UTP Designs

Simon Foster Yakoub Nemouchi Frank Zeyda

April 4, 2018

Contents

1 Design Signature and Core Laws

theory *utp-des-core*
imports *UTP.utp*
begin

In UTP, in order to explicitly record the termination of a program, a subset of alphabetised relations is introduced. These relations are called designs, and their alphabet contains the special boolean observational variable *ok*. It is used to record the start and termination of a program. For more information on designs please see Chapter 3 of the UTP book [3], or the more accessible designs tutorial [2].

1.1 Definitions

Two named theorem sets exist are created to group theorems that, respectively, provide pre-postcondition definitions, and simplify operators to their normal design form.

named-theorems *ndes* **and** *ndes-simp*

alphabet *des-vars* =
  ok :: *bool*

declare *des-vars.defs* [*lens-defs*]

The two locale interpretations below are a technicality to improve automatic proof support via the predicate and relational tactics. This is to enable the (re-)interpretation of state spaces to remove any occurrences of lens types after the proof tactics *pred-simp* and *rel-simp*, or any of their derivatives have been applied. Eventually, it would be desirable to automate both interpretations as part of a custom outer command for defining alphabets.

interpretation *des-vars*: *lens-interp* $\lambda r.\ (ok_v\ r,\ more\ r)$
apply (*unfold-locales*)
apply (*rule injI*)
apply (*clarsimp*)
done

interpretation *des-vars-rel*:
  lens-interp $\lambda(r,\ r').\ (ok_v\ r,\ ok_v\ r',\ more\ r,\ more\ r')$
apply (*unfold-locales*)
apply (*rule injI*)
apply (*clarsimp*)
done

lemma *ok-ord* [*usubst*]:
  $\$ok \prec_v \$ok'$
  by (*simp add*: *var-name-ord-def*)

type-synonym $'\alpha$ *des* = $'\alpha$ *des-vars-scheme*
type-synonym $('\alpha,\ '\beta)$ *rel-des* = $('\alpha$ *des*, $'\beta$ *des*) *urel*
type-synonym $'\alpha$ *hrel-des* = $('\alpha$ *des*) *hrel*

translations
  (*type*) $'\alpha$ *des* <= (*type*) $'\alpha$ *des-vars-scheme*

($type$) $'\alpha$ des <= ($type$) $'\alpha$ $des\text{-}vars\text{-}ext$
($type$) ($'\alpha$, $'\beta$) $rel\text{-}des$ <= ($type$) ($'\alpha$ des, $'\beta$ des) $urel$
($type$) $'\alpha$ $hrel\text{-}des$ <= ($type$) $'\alpha$ des $hrel$

notation $des\text{-}vars\text{-}child\text{-}lens$ (Σ_D)

lemma $ok\text{-}des\text{-}bij\text{-}lens$: $bij\text{-}lens$ $(ok +_L \Sigma_D)$
by ($unfold\text{-}locales$, $simp\text{-}all$ add: $ok\text{-}def$ $des\text{-}vars\text{-}child\text{-}lens\text{-}def$ $lens\text{-}plus\text{-}def$ $prod.case\text{-}eq\text{-}if$)

Define the lens functor for designs

definition $lmap\text{-}des\text{-}vars$ :: $('\alpha \Longrightarrow '\beta) \Rightarrow ('\alpha\ des\text{-}vars\text{-}scheme \Longrightarrow '\beta\ des\text{-}vars\text{-}scheme)$ ($lmap_D$)
where [$lens\text{-}defs$]: $lmap\text{-}des\text{-}vars = lmap[des\text{-}vars]$

lemma $lmap\text{-}des\text{-}vars$: $vwb\text{-}lens\ f \Longrightarrow vwb\text{-}lens\ (lmap\text{-}des\text{-}vars\ f)$
by ($unfold\text{-}locales$, $auto$ $simp$ add: $lens\text{-}defs$)

lemma $lmap\text{-}id$: $lmap_D\ 1_L = 1_L$
by ($simp$ add: $lens\text{-}defs$ $fun\text{-}eq\text{-}iff$)

lemma $lmap\text{-}comp$: $lmap_D\ (f\ ;_L\ g) = lmap_D\ f\ ;_L\ lmap_D\ g$
by ($simp$ add: $lens\text{-}defs$ $fun\text{-}eq\text{-}iff$)

The following notations define liftings from non-design predicates into design predicates using alphabet extensions.

abbreviation $lift\text{-}desr$ ($\lceil\text{-}\rceil_D$)
where $\lceil P\rceil_D \equiv P \oplus_p (\Sigma_D \times_L \Sigma_D)$

abbreviation $lift\text{-}pre\text{-}desr$ ($\lceil\text{-}\rceil_{D<}$)
where $\lceil p\rceil_{D<} \equiv \lceil\lceil p\rceil_<\rceil_D$

abbreviation $lift\text{-}post\text{-}desr$ ($\lceil\text{-}\rceil_{D>}$)
where $\lceil p\rceil_{D>} \equiv \lceil\lceil p\rceil_>\rceil_D$

abbreviation $drop\text{-}desr$ ($\lfloor\text{-}\rfloor_D$)
where $\lfloor P\rfloor_D \equiv P \upharpoonright_e (\Sigma_D \times_L \Sigma_D)$

abbreviation $dcond$ :: $('\alpha, '\beta)\ rel\text{-}des \Rightarrow '\alpha\ upred \Rightarrow ('\alpha, '\beta)\ rel\text{-}des \Rightarrow ('\alpha, '\beta)\ rel\text{-}des$
((3- $\triangleleft$ - $\triangleright_D$/ -) [52,0,53] 52)
where $P \triangleleft b \triangleright_D Q \equiv P \triangleleft \lceil b\rceil_{D<} \triangleright Q$

definition $design$::$('\alpha, '\beta)\ rel\text{-}des \Rightarrow ('\alpha, '\beta)\ rel\text{-}des \Rightarrow ('\alpha, '\beta)\ rel\text{-}des$ (**infixl** $\vdash$ 60) **where**
[$upred\text{-}defs$]: $P \vdash Q = (\$ok \wedge P \Rightarrow \$ok' \wedge Q)$

An rdesign is a design that uses the Isabelle type system to prevent reference to ok in the assumption and commitment.

definition $rdesign$::$('\alpha, '\beta)\ urel \Rightarrow ('\alpha, '\beta)\ urel \Rightarrow ('\alpha, '\beta)\ rel\text{-}des$ (**infixl** $\vdash_r$ 60) **where**
[$upred\text{-}defs$]: $(P \vdash_r Q) = \lceil P\rceil_D \vdash \lceil Q\rceil_D$

An ndesign is a normal design, i.e. where the assumption is a condition

definition $ndesign$::$'\alpha\ cond \Rightarrow ('\alpha, '\beta)\ urel \Rightarrow ('\alpha, '\beta)\ rel\text{-}des$ (**infixl** $\vdash_n$ 60) **where**
[$upred\text{-}defs$]: $(p \vdash_n Q) = (\lceil p\rceil_< \vdash_r Q)$

definition $skip\text{-}d$ :: $'\alpha\ hrel\text{-}des$ (II_D) **where**
[$upred\text{-}defs$]: $II_D \equiv (true \vdash_r II)$

definition *bot-d* :: $('\alpha, '\beta)$ *rel-des* $(\bot_D)$ **where**
[*upred-defs*]: $\bot_D = (false \vdash false)$

definition *pre-design* :: $('\alpha, '\beta)$ *rel-des* $\Rightarrow$ $('\alpha, '\beta)$ *urel* (pre_D) **where**
[*upred-defs*]: $pre_D(P) = \lfloor \neg\ P[\![true,false/\$ok,\$ok']\!] \rfloor_D$

definition *post-design* :: $('\alpha, '\beta)$ *rel-des* $\Rightarrow$ $('\alpha, '\beta)$ *urel* $(post_D)$ **where**
[*upred-defs*]: $post_D(P) = \lfloor P[\![true,true/\$ok,\$ok']\!] \rfloor_D$

syntax
  -ok-f :: *logic* $\Rightarrow$ *logic* (-^f [*1000*] *1000*)
  -ok-t :: *logic* $\Rightarrow$ *logic* (-^t [*1000*] *1000*)
  -top-d :: *logic* $(\top_D)$

translations
  P^f $\rightleftharpoons$ *CONST usubst* (*CONST subst-upd CONST id* (*CONST ovar CONST ok*) *false*) *P*
  P^t $\rightleftharpoons$ *CONST usubst* (*CONST subst-upd CONST id* (*CONST ovar CONST ok*) *true*) *P*
  $\top_D$ => *CONST not-upred* (*CONST utp-expr.var* (*CONST ivar CONST ok*))

1.2 Lifting, Unrestriction, and Substitution

lemma *drop-desr-inv* [*simp*]: $\lfloor \lceil P \rceil_D \rfloor_D = P$
  by (*simp add*: *prod-mwb-lens*)

lemma *lift-desr-inv*:
  fixes P :: $('\alpha, '\beta)$ *rel-des*
  assumes $\$ok \sharp P$ $\$ok' \sharp P$
  shows $\lceil \lfloor P \rfloor_D \rceil_D = P$
proof −
  have *bij-lens* $(\Sigma_D \times_L \Sigma_D +_L (in\text{-}var\ ok +_L out\text{-}var\ ok)$ :: (-, $'\alpha$ *des-vars-scheme* $\times$ $'\beta$ *des-vars-scheme*) *lens*)
    (**is** *bij-lens* (*?P*))
  proof −
    have $?P \approx_L (ok +_L \Sigma_D) \times_L (ok +_L \Sigma_D)$ (**is** $?P \approx_L ?Q$)
      apply (*simp add*: *in-var-def out-var-def prod-as-plus*)
      apply (*simp add*: *prod-as-plus*[*THEN sym*])
     apply (*meson lens-equiv-sym lens-equiv-trans lens-indep-prod lens-plus-comm lens-plus-prod-exchange des-vars-indeps*(*1*))
    done
    moreover have *bij-lens ?Q*
      by (*simp add*: *ok-des-bij-lens prod-bij-lens*)
    ultimately show *?thesis*
      by (*metis bij-lens-equiv lens-equiv-sym*)
  qed

  with *assms* **show** *?thesis*
    apply (*rule-tac aext-arestr*[*of - in-var ok* $+_L$ *out-var ok*])
    apply (*simp add*: *prod-mwb-lens*)
    apply (*simp*)
   apply (*metis alpha-in-var lens-indep-prod lens-indep-sym des-vars-indeps*(*1*) *out-var-def prod-as-plus*)
    using *unrest-var-comp* **apply** *blast*
  done
qed

lemma *unrest-out-des-lift* [*unrest*]: $out\alpha \sharp p \Longrightarrow out\alpha \sharp \lceil p \rceil_D$

by $(pred\text{-}simp)$

lemma $lift\text{-}dist\text{-}seq$ $[simp]$:
$\lceil P \ ;; \ Q\rceil_D = (\lceil P\rceil_D \ ;; \ \lceil Q\rceil_D)$
by $(rel\text{-}auto)$

lemma $lift\text{-}des\text{-}skip\text{-}dr\text{-}unit$ $[simp]$:
$(\lceil P\rceil_D \ ;; \ \lceil II\rceil_D) = \lceil P\rceil_D$
$(\lceil II\rceil_D \ ;; \ \lceil P\rceil_D) = \lceil P\rceil_D$
by $(rel\text{-}auto)+$

lemma $lift\text{-}des\text{-}skip\text{-}dr\text{-}unit\text{-}unrest$: $\$ok\acute{} \ \sharp \ P \Longrightarrow (P \ ;; \ \lceil II\rceil_D) = P$
by $(rel\text{-}auto)$

lemma $state\text{-}subst\text{-}design$ $[usubst]$:
$\lceil \sigma \oplus_s \Sigma_D\rceil_s \dagger (P \vdash_r Q) = (\lceil \sigma\rceil_s \dagger P) \vdash_r (\lceil \sigma\rceil_s \dagger Q)$
by $(rel\text{-}auto)$

lemma $design\text{-}subst$ $[usubst]$:
$[\![\ \$ok \ \sharp \ \sigma; \ \$ok\acute{} \ \sharp \ \sigma \ ]\!] \Longrightarrow \sigma \dagger (P \vdash Q) = (\sigma \dagger P) \vdash (\sigma \dagger Q)$
by $(simp \ add\text{:} \ design\text{-}def \ usubst)$

lemma $design\text{-}msubst$ $[usubst]$:
$(P(x) \vdash Q(x))[\![x \rightarrow v]\!] = (P(x)[\![x \rightarrow v]\!] \vdash Q(x)[\![x \rightarrow v]\!])$
by $(rel\text{-}auto)$

lemma $design\text{-}ok\text{-}false$ $[usubst]$: $(P \vdash Q)[\![false/\$ok]\!] = true$
by $(simp \ add\text{:} \ design\text{-}def \ usubst)$

lemma $ok\text{-}pre$: $(\$ok \land \lceil pre_D(P)\rceil_D) = (\$ok \land (\neg \ P^f))$
by $(pred\text{-}auto \ robust)$

lemma $ok\text{-}post$: $(\$ok \land \lceil post_D(P)\rceil_D) = (\$ok \land (P^t))$
by $(pred\text{-}auto \ robust)$

1.3 Basic Design Laws

lemma $design\text{-}export\text{-}ok$: $P \vdash Q = (P \vdash (\$ok \land Q))$
by $(rel\text{-}auto)$

lemma $design\text{-}export\text{-}ok'$: $P \vdash Q = (P \vdash (\$ok\acute{} \land Q))$
by $(rel\text{-}auto)$

lemma $design\text{-}export\text{-}pre$: $P \vdash (P \land Q) = P \vdash Q$
by $(rel\text{-}auto)$

lemma $design\text{-}export\text{-}spec$: $P \vdash (P \Rightarrow Q) = P \vdash Q$
by $(rel\text{-}auto)$

lemma $design\text{-}ok\text{-}pre\text{-}conj$: $(\$ok \land P) \vdash Q = P \vdash Q$
by $(rel\text{-}auto)$

lemma $true\text{-}is\text{-}design$: $(false \vdash true) = true$
by $(rel\text{-}auto)$

lemma $true\text{-}is\text{-}rdesign$: $(false \vdash_r true) = true$

by (*rel-auto*)

lemma *bot-d-true*: $\bot_D = true$
by (*rel-auto*)

lemma *bot-d-ndes-def* [*ndes-simp*]: $\bot_D = (false \vdash_n true)$
by (*rel-auto*)

lemma *design-false-pre*: $(false \vdash P) = true$
by (*rel-auto*)

lemma *rdesign-false-pre*: $(false \vdash_r P) = true$
by (*rel-auto*)

lemma *ndesign-false-pre*: $(false \vdash_n P) = true$
by (*rel-auto*)

lemma *ndesign-miracle*: $(true \vdash_n false) = \top_D$
by (*rel-auto*)

lemma *top-d-ndes-def* [*ndes-simp*]: $\top_D = (true \vdash_n false)$
by (*rel-auto*)

lemma *skip-d-alt-def*: $II_D = true \vdash II$
by (*rel-auto*)

lemma *skip-d-ndes-def* [*ndes-simp*]: $II_D = true \vdash_n II$
by (*rel-auto*)

lemma *design-subst-ok*:
$(P[\![true/\$ok]\!] \vdash Q[\![true/\$ok]\!]) = (P \vdash Q)$
by (*rel-auto*)

lemma *design-subst-ok-ok'*:
$(P[\![true/\$ok]\!] \vdash Q[\![true,true/\$ok,\$ok']\!]) = (P \vdash Q)$
proof −
have $(P \vdash Q) = ((\$ok \land P) \vdash (\$ok \land \$ok' \land Q))$
by (*pred-auto*)
also have ... $= ((\$ok \land P[\![true/\$ok]\!]) \vdash (\$ok \land (\$ok' \land Q[\![true/\$ok']\!])[\![true/\$ok]\!]))$
by (*metis conj-eq-out-var-subst conj-pos-var-subst upred-eq-true utp-pred-laws.inf-commute ok-vwb-lens*)
also have ... $= ((\$ok \land P[\![true/\$ok]\!]) \vdash (\$ok \land \$ok' \land Q[\![true,true/\$ok,\$ok']\!]))$
by (*simp add*: *usubst*)
also have ... $= (P[\![true/\$ok]\!] \vdash Q[\![true,true/\$ok,\$ok']\!])$
by (*pred-auto*)
finally show *?thesis* **..**
qed

lemma *design-subst-ok'*:
$(P \vdash Q[\![true/\$ok']\!]) = (P \vdash Q)$
proof −
have $(P \vdash Q) = (P \vdash (\$ok' \land Q))$
by (*pred-auto*)
also have ... $= (P \vdash (\$ok' \land Q[\![true/\$ok']\!]))$
by (*metis conj-eq-out-var-subst upred-eq-true utp-pred-laws.inf-commute ok-vwb-lens*)
also have ... $= (P \vdash Q[\![true/\$ok']\!])$

by (*pred-auto*)
finally show *?thesis* **..**
qed

1.4 Sequential Composition Laws

theorem *design-skip-idem* [*simp*]:
$(II_D ;; II_D) = II_D$
by (*rel-auto*)

theorem *design-composition-subst*:
assumes
$\$ok\acute{} \sharp P1$ $\$ok \sharp P2$
shows $((P1 \vdash Q1) ;; (P2 \vdash Q2)) =$
$(((\neg ((\neg P1) ;; true)) \wedge \neg (Q1\llbracket true/\$ok\acute{}\rrbracket ;; (\neg P2))) \vdash (Q1\llbracket true/\$ok\acute{}\rrbracket ;; Q2\llbracket true/\$ok\rrbracket))$
proof −
have $((P1 \vdash Q1) ;; (P2 \vdash Q2)) = (\exists\ ok_0 \cdot ((P1 \vdash Q1)\llbracket \ll ok_0 \gg/\$ok\acute{}\rrbracket ;; (P2 \vdash Q2)\llbracket \ll ok_0 \gg/\$ok\rrbracket))$
by (*rule seqr-middle*, *simp*)
also have ...
$= (((P1 \vdash Q1)\llbracket false/\$ok\acute{}\rrbracket ;; (P2 \vdash Q2)\llbracket false/\$ok\rrbracket)$
$\vee ((P1 \vdash Q1)\llbracket true/\$ok\acute{}\rrbracket ;; (P2 \vdash Q2)\llbracket true/\$ok\rrbracket))$
by (*simp add*: *true-alt-def false-alt-def*, *pred-auto*)
also from *assms*
have ... $= (((\$ok \wedge P1 \Rightarrow Q1\llbracket true/\$ok\acute{}\rrbracket) ;; (P2 \Rightarrow \$ok\acute{} \wedge Q2\llbracket true/\$ok\rrbracket)) \vee ((\neg (\$ok \wedge P1)) ;; true))$
by (*simp add*: *design-def usubst unrest*, *pred-auto*)
also have ... $= ((\neg\$ok ;; true_h) \vee ((\neg P1) ;; true) \vee (Q1\llbracket true/\$ok\acute{}\rrbracket ;; (\neg P2)) \vee (\$ok\acute{} \wedge (Q1\llbracket true/\$ok\acute{}\rrbracket ;; Q2\llbracket true/\$ok\rrbracket)))$
by (*rel-auto*)
also have ... $= (((\neg ((\neg P1) ;; true)) \wedge \neg (Q1\llbracket true/\$ok\acute{}\rrbracket ;; (\neg P2))) \vdash (Q1\llbracket true/\$ok\acute{}\rrbracket ;; Q2\llbracket true/\$ok\rrbracket))$
by (*simp add*: *precond-right-unit design-def unrest*, *rel-auto*)
finally show *?thesis* **.**
qed

