

Reactive Designs in Isabelle/UTP

Simon Foster James Baxter Ana Cavalcanti Jim Woodcock
Samuel Canham

April 19, 2018

Abstract

Reactive designs combine the UTP theories of reactive processes and designs to characterise reactive programs. Whereas sequential imperative programs are expected to run until termination, reactive programs pause at instances to allow interaction with the environment using abstract events, and often do not terminate at all. Thus, whereas a design describes the precondition and postcondition for a program, to characterise initial and final states, a reactive design also has a “pericondition”, which characterises intermediate quiescent observations. This gives rise to a notion of “reactive contract”, which specifies the assumptions a program makes of its environment, and the guarantees it will make of its own behaviour in both intermediate and final observations. This Isabelle/UTP document mechanises the UTP theory of reactive designs, including its healthiness conditions, signature, and a large library of algebraic laws of reactive programming.

Contents

1	Introduction	3
2	Reactive Designs Healthiness Conditions	3
2.1	Preliminaries	3
2.2	Identities	3
2.3	RD1: Divergence yields arbitrary traces	4
2.4	R3c and R3h: Reactive design versions of R3	6
2.5	RD2: A reactive specification cannot require non-termination	9
2.6	Major healthiness conditions	10
2.7	UTP theories	13
3	Reactive Design Specifications	15
3.1	Reactive design forms	15
3.2	Auxiliary healthiness conditions	17
3.3	Composition laws	18
3.4	Refinement introduction laws	25
3.5	Distribution laws	26
4	Reactive Design Triples	26
4.1	Diamond notation	26
4.2	Export laws	27
4.3	Pre-, peri-, and postconditions	28
4.3.1	Definitions	28
4.3.2	Unrestriction laws	28

4.3.3	Substitution laws	29
4.3.4	Healthiness laws	30
4.3.5	Calculation laws	33
4.4	Formation laws	35
4.4.1	Order laws	36
4.5	Composition laws	37
4.6	Refinement introduction laws	41
4.7	Closure laws	42
4.8	Distribution laws	43
4.9	Algebraic laws	45
4.10	Recursion laws	46
5	Normal Reactive Designs	47
5.1	UTP theory	60
6	Syntax for reactive design contracts	61
7	Reactive design tactics	62
8	Reactive design parallel-by-merge	64
8.1	Example basic merge	78
8.2	Simple parallel composition	78
9	Productive Reactive Designs	79
9.1	Healthiness condition	79
9.2	Reactive design calculations	81
9.3	Closure laws	82
10	Guarded Recursion	84
10.1	Traces with a size measure	84
10.2	Guardedness	85
10.3	Tail recursive fixed-point calculations	89
11	Reactive Design Programs	90
11.1	State substitution	91
11.2	Assignment	91
11.3	Conditional	92
11.4	Assumptions	93
11.5	Guarded commands	94
11.6	Generalised Alternation	94
11.7	Choose	96
11.8	State Abstraction	96
11.9	While Loop	97
11.10	Iteration Construction	101
11.11	Substitution Laws	102
11.12	Algebraic Laws	102
11.13	Lifting designs to reactive designs	103
12	Instantaneous Reactive Designs	105
13	Meta-theory for Reactive Designs	108

1 Introduction

This document contains a mechanisation in Isabelle/UTP [3] of our theory of reactive designs. Reactive designs form an important semantic foundation for reactive modelling languages such as Circus [5]. For more details of this work, please see our recent paper [2].

2 Reactive Designs Healthiness Conditions

```
theory utp-rdes-healths
  imports UTP-Reactive.utp-reactive
begin
```

2.1 Preliminaries

```
named-theorems rdes and rdes-def and RD-elim
```

```
type-synonym ('s, 't) rdes = ('s, 't, unit) hrel-rsp
```

```
translations
  (type) ('s, 't) rdes <= (type) ('s, 't, unit) hrel-rsp
```

```
lemma R2-st-ex: R2 ( $\exists$  $st  $\cdot$  P) = ( $\exists$  $st  $\cdot$  R2(P))
  by (rel-auto)
```

```
lemma R2s-st'-eq-st:
  R2s($st' =u $st) = ($st' =u $st)
  by (rel-auto)
```

```
lemma R2c-st'-eq-st:
  R2c($st' =u $st) = ($st' =u $st)
  by (rel-auto)
```

```
lemma R1-des-lift-skip: R1( $\lceil$  II  $\rceil_D$ ) =  $\lceil$  II  $\rceil_D$ 
  by (rel-auto)
```

```
lemma R2-des-lift-skip:
  R2( $\lceil$  II  $\rceil_D$ ) =  $\lceil$  II  $\rceil_D$ 
  apply (rel-auto) using minus-zero-eq by blast
```

```
lemma R1-R2c-ex-st: R1 (R2c ( $\exists$  $st'  $\cdot$  Q1)) = ( $\exists$  $st'  $\cdot$  R1 (R2c Q1))
  by (rel-auto)
```

2.2 Identities

We define two identities fro reactive designs, which correspond to the regular and state-sensitive versions of reactive designs, respectively. The former is the one used in the UTP book and related publications for CSP.

```
definition skip-rea :: ('t::trace, 'α) hrel-rp (IIc) where
skip-rea-def [urel-defs]: IIc = (II  $\vee$  ( $\neg$  $ok  $\wedge$  $tr  $\leq_u$  $tr'))
```

```
definition skip-srea :: ('s, 't::trace, 'α) hrel-rsp (IIR) where
skip-srea-def [urel-defs]: IIR = (( $\exists$  $st  $\cdot$  IIc)  $\triangleleft$  $wait  $\triangleright$  IIc)
```

```
lemma skip-rea-R1-lemma: IIc = R1($ok  $\Rightarrow$  II)
```

by (*rel-auto*)

lemma *skip-rea-form*: $II_c = (II \triangleleft \$ok \triangleright R1(true))$
by (*rel-auto*)

lemma *skip-srea-form*: $II_R = ((\exists \$st \cdot II) \triangleleft \$wait \triangleright II) \triangleleft \$ok \triangleright R1(true)$
by (*rel-auto*)

lemma *R1-skip-rea*: $R1(II_c) = II_c$
by (*rel-auto*)

lemma *R2c-skip-rea*: $R2c\ II_c = II_c$
by (*simp add: skip-rea-def R2c-and R2c-disj R2c-skip-r R2c-not R2c-ok R2c-tr'-ge-tr*)

lemma *R2-skip-rea*: $R2(II_c) = II_c$
by (*metis R1-R2c-is-R2 R1-skip-rea R2c-skip-rea*)

lemma *R2c-skip-srea*: $R2c(II_R) = II_R$
apply (*rel-auto*) **using** *minus-zero-eq* **by** *blast+*

lemma *skip-srea-R1 [closure]*: II_R is $R1$
by (*rel-auto*)

lemma *skip-srea-R2c [closure]*: II_R is $R2c$
by (*simp add: Healthy-def R2c-skip-srea*)

lemma *skip-srea-R2 [closure]*: II_R is $R2$
by (*metis Healthy-def' R1-R2c-is-R2 R2c-skip-srea skip-srea-R1*)

2.3 RD1: Divergence yields arbitrary traces

definition $RD1 :: ('t::trace, 'α, 'β) \text{ rel-rp} \Rightarrow ('t, 'α, 'β) \text{ rel-rp}$ **where**
 $[upred-defs]: RD1(P) = (P \vee (\neg \$ok \wedge \$tr \leq_u \$tr'))$

$RD1$ is essentially $H1$ from the theory of designs, but viewed through the prism of reactive processes.

lemma *RD1-idem*: $RD1(RD1(P)) = RD1(P)$
by (*rel-auto*)

lemma *RD1-Idempotent*: *Idempotent* $RD1$
by (*simp add: Idempotent-def RD1-idem*)

lemma *RD1-mono*: $P \sqsubseteq Q \implies RD1(P) \sqsubseteq RD1(Q)$
by (*rel-auto*)

lemma *RD1-Monotonic*: *Monotonic* $RD1$
using *mono-def RD1-mono* **by** *blast*

lemma *RD1-Continuous*: *Continuous* $RD1$
by (*rel-auto*)

lemma *R1-true-RD1-closed [closure]*: $R1(true)$ is $RD1$
by (*rel-auto*)

lemma *RD1-wait-false [closure]*: P is $RD1 \implies P \llbracket false/\$wait \rrbracket$ is $RD1$

by (rel-auto)

lemma *RD1-wait'-false* [closure]: P is *RD1* $\implies P \llbracket \text{false} / \$\text{wait}' \rrbracket$ is *RD1*
 by (rel-auto)

lemma *RD1-seq*: $RD1(RD1(P) ;; RD1(Q)) = RD1(P) ;; RD1(Q)$
 by (rel-auto)

lemma *RD1-seq-closure* [closure]: $\llbracket P \text{ is } RD1; Q \text{ is } RD1 \rrbracket \implies P ;; Q \text{ is } RD1$
 by (metis *Healthy-def'* *RD1-seq*)

lemma *RD1-R1-commute*: $RD1(R1(P)) = R1(RD1(P))$
 by (rel-auto)

lemma *RD1-R2c-commute*: $RD1(R2c(P)) = R2c(RD1(P))$
 by (rel-auto)

lemma *RD1-via-R1*: $R1(H1(P)) = RD1(R1(P))$
 by (rel-auto)

lemma *RD1-R1-cases*: $RD1(R1(P)) = (R1(P) \triangleleft \$\text{ok} \triangleright R1(\text{true}))$
 by (rel-auto)

lemma *skip-rea-RD1-skip*: $II_c = RD1(II)$
 by (rel-auto)

lemma *skip-srea-RD1* [closure]: II_R is *RD1*
 by (rel-auto)

lemma *RD1-algebraic-intro*:
 assumes
 $P \text{ is } R1 (R1(\text{true}_h) ;; P) = R1(\text{true}_h) (II_c ;; P) = P$
 shows $P \text{ is } RD1$
proof –
 have $P = (II_c ;; P)$
 by (simp add: *assms(3)*)
 also have $\dots = (R1(\$ok \Rightarrow II) ;; P)$
 by (simp add: *skip-rea-R1-lemma*)
 also have $\dots = (((\neg \$ok \wedge R1(\text{true})) ;; P) \vee P)$
 by (metis (no-types, lifting) *R1-def* *seqr-left-unit* *seqr-or-distl* *skip-rea-R1-lemma* *skip-rea-def* *utp-pred-laws.inf-top-left* *utp-pred-laws.sup-commute*)
 also have $\dots = ((R1(\neg \$ok) ;; (R1(\text{true}_h) ;; P)) \vee P)$
 using *dual-order.trans* by (rel-blast)
 also have $\dots = ((R1(\neg \$ok) ;; R1(\text{true}_h)) \vee P)$
 by (simp add: *assms(2)*)
 also have $\dots = (R1(\neg \$ok) \vee P)$
 by (rel-auto)
 also have $\dots = RD1(P)$
 by (rel-auto)
 finally show ?thesis
 by (simp add: *Healthy-def*)
qed

theorem *RD1-left-zero*:
 assumes $P \text{ is } R1$ $P \text{ is } RD1$

shows $(R1(true) ;; P) = R1(true)$
proof –
have $(R1(true) ;; R1(RD1(P))) = R1(true)$
by *(rel-auto)*
thus *?thesis*
by *(simp add: Healthy-if assms(1) assms(2))*
qed

theorem *RD1-left-unit*:
assumes *P is R1 P is RD1*
shows $(II_c ;; P) = P$
proof –
have $(II_c ;; R1(RD1(P))) = R1(RD1(P))$
by *(rel-auto)*
thus *?thesis*
by *(simp add: Healthy-if assms(1) assms(2))*
qed

lemma *RD1-alt-def*:
assumes *P is R1*
shows $R1(P) = (P \triangleleft \$ok \triangleright R1(true))$
proof –
have $R1(R1(P)) = (R1(P) \triangleleft \$ok \triangleright R1(true))$
by *(rel-auto)*
thus *?thesis*
by *(simp add: Healthy-if assms)*
qed

theorem *RD1-algebraic*:
assumes *P is R1*
shows $P \text{ is } RD1 \iff (R1(true_h) ;; P) = R1(true_h) \wedge (II_c ;; P) = P$
using *RD1-algebraic-intro RD1-left-unit RD1-left-zero assms* **by** *blast*

2.4 R3c and R3h: Reactive design versions of R3

definition *R3c* :: $(t::trace, 'a) \text{ hrel-rp} \Rightarrow (t, 'a) \text{ hrel-rp}$ **where**
 $[upred-defs]: R3c(P) = (II_c \triangleleft \$wait \triangleright P)$

definition *R3h* :: $(s, t::trace, 'a) \text{ hrel-rsp} \Rightarrow (s, t, 'a) \text{ hrel-rsp}$ **where**
 $R3h-def [upred-defs]: R3h(P) = ((\exists \$st \cdot II_c) \triangleleft \$wait \triangleright P)$

lemma *R3c-idem*: $R3c(R3c(P)) = R3c(P)$
by *(rel-auto)*

lemma *R3c-Idempotent*: *Idempotent R3c*
by *(simp add: Idempotent-def R3c-idem)*

lemma *R3c-mono*: $P \sqsubseteq Q \implies R3c(P) \sqsubseteq R3c(Q)$
by *(rel-auto)*

lemma *R3c-Monotonic*: *Monotonic R3c*
by *(simp add: mono-def R3c-mono)*

lemma *R3c-Continuous*: *Continuous R3c*
by *(rel-auto)*

lemma *R3h-idem*: $R3h(R3h(P)) = R3h(P)$
by (*rel-auto*)

lemma *R3h-Idempotent*: *Idempotent R3h*
by (*simp add: Idempotent-def R3h-idem*)

lemma *R3h-mono*: $P \sqsubseteq Q \implies R3h(P) \sqsubseteq R3h(Q)$
by (*rel-auto*)

lemma *R3h-Monotonic*: *Monotonic R3h*
by (*simp add: mono-def R3h-mono*)

lemma *R3h-Continuous*: *Continuous R3h*
by (*rel-auto*)

lemma *R3h-inf*: $R3h(P \sqcap Q) = R3h(P) \sqcap R3h(Q)$
by (*rel-auto*)

lemma *R3h-UINF*:
 $A \neq \{\} \implies R3h(\bigsqcap i \in A \cdot P(i)) = (\bigsqcap i \in A \cdot R3h(P(i)))$
by (*rel-auto*)

lemma *R3h-cond*: $R3h(P \triangleleft b \triangleright Q) = (R3h(P) \triangleleft b \triangleright R3h(Q))$
by (*rel-auto*)

lemma *R3c-via-RD1-R3*: $RD1(R3(P)) = R3c(RD1(P))$
by (*rel-auto*)

lemma *R3c-RD1-def*: $P \text{ is } RD1 \implies R3c(P) = RD1(R3(P))$
by (*simp add: Healthy-if R3c-via-RD1-R3*)

lemma *RD1-R3c-commute*: $RD1(R3c(P)) = R3c(RD1(P))$
by (*rel-auto*)

lemma *R1-R3c-commute*: $R1(R3c(P)) = R3c(R1(P))$
by (*rel-auto*)

lemma *R2c-R3c-commute*: $R2c(R3c(P)) = R3c(R2c(P))$
apply (*rel-auto*) **using** *minus-zero-eq* **by** *blast+*

lemma *R1-R3h-commute*: $R1(R3h(P)) = R3h(R1(P))$
by (*rel-auto*)

lemma *R2c-R3h-commute*: $R2c(R3h(P)) = R3h(R2c(P))$
apply (*rel-auto*) **using** *minus-zero-eq* **by** *blast+*

lemma *RD1-R3h-commute*: $RD1(R3h(P)) = R3h(RD1(P))$
by (*rel-auto*)

lemma *R3c-cancels-R3*: $R3c(R3(P)) = R3c(P)$
by (*rel-auto*)

lemma *R3-cancels-R3c*: $R3(R3c(P)) = R3(P)$
by (*rel-auto*)

lemma *R3h-cancels-R3c*: $R3h(R3c(P)) = R3h(P)$
by (*rel-auto*)

lemma *R3c-semir-form*:
 $(R3c(P) ;; R3c(R1(Q))) = R3c(P ;; R3c(R1(Q)))$
by (*rel-simp, safe, auto intro: order-trans*)

lemma *R3h-semir-form*:
 $(R3h(P) ;; R3h(R1(Q))) = R3h(P ;; R3h(R1(Q)))$
by (*rel-simp, safe, auto intro: order-trans, blast+*)

lemma *R3c-seq-closure*:
assumes P is $R3c$ Q is $R3c$ Q is $R1$
shows $(P ;; Q)$ is $R3c$
by (*metis Healthy-def' R3c-semir-form assms*)

lemma *R3h-seq-closure [closure]*:
assumes P is $R3h$ Q is $R3h$ Q is $R1$
shows $(P ;; Q)$ is $R3h$
by (*metis Healthy-def' R3h-semir-form assms*)

lemma *R3c-R3-left-seq-closure*:

assumes P is $R3$ Q is $R3c$
shows $(P ;; Q)$ is $R3c$

proof –

have $(P ;; Q) = ((P ;; Q) \llbracket \text{true}/\$wait \rrbracket \triangleleft \$wait \triangleright (P ;; Q))$

by (*metis cond-var-split cond-var-subst-right in-var-uvar wait-vwb-lens*)

also have $\dots = (((II \triangleleft \$wait \triangleright P) ;; Q) \llbracket \text{true}/\$wait \rrbracket \triangleleft \$wait \triangleright (P ;; Q))$

by (*metis Healthy-def' R3-def assms(1)*)

also have $\dots = ((II \llbracket \text{true}/\$wait \rrbracket ;; Q) \triangleleft \$wait \triangleright (P ;; Q))$

by (*subst-tac*)

also have $\dots = (((II \wedge \$wait') ;; Q) \triangleleft \$wait \triangleright (P ;; Q))$

by (*metis (no-types, lifting) cond-def conj-pos-var-subst seqr-pre-var-out skip-var utp-pred-laws.inf-left-idem wait-vwb-lens*)

also have $\dots = ((II \llbracket \text{true}/\$wait' \rrbracket ;; Q \llbracket \text{true}/\$wait \rrbracket) \triangleleft \$wait \triangleright (P ;; Q))$

by (*metis seqr-pre-transfer seqr-right-one-point true-alt-def uovar-convr upred-eq-true utp-rel.unrest-ouvar vwb-lens-mwb wait-vwb-lens*)

also have $\dots = ((II \llbracket \text{true}/\$wait' \rrbracket ;; (II_c \triangleleft \$wait \triangleright Q) \llbracket \text{true}/\$wait \rrbracket) \triangleleft \$wait \triangleright (P ;; Q))$

by (*metis Healthy-def' R3c-def assms(2)*)

also have $\dots = ((II \llbracket \text{true}/\$wait' \rrbracket ;; II_c \llbracket \text{true}/\$wait \rrbracket) \triangleleft \$wait \triangleright (P ;; Q))$

by (*subst-tac*)

also have $\dots = (((II \wedge \$wait') ;; II_c) \triangleleft \$wait \triangleright (P ;; Q))$

by (*metis seqr-pre-transfer seqr-right-one-point true-alt-def uovar-convr upred-eq-true utp-rel.unrest-ouvar vwb-lens-mwb wait-vwb-lens*)

also have $\dots = ((II ;; II_c) \triangleleft \$wait \triangleright (P ;; Q))$

by (*simp add: cond-def seqr-pre-transfer utp-rel.unrest-ouvar*)

also have $\dots = (II_c \triangleleft \$wait \triangleright (P ;; Q))$

by *simp*

also have $\dots = R3c(P ;; Q)$

by (*simp add: R3c-def*)

finally show *?thesis*

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

qed

lemma *R3c-cases*: $R3c(P) = ((II \triangleleft \$ok \triangleright R1(\text{true})) \triangleleft \$wait \triangleright P)$

by (rel-auto)

lemma *R3h-cases*: $R3h(P) = (((\exists \$st \cdot II) \triangleleft \$ok \triangleright R1(true)) \triangleleft \$wait \triangleright P)$
 by (rel-auto)

lemma *R3h-form*: $R3h(P) = II_R \triangleleft \$wait \triangleright P$
 by (rel-auto)

lemma *R3c-subst-wait*: $R3c(P) = R3c(P_f)$
 by (simp add: R3c-def cond-var-subst-right)

lemma *R3h-subst-wait*: $R3h(P) = R3h(P_f)$
 by (simp add: R3h-cases cond-var-subst-right)

lemma *skip-srea-R3h [closure]*: II_R is *R3h*
 by (rel-auto)

lemma *R3h-wait-true*:
 assumes P is *R3h*
 shows $P_t = II_R t$
proof –
 have $P_t = (II_R \triangleleft \$wait \triangleright P)_t$
 by (metis Healthy-if R3h-form assms)
 also have $\dots = II_R t$
 by (simp add: usubst)
 finally show ?thesis .
qed

2.5 RD2: A reactive specification cannot require non-termination

definition *RD2* where
 [upred-defs]: $RD2(P) = H2(P)$

RD2 is just *H2* since the type system will automatically have J identifying the reactive variables as required.

lemma *RD2-idem*: $RD2(RD2(P)) = RD2(P)$
 by (simp add: H2-idem RD2-def)

lemma *RD2-Idempotent*: *Idempotent RD2*
 by (simp add: Idempotent-def RD2-idem)

lemma *RD2-mono*: $P \sqsubseteq Q \implies RD2(P) \sqsubseteq RD2(Q)$
 by (simp add: H2-def RD2-def seqr-mono)

lemma *RD2-Monotonic*: *Monotonic RD2*
 using mono-def RD2-mono by blast

lemma *RD2-Continuous*: *Continuous RD2*
 by (rel-auto)

lemma *RD1-RD2-commute*: $RD1(RD2(P)) = RD2(RD1(P))$
 by (rel-auto)

lemma *RD2-R3c-commute*: $RD2(R3c(P)) = R3c(RD2(P))$
 by (rel-auto)

lemma *RD2-R3h-commute*: $RD2(R3h(P)) = R3h(RD2(P))$
by (*rel-auto*)

2.6 Major healthiness conditions

definition *RH* :: $(t::trace, 'α) \text{ hrel-rp} \Rightarrow (t, 'α) \text{ hrel-rp}$ (**R**)
where [*upred-defs*]: $RH(P) = R1(R2c(R3c(P)))$

definition *RHS* :: $(s, t::trace, 'α) \text{ hrel-rsp} \Rightarrow (s, t, 'α) \text{ hrel-rsp}$ (**R_s**)
where [*upred-defs*]: $RHS(P) = R1(R2c(R3h(P)))$

definition *RD* :: $(t::trace, 'α) \text{ hrel-rp} \Rightarrow (t, 'α) \text{ hrel-rp}$
where [*upred-defs*]: $RD(P) = RD1(RD2(RP(P)))$

definition *SRD* :: $(s, t::trace, 'α) \text{ hrel-rsp} \Rightarrow (s, t, 'α) \text{ hrel-rsp}$
where [*upred-defs*]: $SRD(P) = RD1(RD2(RHS(P)))$

lemma *RH-comp*: $RH = R1 \circ R2c \circ R3c$
by (*auto simp add: RH-def*)

lemma *RHS-comp*: $RHS = R1 \circ R2c \circ R3h$
by (*auto simp add: RHS-def*)

lemma *RD-comp*: $RD = RD1 \circ RD2 \circ RP$
by (*auto simp add: RD-def*)

lemma *SRD-comp*: $SRD = RD1 \circ RD2 \circ RHS$
by (*auto simp add: SRD-def*)

lemma *RH-idem*: $\mathbf{R}(\mathbf{R}(P)) = \mathbf{R}(P)$
by (*simp add: R1-R2c-commute R1-R3c-commute R1-idem R2c-R3c-commute R2c-idem R3c-idem RH-def*)

lemma *RH-Idempotent*: *Idempotent* **R**
by (*simp add: Idempotent-def RH-idem*)

lemma *RH-Monotonic*: *Monotonic* **R**
by (*metis (no-types, lifting) R1-Monotonic R2c-Monotonic R3c-mono RH-def mono-def*)

lemma *RH-Continuous*: *Continuous* **R**
by (*simp add: Continuous-comp R1-Continuous R2c-Continuous R3c-Continuous RH-comp*)

lemma *RHS-idem*: $\mathbf{R}_s(\mathbf{R}_s(P)) = \mathbf{R}_s(P)$
by (*simp add: R1-R2c-is-R2 R1-R3h-commute R2-idem R2c-R3h-commute R3h-idem RHS-def*)

lemma *RHS-Idempotent* [*closure*]: *Idempotent* **R_s**
by (*simp add: Idempotent-def RHS-idem*)

lemma *RHS-Monotonic*: *Monotonic* **R_s**
by (*simp add: mono-def R1-R2c-is-R2 R2-mono R3h-mono RHS-def*)

lemma *RHS-mono*: $P \sqsubseteq Q \implies \mathbf{R}_s(P) \sqsubseteq \mathbf{R}_s(Q)$
using *mono-def RHS-Monotonic* **by** *blast*

lemma *RHS-Continuous* [*closure*]: *Continuous* **R_s**

by (simp add: Continuous-comp R1-Continuous R2c-Continuous R3h-Continuous RHS-comp)

lemma *RHS-inf*: $\mathbf{R}_s(P \sqcap Q) = \mathbf{R}_s(P) \sqcap \mathbf{R}_s(Q)$
 using Continuous-Disjunctous Disjunctuous-def RHS-Continuous by auto

lemma *RHS-INF*:
 $A \neq \{\} \implies \mathbf{R}_s(\bigsqcap i \in A \cdot P(i)) = (\bigsqcap i \in A \cdot \mathbf{R}_s(P(i)))$
 by (simp add: RHS-def R3h-UIINF R2c-USUP R1-USUP)

lemma *RHS-sup*: $\mathbf{R}_s(P \sqcup Q) = \mathbf{R}_s(P) \sqcup \mathbf{R}_s(Q)$
 by (rel-auto)

lemma *RHS-SUP*:
 $A \neq \{\} \implies \mathbf{R}_s(\bigsqcup i \in A \cdot P(i)) = (\bigsqcup i \in A \cdot \mathbf{R}_s(P(i)))$
 by (rel-auto)

lemma *RHS-cond*: $\mathbf{R}_s(P \triangleleft b \triangleright Q) = (\mathbf{R}_s(P) \triangleleft R2c\ b \triangleright \mathbf{R}_s(Q))$
 by (simp add: RHS-def R3h-cond R2c-condr R1-cond)

lemma *RD-alt-def*: $RD(P) = RD1(RD2(\mathbf{R}(P)))$
 by (simp add: R3c-via-RD1-R3 RD1-R1-commute RD1-R2c-commute RD1-R3c-commute RD1-RD2-commute
 RH-def RD-def RP-def)

lemma *RD1-RH-commute*: $RD1(\mathbf{R}(P)) = \mathbf{R}(RD1(P))$
 by (simp add: RD1-R1-commute RD1-R2c-commute RD1-R3c-commute RH-def)

lemma *RD2-RH-commute*: $RD2(\mathbf{R}(P)) = \mathbf{R}(RD2(P))$
 by (metis R1-H2-commute R2c-H2-commute RD2-R3c-commute RD2-def RH-def)

lemma *RD-idem*: $RD(RD(P)) = RD(P)$
 by (simp add: RD-alt-def RD1-RH-commute RD2-RH-commute RD1-RD2-commute RD2-idem RD1-idem
 RH-idem)

lemma *RD-Monotonic*: *Monotonic RD*
 by (simp add: Monotonic-comp RD1-Monotonic RD2-Monotonic RD-comp RP-Monotonic)

lemma *RD-Continuous*: *Continuous RD*
 by (simp add: Continuous-comp RD1-Continuous RD2-Continuous RD-comp RP-Continuous)

lemma *R3-RD-RP*: $R3(RD(P)) = RP(RD1(RD2(P)))$
 by (metis (no-types, lifting) R1-R2c-is-R2 R2-R3-commute R3-cancels-R3c RD1-RH-commute RD2-RH-commute
 RD-alt-def RH-def RP-def)

lemma *RD1-RHS-commute*: $RD1(\mathbf{R}_s(P)) = \mathbf{R}_s(RD1(P))$
 by (simp add: RD1-R1-commute RD1-R2c-commute RD1-R3h-commute RHS-def)

lemma *RD2-RHS-commute*: $RD2(\mathbf{R}_s(P)) = \mathbf{R}_s(RD2(P))$
 by (metis R1-H2-commute R2c-H2-commute RD2-R3h-commute RD2-def RHS-def)

lemma *SRD-idem*: $SRD(SRD(P)) = SRD(P)$
 by (simp add: RD1-RD2-commute RD1-RHS-commute RD1-idem RD2-RHS-commute RD2-idem RHS-idem
 SRD-def)

lemma *SRD-Idempotent [closure]*: *Idempotent SRD*
 by (simp add: Idempotent-def SRD-idem)

lemma *SRD-Monotonic: Monotonic SRD*

by (simp add: Monotonic-comp RD1-Monotonic RD2-Monotonic RHS-Monotonic SRD-comp)

lemma *SRD-Continuous [closure]: Continuous SRD*

by (simp add: Continuous-comp RD1-Continuous RD2-Continuous RHS-Continuous SRD-comp)

lemma *SRD-RHS-H1-H2: $SRD(P) = \mathbf{R}_s(\mathbf{H}(P))$*

by (rel-auto)

lemma *SRD-healths [closure]:*

assumes *P is SRD*

shows *P is R1 P is R2 P is R3h P is RD1 P is RD2*

apply (metis Healthy-def R1-idem RD1-RHS-commute RD2-RHS-commute RHS-def SRD-def assms)

apply (metis Healthy-def R1-R2c-is-R2 R2-idem RD1-RHS-commute RD2-RHS-commute RHS-def SRD-def assms)

apply (metis Healthy-def R1-R3h-commute R2c-R3h-commute R3h-idem RD1-R3h-commute RD2-R3h-commute RHS-def SRD-def assms)

apply (metis Healthy-def' RD1-idem SRD-def assms)

apply (metis Healthy-def' RD1-RD2-commute RD2-idem SRD-def assms)

done

lemma *SRD-intro:*

assumes *P is R1 P is R2 P is R3h P is RD1 P is RD2*

shows *P is SRD*

by (metis Healthy-def R1-R2c-is-R2 RHS-def SRD-def assms(2) assms(3) assms(4) assms(5))

lemma *SRD-ok-false [usubst]: $P \text{ is } SRD \implies P \llbracket \text{false}/\$ok \rrbracket = R1(\text{true})$*

by (metis (no-types, hide-lams) H1-H2-eq-design Healthy-def R1-ok-false RD1-R1-commute RD1-via-R1 RD2-def SRD-def SRD-healths(1) design-ok-false)

lemma *SRD-ok-true-wait-true [usubst]:*

assumes *P is SRD*

shows $P \llbracket \text{true}, \text{true}/\$ok, \$wait \rrbracket = (\exists \$st \cdot II) \llbracket \text{true}, \text{true}/\$ok, \$wait \rrbracket$

proof –

have $P = (\exists \$st \cdot II) \triangleleft \$ok \triangleright R1 \text{ true} \triangleleft \$wait \triangleright P$

by (metis Healthy-def R3h-cases SRD-healths(3) assms)

moreover have $((\exists \$st \cdot II) \triangleleft \$ok \triangleright R1 \text{ true} \triangleleft \$wait \triangleright P) \llbracket \text{true}, \text{true}/\$ok, \$wait \rrbracket = (\exists \$st \cdot II) \llbracket \text{true}, \text{true}/\$ok, \$wait \rrbracket$

by (simp add: usubst)

ultimately show ?thesis

by (simp)

qed

lemma *SRD-left-zero-1: $P \text{ is } SRD \implies R1(\text{true}) ;; P = R1(\text{true})$*

by (simp add: RD1-left-zero SRD-healths(1) SRD-healths(4))

lemma *SRD-left-zero-2:*

assumes *P is SRD*

shows $(\exists \$st \cdot II) \llbracket \text{true}, \text{true}/\$ok, \$wait \rrbracket ;; P = (\exists \$st \cdot II) \llbracket \text{true}, \text{true}/\$ok, \$wait \rrbracket$

proof –

have $(\exists \$st \cdot II) \llbracket \text{true}, \text{true}/\$ok, \$wait \rrbracket ;; R3h(P) = (\exists \$st \cdot II) \llbracket \text{true}, \text{true}/\$ok, \$wait \rrbracket$

by (rel-auto)

thus ?thesis

by (simp add: Healthy-if SRD-healths(3) assms)

qed

2.7 UTP theories

We create two theory objects: one for reactive designs and one for stateful reactive designs.

