

Theory of Designs in Isabelle/UTP

Simon Foster

Yakoub Nemouchi

Frank Zeyda

October 5, 2018

Abstract

This document describes a mechanisation of the UTP theory of designs in Isabelle/UTP. Designs enrich UTP relations with explicit precondition/postcondition pairs, as present in formal notations like VDM, B, and the refinement calculus. If a program's precondition holds, then it is guaranteed to terminate and establish its postcondition, which is an approach known as total correctness. If the precondition does not hold, the behaviour is maximally nondeterministic, which represents unspecified behaviour. In this mechanisation, we create the theory of designs, including its alphabet, signature, and healthiness conditions. We then use these to prove the key algebraic laws of programming. This development can be used to support program verification based on total correctness.

Contents

1	Design Signature and Core Laws	2
1.1	Definitions	2
1.2	Lifting, Unrestriction, and Substitution	4
1.3	Basic Design Laws	6
1.4	Sequential Composition Laws	7
1.5	Preconditions and Postconditions	10
1.6	Distribution Laws	11
1.7	Refinement Introduction	12
2	Design Healthiness Conditions	13
2.1	H1: No observation is allowed before initiation	13
2.2	H2: A specification cannot require non-termination	16
2.3	Designs as $H1$ - $H2$ predicates	19
2.4	H3: The design assumption is a precondition	22
2.5	Normal Designs as $H1$ - $H3$ predicates	24
2.6	H4: Feasibility	27
2.7	UTP theory of Designs	27
2.8	UTP theories	27
2.9	Galois Connection	28
2.10	Fixed Points	30
3	Design Proof Tactics	32

4	Imperative Programming in Designs	33
4.1	Assignment	33
4.2	Guarded Commands	34
4.3	Alternation	34
4.4	Iteration	38
4.5	Let and Local Variables	41
4.6	Design Hoare Logic	41
5	Design Weakest Preconditions	42
6	Refinement Calculus	43
7	Theory of Invariants	45
7.1	Operation Invariants	45
7.2	State Invariants	45
8	Meta Theory for UTP Designs	46

1 Design Signature and Core Laws

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

UTP designs [2, 4] are a subset of the alphabetised relations that use a boolean observational variable *ok* to record the start and termination of a program. For more information on designs please see Chapter 3 of the UTP book [4], 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

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

translations

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

notation *des-vars-child-lens* (Σ_D)

syntax

-svid-des-alpha :: *svid* (\mathbf{v}_D)

translations

-svid-des-alpha \Rightarrow *CONST des-vars-child-lens*

lemma *ok-des-bij-lens*: *bij-lens* (*ok* $+_L \Sigma_D$)

by (*unfold-locales*, *simp-all add: ok-def des-vars-child-lens-def lens-plus-def prod.case-eq-if*)

Define the lens functor for designs

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

lemma *lmap-des-vars*: *vwb-lens* *f* \Rightarrow *vwb-lens* (*lmap-des-vars* *f*)

by (*unfold-locales*, *auto simp add: lens-defs*)

lemma *lmap-id*: *lmap*_{*D*} *1*_{*L*} = *1*_{*L*}

by (*simp add: lens-defs fun-eq-iff*)

lemma *lmap-comp*: *lmap*_{*D*} (*f* ;_{*L*} *g*) = *lmap*_{*D*} *f* ;_{*L*} *lmap*_{*D*} *g*

by (*simp add: lens-defs fun-eq-iff*)

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

abbreviation *lift-desr* ($\lceil \cdot \rceil_D$)

where $\lceil P \rceil_D \equiv P \oplus_p (\Sigma_D \times_L \Sigma_D)$

abbreviation *lift-pre-desr* ($\lceil \cdot \rceil_{D<}$)

where $\lceil p \rceil_{D<} \equiv \lceil \lceil p \rceil \rceil_{D<}$

abbreviation *lift-post-desr* ($\lceil \cdot \rceil_{D>}$)

where $\lceil p \rceil_{D>} \equiv \lceil \lceil p \rceil \rceil_{D>}$

abbreviation *drop-desr* ($\lfloor \cdot \rfloor_D$)

where $\lfloor P \rfloor_D \equiv P \upharpoonright_e (\Sigma_D \times_L \Sigma_D)$

abbreviation *dcond* :: $(''\alpha, ''\beta) \text{ rel-des} \Rightarrow '\alpha \text{ upred} \Rightarrow (''\alpha, ''\beta) \text{ rel-des} \Rightarrow (''\alpha, ''\beta) \text{ rel-des}$

where *dcond* *P b Q* $\equiv P \triangleleft \lceil b \rceil_{D<} \triangleright Q$

syntax $-dcond :: logic \Rightarrow uexp \Rightarrow logic \Rightarrow logic \ ((\beta - \triangleleft - \triangleright_D / -) \ [52,0,53] \ 52)$
translations $-dcond \ P \ b \ Q == CONST \ dcond \ P \ b \ Q$

definition $design :: ('\alpha, '\beta) \ rel-des \Rightarrow ('\alpha, '\beta) \ rel-des \Rightarrow ('\alpha, '\beta) \ rel-des \ (\mathbf{infixl} \vdash \ 59) \ \mathbf{where}$
 $[upred-defs]: P \vdash Q = (\$ok \wedge P \Rightarrow \$ok' \wedge Q)$

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

definition $rdesign :: ('\alpha, '\beta) \ urel \Rightarrow ('\alpha, '\beta) \ urel \Rightarrow ('\alpha, '\beta) \ rel-des \ (\mathbf{infixl} \vdash_r \ 59) \ \mathbf{where}$
 $[upred-defs]: (P \vdash_r Q) = [\![P]\!]_D \vdash [\![Q]\!]_D$

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

definition $ndesign :: '\alpha \ cond \Rightarrow ('\alpha, '\beta) \ urel \Rightarrow ('\alpha, '\beta) \ rel-des \ (\mathbf{infixl} \vdash_n \ 59) \ \mathbf{where}$
 $[upred-defs]: (p \vdash_n Q) = ([\![p]\!]_{<} \vdash_r Q)$

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

definition $bot-d :: ('\alpha, '\beta) \ rel-des \ (\perp_D) \ \mathbf{where}$
 $[upred-defs]: \perp_D = (false \vdash false)$

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

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

syntax

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

translations

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

1.2 Lifting, Unrestriction, and Substitution

lemma $drop-desr-inv \ [simp]: [\![P]\!]_D = P$
by $(simp \ add: \ prod-mwb-lens)$

lemma $lift-desr-inv:$

fixes $P :: ('\alpha, '\beta) \ rel-des$
assumes $\$ok \ \# \ P \ \$ok' \ \# \ P$
shows $[\![P]\!]_D = P$

proof $-$

have $bij-lens \ (\Sigma_D \times_L \Sigma_D +_L \ (in-var \ ok +_L \ out-var \ ok)) :: (-, '\alpha \ des-vars-scheme \times '\beta \ des-vars-scheme)$
 $lens)$

(is $bij-lens \ (?P))$

proof $-$

have $?P \approx_L \ (ok +_L \ \Sigma_D) \times_L \ (ok +_L \ \Sigma_D) \ (\mathbf{is} \ ?P \approx_L \ ?Q)$

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*]: $out\alpha \# p \implies out\alpha \# [p]_D$
by (*pred-simp*)

lemma *lift-dist-seq* [*simp*]:
 $[P ;; Q]_D = ([P]_D ;; [Q]_D)$
by (*rel-auto*)

lemma *lift-des-skip-dr-unit* [*simp*]:
 $([P]_D ;; [II]_D) = [P]_D$
 $([II]_D ;; [P]_D) = [P]_D$
by (*rel-auto*)+

lemma *lift-des-skip-dr-unit-unrest*: $\$ok' \# P \implies (P ;; [II]_D) = P$
by (*rel-auto*)

lemma *state-subst-design* [*usubst*]:
 $[\sigma \oplus_s \Sigma_D]_s \dagger (P \vdash_r Q) = ([\sigma]_s \dagger P) \vdash_r ([\sigma]_s \dagger Q)$
by (*rel-auto*)

lemma *design-subst* [*usubst*]:
 $\llbracket \$ok \# \sigma; \$ok' \# \sigma \rrbracket \implies \sigma \dagger (P \vdash Q) = (\sigma \dagger P) \vdash (\sigma \dagger Q)$
by (*simp add: design-def usubst*)

lemma *design-msubst* [*usubst*]:
 $(P(x) \vdash Q(x)) \llbracket x \rightarrow v \rrbracket = (P(x) \llbracket x \rightarrow v \rrbracket \vdash Q(x) \llbracket x \rightarrow v \rrbracket)$
by (*rel-auto*)

lemma *design-ok-false* [*usubst*]: $(P \vdash Q) \llbracket false / \$ok \rrbracket = true$
by (*simp add: design-def usubst*)

lemma *ok-pre*: $(\$ok \wedge [pre_D(P)]_D) = (\$ok \wedge (\neg P^f))$
by (*pred-auto robust*)

lemma *ok-post*: $(\$ok \wedge [post_D(P)]_D) = (\$ok \wedge (P^t))$
by (*pred-auto robust*)

1.3 Basic Design Laws

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

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

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

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

lemma *design-ok-pre-conj*: $(\$ok \wedge P) \vdash Q = P \vdash Q$
by (*rel-auto*)

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

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

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

lemma *bot-d-ndes-def* [*ndes-simp*]: $\perp_D = (false \vdash_n true)$
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*:
 $(P \llbracket true/\$ok \rrbracket \vdash Q \llbracket true/\$ok \rrbracket) = (P \vdash Q)$
by (*rel-auto*)

lemma *design-subst-ok-ok'*:
 $(P \llbracket true/\$ok \rrbracket \vdash Q \llbracket true, true/\$ok, \$ok' \rrbracket) = (P \vdash Q)$

proof –

have $(P \vdash Q) = ((\$ok \wedge P) \vdash (\$ok \wedge \$ok' \wedge Q))$

by (*pred-auto*)

also have $\dots = ((\$ok \wedge P[\![true/\$ok]\!]) \vdash (\$ok \wedge (\$ok' \wedge Q[\![true/\$ok']\!])[\![true/\$ok]\!]))$

by (*metis conj-eq-out-var-subst conj-pos-var-subst upred-eq-true utp-pred-laws.inf-commute ok-vwb-lens*)

also have $\dots = ((\$ok \wedge P[\![true/\$ok]\!]) \vdash (\$ok \wedge \$ok' \wedge Q[\![true,true/\$ok,\$ok']\!]))$

by (*simp add: usubst*)

also have $\dots = (P[\![true/\$ok]\!] \vdash Q[\![true,true/\$ok,\$ok']\!])$

by (*pred-auto*)

finally show *?thesis* ..