theorem *design-composition*:
assumes
$\$ok\acute{} \sharp P1$ $\$ok \sharp P2$ $\$ok\acute{} \sharp Q1$ $\$ok \sharp Q2$
shows $((P1 \vdash Q1) ;; (P2 \vdash Q2)) = (((\neg ((\neg P1) ;; true)) \wedge \neg (Q1 ;; (\neg P2))) \vdash (Q1 ;; Q2))$
using *assms* **by** (*simp add*: *design-composition-subst usubst*)

theorem *design-composition-runrest*:
assumes
$\$ok\acute{} \sharp P1$ $\$ok \sharp P2$ $ok \sharp\sharp Q1$ $ok \sharp\sharp Q2$
shows $((P1 \vdash Q1) ;; (P2 \vdash Q2)) = (((\neg ((\neg P1) ;; true)) \wedge \neg (Q1^t ;; (\neg P2))) \vdash (Q1 ;; Q2))$
proof −
have $(\$ok \wedge \$ok\acute{} \wedge (Q1^t ;; Q2\llbracket true/\$ok\rrbracket)) = (\$ok \wedge \$ok\acute{} \wedge (Q1 ;; Q2))$
proof −
have $(\$ok \wedge \$ok\acute{} \wedge (Q1 ;; Q2)) = ((\$ok \wedge Q1) ;; (Q2 \wedge \$ok\acute{}))$
by (*metis* (*no-types*, *lifting*) *conj-comm seqr-post-var-out seqr-pre-var-out*)
also have ... $= ((Q1 \wedge \$ok\acute{}) ;; (\$ok \wedge Q2))$
by (*simp add*: *assms*(*3*) *assms*(*4*) *runrest-ident-var*)
also have ... $= (Q1^t ;; Q2\llbracket true/\$ok\rrbracket)$
by (*metis ok-vwb-lens seqr-pre-transfer seqr-right-one-point true-alt-def uovar-convr upred-eq-true utp-pred-laws.inf.left-idem utp-rel.unrest-ouvar vwb-lens-mwb*)
finally show *?thesis*
by (*metis utp-pred-laws.inf.left-commute utp-pred-laws.inf-left-idem*)

qed
moreover have $(\neg (\neg P1 ;; true) \wedge \neg (Q1^t ;; (\neg P2))) \vdash (Q1^t ;; Q2[\![true/\$ok]\!]) =$
$(\neg (\neg P1 ;; true) \wedge \neg (Q1^t ;; (\neg P2))) \vdash (\$ok \wedge \$ok' \wedge (Q1^t ;; Q2[\![true/\$ok]\!]))$
by (*metis design-export-ok design-export-ok'*)
ultimately show *?thesis* **using** *assms*
by (*simp add*: *design-composition-subst usubst*, *metis design-export-ok design-export-ok'*)
qed

theorem *rdesign-composition*:
$((P1 \vdash_r Q1) ;; (P2 \vdash_r Q2)) = (((\neg ((\neg P1) ;; true)) \wedge \neg (Q1 ;; (\neg P2))) \vdash_r (Q1 ;; Q2))$
by (*simp add*: *rdesign-def design-composition unrest alpha*)

theorem *design-composition-cond*:
assumes
$out\alpha \sharp p1$ $\$ok \sharp P2$ $\$ok' \sharp Q1$ $\$ok \sharp Q2$
shows $((p1 \vdash Q1) ;; (P2 \vdash Q2)) = ((p1 \wedge \neg (Q1 ;; (\neg P2))) \vdash (Q1 ;; Q2))$
using *assms*
by (*simp add*: *design-composition unrest precond-right-unit*)

theorem *rdesign-composition-cond*:
assumes $out\alpha \sharp p1$
shows $((p1 \vdash_r Q1) ;; (P2 \vdash_r Q2)) = ((p1 \wedge \neg (Q1 ;; (\neg P2))) \vdash_r (Q1 ;; Q2))$
using *assms*
by (*simp add*: *rdesign-def design-composition-cond unrest alpha*)

theorem *design-composition-wp*:
assumes
$ok \sharp p1$ $ok \sharp p2$
$\$ok \sharp Q1$ $\$ok' \sharp Q1$ $\$ok \sharp Q2$ $\$ok' \sharp Q2$
shows $((\lceil p1 \rceil_< \vdash Q1) ;; (\lceil p2 \rceil_< \vdash Q2)) = ((\lceil p1 \wedge Q1\ wp\ p2 \rceil_<) \vdash (Q1 ;; Q2))$
using *assms* **by** (*rel-blast*)

theorem *rdesign-composition-wp*:
$((\lceil p1 \rceil_< \vdash_r Q1) ;; (\lceil p2 \rceil_< \vdash_r Q2)) = ((\lceil p1 \wedge Q1\ wp\ p2 \rceil_<) \vdash_r (Q1 ;; Q2))$
by (*rel-blast*)

theorem *ndesign-composition-wp* [*ndes-simp*]:
$((p1 \vdash_n Q1) ;; (p2 \vdash_n Q2)) = ((p1 \wedge Q1\ wp\ p2) \vdash_n (Q1 ;; Q2))$
by (*rel-blast*)

theorem *design-true-left-zero*: $(true ;; (P \vdash Q)) = true$
proof −
have $(true ;; (P \vdash Q)) = (\exists\ ok_0 \cdot true[\![\ll ok_0 \gg/\$ok']\!] ;; (P \vdash Q)[\![\ll ok_0 \gg/\$ok]\!])$
by (*subst seqr-middle*[*of ok*], *simp-all*)
also have ... $= ((true[\![false/\$ok']\!] ;; (P \vdash Q)[\![false/\$ok]\!]) \vee (true[\![true/\$ok']\!] ;; (P \vdash Q)[\![true/\$ok]\!]))$
by (*simp add*: *disj-comm false-alt-def true-alt-def*)
also have ... $= ((true[\![false/\$ok']\!] ;; true_h) \vee (true ;; ((P \vdash Q)[\![true/\$ok]\!])))$
by (*subst-tac*, *rel-auto*)
also have ... $= true$
by (*subst-tac*, *simp add*: *precond-right-unit unrest*)
finally show *?thesis* **.**
qed

theorem *design-left-unit-hom*:
fixes $P\ Q :: '\alpha\ hrel\text{-}des$

shows $(II_D \;;\; (P \vdash_r Q)) = (P \vdash_r Q)$
proof $-$
  have $(II_D \;;\; (P \vdash_r Q)) = ((\mathit{true} \vdash_r II) \;;\; (P \vdash_r Q))$
    by (*simp add*: *skip-d-def*)
  also have ... $= (\mathit{true} \wedge \neg (II \;;\; (\neg P))) \vdash_r (II \;;\; Q)$
  proof $-$
    have $\mathit{out}\alpha \sharp \mathit{true}$
      by *unrest-tac*
    thus *?thesis*
      using *rdesign-composition-cond* **by** *blast*
  qed
  also have ... $= (\neg (\neg P)) \vdash_r Q$
    by *simp*
  finally show *?thesis* **by** *simp*
qed

theorem *rdesign-left-unit* [*simp*]:
  $II_D \;;\; (P \vdash_r Q) = (P \vdash_r Q)$
  by (*rel-auto*)

theorem *design-right-semi-unit*:
  $(P \vdash_r Q) \;;\; II_D = ((\neg (\neg P) \;;\; \mathit{true}) \vdash_r Q)$
  by (*simp add*: *skip-d-def rdesign-composition*)

theorem *design-right-cond-unit* [*simp*]:
  assumes $\mathit{out}\alpha \sharp p$
  shows $(p \vdash_r Q) \;;\; II_D = (p \vdash_r Q)$
  using *assms*
  by (*simp add*: *skip-d-def rdesign-composition-cond*)

theorem *ndesign-left-unit* [*simp*]:
  $II_D \;;\; (p \vdash_n Q) = (p \vdash_n Q)$
  by (*rel-auto*)

theorem *design-bot-left-zero*: $(\bot_D \;;\; (P \vdash Q)) = \bot_D$
  by (*rel-auto*)

theorem *design-top-left-zero*: $(\top_D \;;\; (P \vdash Q)) = \top_D$
  by (*rel-auto*)

1.5 Preconditions and Postconditions

theorem *design-npre*:
  $(P \vdash Q)^f = (\neg \$ok \vee \neg P^f)$
  by (*rel-auto*)

theorem *design-pre*:
  $\neg (P \vdash Q)^f = (\$ok \wedge P^f)$
  by (*simp add*: *design-def*, *subst-tac*)
    (*metis* (*no-types*, *hide-lams*) *not-conj-deMorgans true-not-false*(*2*) *utp-pred-laws.compl-top-eq*
      utp-pred-laws.sup.idem utp-pred-laws.sup-compl-top)

theorem *design-post*:
  $(P \vdash Q)^t = ((\$ok \wedge P^t) \Rightarrow Q^t)$
  by (*rel-auto*)

theorem *rdesign-pre* [*simp*]: $pre_D(P \vdash_r Q) = P$
  by (*pred-auto*)

theorem *rdesign-post* [*simp*]: $post_D(P \vdash_r Q) = (P \Rightarrow Q)$
  by (*pred-auto*)

theorem *ndesign-pre* [*simp*]: $pre_D(p \vdash_n Q) = \lceil p \rceil_<$
  by (*pred-auto*)

theorem *ndesign-post* [*simp*]: $post_D(p \vdash_n Q) = (\lceil p \rceil_< \Rightarrow Q)$
  by (*pred-auto*)

lemma *design-pre-choice* [*simp*]:
  $pre_D(P \sqcap Q) = (pre_D(P) \land pre_D(Q))$
  by (*rel-auto*)

lemma *design-post-choice* [*simp*]:
  $post_D(P \sqcap Q) = (post_D(P) \lor post_D(Q))$
  by (*rel-auto*)

lemma *design-pre-condr* [*simp*]:
  $pre_D(P \triangleleft \lceil b \rceil_D \triangleright Q) = (pre_D(P) \triangleleft b \triangleright pre_D(Q))$
  by (*rel-auto*)

lemma *design-post-condr* [*simp*]:
  $post_D(P \triangleleft \lceil b \rceil_D \triangleright Q) = (post_D(P) \triangleleft b \triangleright post_D(Q))$
  by (*rel-auto*)

lemma *preD-USUP-mem*: $pre_D\ (\bigsqcup\ i{\in}A \cdot P\ i) = (\bigsqcap\ i{\in}A \cdot pre_D(P\ i))$
  by (*rel-auto*)

lemma *preD-USUP-ind*: $pre_D\ (\bigsqcup\ i \cdot P\ i) = (\bigsqcap\ i \cdot pre_D(P\ i))$
  by (*rel-auto*)

1.6 Distribution Laws

theorem *design-choice*:
  $(P_1 \vdash P_2) \sqcap (Q_1 \vdash Q_2) = ((P_1 \land Q_1) \vdash (P_2 \lor Q_2))$
  by (*rel-auto*)

theorem *rdesign-choice*:
  $(P_1 \vdash_r P_2) \sqcap (Q_1 \vdash_r Q_2) = ((P_1 \land Q_1) \vdash_r (P_2 \lor Q_2))$
  by (*rel-auto*)

theorem *ndesign-choice* [*ndes-simp*]:
  $(p_1 \vdash_n P_2) \sqcap (q_1 \vdash_n Q_2) = ((p_1 \land q_1) \vdash_n (P_2 \lor Q_2))$
  by (*rel-auto*)

theorem *ndesign-choice'* [*ndes-simp*]:
  $((p_1 \vdash_n P_2) \lor (q_1 \vdash_n Q_2)) = ((p_1 \land q_1) \vdash_n (P_2 \lor Q_2))$
  by (*rel-auto*)

theorem *design-inf*:
  $(P_1 \vdash P_2) \sqcup (Q_1 \vdash Q_2) = ((P_1 \lor Q_1) \vdash ((P_1 \Rightarrow P_2) \land (Q_1 \Rightarrow Q_2)))$
  by (*rel-auto*)

theorem *rdesign-inf*:
$(P_1 \vdash_r P_2) \sqcup (Q_1 \vdash_r Q_2) = ((P_1 \vee Q_1) \vdash_r ((P_1 \Rightarrow P_2) \wedge (Q_1 \Rightarrow Q_2)))$
by (*rel-auto*)

theorem *ndesign-inf* [*ndes-simp*]:
$(p_1 \vdash_n P_2) \sqcup (q_1 \vdash_n Q_2) = ((p_1 \vee q_1) \vdash_n ((\lceil p_1 \rceil_< \Rightarrow P_2) \wedge (\lceil q_1 \rceil_< \Rightarrow Q_2)))$
by (*rel-auto*)

theorem *design-condr*:
$((P_1 \vdash P_2) \triangleleft b \triangleright (Q_1 \vdash Q_2)) = ((P_1 \triangleleft b \triangleright Q_1) \vdash (P_2 \triangleleft b \triangleright Q_2))$
by (*rel-auto*)

theorem *ndesign-dcond* [*ndes-simp*]:
$((p_1 \vdash_n P_2) \triangleleft b \triangleright_D (q_1 \vdash_n Q_2)) = ((p_1 \triangleleft b \triangleright q_1) \vdash_n (P_2 \triangleleft b \triangleright_r Q_2))$
by (*rel-auto*)

lemma *design-UINF-mem*:
assumes $A \neq \{\}$
shows $(\bigsqcap\ i \in A \cdot P(i) \vdash Q(i)) = (\bigsqcup\ i \in A \cdot P(i)) \vdash (\bigsqcap\ i \in A \cdot Q(i))$
using *assms* **by** (*rel-auto*)

lemma *ndesign-UINF-mem* [*ndes-simp*]:
assumes $A \neq \{\}$
shows $(\bigsqcap\ i \in A \cdot p(i) \vdash_n Q(i)) = (\bigsqcup\ i \in A \cdot p(i)) \vdash_n (\bigsqcap\ i \in A \cdot Q(i))$
using *assms* **by** (*rel-auto*)

lemma *ndesign-UINF-ind* [*ndes-simp*]:
$(\bigsqcap\ i \cdot p(i) \vdash_n Q(i)) = (\bigsqcup\ i \cdot p(i)) \vdash_n (\bigsqcap\ i \cdot Q(i))$
by (*rel-auto*)

lemma *design-USUP-mem*:
$(\bigsqcup\ i \in A \cdot P(i) \vdash Q(i)) = (\bigsqcap\ i \in A \cdot P(i)) \vdash (\bigsqcup\ i \in A \cdot P(i) \Rightarrow Q(i))$
by (*rel-auto*)

lemma *ndesign-USUP-mem* [*ndes-simp*]:
$(\bigsqcup\ i \in A \cdot p(i) \vdash_n Q(i)) = (\bigsqcap\ i \in A \cdot p(i)) \vdash_n (\bigsqcup\ i \in A \cdot \lceil p(i) \rceil_< \Rightarrow Q(i))$
by (*rel-auto*)

lemma *ndesign-USUP-ind* [*ndes-simp*]:
$(\bigsqcup\ i \cdot p(i) \vdash_n Q(i)) = (\bigsqcap\ i \cdot p(i)) \vdash_n (\bigsqcup\ i \cdot \lceil p(i) \rceil_< \Rightarrow Q(i))$
by (*rel-auto*)

1.7 Refinement Introduction

lemma *ndesign-eq-intro*:
assumes $p_1 = q_1$ $P_2 = Q_2$
shows $p_1 \vdash_n P_2 = q_1 \vdash_n Q_2$
by (*simp add*: *assms*)

theorem *design-refinement*:
assumes
$\$ok \sharp P1$ $\$ok' \sharp P1$ $\$ok \sharp P2$ $\$ok' \sharp P2$
$\$ok \sharp Q1$ $\$ok' \sharp Q1$ $\$ok \sharp Q2$ $\$ok' \sharp Q2$
shows $(P1 \vdash Q1 \sqsubseteq P2 \vdash Q2) \longleftrightarrow (`P1 \Rightarrow P2` \wedge `P1 \wedge Q2 \Rightarrow Q1`)$
proof −
have $(P1 \vdash Q1) \sqsubseteq (P2 \vdash Q2) \longleftrightarrow `(\$ok \wedge P2 \Rightarrow \$ok' \wedge Q2) \Rightarrow (\$ok \wedge P1 \Rightarrow \$ok' \wedge Q1)`$

by (*pred-auto*)
also with *assms* **have** ... = ‘$(P2 \Rightarrow \$ok' \wedge Q2) \Rightarrow (P1 \Rightarrow \$ok' \wedge Q1)$‘
by (*subst subst-bool-split*[*of in-var ok*], *simp-all*, *subst-tac*)
also with *assms* **have** ... = ‘$(\neg P2 \Rightarrow \neg P1) \wedge ((P2 \Rightarrow Q2) \Rightarrow P1 \Rightarrow Q1)$‘
by (*subst subst-bool-split*[*of out-var ok*], *simp-all*, *subst-tac*)
also have ... $\longleftrightarrow$ ‘$(P1 \Rightarrow P2)$‘ $\wedge$ ‘$P1 \wedge Q2 \Rightarrow Q1$‘
by (*pred-auto*)
finally show *?thesis* **.**
qed

theorem *rdesign-refinement*:
$(P1 \vdash_r Q1 \sqsubseteq P2 \vdash_r Q2) \longleftrightarrow$ (‘$P1 \Rightarrow P2$‘ $\wedge$ ‘$P1 \wedge Q2 \Rightarrow Q1$‘)
by (*rel-auto*)

lemma *design-refine-intro*:
assumes ‘$P1 \Rightarrow P2$‘ ‘$P1 \wedge Q2 \Rightarrow Q1$‘
shows $P1 \vdash Q1 \sqsubseteq P2 \vdash Q2$
using *assms* **unfolding** *upred-defs*
by (*pred-auto*)

lemma *design-refine-intro'*:
assumes $P_2 \sqsubseteq P_1$ $Q_1 \sqsubseteq (P_1 \wedge Q_2)$
shows $P_1 \vdash Q_1 \sqsubseteq P_2 \vdash Q_2$
using *assms design-refine-intro*[*of* P_1 P_2 Q_2 Q_1] **by** (*simp add*: *refBy-order*)

lemma *rdesign-refine-intro*:
assumes ‘$P1 \Rightarrow P2$‘ ‘$P1 \wedge Q2 \Rightarrow Q1$‘
shows $P1 \vdash_r Q1 \sqsubseteq P2 \vdash_r Q2$
using *assms* **unfolding** *upred-defs*
by (*pred-auto*)

