).\ Q\ x))\ f\ f \longleftrightarrow ((x : P) \Rightarrow_m Q\ x)\ f$
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*]:
$((y\ x :: R^{-1}) \Rightarrow_m (S\ x\ y)^{-1}) = ((x\ y :: R) \Rightarrow_m S\ x\ y)$
by (*intro ext*) *force*

lemma *in-dom-if-rel-if-dep-mono-wrt-rel*:
assumes $((x\ y :: R) \Rightarrow_m S\ x\ y)\ f$
and $R\ x\ y$

shows *in-dom* (*S x y*) (*f x*)
using *assms* **by** (*intro in-domI*) *blast*

end

context
fixes *R* :: *'a* ⇒ *'a* ⇒ *bool* **and** *f* :: *'a* ⇒ *'b*
begin

corollary *in-dom-if-in-dom-if-mono-wrt-rel*:
assumes (*R* ⇛$_m$ *S*) *f*
shows (*in-dom R* ⇛$_m$ *in-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*) ⇛$_m$ *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* ⇛$_m$ *S*) *f*
shows (*in-codom R* ⇛$_m$ *in-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* ⇛$_m$ *S*) *f*
shows (*in-field R* ⇛$_m$ *in-field S*) *f*
using *assms* **by** *fast*

lemma *le-rel-map-if-mono-wrt-rel*:
assumes (*R* ⇛$_m$ *S*) *f*
shows *R* ≤ *rel-map f S*
using *assms* **by** (*intro le-relI*) *auto*

lemma *le-pred-map-if-mono-wrt-pred*:
assumes (*P* ⇛$_m$ *Q*) *f*
shows *P* ≤ *pred-map f Q*
using *assms* **by** (*intro le-predI*) *auto*

lemma *mono-wrt-rel-if-le-rel-map*:
assumes *R* ≤ *rel-map f S*
shows (*R* ⇛$_m$ *S*) *f*
using *assms* **by** (*intro mono-wrt-relI*) *auto*

lemma *mono-wrt-pred-if-le-pred-map*:
assumes *P* ≤ *pred-map f Q*
shows (*P* ⇛$_m$ *Q*) *f*

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

corollary *mono-wrt-rel-iff-le-rel-map*: $(R \Rightarrow_m S)\ f \longleftrightarrow R \leq \textit{rel-map}\ f\ S$
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*: $(P \Rightarrow_m Q)\ f \longleftrightarrow P \leq \textit{pred-map}\ f\ Q$
using *mono-wrt-pred-if-le-pred-map le-pred-map-if-mono-wrt-pred* **by** *auto*

end

definition *mono* :: $((\prime a :: \textit{ord}) \Rightarrow (\prime b :: \textit{ord})) \Rightarrow \textit{bool}$
$\equiv (((\leq) :: \prime a \Rightarrow \prime a \Rightarrow \textit{bool}) \Rightarrow_m ((\leq) :: \prime b \Rightarrow \prime b \Rightarrow \textit{bool}))$

lemma *mono-eq-mono-wrt-le* [*simp*]: (*mono* :: $((\prime a :: \textit{ord}) \Rightarrow (\prime b :: \textit{ord})) \Rightarrow \textit{bool}$)
=
$(((\leq) :: \prime a \Rightarrow \prime a \Rightarrow \textit{bool}) \Rightarrow_m ((\leq) :: \prime b \Rightarrow \prime b \Rightarrow \textit{bool}))$
unfolding *mono-def* **by** *simp*

lemma *mono-eq-mono-wrt-le-uhint* [*uhint*]:
assumes $R \equiv (\leq) :: \prime a \Rightarrow \prime a \Rightarrow \textit{bool}$
and $S \equiv (\leq) :: \prime b \Rightarrow \prime b \Rightarrow \textit{bool}$
shows *mono* :: $((\prime a :: \textit{ord}) \Rightarrow (\prime b :: \textit{ord})) \Rightarrow \textit{bool} \equiv (R \Rightarrow_m S)$
using *assms* **by** *simp*

lemma *mono-iff-mono-wrt-le* [*iff*]: $\textit{mono}\ f \longleftrightarrow ((\leq) \Rightarrow_m (\leq))\ f$ **by** *simp*

lemma *monoI* [*intro*]:
assumes $\bigwedge x\ y.\ x \leq y \Longrightarrow f\ x \leq f\ y$
shows *mono f*
using *assms* **by** (*urule mono-wrt-relI*)

lemma *monoE* [*elim*]:
assumes *mono f*
obtains $\bigwedge x\ y.\ x \leq y \Longrightarrow f\ x \leq f\ y$
using *assms* **by** (*urule* (*e*) *mono-wrt-relE*)

lemma *monoD*:
assumes *mono f*
and $x \leq y$
shows $f\ x \leq f\ y$
using *assms* **by** (*urule mono-wrt-relD*)

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

lemma *antimono-eq-mono-wrt-le-ge* [*simp*]: (*antimono* :: $((\prime a :: \textit{ord}) \Rightarrow (\prime b :: \textit{ord}))$
$\Rightarrow \textit{bool}) =$
$(((\leq) :: \prime a \Rightarrow \prime a \Rightarrow \textit{bool}) \Rightarrow_m ((\geq) :: \prime b \Rightarrow \prime b \Rightarrow \textit{bool}))$
unfolding *antimono-def* **by** *simp*

lemma *antimono-eq-mono-wrt-le-ge-uhint* [*uhint*]:
assumes $R \equiv (\leq) :: 'a \Rightarrow 'a \Rightarrow bool$
and $S \equiv (\geq) :: 'b \Rightarrow 'b \Rightarrow bool$
shows $antimono :: (('a :: ord) \Rightarrow ('b :: ord)) \Rightarrow bool \equiv (R \Rrightarrow_m S)$
using *assms* **by** *simp*

lemma *antimono-iff-mono-wrt-le-ge* [*iff*]: $antimono\ f \longleftrightarrow ((\leq) \Rrightarrow_m (\geq))\ f$ **by** *simp*

lemma *antimonoI* [*intro*]:
assumes $\bigwedge x\ y.\ x \leq y \Longrightarrow f\ x \geq f\ y$
shows $antimono\ f$
by (*urule mono-wrt-relI*) (*urule assms*)

lemma *antimonoE* [*elim*]:
assumes $antimono\ f$
obtains $\bigwedge x\ y.\ x \leq y \Longrightarrow f\ x \geq f\ y$
using *assms* **by** (*urule* (*e*) *mono-wrt-relE*)

lemma *antimonoD*:
assumes $antimono\ f$
and $x \leq y$
shows $f\ x \geq f\ y$
using *assms* **by** (*urule mono-wrt-relD*)

lemma *antimono-Dep-Fun-Rel-rel-left*: $antimono\ (\lambda(R :: 'a \Rightarrow 'b \Rightarrow bool).\ ((x\ y :: R) \Rrightarrow S\ x\ y))$
by (*intro antimonoI*) *auto*

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

lemma *antimono-dep-mono-wrt-rel-left*: $antimono\ (\lambda(R :: 'a \Rightarrow 'a \Rightarrow bool).\ ((x\ y :: R) \Rrightarrow_m S\ x\ y))$
by (*intro antimonoI*) *blast*

lemma *antimono-dep-mono-wrt-pred-left*: $antimono\ (\lambda(P :: 'a \Rightarrow bool).\ ((x : P) \Rrightarrow_m Q\ x))$
by (*intro antimonoI*) *blast*

lemma *Dep-Fun-Rel-rel-if-le-left-if-Dep-Fun-Rel-rel*:
fixes $R\ R' :: 'a \Rightarrow 'b \Rightarrow bool$
assumes $((x\ y :: R) \Rrightarrow S\ x\ y)\ f\ g$
and $R' \leq R$
shows $((x\ y :: R) \Rrightarrow S\ x\ y)\ f\ g$
using *assms* **by** *blast*

lemma *Dep-Fun-Rel-pred-if-le-left-if-Dep-Fun-Rel-pred*:
fixes $P\ P' :: 'a \Rightarrow bool$
assumes $((x : P) \Rrightarrow Q\ x)\ f\ g$
and $P' \leq P$
shows $((x : P') \Rrightarrow Q\ x)\ f\ g$
using *assms* **by** *blast*

lemma *dep-mono-wrt-rel-if-le-left-if-dep-mono-wrt-rel*:
fixes $R\ R' :: 'a \Rightarrow 'a \Rightarrow bool$
assumes $((x\ y :: R) \Rrightarrow_m S\ x\ y)\ f$
and $R' \leq R$
shows $((x\ y :: R') \Rrightarrow_m S\ x\ y)\ f$
using *assms* **by** *blast*

lemma *dep-mono-wrt-pred-if-le-left-if-dep-mono-wrt-pred*:
fixes $P\ P' :: 'a \Rightarrow bool$
assumes $((x : P) \Rrightarrow_m Q\ x)\ f$
and $P' \leq P$
shows $((x : P') \Rrightarrow_m Q\ x)\ f$
using *assms* **by** *blast*

lemma *mono-Dep-Fun-Rel-rel-right*: *mono* $(\lambda(S :: 'a \Rightarrow 'b \Rightarrow 'c \Rightarrow 'd \Rightarrow bool).\ ((x\ y :: R) \Rrightarrow S\ x\ y))$
by (*intro monoI*) *blast*

lemma *mono-Dep-Fun-Rel-pred-right*: *mono* $(\lambda(Q :: 'a \Rightarrow 'b \Rightarrow 'c \Rightarrow bool).\ ((x : P) \Rrightarrow Q\ x))$
by (*intro monoI*) *blast*

lemma *mono-dep-mono-wrt-rel-right*: *mono* $(\lambda(S :: 'a \Rightarrow 'a \Rightarrow 'b \Rightarrow 'b \Rightarrow bool).\ ((x\ y :: R) \Rrightarrow_m S\ x\ y))$
by (*intro monoI*) *blast*

lemma *mono-dep-mono-wrt-pred-right*: *mono* $(\lambda(Q :: 'a \Rightarrow 'b \Rightarrow bool).\ ((x : P) \Rrightarrow_m Q\ x))$
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 $((x\ y :: R) \Rightarrow_m S\ x\ y)\ f$
and $\bigwedge x\ y.\ R\ x\ y \Longrightarrow S\ x\ y\ (f\ x)\ (f\ y) \Longrightarrow T\ x\ y\ (f\ x)\ (f\ y)$
shows $((x\ y :: R) \Rightarrow_m T\ x\ y)\ f$
using *assms* **by** (*intro dep-mono-wrt-relI*) *blast*

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

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

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

lemma *dep-mono-wrt-pred-comp-dep-mono-wrt-rel-compI*:
assumes $((x : P) \Rightarrow_m Q\ x)\ f$
and $\bigwedge x.\ P\ x \Longrightarrow ((x'\ y' :: R\ x) \Rightarrow_m S\ x\ x'\ y')\ f'$
and $\bigwedge x.\ P\ x \Longrightarrow Q\ x\ (f\ x) \Longrightarrow R\ x\ (f\ x)\ (f\ x)$
shows $((x : P) \Rightarrow_m (\lambda y.\ S\ x\ (f\ x)\ (f\ x)\ y\ y))\ (f' \circ f)$
using *assms* **by** (*intro dep-mono-wrt-predI*) *force*

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

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

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

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

Instantiations **lemma** *mono-wrt-rel-self-id*:
fixes $R :: 'a \Rightarrow 'a \Rightarrow bool$
shows $(R \Rrightarrow_m R)\ (id :: 'a \Rightarrow 'a)$
by *auto*

lemma *mono-wrt-pred-self-id*:
fixes $P :: 'a \Rightarrow bool$
shows $(P \Rrightarrow_m P)\ (id :: 'a \Rightarrow 'a)$
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-comp*: $((\leq) \Rrightarrow_m (\leq) \Rrightarrow (\leq))\ (\circ\circ)$ **by** (*intro mono-wrt-relI Fun-Rel-relI le-relI*) *fast*
lemma *mono-rel-inv*: *mono rel-inv* **by** (*intro monoI*) *fast*

lemma *mono-rel-restrict-left*:
$((\leq) \Rrightarrow_m (\leq) \Rrightarrow (\leq))$ (*rel-restrict-left* $:: ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow ('a \Rightarrow bool) \Rightarrow 'a \Rightarrow 'b \Rightarrow bool$)
by (*intro mono-wrt-relI Fun-Rel-relI le-relI*) *fastforce*

lemma *mono-rel-restrict-right*:
$((\leq) \Rrightarrow_m (\leq) \Rrightarrow (\leq))$ (*rel-restrict-right* $:: ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow ('b \Rightarrow bool) \Rightarrow 'a \Rightarrow 'b \Rightarrow bool$)
by (*intro mono-wrt-relI Fun-Rel-relI le-relI*) *fastforce*

lemma *mono-ball*: $((\geq) \Rrightarrow_m (\leq) \Rrightarrow (\leq))$ *ball* **by** *fast*
lemma *mono-bex*: $((\leq) \Rrightarrow_m (\leq) \Rrightarrow (\leq))$ *bex* **by** *fast*

end

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

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

bundle *rev-implies-syntax* **begin notation** *rev-implies* (**infixr** $\longleftarrow$ *25*) **end**
bundle *no-rev-implies-syntax* **begin no-notation** *rev-implies* (**infixr** $\longleftarrow$ *25*)
end
unbundle *rev-implies-syntax*

lemma *rev-imp-eq-imp-inv* [*simp*]: $(\longleftarrow) = (\longrightarrow)^{-1}$
unfolding *rev-implies-def* **by** *simp*

lemma *rev-impI* [*intro*!]:
assumes $Q \Longrightarrow P$
shows $P \longleftarrow Q$
using *assms* **by** *auto*

lemma *rev-impD* [*dest*!]:
assumes $P \longleftarrow Q$
shows $Q \Longrightarrow P$
using *assms* **by** *auto*

lemma *rev-imp-iff-imp*: $(P \longleftarrow Q) \longleftrightarrow (Q \longrightarrow P)$ **by** *auto*

end

Injective

theory *Binary-Relations-Injective*
imports
Functions-Monotone
Reverse-Implies
begin

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

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

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

lemma *antimono-rel-injective-on*:
$((\leq) \Rightarrow\!\!\!\Rightarrow_m (\leq) \Rightarrow\!\!\!\Rightarrow (\geq))$ (*rel-injective-on* :: $('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool$)
by (*intro mono-wrt-relI*) (*auto dest*: *rel-injective-onD intro*!: *rel-injective-onI*)

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

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

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

lemma *rel-injective-on-if-Fun-Rel-imp-if-rel-injective-at*:
  assumes *rel-injective-at* $(Q :: 'b \Rightarrow bool)$ $(R :: 'a \Rightarrow 'b \Rightarrow bool)$
  and $(R \Rrightarrow (\longrightarrow))\ P\ Q$
  shows *rel-injective-on P R*
  using *assms* **by** (*intro rel-injective-onI*) (*auto dest*: *rel-injective-atD*)

lemma *rel-injective-at-if-Fun-Rel-rev-imp-if-rel-injective-on*:
  assumes *rel-injective-on* $(P :: 'a \Rightarrow bool)$ $(R :: 'a \Rightarrow 'b \Rightarrow bool)$
  and $(R \Rrightarrow (\longleftarrow))\ P\ Q$
  shows *rel-injective-at Q R*
  using *assms* **by** (*intro rel-injective-atI*) (*auto dest*: *rel-injective-onD*)

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

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

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

lemma *rel-injective-eq-rel-injective-on-uhint* [*uhint*]:

assumes $P \equiv (\top :: 'a \Rightarrow bool)$
shows *rel-injective* $:: ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow - \equiv$ *rel-injective-on* P
using *assms* **by** (*simp add*: *rel-injective-eq-rel-injective-on*)

lemma *rel-injectiveI* [*intro*]:
assumes $\bigwedge x\ x'\ y.\ R\ x\ y \Longrightarrow R\ x'\ y \Longrightarrow x = x'$
shows *rel-injective* R
using *assms* **by** (*urule rel-injective-onI*)

lemma *rel-injectiveD*:
assumes *rel-injective* R
and $R\ x\ y\ R\ x'\ y$
shows $x = x'$
using *assms* **by** (*urule* (*d*) *rel-injective-onD* **where** *chained* = *insert*) *simp-all*

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

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

lemma *rel-injective-on-if-rel-injective*:
fixes $P :: 'a \Rightarrow bool$ **and** $R :: 'a \Rightarrow 'b \Rightarrow bool$
assumes *rel-injective* R
shows *rel-injective-on* $P\ R$
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* ⟷ *rel-injective R*
using *rel-injective-if-rel-injective-at-in-codom rel-injective-at-if-rel-injective*
by *blast*

lemma *rel-injective-on-compI*:
fixes *P* :: *'a* ⇒ *bool*
assumes *rel-injective* (*R* :: *'a* ⇒ *'b* ⇒ *bool*)
and *rel-injective-on* (*in-codom R* ⊓ *in-dom S*) (*S* :: *'b* ⇒ *'c* ⇒ *bool*)
shows *rel-injective-on P* (*R* ∘∘ *S*)
proof (*rule rel-injective-onI*)
fix *x x' y*
assume *P x P x'* (*R* ∘∘ *S*) *x y* (*R* ∘∘ *S*) *x' y*
then obtain *z z'* **where** *R x z S z y R x' z' S z' y* **by** *blast*
with *assms* **have** *z* = *z'* **by** (*auto dest*: *rel-injective-onD*)
with ‹*R x z*› ‹*R x' z'*› *assms* **show** *x* = *x'* **by** (*auto dest*: *rel-injectiveD*)
qed

end

1.2.5 Agreement

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

consts *rel-agree-on* :: *'a* ⇒ *'b* ⇒ *bool*

overloading
rel-agree-on-pred ≡ *rel-agree-on* :: (*'a* ⇒ *bool*) ⇒ ((*'a* ⇒ *'b* ⇒ *bool*) ⇒ *bool*) ⇒ *bool*
begin
definition *rel-agree-on-pred* (*P* :: *'a* ⇒ *bool*) (*R* :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ *bool*) ≡
∀ *R R'* : *R*. *R*↾*P* = *R'*↾*P*
end

lemma *rel-agree-onI* [*intro*]:
assumes ⋀*R R' x y*. *R R* ⟹ *R R'* ⟹ *P x* ⟹ *R x y* ⟹ *R' x y*
shows *rel-agree-on P R*
using *assms* **unfolding** *rel-agree-on-pred-def* **by** *blast*

lemma *rel-agree-onE*:
assumes *rel-agree-on P R*
obtains ⋀*R R'*. *R R* ⟹ *R R'* ⟹ *R*↾*P* = *R'*↾*P*
using *assms* **unfolding** *rel-agree-on-pred-def* **by** *blast*

lemma *rel-restrict-left-eq-if-rel-agree-onI*:
assumes *rel-agree-on* P $\mathcal{R}$
and $\mathcal{R}$ R $\mathcal{R}$ R'
shows $R{\restriction}_P = R'{\restriction}_P$
using *assms* **by** (*blast elim*: *rel-agree-onE*)

lemma *rel-agree-onD*:
assumes *rel-agree-on* P $\mathcal{R}$
and $\mathcal{R}$ R $\mathcal{R}$ R'
and P x
and R x y
shows R' x y
using *assms* **by** (*blast elim*: *rel-agree-onE dest*: *fun-cong*)

lemma *antimono-rel-agree-on*:
$((\leq) \Rrightarrow_m (\leq) \Rrightarrow (\geq))$ (*rel-agree-on* :: $('a \Rightarrow bool) \Rightarrow (('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool) \Rightarrow bool$)
by (*intro mono-wrt-relI Fun-Rel-relI*) (*fastforce dest*: *rel-agree-onD*)

lemma *le-if-in-dom-le-if-rel-agree-onI*:
assumes *rel-agree-on* A $\mathcal{R}$
and $\mathcal{R}$ R $\mathcal{R}$ R'
and *in-dom* $R \leq A$
shows $R \leq R'$
using *assms* **by** (*auto dest*: *rel-agree-onD*[**where** *?R*=*R*])

lemma *eq-if-in-dom-le-if-rel-agree-onI*:
assumes *rel-agree-on* A $\mathcal{R}$
and $\mathcal{R}$ R $\mathcal{R}$ R'
and *in-dom* $R \leq A$ *in-dom* $R' \leq A$
shows $R = R'$
using *assms le-if-in-dom-le-if-rel-agree-onI* **by** *blast*

end

1.2.6 Dependent Binary Relations

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

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

bundle *bin-rel-syntax*
begin
syntax *-dep-bin-rel* :: $\langle idt \Rightarrow 'a \Rightarrow 'b \Rightarrow 'c \Rightarrow 'd\rangle$ $(\{\sum\}\text{- : -./ - } [41, 41, 40]\ 51)$

notation *bin-rel* (**infixl** {×} *51*)
end
bundle *no-bin-rel-syntax*
begin
no-syntax *-dep-bin-rel* :: ‹*idt* ⇒ *'a* ⇒ *'b* ⇒ *'c* ⇒ *'d*› ({∑}- : -./ - [*41*, *41*, *40*] *51*)
no-notation *bin-rel* (**infixl** {×} *51*)
end
unbundle *bin-rel-syntax*
translations
{∑}*x* : *A*. *B* ⇌ *CONST dep-bin-rel A* (λ*x*. *B*)

definition *dep-bin-rel-pred* (*A* :: *'a* ⇒ *bool*) (*B* :: *'a* ⇒ *'b* ⇒ *bool*) (*R* :: *'a* ⇒ *'b* ⇒ *bool*) ≡
∀*x y*. *R x y* ⟶ *A x* ∧ *B x y*
adhoc-overloading *dep-bin-rel dep-bin-rel-pred*

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

lemma *bin-rel-pred-eq-dep-bin-rel-pred*: *A* {×} *B* = {∑}- : *A*. *B*
unfolding *bin-rel-pred-def* **by** *auto*

lemma *bin-rel-pred-eq-dep-bin-rel-pred-uhint* [*uhint*]:
assumes *A* ≡ *A'*
and *B'* ≡ (λ-. *B*)
shows *A* {×} *B* ≡ {∑}*x* : *A'*. *B' x*
using *assms* **by** (*simp add*: *bin-rel-pred-eq-dep-bin-rel-pred*)

lemma *bin-rel-pred-iff-dep-bin-rel-pred*: (*A* {×} *B*) *R* ⟷ ({∑}- : *A*. *B*) *R*
unfolding *bin-rel-pred-eq-dep-bin-rel-pred* **by** *auto*

lemma *dep-bin-relI* [*intro*]:
assumes ⋀*x y*. *R x y* ⟹ *A x*
and ⋀*x y*. *R x y* ⟹ *A x* ⟹ *B x y*
shows ({∑}*x* : *A*. *B x*) *R*
using *assms* **unfolding** *dep-bin-rel-pred-def* **by** *auto*

lemma *dep-bin-rel-if-bin-rel-and*:
assumes ⋀*x y*. *R x y* ⟹ *A x* ∧ *B x y*
shows ({∑}*x* : *A*. *B x*) *R*
using *assms* **by** *auto*

lemma *dep-bin-relE* [*elim*]:
assumes ({∑}*x* : *A*. *B x*) *R*
and *R x y*
obtains *A x B x y*

using *assms* **unfolding** *dep-bin-rel-pred-def* **by** *auto*

lemma *dep-bin-relE'*:
assumes $(\{\sum\}x : A.\ B\ x)\ R$
obtains $\bigwedge x\ y.\ R\ x\ y \Longrightarrow A\ x \wedge B\ x\ y$
using *assms* **by** *auto*

lemma *bin-relI* [*intro*]:
assumes $\bigwedge x\ y.\ R\ x\ y \Longrightarrow A\ x$
and $\bigwedge x\ y.\ R\ x\ y \Longrightarrow A\ x \Longrightarrow B\ y$
shows $(A\ \{\times\}\ B)\ R$
using *assms* **by** (*urule dep-bin-relI* **where** *chained* = *fact*)

lemma *bin-rel-if-bin-rel-and*:
assumes $\bigwedge x\ y.\ R\ x\ y \Longrightarrow A\ x \wedge B\ y$
shows $(A\ \{\times\}\ B)\ R$
using *assms* **by** (*urule dep-bin-rel-if-bin-rel-and*)

lemma *bin-relE* [*elim*]:
assumes $(A\ \{\times\}\ B)\ R$
and $R\ x\ y$
obtains $A\ x\ B\ y$
using *assms* **by** (*urule* (*e*) *dep-bin-relE*)

lemma *bin-relE'*:
assumes $(A\ \{\times\}\ B)\ R$
obtains $\bigwedge x\ y.\ R\ x\ y \Longrightarrow A\ x \wedge B\ y$
using *assms* **by** (*urule* (*e*) *dep-bin-relE'*)

lemma *dep-bin-rel-cong* [*cong*]:
$[\![A = A'; \bigwedge x.\ A'\ x \Longrightarrow B\ x = B'\ x]\!] \Longrightarrow (\{\sum\}x : A.\ B\ x) = \{\sum\}x : A'.\ B'\ x$
by (*intro ext iffI dep-bin-relI*) *fastforce+*

lemma *le-dep-bin-rel-if-le-dom*:
assumes $A \leq A'$
shows $(\{\sum\}x : A.\ B\ x) \leq (\{\sum\}x : A'.\ B\ x)$
using *assms* **by** (*intro le-predI dep-bin-relI*) *auto*

lemma *dep-bin-rel-covariant-codom*:
assumes $(\{\sum\}x : A.\ B\ x)\ R$
and $\bigwedge x\ y.\ R\ x\ y \Longrightarrow A\ x \Longrightarrow B\ x\ y \Longrightarrow B'\ x\ y$
shows $(\{\sum\}x : A.\ B'\ x)\ R$
using *assms* **by** (*intro dep-bin-relI*) *auto*

lemma *mono-dep-bin-rel*: $((\leq) \Rightarrow_m (\leq) \Rightarrow (\geq) \Rightarrow (\leq))$ *dep-bin-rel*
by (*intro mono-wrt-relI Fun-Rel-relI dep-bin-relI*) *force*

lemma *mono-bin-rel*: $((\leq) \Rightarrow_m (\leq) \Rightarrow (\geq) \Rightarrow (\leq))\ (\{\times\})$
by (*intro mono-wrt-relI Fun-Rel-relI*) *auto*

lemma *in-dom-le-if-dep-bin-rel*:
assumes $(\{\sum\}x : A.\ B\ x)\ R$
shows *in-dom* $R \leq A$
using *assms* **by** *auto*

lemma *in-codom-le-in-codom-on-if-dep-bin-rel*:
assumes $(\{\sum\}x : A.\ B\ x)\ R$
shows *in-codom* $R \leq$ *in-codom-on* $A\ B$
using *assms* **by** *fast*

lemma *rel-restrict-left-eq-self-if-dep-bin-rel* [*simp*]:
assumes $(\{\sum\}x : A.\ B\ x)\ R$
shows $R{\restriction}_A = R$
using *assms rel-restrict-left-eq-self-if-in-dom-le* **by** *auto*

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

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

lemma *mono-bin-rel-dep-bin-rel-bin-rel-rel-comp*:
$(A\ \{\times\}\ B \Rrightarrow_m (\{\sum\}x : B.\ C\ x) \Rrightarrow_m A\ \{\times\}$ *in-codom-on* $B\ C)\ (\circ\circ)$
by *fastforce*

lemma *mono-dep-bin-rel-bin-rel-rel-inv*: $((\{\sum\}x : A.\ B\ x) \Rrightarrow_m$ *in-codom-on* $A\ B$
$\{\times\}\ A)$ *rel-inv*
by *force*

lemma *mono-bin-rel-rel-inv*: $(A\ \{\times\}\ B \Rrightarrow_m B\ \{\times\}\ A)$ *rel-inv*
by *auto*

lemma *mono-dep-bin-rel-top-dep-bin-rel-inf-rel-restrict-left*:
$((\{\sum\}x : A.\ B\ x) \Rrightarrow_m (P : \top) \Rrightarrow_m (\{\sum\}x : A \sqcap P.\ B\ x))$ *rel-restrict-left*
by *fast*

lemma *mono-dep-bin-rel-top-dep-bin-rel-inf-rel-restrict-right*:
$((\{\sum\}x : A.\ B\ x) \Rrightarrow_m (P : \top) \Rrightarrow_m (\{\sum\}x : A.\ (B\ x) \sqcap P))$ *rel-restrict-right*
by *fast*

lemma *le-if-rel-agree-on-if-dep-bin-relI*:
assumes $(\{\sum\}x : A.\ B\ x)\ R$
and *rel-agree-on* $A\ \mathcal{R}$
and $\mathcal{R}\ R\ \mathcal{R}\ R'$
shows $R \leq R'$
using *assms* **by** (*intro le-if-in-dom-le-if-rel-agree-onI in-dom-le-if-dep-bin-rel*)

lemma *eq-if-rel-agree-on-if-dep-bin-relI*:

assumes $(\{\sum\}x : A.\ B\ x)\ R\ (\{\sum\}x : A.\ B'\ x)\ R'$
and *rel-agree-on* $A\ \mathcal{R}$
and $\mathcal{R}\ R\ \mathcal{R}\ R'$
shows $R = R'$
using *assms* **by** (*intro eq-if-in-dom-le-if-rel-agree-onI in-dom-le-if-dep-bin-rel*)

end

1.2.7 Extensions

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

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

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

lemma *extend-leftI* [*iff*]: $(extend\ x\ y\ R)\ x\ y$
unfolding *extend-rel-def* **by** *blast*

lemma *extend-rightI* [*intro*]:
assumes $R\ x'\ y'$
shows $(extend\ x\ y\ R)\ x'\ y'$
unfolding *extend-rel-def* **using** *assms* **by** *blast*

lemma *extendE* [*elim*]:
assumes $(extend\ x\ y\ R)\ x'\ y'$
obtains $x = x'\ y = y' \mid x \neq x' \vee y \neq y'\ R\ x'\ y'$
using *assms* **unfolding** *extend-rel-def* **by** *blast*

lemma *extend-eq-self-if-rel* [*simp*]: $R\ x\ y \Longrightarrow extend\ x\ y\ R = R$
by *auto*

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

lemma *in-dom-extend-eq*: $in\text{-}dom\ (extend\ x\ y\ R) = in\text{-}dom\ R \sqcup (=)\ x$
by (*intro ext*) *auto*

lemma *in-dom-extend-iff*: $in\text{-}dom\ (extend\ x\ y\ R)\ x' \longleftrightarrow in\text{-}dom\ R\ x' \vee x = x'$
by (*auto simp*: *in-dom-extend-eq*)

lemma *codom-extend-eq*: $in\text{-}codom\ (extend\ x\ y\ R) = in\text{-}codom\ R \sqcup (=)\ y$
by (*intro ext*) *auto*

lemma *in-codom-extend-iff*: *in-codom* (*extend x y R*) $y' \longleftrightarrow$ *in-codom R y'* $\vee$ $y = y'$
by (*auto simp*: *codom-extend-eq*)

end

lemma *in-field-extend-eq*: *in-field* (*extend x y R*) = *in-field R* $\sqcup$ (=) *x* $\sqcup$ (=) *y*
by (*intro ext*) *auto*

lemma *in-field-extend-iff*: *in-field* (*extend x y R*) *z* $\longleftrightarrow$ *in-field R z* $\vee$ *z* = *x* $\vee$ *z* = *y*
by (*auto simp*: *in-field-extend-eq*)

lemma *mono-extend*: *mono* (*extend x y* :: (*'a* $\Rightarrow$ *'b* $\Rightarrow$ *bool*) $\Rightarrow$ *'a* $\Rightarrow$ *'b* $\Rightarrow$ *bool*)
by (*intro monoI*) *force*

lemma *dep-mono-dep-bin-rel-extend*:
((*x* : *A*) $\Rrightarrow_m$ *B x* $\Rrightarrow_m$ ($\{\sum\}$*x* : *A'*. *B' x*) $\Rrightarrow_m$ ($\{\sum\}$*x* : *A* $\sqcup$ *A'*. (*B* $\sqcup$ *B'*) *x*)) *extend*
by *fastforce*

consts *glue* :: *'a* $\Rightarrow$ *'b*

definition *glue-rel* ($\mathcal{R}$ :: (*'a* $\Rightarrow$ *'b* $\Rightarrow$ *bool*) $\Rightarrow$ *bool*) *x* $\equiv$ *in-codom-on* $\mathcal{R}$ (λ*R*. *R x*)
adhoc-overloading *glue glue-rel*

lemma *glue-rel-eq-in-codom-on*: *glue* $\mathcal{R}$ *x* = *in-codom-on* $\mathcal{R}$ (λ*R*. *R x*)
unfolding *glue-rel-def* **by** *simp*

lemma *glueI* [*intro*]:
assumes $\mathcal{R}$ *R*
and *R x y*
shows *glue* $\mathcal{R}$ *x y*
using *assms* **unfolding** *glue-rel-def* **by** *blast*

lemma *glueE* [*elim*!]:
assumes *glue* $\mathcal{R}$ *x y*
obtains *R* **where** $\mathcal{R}$ *R R x y*
using *assms* **unfolding** *glue-rel-def* **by** *blast*

lemma *glue-bottom-eq* [*simp*]: *glue* ($\bot$ :: (*'a* $\Rightarrow$ *'b* $\Rightarrow$ *bool*) $\Rightarrow$ *bool*) = $\bot$
by (*intro ext*) *auto*

lemma *glue-eq-rel-eq-self* [*simp*]: *glue* ((=) *R*) = (*R* :: *'a* $\Rightarrow$ *'b* $\Rightarrow$ *bool*)
by (*intro ext*) *auto*

lemma *glue-sup-eq-glue-sup-glue* [*simp*]: *glue* (*A* $\sqcup$ *B*) = *glue A* $\sqcup$ *glue B*
supply *glue-rel-eq-in-codom-on*[*simp*]

by (*rule ext*) (*urule in-codom-on-sup-pred-eq-in-codom-on-sup-in-codom-on*)

lemma *mono-glue*: *mono* (*glue* :: ((*'a* ⇒ *'b* ⇒ *bool*) ⇒ *bool*) ⇒ (*'a* ⇒ *'b* ⇒ *bool*))
by *auto*

lemma *dep-bin-rel-glueI*:
fixes $\mathcal{R}$ **defines** [*simp*]: *D* ≡ *in-codom-on* $\mathcal{R}$ *in-dom*
assumes ⋀*R*. $\mathcal{R}$ *R* ⟹ ∃ *A*. ({∑}*x* : *A*. *B x*) *R*
shows ({∑}*x* : *D*. *B x*) (*glue* $\mathcal{R}$)
using *assms* **by** (*intro dep-bin-relI*) *auto*

end

Right Unique

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

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

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

lemma *right-unique-onI* [*intro*]:
assumes ⋀*x y y'*. *P x* ⟹ *R x y* ⟹ *R x y'* ⟹ *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*

lemma *antimono-right-unique-on*:
((≤) ⇛$_m$ (≤) ⇛ (≥)) (*right-unique-on* :: (*'a* ⇒ *bool*) ⇒ (*'a* ⇒ *'b* ⇒ *bool*) ⇒ *bool*)
by (*fastforce dest*: *right-unique-onD*)

lemma *mono-right-unique-on-top-right-unique-on-inf-rel-restrict-left*:

((*R* : *right-unique-on P*) ⇛$_m$ (*P'* : ⊤) ⇛$_m$ *right-unique-on* (*P* ⊓ *P'*)) *rel-restrict-left*
by (*fast dest*: *right-unique-onD*)

lemma *mono-right-unique-on-comp*:
((*R* : *right-unique-on P*) ⇛$_m$ *right-unique-on* (*in-codom* (*R*↑$_P$)) ⇛$_m$ *right-unique-on P*) (∘∘)
by (*fast dest*: *right-unique-onD*)

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

lemma *right-unique-on-glue-if-right-unique-on-sup*:
assumes ⋀*R R'*. $\mathcal{R}$ *R* ⟹ $\mathcal{R}$ *R'* ⟹ *right-unique-on P* (*R* ⊔ *R'*)
shows *right-unique-on P* (*glue* $\mathcal{R}$)
using *assms* **by** (*fastforce dest*: *right-unique-onD*)

lemma *right-unique-on-if-right-unique-on-glue*:
assumes *right-unique-on P* (*glue* $\mathcal{R}$)
and $\mathcal{R}$ *R*
shows *right-unique-on P R*
using *assms* **by** (*intro right-unique-onI*) (*auto dest*: *right-unique-onD*)

end

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

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

lemma *right-unique-atI* [*intro*]:
assumes ⋀*x y y'*. *P y* ⟹ *P y'* ⟹ *R x y* ⟹ *R x y'* ⟹ *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* ⇒ *bool*) (*R*$^{-1}$ :: *'b* ⇒ *'a* ⇒ *bool*) ⟷ *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*)

lemma *right-unique-on-if-Fun-Rel-imp-if-right-unique-at*:
assumes *right-unique-at* (Q :: $'b \Rightarrow bool$) (R :: $'a \Rightarrow 'b \Rightarrow bool$)
and ($R \Rrightarrow (\longrightarrow)$) P Q
shows *right-unique-on* P R
using *assms* **by** (*intro right-unique-onI*) (*auto dest*: *right-unique-atD*)

lemma *right-unique-at-if-Fun-Rel-rev-imp-if-right-unique-on*:
assumes *right-unique-on* (P :: $'a \Rightarrow bool$) (R :: $'a \Rightarrow 'b \Rightarrow bool$)
and ($R \Rrightarrow (\longleftarrow)$) P Q
shows *right-unique-at* Q R
using *assms* **by** (*intro right-unique-atI*) (*auto dest*: *right-unique-onD*)

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

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

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

lemma *right-unique-eq-right-unique-on-uhint* [*uhint*]:
assumes $P \equiv (\top$ :: $'a \Rightarrow bool)$
shows *right-unique* :: ($'a \Rightarrow 'b \Rightarrow bool$) $\Rightarrow$ - $\equiv$ *right-unique-on* P
using *assms* **by** (*simp only*: *right-unique-eq-right-unique-on*)

lemma *right-uniqueI* [*intro*]:
assumes $\bigwedge x\ y\ y'.\ R\ x\ y \Longrightarrow R\ x\ y' \Longrightarrow y = y'$
shows *right-unique R*
using *assms* **by** (*urule right-unique-onI*)

lemma *right-uniqueD*:
assumes *right-unique R*
and *R x y R x y'*
shows $y = y'$
using *assms* **by** (*urule* (*d*) *right-unique-onD* **where** *chained* = *insert*) *simp-all*

lemma *right-unique-eq-right-unique-at*:
(*right-unique* :: ('*a* ⇒ '*b* ⇒ *bool*) ⇒ *bool*) = *right-unique-at* (⊤ :: '*b* ⇒ *bool*)
by (*intro ext iffI right-uniqueI*) (*auto dest*: *right-unique-atD right-uniqueD*)

lemma *right-unique-eq-right-unique-at-uhint* [*uhint*]:
assumes $P \equiv (\top :: 'b \Rightarrow bool)$
shows *right-unique* :: ('*a* ⇒ '*b* ⇒ *bool*) ⇒ - ≡ *right-unique-at P*
using *assms* **by** (*simp only*: *right-unique-eq-right-unique-at*)

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

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

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

end

1.2.8 Evaluation of Functions as Binary Relations

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

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

definition *eval-rel* $R\ x \equiv$ *THE* y. $R\ x\ y$
adhoc-overloading *eval eval-rel*

bundle *eval-syntax* **begin notation** *eval* ((-‘-) [*999*, *1000*] *999*) **end**
bundle *no-eval-syntax* **begin no-notation** *eval* ((-‘-) [*999*, *1000*] *999*) **end**
unbundle *eval-syntax*

lemma *eval-eq-if-right-unique-onI*:
assumes *right-unique-on* $P\ R$
and $P\ x$
and $R\ x\ y$
shows $R‘x = y$
using *assms* **unfolding** *eval-rel-def* **by** (*auto dest*: *right-unique-onD*)

lemma *eval-eq-if-right-unique-on-eqI*:
assumes *right-unique-on* ((=) x) R
and $R\ x\ y$
shows $R‘x = y$
using *assms* **by** (*auto intro*: *eval-eq-if-right-unique-onI*)

lemma *rel-eval-if-ex1*:
assumes $\exists!y.\ R\ x\ y$

shows $R\ x\ (R`x)$
using *assms* **unfolding** *eval-rel-def* **by** (*rule theI'*)

lemma *rel-if-eval-eq-if-in-dom-if-right-unique-on-eq*:
assumes *right-unique-on* $((=)\ x)\ R$
and *in-dom* $R\ x$
and $R`x = y$
shows $R\ x\ y$
using *assms* **by** (*blast intro*: *rel-eval-if-ex1*[*of* $R\ x$] *dest*: *right-unique-onD*)

corollary *rel-eval-if-in-dom-if-right-unique-on-eq*:
assumes *right-unique-on* $((=)\ x)\ R$
and *in-dom* $R\ x$
shows $R\ x\ (R`x)$
using *assms* **by** (*rule rel-if-eval-eq-if-in-dom-if-right-unique-on-eq*) *simp*

lemma *eval-app-eq-eq* [*simp*]: $(\lambda x.\ (=)\ (f\ x))`x = f\ x$
by (*auto intro*: *eval-eq-if-right-unique-onI*)

lemma *extend-eval-eq-if-not-in-dom* [*simp*]:
assumes $\neg$(*in-dom* $R\ x$)
shows (*extend* $x\ y\ R$)$`x = y$
using *assms* **by** (*force intro*: *eval-eq-if-right-unique-on-eqI*)

corollary *extend-bottom-eval-eq* [*simp*]:
fixes $x :: 'a$ **and** $y :: 'b$
shows (*extend* $x\ y\ (\bot :: 'a \Rightarrow 'b \Rightarrow bool)$)$`x = y$
by (*intro extend-eval-eq-if-not-in-dom*) *auto*

lemma *glue-eval-eqI*:
assumes *right-unique-on* P (*glue* $\mathcal{R}$)
and $\mathcal{R}\ R$
and $P\ x$
and $R\ x\ y$
shows (*glue* $\mathcal{R}$)$`x = y$
using *assms* **by** (*auto intro*: *eval-eq-if-right-unique-onI*)

lemma *glue-eval-eq-evalI*:
assumes *right-unique-on* P (*glue* $\mathcal{R}$)
and $\mathcal{R}\ R$
and $P\ x$
and *in-dom* $R\ x$
shows (*glue* $\mathcal{R}$)$`x = R`x$
using *assms* **by** (*intro glue-eval-eqI*[*of* $P\ \mathcal{R}\ R$])
(*auto intro*: *rel-if-eval-eq-if-in-dom-if-right-unique-on-eq*[*of* x] *dest*: *right-unique-onD*)

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

lemma *extend-eval-eq-if-neq* [*simp*]:
fixes $R :: 'a \Rightarrow 'b \Rightarrow bool$

shows $x \neq y \Longrightarrow (extend\ y\ z\ R)`x = R`x$
unfolding *extend-rel-def eval-rel-def* **by** *auto*

lemma *sup-eval-eq-if-not-in-dom-left* [*simp*]:
fixes $R\ S :: 'a \Rightarrow 'b \Rightarrow bool$
shows $\neg(in\text{-}dom\ S\ x) \Longrightarrow (R \sqcup S)`x = R`x$
unfolding *eval-rel-def* **by** (*cases* $\exists\, y.\ S\ x\ y$) *auto*

lemma *sup-eval-eq-if-not-in-dom-right* [*simp*]:
fixes $R\ S :: 'a \Rightarrow 'b \Rightarrow bool$
shows $\neg(in\text{-}dom\ R\ x) \Longrightarrow (R \sqcup S)`x = S`x$
unfolding *eval-rel-def* **by** (*cases* $\exists\, y.\ R\ x\ y$) *auto*

lemma *rel-restrict-left-eval-eq-if-pred* [*simp*]:
fixes $R :: 'a \Rightarrow 'b \Rightarrow bool$
assumes $P\ x$
shows $(R\restriction_P)`x = R`x$

using *assms* **unfolding** *eval-rel-def rel-restrict-left-pred-def* **by** *auto*

end

Left Total

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

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

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* $P\ R \equiv \forall\, x : P.\ in\text{-}dom\ R\ x$
end

lemma *left-total-onI* [*intro*]:
assumes $\bigwedge x.\ P\ x \Longrightarrow in\text{-}dom\ R\ x$
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 *le-in-dom-if-left-total-on*:

assumes *left-total-on P R*
shows $P \leq$ *in-dom R*
using *assms* **by** *force*

lemma *left-total-on-if-le-in-dom*:
assumes $P \leq$ *in-dom R*
shows *left-total-on P R*
using *assms* **by** *fastforce*

lemma *mono-left-total-on*:
$((\geq) \Rrightarrow_m (\leq) \Rrightarrow (\leq))$ (*left-total-on* :: $('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool$)
by *fastforce*

lemma *le-in-dom-iff-left-total-on*: $P \leq$ *in-dom R* $\longleftrightarrow$ *left-total-on P R*
using *le-in-dom-if-left-total-on left-total-on-if-le-in-dom* **by** *auto*

lemma *mono-left-total-on-top-left-total-on-inf-rel-restrict-left*:
$((R :$ *left-total-on P*$) \Rrightarrow_m (P' : \top) \Rrightarrow_m$ *left-total-on* $(P \sqcap P'))$ *rel-restrict-left*
by *fast*

lemma *mono-left-total-on-comp*:
$((R :$ *left-total-on P*$) \Rrightarrow_m$ *left-total-on* (*in-codom* $(R\restriction_P)) \Rrightarrow_m$ *left-total-on P*$)$ $(\circ\circ)$
by *fast*

consts *left-total* :: $'a \Rightarrow bool$

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

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

lemma *left-total-eq-left-total-on-uhint* [*uhint*]:
assumes $P \equiv \top :: 'a \Rightarrow bool$
shows *left-total* :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow$ - $\equiv$ *left-total-on P*
using *assms* **by** (*simp add*: *left-total-eq-left-total-on*)

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

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

using *assms* **by** (*urule* (*e*) *left-total-onE* **where** *chained* = *insert*) *simp*

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* ⇒ *bool* **and** *R* :: *'a* ⇒ *'b* ⇒ *bool*
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

1.2.9 Functions as Binary Relations

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

Function relations may contain further elements outside their specification.

consts *rel-dep-fun* :: *'a* ⇒ (*'b* ⇒ *'c*) ⇒ *'d* ⇒ *bool*
consts *rel-fun* :: *'a* ⇒ *'b* ⇒ *'c* ⇒ *bool*

bundle *rel-fun-syntax*
begin
syntax
-rel-fun :: *'a* ⇒ *'b* ⇒ *'c* ⇒ *bool* ((-) → (-) [*41*, *40*] *40*)
-rel-dep-fun :: *idt* ⇒ *'a* ⇒ *'b* ⇒ *'c* ⇒ *bool* (*'*(-/ :/ -*'*) → (-) [*41*, *41*, *40*] *40*)
end
bundle *no-rel-fun-syntax*
begin
no-syntax
-rel-fun :: *'a* ⇒ *'b* ⇒ *'c* ⇒ *bool* ((-) → (-) [*41*, *40*] *40*)
-rel-dep-fun :: *idt* ⇒ *'a* ⇒ *'b* ⇒ *'c* ⇒ *bool* (*'*(-/ :/ -*'*) → (-) [*41*, *41*, *40*] *40*)
end
unbundle *rel-fun-syntax*
translations
A → *B* ⇌ *CONST rel-fun A B*
(*x* : *A*) → *B* ⇌ *CONST rel-dep-fun A* (λ*x*. *B*)

definition *rel-dep-fun-pred* (*A* :: *'a* ⇒ *bool*) (*B* :: *'a* ⇒ *'b* ⇒ *bool*) (*R* :: *'a* ⇒ *'b* ⇒ *bool*) ≡
left-total-on A R ∧ *right-unique-on A R* ∧ ((*x* : *A*) ⇛$_m$ *B x*) (*eval R*)

adhoc-overloading *rel-dep-fun rel-dep-fun-pred*

definition *rel-fun-pred* $(A :: 'a \Rightarrow bool)$ $(B :: 'b \Rightarrow bool) :: ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool \equiv$
  rel-dep-fun-pred A $(\lambda(\text{-} :: 'a).\ B)$
adhoc-overloading *rel-fun rel-fun-pred*

lemma *rel-fun-eq-rel-dep-fun*:
  $(((A :: 'a \Rightarrow bool) \to (B :: 'b \Rightarrow bool)) :: ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool) = (((\text{-} :: 'a) : A) \to B)$
  by (*simp add*: *rel-fun-pred-def*)

lemma *rel-fun-eq-rel-dep-fun-uhint* [*uhint*]:
  assumes $(A :: 'a \Rightarrow bool) \equiv A'$
  and $B' \equiv (\lambda(\text{-} :: 'a).\ (B :: 'b \Rightarrow bool))$
  shows $(A \to B) \equiv ((x : A') \to B'\ x)$
  using *assms* **by** (*simp add*: *rel-fun-eq-rel-dep-fun*)

lemma *rel-fun-iff-rel-dep-fun*:
  $((A :: 'a \Rightarrow bool) \to (B :: 'b \Rightarrow bool))\ (R :: 'a \Rightarrow 'b \Rightarrow bool) \longleftrightarrow (((\text{-} :: 'a) : A) \to B)\ R$
  by (*simp add*: *rel-fun-pred-def*)

lemma *rel-dep-funI* [*intro*]:
  assumes *left-total-on A R*
  and *right-unique-on A R*
  and $((x : A) \Rrightarrow_m B\ x)$ $(eval\ R)$
  shows $((x : A) \to B\ x)\ R$
  using *assms* **unfolding** *rel-dep-fun-pred-def* **by** *auto*

lemma *rel-dep-funE* [*elim*]:
  assumes $((x : A) \to B\ x)\ R$
  obtains *left-total-on A R right-unique-on A R* $((x : A) \Rrightarrow_m B\ x)$ $(eval\ R)$
  using *assms* **unfolding** *rel-dep-fun-pred-def* **by** *auto*

lemma *rel-dep-fun-cong* [*cong*]:
  assumes $\bigwedge x.\ A\ x \longleftrightarrow A'\ x$
  and $\bigwedge x\ y.\ A'\ x \Longrightarrow B\ x\ y \longleftrightarrow B'\ x\ y$
  shows $((x : A) \to B\ x) = ((x : A') \to B'\ x)$
  using *assms* **by** (*intro ext*) (*auto intro*!: *rel-dep-funI left-total-onI*
    dep-mono-wrt-predI intro: *right-unique-onD elim*!: *rel-dep-funE*)

lemma *rel-funI* [*intro*]:
  assumes *left-total-on A R*
  and *right-unique-on A R*
  and $(A \Rrightarrow_m B)$ $(eval\ R)$
  shows $(A \to B)\ R$
  using *assms* **by** (*urule rel-dep-funI*)

lemma *rel-funE* [*elim*]:
assumes $(A \to B)\ R$
obtains *left-total-on A R right-unique-on A R* $(A \Rrightarrow_m B)$ *(eval R)*
using *assms* **by** (*urule* (*e*) *rel-dep-funE*)

lemma *mono-rel-dep-fun-mono-wrt-pred-eval*: $(((x : A) \to B\ x) \Rrightarrow_m (x : A) \Rrightarrow_m B\ x)$ *eval*
by *auto*

lemma *ex1-rel-right-if-rel-dep-funI*:
assumes $((x : A) \to B\ x)\ R$
and $A\ x$
shows $\exists!y.\ R\ x\ y$
using *assms* **by** (*blast dest*: *right-unique-onD*)

lemma *eq-if-rel-if-rel-if-rel-dep-funI*:
assumes $((x : A) \to B\ x)\ R$
and $A\ x$
and $R\ x\ y\ R\ x\ y'$
shows $y = y'$
using *assms* **by** (*blast dest*: *right-unique-onD*)

lemma *eval-eq-if-rel-if-rel-dep-funI* [*simp*]:
assumes $((x : A) \to B\ x)\ R$
and $A\ x$
and $R\ x\ y$
shows $R`x = y$
using *assms* **by** (*intro eq-if-rel-if-rel-if-rel-dep-funI*[*OF assms*, *symmetric*])
(*blast intro*!: *rel-eval-if-ex1*[**where** *?R*=*R*] *ex1-rel-right-if-rel-dep-funI*)

lemma *rel-if-eval-eq-if-rel-dep-funI*:
assumes $((x : A) \to B\ x)\ R$
and $A\ x$
and $R`x = y$
shows $R\ x\ y$
by (*rule rel-if-eval-eq-if-in-dom-if-right-unique-on-eq*[**where** *?R*=*R*])
(*use assms* **in** ‹*blast dest*: *right-unique-onD*›)+

corollary *rel-eval-if-rel-dep-funI*:
assumes $((x : A) \to B\ x)\ R$
and $A\ x$
shows $R\ x\ (R`x)$
using *assms* **by** (*rule rel-if-eval-eq-if-rel-dep-funI*) *simp*

corollary *rel-iff-eval-eq-if-rel-dep-funI*:
assumes $((x : A) \to B\ x)\ R$
and $A\ x$
shows $R\ x\ y \longleftrightarrow R`x = y$
using *assms* **by** (*intro iffI*; *rule eval-eq-if-rel-if-rel-dep-funI rel-if-eval-eq-if-rel-dep-funI*)

lemma *rel-dep-fun-relE*:
assumes $((x : A) \to B\ x)\ R$
and $A\ x$
and $R\ x\ y$
obtains $B\ x\ y\ R`x = y$
proof
from *assms* **show** $R`x = y$ **by** *simp*
with *assms* **show** $B\ x\ y$ **by** *blast*
qed

lemma *rel-dep-fun-relE'*:
assumes $((x : A) \to B\ x)\ R$
obtains $\bigwedge x\ y.\ A\ x \Longrightarrow R\ x\ y \Longrightarrow B\ x\ y \wedge R`x = y$
using *assms* **by** (*auto elim*: *rel-dep-fun-relE*)

lemma *rel-codom-if-rel-if-pred-if-rel-dep-fun*:
assumes $((x : A) \to B\ x)\ R$
and $A\ x$
and $R\ x\ y$
shows $B\ x\ y$
using *assms* **by** (*auto elim*: *rel-dep-fun-relE*)

lemma *rel-dep-fun-contravariant-dom*:
assumes $((x : A) \to B\ x)\ R$
and [*dest*]: $\bigwedge x.\ A'\ x \Longrightarrow A\ x$
shows $((x : A') \to B\ x)\ R$
using *assms* **by** (*fast intro*!: *rel-dep-funI dest*: *right-unique-onD*)

lemma *rel-dep-fun-covariant-codom*:
assumes $((x : A) \to B\ x)\ R$
and $\bigwedge x.\ A\ x \Longrightarrow B\ x\ (R`x) \Longrightarrow B'\ x\ (R`x)$
shows $((x : A) \to B'\ x)\ R$
using *assms* **by** (*fast dest*: *right-unique-onD*)

lemma *rel-fun-in-codom-on-if-rel-dep-fun*:
assumes $((x : A) \to B\ x)\ R$
shows $(A \to \textit{in-codom-on}\ A\ B)\ R$
using *assms* **by** *fastforce*

lemma *comp-eq-eval-restrict-left-le-if-rel-dep-fun*:
assumes $((x : A) \to B\ x)\ R$
shows $((=) \circ \textit{eval}\ R)\restriction_A \leq R\restriction_A$
using *assms* **by** (*intro le-relI*) (*force intro*: *rel-eval-if-rel-dep-funI*)

lemma *restrict-left-le-comp-eq-eval-restrict-left-if-rel-dep-fun*:
assumes $((x : A) \to B\ x)\ R$
shows $R\restriction_A \leq ((=) \circ \textit{eval}\ R)\restriction_A$
using *assms* **by** (*intro le-relI*) *force*

corollary *restrict-left-eq-comp-eq-eval-if-rel-dep-fun*:
assumes $((x : A) \to B\ x)\ R$
shows $R\restriction_A = ((=) \circ eval\ R)\restriction_A$
using *assms comp-eq-eval-restrict-left-le-if-rel-dep-fun*
restrict-left-le-comp-eq-eval-restrict-left-if-rel-dep-fun
by (*intro antisym*) *auto*

lemma *eval-eq-if-rel-dep-funI*:
fixes $R :: 'a \Rightarrow 'b \Rightarrow bool$
assumes $((x : A) \to B\ x)\ R\ ((x : A') \to B'\ x)\ R'$
and $R \leq R'$
and $(A \sqcap A')\ x$
shows $R`x = R'`x$
proof −
from *assms* **have** $R'\ x\ (R`x)\ R'\ x\ (R'`x)$ **by** (*auto intro*: *rel-eval-if-rel-dep-funI*)
with *assms* **show** *?thesis* **by** (*intro eval-eq-if-rel-if-rel-dep-funI*[*symmetric*]) *force*+
qed

lemma *rel-agree-on-if-eval-eq-if-rel-dep-fun*:
assumes *crel-dep-fun*: $\bigwedge R.\ \mathcal{R}\ R \Longrightarrow \exists\, B.\ ((x : A) \to B\ x)\ R$
and $\bigwedge R\ R'\ x.\ \mathcal{R}\ R \Longrightarrow \mathcal{R}\ R' \Longrightarrow A\ x \Longrightarrow R`x = R'`x$
shows *rel-agree-on A* $\mathcal{R}$
proof (*rule rel-agree-onI*)
fix *x y R R'* **assume** *hyps*: $\mathcal{R}\ R\ \mathcal{R}\ R'\ A\ x\ R\ x\ y$
with *crel-dep-fun* **have** $y = R`x$ **by** *fastforce*
also from *assms hyps* **have** $... = R'`x$ **by** *blast*
finally have $y = R'`x$ **.**
moreover from *crel-dep-fun*[*OF* ‹$\mathcal{R}\ R'$›] **obtain** B **where** $((x : A) \to B\ x)\ R'$
by *blast*
ultimately show $R'\ x\ y$ **using** ‹$A\ x$› **by** (*auto intro*: *rel-eval-if-rel-dep-funI*)
qed

lemma *mono-rel-dep-fun-top-rel-dep-fun-inf-rel-restrict-left*:
$(((x : A) \to B\ x) \Rrightarrow_m (A' : \top) \Rrightarrow_m (x : A \sqcap A') \to B\ x)$ *rel-restrict-left*
by (*intro mono-wrt-predI dep-mono-wrt-predI rel-dep-funI*

mono-right-unique-on-top-right-unique-on-inf-rel-restrict-left
[*THEN dep-mono-wrt-predD*, *THEN dep-mono-wrt-predD*]
mono-left-total-on-top-left-total-on-inf-rel-restrict-left
[*THEN dep-mono-wrt-predD*, *THEN dep-mono-wrt-predD*])
auto

end

1.2.10 Clean Functions

theory *Binary-Relations-Clean-Function*
imports

Binary-Relations-Function-Base
begin

Clean function relations may not contain further elements outside their specification.

consts *crel-dep-fun* :: $'a \Rightarrow ('b \Rightarrow 'c) \Rightarrow 'd \Rightarrow bool$
consts *crel-fun* :: $'a \Rightarrow 'b \Rightarrow 'c \Rightarrow bool$

bundle *crel-fun-syntax*
begin
syntax
-crel-fun :: $'a \Rightarrow 'b \Rightarrow 'c \Rightarrow bool$ $((-) \rightarrow_c (-)\ [41, 40]\ 40)$
-crel-dep-fun :: $idt \Rightarrow 'a \Rightarrow 'b \Rightarrow 'c \Rightarrow bool$ $('(-/ :/ -') \rightarrow_c (-)\ [41, 41, 40]\ 40)$
end
bundle *no-crel-fun-syntax*
begin
no-syntax
-crel-fun :: $'a \Rightarrow 'b \Rightarrow 'c \Rightarrow bool$ $((-) \rightarrow_c (-)\ [41, 40]\ 40)$
-crel-dep-fun :: $idt \Rightarrow 'a \Rightarrow 'b \Rightarrow 'c \Rightarrow bool$ $('(-/ :/ -') \rightarrow_c (-)\ [41, 41, 40]\ 40)$
end
unbundle *crel-fun-syntax*
translations
$A \rightarrow_c B \rightleftharpoons CONST\ crel\text{-}fun\ A\ B$
$(x : A) \rightarrow_c B \rightleftharpoons CONST\ crel\text{-}dep\text{-}fun\ A\ (\lambda x.\ B)$

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

definition *crel-fun-pred* $(A :: 'a \Rightarrow bool)\ B \equiv (((- :: 'a) : A) \rightarrow_c B)$
adhoc-overloading *crel-fun crel-fun-pred*

lemma *crel-fun-eq-crel-dep-fun*:
$(((A :: 'a \Rightarrow bool) \rightarrow_c (B :: 'b \Rightarrow bool)) :: ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool) = (((- :: 'a) : A) \rightarrow_c B)$
by (*simp add*: *crel-fun-pred-def*)

lemma *crel-fun-eq-crel-dep-fun-uhint* [*uhint*]:
assumes $(A :: 'a \Rightarrow bool) \equiv A'$
and $B' \equiv (\lambda(- :: 'a).\ (B :: 'b \Rightarrow bool))$
shows $(A \rightarrow_c B) \equiv ((x : A') \rightarrow_c B'\ x)$
using *assms* **by** (*simp add*: *crel-fun-eq-crel-dep-fun*)

lemma *crel-fun-iff-crel-dep-fun*:
$((A :: 'a \Rightarrow bool) \rightarrow_c (B :: 'b \Rightarrow bool))\ (R :: 'a \Rightarrow 'b \Rightarrow bool) \longleftrightarrow (((- :: 'a) : A) \rightarrow_c B)\ R$
by (*simp add*: *crel-fun-pred-def*)

lemma *crel-dep-funI* [*intro*]:

assumes $((x : A) \to B\ x)\ R$
and *in-dom* $R \leq A$
shows $((x : A) \to_c B\ x)\ R$
unfolding *crel-dep-fun-pred-def* **using** *assms*
by (*intro conjI antisym le-in-dom-if-left-total-on*) *auto*

lemma *crel-dep-funI'*:
assumes *left-total-on A R*
and *right-unique-on A R*
and $(\{\sum\}x : A.\ B\ x)\ R$
shows $((x : A) \to_c B\ x)\ R$
proof (*intro crel-dep-funI rel-dep-funI dep-mono-wrt-predI*)
fix *x* **assume** *A x*
with *assms* **obtain** *y* **where** *B x y R x y* **by** *auto*
moreover with *assms* **have** $R`x = y$ **by** (*auto intro*: *eval-eq-if-right-unique-onI*)
ultimately show *B x* (*R`x*) **by** *simp*
qed (*use assms* **in** *auto*)

lemma *crel-dep-funE*:
assumes $((x : A) \to_c B\ x)\ R$
obtains $((x : A) \to B\ x)\ R$ *in-dom R* = *A*
using *assms* **unfolding** *crel-dep-fun-pred-def* **by** *auto*

lemma *crel-dep-funE'* [*elim*]:
notes *crel-dep-funE*[*elim*]
assumes $((x : A) \to_c B\ x)\ R$
obtains $((x : A) \to B\ x)\ R\ (\{\sum\}x : A.\ B\ x)\ R$
proof
show $(\{\sum\}x : A.\ B\ x)\ R$
proof (*rule dep-bin-relI*)
fix *x y* **assume** *R x y A x*
with *assms* **have** $R`x = y\ B\ x\ (R`x)$ **by** *auto*
then show *B x y* **by** *simp*
qed (*use assms* **in** *auto*)
qed (*use assms* **in** *auto*)

lemma *crel-dep-fun-cong* [*cong*]:
assumes $\bigwedge x.\ A\ x \longleftrightarrow A'\ x$
and $\bigwedge x\ y.\ A'\ x \Longrightarrow B\ x\ y \longleftrightarrow B'\ x\ y$
shows $((x : A) \to_c B\ x) = ((x : A') \to_c B'\ x)$
using *assms* **by** (*intro ext*) (*auto intro*!: *crel-dep-funI elim*!: *crel-dep-funE*)

lemma *in-dom-eq-if-crel-dep-fun* [*simp*]:
assumes $((x : A) \to_c B\ x)\ R$
shows *in-dom R* = *A*
using *assms* **by** (*auto elim*: *crel-dep-funE*)

lemma *in-codom-le-in-codom-on-if-crel-dep-fun*:
assumes $((x : A) \to_c B\ x)\ R$

shows *in-codom R* ≤ *in-codom-on A B*
using *assms* **by** *fast*

lemma *crel-funI* [*intro*]:
assumes (*A* → *B*) *R*
and *in-dom R* ≤ *A*
shows (*A* →$_c$ *B*) *R*
using *assms* **by** (*urule crel-dep-funI*)

lemma *crel-funI'*:
assumes *left-total-on A R*
and *right-unique-on A R*
and (*A* {×} *B*) *R*
shows (*A* →$_c$ *B*) *R*
using *assms* **by** (*urule crel-dep-funI'*)

lemma *crel-funE*:
assumes (*A* →$_c$ *B*) *R*
obtains (*A* → *B*) *R in-dom R* = *A*
using *assms* **by** (*urule* (*e*) *crel-dep-funE*)

lemma *crel-funE'* [*elim*]:
assumes (*A* →$_c$ *B*) *R*
obtains (*A* → *B*) *R* (*A* {×} *B*) *R*
using *assms* **by** (*urule* (*e*) *crel-dep-funE'*)

lemma *in-dom-eq-if-crel-fun* [*simp*]:
assumes (*A* →$_c$ *B*) *R*
shows *in-dom R* = *A*
using *assms* **by** (*urule in-dom-eq-if-crel-dep-fun*)

lemma *eq-if-rel-if-rel-if-crel-dep-funI*:
assumes ((*x* : *A*) →$_c$ *B x*) *R*
and *R x y R x y'*
shows *y* = *y'*
using *assms* **by** (*auto intro*: *eq-if-rel-if-rel-if-rel-dep-funI*)

lemma *eval-eq-if-rel-if-crel-dep-funI* [*simp*]:
assumes ((*x* : *A*) →$_c$ *B x*) *R*
and *R x y*
shows *R*‘*x* = *y*
using *assms* **by** (*auto intro*: *eval-eq-if-rel-if-rel-dep-funI*)

lemma *crel-dep-fun-relE*:
assumes ((*x* : *A*) →$_c$ *B x*) *R*
and *R x y*
obtains *A x B x y R*‘*x* = *y*
using *assms* **by** (*auto elim*: *rel-dep-fun-relE*)

lemma *crel-dep-fun-relE'*:
assumes $((x : A) \to_c B\ x)\ R$
obtains $\bigwedge x\ y.\ R\ x\ y \Longrightarrow A\ x \wedge B\ x\ y \wedge R`x = y$
using *assms* **by** (*auto elim*: *crel-dep-fun-relE*)

lemma *rel-restrict-left-eq-self-if-crel-dep-fun* [*simp*]:
assumes $((x : A) \to_c B\ x)\ R$
shows $R\restriction_A = R$
using *assms* **by** *auto*

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

lemma *crel-dep-fun-covariant-codom*:
assumes $((x : A) \to_c B\ x)\ R$
and $\bigwedge x.\ A\ x \Longrightarrow B\ x\ (R`x) \Longrightarrow B'\ x\ (R`x)$
shows $((x : A) \to_c B'\ x)\ R$
using *assms* **by** (*force intro*: *rel-dep-fun-covariant-codom*)

lemma *comp-eq-eval-restrict-left-le-if-crel-dep-fun*:
assumes [*uhint*]: $((x : A) \to_c B\ x)\ R$
shows $((=) \circ eval\ R)\restriction_A \leq R$
supply *rel-restrict-left-eq-self-if-crel-dep-fun*[*uhint*]
by (*urule comp-eq-eval-restrict-left-le-if-rel-dep-fun*) (*use assms* **in** *auto*)

lemma *le-comp-eq-eval-restrict-left-if-rel-dep-fun*:
assumes [*uhint*]: $((x : A) \to_c B\ x)\ R$
shows $R \leq ((=) \circ eval\ R)\restriction_A$
supply *rel-restrict-left-eq-self-if-crel-dep-fun*[*uhint*]
by (*urule restrict-left-le-comp-eq-eval-restrict-left-if-rel-dep-fun*) (*use assms* **in** *auto*)

corollary *restrict-left-eq-comp-eq-eval-if-crel-dep-fun*:
assumes $((x : A) \to_c B\ x)\ R$
shows $R = ((=) \circ eval\ R)\restriction_A$
using *assms comp-eq-eval-restrict-left-le-if-crel-dep-fun*
le-comp-eq-eval-restrict-left-if-rel-dep-fun
by (*intro antisym*) *auto*

lemma *eval-eq-if-crel-dep-fun-if-rel-dep-funI*:
fixes $R :: 'a \Rightarrow 'b \Rightarrow bool$
assumes $((x : A) \to B\ x)\ R\ ((x : A') \to_c B'\ x)\ R'$
and $R \leq R'$
and $A\ x$
shows $R`x = R'`x$
proof −
from *assms* **have** $A'\ x$ **by** (*blast elim*: *crel-dep-fun-relE*)
with *assms* **show** *?thesis* **by** (*blast intro*: *eval-eq-if-rel-dep-funI*)
qed

lemma *crel-dep-fun-ext*:

assumes $((x : A) \to_c B\ x)\ R\ ((x : A) \to_c B'\ x)\ R'$
and $\bigwedge x.\ A\ x \Longrightarrow R`x = R'`x$
shows $R = R'$
using *assms*
by (*intro eq-if-rel-agree-on-if-dep-bin-relI*[**where** *?A*=*A* **and** *?B*=*B* **and** *?*$\mathcal{R}$=(=) $R \sqcup (=)\ R'$]
rel-agree-on-if-eval-eq-if-rel-dep-fun)
auto

lemma *eq-if-le-if-crel-dep-fun-if-rel-dep-fun*:
assumes $((x : A) \to B\ x)\ R\ ((x : A) \to_c B'\ x)\ R'$
and $R \leq R'$
shows $R = R'$
proof (*intro ext iffI*)
fix $x\ y$ **assume** $R'\ x\ y$
with *assms* **have** $R'`x = y\ A\ x$ **by** *auto*
moreover with *assms* **have** $R`x = R'`x$ **by** (*blast intro*: *eval-eq-if-crel-dep-fun-if-rel-dep-funI*)
ultimately show $R\ x\ y$ **using** *assms* **by** (*auto intro*: *rel-if-eval-eq-if-rel-dep-funI*)
qed (*use assms* **in** *auto*)

lemma *ex-dom-crel-dep-fun-iff-crel-dep-fun-in-dom*:
$(\exists (A :: 'a \Rightarrow bool).\ ((x : A) \to_c B\ x)\ R) \longleftrightarrow (((x : in\text{-}dom\ R) \to_c B\ x)\ R)$
by *auto*

lemma *crel-fun-bottom-bottom*: $((\bot :: 'a \Rightarrow bool) \to_c A)\ (\bot :: 'a \Rightarrow 'b \Rightarrow bool)$
by *fastforce*

lemma *crel-dep-fun-bottom-iff-eq-bottom* [*iff*]: $((x : (\bot :: 'a \Rightarrow bool)) \to_c B\ x)\ R \longleftrightarrow R = \bot$
by *fastforce*

lemma *mono-crel-dep-fun-top-crel-dep-fun-inf-rel-restrict-left*:
$(((x : A) \to_c B\ x) \Rrightarrow_m (A' : \top) \Rrightarrow_m (x : A \sqcap A') \to_c B\ x)\ rel\text{-}restrict\text{-}left$
by (*intro mono-wrt-predI dep-mono-wrt-predI crel-dep-funI'*

mono-right-unique-on-top-right-unique-on-inf-rel-restrict-left
[*THEN dep-mono-wrt-predD*, *THEN dep-mono-wrt-predD*]
mono-left-total-on-top-left-total-on-inf-rel-restrict-left
[*THEN dep-mono-wrt-predD*, *THEN dep-mono-wrt-predD*]
mono-dep-bin-rel-top-dep-bin-rel-inf-rel-restrict-left
[*THEN mono-wrt-predD*, *THEN dep-mono-wrt-predD*])
auto

lemma *mono-rel-dep-fun-ge-crel-dep-fun-rel-restrict-left*:
$(((x : A) \to B\ x) \Rrightarrow_m (A' : (\geq)\ A) \Rrightarrow_m (x : A') \to_c B\ x)\ rel\text{-}restrict\text{-}left$
proof (*intro mono-wrt-predI dep-mono-wrt-predI crel-dep-funI*)
fix $A\ A' :: 'a \Rightarrow bool$ **and** B **and** $R :: 'a \Rightarrow 'b \Rightarrow bool$ **assume** $((x : A) \to B\ x)\ R$
with *mono-rel-dep-fun-top-rel-dep-fun-inf-rel-restrict-left* **have** $((x : A \sqcap A') \to$