qed

lemma *design-subst-ok'*:

$(P \vdash Q[\![true/\$ok']\!]) = (P \vdash Q)$

proof –

have $(P \vdash Q) = (P \vdash (\$ok' \wedge Q))$

by (*pred-auto*)

also have $\dots = (P \vdash (\$ok' \wedge Q[\![true/\$ok']\!]))$

by (*metis conj-eq-out-var-subst upred-eq-true utp-pred-laws.inf-commute ok-vwb-lens*)

also have $\dots = (P \vdash Q[\![true/\$ok']\!])$

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

$\$ok' \# P1 \ \$ok \# P2$

shows $((P1 \vdash Q1) ;; (P2 \vdash Q2)) =$

$((\neg (\neg P1) ;; true) \wedge \neg (Q1[\![true/\$ok']\!] ;; (\neg P2))) \vdash (Q1[\![true/\$ok']\!] ;; Q2[\![true/\$ok]\!]))$

proof –

have $((P1 \vdash Q1) ;; (P2 \vdash Q2)) = (\exists \ ok_0 \cdot ((P1 \vdash Q1)[\![\ll ok_0 \gg / \$ok']\!] ;; (P2 \vdash Q2)[\![\ll ok_0 \gg / \$ok]\!]))$

by (*rule seqr-middle, simp*)

also have \dots

$= (((P1 \vdash Q1)[\![false/\$ok']\!] ;; (P2 \vdash Q2)[\![false/\$ok]\!])$

$\vee ((P1 \vdash Q1)[\![true/\$ok']\!] ;; (P2 \vdash Q2)[\![true/\$ok]\!]))$

by (*metis (no-types, lifting) calculation disj-comm ok-vwb-lens seqr-bool-split*)

also from *assms*

have $\dots = (((\$ok \wedge P1 \Rightarrow Q1[\![true/\$ok']\!]) ;; (P2 \Rightarrow \$ok' \wedge Q2[\![true/\$ok]\!])) \vee ((\neg (\$ok \wedge P1)) ;;$

$true))$

by (*simp add: design-def usubst unrest, pred-auto*)

also have $\dots = ((\neg \$ok ;; true_h) \vee ((\neg P1) ;; true) \vee (Q1[\![true/\$ok']\!] ;; (\neg P2)) \vee (\$ok' \wedge (Q1[\![true/\$ok']\]$

$;; Q2[\![true/\$ok]\!]))$

by (*rel-auto*)

also have $\dots = (((\neg (\neg P1) ;; true) \wedge \neg (Q1[\![true/\$ok']\!] ;; (\neg P2))) \vdash (Q1[\![true/\$ok']\!] ;; Q2[\![true/\$ok]\!]))$

by (*simp add: precondition-right-unit design-def unrest, rel-auto*)

finally show *?thesis* .

qed

theorem *design-composition*:

assumes

$\$ok' \# P1 \ \$ok \# P2 \ \$ok' \# Q1 \ \$ok \# Q2$
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

$\$ok' \# P1 \ \$ok \# P2 \ ok \#\# Q1 \ ok \#\# Q2$

shows $((P1 \vdash Q1) ;; (P2 \vdash Q2)) = (((\neg ((\neg P1) ;; true)) \wedge \neg (Q1^t ;; (\neg P2))) \vdash (Q1 ;; Q2))$

proof –

have $(\$ok \wedge \$ok' \wedge (Q1^t ;; Q2 \llbracket true/\$ok \rrbracket)) = (\$ok \wedge \$ok' \wedge (Q1 ;; Q2))$

proof –

have $(\$ok \wedge \$ok' \wedge (Q1 ;; Q2)) = ((\$ok \wedge Q1) ;; (Q2 \wedge \$ok'))$

by (*metis (no-types, lifting) conj-comm segr-post-var-out segr-pre-var-out*)

also have $\dots = ((Q1 \wedge \$ok') ;; (\$ok \wedge Q2))$

by (*simp add: assms(3) assms(4) runrest-ident-var*)

also have $\dots = (Q1^t ;; Q2 \llbracket true/\$ok \rrbracket)$

by (*metis ok-vwb-lens segr-pre-transfer segr-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 $(\neg (\neg P1 ;; true) \wedge \neg (Q1^t ;; (\neg P2))) \vdash (Q1^t ;; Q2 \llbracket true/\$ok \rrbracket) =$

$(\neg (\neg P1 ;; true) \wedge \neg (Q1^t ;; (\neg P2))) \vdash (\$ok \wedge \$ok' \wedge (Q1^t ;; Q2 \llbracket true/\$ok \rrbracket))$

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

$out\alpha \# p1 \ \$ok \# P2 \ \$ok' \# Q1 \ \$ok \# Q2$

shows $((p1 \vdash Q1) ;; (P2 \vdash Q2)) = ((p1 \wedge \neg (Q1 ;; (\neg P2))) \vdash (Q1 ;; Q2))$

using *assms*

by (*simp add: design-composition unrest precondition-right-unit*)

theorem *rdesign-composition-cond:*

assumes $out\alpha \# p1$

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

$ok \# p1 \ ok \# p2$

$\$ok \# Q1 \ \$ok' \# Q1 \ \$ok \# Q2 \ \$ok' \# Q2$

shows $((\llbracket p1 \rrbracket_{<} \vdash Q1) ;; (\llbracket p2 \rrbracket_{<} \vdash Q2)) = ((\llbracket p1 \wedge Q1 \ wp \ p2 \rrbracket_{<} \vdash (Q1 ;; Q2))$

using *assms* **by** (*rel-blast*)

theorem *rdesign-composition-wp:*

$((\llbracket p1 \rrbracket_{<} \vdash_r Q1) ;; (\llbracket p2 \rrbracket_{<} \vdash_r Q2)) = ((\llbracket p1 \wedge Q1 \ wp \ p2 \rrbracket_{<} \vdash_r (Q1 ;; Q2))$

by (*rel-blast*)

theorem *ndesign-composition-wp* [*ndes-simp*]:

$((p1 \vdash_n Q1) ;; (p2 \vdash_n Q2)) = ((p1 \wedge Q1 \text{ wp } p2) \vdash_n (Q1 ;; Q2))$
by (*rel-blast*)

theorem *design-true-left-zero*: $(true ;; (P \vdash Q)) = true$

proof –

have $(true ;; (P \vdash Q)) = ((true \llbracket false/\$ok' \rrbracket ;; (P \vdash Q) \llbracket false/\$ok \rrbracket) \vee (true \llbracket true/\$ok' \rrbracket ;; (P \vdash Q) \llbracket true/\$ok \rrbracket))$

by (*rel-auto*)

also have $\dots = ((true \llbracket false/\$ok' \rrbracket ;; true_h) \vee (true ;; ((P \vdash Q) \llbracket true/\$ok \rrbracket)))$

by (*subst-tac, rel-auto*)

also have $\dots = true$

by (*subst-tac, simp add: precondition-right-unit unrest*)

finally show *?thesis* .

qed

theorem *design-left-unit-hom*:

fixes $P Q :: 'a \text{ hrel-des}$

shows $(II_D ;; (P \vdash_r Q)) = (P \vdash_r Q)$

proof –

have $(II_D ;; (P \vdash_r Q)) = ((true \vdash_r II) ;; (P \vdash_r Q))$

by (*simp add: skip-d-def*)

also have $\dots = (true \wedge \neg (II ;; (\neg P))) \vdash_r (II ;; Q)$

proof –

have $out\alpha \nVdash true$

by *unrest-tac*

thus *?thesis*

using *rdesign-composition-cond* **by** *blast*

qed

also have $\dots = (\neg (\neg P)) \vdash_r Q$

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*:

$(P \vdash_r Q) ;; II_D = ((\neg (\neg P) ;; true) \vdash_r Q)$

by (*simp add: skip-d-def rdesign-composition*)

theorem *design-right-cond-unit* [*simp*]:

assumes $out\alpha \nVdash p$

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*: $(\perp_D ;; (P \vdash Q)) = \perp_D$

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]*: $pre_D(p \vdash_n Q) = [p]_<$
by (*pred-auto*)

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

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

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

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

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

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

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

1.6 Distribution Laws

theorem *design-choice*:

$$(P_1 \vdash P_2) \sqcap (Q_1 \vdash Q_2) = ((P_1 \wedge Q_1) \vdash (P_2 \vee Q_2))$$

by (*rel-auto*)

theorem *rdesign-choice*:

$$(P_1 \vdash_r P_2) \sqcap (Q_1 \vdash_r Q_2) = ((P_1 \wedge Q_1) \vdash_r (P_2 \vee Q_2))$$

by (*rel-auto*)

theorem *ndesign-choice* [*ndes-simp*]:

$$(p_1 \vdash_n P_2) \sqcap (q_1 \vdash_n Q_2) = ((p_1 \wedge q_1) \vdash_n (P_2 \vee Q_2))$$

by (*rel-auto*)

theorem *ndesign-choice'* [*ndes-simp*]:

$$((p_1 \vdash_n P_2) \vee (q_1 \vdash_n Q_2)) = ((p_1 \wedge q_1) \vdash_n (P_2 \vee Q_2))$$

by (*rel-auto*)

theorem *design-inf*:

$$(P_1 \vdash P_2) \sqcup (Q_1 \vdash Q_2) = ((P_1 \vee Q_1) \vdash ((P_1 \Rightarrow P_2) \wedge (Q_1 \Rightarrow Q_2)))$$