lemma *rdesign-refine-intro'*:
assumes $P2 \sqsubseteq P1$ $Q1 \sqsubseteq (P1 \wedge Q2)$
shows $P1 \vdash_r Q1 \sqsubseteq P2 \vdash_r Q2$
using *assms* **unfolding** *upred-defs*
by (*pred-auto*)

lemma *ndesign-refine-intro*:
assumes ‘$p1 \Rightarrow p2$‘ ‘$\lceil p1 \rceil_< \wedge Q2 \Rightarrow Q1$‘
shows $p1 \vdash_n Q1 \sqsubseteq p2 \vdash_n Q2$
using *assms* **unfolding** *upred-defs*
by (*pred-auto*)

lemma *design-top*:
$(P \vdash Q) \sqsubseteq \top_D$
by (*rel-auto*)

lemma *design-bottom*:
$\bot_D \sqsubseteq (P \vdash Q)$
by (*rel-auto*)

lemma *design-refine-thms*:
assumes $P \sqsubseteq Q$
shows ‘$pre_D(P) \Rightarrow pre_D(Q)$‘ ‘$pre_D(P) \wedge post_D(Q) \Rightarrow post_D(P)$‘
apply (*metis assms design-pre-choice disj-comm disj-upred-def order-refl rdesign-refinement utp-pred-laws.le-iff-sup*)

apply (*metis assms conj-comm design-post-choice disj-upred-def refBy-order semilattice-sup-class.le-iff-sup utp-pred-laws.inf.coboundedI1*)
done

end

2 Design Healthiness Conditions

theory *utp-des-healths*
imports *utp-des-core*
begin

2.1 H1: No observation is allowed before initiation

definition $H1 :: ('\alpha, '\beta)\ rel\text{-}des \Rightarrow ('\alpha, '\beta)\ rel\text{-}des$ **where**
[*upred-defs*]: $H1(P) = (\$ok \Rightarrow P)$

lemma *H1-idem*:
$H1\ (H1\ P) = H1(P)$
by (*pred-auto*)

lemma *H1-monotone*:
$P \sqsubseteq Q \Longrightarrow H1(P) \sqsubseteq H1(Q)$
by (*pred-auto*)

lemma *H1-Continuous*: *Continuous H1*
by (*rel-auto*)

lemma *H1-below-top*:
$H1(P) \sqsubseteq \top_D$
by (*pred-auto*)

lemma *H1-design-skip*:
$H1(II) = II_D$
by (*rel-auto*)

lemma *H1-cond*: $H1(P \triangleleft b \triangleright Q) = H1(P) \triangleleft b \triangleright H1(Q)$
by (*rel-auto*)

lemma *H1-conj*: $H1(P \wedge Q) = (H1(P) \wedge H1(Q))$
by (*rel-auto*)

lemma *H1-disj*: $H1(P \vee Q) = (H1(P) \vee H1(Q))$
by (*rel-auto*)

lemma *design-export-H1*: $(P \vdash Q) = (P \vdash H1(Q))$
by (*rel-auto*)

The H1 algebraic laws are valid only when $\alpha(R)$ is homogeneous. This should maybe be generalised.

theorem *H1-algebraic-intro*:
assumes
$(true_h\ ;;\ R) = true_h$
$(II_D\ ;;\ R) = R$
shows *R is H1*

proof −
  have $R = (II_D \;;\; R)$ **by** (*simp add*: *assms(2)*)
  also have ... = $(H1(II) \;;\; R)$
    by (*simp add*: *H1-design-skip*)
  also have ... = $((\$ok \Rightarrow II) \;;\; R)$
    by (*simp add*: *H1-def*)
  also have ... = $(((\neg \$ok) \;;\; R) \vee R)$
    by (*simp add*: *impl-alt-def seqr-or-distl*)
  also have ... = $((((\neg \$ok) \;;\; true_h) \;;\; R) \vee R)$
    by (*simp add*: *precond-right-unit unrest*)
  also have ... = $(((\neg \$ok) \;;\; true_h) \vee R)$
    by (*metis assms(1) seqr-assoc*)
  also have ... = $(\$ok \Rightarrow R)$
    by (*simp add*: *impl-alt-def precond-right-unit unrest*)
  finally show *?thesis* **by** (*metis H1-def Healthy-def'*)
qed

lemma *nok-not-false*:
  $(\neg \$ok) \neq false$
  by (*pred-auto*)

theorem *H1-left-zero*:
  assumes *P is H1*
  shows $(true \;;\; P) = true$
proof −
  from *assms* **have** $(true \;;\; P) = (true \;;\; (\$ok \Rightarrow P))$
    by (*simp add*: *H1-def Healthy-def'*)

  also from *assms* **have** ... = $(true \;;\; (\neg \$ok \vee P))$ (**is** - = (*?true* ;; -))
    by (*simp add*: *impl-alt-def*)
  also from *assms* **have** ... = $((?true \;;\; (\neg \$ok)) \vee (?true \;;\; P))$
    using *seqr-or-distr* **by** *blast*
  also from *assms* **have** ... = $(true \vee (true \;;\; P))$
    by (*simp add*: *nok-not-false precond-left-zero unrest*)
  finally show *?thesis*
    by (*simp add*: *upred-defs urel-defs*)
qed

theorem *H1-left-unit*:
  fixes P :: $'\alpha$ *hrel-des*
  assumes *P is H1*
  shows $(II_D \;;\; P) = P$
proof −
  have $(II_D \;;\; P) = ((\$ok \Rightarrow II) \;;\; P)$
    by (*metis H1-def H1-design-skip*)
  also have ... = $(((\neg \$ok) \;;\; P) \vee P)$
    by (*simp add*: *impl-alt-def seqr-or-distl*)
  also from *assms* **have** ... = $((((\neg \$ok) \;;\; true_h) \;;\; P) \vee P)$
    by (*simp add*: *precond-right-unit unrest*)
  also have ... = $(((\neg \$ok) \;;\; (true_h \;;\; P)) \vee P)$
    by (*simp add*: *seqr-assoc*)
  also from *assms* **have** ... = $(\$ok \Rightarrow P)$
    by (*simp add*: *H1-left-zero impl-alt-def precond-right-unit unrest*)
  finally show *?thesis* **using** *assms*
    by (*simp add*: *H1-def Healthy-def'*)

qed

theorem *H1-algebraic*:
  P *is* $H1 \longleftrightarrow (true_h\ ;;\ P) = true_h \wedge (II_D\ ;;\ P) = P$
  using *H1-algebraic-intro H1-left-unit H1-left-zero* **by** *blast*

theorem *H1-nok-left-zero*:
  fixes $P :: {'}\alpha\ hrel\text{-}des$
  assumes P *is* $H1$
  shows $((\neg\ \$ok)\ ;;\ P) = (\neg\ \$ok)$
proof −
  have $((\neg\ \$ok)\ ;;\ P) = (((\neg\ \$ok)\ ;;\ true_h)\ ;;\ P)$
    by (*simp add*: *precond-right-unit unrest*)
  also have ... $= ((\neg\ \$ok)\ ;;\ true_h)$
    by (*metis H1-left-zero assms seqr-assoc*)
  also have ... $= (\neg\ \$ok)$
    by (*simp add*: *precond-right-unit unrest*)
  finally show *?thesis* **.**
qed

lemma *H1-design*:
  $H1(P \vdash Q) = (P \vdash Q)$
  by (*rel-auto*)

lemma *H1-rdesign*:
  $H1(P \vdash_r Q) = (P \vdash_r Q)$
  by (*rel-auto*)

lemma *H1-choice-closed* [*closure*]:
  $[\![\ P$ *is* $H1;\ Q$ *is* $H1\ ]\!] \Longrightarrow P \sqcap Q$ *is* $H1$
  by (*simp add*: *H1-def Healthy-def′ disj-upred-def impl-alt-def semilattice-sup-class.sup-left-commute*)

lemma *H1-inf-closed* [*closure*]:
  $[\![\ P$ *is* $H1;\ Q$ *is* $H1\ ]\!] \Longrightarrow P \sqcup Q$ *is* $H1$
  by (*rel-blast*)

lemma *H1-UINF*:
  assumes $A \neq \{\}$
  shows $H1(\bigsqcap\ i \in A \cdot P(i)) = (\bigsqcap\ i \in A \cdot H1(P(i)))$
  using *assms* **by** (*rel-auto*)

lemma *H1-Sup*:
  assumes $A \neq \{\}\ \forall\ P \in A.\ P$ *is* $H1$
  shows $(\bigsqcap\ A)$ *is* $H1$
proof −
  from *assms*(*2*) **have** $H1\ `\ A = A$
    by (*auto simp add*: *Healthy-def rev-image-eqI*)
  with *H1-UINF*[*of A id*, *OF assms*(*1*)] **show** *?thesis*
    by (*simp add*: *UINF-as-Sup-image Healthy-def*, *presburger*)
qed

lemma *H1-USUP*:
  shows $H1(\bigsqcup\ i \in A \cdot P(i)) = (\bigsqcup\ i \in A \cdot H1(P(i)))$
  by (*rel-auto*)

lemma *H1-Inf* [*closure*]:
  assumes $\forall\ P \in A.\ P$ *is H1*
  shows $(\bigsqcup\ A)$ *is H1*
proof –
  from *assms* **have** *H1* ‘ $A = A$
    by (*auto simp add*: *Healthy-def rev-image-eqI*)
  with *H1-USUP*[*of A id*] **show** *?thesis*
    by (*simp add*: *USUP-as-Inf-image Healthy-def*, *presburger*)
qed

2.2 H2: A specification cannot require non-termination

definition J :: $'\alpha$ *hrel-des* **where**
[*upred-defs*]: $J = ((\$ok \Rightarrow \$ok') \wedge \lceil II \rceil_D)$

definition *H2* **where**
[*upred-defs*]: $H2\ (P)\ \equiv\ P\ ;;\ J$

lemma *J-split*:
  shows $(P\ ;;\ J) = (P^f \vee (P^t \wedge \$ok'))$
proof –
  have $(P\ ;;\ J) = (P\ ;;\ ((\$ok \Rightarrow \$ok') \wedge \lceil II \rceil_D))$
    by (*simp add*: *H2-def J-def design-def*)
  also have ... $= (P\ ;;\ ((\$ok \Rightarrow \$ok \wedge \$ok') \wedge \lceil II \rceil_D))$
    by (*rel-auto*)
  also have ... $= ((P\ ;;\ (\neg\ \$ok \wedge \lceil II \rceil_D)) \vee (P\ ;;\ (\$ok \wedge (\lceil II \rceil_D \wedge \$ok'))))$
    by (*rel-auto*)
  also have ... $= (P^f \vee (P^t \wedge \$ok'))$
  proof –
    have $(P\ ;;\ (\neg\ \$ok \wedge \lceil II \rceil_D)) = P^f$
    proof –
      have $(P\ ;;\ (\neg\ \$ok \wedge \lceil II \rceil_D)) = ((P \wedge \neg\ \$ok')\ ;;\ \lceil II \rceil_D)$
        by (*rel-auto*)
      also have ... $= (\exists\ \$ok' \cdot P \wedge \$ok' =_u \mathit{false})$
        by (*rel-auto*)
      also have ... $= P^f$
        by (*metis C1 one-point out-var-uvar unrest-as-exists ok-vwb-lens vwb-lens-mwb*)
     finally show *?thesis* **.**
    qed
    moreover have $(P\ ;;\ (\$ok \wedge (\lceil II \rceil_D \wedge \$ok'))) = (P^t \wedge \$ok')$
    proof –
      have $(P\ ;;\ (\$ok \wedge (\lceil II \rceil_D \wedge \$ok'))) = (P\ ;;\ (\$ok \wedge II))$
        by (*rel-auto*)
      also have ... $= (P^t \wedge \$ok')$
        by (*rel-auto*)
      finally show *?thesis* **.**
    qed
    ultimately show *?thesis*
      by *simp*
  qed
  finally show *?thesis* **.**
qed

lemma *H2-split*:
  shows $H2(P) = (P^f \vee (P^t \wedge \$ok'))$
  by (*simp add*: *H2-def J-split*)

theorem *H2-equivalence*:
  P *is* $H2 \longleftrightarrow$ ‘$P^f \Rightarrow P^t$‘
proof −
  have ‘$P \Leftrightarrow (P \;;\; J)$‘ $\longleftrightarrow$ ‘$P \Leftrightarrow (P^f \vee (P^t \wedge \$ok'))$‘
    by (*simp add*: *J-split*)
  also have ... $\longleftrightarrow$ ‘$(P \Leftrightarrow P^f \vee P^t \wedge \$ok')^f \wedge (P \Leftrightarrow P^f \vee P^t \wedge \$ok')^t$‘
    by (*simp add*: *subst-bool-split*)
  also have ... = ‘$(P^f \Leftrightarrow P^f) \wedge (P^t \Leftrightarrow P^f \vee P^t)$‘
    by *subst-tac*
  also have ... = ‘$P^t \Leftrightarrow (P^f \vee P^t)$‘
    by (*pred-auto robust*)
  also have ... = ‘$(P^f \Rightarrow P^t)$‘
    by (*pred-auto*)
  finally show *?thesis*
    by (*metis H2-def Healthy-def′ taut-iff-eq*)
qed

lemma *H2-equiv*:
  P *is* $H2 \longleftrightarrow P^t \sqsubseteq P^f$
  using *H2-equivalence refBy-order* **by** *blast*

lemma *H2-design*:
  assumes $\$ok' \sharp P$ $\$ok' \sharp Q$
  shows $H2(P \vdash Q) = P \vdash Q$
  using *assms*
  by (*simp add*: *H2-split design-def usubst unrest*, *pred-auto*)

lemma *H2-rdesign*:
  $H2(P \vdash_r Q) = P \vdash_r Q$
  by (*simp add*: *H2-design unrest rdesign-def*)

theorem *J-idem*:
  $(J \;;\; J) = J$
  by (*rel-auto*)

theorem *H2-idem*:
  $H2(H2(P)) = H2(P)$
  by (*metis H2-def J-idem seqr-assoc*)

theorem *H2-Continuous*: *Continuous H2*
  by (*rel-auto*)

theorem *H2-not-okay*: $H2\ (\neg \$ok) = (\neg \$ok)$
proof −
  have $H2\ (\neg \$ok) = ((\neg \$ok)^f \vee ((\neg \$ok)^t \wedge \$ok'))$
    by (*simp add*: *H2-split*)
  also have ... $= (\neg \$ok \vee (\neg \$ok) \wedge \$ok')$
    by (*subst-tac*)
  also have ... $= (\neg \$ok)$
    by (*pred-auto*)
  finally show *?thesis* **.**
qed

lemma *H2-true*: $H2(true) = true$

by (*rel-auto*)

lemma *H2-choice-closed* [*closure*]:
$\llbracket$ *P is H2*; *Q is H2* $\rrbracket \Longrightarrow$ *P* $\sqcap$ *Q is H2*
by (*metis H2-def Healthy-def′ disj-upred-def seqr-or-distl*)

lemma *H2-inf-closed* [*closure*]:
assumes *P is H2 Q is H2*
shows *P* $\sqcup$ *Q is H2*
proof −
have $P \sqcup Q = (P^f \vee P^t \wedge \$ok') \sqcup (Q^f \vee Q^t \wedge \$ok')$
by (*metis H2-def Healthy-def J-split assms*(*1*) *assms*(*2*))
moreover have *H2*(...) = ...
by (*simp add*: *H2-split usubst*, *pred-auto*)
ultimately show *?thesis*
by (*simp add*: *Healthy-def*)
qed

lemma *H2-USUP*:
shows $H2(\bigsqcap\ i \in A \cdot P(i)) = (\bigsqcap\ i \in A \cdot H2(P(i)))$
by (*rel-auto*)

theorem *H1-H2-commute*:
H1 (*H2 P*) = *H2* (*H1 P*)
proof −
have $H2\ (H1\ P) = ((\$ok \Rightarrow P)\ ;;\ J)$
by (*simp add*: *H1-def H2-def*)
also have ... $= ((\neg\ \$ok \vee P)\ ;;\ J)$
by (*rel-auto*)
also have ... $= (((\neg\ \$ok)\ ;;\ J) \vee (P\ ;;\ J))$
using *seqr-or-distl* **by** *blast*
also have ... $= ((H2\ (\neg\ \$ok)) \vee H2(P))$
by (*simp add*: *H2-def*)
also have ... $= ((\neg\ \$ok) \vee H2(P))$
by (*simp add*: *H2-not-okay*)
also have ... $= H1(H2(P))$
by (*rel-auto*)
finally show *?thesis* **by** *simp*
qed

