

UNIVERSITY *of* York

This is a repository copy of *Theory of Designs in Isabelle/UTP*.

White Rose Research Online URL for this paper:

<https://eprints.whiterose.ac.uk/129380/>

Monograph:

Foster, Simon David orcid.org/0000-0002-9889-9514, Nemouchi, Yakoub and Zeyda, Frank (2018) *Theory of Designs in Isabelle/UTP*. Working Paper. (Unpublished)

Reuse

Items deposited in White Rose Research Online are protected by copyright, with all rights reserved unless indicated otherwise. They may be downloaded and/or printed for private study, or other acts as permitted by national copyright laws. The publisher or other rights holders may allow further reproduction and re-use of the full text version. This is indicated by the licence information on the White Rose Research Online record for the item.

Takedown

If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

Theory of Designs in Isabelle/UTP

Simon Foster

Yakoub Nemouchi

Frank Zeyda

April 6, 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	5
1.4	Sequential Composition Laws	7
1.5	Preconditions and Postconditions	10
1.6	Distribution Laws	10
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	26
2.7	UTP theory of Designs	27
2.8	UTP theories	27
2.9	Galois Connection	28
2.10	Fixed Points	29
3	Design Proof Tactics	32

4	Imperative Programming in Designs	32
4.1	Assignment	32
4.2	Guarded Commands	34
4.3	Alternation	34
4.4	Iteration	38
4.5	Let and Local Variables	41
4.6	Deep Local Variables	41
5	Design Weakest Preconditions	43
6	Refinement Calculus	43
7	Theory of Invariants	45
7.1	Operation Invariants	45
7.2	State Invariants	46
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

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

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)

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

lemma *lmap-des-vars: vwb-lens* $f \implies vwb\text{-lens}$ (*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* ($[_]_D$)
where $[P]_D \equiv P \oplus_p (\Sigma_D \times_L \Sigma_D)$

abbreviation *lift-pre-desr* ($[_]_{D<}$)
where $[p]_{D<} \equiv [[p]_{<}]_D$

abbreviation *lift-post-desr* ($[_]_{D>}$)
where $[p]_{D>} \equiv [[p]_{>}]_D$

abbreviation *drop-desr* ($[_]_D$)
where $[P]_D \equiv P \upharpoonright_e (\Sigma_D \times_L \Sigma_D)$

abbreviation *dcond* :: $(''\alpha, '\beta) \text{ rel-des} \implies '\alpha \text{ upred} \implies (''\alpha, '\beta) \text{ rel-des} \implies (''\alpha, '\beta) \text{ rel-des}$
 $((\beta \prec _ \triangleright_D / _) [52, 0, 53] 52)$
where $P \prec b \triangleright_D Q \equiv P \prec [b]_{D<} \triangleright Q$

definition *design*::('α, 'β) rel-des ⇒ ('α, 'β) rel-des ⇒ ('α, 'β) rel-des (**infixl** ⊢ 60) **where**
[upred-defs]: P ⊢ Q = (\$ok ∧ P ⇒ \$ok' ∧ Q)

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

definition *rdesign*::('α, 'β) urel ⇒ ('α, 'β) urel ⇒ ('α, 'β) rel-des (**infixl** ⊢_r 60) **where**
[upred-defs]: (P ⊢_r Q) = [P]_D ⊢ [Q]_D

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

definition *ndesign*::'α cond ⇒ ('α, 'β) urel ⇒ ('α, 'β) rel-des (**infixl** ⊢_n 60) **where**
[upred-defs]: (p ⊢_n Q) = ([p]_< ⊢_r Q)

definition *skip-d* :: 'α hrel-des (II_D) **where**
[upred-defs]: II_D ≡ (true ⊢_r II)

definition *bot-d* :: ('α, 'β) rel-des (⊥_D) **where**
[upred-defs]: ⊥_D = (false ⊢ false)

definition *pre-design* :: ('α, 'β) rel-des ⇒ ('α, 'β) urel (pre_D) **where**
[upred-defs]: pre_D(P) = [¬ P[[true,false/\$ok,\$ok']]_D

definition *post-design* :: ('α, 'β) rel-des ⇒ ('α, 'β) urel (post_D) **where**
[upred-defs]: post_D(P) = [P[[true,true/\$ok,\$ok']]_D

syntax

-ok-f :: logic ⇒ logic (-^f [1000] 1000)
 -ok-t :: logic ⇒ logic (-^t [1000] 1000)
 -top-d :: logic (⊤_D)

translations

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

1.2 Lifting, Unrestriction, and Substitution

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

lemma *lift-desr-inv*:

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

proof –

have *bij-lens* (Σ_D ×_L Σ_D +_L (in-var ok +_L out-var ok)) :: (-, 'α des-vars-scheme × 'β des-vars-scheme)
lens)

(**is** *bij-lens* (?P))

proof –

have ?P ≈_L (ok +_L Σ_D) ×_L (ok +_L Σ_D) (**is** ?P ≈_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*:

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

by (*rel-auto*)

lemma *design-subst-ok-ok'*:

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

proof –

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

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 ... = $((\$ok \wedge P[\text{true}/\$ok]) \vdash (\$ok \wedge \$ok' \wedge Q[\text{true}, \text{true}/\$ok, \$ok']))$
by (*simp add: usubst*)
also have ... = $(P[\text{true}/\$ok] \vdash Q[\text{true}, \text{true}/\$ok, \$ok'])$
by (*pred-auto*)
finally show ?thesis ..
qed

lemma *design-subst-ok'*:

$(P \vdash Q[\text{true}/\$ok']) = (P \vdash Q)$

proof –

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

by (*pred-auto*)

also have ... = $(P \vdash (\$ok' \wedge Q[\text{true}/\$ok']))$

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

also have ... = $(P \vdash Q[\text{true}/\$ok'])$

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

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

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 ...

$= (((P1 \vdash Q1)[\text{false}/\$ok'] ;; (P2 \vdash Q2)[\text{false}/\$ok]) \vee ((P1 \vdash Q1)[\text{true}/\$ok'] ;; (P2 \vdash Q2)[\text{true}/\$ok]))$

by (*simp add: true-alt-def false-alt-def, pred-auto*)

also from *assms*

have ... = $((\$(ok \wedge P1 \Rightarrow Q1[\text{true}/\$ok']) ;; (P2 \Rightarrow \$ok' \wedge Q2[\text{true}/\$ok])) \vee ((\neg (\$ok \wedge P1)) ;; \text{true}))$

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

also have ... = $((\neg \$ok ;; \text{true}_h) \vee ((\neg P1) ;; \text{true}) \vee (Q1[\text{true}/\$ok'] ;; (\neg P2)) \vee (\$ok' \wedge (Q1[\text{true}/\$ok'] ;; Q2[\text{true}/\$ok])))$

by (*rel-auto*)

also have ... = $((\neg (\neg P1) ;; \text{true})) \wedge \neg (Q1[\text{true}/\$ok'] ;; (\neg P2)) \vdash (Q1[\text{true}/\$ok'] ;; Q2[\text{true}/\$ok]))$

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

finally show ?thesis .

qed

theorem *design-composition*:

assumes

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

shows $((P1 \vdash Q1) ;; (P2 \vdash Q2)) = ((\neg (\neg P1) ;; \text{true})) \wedge \neg (Q1 ;; (\neg P2)) \vdash (Q1 ;; Q2)$