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*]:

$$(p_1 \vdash_n P_2) \sqcup (q_1 \vdash_n Q_2) = ((p_1 \vee q_1) \vdash_n (([p_1]_{<} \Rightarrow P_2) \wedge ([q_1]_{<} \Rightarrow Q_2)))$$

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-UNIF-mem*:

$$\begin{aligned} &\text{assumes } A \neq \{\} \\ &\text{shows } (\prod i \in A \cdot P(i) \vdash Q(i)) = (\prod i \in A \cdot P(i)) \vdash (\prod i \in A \cdot Q(i)) \\ &\text{using } \textit{assms} \text{ by } (\textit{rel-auto}) \end{aligned}$$

lemma *ndesign-UNIF-mem* [*ndes-simp*]:

$$\begin{aligned} &\text{assumes } A \neq \{\} \\ &\text{shows } (\prod i \in A \cdot p(i) \vdash_n Q(i)) = (\prod i \in A \cdot p(i)) \vdash_n (\prod i \in A \cdot Q(i)) \\ &\text{using } \textit{assms} \text{ by } (\textit{rel-auto}) \end{aligned}$$

lemma *ndesign-UNIF-ind* [*ndes-simp*]:

$$(\prod i \cdot p(i) \vdash_n Q(i)) = (\prod i \cdot p(i)) \vdash_n (\prod i \cdot Q(i))$$

by (*rel-auto*)

lemma *design-USUP-mem*:

$$(\prod i \in A \cdot P(i) \vdash Q(i)) = (\prod i \in A \cdot P(i)) \vdash (\prod i \in A \cdot P(i) \Rightarrow Q(i))$$

by (*rel-auto*)

lemma *ndesign-USUP-mem* [*ndes-simp*]:

$(\bigsqcup i \in A \cdot p(i) \vdash_n Q(i)) = (\prod i \in A \cdot p(i)) \vdash_n (\bigsqcup i \in A \cdot \lceil p(i) \rceil_{<} \Rightarrow Q(i))$
by (*rel-auto*)

lemma *ndesign-USUP-ind* [*ndes-simp*]:

$(\bigsqcup i \cdot p(i) \vdash_n Q(i)) = (\prod i \cdot p(i)) \vdash_n (\bigsqcup i \cdot \lceil p(i) \rceil_{<} \Rightarrow Q(i))$
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
 $\$ok \# P1 \ \$ok' \# P1 \ \$ok \# P2 \ \$ok' \# P2$
 $\$ok \# Q1 \ \$ok' \# Q1 \ \$ok \# Q2 \ \$ok' \# Q2$
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** $\dots = '(P2 \Rightarrow \$ok' \wedge Q2) \Rightarrow (P1 \Rightarrow \$ok' \wedge Q1)'$
by (*subst subst-bool-split[of in-var ok], simp-all, subst-tac*)

also with *assms* **have** $\dots = '(\neg P2 \Rightarrow \neg P1) \wedge ((P2 \Rightarrow Q2) \Rightarrow P1 \Rightarrow Q1)'$
by (*subst subst-bool-split[of out-var ok], simp-all, subst-tac*)

also have $\dots \longleftrightarrow '(P1 \Rightarrow P2)' \wedge 'P1 \wedge Q2 \Rightarrow Q1'$
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 $P2 \sqsubseteq P1$ $Q1 \sqsubseteq (P1 \wedge Q2)$
shows $P1 \vdash Q1 \sqsubseteq P2 \vdash Q2$
using *assms* *design-refine-intro*[*of P1 P2 Q2 Q1*] **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-refinement*:

$p1 \vdash_n Q1 \sqsubseteq p2 \vdash_n Q2 \iff (p1 \Rightarrow p2' \wedge '[p1]_< \wedge Q2 \Rightarrow Q1')$
by (*simp add: ndesign-def rdesign-def design-refinement unrest, rel-auto*)

lemma *ndesign-refine-intro*:

assumes ' $p1 \Rightarrow p2'$ ' ' $[p1]_< \wedge Q2 \Rightarrow Q1'$ '
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*:

$\perp_D \sqsubseteq (P \vdash Q)$
by (*rel-auto*)

lemma *design-refine-thms*:

assumes $P \sqsubseteq Q$
shows ' $\text{pre}_D(P) \Rightarrow \text{pre}_D(Q)$ ' ' $\text{pre}_D(P) \wedge \text{post}_D(Q) \Rightarrow \text{post}_D(P)$ '
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* :: $('α, 'β) \text{rel-des} \Rightarrow ('α, 'β) \text{rel-des}$ **where**

$[upred-defs]: H1(P) = (\$ok \Rightarrow P)$

lemma *H1-idem*:

$H1(H1 P) = H1(P)$
by (*pred-auto*)

lemma *H1-monotone*:

$P \sqsubseteq Q \implies H1(P) \sqsubseteq H1(Q)$
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 $\dots = (H1(II) ;; R)$

by (simp add: H1-design-skip)

also have $\dots = (\$ok \Rightarrow II) ;; R$

by (simp add: H1-def)

also have $\dots = (((\neg \$ok) ;; R) \vee R)$

by (simp add: impl-alt-def seqr-or-distl)

also have $\dots = (((\neg \$ok) ;; true_h) ;; R) \vee R$

by (simp add: precondition-right-unit unrest)

also have $\dots = (((\neg \$ok) ;; true_h) \vee R)$

by (metis assms(1) seqr-assoc)

also have $\dots = (\$ok \Rightarrow R)$

by (simp add: impl-alt-def precondition-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 $\dots = (true ;; (\neg \$ok \vee P))$ (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 precondition-left-zero unrest*)
 finally show *?thesis*
 by (*simp add: upred-defs urel-defs*)
 qed

theorem *H1-left-unit:*

fixes $P :: 'a \text{ hrel-des}$

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: precondition-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 precondition-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 $P :: 'a \text{ hrel-des}$

assumes P is *H1*

shows $((\neg \$ok) ;; P) = (\neg \$ok)$

proof –

have $((\neg \$ok) ;; P) = ((\neg \$ok) ;; true_h) ;; P$
 by (*simp add: precondition-right-unit unrest*)
 also have ... = $((\neg \$ok) ;; true_h)$
 by (*metis H1-left-zero assms seqr-assoc*)
 also have ... = $(\neg \$ok)$
 by (*simp add: precondition-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]:*

$\llbracket P \text{ is } H1; Q \text{ is } H1 \rrbracket \implies P \sqcap Q \text{ is } H1$

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*]] \implies *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 $A \neq \{\} \ \forall P \in A. P \text{ is } H1$
shows $(\bigsqcap A) \text{ is } H1$

proof –

from *assms*(2) have $H1 \text{ ' } A = A$
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 $\forall P \in A. P \text{ is } H1$
shows $(\bigsqcup A) \text{ is } H1$

proof –

from *assms* have $H1 \text{ ' } A = A$
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* :: ' α 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 $\dots = (P ;; ((\$ok \Rightarrow \$ok' \wedge \$ok') \wedge \lceil II \rceil_D))$
by (rel-auto)
also have $\dots = ((P ;; (\neg \$ok \wedge \lceil II \rceil_D)) \vee (P ;; (\$ok \wedge (\lceil II \rceil_D \wedge \$ok'))))$
by (rel-auto)
also have $\dots = (P^f \vee (P^t \wedge \$ok'))$

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 $\dots = (\exists \$ok' \cdot P \wedge \$ok' =_u \text{false})$
by *(rel-auto)*
also have $\dots = P^f$
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 $\dots = (P^t \wedge \$ok')$
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*:
 $P \text{ is } H2 \iff 'P^f \Rightarrow P^t'$
proof –
have $'P \Leftrightarrow (P ;; J)' \iff 'P \Leftrightarrow (P^f \vee (P^t \wedge \$ok'))'$
by *(simp add: J-split)*
also have $\dots \iff '(P \Leftrightarrow P^f \vee P^t \wedge \$ok')^f \wedge (P \Leftrightarrow P^f \vee P^t \wedge \$ok')^t'$
by *(simp add: subst-bool-split)*
also have $\dots = '(P^f \Leftrightarrow P^f) \wedge (P^t \Leftrightarrow P^f \vee P^t)'$
by *subst-tac*
also have $\dots = 'P^t \Leftrightarrow (P^f \vee P^t)'$
by *(pred-auto robust)*
also have $\dots = '(P^f \Rightarrow P^t)'$
by *(pred-auto)*
finally show *?thesis*
by *(metis H2-def Healthy-def' taut-iff-eq)*
qed

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

lemma *H2-design*:
assumes $\$ok' \nmid P \ \$ok' \nmid Q$
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 segr-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 $\dots = (\neg \$ok \vee (\neg \$ok) \wedge \$ok')$
 by (subst-tac)
 also have $\dots = (\neg \$ok)$
 by (pred-auto)
 finally show ?thesis .
qed

lemma H2-true: $H2(true) = true$
 by (rel-auto)

lemma H2-choice-closed [closure]:
 $\llbracket P \text{ is } H2; Q \text{ is } H2 \rrbracket \implies P \sqcap Q \text{ is } H2$
 by (metis H2-def Healthy-def' disj-upred-def segr-or-distl)

lemma H2-inf-closed [closure]:
 assumes $P \text{ is } H2 \ Q \text{ is } H2$
 shows $P \sqcup Q \text{ is } H2$
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 $H2(\dots) = \dots$
 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 $\dots = ((\neg \$ok \vee P) ;; J)$
 by (rel-auto)
 also have $\dots = (((\neg \$ok) ;; J) \vee (P ;; J))$

using *segr-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 $H1\text{-}H2 :: ('\alpha, '\beta) \text{rel-des} \Rightarrow ('\alpha, '\beta) \text{rel-des} \text{ (H)}$ where
 $H1\text{-}H2\ P \equiv H1\ (H2\ P)$

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

theorem $H1\text{-}H2\text{-eq-design}$:

$\mathbf{H}(P) = (\neg P^f) \vdash P^t$

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 $H1\text{-}H2\text{-is-design}$:

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 $H1\text{-}H2\text{-eq-rdesign}$:

$\mathbf{H}(P) = \text{pre}_D(P) \vdash_r \text{post}_D(P)$

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 ... = $(\$ok \wedge [\text{pre}_D(P)]_D \Rightarrow \$ok' \wedge \$ok \wedge [\text{post}_D(P)]_D)$
 by (*simp add: ok-post ok-pre*)
 also have ... = $(\$ok \wedge [\text{pre}_D(P)]_D \Rightarrow \$ok' \wedge [\text{post}_D(P)]_D)$
 by (*pred-auto*)
 also have ... = $\text{pre}_D(P) \vdash_r \text{post}_D(P)$
 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 \mathbf{H} Q is \mathbf{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 \mathbf{H} Q is \mathbf{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*: $\mathbf{H} (\mathbf{H} P) = \mathbf{H} P$

by (*simp add: H1-H2-commute H1-idem H2-idem*)

lemma *H1-H2-Idempotent [closure]*: *Idempotent* \mathbf{H}

by (*simp add: Idempotent-def H1-H2-idempotent*)

lemma *H1-H2-monotonic [closure]*: *Monotonic* \mathbf{H}

by (*simp add: H1-monotone H2-def mono-def seqr-mono*)

lemma *H1-H2-Continuous [closure]*: *Continuous* \mathbf{H}

by (*simp add: Continuous-comp H1-Continuous H1-H2-comp H2-Continuous*)

lemma *design-is-H1-H2 [closure]*:

$\llbracket \$ok' \# P; \$ok' \# Q \rrbracket \Longrightarrow (P \vdash Q) \text{ is } \mathbf{H}$

by (*simp add: H1-design H2-design Healthy-def'*)

lemma *rdesign-is-H1-H2 [closure]*:

$(P \vdash_r Q) \text{ is } \mathbf{H}$

by (*simp add: Healthy-def H1-rdesign H2-rdesign*)

lemma *top-d-is-H1-H2 [closure]*: $\top_D \text{ is } \mathbf{H}$

by (*simp add: H1-def H2-not-okay Healthy-intro impl-alt-def*)

lemma *bot-d-is-H1-H2 [closure]*: $\perp_D \text{ is } \mathbf{H}$

by (*simp add: bot-d-def closure unrest*)

lemma *seq-r-H1-H2-closed [closure]*:

assumes P is \mathbf{H} Q is \mathbf{H}

shows $(P ;; Q) \text{ is } \mathbf{H}$

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 $((P_1 \vdash_r P_2) ;; (Q_1 \vdash_r Q_2)) \text{ is } \mathbf{H}$

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 $(\sqcap A)$ is *H1-H2*

proof –

from *assms* **have** $A: A = H1-H2 \text{ ' } A$

by (*auto simp add: Healthy-def rev-image-eqI*)

also have $(\sqcap ...) = (\sqcap P \in A \cdot H1-H2(P))$

by (*simp add: UINF-as-Sup-collect*)

also have $\dots = (\sqcap P \in A \cdot (\neg P^f) \vdash P^t)$

by (*meson H1-H2-eq-design*)

also have $\dots = (\sqcup P \in A \cdot \neg P^f) \vdash (\sqcap P \in A \cdot P^t)$

by (*simp add: design-UINF-mem assms*)

also have \dots is *H1-H2*

by (*simp add: design-is-H1-H2 unrest*)

finally show *?thesis* .

qed

definition *design-inf* :: (α, β) *rel-des set* $\Rightarrow (\alpha, \beta)$ *rel-des* $(\sqcap_D \text{ [900] 900})$ **where**

$\sqcap_D A = (\text{if } (A = \{\}) \text{ then } \top_D \text{ else } \sqcap A)$

abbreviation *design-sup* :: (α, β) *rel-des set* $\Rightarrow (\alpha, \beta)$ *rel-des* $(\sqcup_D \text{ [900] 900})$ **where**

$\sqcup_D A \equiv \sqcup A$

lemma *design-inf-H1-H2-closed*:

assumes $\forall P \in A. P$ is **H**

shows $(\sqcap_D A)$ 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]: $\sqcap_D \{\} = \top_D$

by (*simp add: design-inf-def*)

lemma *design-sup-non-empty* [simp]: $A \neq \{\} \Rightarrow \sqcap_D A = \sqcap A$

by (*simp add: design-inf-def*)

lemma *USUP-mem-H1-H2-closed*:

assumes $\bigwedge i. i \in A \Rightarrow P \text{ } i$ is **H**

shows $(\sqcup i \in A \cdot P \text{ } i)$ is **H**

proof –

from *assms* **have** $(\sqcup i \in A \cdot P \text{ } i) = (\sqcup i \in A \cdot \mathbf{H}(P \text{ } i))$

by (*auto intro: USUP-cong simp add: Healthy-def*)

also have $\dots = (\sqcup i \in A \cdot (\neg (P \text{ } i)^f) \vdash (P \text{ } i)^t)$

by (*meson H1-H2-eq-design*)

also have $\dots = (\sqcap i \in A \cdot \neg (P \text{ } i)^f) \vdash (\sqcup i \in A \cdot \neg (P \text{ } i)^f \Rightarrow (P \text{ } i)^t)$

by (*simp add: design-USUP-mem*)

also have \dots is **H**

by (*simp add: design-is-H1-H2 unrest*)

finally show *?thesis* .

qed

lemma *USUP-ind-H1-H2-closed*:

assumes $\bigwedge i. P \text{ } i$ is **H**

shows $(\sqcup i \cdot P \text{ } i)$ is **H**

using *assms USUP-mem-H1-H2-closed*[of *UNIV P*] by *simp*

lemma *Inf-H1-H2-closed*:

assumes $\forall P \in A. P \text{ is } \mathbf{H}$

shows $(\bigsqcup A) \text{ is } \mathbf{H}$

proof –

from *assms* have $A: A = \mathbf{H} \cdot A$

by (*auto simp add: Healthy-def rev-image-eqI*)

also have $(\bigsqcup \dots) = (\bigsqcup P \in A \cdot \mathbf{H}(P))$

by (*simp add: USUP-as-Inf-collect*)

also have $\dots = (\bigsqcup P \in A \cdot (\neg P^f) \vdash P^t)$

by (*meson H1-H2-eq-design*)

also have $\dots = (\prod P \in A \cdot \neg P^f) \vdash (\bigsqcup P \in A \cdot \neg P^f \Rightarrow P^t)$

by (*simp add: design-USUP-mem*)

also have $\dots \text{ is } \mathbf{H}$

by (*simp add: design-is-H1-H2 unrest*)

finally show *?thesis* .

qed

lemma *rdesign-ref-monos*:

assumes $P \text{ is } \mathbf{H} \ Q \text{ is } \mathbf{H} \ P \sqsubseteq Q$

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 $H3 :: ('\alpha, '\beta) \text{ rel-des} \Rightarrow ('\alpha, '\beta) \text{ rel-des}$ **where**

[*upred-defs*]: $H3(P) \equiv P ;; \Pi_D$

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 $out\alpha \nVdash p$

shows $(p \vdash Q) \text{ is } H3$

proof –

have $((p \vdash Q) ;; \Pi_D) = (\neg((\neg p) ;; true)) \vdash (Q^t ;; \Pi[true/ok])$

by (simp add: skip-d-alt-def design-composition-subst unrest assms)
 also have ... = $p \vdash (Q^t ;; II \llbracket true/\$ok \rrbracket)$
 using assms precondition-equiv segr-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 segr-true-lemma
 also have ... $\longleftrightarrow P = (P ;; true)$
 by (simp add: segr-true-lemma)
 finally show ?thesis .

qed

theorem design-H3-iff-pre:

assumes $\$ok \# P \ \$ok' \# P \ \$ok \# Q \ \$ok' \# Q$

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 ... = $((\text{true} \Rightarrow \text{true}') \wedge \lceil II \rceil_D) \llbracket \text{false} / \text{true}' \rrbracket \;; (\text{true} \vdash II) \llbracket \text{false} / \text{true} \rrbracket$
 $\vee ((\text{true} \Rightarrow \text{true}') \wedge \lceil II \rceil_D) \llbracket \text{true} / \text{true}' \rrbracket \;; (\text{true} \vdash II) \llbracket \text{true} / \text{true} \rrbracket$
by (*rel-auto*)
also have ... = $((\neg \text{true} \wedge \lceil II \rceil_D \;; \text{true}) \vee (\lceil II \rceil_D \;; \text{true}' \wedge \lceil II \rceil_D))$
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 segr-assoc skip-d-absorb-J-1*)

lemma *H3-H2-absorb*:
 $H3 (H2 P) = H3 P$
by (*metis H2-def H3-def segr-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 $\text{true} \# p \text{ out} \alpha \# p \text{ true} \# Q \text{ true}' \# Q$
shows $H3(p \vdash Q) = p \vdash Q$
using *assms*
by (*metis Healthy-def' design-H3-iff-pre precondition-right-unit unrest-out α -var ok-vwb-lens vwb-lens-mwb*)

theorem *H3-rdesign-pre*:
assumes $\text{out} \alpha \# p$
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 $H1$ - $H3$ predicates

A normal design [3] refers only to initial state variables in the precondition.