typeddecl *RDES*

typeddecl *SRDES*

abbreviation *RDES* \equiv *UTHY*(*RDES*, ('t::trace,' α) *rp*)

abbreviation *SRDES* \equiv *UTHY*(*SRDES*, ('s,'t::trace,' α) *rsp*)

overloading

rdes-hcond == *utp-hcond* :: (*RDES*, ('t::trace,' α) *rp*) *uthy* \Rightarrow (('t,' α) *rp* \times ('t,' α) *rp*) *health*
srdes-hcond == *utp-hcond* :: (*SRDES*, ('s,'t::trace,' α) *rsp*) *uthy* \Rightarrow (('s,'t,' α) *rsp* \times ('s,'t,' α) *rsp*) *health*

begin

definition *rdes-hcond* :: (*RDES*, ('t::trace,' α) *rp*) *uthy* \Rightarrow (('t,' α) *rp* \times ('t,' α) *rp*) *health* **where**
 [upred-defs]: *rdes-hcond* *T* = *RD*

definition *srdes-hcond* :: (*SRDES*, ('s,'t::trace,' α) *rsp*) *uthy* \Rightarrow (('s,'t,' α) *rsp* \times ('s,'t,' α) *rsp*) *health*
where

[upred-defs]: *srdes-hcond* *T* = *SRD*

end

interpretation *rdes-theory*: *utp-theory* *UTHY*(*RDES*, ('t::trace,' α) *rp*)

by (*unfold-locales*, *simp-all* add: *rdes-hcond-def* *RD-idem*)

interpretation *rdes-theory-continuous*: *utp-theory-continuous* *UTHY*(*RDES*, ('t::trace,' α) *rp*)

rewrites $\bigwedge P. P \in \text{carrier } (\text{uthy-order } RDES) \longleftrightarrow P \text{ is } RD$

and *carrier* (*uthy-order* *RDES*) \rightarrow *carrier* (*uthy-order* *RDES*) $\equiv \llbracket RD \rrbracket_H \rightarrow \llbracket RD \rrbracket_H$

and *le* (*uthy-order* *RDES*) = *op* \sqsubseteq

and *eq* (*uthy-order* *RDES*) = *op* =

by (*unfold-locales*, *simp-all* add: *rdes-hcond-def* *RD-Continuous*)

interpretation *rdes-rea-galois*:

galois-connection (*RDES* $\leftarrow \langle RD1 \circ RD2, R3 \rangle \rightarrow$ *REA*)

proof (*simp* add: *mk-conn-def*, *rule* *galois-connectionI'*, *simp-all* add: *utp-partial-order* *rdes-hcond-def* *rea-hcond-def*)

show *R3* $\in \llbracket RD \rrbracket_H \rightarrow \llbracket RP \rrbracket_H$

by (*metis* (*no-types*, *lifting*) *Healthy-def'* *Pi-I* *R3-RD-RP* *RP-idem* *mem-Collect-eq*)

show *RD1* \circ *RD2* $\in \llbracket RP \rrbracket_H \rightarrow \llbracket RD \rrbracket_H$

by (*simp* add: *Pi-iff* *Healthy-def*, *metis* *RD-def* *RD-idem*)

show *isotone* (*utp-order* *RD*) (*utp-order* *RP*) *R3*

by (*simp* add: *R3-Monotonic* *isotone-utp-orderI*)

show *isotone* (*utp-order* *RP*) (*utp-order* *RD*) (*RD1* \circ *RD2*)

by (*simp* add: *Monotonic-comp* *RD1-Monotonic* *RD2-Monotonic* *isotone-utp-orderI*)

fix *P* :: ('a, 'b) *hrel-rp*

assume *P* is *RD*

thus *P* \sqsubseteq *RD1* (*RD2* (*R3* *P*))

by (*metis* *Healthy-if* *R3-RD-RP* *RD-def* *RP-idem* *eq-iff*)

next

fix *P* :: ('a, 'b) *hrel-rp*

assume *a*: *P* is *RP*

thus *R3* (*RD1* (*RD2* *P*)) \sqsubseteq *P*

proof –

have *R3* (*RD1* (*RD2* *P*)) = *RP* (*RD1* (*RD2*(*P*)))

by (metis Healthy-if R3-RD-RP RD-def a)
 moreover have $RD1(RD2(P)) \sqsubseteq P$
 by (rel-auto)
 ultimately show ?thesis
 by (metis Healthy-if RP-mono a)
 qed
 qed

interpretation *rdes-rea-retract*:

retract ($RDES \leftarrow \langle RD1 \circ RD2, R3 \rangle \rightarrow REA$)
 by (unfold-locales, simp-all add: mk-conn-def utp-partial-order rdes-hcond-def rea-hcond-def)
 (metis Healthy-if R3-RD-RP RD-def RP-idem eq-refl)

interpretation *srdes-theory*: utp-theory UTHY($SRDES, ('s, 't::trace, 'α) rsp$)

by (unfold-locales, simp-all add: srdes-hcond-def SRD-idem)

interpretation *srdes-theory-continuous*: utp-theory-continuous UTHY($SRDES, ('s, 't::trace, 'α) rsp$)

rewrites $\bigwedge P. P \in \text{carrier (uthy-order } SRDES) \longleftrightarrow P \text{ is } SRD$
 and $P \text{ is } \mathcal{H}_{SRDES} \longleftrightarrow P \text{ is } SRD$
 and $(\mu X \cdot F (\mathcal{H}_{SRDES} X)) = (\mu X \cdot F (SRD X))$
 and $\text{carrier (uthy-order } SRDES) \rightarrow \text{carrier (uthy-order } SRDES) \equiv \llbracket SRD \rrbracket_H \rightarrow \llbracket SRD \rrbracket_H$
 and $\llbracket \mathcal{H}_{SRDES} \rrbracket_H \rightarrow \llbracket \mathcal{H}_{SRDES} \rrbracket_H \equiv \llbracket SRD \rrbracket_H \rightarrow \llbracket SRD \rrbracket_H$
 and $le \text{ (uthy-order } SRDES) = op \sqsubseteq$
 and $eq \text{ (uthy-order } SRDES) = op =$
 by (unfold-locales, simp-all add: srdes-hcond-def SRD-Continuous)

declare *srdes-theory-continuous.top-healthy* [simp del]

declare *srdes-theory-continuous.bottom-healthy* [simp del]

abbreviation *Chaos* :: $('s, 't::trace, 'α) hrel-rsp$ **where**

Chaos $\equiv \perp_{SRDES}$

abbreviation *Miracle* :: $('s, 't::trace, 'α) hrel-rsp$ **where**

Miracle $\equiv \top_{SRDES}$

thm *srdes-theory-continuous.weak.bottom-lower*

thm *srdes-theory-continuous.weak.top-higher*

thm *srdes-theory-continuous.meet-bottom*

thm *srdes-theory-continuous.meet-top*

abbreviation *srd-lfp* (μ_R) **where** $\mu_R F \equiv \mu_{SRDES} F$

abbreviation *srd-gfp* (ν_R) **where** $\nu_R F \equiv \nu_{SRDES} F$

syntax

-srd-mu :: $pttrn \Rightarrow logic \Rightarrow logic (\mu_R \cdot \cdot \cdot [0, 10] 10)$

-srd-nu :: $pttrn \Rightarrow logic \Rightarrow logic (\nu_R \cdot \cdot \cdot [0, 10] 10)$

translations

$\mu_R X \cdot P == \mu_R (\lambda X. P)$

$\nu_R X \cdot P == \mu_R (\lambda X. P)$

The reactive design weakest fixed-point can be defined in terms of relational calculus one.

lemma *srd-mu-equiv*:

assumes *Monotonic* $F F \in \llbracket SRD \rrbracket_H \rightarrow \llbracket SRD \rrbracket_H$

shows $(\mu_R X \cdot F(X)) = (\mu X \cdot F(SRD(X)))$
 by (metis assms srdes-hcond-def srdes-theory-continuous.utp-lfp-def)

end

3 Reactive Design Specifications

theory utp-rdes-designs
 imports utp-rdes-healths
 begin

3.1 Reactive design forms

lemma srdes-skip-def: $II_R = \mathbf{R}_s(\text{true} \vdash (\$tr' =_u \$tr \wedge \neg \$wait' \wedge \lceil II \rceil_R))$
 apply (rel-auto) using minus-zero-eq by blast+

lemma Chaos-def: $\text{Chaos} = \mathbf{R}_s(\text{false} \vdash \text{true})$

proof –

have $\text{Chaos} = SRD(\text{true})$
 by (metis srdes-hcond-def srdes-theory-continuous.healthy-bottom)
 also have $\dots = \mathbf{R}_s(\mathbf{H}(\text{true}))$
 by (simp add: SRD-RHS-H1-H2)
 also have $\dots = \mathbf{R}_s(\text{false} \vdash \text{true})$
 by (metis H1-design H2-true design-false-pre)
 finally show ?thesis .

qed

lemma Miracle-def: $\text{Miracle} = \mathbf{R}_s(\text{true} \vdash \text{false})$

proof –

have $\text{Miracle} = SRD(\text{false})$
 by (metis srdes-hcond-def srdes-theory-continuous.healthy-top)
 also have $\dots = \mathbf{R}_s(\mathbf{H}(\text{false}))$
 by (simp add: SRD-RHS-H1-H2)
 also have $\dots = \mathbf{R}_s(\text{true} \vdash \text{false})$
 by (metis (no-types, lifting) H1-H2-eq-design p-imp-p subst-impl subst-not utp-pred-laws.compl-bot-eq utp-pred-laws.compl-top-eq)
 finally show ?thesis .

qed

lemma RD1-reactive-design: $RD1(\mathbf{R}(P \vdash Q)) = \mathbf{R}(P \vdash Q)$
 by (rel-auto)

lemma RD2-reactive-design:

assumes $\$ok' \nVdash P \ \$ok' \nVdash Q$
 shows $RD2(\mathbf{R}(P \vdash Q)) = \mathbf{R}(P \vdash Q)$
 using assms
 by (metis H2-design RD2-RH-commute RD2-def)

lemma RD1-st-reactive-design: $RD1(\mathbf{R}_s(P \vdash Q)) = \mathbf{R}_s(P \vdash Q)$
 by (rel-auto)

lemma RD2-st-reactive-design:

assumes $\$ok' \nVdash P \ \$ok' \nVdash Q$
 shows $RD2(\mathbf{R}_s(P \vdash Q)) = \mathbf{R}_s(P \vdash Q)$
 using assms

by (metis H2-design RD2-RHS-commute RD2-def)

lemma wait-false-design:

$(P \vdash Q)_f = ((P_f) \vdash (Q_f))$
by (rel-auto)

lemma RD-RH-design-form:

$RD(P) = \mathbf{R}((\neg P_f) \vdash P^t_f)$

proof –

have $RD(P) = RD1(RD2(R1(R2c(R3c(P)))))$
by (simp add: RD-alt-def RH-def)
also have $\dots = RD1(H2(R1(R2s(R3c(P)))))$
by (simp add: R1-R2s-R2c RD2-def)
also have $\dots = RD1(R1(H2(R2s(R3c(P)))))$
by (simp add: R1-H2-commute)
also have $\dots = R1(H1(R1(H2(R2s(R3c(P))))))$
by (simp add: R1-idem RD1-via-R1)
also have $\dots = R1(H1(H2(R2s(R3c(R1(P))))))$
by (simp add: R1-H2-commute R1-R2c-commute R1-R2s-R2c R1-R3c-commute RD1-via-R1)
also have $\dots = R1(R2s(H1(H2(R3c(R1(P))))))$
by (simp add: R2s-H1-commute R2s-H2-commute)
also have $\dots = R1(R2s(H1(R3c(H2(R1(P))))))$
by (metis RD2-R3c-commute RD2-def)
also have $\dots = R2(R1(H1(R3c(H2(R1(P))))))$
by (metis R1-R2-commute R1-idem R2-def)
also have $\dots = R2(R3c(R1(\mathbf{H}(R1(P)))))$
by (simp add: R1-R3c-commute RD1-R3c-commute RD1-via-R1)
also have $\dots = RH(\mathbf{H}(R1(P)))$
by (metis R1-R2s-R2c R1-R3c-commute R2-R1-form RH-def)
also have $\dots = RH(\mathbf{H}(P))$
by (simp add: R1-H2-commute R1-R2c-commute R1-R3c-commute R1-idem RD1-via-R1 RH-def)
also have $\dots = RH((\neg P_f) \vdash P^t_f)$
by (simp add: H1-H2-eq-design)
also have $\dots = \mathbf{R}((\neg P_f) \vdash P^t_f)$
by (metis (no-types, lifting) R3c-subst-wait RH-def subst-not wait-false-design)
finally show ?thesis .

qed

lemma RD-reactive-design:

assumes P is RD
shows $\mathbf{R}((\neg P_f) \vdash P^t_f) = P$
by (metis RD-RH-design-form Healthy-def' assms)

lemma RD-RH-design:

assumes $\$ok' \# P \ \$ok' \# Q$
shows $RD(\mathbf{R}(P \vdash Q)) = \mathbf{R}(P \vdash Q)$
by (simp add: RD1-reactive-design RD2-reactive-design RD-alt-def RH-idem assms(1) assms(2))

lemma RH-design-is-RD:

assumes $\$ok' \# P \ \$ok' \# Q$
shows $\mathbf{R}(P \vdash Q)$ is RD
by (simp add: RD-RH-design Healthy-def' assms(1) assms(2))

lemma SRD-RH-design-form:

$SRD(P) = \mathbf{R}_s((\neg P_f) \vdash P^t_f)$

proof –

have $SRD(P) = R1(R2c(R3h(RD1(RD2(R1(P)))))$
by (*metis* (*no-types*, *lifting*) *R1-H2-commute* *R1-R2c-commute* *R1-R3h-commute* *R1-idem* *R2c-H2-commute* *RD1-R1-commute* *RD1-R2c-commute* *RD1-R3h-commute* *RD2-R3h-commute* *RD2-def* *RHS-def* *SRD-def*)
also have $\dots = R1(R2s(R3h(\mathbf{H}(P))))$
by (*metis* (*no-types*, *lifting*) *R1-H2-commute* *R1-R2c-is-R2* *R1-R3h-commute* *R2-R1-form* *RD1-via-R1* *RD2-def*)
also have $\dots = \mathbf{R}_s(\mathbf{H}(P))$
by (*simp add*: *R1-R2s-R2c* *RHS-def*)
also have $\dots = \mathbf{R}_s((\neg P^f_f) \vdash P^t_f)$
by (*simp add*: *H1-H2-eq-design*)
also have $\dots = \mathbf{R}_s((\neg P^f_f) \vdash P^t_f)$
by (*metis* (*no-types*, *lifting*) *R3h-subst-wait* *RHS-def* *subst-not* *wait-false-design*)
finally show *?thesis* .
qed

lemma *SRD-reactive-design*:

assumes P is *SRD*
shows $\mathbf{R}_s((\neg P^f_f) \vdash P^t_f) = P$
by (*metis* *SRD-RH-design-form* *Healthy-def'* *assms*)

lemma *SRD-RH-design*:

assumes $\$ok' \# P \ \$ok' \# Q$
shows $SRD(\mathbf{R}_s(P \vdash Q)) = \mathbf{R}_s(P \vdash Q)$
by (*simp add*: *RD1-st-reactive-design* *RD2-st-reactive-design* *RHS-idem* *SRD-def* *assms*(1) *assms*(2))

lemma *RHS-design-is-SRD*:

assumes $\$ok' \# P \ \$ok' \# Q$
shows $\mathbf{R}_s(P \vdash Q)$ is *SRD*
by (*simp add*: *Healthy-def'* *SRD-RH-design* *assms*(1) *assms*(2))

lemma *SRD-RHS-H1-H2*: $SRD(P) = \mathbf{R}_s(\mathbf{H}(P))$

by (*metis* (*no-types*, *lifting*) *H1-H2-eq-design* *R3h-subst-wait* *RHS-def* *SRD-RH-design-form* *subst-not* *wait-false-design*)

3.2 Auxiliary healthiness conditions

definition [*upred-defs*]: $R3c\text{-}pre(P) = (true \triangleleft \$wait \triangleright P)$

definition [*upred-defs*]: $R3c\text{-}post(P) = ([II]_D \triangleleft \$wait \triangleright P)$

definition [*upred-defs*]: $R3h\text{-}post(P) = ((\exists \$st \cdot [II]_D) \triangleleft \$wait \triangleright P)$

lemma *R3c-pre-conj*: $R3c\text{-}pre(P \wedge Q) = (R3c\text{-}pre(P) \wedge R3c\text{-}pre(Q))$
by (*rel-auto*)

lemma *R3c-pre-seq*:

$(true ;; Q) = true \implies R3c\text{-}pre(P ;; Q) = (R3c\text{-}pre(P) ;; Q)$
by (*rel-auto*)

lemma *unrest-ok-R3c-pre* [*unrest*]: $\$ok \# P \implies \$ok \# R3c\text{-}pre(P)$
by (*simp add*: *R3c-pre-def* *cond-def* *unrest*)

lemma *unrest-ok'-R3c-pre* [*unrest*]: $\$ok' \# P \implies \$ok' \# R3c\text{-}pre(P)$
by (*simp add*: *R3c-pre-def* *cond-def* *unrest*)

lemma *unrest-ok-R3c-post* [*unrest*]: $\$ok \# P \implies \$ok \# R3c\text{-}post(P)$
 by (*simp add: R3c-post-def cond-def unrest*)

lemma *unrest-ok-R3c-post'* [*unrest*]: $\$ok' \# P \implies \$ok' \# R3c\text{-}post(P)$
 by (*simp add: R3c-post-def cond-def unrest*)

lemma *unrest-ok-R3h-post* [*unrest*]: $\$ok \# P \implies \$ok \# R3h\text{-}post(P)$
 by (*simp add: R3h-post-def cond-def unrest*)

lemma *unrest-ok-R3h-post'* [*unrest*]: $\$ok' \# P \implies \$ok' \# R3h\text{-}post(P)$
 by (*simp add: R3h-post-def cond-def unrest*)

3.3 Composition laws

theorem *R1-design-composition*:

fixes $P\ Q :: ('t::trace, 'α, 'β)\ rel\text{-}rp$

and $R\ S :: ('t, 'β, 'γ)\ rel\text{-}rp$

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

shows

$(R1(P \vdash Q) ;; R1(R \vdash S)) =$
 $R1((\neg (R1(\neg P) ;; R1(true)) \wedge \neg (R1(Q) ;; R1(\neg R))) \vdash (R1(Q) ;; R1(S)))$

proof –

have $(R1(P \vdash Q) ;; R1(R \vdash S)) = (\exists\ ok_0 \cdot (R1(P \vdash Q)) \llbracket \llcorner ok_0 \rceil / \$ok' \rrbracket ;; (R1(R \vdash S)) \llbracket \llcorner ok_0 \rceil / \$ok \rrbracket)$
 using *seqr-middle ok-vwb-lens* **by** *blast*

also from *assms* **have** $\dots = (\exists\ ok_0 \cdot R1((\$ok \wedge P) \Rightarrow (\llcorner ok_0 \rceil \wedge Q)) ;; R1((\llcorner ok_0 \rceil \wedge R) \Rightarrow (\$ok' \wedge S)))$
 $\wedge S)))$

by (*simp add: design-def R1-def usubst unrest*)

also from *assms* **have** $\dots = ((R1((\$ok \wedge P) \Rightarrow (true \wedge Q)) ;; R1((true \wedge R) \Rightarrow (\$ok' \wedge S)))$
 $\vee (R1((\$ok \wedge P) \Rightarrow (false \wedge Q)) ;; R1((false \wedge R) \Rightarrow (\$ok' \wedge S))))$

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

also from *assms* **have** $\dots = ((R1((\$ok \wedge P) \Rightarrow Q) ;; R1(R \Rightarrow (\$ok' \wedge S)))$
 $\vee (R1(\neg (\$ok \wedge P)) ;; R1(true)))$

by *simp*

also from *assms* **have** $\dots = ((R1(\neg \$ok \vee \neg P \vee Q) ;; R1(\neg R \vee (\$ok' \wedge S)))$
 $\vee (R1(\neg \$ok \vee \neg P) ;; R1(true)))$

by (*simp add: impl-alt-def utp-pred-laws.sup.assoc*)

also from *assms* **have** $\dots = (((R1(\neg \$ok \vee \neg P) \vee R1(Q)) ;; R1(\neg R \vee (\$ok' \wedge S)))$
 $\vee (R1(\neg \$ok \vee \neg P) ;; R1(true)))$

by (*simp add: R1-disj utp-pred-laws.disj-assoc*)

also from *assms* **have** $\dots = ((R1(\neg \$ok \vee \neg P) ;; R1(\neg R \vee (\$ok' \wedge S)))$
 $\vee (R1(Q) ;; R1(\neg R \vee (\$ok' \wedge S))))$
 $\vee (R1(\neg \$ok \vee \neg P) ;; R1(true)))$

by (*simp add: seqr-or-distl utp-pred-laws.sup.assoc*)

also from *assms* **have** $\dots = ((R1(Q) ;; R1(\neg R \vee (\$ok' \wedge S)))$
 $\vee (R1(\neg \$ok \vee \neg P) ;; R1(true)))$

by (*rel-blast*)

also from *assms* **have** $\dots = ((R1(Q) ;; (R1(\neg R) \vee R1(S) \wedge \$ok'))$
 $\vee (R1(\neg \$ok \vee \neg P) ;; R1(true)))$

by (*simp add: R1-disj R1-extend-conj utp-pred-laws.inf-commute*)

also have $\dots = ((R1(Q) ;; (R1(\neg R) \vee R1(S) \wedge \$ok'))$
 $\vee ((R1(\neg \$ok) :: ('t, 'α, 'β)\ rel\text{-}rp) ;; R1(true)))$
 $\vee (R1(\neg P) ;; R1(true)))$

by (*simp add: R1-disj seqr-or-distl*)

also have $\dots = ((R1(Q) ;; (R1(\neg R) \vee R1(S) \wedge \$ok'))$
 $\vee (R1(\neg \$ok))$
 $\vee (R1(\neg P) ;; R1(true)))$

proof –
 have $((R1(\neg \$ok) :: ('t, 'α, 'β) \text{ rel-rp}) ;; R1(true)) =$
 $(R1(\neg \$ok) :: ('t, 'α, 'γ) \text{ rel-rp})$
 by (rel-auto)
 thus $?thesis$
 by simp
qed
 also have $\dots = ((R1(Q) ;; (R1(\neg R) \vee (R1(S \wedge \$ok'))))$
 $\vee R1(\neg \$ok)$
 $\vee (R1(\neg P) ;; R1(true)))$
 by $(\text{simp add: } R1\text{-extend-conj})$
 also have $\dots = ((R1(Q) ;; (R1(\neg R)))$
 $\vee (R1(Q) ;; (R1(S \wedge \$ok')))$
 $\vee R1(\neg \$ok)$
 $\vee (R1(\neg P) ;; R1(true)))$
 by $(\text{simp add: } \text{seqr-or-distr utp-pred-laws.sup.assoc})$
 also have $\dots = R1((R1(Q) ;; (R1(\neg R)))$
 $\vee (R1(Q) ;; (R1(S \wedge \$ok')))$
 $\vee (\neg \$ok)$
 $\vee (R1(\neg P) ;; R1(true)))$
 by $(\text{simp add: } R1\text{-disj } R1\text{-seqr})$
 also have $\dots = R1((R1(Q) ;; (R1(\neg R)))$
 $\vee ((R1(Q) ;; R1(S)) \wedge \$ok')$
 $\vee (\neg \$ok)$
 $\vee (R1(\neg P) ;; R1(true)))$
 by (rel-blast)
 also have $\dots = R1(\neg(\$ok \wedge \neg (R1(\neg P) ;; R1(true)) \wedge \neg (R1(Q) ;; (R1(\neg R))))$
 $\vee ((R1(Q) ;; R1(S)) \wedge \$ok'))$
 by (rel-blast)
 also have $\dots = R1((\$ok \wedge \neg (R1(\neg P) ;; R1(true)) \wedge \neg (R1(Q) ;; (R1(\neg R))))$
 $\Rightarrow (\$ok' \wedge (R1(Q) ;; R1(S))))$
 by $(\text{simp add: } \text{impl-alt-def utp-pred-laws.inf-commute})$
 also have $\dots = R1((\neg (R1(\neg P) ;; R1(true)) \wedge \neg (R1(Q) ;; R1(\neg R))) \vdash (R1(Q) ;; R1(S)))$
 by $(\text{simp add: } \text{design-def})$
finally show $?thesis$.
qed