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 $(\$ok \wedge \$ok' \wedge (Q1^t ;; Q2[[true/\$ok]]) = (\$ok \wedge \$ok' \wedge (Q1 ;; Q2))$
proof –
have $(\$ok \wedge \$ok' \wedge (Q1 ;; Q2)) = ((\$ok \wedge Q1) ;; (Q2 \wedge \$ok'))$
by *(metis (no-types, lifting) conj-comm seqr-post-var-out seqr-pre-var-out)*
also have $\dots = ((Q1 \wedge \$ok') ;; (\$ok \wedge Q2))$
by *(simp add: assms(3) assms(4) unrest-ident-var)*
also have $\dots = (Q1^t ;; Q2[[true/\$ok]])$
by *(metis ok-vwb-lens seqr-pre-transfer seqr-right-one-point true-alt-def uovar-convr upred-eq-true utp-pred-laws.inf.left-idem utp-rel.unrest-ouvar vwb-lens-mwb)*
finally show *?thesis*
by *(metis utp-pred-laws.inf.left-commute utp-pred-laws.inf-left-idem)*
qed
moreover have $(\neg (\neg P1 ;; true) \wedge \neg (Q1^t ;; (\neg P2))) \vdash (Q1^t ;; Q2[[true/\$ok]]) =$
 $(\neg (\neg P1 ;; true) \wedge \neg (Q1^t ;; (\neg P2))) \vdash (\$ok \wedge \$ok' \wedge (Q1^t ;; Q2[[true/\$ok]])$
by *(metis design-export-ok design-export-ok')*
ultimately show *?thesis using assms*
by *(simp add: design-composition-subst usubst, metis design-export-ok design-export-ok')*
qed

theorem *rdesign-composition:*

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

theorem *design-composition-cond:*

assumes
 $out\alpha \# p1 \ \$ok \# P2 \ \$ok' \# Q1 \ \$ok \# Q2$
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 $((\lceil p1 \rceil_{<} \vdash Q1) ;; (\lceil p2 \rceil_{<} \vdash Q2)) = ((\lceil p1 \wedge Q1 \ wp \ p2 \rceil_{<} \vdash (Q1 ;; Q2))$
using *assms by (rel-blast)*

theorem *rdesign-composition-wp:*

$((\lceil p1 \rceil_{<} \vdash_r Q1) ;; (\lceil p2 \rceil_{<} \vdash_r Q2)) = ((\lceil p1 \wedge Q1 \ wp \ p2 \rceil_{<} \vdash_r (Q1 ;; Q2))$
by *(rel-blast)*

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

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

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

proof –

have $(true ;; (P \vdash Q)) = (\exists ok_0 \cdot true[\llcorner ok_0 \gg / \$ok \prime] ;; (P \vdash Q)[\llcorner ok_0 \gg / \$ok])$

by (*subst segr-middle*[*of ok*], *simp-all*)

also have $\dots = ((true[\llcorner false / \$ok \prime] ;; (P \vdash Q)[\llcorner false / \$ok]) \vee (true[\llcorner true / \$ok \prime] ;; (P \vdash Q)[\llcorner true / \$ok]))$

by (*simp add: disj-comm false-alt-def true-alt-def*)

also have $\dots = ((true[\llcorner false / \$ok \prime] ;; true_h) \vee (true ;; ((P \vdash Q)[\llcorner true / \$ok])))$

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 $out\alpha \# true$

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 $out\alpha \# p$

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*: $pre_D(\bigsqcup_{i \in A} P i) = (\bigsqcap_{i \in A} pre_D(P i))$

by (*rel-auto*)

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

by (*rel-auto*)

1.6 Distribution Laws

theorem *design-choice*:

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

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-UINF-mem*:

assumes $A \neq \{\}$
 shows $(\prod i \in A \cdot P(i) \vdash Q(i)) = (\bigsqcup i \in A \cdot P(i) \vdash (\prod i \in A \cdot Q(i)))$
 using *assms* by (*rel-auto*)

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

assumes $A \neq \{\}$
 shows $(\prod i \in A \cdot p(i) \vdash_n Q(i)) = (\bigsqcup i \in A \cdot p(i) \vdash_n (\prod i \in A \cdot Q(i)))$
 using *assms* by (*rel-auto*)

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

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

by (*rel-auto*)

lemma *design-USUP-mem*:

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

by (*rel-auto*)

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

$$(\bigsqcup i \in A \cdot p(i) \vdash_n Q(i)) = (\prod i \in A \cdot p(i) \vdash_n (\bigsqcup i \in A \cdot [p(i)]_{<} \Rightarrow Q(i)))$$

by (*rel-auto*)

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

$(\bigsqcup i \cdot p(i) \vdash_n Q(i)) = (\prod i \cdot p(i)) \vdash_n (\bigsqcup i \cdot [p(i)]_< \Rightarrow Q(i))$
 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-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* :: $(\alpha, \beta) \text{ rel-des} \Rightarrow (\alpha, \beta) \text{ rel-des}$ **where**

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

lemma *H1-idem*:

$H1(H1 P) = H1(P)$

by (*pred-auto*)

lemma *H1-monotone*:

$P \sqsubseteq Q \Longrightarrow H1(P) \sqsubseteq H1(Q)$

by (*pred-auto*)

lemma *H1-Continuous*: *Continuous H1*

by (*rel-auto*)

lemma *H1-below-top*:

$H1(P) \sqsubseteq \top_D$

by (*pred-auto*)

lemma *H1-design-skip*:

$H1(\perp) = \perp_D$

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: precond-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 precond-right-unit unrest*)

finally show *?thesis* **by** (*metis H1-def Healthy-def'*)

qed

lemma *nok-not-false*:

$(\neg \$ok) \neq false$

by (*pred-auto*)

theorem *H1-left-zero*:

assumes *P is H1*

shows $(true ;; P) = true$

proof –

from *assms* **have** $(true ;; P) = (true ;; (\$ok \Rightarrow P))$

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

also from *assms* **have** $\dots = (true ;; (\neg \$ok \vee P))$ (**is** $- = (?true ;; -)$)

by (*simp add: impl-alt-def*)

also from *assms* **have** $\dots = ((?true ;; (\neg \$ok)) \vee (?true ;; P))$

using *seqr-or-distr* **by** *blast*

also from *assms* **have** $\dots = (true \vee (true ;; P))$

by (*simp add: nok-not-false precond-left-zero unrest*)

finally show *?thesis*

by (*simp add: upred-defs urel-defs*)

qed

theorem *H1-left-unit*:

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

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

theorem H1-nok-left-zero:
fixes $P :: 'α \text{ hrel-des}$
assumes $P \text{ is } H1$
shows $((\neg \$ok) ;; P) = (\neg \$ok)$
proof –
have $((\neg \$ok) ;; P) = (((\neg \$ok) ;; true_h) ;; P)$
by $(simp \text{ add: precondition-right-unit unrest})$
also have $\dots = ((\neg \$ok) ;; true_h)$
by $(metis \text{ H1-left-zero assms seqr-assoc})$
also have $\dots = (\neg \$ok)$
by $(simp \text{ add: precondition-right-unit unrest})$
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 $(simp \text{ add: H1-def Healthy-def' disj-upred-def impl-alt-def semilattice-sup-class.sup-left-commute})$

lemma H1-inf-closed [closure]:
 $\llbracket P \text{ is } H1; Q \text{ is } H1 \rrbracket \implies P \sqcup Q \text{ is } H1$
by $(rel-blast)$

lemma H1-UINF:
assumes $A \neq \{\}$
shows $H1(\prod i \in A \cdot P(i)) = (\prod i \in A \cdot H1(P(i)))$

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

lemma *H1-Sup*:

assumes $A \neq \{\}$ $\forall P \in A. P$ is *H1*

shows $(\sqcap A)$ 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 $H1(\sqcup i \in A \cdot P(i)) = (\sqcup i \in A \cdot H1(P(i)))$

by (*rel-auto*)

lemma *H1-Inf* [*closure*]:

assumes $\forall P \in A. P$ is *H1*

shows $(\sqcap A)$ 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 ... = $(P ;; ((\$ok \Rightarrow \$ok \wedge \$ok') \wedge \lceil II \rceil_D))$

by (*rel-auto*)

also have ... = $((P ;; (\neg \$ok \wedge \lceil II \rceil_D)) \vee (P ;; (\$ok \wedge (\lceil II \rceil_D \wedge \$ok'))))$

by (*rel-auto*)

also have ... = $(P^f \vee (P^t \wedge \$ok'))$

proof –

have $(P ;; (\neg \$ok \wedge \lceil II \rceil_D)) = P^f$

proof –

have $(P ;; (\neg \$ok \wedge \lceil II \rceil_D)) = ((P \wedge \neg \$ok') ;; \lceil II \rceil_D)$

by (*rel-auto*)

also have ... = $(\exists \$ok' \cdot P \wedge \$ok' =_u \text{false})$

by (*rel-auto*)

also have ... = P^f

by (*metis C1 one-point out-var-uvar unrest-as-exists ok-vwb-lens vwb-lens-mwb*)

finally show ?*thesis* .