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

lemma *H1-H3-comp*: $H1\text{-}H3 = H1 \circ H3$
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: $\mathbf{N} (\mathbf{N} P) = \mathbf{N} P$

by (simp add: $H1$ - $H3$ -commute $H1$ -idem $H3$ -idem)

lemma $H1$ - $H3$ -Idempotent [closure]: Idempotent \mathbf{N}

by (simp add: Idempotent-def $H1$ - $H3$ -idempotent)

lemma $H1$ - $H3$ -monotonic [closure]: Monotonic \mathbf{N}

by (simp add: $H1$ -monotone $H3$ -mono mono-def)

lemma $H1$ - $H3$ -Continuous [closure]: Continuous \mathbf{N}

by (simp add: Continuous-comp $H1$ -Continuous $H1$ - $H3$ -comp $H3$ -Continuous)

lemma $H1$ - $H3$ -intro:

assumes P is \mathbf{H} $out\alpha \nmid pre_D(P)$
shows P is \mathbf{N}
by (metis $H1$ - $H2$ -eq-rdesign $H1$ -rdesign $H3$ -rdesign-pre $Healthy$ -def' $assms$)

lemma $H1$ - $H3$ -impl- $H2$ [closure]: P is $\mathbf{N} \implies P$ 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 $out\alpha \nmid P^f$

proof –

have $P = (\neg (P^f ;; true)) \vdash P^t$

by (metis $H1$ - $H3$ -eq-design $Healthy$ -def $assms$)

also have $out\alpha \nmid (...)^f$

by (simp add: design-def usubst unrest, rel-auto)

finally show ?thesis .

qed

lemma $H3$ -unrest-out-alpha [unrest]: P is $\mathbf{N} \implies out\alpha \nmid pre_D(P)$

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 $\mathbf{N} \implies (\lfloor \text{pre}_D(P) \rfloor_{<} \vdash_n \text{post}_D(P)) = P$
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]: \perp_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 $(P \mathrel{;;} Q)$ 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 $P \mathrel{;;} \top_D = \lfloor \text{pre}_D P \rfloor_{<} \vdash_n \text{false}$
proof –
have $P \mathrel{;;} \top_D = (\lfloor \text{pre}_D(P) \rfloor_{<} \vdash_n \text{post}_D(P)) \mathrel{;;} (\text{true} \vdash_n \text{false})$
by (*simp add: assms ndesign-form ndesign-miracle*)
also have $\dots = \lfloor \text{pre}_D P \rfloor_{<} \vdash_n \text{false}$
by (*simp add: ndesign-composition-wp wp alpha*)
finally show ?thesis .
qed

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

lemma *USUP-ind-H1-H3-closed* [closure]:

$\llbracket \bigwedge i. P \ i \text{ is } \mathbf{N} \rrbracket \implies (\bigsqcup i \cdot P \ i) \text{ is } \mathbf{N}$

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 $H_4 :: ('\alpha, '\beta) \text{ rel-des} \Rightarrow (''\alpha, ''\beta) \text{ rel-des}$ **where**

[upred-defs]: $H_4(P) = ((P;;\text{true}) \Rightarrow P)$

theorem *H4-idem*:

$H_4(H_4(P)) = H_4(P)$

by (*rel-auto*)

lemma *is-H4-alt-def*:

$P \text{ is } H_4 \iff (P;;\text{true}) = \text{true}$

by (*rel-blast*)

end

2.7 UTP theory of Designs

theory *utp-des-theory*

imports *utp-des-healths*

begin

2.8 UTP theories

typedec *DES*

typedec *NDES*

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

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

overloading

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

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

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

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

begin

definition $\text{des-hcond} :: (DES, '\alpha \text{ des}) \text{ uthy} \Rightarrow (''\alpha \text{ des} \times ''\alpha \text{ des}) \text{ health}$ **where**

[upred-defs]: $\text{des-hcond } t = H1\text{-}H2$

definition $\text{des-unit} :: (DES, '\alpha \text{ des}) \text{ uthy} \Rightarrow '\alpha \text{ hrel-des}$ **where**

[upred-defs]: $\text{des-unit } t = II_D$

definition $\text{ndes-hcond} :: (NDES, '\alpha \text{ des}) \text{ uthy} \Rightarrow (''\alpha \text{ des} \times ''\alpha \text{ des}) \text{ health}$ **where**

[upred-defs]: $\text{ndes-hcond } t = H1\text{-}H3$

definition $\text{ndes-unit} :: (NDES, '\alpha \text{ des}) \text{ uthy} \Rightarrow '\alpha \text{ hrel-des}$ **where**

[upred-defs]: $\text{ndes-unit } t = II_D$

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 $\bigwedge P. P \in \text{carrier } (\text{uthy-order } DES) \longleftrightarrow P \text{ is } \mathbf{H}$
 and $\text{carrier } (\text{uthy-order } DES) \rightarrow \text{carrier } (\text{uthy-order } DES) \equiv \llbracket \mathbf{H} \rrbracket_H \rightarrow \llbracket \mathbf{H} \rrbracket_H$
 and $\llbracket \mathcal{H}_{DES} \rrbracket_H \rightarrow \llbracket \mathcal{H}_{DES} \rrbracket_H \equiv \llbracket \mathbf{H} \rrbracket_H \rightarrow \llbracket \mathbf{H} \rrbracket_H$
 and $le (\text{uthy-order } DES) = (\sqsubseteq)$
 and $eq (\text{uthy-order } DES) = (=)$
 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 $\bigwedge P. P \in \text{carrier } (\text{uthy-order } NDES) \longleftrightarrow P \text{ is } \mathbf{N}$
 and $\text{carrier } (\text{uthy-order } NDES) \rightarrow \text{carrier } (\text{uthy-order } NDES) \equiv \llbracket \mathbf{N} \rrbracket_H \rightarrow \llbracket \mathbf{N} \rrbracket_H$
 and $\llbracket \mathcal{H}_{NDES} \rrbracket_H \rightarrow \llbracket \mathcal{H}_{NDES} \rrbracket_H \equiv \llbracket \mathbf{N} \rrbracket_H \rightarrow \llbracket \mathbf{N} \rrbracket_H$
 and $le (\text{uthy-order } NDES) = (\sqsubseteq)$
 and $A \subseteq \text{carrier } (\text{uthy-order } NDES) \longleftrightarrow A \subseteq \llbracket \mathbf{N} \rrbracket_H$
 and $eq (\text{uthy-order } NDES) = (=)$
 by (unfold-locales, simp-all add: ndes-hcond-def H1-H3-Continuous utp-order-def)

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

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

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

lemma *ndesign-lat-bottom*: $\perp_{NDES} = \mathbf{N}(\text{true})$
 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 COM-PASS deliverable D23.5.

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

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

$\bigwedge x. x \in carrier \mathcal{X}_{DES \leftarrow \langle Des, Rel \rangle \rightarrow REL} = (x \text{ is } \mathbf{H}) \text{ and}$

$\bigwedge x. x \in carrier \mathcal{Y}_{DES \leftarrow \langle Des, Rel \rangle \rightarrow REL} = True \text{ and}$

$\pi^*_{DES \leftarrow \langle Des, Rel \rangle \rightarrow REL} = Des \text{ and}$

$\pi^*_{DES \leftarrow \langle Des, Rel \rangle \rightarrow REL} = Rel \text{ and}$

$le \mathcal{X}_{DES \leftarrow \langle Des, Rel \rangle \rightarrow REL} = (\sqsubseteq) \text{ and}$

$le \mathcal{Y}_{DES \leftarrow \langle Des, Rel \rangle \rightarrow REL} = (\sqsubseteq)$

proof (*unfold-locales, simp-all add: rel-hcond-def des-hcond-def*)

show $\bigwedge x. x \text{ is } id$

by (*simp add: Healthy-def*)

next

show $Rel \in \llbracket \mathbf{H} \rrbracket_H \rightarrow \llbracket id \rrbracket_H$

by (*auto simp add: Rel-def rel-hcond-def Healthy-def*)

next

show $Des \in \llbracket id \rrbracket_H \rightarrow \llbracket \mathbf{H} \rrbracket_H$

by (*auto simp add: Des-def des-hcond-def Healthy-def H1-H2-commute H1-idem H2-idem*)

next

fix $R :: 'a \text{ hrel}$

show $R \sqsubseteq Rel (Des R)$

by (*simp add: Des-design Rel-design*)

next

fix $R :: 'a \text{ hrel}$ **and** $D :: 'a \text{ hrel-des}$

assume $a: D \text{ is } \mathbf{H}$

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 $\dots = 'D_1 \wedge R \Rightarrow D_2'$

by (*simp add: rdesign-refinement*)

also have $\dots = ((D_1 \Rightarrow D_2) \sqsubseteq R)$

by (*rel-auto*)

also have $\dots = (Rel D \sqsubseteq R)$

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**
 $\mu_D F \equiv \mu_{DES} F$

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

syntax

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

translations

$\mu_D X \cdot P == \mu_{CONST\ DES} (\lambda X. P)$
 $\nu_D X \cdot P == \nu_{CONST\ DES} (\lambda X. P)$

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

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

Specialise *mu-refine-intro* to designs.

lemma *design-mu-refine-intro*:

assumes $\$ok' \# C \$ok' \# S (C \vdash S) \sqsubseteq F(C \vdash S) \text{ ' } C \Rightarrow (\mu_D F \Leftrightarrow \nu_D F) \text{ '}$
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 $(C \vdash_r S) \sqsubseteq F(C \vdash_r S) \text{ ' } \lceil C \rceil_D \Rightarrow (\mu_D F \Leftrightarrow \nu_D F) \text{ '}$
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 $P \text{ is } \mathbf{H} P \sqsubseteq F(P) \text{ ' } \lceil pre_D(P) \rceil_D \Rightarrow (\mu_D F \Leftrightarrow \nu_D F) \text{ '}$
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 $H: F \in \llbracket \mathbf{H} \rrbracket_H \rightarrow \llbracket \mathbf{H} \rrbracket_H$
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 ∧ [e]_< =_u <<st>>) ⊢_r Q ⊆ μ_D F
using WF proof (induction rule: wf-induct-rule)
  case (less st)
  hence 0: (P ∧ ([e]_<, <<st>>)_u ∈_u <<R>>) ⊢_r Q ⊆ μ_D F
    by rel-blast
  from M H design-theory-continuous.LFP-lemma3 mono-Monotone-utp-order
  have 1: μ_D F ⊆ F (μ_D F)
    by blast
  from 0 1 have 2: (P ∧ ([e]_<, <<st>>)_u ∈_u <<R>>) ⊢_r Q ⊆ F (μ_D F)
    by simp
  have 3: F ((P ∧ ([e]_<, <<st>>)_u ∈_u <<R>>) ⊢_r Q) ⊆ F (μ_D F)
    by (simp add: 0 M monoD)
  have 4: (P ∧ [e]_< =_u <<st>>) ⊢_r Q ⊆ ...
    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 ∈ [H]_H → [H]_H
and       induct-step:
  ∧ st. ((p ∧ e =_u <<st>>) ⊢_n Q) ⊆ F ((p ∧ (e, <<st>>)_u ∈_u <<R>>) ⊢_n Q)
shows (p ⊢_n Q) ⊆ μ_D F
using asms unfolding ndesign-def
by (rule-tac rdesign-mu-wf-refine-intro[of R F [p]_< e], simp-all add: alpha)

```

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