$B\ x)\ R\restriction_{A'}$
  by *force*
 moreover assume $A' \leq A$
 ultimately show $((x : A') \to B\ x)\ R\restriction_{A'}$ **by** (*simp only*: *inf-absorb2*)
qed *auto*

lemma *crel-dep-fun-eq-restrict*: $((x : (A :: 'a \Rightarrow bool)) \to_c (=)\ x)\ (=)\restriction_A$
 by *fastforce*

end

Symmetric

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

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

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

lemma *symmetric-onI* [*intro*]:
 assumes $\bigwedge x\ y.\ P\ x \Longrightarrow P\ y \Longrightarrow R\ x\ y \Longrightarrow R\ y\ x$
 shows *symmetric-on* $P\ R$
 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* $(P :: 'a \Rightarrow bool)\ (R :: 'a \Rightarrow 'a \Rightarrow bool)$
 by (*blast dest*: *symmetric-onD*)

lemma *antimono-symmetric-on*: *antimono* (*symmetric-on* :: $('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$)
 by (*intro antimonoI*) (*auto dest*: *symmetric-onD*)

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

and $P' \leq P$
shows *symmetric-on P' R*
using *assms antimono-symmetric-on* **by** *blast*

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

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

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

lemma *symmetric-eq-symmetric-on-uhint* [*uhint*]:
$P \equiv (\top :: 'a \Rightarrow bool) \Longrightarrow$ (*symmetric* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow$ -) $\equiv$ *symmetric-on P*
by (*simp add*: *symmetric-eq-symmetric-on*)

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

lemma *symmetricD*:
assumes *symmetric R*
and *R x y*
shows *R y x*
using *assms* **by** (*urule* (*d*) *symmetric-onD* **where** *chained* = *insert*) *simp-all*

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

lemma *symmetric-rel-inv-iff-symmetric* [*iff*]: *symmetric* $R^{-1} \longleftrightarrow$ *symmetric* (*R* :: $'a \Rightarrow 'a \Rightarrow bool$)
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* [*iff*]:
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* ($\top$ :: *'a* $\Rightarrow$ *'a* $\Rightarrow$ *bool*)
by (*rule symmetricI*) *auto*

end

Reflexive

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

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

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

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 R*
by *blast*

lemma *mono-reflexive-on*:
$((\geq) \Rrightarrow_m (\leq) \Rrightarrow (\leq))$ (*reflexive-on* $:: ('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$)
by *fastforce*

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

end

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

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

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

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

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

lemma *reflexive-eq-reflexive-on-uhint* [*uhint*]:
$P \equiv$ ($\top$ :: $'a \Rightarrow$ *bool*) $\Longrightarrow$ (*reflexive* :: $('a \Rightarrow 'a \Rightarrow$ *bool*$) \Rightarrow$ -) $\equiv$ *reflexive-on* P
by (*simp add*: *reflexive-eq-reflexive-on*)

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

lemma *reflexiveD*:
assumes *reflexive* R
shows $R\ x\ x$
using *assms* **by** (*urule* (*d*) *reflexive-onD* **where** *chained* = *insert*) *simp*

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

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

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

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

lemma *reflexive-top*: *reflexive* (⊤ :: *'a* ⇒ *'a* ⇒ *bool*)
by (*rule reflexiveI*) *auto*

end

Transitive

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

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

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

lemma *transitive-onI* [*intro*]:
assumes ⋀*x y z*. *P x* ⟹ *P y* ⟹ *P z* ⟹ *R x y* ⟹ *R y z* ⟹ *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 ⋀*x y*. *P x* ⟹ *P y* ⟹ (*R* ∘∘ *R*) *x y* ⟹ *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* ∘∘ *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} ⟷ *transitive-on* (*P* :: *'a* ⇒ *bool*) (*R* :: *'a* ⇒ *'a* ⇒ *bool*)
by (*auto intro*!: *transitive-onI dest*: *transitive-onD*)

lemma *antimono-transitive-on*: *antimono* (*transitive-on* :: (*'a* ⇒ *bool*) ⇒ (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*)

by (*intro antimonoI*) (*auto dest*: *transitive-onD*)

consts *transitive* :: *'a* ⇒ *bool*

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

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

lemma *transitive-eq-transitive-on-uhint* [*uhint*]:
P ≡ (⊤ :: *'a* ⇒ *bool*) ⟹ (*transitive* :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ -) ≡ *transitive-on P*
by (*simp add*: *transitive-eq-transitive-on*)

lemma *transitiveI* [*intro*]:
assumes ⋀*x y z*. *R x y* ⟹ *R y z* ⟹ *R x z*
shows *transitive R*
using *assms* **by** (*urule transitive-onI*)

lemma *transitiveD* [*dest*]:
assumes *transitive R*
and *R x y R y z*
shows *R x z*
using *assms* **by** (*urule* (*d*) *transitive-onD* **where** *chained* = *insert*) *simp-all*

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

lemma *transitive-if-rel-comp-le-self*:
assumes *R* ∘∘ *R* ≤ *R*
shows *transitive R*
by (*urule transitive-on-if-rel-comp-self-imp*) (*use assms* **in** *auto*)

lemma *rel-comp-le-self-if-transitive*:
assumes *transitive R*
shows *R* ∘∘ *R* ≤ *R*
using *assms* **by** *blast*

corollary *transitive-iff-rel-comp-le-self*: *transitive R* ⟷ *R* ∘∘ *R* ≤ *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* [*iff*]:
transitive-on (*in-field R*) *R* ⟷ *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} ⟷ *transitive* (*R* :: *'a* ⇒ *'a* ⇒ *bool*)
by *fast*

Instantiations **lemma** *transitive-eq*: *transitive* (=)
by (*rule transitiveI*) (*rule trans*)

lemma *transitive-top*: *transitive* (⊤ :: *'a* ⇒ *'a* ⇒ *bool*)
by (*rule transitiveI*) *auto*

end

1.2.11 Preorders

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

definition *preorder-on* ≡ *reflexive-on* ⊓ *transitive-on*

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** *auto*

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** *auto*

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-on P R^{-1} $\longleftrightarrow$ *preorder-on* (*P* :: $'a \Rightarrow bool$) (*R* :: $'a \Rightarrow 'a \Rightarrow bool$)
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*

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

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

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

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

using *assms* **by** (*simp add*: *preorder-eq-preorder-on*)

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

lemma *preorderI* [*intro*]:
assumes *reflexive R*
and *transitive R*
shows *preorder R*
using *assms* **by** (*urule preorder-onI*)

lemma *preorderE* [*elim*]:
assumes *preorder R*
obtains *reflexive R transitive R*
using *assms* **by** (*urule* (*e*) *preorder-onE*)

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

end

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

end

1.2.12 Partial Equivalence Relations

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

definition *partial-equivalence-rel-on* $\equiv$ *transitive-on* $\sqcap$ *symmetric-on*

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

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** *auto*

lemma *partial-equivalence-rel-on-rel-self-if-rel-dom*:
assumes *partial-equivalence-rel-on* (*P* :: *'a* ⇒ *bool*) (*R* :: *'a* ⇒ *'a* ⇒ *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* ⇒ *bool*) (*R* :: *'a* ⇒ *'a* ⇒ *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} ⟷ *partial-equivalence-rel-on* (*P* :: *'a* ⇒ *bool*)
(*R* :: *'a* ⇒ *'a* ⇒ *bool*)
by *blast*

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

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

lemma *partial-equivalence-rel-eq-partial-equivalence-rel-on*:
(*partial-equivalence-rel* :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*) = *partial-equivalence-rel-on*
(⊤ :: *'a* ⇒ *bool*)
unfolding *partial-equivalence-rel-def* **..**

lemma *partial-equivalence-rel-eq-partial-equivalence-rel-on-uhint* [*uhint*]:
assumes *P* ≡ ⊤ :: *'a* ⇒ *bool*
shows (*partial-equivalence-rel* :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*) ≡ *partial-equivalence-rel-on*
P
using *assms* **by** (*simp add*: *partial-equivalence-rel-eq-partial-equivalence-rel-on*)

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

lemma *partial-equivalence-relI* [*intro*]:
assumes *transitive R*

and *symmetric R*
shows *partial-equivalence-rel R*
using *assms* **by** (*urule partial-equivalence-rel-onI*)

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* **by** (*urule* (*e*) *partial-equivalence-rel-onE* **where** *chained* = *insert*)
(*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* ⇒ *bool*
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} ⟷ *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* ∘∘ *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* ⟷ *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*

end

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* (⊤ :: *'a* ⇒ *'a* ⇒ *bool*)
using *transitive-top symmetric-top* **by** (*rule partial-equivalence-relI*)

end

1.2.13 Equivalences

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

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

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

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** *auto*

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 ⟷ *equivalence-rel-on* (*in-field R*) *R*
using *equivalence-rel-on-in-field-if-partial-equivalence-rel* **by** *auto*

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

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

lemma *equivalence-rel-eq-equivalence-rel-on*:
(*equivalence-rel* :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*) = *equivalence-rel-on* (⊤ :: *'a* ⇒ *bool*)
unfolding *equivalence-rel-def* **..**

lemma *equivalence-rel-eq-equivalence-rel-on-uhint* [*uhint*]:
assumes *P* ≡ ⊤ :: *'a* ⇒ *bool*
shows *equivalence-rel* :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool* ≡ *equivalence-rel-on P*
using *assms* **by** (*simp add*: *equivalence-rel-eq-equivalence-rel-on*)

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

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

lemma *equivalence-relE* [*elim*]:
assumes *equivalence-rel R*
obtains *partial-equivalence-rel R reflexive R*
using *assms* **by** (*urule* (*e*) *equivalence-rel-onE*)

lemma *equivalence-rel-on-if-equivalence*:
fixes *P* :: *'a* ⇒ *bool*
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*)

end

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

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

end

Antisymmetric

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

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

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

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

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

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

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

lemma *antisymmetric-eq-antisymmetric-on-uhint* [*uhint*]:
$P \equiv (\top :: 'a \Rightarrow bool) \Longrightarrow$ (*antisymmetric* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow$ -) $\equiv$ *antisymmetric-on* P
by (*simp add*: *antisymmetric-eq-antisymmetric-on*)

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

lemma *antisymmetricD*:
assumes *antisymmetric R*
and *R x y R y x*
shows *x = y*
using *assms* **by** (*urule* (*d*) *antisymmetric-onD* **where** *chained = insert*) *simp-all*

lemma *antisymmetric-on-if-antisymmetric*:
fixes *P* :: *'a ⇒ bool* **and** *R* :: *'a ⇒ 'a ⇒ bool*
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* [*iff*]:
antisymmetric-on (*in-field R*) *R* ⟷ *antisymmetric R*
using *antisymmetric-if-antisymmetric-on-in-field antisymmetric-on-if-antisymmetric*
by *blast*

end

1.2.14 Partial Orders

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

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

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

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** *auto*

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*)

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

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

lemma *partial-order-eq-partial-order-on*:
(*partial-order* :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*) = *partial-order-on* (⊤ :: *'a* ⇒ *bool*)
unfolding *partial-order-def* **..**

lemma *partial-order-eq-partial-order-on-uhint* [*uhint*]:
assumes *P* ≡ ⊤ :: *'a* ⇒ *bool*
shows (*partial-order* :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*) ≡ *partial-order-on P*
using *assms* **by** (*simp add*: *partial-order-eq-partial-order-on*)

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

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

lemma *partial-orderE* [*elim*]:
assumes *partial-order R*
obtains *preorder R antisymmetric R*
using *assms* **by** (*urule* (*e*) *partial-order-onE*)

lemma *partial-order-on-if-partial-order*:
fixes *P* :: *'a* ⇒ *bool*
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

end

1.2.15 Restricted Equality

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

Summary Introduces notations and theorems for restricted equalities. An equality (=) can be restricted to only apply to a subset of its elements. The restriction can be formulated, for example, by a predicate or a set.

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

translations
$(=_P) \rightleftharpoons$ *CONST rel-restrict-left* (=) *P*
$x =_P y \rightleftharpoons$ *CONST rel-restrict-left* (=) *P x y*

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* $((=_P) :: 'a \Rightarrow$ -) **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

1.2.16 Composing Functions

theory *Binary-Relations-Function-Composition*
imports
Binary-Relations-Clean-Function
Restricted-Equality
begin

lemma *dep-bin-rel-eval-rel-comp-if-dep-bin-rel-if-crel-dep-fun*:
assumes $((x : A) \to_c B\ x)\ R$
and $\bigwedge x.\ A\ x \Longrightarrow (\{\sum\} y : B\ x.\ C\ x\ y)\ R'$
shows $(\{\sum\} x : A.\ C\ x\ (R`x))\ (R \circ\circ R')$
using *assms* **by** *force*

lemma *crel-dep-fun-eval-rel-comp-if-rel-dep-fun-if-crel-dep-fun*:
assumes $((x : A) \to_c B\ x)\ R$
and $\bigwedge x.\ A\ x \Longrightarrow ((y : B\ x) \to C\ x\ y)\ R'$
shows $((x : A) \to_c C\ x\ (R`x))\ (R \circ\circ R')$
proof (*intro crel-dep-funI' dep-bin-relI*

mono-left-total-on-comp[*THEN dep-mono-wrt-predD*, *THEN mono-wrt-predD*]
mono-right-unique-on-comp[*THEN dep-mono-wrt-predD*, *THEN mono-wrt-predD*])
from *assms* **show** *left-total-on* (*in-codom* $R\restriction_A$) R' **by** *force*
from *assms* **show** *right-unique-on* (*in-codom* $R\restriction_A$) R' **by** (*fast dest*: *right-unique-onD*)
qed (*use assms* **in** ‹*auto elim*: *rel-dep-fun-relE crel-dep-fun-relE*›)

lemma *rel-comp-eval-eq-if-rel-dep-fun-if-crel-dep-funI* [*simp*]:
assumes $((x : A) \to_c B\ x)\ R$
and $\bigwedge x.\ A\ x \Longrightarrow ((y : B\ x) \to C\ x\ y)\ R'$

and $A\ x$
shows $(R \circ\circ R')`x = R'`(R`x)$
proof (*rule eval-eq-if-right-unique-onI*)
from *assms* **have** $((x : A) \to_c C\ x\ (R`x))\ (R \circ\circ R')$
by (*intro crel-dep-fun-eval-rel-comp-if-rel-dep-fun-if-crel-dep-fun*) *auto*
then show *right-unique-on A* $(R \circ\circ R')$ **by** *blast*
from *assms* **show** $(R \circ\circ R')\ x\ (R'`(R`x))$ **by** (*blast intro*: *rel-eval-if-rel-dep-funI*)
qed (*fact assms*)

lemma *eq-restrict-comp-eq-if-crel-dep-fun* [*simp*]:
assumes $((x : A) \to_c B\ x)\ R$
shows $(=_A) \circ\circ R = R$
using *assms* **by** *force*

lemma *comp-eq-restrict-if-crel-dep-fun* [*simp*]:
assumes $(A \to_c B)\ R$
shows $R \circ\circ (=_B) = R$
using *assms* **by** *force*

end

1.2.17 Extending Functions

theory *Binary-Relations-Function-Extend*
imports
Binary-Relations-Clean-Function
begin

lemma *left-total-on-sup-eq-extend-if-left-total-on*:
assumes *left-total-on A R*
shows *left-total-on* $(A \sqcup (=)\ x)$ (*extend x y R*)
using *assms* **by** *fastforce*

lemma *right-unique-on-sup-eq-extend-if-not-in-dom-if-right-unique-on*:
assumes *right-unique-on A R*
and $\neg$(*in-dom R x*)
shows *right-unique-on* $(A \sqcup (=)\ x)$ (*extend x y R*)
using *assms* **by** (*intro right-unique-onI*) (*auto dest*: *right-unique-onD elim*!: *extendE*)

lemma *rel-dep-fun-extend-if-if-rel-dep-funI*:
assumes $((x : A) \to B\ x)\ R$
and $\neg$(*in-dom R x*)
shows $((x' : A \sqcup ((=)\ x)) \to (\textit{if } x' = x \textit{ then } (=)\ y \textit{ else } B\ x'))$ (*extend x y R*)
using *assms* **by** (*intro rel-dep-funI left-total-on-sup-eq-extend-if-left-total-on right-unique-on-sup-eq-extend-if-not-in-dom-if-right-unique-on dep-mono-wrt-predI*) *auto*

lemma *rel-dep-fun-extend-if-rel-dep-funI*:

assumes $((x : A) \to B\ x)\ R$
and $\neg(\mathit{in\text{-}dom}\ R\ x)$
and $B\ x\ y$
shows $((x' : A \sqcup ((=)\ x)) \to B\ x')\ (\mathit{extend}\ x\ y\ R)$
by (*urule rel-dep-fun-covariant-codom*, *urule rel-dep-fun-extend-if-if-rel-dep-funI*)
(*use assms* **in** ‹*force split*: *if-split-asm*›)+

lemma *crel-dep-fun-extend-if-if-crel-dep-funI*:
assumes $((x : A) \to_c B\ x)\ R$
and $\neg(\mathit{in\text{-}dom}\ R\ x)$
shows $((x' : A \sqcup ((=)\ x)) \to_c (\mathit{if}\ x' = x\ \mathit{then}\ (=)\ y\ \mathit{else}\ B\ x'))\ (\mathit{extend}\ x\ y\ R)$
using *assms* **by** (*intro crel-dep-funI rel-dep-fun-extend-if-if-rel-dep-funI*) *fastforce+*

lemma *crel-dep-fun-extend-if-rel-dep-funI*:
assumes $((x : A) \to_c B\ x)\ R$
and $\neg(\mathit{in\text{-}dom}\ R\ x)$
and $B\ x\ y$
shows $((x' : A \sqcup ((=)\ x)) \to_c B\ x')\ (\mathit{extend}\ x\ y\ R)$
using *assms* **by** (*intro crel-dep-funI rel-dep-fun-extend-if-rel-dep-funI*) *fastforce+*

context
fixes $\mathcal{R} :: ('a \Rightarrow 'b \Rightarrow \mathit{bool}) \Rightarrow \mathit{bool}$ **and** $A :: ('a \Rightarrow 'b \Rightarrow \mathit{bool}) \Rightarrow 'a \Rightarrow \mathit{bool}$ **and** D
defines [*simp*]: $D \equiv \mathit{in\text{-}codom\text{-}on}\ \mathcal{R}\ A$
begin

lemma *rel-dep-fun-glue-if-right-unique-if-rel-dep-fun*:
assumes *funs*: $\bigwedge R.\ \mathcal{R}\ R \Longrightarrow ((x : A\ R) \to B\ x)\ R$
and *runique*: *right-unique-on* D (*glue* $\mathcal{R}$)
shows $((x : D) \to B\ x)$ (*glue* $\mathcal{R}$)
proof (*intro rel-dep-funI dep-mono-wrt-predI left-total-onI*)
fix x **assume** $D\ x$
with *funs* **obtain** R **where** *hyps*: $\mathcal{R}\ R\ A\ R\ x\ ((x : A\ R) \to B\ x)\ R$ **by** *auto*
then have $B\ x$ (R‘x) **by** (*auto elim*: *rel-dep-fun-relE*)
moreover have (*glue* $\mathcal{R}$)‘$x = R$‘x
proof (*intro glue-eval-eqI*)
show $\mathcal{R}\ R$ **by** (*fact hyps*)
then have $A\ R \leq D$ **by** *fastforce*
with *runique* **show** *right-unique-on* ($A\ R$) (*glue* $\mathcal{R}$) **using** *antimono-right-unique-on*
by *blast*
qed (*use hyps* **in** ‹*auto elim*: *rel-dep-fun-relE*›)
ultimately show $B\ x$ (*glue* $\mathcal{R}$‘x) **by** *simp*
qed (*use assms* **in** ‹*fastforce+*›)

lemma *crel-dep-fun-glue-if-right-unique-if-crel-dep-fun*:
assumes $\bigwedge R.\ \mathcal{R}\ R \Longrightarrow ((x : A\ R) \to_c B\ x)\ R$
and *right-unique-on* D (*glue* $\mathcal{R}$)
shows $((x : D) \to_c B\ x)$ (*glue* $\mathcal{R}$)

using *assms* **by** (*intro crel-dep-funI rel-dep-fun-glue-if-right-unique-if-rel-dep-fun*) *fastforce+*

end

lemma *right-unique-on-sup-if-rel-agree-on-sup-if-right-unique-on*:
assumes *right-unique-on P R right-unique-on P R′*
and *rel-agree-on* (*P* ⊓ *in-dom R* ⊓ *in-dom R′*) ((=) *R* ⊔ (=) *R′*)
shows *right-unique-on P* (*R* ⊔ *R′*)
proof (*intro right-unique-onI*)
fix *x y y′* **assume** *P x* (*R* ⊔ *R′*) *x y* (*R* ⊔ *R′*) *x y′*
then obtain *R1 R2* **where** *rels*: *R1 x y R2 x y′* **and** *ors*: *R1* = *R* ∨ *R1* = *R′* *R2* = *R* ∨ *R2* = *R′* **by** *auto*
then consider (*neq*) *R1* ≠ *R2* | *R1* = *R2* **by** *auto*
then show *y* = *y′*
proof *cases*
case *neq*
with *rels ors* **obtain** *z z′* **where** *R x z R′ x z′* **and** *yors*: *y* = *z* ∨ *y* = *z′ y′* = *z* ∨ *y′* = *z′*
by *auto*
with ‹*P x*› **have** *R′ x z* **by** (*intro rel-agree-onD*[**where** *?R*=*R* **and** *?R′*=*R′* **and**
?P=*P* ⊓ *in-dom R* ⊓ *in-dom R′* **and** *?R*=(=) *R* ⊔ (=) *R′*] *assms*)
auto
with ‹*P x*› ‹*R′ x z′*› *assms* **have** *z* = *z′* **by** (*blast dest*: *right-unique-onD*)
with *yors* **show** *y* = *y′* **by** *auto*
qed (*use assms* ‹*P x*› *rels ors* **in** ‹*auto dest*: *right-unique-onD*›)
qed

lemma *crel-dep-fun-sup-if-eval-eq-if-crel-dep-fun*:
assumes *dep-funs*: ((*x* : *A*) →*c B x*) *R* ((*x* : *A′*) →*c B x*) *R′*
and ⋀*x*. *A x* ⟹ *A′ x* ⟹ *R*‘*x* = *R′*‘*x*
shows ((*x* : *A* ⊔ *A′*) →*c B x*) (*R* ⊔ *R′*)
proof −
let *?A* = λ*S*. *if S* = *R then A else A′*
from *dep-funs* **have** *A* = *A′* **if** *R* = *R′* **using** *that* **by** (*simp only*: *flip*:
in-dom-eq-if-crel-dep-fun[*OF dep-funs*(*1*)] *in-dom-eq-if-crel-dep-fun*[*OF dep-funs*(*2*)])
then have [*uhint*]: *in-codom-on* ((=) *R* ⊔ (=) *R′*) *?A* = *A* ⊔ *A′*
by (*intro ext iffI*) (*fastforce split*: *if-splits*)+
show *?thesis* **by** (*urule* (*rr*) *crel-dep-fun-glue-if-right-unique-if-crel-dep-fun*
right-unique-on-sup-if-rel-agree-on-sup-if-right-unique-on
rel-agree-on-if-eval-eq-if-rel-dep-fun)
(*use assms* **in** ‹*auto 7 0 intro*: *rel-dep-fun-contravariant-dom dest*: *right-unique-onD*›)
qed

end

1.2.18 Lambda Abstractions

theory *Binary-Relations-Function-Lambda*
imports *Binary-Relations-Clean-Function*
begin

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

definition *rel-lambda-pred* $A\ f\ x\ y \equiv A\ x \wedge f\ x = y$
adhoc-overloading *rel-lambda rel-lambda-pred*

bundle *rel-lambda-syntax*
begin
syntax
-rel-lambda :: *idt* $\Rightarrow 'a \Rightarrow 'b \Rightarrow 'c$ ((*2*λ- : -./ -) *60*)
end
bundle *no-rel-lambda-syntax*
begin
no-syntax
-rel-lambda :: *idt* $\Rightarrow 'a \Rightarrow 'b \Rightarrow 'c$ ((*2*λ- : -./ -) *60*)
end
unbundle *rel-lambda-syntax*
translations
$\lambda x : A.\ f \rightleftharpoons CONST\ rel\text{-}lambda\ A\ (\lambda x.\ f)$

lemma *rel-lambdaI* [*intro*]:
assumes $A\ x$
and $f\ x = y$
shows $(\lambda x : A.\ f\ x)\ x\ y$
using *assms* **unfolding** *rel-lambda-pred-def* **by** *auto*

lemma *rel-lambda-appI*:
assumes $A\ x$
shows $(\lambda x : A.\ f\ x)\ x\ (f\ x)$
using *assms* **by** *auto*

lemma *rel-lambdaE* [*elim*!]:
assumes $(\lambda x : A.\ f\ x)\ x\ y$
obtains $A\ x\ y = f\ x$
using *assms* **unfolding** *rel-lambda-pred-def* **by** *auto*

lemma *rel-lambda-cong* [*cong*]:
$[\![\bigwedge x.\ A\ x \longleftrightarrow A'\ x;\ \bigwedge x.\ A'\ x \Longrightarrow f\ x = f'\ x]\!] \Longrightarrow (\lambda x : A.\ f\ x) = \lambda x : A'.\ f'\ x$
by (*intro ext*) *auto*

lemma *in-dom-rel-lambda-eq* [*simp*]: *in-dom* $(\lambda x : A.\ f\ x) = A$
by *auto*

lemma *in-codom-rel-lambda-eq-has-inverse-on* [*simp*]: *in-codom* $(\lambda x : A.\ f\ x) =$ *has-inverse-on* $A\ f$

by *fastforce*

lemma *left-total-on-rel-lambda*: *left-total-on A* $(\lambda x : A.\ f\ x)$
by *auto*

lemma *right-unique-on-rel-lambda*: *right-unique-on A* $(\lambda x : A.\ f\ x)$
by *auto*

lemma *cdep-fun-rel-lambda*: $((x : A) \to_c ((=)\ (f\ x)))\ (\lambda x : A.\ f\ x)$
by (*intro crel-dep-funI'*) *auto*

Compare the following with (*rel-dep-fun ?A ?B* $\Rrightarrow_m$ *dep-mono-wrt ?A ?B*) *eval*.

lemma *mono-wrt-pred-mono-crel-dep-fun-rel-lambda*:
$(((x : A) \Rrightarrow_m B\ x) \Rrightarrow_m (x : A) \to_c B\ x)$ (*rel-lambda A*)
by (*intro mono-wrt-predI crel-dep-funI'*) *auto*

lemma *rel-lambda-eval-eq* [*simp*]: $A\ x \Longrightarrow (\lambda x : A.\ f\ x)\text{`}x = f\ x$
by (*rule eval-eq-if-right-unique-onI*) *auto*

lemma *app-eq-if-rel-lambda-eqI*:
assumes $(\lambda x : A.\ f\ x) = (\lambda x : A.\ g\ x)$
and $A\ x$
shows $f\ x = g\ x$
using *assms* **by** (*auto dest*: *fun-cong*)

lemma *crel-dep-fun-inf-rel-lambda-inf-if-rel-dep-fun*:
assumes $((x : A) \to B\ x)\ R$
shows $((x : A \sqcap A') \to_c B\ x)\ (\lambda x : A \sqcap A'.\ R\text{`}x)$
using *assms* **by** *force*

corollary *crel-dep-fun-rel-lambda-if-le-if-rel-dep-fun*:
assumes $((x : A) \to B\ x)\ R$
and [*uhint*]: $A' \leq A$
shows $((x : A') \to_c B\ x)\ (\lambda x : A'.\ R\text{`}x)$
supply *inf-absorb2*[*uhint*]
by (*urule crel-dep-fun-inf-rel-lambda-inf-if-rel-dep-fun*) (*fact assms*)

lemma *rel-lambda-ext*:
assumes $((x : A) \to_c B\ x)\ R$
and $\bigwedge x.\ A\ x \Longrightarrow f\ x = R\text{`}x$
shows $(\lambda x : A.\ f\ x) = R$
using *assms* **by** (*intro ext iffI*) (*auto intro*!: *rel-lambdaI intro*: *rel-eval-if-rel-dep-funI*)

lemma *rel-lambda-eval-eq-if-crel-dep-fun* [*simp*]: $((x : A) \to_c B\ x)\ R \Longrightarrow (\lambda x : A.\ R\text{`}x) = R$
by (*rule rel-lambda-ext*) *auto*

Every element of *crel-dep-fun A B* may be expressed as a lambda abstraction.

lemma *eq-rel-lambda-if-crel-dep-funE*:
assumes $((x : A) \rightarrow_c B\ x)\ R$
obtains f **where** $R = (\lambda x : A.\ f\ x)$
proof
let *?f*=$(\lambda x.\ R`x)$
from *assms* **show** $R = (\lambda x : A.\ (\lambda x.\ R`x)\ x)$ **by** *simp*
qed

lemma *rel-restrict-left-eq-rel-lambda-if-le-if-rel-dep-fun*:
assumes $((x : A) \rightarrow B\ x)\ R$
and $A' \leq A$
shows $R\restriction_{A'} = (\lambda x : A'.\ R`x)$
proof −
from *assms mono-rel-dep-fun-ge-crel-dep-fun-rel-restrict-left* **have** $((x : A') \rightarrow_c B\ x)\ R\restriction_{A'}$
by *force*
then show *?thesis*
supply $\langle A' \leq A \rangle$[*uhint*] *inf-absorb2*[*uhint*] **by** (*urule rel-lambda-ext*[*symmetric*]) *auto*
qed

lemma *mono-rel-lambda*: *mono* $(\lambda A.\ \lambda x : A.\ f\ x)$
by *auto*

end

theory *Binary-Relations-Function*
imports
Binary-Relations-Clean-Function
Binary-Relations-Function-Base
Binary-Relations-Function-Composition
Binary-Relations-Function-Evaluation
Binary-Relations-Function-Extend
Binary-Relations-Function-Lambda
begin

end

theory *Binary-Relations-Order*
imports
Binary-Relations-Order-Base
Binary-Relations-Reflexive
Binary-Relations-Symmetric
Binary-Relations-Transitive
begin

Summary Basic results about the order on binary relations.

lemma *in-dom-if-rel-if-rel-comp-le*:
assumes $(R \circ\circ S) \leq (S \circ\circ R)$
and $R\ x\ y\ S\ y\ z$
shows *in-dom* $S\ x$
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* $(R1 :: 'a \Rightarrow 'a \Rightarrow bool)$ *symmetric R2 symmetric R3 symmetric R4*
and $(R1 \circ\circ R2) \leq (R3 \circ\circ R4)$
shows $(R2 \circ\circ R1) \leq (R4 \circ\circ R3)$
proof −
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* $(R1 :: 'a \Rightarrow 'a \Rightarrow bool)$ *symmetric R2 symmetric R3 symmetric R4*
shows $(R1 \circ\circ R2) \leq (R3 \circ\circ R4) \longleftrightarrow (R2 \circ\circ R1) \leq (R4 \circ\circ R3)$
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

Surjective

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

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

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* $\equiv \forall\, y : P.$ *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*:
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* ⇒ *bool*) (*R*$^{-1}$ :: *'b* ⇒ *'a* ⇒ *bool*) ⟷ *left-total-on P R*
by *fast*

lemma *left-total-on-rel-inv-iff-rel-surjective-at* [*iff*]:
left-total-on (*P* :: *'a* ⇒ *bool*) (*R*$^{-1}$ :: *'a* ⇒ *'b* ⇒ *bool*) ⟷ *rel-surjective-at P R*
by *fast*

lemma *mono-rel-surjective-at*:
((≥) ⇛$_m$ (≤) ⇛ (≤)) (*rel-surjective-at* :: (*'b* ⇒ *bool*) ⇒ (*'a* ⇒ *'b* ⇒ *bool*) ⇒ *bool*)
by *fastforce*

lemma *rel-surjective-at-iff-le-codom*:
rel-surjective-at (*P* :: *'b* ⇒ *bool*) (*R* :: *'a* ⇒ *'b* ⇒ *bool*) ⟷ *P* ≤ *in-codom R*
by *force*

lemma *rel-surjective-at-compI*:
fixes *P* :: *'c* ⇒ *bool* **and** *R* :: *'a* ⇒ *'b* ⇒ *bool* **and** *S* :: *'b* ⇒ *'c* ⇒ *bool*
assumes *surj-R*: *rel-surjective-at* (*in-dom S*) *R*
and *surj-S*: *rel-surjective-at P S*
shows *rel-surjective-at P* (*R* ∘∘ *S*)
proof (*rule rel-surjective-atI*)
fix *y* **assume** *P y*
then obtain *x* **where** *S x y* **using** *surj-S* **by** *auto*
moreover then have *in-dom S x* **by** *auto*
moreover then obtain *z* **where** *R z x* **using** *surj-R* **by** *auto*
ultimately show *in-codom* (*R* ∘∘ *S*) *y* **by** *blast*
qed

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

overloading
rel-surjective ≡ *rel-surjective* :: (*'b* ⇒ *'a* ⇒ *bool*) ⇒ *bool*
begin
definition (*rel-surjective* :: (*'b* ⇒ *'a* ⇒ *bool*) ⇒ -) ≡ *rel-surjective-at* (⊤ :: *'a* ⇒ *bool*)
end

lemma *rel-surjective-eq-rel-surjective-at*:
(*rel-surjective* :: (*'b* ⇒ *'a* ⇒ *bool*) ⇒ -) = *rel-surjective-at* (⊤ :: *'a* ⇒ *bool*)

unfolding *rel-surjective-def* **..**

lemma *rel-surjective-eq-rel-surjective-at-uhint* [*uhint*]:
assumes $P \equiv (\top :: 'a \Rightarrow bool)$
shows (*rel-surjective* :: $('b \Rightarrow 'a \Rightarrow bool) \Rightarrow$ -) $\equiv$ *rel-surjective-at P*
using *assms* **by** (*simp add*: *rel-surjective-eq-rel-surjective-at*)

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

lemma *rel-surjectiveE*:
assumes *rel-surjective R*
obtains *x* **where** *R x y*
using *assms* **by** (*urule* (*e*) *rel-surjective-atE* **where** *chained* = *insert*) *simp*

lemma *in-codom-if-rel-surjective*:
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*
by (*urule rel-surjective-at-rel-inv-iff-left-total-on*)

lemma *left-total-rel-inv-iff-rel-surjective* [*iff*]: *left-total* $R^{-1} \longleftrightarrow$ *rel-surjective R*
by (*urule left-total-on-rel-inv-iff-rel-surjective-at*)

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

end

Bi-Total

theory *Binary-Relations-Bi-Total*
imports
Binary-Relations-Left-Total
Binary-Relations-Surjective
begin

definition *bi-total-on P Q* $\equiv$ *left-total-on P* $\sqcap$ *rel-surjective-at Q*

lemma *bi-total-onI* [*intro*]:
assumes *left-total-on P R*

and *rel-surjective-at Q R*
shows *bi-total-on P Q R*
unfolding *bi-total-on-def* **using** *assms* **by** *auto*

lemma *bi-total-onE* [*elim*]:
assumes *bi-total-on P Q R*
obtains *left-total-on P R rel-surjective-at Q R*
using *assms* **unfolding** *bi-total-on-def* **by** *auto*

definition *bi-total* ≡ *bi-total-on* (⊤ :: *'a* ⇒ *bool*) (⊤ :: *'b* ⇒ *bool*) :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ *bool*

lemma *bi-total-eq-bi-total-on*:
(*bi-total* :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ *bool*) = *bi-total-on* (⊤ :: *'a* ⇒ *bool*) (⊤ :: *'b* ⇒ *bool*)
unfolding *bi-total-def* **..**

lemma *bi-total-eq-bi-total-on-uhint* [*uhint*]:
assumes *P* ≡ (⊤ :: *'a* ⇒ *bool*)
and *Q* ≡ (⊤ :: *'b* ⇒ *bool*)
shows (*bi-total* :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ *bool*) ≡ *bi-total-on P Q*
using *assms* **by** (*simp add*: *bi-total-eq-bi-total-on*)

lemma *bi-totalI* [*intro*]:
assumes *left-total R*
and *rel-surjective R*
shows *bi-total R*
using *assms* **by** (*urule bi-total-onI*)

lemma *bi-totalE* [*elim*]:
assumes *bi-total R*
obtains *left-total R rel-surjective R*
using *assms* **by** (*urule* (*e*) *bi-total-onE*)

end

Bi-Unique

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

definition *bi-unique-on* ≡ *right-unique-on* ⊓ *rel-injective-on*

lemma *bi-unique-onI* [*intro*]:
assumes *right-unique-on P R*
and *rel-injective-on P R*

shows *bi-unique-on P R*
unfolding *bi-unique-on-def* **using** *assms* **by** *auto*

lemma *bi-unique-onE* [*elim*]:
assumes *bi-unique-on P R*
obtains *right-unique-on P R rel-injective-on P R*
using *assms* **unfolding** *bi-unique-on-def* **by** *auto*

lemma *bi-unique-on-rel-inv-if-Fun-Rel-iff-if-bi-unique-on*:
assumes *bi-unique-on P R*
and $(R \Rrightarrow (\longleftrightarrow))$ *P Q*
shows *bi-unique-on Q* R^{-1}
using *assms* **by** (*intro bi-unique-onI*
rel-injective-on-if-Fun-Rel-imp-if-rel-injective-at
right-unique-on-if-Fun-Rel-imp-if-right-unique-at)
(*auto 0 3*)

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

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

lemma *bi-unique-eq-bi-unique-on-uhint* [*uhint*]:
assumes $P \equiv (\top :: 'a \Rightarrow bool)$
shows *bi-unique* $\equiv$ (*bi-unique-on P* $:: ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool$)
using *assms* **by** (*simp add*: *bi-unique-eq-bi-unique-on*)

lemma *bi-uniqueI* [*intro*]:
assumes *right-unique R*
and *rel-injective R*
shows *bi-unique R*
using *assms* **by** (*urule bi-unique-onI*)

lemma *bi-uniqueE* [*elim*]:
assumes *bi-unique R*
obtains *right-unique R rel-injective R*
using *assms* **by** (*urule* (*e*) *bi-unique-onE*)

end

Connected

theory *Binary-Relations-Connected*
imports
Binary-Relation-Functions
begin

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

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

lemma *connected-onI* [*intro*]:
  assumes ⋀*x y*. *P x* ⟹ *P y* ⟹ *x* ≠ *y* ⟹ *R x y* ∨ *R y x*
  shows *connected-on P R*
  using *assms* **unfolding** *connected-on-pred-def* **by** *blast*

lemma *connected-onE* [*elim*]:
  assumes *connected-on P R*
  and *P x P y*
  obtains *x* = *y* | *R x y* | *R y x*
  using *assms* **unfolding** *connected-on-pred-def* **by** *auto*

lemma *connected-on-rel-inv-iff-connected-on* [*iff*]:
  connected-on (*P* :: '*a* ⇒ *bool*) (*R* :: '*a* ⇒ '*a* ⇒ *bool*)$^{-1}$ ⟷ *connected-on P R*
  by *blast*

consts *connected* :: '*a* ⇒ *bool*

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

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

lemma *connected-eq-connected-on-uhint* [*uhint*]:
  P ≡ (⊤ :: '*a* ⇒ *bool*) ⟹ (*connected* :: ('*a* ⇒ '*a* ⇒ *bool*) ⇒ -) ≡ *connected-on P*
  by (*simp add*: *connected-eq-connected-on*)

lemma *connectedI* [*intro*]:
  assumes ⋀*x y*. *x* ≠ *y* ⟹ *R x y* ∨ *R y x*
  shows *connected R*
  using *assms* **by** (*urule connected-onI*)

lemma *connectedE* [*elim*]:
  assumes *connected R*
  and *x* ≠ *y*
  obtains *R x y* | *R y x*
  using *assms* **by** (*urule* (*e*) *connected-onE* **where** *chained* = *insert*) *auto*

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

end

Irreflexive

theory *Binary-Relations-Irreflexive*
imports
HOL-Syntax-Bundles-Lattices
ML-Unification.ML-Unification-HOL-Setup
ML-Unification.Unify-Resolve-Tactics
begin

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

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*

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

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

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

lemma *irreflexive-eq-irreflexive-on-uhint* [*uhint*]:
assumes $P \equiv (\top :: 'a \Rightarrow bool)$
shows *irreflexive* $:: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow$ - $\equiv$ *irreflexive-on P*
using *assms* **by** (*simp add*: *irreflexive-eq-irreflexive-on*)

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

lemma *irreflexiveD*:
assumes *irreflexive R*
shows $\neg(R\ x\ x)$
using *assms* **by** (*urule* (*d*) *irreflexive-onD* **where** *chained* = *insert*) *simp*

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

end

1.2.19 Basic Properties

theory *Binary-Relation-Properties*
imports
Binary-Relations-Antisymmetric
Binary-Relations-Bi-Total
Binary-Relations-Bi-Unique
Binary-Relations-Connected
Binary-Relations-Injective
Binary-Relations-Irreflexive
Binary-Relations-Left-Total
Binary-Relations-Reflexive
Binary-Relations-Right-Unique
Binary-Relations-Surjective
Binary-Relations-Symmetric
Binary-Relations-Transitive
begin

end

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

lemma *in-codom-rel-restrict-left-eq* [*simp*]: *in-codom* $R\restriction_P$ = (*in-codom* $R \sqcap P$)
by (*intro ext*) *auto*

lemma *rel-restrict-left-restrict-left-eq* [*simp*]:
fixes $R :: 'a \Rightarrow 'b \Rightarrow bool$ **and** $P\ Q :: 'a \Rightarrow bool$
shows $R\restriction_P\restriction_Q = R\restriction_P \sqcap R\restriction_Q$
by (*intro ext iffI rel-restrict-leftI*) *auto*

lemma *rel-restrict-left-restrict-right-eq* [*simp*]:
fixes $R :: 'a \Rightarrow 'b \Rightarrow bool$ **and** $P :: 'a \Rightarrow bool$ **and** $Q :: 'b \Rightarrow bool$
shows $R\restriction_P\restriction_Q = R\restriction_P \sqcap R\restriction_Q$
by (*intro ext iffI rel-restrict-leftI rel-restrict-rightI*) *auto*

lemma *rel-restrict-right-restrict-left-eq* [*simp*]:
fixes $R :: 'a \Rightarrow 'b \Rightarrow bool$ **and** $P :: 'b \Rightarrow bool$ **and** $Q :: 'a \Rightarrow bool$
shows $R\restriction_P\restriction_Q = R\restriction_P \sqcap R\restriction_Q$
by (*intro ext iffI rel-restrict-leftI rel-restrict-rightI*) *auto*

lemma *rel-restrict-right-restrict-right-eq* [*simp*]:
fixes $R :: 'a \Rightarrow 'b \Rightarrow bool$ **and** $P\ Q :: 'b \Rightarrow bool$
shows $R\restriction_P\restriction_Q = R\restriction_P \sqcap R\restriction_Q$
by (*intro ext iffI*) *auto*

lemma *rel-restrict-left-sup-eq* [*simp*]:
$(R :: 'a \Rightarrow 'b \Rightarrow bool)\restriction_{((P :: 'a \Rightarrow bool) \sqcup Q)} = R\restriction_P \sqcup R\restriction_Q$
by (*intro antisym le-relI*) (*auto elim*!: *rel-restrict-leftE*)

lemma *rel-restrict-left-inf-eq* [*simp*]:
$(R :: 'a \Rightarrow 'b \Rightarrow bool)\restriction_{((P :: 'a \Rightarrow bool) \sqcap Q)} = R\restriction_P \sqcap R\restriction_Q$
by (*intro antisym le-relI*) (*auto elim*!: *rel-restrict-leftE*)

lemma *inf-rel-bimap-and-eq-restrict-left-restrict-right*:
$R \sqcap$ (*rel-bimap* $P\ Q\ (\wedge)$) = $R\restriction_P\restriction_Q$
by (*intro ext*) *auto*

end

theory *LBinary-Relations*
imports
Binary-Relations-Function
Binary-Relations-Order
Binary-Relation-Properties
Binary-Relation-Functions
Binary-Relations-Agree
Binary-Relations-Extend
Binary-Relations-Lattice
Dependent-Binary-Relations

Restricted-Equality
Reverse-Implies
begin

Summary Basic concepts on binary relations.

end

Injective

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

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

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

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

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

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

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

assumes $P \equiv (\top :: 'a \Rightarrow bool)$
shows $injective :: ('a \Rightarrow 'b) \Rightarrow bool \equiv injective\text{-}on\ P$
using *assms* **by** (*simp add*: *injective-eq-injective-on*)

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

lemma *injectiveD*:
assumes *injective f*
and $f\ x = f\ x'$
shows $x = x'$
using *assms* **by** (*urule* (*d*) *injective-onD* **where** *chained* = *insert*) *simp-all*

lemma *injective-on-if-injective*:
fixes $P :: 'a \Rightarrow bool$ **and** $f :: 'a \Rightarrow$ -
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
Binary-Relations-Function-Evaluation
begin

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

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

lemma *inverse-onI* [*intro*]:
assumes $\bigwedge x.\ P\ x \Longrightarrow g\ (f\ x) = x$
shows *inverse-on P f g*
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* $(P :: 'a \Rightarrow bool)\ (f :: 'a \Rightarrow 'b)\ (g :: 'b \Rightarrow 'a)$
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

lemma *inverse-on-compI*:
fixes $P :: 'a \Rightarrow bool$ **and** $P' :: 'b \Rightarrow bool$
and $f :: 'a \Rightarrow 'b$ **and** $g :: 'b \Rightarrow 'a$ **and** $f' :: 'b \Rightarrow 'c$ **and** $g' :: 'c \Rightarrow 'b$
assumes *inverse-on* $P\ f\ g$
and *inverse-on* $P'\ f'\ g'$
and $(P \Rightarrow_m P')\ f$
shows *inverse-on* $P\ (f' \circ f)\ (g \circ g')$
using *assms* **by** (*intro inverse-onI*) (*auto dest*!: *inverse-onD*)

consts *inverse* :: $'a \Rightarrow 'b \Rightarrow bool$

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

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

lemma *inverse-eq-inverse-on-uhint* [*uhint*]:
assumes $P \equiv \top :: 'a \Rightarrow bool$
shows $inverse :: ('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow 'a) \Rightarrow bool \equiv$ *inverse-on* P
using *assms* **by** (*simp add*: *inverse-eq-inverse-on*)

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

lemma *inverseD*:

assumes *inverse f g*
shows *g* (*f x*) = *x*
using *assms* **by** (*urule* (*d*) *inverse-onD* **where** *chained* = *insert*) *simp-all*

lemma *inverse-on-if-inverse*:
fixes *P* :: *'a* ⇒ *bool* **and** *f* :: *'a* ⇒ *'b* **and** *g* :: *'b* ⇒ *'a*
assumes *inverse f g*
shows *inverse-on P f g*
using *assms* **by** (*intro inverse-onI*) (*blast dest*: *inverseD*)

lemma *right-unique-eq-app-if-injective*:
assumes *injective f*
shows *right-unique* (λ*y x*. *y* = *f x*)
using *assms* **by** (*auto dest*: *injectiveD*)

lemma *inverse-eval-eq-app-if-injective*:
assumes *injective f*
shows *inverse f* (*eval* (λ*y x*. *y* = *f x*))
by (*urule inverseI eval-eq-if-right-unique-onI right-unique-eq-app-if-injective*)+
(*use assms* **in** *simp-all*)

lemma *inverse-if-injectiveE*:
assumes *injective* (*f* :: *'a* ⇒ *'b*)
obtains *g* :: *'b* ⇒ *'a* **where** *inverse f g*
using *assms inverse-eval-eq-app-if-injective* **by** *blast*

end

Bijections

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

consts *bijection-on* :: *'a* ⇒ *'b* ⇒ *'c* ⇒ *'d* ⇒ *bool*

definition *bijection-on-pred* (*P* :: *'a* ⇒ *bool*) (*Q* :: *'b* ⇒ *bool*) *f g* ≡
(*P* ⇛$_m$ *Q*) *f* ∧
(*Q* ⇛$_m$ *P*) *g* ∧
inverse-on P f g ∧
inverse-on Q g f
adhoc-overloading *bijection-on bijection-on-pred*

context
fixes *P* :: *'a* ⇒ *bool* **and** *Q* :: *'b* ⇒ *bool* **and** *f* :: *'a* ⇒ *'b* **and** *g* :: *'b* ⇒ *'a*
begin

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

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

lemma *mono-wrt-pred-if-bijection-on-left*:
assumes *bijection-on P Q f g*
shows $(P \Rightarrow_m Q)\ f$
using *assms* **by** (*elim bijection-onE*)

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

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

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

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

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

end

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

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

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

end

lemma *bijection-on-compI*:
fixes $P :: 'a \Rightarrow bool$ **and** $P' :: 'b \Rightarrow bool$ **and** $Q :: 'c \Rightarrow bool$
assumes *bijection-on P P' f g*
and *bijection-on P' Q f' g'*
shows *bijection-on P Q* $(f' \circ f)\ (g \circ g')$
using *assms* **by** (*intro bijection-onI*)
(*auto intro*: *dep-mono-wrt-pred-comp-dep-mono-wrt-pred-compI' inverse-on-compI*
elim!: *bijection-onE simp*: *mono-wrt-pred-eq-dep-mono-wrt-pred*)

consts *bijection* $:: 'a \Rightarrow 'b \Rightarrow bool$

definition (*bijection-rel* $:: ('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow 'a) \Rightarrow bool) \equiv$

bijection-on (⊤ :: *'a* ⇒ *bool*) (⊤ :: *'b* ⇒ *bool*)
adhoc-overloading *bijection bijection-rel*

lemma *bijection-eq-bijection-on*:
(*bijection* :: (*'a* ⇒ *'b*) ⇒ (*'b* ⇒ *'a*) ⇒ *bool*) = *bijection-on* (⊤ :: *'a* ⇒ *bool*) (⊤ :: *'b* ⇒ *bool*)
unfolding *bijection-rel-def* **..**

lemma *bijection-eq-bijection-on-uhint* [*uhint*]:
assumes *P* ≡ (⊤ :: *'a* ⇒ *bool*)
and *Q* ≡ (⊤ :: *'b* ⇒ *bool*)
shows (*bijection* :: (*'a* ⇒ *'b*) ⇒ (*'b* ⇒ *'a*) ⇒ *bool*) = *bijection-on P Q*
using *assms* **by** (*simp add*: *bijection-eq-bijection-on*)

context
fixes *P* :: *'a* ⇒ *bool* **and** *Q* :: *'b* ⇒ *bool* **and** *f* :: *'a* ⇒ *'b* **and** *g* :: *'b* ⇒ *'a*
begin

lemma *bijectionI* [*intro*]:
assumes *inverse f g*
and *inverse g f*
shows *bijection f g*
by (*urule bijection-onI*) (*simp* | *urule assms*)+

lemma *bijectionE* [*elim*]:
assumes *bijection f g*
obtains *inverse f g inverse g f*
using *assms* **by** (*urule* (*e*) *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*)

end

lemma *bijection-right-left-if-bijection-left-right*:
fixes *f* :: *'a* ⇒ *'b* **and** *g* :: *'b* ⇒ *'a*
assumes *bijection f g*
shows *bijection g f*
using *assms* **by** *auto*

Instantiations **lemma** *bijection-on-self-id*: *bijection-on* (*P* :: *'a* ⇒ *bool*) *P id id*

by (*intro bijection-onI inverse-onI mono-wrt-predI*) *simp-all*

end

Surjective

theory *Functions-Surjective*
imports
Functions-Base
begin

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

overloading
surjective-at-pred $\equiv$ *surjective-at* :: $('a \Rightarrow bool) \Rightarrow ('b \Rightarrow 'a) \Rightarrow bool$
begin
definition *surjective-at-pred* $(P :: 'a \Rightarrow bool)$ $(f :: 'b \Rightarrow 'a) \equiv \forall y : P.$ *has-inverse f y*
end

lemma *surjective-atI* [*intro*]:
assumes $\bigwedge y.$ $P\ y \Longrightarrow$ *has-inverse f y*
shows *surjective-at P f*
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*

lemma *has-inverse-if-pred-if-surjective-at*:
assumes *surjective-at P f*
and $P\ y$
shows *has-inverse f y*
using *assms* **by** *blast*

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

overloading
surjective $\equiv$ *surjective* :: $('b \Rightarrow 'a) \Rightarrow bool$
begin
definition (*surjective* :: $('b \Rightarrow 'a) \Rightarrow bool$) $\equiv$ *surjective-at* $(\top :: 'a \Rightarrow bool)$
end

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

lemma *surjective-eq-surjective-at-uhint* [*uhint*]:
assumes $P \equiv \top :: 'a \Rightarrow bool$
shows *surjective* :: $('b \Rightarrow 'a) \Rightarrow bool \equiv$ *surjective-at* P
using *assms* **by** (*simp add*: *surjective-eq-surjective-at*)

lemma *surjectiveI* [*intro*]:
assumes $\bigwedge y.$ *has-inverse* $f\ y$
shows *surjective* f
using *assms* **by** (*urule surjective-atI*)

lemma *surjectiveE*:
assumes *surjective* f
obtains $\bigwedge y.$ *has-inverse* $f\ y$
using *assms* **unfolding** *surjective-eq-surjective-at* **by** (*force dest*: *has-inverseI*)

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

end

1.2.21 Basic Properties

theory *Function-Properties*
imports
Functions-Bijection
Functions-Injective
Functions-Inverse
Functions-Monotone
Functions-Surjective
begin

Summary Basic properties on functions.

end

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

consts *fun-restrict* :: $'a \Rightarrow 'b \Rightarrow 'a$

bundle *fun-restrict-syntax*
begin
notation *fun-restrict* ((-)⌈(-) [*1000*])
end
bundle *no-fun-restrict-syntax*

begin
no-notation *fun-restrict* ((-)↾(_) [*1000*])
end

definition *fun-restrict-pred f P x* ≡ *if P x then f x else undefined*
adhoc-overloading *fun-restrict fun-restrict-pred*

context
  includes *fun-restrict-syntax*
begin

lemma *fun-restrict-eq* [*simp*]:
  assumes *P x*
  shows $f{\restriction}_P\ x = f\ x$
  using *assms* **unfolding** *fun-restrict-pred-def* **by** *auto*

lemma *fun-restrict-eq-if-not* [*simp*]:
  assumes ¬(*P x*)
  shows $f{\restriction}_P\ x = undefined$
  using *assms* **unfolding** *fun-restrict-pred-def* **by** *auto*

lemma *fun-restrict-eq-if*: $f{\restriction}_P\ x = (if\ P\ x\ then\ f\ x\ else\ undefined)$
  by *auto*

lemma *fun-restrict-cong* [*cong*]:
  assumes $\bigwedge x.\ P\ x \longleftrightarrow P'\ x$
  and $\bigwedge x.\ P'\ x \Longrightarrow f\ x = g\ x$
  shows $f{\restriction}_P = g{\restriction}_{P'}$
  using *assms* **by** (*intro ext*) (*auto simp*: *fun-restrict-eq-if*)

end

end

theory *LFunctions*
  imports
    Function-Properties
    Function-Relators
    Functions-Restrict
begin

Summary Basic concepts on functions.

end

1.2.22 Functions On Orders

Basics

theory *Order-Functions-Base*
imports
Functions-Monotone
Binary-Relations-Antisymmetric
Binary-Relations-Symmetric
Preorders
begin

Bi-Relation **consts** $bi\text{-}related :: 'a \Rightarrow 'b \Rightarrow 'b \Rightarrow bool$

overloading
$bi\text{-}related \equiv bi\text{-}related :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$
begin
definition $bi\text{-}related\ (R :: 'a \Rightarrow 'a \Rightarrow bool)\ x\ y \equiv R\ x\ y \wedge R\ y\ x$
end

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

lemma *bi-relatedI* [*intro*]:
assumes $R\ x\ y$
and $R\ y\ x$
shows $x \equiv_R y$
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*

context
fixes $P :: 'a \Rightarrow bool$ **and** $R\ S :: 'a \Rightarrow 'a \Rightarrow bool$ **and** $x\ y :: 'a$
begin

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

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

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

lemma *mono-bi-related*: *mono* $(bi\text{-}related :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a \Rightarrow 'a \Rightarrow bool)$
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* $\leq$ *P*
and $x \equiv_R y$
shows $x = y$
using *assms* **by** (*intro eq-if-bi-related-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'*)

end

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

Inflationary

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

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

Often also called "extensive".

definition *inflationary-on-pred P* ($R :: 'a \Rightarrow 'b \Rightarrow bool$) ($f :: 'a \Rightarrow 'b$) $\equiv \forall x : P.\ R\ x\ (f\ x)$
end

lemma *inflationary-onI* [*intro*]:
assumes $\bigwedge x.\ P\ x \Longrightarrow R\ x\ (f\ x)$
shows *inflationary-on P R f*
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 *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 *mono-inflationary-on-rel*:
$((\geq) \Rightarrow_m (\leq) \Rightarrow (\leq))$ (*inflationary-on* :: $('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool$)
by (*intro mono-wrt-relI Fun-Rel-relI*) *auto*

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

lemma *inflationary-on-if-le-pred-if-inflationary-on*:
assumes *inflationary-on P R f*
and $P' \leq P$
shows *inflationary-on P' R f*
using *assms* **by** *blast*

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

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

consts *inflationary* :: $'a \Rightarrow 'b \Rightarrow bool$

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

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

unfolding *inflationary-def* **..**

lemma *inflationary-eq-inflationary-on-uhint* [*uhint*]:
assumes $P \equiv \top :: 'a \Rightarrow bool$
shows *inflationary* $:: ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool \equiv$ *inflationary-on P*
using *assms* **by** (*simp add*: *inflationary-eq-inflationary-on*)

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

lemma *inflationaryD*:
assumes *inflationary R f*
shows $R\ x\ (f\ x)$
using *assms* **by** (*urule* (*d*) *inflationary-onD* **where** *chained* = *insert*) *simp*

lemma *inflationary-on-if-inflationary*:
fixes $P :: 'a \Rightarrow bool$ **and** $R :: 'a \Rightarrow 'b \Rightarrow bool$ **and** $f :: 'a \Rightarrow 'b$
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*: *inflationary* = *dep-mono-wrt-pred* $\top$
by (*intro ext*) (*fastforce dest*: *inflationaryD*)

Deflationary

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

overloading
deflationary-on-pred $\equiv$ *deflationary-on* $:: ('a \Rightarrow bool) \Rightarrow ('b \Rightarrow 'a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool$
begin
definition *deflationary-on-pred* $(P :: 'a \Rightarrow bool)\ (R :: 'b \Rightarrow 'a \Rightarrow bool) :: ('a \Rightarrow 'b) \Rightarrow bool \equiv$
inflationary-on P R^{-1}
end

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

lemma *deflationary-on-eq-inflationary-on-rel-inv*:
(*deflationary-on P R* $:: ('a \Rightarrow 'b) \Rightarrow bool$) = *inflationary-on P* R^{-1}
unfolding *deflationary-on-pred-def* **..**

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

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

shows *deflationary-on P R f*
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*

end

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

corollary *deflationary-on-rel-inv-eq-inflationary-on* [*simp*]:
(*deflationary-on P* (*S* :: *'a* ⇒ *'b* ⇒ *bool*)$^{-1}$ :: (*'a* ⇒ *'b*) ⇒ *bool*) = *inflationary-on P S*
unfolding *deflationary-on-eq-inflationary-on-rel-inv* **by** *simp*

lemma *deflationary-on-eq-dep-mono-wrt-pred-rel-inv*:
(*deflationary-on P R* :: (*'a* ⇒ *'b*) ⇒ *bool*) = ((*x* : *P*) ⇛$_m$ *R*$^{-1}$ *x*)
by *blast*

lemma *deflationary-on-if-le-rel-if-deflationary-on*:
assumes *deflationary-on P R f*
and ⋀*x*. *P x* ⟹ *R* (*f x*) *x* ⟹ *R'* (*f x*) *x*
shows *deflationary-on P R' f*
using *assms* **by** *auto*

lemma *mono-deflationary-on*:
((≥) ⇛$_m$ (≤) ⇛ (≤)) (*deflationary-on* :: (*'a* ⇒ *bool*) ⇒ (*'b* ⇒ *'a* ⇒ *bool*) ⇒ (*'a* ⇒ *'b*) ⇒ *bool*)
by *blast*

lemma *deflationary-on-if-le-pred-if-deflationary-on*:
assumes *deflationary-on P R f*
and *P'* ≤ *P*
shows *deflationary-on P' R f*
using *assms* **by** *blast*

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*) ⇒ ('*b* ⇒ '*a* ⇒ *bool*) ⇒ ('*a* ⇒ '*b*) ⇒ *bool*) (*P* ⊔ *Q*)
= *deflationary-on P* ⊓ *deflationary-on Q*
unfolding *deflationary-on-eq-inflationary-on-rel-inv* **by** *auto*

end

consts *deflationary* :: '*a* ⇒ '*b* ⇒ *bool*

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

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

lemma *deflationary-eq-deflationary-on-uhint* [*uhint*]:
assumes *P* ≡ ⊤ :: '*a* ⇒ *bool*
shows *deflationary* :: ('*b* ⇒ '*a* ⇒ *bool*) ⇒ ('*a* ⇒ '*b*) ⇒ *bool* ≡ *deflationary-on P*
using *assms* **by** (*simp add*: *deflationary-eq-deflationary-on*)

lemma *deflationaryI* [*intro*]:
assumes ⋀*x*. *R* (*f x*) *x*
shows *deflationary R f*
using *assms* **by** (*urule deflationary-onI*)

lemma *deflationaryD*:
assumes *deflationary R f*
shows *R* (*f x*) *x*
using *assms* **by** (*urule* (*d*) *deflationary-onD* **where** *chained* = *insert*) *simp*

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

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

Relational Equivalence **definition** *rel-equivalence-on* ≡ *inflationary-on* ⊓ *deflationary-on*

lemma *rel-equivalence-on-eq*:
rel-equivalence-on = *inflationary-on* ⊓ *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*

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

lemma *bi-related-if-rel-equivalence-on*:
assumes *rel-equivalence-on P R f*
and *P x*
shows $x \equiv_R f\ x$
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 $x \equiv_R f\ x$
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} :: (*'a* ⇒ *'a*) ⇒ *bool*) = *rel-equivalence-on P R*
by (*intro ext*) *fastforce*

lemma *mono-rel-equivalence-on*:
(($\geq$) $\Rrightarrow_m$ ($\leq$) $\Rrightarrow$ ($\leq$)) (*rel-equivalence-on* :: ('*a* $\Rightarrow$ *bool*) $\Rightarrow$ ('*a* $\Rightarrow$ '*a* $\Rightarrow$ *bool*) $\Rightarrow$ ('*a* $\Rightarrow$ '*a*) $\Rightarrow$ *bool*)
by *blast*

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

lemma *rel-equivalence-on-sup-eq* [*simp*]:
(*rel-equivalence-on* :: ('*a* $\Rightarrow$ *bool*) $\Rightarrow$ ('*a* $\Rightarrow$ '*a* $\Rightarrow$ *bool*) $\Rightarrow$ ('*a* $\Rightarrow$ '*a*) $\Rightarrow$ *bool*) (*P* $\sqcup$ *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* :: ('*a* $\Rightarrow$ '*a*) $\Rightarrow$ *bool*) = *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* :: ('*a* $\Rightarrow$ '*a*) $\Rightarrow$ *bool*) = *rel-equivalence-on P R*
using *assms*
by (*simp add*: *deflationary-on-eq-inflationary-on-rel-inv rel-equivalence-on-eq*)

end

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

overloading
rel-equivalence $\equiv$ *rel-equivalence* :: ('*a* $\Rightarrow$ '*a* $\Rightarrow$ *bool*) $\Rightarrow$ ('*a* $\Rightarrow$ '*a*) $\Rightarrow$ *bool*
begin

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

lemma *rel-equivalence-eq-rel-equivalence-on*:
(*rel-equivalence* :: ('*a* ⇒ '*a* ⇒ *bool*) ⇒ ('*a* ⇒ '*a*) ⇒ *bool*) = *rel-equivalence-on* (⊤ :: '*a* ⇒ *bool*)
unfolding *rel-equivalence-def* **..**

lemma *rel-equivalence-eq-rel-equivalence-on-uhint* [*uhint*]:
assumes *P* ≡ ⊤ :: '*a* ⇒ *bool*
shows *rel-equivalence* :: ('*a* ⇒ '*a* ⇒ *bool*) ⇒ ('*a* ⇒ '*a*) ⇒ *bool* ≡ *rel-equivalence-on P*
using *assms* **by** (*simp add*: *rel-equivalence-eq-rel-equivalence-on*)

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

lemma *rel-equivalenceI* [*intro*]:
assumes *inflationary R f*
and *deflationary R f*
shows *rel-equivalence R f*
using *assms* **by** (*urule rel-equivalence-onI*)

lemma *rel-equivalenceE* [*elim*]:
assumes *rel-equivalence R f*
obtains *inflationary R f deflationary R f*
using *assms* **by** (*urule* (*e*) *rel-equivalence-onE*)

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

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

lemma *rel-equivalence-on-if-rel-equivalence*:
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* ≡$_R$ *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*
by *blast*

end

end

1.2.23 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* $(\leq_L) \Rightarrow_m P$) l
and $(\leq_L) \leq (=)$
and *reflexive-on* P $(\leq_R)$
shows $((\leq_L) \Rightarrow_m (\leq_R))$ l
using *assms* **by** (*intro 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* = η
by (*intro ext*) *simp*

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

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

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

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

lemma *rel-equivalence-on-unit-iff-inflationary-on-if-inverse-on*:
assumes *inverse-on P l r*
shows *rel-equivalence-on P* $(\leq_L)$ $\eta \longleftrightarrow$ *inflationary-on P* $(\leq_L)$ η
using *assms* **by** (*intro iffI rel-equivalence-onI inflationary-onI deflationary-onI*)
(*fastforce dest*: *inverse-onD*)+

lemma *reflexive-on-left-if-inflationary-on-unit-if-inverse-on*:
assumes *inverse-on P l r*
and *inflationary-on P* $(\leq_L)$ η
shows *reflexive-on P* $(\leq_L)$
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* $(\leq_L)$
shows *rel-equivalence-on P* $(\leq_L)$ η
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* $(\leq_L)$ η $\longleftrightarrow$ *reflexive-on P* $(\leq_L)$
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 $[\![((\leq_R) \Rrightarrow_m (\leq_L))\ r;$ *reflexive-on P* $(\leq_R);$ $\bigwedge x\ y.\ [\![P\ x;\ r\ x \leq_L y]\!] \Longrightarrow x \leq_R l\ y]\!]$
$\Longrightarrow$ *inflationary-on P* $(\leq_R)$ ε **for** *P*
by (*fact flip.inflationary-on-unit-if-left-rel-right-if-left-right-relI*)
end

end

end

1.3 Galois

1.3.1 Basic Abbreviations

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

locale *galois* = *order-functors*
begin

The locale *galois* serves to define concepts that ultimately lead to the definition of Galois connections and Galois equivalences. Galois connections and equivalences are special cases of adjoints and adjoint equivalences, respectively, known from category theory. As such, in what follows, we sometimes borrow vocabulary from category theory to highlight this connection.