theorem $R1\text{-design-composition-RR}$:
 assumes $P \text{ is } RR \ Q \text{ is } RR \ R \text{ is } RR \ S \text{ is } RR$
 shows
 $(R1(P \vdash Q) ;; R1(R \vdash S)) = R1(((\neg_r P) \text{ wp}_r \text{ false} \wedge Q \text{ wp}_r R) \vdash (Q ;; S))$
 apply $(\text{subst } R1\text{-design-composition})$
 apply $(\text{simp-all add: } \text{assms unrest wp-rea-def Healthy-if closure})$
 apply (rel-auto)
done

theorem $R1\text{-design-composition-RC}$:
 assumes $P \text{ is } RC \ Q \text{ is } RR \ R \text{ is } RR \ S \text{ is } RR$
 shows
 $(R1(P \vdash Q) ;; R1(R \vdash S)) = R1((P \wedge Q \text{ wp}_r R) \vdash (Q ;; S))$
 by $(\text{simp add: } R1\text{-design-composition-RR assms unrest Healthy-if closure wp})$

lemma $R2s\text{-design}$: $R2s(P \vdash Q) = (R2s(P) \vdash R2s(Q))$
 by $(\text{simp add: } R2s\text{-def design-def usubst})$

lemma *R2c-design*: $R2c(P \vdash Q) = (R2c(P) \vdash R2c(Q))$
 by (simp add: design-def impl-alt-def R2c-disj R2c-not R2c-ok R2c-and R2c-ok')

lemma *R1-R3c-design*:
 $R1(R3c(P \vdash Q)) = R1(R3c\text{-pre}(P) \vdash R3c\text{-post}(Q))$
 by (rel-auto)

lemma *R1-R3h-design*:
 $R1(R3h(P \vdash Q)) = R1(R3c\text{-pre}(P) \vdash R3h\text{-post}(Q))$
 by (rel-auto)

lemma *R3c-R1-design-composition*:
 assumes $\$ok' \# P \$ok' \# Q \$ok \# R \$ok \# S$
 shows $(R3c(R1(P \vdash Q)) ;; R3c(R1(R \vdash S))) =$
 $R3c(R1((\neg (R1(\neg P) ;; R1(true)) \wedge \neg ((R1(Q) \wedge \neg \$wait') ;; R1(\neg R)))$
 $\vdash (R1(Q) ;; (\llbracket II \rrbracket_D \triangleleft \$wait \triangleright R1(S))))$

proof –

have 1: $(\neg (R1(\neg R3c\text{-pre } P) ;; R1 true)) = (R3c\text{-pre } (\neg (R1(\neg P) ;; R1 true)))$
 by (rel-auto)
 have 2: $(\neg (R1(R3c\text{-post } Q) ;; R1(\neg R3c\text{-pre } R))) = R3c\text{-pre}(\neg ((R1 Q \wedge \neg \$wait') ;; R1(\neg R)))$
 by (rel-auto, blast+)
 have 3: $(R1(R3c\text{-post } Q) ;; R1(R3c\text{-post } S)) = R3c\text{-post } (R1 Q ;; (\llbracket II \rrbracket_D \triangleleft \$wait \triangleright R1 S))$
 by (rel-auto)
 show ?thesis
 apply (simp add: R3c-semir-form R1-R3c-commute[THEN sym] R1-R3c-design unrest)
 apply (subst R1-design-composition)
 apply (simp-all add: unrest assms R3c-pre-conj 1 2 3)
 done

qed

lemma *R3h-R1-design-composition*:
 assumes $\$ok' \# P \$ok' \# Q \$ok \# R \$ok \# S$
 shows $(R3h(R1(P \vdash Q)) ;; R3h(R1(R \vdash S))) =$
 $R3h(R1((\neg (R1(\neg P) ;; R1(true)) \wedge \neg ((R1(Q) \wedge \neg \$wait') ;; R1(\neg R)))$
 $\vdash (R1(Q) ;; ((\exists \$st \cdot \llbracket II \rrbracket_D) \triangleleft \$wait \triangleright R1(S))))$

proof –

have 1: $(\neg (R1(\neg R3c\text{-pre } P) ;; R1 true)) = (R3c\text{-pre } (\neg (R1(\neg P) ;; R1 true)))$
 by (rel-auto)
 have 2: $(\neg (R1(R3h\text{-post } Q) ;; R1(\neg R3c\text{-pre } R))) = R3c\text{-pre}(\neg ((R1 Q \wedge \neg \$wait') ;; R1(\neg R)))$
 by (rel-auto, blast+)
 have 3: $(R1(R3h\text{-post } Q) ;; R1(R3h\text{-post } S)) = R3h\text{-post } (R1 Q ;; ((\exists \$st \cdot \llbracket II \rrbracket_D) \triangleleft \$wait \triangleright R1 S))$
 by (rel-auto, blast+)
 show ?thesis
 apply (simp add: R3h-semir-form R1-R3h-commute[THEN sym] R1-R3h-design unrest)
 apply (subst R1-design-composition)
 apply (simp-all add: unrest assms R3c-pre-conj 1 2 3)
 done

qed

lemma *R2-design-composition*:
 assumes $\$ok' \# P \$ok' \# Q \$ok \# R \$ok \# S$
 shows $(R2(P \vdash Q) ;; R2(R \vdash S)) =$
 $R2((\neg (R1(\neg R2c P) ;; R1 true) \wedge \neg (R1(R2c Q) ;; R1(\neg R2c R))) \vdash (R1(R2c Q) ;; R1(R2c S)))$
 apply (simp add: R2-R2c-def R2c-design R1-design-composition assms unrest R2c-not R2c-and R2c-disj

R1-R2c-commute[*THEN sym*] *R2c-idem* *R2c-R1-seq*)
apply (*metis* (*no-types*, *lifting*) *R2c-R1-seq* *R2c-not* *R2c-true*)
done

lemma *RH-design-composition*:

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

shows $(RH(P \vdash Q) ;; RH(R \vdash S)) =$

$RH((\neg (R1 \ (\neg R2s \ P) ;; R1 \ true) \wedge \neg ((R1 \ (R2s \ Q) \wedge (\neg \$wait')) ;; R1 \ (\neg R2s \ R))) \vdash$
 $(R1 \ (R2s \ Q) ;; (\lceil II \rceil_D \triangleleft \$wait \triangleright R1 \ (R2s \ S))))$

proof –

have 1: $R2c \ (R1 \ (\neg R2s \ P) ;; R1 \ true) = (R1 \ (\neg R2s \ P) ;; R1 \ true)$

proof –

have 1: $(R1 \ (\neg R2s \ P) ;; R1 \ true) = (R1(R2 \ (\neg P) ;; R2 \ true))$

by (*rel-auto*)

have $R2c(R1(R2 \ (\neg P) ;; R2 \ true)) = R2c(R1(R2 \ (\neg P) ;; R2 \ true))$

using *R2c-not* **by** *blast*

also have $\dots = R2(R2 \ (\neg P) ;; R2 \ true)$

by (*metis* *R1-R2c-commute* *R1-R2c-is-R2*)

also have $\dots = (R2 \ (\neg P) ;; R2 \ true)$

by (*simp add: R2-seqr-distribute*)

also have $\dots = (R1 \ (\neg R2s \ P) ;; R1 \ true)$

by (*simp add: R2-def R2s-not R2s-true*)

finally show *?thesis*

by (*simp add: 1*)

qed

have 2: $R2c \ ((R1 \ (R2s \ Q) \wedge \neg \$wait') ;; R1 \ (\neg R2s \ R)) = ((R1 \ (R2s \ Q) \wedge \neg \$wait') ;; R1 \ (\neg R2s \ R))$

proof –

have $((R1 \ (R2s \ Q) \wedge \neg \$wait') ;; R1 \ (\neg R2s \ R)) = R1 \ (R2 \ (Q \wedge \neg \$wait') ;; R2 \ (\neg R))$

by (*rel-auto*)

hence $R2c \ ((R1 \ (R2s \ Q) \wedge \neg \$wait') ;; R1 \ (\neg R2s \ R)) = (R2 \ (Q \wedge \neg \$wait') ;; R2 \ (\neg R))$

by (*metis* *R1-R2c-commute* *R1-R2c-is-R2* *R2-seqr-distribute*)

also have $\dots = ((R1 \ (R2s \ Q) \wedge \neg \$wait') ;; R1 \ (\neg R2s \ R))$

by (*rel-auto*)

finally show *?thesis* .

qed

have 3: $R2c((R1 \ (R2s \ Q) ;; (\lceil II \rceil_D \triangleleft \$wait \triangleright R1 \ (R2s \ S)))) = (R1 \ (R2s \ Q) ;; (\lceil II \rceil_D \triangleleft \$wait \triangleright R1 \ (R2s \ S)))$

proof –

have $R2c(((R1 \ (R2s \ Q))\llbracket true/\$wait' \rrbracket ;; (\lceil II \rceil_D \triangleleft \$wait \triangleright R1 \ (R2s \ S))\llbracket true/\$wait \rrbracket))$

$= ((R1 \ (R2s \ Q))\llbracket true/\$wait' \rrbracket ;; (\lceil II \rceil_D \triangleleft \$wait \triangleright R1 \ (R2s \ S))\llbracket true/\$wait \rrbracket)$

proof –

have $R2c(((R1 \ (R2s \ Q))\llbracket true/\$wait' \rrbracket ;; (\lceil II \rceil_D \triangleleft \$wait \triangleright R1 \ (R2s \ S))\llbracket true/\$wait \rrbracket)) =$

$R2c(R1 \ (R2s \ (Q\llbracket true/\$wait' \rrbracket)) ;; \lceil II \rceil_D \llbracket true/\$wait \rrbracket)$

by (*simp add: usubst cond-unit-T R1-def R2s-def*)

also have $\dots = R2c(R2 \ (Q\llbracket true/\$wait' \rrbracket) ;; R2 \ (\lceil II \rceil_D \llbracket true/\$wait \rrbracket))$

by (*metis* *R2-def R2-des-lift-skip R2-subst-wait-true*)

also have $\dots = (R2 \ (Q\llbracket true/\$wait' \rrbracket) ;; R2 \ (\lceil II \rceil_D \llbracket true/\$wait \rrbracket))$

using *R2c-seq* **by** *blast*

also have $\dots = ((R1 \ (R2s \ Q))\llbracket true/\$wait' \rrbracket ;; (\lceil II \rceil_D \triangleleft \$wait \triangleright R1 \ (R2s \ S))\llbracket true/\$wait \rrbracket)$

apply (*simp add: usubst R2-des-lift-skip*)

apply (*metis* *R2-def R2-des-lift-skip R2-subst-wait'-true R2-subst-wait-true*)

done

finally show ?thesis .
 qed
 moreover have $R2c(((R1 \ (R2s \ Q))\llbracket false/\$wait' \rrbracket ;; ([II]_D \triangleleft \$wait \triangleright R1 \ (R2s \ S))\llbracket false/\$wait \rrbracket))$
 $= ((R1 \ (R2s \ Q))\llbracket false/\$wait' \rrbracket ;; ([II]_D \triangleleft \$wait \triangleright R1 \ (R2s \ S))\llbracket false/\$wait \rrbracket)$
 by (simp add: usubst cond-unit-F)
 (metis (no-types, hide-lams) R1-wait'-false R1-wait-false R2-def R2-subst-wait'-false R2-subst-wait-false
 R2c-seq)
 ultimately show ?thesis
 proof –
 have $[II]_D \triangleleft \$wait \triangleright R1 \ (R2s \ S) = R2 \ ([II]_D \triangleleft \$wait \triangleright S)$
 by (simp add: R1-R2c-is-R2 R1-R2s-R2c R2-condr' R2-des-lift-skip R2s-wait)
 then show ?thesis
 by (simp add: R1-R2c-is-R2 R1-R2s-R2c R2c-seq)
 qed
 qed

 have $(R1(R2s(R3c(P \vdash Q))) ;; R1(R2s(R3c(R \vdash S)))) =$
 $((R3c(R1(R2s(P) \vdash R2s(Q)))) ;; R3c(R1(R2s(R) \vdash R2s(S))))$
 by (metis (no-types, hide-lams) R1-R2s-R2c R1-R3c-commute R2c-R3c-commute R2s-design)
 also have ... = $R3c \ (R1 \ ((\neg (R1 \ (\neg R2s \ P)) ;; R1 \ true) \wedge \neg ((R1 \ (R2s \ Q) \wedge \neg \$wait') ;; R1 \ (\neg R2s \ R)))) \vdash$
 $(R1 \ (R2s \ Q) ;; ([II]_D \triangleleft \$wait \triangleright R1 \ (R2s \ S))))$
 by (simp add: R3c-R1-design-composition assms unrest)
 also have ... = $R3c(R1(R2c((\neg (R1 \ (\neg R2s \ P)) ;; R1 \ true) \wedge \neg ((R1 \ (R2s \ Q) \wedge \neg \$wait') ;; R1 \ (\neg R2s \ R)))) \vdash$
 $(R1 \ (R2s \ Q) ;; ([II]_D \triangleleft \$wait \triangleright R1 \ (R2s \ S))))$
 by (simp add: R2c-design R2c-and R2c-not 1 2 3)
 finally show ?thesis
 by (simp add: R1-R2s-R2c R1-R3c-commute R2c-R3c-commute RH-def)
 qed

 lemma *RHS-design-composition*:
 assumes $\$ok' \# P \ \$ok' \# Q \ \$ok \# R \ \$ok \# S$
 shows $(\mathbf{R}_s(P \vdash Q) ;; \mathbf{R}_s(R \vdash S)) =$
 $\mathbf{R}_s((\neg (R1 \ (\neg R2s \ P)) ;; R1 \ true) \wedge \neg ((R1 \ (R2s \ Q) \wedge \neg \$wait') ;; R1 \ (\neg R2s \ R))) \vdash$
 $(R1 \ (R2s \ Q) ;; ((\exists \$st \cdot [II]_D \triangleleft \$wait \triangleright R1 \ (R2s \ S))))$
 proof –
 have 1: $R2c \ (R1 \ (\neg R2s \ P) ;; R1 \ true) = (R1 \ (\neg R2s \ P) ;; R1 \ true)$
 proof –
 have 1: $(R1 \ (\neg R2s \ P) ;; R1 \ true) = (R1(R2 \ (\neg P) ;; R2 \ true))$
 by (rel-auto, blast)
 have $R2c(R1(R2 \ (\neg P) ;; R2 \ true)) = R2c(R1(R2 \ (\neg P) ;; R2 \ true))$
 using R2c-not by blast
 also have ... = $R2(R2 \ (\neg P) ;; R2 \ true)$
 by (metis R1-R2c-commute R1-R2c-is-R2)
 also have ... = $(R2 \ (\neg P) ;; R2 \ true)$
 by (simp add: R2-seqr-distribute)
 also have ... = $(R1 \ (\neg R2s \ P) ;; R1 \ true)$
 by (simp add: R2-def R2s-not R2s-true)
 finally show ?thesis
 by (simp add: 1)
 qed
 have 2: $R2c \ ((R1 \ (R2s \ Q) \wedge \neg \$wait') ;; R1 \ (\neg R2s \ R)) = ((R1 \ (R2s \ Q) \wedge \neg \$wait') ;; R1 \ (\neg R2s \ R))$

proof –
 have $((R1 (R2s Q) \wedge \neg \$wait') ;; R1 (\neg R2s R)) = R1 (R2 (Q \wedge \neg \$wait') ;; R2 (\neg R))$
 by *(rel-auto, blast+)*
 hence $R2c ((R1 (R2s Q) \wedge \neg \$wait') ;; R1 (\neg R2s R)) = (R2 (Q \wedge \neg \$wait') ;; R2 (\neg R))$
 by *(metis (no-types, lifting) R1-R2c-commute R1-R2c-is-R2 R2-seqr-distribute)*
 also have $\dots = ((R1 (R2s Q) \wedge \neg \$wait') ;; R1 (\neg R2s R))$
 by *(rel-auto, blast+)*
 finally show *?thesis* .
qed

have $3: R2c((R1 (R2s Q) ;; ((\exists \$st \cdot [II]_D) \triangleleft \$wait \triangleright R1 (R2s S)))) =$
 $(R1 (R2s Q) ;; ((\exists \$st \cdot [II]_D) \triangleleft \$wait \triangleright R1 (R2s S)))$

proof –
 have $R2c(((R1 (R2s Q))\llbracket true/\$wait' \rrbracket ;; ((\exists \$st \cdot [II]_D) \triangleleft \$wait \triangleright R1 (R2s S))\llbracket true/\$wait \rrbracket))$
 $= ((R1 (R2s Q))\llbracket true/\$wait' \rrbracket ;; ((\exists \$st \cdot [II]_D) \triangleleft \$wait \triangleright R1 (R2s S))\llbracket true/\$wait \rrbracket)$
proof –
 have $R2c(((R1 (R2s Q))\llbracket true/\$wait' \rrbracket ;; ((\exists \$st \cdot [II]_D) \triangleleft \$wait \triangleright R1 (R2s S))\llbracket true/\$wait \rrbracket)) =$
 $R2c(R1 (R2s (Q\llbracket true/\$wait' \rrbracket)) ;; (\exists \$st \cdot [II]_D)\llbracket true/\$wait \rrbracket)$
 by *(simp add: usubst cond-unit-T R1-def R2s-def)*
 also have $\dots = R2c(R2(Q\llbracket true/\$wait' \rrbracket) ;; R2((\exists \$st \cdot [II]_D)\llbracket true/\$wait \rrbracket))$
 by *(metis (no-types, lifting) R2-def R2-des-lift-skip R2-subst-wait-true R2-st-ex)*
 also have $\dots = (R2(Q\llbracket true/\$wait' \rrbracket) ;; R2((\exists \$st \cdot [II]_D)\llbracket true/\$wait \rrbracket))$
 using *R2c-seq* by *blast*
 also have $\dots = ((R1 (R2s Q))\llbracket true/\$wait' \rrbracket ;; ((\exists \$st \cdot [II]_D) \triangleleft \$wait \triangleright R1 (R2s S))\llbracket true/\$wait \rrbracket)$
 apply *(simp add: usubst R2-des-lift-skip)*
 apply *(metis (no-types) R2-def R2-des-lift-skip R2-st-ex R2-subst-wait'-true R2-subst-wait-true)*
 done
 finally show *?thesis* .
qed
 moreover have $R2c(((R1 (R2s Q))\llbracket false/\$wait' \rrbracket ;; ((\exists \$st \cdot [II]_D) \triangleleft \$wait \triangleright R1 (R2s S))\llbracket false/\$wait \rrbracket))$
 $= ((R1 (R2s Q))\llbracket false/\$wait' \rrbracket ;; ((\exists \$st \cdot [II]_D) \triangleleft \$wait \triangleright R1 (R2s S))\llbracket false/\$wait \rrbracket)$
 by *(simp add: usubst)*
(metis (no-types, lifting) R1-wait'-false R1-wait-false R2-R1-form R2-subst-wait'-false R2-subst-wait-false R2c-seq)
 ultimately show *?thesis*
 by *(smt R2-R1-form R2-condr' R2-des-lift-skip R2-st-ex R2c-seq R2s-wait)*
qed

have $(R1(R2s(R3h(P \vdash Q))) ;; R1(R2s(R3h(R \vdash S)))) =$
 $((R3h(R1(R2s(P) \vdash R2s(Q)))) ;; R3h(R1(R2s(R) \vdash R2s(S))))$
 by *(metis (no-types, hide-lams) R1-R2s-R2c R1-R3h-commute R2c-R3h-commute R2s-design)*
 also have $\dots = R3h (R1 ((\neg (R1 (\neg R2s P) ;; R1 true) \wedge \neg ((R1 (R2s Q) \wedge \neg \$wait') ;; R1 (\neg R2s R))) \vdash$
 $(R1 (R2s Q) ;; ((\exists \$st \cdot [II]_D) \triangleleft \$wait \triangleright R1 (R2s S))))$
 by *(simp add: R3h-R1-design-composition assms unrest)*
 also have $\dots = R3h(R1(R2c((\neg (R1 (\neg R2s P) ;; R1 true) \wedge \neg ((R1 (R2s Q) \wedge \neg \$wait') ;; R1 (\neg R2s R))) \vdash$
 $(R1 (R2s Q) ;; ((\exists \$st \cdot [II]_D) \triangleleft \$wait \triangleright R1 (R2s S))))$
 by *(simp add: R2c-design R2c-and R2c-not 1 2 3)*
 finally show *?thesis*
 by *(simp add: R1-R2s-R2c R1-R3h-commute R2c-R3h-commute RHS-def)*
qed

lemma *RHS-R2s-design-composition:*
 assumes

$\$ok' \# P \ \$ok' \# Q \ \$ok \# R \ \$ok \# S$
 $P \text{ is } R2s \ Q \text{ is } R2s \ R \text{ is } R2s \ S \text{ is } R2s$
shows $(\mathbf{R}_s(P \vdash Q) ;; \mathbf{R}_s(R \vdash S)) =$
 $\mathbf{R}_s((\neg (R1 \ (\neg P) ;; R1 \ true) \wedge \neg ((R1 \ Q \wedge \neg \$wait') ;; R1 \ (\neg R))) \vdash$
 $(R1 \ Q ;; ((\exists \$st \cdot \lceil II \rceil_D) \triangleleft \$wait \triangleright R1 \ S)))$
proof –
have $f1: R2s \ P = P$
by $(meson \ Healthy-def \ assms(5))$
have $f2: R2s \ Q = Q$
by $(meson \ Healthy-def \ assms(6))$
have $f3: R2s \ R = R$
by $(meson \ Healthy-def \ assms(7))$
have $R2s \ S = S$
by $(meson \ Healthy-def \ assms(8))$
then show $?thesis$
using $f3 \ f2 \ f1$ **by** $(simp \ add: \ RHS-design-composition \ assms(1) \ assms(2) \ assms(3) \ assms(4))$
qed

lemma $RH\text{-}design\text{-}export\text{-}R1: \mathbf{R}(P \vdash Q) = \mathbf{R}(P \vdash R1(Q))$
by $(rel\text{-}auto)$

lemma $RH\text{-}design\text{-}export\text{-}R2s: \mathbf{R}(P \vdash Q) = \mathbf{R}(P \vdash R2s(Q))$
by $(rel\text{-}auto)$

lemma $RH\text{-}design\text{-}export\text{-}R2c: \mathbf{R}(P \vdash Q) = \mathbf{R}(P \vdash R2c(Q))$
by $(rel\text{-}auto)$

lemma $RHS\text{-}design\text{-}export\text{-}R1: \mathbf{R}_s(P \vdash Q) = \mathbf{R}_s(P \vdash R1(Q))$
by $(rel\text{-}auto)$

lemma $RHS\text{-}design\text{-}export\text{-}R2s: \mathbf{R}_s(P \vdash Q) = \mathbf{R}_s(P \vdash R2s(Q))$
by $(rel\text{-}auto)$

lemma $RHS\text{-}design\text{-}export\text{-}R2c: \mathbf{R}_s(P \vdash Q) = \mathbf{R}_s(P \vdash R2c(Q))$
by $(rel\text{-}auto)$

lemma $RHS\text{-}design\text{-}export\text{-}R2: \mathbf{R}_s(P \vdash Q) = \mathbf{R}_s(P \vdash R2(Q))$
by $(rel\text{-}auto)$

lemma $R1\text{-}design\text{-}R1\text{-}pre:$
 $\mathbf{R}_s(R1(P) \vdash Q) = \mathbf{R}_s(P \vdash Q)$
by $(rel\text{-}auto)$

lemma $RHS\text{-}design\text{-}ok\text{-}wait: \mathbf{R}_s(P \llbracket true, false / \$ok, \$wait \rrbracket \vdash Q \llbracket true, false / \$ok, \$wait \rrbracket) = \mathbf{R}_s(P \vdash Q)$
by $(rel\text{-}auto)$

lemma $RHS\text{-}design\text{-}neg\text{-}R1\text{-}pre:$
 $\mathbf{R}_s((\neg R1 \ P) \vdash R) = \mathbf{R}_s((\neg P) \vdash R)$
by $(rel\text{-}auto)$

lemma $RHS\text{-}design\text{-}conj\text{-}neg\text{-}R1\text{-}pre:$
 $\mathbf{R}_s(((\neg R1 \ P) \wedge Q) \vdash R) = \mathbf{R}_s(((\neg P) \wedge Q) \vdash R)$
by $(rel\text{-}auto)$

lemma $RHS\text{-}pre\text{-}lemma: (\mathbf{R}_s \ P)^f_f = R1(R2c(P^f_f))$

by (rel-auto)

lemma *RHS-design-R2c-pre*:

$\mathbf{R}_s(R2c(P) \vdash Q) = \mathbf{R}_s(P \vdash Q)$

by (rel-auto)

3.4 Refinement introduction laws

lemma *R1-design-refine*:

assumes

P_1 is R1 P_2 is R1 Q_1 is R1 Q_2 is R1

$\$ok \# P_1 \$ok' \# P_1 \$ok \# P_2 \$ok' \# P_2$

$\$ok \# Q_1 \$ok' \# Q_1 \$ok \# Q_2 \$ok' \# Q_2$

shows $R1(P_1 \vdash P_2) \sqsubseteq R1(Q_1 \vdash Q_2) \longleftrightarrow 'P_1 \Rightarrow Q_1' \wedge 'P_1 \wedge Q_2 \Rightarrow P_2'$

proof –

have $R1((\exists \$ok; \$ok' \cdot P_1) \vdash (\exists \$ok; \$ok' \cdot P_2)) \sqsubseteq R1((\exists \$ok; \$ok' \cdot Q_1) \vdash (\exists \$ok; \$ok' \cdot Q_2))$

$\longleftrightarrow 'R1(\exists \$ok; \$ok' \cdot P_1) \Rightarrow R1(\exists \$ok; \$ok' \cdot Q_1)' \wedge 'R1(\exists \$ok; \$ok' \cdot P_1) \wedge R1(\exists \$ok; \$ok' \cdot Q_2) \Rightarrow R1(\exists \$ok; \$ok' \cdot P_2)'$

by (rel-auto, meson+)

thus ?thesis

by (simp-all add: ex-unrest ex-plus Healthy-if assms)

qed

lemma *R1-design-refine-RR*:

assumes P_1 is RR P_2 is RR Q_1 is RR Q_2 is RR

shows $R1(P_1 \vdash P_2) \sqsubseteq R1(Q_1 \vdash Q_2) \longleftrightarrow 'P_1 \Rightarrow Q_1' \wedge 'P_1 \wedge Q_2 \Rightarrow P_2'$

by (simp add: R1-design-refine assms unrest closure)

lemma *RHS-design-refine*:

assumes

P_1 is R1 P_2 is R1 Q_1 is R1 Q_2 is R1

P_1 is R2c P_2 is R2c Q_1 is R2c Q_2 is R2c

$\$ok \# P_1 \$ok' \# P_1 \$ok \# P_2 \$ok' \# P_2$

$\$ok \# Q_1 \$ok' \# Q_1 \$ok \# Q_2 \$ok' \# Q_2$

$\$wait \# P_1 \$wait \# P_2 \$wait \# Q_1 \$wait \# Q_2$

shows $\mathbf{R}_s(P_1 \vdash P_2) \sqsubseteq \mathbf{R}_s(Q_1 \vdash Q_2) \longleftrightarrow 'P_1 \Rightarrow Q_1' \wedge 'P_1 \wedge Q_2 \Rightarrow P_2'$

proof –

have $\mathbf{R}_s(P_1 \vdash P_2) \sqsubseteq \mathbf{R}_s(Q_1 \vdash Q_2) \longleftrightarrow R1(R3h(R2c(P_1 \vdash P_2))) \sqsubseteq R1(R3h(R2c(Q_1 \vdash Q_2)))$

by (simp add: R2c-R3h-commute RHS-def)

also have $\dots \longleftrightarrow R1(R3h(P_1 \vdash P_2)) \sqsubseteq R1(R3h(Q_1 \vdash Q_2))$

by (simp add: Healthy-if R2c-design assms)

also have $\dots \longleftrightarrow R1(R3h(P_1 \vdash P_2)) \llbracket false/\$wait \rrbracket \sqsubseteq R1(R3h(Q_1 \vdash Q_2)) \llbracket false/\$wait \rrbracket$

by (rel-auto, metis+)

also have $\dots \longleftrightarrow R1(P_1 \vdash P_2) \llbracket false/\$wait \rrbracket \sqsubseteq R1(Q_1 \vdash Q_2) \llbracket false/\$wait \rrbracket$

by (rel-auto)

also have $\dots \longleftrightarrow R1(P_1 \vdash P_2) \sqsubseteq R1(Q_1 \vdash Q_2)$

by (simp add: usubst assms closure unrest)

also have $\dots \longleftrightarrow 'P_1 \Rightarrow Q_1' \wedge 'P_1 \wedge Q_2 \Rightarrow P_2'$

by (simp add: R1-design-refine assms)

finally show ?thesis .