```

assumes   WF: wf R
and       M: Monotonic F
and       H: F ∈ [N]_H → [N]_H
and       induct-step:
  ∧ st. ((p ∧ e =_u <<st>>) ⊢_n Q) ⊆ F ((p ∧ (e, <<st>>)_u ∈_u <<R>>) ⊢_n Q)
shows (p ⊢_n Q) ⊆ μ_NDES F

```

proof –

```

{
fix st
have (p ∧ e =_u <<st>>) ⊢_n Q ⊆ μ_NDES F
using WF proof (induction rule: wf-induct-rule)
  case (less st)
  hence 0: (p ∧ (e, <<st>>)_u ∈_u <<R>>) ⊢_n Q ⊆ μ_NDES F
    by rel-blast
  from M H design-theory-continuous.LFP-lemma3 mono-Monotone-utp-order
  have 1: μ_NDES F ⊆ F (μ_NDES F)
    by (simp add: mono-Monotone-utp-order normal-design-theory-continuous.LFP-lemma3)
  from 0 1 have 2: (p ∧ (e, <<st>>)_u ∈_u <<R>>) ⊢_n Q ⊆ F (μ_NDES F)
    by simp

```

```

have 3:  $F ((p \wedge (e, \llst\gg)_u \in_u \ll R \gg) \vdash_n Q) \sqsubseteq F (\mu_{NDES} F)$ 
  by (simp add: 0 M monoD)
have 4:  $(p \wedge e =_u \llst\gg) \vdash_n Q \sqsubseteq \dots$ 
  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:  $\ll P \text{ is } \mathbf{N}; Q(\lfloor pre_D(P) \rfloor_{<} \vdash_n post_D(P)) \rr \implies Q(P)$ 
  by (simp add: ndesign-form)

```

```

lemma ndes-ind-elim:  $\ll \bigwedge i. P \ i \text{ is } \mathbf{N}; Q(\lambda i. \lfloor pre_D(P \ i) \rfloor_{<} \vdash_n post_D(P \ i)) \rr \implies Q(P)$ 
  by (simp add: ndesign-form)

```

```

lemma ndes-split [ND-elim]:  $\ll P \text{ is } \mathbf{N}; \bigwedge pre \ post. Q(pre \vdash_n post) \rr \implies Q(P)$ 
  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* :: $'\alpha \text{ usubst} \Rightarrow '\alpha \text{ hrel-des } (\langle \cdot \rangle_D)$ **where**
 $[upred-defs]: \text{assigns-d } \sigma = (\text{true} \vdash_r \text{assigns-r } \sigma)$

syntax

-assignmenttd :: $svids \Rightarrow uexprs \Rightarrow logic$ (**infixr** :=_D 62)

translations

-assignmenttd $xs \text{ vs} \Rightarrow \text{CONST assigns-d } (-mk-usubst \text{ (CONST id) } xs \text{ vs})$
-assignmenttd $x \text{ v} \leq \text{CONST assigns-d } (\text{CONST subst-upd } (\text{CONST id}) \text{ x v})$
-assignmenttd $x \text{ v} \leq \text{-assignmenttd } (-spvar \text{ x}) \text{ v}$
 $x, y :=_D u, v \leq \text{CONST assigns-d } (\text{CONST subst-upd } (\text{CONST subst-upd } (\text{CONST id}) (\text{CONST svar } x) \text{ u}) (\text{CONST svar } y) \text{ v})$

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]:
 $P \text{ is } \mathbf{H} \implies [\sigma \oplus_s \Sigma_D]_s \uparrow P \text{ is } \mathbf{H}$
by (*metis H1-H2-eq-rdesign Healthy-if rdesign-is-H1-H2 state-subst-design*)

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

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

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

lemma *assign-d-right-comp*:
 $((P \vdash_r Q) ;; \langle f \rangle_D) = ((\neg ((\neg P) ;; \text{true})) \vdash_r (Q ;; \langle f \rangle_a))$
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 $P :: '\alpha \text{ hrel-des}$
assumes $P \text{ is } \mathbf{H}$
shows $(\langle \sigma \rangle_D ;; P) = [\sigma \oplus_s \Sigma_D]_s \uparrow P$
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 $\dots = \lceil \sigma \rceil_s \dagger pre_D(P) \vdash_r \lceil \sigma \rceil_s \dagger post_D(P)$
by (*simp add: assign-d-left-comp*)
also have $\dots = \lceil \sigma \oplus_s \Sigma_D \rceil_s \dagger (pre_D(P) \vdash_r post_D(P))$
by (*rel-auto*)
also have $\dots = \lceil \sigma \oplus_s \Sigma_D \rceil_s \dagger P$
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]:

$P \text{ is } \mathbf{N} \implies \lceil \sigma \oplus_s \Sigma_D \rceil_s \dagger P \text{ is } \mathbf{N}$
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*: $\langle \sigma \rangle_D \text{ is } H4$

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 $\dots = true$
by (*rel-auto*)
ultimately show *?thesis*
using *is-H4-alt-def* **by** *auto*
qed

4.2 Guarded Commands

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

syntax *-GrdCommD* :: *uexp* \Rightarrow *logic* \Rightarrow *logic* ($- \rightarrow_D -$ [60, 61] 61)

translations *-GrdCommD* *b* *P* == *CONST GrdCommD b P*

lemma *GrdCommD-ndes-simp* [ndes-simp]:

$b \rightarrow_D (p_1 \vdash_n P_2) = ((b \Rightarrow p_1) \vdash_n (\lceil b \rceil_{<} \wedge P_2))$
by (*rel-auto*)

lemma *GrdCommD-H1-H3-closed* [closure]: $P \text{ is } \mathbf{N} \implies b \rightarrow_D P \text{ is } \mathbf{N}$

by (*simp add: GrdCommD-def closure*)

lemma *GrdCommD-true* [simp]: $true \rightarrow_D P = P$

by (*rel-auto*)

lemma *GrdCommD-false* [simp]: $false \rightarrow_D P = \top_D$

by (*rel-auto*)

lemma *GrdCommD-abort* [simp]: $b \rightarrow_D true = ((\neg b) \vdash_n false)$

by (*rel-auto*)

4.3 Alternation

consts

ualtern :: $'a \text{ set} \Rightarrow ('a \Rightarrow 'p) \Rightarrow ('a \Rightarrow 'r) \Rightarrow 'r \Rightarrow 'r$

ualtern-list :: $('a \times 'r) \text{ list} \Rightarrow 'r \Rightarrow 'r$

definition *AlternateD* :: 'a set \Rightarrow ('a \Rightarrow 'α upred) \Rightarrow ('a \Rightarrow ('α, 'β) rel-des) \Rightarrow ('α, 'β) rel-des \Rightarrow ('α, 'β) rel-des **where**
[upred-defs, ndes-simp]:
AlternateD A g P Q = $(\bigcap i \in A \cdot g(i) \rightarrow_D P(i)) \sqcap ((\bigwedge i \in A \cdot \neg g(i)) \rightarrow_D Q)$

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 **N**

shows

AlternateD A g P \perp_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 \perp_D =

$(\bigcap i \in A \cdot g(i) \rightarrow_D (\lfloor pre_D(P\ i) \rfloor_{<} \vdash_n post_D(P\ i))) \sqcap ((\bigwedge i \in A \cdot \neg g(i)) \rightarrow_D (false \vdash_n true))$

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* :: ('α upred \times ('α, 'β) rel-des) list \Rightarrow ('α, 'β) rel-des \Rightarrow ('α, 'β) rel-des **where**

[upred-defs, ndes-simp]:

AlternateD-list xs P =

AlternateD {0.. length xs} $(\lambda i. \text{map fst xs } ! i) (\lambda i. \text{map snd xs } ! i) P$

adhoc-overloading

ualtern *AlternateD* **and**

ualtern-list *AlternateD-list*

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

syntax

-altind-els :: ptrn \Rightarrow uexp \Rightarrow uexp \Rightarrow logic \Rightarrow logic \Rightarrow logic (if - \in - · - \rightarrow - else - fi)

-altind :: ptrn \Rightarrow uexp \Rightarrow uexp \Rightarrow logic \Rightarrow logic (if - \in - · - \rightarrow - fi)

-gcomm :: uexp \Rightarrow logic \Rightarrow gcomm (- \rightarrow - [60, 60] 61)

-gcomm-nil :: gcomm \Rightarrow gcomms (-)

-gcomm-cons :: gcomm \Rightarrow gcomms \Rightarrow gcomms (- | - [60, 61] 61)

-gcomm-show :: logic \Rightarrow logic

-altgcomm-els :: gcomms \Rightarrow logic \Rightarrow logic (if / - /else - /fi)

-altgcomm :: gcomms \Rightarrow logic (if / - /fi)

translations

-altind-els x A g P Q \Rightarrow CONST *ualtern* A $(\lambda x. g) (\lambda x. P)$ Q

-altind-els x A g P Q \Leftarrow CONST *ualtern* A $(\lambda x. g) (\lambda x'. P)$ Q

-altind x A g P \Rightarrow CONST *ualtern* A $(\lambda x. g) (\lambda x. P)$ (CONST *Orderings.top*)

-altind x A g P \Leftarrow CONST *ualtern* A $(\lambda x. g) (\lambda x'. P)$ (CONST *Orderings.top*)

$\text{-altgcomm } cs \Rightarrow \text{CONST ualtern-list } cs \text{ (CONST Orderings.top)}$
 $\text{-altgcomm } (-\text{gcomm-show } cs) \leq \text{CONST ualtern-list } cs \text{ (CONST Orderings.top)}$
 $\text{-altgcomm-els } cs \text{ } P \Rightarrow \text{CONST ualtern-list } cs \text{ } P$
 $\text{-altgcomm-els } (-\text{gcomm-show } cs) \text{ } P \leq \text{CONST ualtern-list } cs \text{ } P$