A *Galois connection* between two relations $(\leq_L)$ and $(\leq_R)$ consists of two monotone functions (i.e. order functors) l and r such that $(x \leq_L r\ y) = (l\ x \leq_R y)$. We call this the *Galois property*. l is called the *left adjoint* and r the *right adjoint*. We call $(\leq_L)$ the *left relation* and $(\leq_R)$ the *right relation*. By composing the adjoints, we obtain the unit η and counit ε of the Galois connection.

end

end

1.3.2 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* ≡ *in-codom* $(\leq_R)$ $y \wedge x \leq_L r\ y$

abbreviation *left-Galois* ≡ *Galois*
notation *left-Galois* (**infix** $_L\lessapprox$ *50*)

abbreviation (*input*) *ge-Galois-left* ≡ $(_L\lessapprox)^{-1}$
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* $(\leq_R)$ y
and $x \leq_L r\ y$
shows $x\ _L\lessapprox y$
unfolding *Galois-def* **using** *assms* **by** *blast*

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

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

corollary *left-Galois-iff-in-codom-and-left-rel-right*:
$x\ _L\lessapprox y \longleftrightarrow$ *in-codom* $(\leq_R)$ $y \wedge x \leq_L r\ y$
by *blast*

lemma *left-Galois-restrict-left-eq-left-Galois-left-restrict-left*:
$({}_L\lessapprox)\restriction_P :: {}'a \Rightarrow {}_{bool} = \mathit{galois\text{-}rel.Galois}\ (\leq_L)\restriction_P\ (\leq_R)\ r$
by (*intro ext iffI galois-rel.left-GaloisI rel-restrict-leftI*)
(*auto elim*: *galois-rel.left-GaloisE*)

lemma *left-Galois-restrict-right-eq-left-Galois-right-restrict-right*:
$({}_L\lessapprox)\restriction_P :: {}'d \Rightarrow {}_{bool} = \mathit{galois\text{-}rel.Galois}\ (\leq_L)\ (\leq_R)\restriction_P\ r$
by (*intro ext iffI galois-rel.left-GaloisI rel-restrict-rightI*)
(*auto elim*!: *galois-rel.left-GaloisE rel-restrict-rightE*)

end

end

Equivalences

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

context *order-functors*
begin

definition *order-equivalence* $\equiv$
$((\leq_L) \Rrightarrow_m (\leq_R))\ l \wedge$
$((\leq_R) \Rrightarrow_m (\leq_L))\ r \wedge$
rel-equivalence-on (*in-field* $(\leq_L)$) $(\leq_L)\ \eta \wedge$
rel-equivalence-on (*in-field* $(\leq_R)$) $(\leq_R)\ \varepsilon$

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

lemma *order-equivalenceI* [*intro*]:
assumes $((\leq_L) \Rrightarrow_m (\leq_R))\ l$
and $((\leq_R) \Rrightarrow_m (\leq_L))\ r$
and *rel-equivalence-on* (*in-field* $(\leq_L)$) $(\leq_L)\ \eta$
and *rel-equivalence-on* (*in-field* $(\leq_R)$) $(\leq_R)\ \varepsilon$
shows $((\leq_L) \equiv_o (\leq_R))\ l\ r$
unfolding *order-equivalence-def* **using** *assms* **by** *blast*

lemma *order-equivalenceE* [*elim*]:
assumes $((\leq_L) \equiv_o (\leq_R))\ l\ r$
obtains $((\leq_L) \Rrightarrow_m (\leq_R))\ l\ ((\leq_R) \Rrightarrow_m (\leq_L))\ r$
rel-equivalence-on (*in-field* $(\leq_L)$) $(\leq_L)\ \eta$
rel-equivalence-on (*in-field* $(\leq_R)$) $(\leq_R)\ \varepsilon$
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 0 4 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* = λ*x x'*. *?cx* = *x* $\wedge$ *x* = *x'*
and *?R* = λ*y y'*. (*y* = *cy* $\vee$ *y* = *cy'*) $\wedge$ (*y'* = *cy* $\vee$ *y'* = *cy'*) $\wedge$ (*y* $\neq$ *cy'* $\vee$ *y'* $\neq$ *cy'*)
and *?l* = λ(*a* :: *'a*). *cy*
and *?r* = λ(*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
$\neg$(*transitive-on* (*in-field ?R*) *?R*) **and** $\neg$(*reflexive-on* (*in-field ?R*) *?R*)
using ‹*cy* $\neq$ *cy'*› **by** *auto*
ultimately show *?thesis* **by** *blast*
qed

end

end

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

abbreviation (*input*) *Galois-right* $\equiv$ *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* ($\leq_L$) *x l x* $\leq_R$ *y*
using *assms* **by** *blast*

corollary *Galois-right-iff-in-dom-and-left-right-rel*:
$x \lessapprox_R y \longleftrightarrow$ *in-dom* $(\leq_L)\ x \wedge l\ x \leq_R y$
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* $\equiv \forall x\ y.\ x\ {}_L\lessapprox y \longrightarrow l\ x \leq_R y$

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

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

lemma *half-galois-prop-leftD* [*dest*]:
assumes $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$
and $x\ {}_L\lessapprox y$
shows $l\ x \leq_R y$
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* $(\geq_R)\ (\geq_L)\ r\ l$ **.**

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 $\bigwedge x\ y.\ x \lessapprox_R y \Longrightarrow x \leq_L r\ y$
shows $((\leq_L)\ \trianglelefteq_h\ (\leq_R))\ l\ r$
unfolding *half-galois-prop-right-def* **using** *assms* **by** *blast*

lemma *half-galois-prop-rightD* [*dest*]:
assumes $((\leq_L)\ \trianglelefteq_h\ (\leq_R))\ l\ r$
and $x \lessapprox_R y$
shows $x \leq_L r\ y$
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*]:
$((\leq_R) \trianglelefteq_h (\leq_L))^{-1} = ((\geq_L) \ {}_h\trianglerighteq (\geq_R))$
by (*intro ext*) *blast*

corollary *half-galois-prop-left-rel-inv-iff-half-galois-prop-right* [*iff*]:
$((\geq_L) \ {}_h\trianglerighteq (\geq_R))\ f\ g \longleftrightarrow ((\leq_R) \trianglelefteq_h (\leq_L))\ g\ f$
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* ≡ *flip.Galois*
notation *right-Galois* (**infix** $_R\lessapprox$ *50*)

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

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

abbreviation (*input*) *Galois-left* ≡ *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 $((\leq_L) \Rightarrow_m (\leq_R))\ l$
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** *blast*

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** (*fastforce 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 $\bigwedge(P :: 'c \Rightarrow bool)\ (R :: 'c \Rightarrow 'c \Rightarrow bool).$
$(inflationary\text{-}on\ P\ R^{-1} :: ('c \Rightarrow 'c) \Rightarrow bool) \equiv deflationary\text{-}on\ P\ R$
and $\bigwedge(P :: 'c \Rightarrow bool)\ (R :: 'c \Rightarrow 'c \Rightarrow bool).$
$(deflationary\text{-}on\ P\ R^{-1} :: ('c \Rightarrow 'c) \Rightarrow bool) \equiv inflationary\text{-}on\ P\ R$
and $\bigwedge(P :: 'b \Rightarrow bool).\ reflexive\text{-}on\ P\ (\geq_R) \equiv reflexive\text{-}on\ P\ (\leq_R)$
and $\bigwedge(R :: 'a \Rightarrow 'a \Rightarrow bool).\ transitive\ R^{-1} \equiv transitive\ R$
and $\bigwedge R.\ in\text{-}codom\ R^{-1} \equiv in\text{-}dom\ R$
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
mono-wrt-rel-eq-dep-mono-wrt-rel)

corollary *counit-rel-if-right-rel-if-mono-wrt-relI*:
assumes $((\leq_R) \Rightarrow_m (\leq_L))\ r$
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 $((\leq_R) \Rightarrow_m (\leq_L))\ r$
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 $((\leq_R) \Rightarrow_m (\leq_L))\ r$
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 $((\leq_R) \Rightarrow_m (\leq_L))\ r$
and $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$
and *reflexive-on* $P\ (\leq_R)$
shows *deflationary-on* $P\ (\leq_R)\ \varepsilon$

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

1.3.4 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 elim*: *galois-rel.left-GaloisE*)

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 *in-dom* $(\leq_L)\ x$ *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* ($\leq_L$)) ($\leq_L$) η
and *deflationary-on* (*in-codom* ($\leq_R$)) ($\leq_R$) ε
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

1.3.5 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

1.3.6 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 $(\leq_L)$ η
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

1.3.7 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 $((\leq_L) \equiv_G (\leq_R))$ *l r*
and *reflexive-on* (*in-field* $(\leq_L)$) $(\leq_L)$ *reflexive-on* (*in-field* $(\leq_R)$) $(\leq_R)$

shows $((\leq_L) \equiv_o (\leq_R))\ l\ r$
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* $(\leq_L)$) $(\leq_L)$ *preorder-on* (*in-field* $(\leq_R)$) $(\leq_R)$
shows $((\leq_L) \equiv_G (\leq_R)) = ((\leq_L) \equiv_o (\leq_R))$
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

1.3.8 Relator For Galois Connections

theory *Galois-Relator*
imports
Galois-Relator-Base
Galois-Property
begin

context *galois-prop*
begin

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

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

lemma *Galois-right-if-left-Galois-if-half-galois-prop-left*:
assumes $((\leq_L)\ {}_h\trianglerighteq (\leq_R))\ l\ r$
and $x\ {}_L\lessapprox y$
shows $x \lessapprox_R y$
using *assms* **by** *fast*

corollary *Galois-right-iff-left-Galois-if-galois-prop* [*iff*]:
assumes $((\leq_L) \trianglelefteq (\leq_R))\ l\ r$
shows $x \lessapprox_R y \longleftrightarrow x\ {}_L\lessapprox y$
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 $(\gtrsim_L) = ({}_R\gtrsim)$
using *assms* **by** *blast*

corollary *flip-Galois-rel-inv-iff-Galois-if-galois-prop* [*iff*]:
assumes $((\leq_L) \trianglelefteq (\leq_R))$ *l r*
shows $y \ {}_R\gtrsim \ x \longleftrightarrow x \ {}_L\lesssim \ y$
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 (*galois-rel.Galois* $(\leq_R)^{-1}$ $(\leq_L)^{-1}$ *l*)$^{-1}$ = (*galois-rel.Galois* $(\leq_R)^{-1}$ $(\leq_L)^{-1}$ *l*)$^{-1}$
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 $x \ {}_L\lesssim \ l \ x'$
using *assms*
by (*intro left-Galois-if-Galois-right-if-half-galois-prop-right*) (*auto 5 0*)

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 $x \ {}_L\lesssim \ l \ x$
using *assms* **by** (*intro left-Galois-left-if-left-relI*) *auto*

lemma *left-Galois-left-if-in-codom-if-inflationary-onI*:
assumes $((\leq_L) \Rrightarrow_m (\leq_R))$ *l*

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

lemma *left-Galois-left-if-in-codom-if-inflationary-on-in-codomI*:
assumes $((\leq_L) \Rightarrow_m (\leq_R))$ l
and *inflationary-on* (*in-codom* $(\leq_L)$) $(\leq_L)$ η
and *in-codom* $(\leq_L)$ x
shows x $_L\lessapprox$ l x
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 $((\leq_L) \Rightarrow_m (\leq_R))$ l
and *inflationary-on* (*in-field* $(\leq_L)$) $(\leq_L)$ η
and $x \leq_L x$
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 $((\leq_R) \Rightarrow_m (\leq_L))$ r
and $y \leq_R y'$
shows r y $_L\lessapprox$ y'
using *assms* **by** (*intro left-GaloisI*) *auto*

corollary *right-left-Galois-if-reflexive-onI*:
assumes $((\leq_R) \Rightarrow_m (\leq_L))$ r
and *reflexive-on* P $(\leq_R)$
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 $((\leq_R) \Rightarrow_m (\leq_L))$ r
and *transitive* $(\leq_L)$
and x $_L\lessapprox$ y
and $y \leq_R z$
shows x $_L\lessapprox$ z
using *assms* **by** (*intro left-GaloisI*) (*auto 5 0*)

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

lemma *left-rel-if-right-Galois-if-left-GaloisI*:

assumes $((\leq_R)\ {}_h\trianglelefteq\ (\leq_L))\ r\ l$
and *transitive* $(\leq_L)$
and $x\ {}_L\lessapprox\ y$
and $y\ {}_R\lessapprox\ z$
shows $x \leq_L z$
using *assms* **by** *auto*

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

lemma *left-ge-Galois-eq-left-Galois-if-in-codom-eq-in-dom-if-symmetric*:
assumes *symmetric* $(\leq_L)$
and *in-codom* $(\leq_R)$ = *in-dom* $(\leq_R)$
shows $({}_L\gtrsim) = ({}_L\lessapprox)$ — Note that *galois-rel.Galois* $(\leq_L)^{-1}\ (\leq_R)^{-1}\ r$ = *galois-rel.Galois* $(\leq_L)^{-1}\ (\leq_R)^{-1}\ r$
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 $((\leq_L) \trianglelefteq (\leq_R))\ l\ r$
and *in-codom* $(\leq_L)$ = *in-dom* $(\leq_L)$
and *symmetric* $(\leq_R)$
shows $(\gtrsim_R) = ({}_L\lessapprox)$ — 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 $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$
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 $((\leq_L) \trianglelefteq_h (\leq_R))\ l\ r$
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 $((\leq_L) \trianglelefteq (\leq_R))\ l\ r$
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

1.3.9 Linear Orders

theory *Linear-Orders*
imports
Binary-Relations-Connected
Partial-Orders
begin

definition *linear-order-on* ≡ *partial-order-on* ⊓ *connected-on*

lemma *linear-order-onI* [*intro*]:
assumes *partial-order-on P R*
and *connected-on P R*
shows *linear-order-on P R*
using *assms* **unfolding** *linear-order-on-def* **by** *auto*

lemma *linear-order-onE* [*elim*]:
assumes *linear-order-on P R*
obtains *partial-order-on P R connected-on P R*
using *assms* **unfolding** *linear-order-on-def* **by** *auto*

definition (*linear-order* :: (′*a* ⇒ ′*a* ⇒ *bool*) ⇒ *bool*) ≡ *linear-order-on* (⊤ :: ′*a* ⇒ *bool*)

lemma *linear-order-eq-linear-order-on*:
(*linear-order* :: (′*a* ⇒ ′*a* ⇒ *bool*) ⇒ *bool*) = *linear-order-on* (⊤ :: ′*a* ⇒ *bool*)
unfolding *linear-order-def* **..**

lemma *linear-order-eq-linear-order-on-uhint* [*uhint*]:
assumes *P* ≡ ⊤ :: ′*a* ⇒ *bool*
shows (*linear-order* :: (′*a* ⇒ ′*a* ⇒ *bool*) ⇒ *bool*) ≡ *linear-order-on P*
using *assms* **by** (*simp add*: *linear-order-eq-linear-order-on*)

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

lemma *linear-orderI* [*intro*]:
assumes *partial-order R*
and *connected R*
shows *linear-order R*
using *assms* **by** (*urule linear-order-onI*)

lemma *linear-orderE* [*elim*]:
assumes *linear-order R*
obtains *partial-order R connected R*
using *assms* **by** (*urule* (*e*) *linear-order-onE*)

lemma *linear-order-on-if-linear-order*:
fixes $P :: 'a \Rightarrow bool$
assumes *linear-order R*
shows *linear-order-on P R*
using *assms* **by** (*elim linear-orderE*)
(*intro linear-order-onI partial-order-on-if-partial-order connected-on-if-connected*)

end

end

Closure Operators

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

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

overloading
idempotent-on-pred $\equiv$ *idempotent-on* :: $('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a) \Rightarrow bool$
begin
definition *idempotent-on-pred* $(P :: 'a \Rightarrow bool)$ $(R :: 'a \Rightarrow 'a \Rightarrow bool)$ $(f :: 'a \Rightarrow 'a) \equiv$
rel-equivalence-on P (*rel-map f R*) *f*
end

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

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-pred-def* **using** *assms* **by** *fastforce*

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

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

end

consts *idempotent* :: $'a \Rightarrow 'b \Rightarrow$ *bool*

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

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

lemma *idempotent-eq-idempotent-on-uhint* [*uhint*]:
assumes $P \equiv (\top :: 'a \Rightarrow bool)$
shows *idempotent* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a) \Rightarrow bool \equiv$ *idempotent-on P*
using *assms* **by** (*simp add*: *idempotent-eq-idempotent-on*)

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

lemma *idempotentI* [*intro*]:
assumes $\bigwedge x.\ f\ x \equiv_R f\ (f\ x)$
shows *idempotent R f*
using *assms* **by** (*urule idempotent-onI*)

lemma *idempotentD* [*dest*]:
assumes *idempotent R f*
shows $f\ x \equiv_R f\ (f\ x)$
using *assms* **by** (*urule* (*e*) *idempotent-onE* **where** *chained* = *insert*) *simp*

lemma *idempotent-on-if-idempotent*:
assumes *idempotent R f*
shows *idempotent-on P R f*
using *assms* **by** (*intro idempotent-onI*) *auto*

end

consts *closure-operator* :: $'a \Rightarrow 'b \Rightarrow$ *bool*

overloading
closure-operator ≡ *closure-operator* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a) \Rightarrow bool$
begin
definition *closure-operator* $(R :: 'a \Rightarrow 'a \Rightarrow bool) :: ('a \Rightarrow 'a) \Rightarrow bool \equiv$
$(R \Rrightarrow_m R) \sqcap$ *inflationary-on* (*in-field R*) *R* $\sqcap$ *idempotent-on* (*in-field R*) *R*
end

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* $(R :: 'a \Rightarrow 'a \Rightarrow bool)\ f$
shows $(R \Rrightarrow_m R)\ f$
using *assms* **by** (*elim closure-operatorE*)

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

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

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

1.4 Orders

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

Summary Basic order-theoretic concepts.

end

1.4.1 Bounded Definite Description

theory *Bounded-Definite-Description*

imports
Bounded-Quantifiers
begin

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

bundle *bounded-the-syntax*
begin
syntax *-bthe* :: $[idt, 'a, bool] \Rightarrow 'b$ $((3THE\ -\ :\ -./\ -)\ [0,\ 0,\ 10]\ 10)$
end
bundle *no-bounded-the-syntax*
begin
no-syntax *-bthe* :: $[idt, 'a, bool] \Rightarrow 'b$ $((3THE\ -\ :\ -./\ -)\ [0,\ 0,\ 10]\ 10)$
end
unbundle *bounded-the-syntax*
translations $THE\ x : P.\ Q \rightleftharpoons CONST\ bthe\ P\ (\lambda x.\ Q)$

definition *bthe-pred* $P\ Q \equiv The\ (P \sqcap Q)$
adhoc-overloading *bthe bthe-pred*

lemma *bthe-eqI* [*intro*]:
assumes $Q\ a$
and $P\ a$
and $\bigwedge x.\ [\![P\ x;\ Q\ x]\!] \Longrightarrow x = a$
shows $(THE\ x : P.\ Q\ x) = a$
unfolding *bthe-pred-def* **by** (*auto intro*: *assms*)

lemma *pred-bthe-if-ex1E*:
assumes $\exists!x : P.\ Q\ x$
obtains $P\ (THE\ x : P.\ Q\ x)\ Q\ (THE\ x : P.\ Q\ x)$
unfolding *bthe-pred-def inf-fun-def* **using** *theI'*[*OF assms*[*unfolded bex1-pred-def*]]
by *auto*

lemma *pred-btheI*:
assumes $\exists!x : P.\ Q\ x$
shows $P\ (THE\ x : P.\ Q\ x)$
using *assms* **by** (*elim pred-bthe-if-ex1E*)

lemma *pred-btheI'*:
assumes $\exists!x : P.\ Q\ x$
shows $Q\ (THE\ x : P.\ Q\ x)$
using *assms* **by** (*elim pred-bthe-if-ex1E*)

end

1.5 Predicates

theory *Predicates*

imports
Bounded-Definite-Description
Bounded-Quantifiers
Predicate-Functions
Predicates-Lattice
Predicates-Order
begin

Summary Basic concepts on predicates.

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 relators, e.g. *injective-on* and *dep-mono-wrt*.
3. Basic concepts on orders and relativised order-theoretic properties, e.g. *partial-equivalence-rel-on*.
4. Galois connections, Galois equivalences, order equivalences, and other related concepts on order functors, e.g. *galois.galois-equivalence*.
5. Basic concepts on predicates.
6. Syntax bundles for HOL `HOL_Syntax_Bundles`.
7. Alignments for concepts that have counterparts in the HOL library - see `HOL_Alignments`.

end

theory *HOL-Mem-Of*

imports
HOL.Set
ML-Unification.ML-Unification-HOL-Setup
begin

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

lemma *mem-of-eq-mem-uhint* [*uhint*]:
assumes $x \equiv x'$
and $A \equiv A'$
shows *mem-of* $A\ x \equiv x' \in A'$
using *assms* **by** *simp*

lemma *mem-of-UNIV-eq-top* [*simp*]: *mem-of UNIV* $= \top$
by *auto*

lemma *mem-of-empty-eq-bot* [*simp*]: *mem-of* $\{\} = \bot$
by *auto*

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 ($(-\hat{}-1)$ [*1000*] *999*) **and**
conversep ($(-\hat{}--1)$ [*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 ($(-\hat{}-1)$ [*1000*] *999*) **and**
conversep ($(-\hat{}--1)$ [*1000*] *1000*)

end

end

1.7.1 Alignment With Binary Relation 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* ⇒ *'a* ⇒ *bool*) ⇒ - ≡
antisymmetric-on (*mem-of S*)
end

lemma *antisymmetric-on-set-eq-antisymmetric-on-pred* [*simp*]:
(*antisymmetric-on* (*S* :: *'a set*) :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*) = *antisymmetric-on* (*mem-of S*)
unfolding *antisymmetric-on-set-def* **by** *simp*

lemma *antisymmetric-on-set-eq-antisymmetric-on-pred-uhint* [*uhint*]:
assumes *P* ≡ *mem-of S*
shows *antisymmetric-on* (*S* :: *'a set*) :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool* ≡ *antisymmetric-on P*
using *assms* **by** *simp*

lemma *antisymmetric-on-set-iff-antisymmetric-on-pred* [*iff*]:
antisymmetric-on (*S* :: *'a set*) (*R* :: *'a* ⇒ *'a* ⇒ *bool*) ⟷ *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* ⇒ *'b* ⇒ *bool*) ⇒ - ≡
rel-injective-on (*mem-of S*)
definition *rel-injective-at-set* (*S* :: *'a set*) :: (*'b* ⇒ *'a* ⇒ *bool*) ⇒ - ≡
rel-injective-at (*mem-of S*)
end

lemma *rel-injective-on-set-eq-rel-injective-on-pred* [*simp*]:
(*rel-injective-on* (*S* :: *'a set*) :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ *bool*) =
rel-injective-on (*mem-of S*)
unfolding *rel-injective-on-set-def* **by** *simp*

lemma *rel-injective-on-set-eq-rel-injective-on-pred-uhint* [*uhint*]:
assumes *P* ≡ *mem-of S*
shows *rel-injective-on* (*S* :: *'a set*) :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ *bool* ≡ *rel-injective-on* *P*
using *assms* **by** *simp*

lemma *rel-injective-on-set-iff-rel-injective-on-pred* [*iff*]:
rel-injective-on (*S* :: *'a set*) (*R* :: *'a* ⇒ *'b* ⇒ *bool*) ⟷ *rel-injective-on* (*mem-of* *S*) *R*
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-eq-rel-injective-at-pred-uhint* [*uhint*]:
assumes *P* ≡ *mem-of S*
shows *rel-injective-at* (*S* :: *'a set*) :: (*'b* ⇒ *'a* ⇒ *bool*) ⇒ *bool* ≡ *rel-injective-at* *P*
using *assms* **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* ⇒ *'a* ⇒ *bool*) ⇒ *bool* ≡

irreflexive-on (*mem-of S*)
end

lemma *irreflexive-on-set-eq-irreflexive-on-pred* [*simp*]:
(*irreflexive-on* (*S* :: *'a set*) :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*) =
irreflexive-on (*mem-of S*)
unfolding *irreflexive-on-set-def* **by** *simp*

lemma *irreflexive-on-set-eq-irreflexive-on-pred-uhint* [*uhint*]:
assumes *P* ≡ *mem-of S*
shows *irreflexive-on* (*S* :: *'a set*) :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool* ≡ *irreflexive-on P*
using *assms* **by** *simp*

lemma *irreflexive-on-set-iff-irreflexive-on-pred* [*iff*]:
irreflexive-on (*S* :: *'a set*) (*R* :: *'a* ⇒ *'a* ⇒ *bool*) ⟷ *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* ⇒ *'b* ⇒ *bool*) ⇒ - ≡
left-total-on (*mem-of S*)
end

lemma *left-total-on-set-eq-left-total-on-pred* [*simp*]:
(*left-total-on* (*S* :: *'a set*) :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ *bool*) =
left-total-on (*mem-of S*)
unfolding *left-total-on-set-def* **by** *simp*

lemma *left-total-on-set-eq-left-total-on-pred-uhint* [*uhint*]:
assumes *P* ≡ *mem-of S*
shows *left-total-on* (*S* :: *'a set*) :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ *bool* ≡ *left-total-on P*
using *assms* **by** *simp*

lemma *left-total-on-set-iff-left-total-on-pred* [*iff*]:
left-total-on (*S* :: *'a set*) (*R* :: *'a* ⇒ *'b* ⇒ *bool*) ⟷ *left-total-on* (*mem-of S*) *R*
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* ⇒ *'a* ⇒ *bool*) ⇒ - ≡
reflexive-on (*mem-of S*)
end

lemma *reflexive-on-set-eq-reflexive-on-pred* [*simp*]:
(*reflexive-on* (*S* :: *'a set*) :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*) = *reflexive-on* (*mem-of S*)
unfolding *reflexive-on-set-def* **by** *simp*

lemma *reflexive-on-set-eq-reflexive-on-pred-uhint* [*uhint*]:
assumes *P* ≡ *mem-of S*
shows *reflexive-on* (*S* :: *'a set*) :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool* ≡ *reflexive-on P*
using *assms* **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* ⇒ *'b* ⇒ *bool*) ⇒ - ≡
right-unique-on (*mem-of S*)
definition *right-unique-at-set* (*S* :: *'a set*) :: (*'b* ⇒ *'a* ⇒ *bool*) ⇒ - ≡
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* ⇒ *'b* ⇒ *bool*) ⇒ *bool*) = *right-unique-on* (*mem-of S*)
unfolding *right-unique-on-set-def* **by** *simp*

lemma *right-unique-on-set-eq-right-unique-on-pred-uhint* [*uhint*]:
assumes *P* ≡ *mem-of S*
shows *right-unique-on* (*S* :: *'a set*) :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ *bool* ≡ *right-unique-on P*
using *assms* **by** *simp*

lemma *right-unique-on-set-iff-right-unique-on-pred* [*iff*]:
right-unique-on (*S* :: *'a set*) (*R* :: *'a* ⇒ *'b* ⇒ *bool*) ⟷ *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* ⇒ *'a* ⇒ *bool*) ⇒ - ≡
rel-surjective-at (*mem-of S*)
end

lemma *rel-surjective-at-set-eq-rel-surjective-at-pred* [*simp*]:
(*rel-surjective-at* (*S* :: *'a set*) :: (*'b* ⇒ *'a* ⇒ *bool*) ⇒ *bool*) = *rel-surjective-at* (*mem-of S*)
unfolding *rel-surjective-at-set-def* **by** *simp*

lemma *rel-surjective-at-set-eq-rel-surjective-at-pred-uhint* [*uhint*]:
assumes *P* ≡ *mem-of S*
shows *rel-surjective-at* (*S* :: *'a set*) :: (*'b* ⇒ *'a* ⇒ *bool*) ⇒ *bool* ≡ *rel-surjective-at P*
using *assms* **by** *simp*

lemma *rel-surjective-at-set-iff-rel-surjective-at-pred* [*iff*]:
rel-surjective-at (*S* :: *'a set*) (*R* :: *'b* ⇒ *'a* ⇒ *bool*) ⟷ *rel-surjective-at* (*mem-of S*) *R*
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 ≡ *symmetric-on* :: *'a set* ⇒ (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*
begin
definition *symmetric-on-set* (*S* :: *'a set*) :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ - ≡
symmetric-on (*mem-of S*)
end

lemma *symmetric-on-set-eq-symmetric-on-pred* [*simp*]:
(*symmetric-on* (*S* :: *'a set*) :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*) = *symmetric-on* (*mem-of S*)
unfolding *symmetric-on-set-def* **by** *simp*

lemma *symmetric-on-set-eq-symmetric-on-pred-uhint* [*uhint*]:
assumes *P* ≡ *mem-of S*
shows *symmetric-on* (*S* :: *'a set*) :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool* ≡ *symmetric-on P*
using *assms* **by** *simp*

lemma *symmetric-on-set-iff-symmetric-on-pred* [*iff*]:
symmetric-on (*S* :: *'a set*) (*R* :: *'a* ⇒ *'a* ⇒ *bool*) ⟷ *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 ≡ *transitive-on* :: *'a set* ⇒ (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*
begin
definition *transitive-on-set* (*S* :: *'a set*) :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ - ≡
transitive-on (*mem-of S*)
end

lemma *transitive-on-set-eq-transitive-on-pred* [*simp*]:
(*transitive-on* (*S* :: *'a set*) :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool*) = *transitive-on* (*mem-of S*)
unfolding *transitive-on-set-def* **by** *simp*

lemma *transitive-on-set-eq-transitive-on-pred-uhint* [*uhint*]:
assumes *P* ≡ *mem-of S*
shows *transitive-on* (*S* :: *'a set*) :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool* ≡ *transitive-on P*
using *assms* **by** *simp*

lemma *transitive-on-set-iff-transitive-on-pred* [*iff*]:
transitive-on (*S* :: *'a set*) (*R* :: *'a* ⇒ *'a* ⇒ *bool*) ⟷ *transitive-on* (*mem-of S*) *R*
by *simp*

lemma *transp-eq-transitive* [*HOL-bin-rel-alignment*]: *transp* = *transitive*
by (*intro ext*) (*blast intro*: *transpI dest*: *transpD*)

Bi-Total **lemma** *bi-total-on-set-eq-bi-total-on-pred* [*simp*]:
(*bi-total-on* (*S* :: *'a set*) (*T* :: *'b set*) :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ *bool*) =
bi-total-on (*mem-of S*) (*mem-of T*)
by *auto*

lemma *bi-total-on-set-eq-bi-total-on-pred-uhint* [*uhint*]:
assumes *P* ≡ *mem-of S*
and *Q* ≡ *mem-of T*
shows *bi-total-on* (*S* :: *'a set*) (*T* :: *'b set*) :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ *bool* ≡ *bi-total-on P Q*
using *assms* **by** *simp*

lemma *bi-total-on-set-iff-bi-total-on-pred* [*iff*]:
bi-total-on (*S* :: *'a set*) (*T* :: *'b set*) (*R* :: *'a* ⇒ *'b* ⇒ *bool*) ⟷
bi-total-on (*mem-of S*) (*mem-of T*) *R*
by *simp*

lemma *Transfer-bi-total-eq-bi-total* [*HOL-bin-rel-alignment*]:
Transfer.bi-total = *Binary-Relations-Bi-Total.bi-total*
unfolding *bi-total-alt-def* **by** (*auto simp add*: *HOL-bin-rel-alignment*)

Bi-Unique **lemma** *bi-unique-on-set-eq-bi-unique-on-pred* [*simp*]:
(*bi-unique-on* (*S* :: *'a set*) :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ *bool*) = *bi-unique-on* (*mem-of S*)
by *auto*

lemma *bi-unique-on-set-eq-bi-unique-on-pred-uhint* [*uhint*]:
assumes *P* ≡ *mem-of S*
shows *bi-unique-on* (*S* :: *'a set*) :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ *bool* ≡ *bi-unique-on P*
using *assms* **by** *simp*

lemma *bi-unique-on-set-iff-bi-unique-on-pred* [*iff*]:
bi-unique-on (*S* :: *'a set*) (*R* :: *'a* ⇒ *'b* ⇒ *bool*) ⟷ *bi-unique-on* (*mem-of S*) *R*
by *simp*

lemma *Transfer-bi-unique-eq-bi-unique* [*HOL-bin-rel-alignment*]:
Transfer.bi-unique = *Binary-Relations-Bi-Unique.bi-unique*
unfolding *bi-unique-alt-def* **by** (*auto simp add*: *HOL-bin-rel-alignment*)

Functions **lemma** *Domainp-eq-in-dom* [*HOL-bin-rel-alignment*]: *Domainp* = *in-dom*
by (*intro ext*) *auto*

lemma *Rangep-eq-in-codom* [*HOL-bin-rel-alignment*]: *Rangep* = *in-codom*
by (*intro ext*) *auto*

lemma *relcompp-eq-rel-comp* [*HOL-bin-rel-alignment*]: *relcompp* = *rel-comp*
by (*intro ext*) *auto*

lemma *conversep-eq-rel-inv* [*HOL-bin-rel-alignment*]: *conversep* = *rel-inv*
by (*intro ext*) *auto*

lemma *eq-onp-eq-eq-restrict* [*HOL-bin-rel-alignment*]: *eq-onp* = *rel-restrict-left* (=)
unfolding *eq-onp-def* **by** (*intro ext*) *auto*

definition *rel-restrict-left-set* (*R* :: *'a* ⇒ *'b* ⇒ *bool*) (*S* :: *'a set*) ≡ $R\restriction_{\textit{mem-of } S}$
adhoc-overloading *rel-restrict-left rel-restrict-left-set*

definition *rel-restrict-right-set* (*R* :: *'a* ⇒ *'b* ⇒ *bool*) (*S* :: *'b set*) ≡ $R\downharpoonright_{\textit{mem-of } S}$
adhoc-overloading *rel-restrict-right rel-restrict-right-set*

lemma *rel-restrict-left-set-eq-restrict-left-pred* [*simp*]:
$R\restriction_S = R\restriction_{\textit{mem-of } S}$
unfolding *rel-restrict-left-set-def* **by** *simp*

lemma *rel-restrict-left-set-eq-restrict-left-pred-uhint* [*uhint*]:
assumes *R* ≡ *R'*
and *P* ≡ *mem-of S*
shows $R\restriction_S \equiv R'\restriction_P$
using *assms* **by** *simp*

lemma *restrict-left-set-iff-restrict-left-pred* [*iff*]: $R\restriction_S\ x\ y \longleftrightarrow R\restriction_{\textit{mem-of } S}\ x\ y$
by *simp*

lemma *rel-restrict-right-set-eq-restrict-right-pred* [*simp*]:
$R\downharpoonright_S = R\downharpoonright_{\textit{mem-of } S}$
unfolding *rel-restrict-right-set-def* **by** *simp*

lemma *rel-restrict-right-set-eq-restrict-right-pred-uhint* [*uhint*]:
assumes *R* ≡ *R'*
and *P* ≡ *mem-of S*
shows $R\downharpoonright_S \equiv R'\downharpoonright_P$
using *assms* **by** *simp*

lemma *restrict-right-set-iff-restrict-right-pred* [*iff*]: $R\downharpoonright_S\ x\ y \longleftrightarrow R\downharpoonright_{\textit{mem-of } S}\ x\ y$
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 Function Definitions from HOL.Main

theory *HOL-Alignment-Functions*
imports
HOL-Alignment-Binary-Relations
HOL-Syntax-Bundles-Functions
LFunctions
begin

unbundle *no-HOL-function-syntax*

named-theorems *HOL-fun-alignment*

Functions

Bijection **definition** *bijection-on-set* (*S* :: *'a set*) (*S'* :: *'b set*) :: (*'a* ⇒ *'b*) ⇒ (*'b* ⇒ *'a*) ⇒ *bool* ≡
bijection-on (*mem-of S*) (*mem-of S'*)
adhoc-overloading *bijection-on bijection-on-set*

lemma *bijection-on-set-eq-bijection-on-pred* [*simp*]:
bijection-on (*S* :: *'a set*) (*S'* :: *'b set*) = *bijection-on* (*mem-of S*) (*mem-of S'*)
unfolding *bijection-on-set-def* **by** *simp*

lemma *bijection-on-set-eq-bijection-on-pred-uhint* [*uhint*]:
assumes *P* ≡ *mem-of S*
and *Q* ≡ *mem-of S'*
shows *bijection-on S S'* ≡ *bijection-on P Q*
using *assms* **by** *simp*

lemma *bijection-on-set-iff-bijection-on-pred* [*iff*]:
bijection-on (*S* :: *'a set*) (*S'* :: *'b set*) (*f* :: *'a* ⇒ *'b*) (*g* :: *'b* ⇒ *'a*) ⟷
bijection-on (*mem-of S*) (*mem-of S'*) *f g*
by *simp*

lemma *bij-betw-bijection-onE*:
assumes *bij-betw* (*f* :: *'a* ⇒ *'b*) *S S'*
obtains *g* :: *'b* ⇒ *'a* **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* :: *′a* ⇒ *′b*) (*g* :: *′b* ⇒ *′a*)
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* :: *′a* ⇒ *′b*) *S S′* ⟷ (∃(*g* :: *′b* ⇒ *′a*). *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-eq-injective-on-pred-uhint* [*uhint*]:
assumes *P* ≡ *mem-of S*
shows *injective-on* (*S* :: *′a set*) :: (*′a* ⇒ *′b*) ⇒ - ≡ *injective-on P*
using *assms* **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* ⟷ *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 ≡ *inverse-on* :: *′a set* ⇒ (*′a* ⇒ *′b*) ⇒ (*′b* ⇒ *′a*) ⇒ *bool*
begin
definition *inverse-on-set* (*S* :: *′a set*) :: (*′a* ⇒ *′b*) ⇒ (*′b* ⇒ *′a*) ⇒ *bool* ≡
inverse-on (*mem-of S*)

end

lemma *inverse-on-set-eq-inverse-on-pred* [*simp*]:
(*inverse-on* (*S* :: *'a set*) :: (*'a* ⇒ *'b*) ⇒ (*'b* ⇒ *'a*) ⇒ -) = *inverse-on* (*mem-of S*)
unfolding *inverse-on-set-def* **by** *simp*

lemma *inverse-on-set-eq-inverse-on-pred-uhint* [*uhint*]:
assumes *P* ≡ *mem-of S*
shows *inverse-on* (*S* :: *'a set*) :: (*'a* ⇒ *'b*) ⇒ (*'b* ⇒ *'a*) ⇒ - ≡ *inverse-on P*
using *assms* **by** *simp*

lemma *inverse-on-set-iff-inverse-on-pred* [*iff*]:
inverse-on (*S* :: *'a set*) (*f* :: *'a* ⇒ *'b*) (*g* :: *'b* ⇒ *'a*) ⟷ *inverse-on* (*mem-of S*)
f g
by *simp*

Monotone **lemma** *monotone-on-eq-mono-wrt-rel-restrict-left-right* [*HOL-fun-alignment*]:
monotone-on S R = *mono-wrt-rel R*⌈*S*⌉*S*
by (*intro ext*) (*auto 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 ≡ *surjective-at* :: *'a set* ⇒ (*'b* ⇒ *'a*) ⇒ *bool*
begin
definition *surjective-at-set* (*S* :: *'a set*) :: (*'b* ⇒ *'a*) ⇒ *bool* ≡
surjective-at (*mem-of S*)
end

lemma *surjective-at-set-eq-surjective-at-pred* [*simp*]:
(*surjective-at* (*S* :: *'a set*) :: (*'b* ⇒ *'a*) ⇒ -) = *surjective-at* (*mem-of S*)
unfolding *surjective-at-set-def* **by** *simp*

lemma *surjective-at-set-eq-surjective-at-pred-uhint* [*uhint*]:
assumes *P* ≡ *mem-of S*
shows *surjective-at* (*S* :: *'a set*) :: (*'b* ⇒ *'a*) ⇒ - ≡ *surjective-at P*
using *assms* **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*]: *BNF-Def.rel-fun* = *Fun-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 Order Definitions from HOL

theory *HOL-Alignment-Orders*
imports
HOL−Library.Preorder
HOL-Alignment-Binary-Relations
HOL-Syntax-Bundles-Orders
Orders
begin

named-theorems *HOL-order-alignment*

Functions **definition** *rel R x y* ≡ (*x*, *y*) ∈ *R*
lemma *rel-of-eq* [*simp*]: *rel* = (λ*R x y*. (*x*, *y*) ∈ *R*) **unfolding** *rel-def* **by** *simp*
lemma *rel-of-iff* [*iff*]: *rel R x y* ⟷ (*x*, *y*) ∈ *R* **by** *simp*

Bi-Related **overloading**
bi-related-set ≡ *bi-related* :: ′*a rel* ⇒ ′*a* ⇒ ′*a* ⇒ *bool*
begin
definition *bi-related-set* (*S* :: ′*a rel*) ≡ *bi-related* (*rel S*) :: ′*a* ⇒ ′*a* ⇒ *bool*
end

lemma *bi-related-set-eq-bi-related-pred* [*simp*]:
((≡$_{S\ ::\ 'a\ rel}$) :: ′*a* ⇒ ′*a* ⇒ *bool*) = (≡$_{rel\ S}$)
unfolding *bi-related-set-def* **by** *simp*

lemma *bi-related-set-eq-bi-related-pred-uhint* [*uhint*]:
assumes *R* ≡ *rel S*
shows (≡$_{S\ ::\ 'a\ rel}$) :: ′*a* ⇒ ′*a* ⇒ *bool* ≡ (≡$_{R}$)
using *assms* **by** *simp*

lemma *bi-related-set-iff-bi-related-pred* [*iff*]: (*x* :: ′*a*) ≡$_{(S\ ::\ 'a\ rel)}$ (*y* :: ′*a*) ⟷ *x* ≡$_{rel\ S}$ *y*
by *simp*

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* ⇒ *'b* ⇒ *bool*) ⇒ (*'a* ⇒ *'b*) ⇒ *bool*
begin
definition *inflationary-on-set* (*S* :: *'a set*) :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ (*'a* ⇒ *'b*) ⇒ *bool* ≡
inflationary-on (*mem-of S*)
end

lemma *inflationary-on-set-eq-inflationary-on-pred* [*simp*]:
(*inflationary-on* (*S* :: *'a set*) :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ (*'a* ⇒ *'b*) ⇒ *bool*) =
inflationary-on (*mem-of S*)
unfolding *inflationary-on-set-def* **by** *simp*

lemma *inflationary-on-set-eq-inflationary-on-pred-uhint* [*uhint*]:
assumes *P* ≡ *mem-of S*
shows *inflationary-on* (*S* :: *'a set*) :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ (*'a* ⇒ *'b*) ⇒ *bool* ≡ *inflationary-on P*
using *assms* **by** *simp*

lemma *inflationary-on-set-iff-inflationary-on-pred* [*iff*]:
inflationary-on (*S* :: *'a set*) (*R* :: *'a* ⇒ *'b* ⇒ *bool*) (*f* :: *'a* ⇒ *'b*) ⟷
inflationary-on (*mem-of S*) *R f*
by *simp*

Deflationary overloading
deflationary-on-set ≡ *deflationary-on* :: *'a set* ⇒ (*'b* ⇒ *'a* ⇒ *bool*) ⇒ (*'a* ⇒ *'b*) ⇒ *bool*
begin
definition *deflationary-on-set* (*S* :: *'a set*) :: (*'b* ⇒ *'a* ⇒ *bool*) ⇒ (*'a* ⇒ *'b*) ⇒ *bool* ≡
deflationary-on (*mem-of S*)
end

lemma *deflationary-on-set-eq-deflationary-on-pred* [*simp*]:
(*deflationary-on* (*S* :: *'a set*) :: (*'b* ⇒ *'a* ⇒ *bool*) ⇒ (*'a* ⇒ *'b*) ⇒ *bool*) = *deflationary-on* (*mem-of S*)
unfolding *deflationary-on-set-def* **by** *simp*

lemma *deflationary-on-set-eq-deflationary-on-pred-uhint* [*uhint*]:
assumes *P* ≡ *mem-of S*
shows *deflationary-on* (*S* :: *'a set*) :: (*'b* ⇒ *'a* ⇒ *bool*) ⇒ (*'a* ⇒ *'b*) ⇒ *bool* ≡ *deflationary-on P*
using *assms* **by** *simp*

lemma *deflationary-on-set-iff-deflationary-on-pred* [*iff*]:

deflationary-on (*S* :: *'a set*) (*R* :: *'b* ⇒ *'a* ⇒ *bool*) (*f* :: *'a* ⇒ *'b*) ⟷ *deflationary-on* (*mem-of S*) *R f*
by *simp*

Idempotent **overloading**
idempotent-on-set ≡ *idempotent-on* :: *'a set* ⇒ (*'a* ⇒ *'a* ⇒ *bool*) ⇒ (*'a* ⇒ *'a*) ⇒ *bool*
begin
definition *idempotent-on-set* (*S* :: *'a set*) :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ (*'a* ⇒ *'a*) ⇒ *bool* ≡
idempotent-on (*mem-of S*)
end

lemma *idempotent-on-set-eq-idempotent-on-pred* [*simp*]:
(*idempotent-on* (*S* :: *'a set*) :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ (*'a* ⇒ *'a*) ⇒ *bool*) = *idempotent-on* (*mem-of S*)
unfolding *idempotent-on-set-def* **by** *simp*

lemma *idempotent-on-set-iff-idempotent-on-pred* [*iff*]:
idempotent-on (*S* :: *'a set*) (*R* :: *'a* ⇒ *'a* ⇒ *bool*) (*f* :: *'a* ⇒ *'a*) ⟷ *idempotent-on* (*mem-of S*) *R f*
by *simp*

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

1.8.2 Alignment With Predicate Definitions from HOL

theory *HOL-Alignment-Predicates*
imports
Main
HOL-Mem-Of
Predicates
begin

named-theorems *HOL-predicate-alignment*

Quantifiers **adhoc-overloading** *ball Ball*

lemma *Ball-eq-ball-pred* [*HOL-predicate-alignment*]: $\forall_A = \forall_{mem\text{-}of\ A}$ **by** *auto*

lemma *Ball-eq-ball-pred-uhint* [*uhint*]:
assumes $P \equiv mem\text{-}of\ A$
shows $\forall_A = \forall_P$
using *assms* **by** (*simp add*: *Ball-eq-ball-pred*)

lemma *Ball-iff-ball-pred* [*HOL-predicate-alignment*]: $(\forall x : A.\ Q\ x) \longleftrightarrow (\forall x : mem\text{-}of\ A.\ Q\ x)$
by (*simp add*: *Ball-eq-ball-pred*)

adhoc-overloading *bex Bex*

lemma *Bex-eq-bex-pred* [*HOL-predicate-alignment*]: $\exists_A = \exists_{mem\text{-}of\ A}$ **by** *fast*

lemma *Bex-eq-bex-pred-uhint* [*uhint*]:
assumes $P \equiv mem\text{-}of\ A$
shows $\exists_A = \exists_P$
using *assms* **by** (*simp add*: *Bex-eq-bex-pred*)

lemma *Bex-iff-bex-pred* [*HOL-predicate-alignment*]: $(\exists x : A.\ Q\ x) \longleftrightarrow (\exists x : mem\text{-}of\ A.\ Q\ x)$
by (*simp add*: *Bex-eq-bex-pred*)

end

1.9 HOL Alignments

theory *HOL-Alignments*
imports

HOL-Alignment-Binary-Relations
HOL-Alignment-Functions
HOL-Alignment-Orders
HOL-Alignment-Predicates
begin

Summary Alignment of concepts with HOL counterparts

end

1.9.1 Alignment With Order Definitions from HOL-Algebra

theory *HOL-Algebra-Alignment-Orders*
imports
HOL−Algebra.Order
HOL-Alignment-Orders
begin

named-theorems *HOL-Algebra-order-alignment*

context *equivalence*
begin

lemma *reflexive-on-carrier* [*HOL-Algebra-order-alignment*]:
reflexive-on (*carrier S*) (.=)
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 $\longleftrightarrow$ *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*) ($\sqsubseteq$)
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 Galois Definitions from HOL-Algebra

theory *HOL-Algebra-Alignment-Galois*
imports
HOL−Algebra.Galois-Connection
HOL-Algebra-Alignment-Orders
Galois
begin

named-theorems *HOL-Algebra-galois-alignment*

context *galois-connection*
begin

context
fixes *L R l r*
defines $L \equiv (\sqsubseteq_{\mathcal{X}})\restriction_{carrier\ \mathcal{X}}\uparrow_{carrier\ \mathcal{X}}$ **and** $R \equiv (\sqsubseteq_{\mathcal{Y}})\restriction_{carrier\ \mathcal{Y}}\uparrow_{carrier\ \mathcal{Y}}$

and $l \equiv \pi^*$ **and** $r \equiv \pi_*$
notes *defs*[*simp*] = *L-def R-def l-def r-def* **and** *rel-restrict-right-eq*[*simp*]
and *rel-restrict-leftI*[*intro*!] *rel-restrict-leftE*[*elim*!]
begin

interpretation *galois L R l r* **.**

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** (*intro half-galois-prop-leftI*) *fastforce*

lemma *half-galois-prop-right* [*HOL-Algebra-galois-alignment*]: $(L \trianglelefteq_h R)\ l\ r$
using *galois-property upper-closed* **by** (*intro half-galois-prop-rightI*) *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 $L \equiv (\sqsubseteq_{\mathcal{X}})\restriction_{carrier\ \mathcal{X}}\restriction_{carrier\ \mathcal{X}}$ **and** $R \equiv (\sqsubseteq_{\mathcal{Y}})\restriction_{carrier\ \mathcal{Y}}\restriction_{carrier\ \mathcal{Y}}$
and $l \equiv \pi^*$ **and** $r \equiv \pi_*$
notes *defs*[*simp*] = *L-def R-def l-def r-def* **and** *rel-restrict-right-eq*[*simp*]
and *rel-restrict-leftI*[*intro*!] *rel-restrict-leftE*[*elim*!]
in-codom-rel-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* $\equiv$
  $((\leq_L) \dashv (\leq_R))\ l\ r$
  $\wedge$ *preorder-on* (*in-field* $(\leq_L)$) $(\leq_L)$
  $\wedge$ *preorder-on* (*in-field* $(\leq_R)$) $(\leq_R)$

notation *transport.preorder-galois-connection* (**infix** $\dashv_{pre}$ *50*)

lemma *preorder-galois-connectionI* [*intro*]:
  assumes $((\leq_L) \dashv (\leq_R))\ l\ r$
  and *preorder-on* (*in-field* $(\leq_L)$) $(\leq_L)$
  and *preorder-on* (*in-field* $(\leq_R)$) $(\leq_R)$
  shows $((\leq_L) \dashv_{pre} (\leq_R))\ l\ r$
  unfolding *preorder-galois-connection-def* **using** *assms* **by** *blast*

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

assumes $((\leq_L) \dashv_{pre} (\leq_R))\ l\ r$
obtains $((\leq_L) \dashv (\leq_R))\ l\ r$ *preorder-on* (*in-field* $(\leq_L)$) $(\leq_L)$
preorder-on (*in-field* $(\leq_R)$) $(\leq_R)$
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*]:
$((\leq_R) \dashv_{pre} (\leq_L))^{-1} = ((\geq_L) \dashv_{pre} (\geq_R))$
by (*intro ext*) (*auto intro*!: *t.preorder-galois-connectionI*)

end

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

definition *partial-equivalence-rel-galois-connection* $\equiv$
$((\leq_L) \dashv (\leq_R))\ l\ r$
$\wedge$ *partial-equivalence-rel* $(\leq_L)$
$\wedge$ *partial-equivalence-rel* $(\leq_R)$

notation *transport.partial-equivalence-rel-galois-connection* (**infix** $\dashv_{PER}$ *50*)

lemma *partial-equivalence-rel-galois-connectionI* [*intro*]:
assumes $((\leq_L) \dashv (\leq_R))\ l\ r$
and *partial-equivalence-rel-on* (*in-field* $(\leq_L)$) $(\leq_L)$
and *partial-equivalence-rel-on* (*in-field* $(\leq_R)$) $(\leq_R)$
shows $((\leq_L) \dashv_{PER} (\leq_R))\ l\ r$
unfolding *partial-equivalence-rel-galois-connection-def* **using** *assms* **by** *blast*

lemma *partial-equivalence-rel-galois-connectionE* [*elim*]:
assumes $((\leq_L) \dashv_{PER} (\leq_R))\ l\ r$
obtains $((\leq_L) \dashv_{pre} (\leq_R))\ l\ r$ *symmetric* $(\leq_L)$ *symmetric* $(\leq_R)$
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*]: $((\leq_R) \dashv_{PER} (\leq_L))^{-1} = ((\geq_L) \dashv_{PER} (\geq_R))$
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
Restricted-Equality
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*) (*fastforce dest*: *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* ⇒ *'a* ⇒ *bool*
and *R* :: *'b* ⇒ *'b* ⇒ *bool*
and *l* :: *'a* ⇒ *'b*
and *r* :: *'b* ⇒ *'a*
assumes *reflexive-on-in-field-left*: *reflexive-on* (*in-field L*) *L*
and *reflexive-on-in-field-right*: *reflexive-on* (*in-field R*) *R*
and *transport-bijection*: *transport-bijection L R l r*
begin

sublocale *tbij?* : *transport-bijection L R l r*
rewrites *reflexive-on* (*in-field L*) *L* ≡ *True*
and *reflexive-on* (*in-field R*) *R* ≡ *True*
and ⋀*P*. (*True* ⟹ *P*) ≡ *Trueprop P*
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* ≡ *True*
  and *preorder-on* (*in-field R*) *R* ≡ *True*
  and $\bigwedge P.$ (*True* ⟹ *P*) ≡ *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* ≡ *True*
  and *partial-equivalence-rel R* ≡ *True*
  and $\bigwedge P.$ (*True* ⟹ *P*) ≡ *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*
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*: $({}_L\lessapprox) = (\mathit{galois\text{-}rel.Galois}\ (=)\ (=)\ r)\restriction_P\restriction_Q$
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*
rel-restrict-right-eq rel-inv-eq-self-if-symmetric)+
auto

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** $\leq_{L1}$ *50*)
notation *R1* (**infix** $\leq_{R1}$ *50*)
notation *L2* (**infix** $\leq_{L2}$ *50*)
notation *R2* (**infix** $\leq_{R2}$ *50*)

notation *g1.ge-left* (**infix** $\geq_{L1}$ *50*)
notation *g1.ge-right* (**infix** $\geq_{R1}$ *50*)
notation *g2.ge-left* (**infix** $\geq_{L2}$ *50*)
notation *g2.ge-right* (**infix** $\geq_{R2}$ *50*)

notation *g1.left-Galois* (**infix** $_{L1}\lessapprox$ *50*)
notation *g1.right-Galois* (**infix** $_{R1}\lessapprox$ *50*)
notation *g2.left-Galois* (**infix** $_{L2}\lessapprox$ *50*)
notation *g2.right-Galois* (**infix** $_{R2}\lessapprox$ *50*)

notation *g1.ge-Galois-left* (**infix** $\gtrapprox_{L1}$ *50*)
notation *g1.ge-Galois-right* (**infix** $\gtrapprox_{R1}$ *50*)
notation *g2.ge-Galois-left* (**infix** $\gtrapprox_{L2}$ *50*)
notation *g2.ge-Galois-right* (**infix** $\gtrapprox_{R2}$ *50*)

notation *g1.right-ge-Galois* (**infix** $_{R1}\gtrapprox$ *50*)
notation *g1.Galois-right* (**infix** $\lessapprox_{R1}$ *50*)
notation *g2.right-ge-Galois* (**infix** $_{R2}\gtrapprox$ *50*)
notation *g2.Galois-right* (**infix** $\lessapprox_{R2}$ *50*)

notation *g1.left-ge-Galois* (**infix** $_{L1}\gtrapprox$ *50*)
notation *g1.Galois-left* (**infix** $\lessapprox_{L1}$ *50*)
notation *g2.left-ge-Galois* (**infix** $_{L2}\gtrapprox$ *50*)
notation *g2.Galois-left* (**infix** $\lessapprox_{L2}$ *50*)

notation *g1.unit* (η_1)
notation *g1.counit* (ε_1)
notation *g2.unit* (η_2)
notation *g2.counit* (ε_2)

abbreviation (*input*) $L \equiv L1$

definition $l \equiv l2 \circ l1$

lemma *left-eq-comp*: $l = l2 \circ l1$
  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** $\leq_L$ *50*)
notation R (**infix** $\leq_R$ *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 $((\leq_{L1}) \Rightarrow_m (\leq_{R1}))\ l1\ ((\leq_{L2}) \Rightarrow_m (\leq_{R2}))\ l2$
and $\bigwedge x\ y.\ x \leq_{L1} y \Longrightarrow l1\ x \leq_{R1} l1\ y \Longrightarrow l1\ x \leq_{L2} l1\ y$
shows $((\leq_L) \Rightarrow_m (\leq_R))\ l$
unfolding *left-eq-comp* **using** *assms* **by** (*urule dep-mono-wrt-rel-compI*)

end

context *transport-comp-same*
begin

lemma *mono-wrt-rel-leftI*:
assumes $((\leq_{L1}) \Rightarrow_m (\leq_{R1}))\ l1\ ((\leq_{R1}) \Rightarrow_m (\leq_{R2}))\ l2$
shows $((\leq_L) \Rightarrow_m (\leq_R))\ l$
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*: $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
  and *galois2*: $((\leq_{L2}) \trianglelefteq (\leq_{R2}))$ *l2 r2*
  and *mono-l1*: $(\textit{in-dom}\ (\leq_{L1}) \Rrightarrow_m \textit{in-dom}\ (\leq_{L2}))$ *l1*
  and *mono-r2*: $(\textit{in-codom}\ (\leq_{R2}) \Rrightarrow_m \textit{in-codom}\ (\leq_{R1}))$ *r2*
  and *agree*: $(\textit{in-dom}\ (\leq_{L1}) \Rrightarrow \textit{in-codom}\ (\leq_{R2}) \Rrightarrow (\longleftrightarrow))$
    $(\textit{rel-bimap}\ l1\ r2\ (\leq_{R1}))\ (\textit{rel-bimap}\ l1\ r2\ (\leq_{L2}))$
  shows $((\leq_L) \trianglelefteq (\leq_R))\ l\ r$
proof (*rule galois-prop.galois-propI'*)
  fix *x y* **assume** $\textit{in-dom}\ (\leq_L)\ x\ \textit{in-codom}\ (\leq_R)\ y$
  with *mono-r2 mono-l1* **have** $\textit{in-dom}\ (\leq_{L2})\ (l1\ x)\ \textit{in-codom}\ (\leq_{R1})\ (r2\ y)$ **by** *auto*
  have $x \leq_L r\ y \longleftrightarrow x \leq_{L1} r1\ (r2\ y)$ **by** *simp*
  also from *galois1* ‹$\textit{in-dom}\ (\leq_{L1})\ x$› ‹$\textit{in-codom}\ (\leq_{R1})\ (r2\ y)$›
    have $\ldots \longleftrightarrow l1\ x \leq_{R1} r2\ y$
    by (*rule g1.galois-prop-left-rel-right-iff-left-right-rel*)
  also from *agree* ‹$\textit{in-dom}\ (\leq_{L1})\ x$› ‹$\textit{in-codom}\ (\leq_{R2})\ y$›
    have $\ldots \longleftrightarrow l1\ x \leq_{L2} r2\ y$ **by** *fastforce*
  also from *galois2* ‹$\textit{in-dom}\ (\leq_{L2})\ (l1\ x)$› ‹$\textit{in-codom}\ (\leq_{R2})\ y$›
    have $\ldots \longleftrightarrow l\ x \leq_{R2} y$
    unfolding *l-def*
    by (*simp add*: *g2.galois-prop-left-rel-right-iff-left-right-rel*)
  finally show $x \leq_L r\ y \longleftrightarrow l\ x \leq_R y$ **.**
qed

end

context *transport-comp-same*
begin

corollary *galois-propI*:
  assumes $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
  and $((\leq_{R1}) \trianglelefteq (\leq_{R2}))$ *l2 r2*
  and $(\textit{in-dom}\ (\leq_{L1}) \Rrightarrow_m \textit{in-dom}\ (\leq_{R1}))$ *l1*
  and $(\textit{in-codom}\ (\leq_{R2}) \Rrightarrow_m \textit{in-codom}\ (\leq_{R1}))$ *r2*
  shows $((\leq_L) \trianglelefteq (\leq_R))\ l\ r$
  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*: $((\leq_{L1}) \dashv (\leq_{R1}))$ *l1 r1* $((\leq_{L2}) \dashv (\leq_{R2}))$ *l2 r2*
and *mono-L1-L2-l1*: $\bigwedge x\ y.\ x \leq_{L1} y \Longrightarrow l1\ x \leq_{R1} l1\ y \Longrightarrow l1\ x \leq_{L2} l1\ y$
and *mono-R2-R1-r2*: $\bigwedge x\ y.\ x \leq_{R2} y \Longrightarrow r2\ x \leq_{L2} r2\ y \Longrightarrow r2\ x \leq_{R1} r2\ y$
and (*in-dom* $(\leq_{L1}) \Rrightarrow$ *in-codom* $(\leq_{R2}) \Rrightarrow (\longleftrightarrow)$) (*rel-bimap l1 r2* $(\leq_{R1})$) (*rel-bimap l1 r2* $(\leq_{L2})$)
shows $((\leq_L) \dashv (\leq_R))$ *l r*
proof −
from *galois mono-L1-L2-l1* **have** (*in-dom* $(\leq_{L1}) \Rrightarrow_m$ *in-dom* $(\leq_{L2})$) *l1*
by (*intro mono-wrt-predI*) (*blast elim*!: *in-domE g1.galois-connectionE*)
moreover from *galois mono-R2-R1-r2*
have (*in-codom* $(\leq_{R2}) \Rrightarrow_m$ *in-codom* $(\leq_{R1})$) *r2*
by (*intro 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 $((\leq_{L1}) \dashv (\leq_{R1}))$ *l1 r1* $((\leq_{L2}) \dashv (\leq_{R2}))$ *l2 r2*
and $((\leq_{L1}) \Rrightarrow_m (\leq_{L2}))$ *l1* $((\leq_{R2}) \Rrightarrow_m (\leq_{R1}))$ *r2*
and (*in-dom* $(\leq_{L1}) \Rrightarrow$ *in-codom* $(\leq_{R2}) \Rrightarrow (\longleftrightarrow)$)
(*rel-bimap l1 r2* $(\leq_{R1})$) (*rel-bimap l1 r2* $(\leq_{L2})$)
shows $((\leq_L) \dashv (\leq_R))$ *l r*
using *assms* **by** (*intro galois-connectionI*) *auto*

end

context *transport-comp-same*
begin

corollary *galois-connectionI*:
assumes $((\leq_{L1}) \dashv (\leq_{R1}))$ *l1 r1* $((\leq_{R1}) \dashv (\leq_{R2}))$ *l2 r2*
shows $((\leq_L) \dashv (\leq_R))$ *l r*
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*: $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1* $((\leq_{L2}) \equiv_G (\leq_{R2}))$ *l2 r2*
and *mono-L1-L2-l1*: $\bigwedge x\ y.\ x \leq_{L1} y \Longrightarrow l1\ x \leq_{R1} l1\ y \Longrightarrow l1\ x \leq_{L2} l1\ y$
and *mono-R2-R1-r2*: $\bigwedge x\ y.\ x \leq_{R2} y \Longrightarrow r2\ x \leq_{L2} r2\ y \Longrightarrow r2\ x \leq_{R1} r2\ y$
and $(in\text{-}dom\ (\leq_{L1}) \Rrightarrow in\text{-}codom\ (\leq_{R2}) \Rrightarrow (\longleftrightarrow))$
$(rel\text{-}bimap\ l1\ r2\ (\leq_{R1}))\ (rel\text{-}bimap\ l1\ r2\ (\leq_{L2}))$
and *mono-iff2*: $(in\text{-}dom\ (\leq_{R2}) \Rrightarrow in\text{-}codom\ (\leq_{L1}) \Rrightarrow (\longleftrightarrow))$
$(rel\text{-}bimap\ r2\ l1\ (\leq_{R1}))\ (rel\text{-}bimap\ r2\ l1\ (\leq_{L2}))$
shows $((\leq_L) \equiv_G (\leq_R))$ *l r*
proof −
from *galois mono-L1-L2-l1* **have** $(in\text{-}codom\ (\leq_{L1}) \Rrightarrow_m in\text{-}codom\ (\leq_{L2}))$ *l1*
by (*intro mono-wrt-predI*) *blast*
moreover from *galois mono-R2-R1-r2* **have** $(in\text{-}dom\ (\leq_{R2}) \Rrightarrow_m in\text{-}dom\ (\leq_{R1}))$
r2
by (*intro mono-wrt-predI*) *blast*
moreover from *mono-iff2* **have** $(in\text{-}dom\ (\leq_{R2}) \Rrightarrow in\text{-}codom\ (\leq_{L1}) \Rrightarrow (\longleftrightarrow))$
$(rel\text{-}bimap\ r2\ l1\ (\leq_{L2}))\ (rel\text{-}bimap\ r2\ l1\ (\leq_{R1}))$ **by** *blast*
ultimately show *?thesis* **using** *assms*
by (*intro galois-equivalenceI galois-connectionI flip.galois-propI*) *auto*
qed

lemma *galois-equivalenceI′*:
assumes $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1* $((\leq_{L2}) \equiv_G (\leq_{R2}))$ *l2 r2*
and $((\leq_{L1}) \Rrightarrow_m (\leq_{L2}))$ *l1* $((\leq_{R2}) \Rrightarrow_m (\leq_{R1}))$ *r2*
and $(in\text{-}dom\ (\leq_{L1}) \Rrightarrow in\text{-}codom\ (\leq_{R2}) \Rrightarrow (\longleftrightarrow))$
$(rel\text{-}bimap\ l1\ r2\ (\leq_{R1}))\ (rel\text{-}bimap\ l1\ r2\ (\leq_{L2}))$
and $(in\text{-}dom\ (\leq_{R2}) \Rrightarrow in\text{-}codom\ (\leq_{L1}) \Rrightarrow (\longleftrightarrow))$
$(rel\text{-}bimap\ r2\ l1\ (\leq_{R1}))\ (rel\text{-}bimap\ r2\ l1\ (\leq_{L2}))$
shows $((\leq_L) \equiv_G (\leq_R))$ *l r*
using *assms* **by** (*intro galois-equivalenceI*) *auto*

end

context *transport-comp-same*
begin

lemma *galois-equivalenceI*:
assumes $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1* $((\leq_{R1}) \equiv_G (\leq_{R2}))$ *l2 r2*
shows $((\leq_L) \equiv_G (\leq_R))$ *l r*
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*: $(\textit{in-codom}\ (\leq_{R2}) \Rrightarrow_m \textit{in-codom}\ (\leq_{R1}))$ *r2*
and *r2-L2-self-if-in-codom*: $\bigwedge z.\ \textit{in-codom}\ (\leq_{R2})\ z \Longrightarrow r2\ z \leq_{L2} r2\ z$
shows $({}_L\lessapprox) \leq (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$
proof (*rule le-relI*)
fix *x z* **assume** $x\ {}_L\lessapprox\ z$
then have $x \leq_{L1} r\ z$ *in-codom* $(\leq_R)\ z$ **by** *auto*
with $\langle x \leq_{L1} r\ z\rangle$ *in-codom-mono-r2* **have** $x\ {}_{L1}\lessapprox\ r2\ z$ **by** *fastforce*
moreover from $\langle \textit{in-codom}\ (\leq_{R2})\ z\rangle$ *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*: $((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))$ *r1*
and *trans-L1*: *transitive* $(\leq_{L1})$
and *R1-r2-if-in-codom*: $\bigwedge y\ z.\ \textit{in-codom}\ (\leq_{R2})\ z \Longrightarrow y \leq_{L2} r2\ z \Longrightarrow y \leq_{R1} r2\ z$
shows $(({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox)) \leq ({}_L\lessapprox)$
proof (*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 $x \leq_{L1} r1\ y\ y \leq_{L2} r2\ z$ *in-codom* $(\leq_R)\ z$ **by** *auto*
with *R1-r2-if-in-codom* **have** $y \leq_{R1} r2\ z$ **by** *blast*
with *mono-r1* **have** $r1\ y \leq_{L1} r\ z$ **by** *auto*
with $\langle x \leq_{L1} r1\ y\rangle$ $\langle \textit{in-codom}\ (\leq_R)\ z\rangle$ **show** $x\ {}_L\lessapprox\ z$ **using** *trans-L1* **by** *blast*

qed

corollary *left-Galois-eq-comp-left-GaloisI*:
assumes $((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))$ *r1*
and *transitive* $(\leq_{L1})$
and $\bigwedge z.$ *in-codom* $(\leq_{R2})$ $z \Longrightarrow r2\ z \leq_{L2} r2\ z$
and $\bigwedge y\ z.$ *in-codom* $(\leq_{R2})$ $z \Longrightarrow y \leq_{L2} r2\ z \Longrightarrow y \leq_{R1} r2\ z$
shows $({}_L\lessapprox) = (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$
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 $((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))$ *r1*
and *transitive* $(\leq_{L1})$
and $((\leq_{R2}) \Rrightarrow_m (\leq_{L2}))$ *r2*
and *reflexive-on* (*in-codom* $(\leq_{R2})$) $(\leq_{R2})$
and $\bigwedge y\ z.$ *in-codom* $(\leq_{R2})$ $z \Longrightarrow y \leq_{L2} r2\ z \Longrightarrow y \leq_{R1} r2\ z$
shows $({}_L\lessapprox) = (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$
using *assms* **by** (*intro left-Galois-eq-comp-left-GaloisI*) (*auto 5 0*)

corollary *left-Galois-eq-comp-left-GaloisI''*:
assumes $((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))$ *r1*
and *transitive* $(\leq_{L1})$
and $((\leq_{R2}) \Rrightarrow_m (\leq_{L2}))$ *r2*
and *reflexive-on* (*in-codom* $(\leq_{L2})$) $(\leq_{L2})$
and $\bigwedge y\ z.$ *in-codom* $(\leq_{R2})$ $z \Longrightarrow y \leq_{L2} r2\ z \Longrightarrow y \leq_{R1} r2\ z$
shows $({}_L\lessapprox) = (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$
using *assms* **by** (*intro left-Galois-eq-comp-left-GaloisI*) (*auto 0 6*)

end

context *transport-comp-same*
begin

lemma *left-Galois-eq-comp-left-GaloisI*:
assumes $((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))$ *r1*
and *transitive* $(\leq_{L1})$
and $((\leq_{R2}) \Rrightarrow_m (\leq_{R1}))$ *r2*
and *reflexive-on* (*in-codom* $(\leq_{R2})$) $(\leq_{R2})$
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 $((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))$ *r1*
and *transitive* $(\leq_{L1})$
and *reflexive-on* (*in-codom* $(\leq_{R1})$) $(\leq_{R1})$
and $((\leq_{R2}) \Rrightarrow_m (\leq_{R1}))$ *r2*

shows $(_L\lesssim) = ((_{L1}\lesssim) \circ\circ (_{L2}\lesssim))$
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*: $(P \Rightarrow_m P')$ *l1*
and *mono-r1*: $((\leq_{R1}) \Rightarrow_m (\leq_{L1}))$ *r1*
and *inflationary-unit1*: *inflationary-on* P $(\leq_{L1})$ η_1
and *trans-L1*: *transitive* $(\leq_{L1})$
and *inflationary-unit2*: *inflationary-on* P' $(\leq_{L2})$ η_2
and *L2-le-R1*: $\bigwedge x.\ P\ x \Longrightarrow l1\ x \leq_{L2} r2\ (l\ x) \Longrightarrow l1\ x \leq_{R1} r2\ (l\ x)$
shows *inflationary-on* P $(\leq_L)$ η
proof (*rule inflationary-onI*)
fix x **assume** $P\ x$
with *mono-l1* **have** $P'\ (l1\ x)$ **by** *blast*
with *inflationary-unit2* **have** $l1\ x \leq_{L2} r2\ (l\ x)$ **by** *auto*
with *L2-le-R1* ‹$P\ x$› **have** $l1\ x \leq_{R1} r2\ (l\ x)$ **by** *blast*
with *mono-r1* **have** $\eta_1\ x \leq_{L1} \eta\ x$ **by** *auto*
moreover from *inflationary-unit1* ‹$P\ x$› **have** $x \leq_{L1} \eta_1\ x$ **by** *auto*
ultimately show $x \leq_L \eta\ x$ **using** *trans-L1* **by** *blast*
qed

corollary *inflationary-on-in-field-unitI*:
assumes $((\leq_{L1}) \Rightarrow_m (\leq_{L2}))$ *l1*
and $((\leq_{R1}) \Rightarrow_m (\leq_{L1}))$ *r1*
and *inflationary-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$ η_1
and *transitive* $(\leq_{L1})$
and *inflationary-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$ η_2
and $\bigwedge x.$ *in-field* $(\leq_{L1})$ $x \Longrightarrow l1\ x \leq_{L2} r2\ (l\ x) \Longrightarrow l1\ x \leq_{R1} r2\ (l\ x)$
shows *inflationary-on* (*in-field* $(\leq_L)$) $(\leq_L)$ η
using *assms* **by** (*intro inflationary-on-unitI dep-mono-wrt-predI*) (*auto 5 0*)

Deflationary **context**
begin

interpretation *inv* :
transport-comp-agree $(\geq_{L1})$ $(\geq_{R1})$ *l1 r1* $(\geq_{L2})$ $(\geq_{R2})$ *l2 r2*
rewrites $\bigwedge R\ S.\ (R^{-1} \Rrightarrow_m S^{-1}) \equiv (R \Rrightarrow_m S)$
and $\bigwedge(P :: 'i \Rightarrow bool)\ (R :: 'j \Rightarrow 'i \Rightarrow bool).$
$(\textit{inflationary-on}\ P\ R^{-1} :: ('i \Rightarrow 'j) \Rightarrow bool) \equiv \textit{deflationary-on}\ P\ R$
and $\bigwedge(R :: 'i \Rightarrow 'i \Rightarrow bool).\ \textit{transitive}\ R^{-1} \equiv \textit{transitive}\ R$
and $\bigwedge R.\ \textit{in-field}\ R^{-1} \equiv \textit{in-field}\ R$
by (*simp-all add*: *mono-wrt-rel-eq-dep-mono-wrt-rel*)

lemma *deflationary-on-in-field-unitI*:
assumes $((\leq_{L1}) \Rrightarrow_m (\leq_{L2}))$ *l1*
and $((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))$ *r1*
and *deflationary-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$ η_1
and *transitive* $(\leq_{L1})$
and *deflationary-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$ η_2
and $\bigwedge x.$ *in-field* $(\leq_{L1})$ $x \Longrightarrow r2\ (l\ x) \leq_{L2} l1\ x \Longrightarrow r2\ (l\ x) \leq_{R1} l1\ x$
shows *deflationary-on* (*in-field* $(\leq_L)$) $(\leq_L)$ η
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 $((\leq_{L1}) \Rrightarrow_m (\leq_{L2}))$ *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 *rel-equivalence-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$ η_2
and $\bigwedge x.$ *in-field* $(\leq_{L1})$ $x \Longrightarrow l1\ x \leq_{L2} r2\ (l\ x) \Longrightarrow l1\ x \leq_{R1} r2\ (l\ x)$
and $\bigwedge x.$ *in-field* $(\leq_{L1})$ $x \Longrightarrow r2\ (l\ x) \leq_{L2} l1\ x \Longrightarrow r2\ (l\ x) \leq_{R1} l1\ x$
shows *rel-equivalence-on* (*in-field* $(\leq_L)$) $(\leq_L)$ η
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 $((\leq_{L1}) \equiv_o (\leq_{R1}))$ *l1 r1*
and *transitive* $(\leq_{L1})$
and $((\leq_{L2}) \equiv_o (\leq_{R2}))$ *l2 r2*
and *transitive* $(\leq_{R2})$
and $\bigwedge x\ y.\ x \leq_{L1} y \Longrightarrow l1\ x \leq_{R1} l1\ y \Longrightarrow l1\ x \leq_{L2} l1\ y$
and $\bigwedge x\ y.\ x \leq_{R2} y \Longrightarrow r2\ x \leq_{L2} r2\ y \Longrightarrow r2\ x \leq_{R1} r2\ y$
and $\bigwedge x.$ *in-field* $(\leq_{L1})\ x \Longrightarrow l1\ x \leq_{L2} r2\ (l\ x) \Longrightarrow l1\ x \leq_{R1} r2\ (l\ x)$
and $\bigwedge x.$ *in-field* $(\leq_{L1})\ x \Longrightarrow r2\ (l\ x) \leq_{L2} l1\ x \Longrightarrow r2\ (l\ x) \leq_{R1} l1\ x$
and $\bigwedge x.$ *in-field* $(\leq_{R2})\ x \Longrightarrow r2\ x \leq_{R1} l1\ (r\ x) \Longrightarrow r2\ x \leq_{L2} l1\ (r\ x)$
and $\bigwedge x.$ *in-field* $(\leq_{R2})\ x \Longrightarrow l1\ (r\ x) \leq_{R1} r2\ x \Longrightarrow l1\ (r\ x) \leq_{L2} r2\ x$
shows $((\leq_L) \equiv_o (\leq_R))\ l\ r$
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 mono-wrt-relI)
(*auto elim*!: *g1.order-equivalenceE g2.order-equivalenceE*)

end

context *transport-comp-same*
begin

lemma *order-equivalenceI*:
assumes $((\leq_{L1}) \equiv_o (\leq_{R1}))$ *l1 r1*
and *transitive* $(\leq_{L1})$
and $((\leq_{R1}) \equiv_o (\leq_{R2}))$ *l2 r2*
and *transitive* $(\leq_{R2})$
shows $((\leq_L) \equiv_o (\leq_R))\ l\ r$
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
Equivalence-Relations
Transport-Base
begin

locale *transport-comp* =
t1 : *transport L1 R1 l1 r1* + *t2* : *transport L2 R2 l2 r2*
for *L1* :: $'a \Rightarrow 'a \Rightarrow bool$
and *R1* :: $'b \Rightarrow 'b \Rightarrow bool$
and *l1* :: $'a \Rightarrow 'b$
and *r1* :: $'b \Rightarrow 'a$
and *L2* :: $'b \Rightarrow 'b \Rightarrow bool$
and *R2* :: $'c \Rightarrow 'c \Rightarrow bool$
and *l2* :: $'b \Rightarrow 'c$
and *r2* :: $'c \Rightarrow 'b$
begin

This locale collects results about the composition of transportable components under some generic compatibility conditions on *R1* and *L2* (cf. below). The composition is rather subtle, but in return can cover quite general cases.