qed

lemma *srdes-refine-intro*:

assumes $'P_1 \Rightarrow P_2' \wedge 'P_1 \wedge Q_2 \Rightarrow Q_1'$

shows $\mathbf{R}_s(P_1 \vdash Q_1) \sqsubseteq \mathbf{R}_s(P_2 \vdash Q_2)$

by (simp add: RHS-mono assms design-refine-intro)

3.5 Distribution laws

lemma *RHS-design-choice*: $\mathbf{R}_s(P_1 \vdash Q_1) \sqcap \mathbf{R}_s(P_2 \vdash Q_2) = \mathbf{R}_s((P_1 \wedge P_2) \vdash (Q_1 \vee Q_2))$
 by (*metis RHS-inf design-choice*)

lemma *RHS-design-sup*: $\mathbf{R}_s(P_1 \vdash Q_1) \sqcup \mathbf{R}_s(P_2 \vdash Q_2) = \mathbf{R}_s((P_1 \vee P_2) \vdash ((P_1 \Rightarrow Q_1) \wedge (P_2 \Rightarrow Q_2)))$
 by (*metis RHS-sup design-inf*)

lemma *RHS-design-USUP*:
 assumes $A \neq \{\}$
 shows $(\bigcap i \in A \cdot \mathbf{R}_s(P(i) \vdash Q(i))) = \mathbf{R}_s((\bigcap i \in A \cdot P(i)) \vdash (\bigcap i \in A \cdot Q(i)))$
 by (*subst RHS-INF[OF assms, THEN sym], simp add: design-UINF-mem assms*)

end

4 Reactive Design Triples

theory *utp-rdes-triples*
 imports *utp-rdes-designs*
 begin

4.1 Diamond notation

definition *wait'-cond* ::
 $('t::\text{trace}, 'a, 'b) \text{ rel-rp} \Rightarrow ('t, 'a, 'b) \text{ rel-rp} \Rightarrow ('t, 'a, 'b) \text{ rel-rp}$ (**infixr** \diamond 65) **where**
 $[\text{upred-defs}]: P \diamond Q = (P \triangleleft \$\text{wait}' \triangleright Q)$

lemma *wait'-cond-unrest* [*unrest*]:
 $\llbracket \text{out-var wait} \bowtie x; x \# P; x \# Q \rrbracket \Longrightarrow x \# (P \diamond Q)$
 by (*simp add: wait'-cond-def unrest*)

lemma *wait'-cond-subst* [*usubst*]:
 $\$ \text{wait}' \# \sigma \Longrightarrow \sigma \dagger (P \diamond Q) = (\sigma \dagger P) \diamond (\sigma \dagger Q)$
 by (*simp add: wait'-cond-def usubst unrest*)

lemma *wait'-cond-left-false*: $\text{false} \diamond P = (\neg \$ \text{wait}' \wedge P)$
 by (*rel-auto*)

lemma *wait'-cond-seq*: $((P \diamond Q) ;; R) = ((P ;; (\$ \text{wait}' \wedge R)) \vee (Q ;; (\neg \$ \text{wait}' \wedge R)))$
 by (*simp add: wait'-cond-def cond-def seq-or-distl, rel-blast*)

lemma *wait'-cond-true*: $(P \diamond Q \wedge \$ \text{wait}') = (P \wedge \$ \text{wait}')$
 by (*rel-auto*)

lemma *wait'-cond-false*: $(P \diamond Q \wedge (\neg \$ \text{wait}')) = (Q \wedge (\neg \$ \text{wait}'))$
 by (*rel-auto*)

lemma *wait'-cond-idem*: $P \diamond P = P$
 by (*rel-auto*)

lemma *wait'-cond-conj-exchange*:
 $((P \diamond Q) \wedge (R \diamond S)) = (P \wedge R) \diamond (Q \wedge S)$
 by (*rel-auto*)

lemma *subst-wait'-cond-true* [*usubst*]: $(P \diamond Q) \llbracket \text{true} / \$ \text{wait}' \rrbracket = P \llbracket \text{true} / \$ \text{wait}' \rrbracket$
 by (*rel-auto*)

lemma *subst-wait'-cond-false* [*usubst*]: $(P \diamond Q) \llbracket \text{false} / \$\text{wait}' \rrbracket = Q \llbracket \text{false} / \$\text{wait}' \rrbracket$
by (*rel-auto*)

lemma *subst-wait'-left-subst*: $(P \llbracket \text{true} / \$\text{wait}' \rrbracket \diamond Q) = (P \diamond Q)$
by (*rel-auto*)

lemma *subst-wait'-right-subst*: $(P \diamond Q \llbracket \text{false} / \$\text{wait}' \rrbracket) = (P \diamond Q)$
by (*rel-auto*)

lemma *wait'-cond-split*: $P \llbracket \text{true} / \$\text{wait}' \rrbracket \diamond P \llbracket \text{false} / \$\text{wait}' \rrbracket = P$
by (*simp add: wait'-cond-def cond-var-split*)

lemma *wait-cond'-assoc* [*simp*]: $P \diamond Q \diamond R = P \diamond R$
by (*rel-auto*)

lemma *wait-cond'-shadow*: $(P \diamond Q) \diamond R = P \diamond Q \diamond R$
by (*rel-auto*)

lemma *wait-cond'-conj* [*simp*]: $P \diamond (Q \wedge (R \diamond S)) = P \diamond (Q \wedge S)$
by (*rel-auto*)

lemma *R1-wait'-cond*: $R1(P \diamond Q) = R1(P) \diamond R1(Q)$
by (*rel-auto*)

lemma *R2s-wait'-cond*: $R2s(P \diamond Q) = R2s(P) \diamond R2s(Q)$
by (*simp add: wait'-cond-def R2s-def R2s-def usubst*)

lemma *R2-wait'-cond*: $R2(P \diamond Q) = R2(P) \diamond R2(Q)$
by (*simp add: R2-def R2s-wait'-cond R1-wait'-cond*)

lemma *wait'-cond-R1-closed* [*closure*]:
 $\llbracket P \text{ is } R1; Q \text{ is } R1 \rrbracket \implies P \diamond Q \text{ is } R1$
by (*simp add: Healthy-def R1-wait'-cond*)

lemma *wait'-cond-R2c-closed* [*closure*]: $\llbracket P \text{ is } R2c; Q \text{ is } R2c \rrbracket \implies P \diamond Q \text{ is } R2c$
by (*simp add: R2c-condr wait'-cond-def Healthy-def, rel-auto*)

4.2 Export laws

lemma *RH-design-peri-R1*: $\mathbf{R}(P \vdash R1(Q) \diamond R) = \mathbf{R}(P \vdash Q \diamond R)$
by (*metis (no-types, lifting) R1-idem R1-wait'-cond RH-design-export-R1*)

lemma *RH-design-post-R1*: $\mathbf{R}(P \vdash Q \diamond R1(R)) = \mathbf{R}(P \vdash Q \diamond R)$
by (*metis R1-wait'-cond RH-design-export-R1 RH-design-peri-R1*)

lemma *RH-design-peri-R2s*: $\mathbf{R}(P \vdash R2s(Q) \diamond R) = \mathbf{R}(P \vdash Q \diamond R)$
by (*metis (no-types, lifting) R2s-idem R2s-wait'-cond RH-design-export-R2s*)

lemma *RH-design-post-R2s*: $\mathbf{R}(P \vdash Q \diamond R2s(R)) = \mathbf{R}(P \vdash Q \diamond R)$
by (*metis (no-types, lifting) R2s-idem R2s-wait'-cond RH-design-export-R2s*)

lemma *RH-design-peri-R2c*: $\mathbf{R}(P \vdash R2c(Q) \diamond R) = \mathbf{R}(P \vdash Q \diamond R)$
by (*metis R1-R2s-R2c RH-design-peri-R1 RH-design-peri-R2s*)

lemma *RHS-design-peri-R1*: $\mathbf{R}_s(P \vdash R1(Q) \diamond R) = \mathbf{R}_s(P \vdash Q \diamond R)$

by (metis (no-types, lifting) R1-idem R1-wait'-cond RHS-design-export-R1)

lemma *RHS-design-post-R1*: $\mathbf{R}_s(P \vdash Q \diamond R1(R)) = \mathbf{R}_s(P \vdash Q \diamond R)$

by (metis R1-wait'-cond RHS-design-export-R1 RHS-design-peri-R1)

lemma *RHS-design-peri-R2s*: $\mathbf{R}_s(P \vdash R2s(Q) \diamond R) = \mathbf{R}_s(P \vdash Q \diamond R)$

by (metis (no-types, lifting) R2s-idem R2s-wait'-cond RHS-design-export-R2s)

lemma *RHS-design-post-R2s*: $\mathbf{R}_s(P \vdash Q \diamond R2s(R)) = \mathbf{R}_s(P \vdash Q \diamond R)$

by (metis R2s-wait'-cond RHS-design-export-R2s RHS-design-peri-R2s)

lemma *RHS-design-peri-R2c*: $\mathbf{R}_s(P \vdash R2c(Q) \diamond R) = \mathbf{R}_s(P \vdash Q \diamond R)$

by (metis R1-R2s-R2c RHS-design-peri-R1 RHS-design-peri-R2s)

lemma *RH-design-lemma1*:

$RH(P \vdash (R1(R2c(Q)) \vee R) \diamond S) = RH(P \vdash (Q \vee R) \diamond S)$

by (metis (no-types, lifting) R1-R2c-is-R2 R1-R2s-R2c R2-R1-form R2-disj R2c-idem RH-design-peri-R1 RH-design-peri-R2s)

lemma *RHS-design-lemma1*:

$RHS(P \vdash (R1(R2c(Q)) \vee R) \diamond S) = RHS(P \vdash (Q \vee R) \diamond S)$

by (metis (no-types, lifting) R1-R2c-is-R2 R1-R2s-R2c R2-R1-form R2-disj R2c-idem RHS-design-peri-R1 RHS-design-peri-R2s)

4.3 Pre-, peri-, and postconditions

4.3.1 Definitions

abbreviation $pre_s \equiv [\$ok \mapsto_s true, \$ok' \mapsto_s false, \$wait \mapsto_s false]$

abbreviation $cmt_s \equiv [\$ok \mapsto_s true, \$ok' \mapsto_s true, \$wait \mapsto_s false]$

abbreviation $peri_s \equiv [\$ok \mapsto_s true, \$ok' \mapsto_s true, \$wait \mapsto_s false, \$wait' \mapsto_s true]$

abbreviation $post_s \equiv [\$ok \mapsto_s true, \$ok' \mapsto_s true, \$wait \mapsto_s false, \$wait' \mapsto_s false]$

abbreviation $npre_R(P) \equiv pre_s \dagger P$

definition [*upred-defs*]: $pre_R(P) = (\neg_r npre_R(P))$

definition [*upred-defs*]: $cmt_R(P) = R1(cmt_s \dagger P)$

definition [*upred-defs*]: $peri_R(P) = R1(peri_s \dagger P)$

definition [*upred-defs*]: $post_R(P) = R1(post_s \dagger P)$

4.3.2 Unrestriction laws

lemma *ok-pre-unrest* [*unrest*]: $\$ok \# pre_R P$

by (simp add: pre_R-def unrest usubst)

lemma *ok-peri-unrest* [*unrest*]: $\$ok \# peri_R P$

by (simp add: peri_R-def unrest usubst)

lemma *ok-post-unrest* [*unrest*]: $\$ok \# post_R P$

by (simp add: post_R-def unrest usubst)

lemma *ok-cmt-unrest* [*unrest*]: $\$ok \# cmt_R P$

by (simp add: cmt_R-def unrest usubst)

lemma *ok'-pre-unrest* [*unrest*]: $\$ok' \# pre_R P$

by (simp add: pre_R-def unrest usubst)

lemma *ok'-peri-unrest* [*unrest*]: $\$ok' \# \text{peri}_R P$
 by (*simp add: peri_R-def unrest usubst*)

lemma *ok'-post-unrest* [*unrest*]: $\$ok' \# \text{post}_R P$
 by (*simp add: post_R-def unrest usubst*)

lemma *ok'-cmt-unrest* [*unrest*]: $\$ok' \# \text{cmt}_R P$
 by (*simp add: cmt_R-def unrest usubst*)

lemma *wait-pre-unrest* [*unrest*]: $\$wait \# \text{pre}_R P$
 by (*simp add: pre_R-def unrest usubst*)

lemma *wait-peri-unrest* [*unrest*]: $\$wait \# \text{peri}_R P$
 by (*simp add: peri_R-def unrest usubst*)

lemma *wait-post-unrest* [*unrest*]: $\$wait \# \text{post}_R P$
 by (*simp add: post_R-def unrest usubst*)

lemma *wait-cmt-unrest* [*unrest*]: $\$wait \# \text{cmt}_R P$
 by (*simp add: cmt_R-def unrest usubst*)

lemma *wait'-peri-unrest* [*unrest*]: $\$wait' \# \text{peri}_R P$
 by (*simp add: peri_R-def unrest usubst*)

lemma *wait'-post-unrest* [*unrest*]: $\$wait' \# \text{post}_R P$
 by (*simp add: post_R-def unrest usubst*)

4.3.3 Substitution laws

lemma *pre_s-design*: $\text{pre}_s \dagger (P \vdash Q) = (\neg \text{pre}_s \dagger P)$
 by (*simp add: design-def pre_R-def usubst*)

lemma *peri_s-design*: $\text{peri}_s \dagger (P \vdash Q \diamond R) = \text{peri}_s \dagger (P \Rightarrow Q)$
 by (*simp add: design-def usubst wait'-cond-def*)

lemma *post_s-design*: $\text{post}_s \dagger (P \vdash Q \diamond R) = \text{post}_s \dagger (P \Rightarrow R)$
 by (*simp add: design-def usubst wait'-cond-def*)

lemma *cmt_s-design*: $\text{cmt}_s \dagger (P \vdash Q) = \text{cmt}_s \dagger (P \Rightarrow Q)$
 by (*simp add: design-def usubst wait'-cond-def*)

lemma *pre_s-R1* [*usubst*]: $\text{pre}_s \dagger R1(P) = R1(\text{pre}_s \dagger P)$
 by (*simp add: R1-def usubst*)

lemma *pre_s-R2c* [*usubst*]: $\text{pre}_s \dagger R2c(P) = R2c(\text{pre}_s \dagger P)$
 by (*simp add: R2c-def R2s-def usubst*)

lemma *peri_s-R1* [*usubst*]: $\text{peri}_s \dagger R1(P) = R1(\text{peri}_s \dagger P)$
 by (*simp add: R1-def usubst*)

lemma *peri_s-R2c* [*usubst*]: $\text{peri}_s \dagger R2c(P) = R2c(\text{peri}_s \dagger P)$
 by (*simp add: R2c-def R2s-def usubst*)

lemma *post_s-R1* [*usubst*]: $\text{post}_s \dagger R1(P) = R1(\text{post}_s \dagger P)$
 by (*simp add: R1-def usubst*)

lemma *post_s-R2c* [*usubst*]: $post_s \uparrow R2c(P) = R2c(post_s \uparrow P)$
 by (*simp add: R2c-def R2s-def usubst*)

lemma *cmt_s-R1* [*usubst*]: $cmt_s \uparrow R1(P) = R1(cmt_s \uparrow P)$
 by (*simp add: R1-def usubst*)

lemma *cmt_s-R2c* [*usubst*]: $cmt_s \uparrow R2c(P) = R2c(cmt_s \uparrow P)$
 by (*simp add: R2c-def R2s-def usubst*)

lemma *pre-wait-false*:
 $pre_R(P \llbracket false/\$wait \rrbracket) = pre_R(P)$
 by (*rel-auto*)

lemma *cmt-wait-false*:
 $cmt_R(P \llbracket false/\$wait \rrbracket) = cmt_R(P)$
 by (*rel-auto*)

lemma *rea-pre-RHS-design*: $pre_R(\mathbf{R}_s(P \vdash Q)) = R1(R2c(pre_s \uparrow P))$
 by (*simp add: RHS-def usubst R3h-def pre_R-def pre_s-design R1-negate-R1 R2c-not rea-not-def*)

lemma *rea-cmt-RHS-design*: $cmt_R(\mathbf{R}_s(P \vdash Q)) = R1(R2c(cmt_s \uparrow (P \Rightarrow Q)))$
 by (*simp add: RHS-def usubst R3h-def cmt_R-def cmt_s-design R1-idem*)

lemma *rea-peri-RHS-design*: $peri_R(\mathbf{R}_s(P \vdash Q \diamond R)) = R1(R2c(per_i_s \uparrow (P \Rightarrow_r Q)))$
 by (*simp add: RHS-def usubst peri_R-def R3h-def peri_s-design, rel-auto*)

lemma *rea-post-RHS-design*: $post_R(\mathbf{R}_s(P \vdash Q \diamond R)) = R1(R2c(post_s \uparrow (P \Rightarrow_r R)))$
 by (*simp add: RHS-def usubst post_R-def R3h-def post_s-design, rel-auto*)

lemma *peri-cmt-def*: $peri_R(P) = (cmt_R(P)) \llbracket true/\$wait \rrbracket$
 by (*rel-auto*)

lemma *post-cmt-def*: $post_R(P) = (cmt_R(P)) \llbracket false/\$wait \rrbracket$
 by (*rel-auto*)

lemma *rdes-export-cmt*: $\mathbf{R}_s(P \vdash cmt_s \uparrow Q) = \mathbf{R}_s(P \vdash Q)$
 by (*rel-auto*)

lemma *rdes-export-pre*: $\mathbf{R}_s((P \llbracket true, false/\$ok, \$wait \rrbracket) \vdash Q) = \mathbf{R}_s(P \vdash Q)$
 by (*rel-auto*)

4.3.4 Healthiness laws

lemma *wait'-unrest-pre-SRD* [*unrest*]:
 $\$wait' \# pre_R(P) \Longrightarrow \$wait' \# pre_R(SRD\ P)$
 apply (*rel-auto*)
 using *least-zero apply blast+*
 done

lemma *R1-R2s-cmt-SRD*:
 assumes *P is SRD*
 shows $R1(R2s(cmt_R(P))) = cmt_R(P)$
 by (*metis (no-types, lifting) R1-R2c-commute R1-R2s-R2c R1-idem R2c-idem SRD-reactive-design*
assms rea-cmt-RHS-design)

lemma *R1-R2s- peri -SRD*:
assumes *P is SRD*
shows $R1(R2s(\text{peri}_R(P))) = \text{peri}_R(P)$
by (*metis (no-types, hide-lams) Healthy-def R1-R2s-R2c R2-def R2-idem RHS-def SRD-RH-design-form*
assms R1-idem peri_R -def peri_s -R1 peri_s -R2c)

lemma *R1- peri -SRD*:
assumes *P is SRD*
shows $R1(\text{peri}_R(P)) = \text{peri}_R(P)$
proof –
have $R1(\text{peri}_R(P)) = R1(R1(R2s(\text{peri}_R(P))))$
by (*simp add: R1-R2s- peri -SRD assms*)
also have $\dots = \text{peri}_R(P)$
by (*simp add: R1-idem, simp add: R1-R2s- peri -SRD assms*)
finally show ?thesis .
qed

lemma *peri_R -SRD-R1 [closure]*: *P is SRD $\implies \text{peri}_R(P)$ is R1*
by (*simp add: Healthy-def' R1- peri -SRD*)

lemma *R1-R2c- peri -RHS*:
assumes *P is SRD*
shows $R1(R2c(\text{peri}_R(P))) = \text{peri}_R(P)$
by (*metis R1-R2s-R2c R1-R2s- peri -SRD assms*)

lemma *R1-R2s-post-SRD*:
assumes *P is SRD*
shows $R1(R2s(\text{post}_R(P))) = \text{post}_R(P)$
by (*metis (no-types, hide-lams) Healthy-def R1-R2s-R2c R1-idem R2-def R2-idem RHS-def SRD-RH-design-form*
assms post_R -def post_s -R1 post_s -R2c)

lemma *R2c- peri -SRD*:
assumes *P is SRD*
shows $R2c(\text{peri}_R(P)) = \text{peri}_R(P)$
by (*metis R1-R2c-commute R1-R2c- peri -RHS R1- peri -SRD assms*)

lemma *R1-post-SRD*:
assumes *P is SRD*
shows $R1(\text{post}_R(P)) = \text{post}_R(P)$
proof –
have $R1(\text{post}_R(P)) = R1(R1(R2s(\text{post}_R(P))))$
by (*simp add: R1-R2s-post-SRD assms*)
also have $\dots = \text{post}_R(P)$
by (*simp add: R1-idem, simp add: R1-R2s-post-SRD assms*)
finally show ?thesis .
qed

lemma *R2c-post-SRD*:
assumes *P is SRD*
shows $R2c(\text{post}_R(P)) = \text{post}_R(P)$
by (*metis R1-R2c-commute R1-R2s-R2c R1-R2s-post-SRD R1-post-SRD assms*)

lemma *post_R -SRD-R1 [closure]*: *P is SRD $\implies \text{post}_R(P)$ is R1*
by (*simp add: Healthy-def' R1-post-SRD*)

lemma *R1-R2c-post-RHS*:

assumes P is SRD

shows $R1(R2c(post_R(P))) = post_R(P)$

by (*metis R1-R2s-R2c R1-R2s-post-SRD assms*)

lemma *R2-cmt-conj-wait'*:

P is SRD $\implies R2(cmt_R P \wedge \neg \$wait') = (cmt_R P \wedge \neg \$wait')$

by (*simp add: R2-def R2s-conj R2s-not R2s-wait' R1-extend-conj R1-R2s-cmt-SRD*)

lemma *R2c-preR*:

P is SRD $\implies R2c(pre_R(P)) = pre_R(P)$

by (*metis (no-types, lifting) R1-R2c-commute R2c-idem SRD-reactive-design rea-pre-RHS-design*)

lemma *preR-R2c-closed [closure]*: P is SRD $\implies pre_R(P)$ is R2c

by (*simp add: Healthy-def' R2c-preR*)

lemma *R2c-periR*:

P is SRD $\implies R2c(peri_R(P)) = peri_R(P)$

by (*metis (no-types, lifting) R1-R2c-commute R1-R2s-R2c R1-R2s-peri-SRD R2c-idem*)

lemma *periR-R2c-closed [closure]*: P is SRD $\implies peri_R(P)$ is R2c

by (*simp add: Healthy-def R2c-peri-SRD*)

lemma *R2c-postR*:

P is SRD $\implies R2c(post_R(P)) = post_R(P)$

by (*metis (no-types, hide-lams) R1-R2c-commute R1-R2c-is-R2 R1-R2s-post-SRD R2-def R2s-idem*)

lemma *postR-R2c-closed [closure]*: P is SRD $\implies post_R(P)$ is R2c

by (*simp add: Healthy-def R2c-post-SRD*)

lemma *periR-RR [closure]*: P is SRD $\implies peri_R(P)$ is RR

by (*rule RR-intro, simp-all add: closure unrest*)

lemma *postR-RR [closure]*: P is SRD $\implies post_R(P)$ is RR

by (*rule RR-intro, simp-all add: closure unrest*)

lemma *wpR-trace-ident-pre [wp]*:

$(\$tr' =_u \$tr \wedge \lceil II \rceil_R) \wp_r pre_R P = pre_R P$

by (*rel-auto*)

lemma *R1-preR [closure]*:

$pre_R(P)$ is R1

by (*rel-auto*)

lemma *trace-ident-left-periR*:

$(\$tr' =_u \$tr \wedge \lceil II \rceil_R) ;; peri_R(P) = peri_R(P)$

by (*rel-auto*)

lemma *trace-ident-left-postR*:

$(\$tr' =_u \$tr \wedge \lceil II \rceil_R) ;; post_R(P) = post_R(P)$

by (*rel-auto*)

lemma *trace-ident-right-postR*:

$post_R(P) ;; (\$tr' =_u \$tr \wedge \lceil II \rceil_R) = post_R(P)$

by (*rel-auto*)

lemma *preR-R2-closed* [closure]: P is SRD \implies $\text{pre}_R(P)$ is R2
 by (simp add: R2-comp-def Healthy-comp closure)

lemma *periR-R2-closed* [closure]: P is SRD \implies $\text{peri}_R(P)$ is R2
 by (simp add: Healthy-def' R1-R2c-peri-RHS R2-R2c-def)

lemma *postR-R2-closed* [closure]: P is SRD \implies $\text{post}_R(P)$ is R2
 by (simp add: Healthy-def' R1-R2c-post-RHS R2-R2c-def)

4.3.5 Calculation laws

lemma *wait'-cond-peri-post-cmt* [rdes]:
 $\text{cmt}_R P = \text{peri}_R P \diamond \text{post}_R P$
 by (rel-auto)

lemma *preR-rdes* [rdes]:
 assumes P is RR
 shows $\text{pre}_R(\mathbf{R}_s(P \vdash Q \diamond R)) = P$
 by (simp add: rea-pre-RHS-design unrest usubst assms Healthy-if RR-implies-R2c RR-implies-R1)

lemma *periR-rdes* [rdes]:
 assumes P is RR Q is RR
 shows $\text{peri}_R(\mathbf{R}_s(P \vdash Q \diamond R)) = (P \Rightarrow_r Q)$
 by (simp add: rea-peri-RHS-design unrest usubst assms Healthy-if RR-implies-R2c closure)

lemma *postR-rdes* [rdes]:
 assumes P is RR R is RR
 shows $\text{post}_R(\mathbf{R}_s(P \vdash Q \diamond R)) = (P \Rightarrow_r R)$
 by (simp add: rea-post-RHS-design unrest usubst assms Healthy-if RR-implies-R2c closure)

lemma *preR-Chaos* [rdes]: $\text{pre}_R(\text{Chaos}) = \text{false}$
 by (simp add: Chaos-def, rel-simp)

lemma *periR-Chaos* [rdes]: $\text{peri}_R(\text{Chaos}) = \text{true}_r$
 by (simp add: Chaos-def, rel-simp)

lemma *postR-Chaos* [rdes]: $\text{post}_R(\text{Chaos}) = \text{true}_r$
 by (simp add: Chaos-def, rel-simp)

lemma *preR-Miracle* [rdes]: $\text{pre}_R(\text{Miracle}) = \text{true}_r$
 by (simp add: Miracle-def, rel-auto)

lemma *periR-Miracle* [rdes]: $\text{peri}_R(\text{Miracle}) = \text{false}$
 by (simp add: Miracle-def, rel-auto)

lemma *postR-Miracle* [rdes]: $\text{post}_R(\text{Miracle}) = \text{false}$
 by (simp add: Miracle-def, rel-auto)

lemma *preR-srdes-skip* [rdes]: $\text{pre}_R(\text{II}_R) = \text{true}_r$
 by (rel-auto)

lemma *periR-srdes-skip* [rdes]: $\text{peri}_R(\text{II}_R) = \text{false}$
 by (rel-auto)

lemma *postR-srdes-skip* [rdes]: $\text{post}_R(\text{II}_R) = (\$tr' =_u \$tr \wedge [\text{II}]_R)$

by (rel-auto)

lemma *preR-INF* [rdes]: $A \neq \{\}$ $\implies pre_R(\sqcap A) = (\bigwedge P \in A \cdot pre_R(P))$
by (rel-auto)

lemma *periR-INF* [rdes]: $peri_R(\sqcap A) = (\bigvee P \in A \cdot peri_R(P))$
by (rel-auto)

lemma *postR-INF* [rdes]: $post_R(\sqcap A) = (\bigvee P \in A \cdot post_R(P))$
by (rel-auto)

lemma *preR-UINF* [rdes]: $pre_R(\sqcap i \cdot P(i)) = (\sqcup i \cdot pre_R(P(i)))$
by (rel-auto)

lemma *periR-UINF* [rdes]: $peri_R(\sqcap i \cdot P(i)) = (\sqcap i \cdot peri_R(P(i)))$
by (rel-auto)

lemma *postR-UINF* [rdes]: $post_R(\sqcap i \cdot P(i)) = (\sqcap i \cdot post_R(P(i)))$
by (rel-auto)

lemma *preR-UINF-member* [rdes]: $A \neq \{\}$ $\implies pre_R(\sqcap i \in A \cdot P(i)) = (\sqcup i \in A \cdot pre_R(P(i)))$
by (rel-auto)

lemma *preR-UINF-member-2* [rdes]: $A \neq \{\}$ $\implies pre_R(\sqcap (i,j) \in A \cdot P i j) = (\sqcup (i,j) \in A \cdot pre_R(P i j))$
by (rel-auto)

lemma *preR-UINF-member-3* [rdes]: $A \neq \{\}$ $\implies pre_R(\sqcap (i,j,k) \in A \cdot P i j k) = (\sqcup (i,j,k) \in A \cdot pre_R(P i j k))$
by (rel-auto)

lemma *periR-UINF-member* [rdes]: $peri_R(\sqcap i \in A \cdot P(i)) = (\sqcap i \in A \cdot peri_R(P(i)))$
by (rel-auto)

lemma *periR-UINF-member-2* [rdes]: $peri_R(\sqcap (i,j) \in A \cdot P i j) = (\sqcap (i,j) \in A \cdot peri_R(P i j))$
by (rel-auto)

lemma *periR-UINF-member-3* [rdes]: $peri_R(\sqcap (i,j,k) \in A \cdot P i j k) = (\sqcap (i,j,k) \in A \cdot peri_R(P i j k))$
by (rel-auto)

lemma *postR-UINF-member* [rdes]: $post_R(\sqcap i \in A \cdot P(i)) = (\sqcap i \in A \cdot post_R(P(i)))$
by (rel-auto)

lemma *postR-UINF-member-2* [rdes]: $post_R(\sqcap (i,j) \in A \cdot P i j) = (\sqcap (i,j) \in A \cdot post_R(P i j))$
by (rel-auto)

lemma *postR-UINF-member-3* [rdes]: $post_R(\sqcap (i,j,k) \in A \cdot P i j k) = (\sqcap (i,j,k) \in A \cdot post_R(P i j k))$
by (rel-auto)

lemma *preR-inf* [rdes]: $pre_R(P \sqcap Q) = (pre_R(P) \wedge pre_R(Q))$
by (rel-auto)

lemma *periR-inf* [rdes]: $peri_R(P \sqcap Q) = (peri_R(P) \vee peri_R(Q))$
by (rel-auto)

lemma *postR-inf* [rdes]: $post_R(P \sqcap Q) = (post_R(P) \vee post_R(Q))$

by (rel-auto)

lemma *preR-SUP* [rdes]: $\text{pre}_R(\sqcup A) = (\bigvee P \in A \cdot \text{pre}_R(P))$
by (rel-auto)

lemma *periR-SUP* [rdes]: $A \neq \{\} \implies \text{peri}_R(\sqcup A) = (\bigwedge P \in A \cdot \text{peri}_R(P))$
by (rel-auto)

lemma *postR-SUP* [rdes]: $A \neq \{\} \implies \text{post}_R(\sqcup A) = (\bigwedge P \in A \cdot \text{post}_R(P))$
by (rel-auto)