 $\text{-gcomm } g \text{ } P \Rightarrow (g, P)$
 $\text{-gcomm } g \text{ } P \leq \text{-gcomm-show } (g, P)$
 $\text{-gcomm-cons } c \text{ } cs \Rightarrow c \# cs$
 $\text{-gcomm-cons } (-\text{gcomm-show } c) \text{ } (-\text{gcomm-show } (d \# cs)) \leq \text{-gcomm-show } (c \# d \# cs)$
 $\text{-gcomm-nil } c \Rightarrow [c]$
 $\text{-gcomm-nil } (-\text{gcomm-show } c) \leq \text{-gcomm-show } [c]$

lemma *AlternateD-H1-H3-closed* [closure]:
assumes $\bigwedge i. i \in A \Rightarrow P \text{ } i \text{ is } \mathbf{N} \text{ } Q \text{ } i \text{ is } \mathbf{N}$
shows $\text{if } i \in A \cdot g(i) \rightarrow P(i) \text{ else } Q \text{ } fi \text{ is } \mathbf{N}$
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]:
 $\text{if } i \in A \cdot g(i) \rightarrow (P_1(i) \vdash_n P_2(i)) \text{ else } Q_1 \vdash_n Q_2 \text{ } fi$
 $= ((\bigwedge i \in A \cdot g \text{ } i \Rightarrow P_1 \text{ } i) \wedge ((\bigwedge i \in A \cdot \neg g \text{ } i) \Rightarrow Q_1)) \vdash_n$
 $((\bigvee i \in A \cdot [g \text{ } i]_{<} \wedge P_2 \text{ } i) \vee (\bigwedge i \in A \cdot \neg [g \text{ } i]_{<} \wedge Q_2))$
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 $\bigwedge i. P \text{ } i \sqsubseteq Q \text{ } i \text{ } R \sqsubseteq S$
shows $(\text{if } i \in A \cdot g(i) \rightarrow P(i) \text{ else } R \text{ } fi) \sqsubseteq (\text{if } i \in A \cdot g(i) \rightarrow Q(i) \text{ else } S \text{ } fi)$
using *assms* **by** (rel-auto, meson)

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

lemma *AlternateD-eq*:
assumes $A = B \wedge i. i \in A \Rightarrow g(i) = h(i) \wedge i. i \in A \Rightarrow P(i) = Q(i) \text{ } R = S$

shows $\text{if } i \in A \cdot g(i) \rightarrow P(i) \text{ else } R \text{ fi} = \text{if } i \in B \cdot h(i) \rightarrow Q(i) \text{ else } S \text{ fi}$
by (*insert assms, rel-blast*)

lemma *AlternateD-empty:*

if $i \in \{\}$ **·** $g(i) \rightarrow P(i) \text{ else } Q \text{ fi} = Q$
by (*rel-auto*)

lemma *AlternateD-true-singleton:*

assumes $P \text{ is } \mathbf{N}$
shows $\text{if } \text{true} \rightarrow P \text{ fi} = P$
by (*ndes-eq cls: assms*)

lemma *AlernateD-no-ind:*

assumes $A \neq \{\}$ $P \text{ is } \mathbf{N}$ $Q \text{ is } \mathbf{N}$
shows $\text{if } i \in A \cdot b \rightarrow P \text{ else } Q \text{ fi} = \text{if } b \rightarrow P \text{ else } Q \text{ fi}$
by (*ndes-eq cls: assms*)

lemma *AlernateD-singleton:*

assumes $P \text{ is } \mathbf{N}$ $Q \text{ is } \mathbf{N}$
shows $\text{if } i \in \{k\} \cdot b(i) \rightarrow P(i) \text{ else } Q \text{ fi} = \text{if } b(k) \rightarrow P(k) \text{ else } Q \text{ fi}$ (**is** $?lhs = ?rhs$)

proof –

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

qed

lemma *AlternateD-commute:*

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

lemma *AlternateD-dcond:*

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

lemma *AlternateD-cover:*

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

lemma *UINF-ndes-expand:*

assumes $\bigwedge i. i \in A \implies P(i) \text{ is } \mathbf{N}$
shows $(\bigcap i \in A \cdot \lfloor \text{pre}_D(P(i)) \rfloor < \vdash_n \text{post}_D(P(i))) = (\bigcap i \in A \cdot P(i))$
by (*rule UINF-cong, simp add: assms ndesign-form*)

lemma *USUP-ndes-expand:*

assumes $\bigwedge i. i \in A \implies P(i) \text{ is } \mathbf{N}$
shows $(\bigsqcup i \in A \cdot \lfloor \text{pre}_D(P(i)) \rfloor < \vdash_n \text{post}_D(P(i))) = (\bigsqcup i \in A \cdot P(i))$
by (*rule USUP-cong, simp add: assms ndesign-form*)

lemma *AlternateD-ndes-expand:*

assumes $\bigwedge i. i \in A \implies P(i) \text{ is } \mathbf{N}$ $Q \text{ is } \mathbf{N}$

shows $if\ i \in A \cdot g(i) \rightarrow P(i) \text{ else } Q\ fi =$
 $if\ i \in A \cdot g(i) \rightarrow (\lfloor pre_D(P(i)) \rfloor_{<} \vdash_n post_D(P(i))) \text{ else } \lfloor pre_D(Q) \rfloor_{<} \vdash_n post_D(Q)\ fi$
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 $\bigwedge i. i \in A \implies P(i) \text{ is } \mathbf{N}$
shows $if\ i \in A \cdot g(i) \rightarrow P(i)\ fi = if\ i \in A \cdot g(i) \rightarrow (\lfloor pre_D(P(i)) \rfloor_{<} \vdash_n post_D(P(i)))\ fi$
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 $\bigwedge i. P(i) \text{ is } \mathbf{N}$
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 $\bigwedge i. i \in (insert\ x\ A) \implies P(i) \text{ is } \mathbf{N}\ Q \text{ is } \mathbf{N}$
shows $if\ i \in (insert\ x\ A) \cdot g(i) \rightarrow P(i) \text{ else } Q\ fi =$
 $if\ g(x) \rightarrow P(x) \mid$
 $(\bigvee i \in A \cdot g(i)) \rightarrow if\ i \in A \cdot g(i) \rightarrow P(i)\ fi$
 $\text{else } Q$
 $fi\ (is\ ?lhs = ?rhs)$

proof –

have $?lhs = if\ i \in (insert\ x\ A) \cdot g(i) \rightarrow (\lfloor pre_D(P(i)) \rfloor_{<} \vdash_n post_D(P(i))) \text{ else } (\lfloor pre_D(Q) \rfloor_{<} \vdash_n post_D(Q))\ fi$

using *AlternateD-ndes-expand assms(1) assms(2)* **by** *blast*

also

have ... =

$if\ g(x) \rightarrow (\lfloor pre_D(P(x)) \rfloor_{<} \vdash_n post_D(P(x))) \mid$
 $(\bigvee i \in A \cdot g(i)) \rightarrow if\ i \in A \cdot g(i) \rightarrow \lfloor pre_D(P(i)) \rfloor_{<} \vdash_n post_D(P(i))\ fi$
 $\text{else } \lfloor pre_D(Q) \rfloor_{<} \vdash_n post_D(Q)$
 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*]:

$(p \vdash_n Q) ;; (p \vdash_n Q) \wedge n = ((\bigwedge i \in \{0..n\} \cdot (Q \wedge i) \text{ wp } p) \vdash_n Q \wedge Suc\ n)$

proof (*induct n*)

case 0

then show *?case* **by** (*rel-auto*)

next

case (*Suc n*) **note** *hyp = this*

```

have (p ⊢n Q) ;; (p ⊢n Q) ^ Suc n = (p ⊢n Q) ;; (p ⊢n Q) ;; (p ⊢n Q) ^ n
  by (simp add: upred-semiring.power-Suc)
also have ... = (p ⊢n Q) ;; (⋒ i ∈ {0..n} · Q ^ i wp p) ⊢n Q ^ Suc n
  by (simp add: hyp)
also have ... = (p ∧ Q wp (⋒ i ∈ {0..n} · Q ^ i wp p)) ⊢n (Q ;; Q) ;; Q ^ n
  by (simp add: upred-semiring.power-Suc ndesign-composition-wp segr-assoc)
also have ... = (p ∧ (⋒ i ∈ {0..n} · Q ^ Suc i wp p)) ⊢n (Q ;; Q) ;; Q ^ n
  by (simp add: upred-semiring.power-Suc wp)
also have ... = (p ∧ (⋒ i ∈ {0..n}. Q ^ Suc i wp p)) ⊢n (Q ;; Q) ;; Q ^ n
  by (simp add: USUP-as-Inf-image)
also have ... = (p ∧ (⋒ i ∈ {1..Suc n}. Q ^ i wp p)) ⊢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 ∧ (⋒ i ∈ {1..Suc n}. Q ^ i wp p)) ⊢n (Q ;; Q) ;; Q ^ n
  by (simp add: wp)
also have ... = ((⋒ i ∈ {0..Suc n}. Q ^ i wp p)) ⊢n (Q ;; Q) ;; Q ^ n
  by (simp add: atMost-Suc-eq-insert-0 atLeast0AtMost conj-upred-def image-Suc-atMost)
also have ... = (⋒ i ∈ {0..Suc n} · Q ^ i wp p) ⊢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 ⇒ ('a ⇒ 'p) ⇒ ('a ⇒ 'r) ⇒ 'r
uiterate-list :: ('a × 'r) list ⇒ 'r