Explanations and intuition about the construction can be found in [2].

notation *L1* (**infix** $\leq_{L1}$ *50*)
notation *R1* (**infix** $\leq_{R1}$ *50*)
notation *L2* (**infix** $\leq_{L2}$ *50*)
notation *R2* (**infix** $\leq_{R2}$ *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 *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** $_{R2}\gtrsim$ *50*)
notation *t2.Galois-right* (**infix** $\lessapprox_{R2}$ *50*)

notation *t1.left-ge-Galois* (**infix** $_{L1}\gtrsim$ *50*)
notation *t1.Galois-left* (**infix** $\lessapprox_{L1}$ *50*)
notation *t2.left-ge-Galois* (**infix** $_{L2}\gtrsim$ *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** $\leq_L$ *50*)
notation R (**infix** $\leq_R$ *50*)

lemma *left-relI* [*intro*]:
assumes $x\ _{L1}\overset{\leq}{\approx}\ y$
and $y \leq_{L2} y'$
and $y'\ _{R1}\overset{\leq}{\approx}\ x'$
shows $x \leq_L x'$
unfolding *left-rel-eq-comp* **using** *assms* **by** *blast*

lemma *left-relE* [*elim*]:
assumes $x \leq_L x'$
obtains $y\ y'$ **where** $x\ _{L1}\overset{\leq}{\approx}\ y\ y \leq_{L2} y'\ y'\ _{R1}\overset{\leq}{\approx}\ x'$
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* $(\geq_{L1})$ $(\geq_{R1})$ *l1 r1* $(\geq_{L2})$ $(\geq_{R2})$ *l2 r2* .

lemma *ge-left-rel-eq-left-rel-inv-if-galois-prop* [*simp*]:
assumes $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1* $((\leq_{R1}) \trianglelefteq (\leq_{L1}))$ *r1 l1*
shows $(\geq_L)$ = *transport-comp.L* $(\geq_{L1})$ $(\geq_{R1})$ *l1 r1* $(\geq_{L2})$
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 $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1* $((\leq_{R1}) \trianglelefteq (\leq_{L1}))$ *r1 l1*
shows (*transport-comp.L* $(\geq_{L1})$ $(\geq_{R1})$ *l1 r1* $(\geq_{L2})$) *x x'* $\longleftrightarrow$ $x' \leq_L x$
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 $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*
and $((\leq_{R1})\ {}_h\trianglelefteq (\leq_{L1}))$ *r1 l1*
and *trans-L1*: *transitive* $(\leq_{L1})$
and *mono-l1*: $((\leq_L) \Rrightarrow_m ((\leq_{R1}) \circ\circ (\leq_{R1})))$ *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 $\langle((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*$\rangle$ $\langle x \leq_L x'\rangle$ **have** $x \leq_{L1}$ *r1 y* **by** *blast*
moreover from $\langle((\leq_{R1})\ {}_h\trianglelefteq (\leq_{L1}))$ *r1 l1*$\rangle$ $\langle y \leq_{R1}$ *l1 x'*$\rangle$ $\langle x \leq_L x'\rangle$
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 $\langle((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))\ l1\ r1\rangle$ $\langle x \leq_{L1} x'\rangle$ **have** $x\ {}_{L1}\lessapprox\ y$ **by** *blast*
moreover note $\langle y \leq_{L2} y'\rangle$
moreover from $\langle y' \leq_{R1} l1\ x'\rangle$ $\langle x \leq_{L1} x'\rangle$ **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 $((\leq_{R1})\ {}_h\trianglerighteq\ (\leq_{L1}))$ *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*:
$\exists (L1 :: bool \Rightarrow \text{-})\ (R1 :: bool \Rightarrow \text{-})\ l1\ r1$
$(L2 :: bool \Rightarrow \text{-})\ (R2 :: bool \Rightarrow \text{-})\ l2\ r2.$
$(L1 \equiv_o R1)\ l1\ r1$
$\wedge$ *equivalence-rel L1* $\wedge$ *equivalence-rel R1*
$\wedge\ (L2 \equiv_o R2)\ l2\ r2$
$\wedge$ *equivalence-rel L2* $\wedge$ *equivalence-rel R2*
$\wedge$ *transport-comp.L L1 R1 l1 r1 L2* $\neq$ *L1*
proof (*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 $x \leq_L x'$
and *l1-R1-if-L1-r1*: $\bigwedge y.$ *in-codom* $(\leq_{R1})$ $y \Longrightarrow x \leq_{L1} r1\ y \Longrightarrow l1\ x \leq_{R1} y$
and *t-R1-if-l1-R1*: $\bigwedge y.$ $l1\ x \leq_{R1} y \Longrightarrow t\ y \leq_{R1} y$
and *R2-l2-if-t-L2-if-l1-R1*:
$\bigwedge y\ y'.$ $l1\ x \leq_{R1} y \Longrightarrow t\ y \leq_{L2} y' \Longrightarrow z \leq_{R2} l2\ y'$
and *R1-b-if-R1-l1-if-R1-l1*:
$\bigwedge y\ y'.$ $y \leq_{R1} l1\ x' \Longrightarrow y' \leq_{R1} l1\ x' \Longrightarrow y' \leq_{R1} b\ y$
and *b-L2-r2-if-in-codom-L2-b-if-R1-l1*:
$\bigwedge y.$ $y \leq_{R1} l1\ x' \Longrightarrow$ *in-codom* $(\leq_{L2})$ $(b\ y) \Longrightarrow b\ y \leq_{L2} r2\ z'$
and *in-codom-R2-if-in-codom-L2-b-if-R1-l1*:
$\bigwedge y.$ $y \leq_{R1} l1\ x' \Longrightarrow$ *in-codom* $(\leq_{L2})$ $(b\ y) \Longrightarrow$ *in-codom* $(\leq_{R2})$ z'
and *rel-comp-le*: $(\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1}) \leq (\leq_{L2}) \circ\circ (\leq_{R1})$
and *in-codom-rel-comp-le*: *in-codom* $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-codom* $(\leq_{L2})$
shows $z \leq_R z'$
proof −
from ‹$x \leq_L x'$› **obtain** $yl\ yl'$ **where** $l1\ x \leq_{R1} yl\ yl \leq_{L2} yl'\ yl' \leq_{R1} l1\ x'$
using *l1-R1-if-L1-r1* **by** *blast*
moreover then have $t\ yl \leq_{R1} yl$ **by** (*intro t-R1-if-l1-R1*)
ultimately have $((\leq_{L2}) \circ\circ (\leq_{R1}))$ $(t\ yl)$ $(l1\ x')$ **using** *rel-comp-le* **by** *blast*
then obtain y **where** $t\ yl \leq_{L2} y\ y \leq_{R1} l1\ x'$ **by** *blast*
show $z \leq_R z'$
proof (*rule flip.left-relI*)
from ‹$t\ yl \leq_{L2} y$› ‹$l1\ x \leq_{R1} yl$› **show** $z\ {}_{R2}\lessapprox y$
by (*auto intro*: *R2-l2-if-t-L2-if-l1-R1*)
from ‹$yl' \leq_{R1} l1\ x'$› ‹$y \leq_{R1} l1\ x'$› **show** $y \leq_{R1} b\ yl'$
by (*rule R1-b-if-R1-l1-if-R1-l1*)
show $b\ yl'\ {}_{L2}\lessapprox z'$
proof (*rule t2.left-GaloisI*)
from ‹$yl' \leq_{R1} l1\ x'$› **have** $yl' \leq_{R1} b\ yl'$
by (*intro R1-b-if-R1-l1-if-R1-l1*)
with ‹$l1\ x \leq_{R1} yl$› ‹$yl \leq_{L2} yl'$› *in-codom-rel-comp-le*
have *in-codom* $(\leq_{L2})$ $(b\ yl')$ **by** *blast*
with ‹$yl' \leq_{R1} l1\ x'$› **show** $b\ yl' \leq_{L2} r2\ z'$ *in-codom* $(\leq_{R2})$ 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 $x \leq_L x'$
and *l1-R1-if-L1-r1*: $\bigwedge y.$ *in-codom* $(\leq_{R1})$ $y \Longrightarrow x \leq_{L1} r1\ y \Longrightarrow l1\ x \leq_{R1} y$
and *R1-b-if-R1-l1*: $\bigwedge y.\ y \leq_{R1} l1\ x' \Longrightarrow y \leq_{R1} b\ y$
and *L2-r2-if-L2-b-if-R1-l1*:
$\bigwedge y\ y'.\ y \leq_{R1} l1\ x' \Longrightarrow y' \leq_{L2} b\ y \Longrightarrow y' \leq_{L2} r2\ z'$
and *in-codom-R2-if-L2-b-if-R1-l1*:
$\bigwedge y\ y'.\ y \leq_{R1} l1\ x' \Longrightarrow y' \leq_{L2} b\ y \Longrightarrow$ *in-codom* $(\leq_{R2})\ z'$
and *t-R1-if-R1-l1-if-l1-R1*:
$\bigwedge y\ y'\ y''.\ l1\ x \leq_{R1} y \Longrightarrow l1\ x \leq_{R1} y' \Longrightarrow t\ y \leq_{R1} y'$
and *R2-l2-t-if-in-dom-L2-t-if-l1-R1*:
$\bigwedge y\ y'.\ l1\ x \leq_{R1} y \Longrightarrow$ *in-dom* $(\leq_{L2})\ (t\ y) \Longrightarrow z \leq_{R2} l2\ (t\ y)$
and *in-codom-L2-t-if-in-dom-L2-t-if-l1-R1*:
$\bigwedge y\ y'.\ l1\ x \leq_{R1} y \Longrightarrow$ *in-dom* $(\leq_{L2})\ (t\ y) \Longrightarrow$ *in-codom* $(\leq_{L2})\ (t\ y)$
and *rel-comp-le*: $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{R1}) \circ\circ (\leq_{L2}))$
and *in-dom-rel-comp-le*: *in-dom* $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-dom* $(\leq_{L2})$
shows $z \leq_R z'$
proof −
from ‹$x \leq_L x'$› **obtain** *yl yl'* **where** $l1\ x \leq_{R1} yl\ yl \leq_{L2} yl'\ yl' \leq_{R1} l1\ x'$
using *l1-R1-if-L1-r1* **by** *blast*
moreover then have $yl' \leq_{R1} b\ yl'$ **by** (*intro R1-b-if-R1-l1*)
ultimately have $((\leq_{R1}) \circ\circ (\leq_{L2}))\ (l1\ x)\ (b\ yl')$ **using** *rel-comp-le* **by** *blast*
then obtain *y* **where** $l1\ x \leq_{R1} y\ y \leq_{L2} b\ yl'$ **by** *blast*
show $z \leq_R z'$
proof (*rule flip.left-relI*)
from ‹$yl' \leq_{R1} l1\ x'$› ‹$y \leq_{L2} b\ yl'$›
have *in-codom* $(\leq_{R2})\ z'\ y \leq_{L2} r2\ z'$
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 ‹$l1\ x \leq_{R1} yl$› ‹$l1\ x \leq_{R1} y$› **show** $t\ yl \leq_{R1} y$ **by** (*rule t-R1-if-R1-l1-if-l1-R1*)
show $z\ {}_{R2}\!\lessapprox t\ yl$
proof (*rule flip.t1.left-GaloisI*)
from ‹$l1\ x \leq_{R1} yl$› **have** $t\ yl \leq_{R1} yl$ **by** (*intro t-R1-if-R1-l1-if-l1-R1*)
with ‹$yl \leq_{L2} yl'$› ‹$yl' \leq_{R1} l1\ x'$› *in-dom-rel-comp-le* **have** *in-dom* $(\leq_{L2})$ $(t\ yl)$
by *blast*
with ‹$l1\ x \leq_{R1} yl$›
show $z \leq_{R2} l2\ (t\ yl)$ *in-codom* $(\leq_{L2})\ (t\ yl)$ **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 $((\leq_{L1})\ {}_h\trianglelefteq\ (\leq_{R1}))$ *l1 r1*
and *in-dom* $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-dom* $(\leq_{L2})$
shows $($*in-dom* $(\leq_L) \Rrightarrow_m$ *in-dom* $(\leq_{L2}))$ *l1*
using *assms* **by** (*intro mono-wrt-predI*) *blast*

lemma *mono-in-codom-left-rel-left1-if-in-codom-rel-comp-le*:
assumes $((\leq_{L1})\ {}_h\trianglelefteq\ (\leq_{R1}))$ *l1 r1*
and *in-codom* $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-codom* $(\leq_{L2})$
shows $($*in-codom* $(\leq_L) \Rrightarrow_m$ *in-codom* $(\leq_{L2}))$ *l1*
using *assms* **by** (*intro mono-wrt-predI*) *blast*

Simplification of Compatibility Conditions

Most results will depend on certain compatibility conditions between $(\leq_{R1})$ and $(\leq_{L2})$. We next derive some sufficient assumptions for these conditions.

end

lemma *rel-comp-comp-le-rel-comp-if-rel-comp-comp-if-in-dom-leI*:
assumes *trans-R*: *transitive R*
and *refl-S*: *reflexive-on P S*
and *in-dom-le*: *in-dom* $(R \circ\circ S \circ\circ R) \leq P$
and *rel-comp-le*: $(S \circ\circ R \circ\circ S) \leq (S \circ\circ R)$
shows $(R \circ\circ S \circ\circ R) \leq (S \circ\circ R)$
proof (*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* $(R \circ\circ S \circ\circ R) \leq P$
and *rel-comp-le*: $(S \circ\circ R \circ\circ S) \leq (R \circ\circ S)$
shows $(R \circ\circ S \circ\circ R) \leq (R \circ\circ S)$
proof (*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
thm *mono-rel-comp*
lemma *rel-comp-comp-le-rel-comp-if-rel-comp-le-if-transitive*:
assumes *trans-R*: *transitive R*
and *R-S-le*: $(R \circ\circ S) \leq (S \circ\circ R)$

shows $(R \circ\circ S \circ\circ R) \leq (S \circ\circ R)$
proof −
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 *mono-rel-comp R-S-le* **by** *blast*
also have ... $= S \circ\circ (R \circ\circ R)$ **by** (*simp flip*: *rel-comp-assoc*)
also have ... $\leq (S \circ\circ R)$ **using** *mono-rel-comp R-R-le* **by** *blast*
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 *mono-rel-comp S-R-le* **by** (*auto simp flip*: *rel-comp-assoc*)
also have ... $\leq (R \circ\circ S)$ **using** *mono-rel-comp 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* $\equiv$
$(\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1}) \leq (\leq_{R1}) \circ\circ (\leq_{L2})$

$\wedge$ *in-dom* $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-dom* $(\leq_{L2})$
$\wedge$ $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq ((\leq_{L2}) \circ\circ (\leq_{R1}))$
$\wedge$ *in-dom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-dom* $(\leq_{R1})$

lemma *middle-compatible-domI* [*intro*]:
assumes $(\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1}) \leq (\leq_{R1}) \circ\circ (\leq_{L2})$
and *in-dom* $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-dom* $(\leq_{L2})$
and $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq ((\leq_{L2}) \circ\circ (\leq_{R1}))$
and *in-dom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-dom* $(\leq_{R1})$
shows *middle-compatible-dom*
unfolding *middle-compatible-dom-def* **using** *assms* **by** *blast*

lemma *middle-compatible-domE* [*elim*]:
assumes *middle-compatible-dom*
obtains $(\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1}) \leq (\leq_{R1}) \circ\circ (\leq_{L2})$
and *in-dom* $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-dom* $(\leq_{L2})$
and $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq ((\leq_{L2}) \circ\circ (\leq_{R1}))$
and *in-dom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-dom* $(\leq_{R1})$
using *assms* **unfolding** *middle-compatible-dom-def* **by** *blast*

definition *middle-compatible-codom* $\equiv$
$((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{L2}) \circ\circ (\leq_{R1}))$
$\wedge$ *in-codom* $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-codom* $(\leq_{L2})$
$\wedge$ $(\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2}) \leq (\leq_{R1}) \circ\circ (\leq_{L2})$
$\wedge$ *in-codom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-codom* $(\leq_{R1})$

lemma *middle-compatible-codomI* [*intro*]:
assumes $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{L2}) \circ\circ (\leq_{R1}))$
and *in-codom* $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-codom* $(\leq_{L2})$
and $(\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2}) \leq (\leq_{R1}) \circ\circ (\leq_{L2})$
and *in-codom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-codom* $(\leq_{R1})$
shows *middle-compatible-codom*
unfolding *middle-compatible-codom-def* **using** *assms* **by** *blast*

lemma *middle-compatible-codomE* [*elim*]:
assumes *middle-compatible-codom*
obtains $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{L2}) \circ\circ (\leq_{R1}))$
and *in-codom* $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-codom* $(\leq_{L2})$
and $(\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2}) \leq (\leq_{R1}) \circ\circ (\leq_{L2})$
and *in-codom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-codom* $(\leq_{R1})$
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* $(\leq_{R1})$) $(\leq_{R1})$

and *preorder-on* (*in-field* ($\leq_{L2}$)) ($\leq_{L2}$)
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* ($\leq_{R1}$)) ($\leq_{R1}$)
and *preorder-on* (*in-field* ($\leq_{L2}$)) ($\leq_{L2}$)
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* ($\leq_{R1}$)) ($\leq_{R1}$)
and *preorder-on* (*in-field* ($\leq_{L2}$)) ($\leq_{L2}$)
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* ($\leq_{R1}$)
and (($\leq_{R1}$) $\circ\circ$ ($\leq_{L2}$)) = (($\leq_{L2}$) $\circ\circ$ ($\leq_{R1}$))
shows (($\leq_{R1}$) $\circ\circ$ ($\leq_{L2}$) $\circ\circ$ ($\leq_{R1}$)) $\leq$ (($\leq_{L2}$) $\circ\circ$ ($\leq_{R1}$))
and (($\leq_{R1}$) $\circ\circ$ ($\leq_{L2}$) $\circ\circ$ ($\leq_{R1}$)) $\leq$ (($\leq_{R1}$) $\circ\circ$ ($\leq_{L2}$))
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 (($\leq_{L2}$) $\circ\circ$ ($\leq_{R1}$)) = (($\leq_{R1}$) $\circ\circ$ ($\leq_{L2}$)) $\equiv$ (($\leq_{R1}$) $\circ\circ$ ($\leq_{L2}$)) = (($\leq_{L2}$) $\circ\circ$ ($\leq_{R1}$))
by (*simp only*: *eq-commute*)

lemma *middle-compatible-codom-if-rel-comp-eq-if-transitive*:

assumes *transitive* $(\leq_{R1})$ *transitive* $(\leq_{L2})$
and $((\leq_{R1}) \circ\circ (\leq_{L2})) = ((\leq_{L2}) \circ\circ (\leq_{R1}))$
shows *middle-compatible-codom*
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* $(\leq_{R1})$) $(\leq_{R1})$ *transitive* $(\leq_{L2})$
and $(\leq_{R1}) \leq (\leq_{L2}) \sqcup (=)$
and *in-field* $(\leq_{L2}) \leq$ *in-field* $(\leq_{R1})$
shows *middle-compatible-codom*
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 $(\leq_{R1}) \leq (=)$
and *transitive* $(\leq_{L2})$
and *in-field* $(\leq_{L2}) \leq$ *in-field* $(\leq_{R1})$
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* $= \varepsilon_1$ **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* $(\leq_{L2})$) $(\leq_{L2})$
and $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{R1}) \circ\circ (\leq_{L2}))$
and *in-dom* $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-dom* $(\leq_{L2})$
and *mono-in-codom-r2*: (*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
assume *in-dom* $(\leq_{L2})$ $(l1\ x)$
with *refl-L2* **have** $l1\ x \leq_{L2} l1\ x$ **by** *blast*
with ‹$((\leq_{L2}) \Rightarrow_m (\leq_{R2}))\ l2$› **show** *in-codom* $(\leq_{L2})$ $(l1\ x)\ l\ x \leq_{R2} l2\ (l1\ x)$
by *auto*
qed (*insert assms*, *auto*)
qed

lemma *half-galois-prop-right-left-rightI*:
assumes $((\leq_{R1}) \Rightarrow_m (\leq_{L1}))\ r1$
and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))\ l1\ r1$
and *inflationary-counit1*: *inflationary-on* $(\textit{in-codom}\ (\leq_{R1}))\ (\leq_{R1})\ \varepsilon_1$
and $((\leq_{R2})\ {}_h\trianglelefteq\ (\leq_{L2}))\ r2\ l2$
and *inflationary-unit2*: *inflationary-on* $(\textit{in-dom}\ (\leq_{L2}))\ (\leq_{L2})\ \eta_2$
and *trans-L2*: *transitive* $(\leq_{L2})$
and *mono-in-dom-l1*: $(\textit{in-dom}\ (\leq_L) \Rightarrow_m \textit{in-dom}\ (\leq_{L2}))\ l1$
and $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq ((\leq_{R1}) \circ\circ (\leq_{L2}))$
and $\textit{in-codom}\ ((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq \textit{in-codom}\ (\leq_{R1})$
shows $((\leq_L) \trianglelefteq_h (\leq_R))\ l\ r$
proof (*rule half-galois-prop-rightI*)
fix $x\ z$ **assume** $x \lessapprox_R z$
then show $x \leq_L r\ z$
proof (*intro flip.right-rel-if-left-relI*)
fix y **assume** $r2\ (l\ x) \leq_{L2} y$
moreover have $l1\ x \leq_{L2} r2\ (l\ x)$
proof −
from *mono-in-dom-l1* ‹$x \lessapprox_R z$› **have** $\textit{in-dom}\ (\leq_{L2})\ (l1\ x)$ **by** *blast*
with *inflationary-unit2* **show** $l1\ x \leq_{L2} r2\ (l\ x)$ **by** *auto*
qed
ultimately show $l1\ x \leq_{L2} y$ **using** *trans-L2* **by** *blast*
fix y **assume** $l1\ x \leq_{R1} y$
with ‹$((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))\ l1\ r1$› ‹$x \lessapprox_R z$› **show** $x \leq_{L1} r1\ y$ **by** *blast*
next
assume $\textit{in-codom}\ (\leq_{R1})\ (r2\ z)$
with *inflationary-counit1* **show** $r2\ z \leq_{R1} l1\ (r\ z)$ **by** *auto*
from ‹$((\leq_{R1}) \Rightarrow_m (\leq_{L1}))\ r1$› ‹$\textit{in-codom}\ (\leq_{R1})\ (r2\ z)$› **show** $\textit{in-codom}\ (\leq_{L1})\ (r\ z)$
by (*auto intro*: *in-codom-if-rel-if-dep-mono-wrt-rel*
simp: *mono-wrt-rel-eq-dep-mono-wrt-rel*)
qed (*insert assms*, *auto elim*: *galois-rel.left-GaloisE*)
qed

lemma *half-galois-prop-right-left-rightI'*:
assumes $((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))$ *r1*
and *inflationary-unit1*: *inflationary-on* $(\textit{in-dom}\ (\leq_{L1}))\ (\leq_{L1})\ \eta_1$
and *inflationary-counit1*: $\bigwedge y\ z.\ y \leq_{R1} r2\ z \Longrightarrow y \leq_{R1} l1\ (r\ z)$
and *in-dom* $(\leq_{R1}) \leq$ *in-codom* $(\leq_{R1})$
and $((\leq_{R2})\ {}_h\trianglelefteq\ (\leq_{L2}))$ *r2 l2*
and *inflationary-unit2*: *inflationary-on* $(\textit{in-dom}\ (\leq_{L2}))\ (\leq_{L2})\ \eta_2$
and *trans-L2*: *transitive* $(\leq_{L2})$
and *mono-in-dom-l1*: $(\textit{in-dom}\ (\leq_L) \Rrightarrow_m \textit{in-dom}\ (\leq_{L2}))\ l1$
and $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq ((\leq_{L2}) \circ\circ (\leq_{R1}))$
and *in-dom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq \textit{in-dom}\ (\leq_{R1})$
shows $((\leq_L) \trianglelefteq_h (\leq_R))\ l\ r$
proof (*rule half-galois-prop-rightI*)
fix $x\ z$ **assume** $x \lessapprox_R z$
then show $x \leq_L r\ z$
proof (*intro flip.right-rel-if-left-relI'*)
from $\langle x \lessapprox_R z\rangle$ *inflationary-unit1* **show** $x \leq_{L1} r1\ (l1\ x)$
by (*fastforce elim*: *galois-rel.left-GaloisE*)
fix y **assume** $y \leq_{R1} r2\ z$
with *inflationary-counit1* **show** $y \leq_{R1} l1\ (r\ z)$ **by** *auto*
next
fix y
from *mono-in-dom-l1* $\langle x \lessapprox_R z\rangle$ **have** *in-dom* $(\leq_{L2})\ (l1\ x)$ **by** *blast*
with *inflationary-unit2* **have** $l1\ x \leq_{L2} r2\ (l\ x)$ **by** *auto*
moreover assume $r2\ (l\ x) \leq_{L2} y$
ultimately show $l1\ x \leq_{L2} y$ **using** *trans-L2* **by** *blast*
qed (*insert assms*, *auto elim*: *galois-rel.left-GaloisE*)
qed

lemma *galois-prop-left-rightI*:
assumes $((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))$ *r1*
and $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
and *rel-equivalence-on* $(\textit{in-codom}\ (\leq_{R1}))\ (\leq_{R1})\ \varepsilon_1$
and *transitive* $(\leq_{R1})$
and $((\leq_{L2}) \Rrightarrow_m (\leq_{R2}))\ l2$
and $((\leq_{R2})\ {}_h\trianglelefteq\ (\leq_{L2}))$ *r2 l2*
and *inflationary-on* $(\textit{in-dom}\ (\leq_{L2}))\ (\leq_{L2})\ \eta_2$
and *preorder-on* $(\textit{in-field}\ (\leq_{L2}))\ (\leq_{L2})$
and *middle-compatible-codom*
shows $((\leq_L) \trianglelefteq (\leq_R))\ l\ r$
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 $((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))$ *r1*
and $((\leq_{L1}) \ {}_h\trianglelefteq (\leq_{R1}))$ *l1 r1*
and *inflationary-on* (*in-dom* $(\leq_{L1})$) $(\leq_{L1})$ η_1
and *rel-equiv-counit1*: *rel-equivalence-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$ ε_1
and *trans-R1*: *transitive* $(\leq_{R1})$
and $((\leq_{L2}) \Rrightarrow_m (\leq_{R2}))$ *l2*
and $((\leq_{R2}) \ {}_h\trianglelefteq (\leq_{L2}))$ *r2 l2*
and *inflationary-on* (*in-dom* $(\leq_{L2})$) $(\leq_{L2})$ η_2
and *preorder-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$
and *middle-compatible-dom*
shows $((\leq_L) \trianglelefteq (\leq_R))$ *l r*
proof (*rule galois-propI*)
show $((\leq_L) \ {}_h\trianglelefteq (\leq_R))$ *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 $y \leq_{R1} l1\ (r1\ (r2\ z))$ **if** $y \leq_{R1} r2\ z$ **for** *y z*
proof −
note ‹$y \leq_{R1} r2\ z$›
moreover with *rel-equiv-counit1* **have** $r2\ z \leq_{R1} \varepsilon_1\ (r2\ z)$ **by** *blast*
ultimately show *?thesis* **using** *trans-R1* **by** *auto*
qed
moreover have *in-dom* $(\leq_{R1}) \leq$ *in-codom* $(\leq_{R1})$
proof −
from *rel-equiv-counit1 trans-R1* **have** *reflexive-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
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 $((\leq_L) \trianglelefteq_h (\leq_R))$ *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 $((\leq_{L1}) \ {}_h\trianglelefteq \ (\leq_{R1}))$ *l1 r1*
  and $((\leq_{L2}) \Rrightarrow_m (\leq_{R2}))$ *l2*
  and *inflationary-unit2*: *inflationary-on* (*in-codom* $(\leq_{L2})$) $(\leq_{L2})$ η_2
  and $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{L2}) \circ\circ (\leq_{R1}))$
  and *in-codom* $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-codom* $(\leq_{L2})$
  shows $((\leq_L) \Rrightarrow_m (\leq_R))$ *l*
proof (*rule mono-wrt-relI*)
  fix *x x'* **assume** $x \leq_L x'$
  then show $l\ x \leq_R l\ x'$
  proof (*rule right-rel-if-left-relI*)
    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*
  next
    assume *in-codom* $(\leq_{L2})$ $(l1\ x')$
    with *inflationary-unit2* **show** $l1\ x' \leq_{L2} r2\ (l\ x')$ **by** *auto*
    from ‹*in-codom* $(\leq_{L2})$ $(l1\ x')$› ‹$((\leq_{L2}) \Rrightarrow_m (\leq_{R2}))$ *l2*›
      show *in-codom* $(\leq_{R2})$ $(l\ x')$ **by** *fastforce*
  qed (*insert assms*, *auto*)
qed

lemma *mono-wrt-rel-leftI'*:
  assumes $((\leq_{L1}) \ {}_h\trianglelefteq \ (\leq_{R1}))$ *l1 r1*
  and $((\leq_{L2}) \Rrightarrow_m (\leq_{R2}))$ *l2*
  and $((\leq_{L2}) \trianglelefteq_h (\leq_{R2}))$ *l2 r2*
  and *refl-L2*: *reflexive-on* (*in-dom* $(\leq_{L2})$) $(\leq_{L2})$
  and $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{R1}) \circ\circ (\leq_{L2}))$
  and *in-dom* $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-dom* $(\leq_{L2})$
  shows $((\leq_L) \Rrightarrow_m (\leq_R))$ *l*
proof (*rule mono-wrt-relI*)
  fix *x x'* **assume** $x \leq_L x'$
  then show $l\ x \leq_R l\ x'$
  proof (*rule right-rel-if-left-relI'*)
    fix y' **assume** $y' \leq_{L2} l1\ x'$
    moreover with ‹$((\leq_{L2}) \Rrightarrow_m (\leq_{R2}))$ *l2*› **have** $l2\ y' \leq_{R2} l\ x'$ **by** *auto*
    ultimately show *in-codom* $(\leq_{R2})$ $(l\ x')$ $y' \leq_{L2} r2\ (l\ x')$
      using ‹$((\leq_{L2}) \trianglelefteq_h (\leq_{R2}))$ *l2 r2*› **by** *auto*
  next
    assume *in-dom* $(\leq_{L2})$ $(l1\ x)$
    with *refl-L2* ‹$((\leq_{L2}) \Rrightarrow_m (\leq_{R2}))$ *l2*› **show** $l\ x \leq_{R2} l2\ (l1\ x)$ **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* = ε_1 **and** *flip.t1.counit* ≡ η_2
by (*simp-all only*: *t1.flip-unit-eq-counit t2.flip-counit-eq-unit*)

lemma *galois-connection-left-rightI*:
assumes (($\leq_{R1}$) $\Rrightarrow_m$ ($\leq_{L1}$)) *r1*
and (($\leq_{L1}$) $\trianglelefteq$ ($\leq_{R1}$)) *l1 r1*
and *rel-equivalence-on* (*in-codom* ($\leq_{R1}$)) ($\leq_{R1}$) ε_1
and *transitive* ($\leq_{R1}$)
and (($\leq_{L2}$) $\Rrightarrow_m$ ($\leq_{R2}$)) *l2*
and (($\leq_{R2}$) $_h\trianglelefteq$ ($\leq_{L2}$)) *r2 l2*
and *inflationary-on* (*in-field* ($\leq_{L2}$)) ($\leq_{L2}$) η_2
and *preorder-on* (*in-field* ($\leq_{L2}$)) ($\leq_{L2}$)
and *middle-compatible-codom*
shows (($\leq_L$) ⊣ ($\leq_R$)) *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 (($\leq_{R1}$) $\Rrightarrow_m$ ($\leq_{L1}$)) *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 *inflationary-on* (*in-dom* ($\leq_{L2}$)) ($\leq_{L2}$) η_2
and *preorder-on* (*in-field* ($\leq_{L2}$)) ($\leq_{L2}$)
and *middle-compatible-dom*
shows (($\leq_L$) ⊣ ($\leq_R$)) *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: *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* $= \varepsilon_1$ **and** *flip.t1.counit* $\equiv \eta_2$ **and** *flip.t1.unit* $\equiv \varepsilon_2$
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*: $((\leq_{R1}) \Rightarrow_m (\leq_{L1}))$ *r1*
and *galois-prop1*: $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
and *preorder-R1*: *preorder-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and *rel-comp-le*: $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{R1}) \circ\circ (\leq_{L2}))$
and *mono-in-codom-r2*: (*in-codom* $(\leq_R) \Rightarrow_m$ *in-codom* $(\leq_{R1})$) *r2*
shows $({}_L\lessapprox) \leq (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$
proof (*rule le-relI*)
fix *x z* **assume** *x* ${}_L\lessapprox$ *z*
then have *in-codom* $(\leq_R)$ *z x* $\leq_L$ *r z* **by** *auto*
with *galois-prop1* **obtain** *y y'* **where** *in-dom* $(\leq_{L1})$ *x l1 x* $\leq_{R1}$ *y y* $\leq_{L2}$ *y' y'* $\leq_{R1} \varepsilon_1$ *(r2 z)*
by (*auto elim*!: *left-relE*)
moreover have ε_1 *(r2 z)* $\leq_{R1}$ *r2 z*
proof −
from *mono-in-codom-r2* ‹*in-codom* $(\leq_R)$ *z*› **have** *in-codom* $(\leq_{R1})$ *(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'* $\leq_{R1}$ *r2 z* **using** *preorder-R1* **by** *blast*
with ‹*l1 x* $\leq_{R1}$ *y*› ‹*y* $\leq_{L2}$ *y'*› **have** $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1}))$ *(l1 x) (r2 z)*
by *blast*
with *rel-comp-le* **obtain** *y''* **where** *l1 x* $\leq_{R1}$ *y'' y''* $\leq_{L2}$ *r2 z* **by** *blast*
with *galois-prop1* ‹*in-dom* $(\leq_{L1})$ *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* $(\leq_R)$ *z*› ‹*y''* $\leq_{L2}$ *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*: $((\leq_{R1}) \Rightarrow_m (\leq_{L1}))$ *r1*
and *half-galois-prop-left1*: $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*
and *half-galois-prop-right1*: $((\leq_{R1}) \trianglerighteq_h (\leq_{L1}))$ *r1 l1*
and *refl-R1*: *reflexive-on* (*in-codom* $(\leq_{R1})$) $(\leq_{R1})$
and *mono-l2*: $((\leq_{L2}) \Rightarrow_m (\leq_{R2}))$ *l2*
and *refl-L2*: *reflexive-on* (*in-dom* $(\leq_{L2})$) $(\leq_{L2})$
and *in-codom-rel-comp-le*: *in-codom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-codom* $(\leq_{R1})$
shows $(({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox)) \leq ({}_L\lessapprox)$
proof (*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 ≤_L2 y› show l2 y _R2≲≈ y by blast
    show y ≤_R1 y y _L2≲≈ z by fact+
  qed
  show x ≤_L r z
  proof (intro left-relI)
    show x _L1≲≈ y y ≤_L2 r2 z by fact+
    show r2 z _R1≲≈ r z
    proof (intro flip.t2.left-GaloisI)
      from ‹y ≤_L2 y› ‹y ≤_R1 y› ‹y ≤_L2 r2 z› have ((≤_L2) ∘∘ (≤_R1) ∘∘ (≤_L2)) y
(r2 z)
        by blast
      with in-codom-rel-comp-le have in-codom (≤_R1) (r2 z) by blast
      with refl-R1 have r2 z ≤_R1 r2 z by blast
      with mono-r1 show in-codom (≤_L1) (r z) by auto
      with ‹r2 z ≤_R1 r2 z›  half-galois-prop-right1 mono-r1
        show r2 z ≤_R1 l1 (r z) by (fastforce 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 ((≤_R1) ⇛_m (≤_L1)) r1
  and ((≤_L1) ⊴ (≤_R1)) l1 r1
  and ((≤_R1) ⊴_h (≤_L1)) r1 l1
  and preorder-on (in-field (≤_R1)) (≤_R1)
  and ((≤_L2) ⇛_m (≤_R2)) l2
  and reflexive-on (in-dom (≤_L2)) (≤_L2)
  and ((≤_R1) ∘∘ (≤_L2) ∘∘ (≤_R1)) ≤ ((≤_R1) ∘∘ (≤_L2))
  and (in-codom (≤_R) ⇛_m in-codom (≤_R1)) r2
  and in-codom ((≤_L2) ∘∘ (≤_R1) ∘∘ (≤_L2)) ≤ in-codom (≤_R1)
  shows (_L≲) = ((_L1≲≈) ∘∘ (_L2≲≈))
  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 ((≤_R1) ⇛_m (≤_L1)) r1
  and ((≤_L1) ⊴ (≤_R1)) l1 r1
  and ((≤_R1) ⊴_h (≤_L1)) r1 l1
  and preorder-on (in-field (≤_R1)) (≤_R1)
  and ((≤_L2) ⇛_m (≤_R2)) l2
  and ((≤_R2) _h⊴ (≤_L2)) r2 l2
  and reflexive-on (in-dom (≤_L2)) (≤_L2)
  and ((≤_R1) ∘∘ (≤_L2) ∘∘ (≤_R1)) ≤ ((≤_R1) ∘∘ (≤_L2))
  and in-codom ((≤_L2) ∘∘ (≤_R1) ∘∘ (≤_L2)) ≤ in-codom (≤_R1)
  shows (_L≲) = ((_L1≲≈) ∘∘ (_L2≲≈))
  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 $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1*
and *preorder-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and $((\leq_{R2}) \dashv (\leq_{L2}))$ *r2 l2*
and *reflexive-on* (*in-dom* $(\leq_{L2})$) $(\leq_{L2})$
and $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{R1}) \circ\circ (\leq_{L2}))$
and *in-codom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-codom* $(\leq_{R1})$
shows $({}_L\lessapprox) = (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$
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 $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1*
and *preorder-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and $((\leq_{R2}) \dashv (\leq_{L2}))$ *r2 l2*
and *reflexive-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$
and *in-codom* $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-codom* $(\leq_{L2})$
and $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq ((\leq_{R1}) \circ\circ (\leq_{L2}))$
and *in-codom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-codom* $(\leq_{R1})$
shows $({}_L\lessapprox) = (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$
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 $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and $((\leq_{R2}) \equiv_{pre} (\leq_{L2}))$ *r2 l2*
and *in-codom* $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-codom* $(\leq_{L2})$
and $(\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2}) \leq (\leq_{R1}) \circ\circ (\leq_{L2})$
and *in-codom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-codom* $(\leq_{R1})$
shows $({}_L\lessapprox) = (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$
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*: $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
and *in-dom-L1-le*: *in-dom* $(\leq_{L1}) \leq$ *in-codom* $(\leq_{L1})$
and *refl-R1*: *reflexive-on* (*in-dom* $(\leq_{R1})$) $(\leq_{R1})$
and *refl-L2*: *reflexive-on* (*in-dom* $(\leq_{L2})$) $(\leq_{L2})$
and *mono-in-dom-l1*: (*in-dom* $(\leq_L) \Rrightarrow_m$ *in-dom* $(\leq_{L2})$) *l1*
shows *reflexive-on* (*in-dom* $(\leq_L)$) $(\leq_L)$
proof (*rule reflexive-onI*)
fix *x* **assume** *in-dom* $(\leq_L)$ *x*
then obtain x' **where** $x \leq_L x'$ *in-dom* $(\leq_{L1})$ *x* **by** *blast*
show $x \leq_L x$
proof (*rule left-relI*)
from *refl-R1* **have** *l1 x* $\leq_{R1}$ *l1 x*
proof (*rule reflexive-onD*)
from ‹$x \leq_L x'$› *galois-prop* **show** *in-dom* $(\leq_{R1})$ (*l1 x*) **by** *blast*
qed
then show *x* $_{L1}\lessapprox$ *l1 x*
proof (*intro t1.left-GaloisI*)
from *galois-prop* ‹*in-dom* $(\leq_{L1})$ *x*› ‹*l1 x* $\leq_{R1}$ *l1 x*› **show** *x* $\leq_{L1}$ *r1* (*l1 x*)
by *blast*
qed *blast*
from *refl-L2* **show** *l1 x* $\leq_{L2}$ *l1 x*
proof (*rule reflexive-onD*)
from *mono-in-dom-l1* ‹$x \leq_L x'$› **show** *in-dom* $(\leq_{L2})$ (*l1 x*) **by** *blast*
qed
from ‹*l1 x* $\leq_{R1}$ *l1 x*› **show** *l1 x* $_{R1}\gtrapprox$ *x*
proof (*intro flip1.left-GaloisI*)
from ‹*in-dom* $(\leq_{L1})$ *x*› *in-dom-L1-le* **show** *in-codom* $(\leq_{L1})$ *x* **by** *blast*
qed
qed
qed

lemma *reflexive-on-in-codom-leftI*:
assumes *L1-r1-l1I*: $\bigwedge x$. *in-dom* $(\leq_{L1})$ *x* $\Longrightarrow$ *l1 x* $\leq_{R1}$ *l1 x* $\Longrightarrow$ *x* $\leq_{L1}$ *r1* (*l1 x*)
and *in-codom-L1-le*: *in-codom* $(\leq_{L1}) \leq$ *in-dom* $(\leq_{L1})$
and *refl-R1*: *reflexive-on* (*in-codom* $(\leq_{R1})$) $(\leq_{R1})$
and *refl-L2*: *reflexive-on* (*in-codom* $(\leq_{L2})$) $(\leq_{L2})$
and *mono-in-codom-l1*: (*in-codom* $(\leq_L) \Rrightarrow_m$ *in-codom* $(\leq_{L2})$) *l1*
shows *reflexive-on* (*in-codom* $(\leq_L)$) $(\leq_L)$
proof (*rule reflexive-onI*)

```
  fix x assume in-codom (≤_L) x
  then obtain x' where x' ≤_L x in-codom (≤_L1) x in-codom (≤_R1) (l1 x)
    by blast
  show x ≤_L x
  proof (rule left-relI)
    from refl-R1 ‹in-codom (≤_R1) (l1 x)› have l1 x ≤_R1 l1 x by blast
    show x _L1≲≈ l1 x
    proof (rule t1.left-GaloisI)
      from in-codom-L1-le ‹in-codom (≤_L1) x› have in-dom (≤_L1) x by blast
      with ‹l1 x ≤_R1 l1 x› show x ≤_L1 r1 (l1 x) by (intro L1-r1-l1I)
    qed fact
    from refl-L2 show l1 x ≤_L2 l1 x
    proof (rule reflexive-onD)
      from mono-in-codom-l1 ‹x' ≤_L x› show in-codom (≤_L2) (l1 x) by blast
    qed
    show l1 x _R1≲≈ x by (rule flip1.left-GaloisI) fact+
  qed
qed

corollary reflexive-on-in-field-leftI:
  assumes ((≤_L1) ⊴ (≤_R1)) l1 r1
  and in-codom (≤_L1) = in-dom (≤_L1)
  and reflexive-on (in-field (≤_R1)) (≤_R1)
  and reflexive-on (in-field (≤_L2)) (≤_L2)
  and (in-field (≤_L) ⇛_m in-field (≤_L2)) l1
  shows reflexive-on (in-field (≤_L)) (≤_L)
proof −
  from assms have reflexive-on (in-dom (≤_L)) (≤_L)
    by (intro reflexive-on-in-dom-leftI)
    (auto 0 5 intro: reflexive-on-if-le-pred-if-reflexive-on in-field-if-in-dom)
  moreover from assms have reflexive-on (in-codom (≤_L)) (≤_L)
    by (intro reflexive-on-in-codom-leftI)
    (auto 0 5 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 $(\leq_{R1})$ and $(\leq_{L2})$ has to be transitive and the order of commutativity for the relations.

In the following, we just give two of them that suffice for many purposes.

```
lemma transitive-leftI:
  assumes ((≤_L1) ₕ⊴ (≤_R1)) l1 r1
  and trans-L2: transitive (≤_L2)
  and R1-L2-R1-le: ((≤_R1) ∘∘ (≤_L2) ∘∘ (≤_R1)) ≤ ((≤_L2) ∘∘ (≤_R1))
  shows transitive (≤_L)
proof (rule transitiveI)
  fix x1 x2 x3 assume x1 ≤_L x2 x2 ≤_L x3
```

from ‹$x1 \leq_L x2$› **obtain** $y1\ y2$ **where** $x1\ _{L1}\lessapprox y1\ y1 \leq_{L2} y2\ y2 \leq_{R1} l1\ x2$
by *blast*
from ‹$x2 \leq_L x3$› ‹$((\leq_{L1})\ _h\trianglelefteq (\leq_{R1}))\ l1\ r1$› **obtain** $y3\ y4$ **where**
$l1\ x2 \leq_{R1} y3\ y3 \leq_{L2} y4\ y4 \leq_{R1} l1\ x3$ *in-codom* $(\leq_{L1})\ x3$ **by** *blast*
with *R1-L2-R1-le* **have** $((\leq_{L2}) \circ\circ (\leq_{R1}))\ (l1\ x2)\ (l1\ x3)$ **by** *blast*
then obtain y **where** $l1\ x2 \leq_{L2} y\ y \leq_{R1} l1\ x3$ **by** *blast*
with ‹$y2 \leq_{R1} l1\ x2$› *R1-L2-R1-le* **have** $((\leq_{L2}) \circ\circ (\leq_{R1}))\ y2\ (l1\ x3)$ **by** *blast*
then obtain y' **where** $y2 \leq_{L2} y'\ y' \leq_{R1} l1\ x3$ **by** *blast*
with ‹$y1 \leq_{L2} y2$› **have** $y1 \leq_{L2} y'$ **using** *trans-L2* **by** *blast*
show $x1 \leq_L x3$
proof (*rule left-relI*)
show $x1\ _{L1}\lessapprox y1\ y1 \leq_{L2} y'$ **by** *fact+*
show $y'\ _{R1}\lessapprox x3$ **by** (*rule flip1.left-GaloisI*) *fact+*
qed
qed

lemma *transitive-leftI'*:
assumes *galois-prop*: $((\leq_{L1}) \trianglelefteq (\leq_{R1}))\ l1\ r1$
and *trans-L2*: *transitive* $(\leq_{L2})$
and *R1-L2-R1-le*: $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{R1}) \circ\circ (\leq_{L2}))$
shows *transitive* $(\leq_L)$
proof (*rule transitiveI*)
fix $x1\ x2\ x3$ **assume** $x1 \leq_L x2\ x2 \leq_L x3$
from ‹$x1 \leq_L x2$› *galois-prop* **obtain** $y1\ y2$ **where**
in-dom $(\leq_{L1})\ x1\ l1\ x1 \leq_{R1} y1\ y1 \leq_{L2} y2\ y2 \leq_{R1} l1\ x2$ **by** *blast*
with *R1-L2-R1-le* **have** $((\leq_{R1}) \circ\circ (\leq_{L2}))\ (l1\ x1)\ (l1\ x2)$ **by** *blast*
then obtain y **where** $l1\ x1 \leq_{R1} y\ y \leq_{L2} l1\ x2$ **by** *blast*
moreover from ‹$x2 \leq_L x3$› *galois-prop* **obtain** $y3\ y4$ **where**
$l1\ x2 \leq_{R1} y3\ y3 \leq_{L2} y4\ y4\ _{R1}\lessapprox x3$ **by** *blast*
moreover note *R1-L2-R1-le*
ultimately have $((\leq_{R1}) \circ\circ (\leq_{L2}))\ (l1\ x1)\ y3$ **by** *blast*
then obtain y' **where** $l1\ x1 \leq_{R1} y'\ y' \leq_{L2} y3$ **by** *blast*
with ‹$y3 \leq_{L2} y4$› **have** $y' \leq_{L2} y4$ **using** *trans-L2* **by** *blast*
show $x1 \leq_L x3$
proof (*rule left-relI*)
from ‹*in-dom* $(\leq_{L1})\ x1$› ‹$l1\ x1 \leq_{R1} y'$› *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 $y' \leq_{L2} y4$ **by** *fact*
from ‹$y' \leq_{L2} y4$› ‹$y4\ _{R1}\lessapprox x3$› **show** $y4\ _{R1}\lessapprox x3$ **by** *blast*
qed
qed

Preorders

lemma *preorder-on-in-field-leftI*:
assumes $((\leq_{L1}) \trianglelefteq (\leq_{R1}))\ l1\ r1$
and *in-codom* $(\leq_{L1})$ = *in-dom* $(\leq_{L1})$
and *reflexive-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$