qed

moreover have $(P ;; (\$ok \wedge (\lceil II \rceil_D \wedge \$ok'))) = (P^t \wedge \$ok')$

proof –
have $(P ;; (\$ok \wedge ([II]_D \wedge \$ok')))) = (P ;; (\$ok \wedge II))$
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 $\$ok' \# P \ \$ok' \# Q$
shows $H2(P \vdash Q) = P \vdash Q$
using *assms*
by *(simp add: H2-split design-def usubst unrest, pred-auto)*

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

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

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

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

theorem *H2-not-okay: H2 (¬ \$ok) = (¬ \$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 seqr-or-distl*)

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

assumes $P \text{ is } H2 \quad Q \text{ is } H2$

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 $H2(\prod i \in A \cdot P(i)) = (\prod i \in A \cdot H2(P(i)))$

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 *seqr-or-distl* by *blast*

also have $\dots = ((H2 (\neg \$ok)) \vee H2(P))$

by (*simp add: H2-def*)

also have $\dots = ((\neg \$ok) \vee H2(P))$

by (*simp add: H2-not-okay*)

also have $\dots = H1(H2(P))$

by (*rel-auto*)

finally show *?thesis* by *simp*

qed

2.3 Designs as $H1$ - $H2$ predicates

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

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

theorem $H1$ - $H2$ -*eq-design*:

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

proof –

have $\mathbf{H}(P) = (\$ok \Rightarrow H2(P))$

by (*simp add: H1-def*)

also have $\dots = (\$ok \Rightarrow (P^f \vee (P^t \wedge \$ok')))$

by (*metis H2-split*)

also have $\dots = (\$ok \wedge (\neg P^f) \Rightarrow \$ok' \wedge \$ok \wedge P^t)$

by (*rel-auto*)

also have $\dots = (\neg P^f) \vdash P^t$

by (*rel-auto*)

finally show *?thesis* .

qed

theorem $H1$ - $H2$ -*is-design*:

assumes P is $H1$ P is $H2$

shows $P = (\neg P^f) \vdash P^t$

using *assms* by (*metis H1-H2-eq-design Healthy-def*)

theorem $H1$ - $H2$ -*eq-rdesign*:

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

proof –

have $\mathbf{H}(P) = (\$ok \Rightarrow H2(P))$

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

also have $\dots = (\$ok \Rightarrow (P^f \vee (P^t \wedge \$ok')))$

by (*metis H2-split*)

also have $\dots = (\$ok \wedge (\neg P^f) \Rightarrow \$ok' \wedge P^t)$

by (*pred-auto*)

also have $\dots = (\$ok \wedge (\neg P^f) \Rightarrow \$ok' \wedge \$ok \wedge P^t)$

by (*pred-auto*)

also have $\dots = (\$ok \wedge [\text{pre}_D(P)]_D \Rightarrow \$ok' \wedge \$ok \wedge [\text{post}_D(P)]_D)$

by (*simp add: ok-post ok-pre*)

also have $\dots = (\$ok \wedge [\text{pre}_D(P)]_D \Rightarrow \$ok' \wedge [\text{post}_D(P)]_D)$

by (*pred-auto*)

also have $\dots = \text{pre}_D(P) \vdash_r \text{post}_D(P)$

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

finally show *?thesis* .

qed

theorem $H1$ - $H2$ -*is-rdesign*:

assumes P is $H1$ P is $H2$

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

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 $P \sqsubseteq Q \iff ('pre_D(P) \Rightarrow pre_D(Q))' \wedge 'pre_D(P) \wedge post_D(Q) \Rightarrow post_D(P)'$

by (*metis H1-H2-eq-rdesign Healthy-if assms rdesign-refinement*)

lemma *H1-H2-refines*:

assumes P is **H** Q is **H** $P \sqsubseteq Q$

shows $pre_D(Q) \sqsubseteq pre_D(P)$ $post_D(P) \sqsubseteq (pre_D(P) \wedge post_D(Q))$

using *H1-H2-refinement* *assms* *refBy-order* **by** *auto*

lemma *H1-H2-idempotent*: **H** (**H** P) = **H** P

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

lemma *H1-H2-Idempotent* [*closure*]: *Idempotent* **H**

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

lemma *H1-H2-monotonic* [*closure*]: *Monotonic* **H**

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

lemma *H1-H2-Continuous* [*closure*]: *Continuous* **H**

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

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

$\llbracket \$ok' \# P; \$ok' \# Q \rrbracket \implies (P \vdash Q)$ is **H**

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

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

$(P \vdash_r Q)$ is **H**

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

lemma *top-d-is-H1-H2* [*closure*]: \top_D is **H**

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

lemma *bot-d-is-H1-H2* [*closure*]: \perp_D is **H**

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

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

assumes P is **H** Q is **H**

shows $(P ;; Q)$ is **H**

proof –

obtain P_1 P_2 **where** $P = P_1 \vdash_r P_2$

by (*metis* *H1-H2-commute* *H1-H2-is-rdesign* *H2-idem* *Healthy-def* *assms*(1))

moreover obtain Q_1 Q_2 **where** $Q = Q_1 \vdash_r Q_2$

by (*metis* *H1-H2-commute* *H1-H2-is-rdesign* *H2-idem* *Healthy-def* *assms*(2))

moreover have $((P_1 \vdash_r P_2) ;; (Q_1 \vdash_r Q_2))$ is **H**

by (*simp* *add*: *rdesign-composition* *rdesign-is-H1-H2*)

ultimately show *?thesis* **by** *simp*

qed

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

assumes $A \neq \{\}$ $\forall P \in A.$ P is **H**

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

proof –

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

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

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

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 ... = $(\bigsqcup P \in A \cdot \neg P^f) \vdash (\prod P \in A \cdot P^t)$

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

also have ... is H1-H2

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

finally show ?thesis .

qed

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

$\prod_D A = (\text{if } (A = \{\}) \text{ then } \top_D \text{ else } \prod A)$

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

$\bigsqcup_D A \equiv \bigsqcup A$

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

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

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

by (simp add: design-inf-def)

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

by (simp add: design-inf-def)

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

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

shows $(\bigsqcup_{i \in A} P i)$ is **H**

proof –

from assms have $(\bigsqcup_{i \in A} P i) = (\bigsqcup_{i \in A} P i)$

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

also have ... = $(\bigsqcup_{i \in A} \neg (P i)^f) \vdash (P i)^t$

by (meson H1-H2-eq-design)

also have ... = $(\prod_{i \in A} \neg (P i)^f) \vdash (\bigsqcup_{i \in A} \neg (P i)^f \Rightarrow (P i)^t)$

by (simp add: design-USUP-mem)

also have ... is **H**

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

finally show ?thesis .

qed

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

assumes $\bigwedge i. P i$ is **H**

shows $(\bigsqcup i \cdot P i)$ is **H**

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

lemma *Inf-H1-H2-closed*:

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

shows $(\bigsqcup A)$ is **H**

proof –

from assms have $A: A = \mathbf{H} \text{ ' } A$

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 ... = $(\bigsqcup P \in A \cdot (\neg P^f) \vdash P^t)$
by (*meson H1-H2-eq-design*)
also have ... = $(\prod P \in A \cdot \neg P^f) \vdash (\bigsqcup P \in A \cdot \neg P^f \Rightarrow P^t)$
by (*simp add: design-USUP-mem*)
also have ... **is** **H**
by (*simp add: design-is-H1-H2 unrest*)
finally show *?thesis* .
qed

lemma *rdesign-ref-monos*:

assumes *P is H Q is H* $P \sqsubseteq Q$
shows $pre_D(Q) \sqsubseteq pre_D(P)$ $post_D(P) \sqsubseteq (pre_D(P) \wedge post_D(Q))$

proof –

have $r: P \sqsubseteq Q \longleftrightarrow ('pre_D(P) \Rightarrow pre_D(Q)' \wedge 'pre_D(P) \wedge post_D(Q) \Rightarrow post_D(P)')$
by (*metis H1-H2-eq-rdesign Healthy-if assms(1) assms(2) rdesign-refinement*)
from r **assms** **show** $pre_D(Q) \sqsubseteq pre_D(P)$
by (*auto simp add: refBy-order*)
from r **assms** **show** $post_D(P) \sqsubseteq (pre_D(P) \wedge post_D(Q))$
by (*auto simp add: refBy-order*)

qed

2.4 H3: The design assumption is a precondition

definition *H3* :: $('α, 'β)$ *rel-des* \Rightarrow $('α, 'β)$ *rel-des* **where**
[upred-defs]: H3 (P) $\equiv P$;; Π_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 $out\alpha \nmid p$
shows $(p \vdash Q)$ **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 ... = $p \vdash (Q^t ;; \Pi[true/\$ok])$
using *assms precondition-equiv seqr-true-lemma* **by force**
also have ... = $p \vdash Q$
by (*rel-auto*)
finally show *?thesis*
by (*simp add: H3-def Healthy-def*)

qed

theorem *rdesign-H3-iff-pre*:

$P \vdash_r Q$ is $H3 \iff P = (P ;; true)$
proof –
have $(P \vdash_r Q) ;; II_D = (P \vdash_r Q) ;; (true \vdash_r II)$
by (*simp add: skip-d-def*)
also have $\dots = (\neg ((\neg P) ;; true) \wedge \neg (Q ;; (\neg true))) \vdash_r (Q ;; II)$
by (*simp add: rdesign-composition*)
also have $\dots = (\neg ((\neg P) ;; true) \wedge \neg (Q ;; (\neg true))) \vdash_r Q$
by *simp*
also have $\dots = (\neg ((\neg P) ;; true)) \vdash_r Q$
by (*pred-auto*)
finally have $P \vdash_r Q$ is $H3 \iff P \vdash_r Q = (\neg ((\neg P) ;; true)) \vdash_r Q$
by (*metis H3-def Healthy-def'*)
also have $\dots \iff P = (\neg ((\neg P) ;; true))$
by (*metis rdesign-pre*)
thm *seqr-true-lemma*
also have $\dots \iff P = (P ;; true)$
by (*simp add: seqr-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 $H3 \iff P = (P ;; true)$
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 $H3 \iff \lfloor P \rfloor_D = (\lfloor P \rfloor_D ;; true)$
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 $\dots = (\exists ok_0 \cdot ((\$ok \Rightarrow \$ok') \wedge \lceil II \rceil_D) \llbracket \llcorner ok_0 \gg / \$ok' \rrbracket ;; (true \vdash II) \llbracket \llcorner ok_0 \gg / \$ok \rrbracket$
by (*subst seqr-middle[of ok], simp-all*)
also have $\dots = (((\$ok \Rightarrow \$ok') \wedge \lceil II \rceil_D) \llbracket false / \$ok' \rrbracket ;; (true \vdash II) \llbracket false / \$ok \rrbracket)$
 $\vee (((\$ok \Rightarrow \$ok') \wedge \lceil II \rceil_D) \llbracket true / \$ok' \rrbracket ;; (true \vdash II) \llbracket true / \$ok \rrbracket)$
by (*simp add: disj-comm false-alt-def true-alt-def*)
also have $\dots = ((\neg \$ok \wedge \lceil II \rceil_D ;; true) \vee (\lceil II \rceil_D ;; \$ok' \wedge \lceil II \rceil_D))$
by (*rel-auto*)
also have $\dots = II_D$
by (*rel-auto*)
finally show *?thesis* .

qed

lemma *H2-H3-absorb*:

$$H2 (H3 P) = H3 P$$

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

lemma *H3-H2-absorb*:

$$H3 (H2 P) = H3 P$$

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

theorem *H2-H3-commute*:

$$H2 (H3 P) = H3 (H2 P)$$

by (*simp add: H2-H3-absorb H3-H2-absorb*)

theorem *H3-design-pre*:

assumes $\$ok \# p \text{ out}\alpha \# p \ \$ok \# Q \ \$ok' \# Q$

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

$$H1-H3 p \equiv H1 (H3 p)$$

lemma *H1-H3-comp*: $H1-H3 = H1 \circ H3$

by (*auto*)

theorem *H1-H3-is-design*:

assumes P is *H1* P is *H3*

shows $P = (\neg P^f) \vdash P^t$

by (*metis H1-H2-eq-design H2-H3-absorb Healthy-def' assms(1) assms(2)*)

theorem *H1-H3-is-rdesign*:

assumes P is *H1* P is *H3*

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

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 $P = \lfloor \text{pre}_D(P) \rfloor_{<} \vdash_n \text{post}_D(P)$

by (*metis H1-H3-is-rdesign assms drop-pre-inv ndesign-def precondition-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 H* $\text{out}\alpha \# \text{pre}_D(P)$
 shows *P is N*
 by (*metis H1-H2-eq-rdesign H1-rdesign H3-rdesign-pre Healthy-def' assms*)

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

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

lemma *H1-H3-eq-design*: $\mathbf{N}(P) = (\neg (P^f ;; \text{true})) \vdash P^t$
 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 N*
 shows $\text{out}\alpha \# P^f$
proof –
 have $P = (\neg (P^f ;; \text{true})) \vdash P^t$
 by (*metis H1-H3-eq-design Healthy-def' assms*)
 also have $\text{out}\alpha \# (\dots)^f$
 by (*simp add: design-def usubst unrest, rel-auto*)
 finally show *?thesis* .
qed

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

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

lemma *ndesign-form*: $P \text{ is } \mathbf{N} \implies ([\text{pre}_D(P)]_< \vdash_n \text{post}_D(P)) = P$
 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]*: $\perp_D \text{ is } \mathbf{N}$
 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]*: $\top_D \text{ is } \mathbf{N}$

by (metis ndesign-H1-H3 ndesign-miracle)

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

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

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

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

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

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

qed

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

qed

lemma USUP-ind-H1-H3-closed [closure]:
 $\llbracket \bigwedge i. P i \text{ is } \mathbf{N} \rrbracket \implies (\bigsqcup i \cdot P i) \text{ is } \mathbf{N}$
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]: $H_4(P) = ((P ;; true) \Rightarrow P)$

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

typedecl *DES*

typedecl *NDES*

abbreviation *DES* \equiv *UTHY*(*DES*, ' α *des*)

abbreviation *NDES* \equiv *UTHY*(*NDES*, ' α *des*)

overloading

des-hcond == *utp-hcond* :: (*DES*, ' α *des*) *uthy* \Rightarrow (' α *des* \times ' α *des*) *health*

des-unit == *utp-unit* :: (*DES*, ' α *des*) *uthy* \Rightarrow ' α *hrel-des* (**unchecked**)

ndes-hcond == *utp-hcond* :: (*NDES*, ' α *des*) *uthy* \Rightarrow (' α *des* \times ' α *des*) *health*

ndes-unit == *utp-unit* :: (*NDES*, ' α *des*) *uthy* \Rightarrow ' α *hrel-des* (**unchecked**)

begin

definition *des-hcond* :: (*DES*, ' α *des*) *uthy* \Rightarrow (' α *des* \times ' α *des*) *health* **where**

[*upred-defs*]: *des-hcond* *t* = *H1-H2*

definition *des-unit* :: (*DES*, ' α *des*) *uthy* \Rightarrow ' α *hrel-des* **where**

[*upred-defs*]: *des-unit* *t* = *II_D*

definition *ndes-hcond* :: (*NDES*, ' α *des*) *uthy* \Rightarrow (' α *des* \times ' α *des*) *health* **where**