4.4 Formation laws

lemma *srdes-skip-tri-design* [rdes-def]: $II_R = \mathbf{R}_s(\text{true}_r \vdash \text{false} \diamond II_r)$
by (simp add: srdes-skip-def, rel-auto)

lemma *Chaos-tri-def* [rdes-def]: $\text{Chaos} = \mathbf{R}_s(\text{false} \vdash \text{false} \diamond \text{false})$
by (simp add: Chaos-def design-false-pre)

lemma *Miracle-tri-def* [rdes-def]: $\text{Miracle} = \mathbf{R}_s(\text{true}_r \vdash \text{false} \diamond \text{false})$
by (simp add: Miracle-def R1-design-R1-pre wait'-cond-idem)

lemma *RHS-tri-design-form*:

assumes P_1 is RR P_2 is RR P_3 is RR

shows $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) = (II_R \triangleleft \$wait \triangleright ((\$ok \wedge P_1) \Rightarrow_r (\$ok' \wedge (P_2 \diamond P_3))))$

proof –

have $\mathbf{R}_s(RR(P_1) \vdash RR(P_2) \diamond RR(P_3)) = (II_R \triangleleft \$wait \triangleright ((\$ok \wedge RR(P_1)) \Rightarrow_r (\$ok' \wedge (RR(P_2) \diamond RR(P_3)))))$

apply (rel-auto) using minus-zero-eq by blast

thus ?thesis

by (simp add: Healthy-if asms)

qed

lemma *RHS-design-pre-post-form*:

$\mathbf{R}_s((\neg P^f_f) \vdash P^t_f) = \mathbf{R}_s(\text{pre}_R(P) \vdash \text{cmt}_R(P))$

proof –

have $\mathbf{R}_s((\neg P^f_f) \vdash P^t_f) = \mathbf{R}_s((\neg P^f_f)[\text{true}/\$ok] \vdash P^t_f[\text{true}/\$ok])$

by (simp add: design-subst-ok)

also have $\dots = \mathbf{R}_s(\text{pre}_R(P) \vdash \text{cmt}_R(P))$

by (simp add: preR-def cmtR-def usubst, rel-auto)

finally show ?thesis .

qed

lemma *SRD-as-reactive-design*:

$\text{SRD}(P) = \mathbf{R}_s(\text{pre}_R(P) \vdash \text{cmt}_R(P))$

by (simp add: RHS-design-pre-post-form SRD-RH-design-form)

lemma *SRD-reactive-design-alt*:

assumes P is SRD

shows $\mathbf{R}_s(\text{pre}_R(P) \vdash \text{cmt}_R(P)) = P$

proof –

have $\mathbf{R}_s(\text{pre}_R(P) \vdash \text{cmt}_R(P)) = \mathbf{R}_s((\neg P^f_f) \vdash P^t_f)$

by (simp add: RHS-design-pre-post-form)

thus ?thesis

by (simp add: SRD-reactive-design asms)

qed

lemma *SRD-reactive-tri-design-lemma*:

$SRD(P) = \mathbf{R}_s((\neg P^f_f) \vdash P^t_f \llbracket true/\$wait' \rrbracket \diamond P^t_f \llbracket false/\$wait' \rrbracket)$
 by (*simp add: SRD-RH-design-form wait'-cond-split*)

lemma *SRD-as-reactive-tri-design*:

$SRD(P) = \mathbf{R}_s(pre_R(P) \vdash peri_R(P) \diamond post_R(P))$

proof –

have $SRD(P) = \mathbf{R}_s((\neg P^f_f) \vdash P^t_f \llbracket true/\$wait' \rrbracket \diamond P^t_f \llbracket false/\$wait' \rrbracket)$

by (*simp add: SRD-RH-design-form wait'-cond-split*)

also have $\dots = \mathbf{R}_s(pre_R(P) \vdash peri_R(P) \diamond post_R(P))$

apply (*simp add: usubst*)

apply (*subst design-subst-ok-ok'[THEN sym]*)

apply (*simp add: pre_R-def peri_R-def post_R-def usubst unrest*)

apply (*rel-auto*)

done

finally show *?thesis* .

qed

lemma *SRD-reactive-tri-design*:

assumes *P is SRD*

shows $\mathbf{R}_s(pre_R(P) \vdash peri_R(P) \diamond post_R(P)) = P$

by (*metis Healthy-if SRD-as-reactive-tri-design assms*)

lemma *SRD-elim* [*RD-elim*]: $\llbracket P \text{ is } SRD; Q(\mathbf{R}_s(pre_R(P) \vdash peri_R(P) \diamond post_R(P))) \rrbracket \implies Q(P)$

by (*simp add: SRD-reactive-tri-design*)

lemma *RHS-tri-design-is-SRD* [*closure*]:

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

shows $\mathbf{R}_s(P \vdash Q \diamond R)$ *is SRD*

by (*rule RHS-design-is-SRD, simp-all add: unrest assms*)

lemma *SRD-rdes-intro* [*closure*]:

assumes *P is RR Q is RR R is RR*

shows $\mathbf{R}_s(P \vdash Q \diamond R)$ *is SRD*

by (*rule RHS-tri-design-is-SRD, simp-all add: unrest closure assms*)

lemma *USUP-R1-R2s-cmt-SRD*:

assumes $A \subseteq \llbracket SRD \rrbracket_H$

shows $(\bigsqcup P \in A \cdot R1 (R2s (cmt_R P))) = (\bigsqcup P \in A \cdot cmt_R P)$

by (*rule USUP-cong[of A], metis (mono-tags, lifting) Ball-Collect R1-R2s-cmt-SRD assms*)

lemma *UINF-R1-R2s-cmt-SRD*:

assumes $A \subseteq \llbracket SRD \rrbracket_H$

shows $(\bigsqcap P \in A \cdot R1 (R2s (cmt_R P))) = (\bigsqcap P \in A \cdot cmt_R P)$

by (*rule UINF-cong[of A], metis (mono-tags, lifting) Ball-Collect R1-R2s-cmt-SRD assms*)

4.4.1 Order laws

lemma *preR-antitone*: $P \sqsubseteq Q \implies pre_R(Q) \sqsubseteq pre_R(P)$

by (*rel-auto*)

lemma *periR-monotone*: $P \sqsubseteq Q \implies peri_R(P) \sqsubseteq peri_R(Q)$

by (*rel-auto*)

lemma *postR-monotone*: $P \sqsubseteq Q \implies post_R(P) \sqsubseteq post_R(Q)$

by (*rel-auto*)

4.5 Composition laws

theorem *RH-tri-design-composition*:

assumes $\$ok' \# P \$ok' \# Q_1 \$ok' \# Q_2 \$ok \# R \$ok \# S_1 \$ok \# S_2$

$\$wait' \# Q_2 \$wait \# S_1 \$wait \# S_2$

shows $(RH(P \vdash Q_1 \diamond Q_2) ;; RH(R \vdash S_1 \diamond S_2)) =$

$RH((\neg (R1 (\neg R2s P) ;; R1 true) \wedge \neg ((R1 (R2s Q_2) \wedge \neg \$wait') ;; R1 (\neg R2s R))) \vdash$
 $((Q_1 \vee (R1 (R2s Q_2) ;; R1 (R2s S_1))) \diamond ((R1 (R2s Q_2) ;; R1 (R2s S_2))))$

proof –

have $1: (\neg ((R1 (R2s (Q_1 \diamond Q_2)) \wedge \neg \$wait') ;; R1 (\neg R2s R))) =$
 $(\neg ((R1 (R2s Q_2) \wedge \neg \$wait') ;; R1 (\neg R2s R)))$

by (*metis* (*no-types*, *hide-lams*) *R1-extend-conj R2s-conj R2s-not R2s-wait' wait'-cond-false*)

have $2: (R1 (R2s (Q_1 \diamond Q_2)) ;; ([II]_D \triangleleft \$wait \triangleright R1 (R2s (S_1 \diamond S_2)))) =$

$((R1 (R2s Q_1) \vee (R1 (R2s Q_2) ;; R1 (R2s S_1))) \diamond (R1 (R2s Q_2) ;; R1 (R2s S_2)))$

proof –

have $(R1 (R2s Q_1) ;; (\$wait \wedge ([II]_D \triangleleft \$wait \triangleright R1 (R2s S_1) \diamond R1 (R2s S_2))))$
 $= (R1 (R2s Q_1) \wedge \$wait')$

proof –

have $(R1 (R2s Q_1) ;; (\$wait \wedge ([II]_D \triangleleft \$wait \triangleright R1 (R2s S_1) \diamond R1 (R2s S_2))))$
 $= (R1 (R2s Q_1) ;; (\$wait \wedge [II]_D))$

by (*rel-auto*)

also have $\dots = ((R1 (R2s Q_1) ;; [II]_D) \wedge \$wait')$

by (*rel-auto*)

also from *assms*(2) **have** $\dots = ((R1 (R2s Q_1)) \wedge \$wait')$

by (*simp add: lift-des-skip-dr-unit-unrest unrest*)

finally show *?thesis* .

qed

moreover have $(R1 (R2s Q_2) ;; (\neg \$wait \wedge ([II]_D \triangleleft \$wait \triangleright R1 (R2s S_1) \diamond R1 (R2s S_2))))$
 $= ((R1 (R2s Q_2)) ;; (R1 (R2s S_1) \diamond R1 (R2s S_2)))$

proof –

have $(R1 (R2s Q_2) ;; (\neg \$wait \wedge ([II]_D \triangleleft \$wait \triangleright R1 (R2s S_1) \diamond R1 (R2s S_2))))$
 $= (R1 (R2s Q_2) ;; (\neg \$wait \wedge (R1 (R2s S_1) \diamond R1 (R2s S_2))))$

by (*metis* (*no-types*, *lifting*) *cond-def conj-disj-not-abs utp-pred-laws.double-compl utp-pred-laws.inf.left-idem*
utp-pred-laws.sup-assoc utp-pred-laws.sup-inf-absorb)

also have $\dots = ((R1 (R2s Q_2)) \llbracket false/\$wait' \rrbracket ;; (R1 (R2s S_1) \diamond R1 (R2s S_2)) \llbracket false/\$wait' \rrbracket)$
by (*metis false-alt-def seqr-right-one-point upred-eq-false wait-vwb-lens*)

also have $\dots = ((R1 (R2s Q_2)) ;; (R1 (R2s S_1) \diamond R1 (R2s S_2)))$

by (*simp add: wait'-cond-def usubst unrest assms*)

finally show *?thesis* .

qed

moreover

have $((R1 (R2s Q_1) \wedge \$wait') \vee ((R1 (R2s Q_2)) ;; (R1 (R2s S_1) \diamond R1 (R2s S_2))))$
 $= (R1 (R2s Q_1) \vee (R1 (R2s Q_2) ;; R1 (R2s S_1))) \diamond ((R1 (R2s Q_2) ;; R1 (R2s S_2)))$

by (*simp add: wait'-cond-def cond-seq-right-distr cond-and-T-integrate unrest*)

ultimately show *?thesis*

by (*simp add: R2s-wait'-cond R1-wait'-cond wait'-cond-seq*)

qed

show *?thesis*
apply (*subst RH-design-composition*)
apply (*simp-all add: assms*)
apply (*simp add: assms wait'-cond-def unrest*)
apply (*simp add: assms wait'-cond-def unrest*)
apply (*simp add: 1 2*)
apply (*simp add: R1-R2s-R2c RH-design-lemma1*)
done
qed

theorem *R1-design-composition-RR*:
assumes *P is RR Q is RR R is RR S is RR*
shows
 $(R1(P \vdash Q) ;; R1(R \vdash S)) = R1(((\neg_r P) \text{ wp}_r \text{ false} \wedge Q \text{ wp}_r R) \vdash (Q ;; S))$
apply (*subst R1-design-composition*)
apply (*simp-all add: assms unrest wp-rea-def Healthy-if closure*)
apply (*rel-auto*)
done

theorem *R1-design-composition-RC*:
assumes *P is RC Q is RR R is RR S is RR*
shows
 $(R1(P \vdash Q) ;; R1(R \vdash S)) = R1((P \wedge Q \text{ wp}_r R) \vdash (Q ;; S))$
by (*simp add: R1-design-composition-RR assms unrest Healthy-if closure wp*)

theorem *RHS-tri-design-composition*:
assumes $\$ok' \# P \$ok' \# Q_1 \$ok' \# Q_2 \$ok \# R \$ok \# S_1 \$ok \# S_2$
 $\$wait \# R \$wait' \# Q_2 \$wait \# S_1 \$wait \# S_2$
shows $(\mathbf{R}_s(P \vdash Q_1 \diamond Q_2) ;; \mathbf{R}_s(R \vdash S_1 \diamond S_2)) =$
 $\mathbf{R}_s((\neg (R1 (\neg R2s P) ;; R1 \text{ true}) \wedge \neg (R1(R2s Q_2) ;; R1 (\neg R2s R))) \vdash$
 $((\exists \$st' \cdot Q_1) \vee (R1 (R2s Q_2) ;; R1 (R2s S_1))) \diamond ((R1 (R2s Q_2) ;; R1 (R2s S_2))))$

proof –
have $1: (\neg ((R1 (R2s (Q_1 \diamond Q_2)) \wedge \neg \$wait') ;; R1 (\neg R2s R))) =$
 $(\neg ((R1 (R2s Q_2) \wedge \neg \$wait') ;; R1 (\neg R2s R)))$
by (*metis (no-types, hide-lams) R1-extend-conj R2s-conj R2s-not R2s-wait' wait'-cond-false*)
have $2: (R1 (R2s (Q_1 \diamond Q_2)) ;; ((\exists \$st \cdot \lceil II \rceil_D) \triangleleft \$wait \triangleright R1 (R2s (S_1 \diamond S_2)))) =$
 $((\exists \$st' \cdot R1 (R2s Q_1)) \vee (R1 (R2s Q_2) ;; R1 (R2s S_1))) \diamond (R1 (R2s Q_2) ;; R1 (R2s S_2)))$

proof –
have $(R1 (R2s Q_1) ;; (\$wait \wedge ((\exists \$st \cdot \lceil II \rceil_D) \triangleleft \$wait \triangleright R1 (R2s S_1) \diamond R1 (R2s S_2))))$
 $= (\exists \$st' \cdot ((R1 (R2s Q_1)) \wedge \$wait'))$

proof –
have $(R1 (R2s Q_1) ;; (\$wait \wedge ((\exists \$st \cdot \lceil II \rceil_D) \triangleleft \$wait \triangleright R1 (R2s S_1) \diamond R1 (R2s S_2))))$
 $= (R1 (R2s Q_1) ;; (\$wait \wedge (\exists \$st \cdot \lceil II \rceil_D)))$
by (*rel-auto, blast+*)
also have $\dots = ((R1 (R2s Q_1) ;; (\exists \$st \cdot \lceil II \rceil_D)) \wedge \$wait')$
by (*rel-auto*)
also from *assms(2)* **have** $\dots = (\exists \$st' \cdot ((R1 (R2s Q_1)) \wedge \$wait'))$
by (*rel-auto, blast*)
finally show *?thesis* .

qed
moreover have $(R1 (R2s Q_2) ;; (\neg \$wait \wedge ((\exists \$st \cdot \lceil II \rceil_D) \triangleleft \$wait \triangleright R1 (R2s S_1) \diamond R1 (R2s S_2))))$
 $= ((R1 (R2s Q_2)) ;; (R1 (R2s S_1) \diamond R1 (R2s S_2)))$

proof –
have $(R1 (R2s Q_2) ;; (\neg \$wait \wedge ((\exists \$st \cdot \lceil II \rceil_D) \triangleleft \$wait \triangleright R1 (R2s S_1) \diamond R1 (R2s S_2))))$
 $= (R1 (R2s Q_2) ;; (\neg \$wait \wedge (R1 (R2s S_1) \diamond R1 (R2s S_2))))$
by $(metis (no-types, lifting) cond-def conj-disj-not-abs utp-pred-laws.double-compl utp-pred-laws.inf.left-idem utp-pred-laws.sup-assoc utp-pred-laws.sup-inf-absorb)$

also have $\dots = ((R1 (R2s Q_2)) \llbracket false / \$wait \rrbracket ;; (R1 (R2s S_1) \diamond R1 (R2s S_2)) \llbracket false / \$wait \rrbracket)$
by $(metis false-alt-def segr-right-one-point upred-eq-false wait-vwb-lens)$

also have $\dots = ((R1 (R2s Q_2)) ;; (R1 (R2s S_1) \diamond R1 (R2s S_2)))$
by $(simp add: wait'-cond-def usubst unrest assms)$

finally show $?thesis$.
qed

moreover
have $((R1 (R2s Q_1) \wedge \$wait') \vee ((R1 (R2s Q_2)) ;; (R1 (R2s S_1) \diamond R1 (R2s S_2))))$
 $= (R1 (R2s Q_1) \vee (R1 (R2s Q_2) ;; R1 (R2s S_1))) \diamond ((R1 (R2s Q_2) ;; R1 (R2s S_2)))$
by $(simp add: wait'-cond-def cond-seq-right-distr cond-and-T-integrate unrest)$

ultimately show $?thesis$
by $(simp add: R2s-wait'-cond R1-wait'-cond wait'-cond-seq ex-conj-contr-right unrest)$
 $(simp add: cond-and-T-integrate cond-seq-right-distr unrest-var wait'-cond-def)$
qed

from $assms(7,8)$ **have** $\exists: (R1 (R2s Q_2) \wedge \neg \$wait') ;; R1 (\neg R2s R) = R1 (R2s Q_2) ;; R1 (\neg R2s R)$
by $(rel-auto, blast, meson)$

show $?thesis$
apply $(subst RHS-design-composition)$
apply $(simp-all add: assms)$
apply $(simp add: assms wait'-cond-def unrest)$
apply $(simp add: assms wait'-cond-def unrest)$
apply $(simp add: 1 2 3)$
apply $(simp add: R1-R2s-R2c RHS-design-lemma1)$
apply $(metis R1-R2c-ex-st RHS-design-lemma1)$
done
qed

theorem $RHS-tri-design-composition-wp$:
assumes $\$ok' \# P \$ok' \# Q_1 \$ok' \# Q_2 \$ok \# R \$ok \# S_1 \$ok \# S_2$
 $\$wait \# R \$wait' \# Q_2 \$wait \# S_1 \$wait \# S_2$
 $P \text{ is } R2c \ Q_1 \text{ is } R1 \ Q_1 \text{ is } R2c \ Q_2 \text{ is } R1 \ Q_2 \text{ is } R2c$
 $R \text{ is } R2c \ S_1 \text{ is } R1 \ S_1 \text{ is } R2c \ S_2 \text{ is } R1 \ S_2 \text{ is } R2c$
shows $\mathbf{R}_s(P \vdash Q_1 \diamond Q_2) ;; \mathbf{R}_s(R \vdash S_1 \diamond S_2) =$
 $\mathbf{R}_s(((\neg_r P) \text{ wp}_r false \wedge Q_2 \text{ wp}_r R) \vdash (((\exists \$st' \cdot Q_1) \sqcap (Q_2 ;; S_1)) \diamond (Q_2 ;; S_2)))$ **(is ?lhs = ?rhs)**
proof –
have $?lhs = \mathbf{R}_s((\neg R1 (\neg P) ;; R1 true \wedge \neg Q_2 ;; R1 (\neg R)) \vdash ((\exists \$st' \cdot Q_1) \sqcap Q_2 ;; S_1) \diamond Q_2 ;; S_2)$
by $(simp add: RHS-tri-design-composition assms Healthy-if R2c-healthy-R2s disj-upred-def)$
 $(metis (no-types, hide-lams) R1-negate-R1 R2c-healthy-R2s assms(11,16))$
also have $\dots = ?rhs$
by $(rel-auto)$

finally show *?thesis* .
qed

theorem *RHS-tri-design-composition-RR-wp*:

assumes *P is RR Q₁ is RR Q₂ is RR*

R is RR S₁ is RR S₂ is RR

shows $\mathbf{R}_s(P \vdash Q_1 \diamond Q_2) ;; \mathbf{R}_s(R \vdash S_1 \diamond S_2) =$

$\mathbf{R}_s(((\neg_r P) \text{ wp}_r \text{ false} \wedge Q_2 \text{ wp}_r R) \vdash ((\exists \$st' \cdot Q_1) \sqcap (Q_2 ;; S_1)) \diamond (Q_2 ;; S_2)))$ (*is ?lhs = ?rhs*)

by (*simp add: RHS-tri-design-composition-wp add: closure assms unrest RR-implies-R2c*)

lemma *RHS-tri-normal-design-composition*:

assumes

$\$ok' \# P \$ok' \# Q_1 \$ok' \# Q_2 \$ok \# R \$ok \# S_1 \$ok \# S_2$

$\$wait \# R \$wait' \# Q_2 \$wait \# S_1 \$wait \# S_2$

P is R2c Q₁ is R1 Q₁ is R2c Q₂ is R1 Q₂ is R2c

R is R2c S₁ is R1 S₁ is R2c S₂ is R1 S₂ is R2c

R1 ($\neg P$) ;; R1(true) = R1($\neg P$) \$st' # Q₁

shows $\mathbf{R}_s(P \vdash Q_1 \diamond Q_2) ;; \mathbf{R}_s(R \vdash S_1 \diamond S_2)$

$= \mathbf{R}_s((P \wedge Q_2 \text{ wp}_r R) \vdash (Q_1 \vee (Q_2 ;; S_1)) \diamond (Q_2 ;; S_2))$

proof –

have $\mathbf{R}_s(P \vdash Q_1 \diamond Q_2) ;; \mathbf{R}_s(R \vdash S_1 \diamond S_2) =$

$\mathbf{R}_s((R1 (\neg P) \text{ wp}_r \text{ false} \wedge Q_2 \text{ wp}_r R) \vdash ((\exists \$st' \cdot Q_1) \sqcap (Q_2 ;; S_1)) \diamond (Q_2 ;; S_2))$

by (*simp-all add: RHS-tri-design-composition-wp rea-not-def assms unrest*)

also have ... = $\mathbf{R}_s((P \wedge Q_2 \text{ wp}_r R) \vdash (Q_1 \vee (Q_2 ;; S_1)) \diamond (Q_2 ;; S_2))$

by (*simp add: assms wp-rea-def ex-unrest, rel-auto*)

finally show *?thesis* .

qed

lemma *RHS-tri-normal-design-composition' [rdes-def]*:

assumes *P is RC Q₁ is RR \$st' # Q₁ Q₂ is RR R is RR S₁ is RR S₂ is RR*

shows $\mathbf{R}_s(P \vdash Q_1 \diamond Q_2) ;; \mathbf{R}_s(R \vdash S_1 \diamond S_2)$

$= \mathbf{R}_s((P \wedge Q_2 \text{ wp}_r R) \vdash (Q_1 \vee (Q_2 ;; S_1)) \diamond (Q_2 ;; S_2))$

proof –

have *R1 ($\neg P$) ;; R1 true = R1($\neg P$)*

using *RC-implies-RC1[OF assms(1)]*

by (*simp add: Healthy-def RC1-def rea-not-def*)

(*metis R1-negate-R1 R1-seqr utp-pred-laws.double-compl*)

thus *?thesis*

by (*simp add: RHS-tri-normal-design-composition assms closure unrest RR-implies-R2c*)

qed

lemma *RHS-tri-design-right-unit-lemma*:

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

shows $\mathbf{R}_s(P \vdash Q \diamond R) ;; II_R = \mathbf{R}_s((\neg_r (\neg_r P) ;; \text{true}_r) \vdash ((\exists \$st' \cdot Q) \diamond R))$

proof –

have $\mathbf{R}_s(P \vdash Q \diamond R) ;; II_R = \mathbf{R}_s(P \vdash Q \diamond R) ;; \mathbf{R}_s(\text{true} \vdash \text{false} \diamond (\$tr' =_u \$tr \wedge [II]_R))$

by (*simp add: srdes-skip-tri-design, rel-auto*)

also have ... = $\mathbf{R}_s((\neg R1 (\neg R2s P) ;; R1 \text{true}) \vdash (\exists \$st' \cdot Q) \diamond (R1 (R2s R) ;; R1 (R2s (\$tr' =_u \$tr \wedge [II]_R))))$

by (*simp-all add: RHS-tri-design-composition assms unrest R2s-true R1-false R2s-false*)

also have ... = $\mathbf{R}_s((\neg R1 (\neg R2s P) ;; R1 \text{true}) \vdash (\exists \$st' \cdot Q) \diamond R1 (R2s R))$

proof –

from *assms(3,4)* have $(R1 (R2s R) ;; R1 (R2s (\$tr' =_u \$tr \wedge [II]_R))) = R1 (R2s R)$

by (*rel-auto, metis (no-types, lifting) minus-zero-eq, meson order-refl trace-class.diff-cancel*)

thus ?thesis
 by simp
 qed
 also have ... = $\mathbf{R}_s((\neg (\neg P) ;; R1 \text{ true}) \vdash ((\exists \$st' \cdot Q) \diamond R))$
 by (metis (no-types, lifting) R1-R2s-R1-true-lemma R1-R2s-R2c R2c-not RHS-design-R2c-pre RHS-design-neg-R1-pre
 RHS-design-post-R1 RHS-design-post-R2s)
 also have ... = $\mathbf{R}_s((\neg_r (\neg_r P) ;; true_r) \vdash ((\exists \$st' \cdot Q) \diamond R))$
 by (rel-auto)
 finally show ?thesis .
 qed

lemma SRD-composition-wp:

assumes P is SRD Q is SRD

shows $(P ;; Q) = \mathbf{R}_s(((\neg_r pre_R P) wp_r false \wedge post_R P wp_r pre_R Q) \vdash$
 $((\exists \$st' \cdot peri_R P) \vee (post_R P ;; peri_R Q)) \diamond (post_R P ;; post_R Q))$