2.3 Designs as $H1$-$H2$ predicates

abbreviation *H1-H2* :: ($'\alpha$, $'\beta$) *rel-des* $\Rightarrow$ ($'\alpha$, $'\beta$) *rel-des* (**H**) **where**
H1-H2 P $\equiv$ *H1* (*H2 P*)

lemma *H1-H2-comp*: $\mathbf{H} = H1 \circ H2$
by (*auto*)

theorem *H1-H2-eq-design*:
$\mathbf{H}(P) = (\neg\ P^f) \vdash P^t$
proof −
have $\mathbf{H}(P) = (\$ok \Rightarrow H2(P))$
by (*simp add*: *H1-def*)
also have ... $= (\$ok \Rightarrow (P^f \vee (P^t \wedge \$ok')))$
by (*metis H2-split*)
also have ... $= (\$ok \wedge (\neg\ P^f) \Rightarrow \$ok' \wedge \$ok \wedge P^t)$

by (*rel-auto*)
also have ... = $(\neg\ P^f) \vdash P^t$
by (*rel-auto*)
finally show *?thesis* **.**
qed

theorem *H1-H2-is-design*:
assumes *P is H1 P is H2*
shows $P = (\neg\ P^f) \vdash P^t$
using *assms* **by** (*metis H1-H2-eq-design Healthy-def*)

theorem *H1-H2-eq-rdesign*:
$\mathbf{H}(P) = pre_D(P) \vdash_r post_D(P)$
proof −
have $\mathbf{H}(P) = (\$ok \Rightarrow H2(P))$
by (*simp add*: *H1-def Healthy-def′*)
also have ... = $(\$ok \Rightarrow (P^f \vee (P^t \wedge \$ok')))$
by (*metis H2-split*)
also have ... = $(\$ok \wedge (\neg\ P^f) \Rightarrow \$ok' \wedge P^t)$
by (*pred-auto*)
also have ... = $(\$ok \wedge (\neg\ P^f) \Rightarrow \$ok' \wedge \$ok \wedge P^t)$
by (*pred-auto*)
also have ... = $(\$ok \wedge \lceil pre_D(P)\rceil_D \Rightarrow \$ok' \wedge \$ok \wedge \lceil post_D(P)\rceil_D)$
by (*simp add*: *ok-post ok-pre*)
also have ... = $(\$ok \wedge \lceil pre_D(P)\rceil_D \Rightarrow \$ok' \wedge \lceil post_D(P)\rceil_D)$
by (*pred-auto*)
also have ... = $pre_D(P) \vdash_r post_D(P)$
by (*simp add*: *rdesign-def design-def*)
finally show *?thesis* **.**
qed

theorem *H1-H2-is-rdesign*:
assumes *P is H1 P is H2*
shows $P = pre_D(P) \vdash_r post_D(P)$
by (*metis H1-H2-eq-rdesign Healthy-def assms*(*1*) *assms*(*2*))

lemma *H1-H2-refinement*:
assumes *P is* **H** *Q is* **H**
shows $P \sqsubseteq Q \longleftrightarrow (`pre_D(P) \Rightarrow pre_D(Q)` \wedge `pre_D(P) \wedge post_D(Q) \Rightarrow post_D(P)`)$
by (*metis H1-H2-eq-rdesign Healthy-if assms rdesign-refinement*)

lemma *H1-H2-refines*:
assumes *P is* **H** *Q is* **H** $P \sqsubseteq Q$
shows $pre_D(Q) \sqsubseteq pre_D(P)$ $post_D(P) \sqsubseteq (pre_D(P) \wedge post_D(Q))$
using *H1-H2-refinement assms refBy-order* **by** *auto*

lemma *H1-H2-idempotent*: **H** (**H** *P*) = **H** *P*
by (*simp add*: *H1-H2-commute H1-idem H2-idem*)

lemma *H1-H2-Idempotent* [*closure*]: *Idempotent* **H**
by (*simp add*: *Idempotent-def H1-H2-idempotent*)

lemma *H1-H2-monotonic* [*closure*]: *Monotonic* **H**
by (*simp add*: *H1-monotone H2-def mono-def seqr-mono*)

lemma *H1-H2-Continuous* [*closure*]: *Continuous* **H**
  by (*simp add*: *Continuous-comp H1-Continuous H1-H2-comp H2-Continuous*)

lemma *design-is-H1-H2* [*closure*]:
  $[\![\$ok' \sharp P; \$ok' \sharp Q]\!] \Longrightarrow (P \vdash Q)$ *is* **H**
  by (*simp add*: *H1-design H2-design Healthy-def'*)

lemma *rdesign-is-H1-H2* [*closure*]:
  $(P \vdash_r Q)$ *is* **H**
  by (*simp add*: *Healthy-def H1-rdesign H2-rdesign*)

lemma *top-d-is-H1-H2* [*closure*]: $\top_D$ *is* **H**
  by (*simp add*: *H1-def H2-not-okay Healthy-intro impl-alt-def*)

lemma *bot-d-is-H1-H2* [*closure*]: $\bot_D$ *is* **H**
  by (*simp add*: *bot-d-def closure unrest*)

lemma *seq-r-H1-H2-closed* [*closure*]:
  assumes P *is* **H** Q *is* **H**
  shows $(P \;;;\; Q)$ *is* **H**
proof −
  obtain P_1 P_2 **where** $P = P_1 \vdash_r P_2$
    by (*metis H1-H2-commute H1-H2-is-rdesign H2-idem Healthy-def assms*(*1*))
  moreover obtain Q_1 Q_2 **where** $Q = Q_1 \vdash_r Q_2$
   by (*metis H1-H2-commute H1-H2-is-rdesign H2-idem Healthy-def assms*(*2*))
  moreover have $((P_1 \vdash_r P_2) \;;;\; (Q_1 \vdash_r Q_2))$ *is* **H**
    by (*simp add*: *rdesign-composition rdesign-is-H1-H2*)
  ultimately show *?thesis* **by** *simp*
qed

lemma *UINF-H1-H2-closed* [*closure*]:
  assumes $A \neq \{\}$ $\forall\ P \in A.\ P$ *is* **H**
  shows $(\bigsqcap A)$ *is H1-H2*
proof −
  from *assms* **have** *A*: $A = $ *H1-H2* ‘ A
    by (*auto simp add*: *Healthy-def rev-image-eqI*)
  also have $(\bigsqcap ...) = (\bigsqcap P \in A \cdot H1\text{-}H2(P))$
    by (*simp add*: *UINF-as-Sup-collect*)
  also have $... = (\bigsqcap P \in A \cdot (\neg P^f) \vdash P^t)$
    by (*meson H1-H2-eq-design*)
  also have $... = (\bigsqcup P \in A \cdot \neg P^f) \vdash (\bigsqcap P \in A \cdot P^t)$
    by (*simp add*: *design-UINF-mem assms*)
  also have ... *is H1-H2*
    by (*simp add*: *design-is-H1-H2 unrest*)
  finally show *?thesis* **.**
qed

definition *design-inf* :: $('\alpha, '\beta)$ *rel-des set* $\Rightarrow$ $('\alpha, '\beta)$ *rel-des* ($\bigsqcap_D$- [*900*] *900*) **where**
$\bigsqcap_D A = (if\ (A = \{\})\ then\ \top_D\ else\ \bigsqcap A)$

abbreviation *design-sup* :: $('\alpha, '\beta)$ *rel-des set* $\Rightarrow$ $('\alpha, '\beta)$ *rel-des* ($\bigsqcup_D$- [*900*] *900*) **where**
$\bigsqcup_D A \equiv \bigsqcup A$

lemma *design-inf-H1-H2-closed*:
  assumes $\forall\ P \in A.\ P$ *is* **H**

shows $(\bigsqcap_D\ A)$ *is* **H**
apply (*auto simp add*: *design-inf-def closure*)
apply (*simp add*: *H1-def H2-not-okay Healthy-def impl-alt-def*)
apply (*metis H1-def Healthy-def UINF-H1-H2-closed assms empty-iff impl-alt-def*)
done

lemma *design-sup-empty* [*simp*]: $\bigsqcap_D\ \{\} = \top_D$
by (*simp add*: *design-inf-def*)

lemma *design-sup-non-empty* [*simp*]: $A \neq \{\} \Longrightarrow \bigsqcap_D\ A = \bigsqcap\ A$
by (*simp add*: *design-inf-def*)

lemma *USUP-mem-H1-H2-closed*:
assumes $\bigwedge\ i.\ i \in A \Longrightarrow P\ i$ *is* **H**
shows $(\bigsqcup\ i{\in}A \cdot P\ i)$ *is* **H**
proof −
from *assms* **have** $(\bigsqcup\ i{\in}A \cdot P\ i) = (\bigsqcup\ i{\in}A \cdot \mathbf{H}(P\ i))$
by (*auto intro*: *USUP-cong simp add*: *Healthy-def*)
also have ... $= (\bigsqcup\ i{\in}A \cdot (\neg\ (P\ i)^f) \vdash (P\ i)^t)$
by (*meson H1-H2-eq-design*)
also have ... $= (\bigsqcap\ i{\in}A \cdot \neg\ (P\ i)^f) \vdash (\bigsqcup\ i{\in}A \cdot \neg\ (P\ i)^f \Rightarrow (P\ i)^t)$
by (*simp add*: *design-USUP-mem*)
also have ... *is* **H**
by (*simp add*: *design-is-H1-H2 unrest*)
finally show *?thesis* **.**
qed

lemma *USUP-ind-H1-H2-closed*:
assumes $\bigwedge\ i.\ P\ i$ *is* **H**
shows $(\bigsqcup\ i \cdot P\ i)$ *is* **H**
using *assms USUP-mem-H1-H2-closed*[*of UNIV P*] **by** *simp*

lemma *Inf-H1-H2-closed*:
assumes $\forall\ P \in A.\ P$ *is* **H**
shows $(\bigsqcup\ A)$ *is* **H**
proof −
from *assms* **have** A: $A = \mathbf{H}$ ‘ A
by (*auto simp add*: *Healthy-def rev-image-eqI*)
also have $(\bigsqcup\ ...) = (\bigsqcup\ P \in A \cdot \mathbf{H}(P))$
by (*simp add*: *USUP-as-Inf-collect*)
also have ... $= (\bigsqcup\ P \in A \cdot (\neg\ P^f) \vdash P^t)$
by (*meson H1-H2-eq-design*)
also have ... $= (\bigsqcap\ P \in A \cdot \neg\ P^f) \vdash (\bigsqcup\ P \in A \cdot \neg\ P^f \Rightarrow P^t)$
by (*simp add*: *design-USUP-mem*)
also have ... *is* **H**
by (*simp add*: *design-is-H1-H2 unrest*)
finally show *?thesis* **.**
qed

lemma *rdesign-ref-monos*:
assumes P *is* **H** Q *is* **H** $P \sqsubseteq Q$
shows $pre_D(Q) \sqsubseteq pre_D(P)\ post_D(P) \sqsubseteq (pre_D(P) \wedge post_D(Q))$
proof −
have r: $P \sqsubseteq Q \longleftrightarrow$ (‘$pre_D(P) \Rightarrow pre_D(Q)$‘ $\wedge$ ‘$pre_D(P) \wedge post_D(Q) \Rightarrow post_D(P)$‘)
by (*metis H1-H2-eq-rdesign Healthy-if assms*(*1*) *assms*(*2*) *rdesign-refinement*)

from *r assms* **show** $pre_D(Q) \sqsubseteq pre_D(P)$
by (*auto simp add*: *refBy-order*)
from *r assms* **show** $post_D(P) \sqsubseteq (pre_D(P) \wedge post_D(Q))$
by (*auto simp add*: *refBy-order*)
qed

2.4 H3: The design assumption is a precondition

definition $H3 :: ('\alpha, '\beta)$ *rel-des* $\Rightarrow$ $('\alpha, '\beta)$ *rel-des* **where**
[*upred-defs*]: $H3\ (P) \equiv P\ ;;\ II_D$

theorem *H3-idem*:
$H3(H3(P)) = H3(P)$
by (*metis H3-def design-skip-idem seqr-assoc*)

theorem *H3-mono*:
$P \sqsubseteq Q \Longrightarrow H3(P) \sqsubseteq H3(Q)$
by (*simp add*: *H3-def seqr-mono*)

theorem *H3-Monotonic*:
Monotonic H3
by (*simp add*: *H3-mono mono-def*)

theorem *H3-Continuous*: *Continuous H3*
by (*rel-auto*)

theorem *design-condition-is-H3*:
assumes $out\alpha \sharp p$
shows $(p \vdash Q)$ *is H3*
proof −
have $((p \vdash Q)\ ;;\ II_D) = (\neg\ ((\neg\ p)\ ;;\ true)) \vdash (Q^t\ ;;\ II[\![true/\$ok]\!])$
by (*simp add*: *skip-d-alt-def design-composition-subst unrest assms*)
also have ... $= p \vdash (Q^t\ ;;\ II[\![true/\$ok]\!])$
using *assms precond-equiv seqr-true-lemma* **by** *force*
also have ... $= p \vdash Q$
by (*rel-auto*)
finally show *?thesis*
by (*simp add*: *H3-def Healthy-def'*)
qed

theorem *rdesign-H3-iff-pre*:
$P \vdash_r Q$ *is H3* $\longleftrightarrow P = (P\ ;;\ true)$
proof −
have $(P \vdash_r Q)\ ;;\ II_D = (P \vdash_r Q)\ ;;\ (true \vdash_r II)$
by (*simp add*: *skip-d-def*)
also have ... $= (\neg\ ((\neg\ P)\ ;;\ true) \wedge \neg\ (Q\ ;;\ (\neg\ true))) \vdash_r (Q\ ;;\ II)$
by (*simp add*: *rdesign-composition*)
also have ... $= (\neg\ ((\neg\ P)\ ;;\ true) \wedge \neg\ (Q\ ;;\ (\neg\ true))) \vdash_r Q$
by *simp*
also have ... $= (\neg\ ((\neg\ P)\ ;;\ true)) \vdash_r Q$
by (*pred-auto*)
finally have $P \vdash_r Q$ *is H3* $\longleftrightarrow P \vdash_r Q = (\neg\ ((\neg\ P)\ ;;\ true)) \vdash_r Q$
by (*metis H3-def Healthy-def'*)
also have ... $\longleftrightarrow P = (\neg\ ((\neg\ P)\ ;;\ true))$
by (*metis rdesign-pre*)
thm *seqr-true-lemma*

also have ... $\longleftrightarrow P = (P ;; true)$
by (*simp add*: *seqr-true-lemma*)
finally show *?thesis* **.**
qed

theorem *design-H3-iff-pre*:
assumes $\$ok \sharp P$ $\$ok' \sharp P$ $\$ok \sharp Q$ $\$ok' \sharp Q$
shows $P \vdash Q$ *is H3* $\longleftrightarrow P = (P ;; true)$
proof −
have $P \vdash Q = \lfloor P \rfloor_D \vdash_r \lfloor Q \rfloor_D$
by (*simp add*: *assms lift-desr-inv rdesign-def*)
moreover hence $\lfloor P \rfloor_D \vdash_r \lfloor Q \rfloor_D$ *is H3* $\longleftrightarrow \lfloor P \rfloor_D = (\lfloor P \rfloor_D ;; true)$
using *rdesign-H3-iff-pre* **by** *blast*
ultimately show *?thesis*
by (*metis assms*(*1*,*2*) *drop-desr-inv lift-desr-inv lift-dist-seq aext-true*)
qed

theorem *H1-H3-commute*:
H1 (*H3 P*) = *H3* (*H1 P*)
by (*rel-auto*)

lemma *skip-d-absorb-J-1*:
$(II_D ;; J) = II_D$
by (*metis H2-def H2-rdesign skip-d-def*)

lemma *skip-d-absorb-J-2*:
$(J ;; II_D) = II_D$
proof −
have $(J ;; II_D) = ((\$ok \Rightarrow \$ok') \wedge \lceil II \rceil_D) ;; (true \vdash II)$
by (*simp add*: *J-def skip-d-alt-def*)
also have ... $= (\exists\ ok_0 \cdot ((\$ok \Rightarrow \$ok') \wedge \lceil II \rceil_D)[\![\ll ok_0 \gg/\$ok']\!] ;; (true \vdash II)[\![\ll ok_0 \gg/\$ok]\!])$
by (*subst seqr-middle*[*of ok*], *simp-all*)
also have ... $= ((((\$ok \Rightarrow \$ok') \wedge \lceil II \rceil_D)[\![false/\$ok']\!] ;; (true \vdash II)[\![false/\$ok]\!])$
$\vee (((\$ok \Rightarrow \$ok') \wedge \lceil II \rceil_D)[\![true/\$ok']\!] ;; (true \vdash II)[\![true/\$ok]\!]))$
by (*simp add*: *disj-comm false-alt-def true-alt-def*)
also have ... $= ((\neg \$ok \wedge \lceil II \rceil_D ;; true) \vee (\lceil II \rceil_D ;; \$ok' \wedge \lceil II \rceil_D))$
by (*rel-auto*)
also have ... $= II_D$
by (*rel-auto*)
finally show *?thesis* **.**
qed

lemma *H2-H3-absorb*:
H2 (*H3 P*) = *H3 P*
by (*metis H2-def H3-def seqr-assoc skip-d-absorb-J-1*)

lemma *H3-H2-absorb*:
H3 (*H2 P*) = *H3 P*
by (*metis H2-def H3-def seqr-assoc skip-d-absorb-J-2*)

theorem *H2-H3-commute*:
H2 (*H3 P*) = *H3* (*H2 P*)
by (*simp add*: *H2-H3-absorb H3-H2-absorb*)

theorem *H3-design-pre*:

assumes $\$ok \sharp p$ $out\alpha \sharp p$ $\$ok \sharp Q$ $\$ok' \sharp Q$
shows $H3(p \vdash Q) = p \vdash Q$
using *assms*
by (*metis Healthy-def′ design-H3-iff-pre precond-right-unit unrest-out*α*-var ok-vwb-lens vwb-lens-mwb*)

theorem *H3-rdesign-pre*:
assumes $out\alpha \sharp p$
shows $H3(p \vdash_r Q) = p \vdash_r Q$
using *assms*
by (*simp add*: *H3-def*)

theorem *H3-ndesign*: $H3(p \vdash_n Q) = (p \vdash_n Q)$
by (*simp add*: *H3-def ndesign-def unrest-pre-out*α)

theorem *ndesign-is-H3* [*closure*]: $p \vdash_n Q$ *is H3*
by (*simp add*: *H3-ndesign Healthy-def*)

2.5 Normal Designs as $H1$-$H2$ predicates

abbreviation *H1-H3* :: $('\alpha, '\beta)$ *rel-des* $\Rightarrow$ $('\alpha, '\beta)$ *rel-des* (**N**) **where**
H1-H3 p $\equiv$ *H1* (*H3 p*)

lemma *H1-H3-comp*: *H1-H3* = *H1* $\circ$ *H3*
by (*auto*)

theorem *H1-H3-is-design*:
assumes *P is H1 P is H3*
shows $P = (\neg P^f) \vdash P^t$
by (*metis H1-H2-eq-design H2-H3-absorb Healthy-def′ assms*(*1*) *assms*(*2*))

theorem *H1-H3-is-rdesign*:
assumes *P is H1 P is H3*
shows $P = pre_D(P) \vdash_r post_D(P)$
by (*metis H1-H2-is-rdesign H2-H3-absorb Healthy-def′ assms*)

theorem *H1-H3-is-normal-design*:
assumes *P is H1 P is H3*
shows $P = \lfloor pre_D(P) \rfloor_< \vdash_n post_D(P)$
by (*metis H1-H3-is-rdesign assms drop-pre-inv ndesign-def precond-equiv rdesign-H3-iff-pre*)

lemma *H1-H3-idempotent*: **N** (**N** *P*) = **N** *P*
by (*simp add*: *H1-H3-commute H1-idem H3-idem*)

lemma *H1-H3-Idempotent* [*closure*]: *Idempotent* **N**
by (*simp add*: *Idempotent-def H1-H3-idempotent*)

lemma *H1-H3-monotonic* [*closure*]: *Monotonic* **N**
by (*simp add*: *H1-monotone H3-mono mono-def*)

lemma *H1-H3-Continuous* [*closure*]: *Continuous* **N**
by (*simp add*: *Continuous-comp H1-Continuous H1-H3-comp H3-Continuous*)

lemma *H1-H3-intro*:
assumes *P is* **H** $out\alpha \sharp pre_D(P)$
shows *P is* **N**
by (*metis H1-H2-eq-rdesign H1-rdesign H3-rdesign-pre Healthy-def′ assms*)

lemma *H1-H3-impl-H2* [*closure*]: P *is* $\mathbf{N} \Longrightarrow P$ *is* $\mathbf{H}$
by (*metis H1-H2-commute H1-idem H2-H3-absorb Healthy-def'*)

lemma *H1-H3-eq-design-d-comp*: $\mathbf{N}(P) = ((\neg\ P^f) \vdash P^t)$;; II_D
by (*metis H1-H2-eq-design H1-H3-commute H3-H2-absorb H3-def*)

lemma *H1-H3-eq-design*: $\mathbf{N}(P) = (\neg\ (P^f\ ;;\ true)) \vdash P^t$
apply (*simp add*: *H1-H3-eq-design-d-comp skip-d-alt-def*)
apply (*subst design-composition-subst*)
apply (*simp-all add*: *usubst unrest*)
apply (*rel-auto*)
done