[*upred-defs*]: *ndes-hcond* *t* = *H1-H3*

definition *ndes-unit* :: (*NDES*, ' α *des*) *uthy* \Rightarrow ' α *hrel-des* **where**

[*upred-defs*]: *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 $\bigwedge P. P \in \text{carrier (uthy-order DES)} \longleftrightarrow P \text{ is } \mathbf{H}$
and *carrier (uthy-order DES) \rightarrow carrier (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 (uthy-order DES) = op \sqsubseteq*
and *eq (uthy-order DES) = op =*
by (*unfold-locales, simp-all add: des-hcond-def H1-H2-Continuous utp-order-def*)

interpretation *normal-design-theory-continuous: utp-theory-continuous NDES*
rewrites $\bigwedge P. P \in \text{carrier (uthy-order NDES)} \longleftrightarrow P \text{ is } \mathbf{N}$
and *carrier (uthy-order NDES) \rightarrow carrier (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 (uthy-order NDES) = op \sqsubseteq*
and *$A \subseteq \text{carrier (uthy-order NDES)} \longleftrightarrow A \subseteq \llbracket \mathbf{N} \rrbracket_H$*
and *eq (uthy-order NDES) = op =*
by (*unfold-locales, simp-all add: ndes-hcond-def H1-H3-Continuous utp-order-def*)

lemma *design-lat-top: $\top_{DES} = \mathbf{H}(\text{false})$*
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*]: $Des(R) = \mathbf{H}(\llbracket R \rrbracket_D \wedge \$ok')$

definition [*upred-defs*]: $Rel(D) = \llbracket D[\text{true}, \text{true}/\$ok, \$ok'] \rrbracket_D$

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

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

interpretation *Des-Rel-coretract*:
 $\text{coretract } DES \leftarrow \langle Des, Rel \rangle \rightarrow REL$
rewrites
 $\bigwedge x. x \in \text{carrier } \mathcal{X}_{DES \leftarrow \langle Des, Rel \rangle \rightarrow REL} = (x \text{ is } \mathbf{H}) \text{ and}$
 $\bigwedge x. x \in \text{carrier } \mathcal{Y}_{DES \leftarrow \langle Des, Rel \rangle \rightarrow REL} = \text{True and}$
 $\pi^*_{DES \leftarrow \langle Des, Rel \rangle \rightarrow REL} = Des \text{ and}$
 $\pi^*_{DES \leftarrow \langle Des, Rel \rangle \rightarrow REL} = Rel \text{ and}$
 $le \mathcal{X}_{DES \leftarrow \langle Des, Rel \rangle \rightarrow REL} = op \sqsubseteq \text{ and}$
 $le \mathcal{Y}_{DES \leftarrow \langle Des, Rel \rangle \rightarrow REL} = op \sqsubseteq$
proof (*unfold-locales, simp-all add: rel-hcond-def des-hcond-def*)
show $\bigwedge x. x \text{ is } id$
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* $:: ('\alpha \text{ hrel-des} \Rightarrow '\alpha \text{ hrel-des}) \Rightarrow '\alpha \text{ hrel-des } (\mu_D)$ **where**
 $\mu_D F \equiv \mu_{DES} F$

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

syntax

-dmu :: ptnrn ⇒ logic ⇒ logic (μ_D - · - [0, 10] 10)
-dnu :: ptnrn ⇒ logic ⇒ logic (ν_D - · - [0, 10] 10)

translations

μ_D X · P == μ_{CONST DES} (λ X. P)
ν_D X · P == ν_{CONST DES} (λ 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 ⊢ S) ⊆ F(C ⊢ S) 'C ⇒ (μ_D F ⇔ ν_D F)'
shows (C ⊢ S) ⊆ μ_D F

proof –

from *assms* **have** (C ⊢ S) ⊆ ν_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 ⊢_r S) ⊆ F(C ⊢_r S) '[C]_D ⇒ (μ_D F ⇔ ν_D F)'

shows (C ⊢_r S) ⊆ μ_D F

using *assms* **by** (*simp add: rdesign-def design-mu-refine-intro unrest*)

lemma *H1-H2-mu-refine-intro*:

assumes P is **H** P ⊆ F(P) '[pre_D(P)]_D ⇒ (μ_D F ⇔ ν_D F)'

shows P ⊆ μ_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 ∈ [**H**]_H → [**H**]_H

and *induct-step*:

∧*st*. (P ∧ [e]_< =_u <st>) ⊢_r Q ⊆ F ((P ∧ ([e]_<, <st>)_u ∈_u <R>) ⊢_r Q)

shows (P ⊢_r Q) ⊆ μ_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: $F ((P \wedge ([e]_{<}, \ll st \gg)_u \in_u \ll R \gg)) \vdash_r Q \sqsubseteq F (\mu_D F)$
 by (*simp add: 0 M monoD*)
 have 4: $(P \wedge [e]_{<} =_u \ll st \gg) \vdash_r Q \sqsubseteq \dots$
 by (*rule induct-step*)
 show ?*case*
 using *order-trans[OF 3 4] H M design-theory-continuous.LFP-lemma2 dual-order.trans mono-Monotone-utp-order*

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

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

theorem *ndesign-mu-wf-refine-intro*:
 assumes $WF: wf R$
 and $M: Monotonic F$
 and $H: F \in \ll \mathbf{N} \gg_H \rightarrow \ll \mathbf{N} \gg_H$
 and *induct-step*:
 $\bigwedge st. ((p \wedge e =_u \ll st \gg) \vdash_n Q) \sqsubseteq F ((p \wedge (e, \ll st \gg)_u \in_u \ll R \gg)) \vdash_n Q$
 shows $(p \vdash_n Q) \sqsubseteq \mu_{NDES} F$

proof –

{
 fix *st*
 have $(p \wedge e =_u \ll st \gg) \vdash_n Q \sqsubseteq \mu_{NDES} F$
 using *WF proof (induction rule: wf-induct-rule)*
 case (*less st*)
 hence 0: $(p \wedge (e, \ll st \gg)_u \in_u \ll R \gg) \vdash_n Q \sqsubseteq \mu_{NDES} F$
 by *rel-blast*
 from *M H design-theory-continuous.LFP-lemma3 mono-Monotone-utp-order*
 have 1: $\mu_{NDES} F \sqsubseteq F (\mu_{NDES} F)$
 by (*simp add: mono-Monotone-utp-order normal-design-theory-continuous.LFP-lemma3*)
 from 0 1 have 2: $(p \wedge (e, \ll st \gg)_u \in_u \ll R \gg) \vdash_n Q \sqsubseteq F (\mu_{NDES} F)$
 by *simp*
 have 3: $F ((p \wedge (e, \ll st \gg)_u \in_u \ll R \gg) \vdash_n Q) \sqsubseteq F (\mu_{NDES} F)$
 by (*simp add: 0 M monoD*)
 have 4: $(p \wedge e =_u \ll st \gg) \vdash_n Q \sqsubseteq \dots$
 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:  $\llbracket P \text{ is } \mathbf{N}; Q(\lfloor \text{pre}_D(P) \rfloor_{<} \vdash_n \text{post}_D(P)) \rrbracket \implies Q(P)$ 
  by (simp add: ndesign-form)

```

```

lemma ndes-ind-elim:  $\llbracket \bigwedge i. P \ i \text{ is } \mathbf{N}; Q(\lambda i. \lfloor \text{pre}_D(P \ i) \rfloor_{<} \vdash_n \text{post}_D(P \ i)) \rrbracket \implies Q(P)$ 
  by (simp add: ndesign-form)

```