(is ?lhs = ?rhs)

proof –

have $(P ;; Q) = (\mathbf{R}_s(pre_R(P) \vdash peri_R(P) \diamond post_R(P)) ;; \mathbf{R}_s(pre_R(Q) \vdash peri_R(Q) \diamond post_R(Q)))$

by (simp add: SRD-reactive-tri-design assms(1) assms(2))

also from assms

have ... = ?rhs

by (simp add: RHS-tri-design-composition-wp disj-upred-def unrest assms closure)

finally show ?thesis .

qed

4.6 Refinement introduction laws

lemma RHS-tri-design-refine:

assumes P_1 is RR P_2 is RR P_3 is RR Q_1 is RR Q_2 is RR Q_3 is RR

shows $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) \sqsubseteq \mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3) \longleftrightarrow 'P_1 \Rightarrow Q_1' \wedge 'P_1 \wedge Q_2 \Rightarrow P_2' \wedge 'P_1 \wedge Q_3 \Rightarrow$
 P_3'

(is ?lhs = ?rhs)

proof –

have $?lhs \longleftrightarrow 'P_1 \Rightarrow Q_1' \wedge 'P_1 \wedge Q_2 \diamond Q_3 \Rightarrow P_2 \diamond P_3'$

by (simp add: RHS-design-refine assms closure RR-implies-R2c unrest ex-unrest)

also have ... $\longleftrightarrow 'P_1 \Rightarrow Q_1' \wedge '(P_1 \wedge Q_2) \diamond (P_1 \wedge Q_3) \Rightarrow P_2 \diamond P_3'$

by (rel-auto)

also have ... $\longleftrightarrow 'P_1 \Rightarrow Q_1' \wedge '((P_1 \wedge Q_2) \diamond (P_1 \wedge Q_3) \Rightarrow P_2 \diamond P_3)[[true/\$wait]]' \wedge '((P_1 \wedge Q_2) \diamond$
 $(P_1 \wedge Q_3) \Rightarrow P_2 \diamond P_3)[[false/\$wait]]'$

by (rel-auto, metis)

also have ... $\longleftrightarrow ?rhs$

by (simp add: usubst unrest assms)

finally show ?thesis .

qed

lemma sdes-tri-refine-intro:

assumes $'P_1 \Rightarrow P_2' \wedge 'P_1 \wedge Q_2 \Rightarrow Q_1' \wedge 'P_1 \wedge R_2 \Rightarrow R_1'$

shows $\mathbf{R}_s(P_1 \vdash Q_1 \diamond R_1) \sqsubseteq \mathbf{R}_s(P_2 \vdash Q_2 \diamond R_2)$

using assms

by (rule-tac sdes-refine-intro, simp-all, rel-auto)

lemma sdes-tri-eq-intro:

assumes $P_1 = Q_1 \wedge P_2 = Q_2 \wedge P_3 = Q_3$

shows $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) = \mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3)$

using assms by (simp)

lemma *srdes-tri-refine-intro'*:

assumes $P_2 \sqsubseteq P_1 \quad Q_1 \sqsubseteq (P_1 \wedge Q_2) \quad R_1 \sqsubseteq (P_1 \wedge R_2)$
shows $\mathbf{R}_s(P_1 \vdash Q_1 \diamond R_1) \sqsubseteq \mathbf{R}_s(P_2 \vdash Q_2 \diamond R_2)$
using *assms*
by (*rule-tac srdes-tri-refine-intro, simp-all add: refBy-order*)

lemma *SRD-peri-under-pre*:

assumes P is SRD $\$wait' \nmid pre_R(P)$
shows $(pre_R(P) \Rightarrow_r peri_R(P)) = peri_R(P)$

proof –

have $peri_R(P) =$
 $peri_R(\mathbf{R}_s(pre_R(P) \vdash peri_R(P) \diamond post_R(P)))$
by (*simp add: SRD-reactive-tri-design assms*)
also have $\dots = (pre_R P \Rightarrow_r peri_R P)$
by (*simp add: rea-pre-RHS-design rea-peri-RHS-design assms*
 $unrest usubst R1-peri-SRD R2c-preR R1-rea-impl R2c-rea-impl R2c-periR$)
finally show *?thesis* ..

qed

lemma *SRD-post-under-pre*:

assumes P is SRD $\$wait' \nmid pre_R(P)$
shows $(pre_R(P) \Rightarrow_r post_R(P)) = post_R(P)$

proof –

have $post_R(P) =$
 $post_R(\mathbf{R}_s(pre_R(P) \vdash peri_R(P) \diamond post_R(P)))$
by (*simp add: SRD-reactive-tri-design assms*)
also have $\dots = (pre_R P \Rightarrow_r post_R P)$
by (*simp add: rea-pre-RHS-design rea-post-RHS-design assms*
 $unrest usubst R1-post-SRD R2c-preR R1-rea-impl R2c-rea-impl R2c-postR$)
finally show *?thesis* ..

qed

lemma *SRD-refine-intro*:

assumes
 P is SRD Q is SRD
 $'pre_R(P) \Rightarrow pre_R(Q)'$ $'pre_R(P) \wedge peri_R(Q) \Rightarrow peri_R(P)'$ $'pre_R(P) \wedge post_R(Q) \Rightarrow post_R(P)'$
shows $P \sqsubseteq Q$
by (*metis SRD-reactive-tri-design assms(1) assms(2) assms(3) assms(4) assms(5) srdes-tri-refine-intro*)

lemma *SRD-refine-intro'*:

assumes
 P is SRD Q is SRD
 $'pre_R(P) \Rightarrow pre_R(Q)'$ $peri_R(P) \sqsubseteq (pre_R(P) \wedge peri_R(Q))$ $post_R(P) \sqsubseteq (pre_R(P) \wedge post_R(Q))$
shows $P \sqsubseteq Q$
using *assms* **by** (*rule-tac SRD-refine-intro, simp-all add: refBy-order*)

lemma *SRD-eq-intro*:

assumes
 P is SRD Q is SRD $pre_R(P) = pre_R(Q)$ $peri_R(P) = peri_R(Q)$ $post_R(P) = post_R(Q)$
shows $P = Q$
by (*metis SRD-reactive-tri-design assms*)

4.7 Closure laws

lemma *SRD-srdes-skip [closure]*: II_R is SRD

by (*simp add: srdes-skip-def RHS-design-is-SRD unrest*)

lemma *SRD-seqr-closure* [closure]:
assumes P is SRD Q is SRD
shows $(P ;; Q)$ is SRD
proof –
 have $(P ;; Q) = \mathbf{R}_s(((\neg_r \text{pre}_R P) \text{wp}_r \text{false} \wedge \text{post}_R P \text{wp}_r \text{pre}_R Q) \vdash$
 $((\exists \$st' \cdot \text{peri}_R P) \vee \text{post}_R P ;; \text{peri}_R Q) \diamond \text{post}_R P ;; \text{post}_R Q)$
 by (simp add: SRD-composition-wp assms(1) assms(2))
 also have ... is SRD
 by (rule RHS-design-is-SRD, simp-all add: wp-rea-def unrest)
 finally show ?thesis .
qed

lemma *SRD-power-Suc* [closure]: P is SRD $\implies P^\wedge(\text{Suc } n)$ is SRD
proof (induct n)
 case 0
 then show ?case
 by (simp)
next
 case (Suc n)
 then show ?case
 using SRD-seqr-closure by (simp add: SRD-seqr-closure upred-semiring.power-Suc)
qed

lemma *SRD-power-comp* [closure]: P is SRD $\implies P ;; P^\wedge n$ is SRD
 by (metis SRD-power-Suc upred-semiring.power-Suc)

lemma *uplus-SRD-closed* [closure]: P is SRD $\implies P^+$ is SRD
 by (simp add: uplus-power-def closure)

lemma *SRD-Sup-closure* [closure]:
assumes $A \subseteq \llbracket \text{SRD} \rrbracket_H$ $A \neq \{\}$
shows $(\bigcap A)$ is SRD
proof –
 have $\text{SRD } (\bigcap A) = (\bigcap (\text{SRD } A))$
 by (simp add: ContinuousD SRD-Continuous assms(2))
 also have ... = $(\bigcap A)$
 by (simp only: Healthy-carrier-image assms)
 finally show ?thesis by (simp add: Healthy-def)
qed

4.8 Distribution laws

lemma *RHS-tri-design-choice* [rdes-def]:
 $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) \sqcap \mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3) = \mathbf{R}_s((P_1 \wedge Q_1) \vdash (P_2 \vee Q_2) \diamond (P_3 \vee Q_3))$
 apply (simp add: RHS-design-choice)
 apply (rule cong[of \mathbf{R}_s \mathbf{R}_s])
 apply (simp)
 apply (rel-auto)
 done

lemma *RHS-tri-design-disj* [rdes-def]:
 $(\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) \vee \mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3)) = \mathbf{R}_s((P_1 \wedge Q_1) \vdash (P_2 \vee Q_2) \diamond (P_3 \vee Q_3))$
 by (simp add: RHS-tri-design-choice disj-upred-def)

lemma *RHS-tri-design-sup* [rdes-def]:

$\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) \sqcup \mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3) = \mathbf{R}_s((P_1 \vee Q_1) \vdash ((P_1 \Rightarrow_r P_2) \wedge (Q_1 \Rightarrow_r Q_2)) \diamond ((P_1 \Rightarrow_r P_3) \wedge (Q_1 \Rightarrow_r Q_3)))$
 by (*simp add: RHS-design-sup, rel-auto*)

lemma *RHS-tri-design-conj* [*rdes-def*]:

$(\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) \wedge \mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3)) = \mathbf{R}_s((P_1 \vee Q_1) \vdash ((P_1 \Rightarrow_r P_2) \wedge (Q_1 \Rightarrow_r Q_2)) \diamond ((P_1 \Rightarrow_r P_3) \wedge (Q_1 \Rightarrow_r Q_3)))$
 by (*simp add: RHS-tri-design-sup conj-upred-def*)

lemma *SRD-UINF* [*rdes-def*]:

assumes $A \neq \{\}$ $A \subseteq \llbracket \text{SRD} \rrbracket_H$

shows $\sqcap A = \mathbf{R}_s((\bigwedge P \in A \cdot \text{pre}_R(P)) \vdash (\bigvee P \in A \cdot \text{peri}_R(P)) \diamond (\bigvee P \in A \cdot \text{post}_R(P)))$

proof –

have $\sqcap A = \mathbf{R}_s(\text{pre}_R(\sqcap A) \vdash \text{peri}_R(\sqcap A) \diamond \text{post}_R(\sqcap A))$

by (*metis SRD-as-reactive-tri-design assms srdes-hcond-def*

srdes-theory-continuous.healthy-inf srdes-theory-continuous.healthy-inf-def)

also have $\dots = \mathbf{R}_s((\bigwedge P \in A \cdot \text{pre}_R(P)) \vdash (\bigvee P \in A \cdot \text{peri}_R(P)) \diamond (\bigvee P \in A \cdot \text{post}_R(P)))$

by (*simp add: preR-INF periR-INF postR-INF assms*)

finally show *?thesis* .

qed

lemma *RHS-tri-design-USUP* [*rdes-def*]:

assumes $A \neq \{\}$

shows $(\sqcap i \in A \cdot \mathbf{R}_s(P(i) \vdash Q(i) \diamond R(i))) = \mathbf{R}_s((\bigwedge i \in A \cdot P(i)) \vdash (\sqcap i \in A \cdot Q(i)) \diamond (\sqcap i \in A \cdot R(i)))$

by (*subst RHS-INF[OF assms, THEN sym], simp add: design-UINF-mem assms, rel-auto*)

lemma *SRD-UINF-mem*:

assumes $A \neq \{\}$ $\bigwedge i. P \ i \text{ is } \text{SRD}$

shows $(\sqcap i \in A \cdot P \ i) = \mathbf{R}_s((\bigwedge i \in A \cdot \text{pre}_R(P \ i)) \vdash (\bigvee i \in A \cdot \text{peri}_R(P \ i)) \diamond (\bigvee i \in A \cdot \text{post}_R(P \ i)))$

(*is ?lhs = ?rhs*)

proof –

have $?lhs = (\sqcap (P \ ' A))$

by (*rel-auto*)

also have $\dots = \mathbf{R}_s((\bigwedge Pa \in P \ ' A \cdot \text{pre}_R \ Pa) \vdash (\sqcap Pa \in P \ ' A \cdot \text{peri}_R \ Pa) \diamond (\sqcap Pa \in P \ ' A \cdot \text{post}_R \ Pa))$

by (*subst rdes-def, simp-all add: assms image-subsetI*)

also have $\dots = ?rhs$

by (*rel-auto*)

finally show *?thesis* .

qed

lemma *RHS-tri-design-UINF-ind* [*rdes-def*]:

$(\sqcap i \cdot \mathbf{R}_s(P_1(i) \vdash P_2(i) \diamond P_3(i))) = \mathbf{R}_s((\bigwedge i \cdot P_1 \ i) \vdash (\bigvee i \cdot P_2(i)) \diamond (\bigvee i \cdot P_3(i)))$

by (*rel-auto*)

lemma *cond-srea-form* [*rdes-def*]:

$\mathbf{R}_s(P \vdash Q_1 \diamond Q_2) \triangleleft b \triangleright_R \mathbf{R}_s(R \vdash S_1 \diamond S_2) =$

$\mathbf{R}_s((P \triangleleft b \triangleright_R R) \vdash (Q_1 \triangleleft b \triangleright_R S_1) \diamond (Q_2 \triangleleft b \triangleright_R S_2))$

proof –

have $\mathbf{R}_s(P \vdash Q_1 \diamond Q_2) \triangleleft b \triangleright_R \mathbf{R}_s(R \vdash S_1 \diamond S_2) = \mathbf{R}_s(P \vdash Q_1 \diamond Q_2) \triangleleft R2c(\lceil b \rceil_{S<}) \triangleright \mathbf{R}_s(R \vdash S_1 \diamond S_2)$

by (*pred-auto*)

also have $\dots = \mathbf{R}_s(P \vdash Q_1 \diamond Q_2 \triangleleft b \triangleright_R R \vdash S_1 \diamond S_2)$

by (*simp add: RHS-cond lift-cond-srea-def*)

also have ... = $\mathbf{R}_s((P \triangleleft b \triangleright_R R) \vdash (Q_1 \diamond Q_2 \triangleleft b \triangleright_R S_1 \diamond S_2))$
 by (simp add: design-condr lift-cond-srea-def)
 also have ... = $\mathbf{R}_s((P \triangleleft b \triangleright_R R) \vdash (Q_1 \triangleleft b \triangleright_R S_1) \diamond (Q_2 \triangleleft b \triangleright_R S_2))$
 by (rule cong[of $\mathbf{R}_s \mathbf{R}_s$], simp, rel-auto)
 finally show ?thesis .
 qed

lemma *SRD-cond-srea [closure]*:

assumes *P is SRD Q is SRD*

shows $P \triangleleft b \triangleright_R Q$ is SRD

proof –

have $P \triangleleft b \triangleright_R Q = \mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \diamond \text{post}_R(P)) \triangleleft b \triangleright_R \mathbf{R}_s(\text{pre}_R(Q) \vdash \text{peri}_R(Q) \diamond \text{post}_R(Q))$

by (simp add: SRD-reactive-tri-design assms)

also have ... = $\mathbf{R}_s((\text{pre}_R P \triangleleft b \triangleright_R \text{pre}_R Q) \vdash (\text{peri}_R P \triangleleft b \triangleright_R \text{peri}_R Q) \diamond (\text{post}_R P \triangleleft b \triangleright_R \text{post}_R Q))$

by (simp add: cond-srea-form)

also have ... is SRD

by (simp add: RHS-tri-design-is-SRD lift-cond-srea-def unrest)

finally show ?thesis .

qed

4.9 Algebraic laws

lemma *SRD-left-unit*:

assumes *P is SRD*

shows $\Pi_R ;; P = P$

by (simp add: SRD-composition-wp closure rdes wp C1 R1-negate-R1 R1-false
 rpred trace-ident-left-periR trace-ident-left-postR SRD-reactive-tri-design assms)

lemma *skip-srea-self-unit [simp]*:

$\Pi_R ;; \Pi_R = \Pi_R$

by (simp add: SRD-left-unit closure)

lemma *SRD-right-unit-tri-lemma*:

assumes *P is SRD*

shows $P ;; \Pi_R = \mathbf{R}_s((\neg_r \text{pre}_R P) \text{wp}_r \text{false} \vdash (\exists \$st' \cdot \text{peri}_R P) \diamond \text{post}_R P)$

by (simp add: SRD-composition-wp closure rdes wp rpred trace-ident-right-postR assms)

lemma *Miracle-left-zero*:

assumes *P is SRD*

shows *Miracle* ;; $P = \text{Miracle}$

proof –

have $\text{Miracle} ;; P = \mathbf{R}_s(\text{true} \vdash \text{false}) ;; \mathbf{R}_s(\text{pre}_R(P) \vdash \text{cmt}_R(P))$

by (simp add: Miracle-def SRD-reactive-design-alt assms)

also have ... = $\mathbf{R}_s(\text{true} \vdash \text{false})$

by (simp add: RHS-design-composition unrest R1-false R2s-false R2s-true)

also have ... = *Miracle*

by (simp add: Miracle-def)

finally show ?thesis .

qed

lemma *Chaos-left-zero*:

assumes *P is SRD*

shows (*Chaos* ;; P) = *Chaos*

proof –

have $\text{Chaos} ;; P = \mathbf{R}_s(\text{false} \vdash \text{true}) ;; \mathbf{R}_s(\text{pre}_R(P) \vdash \text{cmt}_R(P))$

by (simp add: Chaos-def SRD-reactive-design-alt assms)
 also have ... = $\mathbf{R}_s ((\neg R1 \text{ true} \wedge \neg (R1 \text{ true} \wedge \neg \$wait')) ;; R1 (\neg R2s (pre_R P))) \vdash$
 $R1 \text{ true} ;; ((\exists \$st \cdot [II]_D) \triangleleft \$wait \triangleright R1 (R2s (cmt_R P)))$
 by (simp add: RHS-design-composition unrest R2s-false R2s-true R1-false)
 also have ... = $\mathbf{R}_s ((false \wedge \neg (R1 \text{ true} \wedge \neg \$wait')) ;; R1 (\neg R2s (pre_R P))) \vdash$
 $R1 \text{ true} ;; ((\exists \$st \cdot [II]_D) \triangleleft \$wait \triangleright R1 (R2s (cmt_R P)))$
 by (simp add: RHS-design-conj-neg-R1-pre)
 also have ... = $\mathbf{R}_s(true)$
 by (simp add: design-false-pre)
 also have ... = $\mathbf{R}_s(false \vdash true)$
 by (simp add: design-def)
 also have ... = Chaos
 by (simp add: Chaos-def)
 finally show ?thesis .
 qed

lemma *SRD-right-Chaos-tri-lemma*:

assumes *P is SRD*
 shows $P ;; Chaos = \mathbf{R}_s (((\neg_r pre_R P) wp_r false \wedge post_R P wp_r false) \vdash (\exists \$st' \cdot peri_R P) \diamond false)$
 by (simp add: SRD-composition-wp closure rdes assms wp, rel-auto)

lemma *SRD-right-Miracle-tri-lemma*:

assumes *P is SRD*
 shows $P ;; Miracle = \mathbf{R}_s (((\neg_r pre_R P) wp_r false \vdash (\exists \$st' \cdot peri_R P) \diamond false)$
 by (simp add: SRD-composition-wp closure rdes assms wp, rel-auto)

Stateful reactive designs are left unital

overloading

$srdes\text{-}unit == utp\text{-}unit :: (SRDES, ('s, 't :: trace, 'a) rsp) \text{ uthy} \Rightarrow ('s, 't, 'a) \text{ hrel-rsp}$

begin

definition $srdes\text{-}unit :: (SRDES, ('s, 't :: trace, 'a) rsp) \text{ uthy} \Rightarrow ('s, 't, 'a) \text{ hrel-rsp}$ **where**
 $srdes\text{-}unit T = II_R$

end

interpretation *srdes-left-unital*: *utp-theory-left-unital SRDES*

by (unfold-locales, simp-all add: srdes-hcond-def srdes-unit-def SRD-seqr-closure SRD-srdes-skip SRD-left-unit)

4.10 Recursion laws

lemma *mono-srd-iter*:

assumes $mono F F \in \llbracket SRD \rrbracket_H \rightarrow \llbracket SRD \rrbracket_H$
 shows $mono (\lambda X. \mathbf{R}_s(pre_R(F X) \vdash peri_R(F X) \diamond post_R(F X)))$
 apply (rule monoI)
 apply (rule srdes-tri-refine-intro')
 apply (meson assms(1) monoE preR-antitone utp-pred-laws.le-infI2)
 apply (meson assms(1) monoE periR-monotone utp-pred-laws.le-infI2)
 apply (meson assms(1) monoE postR-monotone utp-pred-laws.le-infI2)
 done

lemma *mu-srd-SRD*:

assumes $mono F F \in \llbracket SRD \rrbracket_H \rightarrow \llbracket SRD \rrbracket_H$
 shows $(\mu X \cdot \mathbf{R}_s(pre_R(F X) \vdash peri_R(F X) \diamond post_R(F X))) \text{ is SRD}$
 apply (subst gfp-unfold)
 apply (simp add: mono-srd-iter assms)
 apply (rule RHS-tri-design-is-SRD)
 apply (simp-all add: unrest)

done

lemma *mu-srd-iter*:

assumes *mono* $F \in \llbracket \text{SRD} \rrbracket_H \rightarrow \llbracket \text{SRD} \rrbracket_H$
shows $(\mu \ X \cdot \mathbf{R}_s(\text{pre}_R(F(X)) \vdash \text{peri}_R(F(X)) \diamond \text{post}_R(F(X)))) = F(\mu \ X \cdot \mathbf{R}_s(\text{pre}_R(F(X)) \vdash \text{peri}_R(F(X)) \diamond \text{post}_R(F(X))))$
apply (*subst gfp-unfold*)
apply (*simp add: mono-srd-iter assms*)
apply (*subst SRD-as-reactive-tri-design[THEN sym]*)
using *Healthy-func assms(1) assms(2) mu-srd-SRD* **apply** *blast*
done

lemma *mu-srd-form*:

assumes *mono* $F \in \llbracket \text{SRD} \rrbracket_H \rightarrow \llbracket \text{SRD} \rrbracket_H$
shows $\mu_R F = (\mu \ X \cdot \mathbf{R}_s(\text{pre}_R(F(X)) \vdash \text{peri}_R(F(X)) \diamond \text{post}_R(F(X))))$
proof –
have 1: $F(\mu \ X \cdot \mathbf{R}_s(\text{pre}_R(F(X)) \vdash \text{peri}_R(F(X)) \diamond \text{post}_R(F(X))))$ is *SRD*
by (*simp add: Healthy-apply-closed assms(1) assms(2) mu-srd-SRD*)
have 2: *Mono_{uthy-order} SRDES* F
by (*simp add: assms(1) mono-Monotone-utp-order*)
hence 3: $\mu_R F = F(\mu_R F)$
by (*simp add: srdes-theory-continuous.LFP-unfold[THEN sym] assms*)
hence $\mathbf{R}_s(\text{pre}_R(F(F(\mu_R F))) \vdash \text{peri}_R(F(F(\mu_R F))) \diamond \text{post}_R(F(F(\mu_R F)))) = \mu_R F$
using *SRD-reactive-tri-design* **by** *force*
hence $(\mu \ X \cdot \mathbf{R}_s(\text{pre}_R(F(X)) \vdash \text{peri}_R(F(X)) \diamond \text{post}_R(F(X)))) \sqsubseteq F(\mu_R F)$
by (*simp add: 2 srdes-theory-continuous.weak.LFP-lemma3 gfp-upperbound assms*)
thus *?thesis*
using *assms 1 3 srdes-theory-continuous.weak.LFP-lowerbound eq-iff mu-srd-iter*
by (*metis (mono-tags, lifting)*)
qed

lemma *Monotonic-SRD-comp [closure]*: *Monotonic* (*op* ;; $P \circ \text{SRD}$)

by (*simp add: mono-def R1-R2c-is-R2 R2-mono R3h-mono RD1-mono RD2-mono RHS-def SRD-def seqr-mono*)

end

5 Normal Reactive Designs

theory *utp-rdes-normal*

imports

utp-rdes-triples

UTP-KAT.utp-kleene

begin

This additional healthiness condition is analogous to H3

definition *RD3* **where**

[upred-defs]: $\text{RD3}(P) = P \text{ ;; } \text{II}_R$

lemma *RD3-idem*: $\text{RD3}(\text{RD3}(P)) = \text{RD3}(P)$

proof –

have *a*: $\text{II}_R \text{ ;; } \text{II}_R = \text{II}_R$

by (*simp add: SRD-left-unit SRD-srdes-skip*)

show *?thesis*

by (*simp add: RD3-def seqr-assoc a*)

qed

lemma *RD3-Idempotent [closure]: Idempotent RD3*
 by (simp add: Idempotent-def RD3-idem)

lemma *RD3-continuous: RD3($\prod A$) = ($\prod P \in A. RD3(P)$)*
 by (simp add: RD3-def seq-Sup-distr)

lemma *RD3-Continuous [closure]: Continuous RD3*
 by (simp add: Continuous-def RD3-continuous)

lemma *RD3-right-subsumes-RD2: RD2(RD3(P)) = RD3(P)*

proof –

have $a:II_R \;; J = II_R$
 by (rel-auto)
 show ?thesis
 by (metis (no-types, hide-lams) H2-def RD2-def RD3-def a seqr-assoc)

qed

lemma *RD3-left-subsumes-RD2: RD3(RD2(P)) = RD3(P)*

proof –

have $a:J \;; II_R = II_R$
 by (rel-simp, safe, blast+)
 show ?thesis
 by (metis (no-types, hide-lams) H2-def RD2-def RD3-def a seqr-assoc)

qed

lemma *RD3-implies-RD2: P is RD3 \implies P is RD2*
 by (metis Healthy-def RD3-right-subsumes-RD2)

lemma *RD3-intro-pre:*

assumes P is SRD $(\neg_r pre_R(P)) \;; true_r = (\neg_r pre_R(P)) \$st' \# peri_R(P)$
 shows P is RD3

proof –

have $RD3(P) = \mathbf{R}_s ((\neg_r pre_R P) wp_r false \vdash (\exists \$st' \cdot peri_R P) \diamond post_R P)$
 by (simp add: RD3-def SRD-right-unit-tri-lemma assms)
 also have $\dots = \mathbf{R}_s ((\neg_r pre_R P) wp_r false \vdash peri_R P \diamond post_R P)$
 by (simp add: assms(3) ex-unrest)
 also have $\dots = \mathbf{R}_s ((\neg_r pre_R P) wp_r false \vdash cmt_R P)$
 by (simp add: wait'-cond-peri-post-cmt)
 also have $\dots = \mathbf{R}_s (pre_R P \vdash cmt_R P)$
 by (simp add: assms(2) rpred wp-rea-def R1-preR)
 finally show ?thesis
 by (metis Healthy-def SRD-as-reactive-design assms(1))

qed

lemma *RHS-tri-design-right-unit-lemma:*

assumes $\$ok' \# P \$ok' \# Q \$ok' \# R \$wait' \# R$
 shows $\mathbf{R}_s(P \vdash Q \diamond R) \;; II_R = \mathbf{R}_s((\neg_r (\neg_r P) \;; true_r) \vdash ((\exists \$st' \cdot Q) \diamond R))$

proof –

have $\mathbf{R}_s(P \vdash Q \diamond R) \;; II_R = \mathbf{R}_s(P \vdash Q \diamond R) \;; \mathbf{R}_s(true \vdash false \diamond (\$tr' =_u \$tr \wedge [II]_R))$
 by (simp add: srdes-skip-tri-design, rel-auto)
 also have $\dots = \mathbf{R}_s ((\neg R1 (\neg R2s P) \;; R1 true) \vdash (\exists \$st' \cdot Q) \diamond (R1 (R2s R) \;; R1 (R2s (\$tr' =_u \$tr \wedge [II]_R))))$
 by (simp-all add: RHS-tri-design-composition assms unrest R2s-true R1-false R2s-false)

also have ... = $\mathbf{R}_s((\neg R1 (\neg R2s P) ;; R1 \text{ true}) \vdash (\exists \$st' \cdot Q) \diamond R1 (R2s R))$
proof –
from *assms*(3,4) **have** ($R1 (R2s R) ;; R1 (R2s (\$tr' =_u \$tr \wedge [II]_R))$) = $R1 (R2s R)$
by (*rel-auto*, *metis* (*no-types*, *lifting*) *minus-zero-eq*, *meson* *order-refl* *trace-class.diff-cancel*)
thus *?thesis*
by *simp*
qed
also have ... = $\mathbf{R}_s((\neg (\neg P) ;; R1 \text{ true}) \vdash ((\exists \$st' \cdot Q) \diamond R))$
by (*metis* (*no-types*, *lifting*) *R1-R2s-R1-true-lemma* *R1-R2s-R2c* *R2c-not RHS-design-R2c-pre* *RHS-design-neg-R1-pre* *RHS-design-post-R1* *RHS-design-post-R2s*)
also have ... = $\mathbf{R}_s((\neg_r (\neg_r P) ;; \text{true}_r) \vdash ((\exists \$st' \cdot Q) \diamond R))$
by (*rel-auto*)
finally show *?thesis* .
qed

lemma *RHS-tri-design-RD3-intro*:

assumes
 $\$ok' \# P \ \$ok' \# Q \ \$ok' \# R \ \$st' \# Q \ \$wait' \# R$
 $P \text{ is } R1 (\neg_r P) ;; \text{true}_r = (\neg_r P)$
shows $\mathbf{R}_s(P \vdash Q \diamond R)$ *is* *RD3*
apply (*simp* *add: Healthy-def RD3-def*)
apply (*subst* *RHS-tri-design-right-unit-lemma*)
apply (*simp-all* *add:assms ex-unrest rpred*)
done