lemma *H3-unrest-out-alpha-nok* [*unrest*]:
assumes P *is* $\mathbf{N}$
shows $out\alpha \sharp P^f$
proof –
have $P = (\neg\ (P^f\ ;;\ true)) \vdash P^t$
by (*metis H1-H3-eq-design Healthy-def assms*)
also have $out\alpha \sharp (...^f)$
by (*simp add*: *design-def usubst unrest*, *rel-auto*)
finally show *?thesis* **.**
qed

lemma *H3-unrest-out-alpha* [*unrest*]: P *is* $\mathbf{N} \Longrightarrow out\alpha \sharp pre_D(P)$
by (*metis H1-H3-commute H1-H3-is-rdesign H1-idem Healthy-def' precond-equiv rdesign-H3-iff-pre*)

lemma *ndesign-H1-H3* [*closure*]: $p \vdash_n Q$ *is* $\mathbf{N}$
by (*simp add*: *H1-rdesign H3-def Healthy-def' ndesign-def unrest-pre-out*α)

lemma *ndesign-form*: P *is* $\mathbf{N} \Longrightarrow (\lfloor pre_D(P) \rfloor_< \vdash_n post_D(P)) = P$
by (*metis H1-H2-eq-rdesign H1-H3-impl-H2 H3-unrest-out-alpha Healthy-def drop-pre-inv ndesign-def*)

lemma *des-bot-H1-H3* [*closure*]: $\bot_D$ *is* $\mathbf{N}$
by (*metis H1-design H3-def Healthy-def' design-false-pre design-true-left-zero skip-d-alt-def bot-d-def*)

lemma *des-top-is-H1-H3* [*closure*]: $\top_D$ *is* $\mathbf{N}$
by (*metis ndesign-H1-H3 ndesign-miracle*)

lemma *skip-d-is-H1-H3* [*closure*]: II_D *is* $\mathbf{N}$
by (*simp add*: *ndesign-H1-H3 skip-d-ndes-def*)

lemma *seq-r-H1-H3-closed* [*closure*]:
assumes P *is* $\mathbf{N}$ Q *is* $\mathbf{N}$
shows (P ;; Q) *is* $\mathbf{N}$
by (*metis* (*no-types*) *H1-H2-eq-design H1-H3-eq-design-d-comp H1-H3-impl-H2 Healthy-def assms*(*1*) *assms*(*2*) *seq-r-H1-H2-closed seqr-assoc*)

lemma *dcond-H1-H2-closed* [*closure*]:
assumes P *is* $\mathbf{N}$ Q *is* $\mathbf{N}$
shows $(P \triangleleft b \triangleright_D Q)$ *is* $\mathbf{N}$
by (*metis assms ndesign-H1-H3 ndesign-dcond ndesign-form*)

lemma *inf-H1-H2-closed* [*closure*]:

assumes P *is* $\mathbf{N}$ Q *is* $\mathbf{N}$
shows $(P \sqcap Q)$ *is* $\mathbf{N}$
by (*metis assms ndesign-H1-H3 ndesign-choice ndesign-form*)

lemma *sup-H1-H2-closed* [*closure*]:
assumes P *is* $\mathbf{N}$ Q *is* $\mathbf{N}$
shows $(P \sqcup Q)$ *is* $\mathbf{N}$
by (*metis assms ndesign-H1-H3 ndesign-inf ndesign-form*)

lemma *ndes-seqr-miracle*:
assumes P *is* $\mathbf{N}$
shows P ;; $\top_D = \lfloor pre_D\ P \rfloor_< \vdash_n false$
proof −
have P ;; $\top_D = (\lfloor pre_D(P) \rfloor_< \vdash_n post_D(P))$;; $(true \vdash_n false)$
by (*simp add*: *assms ndesign-form ndesign-miracle*)
also have ... $= \lfloor pre_D\ P \rfloor_< \vdash_n false$
by (*simp add*: *ndesign-composition-wp wp alpha*)
finally show *?thesis* **.**
qed

lemma *ndes-seqr-abort*:
assumes P *is* $\mathbf{N}$
shows P ;; $\bot_D = (\lfloor pre_D\ P \rfloor_< \wedge post_D\ P\ wp\ false) \vdash_n false$
proof −
have P ;; $\bot_D = (\lfloor pre_D(P) \rfloor_< \vdash_n post_D(P))$;; $(false \vdash_n false)$
by (*simp add*: *assms bot-d-true ndesign-false-pre ndesign-form*)
also have ... $= (\lfloor pre_D\ P \rfloor_< \wedge post_D\ P\ wp\ false) \vdash_n false$
by (*simp add*: *ndesign-composition-wp alpha*)
finally show *?thesis* **.**
qed

lemma *USUP-ind-H1-H3-closed* [*closure*]:
$[\![\bigwedge i.\ P\ i\ is\ \mathbf{N}]\!] \Longrightarrow (\bigsqcup\ i \cdot P\ i)\ is\ \mathbf{N}$
by (*rule H1-H3-intro*, *simp-all add*: *H1-H3-impl-H2 USUP-ind-H1-H2-closed preD-USUP-ind unrest*)

2.6 H4: Feasibility

definition *H4* :: $('\alpha, '\beta)$ *rel-des* $\Rightarrow$ $('\alpha, '\beta)$ *rel-des* **where**
[*upred-defs*]: $H4(P) = ((P;;true) \Rightarrow P)$

theorem *H4-idem*:
$H4(H4(P)) = H4(P)$
by (*rel-auto*)

lemma *is-H4-alt-def*:
P *is* $H4 \longleftrightarrow (P$;; $true) = true$
by (*rel-blast*)

end

2.7 UTP theory of Designs

theory *utp-des-theory*
imports *utp-des-healths*
begin

2.8 UTP theories

typedecl *DES*
typedecl *NDES*

abbreviation $DES \equiv UTHY(DES, {'\alpha}\ des)$
abbreviation $NDES \equiv UTHY(NDES, {'\alpha}\ des)$

overloading
  $des\text{-}hcond == utp\text{-}hcond :: (DES, {'\alpha}\ des)\ uthy \Rightarrow ({'\alpha}\ des \times {'\alpha}\ des)\ health$
  $des\text{-}unit == utp\text{-}unit :: (DES, {'\alpha}\ des)\ uthy \Rightarrow {'\alpha}\ hrel\text{-}des$ (**unchecked**)

  $ndes\text{-}hcond == utp\text{-}hcond :: (NDES, {'\alpha}\ des)\ uthy \Rightarrow ({'\alpha}\ des \times {'\alpha}\ des)\ health$
  $ndes\text{-}unit == utp\text{-}unit :: (NDES, {'\alpha}\ des)\ uthy \Rightarrow {'\alpha}\ hrel\text{-}des$ (**unchecked**)

begin
  definition $des\text{-}hcond :: (DES, {'\alpha}\ des)\ uthy \Rightarrow ({'\alpha}\ des \times {'\alpha}\ des)\ health$ **where**
  [*upred-defs*]: $des\text{-}hcond\ t = H1\text{-}H2$

  definition $des\text{-}unit :: (DES, {'\alpha}\ des)\ uthy \Rightarrow {'\alpha}\ hrel\text{-}des$ **where**
  [*upred-defs*]: $des\text{-}unit\ t = II_D$

  definition $ndes\text{-}hcond :: (NDES, {'\alpha}\ des)\ uthy \Rightarrow ({'\alpha}\ des \times {'\alpha}\ des)\ health$ **where**
  [*upred-defs*]: $ndes\text{-}hcond\ t = H1\text{-}H3$

  definition $ndes\text{-}unit :: (NDES, {'\alpha}\ des)\ uthy \Rightarrow {'\alpha}\ hrel\text{-}des$ **where**
  [*upred-defs*]: $ndes\text{-}unit\ t = II_D$

end

interpretation *des-utp-theory*: *utp-theory DES*
  by (*simp add*: *H1-H2-commute H1-idem H2-idem des-hcond-def utp-theory-def*)

interpretation *ndes-utp-theory*: *utp-theory NDES*
  by (*simp add*: *H1-H3-commute H1-idem H3-idem ndes-hcond-def utp-theory.intro*)

interpretation *des-left-unital*: *utp-theory-left-unital DES*
  apply (*unfold-locales*)
  apply (*simp-all add*: *des-hcond-def des-unit-def*)
  using *seq-r-H1-H2-closed* **apply** *blast*
  apply (*simp add*: *rdesign-is-H1-H2 skip-d-def*)
  apply (*metis H1-idem H1-left-unit Healthy-def'*)
done

interpretation *ndes-unital*: *utp-theory-unital NDES*
  apply (*unfold-locales*, *simp-all add*: *ndes-hcond-def ndes-unit-def*)
  using *seq-r-H1-H3-closed* **apply** *blast*
  apply (*metis H1-rdesign H3-def Healthy-def' design-skip-idem skip-d-def*)
  apply (*metis H1-idem H1-left-unit Healthy-def'*)
  apply (*metis H1-H3-commute H3-def H3-idem Healthy-def'*)
done

interpretation *design-theory-continuous*: *utp-theory-continuous DES*
  rewrites $\bigwedge P.\ P \in carrier\ (uthy\text{-}order\ DES) \longleftrightarrow P\ is\ \mathbf{H}$
  and $carrier\ (uthy\text{-}order\ DES) \to carrier\ (uthy\text{-}order\ DES) \equiv [\![\mathbf{H}]\!]_H \to [\![\mathbf{H}]\!]_H$
  and $[\![\mathcal{H}_{DES}]\!]_H \to [\![\mathcal{H}_{DES}]\!]_H \equiv [\![\mathbf{H}]\!]_H \to [\![\mathbf{H}]\!]_H$

and $le\ (uthy\text{-}order\ DES) = op\ \sqsubseteq$
and $eq\ (uthy\text{-}order\ DES) = op\ =$
by (*unfold-locales*, *simp-all add*: *des-hcond-def H1-H2-Continuous utp-order-def*)

interpretation *normal-design-theory-continuous*: *utp-theory-continuous NDES*
rewrites $\bigwedge P.\ P \in carrier\ (uthy\text{-}order\ NDES) \longleftrightarrow P\ is\ \mathbf{N}$
and $carrier\ (uthy\text{-}order\ NDES) \to carrier\ (uthy\text{-}order\ NDES) \equiv [\![\mathbf{N}]\!]_H \to [\![\mathbf{N}]\!]_H$
and $[\![\mathcal{H}_{NDES}]\!]_H \to [\![\mathcal{H}_{NDES}]\!]_H \equiv [\![\mathbf{N}]\!]_H \to [\![\mathbf{N}]\!]_H$
and $le\ (uthy\text{-}order\ NDES) = op\ \sqsubseteq$
and $A \subseteq carrier\ (uthy\text{-}order\ NDES) \longleftrightarrow A \subseteq [\![\mathbf{N}]\!]_H$
and $eq\ (uthy\text{-}order\ NDES) = op\ =$
by (*unfold-locales*, *simp-all add*: *ndes-hcond-def H1-H3-Continuous utp-order-def*)

lemma *design-lat-top*: $\top_{DES} = \mathbf{H}(false)$
by (*simp add*: *design-theory-continuous.healthy-top*, *simp add*: *des-hcond-def*)

lemma *design-lat-bottom*: $\bot_{DES} = \mathbf{H}(true)$
by (*simp add*: *design-theory-continuous.healthy-bottom*, *simp add*: *des-hcond-def*)

lemma *ndesign-lat-top*: $\top_{NDES} = \mathbf{N}(false)$
by (*metis ndes-hcond-def normal-design-theory-continuous.healthy-top*)

lemma *ndesign-lat-bottom*: $\bot_{NDES} = \mathbf{N}(true)$
by (*metis ndes-hcond-def normal-design-theory-continuous.healthy-bottom*)

2.9 Galois Connection

Example Galois connection between designs and relations. Based on Jim's example in COMPASS deliverable D23.5.

definition [*upred-defs*]: $Des(R) = \mathbf{H}(\lceil R \rceil_D \wedge \$ok')$
definition [*upred-defs*]: $Rel(D) = \lfloor D[\![true,true/\$ok,\$ok']\!] \rfloor_D$

lemma *Des-design*: $Des(R) = true \vdash_r R$
by (*rel-auto*)

lemma *Rel-design*: $Rel(P \vdash_r Q) = (P \Rightarrow Q)$
by (*rel-auto*)

interpretation *Des-Rel-coretract*:
coretract $DES \leftarrow\langle Des,Rel\rangle\rightarrow REL$
rewrites
$\bigwedge x.\ x \in carrier\ \mathcal{X}_{DES \leftarrow\langle Des,Rel\rangle\rightarrow REL} = (x\ is\ \mathbf{H})$ **and**
$\bigwedge x.\ x \in carrier\ \mathcal{Y}_{DES \leftarrow\langle Des,Rel\rangle\rightarrow REL} = True$ **and**
$\pi_{*DES \leftarrow\langle Des,Rel\rangle\rightarrow REL} = Des$ **and**
$\pi^{*}{}_{DES \leftarrow\langle Des,Rel\rangle\rightarrow REL} = Rel$ **and**
$le\ \mathcal{X}_{DES \leftarrow\langle Des,Rel\rangle\rightarrow REL} = op\ \sqsubseteq$ **and**
$le\ \mathcal{Y}_{DES \leftarrow\langle Des,Rel\rangle\rightarrow REL} = op\ \sqsubseteq$
proof (*unfold-locales*, *simp-all add*: *rel-hcond-def des-hcond-def*)
show $\bigwedge x.\ x\ is\ id$
by (*simp add*: *Healthy-def*)
next
show $Rel \in [\![\mathbf{H}]\!]_H \to [\![id]\!]_H$
by (*auto simp add*: *Rel-def rel-hcond-def Healthy-def*)
next

show *Des* $\in [\![id]\!]_H \rightarrow [\![\mathbf{H}]\!]_H$
  by (*auto simp add*: *Des-def des-hcond-def Healthy-def H1-H2-commute H1-idem H2-idem*)
next
  fix R :: *'a hrel*
  show $R \sqsubseteq$ *Rel* (*Des* R)
    by (*simp add*: *Des-design Rel-design*)
next
  fix R :: *'a hrel* **and** D :: *'a hrel-des*
  assume *a*: D *is* **H**
  then obtain D_1 D_2 **where** D: $D = D_1 \vdash_r D_2$
    by (*metis H1-H2-commute H1-H2-is-rdesign H1-idem Healthy-def'*)
  show (*Rel* $D \sqsubseteq R$) = ($D \sqsubseteq$ *Des* R)
  proof −
    have ($D \sqsubseteq$ *Des* R) = ($D_1 \vdash_r D_2 \sqsubseteq$ *true* $\vdash_r R$)
      by (*simp add*: *D Des-design*)
    also have ... = ‘$D_1 \wedge R \Rightarrow D_2$‘
      by (*simp add*: *rdesign-refinement*)
    also have ... = (($D_1 \Rightarrow D_2$) $\sqsubseteq R$)
      by (*rel-auto*)
    also have ... = (*Rel* $D \sqsubseteq R$)
      by (*simp add*: *D Rel-design*)
    finally show *?thesis* **..**
  qed
qed

From this interpretation we gain many Galois theorems. Some require simplification to remove superfluous assumptions.

thm *Des-Rel-coretract.deflation*[*simplified*]
thm *Des-Rel-coretract.inflation*
thm *Des-Rel-coretract.upper-comp*[*simplified*]
thm *Des-Rel-coretract.lower-comp*

2.10 Fixed Points

abbreviation *design-lfp* :: ($'\alpha$ *hrel-des* $\Rightarrow$ $'\alpha$ *hrel-des*) $\Rightarrow$ $'\alpha$ *hrel-des* (μ_D) **where**
$\mu_D\ F \equiv \boldsymbol{\mu}_{DES}\ F$

abbreviation *design-gfp* :: ($'\alpha$ *hrel-des* $\Rightarrow$ $'\alpha$ *hrel-des*) $\Rightarrow$ $'\alpha$ *hrel-des* (ν_D) **where**
$\nu_D\ F \equiv \boldsymbol{\nu}_{DES}\ F$

syntax
  -dmu :: *pttrn* $\Rightarrow$ *logic* $\Rightarrow$ *logic* (μ_D - $\cdot$ - [*0*, *10*] *10*)
  -dnu :: *pttrn* $\Rightarrow$ *logic* $\Rightarrow$ *logic* (ν_D - $\cdot$ - [*0*, *10*] *10*)

translations
  $\mu_D\ X \cdot P == \boldsymbol{\mu}_{CONST\ DES}\ (\lambda\ X.\ P)$
  $\nu_D\ X \cdot P == \boldsymbol{\nu}_{CONST\ DES}\ (\lambda\ X.\ P)$

thm *design-theory-continuous.GFP-unfold*
thm *design-theory-continuous.LFP-unfold*

Specialise *mu-refine-intro* to designs.

lemma *design-mu-refine-intro*:
  assumes \$*ok*´ $\sharp$ C \$*ok*´ $\sharp$ S ($C \vdash S$) $\sqsubseteq$ F($C \vdash S$) ‘$C \Rightarrow (\mu_D\ F \Leftrightarrow \nu_D\ F)$‘
  shows ($C \vdash S$) $\sqsubseteq \mu_D\ F$

proof −
  from *assms* **have** $(C \vdash S) \sqsubseteq \nu_D\ F$
    thm *design-theory-continuous.weak.GFP-upperbound*
    by (*simp add*: *design-is-H1-H2 design-theory-continuous.weak.GFP-upperbound*)
  with *assms* **show** *?thesis*
    by (*rel-auto*, *metis* (*no-types*, *lifting*))
qed

lemma *rdesign-mu-refine-intro*:
  assumes $(C \vdash_r S) \sqsubseteq F(C \vdash_r S)$ ‘$\lceil C \rceil_D \Rightarrow (\mu_D\ F \Leftrightarrow \nu_D\ F)$‘
  shows $(C \vdash_r S) \sqsubseteq \mu_D\ F$
  using *assms* **by** (*simp add*: *rdesign-def design-mu-refine-intro unrest*)

lemma *H1-H2-mu-refine-intro*:
  assumes P *is* $\mathbf{H}$ $P \sqsubseteq F(P)$ ‘$\lceil pre_D(P) \rceil_D \Rightarrow (\mu_D\ F \Leftrightarrow \nu_D\ F)$‘
  shows $P \sqsubseteq \mu_D\ F$
  by (*metis H1-H2-eq-rdesign Healthy-if assms rdesign-mu-refine-intro*)

Foundational theorem for recursion introduction using a well-founded relation. Contributed by Dr. Yakoub Nemouchi.