```

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

```

syntax

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

translations

-assignmentd xs vs \Rightarrow CONST assigns-d (-mk-ustubst (CONST id) xs vs)

-assignmentd x v \Leftarrow CONST assigns-d (CONST subst-upd (CONST id) x v)

-assignmentd x v \Leftarrow -assignmentd (-spvar x) v

$x, y :=_D u, v \Leftarrow$ CONST assigns-d (CONST subst-upd (CONST subst-upd (CONST id) (CONST svar x) u) (CONST svar y) v)

lemma assigns-d-is-H1-H2 [closure]: $\langle \sigma \rangle_D$ is **H**

by (simp add: assigns-d-def rdesign-is-H1-H2)

lemma assigns-d-H1-H3 [closure]: $\langle \sigma \rangle_D$ is **N**

by (metis H1-rdesign H3-ndesign Healthy-def' aext-true assigns-d-def ndesign-def)

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

lemma state-subst-H1-H2-closed [closure]:

P is **H** \implies $[\sigma \oplus_s \Sigma_D]_s \dagger P$ is **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]: $\langle \text{id} \rangle_D = \text{ID}$

by (rel-auto)

lemma assign-d-left-comp:

$(\langle f \rangle_D ;; (P \vdash_r Q)) = ([f]_s \dagger P \vdash_r [f]_s \dagger Q)$

by (simp add: assigns-d-def rdesign-composition assigns-r-comp subst-not)

lemma assign-d-right-comp:

$((P \vdash_r Q) ;; \langle f \rangle_D) = ((\neg ((\neg P) ;; \text{true})) \vdash_r (Q ;; \langle f \rangle_a))$

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 $P :: \alpha$ hrel-des

assumes P is **H**

shows $(\langle \sigma \rangle_D ;; P) = [\sigma \oplus_s \Sigma_D]_s \dagger P$

proof –

have $\langle \sigma \rangle_D ;; P = \langle \sigma \rangle_D ;; (\text{pre}_D(P) \vdash_r \text{post}_D(P))$

by (metis H1-H2-commute H1-H2-is-rdesign H2-idem Healthy-def' assms)

also have $\dots = [\sigma]_s \dagger \text{pre}_D(P) \vdash_r [\sigma]_s \dagger \text{post}_D(P)$

by (simp add: assign-d-left-comp)

also have $\dots = [\sigma \oplus_s \Sigma_D]_s \dagger (\text{pre}_D(P) \vdash_r \text{post}_D(P))$

by (rel-auto)

also have $\dots = [\sigma \oplus_s \Sigma_D]_s \dagger P$

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]:

$P \text{ is } \mathbf{N} \implies [\sigma \oplus_s \Sigma_D]_s \dagger P \text{ is } \mathbf{N}$

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 $(\langle \sigma \rangle_D ;; (\text{false} \vdash_r \text{true}_h)) = (\text{false} \vdash_r \text{true})$

by (*simp add: assigns-d-def rdesign-composition assigns-r-feasible*)

moreover have $\dots = \text{true}$

by (*rel-auto*)

ultimately show *?thesis*

using *is-H4-alt-def* by *auto*

qed

4.2 Guarded Commands

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

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

$b \rightarrow_D (p_1 \vdash_n P_2) = ((b \Rightarrow p_1) \vdash_n ([b]_{<} \wedge P_2))$

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*]: $\text{true} \rightarrow_D P = P$

by (*rel-auto*)

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

by (*rel-auto*)

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

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

[*upred-defs, ndes-simp*]:

$\text{AlternateD } A \ g \ P \ Q = (\prod_{i \in A} i \cdot g(i) \rightarrow_D P(i)) \sqcap (\bigwedge_{i \in A} i \cdot \neg g(i) \rightarrow_D Q)$

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 $\bigwedge i. P(i) \text{ is } \mathbf{N}$

shows

$AlternateD\ A\ g\ P\ \perp_D =$
 $((\bigvee i \in A \cdot g(i)) \wedge (\bigwedge i \in A \cdot g(i) \Rightarrow \lfloor pre_D(P\ i) \rfloor_{<})) \vdash_n (\bigvee i \in A \cdot \lfloor g(i) \rfloor_{<} \wedge post_D(P\ i))$
proof (*cases* $A = \{\}$)
case *False*
have $AlternateD\ A\ g\ P\ \perp_D =$
 $(\prod i \in A \cdot g(i) \rightarrow_D (\lfloor pre_D(P\ i) \rfloor_{<} \vdash_n post_D(P\ i))) \sqcap (\bigwedge i \in A \cdot \neg g(i) \rightarrow_D (false \vdash_n true))$
by (*simp add: AlternateD-def ndesign-form bot-d-ndes-def assms*)
also have $\dots = ((\bigvee i \in A \cdot g(i)) \wedge (\bigwedge i \in A \cdot g(i) \Rightarrow \lfloor pre_D(P\ i) \rfloor_{<})) \vdash_n (\bigvee i \in A \cdot \lfloor g(i) \rfloor_{<} \wedge post_D(P\ i))$
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 $AlternateD\text{-list} :: ('\alpha\ upred \times ('\alpha, '\beta)\ rel\text{-des})\ list \Rightarrow ('\alpha, '\beta)\ rel\text{-des} \Rightarrow ('\alpha, '\beta)\ rel\text{-des}$
where

[*upred-defs, ndes-simp*]:

$AlternateD\text{-list}\ xs\ P =$

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

adhoc-overloading

ualtern $AlternateD$ **and**

ualtern-list $AlternateD\text{-list}$

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

syntax

-altind-els $::\ ptrn \Rightarrow logic \Rightarrow logic \Rightarrow logic \Rightarrow logic \Rightarrow logic\ (if\ \neg\ \cdot\ \rightarrow\ \cdot\ \text{else}\ \cdot\ fi)$

-altind $::\ ptrn \Rightarrow logic \Rightarrow logic \Rightarrow logic \Rightarrow logic\ (if\ \neg\ \cdot\ \rightarrow\ \cdot\ \text{else}\ \cdot\ fi)$

-gcomm $::\ logic \Rightarrow logic \Rightarrow gcomm\ (\rightarrow\ \cdot\ \text{else}\ \cdot\ [65, 66]\ 65)$

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

-gcomm-cons $::\ gcomm \Rightarrow gcomms \Rightarrow gcomms\ (\cdot\ | \cdot\ \text{else}\ \cdot\ [60, 61]\ 61)$

-gcomm-show $::\ logic \Rightarrow logic$

-altgcomm-els $::\ gcomms \Rightarrow logic \Rightarrow logic\ (if\ \cdot\ \text{else}\ \cdot\ fi)$

-altgcomm $::\ gcomms \Rightarrow logic\ (if\ \cdot\ \text{else}\ \cdot\ fi)$

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

-altgcomm $cs \Rightarrow CONST\ ualtern\text{-list}\ cs\ (CONST\ Orderings.top)$

-altgcomm $(\text{-gcomm-show}\ cs) \Leftarrow CONST\ ualtern\text{-list}\ cs\ (CONST\ Orderings.top)$

-altgcomm-els $cs\ P \Rightarrow CONST\ ualtern\text{-list}\ cs\ P$

-altgcomm-els $(\text{-gcomm-show}\ cs)\ P \Leftarrow CONST\ ualtern\text{-list}\ cs\ P$

-gcomm $g\ P \Rightarrow (g, P)$

-gcomm $g\ P \Leftarrow \text{-gcomm-show}\ (g, P)$

-gcomm-cons $c\ cs \Rightarrow c\ \#\ cs$

-gcomm-cons $(\text{-gcomm-show}\ c)\ (\text{-gcomm-show}\ (d\ \#\ cs)) \Leftarrow \text{-gcomm-show}\ (c\ \#\ d\ \#\ cs)$

-gcomm-nil $c \Rightarrow [c]$

-gcomm-nil $(\text{-gcomm-show}\ c) \Leftarrow \text{-gcomm-show}\ [c]$

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

lemma *AlternateD-eq*:
assumes $A = B \ \bigwedge i. i \in A \implies g(i) = h(i) \ \bigwedge i. i \in A \implies P(i) = Q(i) \ R = S$
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*:
 $\text{if } i \in \{\} \cdot g(i) \rightarrow P(i) \text{ else } Q \ \text{fi} = Q$
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 is \mathbf{N} Q 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 k is \mathbf{N} Q 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 is \mathbf{N} Q 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 is \mathbf{N} Q 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 is \mathbf{N} Q 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 $\bigwedge i. i \in A \implies P(i)$ is \mathbf{N}
shows $(\prod i \in A \cdot \lfloor \text{pre}_D(P(i)) \rfloor < \vdash_n \text{post}_D(P(i))) = (\prod i \in A \cdot P(i))$
by (*rule UINF-cong, simp add: assms ndesign-form*)

lemma *USUP-ndes-expand*:

assumes $\bigwedge i. i \in A \implies P(i)$ 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 $\bigwedge i. i \in A \implies P(i)$ is \mathbf{N} Q is \mathbf{N}
shows $\text{if } i \in A \cdot g(i) \rightarrow P(i) \text{ else } Q \text{ fi} =$
 $\text{if } i \in A \cdot g(i) \rightarrow (\lfloor \text{pre}_D(P(i)) \rfloor < \vdash_n \text{post}_D(P(i))) \text{ else } \lfloor \text{pre}_D(Q) \rfloor < \vdash_n \text{post}_D(Q) \text{ fi}$
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 $\bigwedge i. i \in A \implies P(i)$ 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 $\bigwedge i. P(i)$ *is N*
shows $(\lambda i. \lfloor pre_D(P(i)) \rfloor_{<} \vdash_n post_D(P(i))) = P$
by (*simp add: assms ndesign-form*)

lemma *AlternateD-insert:*

assumes $\bigwedge i. i \in (insert\ x\ A) \implies P(i)$ *is N* Q *is N*
shows $if\ i \in (insert\ x\ A) \cdot g(i) \rightarrow P(i)$ *else* $Q\ fi =$
 $if\ g(x) \rightarrow P(x) \mid$
 $(\bigvee i \in A \cdot g(i)) \rightarrow if\ i \in A \cdot g(i) \rightarrow P(i)\ fi$
else Q
 fi (**is** $?lhs = ?rhs$)

proof –

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

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

also

have ... =

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

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

proof (*induct n*)

case 0

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

next

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

have $(p \vdash_n Q) ;; (p \vdash_n Q) \hat{=} Suc\ n = (p \vdash_n Q) ;; (p \vdash_n Q) ;; (p \vdash_n Q) \hat{=} n$

by (*simp add: upred-semiring.power-Suc*)

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

by (*simp add: hyp*)

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

by (*simp add: upred-semiring.power-Suc ndesign-composition-wp seqr-assoc*)

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