RD3 reactive designs are those whose assumption can be written as a conjunction of a precondition on (undashed) program variables, and a negated statement about the trace. The latter allows us to state that certain events must not occur in the trace – which are effectively safety properties.

lemma *R1-right-unit-lemma*:

$\llbracket \text{out}\alpha \# b; \text{out}\alpha \# e \rrbracket \implies (\neg_r b \vee \$tr \hat{^}_u e \leq_u \$tr') ;; R1(\text{true}) = (\neg_r b \vee \$tr \hat{^}_u e \leq_u \$tr')$
by (*rel-auto*, *blast*, *metis* (*no-types*, *lifting*) *dual-order.trans*)

lemma *RHS-tri-design-RD3-intro-form*:

assumes
 $\text{out}\alpha \# b \ \text{out}\alpha \# e \ \$ok' \# Q \ \$st' \# Q \ \$ok' \# R \ \$wait' \# R$
shows $\mathbf{R}_s((b \wedge \neg_r \$tr \hat{^}_u e \leq_u \$tr') \vdash Q \diamond R)$ *is* *RD3*
apply (*rule* *RHS-tri-design-RD3-intro*)
apply (*simp-all* *add: assms unrest closure rpred*)
apply (*subst* *R1-right-unit-lemma*)
apply (*simp-all* *add: assms unrest*)
done

definition *NSRD* :: (*'s, 't::trace, 'α*) *hrel-rsp* \Rightarrow (*'s, 't, 'α*) *hrel-rsp*
where [*upred-defs*]: *NSRD* = *RD1* \circ *RD3* \circ *RHS*

lemma *RD1-RD3-commute*: $RD1(RD3(P)) = RD3(RD1(P))$
by (*rel-auto*, *blast*+)

lemma *NSRD-is-SRD [closure]*: $P \text{ is } NSRD \implies P \text{ is } SRD$

by (*simp* *add: Healthy-def NSRD-def SRD-def*, *metis* *Healthy-def* *RD1-RD3-commute* *RD2-RHS-commute* *RD3-def* *RD3-right-subsumes-RD2* *SRD-def* *SRD-idem* *SRD-seqr-closure* *SRD-srdes-skip*)

lemma *NSRD-elim [RD-elim]*:

$\llbracket P \text{ is } NSRD; Q(\mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \diamond \text{post}_R(P))) \rrbracket \implies Q(P)$

by (simp add: RD-elim closure)

lemma NSRD-Idempotent [closure]: Idempotent NSRD

by (clarsimp simp add: Idempotent-def NSRD-def, metis (no-types, hide-lams) Healthy-def RD1-RD3-commute RD3-def RD3-idem RD3-left-subsumes-RD2 SRD-def SRD-idem SRD-seqr-closure SRD-srdes-skip)

lemma NSRD-Continuous [closure]: Continuous NSRD

by (simp add: Continuous-comp NSRD-def RD1-Continuous RD3-Continuous RHS-Continuous)

lemma NSRD-form:

$NSRD(P) = \mathbf{R}_s((\neg_r (\neg_r pre_R(P)) ;; R1 true) \vdash ((\exists \$st' \cdot peri_R(P)) \diamond post_R(P)))$

proof –

have $NSRD(P) = RD3(SRD(P))$

by (metis (no-types, lifting) NSRD-def RD1-RD3-commute RD3-left-subsumes-RD2 SRD-def comp-def)

also have $\dots = RD3(\mathbf{R}_s(pre_R(P) \vdash peri_R(P) \diamond post_R(P)))$

by (simp add: SRD-as-reactive-tri-design)

also have $\dots = \mathbf{R}_s(pre_R(P) \vdash peri_R(P) \diamond post_R(P)) ;; II_R$

by (simp add: RD3-def)

also have $\dots = \mathbf{R}_s((\neg_r (\neg_r pre_R(P)) ;; R1 true) \vdash ((\exists \$st' \cdot peri_R(P)) \diamond post_R(P)))$

by (simp add: RHS-tri-design-right-unit-lemma unrest)

finally show ?thesis .

qed

lemma NSRD-healthy-form:

assumes P is NSRD

shows $\mathbf{R}_s((\neg_r (\neg_r pre_R(P)) ;; R1 true) \vdash ((\exists \$st' \cdot peri_R(P)) \diamond post_R(P))) = P$

by (metis Healthy-def NSRD-form assms)

lemma NSRD-Sup-closure [closure]:

assumes $A \subseteq \llbracket NSRD \rrbracket_H$ $A \neq \{\}$

shows $\sqcap A$ is NSRD

proof –

have $NSRD (\sqcap A) = (\sqcap (NSRD 'A))$

by (simp add: ContinuousD NSRD-Continuous assms(2))

also have $\dots = (\sqcap A)$

by (simp only: Healthy-carrier-image assms)

finally show ?thesis by (simp add: Healthy-def)

qed

lemma intChoice-NSRD-closed [closure]:

assumes P is NSRD Q is NSRD

shows $P \sqcap Q$ is NSRD

using NSRD-Sup-closure[of $\{P, Q\}$] by (simp add: assms)

lemma NRSD-SUP-closure [closure]:

$\llbracket \bigwedge i. i \in A \implies P(i) \text{ is NSRD}; A \neq \{\} \rrbracket \implies (\sqcap i \in A. P(i)) \text{ is NSRD}$

by (rule NSRD-Sup-closure, auto)

lemma NSRD-neg-pre-unit:

assumes P is NSRD

shows $(\neg_r pre_R(P)) ;; true_r = (\neg_r pre_R(P))$

proof –

have $(\neg_r pre_R(P)) = (\neg_r pre_R(\mathbf{R}_s((\neg_r (\neg_r pre_R(P)) ;; R1 true) \vdash ((\exists \$st' \cdot peri_R(P)) \diamond post_R(P))))$

by (simp add: NSRD-healthy-form assms)

also have $\dots = R1 (R2c ((\neg_r pre_R P) ;; R1 true))$

by (simp add: rea-pre-RHS-design R1-negate-R1 R1-idem R1-rea-not' R2c-rea-not usubst rpred unrest closure)
 also have ... = $(\neg_r \text{pre}_R P) ;; R1 \text{ true}$
 by (simp add: R1-R2c-seqr-distribute closure assms)
 finally show ?thesis
 by (simp add: rea-not-def)
 qed

lemma NSRD-neg-pre-left-zero:
 assumes P is NSRD Q is R1 Q is RD1
 shows $(\neg_r \text{pre}_R(P)) ;; Q = (\neg_r \text{pre}_R(P))$
 by (metis (no-types, hide-lams) NSRD-neg-pre-unit RD1-left-zero assms(1) assms(2) assms(3) seqr-assoc)

lemma NSRD-st'-unrest-peri [unrest]:
 assumes P is NSRD
 shows $\$st' \# \text{peri}_R(P)$
proof –
 have $\text{peri}_R(P) = \text{peri}_R(\mathbf{R}_s((\neg_r (\neg_r \text{pre}_R(P)) ;; R1 \text{ true}) \vdash ((\exists \$st' \cdot \text{peri}_R(P)) \diamond \text{post}_R(P))))$
 by (simp add: NSRD-healthy-form assms)
 also have ... = $R1 (R2c (\neg_r (\neg_r \text{pre}_R P) ;; R1 \text{ true}) \Rightarrow_r (\exists \$st' \cdot \text{peri}_R P))$
 by (simp add: rea-peri-RHS-design usubst unrest)
 also have $\$st' \# \dots$
 by (simp add: R1-def R2c-def unrest)
 finally show ?thesis .
 qed

lemma NSRD-wait'-unrest-pre [unrest]:
 assumes P is NSRD
 shows $\$wait' \# \text{pre}_R(P)$
proof –
 have $\text{pre}_R(P) = \text{pre}_R(\mathbf{R}_s((\neg_r (\neg_r \text{pre}_R(P)) ;; R1 \text{ true}) \vdash ((\exists \$st' \cdot \text{peri}_R(P)) \diamond \text{post}_R(P))))$
 by (simp add: NSRD-healthy-form assms)
 also have ... = $(R1 (R2c (\neg_r (\neg_r \text{pre}_R P) ;; R1 \text{ true})))$
 by (simp add: rea-pre-RHS-design usubst unrest)
 also have $\$wait' \# \dots$
 by (simp add: R1-def R2c-def unrest)
 finally show ?thesis .
 qed

lemma NSRD-st'-unrest-pre [unrest]:
 assumes P is NSRD
 shows $\$st' \# \text{pre}_R(P)$
proof –
 have $\text{pre}_R(P) = \text{pre}_R(\mathbf{R}_s((\neg_r (\neg_r \text{pre}_R(P)) ;; R1 \text{ true}) \vdash ((\exists \$st' \cdot \text{peri}_R(P)) \diamond \text{post}_R(P))))$
 by (simp add: NSRD-healthy-form assms)
 also have ... = $R1 (R2c (\neg_r (\neg_r \text{pre}_R P) ;; R1 \text{ true}))$
 by (simp add: rea-pre-RHS-design usubst unrest)
 also have $\$st' \# \dots$
 by (simp add: R1-def R2c-def unrest)
 finally show ?thesis .
 qed

lemma NSRD-alt-def: $\text{NSRD}(P) = \text{RD3}(\text{SRD}(P))$
 by (metis NSRD-def RD1-RD3-commute RD3-left-subsumes-RD2 SRD-def comp-eq-dest-lhs)

lemma *preR-RR [closure]*: P is NSRD \implies $\text{pre}_R(P)$ is RR
 by (rule RR-intro, simp-all add: closure unrest)

lemma *NSRD-neg-pre-RC [closure]*:
 assumes P is NSRD
 shows $\text{pre}_R(P)$ is RC
 by (rule RC-intro, simp-all add: closure assms NSRD-neg-pre-unit rpred)

lemma *NSRD-intro*:
 assumes P is SRD $(\neg_r \text{pre}_R(P)) \;; \text{true}_r = (\neg_r \text{pre}_R(P)) \ \$st' \ \# \ \text{peri}_R(P)$
 shows P is NSRD

proof –

have $\text{NSRD}(P) = \mathbf{R}_s((\neg_r (\neg_r \text{pre}_R(P)) \;; R1 \text{ true}) \vdash ((\exists \ \$st' \cdot \text{peri}_R(P)) \diamond \text{post}_R(P)))$
 by (simp add: NSRD-form)
 also have $\dots = \mathbf{R}_s(\text{pre}_R P \vdash \text{peri}_R P \diamond \text{post}_R P)$
 by (simp add: assms ex-unrest rpred closure)
 also have $\dots = P$
 by (simp add: SRD-reactive-tri-design assms(1))
 finally show ?thesis
 using Healthy-def by blast

qed

lemma *NSRD-intro'*:
 assumes P is R2 P is R3h P is RD1 P is RD3
 shows P is NSRD
 by (metis (no-types, hide-lams) Healthy-def NSRD-def R1-R2c-is-R2 RHS-def assms comp-apply)

lemma *NSRD-RC-intro*:
 assumes P is SRD $\text{pre}_R(P)$ is RC $\$st' \ \# \ \text{peri}_R(P)$
 shows P is NSRD
 by (metis Healthy-def NSRD-form SRD-reactive-tri-design assms(1) assms(2) assms(3)
 ex-unrest rea-not-false wp-rea-RC-false wp-rea-def)

lemma *NSRD-rdes-intro [closure]*:
 assumes P is RC Q is RR R is RR $\$st' \ \# \ Q$
 shows $\mathbf{R}_s(P \vdash Q \diamond R)$ is NSRD
 by (rule NSRD-RC-intro, simp-all add: rdes closure assms unrest)

lemma *SRD-RD3-implies-NSRD*:
 $\llbracket P \text{ is SRD}; P \text{ is RD3} \rrbracket \implies P \text{ is NSRD}$
 by (metis (no-types, lifting) Healthy-def NSRD-def RHS-idem SRD-healths(4) SRD-reactive-design comp-apply)

lemma *NSRD-iff*:
 $P \text{ is NSRD} \iff ((P \text{ is SRD}) \wedge (\neg_r \text{pre}_R(P)) \;; R1(\text{true}) = (\neg_r \text{pre}_R(P)) \wedge (\$st' \ \# \ \text{peri}_R(P)))$
 by (meson NSRD-intro NSRD-is-SRD NSRD-neg-pre-unit NSRD-st'-unrest-peri)

lemma *NSRD-is-RD3 [closure]*:
 assumes P is NSRD
 shows P is RD3
 by (simp add: NSRD-is-SRD NSRD-neg-pre-unit NSRD-st'-unrest-peri RD3-intro-pre assms)

lemma *NSRD-refine-elim*:
 assumes

$P \sqsubseteq Q$ *P is NSRD Q is NSRD*
 $\llbracket 'pre_R(P) \Rightarrow pre_R(Q)'; 'pre_R(P) \wedge peri_R(Q) \Rightarrow peri_R(P)'; 'pre_R(P) \wedge post_R(Q) \Rightarrow post_R(P)' \rrbracket$
 $\Rightarrow R$
shows R
proof –
have $\mathbf{R}_s(pre_R(P) \vdash peri_R(P) \diamond post_R(P)) \sqsubseteq \mathbf{R}_s(pre_R(Q) \vdash peri_R(Q) \diamond post_R(Q))$
by (*simp add: NSRD-is-SRD SRD-reactive-tri-design assms(1) assms(2) assms(3)*)
hence $1: 'pre_R P \Rightarrow pre_R Q'$ **and** $2: 'pre_R P \wedge peri_R Q \Rightarrow peri_R P'$ **and** $3: 'pre_R P \wedge post_R Q \Rightarrow post_R P'$
by (*simp-all add: RHS-tri-design-refine assms closure*)
with *assms(4)* **show** *?thesis*
by *simp*
qed

lemma *NSRD-right-unit: P is NSRD $\Rightarrow P ;; II_R = P$*
by (*metis Healthy-if NSRD-is-RD3 RD3-def*)

lemma *NSRD-composition-wp:*
assumes *P is NSRD Q is SRD*
shows $P ;; Q =$
 $\mathbf{R}_s((pre_R P \wedge post_R P wp_r pre_R Q) \vdash (peri_R P \vee (post_R P ;; peri_R Q)) \diamond (post_R P ;; post_R Q))$
by (*simp add: SRD-composition-wp assms NSRD-is-SRD wp-rea-def NSRD-neg-pre-unit NSRD-st'-unrest-peri R1-negate-R1 R1-preR ex-unrest rpred*)

lemma *preR-NSRD-seq-lemma:*
assumes *P is NSRD Q is SRD*
shows $R1 (R2c (post_R P ;; (\neg_r pre_R Q))) = post_R P ;; (\neg_r pre_R Q)$
proof –
have $post_R P ;; (\neg_r pre_R Q) = R1(R2c(post_R P)) ;; R1(R2c(\neg_r pre_R Q))$
by (*simp add: NSRD-is-SRD R1-R2c-post-RHS R1-rea-not R2c-preR R2c-rea-not assms(1) assms(2)*)
also have $\dots = R1 (R2c (post_R P ;; (\neg_r pre_R Q)))$
by (*simp add: R1-seqr R2c-R1-seq calculation*)
finally show *?thesis ..*
qed

lemma *preR-NSRD-seq [rdes]:*
assumes *P is NSRD Q is SRD*
shows $pre_R(P ;; Q) = (pre_R P \wedge post_R P wp_r pre_R Q)$
by (*simp add: NSRD-composition-wp assms rea-pre-RHS-design usubst unrest wp-rea-def R2c-disj R1-disj R2c-and R2c-preR R1-R2c-commute[THEN sym] R1-extend-conj' R1-idem R2c-not closure*)
(metis (no-types, lifting) Healthy-def Healthy-if NSRD-is-SRD R1-R2c-commute R1-R2c-seqr-distribute R1-seqr-closure assms(1) assms(2) postR-R2c-closed postR-SRD-R1 preR-R2c-closed rea-not-R1 rea-not-R2c)

lemma *periR-NSRD-seq [rdes]:*
assumes *P is NSRD Q is NSRD*
shows $peri_R(P ;; Q) = ((pre_R P \wedge post_R P wp_r pre_R Q) \Rightarrow_r (peri_R P \vee (post_R P ;; peri_R Q)))$
by (*simp add: NSRD-composition-wp assms closure rea-peri-RHS-design usubst unrest wp-rea-def R1-extend-conj' R1-disj R1-R2c-seqr-distribute R2c-disj R2c-and R2c-rea-impl R1-rea-impl' R2c-preR R2c-periR R1-rea-not' R2c-rea-not R1-peri-SRD*)

lemma *postR-NSRD-seq [rdes]:*
assumes *P is NSRD Q is NSRD*
shows $post_R(P ;; Q) = ((pre_R P \wedge post_R P wp_r pre_R Q) \Rightarrow_r (post_R P ;; post_R Q))$

by (simp add: NSRD-composition-wp assms closure rea-post-RHS-design usubst unrest wp-rea-def
 R1-extend-conj' R1-disj R1-R2c-seqr-distribute R2c-disj R2c-and R2c-rea-impl R1-rea-impl'
 R2c-preR R2c-periR R1-rea-not' R2c-rea-not)

lemma NSRD-seqr-closure [closure]:

assumes P is NSRD Q is NSRD

shows $(P ;; Q)$ is NSRD

proof –

have $(\neg_r \text{post}_R P \text{wp}_r \text{pre}_R Q) ;; \text{true}_r = (\neg_r \text{post}_R P \text{wp}_r \text{pre}_R Q)$

by (simp add: wp-rea-def rpred assms closure seqr-assoc NSRD-neg-pre-unit)

moreover have $\$st' \# \text{pre}_R P \wedge \text{post}_R P \text{wp}_r \text{pre}_R Q \Rightarrow_r \text{peri}_R P \vee \text{post}_R P ;; \text{peri}_R Q$

by (simp add: unrest assms wp-rea-def)

ultimately show ?thesis

by (rule-tac NSRD-intro, simp-all add: seqr-or-distl NSRD-neg-pre-unit assms closure rdes unrest)

qed

lemma RHS-tri-normal-design-composition:

assumes

$\$ok' \# P \$ok' \# Q_1 \$ok' \# Q_2 \$ok \# R \$ok \# S_1 \$ok \# S_2$

$\$wait \# R \$wait' \# Q_2 \$wait \# S_1 \$wait \# S_2$

P is R2c Q_1 is R1 Q_1 is R2c Q_2 is R1 Q_2 is R2c

R is R2c S_1 is R1 S_1 is R2c S_2 is R1 S_2 is R2c

$R1(\neg P) ;; R1(\text{true}) = R1(\neg P) \$st' \# Q_1$

shows $\mathbf{R}_s(P \vdash Q_1 \diamond Q_2) ;; \mathbf{R}_s(R \vdash S_1 \diamond S_2)$

$= \mathbf{R}_s((P \wedge Q_2 \text{wp}_r R) \vdash (Q_1 \vee (Q_2 ;; S_1)) \diamond (Q_2 ;; S_2))$

proof –

have $\mathbf{R}_s(P \vdash Q_1 \diamond Q_2) ;; \mathbf{R}_s(R \vdash S_1 \diamond S_2) =$

$\mathbf{R}_s((R1(\neg P) \text{wp}_r \text{false} \wedge Q_2 \text{wp}_r R) \vdash ((\exists \$st' \cdot Q_1) \sqcap (Q_2 ;; S_1)) \diamond (Q_2 ;; S_2))$

by (simp-all add: RHS-tri-design-composition-wp rea-not-def assms unrest)

also have $\dots = \mathbf{R}_s((P \wedge Q_2 \text{wp}_r R) \vdash (Q_1 \vee (Q_2 ;; S_1)) \diamond (Q_2 ;; S_2))$

by (simp add: assms wp-rea-def ex-unrest, rel-auto)

finally show ?thesis .

qed

lemma RHS-tri-normal-design-composition' [rdes-def]:

assumes P is RC Q_1 is RR $\$st' \# Q_1$ Q_2 is RR R is RR S_1 is RR S_2 is RR

shows $\mathbf{R}_s(P \vdash Q_1 \diamond Q_2) ;; \mathbf{R}_s(R \vdash S_1 \diamond S_2)$

$= \mathbf{R}_s((P \wedge Q_2 \text{wp}_r R) \vdash (Q_1 \vee (Q_2 ;; S_1)) \diamond (Q_2 ;; S_2))$

proof –

have $R1(\neg P) ;; R1 \text{true} = R1(\neg P)$

using RC-implies-RC1[OF assms(1)]

by (simp add: Healthy-def RC1-def rea-not-def)

(metis R1-negate-R1 R1-seqr utp-pred-laws.double-compl)

thus ?thesis

by (simp add: RHS-tri-normal-design-composition assms closure unrest RR-implies-R2c)

qed

If a normal reactive design has postcondition false, then it is a left zero for sequential composition.

lemma NSRD-seq-post-false:

assumes P is NSRD Q is SRD $\text{post}_R(P) = \text{false}$

shows $P ;; Q = P$

apply (simp add: NSRD-composition-wp assms wp rpred closure)

using NSRD-is-SRD SRD-reactive-tri-design assms(1,3) apply fastforce

done

lemma *NSRD-srd-skip* [closure]: *II_R is NSRD*
 by (rule *NSRD-intro*, simp-all add: rdes closure unrest)

lemma *NSRD-Chaos* [closure]: *Chaos is NSRD*
 by (rule *NSRD-intro*, simp-all add: closure rdes unrest)

lemma *NSRD-Miracle* [closure]: *Miracle is NSRD*
 by (rule *NSRD-intro*, simp-all add: closure rdes unrest)

Post-composing a miracle filters out the non-terminating behaviours

lemma *NSRD-right-Miracle-tri-lemma*:
 assumes *P is NSRD*
 shows $P \;; \text{Miracle} = \mathbf{R}_s (pre_R P \vdash peri_R P \diamond false)$
 by (simp add: *NSRD-composition-wp closure assms rdes wp rpred*)

The set of non-terminating behaviours is a subset

lemma *NSRD-right-Miracle-refines*:
 assumes *P is NSRD*
 shows $P \sqsubseteq P \;; \text{Miracle}$
proof –
 have $\mathbf{R}_s (pre_R P \vdash peri_R P \diamond post_R P) \sqsubseteq \mathbf{R}_s (pre_R P \vdash peri_R P \diamond false)$
 by (rule *sdes-tri-refine-intro*, rel-auto+)
 thus ?thesis
 by (simp add: *NSRD-elim NSRD-right-Miracle-tri-lemma assms*)
qed

lemma *upower-Suc-NSRD-closed* [closure]:
 $P \text{ is NSRD} \implies P \wedge Suc\ n \text{ is NSRD}$
proof (induct *n*)
 case 0
 then show ?case
 by (simp)
next
 case (Suc *n*)
 then show ?case
 by (simp add: *NSRD-seqr-closure upred-semiring.power-Suc*)
qed

lemma *NSRD-power-Suc* [closure]:
 $P \text{ is NSRD} \implies P \;; P \wedge n \text{ is NSRD}$
 by (metis *upower-Suc-NSRD-closed upred-semiring.power-Suc*)

lemma *uplus-NSRD-closed* [closure]: $P \text{ is NSRD} \implies P^+ \text{ is NSRD}$
 by (simp add: *uplus-power-def closure*)

lemma *preR-power*:
 assumes *P is NSRD*
 shows $pre_R(P \;; P \wedge n) = (\bigsqcup_{i \in \{0..n\}} (post_R(P) \wedge i) \text{ wp}_r (pre_R(P)))$
proof (induct *n*)
 case 0
 then show ?case
 by (simp add: *wp closure*)
next
 case (Suc *n*) **note** *hyp = this*

```

have preR (P ^ (Suc n + 1)) = preR (P ;; P ^ (n+1))
  by (simp add: upred-semiring.power-Suc)
also have ... = (preR P ^ postR P wpr preR (P ^ (Suc n)))
  using NSRD-iff assms preR-NSRD-seq upower-Suc-NSRD-closed by fastforce
also have ... = (preR P ^ postR P wpr (⋂ i∈{0..n}. postR P ^ i wpr preR P))
  by (simp add: hyp upred-semiring.power-Suc)
also have ... = (preR P ^ (⋂ i∈{0..n}. postR P wpr (postR P ^ i wpr preR P)))
  by (simp add: wp)
also have ... = (preR P ^ (⋂ i∈{0..n}. (postR P ^ (i+1) wpr preR P)))
proof -
  have ∧ i. R1 (postR P ^ i ;; (¬r preR P)) = (postR P ^ i ;; (¬r preR P))
    by (induct-tac i, simp-all add: closure Healthy-if assms)
  thus ?thesis
    by (simp add: wp-rea-def upred-semiring.power-Suc seqr-assoc rpred closure assms)
qed
also have ... = (postR P ^ 0 wpr preR P ^ (⋂ i∈{0..n}. (postR P ^ (i+1) wpr preR P)))
  by (simp add: wp assms closure)
also have ... = (postR P ^ 0 wpr preR P ^ (⋂ i∈{1..Suc n}. (postR P ^ i wpr preR P)))
proof -
  have (⋂ i∈{0..n}. (postR P ^ (i+1) wpr preR P)) = (⋂ i∈{1..Suc n}. (postR P ^ i wpr preR P))
    by (rule cong[of Inf], simp-all add: fun-eq-iff)
    (metis (no-types, lifting) image-Suc-atLeastAtMost image-cong image-image)
  thus ?thesis by simp
qed
also have ... = (⋂ i∈insert 0 {1..Suc n}. (postR P ^ i wpr preR P))
  by (simp add: conj-upred-def)
also have ... = (⋂ i∈{0..Suc n}. postR P ^ i wpr preR P)
  by (simp add: atLeast0-atMost-Suc-eq-insert-0)
finally show ?case by (simp add: upred-semiring.power-Suc)
qed

```

```

lemma preR-power' [rdes]:
  assumes P is NSRD
  shows preR(P ;; Pn) = (⋂ i∈{0..n}. (postR(P) ^ i) wpr (preR(P)))
  by (simp add: preR-power assms USUP-as-Inf[THEN sym])

```

```

lemma preR-power-Suc [rdes]:
  assumes P is NSRD
  shows preR(PSuc n) = (⋂ i∈{0..n}. (postR(P) ^ i) wpr (preR(P)))
  by (simp add: upred-semiring.power-Suc rdes assms)

```

```

declare upred-semiring.power-Suc [simp]

```

```

lemma periR-power:
  assumes P is NSRD
  shows periR(P ;; Pn) = (preR(PSuc n) ⇒r (⋂ i∈{0..n}. postR(P) ^ i ;; periR(P)))
proof (induct n)
  case 0
  then show ?case
    by (simp add: NSRD-is-SRD NSRD-wait'-unrest-pre SRD-peri-under-pre assms)
next
  case (Suc n) note hyp = this
  have periR (P ^ (Suc n + 1)) = periR (P ;; P ^ (n+1))
    by (simp)
  also have ... = (preR(P ^ (Suc n + 1)) ⇒r (periR P ∨ postR P ;; periR (P ;; P ^ n)))