```
  and preorder-on (in-field (≤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 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 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 ((≤L1) ◃ (≤R1)) l1 r1
  and in-codom (≤L1) = in-dom (≤L1)
  and symmetric (≤R1)
  and symmetric (≤L2)
  shows symmetric (≤L)
proof −
```

from *assms* **have** $(\gtrsim_{R1}) = ({}_{L1}\lesssim)$ **by** (*intro*
t1.ge-Galois-right-eq-left-Galois-if-symmetric-if-in-codom-eq-in-dom-if-galois-prop)
moreover then have $({}_{R1}\lesssim) = (\gtrsim_{L1})$
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 $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
and *in-codom* $(\leq_{L1})$ = *in-dom* $(\leq_{L1})$
and *symmetric* $(\leq_{R1})$
and *partial-equivalence-rel* $(\leq_{L2})$
and $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{L2}) \circ\circ (\leq_{R1}))$
shows *partial-equivalence-rel* $(\leq_L)$
using *assms* **by** (*intro partial-equivalence-relI transitive-leftI symmetric-leftI*)
auto

lemma *partial-equivalence-rel-leftI'*:
assumes $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
and *in-codom* $(\leq_{L1})$ = *in-dom* $(\leq_{L1})$
and *symmetric* $(\leq_{R1})$
and *partial-equivalence-rel* $(\leq_{L2})$
and $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq ((\leq_{R1}) \circ\circ (\leq_{L2}))$
shows *partial-equivalence-rel* $(\leq_L)$
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 $((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))$ *r1*
and $((\leq_{L1}) \ {}_h\trianglelefteq (\leq_{R1}))$ *l1 r1*
and *inflationary-unit1*: *inflationary-on* (*in-dom* $(\leq_{L1})$) $(\leq_{L1})$ η_1
and *inflationary-counit1*: *inflationary-on* (*in-codom* $(\leq_{R1})$) $(\leq_{R1})$ ε_1
and *refl-R1*: *reflexive-on* (*in-dom* $(\leq_{R1})$) $(\leq_{R1})$
and *inflationary-unit2*: *inflationary-on* (*in-dom* $(\leq_{L2})$) $(\leq_{L2})$ η_2
and *refl-L2*: *reflexive-on* (*in-dom* $(\leq_{L2})$) $(\leq_{L2})$
and *mono-in-dom-l1*: (*in-dom* $(\leq_L) \Rrightarrow_m$ *in-dom* $(\leq_{L2})$) *l1*
and *in-codom-rel-comp-le*: *in-codom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-codom* $((\leq_{R1}))$
shows *inflationary-on* (*in-dom* $(\leq_L)$) $(\leq_L)$ η
proof (*rule inflationary-onI*)
fix *x* **assume** *in-dom* $(\leq_L)$ *x*
show $x \leq_L \eta\ x$
proof (*rule left-relI*)
from ‹*in-dom* $(\leq_L)$ *x*› ‹$((\leq_{L1}) \ {}_h\trianglelefteq (\leq_{R1}))$ *l1 r1*› **have** *in-dom* $(\leq_{R1})$ (*l1 x*) **by** *blast*
with *refl-R1* **have** $l1\ x \leq_{R1} l1\ x$ **by** *blast*
moreover from ‹*in-dom* $(\leq_L)$ *x*› **have** *in-dom* $(\leq_{L1})$ *x* **by** *blast*
moreover note *inflationary-unit1*
ultimately show $x \ {}_{L1}\lessapprox l1\ x$ **by** (*intro t1.left-GaloisI*) *auto*
from ‹*in-dom* $(\leq_L)$ *x*› *mono-in-dom-l1* **have** *in-dom* $(\leq_{L2})$ (*l1 x*) **by** *blast*
with *inflationary-unit2* **show** $l1\ x \leq_{L2} r2\ (l\ x)$ **by** *auto*
show $r2\ (l\ x) \ {}_{R1}\gtrapprox \eta\ x$
proof (*rule flip.t2.left-GaloisI*)
from *refl-L2* ‹*in-dom* $(\leq_{L2})$ (*l1 x*)› **have** $l1\ x \leq_{L2} l1\ x$ **by** *blast*
with *in-codom-rel-comp-le* ‹$l1\ x \leq_{R1} l1\ x$› ‹$l1\ x \leq_{L2} r2\ (l\ x)$›
have *in-codom* $(\leq_{R1})$ $(r2\ (l\ x))$ **by** *blast*
with ‹$((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))$ *r1*› **show** *in-codom* $(\leq_{L1})$ $(\eta\ x)$
by (*auto intro*: *in-codom-if-rel-if-dep-mono-wrt-rel simp*: *mono-wrt-rel-eq-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

lemma *inflationary-on-in-codom-unitI*:
assumes $((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))$ *r1*
and *inflationary-unit1*: *inflationary-on* (*in-codom* $(\leq_{L1})$) $(\leq_{L1})$ η_1
and *inflationary-counit1*: *inflationary-on* (*in-codom* $(\leq_{R1})$) $(\leq_{R1})$ ε_1
and *refl-R1*: *reflexive-on* (*in-codom* $(\leq_{R1})$) $(\leq_{R1})$
and *inflationary-unit2*: *inflationary-on* (*in-codom* $(\leq_{L2})$) $(\leq_{L2})$ η_2
and *refl-L2*: *reflexive-on* (*in-codom* $(\leq_{L2})$) $(\leq_{L2})$
and *mono-in-codom-l1*: (*in-codom* $(\leq_L) \Rrightarrow_m$ *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 $r2\ (l\ x)\ {}_{R1}\gtrapprox \eta\ x$
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* $(\leq_{L1})$ $(\eta\ x)$
by (*auto intro*: *in-codom-if-rel-if-dep-mono-wrt-rel simp*: *mono-wrt-rel-eq-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* $(\leq_L)$) $(\leq_L)$ 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 $((\leq_{L1}) \Rightarrow_m (\leq_{R1}))$ *l1* $((\leq_{R1}) \Rightarrow_m (\leq_{L1}))$ *r1*
and *refl-L1*: *reflexive-on* (*in-dom* $(\leq_{L1})$) $(\leq_{L1})$
and *in-dom-R1-le-in-codom-R1*: *in-dom* $(\leq_{R1}) \leq$ *in-codom* $(\leq_{R1})$
and *deflationary-L2*: *deflationary-on* (*in-dom* $(\leq_{L2})$) $(\leq_{L2})$ η_2
and *refl-L2*: *reflexive-on* (*in-dom* $(\leq_{L2})$) $(\leq_{L2})$
and *mono-in-dom-l1*: (*in-dom* $(\leq_L) \Rightarrow_m$ *in-dom* $(\leq_{L2})$) *l1*
and *in-dom-rel-comp-le*: *in-dom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-dom* $((\leq_{R1}))$
shows *deflationary-on* (*in-dom* $(\leq_L)$) $(\leq_L)$ η
proof (*rule deflationary-onI*)
fix *x* **assume** *in-dom* $(\leq_L)$ *x*
show $\eta\ x \leq_L x$
proof (*rule left-relI*)
from *refl-L1* ‹*in-dom* $(\leq_L)$ *x*› **have** $x \leq_{L1} x$ **by** *blast*
moreover with ‹$((\leq_{L1}) \Rightarrow_m (\leq_{R1}))$ *l1*› **have** *l1 x* $\leq_{R1}$ *l1 x* **by** *blast*
ultimately show *l1 x* $_{R1}\lesssim_{\approx} x$ **by** *auto*
from *mono-in-dom-l1* ‹*in-dom* $(\leq_L)$ *x*› **have** *in-dom* $(\leq_{L2})$ (*l1 x*) **by** *blast*
with *deflationary-L2* **show** *r2* (*l x*) $\leq_{L2}$ *l1 x* **by** *auto*
show $\eta\ x$ $_{L1}\lesssim_{\approx}$ *r2* (*l x*)
proof (*rule t1.left-GaloisI*)
from *refl-L2* ‹*in-dom* $(\leq_{L2})$ (*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 ‹$((\leq_{R1}) \Rightarrow_m (\leq_{L1}))$ *r1*› **have** *in-dom* $(\leq_{L1})$ $(\eta\ x)$
by (*auto intro*: *in-dom-if-rel-if-dep-mono-wrt-rel simp*: *mono-wrt-rel-eq-dep-mono-wrt-rel*)
with *refl-L1* **show** $\eta\ x \leq_{L1}$ *r1* (*r2* (*l x*))
by (*auto intro*: *in-field-if-in-codom*)
from ‹*in-dom* $(\leq_{R1})$ (*r2* (*l x*))› *in-dom-R1-le-in-codom-R1*
show *in-codom* $(\leq_{R1})$ (*r2* (*l x*)) **by** *blast*
qed
qed
qed

lemma *deflationary-on-in-codom-unitI*:
assumes $((\leq_{L1}) \Rightarrow_m (\leq_{R1}))$ *l1* $((\leq_{R1}) \Rightarrow_m (\leq_{L1}))$ *r1*
and *refl-L1*: *reflexive-on* (*in-codom* $(\leq_{L1})$) $(\leq_{L1})$
and *in-dom-R1-le-in-codom-R1*: *in-dom* $(\leq_{R1}) \leq$ *in-codom* $(\leq_{R1})$
and *deflationary-L2*: *deflationary-on* (*in-codom* $(\leq_{L2})$) $(\leq_{L2})$ η_2
and *refl-L2*: *reflexive-on* (*in-codom* $(\leq_{L2})$) $(\leq_{L2})$
and *mono-in-codom-l1*: (*in-codom* $(\leq_L) \Rightarrow_m$ *in-codom* $(\leq_{L2})$) *l1*
and *in-dom-rel-comp-le*: *in-dom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-dom* $((\leq_{R1}))$
shows *deflationary-on* (*in-codom* $(\leq_L)$) $(\leq_L)$ η
proof (*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 ‹$((\leq_{L1}) \Rightarrow_m (\leq_{R1}))$ *l1*› **have** *l1 x* $\leq_{R1}$ *l1 x* **by** *blast*
ultimately show *l1 x* $_{R1}\lesssim_{\approx} x$ **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 η *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 simp*: *mono-wrt-rel-eq-dep-mono-wrt-rel*)
with *refl-L1* **show** η *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 $((\leq_{L1}) \ {}_h\trianglelefteq (\leq_{R1}))$ *l1 r1*
and *inflationary-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$ η_1
and *inflationary-on* (*in-codom* $(\leq_{R1})$) $(\leq_{R1})$ ε_1
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and *reflexive-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and *rel-equivalence-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$ η_2
and *reflexive-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$
and (*in-dom* $(\leq_L) \Rrightarrow_m$ *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* (($\leq_{R1}$))
and *in-codom* (($\leq_{L2}$) $\circ\circ$ ($\leq_{R1}$) $\circ\circ$ ($\leq_{L2}$)) $\leq$ *in-codom* (($\leq_{R1}$))
shows *rel-equivalence-on* (*in-field* ($\leq_L$)) ($\leq_L$) η
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 (($\leq_{L1}$) $\Rrightarrow_m$ ($\leq_{R1}$)) *l1* (($\leq_{R1}$) $\Rrightarrow_m$ ($\leq_{L1}$)) *r1*
and (($\leq_{L1}$) $_h\trianglelefteq$ ($\leq_{R1}$)) *l1 r1*
and *inflationary-on* (*in-field* ($\leq_{L1}$)) ($\leq_{L1}$) η_1
and *rel-equiv-counit1*: *rel-equivalence-on* (*in-field* ($\leq_{R1}$)) ($\leq_{R1}$) ε_1
and *reflexive-on* (*in-field* ($\leq_{L1}$)) ($\leq_{L1}$)
and *reflexive-on* (*in-field* ($\leq_{R1}$)) ($\leq_{R1}$)
and (($\leq_{R2}$) $\Rrightarrow_m$ ($\leq_{L2}$)) *r2* (($\leq_{L2}$) $\Rrightarrow_m$ ($\leq_{R2}$)) *l2*
and (($\leq_{R2}$) $_h\trianglelefteq$ ($\leq_{L2}$)) *r2 l2*
and *rel-equiv-unit2*: *rel-equivalence-on* (*in-field* ($\leq_{L2}$)) ($\leq_{L2}$) η_2
and *inflationary-on* (*in-field* ($\leq_{R2}$)) ($\leq_{R2}$) ε_2
and *reflexive-on* (*in-field* ($\leq_{L2}$)) ($\leq_{L2}$)
and *reflexive-on* (*in-field* ($\leq_{R2}$)) ($\leq_{R2}$)
and *middle-compatible*: *middle-compatible-codom*
shows (($\leq_L$) $\equiv_o$ ($\leq_R$)) *l r*
proof (*rule order-equivalenceI*)
show (($\leq_L$) $\Rrightarrow_m$ ($\leq_R$)) *l* **using** *rel-equiv-unit2* ‹(($\leq_{L1}$) $_h\trianglelefteq$ ($\leq_{R1}$)) *l1 r1*›
‹(($\leq_{L2}$) $\Rrightarrow_m$ ($\leq_{R2}$)) *l2*› *middle-compatible*
by (*intro mono-wrt-rel-leftI*) *auto*
show (($\leq_R$) $\Rrightarrow_m$ ($\leq_L$)) *r* **using** *rel-equiv-counit1* ‹(($\leq_{R2}$) $_h\trianglelefteq$ ($\leq_{L2}$)) *r2 l2*›
‹(($\leq_{R1}$) $\Rrightarrow_m$ ($\leq_{L1}$)) *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 $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-dom* $(\leq_{L2})$
in-dom $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-dom* $((\leq_{R1}))$
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* $(\leq_L) \Rrightarrow_m$ *in-dom* $(\leq_{L2})$) *l1*
and (*in-codom* $(\leq_L) \Rrightarrow_m$ *in-codom* $(\leq_{L2})$) *l1*
using ‹$((\leq_{L1})\ {}_h\trianglelefteq\ (\leq_{R1}))$ *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* $(\leq_L)$) $(\leq_L)$ 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* $(\leq_R) \Rrightarrow_m$ *in-dom* $(\leq_{R1})$) *r2*
and (*in-codom* $(\leq_R) \Rrightarrow_m$ *in-codom* $(\leq_{R1})$) *r2*
using ‹$((\leq_{R2})\ {}_h\trianglelefteq\ (\leq_{L2}))$ *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* $(\leq_R)$) $(\leq_R)$ 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 $((\leq_{L1}) \equiv_o (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and *transitive* $(\leq_{R1})$
and $((\leq_{L2}) \equiv_o (\leq_{R2}))$ *l2 r2*
and *transitive* $(\leq_{L2})$
and *reflexive-on* (*in-field* $(\leq_{R2})$) $(\leq_{R2})$
and *middle-compatible-codom*
shows $((\leq_L) \equiv_o (\leq_R))$ *l r*
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 $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and *reflexive-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and $((\leq_{L2}) \equiv_G (\leq_{R2}))$ *l2 r2*
and *reflexive-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$
and *reflexive-on* (*in-field* $(\leq_{R2})$) $(\leq_{R2})$

and *middle-compatible-codom*
shows $((\leq_L) \equiv_o (\leq_R))$ *l r*
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 $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and *preorder-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and $((\leq_{L2}) \equiv_G (\leq_{R2}))$ *l2 r2*
and *preorder-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$
and *reflexive-on* (*in-field* $(\leq_{R2})$) $(\leq_{R2})$
and *middle-compatible-codom*
shows $((\leq_L) \dashv_{pre} (\leq_R))$ *l r*
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 $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and $((\leq_{R2}) \dashv_{pre} (\leq_{L2}))$ *r2 l2*
and *middle-compatible-codom*
shows $({}_L\lessapprox) = (({}_{L1}\lessapprox) \circ\circ ({}_{L2}\lessapprox))$
using *assms* **by** (*intro left-Galois-eq-comp-left-Galois-if-galois-connection-if-galois-equivalenceI*)
auto

For theorems with weaker assumptions, see $[\![((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))\ r1$; *t1.galois-prop l1 r1*; *flip.t2.half-galois-prop-right*; *preorder-on* (*in-field* $(\leq_{R1})$)

$(\leq_{R1})$; $((\leq_{L2}) \Rightarrow_m (\leq_{R2}))$ *l2*; *flip.t1.half-galois-prop-left*; *reflexive-on* (*in-dom* $(\leq_{L2})$) $(\leq_{L2})$; $(\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1}) \leq (\leq_{R1}) \circ\circ (\leq_{L2})$; *in-codom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-codom* $(\leq_{R1})$⟧ ⟹ *flip.right-Galois* = *flip.t2.right-Galois* ∘∘ *flip.t1.right-Galois*

⟦*t1.galois-equivalence*; *preorder-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$; *flip.t1.galois-connection*; *reflexive-on* (*in-field* $(\leq_{L2})$) $(\leq_{L2})$; *in-codom* $((\leq_{R1}) \circ\circ (\leq_{L2}) \circ\circ (\leq_{R1})) \leq$ *in-codom* $(\leq_{L2})$; $(\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2}) \leq (\leq_{R1}) \circ\circ (\leq_{L2})$; *in-codom* $((\leq_{L2}) \circ\circ (\leq_{R1}) \circ\circ (\leq_{L2})) \leq$ *in-codom* $(\leq_{R1})$⟧ ⟹ *flip.right-Galois* = *flip.t2.right-Galois* ∘∘ *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* ⇒ *bool*) (*R* :: *'a* ⇒ -)
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* $R^{\oplus}$
by (*urule symmetric-on-if-le-pred-if-symmetric-on*)
(*use assms* **in** ‹*urule Refl-Rel-symmetric-onI*›, *simp*)

lemma *Refl-Rel-partial-equivalence-rel-onI* [*intro*]:
assumes *partial-equivalence-rel-on* (*P* :: *'a* ⇒ *bool*) (*R* :: *'a* ⇒ -)
shows *partial-equivalence-rel-on P* $R^{\oplus}$
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* ⇛$_m$ *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 ‹$(S \Rightarrow_m R)$ 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 $(S \Rightarrow_m R)$ 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 5 0 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 $(R \Rightarrow_m S)$ f
shows $(R^{\oplus} \Rightarrow_m S^{\oplus})$ f
using *assms* **by** (*intro mono-wrt-relI*) *blast*

context *galois*
begin

interpretation *gR* : *galois* $(\leq_L)^{\oplus}$ $(\leq_R)^{\oplus}$ l r **.**

lemma *Galois-Refl-RelI*:
assumes $((\leq_R) \Rightarrow_m (\leq_L))$ r
and *in-field* $(\leq_L)^{\oplus}$ x
and *in-field* $(\leq_R)^{\oplus}$ y
and *in-codom* $(\leq_R)$ $y \Longrightarrow x\ {}_L\overset{\leq}{\approx}\ y$
shows (*galois-rel.Galois* $((\leq_L)^{\oplus})$ $((\leq_R)^{\oplus})$ 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 $((\leq_L) \Rightarrow_m (\leq_R))$ l
and $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))$ l r
shows $((\leq_L)^{\oplus}\ {}_h\trianglelefteq\ (\leq_R)^{\oplus})$ 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 $((\geq_R) \Rightarrow_m (\geq_L)) \equiv ((\leq_R) \Rightarrow_m (\leq_L))$
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*
mono-wrt-rel-eq-dep-mono-wrt-rel)

lemma *half-galois-prop-right-Refl-Rel-right-leftI*:
assumes $((\leq_R) \Rightarrow_m (\leq_L))\ r$
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 $((\leq_L) \Rightarrow_m (\leq_R))\ l$
and $((\leq_R) \Rightarrow_m (\leq_L))\ r$
and *inflationary-on* (*in-dom* $(\leq_L)$) $(\leq_L)\ \eta$
shows *inflationary-on* (*in-field* $(\leq_L)^{\oplus}$) $(\leq_L)^{\oplus}\ \eta$
using *assms*
by (*intro inflationary-onI Refl-RelI*) (*auto 0 3 elim*!: *in-fieldE Refl-RelE*)

lemma *inflationary-on-in-field-Refl-Rel-left′*:
assumes $((\leq_L) \Rightarrow_m (\leq_R))\ l$
and $((\leq_R) \Rightarrow_m (\leq_L))\ r$
and *inflationary-on* (*in-codom* $(\leq_L)$) $(\leq_L)\ \eta$
shows *inflationary-on* (*in-field* $(\leq_L)^{\oplus}$) $(\leq_L)^{\oplus}\ \eta$
using *assms*

by (*intro inflationary-onI Refl-RelI*) (*auto 0 3 elim*!: *in-fieldE Refl-RelE*)

interpretation *inv* : *galois* $(\geq_L)$ $(\geq_R)$ *l r*
rewrites $((\geq_L) \Rightarrow_m (\geq_R)) \equiv ((\leq_L) \Rightarrow_m (\leq_R))$
and $((\geq_R) \Rightarrow_m (\geq_L)) \equiv ((\leq_R) \Rightarrow_m (\leq_L))$
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 $\bigwedge(P :: 'c \Rightarrow bool)\ (R :: 'd \Rightarrow 'c \Rightarrow bool).$
(*inflationary-on* P $R^{-1} :: ('c \Rightarrow 'd) \Rightarrow bool) \equiv$ *deflationary-on* P R
by (*simp-all add*: *mono-wrt-rel-eq-dep-mono-wrt-rel*)

lemma *deflationary-on-in-field-Refl-Rel-leftI*:
assumes $((\leq_L) \Rightarrow_m (\leq_R))$ *l*
and $((\leq_R) \Rightarrow_m (\leq_L))$ *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 $((\leq_L) \Rightarrow_m (\leq_R))$ *l*
and $((\leq_R) \Rightarrow_m (\leq_L))$ *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 $((\leq_L) \Rightarrow_m (\leq_R))$ *l*
and $((\leq_R) \Rightarrow_m (\leq_L))$ *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 $((\leq_L) \Rightarrow_m (\leq_R))$ *l*
and $((\leq_R) \Rightarrow_m (\leq_L))$ *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* :: *'a* ⇒ *'a* ⇒ *bool*) (*S* :: *'a* ⇒ *'a* ⇒ *'b* ⇒ *'b* ⇒ *bool*) ≡
  ((*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* ⇒ *'a* ⇒ *bool*) ⇒ (*'b* ⇒ *'b* ⇒ *bool*) ⇒ (*'a* ⇒ *'b*) ⇒
    (*'a* ⇒ *'b*) ⇒ *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* ⇒ *'a* ⇒ *bool*) ⇒ (*'b* ⇒ *'b* ⇒ *bool*) ⇒ (*'a* ⇒ *'b*) ⇒
    (*'a* ⇒ *'b*) ⇒ *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
unbundle *Mono-Dep-Fun-Rel-syntax*

translations
R ⇛⊕ *S* ⇌ *CONST Mono-Fun-Rel R S*
(*x y* :: *R*) ⇛⊕ *S* ⇌ *CONST Mono-Dep-Fun-Rel R* (λ*x y*. *S*)
(*x y* :: *R* | *B*) ⇛⊕ *S* ⇌ *CONST Mono-Dep-Fun-Rel R* (λ*x y*. *CONST rel-if B S*)

locale *Dep-Fun-Rel-orders* =
fixes *L* :: *'a* ⇒ *'b* ⇒ *bool*
and *R* :: *'a* ⇒ *'b* ⇒ *'c* ⇒ *'d* ⇒ *bool*
begin

sublocale *o* : *orders L R a b* **for** *a b* **.**

notation *L* (**infix** ≤$_L$ *50*)
notation *o.ge-left* (**infix** ≥$_L$ *50*)

notation *R* ((≤$_{R\ (-)\ (-)}$) *50*)
abbreviation *right-infix c a b d* ≡ (≤$_{R\ a\ b}$) *c d*
notation *right-infix* ((-) ≤$_{R\ (-)\ (-)}$ (-) [*51*,*51*,*51*,*51*] *50*)

notation *o.ge-right* ((≥$_{R\ (-)\ (-)}$) *50*)

abbreviation (*input*) *ge-right-infix d a b c* ≡ (≥$_{R\ a\ b}$) *d c*
notation *ge-right-infix* ((-) ≥$_{R\ (-)\ (-)}$ (-) [*51*,*51*,*51*,*51*] *50*)

abbreviation (*input*) *DFR* ≡ ((*a b* :: *L*) ⇛ *R a b*)

end

locale *hom-Dep-Fun-Rel-orders* = *Dep-Fun-Rel-orders L R*
for *L* :: *'a* ⇒ *'a* ⇒ *bool*
and *R* :: *'a* ⇒ *'a* ⇒ *'b* ⇒ *'b* ⇒ *bool*
begin

sublocale *ho* : *hom-orders L R a b* **for** *a b* **.**

lemma *Mono-Dep-Fun-Refl-Rel-right-eq-Mono-Dep-Fun-if-le-if-reflexive-onI*:
assumes *refl-L*: *reflexive-on* (*in-field* (≤$_L$)) (≤$_L$)
and ⋀*x1 x2*. *x1* ≤$_L$ *x2* ⟹ (≤$_{R\ x2\ x2}$) ≤ (≤$_{R\ x1\ x2}$)
and ⋀*x1 x2*. *x1* ≤$_L$ *x2* ⟹ (≤$_{R\ x1\ x1}$) ≤ (≤$_{R\ x1\ x2}$)
shows ((*x y* :: (≤$_L$)) ⇛⊕ (≤$_{R\ x\ y}$)$^{\oplus}$) = ((*x y* :: (≤$_L$)) ⇛⊕ (≤$_{R\ x\ y}$))
proof −
{

fix *f g x1 x2*
assume $((x\ y :: (\leq_L)) \Rrightarrow (\leq_R\ x\ y))\ f\ g\ x1 \leq_L x1\ x1 \leq_L x2$
with *assms* **have** $f\ x1 \leq_R x1\ x2\ g\ x1\ f\ x2 \leq_R x1\ x2\ g\ x2$ **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* $(\leq_L)$) $(\leq_L)$
and $((x1\ x2 :: (\geq_L)) \Rrightarrow_m (x3\ x4 :: (\leq_L) \mid x1 \leq_L x3) \Rrightarrow (\leq))\ R$
shows $((x\ y :: (\leq_L)) \Rrightarrow\oplus (\leq_R\ x\ y)^{\oplus}) = ((x\ y :: (\leq_L)) \Rrightarrow\oplus (\leq_R\ x\ y))$
using *assms*
by (*intro Mono-Dep-Fun-Refl-Rel-right-eq-Mono-Dep-Fun-if-le-if-reflexive-onI*)
(*auto 6 0*)

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* $(\leq_L)$) $(\leq_L)$
shows $((\leq_L) \Rrightarrow\oplus (\leq_R)^{\oplus}) = ((\leq_L) \Rrightarrow\oplus (\leq_R))$
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* $\equiv$ *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* $\longleftrightarrow$ *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** $\leq_{L1}$ *50*)
notation *R1* (**infix** $\leq_{R1}$ *50*)

notation *t1.ge-left* (**infix** $\geq_{L1}$ *50*)
notation *t1.ge-right* (**infix** $\geq_{R1}$ *50*)

notation *t1.left-Galois* (**infix** $_{L1}\lessapprox$ *50*)
notation *t1.ge-Galois-left* (**infix** $\gtrapprox_{L1}$ *50*)
notation *t1.right-Galois* (**infix** $_{R1}\lessapprox$ *50*)
notation *t1.ge-Galois-right* (**infix** $\gtrapprox_{R1}$ *50*)
notation *t1.right-ge-Galois* (**infix** $_{R1}\gtrapprox$ *50*)
notation *t1.Galois-right* (**infix** $\lessapprox_{R1}$ *50*)
notation *t1.left-ge-Galois* (**infix** $_{L1}\gtrapprox$ *50*)
notation *t1.Galois-left* (**infix** $\lessapprox_{L1}$ *50*)

notation *t1.unit* (η_1)
notation *t1.counit* (ε_1)

notation *L2* (($\leq_{L2}$ (-) (-)) *50*)
notation *R2* (($\leq_{R2}$ (-) (-)) *50*)

notation *dfro1.right-infix* ((-) $\leq_{L2}$ (-) (-) (-) [*51*,*51*,*51*,*51*] *50*)

notation *dfro2.right-infix* $((\text{-}) \leq_{R2\ (\text{-})\ (\text{-})} (\text{-})\ [51,51,51,51]\ 50)$

notation *dfro1.o.ge-right* $((\geq_{L2\ (\text{-})\ (\text{-})})\ 50)$
notation *dfro2.o.ge-right* $((\geq_{R2\ (\text{-})\ (\text{-})})\ 50)$

notation *dfro1.ge-right-infix* $((\text{-}) \geq_{L2\ (\text{-})\ (\text{-})} (\text{-})\ [51,51,51,51]\ 50)$
notation *dfro2.ge-right-infix* $((\text{-}) \geq_{R2\ (\text{-})\ (\text{-})} (\text{-})\ [51,51,51,51]\ 50)$

notation *l2* $(l2_{(\text{-})\ (\text{-})})$
notation *r2* $(r2_{(\text{-})\ (\text{-})})$

sublocale *t2* : *transport* $(\leq_{L2\ x\ (r1\ x')})\ (\leq_{R2\ (l1\ x)\ x'})\ l2_{x'\ x}\ r2_{x\ x'}$ **for** $x\ x'$ **.**

notation *t2.left-Galois* $((_{L2\ (\text{-})\ (\text{-})}\lessapprox)\ 50)$
notation *t2.right-Galois* $((_{R2\ (\text{-})\ (\text{-})}\lessapprox)\ 50)$

abbreviation *left2-Galois-infix* $y\ x\ x'\ y' \equiv (_{L2\ x\ x'}\lessapprox)\ y\ y'$
notation *left2-Galois-infix* $((\text{-})\ _{L2\ (\text{-})\ (\text{-})}\lessapprox (\text{-})\ [51,51,51,51]\ 50)$
abbreviation *right2-Galois-infix* $y'\ x\ x'\ y \equiv (_{R2\ x\ x'}\lessapprox)\ y'\ y$
notation *right2-Galois-infix* $((\text{-})\ _{R2\ (\text{-})\ (\text{-})}\lessapprox (\text{-})\ [51,51,51,51]\ 50)$

notation *t2.ge-Galois-left* $((\gtrapprox_{L2\ (\text{-})\ (\text{-})})\ 50)$
notation *t2.ge-Galois-right* $((\gtrapprox_{R2\ (\text{-})\ (\text{-})})\ 50)$

abbreviation (*input*) *ge-Galois-left-left2-infix* $y'\ x\ x'\ y \equiv (\gtrapprox_{L2\ x\ x'})\ y'\ y$
notation *ge-Galois-left-left2-infix* $((\text{-}) \gtrapprox_{L2\ (\text{-})\ (\text{-})} (\text{-})\ [51,51,51,51]\ 50)$
abbreviation (*input*) *ge-Galois-left-right2-infix* $y\ x\ x'\ y' \equiv (\gtrapprox_{R2\ x\ x'})\ y\ y'$
notation *ge-Galois-left-right2-infix* $((\text{-}) \gtrapprox_{R2\ (\text{-})\ (\text{-})} (\text{-})\ [51,51,51,51]\ 50)$

notation *t2.right-ge-Galois* $((_{R2\ (\text{-})\ (\text{-})}\gtrapprox)\ 50)$
notation *t2.left-ge-Galois* $((_{L2\ (\text{-})\ (\text{-})}\gtrapprox)\ 50)$

abbreviation *left2-ge-Galois-left-infix* $y\ x\ x'\ y' \equiv (_{L2\ x\ x'}\gtrapprox)\ y\ y'$
notation *left2-ge-Galois-left-infix* $((\text{-})\ _{L2\ (\text{-})\ (\text{-})}\gtrapprox (\text{-})\ [51,51,51,51]\ 50)$
abbreviation *right2-ge-Galois-left-infix* $y'\ x\ x'\ y \equiv (_{R2\ x\ x'}\gtrapprox)\ y'\ y$
notation *right2-ge-Galois-left-infix* $((\text{-})\ _{R2\ (\text{-})\ (\text{-})}\gtrapprox (\text{-})\ [51,51,51,51]\ 50)$

notation *t2.Galois-right* $((\lessapprox_{R2\ (\text{-})\ (\text{-})})\ 50)$
notation *t2.Galois-left* $((\lessapprox_{L2\ (\text{-})\ (\text{-})})\ 50)$

abbreviation (*input*) *Galois-left2-infix* $y'\ x\ x'\ y \equiv (\lessapprox_{L2\ x\ x'})\ y'\ y$
notation *Galois-left2-infix* $((\text{-}) \lessapprox_{L2\ (\text{-})\ (\text{-})} (\text{-})\ [51,51,51,51]\ 50)$
abbreviation (*input*) *Galois-right2-infix* $y\ x\ x'\ y' \equiv (\lessapprox_{R2\ x\ x'})\ y\ y'$
notation *Galois-right2-infix* $((\text{-}) \lessapprox_{R2\ (\text{-})\ (\text{-})} (\text{-})\ [51,51,51,51]\ 50)$

abbreviation *t2-unit* $x\ x' \equiv t2.unit\ x'\ x$
notation *t2-unit* $(\eta_{2\ (\text{-})\ (\text{-})})$

abbreviation *t2-counit x x′* ≡ *t2.counit x′ x*
notation *t2-counit* (*ε*$_{2}$ (-) (-))

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* :: (≤$_{L1}$)) ⇛ (≤$_{L2\ x1\ x2}$)

lemma *left-rel-eq-Dep-Fun-Rel*: *L* = ((*x1 x2* :: (≤$_{L1}$)) ⇛ (≤$_{L2\ x1\ x2}$))
  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′* :: (≤$_{R1}$)) ⇛ (≤$_{R2\ x1'\ x2'}$))
  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** $\leq_L$ *50*)
notation R (**infix** $\leq_R$ *50*)

lemma *left-relI* [*intro*]:
assumes $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow f\ x1 \leq_{L2\ x1\ x2} f'\ x2$
shows $f \leq_L f'$
unfolding *left-rel-eq-Dep-Fun-Rel* **using** *assms* **by** *blast*

lemma *left-relE* [*elim*]:
assumes $f \leq_L f'$
and $x1 \leq_{L1} x2$
obtains $f\ x1 \leq_{L2\ x1\ x2} f'\ x2$
using *assms* **unfolding** *left-rel-eq-Dep-Fun-Rel* **by** *blast*

interpretation *flip-inv* :
transport-Dep-Fun-Rel $(\geq_{R1})$ $(\geq_{L1})$ *r1 l1 flip2 R2 flip2 L2 r2 l2* .

lemma *flip-inv-right-eq-ge-left*: *flip-inv.R* $= (\geq_L)$
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* $\equiv (\geq_R)$
unfolding *flip.flip-inv-right-eq-ge-left* .

Useful Rewritings for Dependent Relation **lemma** *left-rel2-unit-eqs-left-rel2I*:
assumes $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow (\leq_{L2\ (\eta_1\ x)\ x}) \leq (\leq_{L2\ x\ x})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow (\leq_{L2\ x\ (\eta_1\ x)}) \leq (\leq_{L2\ x\ x})$
and $x \leq_{L1} x$
and $x \equiv_{L1} \eta_1\ x$
shows $(\leq_{L2\ (\eta_1\ x)\ x}) = (\leq_{L2\ x\ x})$
and $(\leq_{L2\ x\ (\eta_1\ x)}) = (\leq_{L2\ x\ x})$
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** $_L\gtrsim$ *50*)
notation *tdfr.Galois-left* (**infix** $\lesssim_L$ *50*)

notation *tdfr.unit* (η)
notation *tdfr.counit* (ε)

lemma *left-rel-eq-Fun-Rel*: $(\leq_L) = ((\leq_{L1}) \Rrightarrow (\leq_{L2}))$
  by (*urule tdfr.left-rel-eq-Dep-Fun-Rel*)

lemma *left-eq-fun-map*: $l = (r1 \rightsquigarrow l2)$
  by (*intro ext*) *simp*

interpretation *flip* : *transport-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2* **.**

lemma *right-rel-eq-Fun-Rel*: $(\leq_R) = ((\leq_{R1}) \Rrightarrow (\leq_{R2}))$
  unfolding *flip.left-rel-eq-Fun-Rel* **..**

lemma *right-eq-fun-map*: $r = (l1 \rightsquigarrow r2)$
  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* :: $'a1 \Rightarrow 'a1 \Rightarrow$ *bool*
  and *R1* :: $'a2 \Rightarrow 'a2 \Rightarrow$ *bool*
  and *l1* :: $'a1 \Rightarrow 'a2$
  and *r1* :: $'a2 \Rightarrow 'a1$
  and *L2* :: $'a1 \Rightarrow 'a1 \Rightarrow 'b1 \Rightarrow 'b1 \Rightarrow$ *bool*
  and *R2* :: $'a2 \Rightarrow 'a2 \Rightarrow 'b2 \Rightarrow 'b2 \Rightarrow$ *bool*
  and *l2* :: $'a2 \Rightarrow 'a1 \Rightarrow 'b1 \Rightarrow 'b2$
  and *r2* :: $'a1 \Rightarrow 'a2 \Rightarrow 'b2 \Rightarrow 'b1$
begin

definition $L \equiv tdfr.L^{\oplus}$

lemma *left-rel-eq-tdfr-left-Refl-Rel*: $L = tdfr.L^{\oplus}$
  unfolding *L-def* **..**

lemma *left-rel-eq-Mono-Dep-Fun-Rel*: $L = ((x1\ x2 :: (\leq_{L1})) \Rrightarrow^{\oplus} (\leq_{L2\ x1\ x2}))$
  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 $l2 :: 'b1 \Rightarrow 'b2$
and $r2 :: 'b2 \Rightarrow 'b1$
begin

notation *tpdfr.L* (L)
notation *tpdfr.R* (R)

abbreviation $l \equiv$ *tpdfr.l*
abbreviation $r \equiv$ *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*: $l = (r1 \rightsquigarrow l2)$
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*: $r = (l1 \rightsquigarrow r2)$
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*: $((\leq_{R1}) \Rightarrow_m (\leq_{L1}))\ r1$
and *mono-l2*: $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow ((\leq_{L2\ (r1\ x1')\ (r1\ x2')}) \Rightarrow_m (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}))\ (l2_{x2'\ (r1\ x1')})$
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*: $\bigwedge x1'\ x2'\ y.\ x1' \leq_{R1} x2' \Longrightarrow in\text{-}dom\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')})\ y \Longrightarrow (\leq_{R2\ x1'\ x2'})\ (l2_{x2'\ (r1\ x1')}\ y) \leq (\leq_{R2\ x1'\ x2'})\ (l2_{x1'\ (r1\ x1')}\ y)$
and *ge-R2-l2-le2*: $\bigwedge x1'\ x2'\ y.\ x1' \leq_{R1} x2' \Longrightarrow in\text{-}codom\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')})\ y \Longrightarrow (\geq_{R2\ x1'\ x2'})\ (l2_{x2'\ (r1\ x1')}\ y) \leq (\geq_{R2\ x1'\ x2'})\ (l2_{x2'\ (r1\ x2')}\ y)$
shows $((\leq_L) \Rightarrow_m (\leq_R))\ l$
proof (*intro 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 $(in\text{-}dom\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')}) \Rightarrow (\leq_{R2\ x1'\ x2'}))\ (l2_{x1'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x1')})$
and *transitive* $(\leq_{R2\ x1'\ x2'})$
and $x1' \leq_{R1} x2'$
and $in\text{-}dom\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')})\ y$
shows $(\leq_{R2\ x1'\ x2'})\ (l2_{x2'\ (r1\ x1')}\ y) \leq (\leq_{R2\ x1'\ x2'})\ (l2_{x1'\ (r1\ x1')}\ y)$
using *assms* **by** *blast*

lemma *mono-wrt-rel-left-in-codom-mono-left-assm*:
assumes $(in\text{-}codom\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')}) \Rightarrow (\leq_{R2\ x1'\ x2'}))\ (l2_{x2'\ (r1\ x1')})$

$(l2_{x2'\ (r1\ x2')})$
and *transitive* $(\leq_{R2\ x1'\ x2'})$
and $x1' \leq_{R1} x2'$
and *in-codom* $(\leq_{L2\ (r1\ x1')\ (r1\ x2')})\ y$
shows $(\geq_{R2\ x1'\ x2'})\ (l2_{x2'\ (r1\ x1')}\ y) \leq (\geq_{R2\ x1'\ x2'})\ (l2_{x2'\ (r1\ x2')}\ y)$
using *assms* **by** *blast*

lemma *mono-wrt-rel-left-if-transitiveI*:
assumes $((\leq_{R1}) \Rightarrow_m (\leq_{L1}))\ r1$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$((\leq_{L2\ (r1\ x1')\ (r1\ x2')}) \Rightarrow_m (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}))\ (l2_{x2'\ (r1\ x1')})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$(\textit{in-dom}\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')}) \Rrightarrow (\leq_{R2\ x1'\ x2'}))\ (l2_{x1'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x1')})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$(\textit{in-codom}\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')}) \Rrightarrow (\leq_{R2\ x1'\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x2')})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$ *transitive* $(\leq_{R2\ x1'\ x2'})$
shows $((\leq_L) \Rightarrow_m (\leq_R))\ l$
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 $((\leq_{R1}) \Rightarrow_m (\leq_{L1}))\ r1$
and $\bigwedge x\ x'.\ x\ {}_{L1}\!\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \Rightarrow_m (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})$
shows $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$((\leq_{L2\ (r1\ x1')\ (r1\ x2')}) \Rightarrow_m (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}))\ (l2_{x2'\ (r1\ x1')})$
using *assms* **by** (*intro mono-wrt-relI*) *fastforce*

interpretation *flip-inv* :
transport-Dep-Fun-Rel $(\geq_{R1})\ (\geq_{L1})\ r1\ l1\ flip2\ R2\ flip2\ L2\ r2\ l2$
rewrites *flip-inv.R* $\equiv (\geq_L)$ **and** *flip-inv.L* $\equiv (\geq_R)$
and *flip-inv.t1.counit* $\equiv \eta_1$
and $\bigwedge R\ x\ y.\ (flip2\ R\ x\ y)^{-1} \equiv R\ y\ x$
and $\bigwedge R\ x1\ x2.\ in\text{-}dom\ (flip2\ R\ x1\ x2) \equiv in\text{-}codom\ (R\ x2\ x1)$
and $\bigwedge R\ x1\ x2.\ in\text{-}codom\ (flip2\ R\ x1\ x2) \equiv in\text{-}dom\ (R\ x2\ x1)$
and $\bigwedge R\ S.\ (R^{-1} \Rightarrow_m S^{-1}) \equiv (R \Rightarrow_m S)$
and $\bigwedge x1\ x2\ x1'\ x2'.\ (flip2\ R2\ x1'\ x2' \Rightarrow_m flip2\ L2\ x1\ x2) \equiv$
$((\leq_{R2\ x2'\ x1'}) \Rightarrow_m (\leq_{L2\ x2\ x1}))$
and $\bigwedge x1\ x2\ x3\ x4.\ flip2\ L2\ x1\ x2 \leq flip2\ L2\ x3\ x4 \equiv (\leq_{L2\ x2\ x1}) \leq (\leq_{L2\ x4\ x3})$
and $\bigwedge x1'\ x2'\ y1\ y2.$
$flip\text{-}inv.dfro2.right\text{-}infix\ y1\ x1'\ x2' \leq flip\text{-}inv.dfro2.right\text{-}infix\ y2\ x1'\ x2' \equiv$
$(\geq_{L2\ x2'\ x1'})\ y1 \leq (\geq_{L2\ x2'\ x1'})\ y2$
and $\bigwedge(P :: 'f \Rightarrow bool)\ x1\ x2.\ (P \Rrightarrow flip2\ L2\ x1\ x2) \equiv (P \Rrightarrow (\geq_{L2\ x2\ x1}))$
and $\bigwedge(P :: 'f \Rightarrow bool)\ (R :: 'g \Rightarrow 'g \Rightarrow bool).\ (P \Rrightarrow R^{-1}) \equiv (P \Rrightarrow R)^{-1}$
and $\bigwedge x1\ x2.\ transitive\ (flip2\ L2\ x1\ x2) \equiv transitive\ (\leq_{L2\ x2\ x1})$

by (*simp-all add*: *flip-inv-left-eq-ge-right flip-inv-right-eq-ge-left*
t1.flip-counit-eq-unit mono-wrt-rel-eq-dep-mono-wrt-rel Fun-Rel-pred-eq-Dep-Fun-Rel-pred
del: *rel-inv-iff-rel*)

lemma *mono-wrt-rel-rightI*:
assumes $((\leq_{L1}) \Rightarrow_m (\leq_{R1}))$ *l1*
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow ((\leq_{R2\ (l1\ x1)\ (l1\ x2)}) \Rightarrow_m (\leq_{L2\ x1\ (\eta_1\ x2)}))\ (r2_{x1\ (l1\ x2)})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2\ y'.\ x1 \leq_{L1} x2 \Longrightarrow$ *in-codom* $(\leq_{R2\ (l1\ x1)\ (l1\ x2)})\ y' \Longrightarrow$
$(\geq_{L2\ x1\ x2})\ (r2_{x1\ (l1\ x2)}\ y') \leq (\geq_{L2\ x1\ x2})\ (r2_{x2\ (l1\ x2)}\ y')$
and $\bigwedge x1\ x2\ y'.\ x1 \leq_{L1} x2 \Longrightarrow$ *in-dom* $(\leq_{R2\ (l1\ x1)\ (l1\ x2)})\ y' \Longrightarrow$
$(\leq_{L2\ x1\ x2})\ (r2_{x1\ (l1\ x2)}\ y') \leq (\leq_{L2\ x1\ x2})\ (r2_{x1\ (l1\ x1)}\ y')$
shows $((\leq_R) \Rightarrow_m (\leq_L))\ r$
using *assms* **by** (*intro flip-inv.mono-wrt-rel-leftI*[*simplified rel-inv-iff-rel*])

lemma *mono-wrt-rel-right-if-transitiveI*:
assumes $((\leq_{L1}) \Rightarrow_m (\leq_{R1}))$ *l1*
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow ((\leq_{R2\ (l1\ x1)\ (l1\ x2)}) \Rightarrow_m (\leq_{L2\ x1\ (\eta_1\ x2)}))\ (r2_{x1\ (l1\ x2)})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ (*in-codom* $(\leq_{R2\ (l1\ x1)\ (l1\ x2)}) \Rrightarrow (\leq_{L2\ x1\ x2})$)
$(r2_{x1\ (l1\ x2)})\ (r2_{x2\ (l1\ x2)})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ (*in-dom* $(\leq_{R2\ (l1\ x1)\ (l1\ x2)}) \Rrightarrow (\leq_{L2\ x1\ x2})$)
$(r2_{x1\ (l1\ x1)})\ (r2_{x1\ (l1\ x2)})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
shows $((\leq_R) \Rightarrow_m (\leq_L))\ r$
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*: $((\leq_{L1}) \Rightarrow_m (\leq_{R1}))$ *l1* $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*
and *mono-r2*: $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rightarrow_m (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
shows $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow ((\leq_{R2\ (l1\ x1)\ (l1\ x2)}) \Rightarrow_m (\leq_{L2\ x1\ (\eta_1\ x2)}))$
$(r2_{x1\ (l1\ x2)})$
proof −
show $((\leq_{R2\ (l1\ x1)\ (l1\ x2)}) \Rightarrow_m (\leq_{L2\ x1\ (\eta_1\ x2)}))\ (r2_{x1\ (l1\ x2)})$ **if** $x1 \leq_{L1} x2$
for *x1 x2*
proof −
from ‹$x1 \leq_{L1} x2$› **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 $((\leq_{R1}) \Rightarrow_m (\leq_{L1}))$ *r1*
and $((\leq_{L2}) \Rightarrow_m (\leq_{R2}))$ *l2*
shows $((\leq_L) \Rightarrow_m (\leq_R))$ *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*: $((\leq_{R1}) \Rightarrow_m (\leq_{L1}))$ *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 $((in\text{-}codom\ (\leq_{R2\ (\varepsilon_1\ x')\ x'}))) \Rrightarrow (\leq_{L2\ (r1\ x')\ (r1\ x')}))$
$(r2_{(r1\ x')\ (\varepsilon_1\ x')})\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $x' \leq_{R1} x'$
and *in-codom* $(\leq_{R2\ (\varepsilon_1\ x')\ x'})\ y'$
shows $(\geq_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')}\ y') \leq (\geq_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ x'}$

y')
using *mono-wrt-relD*[*OF mono-r1*] *assms*(2−) **by** *blast*

interpretation *flip-inv* :
transport-Dep-Fun-Rel $(\geq_{R1})$ $(\geq_{L1})$ *r1 l1 flip2 R2 flip2 L2 r2 l2*
rewrites *flip-inv.L* $\equiv (\geq_R)$ **and** *flip-inv.R* $\equiv (\geq_L)$
and *flip-inv.t1.counit* $\equiv \eta_1$
and $\bigwedge R\ x\ y.\ (\mathit{flip2}\ R\ x\ y)^{-1} \equiv R\ y\ x$
and $\bigwedge R.$ *in-dom* $R^{-1} \equiv$ *in-codom* R
and $\bigwedge R\ x1\ x2.$ *in-codom* $(\mathit{flip2}\ R\ x1\ x2) \equiv$ *in-dom* $(R\ x2\ x1)$
and $\bigwedge R\ S.\ (R^{-1} \Rrightarrow_m S^{-1}) \equiv (R \Rrightarrow_m S)$
and $\bigwedge R\ S\ x1\ x2\ x1'\ x2'.\ (\mathit{flip2}\ R\ x1\ x2\ {}_h\trianglelefteq\ \mathit{flip2}\ S\ x1'\ x2') \equiv (S\ x2'\ x1' \trianglelefteq_h R\ x2\ x1)^{-1}$
and $\bigwedge R\ S.\ (R^{-1}\ {}_h\trianglelefteq\ S^{-1}) \equiv (S \trianglelefteq_h R)^{-1}$
and $\bigwedge x1\ x2\ x3\ x4.\ \mathit{flip2}\ L2\ x1\ x2 \leq \mathit{flip2}\ L2\ x3\ x4 \equiv (\leq_{L2\ x2\ x1}) \leq (\leq_{L2\ x4\ x3})$
and $\bigwedge(R :: 'z \Rightarrow 'z \Rightarrow \mathit{bool})\ (P :: 'z \Rightarrow \mathit{bool}).\ \mathit{reflexive\text{-}on}\ P\ R^{-1} \equiv \mathit{reflexive\text{-}on}\ P\ R$
and $\bigwedge R\ x1\ x2.\ \mathit{transitive}\ (\mathit{flip2}\ R\ x1\ x2 :: 'z \Rightarrow 'z \Rightarrow \mathit{bool}) \equiv \mathit{transitive}\ (R\ x2\ x1)$
and $\bigwedge x\ x.\ ((\mathit{in\text{-}dom}\ (\leq_{L2\ x'\ (\eta_1\ x')})) \Rrightarrow \mathit{flip2}\ R2\ (l1\ x')\ (l1\ x'))$
$\equiv ((\mathit{in\text{-}dom}\ (\leq_{L2\ x'\ (\eta_1\ x')})) \Rrightarrow (\leq_{R2\ (l1\ x')\ (l1\ x')}))^{-1}$
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
mono-wrt-rel-eq-dep-mono-wrt-rel Fun-Rel-pred-eq-Dep-Fun-Rel-pred)

lemma *left-rel-right-if-left-right-relI*:
assumes $((\leq_{L1}) \Rrightarrow_m (\leq_{R1}))$ *l1*
and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*
and *reflexive-on* $(\mathit{in\text{-}codom}\ (\leq_{L1}))\ (\leq_{L1})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow ((\leq_{L2\ x\ (\eta_1\ x)}) \trianglelefteq_h (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (r2_{x\ (l1\ x)})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow (\leq_{L2\ x\ (\eta_1\ x)}) \leq (\leq_{L2\ x\ x})$
and $\bigwedge x\ y.\ x \leq_{L1} x \Longrightarrow \mathit{in\text{-}dom}\ (\leq_{L2\ x\ (\eta_1\ x)})\ y \Longrightarrow$
$(\leq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ (\eta_1\ x)}\ y) \leq (\leq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ x}\ y)$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow \mathit{transitive}\ (\leq_{L2\ x1\ x2})$
and $f \leq_L f$
and $l\ f \leq_R g$
shows $f \leq_L r\ g$
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 $((\leq_{L1}) \Rrightarrow_m (\leq_{R1}))$ *l1*
and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))^{-1}$ *r1 l1*
and $((\mathit{in\text{-}dom}\ (\leq_{L2\ x\ (\eta_1\ x)})) \Rrightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (l2_{(l1\ x)\ (\eta_1\ x)})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow \mathit{transitive}\ (\leq_{R2\ x1'\ x2'})$
and $x \leq_{L1} x$

and *in-dom* $(\leq_{L2\ x\ (\eta_1\ x)})\ y$
shows $(\leq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ (\eta_1\ x)}\ y) \leq (\leq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ x}\ y)$
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 $((\leq_{L1}) \dashv (\leq_{R1}))\ l1\ r1$
and *reflexive-on* $(in\text{-}codom\ (\leq_{L1}))\ (\leq_{L1})$
and *reflexive-on* $(in\text{-}dom\ (\leq_{R1}))\ (\leq_{R1})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$
$((\leq_{L2\ (r1\ x')\ (r1\ x')})\ {}_h\trianglelefteq\ (\leq_{R2\ (\varepsilon_1\ x')\ x'}))\ (l2_{\ x'\ (r1\ x')})\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow ((\leq_{L2\ x\ (\eta_1\ x)}) \trianglelefteq_h (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (r2_{x\ (l1\ x)})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow (\leq_{L2\ x\ (\eta_1\ x)}) \leq (\leq_{L2\ x\ x})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x')\ x'}) \leq (\leq_{R2\ x'\ x'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x\ y.\ x \leq_{L1} x \Longrightarrow in\text{-}dom\ (\leq_{L2\ x\ (\eta_1\ x)})\ y \Longrightarrow$
$(\leq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ (\eta_1\ x)}\ y) \leq (\leq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ x}\ y)$
and $\bigwedge x'\ y'.\ x' \leq_{R1} x' \Longrightarrow in\text{-}codom\ (\leq_{R2\ (\varepsilon_1\ x')\ x'})\ y' \Longrightarrow$
$(\geq_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')}\ y') \leq (\geq_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ x'}$
$y')$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow transitive\ (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow transitive\ (\leq_{R2\ x1'\ x2'})$
and $f \leq_L f$
and $g \leq_R g$
shows $f \leq_L r\ g \longleftrightarrow l\ f \leq_R g$
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 $((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))\ r1$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')})\ {}_h\trianglelefteq\ (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $x' \leq_{R1} x'$
shows $((\leq_{L2\ (r1\ x')\ (r1\ x')})\ {}_h\trianglelefteq\ (\leq_{R2\ (\varepsilon_1\ x')\ x'}))\ (l2_{\ x'\ (r1\ x')})\ (r2_{(r1\ x')\ x'})$
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 $((\leq_{L1}) \Rrightarrow_m (\leq_{R1}))\ l1$
and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))\ l1\ r1$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \trianglelefteq_h (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $x \leq_{L1} x$
shows $((\leq_{L2\ x\ (\eta_1\ x)}) \trianglelefteq_h (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (r2_{x\ (l1\ x)})$
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 $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$
$((in\text{-}dom\ (\leq_{L2\ x\ (\eta_1\ x)})) \Rrightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (l2_{(l1\ x)\ (\eta_1\ x)})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$
$((in\text{-}codom\ (\leq_{R2\ (\varepsilon_1\ x')\ x'})) \Rrightarrow (\leq_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow transitive\ (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow transitive\ (\leq_{R2\ x1'\ x2'})$
and $f \leq_L f$
and $g \leq_R g$
shows $f \leq_L r\ g \longleftrightarrow l\ f \leq_R g$
proof −
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 $\langle x\ {}_{L1}\lessapprox\ x' \rangle$ **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** $x2 \leq_{L1} x2\ \eta_1\ x2 \leq_{L1} \eta_1\ x2$ **by** *auto*

moreover note *dep-mono-wrt-relD*[*OF antimono-L2* ‹*x1* $\leq_{L1}$ *x2*›]
and *dep-mono-wrt-relD*[*OF antimono-L2* ‹*x1* $\leq_{L1}$ *x1*›]
ultimately show $(\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})\ (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
using ‹*x1* $\leq_{L1}$ *x2*› **by** *auto*
qed

lemma *left-rel-right-iff-left-right-rel-if-galois-prop-le-assms-rightI*:
assumes *galois-conn1*: $((\leq_{L1}) \dashv (\leq_{R1}))$ *l1 r1*
and *refl-R1*: *reflexive-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and *mono-R2*:
$((x1'\ x2' :: (\leq_{R1}) \mid \varepsilon_1\ x2' \leq_{R1} x1') \Rrightarrow_m (x3'\ x4' :: (\leq_{R1}) \mid x2' \leq_{R1} x3') \Rrightarrow (\leq))\ R2$
shows $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$
proof −
fix *x1' x2'* **assume** $x1' \leq_{R1} x2'$
with *galois-conn1 refl-R1* **have** $x2' \leq_{R1} x2'\ \varepsilon_1\ x1' \leq_{R1} x1'$
by (*blast intro*:
t1.counit-rel-if-right-rel-if-half-galois-prop-left-if-mono-wrt-rel)+
moreover with *refl-R1* **have** $x1' \leq_{R1} x1'\ \varepsilon_1\ x1' \leq_{R1} \varepsilon_1\ x1'$ **by** *auto*
moreover note *dep-mono-wrt-relD*[*OF mono-R2* ‹$\varepsilon_1\ x1' \leq_{R1} x1'$›]
and *dep-mono-wrt-relD*[*OF mono-R2* ‹$x1' \leq_{R1} x1'$›]
ultimately show $(\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})\ (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$
using ‹$x1' \leq_{R1} x2'$› **by** *auto*
qed

corollary *left-rel-right-iff-left-right-rel-if-monoI*:
assumes $((\leq_{L1}) \dashv (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and *reflexive-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\overset{\leq}{\approx}\ x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \trianglelefteq (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
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 $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$
$((\textit{in-dom}\ (\leq_{L2\ x\ (\eta_1\ x)})) \Rrightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (l2_{(l1\ x)\ (\eta_1\ x)})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$
$((\textit{in-codom}\ (\leq_{R2\ (\varepsilon_1\ x')\ x'})) \Rrightarrow (\leq_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow \textit{transitive}\ (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow \textit{transitive}\ (\leq_{R2\ x1'\ x2'})$
and $f \leq_L f$
and $g \leq_R g$
shows $f \leq_L r\ g \longleftrightarrow l\ f \leq_R g$
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 $((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))$ *r1*
and $((\leq_{L1}) \ {}_h\trianglelefteq (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-dom* $(\leq_{R1})$) $(\leq_{R1})$
and $((\leq_{L2}) \ {}_h\trianglelefteq (\leq_{R2}))$ *l2 r2*
and *transitive* $(\leq_{R2})$
and $g \leq_R g$
and $f \leq_L r\ g$
shows $l\ f \leq_R g$
using *assms* **by** (*intro tdfr.left-right-rel-if-left-rel-rightI*) *simp-all*

corollary *left-rel-right-if-left-right-relI*:
assumes $((\leq_{L1}) \Rrightarrow_m (\leq_{R1}))$ *l1*
and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-codom* $(\leq_{L1})$) $(\leq_{L1})$
and $((\leq_{L2}) \trianglelefteq_h (\leq_{R2}))$ *l2 r2*
and *transitive* $(\leq_{L2})$
and $f \leq_L f$
and $l\ f \leq_R g$
shows $f \leq_L r\ g$
using *assms* **by** (*intro tdfr.left-rel-right-if-left-right-relI*) *simp-all*

corollary *left-rel-right-iff-left-right-relI*:
assumes $((\leq_{L1}) \dashv (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-codom* $(\leq_{L1})$) $(\leq_{L1})$
and *reflexive-on* (*in-dom* $(\leq_{R1})$) $(\leq_{R1})$
and $((\leq_{L2}) \trianglelefteq (\leq_{R2}))$ *l2 r2*
and *transitive* $(\leq_{L2})$
and *transitive* $(\leq_{R2})$
and $f \leq_L f$
and $g \leq_R g$
shows $f \leq_L r\ g \longleftrightarrow l\ f \leq_R g$
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 $((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))$ *r1*

and $((\leq_{L1})\ {}_h\trianglelefteq\ (\leq_{R1}))\ l1\ r1$
and *reflexive-on* (*in-dom* $(\leq_{R1})$) $(\leq_{R1})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$
$((\leq_{L2}\ (r1\ x')\ (r1\ x'))\ {}_h\trianglelefteq\ (\leq_{R2}\ (\varepsilon_1\ x')\ x'))\ (l2\ x'\ (r1\ x'))\ (r2\ (r1\ x')\ x')$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow (\leq_{R2}\ (\varepsilon_1\ x')\ x') \leq (\leq_{R2\ x'\ x'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x'\ y'.\ x' \leq_{R1} x' \Longrightarrow$ *in-codom* $(\leq_{R2}\ (\varepsilon_1\ x')\ x')\ y' \Longrightarrow$
$(\geq_{L2}\ (r1\ x')\ (r1\ x'))\ (r2\ (r1\ x')\ (\varepsilon_1\ x')\ y') \leq (\geq_{L2}\ (r1\ x')\ (r1\ x'))\ (r2\ (r1\ x')\ x'\ y')$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$ *transitive* $(\leq_{R2\ x1'\ x2'})$
shows $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$
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 $((\leq_{L1}) \Rrightarrow_m (\leq_{R1}))\ l1$
and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))\ l1\ r1$
and *reflexive-on* (*in-codom* $(\leq_{L1})$) $(\leq_{L1})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow ((\leq_{L2\ x\ (\eta_1\ x)}) \trianglelefteq_h (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2\ (l1\ x)\ x)\ (r2\ x\ (l1\ x))$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow (\leq_{L2\ x\ (\eta_1\ x)}) \leq (\leq_{L2\ x\ x})$
and $\bigwedge x\ y.\ x \leq_{L1} x \Longrightarrow$ *in-dom* $(\leq_{L2\ x\ (\eta_1\ x)})\ y \Longrightarrow$
$(\leq_{R2\ (l1\ x)\ (l1\ x)})\ (l2\ (l1\ x)\ (\eta_1\ x)\ y) \leq (\leq_{R2\ (l1\ x)\ (l1\ x)})\ (l2\ (l1\ x)\ x\ y)$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
shows $((\leq_L) \trianglelefteq_h (\leq_R))\ l\ r$
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 $((\leq_{L1}) \dashv (\leq_{R1}))\ l1\ r1$
and *reflexive-on* (*in-codom* $(\leq_{L1})$) $(\leq_{L1})$
and *reflexive-on* (*in-dom* $(\leq_{R1})$) $(\leq_{R1})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$
$((\leq_{L2}\ (r1\ x')\ (r1\ x'))\ {}_h\trianglelefteq\ (\leq_{R2}\ (\varepsilon_1\ x')\ x'))\ (l2\ x'\ (r1\ x'))\ (r2\ (r1\ x')\ x')$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow ((\leq_{L2\ x\ (\eta_1\ x)}) \trianglelefteq_h (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2\ (l1\ x)\ x)\ (r2\ x\ (l1\ x))$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow (\leq_{L2\ x\ (\eta_1\ x)}) \leq (\leq_{L2\ x\ x})$

and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x')\ x'}) \leq (\leq_{R2\ x'\ x'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x\ y.\ x \leq_{L1} x \Longrightarrow$ *in-dom* $(\leq_{L2\ x\ (\eta_1\ x)})\ y \Longrightarrow$
$(\leq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ (\eta_1\ x)}\ y) \leq (\leq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ x}\ y)$
and $\bigwedge x'\ y'.\ x' \leq_{R1} x' \Longrightarrow$ *in-codom* $(\leq_{R2\ (\varepsilon_1\ x')\ x'})\ y' \Longrightarrow$
$(\geq_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')}\ y') \leq (\geq_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ x'}\ y')$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$ *transitive* $(\leq_{R2\ x1'\ x2'})$
shows $((\leq_L) \trianglelefteq (\leq_R))\ l\ r$
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 $((\leq_{L1}) \dashv (\leq_{R1}))\ l1\ r1$
and *reflexive-on* $(in\text{-}codom\ (\leq_{L1}))\ (\leq_{L1})$
and *reflexive-on* $(in\text{-}dom\ (\leq_{R1}))\ (\leq_{R1})$
and *galois-prop2*: $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow$
$((\leq_{L2\ x\ (r1\ x')}) \trianglelefteq (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow (\leq_{L2\ x\ (\eta_1\ x)}) \leq (\leq_{L2\ x\ x})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x')\ x'}) \leq (\leq_{R2\ x'\ x'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$
$((in\text{-}dom\ (\leq_{L2\ x\ (\eta_1\ x)})) \Rrightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (l2_{(l1\ x)\ (\eta_1\ x)})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$
$((in\text{-}codom\ (\leq_{R2\ (\varepsilon_1\ x')\ x'})) \Rrightarrow (\leq_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$ *transitive* $(\leq_{R2\ x1'\ x2'})$
shows $((\leq_L) \trianglelefteq (\leq_R))\ l\ r$
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 $\langle x\ {}_{L1}\lessapprox x' \rangle$ **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 $((\leq_{L1}) \dashv (\leq_{R1}))\ l1\ r1$
and *reflexive-on* $(in\text{-}field\ (\leq_{L1}))\ (\leq_{L1})$
and *reflexive-on* $(in\text{-}field\ (\leq_{R1}))\ (\leq_{R1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \trianglelefteq (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
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 $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$
$((in\text{-}dom\ (\leq_{L2\ x\ (\eta_1\ x)})) \Rrightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (l2_{(l1\ x)\ (\eta_1\ x)})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$
$((in\text{-}codom\ (\leq_{R2\ (\varepsilon_1\ x')\ x'})) \Rrightarrow (\leq_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow transitive\ (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow transitive\ (\leq_{R2\ x1'\ x2'})$
shows $((\leq_L) \trianglelefteq (\leq_R))\ l\ r$
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 $[\![(tdfr.L \Rrightarrow_m tdfr.R)\ l;\ (tdfr.R \Rrightarrow_m tdfr.L)\ r;\ t1.galois\text{-}connection;\ reflexive\text{-}on\ (in\text{-}field\ (\leq_{L1}))\ (\leq_{L1});\ reflexive\text{-}on\ (in\text{-}field\ (\leq_{R1}))\ (\leq_{R1});\ \bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow t2.galois\text{-}prop\ x\ x'\ l2_{x'\ x}\ r2_{x\ x'};\ ((x1\ x2 :: (\leq_{L1})) \Rrightarrow_m (x3\ x4 :: (\leq_{L1})) \Rrightarrow (x2 \leq_{L1} x3 \wedge x4 \leq_{L1} \eta_1\ x3) \longrightarrow (\lambda x\ y.\ y \leq x))\ L2;\ ((x1'\ x2' :: (\leq_{R1})) \Rrightarrow_m \varepsilon_1\ x2' \leq_{R1} x1' \longrightarrow ((x3'\ x4' :: (\leq_{R1})) \Rrightarrow x2' \leq_{R1} x3' \longrightarrow (\leq)))\ R2;\ \bigwedge x.\ x \leq_{L1} x \Longrightarrow (in\text{-}dom\ (\leq_{L2\ x\ \eta_1\ x}) \Rrightarrow \leq_{R2\ l1\ x\ l1\ x})\ l2_{l1\ x\ x}\ l2_{l1\ x\ \eta_1\ x};\ \bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow (in\text{-}codom\ (\leq_{R2\ \varepsilon_1\ x'\ x'}) \Rrightarrow \leq_{L2\ r1\ x'\ r1\ x'})\ r2_{r1\ x'\ \varepsilon_1\ x'}\ r2_{r1\ x'\ x'};\ \bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow transitive\ (\leq_{L2\ x1\ x2});\ \bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow transitive\ (\leq_{R2\ x1'\ x2'})]\!] \Longrightarrow galois\text{-}prop\ l\ r$, as we will do later for Galois connections, by applying $[\![((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))\ r1;\ \bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{L2\ r1\ x1'\ r1\ x2'} \Rrightarrow_m \leq_{R2\ \varepsilon_1\ x1'\ x2'})\ l2_{x2'\ r1\ x1'};\ \bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ \varepsilon_1\ x1'\ x2'}) \leq (\leq_{R2\ x1'\ x2'});\ \bigwedge x1'\ x2'\ y.\ [\![x1' \leq_{R1} x2';\ in\text{-}dom\ (\leq_{L2\ r1\ x1'\ r1\ x2'})\ y]\!] \Longrightarrow dfro2.right\text{-}infix\ (l2_{x2'\ r1\ x1'}\ y)\ x1'\ x2' \leq dfro2.right\text{-}infix\ (l2_{x1'\ r1\ x1'}\ y)\ x1'\ x2';\ \bigwedge x1'\ x2'\ y.\ [\![x1' \leq_{R1} x2';\ in\text{-}codom\ (\leq_{L2\ r1\ x1'\ r1\ x2'})\ y]\!] \Longrightarrow (\leq_{R2\ x1'\ x2'})^{-1}\ (l2_{x2'\ r1\ x1'}\ y) \leq (\leq_{R2\ x1'\ x2'})^{-1}\ (l2_{x2'\ r1\ x2'}\ y)]\!] \Longrightarrow (tdfr.L \Rrightarrow_m tdfr.R)\ l$ and $[\![((\leq_{L1}) \Rrightarrow_m (\leq_{R1}))\ l1;\ \bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{R2\ l1\ x1\ l1\ x2} \Rrightarrow_m \leq_{L2\ x1\ \eta_1\ x2})\ r2_{x1\ l1\ x2};\ \bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ \eta_1\ x2}) \leq (\leq_{L2\ x1\ x2});\ \bigwedge x1\ x2\ y'.\ [\![x1 \leq_{L1} x2;\ in\text{-}codom\ (\leq_{R2\ l1\ x1\ l1\ x2})\ y']\!] \Longrightarrow (\leq_{L2\ x1\ x2})^{-1}\ (r2_{x1\ l1\ x2}\ y') \leq (\leq_{L2\ x1\ x2})^{-1}\ (r2_{x2\ l1\ x2}\ y');\ \bigwedge x1\ x2\ y'.\ [\![x1 \leq_{L1} x2;\ in\text{-}dom\ (\leq_{R2\ l1\ x1\ l1\ x2})\ y']\!] \Longrightarrow dfro1.right\text{-}infix\ (r2_{x1\ l1\ x2}\ y')\ x1\ x2 \leq dfro1.right\text{-}infix\ (r2_{x1\ l1\ x1}\ y')\ x1\ x2]\!] \Longrightarrow (tdfr.R \Rrightarrow_m tdfr.L)\ r$ 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 $[\![((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))\ r1;$ $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{L2\ r1\ x1'\ r1\ x2'} \Rrightarrow_m \leq_{R2\ \varepsilon_1\ x1'\ x2'})\ l2_{x2'\ r1\ x1'};$ $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ \varepsilon_1\ x1'\ x2'}) \leq (\leq_{R2\ x1'\ x2'});\ \bigwedge x1'\ x2'\ y.$ $[\![x1' \leq_{R1} x2';$ *in-dom* $(\leq_{L2\ r1\ x1'\ r1\ x2'})\ y]\!] \Longrightarrow$ *dfro2.right-infix* $(l2_{x2'\ r1\ x1'}$ $y)\ x1'\ x2' \leq$ *dfro2.right-infix* $(l2_{x1'\ r1\ x1'}\ y)\ x1'\ x2';\ \bigwedge x1'\ x2'\ y.\ [\![x1'$ $\leq_{R1}\ x2';$ *in-codom* $(\leq_{L2\ r1\ x1'\ r1\ x2'})\ y]\!] \Longrightarrow (\leq_{R2\ x1'\ x2'})^{-1}\ (l2_{x2'\ r1\ x1'}$ $y) \leq (\leq_{R2\ x1'\ x2'})^{-1}\ (l2_{x2'\ r1\ x2'}\ y)]\!] \Longrightarrow$ $(tdfr.L \Rrightarrow_m tdfr.R)\ l$ and $[\![((\leq_{L1})$ $\Rrightarrow_m (\leq_{R1}))\ l1;\ \bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{R2\ l1\ x1\ l1\ x2} \Rrightarrow_m \leq_{L2\ x1\ \eta_1\ x2})$ $r2_{x1\ l1\ x2};\ \bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ \eta_1\ x2}) \leq (\leq_{L2\ x1\ x2});\ \bigwedge x1\ x2\ y'.$ $[\![x1 \leq_{L1} x2;$ *in-codom* $(\leq_{R2\ l1\ x1\ l1\ x2})\ y']\!] \Longrightarrow (\leq_{L2\ x1\ x2})^{-1}\ (r2_{x1\ l1\ x2}\ y')$ $\leq (\leq_{L2\ x1\ x2})^{-1}\ (r2_{x2\ l1\ x2}\ y');\ \bigwedge x1\ x2\ y'.\ [\![x1 \leq_{L1} x2;$ *in-dom* $(\leq_{R2\ l1\ x1\ l1\ x2})$ $y']\!] \Longrightarrow$ *dfro1.right-infix* $(r2_{x1\ l1\ x2}\ y')\ x1\ x2 \leq$ *dfro1.right-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 $((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))\ r1$