by (*simp add: upred-semiring.power-Suc wp*)

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

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

also have ... = $(p \wedge (\bigwedge i \in \{1..Suc\ n\} \cdot Q \hat{=} i)\ wp\ p) \vdash_n (Q ;; Q) ;; Q \hat{=} n$

by (*metis* (*no-types*, *lifting*) *One-nat-def image-Suc-atLeastAtMost image-cong image-image*)
also have ... = $(Q \hat{=} 0 \text{ wp } p \wedge (\bigsqcup i \in \{1..Suc\ n\}. Q \hat{=} i \text{ wp } p)) \vdash_n (Q ;; Q) ;; Q \hat{=} n$
 by (*simp add*: *wp*)
also have ... = $(\bigsqcup i \in \{0..Suc\ n\}. Q \hat{=} i \text{ wp } p) \vdash_n (Q ;; Q) ;; Q \hat{=} n$
 by (*simp add*: *Iic-Suc-eq-insert-0 atLeast0AtMost conj-upred-def image-Suc-atMost*)
also have ... = $(\bigsqcup i \in \{0..Suc\ n\} \cdot Q \hat{=} i \text{ wp } p) \vdash_n Q \hat{=} Suc\ (Suc\ n)$
 by (*simp add*: *upred-semiring.power-Suc USUP-as-Inf-image upred-semiring.mult-assoc*)
finally show ?*case* .
qed

Overloadable Syntax

consts

uiterate :: $'a \text{ set} \Rightarrow ('a \Rightarrow 'p) \Rightarrow ('a \Rightarrow 'r) \Rightarrow 'r$
uiterate-list :: $('a \times 'r) \text{ list} \Rightarrow 'r$

syntax

-iterind :: $p\text{trn} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic} \ (do \ -\in\ - \cdot - \rightarrow - \text{ od})$
-itergcomm :: $g\text{comms} \Rightarrow \text{logic} \ (do \ - \text{ od})$

translations

-iterind $x\ A\ g\ P \Rightarrow \text{CONST } uiterate\ A\ (\lambda\ x.\ g)\ (\lambda\ x.\ P)$
-iterind $x\ A\ g\ P \Leftarrow \text{CONST } uiterate\ A\ (\lambda\ x.\ g)\ (\lambda\ x'.\ P)$
-itergcomm $cs \Rightarrow \text{CONST } uiterate-list\ cs$
-itergcomm $(-gcomm-show\ cs) \Leftarrow \text{CONST } uiterate-list\ cs$

definition *IterateD* :: $'a \text{ set} \Rightarrow ('a \Rightarrow 'a \text{ upred}) \Rightarrow ('a \Rightarrow 'a \text{ hrel-des}) \Rightarrow 'a \text{ hrel-des}$ **where**
 [*upred-defs*, *ndes-simp*]:

IterateD $A\ g\ P = (\mu_{NDES}\ X \cdot \text{if } i \in A \cdot g(i) \rightarrow P(i) ;; X \text{ else } II_D\ fi)$

definition *IterateD-list* :: $('a \text{ upred} \times 'a \text{ hrel-des}) \text{ list} \Rightarrow 'a \text{ hrel-des}$ **where**
 [*upred-defs*, *ndes-simp*]:

IterateD-list $xs = IterateD\ \{0..<length\ xs\} (\lambda\ i.\ \text{fst}\ (nth\ xs\ i)) (\lambda\ i.\ \text{snd}\ (nth\ xs\ i))$

adhoc-overloading

uiterate *IterateD* **and**
uiterate-list *IterateD-list*

lemma *IterateD-H1-H3-closed* [*closure*]:

assumes $\bigwedge i.\ i \in A \Longrightarrow P\ i$ **is** \mathbf{N}
shows $do\ i \in A \cdot g(i) \rightarrow P(i)$ **od** **is** \mathbf{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*:

$do\ i \in \{\} \cdot g(i) \rightarrow P(i)$ **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*:

$do\ b \rightarrow P\ od = (\mu_{NDES}\ X \cdot if\ b \rightarrow P\ ;;\ X\ else\ II_D\ fi)$
oops

lemma *IterateD-singleton:*

assumes $P\ is\ N$
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
 $\bigwedge i. P\ i\ is\ N \wedge i. Q\ i\ is\ N$
 $\bigwedge i. P\ i \sqsubseteq Q\ i$
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 segr-mono assms*)
done

lemma *IterateD-single-refine:*

assumes
 $P\ is\ N\ Q\ is\ N\ P \sqsubseteq Q$
shows $(do\ g \rightarrow P\ od) \sqsubseteq (do\ g \rightarrow Q\ od)$
oops

lemma *IterateD-refine-intro:*

fixes $V :: (nat, 'a)\ uexpr$
assumes *vwb-lens w*
shows
 $I \vdash_n (w : [I \wedge \neg (\bigvee i \in A \cdot g(i))]_{>}) \sqsubseteq$
 $do\ i \in A \cdot g(i) \rightarrow (I \wedge g(i)) \vdash_n (w : [I]_{>} \wedge [V]_{>} <_u [V]_{<})\ od$
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 $V :: (\text{nat}, 'a) \text{ uexpr}$

assumes *vwb-lens w*

shows

$I \vdash_n (w:([I \wedge \neg g]_{>})) \sqsubseteq$
do $g \rightarrow ((I \wedge g) \vdash_n (w:([I]_{>} \wedge [V]_{>} <_u [V]_{<})) \text{ od}$

apply (*rule order-trans*)

defer

apply (*rule IterateD-refine-intro[of w {0} λ i. g I V, simplified, OF assms(1)]*)

oops

4.5 Let and Local Variables

definition *LetD* $:: ('a, 'α) \text{ uexpr} \Rightarrow ('a \Rightarrow 'α \text{ hrel-des}) \Rightarrow 'α \text{ hrel-des}$ **where**
[upred-defs]: $\text{LetD } v \ P = (P \ x) \llbracket x \rightarrow [v]_{D<} \rrbracket$

syntax

-LetD $:: [\text{letbinds}, 'a] \Rightarrow 'a \quad ((\text{let}_D (-) / \text{in } (-)) [0, 10] 10)$

translations

-LetD (-binds b bs) e \equiv *-LetD b (-LetD bs e)*

let_D x = a in e \equiv *CONST LetD a (λx. e)*

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

$\text{LetD } v \ (\lambda \ x. \ p(x) \vdash_n \ Q(x)) = (p(x) \llbracket x \rightarrow v \rrbracket) \vdash_n \ (Q(x) \llbracket x \rightarrow [v]_{<} \rrbracket)$

by (*rel-auto*)

lemma *LetD-H1-H3-closed* [*closure*]:

$\llbracket \bigwedge \ x. \ P(x) \text{ is } \mathbf{N} \rrbracket \Longrightarrow \text{LetD } v \ P \text{ is } \mathbf{N}$

by (*rel-auto*)

4.6 Deep Local Variables

definition *des-local-state* $::$

'a::countable itself $\Rightarrow ((\text{nat}, 's) \text{ local-scheme } \text{des}, 's, \text{nat}, 'a::\text{countable}) \text{ local-prim}$ **where**
des-local-state t = (*sstate* = Σ_D , *sassigns* = *assigns-d*, *inj-local* = *nat-inj-univ*)

syntax

-des-local-state-type $:: \text{type} \Rightarrow \text{logic } (\mathcal{L}_D[-])$

-des-var-scope-type $:: \text{id} \Rightarrow \text{type} \Rightarrow \text{logic} \Rightarrow \text{logic } (\text{var}_D - :: - \cdot - [0, 0, 10] 10)$

translations

$\mathcal{L}_D['a] == \text{CONST } \text{des-local-state } \text{TYPE}('a)$

-des-var-scope-type x t P \Rightarrow *-var-scope-type (-des-local-state-type t) x t P*

var_D x :: 'a · P <= var[$\mathcal{L}_D['a]$] x · P

lemma *get-rel-local* [*lens-defs*]:

$\text{get}_S \mathcal{L}_D['a::\text{countable}] = \text{get}_{\Sigma_D}$

by (*simp add: des-local-state-def*)

lemma *des-local-state* [*simp*]: *utp-local-state* $\mathcal{L}_D['a::\text{countable}]$

by (*unfold-locales, simp-all add: upred-defs assigns-comp des-local-state-def, rel-auto*)

(metis local.cases-scheme)

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

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

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

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

lemma *msubst-H1-H3-closed* [closure]:
 $\llbracket \$ok \# v; out\alpha \# v; (\bigwedge x. P\ x\ is\ \mathbf{N}) \rrbracket \implies (P(x)\llbracket x \rightarrow v \rrbracket)$ is \mathbf{N}
 by (rel-auto, metis+)

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

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

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

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

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

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

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

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

end

5 Design Weakest Preconditions

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

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

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

theorem *wpd-eq-intro*: $\llbracket \bigwedge r. (p_1 \vdash_n Q_1) \text{ wp}_D r = (p_2 \vdash_n Q_2) \text{ wp}_D r \rrbracket \Longrightarrow (p_1 \vdash_n Q_1) = (p_2 \vdash_n Q_2)$
apply (*rel-simp robust*; *metis curry-conv*)
done

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

lemma *wp-assigns-d* [*wp*]: $\langle \sigma \rangle_D \text{ wp}_D r = \sigma \dagger r$
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*]:
 $(p \vdash_n Q) \text{ wp}_D r = (p \wedge Q \text{ wp } r)$
by (*simp add: ndesign-def rdesign-wp*)

theorem *wpd-seq-r*:
fixes $Q1 \ Q2 :: '\alpha \text{ hrel}$
shows $((\lfloor p1 \rfloor_{<} \vdash_r Q1) ;; (\lfloor p2 \rfloor_{<} \vdash_r Q2)) \text{ wp}_D r = (\lfloor p1 \rfloor_{<} \vdash_r Q1) \text{ wp}_D ((\lfloor p2 \rfloor_{<} \vdash_r Q2) \text{ wp}_D r)$
apply (*simp add: wp*)
apply (*subst rdesign-composition-wp*)
apply (*simp only: wp*)
apply (*rel-auto*)
done

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

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

end

6 Refinement Calculus

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

begin

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

syntax

-*init-var* :: *logic*
-*des-spec* :: *salpha* \Rightarrow *logic* \Rightarrow *logic* \Rightarrow *logic* ($:-$ [$-$,/ $-$] $_D$ [99,0,0] 100)
-*des-log-const* :: *pttrn* \Rightarrow *logic* \Rightarrow *logic* (con_D - · - [0, 10] 10)

translations

-*des-spec* $x\ p\ q \Rightarrow CONST\ des-spec\ x\ p\ (\lambda\ -init-var.\ q)$
-*des-spec* ($-salphaset\ (-salphamk\ x)$) $p\ q \Leftarrow CONST\ des-spec\ x\ p\ (\lambda\ iv.\ q)$
-*des-log-const* $x\ P \Rightarrow \sqcup\ x \cdot P$

parse-translation

\ll
let
 fun *init-var-tr* [] = *Syntax.free iv*
 | *init-var-tr* - = *raise Match*;
in
[[@{*syntax-const* -*init-var*}, *K init-var-tr*]]
end
 \gg

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 $V :: (nat, 'a)\ uexpr$
assumes *vwb-lens* w