theorem *rdesign-mu-wf-refine-intro*:
  assumes *WF*: *wf R*
    and *M*: *Monotonic F*
    and *H*: $F \in [\![\mathbf{H}]\!]_H \to [\![\mathbf{H}]\!]_H$
    and *induct-step*:
    $\bigwedge st.\ (P \wedge \lceil e \rceil_< =_u \ll st \gg) \vdash_r Q \sqsubseteq F\ ((P \wedge (\lceil e \rceil_<, \ll st \gg)_u \in_u \ll R \gg) \vdash_r Q)$
  shows $(P \vdash_r Q) \sqsubseteq \mu_D\ F$
proof −
  {
  fix *st*
  have $(P \wedge \lceil e \rceil_< =_u \ll st \gg) \vdash_r Q \sqsubseteq \mu_D\ F$
  using *WF* **proof** (*induction rule*: *wf-induct-rule*)
    case (*less st*)
    hence *0*: $(P \wedge (\lceil e \rceil_<, \ll st \gg)_u \in_u \ll R \gg) \vdash_r Q \sqsubseteq \mu_D\ F$
      by *rel-blast*
    from *M H design-theory-continuous.LFP-lemma3 mono-Monotone-utp-order*
    have *1*: $\mu_D\ F \sqsubseteq\ F\ (\mu_D\ F)$
      by *blast*
    from *0 1* **have** *2*:$(P \wedge (\lceil e \rceil_<,\ll st \gg)_u{\in_u}\ll R \gg) \vdash_r Q \sqsubseteq F\ (\mu_D\ F)$
      by *simp*
    have *3*: $F\ ((P \wedge (\lceil e \rceil_<, \ll st \gg)_u \in_u \ll R \gg) \vdash_r Q) \sqsubseteq F\ (\mu_D\ F)$
      by (*simp add*: *0 M monoD*)
    have *4*:$(P \wedge \lceil e \rceil_< =_u \ll st \gg) \vdash_r Q \sqsubseteq \ldots$
      by (*rule induct-step*)
    show *?case*
    using *order-trans*[*OF 3 4*] *H M design-theory-continuous.LFP-lemma2 dual-order.trans mono-Monotone-utp-order*

      by *blast*
  qed
  }
  thus *?thesis*
    by (*pred-simp*)
qed

theorem *ndesign-mu-wf-refine-intro′*:

assumes *WF*: *wf R*
and *M*: *Monotonic F*
and *H*: $F \in [\![\mathbf{H}]\!]_H \rightarrow [\![\mathbf{H}]\!]_H$
and *induct-step*:
$\bigwedge st.\ ((p \wedge e =_u \ll st\gg) \vdash_n Q) \sqsubseteq F\ ((p \wedge (e, \ll st\gg)_u \in_u \ll R\gg) \vdash_n Q)$
shows $(p \vdash_n Q) \sqsubseteq \mu_D\ F$
using *assms* **unfolding** *ndesign-def*
by (*rule-tac rdesign-mu-wf-refine-intro*[*of R F* $\lceil p \rceil_<$ *e*], *simp-all add*: *alpha*)

theorem *ndesign-mu-wf-refine-intro*:
assumes *WF*: *wf R*
and *M*: *Monotonic F*
and *H*: $F \in [\![\mathbf{N}]\!]_H \rightarrow [\![\mathbf{N}]\!]_H$
and *induct-step*:
$\bigwedge st.\ ((p \wedge e =_u \ll st\gg) \vdash_n Q) \sqsubseteq F\ ((p \wedge (e, \ll st\gg)_u \in_u \ll R\gg) \vdash_n Q)$
shows $(p \vdash_n Q) \sqsubseteq \boldsymbol{\mu}_{NDES}\ F$
proof −
{
fix *st*
have $(p \wedge e =_u \ll st\gg) \vdash_n Q \sqsubseteq \boldsymbol{\mu}_{NDES}\ F$
using *WF* **proof** (*induction rule*: *wf-induct-rule*)
case (*less st*)
hence *0*: $(p \wedge (e, \ll st\gg)_u \in_u \ll R\gg) \vdash_n Q \sqsubseteq \boldsymbol{\mu}_{NDES}\ F$
by *rel-blast*
from *M H design-theory-continuous.LFP-lemma3 mono-Monotone-utp-order*
have *1*: $\boldsymbol{\mu}_{NDES}\ F \sqsubseteq\ F\ (\boldsymbol{\mu}_{NDES}\ F)$
by (*simp add*: *mono-Monotone-utp-order normal-design-theory-continuous.LFP-lemma3*)
from *0 1* **have** *2*:$(p \wedge (e, \ll st\gg)_u{\in_u}\ll R\gg) \vdash_n Q \sqsubseteq F\ (\boldsymbol{\mu}_{NDES}\ F)$
by *simp*
have *3*: $F\ ((p \wedge (e, \ll st\gg)_u \in_u \ll R\gg) \vdash_n Q) \sqsubseteq F\ (\boldsymbol{\mu}_{NDES}\ F)$
by (*simp add*: *0 M monoD*)
have *4*:$(p \wedge e =_u \ll st\gg) \vdash_n Q \sqsubseteq \ldots$
by (*rule induct-step*)
show *?case*
using *order-trans*[*OF 3 4*] *H M normal-design-theory-continuous.LFP-lemma2 dual-order.trans mono-Monotone-utp-order*
by *blast*
qed
}
thus *?thesis*
by (*pred-simp*)
qed

end

3 Design Proof Tactics

theory *utp-des-tactics*
imports *utp-des-theory*
begin

The tactics split apart a healthy normal design predicate into its pre-postcondition form, using elimination rules, and then attempt to prove refinement conjectures.

named-theorems *ND-elim*

lemma *ndes-elim*: $[\![\ P\ is\ \mathbf{N};\ Q(\lfloor pre_D(P)\rfloor_< \vdash_n post_D(P))\ ]\!] \Longrightarrow Q(P)$
by (*simp add*: *ndesign-form*)

lemma *ndes-ind-elim*: $[\![\ \bigwedge i.\ P\ i\ is\ \mathbf{N};\ Q(\lambda\ i.\ \lfloor pre_D(P\ i)\rfloor_< \vdash_n post_D(P\ i))\ ]\!] \Longrightarrow Q(P)$
by (*simp add*: *ndesign-form*)

lemma *ndes-split* [*ND-elim*]: $[\![\ P\ is\ \mathbf{N};\ \bigwedge pre\ post.\ Q(pre \vdash_n post)\ ]\!] \Longrightarrow Q(P)$
by (*metis H1-H2-eq-rdesign H1-H3-impl-H2 H3-unrest-out-alpha Healthy-def drop-pre-inv ndesign-def*)

Use given closure laws (cls) to expand normal design predicates

method *ndes-expand* **uses** *cls* = (*insert cls*, (*erule ND-elim*)+)

Expand and simplify normal designs

method *ndes-simp* **uses** *cls* =
((*ndes-expand cls*: *cls*)?, (*simp add*: *ndes-simp closure alpha usubst unrest wp prod.case-eq-if*))

Attempt to discharge a refinement between two normal designs

method *ndes-refine* **uses** *cls* =
(*ndes-simp cls*: *cls*; *rule-tac ndesign-refine-intro*; (*insert cls*; *rel-simp*; *auto?*))

Attempt to discharge an equality between two normal designs

method *ndes-eq* **uses** *cls* =
(*ndes-simp cls*: *cls*; *rule-tac antisym*; *rule-tac ndesign-refine-intro*; (*insert cls*; *rel-simp*; *auto?*))

end

4 Imperative Programming in Designs

theory *utp-des-prog*
imports *utp-des-tactics*
begin

4.1 Assignment

definition *assigns-d* :: $'\alpha$ *usubst* $\Rightarrow$ $'\alpha$ *hrel-des* ($\langle\text{-}\rangle_D$) **where**
[*upred-defs*]: *assigns-d* σ = (*true* $\vdash_r$ *assigns-r* σ)

syntax
-assignmentd :: *svids* $\Rightarrow$ *uexprs* $\Rightarrow$ *logic* (**infixr** $:=_D$ *72*)

translations
-assignmentd xs vs => *CONST assigns-d* (*-mk-usubst* (*CONST id*) *xs vs*)
-assignmentd x v <= *CONST assigns-d* (*CONST subst-upd* (*CONST id*) *x v*)
-assignmentd x v <= *-assignmentd* (*-spvar x*) *v*
x,*y* $:=_D$ *u*,*v* <= *CONST assigns-d* (*CONST subst-upd* (*CONST subst-upd* (*CONST id*) (*CONST svar x*) *u*) (*CONST svar y*) *v*)

lemma *assigns-d-is-H1-H2* [*closure*]: $\langle\sigma\rangle_D$ *is* **H**
by (*simp add*: *assigns-d-def rdesign-is-H1-H2*)

lemma *assigns-d-H1-H3* [*closure*]: $\langle\sigma\rangle_D$ *is* **N**
by (*metis H1-rdesign H3-ndesign Healthy-def' aext-true assigns-d-def ndesign-def*)

Designs are closed under substitutions on state variables only (via lifting)

lemma *state-subst-H1-H2-closed* [*closure*]:
P *is* $\mathbf{H} \Longrightarrow \lceil \sigma \oplus_s \Sigma_D \rceil_s \dagger P$ *is* $\mathbf{H}$
by (*metis H1-H2-eq-rdesign Healthy-if rdesign-is-H1-H2 state-subst-design*)

lemma *assigns-d-ndes-def* [*ndes-simp*]:
$\langle \sigma \rangle_D = (true \vdash_n \langle \sigma \rangle_a)$
by (*rel-auto*)

lemma *assigns-d-id* [*simp*]: $\langle id \rangle_D = II_D$
by (*rel-auto*)

lemma *assign-d-left-comp*:
$(\langle f \rangle_D \;;;\; (P \vdash_r Q)) = (\lceil f \rceil_s \dagger P \vdash_r \lceil f \rceil_s \dagger Q)$
by (*simp add*: *assigns-d-def rdesign-composition assigns-r-comp subst-not*)

lemma *assign-d-right-comp*:
$((P \vdash_r Q) \;;;\; \langle f \rangle_D) = ((\neg ((\neg P) \;;;\; true)) \vdash_r (Q \;;;\; \langle f \rangle_a))$
by (*simp add*: *assigns-d-def rdesign-composition*)

lemma *assigns-d-comp*:
$(\langle f \rangle_D \;;;\; \langle g \rangle_D) = \langle g \circ f \rangle_D$
by (*simp add*: *assigns-d-def rdesign-composition assigns-comp*)

lemma *assigns-d-comp-ext*:
fixes P :: $'\alpha$ *hrel-des*
assumes P *is* $\mathbf{H}$
shows $(\langle \sigma \rangle_D \;;;\; P) = \lceil \sigma \oplus_s \Sigma_D \rceil_s \dagger P$
proof −
have $\langle \sigma \rangle_D \;;;\; P = \langle \sigma \rangle_D \;;;\; (pre_D(P) \vdash_r post_D(P))$
by (*metis H1-H2-commute H1-H2-is-rdesign H2-idem Healthy-def' assms*)
also have ... $= \lceil \sigma \rceil_s \dagger pre_D(P) \vdash_r \lceil \sigma \rceil_s \dagger post_D(P)$
by (*simp add*: *assign-d-left-comp*)
also have ... $= \lceil \sigma \oplus_s \Sigma_D \rceil_s \dagger (pre_D(P) \vdash_r post_D(P))$
by (*rel-auto*)
also have ... $= \lceil \sigma \oplus_s \Sigma_D \rceil_s \dagger P$
by (*metis H1-H2-commute H1-H2-is-rdesign H2-idem Healthy-def' assms*)
finally show *?thesis* **.**
qed

Normal designs are closed under substitutions on state variables only

lemma *state-subst-H1-H3-closed* [*closure*]:
P *is* $\mathbf{N} \Longrightarrow \lceil \sigma \oplus_s \Sigma_D \rceil_s \dagger P$ *is* $\mathbf{N}$
by (*metis H1-H2-eq-rdesign H1-H3-impl-H2 Healthy-if assign-d-left-comp assigns-d-H1-H3 seq-r-H1-H3-closed state-subst-design*)

lemma *H4-assigns-d*: $\langle \sigma \rangle_D$ *is H4*
proof −
have $(\langle \sigma \rangle_D \;;;\; (false \vdash_r true_h)) = (false \vdash_r true)$
by (*simp add*: *assigns-d-def rdesign-composition assigns-r-feasible*)
moreover have ... $= true$
by (*rel-auto*)
ultimately show *?thesis*
using *is-H4-alt-def* **by** *auto*
qed

4.2 Guarded Commands

definition *GrdCommD* :: $'\alpha$ *upred* $\Rightarrow$ ($'\alpha$, $'\beta$) *rel-des* $\Rightarrow$ ($'\alpha$, $'\beta$) *rel-des* (- $\to_D$ - [*85*, *86*] *85*) **where**
[*upred-defs*]: $b \to_D P = P \triangleleft b \triangleright_D \top_D$

lemma *GrdCommD-ndes-simp* [*ndes-simp*]:
  $b \to_D (p_1 \vdash_n P_2) = ((b \Rightarrow p_1) \vdash_n (\lceil b \rceil_< \wedge P_2))$
  by (*rel-auto*)

lemma *GrdCommD-H1-H3-closed* [*closure*]: P *is* $\mathbf{N} \Longrightarrow b \to_D P$ *is* $\mathbf{N}$
  by (*simp add*: *GrdCommD-def closure*)

lemma *GrdCommD-true* [*simp*]: *true* $\to_D P = P$
  by (*rel-auto*)

lemma *GrdCommD-false* [*simp*]: *false* $\to_D P = \top_D$
  by (*rel-auto*)

lemma *GrdCommD-abort* [*simp*]: $b \to_D$ *true* $= ((\neg\ b) \vdash_n$ *false*$)$
  by (*rel-auto*)

4.3 Alternation

consts
  ualtern :: $'a$ *set* $\Rightarrow$ ($'a \Rightarrow 'p$) $\Rightarrow$ ($'a \Rightarrow 'r$) $\Rightarrow 'r \Rightarrow 'r$
  ualtern-list :: ($'a \times 'r$) *list* $\Rightarrow 'r \Rightarrow 'r$

definition *AlternateD* :: $'a$ *set* $\Rightarrow$ ($'a \Rightarrow '\alpha$ *upred*) $\Rightarrow$ ($'a \Rightarrow$ ($'\alpha$, $'\beta$) *rel-des*) $\Rightarrow$ ($'\alpha$, $'\beta$) *rel-des* $\Rightarrow$ ($'\alpha$, $'\beta$) *rel-des* **where**
[*upred-defs*, *ndes-simp*]:
AlternateD $A\ g\ P\ Q = (\bigsqcap\ i{\in}A \cdot g(i) \to_D P(i)) \sqcap (\bigwedge\ i{\in}A \cdot \neg\ g(i)) \to_D Q$

This lemma shows that our generalised alternation is the same operator as Marcel Oliveira's definition of alternation when the else branch is abort.

lemma *AlternateD-abort-alternate*:
  assumes $\bigwedge\ i.\ P(i)$ *is* $\mathbf{N}$
  shows
  AlternateD $A\ g\ P\ \bot_D =$
  $((\bigvee\ i{\in}A \cdot g(i)) \wedge (\bigwedge\ i{\in}A \cdot g(i) \Rightarrow \lfloor pre_D(P\ i)\rfloor_<)) \vdash_n (\bigvee\ i{\in}A \cdot \lceil g(i)\rceil_< \wedge post_D(P\ i))$
proof (*cases* $A = \{\}$)
  case *False*
  have *AlternateD* $A\ g\ P\ \bot_D =$
    $(\bigsqcap\ i{\in}A \cdot g(i) \to_D (\lfloor pre_D(P\ i)\rfloor_< \vdash_n post_D(P\ i))) \sqcap (\bigwedge\ i{\in}A \cdot \neg\ g(i)) \to_D (false \vdash_n true)$
    by (*simp add*: *AlternateD-def ndesign-form bot-d-ndes-def assms*)
  also have ... $= ((\bigvee\ i{\in}A \cdot g(i)) \wedge (\bigwedge\ i{\in}A \cdot g(i) \Rightarrow \lfloor pre_D(P\ i)\rfloor_<)) \vdash_n (\bigvee\ i{\in}A \cdot \lceil g(i)\rceil_< \wedge post_D(P\ i))$
    by (*simp add*: *ndes-simp False*, *rel-auto*)
  finally show *?thesis* **by** *simp*
next
  case *True*
  thus *?thesis*
    by (*simp add*: *AlternateD-def*, *rel-auto*)
qed

definition *AlternateD-list* :: ($'\alpha$ *upred* $\times$ ($'\alpha$, $'\beta$) *rel-des*) *list* $\Rightarrow$ ($'\alpha$, $'\beta$) *rel-des* $\Rightarrow$ ($'\alpha$, $'\beta$) *rel-des*
where

[*upred-defs*, *ndes-simp*]:
AlternateD-list xs P =
  AlternateD {*0..<length xs*} (λ *i. map fst xs ! i*) (λ *i. map snd xs ! i*) *P*

adhoc-overloading
  ualtern AlternateD **and**
  ualtern-list AlternateD-list

nonterminal *gcomm* **and** *gcomms*

syntax
  -altind-els :: *pttrn* ⇒ *logic* ⇒ *logic* ⇒ *logic* ⇒ *logic* ⇒ *logic* (*if* -∈- • - → - *else* - *fi*)
  -altind :: *pttrn* ⇒ *logic* ⇒ *logic* ⇒ *logic* ⇒ *logic* (*if* -∈- • - → - *fi*)
  -gcomm :: *logic* ⇒ *logic* ⇒ *gcomm* (- → - [*65*, *66*] *65*)
  -gcomm-nil :: *gcomm* ⇒ *gcomms* (-)
  -gcomm-cons :: *gcomm* ⇒ *gcomms* ⇒ *gcomms* (- | - [*60*, *61*] *61*)
  -gcomm-show :: *logic* ⇒ *logic*
  -altgcomm-els :: *gcomms* ⇒ *logic* ⇒ *logic* (*if* - *else* - *fi*)
  -altgcomm :: *gcomms* ⇒ *logic* (*if* - *fi*)

translations
  -altind-els x A g P Q => *CONST ualtern A* (λ *x. g*) (λ *x. P*) *Q*
  -altind-els x A g P Q <= *CONST ualtern A* (λ *x. g*) (λ *x'. P*) *Q*
  -altind x A g P => *CONST ualtern A* (λ *x. g*) (λ *x. P*) (*CONST Orderings.top*)
  -altind x A g P <= *CONST ualtern A* (λ *x. g*) (λ *x'. P*) (*CONST Orderings.top*)
  -altgcomm cs => *CONST ualtern-list cs* (*CONST Orderings.top*)
  -altgcomm (*-gcomm-show cs*) <= *CONST ualtern-list cs* (*CONST Orderings.top*)
  -altgcomm-els cs P => *CONST ualtern-list cs P*
  -altgcomm-els (*-gcomm-show cs*) *P* <= *CONST ualtern-list cs P*