```

syntax

```

-iterind      :: ptnr ⇒ uexp ⇒ uexp ⇒ 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 = (μ_{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))

ad hoc-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  $i \in \{\}$  ·  $g(i) \rightarrow P(i)$  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  $b \rightarrow P$  od = ( $\mu_{NDES} X \cdot \text{if } b \rightarrow P ;; X \text{ else } II_D \text{ fi}$ )
oops

lemma IterateD-singleton:
  assumes  $P \text{ is } \mathbf{N}$ 
  shows do  $b \rightarrow P$  od = do  $i \in \{0\} \cdot b \rightarrow P$  od
  apply (simp add: IterateD-list-def IterateD-def AlternateD-singleton assms)
  apply (subst AlternateD-singleton)
  apply (simp)
  apply (rel-auto)
oops

lemma IterateD-mono-refine:
  assumes
     $\bigwedge i. P \ i \text{ is } \mathbf{N} \bigwedge i. Q \ i \text{ is } \mathbf{N}$ 
     $\bigwedge i. P \ i \sqsubseteq Q \ i$ 
  shows (do  $i \in A \cdot g(i) \rightarrow P(i)$  od)  $\sqsubseteq$  (do  $i \in A \cdot g(i) \rightarrow Q(i)$  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
     $P \text{ is } \mathbf{N} \ Q \text{ is } \mathbf{N} \ P \sqsubseteq Q$ 
  shows (do  $g \rightarrow P$  od)  $\sqsubseteq$  (do  $g \rightarrow Q$  od)
oops

lemma IterateD-refine-intro:
  fixes  $V :: (\text{nat}, 'a) \text{ uexpr}$ 
  assumes  $\text{vwb-lens } w$ 
  shows
     $I \vdash_n (w: [\![ I \wedge \neg (\bigvee i \in A \cdot g(i)) \]\!]_{>}) \sqsubseteq$ 
    do  $i \in A \cdot g(i) \rightarrow (I \wedge g(i)) \vdash_n (w: [\![ I \]\!]_{>} \wedge [\![ V \]\!]_{>} <_u [\![ V \]\!]_{<})$  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
     $I \vdash_n (w: [I \wedge \neg g]_{>}) \sqsubseteq$ 
    do  $g \rightarrow ((I \wedge g) \vdash_n (w: [I]_{>} \wedge [V]_{>} <_u [V]_{<}))$  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 $LetD :: ('a, 'a) uexpr \Rightarrow ('a \Rightarrow 'a \text{ hrel-des}) \Rightarrow 'a \text{ hrel-des}$ **where**
 $[upred-defs]: LetD \ v \ P = (P \ x) \llbracket x \rightarrow [v]_{D<} \rrbracket$

syntax
 $-LetD \quad :: [letbinds, 'a] \Rightarrow 'a \quad ((let_D \ (-) / in \ (-)) \ [0, 10] \ 10)$

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

lemma $LetD\text{-}ndes\text{-}simp \ [ndes\text{-}simp]:$
 $LetD \ v \ (\lambda x. \ p(x) \vdash_n \ Q(x)) = (p(x) \llbracket x \rightarrow v \rrbracket) \vdash_n \ (Q(x) \llbracket x \rightarrow [v]_{<} \rrbracket)$
by (rel-auto)

lemma $LetD\text{-}H1\text{-}H3\text{-}closed \ [closure]:$
 $\llbracket \bigwedge x. \ P(x) \text{ is } \mathbf{N} \rrbracket \implies LetD \ v \ P \text{ is } \mathbf{N}$
by (rel-auto)

end

4.6 Design Hoare Logic

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

definition $HoareD :: 's \ upred \Rightarrow 's \ hrel\text{-}des \Rightarrow 's \ upred \Rightarrow bool \ (\{-\}\{-\}_D)$ **where**
 $[upred-defs, ndes-simp]: HoareD \ p \ S \ q = ((p \vdash_n [q]_{>}) \sqsubseteq S)$

lemma *assigns-hoare-d* [*hoare-safe*]: $'p \Rightarrow \sigma \dagger q' \Rightarrow \{p\}\langle\sigma\rangle_D\{q\}_D$
by *rel-auto*

lemma *skip-hoare-d*: $\{p\}II_D\{p\}_D$
by (*rel-auto*)

lemma *assigns-backward-hoare-d*:
 $\{\sigma \dagger p\}\langle\sigma\rangle_D\{p\}_D$
by *rel-auto*

lemma *seq-hoare-d*:
assumes C is **N** D is **N** $\{p\}C\{q\}_D \{q\}D\{r\}_D$
shows $\{p\}C ;; D\{r\}_D$

proof –

obtain $c_1 C_2$ **where** $C: C = c_1 \vdash_n C_2$
by (*metis assms(1) ndesign-form*)

obtain $d_1 D_2$ **where** $D: D = d_1 \vdash_n D_2$
by (*metis assms(2) ndesign-form*)

from *assms(3-4)* **show** *?thesis*

apply (*simp add: C D*)

apply (*ndes-simp*)

apply (*simp add: ndesign-refinement*)

apply (*rel-blast*)

done

qed

end

5 Design Weakest Preconditions

theory *utp-des-wp*

imports *utp-des-prog utp-des-hoare*

begin

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

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

theorem *wpd-eq-intro*: $\llbracket \bigwedge r. (p_1 \vdash_n Q_1) \text{wp}_D r = (p_2 \vdash_n Q_2) \text{wp}_D r \rrbracket \Rightarrow (p_1 \vdash_n Q_1) = (p_2 \vdash_n Q_2)$

apply (*rel-simp robust; metis curry-conv*)

done

theorem *wpd-H3-eq-intro*: $\llbracket P \text{ is } H1-H3; Q \text{ is } H1-H3; \bigwedge r. P \text{wp}_D r = Q \text{wp}_D r \rrbracket \Rightarrow P = Q$
by (*metis H1-H3-commute H1-H3-is-normal-design H3-idem Healthy-def' wpd-eq-intro*)

lemma *wp-d-abort* [*wp*]: $\text{true wp}_D p = \text{false}$
by (*rel-auto*)

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

theorem *rdesign-wp* [*wp*]:
 $(\lfloor p \rfloor_{<} \vdash_r Q) \text{wp}_D r = (p \wedge Q \text{wp } r)$

```

by (rel-auto)

theorem ndesign-wp [wp]:
  (p ⊢n Q) wpD r = (p ∧ Q wp r)
by (simp add: ndesign-def rdesign-wp)

theorem wpd-seq-r:
  fixes Q1 Q2 :: 'α hrel
  shows ((⌈p1⌉< ⊢r Q1) ;; (⌈p2⌉< ⊢r Q2)) wpD r = (⌈p1⌉< ⊢r Q1) wpD ((⌈p2⌉< ⊢r Q2) wpD 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 ⊢n Q1) ;; (p2 ⊢n Q2)) wpD r = (p1 ⊢n Q1) wpD ((p2 ⊢n Q2) wpD 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) wpD r = P wpD (Q wpD r)
  by (metis H1-H3-commute H1-H3-is-normal-design H1-idem Healthy-def' assms(1) assms(2) wpnd-seq-r)

theorem wp-hoare-d-link:
  assumes Q is N
  shows {p} Q {r}D ⟷ (Q wpD r ⊆ p)
  by (ndes-simp cls: assms, rel-auto)

end

```

6 Refinement Calculus

```

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

```

definition *des-spec* :: ('a ⇒ 'α) ⇒ 'α upred ⇒ ('α ⇒ 'α upred) ⇒ 'α hrel-des **where**
[upred-defs]: *des-spec* x p q = (⌊ v • ((p ∧ &v =_u «v»)) ⊢_n x:⌈q(v)⌉_>⌋)

syntax

```

-init-var      :: logic
-des-spec      :: salpha ⇒ logic ⇒ logic ⇒ logic (-: [-, / -]D [99, 0, 0] 100)
-des-log-const :: ptttrn ⇒ logic ⇒ logic (conD - · - [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 => ⌊ 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*:
 $x:[pre,post]_D = (pre \vdash_n x:[post]_{>})$
by (*rel-auto*)

lemma *des-spec-abort*:
 $x:[false,post]_D = \perp_D$
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 $V :: (nat, 'a) \text{ uepr}$
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 \sqsubseteq *?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 $\dots = ((\$ok \wedge \neg D^f \Rightarrow \$ok' \wedge D^t) \wedge (\$ok \wedge \neg D^f \Rightarrow \psi))$

by (*simp add: design-def*)

also have $\dots = ((\neg D^f) \vdash (D^t \wedge \psi))$

by (*pred-auto*)

finally show *?thesis* .

qed

lemma $OIH-idem$:

assumes D is $H1-H2$ $\$ok' \# \psi$

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 $OIH-of-design$:

$\$ok' \# P \Longrightarrow OIH(\psi)(P \vdash Q) = (P \vdash (Q \wedge \psi))$

by (*simp add: OIH-def design-def usubst, rel-auto*)

7.2 State Invariants

definition $ISH(\psi)(D) = (D \vee (\$ok \wedge \neg D^f \wedge [\psi]_{<} \Rightarrow \$ok' \wedge D^t))$

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

lemma $ISH-design$: $ISH(\psi)(D) = (\neg D^f \wedge [\psi]_{<}) \vdash D^t$

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 $ISH-of-design$:

$\llbracket \$ok' \# P; \$ok' \# Q \rrbracket \Longrightarrow ISH(\psi)(P \vdash Q) = ((P \wedge [\psi]_{<}) \vdash Q)$

by (*simp add: ISH-design design-def usubst, pred-auto*)

definition $OSH(\psi)(D) = (D \wedge (\$ok \wedge \neg D^f \wedge [\psi]_{<} \Rightarrow [\psi]_{>}))$

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*:

$\llbracket \$ok' \# P; \$ok' \# Q \rrbracket \Longrightarrow OSH(\psi)(P \vdash Q) = (P \vdash (Q \wedge (\lceil\psi\rceil_{<} \Rightarrow \lceil\psi\rceil_{>})))$
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*:

$\llbracket \$ok' \# P; \$ok' \# Q; ok \# \psi \rrbracket \Longrightarrow SIH(\psi)(P \vdash Q) = ((P \wedge \lceil\psi\rceil_{<}) \vdash (Q \wedge \lceil\psi\rceil_{>}))$
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-hoare
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] W. Guttman and B. Möller. Normal design algebra. *Journal of Logic and Algebraic Programming*, 79(2):144–173, February 2010.
- [4] T. Hoare and J. He. *Unifying Theories of Programming*. Prentice-Hall, 1998.