```

shows  $w:[ivr, ivr \wedge \neg (\bigvee_{i \in A} \cdot g(i))]_D$ 
       $\sqsubseteq (do\ i \in A \cdot g(i) \rightarrow \bigsqcup\ iv \cdot w:[ivr \wedge g(i) \wedge \ll iv \gg =_u \&\mathbf{v}, ivr \wedge (V <_u V[\ll iv \gg / \mathbf{v}])]_D\ od)\ (\mathbf{is}$ 
 $\ ?lhs \sqsubseteq\ ?rhs)$ 
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 $OIH\text{-def}$ [*upred-defs*]

lemma $OIH\text{-design}$:

```

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 $OIH\text{-idem}$:

```

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 $OIH\text{-of-design}$:

```

 $\$ok' \# P \Longrightarrow OIH(\psi)(P \vdash Q) = (P \vdash (Q \wedge \psi))$ 
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 $OSH-as-OIH$:
 $OSH(\psi)(D) = OIH([\psi]_{<} \Rightarrow [\psi]_{>})(D)$
by (*simp add: OSH-def OIH-def, pred-auto*)

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

lemma $OSH-of-design$:
 $\llbracket \$ok' \# P; \$ok' \# Q \rrbracket \Longrightarrow OSH(\psi)(P \vdash Q) = (P \vdash (Q \wedge ([\psi]_{<} \Rightarrow [\psi]_{>})))$
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 $SIH-of-design$:
 $\llbracket \$ok' \# P; \$ok' \# Q; ok \# \psi \rrbracket \Longrightarrow SIH(\psi)(P \vdash Q) = ((P \wedge [\psi]_{<}) \vdash (Q \wedge [\psi]_{>}))$
by (*simp add: SIH-def OSH-of-design ISH-of-design unrest, pred-auto*)

end

8 Meta Theory for UTP Designs

theory *utp-designs*

imports

utp-des-core
utp-des-healths
utp-des-theory
utp-des-tactics
utp-des-prog
utp-des-wp
utp-des-refcalc
utp-des-invariants

begin end

References

- [1] A. Cavalcanti, A. Wellings, and J. Woodcock. The Safety-Critical Java memory model formalised. *Formal Aspects of Computing*, 25(1):37–57, 2012.
- [2] A. Cavalcanti and J. Woodcock. A tutorial introduction to designs in unifying theories of programming. In *Proc. 4th Intl. Conf. on Integrated Formal Methods (IFM)*, volume 2999 of *LNCS*, pages 40–66. Springer, 2004.
- [3] 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.