and $((\leq_{L1})\ {}_h\trianglelefteq\ (\leq_{R1}))\ l1\ r1$
and *reflexive-on* $(in\text{-}dom\ (\leq_{R1}))\ (\leq_{R1})$
and $((\leq_{L2}) \Rrightarrow_m (\leq_{R2}))\ l2$
and $((\leq_{L2})\ {}_h\trianglelefteq\ (\leq_{R2}))\ l2\ r2$
and *transitive* $(\leq_{R2})$
shows $((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))\ l\ r$
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 $((\leq_{L1}) \Rrightarrow_m (\leq_{R1}))\ l1$
and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))\ l1\ r1$
and *reflexive-on* $(in\text{-}codom\ (\leq_{L1}))\ (\leq_{L1})$
and $((\leq_{R2}) \Rrightarrow_m (\leq_{L2}))\ r2$
and $((\leq_{L2}) \trianglelefteq_h (\leq_{R2}))\ l2\ r2$
and *transitive* $(\leq_{L2})$
shows $((\leq_L) \trianglelefteq_h (\leq_R))\ l\ r$
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 $((\leq_{L1}) \dashv (\leq_{R1}))\ l1\ r1$
and *reflexive-on* $(in\text{-}codom\ (\leq_{L1}))\ (\leq_{L1})$
and *reflexive-on* $(in\text{-}dom\ (\leq_{R1}))\ (\leq_{R1})$

and $((\leq_{L2}) \dashv (\leq_{R2}))$ *l2 r2*
and *transitive* $(\leq_{L2})$
and *transitive* $(\leq_{R2})$
shows $((\leq_L) \trianglelefteq (\leq_R))$ *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*: $((\leq_{L1}) \dashv (\leq_{R1}))$ *l1 r1*
and *refl-R1*: *reflexive-on* $(in\text{-}codom\ (\leq_{R1}))\ (\leq_{R1})$
and *R2-le1*: $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and *mono-l2-2*: $((x' : in\text{-}codom\ (\leq_{R1})) \Rrightarrow_m (x1\ x2 :: (\leq_{L1}) \mid x2\ _{L1}\lessapprox x') \Rrightarrow_m$
$(in\text{-}field\ (\leq_{L2\ x1\ (r1\ x')})) \Rrightarrow (\leq_{R2\ (l1\ x1)\ x'}))\ l2$
shows $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$((in\text{-}codom\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')})) \Rrightarrow (\leq_{R2\ x1'\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x2')})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$
$((in\text{-}dom\ (\leq_{L2\ x\ (\eta_1\ x)})) \Rrightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (l2_{(l1\ x)\ (\eta_1\ x)})$
proof −
show $((in\text{-}codom\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')})) \Rrightarrow (\leq_{R2\ x1'\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x2')})$
if $x1' \leq_{R1} x2'$ **for** $x1'\ x2'$
proof −
from *galois-conn1* $\langle x1' \leq_{R1} x2' \rangle$ **have** $r1\ x1' \leq_{L1} r1\ x2'\ r1\ x2'\ _{L1}\lessapprox x2'$
using *refl-R1* **by** (*auto intro*: *t1.right-left-Galois-if-reflexive-onI*)
with *mono-l2-2* **show** *?thesis* **using** *R2-le1* $\langle x1' \leq_{R1} x2' \rangle$ **by** *fastforce*
qed
show $((in\text{-}dom\ (\leq_{L2\ x\ (\eta_1\ x)})) \Rrightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (l2_{(l1\ x)\ (\eta_1\ x)})$
if $x \leq_{L1} x$ **for** x
proof −
from *galois-conn1* $\langle x \leq_{L1} x \rangle$ **have** $x \leq_{L1} \eta_1\ x\ \eta_1\ x\ _{L1}\lessapprox l1\ x$
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: ((≤L1) ⊣ (≤R1)) l1 r1
  and refl-R1: reflexive-on (in-field (≤R1)) (≤R1)
  and R2-le1: ⋀x1' x2'. x1' ≤R1 x2' ⟹ (≤R2 (ε1 x1') x2') ≤ (≤R2 x1' x2')
  and mono-l2: ((x1' x2' :: (≤R1)) ⇛m (x1 x2 :: (≤L1) | x2 L1≲≈ x1') ⇛
    in-field (≤L2 x1 (r1 x2')) ⇛ (≤R2 (l1 x1) x2')) l2
  shows ⋀x1' x2'. x1' ≤R1 x2' ⟹
   ((in-dom (≤L2 (r1 x1') (r1 x2'))) ⇛ (≤R2 x1' x2')) (l2x1' (r1 x1')) (l2x2' (r1 x1'))
  and ((x' : in-codom (≤R1)) ⇛m (x1 x2 :: (≤L1) | x2 L1≲≈ x') ⇛m
    (in-field (≤L2 x1 (r1 x')) ⇛ (≤R2 (l1 x1) x')) l2
proof −
  show ((in-dom (≤L2 (r1 x1') (r1 x2'))) ⇛ (≤R2 x1' x2')) (l2x1' (r1 x1')) (l2x2' (r1 x1'))
    if x1' ≤R1 x2' for x1' x2'
  proof −
    from galois-conn1 ‹x1' ≤R1 x2'› have r1 x1' ≤L1 r1 x1' r1 x1' L1≲≈ x1'
      using refl-R1 by force+
    with mono-l2 show ?thesis using ‹x1' ≤R1 x2'› R2-le1 by (auto 11 0)
  qed
  from mono-l2 show ((x' : in-codom (≤R1)) ⇛m (x1 x2 :: (≤L1) | x2 L1≲≈ x')
⇛m
    (in-field (≤L2 x1 (r1 x')) ⇛ (≤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: ((≤L1) ⊣ (≤R1)) l1 r1
  and refl-L1: reflexive-on (in-dom (≤L1)) (≤L1)
  and L2-le2: ⋀x1 x2. x1 ≤L1 x2 ⟹ (≤L2 x1 (η1 x2)) ≤ (≤L2 x1 x2)
  and mono-r2-2: ((x : in-dom (≤L1)) ⇛m (x1' x2' :: (≤R1) | x L1≲≈ x1') ⇛m
    (in-field (≤R2 (l1 x) x2')) ⇛ (≤L2 x (r1 x2'))) r2
  shows ⋀x1 x2. x1 ≤L1 x2 ⟹
    ((in-dom (≤R2 (l1 x1) (l1 x2))) ⇛ (≤L2 x1 x2)) (r2x1 (l1 x1)) (r2x1 (l1 x2))
  and ⋀x'. x' ≤R1 x' ⟹
   ((in-codom (≤R2 (ε1 x') x')) ⇛ (≤L2 (r1 x') (r1 x'))) (r2(r1 x') (ε1 x')) (r2(r1 x') x')
proof −
  show ((in-dom (≤R2 (l1 x1) (l1 x2))) ⇛ (≤L2 x1 x2)) (r2x1 (l1 x1)) (r2x1 (l1 x2))
    if x1 ≤L1 x2 for x1 x2
  proof −
    from galois-conn1 ‹x1 ≤L1 x2› have x1 L1≲≈ l1 x1 l1 x1 ≤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* ‹$x1 \leq_{L1} x2$› **by** (*auto 12 0*)
qed
show ((*in-codom* ($\leq_{R2}$ (ε_1 x') x')) $\Rrightarrow$ ($\leq_{L2}$ (*r1* x') (*r1* x'))) ($r2_{(r1\ x')\ (\varepsilon_1\ x')}$) ($r2_{(r1\ x')\ x'}$)
if $x' \leq_{R1} x'$ **for** x'
proof −
from *galois-conn1* ‹$x' \leq_{R1} x'$› **have** *r1* x' $_{L1}\lessapprox$ ε_1 x' ε_1 x' $\leq_{R1}$ 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*: ((≤$_{L1}$) ⊣ (≤$_{R1}$)) *l1 r1*
and *refl-L1*: *reflexive-on* (*in-field* (≤$_{L1}$)) (≤$_{L1}$)
and *L2-le2*: $\bigwedge$*x1 x2*. *x1* ≤$_{L1}$ *x2* ⟹ (≤$_{L2\ x1\ (\eta_1\ x2)}$) ≤ (≤$_{L2\ x1\ x2}$)
and *mono-r2*: ((*x1 x2* :: (≤$_{L1}$)) $\Rrightarrow_m$ (*x1'* *x2'* :: (≤$_{R1}$) | *x2* $_{L1}\lessapprox$ *x1'*) $\Rrightarrow$
in-field (≤$_{R2\ (l1\ x1)\ x2'}$) $\Rrightarrow$ (≤$_{L2\ x1\ (r1\ x2')}$)) *r2*
shows $\bigwedge$*x1 x2*. *x1* ≤$_{L1}$ *x2* ⟹
((*in-codom* (≤$_{R2\ (l1\ x1)\ (l1\ x2)}$)) $\Rrightarrow$ (≤$_{L2\ x1\ x2}$)) ($r2_{x1\ (l1\ x2)}$) ($r2_{x2\ (l1\ x2)}$)
and ((*x* : *in-dom* (≤$_{L1}$)) $\Rrightarrow_m$ (*x1'* *x2'* :: (≤$_{R1}$) | *x* $_{L1}\lessapprox$ *x1'*) $\Rrightarrow_m$
(*in-field* (≤$_{R2\ (l1\ x)\ x2'}$)) $\Rrightarrow$ (≤$_{L2\ x\ (r1\ x2')}$)) *r2*
proof −
show ((*in-codom* (≤$_{R2\ (l1\ x1)\ (l1\ x2)}$)) $\Rrightarrow$ (≤$_{L2\ x1\ x2}$)) ($r2_{x1\ (l1\ x2)}$) ($r2_{x2\ (l1\ x2)}$)
if *x1* ≤$_{L1}$ *x2* **for** *x1 x2*
proof −
from *galois-conn1* ‹*x1* ≤$_{L1}$ *x2*› **have** *x2* $_{L1}\lessapprox$ *l1 x2 l1 x2* ≤$_{R1}$ *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* ≤$_{L1}$ *x2*› *L2-le2* **by** *fastforce*
qed
from *mono-r2* **show** ((*x* : *in-dom* (≤$_{L1}$)) $\Rrightarrow_m$ (*x1'* *x2'* :: (≤$_{R1}$) | *x* $_{L1}\lessapprox$ *x1'*) $\Rrightarrow_m$
(*in-field* (≤$_{R2\ (l1\ x)\ x2'}$)) $\Rrightarrow$ (≤$_{L2\ x\ (r1\ x2')}$)) *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 ((≤$_{L1}$) ⊣ (≤$_{R1}$)) *l1 r1*
and *reflexive-on* (*in-codom* (≤$_{L1}$)) (≤$_{L1}$)

and *reflexive-on* (*in-dom* ($\leq_{R1}$)) ($\leq_{R1}$)
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \trianglelefteq (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow (\leq_{L2\ x\ (\eta_1\ x)}) \leq (\leq_{L2\ x\ x})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x')\ x'}) \leq (\leq_{R2\ x'\ x'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$
$((in\text{-}dom\ (\leq_{L2\ x\ (\eta_1\ x)})) \Rrightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (l2_{(l1\ x)\ (\eta_1\ x)})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$
$((in\text{-}codom\ (\leq_{R2\ (\varepsilon_1\ x')\ x'})) \Rrightarrow (\leq_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* ($\leq_{L2\ x1\ x2}$)
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$ *transitive* ($\leq_{R2\ x1'\ x2'}$)
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

lemma *galois-connection-left-rightI'*:
assumes $((\leq_{L1}) \dashv (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-codom* ($\leq_{L1}$)) ($\leq_{L1}$)
and *reflexive-on* (*in-dom* ($\leq_{R1}$)) ($\leq_{R1}$)
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$((\leq_{L2\ (r1\ x1')\ (r1\ x2')}) \Rrightarrow_m (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}))\ (l2_{x2'\ (r1\ x1')})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow ((\leq_{R2\ (l1\ x1)\ (l1\ x2)}) \Rrightarrow_m (\leq_{L2\ x1\ (\eta_1\ x2)}))\ (r2_{x1\ (l1\ x2)})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \trianglelefteq (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$((in\text{-}dom\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')})) \Rrightarrow (\leq_{R2\ x1'\ x2'}))\ (l2_{x1'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x1')})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$((in\text{-}codom\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')})) \Rrightarrow (\leq_{R2\ x1'\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x2')})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$
$((in\text{-}dom\ (\leq_{L2\ x\ (\eta_1\ x)})) \Rrightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (l2_{(l1\ x)\ (\eta_1\ x)})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$
$((in\text{-}codom\ (\leq_{R2\ (l1\ x1)\ (l1\ x2)})) \Rrightarrow (\leq_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x2)})\ (r2_{x2\ (l1\ x2)})$
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 $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* ($\leq_{L2\ x1\ x2}$)
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$ *transitive* ($\leq_{R2\ x1'\ x2'}$)
shows $((\leq_L) \dashv (\leq_R))\ l\ r$
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 $((\leq_{L1}) \dashv (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-codom* $(\leq_{L1})$) $(\leq_{L1})$
and *reflexive-on* (*in-dom* $(\leq_{R1})$) $(\leq_{R1})$
and $\bigwedge x\ x'.\ x\ _{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \dashv (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$((in\text{-}dom\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')}))) \Rrightarrow (\leq_{R2\ x1'\ x2'}))\ (l2_{x1'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x1')})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$((in\text{-}codom\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')}))) \Rrightarrow (\leq_{R2\ x1'\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x2')})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$
$((in\text{-}dom\ (\leq_{L2\ x\ (\eta_1\ x)}))) \Rrightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (l2_{(l1\ x)\ (\eta_1\ x)})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$
$((in\text{-}codom\ (\leq_{R2\ (l1\ x1)\ (l1\ x2)}))) \Rrightarrow (\leq_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x2)})\ (r2_{x2\ (l1\ x2)})$
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 $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$ *transitive* $(\leq_{R2\ x1'\ x2'})$
shows $((\leq_L) \dashv (\leq_R))\ l\ r$
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 $((\leq_{L1}) \dashv (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and *reflexive-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and $\bigwedge x\ x'.\ x\ _{L1}\lessapprox x' \Longrightarrow$
$((\leq_{L2\ x\ (r1\ x')}) \dashv (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$((in\text{-}dom\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')}))) \Rrightarrow (\leq_{R2\ x1'\ x2'}))\ (l2_{x1'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x1')})$
and $((x' : in\text{-}codom\ (\leq_{R1})) \Rrightarrow_m (x1\ x2 :: (\leq_{L1}) \mid x2\ _{L1}\lessapprox x') \Rrightarrow_m$

(*in-field* ($\leq_{L2\ x1\ (r1\ x')}$)) $\Rrightarrow$ ($\leq_{R2\ (l1\ x1)\ x'}$)) *l2*
and $\bigwedge$*x1 x2*. *x1* $\leq_{L1}$ *x2* $\Longrightarrow$
((*in-codom* ($\leq_{R2\ (l1\ x1)\ (l1\ x2)}$)) $\Rrightarrow$ ($\leq_{L2\ x1\ x2}$)) ($r2_{x1\ (l1\ x2)}$) ($r2_{x2\ (l1\ x2)}$)
and ((*x* : *in-dom* ($\leq_{L1}$)) $\Rrightarrow_m$ (*x1' x2'* :: ($\leq_{R1}$) | *x* $_{L1}\lessapprox$ *x1'*) $\Rrightarrow_m$
(*in-field* ($\leq_{R2\ (l1\ x)\ x2'}$)) $\Rrightarrow$ ($\leq_{L2\ x\ (r1\ x2')}$)) *r2*
and $\bigwedge$*x1 x2*. *x1* $\leq_{L1}$ *x2* $\Longrightarrow$ *transitive* ($\leq_{L2\ x1\ x2}$)
and $\bigwedge$*x1' x2'*. *x1'* $\leq_{R1}$ *x2'* $\Longrightarrow$ *transitive* ($\leq_{R2\ x1'\ x2'}$)
shows (($\leq_L$) $\dashv$ ($\leq_R$)) *l r*
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 (($\leq_{L1}$) $\dashv$ ($\leq_{R1}$)) *l1 r1*
and *reflexive-on* (*in-field* ($\leq_{L1}$)) ($\leq_{L1}$)
and *reflexive-on* (*in-field* ($\leq_{R1}$)) ($\leq_{R1}$)
and $\bigwedge$*x x'*. *x* $_{L1}\lessapprox$ *x'* $\Longrightarrow$ (($\leq_{L2\ x\ (r1\ x')}$) $\dashv$ ($\leq_{R2\ (l1\ x)\ x'}$)) ($l2_{x'\ x}$) ($r2_{x\ x'}$)
and $\bigwedge$*x1 x2*. *x1* $\leq_{L1}$ *x2* $\Longrightarrow$ ($\leq_{L2\ x2\ x2}$) $\leq$ ($\leq_{L2\ x1\ x2}$)
and $\bigwedge$*x1 x2*. *x1* $\leq_{L1}$ *x2* $\Longrightarrow$ ($\leq_{L2\ x1\ (\eta_1\ x2)}$) $\leq$ ($\leq_{L2\ x1\ x2}$)
and $\bigwedge$*x1' x2'*. *x1'* $\leq_{R1}$ *x2'* $\Longrightarrow$ ($\leq_{R2\ (\varepsilon_1\ x1')\ x2'}$) $\leq$ ($\leq_{R2\ x1'\ x2'}$)
and $\bigwedge$*x1' x2'*. *x1'* $\leq_{R1}$ *x2'* $\Longrightarrow$ ($\leq_{R2\ x1'\ x1'}$) $\leq$ ($\leq_{R2\ x1'\ x2'}$)
and ((*x1' x2'* :: ($\leq_{R1}$)) $\Rrightarrow_m$ (*x1 x2* :: ($\leq_{L1}$) | *x2* $_{L1}\lessapprox$ *x1'*) $\Rrightarrow$
in-field ($\leq_{L2\ x1\ (r1\ x2')}$) $\Rrightarrow$ ($\leq_{R2\ (l1\ x1)\ x2'}$)) *l2*
and ((*x1 x2* :: ($\leq_{L1}$)) $\Rrightarrow_m$ (*x1' x2'* :: ($\leq_{R1}$) | *x2* $_{L1}\lessapprox$ *x1'*) $\Rrightarrow$
in-field ($\leq_{R2\ (l1\ x1)\ x2'}$) $\Rrightarrow$ ($\leq_{L2\ x1\ (r1\ x2')}$)) *r2*
and $\bigwedge$*x1 x2*. *x1* $\leq_{L1}$ *x2* $\Longrightarrow$ *transitive* ($\leq_{L2\ x1\ x2}$)
and $\bigwedge$*x1' x2'*. *x1'* $\leq_{R1}$ *x2'* $\Longrightarrow$ *transitive* ($\leq_{R2\ x1'\ x2'}$)
shows (($\leq_L$) $\dashv$ ($\leq_R$)) *l r*
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 (($\leq_{L1}$) $\dashv$ ($\leq_{R1}$)) *l1 r1*
and *reflexive-on* (*in-field* ($\leq_{L1}$)) ($\leq_{L1}$)
and *reflexive-on* (*in-field* ($\leq_{R1}$)) ($\leq_{R1}$)
and $\bigwedge$*x x'*. *x* $_{L1}\lessapprox$ *x'* $\Longrightarrow$ (($\leq_{L2\ x\ (r1\ x')}$) $\dashv$ ($\leq_{R2\ (l1\ x)\ x'}$)) ($l2_{x'\ x}$) ($r2_{x\ x'}$)
and ((- *x2* :: ($\leq_{L1}$)) $\Rrightarrow_m$ (*x3 x4* :: ($\leq_{L1}$) | (*x2* $\leq_{L1}$ *x3* $\wedge$ *x4* $\leq_{L1}$ η_1 *x3*)) $\Rrightarrow$
($\geq$)) *L2*
and ((*x1' x2'* :: ($\leq_{R1}$) | ε_1 *x2'* $\leq_{R1}$ *x1'*) $\Rrightarrow_m$ (*x3'* - :: ($\leq_{R1}$) | *x2'* $\leq_{R1}$ *x3'*) $\Rrightarrow$
($\leq$)) *R2*
and ((*x1' x2'* :: ($\leq_{R1}$)) $\Rrightarrow_m$ (*x1 x2* :: ($\leq_{L1}$) | *x2* $_{L1}\lessapprox$ *x1'*) $\Rrightarrow$
in-field ($\leq_{L2\ x1\ (r1\ x2')}$) $\Rrightarrow$ ($\leq_{R2\ (l1\ x1)\ x2'}$)) *l2*
and ((*x1 x2* :: ($\leq_{L1}$)) $\Rrightarrow_m$ (*x1' x2'* :: ($\leq_{R1}$) | *x2* $_{L1}\lessapprox$ *x1'*) $\Rrightarrow$
in-field ($\leq_{R2\ (l1\ x1)\ x2'}$) $\Rrightarrow$ ($\leq_{L2\ x1\ (r1\ x2')}$)) *r2*

and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$ *transitive* $(\leq_{R2\ x1'\ x2'})$
shows $((\leq_L) \dashv (\leq_R))\ l\ r$
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 $((\leq_{L1}) \dashv (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-codom* $(\leq_{L1})$) $(\leq_{L1})$
and *reflexive-on* (*in-dom* $(\leq_{R1})$) $(\leq_{R1})$
and $((\leq_{L2}) \dashv (\leq_{R2}))$ *l2 r2*
and *transitive* $(\leq_{L2})$
and *transitive* $(\leq_{R2})$
shows $((\leq_L) \dashv (\leq_R))\ l\ r$
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* $(\leq_L)$) $(\leq_L)$
and *R-le-R-if-L*: $\bigwedge x1\ x2.\ x1 \leq_L x2 \Longrightarrow (\leq_{R\ x2\ x2}) \leq (\leq_{R\ x1\ x2})$
and *pequiv-R*: $\bigwedge x1\ x2.\ x1 \leq_L x2 \Longrightarrow$ *partial-equivalence-rel* $(\leq_{R\ x1\ x2})$
shows *reflexive-on* (*in-dom DFR*) *DFR*
proof (*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* $\leq_L$ *x2*
moreover with *refl-L* **have** *x2* $\leq_L$ *x2* **by** *blast*
ultimately have *f x1* $\leq_{R\ x1\ x2}$ *g x2 f x2* $\leq_{R\ x1\ x2}$ *g x2*
using *R-le-R-if-L* **by** *auto*
moreover with *pequiv-R* ‹*x1* $\leq_L$ *x2*› **have** *g x2* $\leq_{R\ x1\ x2}$ *f x2*
by (*blast dest*: *symmetricD*)
ultimately show *f x1* $\leq_{R\ x1\ x2}$ *f x2* **using** *pequiv-R* ‹*x1* $\leq_L$ *x2*› **by** *blast*
qed

lemma *reflexive-on-in-codomI*:
assumes *refl-L*: *reflexive-on* (*in-dom* ($\leq_L$)) ($\leq_L$)
and *R-le-R-if-L*: $\bigwedge$*x1 x2*. *x1* $\leq_L$ *x2* $\Longrightarrow$ ($\leq_{R\ x1\ x1}$) $\leq$ ($\leq_{R\ x1\ x2}$)
and *pequiv-R*: $\bigwedge$*x1 x2*. *x1* $\leq_L$ *x2* $\Longrightarrow$ *partial-equivalence-rel* ($\leq_{R\ x1\ x2}$)
shows *reflexive-on* (*in-codom DFR*) *DFR*
proof (*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* $\leq_L$ *x2*
moreover with *refl-L* **have** *x1* $\leq_L$ *x1* **by** *blast*
ultimately have *g x1* $\leq_{R\ x1\ x2}$ *f x2 g x1* $\leq_{R\ x1\ x2}$ *f x1*
using *R-le-R-if-L* **by** *auto*
moreover with *pequiv-R* ‹*x1* $\leq_L$ *x2*› **have** *f x1* $\leq_{R\ x1\ x2}$ *g x1*
by (*blast dest*: *symmetricD*)
ultimately show *f x1* $\leq_{R\ x1\ x2}$ *f x2* **using** *pequiv-R* ‹*x1* $\leq_L$ *x2*› **by** *blast*
qed

corollary *reflexive-on-in-fieldI*:
assumes *reflexive-on* (*in-field* ($\leq_L$)) ($\leq_L$)
and $\bigwedge$*x1 x2*. *x1* $\leq_L$ *x2* $\Longrightarrow$ ($\leq_{R\ x2\ x2}$) $\leq$ ($\leq_{R\ x1\ x2}$)
and $\bigwedge$*x1 x2*. *x1* $\leq_L$ *x2* $\Longrightarrow$ ($\leq_{R\ x1\ x1}$) $\leq$ ($\leq_{R\ x1\ x2}$)
and $\bigwedge$*x1 x2*. *x1* $\leq_L$ *x2* $\Longrightarrow$ *partial-equivalence-rel* ($\leq_{R\ x1\ x2}$)
shows *reflexive-on* (*in-field DFR*) *DFR*
proof −
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* $\leq_{R\ y\ x}$ *g x* **by** *blast*
with *sym-R* ‹*y* $\leq_L$ *x*› **have** *g x* $\leq_{R\ y\ x}$ *f y* **by** (*blast dest*: *symmetricD*)

with *R-le-R-if-L* ‹$y \leq_L x$› **show** $g\ x \leq_{R\ x\ y} f\ y$ **by** *blast*
qed

corollary *partial-equivalence-relI*:
assumes *reflexive-on* (*in-field* ($\leq_L$)) ($\leq_L$)
and *sym-L*: *symmetric* ($\leq_L$)
and *mono-R*: (($x1\ x2$ :: ($\leq_L$)) $\Rightarrow_m$ ($x3\ x4$ :: ($\leq_L$) | $x1 \leq_L x3$) $\Rightarrow$ ($\leq$)) R
and $\bigwedge x1\ x2.\ x1 \leq_L x2 \Longrightarrow$ *partial-equivalence-rel* ($\leq_{R\ x1\ x2}$)
shows *partial-equivalence-rel DFR*
proof −
have ($\leq_{R\ x1\ x2}$) ≤ ($\leq_{R\ x2\ x1}$) **if** $x1 \leq_L x2$ **for** *x1 x2*
proof −
from *sym-L* ‹$x1 \leq_L x2$› **have** $x2 \leq_L x1$ **by** (*rule symmetricD*)
with *mono-R* ‹$x1 \leq_L x2$› **show** *?thesis* **by** *blast*
qed
with *assms* **show** *?thesis*
by (*intro partial-equivalence-relI transitiveI symmetricI*)
(*blast 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 (($\leq_{L1}$) $\Rightarrow_m$ ($\leq_{R1}$)) *l1*
and (($\leq_{L1}$) $\trianglelefteq_h$ ($\leq_{R1}$)) *l1 r1*
and *L2-eq*: ($\leq_{L2\ x1\ x2}$) = ($\leq_{L2\ x1\ (\eta_1\ x2)}$)
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow$ *transitive* ($\leq_{L2\ x\ (r1\ x')}$)
and $x1 \leq_{L1} x2$
shows *transitive* ($\leq_{L2\ x1\ x2}$)
by (*subst L2-eq*) (*auto intro*!: *assms t1.left-Galois-left-if-left-relI*)

lemma *symmetric-left2-if-symmetric-left2-if-left-GaloisI*:
assumes (($\leq_{L1}$) $\Rightarrow_m$ ($\leq_{R1}$)) *l1*
and (($\leq_{L1}$) $\trianglelefteq_h$ ($\leq_{R1}$)) *l1 r1*
and *L2-eq*: ($\leq_{L2\ x1\ x2}$) = ($\leq_{L2\ x1\ (\eta_1\ x2)}$)
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow$ *symmetric* ($\leq_{L2\ x\ (r1\ x')}$)

and *x1* $\leq_{L1}$ *x2*
shows *symmetric* $(\leq_{L2\ x1\ x2})$
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 $((\leq_{L1}) \Rightarrow_m (\leq_{R1}))$ *l1*
and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*
and *L2-eq*: $(\leq_{L2\ x1\ x2}) = (\leq_{L2\ x1\ (\eta_1\ x2)})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow$ *partial-equivalence-rel* $(\leq_{L2\ x\ (r1\ x')})$
and *x1* $\leq_{L1}$ *x2*
shows *partial-equivalence-rel* $(\leq_{L2\ x1\ x2})$
by (*subst L2-eq*) (*auto intro*!: *assms t1.left-Galois-left-if-left-relI*)

context
assumes *galois-prop*: $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
begin

interpretation *flip-inv* :
transport-Dep-Fun-Rel $(\geq_{R1})$ $(\geq_{L1})$ *r1 l1 flip2 R2 flip2 L2 r2 l2*
rewrites *flip-inv.t1.unit* $\equiv \varepsilon_1$
and $\bigwedge R\ x\ y.$ (*flip2 R x y*) $\equiv (R\ y\ x)^{-1}$
and $\bigwedge R\ S.\ R^{-1} = S^{-1} \equiv R = S$
and $\bigwedge R\ S.\ (R^{-1} \Rightarrow_m S^{-1}) \equiv (R \Rightarrow_m S)$
and $\bigwedge x\ x'.\ x'\ {}_{R1}\gtrapprox\ x \equiv x\ {}_{L1}\lessapprox\ x'$
and $((\geq_{R1}) \trianglelefteq_h (\geq_{L1}))$ *r1 l1* $\equiv$ *True*
and $\bigwedge(R :: 'z \Rightarrow 'z \Rightarrow bool).$ *transitive* $R^{-1} \equiv$ *transitive R*
and $\bigwedge(R :: 'z \Rightarrow 'z \Rightarrow bool).$ *symmetric* $R^{-1} \equiv$ *symmetric R*
and $\bigwedge(R :: 'z \Rightarrow 'z \Rightarrow bool).$ *partial-equivalence-rel* $R^{-1} \equiv$ *partial-equivalence-rel R*
and $\bigwedge P.$ (*True* $\Longrightarrow P) \equiv$ *Trueprop P*
and $\bigwedge P\ Q.$ (*True* $\Longrightarrow$ *PROP P* $\Longrightarrow$ *PROP Q*) $\equiv$ (*PROP P* $\Longrightarrow$ *True* $\Longrightarrow$ *PROP Q*)
using *galois-prop*
by (*auto intro*!: *Eq-TrueI simp*: *t1.flip-unit-eq-counit*
galois-prop.half-galois-prop-right-rel-inv-iff-half-galois-prop-left
mono-wrt-rel-eq-dep-mono-wrt-rel
simp del: *rel-inv-iff-rel*)

lemma *transitive-right2-if-transitive-right2-if-left-GaloisI*:
assumes $((\leq_{R1}) \Rightarrow_m (\leq_{L1}))$ *r1*
and $(\leq_{R2\ x1\ x2}) = (\leq_{R2\ (\varepsilon_1\ x1)\ x2})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow$ *transitive* $(\leq_{R2\ (l1\ x)\ x'})$
and *x1* $\leq_{R1}$ *x2*
shows *transitive* $(\leq_{R2\ x1\ x2})$
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 $((\leq_{R1}) \Rightarrow_m (\leq_{L1}))$ *r1*
and $(\leq_{R2\ x1\ x2}) = (\leq_{R2\ (\varepsilon_1\ x1)\ x2})$
and $\bigwedge x\ x'.\ x\ _{L1}\lessapprox x' \Longrightarrow$ *symmetric* $(\leq_{R2\ (l1\ x)\ x'})$
and $x1 \leq_{R1} x2$
shows *symmetric* $(\leq_{R2\ x1\ x2})$
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 $((\leq_{R1}) \Rightarrow_m (\leq_{L1}))$ *r1*
and $(\leq_{R2\ x1\ x2}) = (\leq_{R2\ (\varepsilon_1\ x1)\ x2})$
and $\bigwedge x\ x'.\ x\ _{L1}\lessapprox x' \Longrightarrow$ *partial-equivalence-rel* $(\leq_{R2\ (l1\ x)\ x'})$
and $x1 \leq_{R1} x2$
shows *partial-equivalence-rel* $(\leq_{R2\ x1\ x2})$
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*: $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and $((x1\ x2 :: (\geq_{L1})) \Rightarrow_m (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rightarrow (\leq))$ *L2*
and $\bigwedge x\ x'.\ x\ _{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_{pre} (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $x1 \leq_{L1} x2$
shows *transitive* $(\leq_{L2\ x1\ x2})$
proof −
from ‹$x1 \leq_{L1} x2$› *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** $(\leq_{L2\ x1\ x2}) = (\leq_{L2\ x1\ (\eta_1\ x2)})$
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*: $((\leq_{L1}) \equiv_{PER} (\leq_{R1}))$ *l1 r1*
and $((x1\ x2 :: (\geq_{L1})) \Rightarrow_m (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rightarrow (\leq))$ *L2*
and $\bigwedge x\ x'.\ x\ _{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_{PER} (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $x1 \leq_{L1} x2$
shows *symmetric* $(\leq_{L2\ x1\ x2})$
proof −
from ‹$x1 \leq_{L1} x2$› *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** $(\leq_{L2\ x1\ x2}) = (\leq_{L2\ x1\ (\eta_1\ x2)})$
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*: $((\leq_{L1}) \equiv_{PER} (\leq_{R1}))$ *l1 r1*
and $((x1\ x2 :: (\geq_{L1})) \Rightarrow_m (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rrightarrow (\leq))$ *L2*
and $\bigwedge x\ x'.\ x\ {}_{L1}\!\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_{PER} (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $x1 \leq_{L1} x2$
shows *partial-equivalence-rel* $(\leq_{L2\ x1\ x2})$
proof −
from ‹$x1 \leq_{L1} x2$› *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** $(\leq_{L2\ x1\ x2}) = (\leq_{L2\ x1\ (\eta_1\ x2)})$
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*: $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and $((x1'\ x2' :: (\geq_{R1})) \Rightarrow_m (x3'\ x4' :: (\leq_{R1}) \mid x1' \leq_{R1} x3') \Rrightarrow (\leq))$ *R2*
and $\bigwedge x\ x'.\ x\ {}_{L1}\!\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_{pre} (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $x1' \leq_{R1} x2'$
shows *transitive* $(\leq_{R2\ x1'\ x2'})$
proof −
from ‹$x1' \leq_{R1} x2'$› *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** $(\leq_{R2\ x1'\ x2'}) = (\leq_{R2\ (\varepsilon_1\ x1')\ x2'})$
by (*intro flip.left2-eq-if-bi-related-if-monoI*) *blast+*
with *assms* **show** *?thesis*
by (*intro 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 $((x1'\ x2' :: (\geq_{R1})) \Rightarrow_m (x3'\ x4' :: (\leq_{R1}) \mid x1' \leq_{R1} x3') \Rrightarrow (\leq))\ R2$
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 $\langle x1' \leq_{R1} x2' \rangle$ *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 $((x1'\ x2' :: (\geq_{R1})) \Rightarrow_m (x3'\ x4' :: (\leq_{R1}) \mid x1' \leq_{R1} x3') \Rrightarrow (\leq))\ R2$
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 $\langle x1' \leq_{R1} x2' \rangle$ *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 $((\leq_{L1}) \Rightarrow_m (\leq_{R1}))$ *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 $((\leq_{R1}) \Rightarrow_m (\leq_{L1}))$ *r1*
and *half-galois-prop-right2*: $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow$
$((\leq_{R2}\ (l1\ x)\ x') \trianglelefteq_h\ (\leq_{L2}\ x\ (r1\ x'))) \ (r2_{x\ x'})\ (l2_{x'\ x})$
and $(\leq_{R2}\ (\varepsilon_1\ x')\ x') = (\leq_{R2}\ x'\ x')$

and $(\leq_{R2\ x'\ (\varepsilon_1\ x')}) = (\leq_{R2\ x'\ x'})$
and $x' \leq_{R1} x'$
shows $((\leq_{R2\ x'\ (\varepsilon_1\ x')}) \trianglelefteq_h (\leq_{L2\ (r1\ x')\ (r1\ x')})) (r2_{(r1\ x')\ x'}) (l2_{x'\ (r1\ x')})$
proof −
from *assms* **have** $r1\ x'\ _{L1}\lessapprox x'$ **by** (*intro t1.right-left-Galois-if-right-relI*) *auto*
with *half-galois-prop-right2*
have $((\leq_{R2\ (\varepsilon_1\ x')\ x'}) \trianglelefteq_h (\leq_{L2\ (r1\ x')\ (r1\ x')})) (r2_{(r1\ x')\ x'}) (l2_{x'\ (r1\ x')})$ **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* $\equiv \eta_1$ **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-galois-equivalence-mono-assms-leftI*:
assumes *galois-equiv1*: $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1*
and *preorder-L1*: *preorder-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and *mono-L2*: $((x1\ x2 :: (\geq_{L1})) \Rrightarrow_m (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rrightarrow (\leq))\ L2$
shows $((x1\ x2 :: (\leq_{L1}) \mid \eta_1\ x2 \leq_{L1} x1) \Rrightarrow_m (x3\ x4 :: (\leq_{L1}) \mid x2 \leq_{L1} x3) \Rrightarrow (\leq))\ L2$ (**is** *?goal1*)
and $((x1\ x2 :: (\leq_{L1})) \Rrightarrow_m (x3\ x4 :: (\leq_{L1}) \mid (x2 \leq_{L1} x3 \wedge x4 \leq_{L1} \eta_1\ x3)) \Rrightarrow (\geq))\ L2$ (**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** $x1 \leq_{L1} x2$
moreover with *galois-equiv1 preorder-L1* **have** $x2 \leq_{L1} \eta_1\ x2$
by (*blast intro*: *t1.rel-unit-if-reflexive-on-if-galois-connection*)
moreover assume $\eta_1\ x2 \leq_{L1} x1$
ultimately have $x2 \equiv_{L1} x1$ **using** *preorder-L1* **by** *blast*
moreover assume $x3 \leq_{L1} x4\ x2 \leq_{L1} x3$
ultimately show $(\leq_{L2\ x1\ x3}) \leq (\leq_{L2\ x2\ x4})$ **using** *preorder-L1 mono-L2*
by (*intro le-relI*) (*blast dest*!: *rel-ifD elim*!: *dep-mono-wrt-relE*)
qed
show *?goal2*
proof (*intro dep-mono-wrt-relI rel-if-if-impI Dep-Fun-Rel-relI*)
fix *x1 x2 x3 x4* **presume** $x3 \leq_{L1} x4\ x4 \leq_{L1} \eta_1\ x3$
moreover with *galois-equiv1 preorder-L1* **have** $\eta_1\ x3 \leq_{L1} x3$
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 $x1 \leq_{L1} x2\ x2 \leq_{L1} x3$
ultimately show $(\leq_{L2\ x2\ x4}) \leq (\leq_{L2\ x1\ x3})$ **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*: $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1*

and *refl-L1*: *reflexive-on* (*in-field* ($\leq_{L1}$)) ($\leq_{L1}$)
and *refl-R1*: *reflexive-on* (*in-field* ($\leq_{R1}$)) ($\leq_{R1}$)
and *mono-L2*: ((*x1 x2* :: ($\geq_{L1}$)) $\Rightarrow_m$ (*x3 x4* :: ($\leq_{L1}$) | *x1* $\leq_{L1}$ *x3*) $\Rrightarrow$ ($\leq$)) *L2*
and *mono-R2*: ((*x1' x2'* :: ($\geq_{R1}$)) $\Rightarrow_m$ (*x3' x4'* :: ($\leq_{R1}$) | *x1'* $\leq_{R1}$ *x3'*) $\Rrightarrow$ ($\leq$)) *R2*
and *mono-l2*: ((*x1' x2'* :: ($\leq_{R1}$)) $\Rightarrow_m$ (*x1 x2* :: ($\leq_{L1}$) | *x2* $_{L1}\lessapprox$ *x1'*) $\Rrightarrow$ *in-field* ($\leq_{L2\ x1\ (r1\ x2')}$) $\Rrightarrow$ ($\leq_{R2\ (l1\ x1)\ x2'}$)) *l2*
and $x \leq_{L1} x$
shows (*in-codom* ($\leq_{L2\ (\eta_1\ x)\ x}$) $\Rrightarrow$ ($\leq_{R2\ (l1\ x)\ (l1\ x)}$)) ($^{l2}_{(l1\ x)\ (\eta_1\ x)}$) ($^{l2}_{(l1\ x)\ x}$)
proof (*intro Fun-Rel-predI*)
fix *y* **assume** *in-codom* ($\leq_{L2\ (\eta_1\ x)\ x}$) *y*
moreover from ‹$x \leq_{L1} x$› *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** $\eta_1\ x \leq_{L1} \eta_1\ x$ **by** *blast*
ultimately have *in-codom* ($\leq_{L2\ (\eta_1\ x)\ (\eta_1\ x)}$) *y* **using** *mono-L2* **by** *blast*
moreover from ‹$x \leq_{L1} x$› *galois-equiv1*
have *l1 x* $\leq_{R1}$ *l1 x* η_1 *x* $\leq_{L1}$ *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* $\leq_{R1}$ *l1 x*›] ‹η_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 note ‹*l1 x* $\leq_{R1}$ *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* $\leq_{R2\ (l1\ x)\ (l1\ x)}$ $^{l2}_{(l1\ x)\ x}$ *y*
using *mono-R2* **by** *blast*
qed

lemma *galois-equivalence-if-mono-if-galois-equivalence-Dep-Fun-Rel-pred-assm-right*:
assumes *galois-equiv1*: (($\leq_{L1}$) $\equiv_G$ ($\leq_{R1}$)) *l1 r1*
and *refl-R1*: *reflexive-on* (*in-field* ($\leq_{R1}$)) ($\leq_{R1}$)
and *mono-L2*: ((*x1 x2* :: ($\geq_{L1}$)) $\Rightarrow_m$ (*x3 x4* :: ($\leq_{L1}$) | *x1* $\leq_{L1}$ *x3*) $\Rrightarrow$ ($\leq$)) *L2*
and *mono-R2*: ((*x1' x2'* :: ($\geq_{R1}$)) $\Rightarrow_m$ (*x3' x4'* :: ($\leq_{R1}$) | *x1'* $\leq_{R1}$ *x3'*) $\Rrightarrow$ ($\leq$)) *R2*
and *mono-r2*: ((*x1 x2* :: ($\leq_{L1}$)) $\Rightarrow_m$ (*x1' x2'* :: ($\leq_{R1}$) | *x2* $_{L1}\lessapprox$ *x1'*) $\Rrightarrow$ *in-field* ($\leq_{R2\ (l1\ x1)\ x2'}$) $\Rrightarrow$ ($\leq_{L2\ x1\ (r1\ x2')}$)) *r2*
and $x' \leq_{R1} x'$
shows (*in-dom* ($\leq_{R2\ x'\ (\varepsilon_1\ x')}$) $\Rrightarrow$ ($\leq_{L2\ (r1\ x')\ (r1\ x')}$)) ($^{r2}_{(r1\ x')\ x'}$) ($^{r2}_{(r1\ x')\ (\varepsilon_1\ x')}$)
proof (*intro Fun-Rel-predI*)
fix *y* **assume** *in-dom* ($\leq_{R2\ x'\ (\varepsilon_1\ x')}$) *y*
moreover from ‹$x' \leq_{R1} x'$› *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** $\varepsilon_1\ x' \leq_{R1} \varepsilon_1\ x'$ **by** *blast*

ultimately have *in-dom* $(\leq_{R2\ (\varepsilon_1\ x')\ (\varepsilon_1\ x'')})\ y$ **using** *mono-R2* **by** *blast*
moreover from $\langle x' \leq_{R1} x'' \rangle$ *galois-equiv1*
have $r1\ x' \leq_{L1} r1\ x''\ x'' \leq_{R1} \varepsilon_1\ x'\ r1\ x''\ {}_{L1}\lessapprox x'$
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* $\langle r1\ x' \leq_{L1} r1\ x'' \rangle$] $\langle x' \leq_{R1} \varepsilon_1\ x'' \rangle$]
ultimately have ${}^{r2}{}_{(r1\ x'')\ x'}\ y \leq_{L2\ (r1\ x')\ (\eta_1\ (r1\ x''))}\ {}^{r2}{}_{(r1\ x')\ (\varepsilon_1\ x'')}\ y$ **by** *auto*
moreover note $\langle r1\ x' \leq_{L1} r1\ x'' \rangle$
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 ${}^{r2}{}_{(r1\ x'')\ x'}\ y \leq_{L2\ (r1\ x')\ (r1\ x'')}\ {}^{r2}{}_{(r1\ x')\ (\varepsilon_1\ x'')}\ y$
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*: $((\leq_{L1}) \equiv_G (\leq_{R1}))\ l1\ r1$
and *refl-L1*: *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and *reflexive-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and *galois-equiv2*: $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow$
$((\leq_{L2\ x\ (r1\ x')}) \equiv_G (\leq_{R2\ (l1\ x)\ x''}))\ (l2_{x'\ x})\ (r2_{x\ x''})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow (\leq_{L2\ (\eta_1\ x)\ x}) \leq (\leq_{L2\ x\ x})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x2'\ x2''}) \leq (\leq_{R2\ x1'\ x2''})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2''}) \leq (\leq_{R2\ x1'\ x2''})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1''}) \leq (\leq_{R2\ x1'\ x2''})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow (\leq_{R2\ x'\ (\varepsilon_1\ x')}) \leq (\leq_{R2\ x'\ x'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$(in\text{-}dom\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')}) \Rrightarrow (\leq_{R2\ x1'\ x2''}))\ (l2_{x1'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x1'')})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$(in\text{-}codom\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')}) \Rrightarrow (\leq_{R2\ x1'\ x2''}))\ (l2_{x2'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x2'')})$

and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$
$(\mathit{in\text{-}dom}\ (\leq_{L2\ x\ (\eta_1\ x)}) \Rrightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ x})\ (l2_{(l1\ x)\ (\eta_1\ x)})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$
$(\mathit{in\text{-}codom}\ (\leq_{L2\ (\eta_1\ x)\ x}) \Rrightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ (\eta_1\ x)})\ (l2_{(l1\ x)\ x})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$
$(\mathit{in\text{-}codom}\ (\leq_{R2\ (l1\ x1)\ (l1\ x2)}) \Rrightarrow (\leq_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x2)})\ (r2_{x2\ (l1\ x2)})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$
$(\mathit{in\text{-}dom}\ (\leq_{R2\ (l1\ x1)\ (l1\ x2)}) \Rrightarrow (\leq_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x1)})\ (r2_{x1\ (l1\ x2)})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$
$(\mathit{in\text{-}codom}\ (\leq_{R2\ (\varepsilon_1\ x')\ x'}) \Rrightarrow (\leq_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$
$(\mathit{in\text{-}dom}\ (\leq_{R2\ x'\ (\varepsilon_1\ x')}) \Rrightarrow (\leq_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ x'})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow \mathit{transitive}\ (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow \mathit{transitive}\ (\leq_{R2\ x1'\ x2'})$
shows $((\leq_L) \equiv_G (\leq_R))\ l\ r$
proof −
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 $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and *reflexive-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_G (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow (\leq_{L2\ (\eta_1\ x)\ x}) \leq (\leq_{L2\ x\ x})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x2'\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow (\leq_{R2\ x'\ (\varepsilon_1\ x')}) \leq (\leq_{R2\ x'\ x'})$
and $((x1'\ x2' :: (\leq_{R1})) \Rrightarrow_m (x1\ x2 :: (\leq_{L1}) \mid x2\ {}_{L1}\lessapprox x1') \Rrightarrow$
in-field $(\leq_{L2\ x1\ (r1\ x2')}) \Rrightarrow (\leq_{R2\ (l1\ x1)\ x2'}))\ l2$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$
$(\mathit{in\text{-}codom}\ (\leq_{L2\ (\eta_1\ x)\ x}) \Rrightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ (\eta_1\ x)})\ (l2_{(l1\ x)\ x})$
and $((x1\ x2 :: (\leq_{L1})) \Rrightarrow_m (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$
$(\mathit{in\text{-}dom}\ (\leq_{R2\ x'\ (\varepsilon_1\ x')}) \Rrightarrow (\leq_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ x'})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$ *transitive* $(\leq_{R2\ x1'\ x2'})$
shows $((\leq_L) \equiv_G (\leq_R))\ l\ r$
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 $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and *reflexive-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_G (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $((x1\ x2 :: (\leq_{L1}) \mid \eta_1\ x2 \leq_{L1} x1) \Rrightarrow_m (x3\ x4 :: (\leq_{L1}) \mid x2 \leq_{L1} x3) \Rrightarrow (\leq))$ *L2*
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 $((x1\,'\,x2\,' :: (\leq_{R1})) \Rightarrow_m (x3\,'\,x4\,' :: (\leq_{R1}) \mid (x2\,' \leq_{R1} x3\,' \wedge x4\,' \leq_{R1} \varepsilon_1\ x3\,'))$
$\Rightarrow (\geq))\ R2$
and $((x1\,'\,x2\,' :: (\leq_{R1})) \Rightarrow_m (x1\ x2 :: (\leq_{L1}) \mid x2\ _{L1}\lessapprox x1\,') \Rightarrow$
in-field $(\leq_{L2\ x1\ (r1\ x2\,')}) \Rightarrow (\leq_{R2\ (l1\ x1)\ x2\,'}))\ l2$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$
$(\textit{in-codom}\ (\leq_{L2\ (\eta_1\ x)\ x}) \Rightarrow (\leq_{R2\ (l1\ x)\ (l1\ x)}))\ (l2_{(l1\ x)\ (\eta_1\ x)})\ (l2_{(l1\ x)\ x})$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow_m (x1\,'\,x2\,' :: (\leq_{R1}) \mid x2\ _{L1}\lessapprox x1\,') \Rightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2\,'}) \Rightarrow (\leq_{L2\ x1\ (r1\ x2\,')}))\ r2$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$
$(\textit{in-dom}\ (\leq_{R2\ x'\ (\varepsilon_1\ x')}) \Rightarrow (\leq_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ x'})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow \textit{transitive}\ (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\,'\ x2\,'.\ x1\,' \leq_{R1} x2\,' \Longrightarrow \textit{transitive}\ (\leq_{R2\ x1\,'\ x2\,'})$
shows $((\leq_L) \equiv_G (\leq_R))\ l\ r$
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 $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))\ l1\ r1$
and $\bigwedge x\ x'.\ x\ _{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_G (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $((x1\ x2 :: (\geq_{L1})) \Rightarrow_m (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rightarrow (\leq))\ L2$
and $((x1\,'\,x2\,' :: (\geq_{R1})) \Rightarrow_m (x3\,'\,x4\,' :: (\leq_{R1}) \mid x1\,' \leq_{R1} x3\,') \Rightarrow (\leq))\ R2$
and $((x1\,'\,x2\,' :: (\leq_{R1})) \Rightarrow_m (x1\ x2 :: (\leq_{L1}) \mid x2\ _{L1}\lessapprox x1\,') \Rightarrow$
in-field $(\leq_{L2\ x1\ (r1\ x2\,')}) \Rightarrow (\leq_{R2\ (l1\ x1)\ x2\,'}))\ l2$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow_m (x1\,'\,x2\,' :: (\leq_{R1}) \mid x2\ _{L1}\lessapprox x1\,') \Rightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2\,'}) \Rightarrow (\leq_{L2\ x1\ (r1\ x2\,')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow \textit{transitive}\ (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\,'\ x2\,'.\ x1\,' \leq_{R1} x2\,' \Longrightarrow \textit{transitive}\ (\leq_{R2\ x1\,'\ x2\,'})$
shows $((\leq_L) \equiv_G (\leq_R))\ l\ r$
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 $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))\ l1\ r1$
and $\bigwedge x\ x'.\ x\ _{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_{pre} (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$

and $((x1\ x2 :: (\geq_{L1})) \Rightarrow_m (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rightarrow (\leq))\ L2$
and $((x1'\ x2' :: (\geq_{R1})) \Rightarrow_m (x3'\ x4' :: (\leq_{R1}) \mid x1' \leq_{R1} x3') \Rightarrow (\leq))\ R2$
and $((x1'\ x2' :: (\leq_{R1})) \Rightarrow_m (x1\ x2 :: (\leq_{L1}) \mid x2\ _{L1}\lessapprox x1') \Rightarrow$
in-field $(\leq_{L2\ x1\ (r1\ x2')}) \Rightarrow (\leq_{R2\ (l1\ x1)\ x2'}))\ l2$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow_m (x1'\ x2' :: (\leq_{R1}) \mid x2\ _{L1}\lessapprox x1') \Rightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
shows $((\leq_L) \equiv_G (\leq_R))\ l\ r$
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 $((\leq_{L1}) \equiv_G (\leq_{R1}))\ l1\ r1$
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and *reflexive-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and $((\leq_{L2}) \equiv_G (\leq_{R2}))\ l2\ r2$
and *transitive* $(\leq_{L2})$
and *transitive* $(\leq_{R2})$
shows $((\leq_L) \equiv_G (\leq_R))\ l\ r$
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 ($\leq_L$) and ($\leq_R$) are reflexive.

lemma *reflexive-on-in-field-left-rel2-le-assmI*:
  assumes *refl-L1*: *reflexive-on* (*in-dom* ($\leq_{L1}$)) ($\leq_{L1}$)
  and *mono-L2*: ((*x1* : $\top$) $\Rightarrow_m$ (*x2 x3* :: ($\leq_{L1}$) | *x1* $\leq_{L1}$ *x2*) $\Rightarrow_m$ ($\leq$)) *L2*
  and *x1* $\leq_{L1}$ *x2*
  shows ($\leq_{L2\ x1\ x1}$) $\leq$ ($\leq_{L2\ x1\ x2}$)
proof −
  from *refl-L1* ‹*x1* $\leq_{L1}$ *x2*› **have** *x1* $\leq_{L1}$ *x1* **by** *blast*
  with *dep-mono-wrt-relD*[*OF dep-mono-wrt-predD*[*OF mono-L2*] ‹*x1* $\leq_{L1}$ *x2*›]
    show ($\leq_{L2\ x1\ x1}$) $\leq$ ($\leq_{L2\ x1\ x2}$) **by** *auto*
qed

lemma *reflexive-on-in-field-mono-assm-left2I*:
  assumes *mono-L2*: ((*x1 x2* :: ($\geq_{L1}$)) $\Rightarrow_m$ (*x3 x4* :: ($\leq_{L1}$) | *x1* $\leq_{L1}$ *x3*) $\Rightarrow$ ($\leq$)) *L2*
  and *refl-L1*: *reflexive-on* (*in-dom* ($\leq_{L1}$)) ($\leq_{L1}$)
  shows ((*x1* : $\top$) $\Rightarrow_m$ (*x2 x3* :: ($\leq_{L1}$) | *x1* $\leq_{L1}$ *x2*) $\Rightarrow_m$ ($\leq$)) *L2*
proof (*intro dep-mono-wrt-predI dep-mono-wrt-relI rel-if-if-impI*)
  fix *x1 x2 x3* **assume** *x1* $\leq_{L1}$ *x2 x2* $\leq_{L1}$ *x3*
  with *refl-L1* **have** *x1* $\geq_{L1}$ *x1* **by** *blast*
  from *Dep-Fun-Rel-relD*[*OF dep-mono-wrt-relD*[*OF mono-L2* ‹*x1* $\geq_{L1}$ *x1*›]
    ‹*x2* $\leq_{L1}$ *x3*›] ‹*x1* $\leq_{L1}$ *x2*›
    show ($\leq_{L2\ x1\ x2}$) $\leq$ ($\leq_{L2\ x1\ x3}$) **by** *blast*
qed

lemma *reflexive-on-in-field-left-if-equivalencesI*:
  assumes (($\leq_{L1}$) $\equiv_G$ ($\leq_{R1}$)) *l1 r1*
  and *preorder-on* (*in-field* ($\leq_{L1}$)) ($\leq_{L1}$)
  and ((*x1 x2* :: ($\geq_{L1}$)) $\Rightarrow_m$ (*x3 x4* :: ($\leq_{L1}$) | *x1* $\leq_{L1}$ *x3*) $\Rightarrow$ ($\leq$)) *L2*
  and $\bigwedge$*x1 x2*. *x1* $\leq_{L1}$ *x2* $\Longrightarrow$ *partial-equivalence-rel* ($\leq_{L2\ x1\ x2}$)
  shows *reflexive-on* (*in-field* ($\leq_L$)) ($\leq_L$)
  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* ($\leq_{L1}$)) ($\leq_{L1}$)
and $\bigwedge$*x1 x2*. *x1* $\leq_{L1}$ *x2* $\Longrightarrow$ ($\leq_{L2\ x2\ x2}$) $\leq$ ($\leq_{L2\ x1\ x2}$)
and $\bigwedge$*x1 x2*. *x1* $\leq_{L1}$ *x2* $\Longrightarrow$ ($\leq_{L2\ x1\ x1}$) $\leq$ ($\leq_{L2\ x1\ x2}$)
and $\bigwedge$*x1 x2*. *x1* $\leq_{L1}$ *x2* $\Longrightarrow$ *partial-equivalence-rel* ($\leq_{L2\ x1\ x2}$)
shows ($\leq_L$) = *tdfr.L*
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 (($\leq_{L1}$) $\equiv_G$ ($\leq_{R1}$)) *l1 r1*
and *preorder-on* (*in-field* ($\leq_{L1}$)) ($\leq_{L1}$)
and ((*x1 x2* :: ($\geq_{L1}$)) $\Rrightarrow_m$ (*x3 x4* :: ($\leq_{L1}$) | *x1* $\leq_{L1}$ *x3*) $\Rrightarrow$ ($\leq$)) *L2*
and $\bigwedge$*x1 x2*. *x1* $\leq_{L1}$ *x2* $\Longrightarrow$ *partial-equivalence-rel* ($\leq_{L2\ x1\ x2}$)
shows ($\leq_L$) = *tdfr.L*
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* ($\leq_{L1}$)) ($\leq_{L1}$)
and *partial-equivalence-rel* ($\leq_{L2}$)
shows ($\leq_L$) = *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 $((\leq_{L1})\ {}_h\trianglelefteq\ (\leq_{R1}))$ *l1 r1*
and *refl-L1*: *reflexive-on* (*in-dom* $(\leq_{L1})$) $(\leq_{L1})$
and *mono-r2*: $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rrightarrow_m (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and *L2-le2*: $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
and *ge-L2-r2-le2*: $\bigwedge x\ x'\ y'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow$ *in-dom* $(\leq_{R2\ (l1\ x)\ x'})\ y' \Longrightarrow$
$(\geq_{L2\ x\ (r1\ x')})\ (r2_{x\ (l1\ x)}\ y') \leq (\geq_{L2\ x\ (r1\ x')})\ (r2_{x\ x'}\ y')$
and *trans-L2*: $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $g \leq_R g$
and $f\ {}_{L}\lessapprox g$
shows $((x\ x' :: ({}_{L1}\lessapprox)) \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
proof (*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 $\langle x\ {}_{L1}\lessapprox x'\rangle$ $\langle((\leq_{L1})\ {}_h\trianglelefteq\ (\leq_{R1}))\ l1\ r1\rangle$ **have** $x \leq_{L1} r1\ x'\ l1\ x \leq_{R1} x'$ **by** *auto*
with $\langle g \leq_R g\rangle$ **have** $g\ (l1\ x) \leq_{R2\ (l1\ x)\ x'} g\ x'$ **by** *blast*
then show *in-codom* $(\leq_{R2\ (l1\ x)\ x'})\ (g\ x')$ **by** *blast*

from $\langle f\ {}_{L}\lessapprox g\rangle$ **have** $f \leq_L r\ g$ **by** *blast*
moreover from *refl-L1* $\langle x\ {}_{L1}\lessapprox x'\rangle$ **have** $x \leq_{L1} x$ **by** *blast*
ultimately have $f\ x \leq_{L2\ x\ x} (r\ g)\ x$ **by** *blast*
with *L2-le2* $\langle x \leq_{L1} r1\ x'\rangle$ **have** $f\ x \leq_{L2\ x\ (r1\ x')} (r\ g)\ x$ **by** *blast*
then have $f\ x \leq_{L2\ x\ (r1\ x')} r2_{x\ (l1\ x)}\ (g\ (l1\ x))$ **by** *simp*
with *ge-L2-r2-le2* **have** $f\ x \leq_{L2\ x\ (r1\ x')} r2_{x\ x'}\ (g\ (l1\ x))$
using $\langle x\ {}_{L1}\lessapprox x'\rangle$ $\langle g\ (l1\ x) \leq_{R2\ (l1\ x)\ x'} \text{-}\rangle$ **by** *blast*
moreover have $\ldots \leq_{L2\ x\ (r1\ x')} r2_{x\ x'}\ (g\ x')$
using *mono-r2* $\langle x\ {}_{L1}\lessapprox x'\rangle$ $\langle g\ (l1\ x) \leq_{R2\ (l1\ x)\ x'} g\ x'\rangle$ **by** *blast*
ultimately show $f\ x \leq_{L2\ x\ (r1\ x')} r2_{x\ x'}\ (g\ x')$
using *trans-L2* $\langle x\ {}_{L1}\lessapprox x'\rangle$ **by** *blast*
qed
qed

lemma *left-rel-right-if-Dep-Fun-Rel-left-GaloisI*:
assumes *mono-l1*: $((\leq_{L1}) \Rrightarrow_m (\leq_{R1}))\ l1$
and *half-galois-prop-right1*: $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))\ l1\ r1$
and *L2-unit-le2*: $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and *ge-L2-r2-le1*: $\bigwedge x1\ x2\ y'.\ x1 \leq_{L1} x2 \Longrightarrow$ *in-codom* $(\leq_{R2\ (l1\ x1)\ (l1\ x2)})\ y' \Longrightarrow$
$(\geq_{L2\ x1\ x2})\ (r2_{x1\ (l1\ x2)}\ y') \leq (\geq_{L2\ x1\ x2})\ (r2_{x2\ (l1\ x2)}\ y')$
and *rel-f-g*: $((x\ x' :: ({}_{L1}\lessapprox)) \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
shows $f \leq_L r\ g$
proof (*intro left-relI*)
fix *x1 x2* **assume** $x1 \leq_{L1} x2$
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* $(\leq_{R2\ (l1\ x1)\ (l1\ x2)})\ (g\ (l1\ x2))$
$f\ x1 \leq_{L2\ x1\ (\eta_1\ x2)}\ r2_{x1\ (l1\ x2)}\ (g\ (l1\ x2))$ **by** *auto*
with *L2-unit-le2* ‹$x1 \leq_{L1}\ x2$› **have** $f\ x1 \leq_{L2\ x1\ x2}\ r2_{x1\ (l1\ x2)}\ (g\ (l1\ x2))$ **by** *blast*
with *ge-L2-r2-le1* ‹$x1 \leq_{L1}\ x2$› ‹*in-codom* $(\leq_{R2\ (l1\ x1)\ (l1\ x2)})\ (g\ (l1\ x2))$›
have $f\ x1 \leq_{L2\ x1\ x2}\ r2_{x2\ (l1\ x2)}\ (g\ (l1\ x2))$ **by** *blast*
then show $f\ x1 \leq_{L2\ x1\ x2}\ r\ g\ x2$ **by** *simp*
qed

lemma *left-Galois-if-Dep-Fun-Rel-left-GaloisI*:
assumes $((\leq_{L1}) \Rrightarrow_m (\leq_{R1}))\ l1$
and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))\ l1\ r1$
and $\bigwedge x1\ x2.\ x1 \leq_{L1}\ x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2\ y'.\ x1 \leq_{L1}\ x2 \Longrightarrow$ *in-codom* $(\leq_{R2\ (l1\ x1)\ (l1\ x2)})\ y' \Longrightarrow$
$(\geq_{L2\ x1\ x2})\ (r2_{x1\ (l1\ x2)}\ y') \leq (\geq_{L2\ x1\ x2})\ (r2_{x2\ (l1\ x2)}\ y')$
and *in-codom* $(\leq_R)\ g$
and $((x\ x' :: (_{L1}\lessapprox))) \Rrightarrow (_{L2\ x\ x'}\lessapprox))\ f\ g$
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*: $((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))\ r1$
and *half-galois-prop-left2*: $\bigwedge x1'\ x2'.\ x1' \leq_{R1}\ x2' \Longrightarrow$
$((\leq_{L2\ (r1\ x1')\ (r1\ x2')})\ {}_h\trianglelefteq\ (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (r2_{(r1\ x1')\ x2'})$
and *R2-le1*: $\bigwedge x1'\ x2'.\ x1' \leq_{R1}\ x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and *R2-l2-le1*: $\bigwedge x1'\ x2'\ y.\ x1' \leq_{R1}\ x2' \Longrightarrow$ *in-dom* $(\leq_{L2\ (r1\ x1')\ (r1\ x2')})\ y \Longrightarrow$
$(\leq_{R2\ x1'\ x2'})\ (l2_{x2'\ (r1\ x1')}\ y) \leq (\leq_{R2\ x1'\ x2'})\ (l2_{x1'\ (r1\ x1')}\ y)$
and *rel-f-g*: $((x\ x' :: (_{L1}\lessapprox))) \Rrightarrow (_{L2\ x\ x'}\lessapprox))\ f\ g$
shows $l\ f \leq_R\ g$
proof (*rule flip.left-relI*)
fix $x1'\ x2'$ **assume** $x1' \leq_{R1}\ x2'$
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* ‹$x1' \leq_{R1}\ x2'$›]
have $l2_{x2'\ (r1\ x1')}\ (f\ (r1\ x1')) \leq_{R2\ (\varepsilon_1\ x1')\ x2'}\ g\ x2'$ **by** *auto*
with *R2-le1* ‹$x1' \leq_{R1}\ x2'$› **have** $l2_{x2'\ (r1\ x1')}\ (f\ (r1\ x1')) \leq_{R2\ x1'\ x2'}\ g\ x2'$
by *blast*
with *R2-l2-le1* ‹$x1' \leq_{R1}\ x2'$› ‹$f\ (r1\ x1')\ _{L2\ (r1\ x1')\ x2'}\lessapprox$ -›
have $l2_{x1'\ (r1\ x1')}\ (f\ (r1\ x1')) \leq_{R2\ x1'\ x2'}\ g\ x2'$ **by** *blast*
then show $l\ f\ x1' \leq_{R2\ x1'\ x2'}\ g\ x2'$ **by** *simp*
qed

lemma *left-Galois-if-Dep-Fun-Rel-left-GaloisI'*:
assumes $((\leq_{L1}) \Rrightarrow_m (\leq_{R1}))\ l1$ **and** $((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))\ r1$
and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))\ l1\ r1$

and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$((\leq_{L2\ (r1\ x1')\ (r1\ x2')})\ {}_h\trianglelefteq\ (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (r2_{(r1\ x1')\ x2'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1\ x2\ y'.\ x1 \leq_{L1} x2 \Longrightarrow$ *in-codom* $(\leq_{R2\ (l1\ x1)\ (l1\ x2)})\ y' \Longrightarrow$
$(\geq_{L2\ x1\ x2})\ (r2_{x1\ (l1\ x2)}\ y') \leq (\geq_{L2\ x1\ x2})\ (r2_{x2\ (l1\ x2)}\ y')$
and $\bigwedge x1'\ x2'\ y.\ x1' \leq_{R1} x2' \Longrightarrow$ *in-dom* $(\leq_{L2\ (r1\ x1')\ (r1\ x2')})\ y \Longrightarrow$
$(\leq_{R2\ x1'\ x2'})\ (l2_{x2'\ (r1\ x1')}\ y) \leq (\leq_{R2\ x1'\ x2'})\ (l2_{x1'\ (r1\ x1')}\ y)$
and $((x\ x' :: (_{L1}\lessapprox))) \Rrightarrow (_{L2\ x\ x'}\lessapprox))\ f\ g$
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 $((\leq_{L1}) \Rrightarrow_m (\leq_{R1}))\ l1$
and $((\leq_{L1}) \trianglelefteq (\leq_{R1}))\ l1\ r1$
and *reflexive-on* (*in-dom* $(\leq_{L1})$) $(\leq_{L1})$
and $\bigwedge x\ x'.\ x\ _{L1}\lessapprox\ x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rrightarrow_m (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2\ y'.\ x1 \leq_{L1} x2 \Longrightarrow$ *in-codom* $(\leq_{R2\ (l1\ x1)\ (l1\ x2)})\ y' \Longrightarrow$
$(\geq_{L2\ x1\ x2})\ (r2_{x1\ (l1\ x2)}\ y') \leq (\geq_{L2\ x1\ x2})\ (r2_{x2\ (l1\ x2)}\ y')$
and $\bigwedge x\ x'\ y'.\ x\ _{L1}\lessapprox\ x' \Longrightarrow$ *in-dom* $(\leq_{R2\ (l1\ x)\ x'})\ y' \Longrightarrow$
$(\geq_{L2\ x\ (r1\ x')})\ (r2_{x\ (l1\ x)}\ y') \leq (\geq_{L2\ x\ (r1\ x')})\ (r2_{x\ x'}\ y')$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $g \leq_R g$
shows $f\ _{L}\lessapprox\ g \longleftrightarrow ((x\ x' :: (_{L1}\lessapprox))) \Rrightarrow (_{L2\ x\ x'}\lessapprox))\ f\ g$
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 $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$
$((\text{in-codom}\ (\leq_{R2\ (l1\ x1)\ (l1\ x2)})) \Rrightarrow (\leq_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x2)})\ (r2_{x2\ (l1\ x2)})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
shows $\bigwedge x1\ x2\ y'.\ x1 \leq_{L1} x2 \Longrightarrow$ *in-codom* $(\leq_{R2\ (l1\ x1)\ (l1\ x2)})\ y' \Longrightarrow$
$(\geq_{L2\ x1\ x2})\ (r2_{x1\ (l1\ x2)}\ y') \leq (\geq_{L2\ x1\ x2})\ (r2_{x2\ (l1\ x2)}\ y')$
using *assms* **by** *blast*

lemma *left-Galois-iff-Dep-Fun-Rel-left-Galois-ge-left-rel2-assmI'*:
assumes $\bigwedge x\ x'.\ x\ _{L1}\lessapprox\ x' \Longrightarrow$
$((\text{in-dom}\ (\leq_{R2\ (l1\ x)\ x'})) \Rrightarrow (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ (l1\ x)})\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
shows $\bigwedge x\ x'\ y'.\ x\ _{L1}\lessapprox\ x' \Longrightarrow$ *in-dom* $(\leq_{R2\ (l1\ x)\ x'})\ y' \Longrightarrow$
$(\geq_{L2\ x\ (r1\ x')})\ (r2_{x\ (l1\ x)}\ y') \leq (\geq_{L2\ x\ (r1\ x')})\ (r2_{x\ x'}\ y')$
using *assms* **by** *blast*

lemma *left-Galois-iff-Dep-Fun-Rel-left-Galois-mono-assm-in-codom-rightI*:
assumes *mono-l1*: $((\leq_{L1}) \Rightarrow_m (\leq_{R1}))\ l1$
and *half-galois-prop-right1*: $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))\ l1\ r1$
and *refl-L1*: *reflexive-on* (*in-codom* $(\leq_{L1})$) $(\leq_{L1})$
and *L2-le-unit2*: $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and *mono-r2*: $((x1\ x2 :: (\leq_{L1})) \Rightarrow_m (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox x1') \Rightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rightarrow (\leq_{L2\ x1\ (r1\ x2')})\ r2$
and $x1 \leq_{L1} x2$
shows $((\textit{in-codom}\ (\leq_{R2\ (l1\ x1)\ (l1\ x2)})) \Rightarrow (\leq_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x2)})\ (r2_{x2\ (l1\ x2)})$
proof (*intro Fun-Rel-predI*)
from *mono-l1 half-galois-prop-right1 refl-L1* ‹$x1 \leq_{L1} x2$› **have** $l1\ x2 \leq_{R1} l1\ x2\ x2\ {}_{L1}\lessapprox l1\ x2$
by (*blast intro*: *t1.left-Galois-left-if-left-relI*)+
fix y' **assume** *in-codom* $(\leq_{R2\ (l1\ x1)\ (l1\ x2)})\ y'$
with *Dep-Fun-Rel-relD*[*OF*
dep-mono-wrt-relD[*OF mono-r2* ‹$x1 \leq_{L1} x2$›] ‹$l1\ x2 \leq_{R1} l1\ x2$›]
have $r2_{x1\ (l1\ x2)}\ y' \leq_{L2\ x1\ (\eta_1\ x2)} r2_{x2\ (l1\ x2)}\ y'$
using ‹$x2\ {}_{L1}\lessapprox l1\ x2$› **by** (*auto dest*: *in-field-if-in-codom*)
with *L2-le-unit2* ‹$x1 \leq_{L1} x2$› **show** $r2_{x1\ (l1\ x2)}\ y' \leq_{L2\ x1\ x2} r2_{x2\ (l1\ x2)}\ y'$
by *blast*
qed

lemma *left-Galois-iff-Dep-Fun-Rel-left-Galois-mono-assm-in-dom-rightI*:
assumes *mono-l1*: $((\leq_{L1}) \Rightarrow_m (\leq_{R1}))\ l1$
and *half-galois-prop-right1*: $((\leq_{L1}) \trianglelefteq (\leq_{R1}))\ l1\ r1$
and *refl-L1*: *reflexive-on* (*in-dom* $(\leq_{L1})$) $(\leq_{L1})$
and *mono-r2*: $((x1\ x2 :: (\leq_{L1})) \Rightarrow_m (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox x1') \Rightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rightarrow (\leq_{L2\ x1\ (r1\ x2')})\ r2$
and $x\ {}_{L1}\lessapprox x'$
shows $((\textit{in-dom}\ (\leq_{R2\ (l1\ x)\ x'})) \Rightarrow (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ (l1\ x)})\ (r2_{x\ x'})$
proof −
from *mono-l1 half-galois-prop-right1 refl-L1* ‹$x\ {}_{L1}\lessapprox x'$›
have $x \leq_{L1} x\ l1\ x \leq_{R1} x'\ x\ {}_{L1}\lessapprox l1\ x$
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* ‹$x \leq_{L1} x$›] ‹$l1\ x \leq_{R1} x'$›]
show *?thesis* **by** *blast*
qed

lemma *left-Galois-iff-Dep-Fun-Rel-left-Galois-if-monoI*:
assumes $((\leq_{L1}) \Rightarrow_m (\leq_{R1}))\ l1$
and $((\leq_{L1}) \trianglelefteq (\leq_{R1}))\ l1\ r1$
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rightarrow_m (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$

and $((x1\ x2 :: (\leq_{L1})) \Rrightarrow_m (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $g \leq_R g$
shows $f\ {}_L\lessapprox g \longleftrightarrow ((x\ x' :: ({}_{L1}\lessapprox)) \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
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* $(in\text{-}dom\ (\leq_{L1}))\ (\leq_{L1})$
and *mono-L2*: $((x1 : \top) \Rrightarrow_m (x2\ \text{-} :: (\leq_{L1}) \mid x1 \leq_{L1} x2) \Rrightarrow_m (\leq))\ L2$
and $x1 \leq_{L1} x2$
shows $(\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
proof −
from *refl-L1* ‹$x1 \leq_{L1} x2$› **have** $x1 \leq_{L1} x1$ **by** *blast*
with *dep-mono-wrt-relD*[*OF dep-mono-wrt-predD*[*OF mono-L2*] ‹$x1 \leq_{L1} x2$›]
show $(\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$ **by** *auto*
qed

lemma *left-Galois-iff-Dep-Fun-Rel-left-Galois-left-rel2-unit1-le-assmI*:
assumes *mono-l1*: $((\leq_{L1}) \Rrightarrow_m (\leq_{R1}))\ l1$
and *half-galois-prop-right1*: $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))\ l1\ r1$
and *refl-L1*: *reflexive-on* $(in\text{-}codom\ (\leq_{L1}))\ (\leq_{L1})$
and *antimono-L2*:
$((x1 : \top) \Rrightarrow_m (x2\ x3 :: (\leq_{L1}) \mid (x1 \leq_{L1} x2 \wedge x3 \leq_{L1} \eta_1\ x2)) \Rrightarrow_m (\geq))\ L2$
and $x1 \leq_{L1} x2$
shows $(\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
proof −
from *mono-l1 half-galois-prop-right1 refl-L1* ‹$x1 \leq_{L1} x2$› **have** $x2 \leq_{L1} \eta_1\ x2$
by (*blast intro*: *t1.rel-unit-if-reflexive-on-if-half-galois-prop-right-if-mono-wrt-rel*)
with *refl-L1* **have** $\eta_1\ x2 \leq_{L1} \eta_1\ x2$ **by** *blast*
with *dep-mono-wrt-relD*[*OF dep-mono-wrt-predD*[*OF antimono-L2*] ‹$x2 \leq_{L1} \eta_1\ x2$›]
show $(\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$ **using** ‹$x1 \leq_{L1} x2$› **by** *auto*
qed

lemma *left-Galois-iff-Dep-Fun-Rel-left-Galois-if-monoI'*:
assumes $((\leq_{L1}) \Rrightarrow_m (\leq_{R1}))\ l1$