  -gcomm g P => (*g*, *P*)
  -gcomm g P <= *-gcomm-show* (*g*, *P*)
  -gcomm-cons c cs => *c* # *cs*
  -gcomm-cons (*-gcomm-show c*) (*-gcomm-show* (*d* # *cs*)) <= *-gcomm-show* (*c* # *d* # *cs*)
  -gcomm-nil c => [*c*]
  -gcomm-nil (*-gcomm-show c*) <= *-gcomm-show* [*c*]

lemma *AlternateD-H1-H3-closed* [*closure*]:
  assumes $\bigwedge$ *i. i* ∈ *A* ⟹ *P i is* **N** *Q is* **N**
  shows *if i*∈*A* • *g*(*i*) → *P*(*i*) *else Q fi is* **N**
proof (*cases A* = {})
  case *True*
  then show *?thesis*
    by (*simp add*: *AlternateD-def closure false-upred-def assms*)
next
  case *False*
  then show *?thesis*
    by (*simp add*: *AlternateD-def closure assms*)
qed

lemma *AltD-ndes-simp* [*ndes-simp*]:
  if i∈*A* • *g*(*i*) → ($P_1(i) \vdash_n P_2(i)$) *else* $Q_1 \vdash_n Q_2$ *fi*
   $= ((\bigwedge i \in A \cdot g\ i \Rightarrow P_1\ i) \wedge ((\bigwedge i \in A \cdot \neg\ g\ i) \Rightarrow Q_1)) \vdash_n$
     $((\bigvee i \in A \cdot \lceil g\ i \rceil_< \wedge P_2\ i) \vee (\bigwedge i \in A \cdot \neg\ \lceil g\ i \rceil_<) \wedge Q_2)$
proof (*cases A* = {})

case *True*
then show *?thesis* **by** (*simp add*: *AlternateD-def*)
next
case *False*
then show *?thesis*
by (*simp add*: *ndes-simp*, *rel-auto*)
qed

declare *UINF-upto-expand-first* [*ndes-simp*]
declare *UINF-Suc-shift* [*ndes-simp*]
declare *USUP-upto-expand-first* [*ndes-simp*]
declare *USUP-Suc-shift* [*ndes-simp*]
declare *true-upred-def* [*THEN sym*, *ndes-simp*]

lemma *AlternateD-mono-refine*:
assumes $\bigwedge$ *i*. $P\ i \sqsubseteq Q\ i\ R \sqsubseteq S$
shows ($if\ i{\in}A \cdot g(i) \rightarrow P(i)\ else\ R\ fi) \sqsubseteq (if\ i{\in}A \cdot g(i) \rightarrow Q(i)\ else\ S\ fi$)
using *assms* **by** (*rel-auto*, *meson*)

lemma *Monotonic-AlternateD* [*closure*]:
$[\![\bigwedge i.\ Monotonic\ (F\ i);\ Monotonic\ G\ ]\!] \Longrightarrow Monotonic\ (\lambda\ X.\ if\ i{\in}A \cdot g(i) \rightarrow F\ i\ X\ else\ G(X)\ fi)$
by (*rel-auto*, *meson*)

lemma *AlternateD-eq*:
assumes $A = B \bigwedge i.\ i{\in}A \Longrightarrow g(i) = h(i) \bigwedge i.\ i{\in}A \Longrightarrow P(i) = Q(i)\ R = S$
shows $if\ i{\in}A \cdot g(i) \rightarrow P(i)\ else\ R\ fi = if\ i{\in}B \cdot h(i) \rightarrow Q(i)\ else\ S\ fi$
by (*insert assms*, *rel-blast*)

lemma *AlternateD-empty*:
$if\ i{\in}\{\} \cdot g(i) \rightarrow P(i)\ else\ Q\ fi = Q$
by (*rel-auto*)

lemma *AlternateD-true-singleton*:
assumes *P is* **N**
shows $if\ true \rightarrow P\ fi = P$
by (*ndes-eq cls*: *assms*)

lemma *AlernateD-no-ind*:
assumes $A \neq \{\}$ *P is* **N** *Q is* **N**
shows $if\ i{\in}A \cdot b \rightarrow P\ else\ Q\ fi = if\ b \rightarrow P\ else\ Q\ fi$
by (*ndes-eq cls*: *assms*)

lemma *AlernateD-singleton*:
assumes *P k is* **N** *Q is* **N**
shows $if\ i{\in}\{k\} \cdot b(i) \rightarrow P(i)\ else\ Q\ fi = if\ b(k) \rightarrow P(k)\ else\ Q\ fi$ (**is** *?lhs* = *?rhs*)
proof −
have $?lhs = if\ i{\in}\{k\} \cdot b(k) \rightarrow P(k)\ else\ Q\ fi$
by (*auto intro*: *AlternateD-eq simp add*: *assms ndesign-form*)
also have ... = *?rhs*
by (*simp add*: *AlernateD-no-ind assms closure*)
finally show *?thesis* **.**
qed

lemma *AlternateD-commute*:
assumes *P is* **N** *Q is* **N**

shows $\mathit{if}\ g_1 \rightarrow P \mid g_2 \rightarrow Q\ \mathit{fi} = \mathit{if}\ g_2 \rightarrow Q \mid g_1 \rightarrow P\ \mathit{fi}$
by (*ndes-eq cls*:*assms*)

lemma *AlternateD-dcond*:
assumes P *is* **N** Q *is* **N**
shows $\mathit{if}\ g \rightarrow P\ \mathit{else}\ Q\ \mathit{fi} = P \triangleleft g \triangleright_D Q$
by (*ndes-eq cls*:*assms*)

lemma *AlternateD-cover*:
assumes P *is* **N** Q *is* **N**
shows $\mathit{if}\ g \rightarrow P\ \mathit{else}\ Q\ \mathit{fi} = \mathit{if}\ g \rightarrow P \mid (\neg\ g) \rightarrow Q\ \mathit{fi}$
by (*ndes-eq cls*: *assms*)

lemma *UINF-ndes-expand*:
assumes $\bigwedge\ i.\ i{\in}A \Longrightarrow P(i)$ *is* **N**
shows $(\bigsqcap\ i \in A \cdot \lfloor pre_D(P(i)) \rfloor_< \vdash_n post_D(P(i))) = (\bigsqcap\ i \in A \cdot P(i))$
by (*rule UINF-cong*, *simp add*: *assms ndesign-form*)

lemma *USUP-ndes-expand*:
assumes $\bigwedge\ i.\ i{\in}A \Longrightarrow P(i)$ *is* **N**
shows $(\bigsqcup\ i \in A \cdot \lfloor pre_D(P(i)) \rfloor_< \vdash_n post_D(P(i))) = (\bigsqcup\ i \in A \cdot P(i))$
by (*rule USUP-cong*, *simp add*: *assms ndesign-form*)

lemma *AlternateD-ndes-expand*:
assumes $\bigwedge\ i.\ i{\in}A \Longrightarrow P(i)$ *is* **N** Q *is* **N**
shows $\mathit{if}\ i{\in}A \cdot g(i) \rightarrow P(i)\ \mathit{else}\ Q\ \mathit{fi} =$
$\mathit{if}\ i{\in}A \cdot g(i) \rightarrow (\lfloor pre_D(P(i)) \rfloor_< \vdash_n post_D(P(i)))\ \mathit{else}\ \lfloor pre_D(Q) \rfloor_< \vdash_n post_D(Q)\ \mathit{fi}$
apply (*simp add*: *AlternateD-def*)
apply (*subst UINF-ndes-expand*[*THEN sym*])
apply (*simp add*: *assms closure*)
apply (*ndes-simp cls*: *assms*)
apply (*rel-auto*)
done

lemma *AlternateD-ndes-expand'*:
assumes $\bigwedge\ i.\ i{\in}A \Longrightarrow P(i)$ *is* **N**
shows $\mathit{if}\ i{\in}A \cdot g(i) \rightarrow P(i)\ \mathit{fi} = \mathit{if}\ i{\in}A \cdot g(i) \rightarrow (\lfloor pre_D(P(i)) \rfloor_< \vdash_n post_D(P(i)))\ \mathit{fi}$
apply (*simp add*: *AlternateD-def*)
apply (*subst UINF-ndes-expand*[*THEN sym*])
apply (*simp add*: *assms closure*)
apply (*ndes-simp cls*: *assms*)
apply (*rel-auto*)
done

lemma *ndesign-ind-form*:
assumes $\bigwedge\ i.\ P(i)$ *is* **N**
shows $(\lambda\ i.\ \lfloor pre_D(P(i)) \rfloor_< \vdash_n post_D(P(i))) = P$
by (*simp add*: *assms ndesign-form*)

lemma *AlternateD-insert*:
assumes $\bigwedge\ i.\ i{\in}(\mathit{insert}\ x\ A) \Longrightarrow P(i)$ *is* **N** Q *is* **N**
shows $\mathit{if}\ i{\in}(\mathit{insert}\ x\ A) \cdot g(i) \rightarrow P(i)\ \mathit{else}\ Q\ \mathit{fi} =$
$\mathit{if}\ g(x) \rightarrow P(x) \mid$
$(\bigvee\ i{\in}A \cdot g(i)) \rightarrow \mathit{if}\ i{\in}A \cdot g(i) \rightarrow P(i)\ \mathit{fi}$
$\mathit{else}\ Q$

fi (**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = *if* $i \in (insert\ x\ A) \cdot g(i) \to (\lfloor pre_D(P(i)) \rfloor_< \vdash_n post_D(P(i)))$ *else* $(\lfloor pre_D(Q) \rfloor_< \vdash_n post_D(Q))$ *fi*
using *AlternateD-ndes-expand assms(1) assms(2)* **by** *blast*
also
have ... =
if $g(x) \to (\lfloor pre_D(P(x)) \rfloor_< \vdash_n post_D(P(x)))$ |
$(\bigvee i \in A \cdot g(i)) \to$ *if* $i \in A \cdot g(i) \to \lfloor pre_D(P(i)) \rfloor_< \vdash_n post_D(P(i))$ *fi*
else $\lfloor pre_D(Q) \rfloor_< \vdash_n post_D(Q)$
fi
by (*ndes-simp cls*:*assms*, *rel-auto*)
also have ... = *?rhs*
by (*simp add*: *AlternateD-ndes-expand′ ndesign-form assms*)
finally show *?thesis* **.**
qed

4.4 Iteration

theorem *ndesign-iteration-wp* [*ndes-simp*]:
$(p \vdash_n Q)$;; $(p \vdash_n Q)$ ^ $n = ((\bigwedge i \in \{0..n\} \cdot (Q$ ^ $i)$ *wp* $p) \vdash_n Q$ ^ *Suc n*)
proof (*induct n*)
case *0*
then show *?case* **by** (*rel-auto*)
next
case (*Suc n*) **note** *hyp* = *this*
have $(p \vdash_n Q)$;; $(p \vdash_n Q)$ ^ *Suc n* = $(p \vdash_n Q)$;; $(p \vdash_n Q)$;; $(p \vdash_n Q)$ ^ *n*
by (*simp add*: *upred-semiring.power-Suc*)
also have ... = $(p \vdash_n Q)$;; $((\bigsqcup i \in \{0..n\} \cdot Q$ ^ i *wp* $p) \vdash_n Q$ ^ *Suc n*)
by (*simp add*: *hyp*)
also have ... = $(p \wedge Q$ *wp* $(\bigsqcup i \in \{0..n\} \cdot Q$ ^ i *wp* $p)) \vdash_n (Q$;; $Q)$;; Q ^ *n*
by (*simp add*: *upred-semiring.power-Suc ndesign-composition-wp seqr-assoc*)
also have ... = $(p \wedge (\bigsqcup i \in \{0..n\} \cdot Q$ ^ *Suc i wp* $p)) \vdash_n (Q$;; $Q)$;; Q ^ *n*
by (*simp add*: *upred-semiring.power-Suc wp*)
also have ... = $(p \wedge (\bigsqcup i \in \{0..n\}.\ Q$ ^ *Suc i wp* $p)) \vdash_n (Q$;; $Q)$;; Q ^ *n*
by (*simp add*: *USUP-as-Inf-image*)
also have ... = $(p \wedge (\bigsqcup i \in \{1..Suc\ n\}.\ Q$ ^ i *wp* $p)) \vdash_n (Q$;; $Q)$;; Q ^ *n*
by (*metis* (*no-types*, *lifting*) *One-nat-def image-Suc-atLeastAtMost image-cong image-image*)
also have ... = $(Q$ ^ 0 *wp* $p \wedge (\bigsqcup i \in \{1..Suc\ n\}.\ Q$ ^ i *wp* $p)) \vdash_n (Q$;; $Q)$;; Q ^ *n*
by (*simp add*: *wp*)
also have ... = $((\bigsqcup i \in \{0..Suc\ n\}.\ Q$ ^ i *wp* $p)) \vdash_n (Q$;; $Q)$;; Q ^ *n*
by (*simp add*: *Iic-Suc-eq-insert-0 atLeast0AtMost conj-upred-def image-Suc-atMost*)
also have ... = $(\bigsqcup i \in \{0..Suc\ n\} \cdot Q$ ^ i *wp* $p) \vdash_n Q$ ^ *Suc* (*Suc n*)
by (*simp add*: *upred-semiring.power-Suc USUP-as-Inf-image upred-semiring.mult-assoc*)
finally show *?case* **.**
qed

Overloadable Syntax

consts
uiterate :: $'a$ *set* $\Rightarrow ('a \Rightarrow 'p) \Rightarrow ('a \Rightarrow 'r) \Rightarrow 'r$
uiterate-list :: $('a \times 'r)$ *list* $\Rightarrow 'r$

syntax
-iterind :: *pttrn* ⇒ *logic* ⇒ *logic* ⇒ *logic* ⇒ *logic* (*do* -∈- · - → - *od*)
-itergcomm :: *gcomms* ⇒ *logic* (*do* - *od*)

translations
  -iterind x A g P => *CONST uiterate A* (λ *x. g*) (λ *x. P*)
  -iterind x A g P <= *CONST uiterate A* (λ *x. g*) (λ *x'. P*)
  -itergcomm cs => *CONST uiterate-list cs*
  -itergcomm (*-gcomm-show cs*) <= *CONST uiterate-list cs*

definition *IterateD* :: *'a set* ⇒ (*'a* ⇒ *'α upred*) ⇒ (*'a* ⇒ *'α hrel-des*) ⇒ *'α hrel-des* **where**
[*upred-defs*, *ndes-simp*]:
IterateD A g P = ($\boldsymbol{\mu}_{NDES}$ *X* · *if i∈A* · *g*(*i*) → *P*(*i*) ;; *X else* II_D *fi*)

definition *IterateD-list* :: (*'α upred* × *'α hrel-des*) *list* ⇒ *'α hrel-des* **where**
[*upred-defs*, *ndes-simp*]:
IterateD-list xs = *IterateD* {*0..<length xs*} (λ *i. fst* (*nth xs i*)) (λ *i. snd* (*nth xs i*))

adhoc-overloading
  uiterate IterateD **and**
  uiterate-list IterateD-list

lemma *IterateD-H1-H3-closed* [*closure*]:
  assumes ⋀ *i. i* ∈ *A* ⟹ *P i is* **N**
  shows *do i∈A* · *g*(*i*) → *P*(*i*) *od is* **N**
proof (*cases A* = {})
  case *True*
  then show *?thesis*
    by (*simp add*: *IterateD-def closure assms*)
next
  case *False*
  then show *?thesis*
    by (*simp add*: *IterateD-def closure assms*)
qed

lemma *IterateD-empty*:
  do i∈{} · *g*(*i*) → *P*(*i*) *od* = II_D
  by (*simp add*: *IterateD-def AlternateD-empty normal-design-theory-continuous.LFP-const skip-d-is-H1-H3*)

lemma *IterateD-list-single-expand*:
  do b → *P od* = ($\boldsymbol{\mu}_{NDES}$ *X* · *if b* → *P* ;; *X else* II_D *fi*)
oops

lemma *IterateD-singleton*:
  assumes *P is* **N**
  shows *do b* → *P od* = *do i∈*{*0*} · *b* → *P od*
  apply (*simp add*: *IterateD-list-def IterateD-def AlernateD-singleton assms*)
  apply (*subst AlernateD-singleton*)
  apply (*simp*)
  apply (*rel-auto*)
oops

lemma *IterateD-mono-refine*:
  assumes
    ⋀ *i. P i is* **N** ⋀ *i. Q i is* **N**
    ⋀ *i. P i* ⊑ *Q i*
  shows (*do i∈A* · *g*(*i*) → *P*(*i*) *od*) ⊑ (*do i∈A* · *g*(*i*) → *Q*(*i*) *od*)
  apply (*simp add*: *IterateD-def normal-design-theory-continuous.utp-lfp-def*)
  apply (*subst normal-design-theory-continuous.utp-lfp-def*)

apply (*simp-all add*: *closure assms*)
apply (*subst normal-design-theory-continuous.utp-lfp-def*)
apply (*simp-all add*: *closure assms*)
apply (*simp add*: *ndes-hcond-def*)
apply (*rule gfp-mono*)
apply (*rule AlternateD-mono-refine*)
apply (*simp-all add*: *closure seqr-mono assms*)
done

lemma *IterateD-single-refine*:
assumes
P *is* $\mathbf{N}$ Q *is* $\mathbf{N}$ $P \sqsubseteq Q$
shows $(do\ g \rightarrow P\ od) \sqsubseteq (do\ g \rightarrow Q\ od)$
oops

lemma *IterateD-refine-intro*:
fixes $V :: (nat,\ 'a)\ uexpr$
assumes *vwb-lens w*
shows
$I \vdash_n (w{:}[\lceil I \wedge \neg (\bigvee i{\in}A \cdot g(i))\rceil_{>}]) \sqsubseteq$
$do\ i{\in}A \cdot g(i) \rightarrow (I \wedge g(i)) \vdash_n (w{:}[\lceil I\rceil_{>} \wedge \lceil V\rceil_{>} <_u \lceil V\rceil_{<}])\ od$
proof $(cases\ A = \{\})$
case *True*
with *assms* **show** *?thesis*
by (*simp add*: *IterateD-empty*, *rel-auto*)
next
case *False*
then show *?thesis*
using *assms*
apply (*simp add*: *IterateD-def*)
apply (*rule ndesign-mu-wf-refine-intro*[**where** $e{=}V$ **and** $R{=}\{(x,\ y).\ x < y\}$])
apply (*simp-all add*: *wf closure*)
apply (*simp add*: *ndes-simp unrest*)
apply (*rule ndesign-refine-intro*)
apply (*rel-auto*)
apply (*rel-auto*)
apply (*metis mwb-lens.put-put vwb-lens-mwb*)
done
qed

lemma *IterateD-single-refine-intro*:
fixes $V :: (nat,\ 'a)\ uexpr$
assumes *vwb-lens w*
shows
$I \vdash_n (w{:}[\lceil I \wedge \neg\ g\rceil_{>}]) \sqsubseteq$
$do\ g \rightarrow ((I \wedge g) \vdash_n (w{:}[\lceil I\rceil_{>} \wedge \lceil V\rceil_{>} <_u \lceil V\rceil_{<}]))\ od$
apply (*rule order-trans*)
defer
apply (*rule IterateD-refine-intro*[*of* w $\{0\}$ $\lambda\ i.\ g\ I\ V$, *simplified*, *OF assms*(*1*)])
oops