and $((\leq_{L1}) \trianglelefteq (\leq_{R1}))\ l1\ r1$
and *reflexive-on* $(in\text{-}field\ (\leq_{L1}))\ (\leq_{L1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rrightarrow_m (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $((x1 : \top) \Rrightarrow_m (x2\ \text{-} :: (\leq_{L1}) \mid x1 \leq_{L1} x2) \Rrightarrow_m (\leq))\ L2$
and $((x1 : \top) \Rrightarrow_m (x2\ x3 :: (\leq_{L1}) \mid (x1 \leq_{L1} x2 \wedge x3 \leq_{L1} \eta_1\ x2)) \Rrightarrow_m (\geq))\ L2$
and $((x1\ x2 :: (\leq_{L1})) \Rrightarrow_m (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox x1') \Rrightarrow$

in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')})$) *r2*
and $\bigwedge$*x1 x2*. *x1* $\leq_{L1}$ *x2* $\Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $g \leq_R g$
shows $f\ {}_L\lessapprox\ g \longleftrightarrow ((x\ x' :: ({}_{L1}\lessapprox)) \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
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 $((\leq_{L1}) \dashv (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rrightarrow_m (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $((x1 : \top) \Rrightarrow_m (x2\ \text{-} :: (\leq_{L1}) \mid x1 \leq_{L1} x2) \Rrightarrow_m (\leq))$ *L2*
and $((x1 : \top) \Rrightarrow_m (x2\ x3 :: (\leq_{L1}) \mid (x1 \leq_{L1} x2 \wedge x3 \leq_{L1} \eta_1\ x2)) \Rrightarrow_m (\geq))$ *L2*
and $((x1\ x2 :: (\leq_{L1})) \Rrightarrow_m (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')})$) *r2*
and $\bigwedge$*x1 x2*. *x1* $\leq_{L1}$ *x2* $\Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $g \leq_R g$
shows $f\ {}_L\lessapprox\ g \longleftrightarrow ((x\ x' :: ({}_{L1}\lessapprox)) \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
using *assms* **by** (*intro left-Galois-iff-Dep-Fun-Rel-left-Galois-if-monoI'*) *auto*

interpretation *flip-inv* : *galois* $(\geq_{R1})$ $(\geq_{L1})$ *r1 l1* **.**

lemma *left-Galois-iff-Dep-Fun-Rel-left-Galois-left-rel2-unit1-le-assm-if-galois-equivI*:
assumes *galois-equiv1*: $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1*
and *refl-L1*: *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and *mono-L2*: $((x1 : \top) \Rrightarrow_m (x2\ \text{-} :: (\leq_{L1}) \mid x1 \leq_{L1} x2) \Rrightarrow_m (\leq))$ *L2*
and *x1* $\leq_{L1}$ *x2*
shows $(\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
proof −
from *refl-L1* ‹*x1* $\leq_{L1}$ *x2*› **have** *x1* $\leq_{L1}$ *x1* **by** *blast*
from *galois-equiv1 refl-L1* ‹*x1* $\leq_{L1}$ *x2*› **have** η_1 *x2* $\leq_{L1}$ *x2* **by** (*intro*
flip.t1.counit-rel-if-reflexive-on-if-half-galois-prop-left-if-mono-wrt-rel)
blast+
have *x1* $\leq_{L1}$ η_1 *x2* **by** (*rule t1.rel-unit-if-left-rel-if-mono-wrt-relI*)
(*insert galois-equiv1 refl-L1* ‹*x1* $\leq_{L1}$ *x2*›, *auto*)
with *dep-mono-wrt-relD*[*OF dep-mono-wrt-predD*[*OF mono-L2*] ‹η_1 *x2* $\leq_{L1}$ *x2*›]
show $(\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$ **by** *auto*
qed

lemma *left-Galois-iff-Dep-Fun-Rel-left-Galois-if-galois-equivalenceI*:
assumes $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rrightarrow_m (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $((x1 : \top) \Rrightarrow_m (x2\ \text{-} :: (\leq_{L1}) \mid x1 \leq_{L1} x2) \Rrightarrow_m (\leq))$ *L2*
and $((x1\ x2 :: (\leq_{L1})) \Rrightarrow_m (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1') \Rrightarrow$

in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')})\ r2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $g \leq_R g$
shows $f\ {}_L\lessapprox\ g \longleftrightarrow ((x\ x' :: ({}_{L1}\lessapprox)) \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
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 $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rrightarrow_m (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
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-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $g \leq_R g$
shows $f\ {}_L\lessapprox\ g \longleftrightarrow ((x\ x' :: ({}_{L1}\lessapprox)) \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
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 $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rrightarrow_m (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
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-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $g \leq_R g$
shows $f\ {}_L\lessapprox\ g \longleftrightarrow ((x\ x' :: ({}_{L1}\lessapprox)) \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
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 $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_{pre} (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $((x1\ x2 :: (\geq_{L1})) \Rrightarrow_m (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rrightarrow (\leq))\ L2$
and $((x1\ x2 :: (\leq_{L1})) \Rrightarrow_m (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $g \leq_R g$
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 $((\leq_{L1}) \Rightarrow_m (\leq_{R1}))$ *l1* **and** $((\leq_{R1}) \Rightarrow_m (\leq_{L1}))$ *r1*
and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$((\leq_{L2\ (r1\ x1')\ (r1\ x2')})\ {}_h\trianglelefteq\ (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (r2_{(r1\ x1')\ x2'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'\ y.\ x1' \leq_{R1} x2' \Longrightarrow$ *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 $\bigwedge x1\ x2\ y'.\ x1 \leq_{L1} x2 \Longrightarrow$ *in-codom* $(\leq_{R2\ (l1\ x1)\ (l1\ x2)})\ y' \Longrightarrow$
$(\geq_{L2\ x1\ x2})\ (r2_{x1\ (l1\ x2)}\ y') \leq (\geq_{L2\ x1\ x2})\ (r2_{x2\ (l1\ x2)}\ y')$
shows $((x\ x' :: ({}_{L1}\lessapprox))) \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\restriction_{\textit{in-dom}\ (\leq_L)}{}^{\uparrow}{}_{\textit{in-codom}\ (\leq_R)}$
$= ((x\ x' :: ({}_{L1}\lessapprox))) \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))$
using *assms* **by** (*intro ext iffI rel-restrict-leftI rel-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 $((\leq_{L1}) \dashv (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and *reflexive-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$((\leq_{L2\ (r1\ x1')\ (r1\ x2')})\ {}_h\trianglelefteq\ (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (r2_{(r1\ x1')\ x2'})$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow_m (x3\ x4 :: (\leq_{L1}) \mid (x2 \leq_{L1} x3 \wedge x4 \leq_{L1} \eta_1\ x3)) \Rrightarrow (\geq))\ L2$
and $((x1'\ x2' :: (\leq_{R1}) \mid \varepsilon_1\ x2' \leq_{R1} x1') \Rightarrow_m (x3'\ x4' :: (\leq_{R1}) \mid x2' \leq_{R1} x3') \Rrightarrow (\leq))\ R2$
and $((x1'\ x2' :: (\leq_{R1})) \Rightarrow_m (x1\ x2 :: (\leq_{L1}) \mid x2\ {}_{L1}\lessapprox\ x1') \Rrightarrow$
in-field $(\leq_{L2\ x1\ (r1\ x2')}) \Rrightarrow (\leq_{R2\ (l1\ x1)\ x2'}))\ l2$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow_m (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$ *transitive* $(\leq_{R2\ x1'\ x2'})$
shows $((x\ x' :: ({}_{L1}\lessapprox))) \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))\restriction_{\textit{in-dom}\ (\leq_L)}{}^{\uparrow}{}_{\textit{in-codom}\ (\leq_R)}$
$= ((x\ x' :: ({}_{L1}\lessapprox))) \Rrightarrow ({}_{L2\ x\ x'}\lessapprox))$
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 $((\leq_{L1}) \ {}_h\trianglelefteq \ (\leq_{R1}))$ *l1 r1*
shows $((x\ x' :: (_{L1}\overset{\leq}{\approx})) \Rrightarrow (S\ x\ x')\restriction_{in\text{-}dom\ (\leq_{L2}\ x\ (r1\ x'))}\rceil_{in\text{-}codom\ (\leq_{R2}\ (l1\ x)\ x')})$
$\lceil_{in\text{-}dom\ (\leq_L)}\rceil_{in\text{-}codom\ (\leq_R)} =$
$((x\ x' :: (_{L1}\overset{\leq}{\approx})) \Rrightarrow S\ x\ x')\restriction_{in\text{-}dom\ (\leq_L)}\rceil_{in\text{-}codom\ (\leq_R)}$ (**is** *?lhs = ?rhs*)
proof −
have *?lhs* =
$((x\ x' :: (_{L1}\overset{\leq}{\approx})) \Rrightarrow (S\ x\ x')\rceil_{in\text{-}codom\ (\leq_{R2}\ (l1\ x)\ x')})$
$\lceil_{in\text{-}dom\ (\leq_L)}\rceil_{in\text{-}codom\ (\leq_R)}$
by (*subst rel-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 rel-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: *rel-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 $((\leq_{L1}) \ {}_h\trianglelefteq \ (\leq_{R1}))$ *l1 r1*
and *reflexive-on* $(in\text{-}dom\ (\leq_{L1}))\ (\leq_{L1})$
and $((\leq_{R2}) \Rrightarrow_m (\leq_{L2}))$ *r2*
and *transitive* $(\leq_{L2})$
and $g \leq_R g$
and $f\ {}_L\overset{\leq}{\approx}\ g$
shows $((_{L1}\overset{\leq}{\approx}) \Rrightarrow (_{L2}\overset{\leq}{\approx}))\ f\ g$
by (*urule tdfr.Dep-Fun-Rel-left-Galois-if-left-GaloisI assms* | *simp*)+

corollary *left-Galois-if-Fun-Rel-left-GaloisI*:
assumes $((\leq_{L1}) \Rrightarrow_m (\leq_{R1}))$ *l1*
and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*
and *in-codom* $(\leq_R)$ *g*
and $((_{L1}\overset{\leq}{\approx}) \Rrightarrow (_{L2}\overset{\leq}{\approx}))\ f\ g$
shows $f\ {}_L\overset{\leq}{\approx}\ g$
by (*urule tdfr.left-Galois-if-Dep-Fun-Rel-left-GaloisI assms* | *simp*)+

lemma *left-Galois-if-Fun-Rel-left-GaloisI'*:
assumes $((\leq_{L1}) \Rrightarrow_m (\leq_{R1}))$ *l1* **and** $((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))$ *r1*

and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*
and $((\leq_{L2}) \ {}_h\trianglelefteq (\leq_{R2}))$ *l2 r2*
and $((_{L1}\lessapprox) \Rrightarrow (_{L2}\lessapprox))$ *f g*
shows $f \ {}_{L}\lessapprox \ g$
by (*urule tdfr.left-Galois-if-Dep-Fun-Rel-left-GaloisI'* *assms* | *simp*)+

corollary *left-Galois-iff-Fun-Rel-left-GaloisI*:
assumes $((\leq_{L1}) \Rrightarrow_m (\leq_{R1}))$ *l1*
and $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-dom* $(\leq_{L1})$) $(\leq_{L1})$
and $((\leq_{R2}) \Rrightarrow_m (\leq_{L2}))$ *r2*
and *transitive* $(\leq_{L2})$
and $g \leq_R g$
shows $f \ {}_{L}\lessapprox \ g \longleftrightarrow ((_{L1}\lessapprox) \Rrightarrow (_{L2}\lessapprox)) \ f \ g$
by (*urule tdfr.left-Galois-iff-Dep-Fun-Rel-left-GaloisI* *assms* | *simp*)+

Simplification of Restricted Function Relator **lemma** *Fun-Rel-left-Galois-restrict-left-right-eq-Fun*
assumes $((\leq_{L1}) \Rrightarrow_m (\leq_{R1}))$ *l1* **and** $((\leq_{R1}) \Rrightarrow_m (\leq_{L1}))$ *r1*
and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*
and $((\leq_{L2}) \ {}_h\trianglelefteq (\leq_{R2}))$ *l2 r2*
shows $((_{L1}\lessapprox) \Rrightarrow (_{L2}\lessapprox))\restriction_{in\text{-}dom\ (\leq_L)}\restriction_{in\text{-}codom\ (\leq_R)} = ((_{L1}\lessapprox) \Rrightarrow (_{L2}\lessapprox))$
by (*urule tdfr.Dep-Fun-Rel-left-Galois-restrict-left-right-eq-Dep-Fun-Rel-left-GaloisI*
assms
| *simp*)+

Simplification of Restricted Function Relator for Nested Transports

lemma *Fun-Rel-left-Galois-restrict-left-right-restrict-left-right-eq*:
fixes $S :: 'b1 \Rightarrow 'b2 \Rightarrow bool$
assumes $((\leq_{L1}) \ {}_h\trianglelefteq (\leq_{R1}))$ *l1 r1*
shows $((_{L1}\lessapprox) \Rrightarrow S\restriction_{in\text{-}dom\ (\leq_{L2})}\restriction_{in\text{-}codom\ (\leq_{R2})})\restriction_{in\text{-}dom\ (\leq_L)}\restriction_{in\text{-}codom\ (\leq_R)}$
=
$((_{L1}\lessapprox) \Rrightarrow S)\restriction_{in\text{-}dom\ (\leq_L)}\restriction_{in\text{-}codom\ (\leq_R)}$
by (*urule tdfr.Dep-Fun-Rel-left-Galois-restrict-left-right-restrict-left-right-eq* *assms*
| *simp*)+

end

Monotone Dependent Function Relator **context** *transport-Mono-Dep-Fun-Rel*
begin

lemma *Dep-Fun-Rel-left-Galois-if-left-GaloisI*:
assumes $((\leq_{L1}) \ {}_h\trianglelefteq (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-dom* $(\leq_{L1})$) $(\leq_{L1})$
and $\bigwedge x\ x'.\ x \ {}_{L1}\lessapprox \ x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rrightarrow_m (\leq_{L2\ x\ (r1\ x')})) \ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x\ x'\ y'.\ x \ {}_{L1}\lessapprox \ x' \Longrightarrow in\text{-}dom\ (\leq_{R2\ (l1\ x)\ x'})\ y' \Longrightarrow$
$(\geq_{L2\ x\ (r1\ x')}) \ (r2_{x\ (l1\ x)\ y'}) \leq (\geq_{L2\ x\ (r1\ x')}) \ (r2_{x\ x'\ y'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow transitive\ (\leq_{L2\ x1\ x2})$

and $f\ {}_{L}\lessapprox\ g$
shows $((x\ x' :: ({}_{L1}\lessapprox)) \Rightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
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 $(tdfr.R \Rightarrow_m tdfr.L)\ r$
and $((\leq_{L1}) \Rightarrow_m (\leq_{R1}))\ l1$
and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))\ l1\ r1$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2\ y'.\ x1 \leq_{L1} x2 \Longrightarrow in\text{-}codom\ (\leq_{R2\ (l1\ x1)\ (l1\ x2)})\ y' \Longrightarrow$
$(\geq_{L2\ x1\ x2})\ (r2_{x1\ (l1\ x2)}\ y') \leq (\geq_{L2\ x1\ x2})\ (r2_{x2\ (l1\ x2)}\ y')$
and $in\text{-}dom\ (\leq_L)\ f$
and $in\text{-}codom\ (\leq_R)\ g$
and $((x\ x' :: ({}_{L1}\lessapprox)) \Rightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
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 $(tdfr.R \Rightarrow_m tdfr.L)\ r$
and $((\leq_{L1}) \Rightarrow_m (\leq_{R1}))\ l1$
and $((\leq_{L1}) \trianglelefteq (\leq_{R1}))\ l1\ r1$
and $reflexive\text{-}on\ (in\text{-}field\ (\leq_{L1}))\ (\leq_{L1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rightarrow_m (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow_m (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox\ x1') \Rightarrow$
$in\text{-}field\ (\leq_{R2\ (l1\ x1)\ x2'}) \Rightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow transitive\ (\leq_{L2\ x1\ x2})$
and $in\text{-}dom\ (\leq_L)\ f$
and $in\text{-}codom\ (\leq_R)\ g$
shows $f\ {}_{L}\lessapprox\ g \longleftrightarrow ((x\ x' :: ({}_{L1}\lessapprox)) \Rightarrow ({}_{L2\ x\ x'}\lessapprox))\ f\ g$
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*: $((\leq_{L1}) \dashv (\leq_{R1}))\ l1\ r1$
and *refl-L1*: $reflexive\text{-}on\ (in\text{-}field\ (\leq_{L1}))\ (\leq_{L1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox\ x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rightarrow_m (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$

and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
and *L2-le-unit2*: $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and *mono-r2*: $((x1\ x2 :: (\leq_{L1})) \Rightarrow_m (x1'\ x2' :: (\leq_{R1}) \mid x2\ _{L1}\lessapprox x1') \Rightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and *trans-L2*: $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and *in-dom* $(\leq_L)\ f$
and *in-codom* $(\leq_R)\ g$
shows $f\ _L\lessapprox\ g \longleftrightarrow ((x\ x' :: (_{L1}\lessapprox)) \Rightarrow (_{L2\ x\ x'}\lessapprox))\ f\ g$ (**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof −
have $(\leq_{L2\ x1\ x2})\ (r2_{x1\ (l1\ x2)}\ y') \leq (\leq_{L2\ x1\ x2})\ (r2_{x1\ (l1\ x1)}\ y')$
if *hyps*: *x1* $\leq_{L1}$ *x2 in-dom* $(\leq_{R2\ (l1\ x1)\ (l1\ x2)})\ y'$ **for** *x1 x2 y'*
proof −
have $((in\text{-}dom\ (\leq_{R2\ (l1\ x1)\ (l1\ x2)})) \Rightarrow (\leq_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x1)})\ (r2_{x1\ (l1\ x2)})$
proof (*intro Fun-Rel-predI*)
from *galois-conn1 refl-L1* ‹*x1* $\leq_{L1}$ *x2*›
have *x1* $\leq_{L1}$ *x1 l1 x1* $\leq_{R1}$ *l1 x2 x1* $_{L1}\lessapprox$ *l1 x1*
by (*blast intro*: *t1.left-Galois-left-if-left-relI*)+
fix *y'* **assume** *in-dom* $(\leq_{R2\ (l1\ x1)\ (l1\ x2)})\ y'$
with *Dep-Fun-Rel-relD*[*OF dep-mono-wrt-relD*[*OF mono-r2* ‹*x1* $\leq_{L1}$ *x1*›]
‹*l1 x1* $\leq_{R1}$ *l1 x2*›]
have $r2_{x1\ (l1\ x1)}\ y' \leq_{L2\ x1\ (\eta_1\ x2)}\ r2_{x1\ (l1\ x2)}\ y'$
using ‹*x1* $_{L1}\lessapprox$ *l1 x1*› **by** (*auto dest*: *in-field-if-in-dom*)
with *L2-le-unit2* ‹*x1* $\leq_{L1}$ *x2*› **show** $r2_{x1\ (l1\ x1)}\ y' \leq_{L2\ x1\ x2}\ r2_{x1\ (l1\ x2)}\ y'$
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 $((\leq_{L1}) \dashv (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and $\bigwedge x\ x'.\ x\ _{L1}\lessapprox\ x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rightarrow_m (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $((x1 : \top) \Rightarrow_m (x2\ \text{-} :: (\leq_{L1}) \mid x1 \leq_{L1} x2) \Rightarrow_m (\leq))\ L2$
and $((x1 : \top) \Rightarrow_m (x2\ x3 :: (\leq_{L1}) \mid (x1 \leq_{L1} x2 \wedge x3 \leq_{L1} \eta_1\ x2)) \Rightarrow_m (\geq))\ L2$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow_m (x1'\ x2' :: (\leq_{R1}) \mid x2\ _{L1}\lessapprox\ x1') \Rightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and *in-dom* $(\leq_L)\ f$

and *in-codom* $(\leq_R)$ g
shows $f\ {}_L\!\lessapprox\ g \longleftrightarrow ((x\ x' :: ({}_{L1}\!\lessapprox)) \Rrightarrow ({}_{L2\ x\ x'}\!\lessapprox))\ f\ g$ (**is** *?lhs* $\longleftrightarrow$ *?rhs*)
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 $((\leq_{L1}) \dashv (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\!\lessapprox\ x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rrightarrow_m (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
and $((x1 : \top) \Rrightarrow_m (x2 \text{-} :: (\leq_{L1}) \mid x1 \leq_{L1} x2) \Rrightarrow_m (\leq))\ L2$
and $((x1 : \top) \Rrightarrow_m (x2\ x3 :: (\leq_{L1}) \mid (x1 \leq_{L1} x2 \wedge x3 \leq_{L1} \eta_1\ x2)) \Rrightarrow_m (\geq))\ L2$
and $((x1\ x2 :: (\leq_{L1})) \Rrightarrow_m (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\!\lessapprox\ x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
shows $({}_L\!\lessapprox) = ((x\ x' :: ({}_{L1}\!\lessapprox)) \Rrightarrow ({}_{L2\ x\ x'}\!\lessapprox))\restriction_{\textit{in-dom}\ (\leq_L)}\restriction_{\textit{in-codom}\ (\leq_R)}$
using *assms* **by** (*intro ext iffI rel-restrict-leftI rel-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 $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\!\lessapprox\ x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rrightarrow_m (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
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-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and *in-dom* $(\leq_L)$ f
and *in-codom* $(\leq_R)$ g
shows $f\ {}_L\!\lessapprox\ g \longleftrightarrow ((x\ x' :: ({}_{L1}\!\lessapprox)) \Rrightarrow ({}_{L2\ x\ x'}\!\lessapprox))\ f\ g$
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 $((\leq_{L1}) \equiv_G (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\!\lessapprox\ x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rrightarrow_m (\leq_{L2\ x\ (r1\ x')}))\ (r2_{x\ x'})$
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-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$

shows $(_{L}\lessapprox) = ((x\ x' :: (_{L1}\lessapprox)) \Rightarrow (_{L2\ x\ x'}\lessapprox))\restriction_{in\text{-}dom\ (\leq_L)}\restriction_{in\text{-}codom\ (\leq_R)}$
using *assms* **by** (*intro ext iffI rel-restrict-leftI rel-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 $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and $\bigwedge x\ x'.\ x\ _{L1}\lessapprox\ x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rrightarrow_m (\leq_{L2\ x\ (r1\ x')})) (r2_{x\ x'})$
and $((x1\ x2 :: (\geq_{L1})) \Rrightarrow_m (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rightarrow (\leq))\ L2$
and $((x1\ x2 :: (\leq_{L1})) \Rrightarrow_m (x1'\ x2' :: (\leq_{R1}) \mid x2\ _{L1}\lessapprox\ x1') \Rightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and *in-dom* $(\leq_L)\ f$
and *in-codom* $(\leq_R)\ g$
shows $f\ _{L}\lessapprox\ g \longleftrightarrow ((x\ x' :: (_{L1}\lessapprox)) \Rightarrow (_{L2\ x\ x'}\lessapprox))\ f\ g$
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 $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and $\bigwedge x\ x'.\ x\ _{L1}\lessapprox\ x' \Longrightarrow ((\leq_{R2\ (l1\ x)\ x'}) \Rrightarrow_m (\leq_{L2\ x\ (r1\ x')})) (r2_{x\ x'})$
and $((x1\ x2 :: (\geq_{L1})) \Rrightarrow_m (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rightarrow (\leq))\ L2$
and $((x1\ x2 :: (\leq_{L1})) \Rrightarrow_m (x1'\ x2' :: (\leq_{R1}) \mid x2\ _{L1}\lessapprox\ x1') \Rightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
shows $(_{L}\lessapprox) = ((x\ x' :: (_{L1}\lessapprox)) \Rightarrow (_{L2\ x\ x'}\lessapprox))\restriction_{in\text{-}dom\ (\leq_L)}\restriction_{in\text{-}codom\ (\leq_R)}$
using *assms* **by** (*intro ext iffI rel-restrict-leftI rel-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 $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and $\bigwedge x\ x'.\ x\ _{L1}\lessapprox\ x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_{pre} (\leq_{R2\ (l1\ x)\ x'})) (l2_{x'\ x}) (r2_{x\ x'})$
and $((x1\ x2 :: (\geq_{L1})) \Rrightarrow_m (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rightarrow (\leq))\ L2$
and $((x1\ x2 :: (\leq_{L1})) \Rrightarrow_m (x1'\ x2' :: (\leq_{R1}) \mid x2\ _{L1}\lessapprox\ x1') \Rightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and *in-dom* $(\leq_L)\ f$
and *in-codom* $(\leq_R)\ g$
shows $f\ _{L}\lessapprox\ g \longleftrightarrow ((x\ x' :: (_{L1}\lessapprox)) \Rightarrow (_{L2\ x\ x'}\lessapprox))\ f\ g$
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 $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and $\bigwedge x\ x'.\ x\ _{L1}\lessapprox\ x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_{pre} (\leq_{R2\ (l1\ x)\ x'})) (l2_{x'\ x}) (r2_{x\ x'})$
and $((x1\ x2 :: (\geq_{L1})) \Rrightarrow_m (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rightarrow (\leq))\ L2$

and $((x1\ x2 :: (\leq_{L1})) \Rightarrow_m (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lesssim\!\!\approx\ x1') \Rightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
shows $({}_{L}\lesssim\!\!\approx) = ((x\ x' :: ({}_{L1}\lesssim\!\!\approx)) \Rightarrow ({}_{L2\ x\ x'}\lesssim\!\!\approx))\restriction_{in\text{-}dom\ (\leq_L)}{}^{\uparrow}{}_{in\text{-}codom\ (\leq_R)}$
using *assms* **by** (*intro ext iffI rel-restrict-leftI rel-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-field* $(\leq_{L1})$) $(\leq_{L1})$
and *reflexive-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$((\leq_{L2\ (r1\ x1')\ (r1\ x2')})\ {}_{h}\trianglelefteq\ (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (r2_{(r1\ x1')\ x2'})$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow_m (x3\ x4 :: (\leq_{L1}) \mid (x2 \leq_{L1} x3 \wedge x4 \leq_{L1} \eta_1\ x3)) \Rightarrow (\geq))\ L2$
and $((x1'\ x2' :: (\leq_{R1}) \mid \varepsilon_1\ x2' \leq_{R1} x1') \Rightarrow_m (x3'\ x4' :: (\leq_{R1}) \mid x2' \leq_{R1} x3') \Rightarrow (\leq))\ R2$
and $((x1'\ x2' :: (\leq_{R1})) \Rightarrow_m (x1\ x2 :: (\leq_{L1}) \mid x2\ {}_{L1}\lesssim\!\!\approx\ x1') \Rightarrow$
in-field $(\leq_{L2\ x1\ (r1\ x2')}) \Rightarrow (\leq_{R2\ (l1\ x1)\ x2'}))\ l2$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow_m (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lesssim\!\!\approx\ x1') \Rightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
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 $((x\ x' :: ({}_{L1}\lesssim\!\!\approx)) \Rightarrow ({}_{L2\ x\ x'}\lesssim\!\!\approx))\restriction_{in\text{-}dom\ (\leq_L)}{}^{\uparrow}{}_{in\text{-}codom\ (\leq_R)}$
$= ((x\ x' :: ({}_{L1}\lesssim\!\!\approx)) \Rightarrow ({}_{L2\ x\ x'}\lesssim\!\!\approx))$
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 $((x1\ x2 :: (\geq_{L1})) \Rightarrow_m (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rightarrow (\leq))\ L2$
and $((x1'\ x2' :: (\geq_{R1})) \Rightarrow_m (x3'\ x4' :: (\leq_{R1}) \mid x1' \leq_{R1} x3') \Rightarrow (\leq))\ R2$
and $((x1'\ x2' :: (\leq_{R1})) \Rightarrow_m (x1\ x2 :: (\leq_{L1}) \mid x2\ {}_{L1}\lesssim\!\!\approx\ x1') \Rightarrow$
in-field $(\leq_{L2\ x1\ (r1\ x2')}) \Rightarrow (\leq_{R2\ (l1\ x1)\ x2'}))\ l2$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow_m (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lesssim\!\!\approx\ x1') \Rightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
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 $((x\ x' :: ({}_{L1}\lesssim\!\!\approx)) \Rightarrow ({}_{L2\ x\ x'}\lesssim\!\!\approx))\restriction_{in\text{-}dom\ (\leq_L)}{}^{\uparrow}{}_{in\text{-}codom\ (\leq_R)}$

$= ((x\ x' :: (_{L1}\lessapprox)) \Rrightarrow (_{L2\ x\ x'}\lessapprox))$
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 $((\leq_{L1})\ {}_h\trianglelefteq\ (\leq_{R1}))$ *l1 r1*
shows $((x\ x' :: (_{L1}\lessapprox)) \Rrightarrow (S\ x\ x')\restriction_{in\text{-}dom\ (\leq_{L2\ x\ (r1\ x')})}\restriction_{in\text{-}codom\ (\leq_{R2\ (l1\ x)\ x'})}$
$\restriction_{in\text{-}dom\ (\leq_L)}\restriction_{in\text{-}codom\ (\leq_R)} =$
$((x\ x' :: (_{L1}\lessapprox)) \Rrightarrow S\ x\ x')\restriction_{in\text{-}dom\ (\leq_L)}\restriction_{in\text{-}codom\ (\leq_R)}$
(**is** $?lhs\restriction_{?DL}\restriction_{?CR} = ?rhs\restriction_{?DL}\restriction_{?CR}$)
proof (*intro ext*)
fix *f g*
have $?lhs\restriction_{?DL}\restriction_{?CR}\ f\ g \longleftrightarrow ?lhs\ f\ g \wedge ?DL\ f \wedge ?CR\ g$ **by** *blast*
also have ... $\longleftrightarrow ?lhs\restriction_{in\text{-}dom\ tdfr.L}\restriction_{in\text{-}codom\ tdfr.R}\ f\ g \wedge ?DL\ f \wedge ?CR\ g$
unfolding *left-rel-eq-tdfr-left-Refl-Rel right-rel-eq-tdfr-right-Refl-Rel*
by *blast*
also with *assms* **have** ... $\longleftrightarrow ?rhs\restriction_{in\text{-}dom\ tdfr.L}\restriction_{in\text{-}codom\ tdfr.R}\ f\ g \wedge ?DL\ f \wedge ?CR\ g$
by (*simp only*:
tdfr.Dep-Fun-Rel-left-Galois-restrict-left-right-restrict-left-right-eq)
also have ... $\longleftrightarrow ?rhs\restriction_{?DL}\restriction_{?CR}\ f\ g$
unfolding *left-rel-eq-tdfr-left-Refl-Rel right-rel-eq-tdfr-right-Refl-Rel*
by *blast*
finally show $?lhs\restriction_{?DL}\restriction_{?CR}\ f\ g \longleftrightarrow ?rhs\restriction_{?DL}\restriction_{?CR}\ f\ g$.
qed

end

Monotone Function Relator **context** *transport-Mono-Fun-Rel* **begin**

corollary *Fun-Rel-left-Galois-if-left-GaloisI*:
assumes $((\leq_{L1})\ {}_h\trianglelefteq\ (\leq_{R1}))$ *l1 r1*
and *reflexive-on* $(in\text{-}dom\ (\leq_{L1}))\ (\leq_{L1})$
and $((\leq_{R2}) \Rrightarrow_m (\leq_{L2}))\ (r2)$
and *transitive* $(\leq_{L2})$
and $f\ {}_L\lessapprox\ g$
shows $((_{L1}\lessapprox) \Rrightarrow (_{L2}\lessapprox))\ f\ g$
by (*urule tpdfr.Dep-Fun-Rel-left-Galois-if-left-GaloisI assms* | *simp*)+

interpretation *flip* : *transport-Mono-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2* .

lemma *left-Galois-if-Fun-Rel-left-GaloisI*:
assumes $((\leq_{L1}) \Rightarrow_m (\leq_{R1}))$ *l1*
and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*
and $((\leq_{R2}) \Rightarrow_m (\leq_{L2}))$ *r2*
and *in-dom* $(\leq_L)$ *f*
and *in-codom* $(\leq_R)$ *g*
and $((_{L1}\lessapprox) \Rightarrow (_{L2}\lessapprox))$ *f g*
shows *f* $_{L}\lessapprox$ *g*
by (*urule tpdfr.left-Galois-if-Dep-Fun-Rel-left-GaloisI flip.tfr.mono-wrt-rel-leftI*)+
(*urule assms* | *simp*)+

corollary *left-Galois-iff-Fun-Rel-left-GaloisI*:
assumes $((\leq_{L1}) \Rightarrow_m (\leq_{R1}))$ *l1*
and $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-dom* $(\leq_{L1})$) $(\leq_{L1})$
and $((\leq_{R2}) \Rightarrow_m (\leq_{L2}))$ $(r2)$
and *transitive* $(\leq_{L2})$
and *in-dom* $(\leq_L)$ *f*
and *in-codom* $(\leq_R)$ *g*
shows *f* $_{L}\lessapprox$ *g* $\longleftrightarrow$ $((_{L1}\lessapprox) \Rightarrow (_{L2}\lessapprox))$ *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 $((\leq_{L1}) \Rightarrow_m (\leq_{R1}))$ *l1*
and $((\leq_{L1}) \trianglelefteq (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-dom* $(\leq_{L1})$) $(\leq_{L1})$
and $((\leq_{R2}) \Rightarrow_m (\leq_{L2}))$ *r2*
and *transitive* $(\leq_{L2})$
shows $(_{L}\lessapprox) = ((_{L1}\lessapprox) \Rightarrow (_{L2}\lessapprox))\restriction_{\textit{in-dom}\ (\leq_L)}\restriction^{\textit{in-codom}\ (\leq_R)}$
using *assms* **by** (*intro ext iffI rel-restrict-leftI rel-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 $((\leq_{L1}) \Rightarrow_m (\leq_{R1}))$ *l1* **and** $((\leq_{R1}) \Rightarrow_m (\leq_{L1}))$ *r1*
and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*
and $((\leq_{L2}) \ {}_h\trianglelefteq (\leq_{R2}))$ *l2 r2*
shows $((_{L1}\lessapprox) \Rightarrow (_{L2}\lessapprox))\restriction_{\textit{in-dom}\ (\leq_L)}\restriction^{\textit{in-codom}\ (\leq_R)} = ((_{L1}\lessapprox) \Rightarrow (_{L2}\lessapprox))$
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 $((\leq_{L1}) \Rightarrow_m (\leq_{R1}))$ *l1* **and** $((\leq_{R1}) \Rightarrow_m (\leq_{L1}))$ *r1*
and $((\leq_{L1}) \trianglelefteq_h (\leq_{R1}))$ *l1 r1*

and *reflexive-on* (*in-field* ($\leq_{L1}$)) ($\leq_{L1}$)
and *reflexive-on* (*in-field* ($\leq_{R1}$)) ($\leq_{R1}$)
and (($\leq_{L2}$) $\mathrel{\wr\trianglelefteq}$ ($\leq_{R2}$)) *l2 r2*
and *partial-equivalence-rel* ($\leq_{L2}$)
and *partial-equivalence-rel* ($\leq_{R2}$)
shows (($_{L1}\lessapprox$) $\Rrightarrow$ ($_{L2}\lessapprox$))$\restriction_{in\text{-}dom\ (\leq_L)}{}^{\uparrow}{}_{in\text{-}codom\ (\leq_R)}$ = (($_{L1}\lessapprox$) $\Rrightarrow$ ($_{L2}\lessapprox$))
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 (($\leq_{L1}$) $\mathrel{\wr\trianglelefteq}$ ($\leq_{R1}$)) *l1 r1*
shows (($_{L1}\lessapprox$) $\Rrightarrow$ $S\restriction_{in\text{-}dom\ (\leq_{L2})}{}^{\uparrow}{}_{in\text{-}codom\ (\leq_{R2})}\restriction_{in\text{-}dom\ (\leq_L)}{}^{\uparrow}{}_{in\text{-}codom\ (\leq_R)}$
=
(($_{L1}\lessapprox$) $\Rrightarrow$ S)$\restriction_{in\text{-}dom\ (\leq_L)}{}^{\uparrow}{}_{in\text{-}codom\ (\leq_R)}$
by (*urule tpdfr.Dep-Fun-Rel-left-Galois-restrict-left-right-restrict-left-right-eq assms*
| *simp*)+

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* ($\leq_{L1}$)) ($\leq_{L1}$) η_1
and *refl-L1*: *reflexive-on* (*in-codom* ($\leq_{L1}$)) ($\leq_{L1}$)
and *trans-L1*: *transitive* ($\leq_{L1}$)
and *mono-l2*: $\bigwedge x.\ x \leq_{L1} x \Longrightarrow ((\leq_{L2\ x\ x}) \Rrightarrow_m (\leq_{R2\ (l1\ x)\ (l1\ x)})) (l2_{(l1\ x)\ x})$
and *mono-r2*: $\bigwedge x.\ x \leq_{L1} x \Longrightarrow ((\leq_{R2\ (l1\ x)\ (l1\ x)}) \Rrightarrow_m (\leq_{L2\ x\ (\eta_1\ x)})) (r2_{x\ (l1\ x)})$
and *inflationary-unit2*: $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$
inflationary-on (*in-codom* ($\leq_{L2\ x\ x}$)) ($\leq_{L2\ x\ x}$) ($\eta_{2\ x\ (l1\ x)}$)
and *L2-le1*: $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and *L2-unit-le2*: $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$

and *ge-R2-l2-le2*: $\bigwedge x\ y.\ x \leq_{L1} x \Longrightarrow$ *in-codom* $(\leq_{L2\ x\ (\eta_1\ x)})\ y \Longrightarrow$
$(\geq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ x}\ y) \leq (\geq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ (\eta_1\ x)}\ y)$
and *trans-L2*: $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $f \leq_L f$
shows $f \leq_L \eta\ f$
proof (*intro left-relI*)
fix *x1 x2* **assume** [*iff*]: $x1 \leq_{L1} x2$
moreover with *inflationary-unit1* **have** $x2 \leq_{L1} \eta_1\ x2$ **by** *blast*
ultimately have $x1 \leq_{L1} \eta_1\ x2$ **using** *trans-L1* **by** *blast*
with $\langle f \leq_L f \rangle$ **have** $f\ x1 \leq_{L2\ x1\ (\eta_1\ x2)} f\ (\eta_1\ x2)$ **by** *blast*
with *L2-unit-le2* **have** $f\ x1 \leq_{L2\ x1\ x2} f\ (\eta_1\ x2)$ **by** *blast*
moreover have ... $\leq_{L2\ x1\ x2} \eta\ f\ x2$
proof −
from *refl-L1* $\langle x2 \leq_{L1} \eta_1\ x2 \rangle$ **have** $\eta_1\ x2 \leq_{L1} \eta_1\ x2$ **by** *blast*
with $\langle f \leq_L f \rangle$ **have** $f\ (\eta_1\ x2) \leq_{L2\ (\eta_1\ x2)\ (\eta_1\ x2)} f\ (\eta_1\ x2)$ **by** *blast*
with *L2-le1* **have** $f\ (\eta_1\ x2) \leq_{L2\ x2\ (\eta_1\ x2)} f\ (\eta_1\ x2)$
using $\langle x2 \leq_{L1} \eta_1\ x2 \rangle$ **by** *blast*
moreover from *refl-L1* $\langle x1 \leq_{L1} x2 \rangle$ **have** [*iff*]: $x2 \leq_{L1} x2$ **by** *blast*
ultimately have $f\ (\eta_1\ x2) \leq_{L2\ x2\ x2} f\ (\eta_1\ x2)$ **using** *L2-unit-le2* **by** *blast*
with *inflationary-unit2* **have** $f\ (\eta_1\ x2) \leq_{L2\ x2\ x2}\ \eta_{2\ x2\ (l1\ x2)}\ (f\ (\eta_1\ x2))$ **by** *blast*
moreover have ... $\leq_{L2\ x2\ x2} \eta\ f\ x2$
proof −
from $\langle f\ (\eta_1\ x2) \leq_{L2\ x2\ x2} f\ (\eta_1\ x2) \rangle$ *mono-l2*
have $l2_{(l1\ x2)\ x2}\ (f\ (\eta_1\ x2)) \leq_{R2\ (l1\ x2)\ (l1\ x2)}\ l2_{(l1\ x2)\ x2}\ (f\ (\eta_1\ x2))$
by *blast*
with *ge-R2-l2-le2*
have $l2_{(l1\ x2)\ x2}\ (f\ (\eta_1\ x2)) \leq_{R2\ (l1\ x2)\ (l1\ x2)}\ l2_{(l1\ x2)\ (\eta_1\ x2)}\ (f\ (\eta_1\ x2))$
using $\langle f\ (\eta_1\ x2) \leq_{L2\ x2\ (\eta_1\ x2)} f\ (\eta_1\ x2) \rangle$ **by** *blast*
with *mono-r2* **have** $\eta_{2\ x2\ (l1\ x2)}\ (f\ (\eta_1\ x2)) \leq_{L2\ x2\ (\eta_1\ x2)} \eta\ f\ x2$
by *auto*
with *L2-unit-le2* **show** *?thesis* **by** *blast*
qed
ultimately have $f\ (\eta_1\ x2) \leq_{L2\ x2\ x2} \eta\ f\ x2$ **using** *trans-L2* **by** *blast*
with *L2-le1* **show** *?thesis* **by** *blast*
qed
ultimately show $f\ x1 \leq_{L2\ x1\ x2} \eta\ f\ x2$ **using** *trans-L2* **by** *blast*
qed

Deflationary interpretation *flip-inv* :
transport-Dep-Fun-Rel $(\geq_{R1})$ $(\geq_{L1})$ *r1 l1 flip2 R2 flip2 L2 r2 l2*
rewrites *flip-inv.L* $\equiv (\geq_R)$ **and** *flip-inv.R* $\equiv (\geq_L)$
and *flip-inv.unit* $\equiv \varepsilon$
and *flip-inv.t1.unit* $\equiv \varepsilon_1$
and $\bigwedge x\ y.$ *flip-inv.t2-unit x y* $\equiv \varepsilon_{2\ y\ x}$
and $\bigwedge R\ x\ y.\ (flip2\ R\ x\ y)^{-1} \equiv R\ y\ x$
and $\bigwedge R.$ *in-codom* $R^{-1} \equiv$ *in-dom R*
and $\bigwedge R\ x1\ x2.$ *in-codom* $(flip2\ R\ x1\ x2) \equiv$ *in-dom* $(R\ x2\ x1)$

and $\bigwedge x1\ x2\ x1'\ x2'.\ (\mathit{flip2}\ R2\ x1'\ x2' \Rightarrow_m \mathit{flip2}\ L2\ x1\ x2) \equiv ((\leq_{R2\ x2'\ x1'}) \Rightarrow_m (\leq_{L2\ x2\ x1}))$
and $\bigwedge x1\ x2\ x1'\ x2'.\ (\mathit{flip2}\ L2\ x1\ x2 \Rightarrow_m \mathit{flip2}\ R2\ x1'\ x2') \equiv ((\leq_{L2\ x2\ x1}) \Rightarrow_m (\leq_{R2\ x2'\ x1'}))$
and $\bigwedge(R :: 'z \Rightarrow 'z \Rightarrow \mathit{bool})\ (P :: 'z \Rightarrow \mathit{bool}).$
$(\mathit{inflationary\text{-}on}\ P\ R^{-1} :: ('z \Rightarrow 'z) \Rightarrow \mathit{bool}) \equiv \mathit{deflationary\text{-}on}\ P\ R$
and $\bigwedge(P :: 'b2 \Rightarrow \mathit{bool})\ x.$
$(\mathit{inflationary\text{-}on}\ P\ (\mathit{flip2}\ R2\ x\ x) :: ('b2 \Rightarrow 'b2) \Rightarrow \mathit{bool}) \equiv \mathit{deflationary\text{-}on}\ P\ (\leq_{R2\ x\ x})$
and $\bigwedge x1\ x2\ x3\ x4.\ \mathit{flip2}\ R2\ x1\ x2 \leq \mathit{flip2}\ R2\ x3\ x4 \equiv (\leq_{R2\ x2\ x1}) \leq (\leq_{R2\ x4\ x3})$
and $\bigwedge(R :: 'z \Rightarrow 'z \Rightarrow \mathit{bool})\ (P :: 'z \Rightarrow \mathit{bool}).\ \mathit{reflexive\text{-}on}\ P\ R^{-1} \equiv \mathit{reflexive\text{-}on}\ P\ R$
and $\bigwedge(R :: 'z \Rightarrow 'z \Rightarrow \mathit{bool}).\ \mathit{transitive}\ R^{-1} \equiv \mathit{transitive}\ R$
and $\bigwedge x1'\ x2'.\ \mathit{transitive}\ (\mathit{flip2}\ R2\ x1'\ x2') \equiv \mathit{transitive}\ (\leq_{R2\ x2'\ x1'})$
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
mono-wrt-rel-eq-dep-mono-wrt-rel)

lemma *counit-rel-self-if-rel-selfI*:
assumes *deflationary-on* (*in-dom* $(\leq_{R1})$) $(\leq_{R1})\ \varepsilon_1$
and *reflexive-on* (*in-dom* $(\leq_{R1})$) $(\leq_{R1})$
and *transitive* $(\leq_{R1})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow ((\leq_{L2\ (r1\ x')\ (r1\ x')}) \Rightarrow_m (\leq_{R2\ (\varepsilon_1\ x')\ x'}))\ (l2\ x'\ (r1\ x'))$
and $\bigwedge x'\ x'.\ x' \leq_{R1} x' \Longrightarrow ((\leq_{R2\ x'\ x'}) \Rightarrow_m (\leq_{L2\ (r1\ x')\ (r1\ x')}))\ (r2_{(r1\ x')\ x'})$
and $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow \mathit{deflationary\text{-}on}\ (\mathit{in\text{-}dom}\ (\leq_{R2\ x'\ x'}))\ (\leq_{R2\ x'\ x'})\ (\varepsilon_{2\ (r1\ x')\ x'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x'\ y'.\ x' \leq_{R1} x' \Longrightarrow \mathit{in\text{-}dom}\ (\leq_{R2\ (\varepsilon_1\ x')\ x'})\ y' \Longrightarrow$
$(\leq_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ x'\ y'}) \leq (\leq_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')\ y'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow \mathit{transitive}\ (\leq_{R2\ x1'\ x2'})$
and $g \leq_R g$
shows $\varepsilon\ g \leq_R g$
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*: *rel-equivalence-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})\ \eta_1$
and *mono-r2*: $\bigwedge x.\ x \leq_{L1} x \Longrightarrow ((\leq_{R2\ (l1\ x)\ (l1\ x)}) \Rightarrow_m (\leq_{L2\ x\ x}))\ (r2_{x\ (l1\ x)})$
and *rel-equiv-unit2*: $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$
rel-equivalence-on (*in-field* $(\leq_{L2\ x\ x})$) $(\leq_{L2\ x\ x})\ (\eta_{2\ x\ (l1\ x)})$
and *L2-le1*: $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and *L2-le2*: $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
and [*iff*]: $x \leq_{L1} x$
shows $((\leq_{R2\ (l1\ x)\ (l1\ x)}) \Rightarrow_m (\leq_{L2\ x\ (\eta_1\ x)}))\ (r2_{x\ (l1\ x)})$
and $((\leq_{R2\ (l1\ x)\ (l1\ x)}) \Rightarrow_m (\leq_{L2\ (\eta_1\ x)\ x}))\ (r2_{x\ (l1\ x)})$
and *deflationary-on* (*in-dom* $(\leq_{L2\ x\ x})$) $(\leq_{L2\ x\ x})\ \eta_{2\ x\ (l1\ x)}$

and *inflationary-on* (*in-codom* ($\leq_{L2\ x\ x}$)) ($\leq_{L2\ x\ x}$) $\eta_{2\ x\ (l1\ x)}$
proof −
from *rel-equiv-unit1* **have** $x \equiv_{L1} \eta_1\ x$ **by** *blast*
with *mono-r2* **show** $((\leq_{R2\ (l1\ x)\ (l1\ x)}) \Rrightarrow_m (\leq_{L2\ x\ (\eta_1\ x)}))\ (r2_{x\ (l1\ x)})$
and $((\leq_{R2\ (l1\ x)\ (l1\ x)}) \Rrightarrow_m (\leq_{L2\ (\eta_1\ x)\ x}))\ (r2_{x\ (l1\ x)})$
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* ($\leq_{L1}$)) ($\leq_{L1}$) η_1
and *trans-L1*: *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 $\bigwedge x.\ x \leq_{L1} x \Longrightarrow ((\leq_{R2\ (l1\ x)\ (l1\ x)}) \Rrightarrow_m (\leq_{L2\ x\ x}))\ (r2_{x\ (l1\ x)})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$
rel-equivalence-on (*in-field* ($\leq_{L2\ x\ x}$)) ($\leq_{L2\ x\ x}$) ($\eta_{2\ x\ (l1\ x)}$)
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ (\eta_1\ x1)\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x\ y.\ x \leq_{L1} x \Longrightarrow$ *in-dom* $(\leq_{L2\ (\eta_1\ x)\ x})\ y \Longrightarrow$
$(\leq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ x\ y}) \leq (\leq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ (\eta_1\ x)\ y})$
and $\bigwedge x\ y.\ x \leq_{L1} x \Longrightarrow$ *in-codom* $(\leq_{L2\ x\ (\eta_1\ x)})\ y \Longrightarrow$
$(\geq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ x\ y}) \leq (\geq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ (\eta_1\ x)\ y})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* ($\leq_{L2\ x1\ x2}$)
and $f \leq_L f$
shows $f \equiv_L \eta\ f$
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 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*: $((\leq_{L1}) \equiv_o (\leq_{R1}))$ *l1 r1*
and *refl-R1*: *reflexive-on* (*in-field* ($\leq_{R1}$)) ($\leq_{R1}$)

and *R2-counit-le1*: $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and *mono-l2*: $((x1'\ x2' :: (\leq_{R1})) \Rightarrow_m (x1\ x2 :: (\leq_{L1}) \mid x2\ _{L1}\lessapprox x1') \Rrightarrow$
in-field $(\leq_{L2\ x1\ (r1\ x2')}) \Rrightarrow (\leq_{R2\ (l1\ x1)\ x2'}))\ l2$
and [*iff*]: $x1' \leq_{R1} x2'$
shows $((\textit{in-dom}\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')})) \Rrightarrow (\leq_{R2\ x1'\ x2'}))\ (l2_{x1'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x1')})$
and $((\textit{in-codom}\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')})) \Rrightarrow (\leq_{R2\ x1'\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x2')})$
proof −
from *refl-R1* **have** $x1' \leq_{R1} x1'\ x2' \leq_{R1} x2'$ **by** *auto*
moreover with *order-equiv1*
have $r1\ x1' \leq_{L1} r1\ x2'\ r1\ x1' \leq_{L1} r1\ x1'\ r1\ x2' \leq_{L1} r1\ x2'$ **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* ‹$x1' \leq_{R1} x2'$›]
‹$r1\ x1' \leq_{L1} r1\ x1'$›]
with ‹$r1\ x1'\ _{L1}\lessapprox x1'$› *R2-counit-le1*
show $((\textit{in-dom}\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')})) \Rrightarrow (\leq_{R2\ x1'\ x2'}))\ (l2_{x1'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x1')})$
by (*intro Fun-Rel-predI*) (*auto dest*!: *in-field-if-in-dom*)
note *Dep-Fun-Rel-relD*[*OF dep-mono-wrt-relD*[*OF mono-l2* ‹$x2' \leq_{R1} x2'$›]
‹$r1\ x1' \leq_{L1} r1\ x2'$›]
with ‹$r1\ x2'\ _{L1}\lessapprox x2'$› *R2-counit-le1*
show $((\textit{in-codom}\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')})) \Rrightarrow (\leq_{R2\ x1'\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x2')})$
by (*intro Fun-Rel-predI*) (*auto dest*!: *in-field-if-in-codom*)
qed

lemma *order-equivalence-if-order-equivalence-mono-assms-rightI*:
assumes *order-equiv1*: $((\leq_{L1}) \equiv_o (\leq_{R1}))\ l1\ r1$
and *refl-L1*: *reflexive-on* $(\textit{in-field}\ (\leq_{L1}))\ (\leq_{L1})$
and *L2-unit-le2*: $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and *mono-r2*: $((x1\ x2 :: (\leq_{L1})) \Rightarrow_m (x1'\ x2' :: (\leq_{R1}) \mid x2\ _{L1}\lessapprox x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and [*iff*]: $x1 \leq_{L1} x2$
shows $((\textit{in-codom}\ (\leq_{R2\ (l1\ x1)\ (l1\ x2)})) \Rrightarrow (\leq_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x2)})\ (r2_{x2\ (l1\ x2)})$
and $((\textit{in-dom}\ (\leq_{R2\ (l1\ x1)\ (l1\ x2)})) \Rrightarrow (\leq_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x1)})\ (r2_{x1\ (l1\ x2)})$
proof −
from *refl-L1* **have** $x1 \leq_{L1} x1\ x2 \leq_{L1} x2$ **by** *auto*
moreover with *order-equiv1*
have $l1\ x1 \leq_{R1} l1\ x2\ l1\ x1 \leq_{R1} l1\ x1\ l1\ x2 \leq_{R1} l1\ x2$ **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* ‹$x1 \leq_{L1} x2$›]
‹$l1\ x2 \leq_{R1} l1\ x2$›]
with ‹$x2\ _{L1}\lessapprox l1\ x2$› *L2-unit-le2*
show $((\textit{in-codom}\ (\leq_{R2\ (l1\ x1)\ (l1\ x2)})) \Rrightarrow (\leq_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x2)})\ (r2_{x2\ (l1\ x2)})$
by (*intro Fun-Rel-predI*) (*auto dest*!: *in-field-if-in-codom*)
note *Dep-Fun-Rel-relD*[*OF dep-mono-wrt-relD*[*OF mono-r2* ‹$x1 \leq_{L1} x1$›]
‹$l1\ x1 \leq_{R1} l1\ x2$›]
with ‹$x1\ _{L1}\lessapprox l1\ x1$› *L2-unit-le2*

show $((\textit{in-dom}\ (\leq_{R2}\ (l1\ x1)\ (l1\ x2)))) \Rrightarrow (\leq_{L2}\ x1\ x2))\ (r2_{x1}\ (l1\ x1))\ (r2_{x1}\ (l1\ x2))$
by (*intro 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*: $((x1'\ x2' :: (\leq_{R1})) \Rrightarrow_m (x1\ x2 :: (\leq_{L1}) \mid x2\ {}_{L1}\lessapprox x1') \Rrightarrow$
$\textit{in-field}\ (\leq_{L2\ x1\ (r1\ x2')}) \Rrightarrow (\leq_{R2\ (l1\ x1)\ x2'}))\ l2$
and $x \leq_{L1} x$
and $\textit{in-field}\ (\leq_{L2\ x\ x})\ y$
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 ‹$\textit{in-field}\ (\leq_{L2\ x\ x})\ y$›
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 $\textit{in-field}\ (\leq_{L2\ (\eta_1\ x)\ (\eta_1\ x)})\ y\ \textit{in-field}\ (\leq_{L2\ x\ (\eta_1\ x)})\ y$
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 ‹$\eta_1\ x\ {}_{L1}\lessapprox l1\ x$› ‹$\textit{in-field}\ (\leq_{L2\ x\ (\eta_1\ x)})\ y$›
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′* :: ($\leq_{R1}$)) $\Rightarrow_m$ (*x3′ x4′* :: ($\leq_{R1}$)) | (*x2′* $\leq_{R1}$ *x3′* $\wedge$ *x4′* $\leq_{R1}$ ε_1 *x3′*))
$\Rightarrow$ ($\geq$)) *R2*
and *mono-r2*: ((*x1 x2* :: ($\leq_{L1}$)) $\Rightarrow_m$ (*x1′ x2′* :: ($\leq_{R1}$) | *x2* $_{L1}\lessapprox$ *x1′*) $\Rightarrow$
in-field ($\leq_{R2\ (l1\ x1)\ x2'}$) $\Rightarrow$ ($\leq_{L2\ x1\ (r1\ x2')}$)) *r2*
and $x' \leq_{R1} x'$
and *in-field* ($\leq_{R2\ x'\ x'}$) y'
shows $r2_{(r1\ x')\ (\varepsilon_1\ x')}\ y' \equiv_{L2\ (r1\ x')\ (r1\ x')}\ r2_{(r1\ x')\ x'}\ y'$
proof (*rule bi-relatedI*)
note *t1.preorder-equivalence-order-equivalenceE*[*elim*!]
from ‹$x' \leq_{R1} x'$› *pre-equiv1* **have** *r1 x′* $\leq_{L1}$ *r1 x′ x′* $\leq_{R1}$ ε_1 *x′* ε_1 *x′* $\leq_{R1}$ *x′*
by *blast*+
with *pre-equiv1* **have** *r1 x′* $_{L1}\lessapprox$ *x′ r1 x′* $_{L1}\lessapprox$ ε_1 *x′* **by** *fastforce*+
from *pre-equiv1* ‹$x' \leq_{R1} \varepsilon_1\ x'$› **have** $x' \leq_{R1} \varepsilon_1\ (\varepsilon_1\ x')$ **by** *fastforce*
moreover note ‹*in-field* ($\leq_{R2\ x'\ x'}$) y'›
Dep-Fun-Rel-relD[*OF dep-mono-wrt-relD*[*OF mono-R2* ‹$\varepsilon_1\ x' \leq_{R1} x'$›] ‹$\varepsilon_1\ x' \leq_{R1} x'$›]
Dep-Fun-Rel-relD[*OF dep-mono-wrt-relD*[*OF mono-R2* ‹$\varepsilon_1\ x' \leq_{R1} x'$›] ‹$x' \leq_{R1} x'$›]
ultimately have *in-field* ($\leq_{R2\ (\varepsilon_1\ x')\ (\varepsilon_1\ x')}$) y' *in-field* ($\leq_{R2\ (\varepsilon_1\ x')\ x'}$) y'
using ‹$x' \leq_{R1} \varepsilon_1\ x'$› ‹$x' \leq_{R1} x'$› **by** *blast*+
moreover note ‹*r1 x′* $_{L1}\lessapprox$ ε_1 *x′*›
Dep-Fun-Rel-relD[*OF dep-mono-wrt-relD*[*OF mono-r2* ‹*r1 x′* $\leq_{L1}$ *r1 x′*›] ‹$\varepsilon_1\ x' \leq_{R1} x'$›]
ultimately show $r2_{(r1\ x')\ (\varepsilon_1\ x')}\ y' \leq_{L2\ (r1\ x')\ (r1\ x')}\ r2_{(r1\ x')\ x'}\ y'$ **by** *auto*
note ‹*r1 x′* $_{L1}\lessapprox$ *x′*› ‹*in-field* ($\leq_{R2\ (\varepsilon_1\ x')\ (\varepsilon_1\ x')}$) y'›
Dep-Fun-Rel-relD[*OF dep-mono-wrt-relD*[*OF mono-r2* ‹*r1 x′* $\leq_{L1}$ *r1 x′*›] ‹$x' \leq_{R1} \varepsilon_1\ x'$›]
then have $r2_{(r1\ x')\ x'}\ y' \leq_{L2\ (r1\ x')\ (\eta_1\ (r1\ x'))}\ r2_{(r1\ x')\ (\varepsilon_1\ x')}\ y'$ **by** *auto*
moreover from *pre-equiv1* ‹*r1 x′* $\leq_{L1}$ *r1 x′*›
have $\eta_1\ (r1\ x') \leq_{L1} r1\ x'\ r1\ x' \leq_{L1} \eta_1\ (r1\ x')$ **by** *fastforce*+
moreover note *Dep-Fun-Rel-relD*[*OF dep-mono-wrt-relD*
[*OF mono-L2* ‹*r1 x′* $\leq_{L1}$ *r1 x′*›] ‹*r1 x′* $\leq_{L1}$ $\eta_1\ (r1\ x')$›]
ultimately show $r2_{(r1\ x')\ x'}\ y' \leq_{L2\ (r1\ x')\ (r1\ x')}\ r2_{(r1\ x')\ (\varepsilon_1\ x')}\ y'$
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* ($\leq_{L1}$)) ($\leq_{L1}$) η_1
and *reflexive-on* (*in-codom* ($\leq_{L1}$)) ($\leq_{L1}$)
and *transitive* ($\leq_{L1}$)
and (($\leq_{L2}$) $\Rightarrow_m$ ($\leq_{R2}$)) *l2*

and $((\leq_{R2}) \Rightarrow_m (\leq_{L2}))$ *r2*
and *inflationary-on* $(\textit{in-codom}\ (\leq_{L2}))\ (\leq_{L2})\ \eta_2$
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* $(\textit{in-dom}\ (\leq_{R1}))\ (\leq_{R1})\ \varepsilon_1$
and *reflexive-on* $(\textit{in-dom}\ (\leq_{R1}))\ (\leq_{R1})$
and *transitive* $(\leq_{R1})$
and $((\leq_{L2}) \Rightarrow_m (\leq_{R2}))$ *l2*
and $((\leq_{R2}) \Rightarrow_m (\leq_{L2}))$ *r2*
and *deflationary-on* $(\textit{in-dom}\ (\leq_{R2}))\ (\leq_{R2})\ \varepsilon_2$
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* $(\textit{in-field}\ (\leq_{L1}))\ (\leq_{L1})\ \eta_1$
and *transitive* $(\leq_{L1})$
and $((\leq_{L2}) \Rightarrow_m (\leq_{R2}))$ *l2*
and $((\leq_{R2}) \Rightarrow_m (\leq_{L2}))$ *r2*
and *rel-equivalence-on* $(\textit{in-field}\ (\leq_{L2}))\ (\leq_{L2})\ \eta_2$
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 $(\textit{tdfr.L} \Rightarrow_m \textit{tdfr.R})\ l$ **and** $(\textit{tdfr.R} \Rightarrow_m \textit{tdfr.L})\ r$
and *inflationary-on* $(\textit{in-codom}\ (\leq_{L1}))\ (\leq_{L1})\ \eta_1$
and *reflexive-on* $(\textit{in-codom}\ (\leq_{L1}))\ (\leq_{L1})$
and *transitive* $(\leq_{L1})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow ((\leq_{L2}\ x\ x) \Rightarrow_m (\leq_{R2}\ (l1\ x)\ (l1\ x)))\ (l2_{(l1\ x)\ x})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow ((\leq_{R2}\ (l1\ x)\ (l1\ x)) \Rightarrow_m (\leq_{L2}\ x\ (\eta_1\ x)))\ (r2_{x\ (l1\ x)})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow \textit{inflationary-on}\ (\textit{in-codom}\ (\leq_{L2}\ x\ x))\ (\leq_{L2}\ x\ x)\ (\eta_2\ x\ (l1\ x))$
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 $\bigwedge x\ y.\ x \leq_{L1} x \Longrightarrow \textit{in-codom}\ (\leq_{L2}\ x\ (\eta_1\ x))\ y \Longrightarrow$
$(\geq_{R2}\ (l1\ x)\ (l1\ x))\ (l2_{(l1\ x)\ x\ y}) \leq (\geq_{R2}\ (l1\ x)\ (l1\ x))\ (l2_{(l1\ x)\ (\eta_1\ x)\ y})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow \textit{transitive}\ (\leq_{L2}\ x1\ x2)$

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 6 2 intro*: *tdfr.rel-unit-self-if-rel-selfI*[*simplified unit-eq*]
elim!: *Refl-RelE in-fieldE*)

Deflationary **lemma** *deflationary-on-counitI*:
assumes (*tdfr.L* $\Rrightarrow_m$ *tdfr.R*) *l* **and** (*tdfr.R* $\Rrightarrow_m$ *tdfr.L*) *r*
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 $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow ((\leq_{L2\ (r1\ x')\ (r1\ x')}) \Rrightarrow_m (\leq_{R2\ (\varepsilon_1\ x')\ x'}))\ (l2_{\ x'\ (r1\ x')})$
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 $\bigwedge x'.\ x' \leq_{R1} x' \Longrightarrow$ *deflationary-on* (*in-dom* ($\leq_{R2\ x'\ x'}$)) ($\leq_{R2\ x'\ x'}$) ($\varepsilon_{2\ (r1\ x')\ x'}$)
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x'\ y'.\ x' \leq_{R1} x' \Longrightarrow$ *in-dom* ($\leq_{R2\ (\varepsilon_1\ x')\ x'}$) $y' \Longrightarrow$
$(\leq_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ x'}\ y') \leq (\leq_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')}$
$y')$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$ *transitive* ($\leq_{R2\ x1'\ x2'}$)
shows *deflationary-on* (*in-field* ($\leq_R$)) ($\leq_R$) ε
unfolding *right-rel-eq-tdfr-right-Refl-Rel* **using** *assms*
by (*intro deflationary-onI Refl-RelI*)
(*auto 6 2 intro*: *tdfr.counit-rel-self-if-rel-selfI*[*simplified counit-eq*]
elim!: *Refl-RelE in-fieldE*)

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* (*in-field* ($\leq_{L1}$)) ($\leq_{L1}$) η_1
and *trans-L1*: *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 $\bigwedge x.\ x \leq_{L1} x \Longrightarrow ((\leq_{R2\ (l1\ x)\ (l1\ x)}) \Rrightarrow_m (\leq_{L2\ x\ x}))\ (r2_{x\ (l1\ x)})$
and $\bigwedge x.\ x \leq_{L1} x \Longrightarrow$ *rel-equivalence-on* (*in-field* ($\leq_{L2\ x\ x}$)) ($\leq_{L2\ x\ x}$) ($\eta_{2\ x\ (l1\ x)}$)
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ (\eta_1\ x1)\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x\ y.\ x \leq_{L1} x \Longrightarrow$ *in-dom* ($\leq_{L2\ (\eta_1\ x)\ x}$) $y \Longrightarrow$

$(\leq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ x\ y}) \leq (\leq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ (\eta_1\ x)\ y})$
and $\bigwedge x\ y.\ x \leq_{L1} x \Longrightarrow$ *in-codom* $(\leq_{L2\ x\ (\eta_1\ x)})\ y \Longrightarrow$
$(\geq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ x\ y}) \leq (\geq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ (\eta_1\ x)\ y})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
shows *rel-equivalence-on* (*in-field* $(\leq_L)$) $(\leq_L)\ \eta$
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* $(\leq_{L1})$) $(\leq_{L1})\ \eta_1$
and *rel-equivalence-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})\ \varepsilon_1$
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 $\bigwedge x\ y.\ x \leq_{L1} x \Longrightarrow$ *in-dom* $(\leq_{L2\ (\eta_1\ x)\ x})\ y \Longrightarrow$
$(\leq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ x\ y}) \leq (\leq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ (\eta_1\ x)\ y})$
and $\bigwedge x\ y.\ x \leq_{L1} x \Longrightarrow$ *in-codom* $(\leq_{L2\ x\ (\eta_1\ x)})\ y \Longrightarrow$
$(\geq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ x\ y}) \leq (\geq_{R2\ (l1\ x)\ (l1\ x)})\ (l2_{(l1\ x)\ (\eta_1\ x)\ y})$
and $\bigwedge x'\ y'.\ x' \leq_{R1} x' \Longrightarrow$ *in-dom* $(\leq_{R2\ (\varepsilon_1\ x')\ x'})\ y' \Longrightarrow$
$(\leq_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ x'\ y'}) \leq (\leq_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')\ y'})$
and $\bigwedge x'\ y'.\ x' \leq_{R1} x' \Longrightarrow$ *in-codom* $(\leq_{R2\ x'\ (\varepsilon_1\ x')})\ y' \Longrightarrow$
$(\geq_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ x'\ y'}) \leq (\geq_{L2\ (r1\ x')\ (r1\ x')})\ (r2_{(r1\ x')\ (\varepsilon_1\ x')\ y'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{R1} x2 \Longrightarrow$ *transitive* $(\leq_{R2\ x1\ x2})$
shows $((\leq_L) \equiv_o (\leq_R))\ l\ r$
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*: $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))\ l1\ r1$
and *order-equiv2*: $\bigwedge x\ x'.\ x\ {}_{L1}\!\lessapprox\!\gtrapprox x' \Longrightarrow$
$((\leq_{L2\ x\ (r1\ x')}) \equiv_o (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and *L2-les*: $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
$\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ (\eta_1\ x1)\ x2}) \leq (\leq_{L2\ x1\ x2})$
$\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
$\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and *R2-les*: $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x2'\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
$\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
$\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$
$\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ (\varepsilon_1\ x2')}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$((\mathit{in\text{-}dom}\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')})) \Rrightarrow (\leq_{R2\ x1'\ x2'}))\ (l2_{x1'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x1')})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow$
$((\mathit{in\text{-}codom}\ (\leq_{L2\ (r1\ x1')\ (r1\ x2')})) \Rrightarrow (\leq_{R2\ x1'\ x2'}))\ (l2_{x2'\ (r1\ x1')})\ (l2_{x2'\ (r1\ x2')})$
and *l2-bi-rel*: $\bigwedge x\ y.\ x \leq_{L1} x \Longrightarrow$ *in-field* $(\leq_{L2\ x\ x})\ y \Longrightarrow$
$l2_{(l1\ x)\ (\eta_1\ x)\ y} \equiv_{R2\ (l1\ x)\ (l1\ x)} l2_{(l1\ x)\ x\ y}$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$
$((\mathit{in\text{-}codom}\ (\leq_{R2\ (l1\ x1)\ (l1\ x2)})) \Rrightarrow (\leq_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x2)})\ (r2_{x2\ (l1\ x2)})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$
$((\mathit{in\text{-}dom}\ (\leq_{R2\ (l1\ x1)\ (l1\ x2)})) \Rrightarrow (\leq_{L2\ x1\ x2}))\ (r2_{x1\ (l1\ x1)})\ (r2_{x1\ (l1\ x2)})$
and *r2-bi-rel*: $\bigwedge x'\ y'.\ x' \leq_{R1} x' \Longrightarrow$ *in-field* $(\leq_{R2\ x'\ x'})\ y' \Longrightarrow$
$r2_{(r1\ x')\ (\varepsilon_1\ x')\ y'} \equiv_{L2\ (r1\ x')\ (r1\ x')} r2_{(r1\ x')\ x'\ y'}$
and *trans-L2*: $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$

and *trans-R2*: $\bigwedge x1\ x2.\ x1 \leq_{R1} x2 \Longrightarrow transitive\ (\leq_{R2\ x1\ x2})$
shows $((\leq_L) \equiv_o (\leq_R))\ l\ r$
proof −
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** (*intro t1.left-GaloisI*)
(*auto elim*!: *t1.preorder-equivalence-order-equivalenceE t1.order-equivalenceE bi-relatedE*)
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 elim*!: *le-relE*)
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 $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow ((\leq_{R2}\ (l1\ x1)\ (l1\ x2)) \Rrightarrow_m (\leq_{L2}\ x1\ (\eta_1\ x2)))\ (r2_{x1}\ (l1\ x2))$
using *pre-equiv1* **by** *blast*
moreover with *L2-les*
have $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow ((\leq_{R2}\ (l1\ x1)\ (l1\ x2)) \Rrightarrow_m (\leq_{L2}\ x1\ x2))\ (r2_{x1}\ (l1\ x2))$
by *blast*
moreover have *in-dom* $(\leq_{L2}\ (\eta_1\ x)\ x)\ y \Longrightarrow$
$(\leq_{R2}\ (l1\ x)\ (l1\ x))\ (l2_{(l1\ x)\ x}\ y) \leq (\leq_{R2}\ (l1\ x)\ (l1\ x))\ (l2_{(l1\ x)\ (\eta_1\ x)}\ y)$
(**is** - $\Longrightarrow$ *?goal1*)
and *in-codom* $(\leq_{L2\ x\ (\eta_1\ x)})\ y \Longrightarrow$
$(\geq_{R2}\ (l1\ x)\ (l1\ x))\ (l2_{(l1\ x)\ x}\ y) \leq (\geq_{R2}\ (l1\ x)\ (l1\ x))\ (l2_{(l1\ x)\ (\eta_1\ x)}\ y)$
(**is** - $\Longrightarrow$ *?goal2*)
if [*iff*]: $x \leq_{L1} x$ **for** *x y*
proof −
presume *in-dom* $(\leq_{L2}\ (\eta_1\ x)\ x)\ y \vee$ *in-codom* $(\leq_{L2\ x\ (\eta_1\ x)})\ y$
then have *in-field* $(\leq_{L2\ x\ x})\ y$ **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** $\langle l1\ x \leq_{R1} l1\ x\rangle$ **by** *blast*
ultimately show *?goal1 ?goal2* **using** *trans-R2* **by** *blast+*
qed *auto*
moreover have *in-dom* $(\leq_{R2}\ (\varepsilon_1\ x')\ x')\ y' \Longrightarrow$
$(\leq_{L2}\ (r1\ x')\ (r1\ x'))\ (r2_{(r1\ x')\ x'}\ y') \leq (\leq_{L2}\ (r1\ x')\ (r1\ x'))\ (r2_{(r1\ x')\ (\varepsilon_1\ x')}\ y')$
(**is** - $\Longrightarrow$ *?goal1*)
and *in-codom* $(\leq_{R2\ x'\ (\varepsilon_1\ x')})\ y' \Longrightarrow$
$(\geq_{L2}\ (r1\ x')\ (r1\ x'))\ (r2_{(r1\ x')\ x'}\ y') \leq (\geq_{L2}\ (r1\ x')\ (r1\ x'))\ (r2_{(r1\ x')\ (\varepsilon_1\ x')}\ y')$
(**is** - $\Longrightarrow$ *?goal2*)
if [*iff*]: $x' \leq_{R1} x'$ **for** $x'\ y'$
proof −
presume *in-dom* $(\leq_{R2}\ (\varepsilon_1\ x')\ x')\ y' \vee$ *in-codom* $(\leq_{R2\ x'\ (\varepsilon_1\ x')})\ y'$
then have *in-field* $(\leq_{R2\ x'\ x'})\ y'$ **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** $\langle r1\ x' \leq_{L1} r1\ x'\rangle$ **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 $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_o (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x2\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ (\eta_1\ x1)\ x2}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ x1}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow (\leq_{L2\ x1\ (\eta_1\ x2)}) \leq (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x2'\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ (\varepsilon_1\ x1')\ x2'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ x1'}) \leq (\leq_{R2\ x1'\ x2'})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow (\leq_{R2\ x1'\ (\varepsilon_1\ x2')}) \leq (\leq_{R2\ x1'\ x2'})$
and $((x1'\ x2' :: (\leq_{R1})) \Rrightarrow_m (x1\ x2 :: (\leq_{L1}) \mid x2\ {}_{L1}\lessapprox x1') \Rrightarrow$
in-field $(\leq_{L2\ x1\ (r1\ x2')}) \Rrightarrow (\leq_{R2\ (l1\ x1)\ x2'}))$ *l2*
and $\bigwedge x\ y.\ x \leq_{L1} x \Longrightarrow$ *in-field* $(\leq_{L2\ x\ x})\ y \Longrightarrow$
$l2_{(l1\ x)\ (\eta_1\ x)}\ y \equiv_{R2\ (l1\ x)\ (l1\ x)}\ l2_{(l1\ x)\ x}\ y$
and $((x1\ x2 :: (\leq_{L1})) \Rrightarrow_m (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))$ *r2*
and $\bigwedge x'\ y'.\ x' \leq_{R1} x' \Longrightarrow$ *in-field* $(\leq_{R2\ x'\ x'})\ y' \Longrightarrow$
$r2_{(r1\ x')\ (\varepsilon_1\ x')}\ y' \equiv_{L2\ (r1\ x')\ (r1\ x')}\ r2_{(r1\ x')\ x'}\ y'$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow$ *transitive* $(\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{R1} x2 \Longrightarrow$ *transitive* $(\leq_{R2\ x1\ x2})$
shows $((\leq_L) \equiv_o (\leq_R))\ l\ r$
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 $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_o (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $((x1\ x2 :: (\leq_{L1}) \mid \eta_1\ x2 \leq_{L1} x1) \Rrightarrow_m (x3\ x4 :: (\leq_{L1}) \mid x2 \leq_{L1} x3) \Rrightarrow (\leq))$ *L2*
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 $((x1'\ x2' :: (\leq_{R1})) \Rrightarrow_m (x3'\ x4' :: (\leq_{R1}) \mid (x2' \leq_{R1} x3' \wedge x4' \leq_{R1} \varepsilon_1\ x3')) \Rrightarrow (\geq))$ *R2*
and $((x1'\ x2' :: (\leq_{R1})) \Rrightarrow_m (x1\ x2 :: (\leq_{L1}) \mid x2\ {}_{L1}\lessapprox x1') \Rrightarrow$
in-field $(\leq_{L2\ x1\ (r1\ x2')}) \Rrightarrow (\leq_{R2\ (l1\ x1)\ x2'}))$ *l2*
and $((x1\ x2 :: (\leq_{L1})) \Rrightarrow_m (x1'\ x2' :: (\leq_{R1}) \mid x2\ {}_{L1}\lessapprox x1') \Rrightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))$ *r2*

and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow transitive\ (\leq_{L2\ x1\ x2})$
and $\bigwedge x1\ x2.\ x1 \leq_{R1} x2 \Longrightarrow transitive\ (\leq_{R2\ x1\ x2})$
shows $((\leq_L) \equiv_o (\leq_R))\ l\ r$
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 $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))\ l1\ r1$
and $\bigwedge x\ x'.\ x\ _{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_{pre} (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $((x1\ x2 :: (\geq_{L1})) \Rightarrow_m (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rightarrow (\leq))\ L2$
and $((x1'\ x2' :: (\geq_{R1})) \Rightarrow_m (x3'\ x4' :: (\leq_{R1}) \mid x1' \leq_{R1} x3') \Rightarrow (\leq))\ R2$
and $((x1'\ x2' :: (\leq_{R1})) \Rightarrow_m (x1\ x2 :: (\leq_{L1}) \mid x2\ _{L1}\gtrapprox x1') \Rightarrow$
in-field $(\leq_{L2\ x1\ (r1\ x2')}) \Rightarrow (\leq_{R2\ (l1\ x1)\ x2'}))\ l2$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow_m (x1'\ x2' :: (\leq_{R1}) \mid x2\ _{L1}\gtrapprox x1') \Rightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
shows $((\leq_L) \equiv_o (\leq_R))\ l\ r$
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 $((\leq_{L1}) \Rightarrow_m (\leq_{R1}))\ l1$
and $((\leq_{R1}) \Rightarrow_m (\leq_{L1}))\ r1$
and *inflationary-on* $(in\text{-}codom\ (\leq_{L1}))\ (\leq_{L1})\ \eta_1$
and *reflexive-on* $(in\text{-}codom\ (\leq_{L1}))\ (\leq_{L1})$
and *transitive* $(\leq_{L1})$

and $((\leq_{L2}) \Rrightarrow_m (\leq_{R2}))$ *l2*
and $((\leq_{R2}) \Rrightarrow_m (\leq_{L2}))$ *r2*
and *inflationary-on* (*in-codom* $(\leq_{L2})$) $(\leq_{L2})$ η_2
and *transitive* $(\leq_{L2})$
shows *inflationary-on* (*in-field* $(\leq_L)$) $(\leq_L)$ η
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 $((\leq_{L1}) \dashv (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$
and *reflexive-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$
and $\bigwedge x\ x'.\ x\ {}_{L1}\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \dashv (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $((\text{-}\ x2 :: (\leq_{L1})) \Rrightarrow_m (x3\ x4 :: (\leq_{L1}) \mid (x2 \leq_{L1} x3 \wedge x4 \leq_{L1} \eta_1\ x3)) \Rrightarrow (\geq))\ L2$
and $((x1'\ x2' :: (\leq_{R1}) \mid \varepsilon_1\ x2' \leq_{R1} x1') \Rrightarrow_m (x3'\ \text{-} :: (\leq_{R1}) \mid x2' \leq_{R1} x3') \Rrightarrow (\leq))\ R2$
and $((x1'\ x2' :: (\leq_{R1})) \Rrightarrow_m (x1\ x2 :: (\leq_{L1}) \mid x2\ {}_{L1}\lessapprox x1') \Rrightarrow$
in-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-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
and $\bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow transitive\ (\leq_{L2\ x1\ x2})$
and $\bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow transitive\ (\leq_{R2\ x1'\ x2'})$
shows $((\leq_L) \dashv_{pre} (\leq_R))\ l\ r$
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 $((x1\ \text{-} :: (\geq_{L1})) \Rightarrow_m (x3\ \text{-} :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rightarrow (\leq))\ L2$
and $((x1'\ \text{-} :: (\geq_{R1})) \Rightarrow_m (x3'\ \text{-} :: (\leq_{R1}) \mid x1' \leq_{R1} x3') \Rightarrow (\leq))\ R2$
and $((x1'\ x2' :: (\leq_{R1})) \Rightarrow_m (x1\ x2 :: (\leq_{L1}) \mid x2\ _{L1}\lessapprox x1') \Rightarrow$
in-field $(\leq_{L2\ x1\ (r1\ x2')}) \Rightarrow (\leq_{R2\ (l1\ x1)\ x2'}))\ l2$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow_m (x1'\ x2' :: (\leq_{R1}) \mid x2\ _{L1}\lessapprox x1') \Rightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
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 $((x1\ \text{-} :: (\geq_{L1})) \Rightarrow_m (x3\ \text{-} :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rightarrow (\leq))\ L2$
and $((x1'\ \text{-} :: (\geq_{R1})) \Rightarrow_m (x3'\ \text{-} :: (\leq_{R1}) \mid x1' \leq_{R1} x3') \Rightarrow (\leq))\ R2$
and $((x1'\ x2' :: (\leq_{R1})) \Rightarrow_m (x1\ x2 :: (\leq_{L1}) \mid x2\ _{L1}\lessapprox x1') \Rightarrow$
in-field $(\leq_{L2\ x1\ (r1\ x2')}) \Rightarrow (\leq_{R2\ (l1\ x1)\ x2'}))\ l2$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow_m (x1'\ x2' :: (\leq_{R1}) \mid x2\ _{L1}\lessapprox x1') \Rightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
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 $[\![t1.galois\text{-}equivalence;$ $preorder\text{-}on\ (in\text{-}field\ (\leq_{L1}))\ (\leq_{L1});\ ((x1\ x2 :: (\leq_{L1})^{-1}) \Rrightarrow_m (x3\ x4 :: (\leq_{L1}))$ $\Rrightarrow x1 \leq_{L1} x3 \longrightarrow (\leq))\ L2;\ \bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow partial\text{-}equivalence\text{-}rel$ $(\leq_{L2\ x1\ x2})]\!] \Longrightarrow flip.R = flip.tdfr.R.$

Simplification of Galois relator See $[\![t1.galois\text{-}connection;\ reflexive\text{-}on\ (in\text{-}field\ (\leq_{L1}))\ (\leq_{L1});\ \bigwedge x\ x'.\ flip.t1.right\text{-}Galois\ x\ x' \Longrightarrow (\leq_{R2\ l1\ x\ x'}$ $\Rrightarrow_m \leq_{L2\ x\ r1\ x'})\ r2_{x\ x'};\ ((x1 : \top) \Rrightarrow_m (x2\ \text{-} :: (\leq_{L1})) \Rrightarrow_m x1 \leq_{L1} x2 \longrightarrow$ $(\leq))\ L2;\ ((x1 : \top) \Rrightarrow_m (x2\ x3 :: (\leq_{L1})) \Rrightarrow_m (x1 \leq_{L1} x2 \wedge x3 \leq_{L1} \eta_1$ $x2) \longrightarrow (\lambda x\ y.\ y \leq x))\ L2;\ ((x1\ x2 :: (\leq_{L1})) \Rrightarrow_m (x1'\ x2' :: (\leq_{R1}))$ $\Rrightarrow flip.t1.right\text{-}Galois\ x2\ x1' \longrightarrow (in\text{-}field\ (\leq_{R2\ l1\ x1\ x2'}) \Rrightarrow \leq_{L2\ x1\ r1\ x2'}))$ $r2;\ \bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow transitive\ (\leq_{L2\ x1\ x2})]\!] \Longrightarrow flip.right\text{-}Galois =$ $(Dep\text{-}Fun\text{-}Rel\ flip.t1.right\text{-}Galois\ t2.left\text{-}Galois)\restriction_{in\text{-}dom\ flip.R}\restriction_{in\text{-}codom\ flip.L}$

$[\![t1.preorder\text{-}equivalence;\ \bigwedge x\ x'.\ flip.t1.right\text{-}Galois\ x\ x' \Longrightarrow (\leq_{R2\ l1\ x\ x'}$ $\Rrightarrow_m \leq_{L2\ x\ r1\ x'})\ r2_{x\ x'};\ ((x1\ x2 :: (\leq_{L1})^{-1}) \Rrightarrow_m (x3\ x4 :: (\leq_{L1})) \Rrightarrow x1 \leq_{L1}$ $x3 \longrightarrow (\leq))\ L2;\ ((x1\ x2 :: (\leq_{L1})) \Rrightarrow_m (x1'\ x2' :: (\leq_{R1})) \Rrightarrow flip.t1.right\text{-}Galois$ $x2\ x1' \longrightarrow (in\text{-}field\ (\leq_{R2\ l1\ x1\ x2'}) \Rrightarrow \leq_{L2\ x1\ r1\ x2'}))\ r2;\ \bigwedge x1\ x2.\ x1 \leq_{L1} x2$ $\Longrightarrow transitive\ (\leq_{L2\ x1\ x2})]\!] \Longrightarrow flip.right\text{-}Galois = (Dep\text{-}Fun\text{-}Rel\ flip.t1.right\text{-}Galois$ $t2.left\text{-}Galois)\restriction_{in\text{-}dom\ flip.R}\restriction_{in\text{-}codom\ flip.L}$

$[\![t1.preorder\text{-}equivalence;\ \bigwedge x\ x'.\ flip.t1.right\text{-}Galois\ x\ x' \Longrightarrow t2.preorder\text{-}equivalence$ $x\ x';\ ((x1\ x2 :: (\leq_{L1})^{-1}) \Rrightarrow_m (x3\ x4 :: (\leq_{L1})) \Rrightarrow x1 \leq_{L1} x3 \longrightarrow (\leq))$ $L2;\ ((x1\ x2 :: (\leq_{L1})) \Rrightarrow_m (x1'\ x2' :: (\leq_{R1})) \Rrightarrow flip.t1.right\text{-}Galois\ x2\ x1'$ $\longrightarrow (in\text{-}field\ (\leq_{R2\ l1\ x1\ x2'}) \Rrightarrow \leq_{L2\ x1\ r1\ x2'}))\ r2]\!] \Longrightarrow flip.right\text{-}Galois =$ $(Dep\text{-}Fun\text{-}Rel\ flip.t1.right\text{-}Galois\ t2.left\text{-}Galois)\restriction_{in\text{-}dom\ flip.R}\restriction_{in\text{-}codom\ flip.L}$

$[\![t1.preorder\text{-}equivalence;\ \bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow ((\leq_{L2\ r1\ x1'\ r1\ x2'})$ $_h\trianglelefteq (\leq_{R2\ \varepsilon_1\ x1'\ x2'}))\ l2_{x2'\ r1\ x1'}\ r2_{r1\ x1'\ x2'};\ ((x1\ x2 :: (\leq_{L1})^{-1}) \Rrightarrow_m (x3\ x4$ $:: (\leq_{L1})) \Rrightarrow x1 \leq_{L1} x3 \longrightarrow (\leq))\ L2;\ ((x1'\ x2' :: (\leq_{R1})^{-1}) \Rrightarrow_m (x3'\ x4' ::$ $(\leq_{R1})) \Rrightarrow x1' \leq_{R1} x3' \longrightarrow (\leq))\ R2;\ ((x1'\ x2' :: (\leq_{R1})) \Rrightarrow_m (x1\ x2 :: (\leq_{L1}))$ $\Rrightarrow flip.t1.right\text{-}Galois\ x2\ x1' \longrightarrow (in\text{-}field\ (\leq_{L2\ x1\ r1\ x2'}) \Rrightarrow \leq_{R2\ l1\ x1\ x2'}))$ $l2;\ ((x1\ x2 :: (\leq_{L1})) \Rrightarrow_m (x1'\ x2' :: (\leq_{R1})) \Rrightarrow flip.t1.right\text{-}Galois\ x2\ x1' \longrightarrow$ $(in\text{-}field\ (\leq_{R2\ l1\ x1\ x2'}) \Rrightarrow \leq_{L2\ x1\ r1\ x2'}))\ r2;\ \bigwedge x1\ x2.\ x1 \leq_{L1} x2 \Longrightarrow par\text{-}$ $tial\text{-}equivalence\text{-}rel\ (\leq_{L2\ x1\ x2});\ \bigwedge x1'\ x2'.\ x1' \leq_{R1} x2' \Longrightarrow partial\text{-}equivalence\text{-}rel$ $(\leq_{R2\ x1'\ x2'})]\!] \Longrightarrow (Dep\text{-}Fun\text{-}Rel\ flip.t1.right\text{-}Galois\ t2.left\text{-}Galois)\restriction_{in\text{-}dom\ flip.R}\restriction_{in\text{-}codom\ flip.L}$ $= Dep\text{-}Fun\text{-}Rel\ flip.t1.right\text{-}Galois\ t2.left\text{-}Galois$

$t1.half\text{-}galois\text{-}prop\text{-}left \Longrightarrow ((x\ x' :: flip.t1.right\text{-}Galois) \Rrightarrow (?S\ x\ x')\restriction_{in\text{-}dom\ (\leq_{L2\ x\ r1\ x'})}\restriction_{in\text{-}codom\ (\leq}$ $= (Dep\text{-}Fun\text{-}Rel\ flip.t1.right\text{-}Galois\ ?S)\restriction_{in\text{-}dom\ flip.R}\restriction_{in\text{-}codom\ flip.L}$

end

Monotone Function Relator **context** *transport-Mono-Fun-Rel*
begin

interpretation *flip* : *transport-Mono-Fun-Rel R1 L1 r1 l1 R2 L2 r2 l2* **.**

Closure of Order and Galois Concepts **lemma** *preorder-galois-connection-if-galois-connectionI*:

assumes $((\leq_{L1}) \dashv (\leq_{R1}))$ *l1 r1*
and *reflexive-on* (*in-codom* $(\leq_{L1})$) $(\leq_{L1})$ *reflexive-on* (*in-dom* $(\leq_{R1})$) $(\leq_{R1})$
and $((\leq_{L2}) \dashv (\leq_{R2}))$ *l2 r2*
and *transitive* $(\leq_{L2})$ *transitive* $(\leq_{R2})$
shows $((\leq_L) \dashv_{pre} (\leq_R))$ *l r*
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 $[\![$*reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$; *partial-equivalence-rel* $(\leq_{L2})]\!] \Longrightarrow$ *flip.tpdfr.R* = *flip.tfr.tdfr.R*.

Simplification of Galois relator See $[\![((\leq_{L1}) \Rightarrow_m (\leq_{R1}))\ l1;$ *tdfrs.t1.galois-prop l1 r1*; *reflexive-on* (*in-dom* $(\leq_{L1})$) $(\leq_{L1})$; $((\leq_{R2}) \Rightarrow_m (\leq_{L2}))\ r2$; *transitive* $(\leq_{L2})]\!] \Longrightarrow$ *flip.tpdfr.right-Galois* = (*flip.tdfrs.t1.right-Galois* $\Rrightarrow$ *flip.tdfrs.t2.right-Galois*)$\restriction_{\textit{in-dom flip.tpdfr.R}}\restriction_{\textit{in-codom flip.tpdfr.L}}$

$[\![((\leq_{L1}) \Rightarrow_m (\leq_{R1}))\ l1; ((\leq_{R1}) \Rightarrow_m (\leq_{L1}))\ r1;$ *tdfrs.t1.half-galois-prop-right*; *reflexive-on* (*in-field* $(\leq_{L1})$) $(\leq_{L1})$; *reflexive-on* (*in-field* $(\leq_{R1})$) $(\leq_{R1})$; *tdfrs.t2.half-galois-prop-left*; *partial-equivalence-rel* $(\leq_{L2})$; *partial-equivalence-rel* $(\leq_{R2})]\!] \Longrightarrow$ (*flip.tdfrs.t1.right-Galois* $\Rrightarrow$ *flip.tdfrs.t2.right-Galois*)$\restriction_{\textit{in-dom flip.tpdfr.R}}\restriction_{\textit{in-codom flip.tpdfr.L}}$ = (*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*$\restriction_{\textit{in-dom}\ (\leq_{L2})}\restriction_{\textit{in-codom}\ (\leq_{R2})}$)$\restriction_{\textit{in-d}}$ = (*flip.tdfrs.t1.right-Galois* $\Rrightarrow$ *?S*)$\restriction_{\textit{in-dom flip.tpdfr.R}}\restriction_{\textit{in-codom flip.tpdfr.L}}$.

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*; $((x1\ x2 :: ?L1.0^{-1}) \Rightarrow_m (x3\ x4 :: ?L1.0) \Rightarrow ?L1.0\ x1\ x3 \longrightarrow (\leq))\ ?L2.0$; $\bigwedge x1\ x2.$ *?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 $((\leq_{L1}) \equiv_{PER} (\leq_{R1}))$ *l1 r1*
and $\bigwedge x\ x'.\ x\ _{L1}\!\lessapprox x' \Longrightarrow ((\leq_{L2\ x\ (r1\ x')}) \equiv_{PER} (\leq_{R2\ (l1\ x)\ x'}))\ (l2_{x'\ x})\ (r2_{x\ x'})$
and $((x1\ x2 :: (\geq_{L1})) \Rightarrow_m (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rightarrow (\leq))\ L2$
and $((x1'\ x2' :: (\geq_{R1})) \Rightarrow_m (x3'\ x4' :: (\leq_{R1}) \mid x1' \leq_{R1} x3') \Rightarrow (\leq))\ R2$
and $((x1'\ x2' :: (\leq_{R1})) \Rightarrow_m (x1\ x2 :: (\leq_{L1}) \mid x2\ _{L1}\!\lessapprox x1') \Rightarrow$
in-field $(\leq_{L2\ x1\ (r1\ x2')}) \Rightarrow (\leq_{R2\ (l1\ x1)\ x2'}))\ l2$
and $((x1\ x2 :: (\leq_{L1})) \Rightarrow_m (x1'\ x2' :: (\leq_{R1}) \mid x2\ _{L1}\!\lessapprox x1') \Rightarrow$
in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rightarrow (\leq_{L2\ x1\ (r1\ x2')}))\ r2$
shows $((\leq_L) \equiv_{PER} (\leq_R))$ *l r*
proof −
from *assms* **have** $((\leq_L) \equiv_{PER} (\leq_R)) = (tpdfr.L \equiv_{PER} tpdfr.R)$
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 $((\leq_{L1}) \equiv_{PER} (\leq_{R1}))$ *l1 r1*
and $((\leq_{L2}) \equiv_{PER} (\leq_{R2}))$ *l2 r2*
shows $((\leq_{L}) \equiv_{PER} (\leq_{R}))$ *l r*
proof −
from *assms* **have** $((\leq_{L}) \equiv_{PER} (\leq_{R})) = (tpfr.tpdfr.L \equiv_{PER} tpfr.tpdfr.R)$
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*: $({}_{L}\lessapprox) = (\leq_{L})$

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* ⊆ {(*x*, *y*). *R1 x y*}
and *Fset2 z* ⊆ {(*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* ⟷
(*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 ⋀*x1 y1*. *x1* ∈ *Fset1 x* ⟹ *y1* ∈ *Fset1 y* ⟹ *R1 x1 y1* ⟹ *S1 x1 y1*
and ⋀*x2 y2*. *x2* ∈ *Fset2 x* ⟹ *y2* ∈ *Fset2 y* ⟹ *R2 x2 y2* ⟹ *S2 x2 y2*
and ⋀*x3 y3*. *x3* ∈ *Fset3 x* ⟹ *y3* ∈ *Fset3 y* ⟹ *R3 x3 y3* ⟹ *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 ≤ S1 R2 ≤ S2 R3 ≤ S3
 shows Frel R1 R2 R3 ≤ 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 ⋀x1. x1 ∈ Fset1 x ⟹ R1 x1 x1
 and ⋀x2. x2 ∈ Fset2 x ⟹ R2 x2 x2
 and ⋀x3. x3 ∈ Fset3 x ⟹ R3 x3 x3
 shows Frel R1 R2 R3 x x
 by (rule Frel-mono-strong[OF Frel-eq-self[of x]];
  drule assms, hypsubst, assumption)

lemma Frel-cong:
 assumes ⋀x1 y1. x1 ∈ Fset1 x ⟹ y1 ∈ Fset1 y ⟹ R1 x1 y1 ⟷ R1' x1 y1
 and ⋀x2 y2. x2 ∈ Fset2 x ⟹ y2 ∈ Fset2 y ⟹ R2 x2 y2 ⟷ R2' x2 y2
 and ⋀x3 y3. x3 ∈ Fset3 x ⟹ y3 ∈ Fset3 y ⟹ R3 x3 y3 ⟷ R3' x3 y3
 shows Frel R1 R2 R3 x y = Frel R1' R2' R3' x y
 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*: $\bigwedge$*R1 R2 R3 S1 S2 S3*.
Frel R1 R2 R3 $\circ\circ$ *Frel S1 S2 S3* $\leq$ *Frel* (*R1* $\circ\circ$ *S1*) (*R2* $\circ\circ$ *S2*) (*R3* $\circ\circ$ *S3*)

lemma *fst-sndOp-eq-snd-fstOp*: *fst* $\circ$ *BNF-Def.sndOp P Q* = *snd* $\circ$ *BNF-Def.fstOp P Q*
unfolding *fstOp-def sndOp-def* **by** (*intro ext*) *simp*

lemma *Frel-rel-comp-le-Frel-comp*:
Frel (*R1* $\circ\circ$ *S1*) (*R2* $\circ\circ$ *S2*) (*R3* $\circ\circ$ *S3*) $\leq$ (*Frel R1 R2 R3* $\circ\circ$ *Frel S1 S2 S3*)
apply (*rule le-relI*)
apply (*erule FrelE*)
apply (*rule rel-compI*[**where** *?y=Fmap* (*snd* $\circ$ *BNF-Def.fstOp R1 S1*)
(*snd* $\circ$ *BNF-Def.fstOp R2 S2*) (*snd* $\circ$ *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*, *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*, *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* :: $('a \Rightarrow bool) \Rightarrow ('b \Rightarrow bool) \Rightarrow ('c \Rightarrow bool) \Rightarrow$
$('d, 'a, 'b, 'c)\ F \Rightarrow bool$
where $Fpred\ P1\ P2\ P3\ x \equiv Frel\ ((=)\restriction_{P1})\ ((=)\restriction_{P2})\ ((=)\restriction_{P3})\ x\ x$

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 rel-restrict-leftE,
rule rel-restrict-leftI,
assumption,
rule assms,
assumption+)
done

lemma *Fpred-top*: $Fpred\ \top\ \top\ \top\ x$
apply (*subst Fpred-def*)
apply (*rule Frel-refl-strong*;
subst rel-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-rel-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 Set.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 rel-restrict-leftI,
rule refl,
drule rev-subsetD,
assumption,
elim CollectE case-prodE rel-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* ⇒ *'a2* ⇒ *bool*
and *R2* :: *'b1* ⇒ *'b2* ⇒ *bool*
and *R3* :: *'c1* ⇒ *'c2* ⇒ *bool*
and *P1* :: *'a1* ⇒ *bool*
and *P2* :: *'b1* ⇒ *bool*
and *P3* :: *'c1* ⇒ *bool*
shows (*Frel R1 R2 R3* :: (*'d*, *'a1*, *'b1*, *'c1*) *F* ⇒ -)↾$_{Fpred\ P1\ P2\ P3 :: ('d,\ 'a1,\ 'b1,\ 'c1)\ F \Rightarrow -}$
=
Frel (*R1*↾$_{P1}$) (*R2*↾$_{P2}$) (*R3*↾$_{P3}$)
apply (*intro ext*)
apply (*rule iffI*)

apply (*erule rel-restrict-leftE*)
apply (*elim FpredE*)
apply (*rule Frel-mono-strong*,
assumption;
rule rel-restrict-leftI,
assumption+)
apply (*rule rel-restrict-leftI*)
apply (*rule Frel-mono-strong*,
assumption;
erule rel-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-rel-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* ⇒ -)$\restriction_{Fpred\ P1\ P2\ P3}$ :: (*'d*, *'a2*, *'b2*, *'c2*) *F* ⇒ -
=
Frel ($R1\restriction_{P1}$) ($R2\restriction_{P2}$) ($R3\restriction_{P3}$)
apply (*subst rel-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 rel-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 $L3 :: 'a3 \Rightarrow 'a3 \Rightarrow bool$
and $R3 :: 'b3 \Rightarrow 'b3 \Rightarrow bool$
and $l3 :: 'a3 \Rightarrow 'b3$
and $r3 :: 'b3 \Rightarrow 'a3$
begin

notation $L1$ (**infix** $\leq_{L1}$ 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 $t1.ge\text{-}left$ (**infix** $\geq_{L1}$ 50)
notation $t1.ge\text{-}right$ (**infix** $\geq_{R1}$ 50)
notation $t2.ge\text{-}left$ (**infix** $\geq_{L2}$ 50)
notation $t2.ge\text{-}right$ (**infix** $\geq_{R2}$ 50)
notation $t3.ge\text{-}left$ (**infix** $\geq_{L3}$ 50)
notation $t3.ge\text{-}right$ (**infix** $\geq_{R3}$ 50)

notation $t1.left\text{-}Galois$ (**infix** ${}_{L1}\lessapprox$ 50)
notation $t1.right\text{-}Galois$ (**infix** ${}_{R1}\lessapprox$ 50)
notation $t2.left\text{-}Galois$ (**infix** ${}_{L2}\lessapprox$ 50)
notation $t2.right\text{-}Galois$ (**infix** ${}_{R2}\lessapprox$ 50)
notation $t3.left\text{-}Galois$ (**infix** ${}_{L3}\lessapprox$ 50)
notation $t3.right\text{-}Galois$ (**infix** ${}_{R3}\lessapprox$ 50)

notation $t1.ge\text{-}Galois\text{-}left$ (**infix** $\gtrapprox_{L1}$ 50)
notation $t1.ge\text{-}Galois\text{-}right$ (**infix** $\gtrapprox_{R1}$ 50)
notation $t2.ge\text{-}Galois\text{-}left$ (**infix** $\gtrapprox_{L2}$ 50)
notation $t2.ge\text{-}Galois\text{-}right$ (**infix** $\gtrapprox_{R2}$ 50)
notation $t3.ge\text{-}Galois\text{-}left$ (**infix** $\gtrapprox_{L3}$ 50)
notation $t3.ge\text{-}Galois\text{-}right$ (**infix** $\gtrapprox_{R3}$ 50)

notation $t1.right\text{-}ge\text{-}Galois$ (**infix** ${}_{R1}\gtrapprox$ 50)
notation $t1.Galois\text{-}right$ (**infix** $\lessapprox_{R1}$ 50)
notation $t2.right\text{-}ge\text{-}Galois$ (**infix** ${}_{R2}\gtrapprox$ 50)
notation $t2.Galois\text{-}right$ (**infix** $\lessapprox_{R2}$ 50)
notation $t3.right\text{-}ge\text{-}Galois$ (**infix** ${}_{R3}\gtrapprox$ 50)
notation $t3.Galois\text{-}right$ (**infix** $\lessapprox_{R3}$ 50)

notation $t1.left\text{-}ge\text{-}Galois$ (**infix** ${}_{L1}\gtrapprox$ 50)
notation $t1.Galois\text{-}left$ (**infix** $\lessapprox_{L1}$ 50)
notation $t2.left\text{-}ge\text{-}Galois$ (**infix** ${}_{L2}\gtrapprox$ 50)
notation $t2.Galois\text{-}left$ (**infix** $\lessapprox_{L2}$ 50)
notation $t3.left\text{-}ge\text{-}Galois$ (**infix** ${}_{L3}\gtrapprox$ 50)
notation $t3.Galois\text{-}left$ (**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*: ε = *Fmap* ε_1 ε_2 ε_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 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*
rel-restrict-right-eq[*of* $\lambda x\ y.\ x \leq_{L1} r1\ y$ *in-codom* $(\leq_{R1})$,
unfolded rel-restrict-left-pred-def rel-inv-iff-rel]
rel-restrict-right-eq[*of* $\lambda x\ y.\ x \leq_{L2} r2\ y$ *in-codom* $(\leq_{R2})$,
unfolded rel-restrict-left-pred-def rel-inv-iff-rel]
rel-restrict-right-eq[*of* $\lambda x\ y.\ x \leq_{L3} r3\ y$ *in-codom* $(\leq_{R3})$,

unfolded rel-restrict-left-pred-def rel-inv-iff-rel])
apply (*subst* (*asm*) *Frel-restrict-right-Fpred-eq-Frel-restrict-right*[*symmetric*])
apply (*erule rel-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

context
fixes *R1* :: $'a \Rightarrow 'a \Rightarrow bool$ **and** *R2* :: $'b \Rightarrow 'b \Rightarrow bool$ **and** *R3* :: $'c \Rightarrow 'c \Rightarrow bool$
begin

lemma *reflexive-on-in-field-FrelI*:
assumes *reflexive-on* (*in-field R1*) *R1*
and *reflexive-on* (*in-field R2*) *R2*
and *reflexive-on* (*in-field R3*) *R3*
defines $R \equiv Frel\ R1\ R2\ R3$
shows *reflexive-on* (*in-field R*) *R*
apply (*subst reflexive-on-iff-eq-restrict-le*)
apply (*subst Frel-eq*[*symmetric*])
apply (*unfold R-def*)
apply (*subst in-field-Frel-eq-Fpred-in-in-field*)
apply (*subst rel-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-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

end

context *transport-natural-functor*
begin

lemmas *reflexive-on-in-field-leftI* = *reflexive-on-in-field-FrelI*
[*of L1 L2 L3*, *folded transport-defs*]

lemmas *transitive-leftI* = *transitive-FrelI*[*of L1 L2 L3*, *folded transport-defs*]

lemmas *preorder-on-in-field-leftI* = *preorder-on-in-field-FrelI*
[*of L1 L2 L3*, *folded transport-defs*]

lemmas *symmetricI* = *symmetric-FrelI*[*of L1 L2 L3*, *folded transport-defs*]

lemmas *partial-equivalence-rel-leftI* = *partial-equivalence-rel-FrelI*
[*of L1 L2 L3*, *folded transport-defs*]

end

end

2.10.5 Order Equivalence

theory *Transport-Natural-Functors-Order-Equivalence*
imports
Transport-Natural-Functors-Base
begin

context
fixes *R1* :: *'a* ⇒ *'a* ⇒ *bool* **and** *R2* :: *'b* ⇒ *'b* ⇒ *bool* **and** *R3* :: *'c* ⇒ *'c* ⇒ *bool*
and *f1* :: *'a* ⇒ *'a* **and** *f2* :: *'b* ⇒ *'b* **and** *f3* :: *'c* ⇒ *'c*
and *R* :: (*'d*, *'a*, *'b*, *'c*) *F* ⇒ (*'d*, *'a*, *'b*, *'c*) *F* ⇒ *bool*
and *f* :: (*'d*, *'a*, *'b*, *'c*) *F* ⇒ (*'d*, *'a*, *'b*, *'c*) *F*
defines *R* ≡ *Frel R1 R2 R3* **and** *f* ≡ *Fmap f1 f2 f3*
begin

lemma *inflationary-on-in-dom-FrelI*:
assumes *inflationary-on* (*in-dom R1*) *R1 f1*
and *inflationary-on* (*in-dom R2*) *R2 f2*
and *inflationary-on* (*in-dom R3*) *R3 f3*
shows *inflationary-on* (*in-dom R*) *R f*

apply (*unfold R-def f-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*
shows *inflationary-on* (*in-codom R*) *R f*
apply (*unfold R-def f-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

end

context
fixes *R1* :: $'a \Rightarrow 'a \Rightarrow$ *bool* **and** *R2* :: $'b \Rightarrow 'b \Rightarrow$ *bool* **and** *R3* :: $'c \Rightarrow 'c \Rightarrow$ *bool*
and *f1* :: $'a \Rightarrow 'a$ **and** *f2* :: $'b \Rightarrow 'b$ **and** *f3* :: $'c \Rightarrow 'c$
and *R* :: $('d, 'a, 'b, 'c)\ F \Rightarrow ('d, 'a, 'b, 'c)\ F \Rightarrow$ *bool*
and *f* :: $('d, 'a, 'b, 'c)\ F \Rightarrow ('d, 'a, 'b, 'c)\ F$
defines *R* $\equiv$ *Frel R1 R2 R3* **and** *f* $\equiv$ *Fmap f1 f2 f3*
begin

lemma *inflationary-on-in-field-FrelI*:
assumes *inflationary-on* (*in-field R1*) *R1 f1*
and *inflationary-on* (*in-field R2*) *R2 f2*
and *inflationary-on* (*in-field R3*) *R3 f3*
shows *inflationary-on* (*in-field R*) *R f*
apply (*unfold R-def f-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*
shows *deflationary-on* (*in-dom R*) *R f*
apply (*unfold R-def f-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*
shows *deflationary-on* (*in-codom R*) *R f*
apply (*unfold R-def f-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

end

context
fixes *R1* :: $'a \Rightarrow 'a \Rightarrow$ *bool* **and** *R2* :: $'b \Rightarrow 'b \Rightarrow$ *bool* **and** *R3* :: $'c \Rightarrow 'c \Rightarrow$ *bool*
and *f1* :: $'a \Rightarrow 'a$ **and** *f2* :: $'b \Rightarrow 'b$ **and** *f3* :: $'c \Rightarrow 'c$
and *R* :: ($'d$, $'a$, $'b$, $'c$) *F* $\Rightarrow$ ($'d$, $'a$, $'b$, $'c$) *F* $\Rightarrow$ *bool*
and *f* :: ($'d$, $'a$, $'b$, $'c$) *F* $\Rightarrow$ ($'d$, $'a$, $'b$, $'c$) *F*
defines *R* $\equiv$ *Frel R1 R2 R3* **and** *f* $\equiv$ *Fmap f1 f2 f3*
begin

lemma *deflationary-on-in-field-FrelI*:
assumes *deflationary-on* (*in-field R1*) *R1 f1*
and *deflationary-on* (*in-field R2*) *R2 f2*
and *deflationary-on* (*in-field R3*) *R3 f3*
shows *deflationary-on* (*in-field R*) *R f*
apply (*unfold R-def f-def*)
apply (*subst deflationary-on-eq-inflationary-on-rel-inv*)
apply (*subst in-field-rel-inv-eq-in-field*[*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-in-field,
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*
shows *rel-equivalence-on* (*in-field R*) *R f*
apply (*unfold R-def f-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*)
apply (*rule inflationary-on-in-field-FrelI*; *assumption*)
apply (*fold R-def f-def*)
apply (*rule deflationary-on-in-field-FrelI*; *assumption*)
done

end

context *transport-natural-functor*
begin

lemmas *inflationary-on-in-field-unitI* = *inflationary-on-in-field-FrelI*
[*of L1* η_1 *L2* η_2 *L3* η_3, *folded transport-defs unit-eq-Fmap*]

lemmas *deflationary-on-in-field-unitI* = *deflationary-on-in-field-FrelI*
[*of L1* η_1 *L2* η_2 *L3* η_3, *folded transport-defs unit-eq-Fmap*]

lemmas *rel-equivalence-on-in-field-unitI* = *rel-equivalence-on-in-field-FrelI*
[*of L1* η_1 *L2* η_2 *L3* η_3, *folded transport-defs unit-eq-Fmap*]

interpretation *flip* :
transport-natural-functor R1 L1 r1 l1 R2 L2 r2 l2 R3 L3 r3 l3
rewrites *flip.unit* $\equiv \varepsilon$ **and** *flip.t1.unit* $\equiv \varepsilon_1$

and *flip.t2.unit* $\equiv \varepsilon_2$ **and** *flip.t3.unit* $\equiv \varepsilon_3$
by (*simp-all only*: *order-functors.flip-counit-eq-unit*)

lemma *order-equivalenceI*:
assumes $((\leq_{L1}) \equiv_o (\leq_{R1}))$ *l1 r1*
and $((\leq_{L2}) \equiv_o (\leq_{R2}))$ *l2 r2*
and $((\leq_{L3}) \equiv_o (\leq_{R3}))$ *l3 r3*
shows $((\leq_L) \equiv_o (\leq_R))$ *l r*
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 $((\leq_{L1}) \equiv_{pre} (\leq_{R1}))$ *l1 r1*
and $((\leq_{L2}) \equiv_{pre} (\leq_{R2}))$ *l2 r2*
and $((\leq_{L3}) \equiv_{pre} (\leq_{R3}))$ *l3 r3*
shows $((\leq_L) \equiv_{pre} (\leq_R))$ *l r*
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 $((\leq_{L1}) \equiv_{PER} (\leq_{R1}))$ *l1 r1*
and $((\leq_{L2}) \equiv_{PER} (\leq_{R2}))$ *l2 r2*
and $((\leq_{L3}) \equiv_{PER} (\leq_{R3}))$ *l3 r3*
shows $((\leq_{L}) \equiv_{PER} (\leq_{R}))$ *l r*
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 $((x\ y ::$ *rel-if B R*$) \Rrightarrow_m$ (*rel-if B'* $(S\ x\ y)))\ f$
using *assms* **by** (*intro dep-mono-wrt-relI*) *auto*

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 (*intro transitive-onI*) (*force dest*: *transitive-onD*)

end

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

corollary *preorder-rel-if-if-preorderI* [*intro*]:
assumes $B \Longrightarrow$ *preorder R*
shows *preorder* (*rel-if B R*)
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*

end

context *galois-prop*
begin

interpretation *rel-if* : *galois-prop rel-if B* $(\leq_L)$ *rel-if B'* $(\leq_R)$ *l r* **.**
interpretation *flip-inv* : *galois-prop* $(\geq_R)$ $(\geq_L)$ *r l* **.**

lemma *rel-if-half-galois-prop-left-if-iff-if-half-galois-prop-leftI*:
assumes $B \Longrightarrow B' \Longrightarrow ((\leq_L)\ {}_h\trianglelefteq\ (\leq_R))$ *l r*
and $B \longleftrightarrow B'$
shows $((\textit{rel-if}\ B\ (\leq_L))\ {}_h\trianglelefteq\ (\textit{rel-if}\ B'\ (\leq_R)))$ *l r*
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 $B \Longrightarrow B' \Longrightarrow ((\leq_L)\ \trianglelefteq_h\ (\leq_R))$ *l r*
and $B \longleftrightarrow B'$
shows $((\textit{rel-if}\ B\ (\leq_L))\ \trianglelefteq_h\ (\textit{rel-if}\ B'\ (\leq_R)))$ *l r*
using *assms* **by** (*intro rel-if.half-galois-prop-rightI*) *fastforce*

lemma *rel-if-galois-prop-if-iff-if-galois-propI*:
assumes $B \Longrightarrow B' \Longrightarrow ((\leq_L) \trianglelefteq (\leq_R))$ *l r*
and $B \longleftrightarrow B'$
shows $((\textit{rel-if}\ B\ (\leq_L)) \trianglelefteq (\textit{rel-if}\ B'\ (\leq_R)))$ *l r*
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* ⇒ *'a1* ⇒ *bool*
and *R1* :: *'a2* ⇒ *'a2* ⇒ *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 $tdfr.ge\text{-}left$ (**infix** $\geq_L$ *50*)
notation $tdfr.ge\text{-}right$ (**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 $((x1\ x2 :: (\geq_{L1})) \Rightarrow_m (x3\ x4 :: (\leq_{L1}) \mid x1 \leq_{L1} x3) \Rightarrow (\leq))$ *L2*
and $((x1'\ x2' :: (\geq_{R1})) \Rightarrow_m (x3'\ x4' :: (\leq_{R1}) \mid x1' \leq_{R1} x3') \Rightarrow (\leq))$ *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 $((x1'\ x2' :: (\leq_{R1})) \Rightarrow_m (x1\ x2 :: (\leq_{L1}) \mid x2\ _{L1}\lessapprox x1') \Rightarrow$
in-field $(\leq_{L2\ x1\ (r1\ x2')}) \Rightarrow (\leq_{R2\ (l1\ x1)\ x2'}))$ $(\lambda\text{-}\ \text{-}.\ l2)$
by (*intro dep-mono-wrt-relI Dep-Fun-Rel-relI Dep-Fun-Rel-predI rel-if-if-impI*)
(*auto 10 0 dest*!: *per2I*)
moreover have
$((x1\ x2 :: (\leq_{L1})) \Rightarrow_m (x1'\ x2' :: (\leq_{R1}) \mid x2\ _{L1}\lessapprox x1') \Rightarrow$

in-field $(\leq_{R2\ (l1\ x1)\ x2'}) \Rrightarrow (\leq_{L2\ x1\ (r1\ x2')})$) $(\lambda\text{- -}.\ r2)$
by (*intro dep-mono-wrt-relI Dep-Fun-Rel-relI Dep-Fun-Rel-predI rel-if-if-impI*)
(*auto 10 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-equivalence-rel-equivalence ?L1.0 ?R1.0 ?l1.0 ?r1.0*; $\bigwedge x\ x'$. *galois-rel.Galois ?L1.0 ?R1.0 ?r1.0 x x'* $\Longrightarrow$ *transport.partial-equivalence-rel-equivalence* (*?L2.0 x* (*?r1.0 x'*)) (*?R2.0* (*?l1.0 x*) *x'*) *?l2.0 ?r2.0*; $((x1\ x2 :: ?L1.0^{-1})$ $\Rrightarrow_m$ $(x3\ x4 :: ?L1.0) \Rrightarrow ?L1.0\ x1\ x3 \longrightarrow (\leq))$ *?L2.0*; $((x1'\ x2' :: ?R1.0^{-1})$ $\Rrightarrow_m$ $(x3'\ x4' :: ?R1.0) \Rrightarrow ?R1.0\ x1'\ x3' \longrightarrow (\leq))$ *?R2.0*$]\!] \Longrightarrow$ *transport.partial-equivalence-rel-equivalence* (*transport-Dep-Fun-Rel.L ?L1.0 ?L2.0*) (*transport-Dep-Fun-Rel.L ?R1.0 ?R2.0*) (*transport-Dep-Fun-Rel.l ?r1.0* (λ- -. *?l2.0*)) (*transport-Dep-Fun-Rel.l ?l1.0* (λ- -. *?r2.0*)): The relations can be dependent, but the functions must be simple. This is not production ready, but a proof of concept.

The package provides a command **trp-term**, which sets up the required goals to prove a given term. See the examples in this directory for some use cases and refer to [2] for more details.

Theorem Setups **context** *transport*
begin

lemma *left-Galois-left-if-left-rel-if-partial-equivalence-rel-equivalence*:

assumes $((\leq_L) \equiv_{PER} (\leq_R))$ *l r*
and $x \leq_L x'$
shows $x\ {}_L\lessapprox\ l\ x$
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* $\equiv ((\leq_L) \equiv_{PER} (\leq_R))\ l\ r \wedge x\ {}_L\lessapprox\ y$

The choice of x' is arbitrary. All we need is *in-dom* $(\leq_L)$ *x*.

lemma *transport-per-start*:
assumes $((\leq_L) \equiv_{PER} (\leq_R))$ *l r*
and $x \leq_L x'$
shows *transport-per x* (*l x*)
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 $((\leq_{L1}) \equiv_{PER} (\leq_{R1}))$ *l1 r1*
and $((\leq_{L2}) \equiv_{PER} (\leq_{R2}))$ *l2 r2*
shows $({}_L\lessapprox) = (({}_{L1}\lessapprox) \Rrightarrow ({}_{L2}\lessapprox))$
proof (*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 $g \leq_R g$
proof −
from *assms* **have** *per*: $((\leq_L) \equiv_{PER} (\leq_R))\ l\ r$
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 $((_{L1}\lessapprox) \Rightarrow (_{L2}\lessapprox))\ f\ g$
with *assms* **have** $((_{L1}\lessapprox) \Rightarrow (_{L2}\lessapprox))\restriction_{in\text{-}dom\ (\leq_L)}{}^{\uparrow}{}_{in\text{-}codom\ (\leq_R)}\ f\ g$
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* = *Fun-Rel-relD*[*rotated*]

lemma *related-Fun-Rel-lambdaI*:
assumes $\bigwedge x\ y.\ R\ x\ y \Longrightarrow S\ (f\ x)\ (g\ y)$
and $T = (R \Rightarrow S)$
shows $T\ f\ g$
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‹
structure A = *Standard-Mixed-Unification*
›
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*
|> *Type-Unification.e-unify Unification-Util.unify-types*),
prems-unifier = *SOME* (*Transport-Mixed-Unification.first-higherp-decomp-comb-higher-unify*
|> *Unification-Combinator.norm-unifier Envir-Normalisation.beta-norm-term-unif*),
normalisers = *SOME Transport-Mixed-Unification.norms-first-higherp-decomp-comb-higher-unify*,
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*
(*Unification-Util.inst-norm-term′*
Transport-Mixed-Unification.norms-first-higherp-decomp-comb-higher-unify)
|> *K*)
(*Transport-Unification-Combine.default-metadata Transport-Unification-Hints.binding*)›]]

Prototype **ML-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* (′((-)≲≈(-) (-))′)

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-Alignment-Binary-Relations
HOL−Library.IArray
begin

Summary Dependent function relator examples from [2]. Refer to the paper for more details.

context
includes *galois-rel-syntax transport-syntax*
notes
transport.rel-if-partial-equivalence-rel-equivalence-if-iff-if-partial-equivalence-rel-equivalenceI
[*rotated*, *per-intro*]
transport-Dep-Fun-Rel-no-dep-fun.partial-equivalence-rel-equivalenceI
[*ML-Krattr* ‹*Drule.rearrange-prems* [*1*] #> *Drule.rearrange-prems* [*2*,*3*]›,

per-intro]
begin

interpretation *transport L R l r* **for** *L R l r* **.**

abbreviation *Zpos* ≡ $((=_{(\leq)(0 :: int)}) :: int \Rightarrow -)$

lemma *Zpos-per* [*per-intro*]: $(Zpos \equiv_{PER} (=))$ *nat int*
by *fastforce*

lemma *sub-parametric* [*trp-in-dom*]:
$((i\text{ -} :: Zpos) \Rrightarrow (j\text{ -} :: Zpos \mid j \leq i) \Rrightarrow Zpos)\ (-)\ (-)$
by *fastforce*

trp-term *nat-sub* :: *nat* ⇒ *nat* ⇒ *nat* **where** $x = (-) :: int \Rightarrow -$
and $L = (i\text{ -} :: Zpos) \Rrightarrow (j\text{ -} :: Zpos \mid j \leq i) \Rrightarrow Zpos$
and $R = (n\text{ -} :: (=)) \Rrightarrow (m\text{ -} :: (=) \mid m \leq n) \Rrightarrow (=)$

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 [*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*]:
$((xs\text{ -} :: LRel\ R) \Rrightarrow (i\text{ -} :: (=) \mid i < length\ xs) \Rrightarrow R)\ (!)\ (!)$
by (*fastforce simp*: *list-all2-lengthD list-all2-nthD2*)

context
fixes *R* :: *′a* ⇒ *′a* ⇒ *bool* **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 $L = ((xs\ -) :: LRel\ R) \Rrightarrow (i\ -) :: (=) \mid i < length\ xs) \Rrightarrow R)$
and $R = ((xs\ -) :: IARel\ R) \Rrightarrow (i\ -) :: (=) \mid i < IArray.length\ xs) \Rrightarrow R)$
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 $LFSL\ xs\ xs' \equiv fset\text{-}of\text{-}list\ xs = fset\text{-}of\text{-}list\ xs'$
abbreviation (*input*) $(LFSR :: 'a\ fset \Rightarrow -) \equiv (=)$
definition $LSL\ xs\ xs' \equiv set\ xs = set\ xs'$
abbreviation (*input*) $(LSR :: 'a\ set \Rightarrow -) \equiv (=_{finite\ ::\ 'a\ set \Rightarrow bool})$

interpretation $t : transport\ LSL\ R\ l\ r$ **for** $LSL\ R\ l\ r$ **.**

Proofs of equivalences.

lemma *list-fset-PER* [*per-intro*]: $(LFSL \equiv_{PER} LFSR)\ fset\text{-}of\text{-}list\ sorted\text{-}list\text{-}of\text{-}fset$
unfolding *LFSL-def* **by** *fastforce*

lemma *list-set-PER* [*per-intro*]: $(LSL \equiv_{PER} LSR)\ set\ sorted\text{-}list\text{-}of\text{-}set$
unfolding *LSL-def* **by** *fastforce*

We can rewrite the Galois relators in the following theorems to the relator of the paper.

definition $LFS\ xs\ s \equiv fset\text{-}of\text{-}list\ xs = s$
definition $LS\ xs\ s \equiv set\ xs = s$

lemma *LFSL-Galois-eq-LFS*: $(_{LFSL}\overset{<}{\approx}_{LFSR\ sorted\text{-}list\text{-}of\text{-}fset}) \equiv LFS$
unfolding *LFS-def LFSL-def* **by** (*intro eq-reflection ext*) (*auto*)
lemma *LFSR-Galois-eq-inv-LFS*: $(_{LFSR}\overset{<}{\approx}_{LFSL\ fset\text{-}of\text{-}list}) \equiv LFS^{-1}$
unfolding *LFS-def LFSL-def* **by** (*intro eq-reflection ext*) (*auto*)
lemma *LSL-Galois-eq-LS*: $(_{LSL}\overset{<}{\approx}_{LSR\ sorted\text{-}list\text{-}of\text{-}set}) \equiv LS$
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 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* ≡$_{PER}$ *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 Fun-Rel-relI*) *simp*

lemma [*trp-uhint*]: *P* ≡ (=) ⟹ *P* ≡ (=) ⇛ (=) **by** *simp*
lemma [*trp-uhint*]: *P* ≡ ⊤ ⟹ (=$_{P\ ::\ 'a\ \Rightarrow\ bool}$) ≡ ((=) :: *′a* ⇒ -) **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 Fun-Rel-relI*) *simp*

trp-term *filter-set* :: (*′a* :: *linorder* ⇒ *bool*) ⇒ *′a set* ⇒ *′a set*
where *x* = *filter* :: (*′a* :: *linorder* ⇒ *bool*) ⇒ *′a list* ⇒ *′a list*
by *trp-prover*

lemma (((=) ⇛ (=)) ⇛ *LS* ⇛ *LS*) *filter filter-set* **by** (*trp-hints-resolve filter-set-related′*)

lemma *append-parametric* [*trp-in-dom*]:
(*LSL* ⇛ *LSL* ⇛ *LSL*) *append append*
unfolding *LSL-def* **by** (*intro Fun-Rel-relI*) *simp*

trp-term *append-set* :: ('*a* :: *linorder*) *set* ⇒ '*a set* ⇒ '*a set*
where *x* = *append* :: ('*a* :: *linorder*) *list* ⇒ '*a list* ⇒ '*a list*
by *trp-prover*

lemma (*LS* ⇛ *LS* ⇛ *LS*) *append append-set* **by** (*trp-hints-resolve append-set-related'*)

The prototype also provides a simplified definition.

lemma *append-set s s'* ≡ *set* (*sorted-list-of-set s*) ∪ *set* (*sorted-list-of-set s'*)
by (*fact append-set-app-eq*)

lemma *finite s* ⟹ *finite s'* ⟹ *append-set s s'* = *s* ∪ *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* (≤$_L$) *l r T*
shows *T* = *t.Galois*
proof (*intro ext iffI*)
fix *x y* **assume** *T x y*
with *assms* **have** *x* ≤$_L$ *x l x* = *y* **using** *Quotient-cr-rel* **by** *auto*
with *assms* **have** *r* (*l x*) ≤$_L$ *x r* (*l x*) ≤$_L$ *r y*
using *Quotient-rep-abs Quotient-rep-reflp* **by** *auto*
with *assms* **have** *x* ≤$_L$ *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 $((\leq_L) \equiv_{pre} (=))\ l\ r$
shows *Quotient* $(\leq_L)\ l\ r$ *t.Galois*
proof (*rule QuotientI*)
from *assms* **show** *g2*: $l\ (r\ y) = y$ **for** y **by** *fastforce*
from *assms* **show** $r\ y \leq_L r\ y$ **for** y **by** *blast*
show *g1*: $x \leq_L x' \longleftrightarrow x \leq_L x \wedge x' \leq_L x' \wedge l\ x = l\ x'$
(**is** *?lhs* $\longleftrightarrow$ *?rhs*) **for** $x\ x'$
proof (*rule iffI*)
assume *?rhs*
with *assms* **have** $\eta\ x \leq_L \eta\ x'$ **by** *fastforce*
moreover from ‹*?rhs*› *assms* **have** $x \leq_L \eta\ x\ \eta\ x' \leq_L x'$
by (*blast elim*: *t.preorder-equivalence-order-equivalenceE*)+
moreover from *assms* **have** *transitive* $(\leq_L)$ **by** *blast*
ultimately show $x \leq_L x'$ **by** *blast*
next
assume *?lhs*
with *assms* **show** *?rhs* **by** *blast*
qed
from *assms* **show** *t.Galois* $= (\lambda x\ y.\ x \leq_L x \wedge l\ x = y)$
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* $(\leq_L)\ l\ r\ T$
shows $((\leq_L) \equiv_{PER} (=))\ l\ r$
proof (*rule t.partial-equivalence-rel-equivalence-if-order-equivalenceI*)
from *Quotient-part-equivp*[*OF assms*] **show** *partial-equivalence-rel* $(\leq_L)$
by (*blast elim*: *part-equivpE dest*: *transpD sympD*)
have $x \equiv_L r\ (l\ x)$ **if** *in-field* $(\leq_L)\ x$ **for** x
proof −
from *assms* ‹*in-field* $(\leq_L)\ x$› **have** $x \leq_L x$
using *Quotient-refl1 Quotient-refl2* **by** *fastforce*
with *assms Quotient-rep-abs Quotient-symp* **show** *?thesis*
by (*fastforce dest*: *sympD*)
qed
with *assms* **show** $((\leq_L) \equiv_o (=))\ l\ r$
using *Quotient-abs-rep Quotient-rel-abs Quotient-rep-reflp*
Quotient-abs-rep[*symmetric*]
by (*intro t.order-equivalenceI mono-wrt-relI rel-equivalence-onI*

inflationary-onI deflationary-onI)
auto
qed *auto*

corollary *Quotient-iff-partial-equivalence-rel-equivalence*:
Quotient ($\leq_L$) *l r t.Galois* $\longleftrightarrow$ (($\leq_L$) $\equiv_{PER}$ (=)) *l r*
using *Quotient-if-preorder-equivalence partial-equivalence-rel-equivalence-if-Quotient*
by *blast*

corollary *Quotient-T-eq-ge-Galois-right*:
assumes *Quotient* ($\leq_L$) *l r T*
shows *T* = *t.ge-Galois-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* ⇒ *'a* ⇒ *bool* ≡ ($=_A$)
abbreviation (*input*) *R* :: *'b* ⇒ *'b* ⇒ *bool* ≡ (=)

sublocale *transport?* :
transport-eq-restrict-bijection mem-of A ⊤ :: *'b* ⇒ *bool Abs Rep*
rewrites ($=_{mem\text{-}of\ A}$) ≡ *L*
and ($=_{\top\ ::\ 'b\ \Rightarrow\ bool}$) ≡ *R*
and (*galois-rel.Galois* (=) (=) *Rep*)$\restriction_{mem\text{-}of\ A}\restriction_{\top\ ::\ 'b\ \Rightarrow\ bool}$ ≡
(*galois-rel.Galois* (=) (=) *Rep*)
using *Abs-inverse Rep-inverse*
by (*intro transport-eq-restrict-bijection.intro bijection-onI*)
(*auto intro*!: *eq-reflection galois-rel.left-GaloisI Rep elim*: *galois-rel.left-GaloisE*)

interpretation *galois L R Abs Rep* **.**

lemma *Rep-left-Galois-self*: $Rep\ y\ {}_L\overset{\leq}{\approx}\ y$
using *Rep* **by** (*intro left-GaloisI*) *auto*

definition $AR\ x\ y \equiv x = Rep\ y$

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
HOL-Alignment-Functions
begin

context
includes *galois-rel-syntax transport-syntax*
begin

typedef *pint* = $\{i :: int.\ 0 \leq i\}$ **by** *auto*

interpretation *typedef-pint* : *type-definition Rep-pint Abs-pint* $\{i :: int.\ 0 \leq i\}$
by (*fact type-definition-pint*)

lemma [*trp-relator-rewrite*, *trp-uhint*]:
$(=_{Collect\ ((\leq)\ (0\ ::\ int))})\overset{\leq}{\approx}(=)\ Rep\text{-}pint \equiv typedef\text{-}pint.AR$
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*]:
$(=_{\{s :: 'a\ set.\ finite\ s\}}) :: 'a\ set \Rightarrow \text{-}\overset{\leq}{\approx}(=)\ Rep\text{-}fset \equiv typedef\text{-}fset.AR$
using *typedef-fset.left-Galois-eq-AR* **by** (*intro eq-reflection*) *simp*

lemma *eq-restrict-set-eq-eq-uhint* [*trp-uhint*]:
$(\bigwedge x.\ P\ x \equiv x \in A) \Longrightarrow ((=_{A\ ::\ 'a\ set}) :: 'a \Rightarrow \text{-}) \equiv (=_P)$

by (*intro eq-reflection*) (*auto simp*: *fun-eq-iff*)

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 Fun-Rel-relI*) *fastforce*

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 $L = ((=) \Rrightarrow$ *typedef-fset.L* ⇛ *typedef-fset.L*)
and $R = ((=) \Rrightarrow$ *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* ⇛ *rel-set R* ⇛ *rel-set R*) *insert insert*
by (*intro Fun-Rel-relI rel-setI*) (*auto dest*: *rel-setD1 rel-setD2*)

context
assumes [*trp-uhint*]:

L ≡ *rel-set* (*L1* :: *int* ⇒ *int* ⇒ *bool*) ⟹ *R* ≡ *rel-set* (*R1* :: *pint* ⇒ *pint* ⇒ *bool*)
⟹ *r* ≡ *image r1* ⟹ *S* ≡ (*L1*$\lessapprox$*R1 r1*) ⟹ (*L*$\lessapprox$*R r*) ≡ *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*]:
(((=) ⇛ (=)) ⇛ *typedef-fset.L* ⇛ *typedef-fset.L*) *image image*
by (*intro Fun-Rel-relI*) *auto*

trp-term *fimage* :: ('*a* ⇒ '*b*) ⇒ '*a fset* ⇒ '*b fset* **where** *x* = *image*
by *trp-prover*

lemma *image-parametric'* [*trp-related-intro*]:
(*R* ⇛ *S*) ⇛ *rel-set R* ⇛ *rel-set S*) *image image*
using *transfer-raw*[*simplified HOL-fun-alignment Transfer.Rel-def*]
by *simp*

lemma *Galois-id-hint* [*trp-uhint*]:
(*L* :: '*a* ⇒ '*a* ⇒ *bool*) ≡ *R* ⟹ *r* ≡ *id* ⟹ *E* ≡ *L* ⟹ (*L*$\lessapprox$*R r*) ≡ *E*
by (*simp only*: *eq-reflection*[*OF transport-id.left-Galois-eq-left*])

lemma *Freq* [*trp-uhint*]: *L* ≡ (=) ⇛ (=) ⟹ *L* ≡ (=)
by *auto*

context
fixes *L1 R1 l1 r1 L R l r*
assumes *per1*: (*L1* ≡$_{PER}$ *R1*) *l1 r1*
defines *L* ≡ *rel-set L1* **and** *R* ≡ *rel-set R1*
and *l* ≡ *image l1* **and** *r* ≡ *image r1*
begin

interpretation *transport L R l r* **.**

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* ⇒ *'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* ≡ *image* ((+) (*1* :: *int*))

lemma *set-succ-parametric* [*trp-in-dom*]:
(*typedef-fset.L* ⇛ *typedef-fset.L*) *set-succ set-succ*
unfolding *set-succ-def* **by** *auto*

trp-term *fset-succ* :: *int fset* ⇒ *int fset*
where *x* = *set-succ*
and *L* = *typedef-fset.L* ⇛ *typedef-fset.L*
and *R* = *typedef-fset.R* ⇛ *typedef-fset.R*
by *trp-prover*

trp-term *fset-succ'* :: *int fset* ⇒ *int fset*
where *x* = *set-succ*
and *L* = *typedef-fset.L* ⇛ *typedef-fset.L*
and *R* = *typedef-fset.R* ⇛ *typedef-fset.R*
unfold set-succ-def !

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* ⇒ -))
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: *galois-prop.galois-prop ?L ?R ?l ?r* $\Longrightarrow$ (*galois-rel.Galois ?R*$^{-1}$ *?L*$^{-1}$ *?l*)$^{-1}$ *?x ?y* = *Galois-infix ?x ?L ?R ?r ?y*
- Section 3.4: Partial Galois Connections and Equivalences can be found in *Transport.Half-Galois-Property*, *Transport.Galois-Property*, *Transport.Galois-Connections*, *Transport.Galois-Equivalences*, and *Transport.Order-Equivalences*.
  - Lemma 2: *Transport.Transport-Partial-Quotient-Types* (*results from Appendix*)
  - Lemma 4: ⟦*order-functors.order-equivalence ?L ?R ?l ?r*; *transitive ?L*; *transitive ?R*⟧ $\Longrightarrow$ *galois.galois-equivalence ?L ?R ?l ?r*
  - Lemma 5: ⟦*galois.galois-equivalence ?L ?R ?l ?r*; *reflexive-on* (*in-field ?L*) *?L*; *reflexive-on* (*in-field ?R*) *?R*⟧ $\Longrightarrow$ *order-functors.order-equivalence ?L ?R ?l ?r*
- Section 4.1: Closure (Dependent) Function Relator can be found in `Functions`.
  - Monotone function relator *Transport.Monotone-Function-Relator*.
  - Setup of construction *Transport.Transport-Functions-Base*.
  - Theorem 1: see *Transport.Transport-Functions*
  - Theorem 2: see ⟦*transport.preorder-equivalence ?L1.0 ?R1.0 ?l1.0 ?r1.0*; $\bigwedge$*x x'*. *Galois-infix x ?L1.0 ?R1.0 ?r1.0 x'* $\Longrightarrow$ *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* $\longrightarrow$ ($\leq$)) *?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*; *in-dom* (*transport-Mono-Dep-Fun-Rel.L ?L1.0 ?L2.0*) *?f*; *in-codom* (*transport-Mono-Dep-Fun-Rel.L ?R1.0 ?R2.0*) *?g*⟧ $\Longrightarrow$ *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: ⟦*galois.galois-connection ?L1.0 ?R1.0 ?l1.0 ?r1.0*; *reflexive-on* (*in-codom ?L1.0*) *?L1.0*; *reflexive-on* (*in-dom ?R1.0*) *?R1.0*; *galois.galois-connection ?L2.0 ?R2.0 ?l2.0 ?r2.0*; *transitive ?L2.0*; *transitive ?R2.0*⟧ ⟹ *galois.galois-connection* (*transport-Mono-Dep-Fun-Rel.L ?L1.0* (λ- -. *?L2.0*)) (*transport-Mono-Dep-Fun-Rel.L ?R1.0* (λ- -. *?R2.0*)) (*transport-Dep-Fun-Rel.l ?r1.0* (λ- -. *?l2.0*)) (*transport-Dep-Fun-Rel.l ?l1.0* (λ- -. *?r2.0*))

– Lemma 7: ⟦(*?L1.0* ⇛$_m$ *?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* ⇛$_m$ *?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*⟧ ⟹ *Galois-infix ?f* (*transport-Mono-Dep-Fun-Rel.L ?L1.0* (λ- -. *?L2.0*)) (*transport-Mono-Dep-Fun-Rel.L ?R1.0* (λ- -. *?R2.0*)) (*transport-Dep-Fun-Rel.l ?l1.0* (λ- -. *?r2.0*)) *?g* = (*galois-rel.Galois ?L1.0 ?R1.0 ?r1.0* ⇛ *galois-rel.Galois ?L2.0 ?R2.0 ?r2.0*) *?f ?g*

– Theorem 7: ⟦*galois.galois-connection ?L1.0 ?R1.0 ?l1.0 ?r1.0*; *reflexive-on* (*in-field ?L1.0*) *?L1.0*; *reflexive-on* (*in-field ?R1.0*) *?R1.0*; ⋀*x x'*. *Galois-infix x ?L1.0 ?R1.0 ?r1.0 x'* ⟹ *galois.galois-connection* (*?L2.0 x* (*?r1.0 x'*)) (*?R2.0* (*?l1.0 x*) *x'*) (*?l2.0 x' x*) (*?r2.0 x x'*); ((- *x2* :: *?L1.0*) ⇛$_m$ (*x3 x4* :: *?L1.0*) ⇛ (*?L1.0 x2 x3* ∧ *?L1.0 x4* (*order-functors.unit ?l1.0 ?r1.0 x3*)) ⟶ (λ*x y*. *y* ≤ *x*)) *?L2.0*; ((*x1' x2'* :: *?R1.0*) ⇛$_m$ *?R1.0* (*order-functors.counit ?l1.0 ?r1.0 x2'*) *x1'* ⟶ ((*x3'* - :: *?R1.0*) ⇛ *?R1.0 x2' x3'* ⟶ (≤))) *?R2.0*; ((*x1' x2'* :: *?R1.0*) ⇛$_m$ (*x1 x2* :: *?L1.0*) ⇛ *Galois-infix x2 ?L1.0 ?R1.0 ?r1.0 x1'* ⟶ (*in-field* (*?L2.0 x1* (*?r1.0 x2'*)) ⇛ *?R2.0* (*?l1.0 x1*) *x2'*)) *?l2.0*; ((*x1 x2* :: *?L1.0*) ⇛$_m$ (*x1' x2'* :: *?R1.0*) ⇛ *Galois-infix x2 ?L1.0 ?R1.0 ?r1.0 x1'* ⟶ (*in-field* (*?R2.0* (*?l1.0 x1*) *x2'*) ⇛ *?L2.0 x1* (*?r1.0 x2'*))) *?r2.0*; ⋀*x1 x2*. *?L1.0 x1 x2* ⟹ *transitive* (*?L2.0 x1 x2*); ⋀*x1' x2'*. *?R1.0 x1' x2'* ⟹ *transitive* (*?R2.0 x1' x2'*)⟧ ⟹ *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: ⟦*galois.galois-connection ?L1.0 ?R1.0 ?l1.0 ?r1.0*; *reflexive-on* (*in-field ?L1.0*) *?L1.0*; ⋀*x x'*. *Galois-infix x ?L1.0 ?R1.0 ?r1.0 x'* ⟹ (*?R2.0* (*?l1.0 x*) *x'* ⇛$_m$ *?L2.0 x* (*?r1.0 x'*)) (*?r2.0 x x'*); ((*x1* : ⊤) ⇛$_m$ (*x2* - :: *?L1.0*) ⇛$_m$ *?L1.0 x1 x2* ⟶ (≤)) *?L2.0*; ((*x1* : ⊤) ⇛$_m$ (*x2 x3* :: *?L1.0*) ⇛$_m$ (*?L1.0 x1 x2* ∧ *?L1.0 x3* (*order-functors.unit ?l1.0 ?r1.0 x2*)) ⟶ (λ*x y*. *y* ≤ *x*)) *?L2.0*; ((*x1 x2* :: *?L1.0*) ⇛$_m$ (*x1' x2'* :: *?R1.0*) ⇛ *Galois-infix x2 ?L1.0 ?R1.0 ?r1.0 x1'* ⟶ (*in-field* (*?R2.0* (*?l1.0 x1*) *x2'*) ⇛ *?L2.0 x1* (*?r1.0 x2'*))) *?r2.0*; ⋀*x1 x2*. *?L1.0 x1 x2* ⟹ *transitive*

(*?L2.0 x1 x2*); *in-dom* (*transport-Mono-Dep-Fun-Rel.L ?L1.0 ?L2.0*) *?f*; *in-codom* (*transport-Mono-Dep-Fun-Rel.L ?R1.0 ?R2.0*) *?g*⟧ ⟹ *Galois-infix ?f* (*transport-Mono-Dep-Fun-Rel.L ?L1.0 ?L2.0*) (*transport-Mono-Dep-Fun-Rel.L ?R1.0 ?R2.0*) (*transport-Dep-Fun-Rel.l ?l1.0 ?r2.0*) *?g* = ((*x x'* :: *galois-rel.Galois ?L1.0 ?R1.0 ?r1.0*) ⇛ *galois-rel.Galois* (*?L2.0 x* (*?r1.0 x'*)) (*?R2.0* (*?l1.0 x*) *x'*) (*?r2.0 x x'*)) *?f ?g*

- Lemma 8 ⟦*galois.galois-equivalence ?L1.0 ?R1.0 ?l1.0 ?r1.0*; *Preorders.preorder-on* (*in-field ?L1.0*) *?L1.0*; ((*x1 x2* :: *?L1.0*$^{-1}$) ⇛$_m$ (*x3 x4* :: *?L1.0*) ⇛ *?L1.0 x1 x3* ⟶ (≤)) *?L2.0*; ⋀*x1 x2*. *?L1.0 x1 x2* ⟹ *partial-equivalence-rel* (*?L2.0 x1 x2*)⟧ ⟹ *transport-Mono-Dep-Fun-Rel.L ?L1.0 ?L2.0* = *transport-Dep-Fun-Rel.L ?L1.0 ?L2.0*
- Lemma 9: ⟦*reflexive-on* (*in-field ?L1.0*) *?L1.0*; *partial-equivalence-rel ?L2.0*⟧ ⟹ *transport-Mono-Dep-Fun-Rel.L ?L1.0* (λ- -. *?L2.0*) = *transport-Dep-Fun-Rel.L ?L1.0* (λ- -. *?L2.0*)

- Section 4.2: Closure Natural Functors can be found in `Natural_Functors`.
  - Theorem 3: see *Transport.Transport-Natural-Functors*
  - Theorem 4: *galois-rel.Galois* (*transport-natural-functor.L ?L1.0 ?L2.0 ?L3.0*) (*transport-natural-functor.L ?R1.0 ?R2.0 ?R3.0*) (*transport-natural-functor.l ?r1.0 ?r2.0 ?r3.0*) = *Frel* (*galois-rel.Galois ?L1.0 ?R1.0 ?r1.0*) (*galois-rel.Galois ?L2.0 ?R2.0 ?r2.0*) (*galois-rel.Galois ?L3.0 ?R3.0 ?r3.0*)

- Section 4.3: Closure Compositions can be found in `Compositions`.
  - Setup for simple case in *Transport.Transport-Compositions-Agree-Base*
  - Setup for generic case in *Transport.Transport-Compositions-Generic-Base*
  - Theorem 5: ⟦*transport.preorder-equivalence ?L1.0 ?R1.0 ?l1.0 ?r1.0*; *transport.preorder-equivalence ?L2.0 ?R2.0 ?l2.0 ?r2.0*; *transport-comp.middle-compatible-codom ?R1.0 ?L2.0*⟧ ⟹ *transport.preorder-equivalence* (*transport-comp.L ?L1.0 ?R1.0 ?l1.0 ?r1.0 ?L2.0*) (*transport-comp.L ?R2.0 ?L2.0 ?r2.0 ?l2.0 ?R1.0*) (*transport-comp.l ?l1.0 ?l2.0*) (*transport-comp.l ?r2.0 ?r1.0*) and ⟦*transport.partial-equivalence-rel-equivalence ?L1.0 ?R1.0 ?l1.0 ?r1.0*; *transport.partial-equivalence-rel-equivalence ?L2.0 ?R2.0 ?l2.0 ?r2.0*; *transport-comp.middle-compatible-codom ?R1.0 ?L2.0*⟧ ⟹ *transport.partial-equivalence-rel-equivalence* (*transport-comp.L ?L1.0 ?R1.0 ?l1.0 ?r1.0 ?L2.0*) (*transport-comp.L ?R2.0 ?L2.0 ?r2.0 ?l2.0 ?R1.0*) (*transport-comp.l ?l1.0 ?l2.0*) (*transport-comp.l ?r2.0 ?r1.0*)

- Theorem 6: ⟦*transport.preorder-equivalence ?L1.0 ?R1.0 ?l1.0 ?r1.0*; *transport.preorder-galois-connection ?R2.0 ?L2.0 ?r2.0 ?l2.0*; *transport-comp.middle-compatible-codom ?R1.0 ?L2.0*⟧ ⟹ *galois-rel.Galois* (*transport-comp.L ?L1.0 ?R1.0 ?l1.0 ?r1.0 ?L2.0*) (*transport-comp.L ?R2.0 ?L2.0 ?r2.0 ?l2.0 ?R1.0*) (*transport-comp.l ?r2.0 ?r1.0*) = *galois-rel.Galois ?L1.0 ?R1.0 ?r1.0* ∘∘ *galois-rel.Galois ?L2.0 ?R2.0 ?r2.0*

  (*results from Appendix*)
- Theorem 9: see `Compositions/Agree`, results in *transport-comp-same*.
- Theorem 10: ⟦*galois.galois-equivalence ?L1.0 ?R1.0 ?l1.0 ?r1.0*; *Preorders.preorder-on* (*in-field ?R1.0*) *?R1.0*; *galois.galois-equivalence ?L2.0 ?R2.0 ?l2.0 ?r2.0*; *Preorders.preorder-on* (*in-field ?L2.0*) *?L2.0*; *transport-comp.middle-compatible-codom ?R1.0 ?L2.0*⟧ ⟹ *galois.galois-connection* (*transport-comp.L ?L1.0 ?R1.0 ?l1.0 ?r1.0 ?L2.0*) (*transport-comp.L ?R2.0 ?L2.0 ?r2.0 ?l2.0 ?R1.0*) (*transport-comp.l ?l1.0 ?l2.0*) (*transport-comp.l ?r2.0 ?r1.0*)
- Theorem 11: ⟦(*?R1.0* ⇛$_m$ *?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* ⇛$_m$ *?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* ∘∘ *?L2.0* ∘∘ *?R1.0* ≤ *?R1.0* ∘∘ *?L2.0*; *in-codom* (*?L2.0* ∘∘ *?R1.0* ∘∘ *?L2.0*) ≤ *in-codom ?R1.0*⟧ ⟹ *galois-rel.Galois* (*transport-comp.L ?L1.0 ?R1.0 ?l1.0 ?r1.0 ?L2.0*) (*transport-comp.L ?R2.0 ?L2.0 ?r2.0 ?l2.0 ?R1.0*) (*transport-comp.l ?r2.0 ?r1.0*) = *galois-rel.Galois ?L1.0 ?R1.0 ?r1.0* ∘∘ *galois-rel.Galois ?L2.0 ?R2.0 ?r2.0*

- Section 5:
  - Implementation *Transport.Transport-Prototype*: Note: the command "trp" from the paper is called **trp-term** and the method "trprover" is called "trp_term_prover".
  - Example 1: *Transport.Transport-Lists-Sets-Examples*
  - Example 2: *Transport.Transport-Dep-Fun-Rel-Examples*
  - Example 3: https://github.com/kappelmann/Isabelle-Set/blob/fdf59444d9a53b5279080fb4d24893c9efa31160/Isabelle_Set/Integers/Integers_Transport.thy
- Proof: Partial Quotient Types are a special case: *Transport.Transport-Partial-Quotient-Types*
- Proof: Typedefs are a special case: *Transport.Transport-Typedef-Base*

- Proof: Set-Extensions are a special case: https://github.com/kappelmann/Isabelle-Set/blob/fdf59444d9a53b5279080fb4d24893c9efa31160/Isabelle_Set/Set_Extensions/Set_Extensions_Transport.thy
- Proof: Bijections as special case: *Transport.Transport-Bijections*

end

Bibliography

[1] Huffman, Brian and Kunčar, Ondřej. Lifting and Transfer: A modular design for quotients in Isabelle/HOL. In *Certified Programs and Proofs: Third International Conference, CPP 2013, Melbourne, VIC, Australia, December 11-13, 2013, Proceedings 3*, pages 131–146. Springer, 2013.

[2] K. Kappelmann. Transport via partial galois connections and equivalences. In C.-K. Hur, editor, *Programming Languages and Systems*, pages 225–245, Singapore, 2023. Springer Nature Singapore.