4.5 Let and Local Variables

definition $LetD :: ('a,\ '\alpha)\ uexpr \Rightarrow ('a \Rightarrow '\alpha\ hrel\text{-}des) \Rightarrow '\alpha\ hrel\text{-}des$ **where**
[*upred-defs*]: $LetD\ v\ P = (P\ x)[\![x \rightarrow \lceil v\rceil_{D<}]\!]$

syntax
  -LetD :: [*letbinds*, *'a*] ⇒ *'a* ((*let*$_D$ (-)/ *in* (-)) [*0*, *10*] *10*)

translations
  -LetD (*-binds b bs*) *e* ⇌ *-LetD b* (*-LetD bs e*)
  $let_D\ x = a\ in\ e$ ⇌ *CONST LetD a* ($\lambda x.\ e$)

lemma *LetD-ndes-simp* [*ndes-simp*]:
  $LetD\ v\ (\lambda\ x.\ p(x) \vdash_n Q(x)) = (p(x)[\![x \to v]\!]) \vdash_n (Q(x)[\![x \to \lceil v \rceil_<]\!])$
  by (*rel-auto*)

lemma *LetD-H1-H3-closed* [*closure*]:
  $[\![\bigwedge x.\ P(x)\ is\ \mathbf{N}]\!] \Longrightarrow LetD\ v\ P\ is\ \mathbf{N}$
  by (*rel-auto*)

4.6 Deep Local Variables

definition *des-local-state* ::
  'a::countable itself ⇒ ((*nat*, *'s*) *local-scheme des*, *'s*, *nat*, *'a::countable*) *local-prim* **where**
  des-local-state t = (| *sstate* = Σ_D, *sassigns* = *assigns-d*, *inj-local* = *nat-inj-univ* |)

syntax
  -des-local-state-type :: *type* ⇒ *logic* ($\mathcal{L}_D$[-])
  -des-var-scope-type :: *id* ⇒ *type* ⇒ *logic* ⇒ *logic* (*var*$_D$ - :: - · - [*0*, *0*, *10*] *10*)

translations
  $\mathcal{L}_D['a]$ == *CONST des-local-state TYPE*(*'a*)
  -des-var-scope-type x t P => *-var-scope-type* (*-des-local-state-type t*) *x t P*
  $var_D\ x :: 'a \cdot P$ <= $var[\mathcal{L}_D['a]]\ x \cdot P$

lemma *get-rel-local* [*lens-defs*]:
  $get_{\mathbf{s}_{\mathcal{L}_D['a::countable]}} = get_{\Sigma_D}$
  by (*simp add*: *des-local-state-def*)

lemma *des-local-state* [*simp*]: *utp-local-state* $\mathcal{L}_D['a::countable]$
  by (*unfold-locales*, *simp-all add*: *upred-defs assigns-comp des-local-state-def*, *rel-auto*)
    (*metis local.cases-scheme*)

lemma *sassigns-des-state* [*simp*]: $\langle\sigma\rangle_{\mathcal{L}_D['a::countable]} = \langle\sigma\rangle_D$
  by (*simp add*: *des-local-state-def*)

lemma *des-var-open-H1-H3-closed* [*closure*]:
  $open[\mathcal{L}_D['a::countable]]\ is\ \mathbf{N}$
  by (*simp add*: *utp-local-state.var-open-def closure*)

lemma *des-var-close-H1-H3-closed* [*closure*]:
  $close[\mathcal{L}_D['a::countable]]\ is\ \mathbf{N}$
  by (*simp add*: *utp-local-state.var-close-def closure*)

lemma *unrest-ok-vtop-des* [*unrest*]: $ok \sharp top[\mathcal{L}_D['a::countable]]$
  by (*simp add*: *utp-local-state.top-var-def*, *simp add*: *des-local-state-def unrest*)

lemma *msubst-H1-H3-closed* [*closure*]:
  $[\![\$ok \sharp v;\ out\alpha \sharp v;\ (\bigwedge x.\ P\ x\ is\ \mathbf{N})]\!] \Longrightarrow (P(x)[\![x \to v]\!])\ is\ \mathbf{N}$
  by (*rel-auto*, *metis+*)

lemma *var-block-H1-H3-closed* [*closure*]:
($\bigwedge x.\ P\ x\ is\ \mathbf{N}$) $\Longrightarrow$ $\mathcal{V}[\mathcal{L}_D['a{::}countable],\ P]\ is\ \mathbf{N}$
by (*simp add*: *utp-local-state.var-scope-def closure unrest*)

lemma *inj-local-rel* [*simp*]: $inj\text{-}local\ R_l = \mathcal{U}_{\mathbb{N}}$
by (*simp add*: *rel-local-state-def*)

lemma *sstate-rel* [*simp*]: $\mathbf{s}_{R_l} = 1_L$
by (*simp add*: *rel-local-state-def*)

lemma *inj-local-des* [*simp*]:
$inj\text{-}local\ \mathcal{L}_D['a{::}countable] = \mathcal{U}_{\mathbb{N}}$
by (*simp add*: *des-local-state-def*)

lemma *sstate-des* [*simp*]: $\mathbf{s}_{\mathcal{L}_D['a{::}countable]} = \Sigma_D$
by (*simp add*: *des-local-state-def*)

lemma *ndesign-msubst-top* [*usubst*]:
$(p\ x \vdash_n Q\ x)[\![x \rightarrow \lceil top[\mathcal{L}_D['a{::}countable]]\rceil_<]\!] = ((p\ x)[\![x \rightarrow top[R_l['a]]]\!] \vdash_n (Q\ x)[\![x \rightarrow \lceil top[R_l['a]]\rceil_<]\!])$
by (*rel-auto'*)

First attempt at a law for expanding design variable blocks. Far from adequate at the moment though.

lemma *ndesign-local-expand-1* [*ndes-simp*]:
$(var_D\ x :: 'a :: countable \cdot p(x) \vdash_n Q(x)) =$
$(\bigsqcup\ v \cdot (p\ x)[\![x \rightarrow top[R_l]]\!][\![\&store\ \hat{}_u\ \langle \ll v \gg \rangle / store]\!]) \vdash_n$
$(\bigsqcap\ v \cdot store := \&store\ \hat{}_u\ \langle \ll v \gg \rangle\ ;;\ (Q\ x)[\![x \rightarrow \lceil top[R_l]\rceil_<]\!]\ ;;\ store := (front_u(\&store) \triangleleft 0 <_u \#_u(\&store) \triangleright \&store))$
apply (*simp add*: *utp-local-state.var-scope-def utp-local-state.var-open-def utp-local-state.var-close-def seq-UINF-distr' usubst*)
apply (*simp add*: *ndes-simp wp unrest*)
apply (*rel-auto'*)
done

end

5 Design Weakest Preconditions

theory *utp-des-wp*
imports *utp-des-prog*
begin

definition *wp-design* :: $('\alpha, '\beta)\ rel\text{-}des \Rightarrow '\beta\ cond \Rightarrow '\alpha\ cond$ (**infix** wp_D *60*) **where**
[*upred-defs*]: $Q\ wp_D\ r = (\lfloor pre_D(Q)\ ;;\ true :: ('\alpha, '\beta)\ urel \rfloor_< \wedge (post_D(Q)\ wp\ r))$

If two normal designs have the same weakest precondition for any given postcondition, then the two designs are equivalent.

theorem *wpd-eq-intro*: $[\![\bigwedge r.\ (p_1 \vdash_n Q_1)\ wp_D\ r = (p_2 \vdash_n Q_2)\ wp_D\ r]\!] \Longrightarrow (p_1 \vdash_n Q_1) = (p_2 \vdash_n Q_2)$
apply (*rel-simp robust*; *metis curry-conv*)
done

theorem *wpd-H3-eq-intro*: $[\![P\ is\ H1\text{-}H3;\ Q\ is\ H1\text{-}H3;\ \bigwedge r.\ P\ wp_D\ r = Q\ wp_D\ r]\!] \Longrightarrow P = Q$
by (*metis H1-H3-commute H1-H3-is-normal-design H3-idem Healthy-def' wpd-eq-intro*)

lemma *wp-assigns-d* [*wp*]: $\langle\sigma\rangle_D$ wp_D $r = \sigma \dagger r$
  by (*rel-auto*)

theorem *rdesign-wp* [*wp*]:
  $(\lceil p \rceil_< \vdash_r Q)$ wp_D $r = (p \wedge Q$ *wp* $r)$
  by (*rel-auto*)

theorem *ndesign-wp* [*wp*]:
  $(p \vdash_n Q)$ wp_D $r = (p \wedge Q$ *wp* $r)$
  by (*simp add*: *ndesign-def rdesign-wp*)

theorem *wpd-seq-r*:
  fixes *Q1 Q2* :: *'α hrel*
  shows $((\lceil p1 \rceil_< \vdash_r Q1)$;; $(\lceil p2 \rceil_< \vdash_r Q2))$ wp_D $r = (\lceil p1 \rceil_< \vdash_r Q1)$ wp_D $((\lceil p2 \rceil_< \vdash_r Q2)$ wp_D $r)$
  apply (*simp add*: *wp*)
  apply (*subst rdesign-composition-wp*)
  apply (*simp only*: *wp*)
  apply (*rel-auto*)
done

theorem *wpnd-seq-r* [*wp*]:
  fixes *Q1 Q2* :: *'α hrel*
  shows $((p1 \vdash_n Q1)$;; $(p2 \vdash_n Q2))$ wp_D $r = (p1 \vdash_n Q1)$ wp_D $((p2 \vdash_n Q2)$ wp_D $r)$
  by (*simp add*: *ndesign-def wpd-seq-r*)

theorem *wpd-seq-r-H1-H3* [*wp*]:
  fixes *P Q* :: *'α hrel-des*
  assumes *P is* **N** *Q is* **N**
  shows $(P$;; $Q)$ wp_D $r = P$ wp_D $(Q$ wp_D $r)$
  by (*metis H1-H3-commute H1-H3-is-normal-design H1-idem Healthy-def' assms*(*1*) *assms*(*2*) *wpnd-seq-r*)

end

6 Refinement Calculus

theory *utp-des-refcalc*
  imports *utp-des-prog*
begin

definition *des-spec* :: $('a \Longrightarrow 'α) \Rightarrow$ *'α upred* $\Rightarrow$ $('α \Rightarrow$ *'α upred*$)$ $\Rightarrow$ *'α hrel-des* **where**
[*upred-defs*]: *des-spec x p q* $= (\bigsqcup\ v \cdot ((p \wedge \&\mathbf{v} =_u \ll v \gg) \vdash_n x{:}[\lceil q(v) \rceil_>]))$

syntax
  -init-var :: *logic*
  -des-spec :: *salpha* ⇒ *logic* ⇒ *logic* ⇒ *logic* $(\text{-:[-,/ -]}_D$ $[99,0,0]$ $100)$
  -des-log-const :: *pttrn* ⇒ *logic* ⇒ *logic* $(con_D$ - $\cdot$ - $[0,\ 10]$ $10)$

translations
  -des-spec x p q => *CONST des-spec x p* (λ *-init-var*. *q*)
  -des-spec (*-salphaset* (*-salphamk x*)) *p q* <= *CONST des-spec x p* (λ *iv*. *q*)
  -des-log-const x P => $\bigsqcup$ *x* · *P*

parse-translation ⟪
let

fun init-var-tr [] = *Syntax.free iv*
  | *init-var-tr* - = *raise Match*;
in
[(@{*syntax-const -init-var*}, *K init-var-tr*)]
end
⟫

abbreviation $choose_D\ x \equiv \{\&x\}{:}[true,true]_D$

lemma *des-spec-simple-def*:
  $x{:}[pre,post]_D = (pre \vdash_n x{:}[\lceil post \rceil_>])$
  by (*rel-auto*)

lemma *des-spec-abort*:
  $x{:}[false,post]_D = \bot_D$
  by (*rel-auto*)

lemma *des-spec-skip*: $\emptyset{:}[true,true]_D = II_D$
  by (*rel-auto*)

lemma *des-spec-strengthen-post*:
  assumes ‘$post' \Rightarrow post$‘
  shows $w{:}[pre,\ post]_D \sqsubseteq w{:}[pre,\ post']_D$
  using *assms* **by** (*rel-auto*)

lemma *des-spec-weaken-pre*:
  assumes ‘$pre \Rightarrow pre'$‘
  shows $w{:}[pre,\ post]_D \sqsubseteq w{:}[pre',\ post]_D$
  using *assms* **by** (*rel-auto*)

lemma *des-spec-refine-skip*:
  assumes *vwb-lens w* ‘$pre \Rightarrow post$‘
  shows $w{:}[pre,\ post]_D \sqsubseteq II_D$
  using *assms* **by** (*rel-auto*)

lemma *rc-iter*:
  fixes V :: (*nat*, *'a*) *uexpr*
  assumes *vwb-lens w*
  shows $w{:}[ivr,\ ivr \wedge \neg\ (\bigvee\ i{\in}A \cdot g(i))]_D$
    $\sqsubseteq (do\ i{\in}A \cdot g(i) \rightarrow \bigsqcup\ iv \cdot w{:}[ivr \wedge g(i) \wedge \ll iv \gg =_u \&\mathbf{v},\ ivr \wedge (V <_u V[\![\ll iv \gg/\mathbf{v}]\!])]_D\ od)$ (**is** *?lhs* ⊑ *?rhs*)
  apply (*rule order-trans*)
   defer
   apply (*simp add*: *des-spec-simple-def*)
   apply (*rule IterateD-refine-intro*[*of - - - - V*])
   apply (*simp add*: *assms*)
  apply (*rule IterateD-mono-refine*)
    apply (*simp-all add*: *ndes-simp closure*)
   apply (*rel-auto*)
  using *assms*
  apply (*rel-auto*)
  done

end

7 Theory of Invariants

theory *utp-des-invariants*
  imports *utp-des-theory*
begin

The theory of invariants formalises operation and state invariants based on the theory of designs. For more information, please see the associated paper [1, Section 4].

7.1 Operation Invariants

definition $OIH(\psi)(D) = (D \wedge (\$ok \wedge \neg D^f \Rightarrow \psi))$

declare *OIH-def* [*upred-defs*]

lemma *OIH-design*:
  assumes *D is H1-H2*
  shows $OIH(\psi)(D) = ((\neg D^f) \vdash (D^t \wedge \psi))$
proof −
  have $OIH(\psi)(D) = (((\neg D^f) \vdash D^t) \wedge (\$ok \wedge \neg D^f \Rightarrow \psi))$
    by (*metis H1-H2-commute H1-H2-is-design H1-idem Healthy-def′ OIH-def assms*)
  also have ... $= ((\$ok \wedge \neg D^f \Rightarrow \$ok' \wedge D^t) \wedge (\$ok \wedge \neg D^f \Rightarrow \psi))$
    by (*simp add*: *design-def*)
  also have ... $= ((\neg D^f) \vdash (D^t \wedge \psi))$
    by (*pred-auto*)
  finally show *?thesis* **.**
qed

lemma *OIH-idem*:
  assumes *D is H1-H2* $\$ok' \sharp \psi$
  shows $OIH(\psi)(OIH(\psi)(D)) = OIH(\psi)(D)$
  using *assms*
  by (*simp add*: *OIH-design design-is-H1-H2 unrest*) (*simp add*: *design-def usubst*, *rel-auto*)

lemma *OIH-of-design*:
  $\$ok' \sharp P \Longrightarrow OIH(\psi)(P \vdash Q) = (P \vdash (Q \wedge \psi))$
  by (*simp add*: *OIH-def design-def usubst*, *rel-auto*)

7.2 State Invariants

definition $ISH(\psi)(D) = (D \vee (\$ok \wedge \neg D^f \wedge \lceil\psi\rceil_< \Rightarrow \$ok' \wedge D^t))$

declare *ISH-def* [*upred-defs*]

lemma *ISH-design*: $ISH(\psi)(D) = (\neg D^f \wedge \lceil\psi\rceil_<) \vdash D^t$
  by (*rel-auto*, *metis+*)

lemma *ISH-idem*: $ISH(\psi)(ISH(\psi)(D)) = ISH(\psi)(D)$
  by (*simp add*: *ISH-design usubst design-def*, *pred-auto*)

lemma *ISH-of-design*:
  $[\![\$ok' \sharp P; \$ok' \sharp Q]\!] \Longrightarrow ISH(\psi)(P \vdash Q) = ((P \wedge \lceil\psi\rceil_<) \vdash Q)$
  by (*simp add*: *ISH-design design-def usubst*, *pred-auto*)

definition $OSH(\psi)(D) = (D \wedge (\$ok \wedge \neg D^f \wedge \lceil\psi\rceil_< \Rightarrow \lceil\psi\rceil_>))$

declare *OSH-def* [*upred-defs*]

lemma *OSH-as-OIH*:
$OSH(\psi)(D) = OIH(\lceil\psi\rceil_< \Rightarrow \lceil\psi\rceil_>)(D)$
by (*simp add*: *OSH-def OIH-def*, *pred-auto*)

lemma *OSH-design*:
assumes *D is H1-H2*
shows $OSH(\psi)(D) = ((\neg D^f) \vdash (D^t \wedge (\lceil\psi\rceil_< \Rightarrow \lceil\psi\rceil_>)))$
by (*simp add*: *OSH-as-OIH OIH-design assms*)

lemma *OSH-of-design*:
$\llbracket \$ok' \sharp P; \$ok' \sharp Q \rrbracket \Longrightarrow OSH(\psi)(P \vdash Q) = (P \vdash (Q \wedge (\lceil\psi\rceil_< \Rightarrow \lceil\psi\rceil_>)))$
by (*simp add*: *OSH-design design-is-H1-H2 unrest*, *simp add*: *design-def usubst*, *pred-auto*)

definition $SIH(\psi) = ISH(\psi) \circ OSH(\psi)$

declare *SIH-def* [*upred-defs*]

lemma *SIH-of-design*:
$\llbracket \$ok' \sharp P; \$ok' \sharp Q; ok \sharp \psi \rrbracket \Longrightarrow SIH(\psi)(P \vdash Q) = ((P \wedge \lceil\psi\rceil_<) \vdash (Q \wedge \lceil\psi\rceil_>))$
by (*simp add*: *SIH-def OSH-of-design ISH-of-design unrest*, *pred-auto*)

end

8 Meta Theory for UTP Designs

theory *utp-designs*
imports
utp-des-core
utp-des-healths
utp-des-theory
utp-des-tactics
utp-des-prog
utp-des-wp
utp-des-refcalc
utp-des-invariants
begin end

References

[1] A. Cavalcanti, A. Wellings, and J. Woodcock. The Safety-Critical Java memory model formalised. *Formal Aspects of Computing*, 25(1):37–57, 2012.

[2] A. Cavalcanti and J. Woodcock. A tutorial introduction to designs in unifying theories of programming. In *Proc. 4th Intl. Conf. on Integrated Formal Methods (IFM)*, volume 2999 of *LNCS*, pages 40–66. Springer, 2004.

[3] T. Hoare and J. He. *Unifying Theories of Programming.* Prentice-Hall, 1998.