```



```

    by (simp add: closure assms rdes)
  also have ... = (preR(P ^ (Suc n + 1)) ⇒r (periR P ∨ postR P ;; (preR (P ^ (Suc n)) ⇒r (⋂ i∈{0..n}.
postR P ^ i) ;; periR P)))
    by (simp only: hyp)
  also
  have ... = (preR P ⇒r periR P ∨ (postR P wpr preR (P ;; P ^ n) ⇒r postR P ;; (preR (P ;; P ^ n)
⇒r (⋂ i∈{0..n}. postR P ^ i) ;; periR P)))
    by (simp add: rdes closure assms, rel-blast)
  also
  have ... = (preR P ⇒r periR P ∨ (postR P wpr preR (P ;; P ^ n) ⇒r postR P ;; ((⋂ i∈{0..n}. postR
P ^ i) ;; periR P)))
  proof -
    have (⋂ i∈{0..n}. postR P ^ i) is R1
    by (simp add: NSRD-is-SRD R1-Continuous R1-power Sup-Continuous-closed assms postR-SRD-R1)
    hence 1:((⋂ i∈{0..n}. postR P ^ i) ;; periR P) is R1
      by (simp add: closure assms)
    hence (preR (P ;; P ^ n) ⇒r (⋂ i∈{0..n}. postR P ^ i) ;; periR P) is R1
      by (simp add: closure)
    hence (postR P wpr preR (P ;; P ^ n) ⇒r postR P ;; (preR (P ;; P ^ n) ⇒r (⋂ i∈{0..n}. postR P
^ i) ;; periR P))
      = (postR P wpr preR (P ;; P ^ n) ⇒r R1(postR P) ;; R1(preR (P ;; P ^ n) ⇒r (⋂ i∈{0..n}.
postR P ^ i) ;; periR P))
      by (simp add: Healthy-if R1-post-SRD assms closure)
    thus ?thesis
      by (simp only: wp-rea-impl-lemma, simp add: Healthy-if 1, simp add: R1-post-SRD assms closure)
  qed
  also
  have ... = (preR P ∧ postR P wpr preR (P ;; P ^ n) ⇒r periR P ∨ postR P ;; ((⋂ i∈{0..n}. postR
P ^ i) ;; periR P))
    by (pred-auto)
  also
  have ... = (preR P ∧ postR P wpr preR (P ;; P ^ n) ⇒r periR P ∨ ((⋂ i∈{0..n}. postR P ^ (Suc
i)) ;; periR P))
    by (simp add: seq-Sup-distl seqr-assoc[THEN sym])
  also
  have ... = (preR P ∧ postR P wpr preR (P ;; P ^ n) ⇒r periR P ∨ ((⋂ i∈{1..Suc n}. postR P ^ i)
;; periR P))
  proof -
    have (⋂ i∈{0..n}. postR P ^ Suc i) = (⋂ i∈{1..Suc n}. postR P ^ i)
      apply (rule cong[of Sup], auto)
    apply (metis atLeast0AtMost atMost-iff image-Suc-atLeastAtMost rev-image-eqI upred-semiring.power-Suc)
      using Suc-le-D apply fastforce
    done
    thus ?thesis by simp
  qed
  also
  have ... = (preR P ∧ postR P wpr preR (P ;; P ^ n) ⇒r ((⋂ i∈{0..Suc n}. postR P ^ i) ;; periR P)
    by (simp add: SUP-atLeastAtMost-first uinf-or seqr-or-distl seqr-or-distr)
  also
  have ... = (preR(P ^ (Suc (Suc n)))) ⇒r ((⋂ i∈{0..Suc n}. postR P ^ i) ;; periR P)
    by (simp add: rdes closure assms)
  finally show ?case by (simp)
qed

```

lemma peri_R-power' [rdes]:

assumes P is NSRD
shows $\text{peri}_R(P ;; P^{\wedge} n) = (\text{pre}_R(P^{\wedge}(\text{Suc } n)) \Rightarrow_r (\bigwedge i \in \{0..n\} \cdot \text{post}_R(P) \wedge i) ;; \text{peri}_R(P))$
by (*simp add: periR-power assms UINF-as-Sup[THEN sym]*)

lemma *periR-power-Suc* [rdes]:
assumes P is NSRD
shows $\text{peri}_R(P^{\wedge}(\text{Suc } n)) = (\text{pre}_R(P^{\wedge}(\text{Suc } n)) \Rightarrow_r (\bigwedge i \in \{0..n\} \cdot \text{post}_R(P) \wedge i) ;; \text{peri}_R(P))$
by (*simp add: rdes assms*)

lemma *postR-power* [rdes]:
assumes P is NSRD
shows $\text{post}_R(P ;; P^{\wedge} n) = (\text{pre}_R(P^{\wedge}(\text{Suc } n)) \Rightarrow_r \text{post}_R(P) \wedge \text{Suc } n)$
proof (*induct n*)
case 0
then show ?case
by (*simp add: NSRD-is-SRD NSRD-wait'-unrest-pre SRD-post-under-pre assms*)
next
case ($\text{Suc } n$) **note** *hyp = this*
have $\text{post}_R(P \wedge (\text{Suc } n + 1)) = \text{post}_R(P ;; P^{\wedge} (n+1))$
by (*simp*)
also have $\dots = (\text{pre}_R(P \wedge (\text{Suc } n + 1)) \Rightarrow_r (\text{post}_R P ;; \text{post}_R(P ;; P^{\wedge} n)))$
by (*simp add: closure assms rdes*)
also have $\dots = (\text{pre}_R(P \wedge (\text{Suc } n + 1)) \Rightarrow_r (\text{post}_R P ;; (\text{pre}_R(P \wedge \text{Suc } n) \Rightarrow_r \text{post}_R P \wedge \text{Suc } n)))$
by (*simp only: hyp*)
also
have $\dots = (\text{pre}_R P \Rightarrow_r (\text{post}_R P \text{ wp}_r \text{pre}_R(P \wedge \text{Suc } n) \Rightarrow_r \text{post}_R P ;; (\text{pre}_R(P \wedge \text{Suc } n) \Rightarrow_r \text{post}_R P \wedge \text{Suc } n)))$
by (*simp add: rdes closure assms, pred-auto*)
also
have $\dots = (\text{pre}_R P \Rightarrow_r (\text{post}_R P \text{ wp}_r \text{pre}_R(P \wedge \text{Suc } n) \Rightarrow_r \text{post}_R P ;; \text{post}_R P \wedge \text{Suc } n))$
by (*metis (no-types, lifting) Healthy-if NSRD-is-SRD NSRD-power-Suc R1-power assms hyp postR-SRD-R1 upred-semiring.power-Suc wp-rea-impl-lemma*)
also
have $\dots = (\text{pre}_R P \wedge \text{post}_R P \text{ wp}_r \text{pre}_R(P \wedge \text{Suc } n) \Rightarrow_r \text{post}_R P \wedge \text{Suc } (\text{Suc } n))$
by (*pred-auto*)
also have $\dots = (\text{pre}_R(P^{\wedge}(\text{Suc } (\text{Suc } n))) \Rightarrow_r \text{post}_R P \wedge \text{Suc } (\text{Suc } n))$
by (*simp add: rdes closure assms*)
finally show ?case **by** (*simp*)
qed

lemma *postR-power-Suc* [rdes]:
assumes P is NSRD
shows $\text{post}_R(P^{\wedge}(\text{Suc } n)) = (\text{pre}_R(P^{\wedge}(\text{Suc } n)) \Rightarrow_r \text{post}_R(P) \wedge \text{Suc } n)$
by (*simp add: rdes assms*)

lemma *power-rdes-def* [rdes-def]:
assumes P is RC Q is RR R is RR $\$st' \# Q$
shows $(\mathbf{R}_s(P \vdash Q \diamond R))^{\wedge}(\text{Suc } n)$
 $= \mathbf{R}_s((\bigwedge i \in \{0..n\} \cdot (R \wedge i) \text{ wp}_r P) \vdash ((\bigwedge i \in \{0..n\} \cdot R \wedge i) ;; Q) \diamond (R \wedge \text{Suc } n))$
proof (*induct n*)
case 0
then show ?case
by (*simp add: wp assms closure*)
next
case ($\text{Suc } n$)

have 1: $(P \wedge (\bigsqcup i \in \{0..n\} \cdot R \text{ wp}_r (R \hat{\ } i \text{ wp}_r P))) = (\bigsqcup i \in \{0..Suc\ n\} \cdot R \hat{\ } i \text{ wp}_r P)$
 (is ?lhs = ?rhs)
proof –
have ?lhs = $(P \wedge (\bigsqcup i \in \{0..n\} \cdot (R \hat{\ } Suc\ i \text{ wp}_r P)))$
by (simp add: wp closure assms)
also have ... = $(P \wedge (\bigsqcup i \in \{0..n\} \cdot (R \hat{\ } Suc\ i \text{ wp}_r P)))$
by (simp only: USUP-as-Inf-collect)
also have ... = $(P \wedge (\bigsqcup i \in \{1..Suc\ n\} \cdot (R \hat{\ } i \text{ wp}_r P)))$
by (metis (no-types, lifting) INF-cong One-nat-def image-Suc-atLeastAtMost image-image)
also have ... = $(\bigsqcup i \in insert\ 0\ \{1..Suc\ n\} \cdot (R \hat{\ } i \text{ wp}_r P))$
by (simp add: wp assms closure conj-upred-def)
also have ... = $(\bigsqcup i \in \{0..Suc\ n\} \cdot (R \hat{\ } i \text{ wp}_r P))$
by (simp add: atLeastAtMost-insertL)
finally show ?thesis
by (simp add: USUP-as-Inf-collect)
qed

have 2: $(Q \vee R ;; (\prod i \in \{0..n\} \cdot R \hat{\ } i) ;; Q) = (\prod i \in \{0..Suc\ n\} \cdot R \hat{\ } i) ;; Q$
 (is ?lhs = ?rhs)
proof –
have ?lhs = $(Q \vee (\prod i \in \{0..n\} \cdot R \hat{\ } Suc\ i) ;; Q)$
by (simp add: seqr-assoc[THEN sym] seq-UINF-distl)
also have ... = $(Q \vee (\prod i \in \{0..n\} \cdot R \hat{\ } Suc\ i) ;; Q)$
by (simp only: UINF-as-Sup-collect)
also have ... = $(Q \vee (\prod i \in \{1..Suc\ n\} \cdot R \hat{\ } i) ;; Q)$
by (metis One-nat-def image-Suc-atLeastAtMost image-image)
also have ... = $((\prod i \in insert\ 0\ \{1..Suc\ n\} \cdot R \hat{\ } i) ;; Q)$
by (simp add: disj-upred-def[THEN sym] seqr-or-distl)
also have ... = $((\prod i \in \{0..Suc\ n\} \cdot R \hat{\ } i) ;; Q)$
by (simp add: atLeastAtMost-insertL)
finally show ?thesis
by (simp add: UINF-as-Sup-collect)
qed

have 3: $(\prod i \in \{0..n\} \cdot R \hat{\ } i) ;; Q$ is RR
proof –
have $(\prod i \in \{0..n\} \cdot R \hat{\ } i) ;; Q = (\prod i \in \{0..n\} \cdot R \hat{\ } i) ;; Q$
by (simp add: UINF-as-Sup-collect)
also have ... = $(\prod i \in insert\ 0\ \{1..n\} \cdot R \hat{\ } i) ;; Q$
by (simp add: atLeastAtMost-insertL)
also have ... = $(Q \vee (\prod i \in \{1..n\} \cdot R \hat{\ } i) ;; Q)$
by (metis (no-types, lifting) SUP-insert disj-upred-def seqr-left-unit seqr-or-distl upred-semiring.power-0)
also have ... = $(Q \vee (\prod i \in \{0..<n\} \cdot R \hat{\ } Suc\ i) ;; Q)$
by (metis One-nat-def atLeastLessThanSuc-atLeastAtMost image-Suc-atLeastLessThan image-image)
also have ... = $(Q \vee (\prod i \in \{0..<n\} \cdot R \hat{\ } Suc\ i) ;; Q)$
by (simp add: UINF-as-Sup-collect)
also have ... is RR
by (simp-all add: closure assms)
finally show ?thesis .
qed
from 1 2 3 **Suc show** ?case
by (simp add: Suc RHS-tri-normal-design-composition' closure assms wp)
qed

declare *upred-semiring.power-Suc* [simp del]

theorem *uplus-rdes-def* [rdes-def]:

assumes *P is RC Q is RR R is RR \$st' # Q*

shows $(\mathbf{R}_s(P \vdash Q \diamond R))^+ = \mathbf{R}_s(R^{*r} \text{ wp}_r P \vdash R^{*r} ;; Q \diamond R^+)$

proof –

have $1: (\bigcap i \cdot R \hat{=} i) ;; Q = R^{*r} ;; Q$

by (*metis* (*no-types*) *RA1* *assms*(2) *rea-skip-unit*(2) *rrel-thy.Star-def* *ustar-alt-def*)

show *?thesis*

by (*simp add: uplus-power-def seq-UINF-distr wp closure assms rdes-def*)
(metis 1 seq-UINF-distr')

qed

5.1 UTP theory

typeddecl *NSRDES*

abbreviation *NSRDES* \equiv *UTHY*(*NSRDES*, (*'s, 't::trace, 'α*) *rsp*)

overloading

nsrdes-hcond \equiv *utp-hcond* :: (*NSRDES*, (*'s, 't::trace, 'α*) *rsp*) *uthy* \Rightarrow ((*'s, 't, 'α*) *rsp* \times (*'s, 't, 'α*) *rsp*)

health

nsrdes-unit \equiv *utp-unit* :: (*NSRDES*, (*'s, 't::trace, 'α*) *rsp*) *uthy* \Rightarrow (*'s, 't, 'α*) *hrel-rsp*

begin

definition *nsrdes-hcond* :: (*NSRDES*, (*'s, 't::trace, 'α*) *rsp*) *uthy* \Rightarrow ((*'s, 't, 'α*) *rsp* \times (*'s, 't, 'α*) *rsp*)

health **where**

[*upred-defs*]: *nsrdes-hcond* *T* = *NSRD*

definition *nsrdes-unit* :: (*NSRDES*, (*'s, 't::trace, 'α*) *rsp*) *uthy* \Rightarrow (*'s, 't, 'α*) *hrel-rsp* **where**

[*upred-defs*]: *nsrdes-unit* *T* = *II_R*

end

Here, we show that normal stateful reactive designs form a Kleene UTP theory, and thus a Kleene algebra [4, 1]. This provides the basis for reasoning about iterative reactive contracts.

interpretation *nsrd-thy*: *utp-theory-kleene* *UTHY*(*NSRDES*, (*'s, 't::trace, 'α*) *rsp*)

rewrites $\bigwedge P. P \in \text{carrier } (\text{uthy-order } \text{NSRDES}) \longleftrightarrow P \text{ is } \text{NSRD}$

and *P is* $\mathcal{H}_{\text{NSRDES}} \longleftrightarrow P \text{ is } \text{NSRD}$

and $(\mu X \cdot F (\mathcal{H}_{\text{NSRDES}} X)) = (\mu X \cdot F (\text{NSRD } X))$

and $\text{carrier } (\text{uthy-order } \text{NSRDES}) \rightarrow \text{carrier } (\text{uthy-order } \text{NSRD}) \equiv \llbracket \text{NSRD} \rrbracket_H \rightarrow \llbracket \text{NSRD} \rrbracket_H$

and $\llbracket \mathcal{H}_{\text{NSRDES}} \rrbracket_H \rightarrow \llbracket \mathcal{H}_{\text{NSRDES}} \rrbracket_H \equiv \llbracket \text{NSRD} \rrbracket_H \rightarrow \llbracket \text{NSRD} \rrbracket_H$

and $\top_{\text{NSRDES}} = \text{Miracle}$

and $\mathcal{II}_{\text{NSRDES}} = \text{II}_R$

and $\text{le } (\text{uthy-order } \text{NSRDES}) = \text{op } \sqsubseteq$

proof –

interpret *lat*: *utp-theory-continuous* *UTHY*(*NSRDES*, (*'s, 't, 'α*) *rsp*)

by (*unfold-locales*, *simp-all add: nsrdes-hcond-def nsrdes-unit-def closure Healthy-if*)

show $1: \top_{\text{NSRDES}} = (\text{Miracle} :: (\text{'s, 't, 'α}) \text{ hrel-rsp})$

by (*metis* *NSRD-Miracle* *NSRD-is-SRD* *lat.top-healthy* *lat.utp-theory-continuous-axioms* *nsrdes-hcond-def* *srdes-theory-continuous.meet-top* *upred-semiring.add-commute* *utp-theory-continuous.meet-top*)

thus *utp-theory-kleene* *UTHY*(*NSRDES*, (*'s, 't, 'α*) *rsp*)

by (*unfold-locales*, *simp-all add: nsrdes-hcond-def nsrdes-unit-def closure Healthy-if* *Miracle-left-zero* *SRD-left-unit* *NSRD-right-unit*)

qed (*simp-all add: nsrdes-hcond-def nsrdes-unit-def closure Healthy-if*)

declare *nsrd-thy.top-healthy* [simp del]

declare *nsrd-thy.bottom-healthy* [*simp del*]

abbreviation *TestR* (*test_R*) **where**
test_R *P* \equiv *utest NSRDES P*

abbreviation *StarR* :: (*'s*, *'t::trace*, *'α*) *hrel-rsp* \Rightarrow (*'s*, *'t*, *'α*) *hrel-rsp* (*-^{*R}* [999] 999) **where**
StarR P \equiv *P★NSRDES*

We also show how to calculate the Kleene closure of a reactive design.

lemma *StarR-rdes-def* [*rdes-def*]:

assumes *P* is RC *Q* is RR *R* is RR \$st' $\#$ *Q*

shows ($\mathbf{R}_s(P \vdash Q \diamond R)^{*R} = \mathbf{R}_s((R^{*r} \text{ wp}_r P) \vdash R^{*r} ;; Q \diamond R^{*r})$)

by (*simp add: rrel-thy.Star-alt-def nsrd-thy.Star-alt-def assms closure rdes-def unrest rpred disj-upred-def*)

end

6 Syntax for reactive design contracts

theory *utp-rdes-contracts*

imports *utp-rdes-normal*

begin

We give an experimental syntax for reactive design contracts $[P \vdash Q|R]_R$, where *P* is a precondition on undashed state variables only, *Q* is a pericondition that can refer to the trace and before state but not the after state, and *R* is a postcondition. Both *Q* and *R* can refer only to the trace contribution through a HOL variable *trace* which is bound to $\&tt$.

definition *mk-RD* :: (*'s upred* \Rightarrow (*'t::trace* \Rightarrow *'s upred*) \Rightarrow (*'t* \Rightarrow *'s hrel*) \Rightarrow (*'s*, *'t*, *'a*) *hrel-rsp*) **where**
mk-RD P Q R = $\mathbf{R}_s([\![P]\!]_{S<} \vdash [\![Q(x)]\!]_{S<} [\![x \rightarrow \&tt]\!] \diamond [\![R(x)]\!]_S [\![x \rightarrow \&tt]\!])$

definition *trace-pred* :: (*'t::trace* \Rightarrow *'s upred*) \Rightarrow (*'s*, *'t*, *'α*) *hrel-rsp* **where**
[upred-defs]: trace-pred P = $[\![P\ x]\!]_{S<} [\![x \rightarrow \&tt]\!]$

syntax

-trace-var :: *logic*

-mk-RD :: *logic* \Rightarrow *logic* \Rightarrow *logic* \Rightarrow *logic* (*[-/ \vdash -/ | -]_R*)

-trace-pred :: *logic* \Rightarrow *logic* (*[-]_t*)

parse-translation \ll

let

fun trace-var-tr [] = *Syntax.free trace*
| *trace-var-tr* - = *raise Match*;

in

[(@{syntax-const -trace-var}, K trace-var-tr)]

end

\gg

translations

$[P \vdash Q \mid R]_R \Rightarrow \text{CONST } mk\text{-RD } P (\lambda \text{-trace-var. } Q) (\lambda \text{-trace-var. } R)$

$[P \vdash Q \mid R]_R \Leftarrow \text{CONST } mk\text{-RD } P (\lambda x. Q) (\lambda y. R)$

$[P]_t \Rightarrow \text{CONST } trace\text{-pred } (\lambda \text{-trace-var. } P)$

$[P]_t \Leftarrow \text{CONST } trace\text{-pred } (\lambda t. P)$

lemma *SRD-mk-RD* [*closure*]: $[P \vdash Q(\text{trace}) \mid R(\text{trace})]_R$ is SRD

by (simp add: mk-RD-def closure unrest)

lemma *preR-mk-RD [rdes]*: $\text{pre}_R([P \vdash Q(\text{trace}) \mid R(\text{trace})]_R) = R1(\lceil P \rceil_{S<})$
 by (simp add: mk-RD-def rea-pre-RHS-design usubst unrest R2c-not R2c-lift-state-pre)

lemma *trace-pred-RR-closed [closure]*:
 $\lceil P \text{ trace} \rceil_t$ is RR
 by (rel-auto)

lemma *unrest-trace-pred-st' [unrest]*:
 $\$st' \# \lceil P \text{ trace} \rceil_t$
 by (rel-auto)

lemma *R2c-msubst-tt*: $R2c(\text{msubst}(\lambda x. \lceil Q \ x \rceil_S) \ \&tt) = (\text{msubst}(\lambda x. \lceil Q \ x \rceil_S) \ \&tt)$
 by (rel-auto)

lemma *periR-mk-RD [rdes]*: $\text{peri}_R([P \vdash Q(\text{trace}) \mid R(\text{trace})]_R) = (\lceil P \rceil_{S<} \Rightarrow_r R1((\lceil Q(\text{trace}) \rceil_{S<}) \llbracket \text{trace} \rightarrow \&tt \rrbracket))$
 by (simp add: mk-RD-def rea-peri-RHS-design usubst unrest R2c-not R2c-lift-state-pre
 R2c-disj R2c-msubst-tt R1-disj R2c-rea-impl R1-rea-impl)

lemma *postR-mk-RD [rdes]*: $\text{post}_R([P \vdash Q(\text{trace}) \mid R(\text{trace})]_R) = (\lceil P \rceil_{S<} \Rightarrow_r R1((\lceil R(\text{trace}) \rceil_S) \llbracket \text{trace} \rightarrow \&tt \rrbracket))$
 by (simp add: mk-RD-def rea-post-RHS-design usubst unrest R2c-not R2c-lift-state-pre
 impl-alt-def R2c-disj R2c-msubst-tt R2c-rea-impl R1-rea-impl)

Refinement introduction law for contracts

lemma *RD-contract-refine*:

assumes

Q is SRD ‘ $\lceil P_1 \rceil_{S<} \Rightarrow \text{pre}_R Q$ ’
 ‘ $\lceil P_1 \rceil_{S<} \wedge \text{peri}_R Q \Rightarrow \lceil P_2 \ x \rceil_{S<} \llbracket x \rightarrow \&tt \rrbracket$ ’
 ‘ $\lceil P_1 \rceil_{S<} \wedge \text{post}_R Q \Rightarrow \lceil P_3 \ x \rceil_S \llbracket x \rightarrow \&tt \rrbracket$ ’

shows $[P_1 \vdash P_2(\text{trace}) \mid P_3(\text{trace})]_R \sqsubseteq Q$

proof –

have $[P_1 \vdash P_2(\text{trace}) \mid P_3(\text{trace})]_R \sqsubseteq \mathbf{R}_s(\text{pre}_R(Q) \vdash \text{peri}_R(Q) \diamond \text{post}_R(Q))$

using *assms*

by (simp add: mk-RD-def, rule-tac srdes-tri-refine-intro, simp-all)

thus ?thesis

by (simp add: SRD-reactive-tri-design assms(1))

qed

end

7 Reactive design tactics

theory *utp-rdes-tactics*

imports *utp-rdes-triples*

begin

Theorems for normalisation

lemmas *rdes-rel-norms* =

prod.case-eq-if

conj-assoc

disj-assoc

conj-disj-distr

conj-UINF-dist

conj-UINF-ind-dist

seqr-or-distl
seqr-or-distr
seq-UINF-distl
seq-UINF-distl'
seq-UINF-distr
seq-UINF-distr'

The following tactic can be used to simply and evaluate reactive predicates.

method *rpred-simp* = (*uepr-simp* *simps*: *rpred* *usubst* *closure* *unrest*)

Tactic to expand out healthy reactive design predicates into the syntactic triple form.

method *rdes-expand* **uses** *cls* = (*insert* *cls*, (*erule* *RD-elim*)*+*)

Tactic to simplify the definition of a reactive design

method *rdes-simp* **uses** *cls* *cong* *simps* =
 ((*rdes-expand* *cls*: *cls*)?, (*simp* *add*: *closure*)?, (*simp* *add*: *rdes-def* *rdes-rel-norms* *rdes* *rpred* *cls* *closure*
alpha *usubst* *unrest* *wp* *simps* *cong*: *cong*))

Tactic to split a refinement conjecture into three POs

method *rdes-refine-split* **uses** *cls* *cong* *simps* =
 (*rdes-simp* *cls*: *cls* *cong*: *cong* *simps*: *simps*; *rule-tac* *srdes-tri-refine-intro*)

Tactic to split an equality conjecture into three POs

method *rdes-eq-split* **uses** *cls* *cong* *simps* =
 (*rdes-simp* *cls*: *cls* *cong*: *cong* *simps*: *simps*; (*rule-tac* *srdes-tri-eq-intro*))

Tactic to prove a refinement

method *rdes-refine* **uses** *cls* *cong* *simps* =
 (*rdes-refine-split* *cls*: *cls* *cong*: *cong* *simps*: *simps*; (*insert* *cls*; *rel-auto*))

Tactics to prove an equality

method *rdes-eq* **uses** *cls* *cong* *simps* =
 (*rdes-eq-split* *cls*: *cls* *cong*: *cong* *simps*: *simps*; *rel-auto*)

Via antisymmetry

method *rdes-eq-anti* **uses** *cls* *cong* *simps* =
 (*rdes-simp* *cls*: *cls* *cong*: *cong* *simps*: *simps*; (*rule-tac* *antisym*; (*rule-tac* *srdes-tri-refine-intro*; *rel-auto*)))

Tactic to calculate pre/peri/postconditions from reactive designs

method *rdes-calc* = (*simp* *add*: *rdes* *rpred* *closure* *alpha* *usubst* *unrest* *wp* *prod.case-eq-if*)

The following tactic attempts to prove a reactive design refinement by calculation of the pre-, peri-, and postconditions and then showing three implications between them using *rel-blast*.

method *rdspl-refine* =
 (*rule-tac* *SRD-refine-intro*; (*simp* *add*: *closure* *rdes* *unrest* *usubst* ; *rel-blast*?))

The following tactic combines antisymmetry with the previous tactic to prove an equality.

method *rdspl-eq* =
 (*rule-tac* *antisym*, *rdes-refine*, *rdes-refine*)

end

8 Reactive design parallel-by-merge

```

theory utp-rdes-parallel
  imports
    utp-rdes-normal
    utp-rdes-tactics
begin

```

R3h implicitly depends on RD1, and therefore it requires that both sides be RD1. We also require that both sides are R3c, and that $wait_m$ is a quasi-unit, and div_m yields divergence.

```

lemma st-U0-alpha:  $\lceil \exists \$st \cdot II \rceil_0 = (\exists \$st \cdot \lceil II \rceil_0)$ 
  by (rel-auto)

```

```

lemma st-U1-alpha:  $\lceil \exists \$st \cdot II \rceil_1 = (\exists \$st \cdot \lceil II \rceil_1)$ 
  by (rel-auto)

```

```

definition skip-rm :: ('s, 't::trace, 'α) rsp merge (IIRM) where
  [upred-defs]: IIRM =  $(\exists \$st_{<} \cdot skip_m \vee (\neg \$ok_{<} \wedge \$tr_{<} \leq_u \$tr'))$ 

```

```

definition [upred-defs]: R3hm(M) = (IIRM ◁ $wait_{<} ▷ M)

```

```

lemma R3hm-idem: R3hm(R3hm(P)) = R3hm(P)
  by (rel-auto)

```

```

lemma R3h-par-by-merge [closure]:
  assumes P is R3h Q is R3h M is R3hm
  shows (P ||M Q) is R3h

```

proof –

```

  have (P ||M Q) = (((P ||M Q)⟦true/$ok⟧ ◁ $ok ▷ (P ||M Q)⟦false/$ok⟧)⟦true/$wait⟧ ◁ $wait ▷ (P ||M Q))

```

```

    by (simp add: cond-var-subst-left cond-var-subst-right)

```

```

  also have ... = (((P ||M Q)⟦true,true/$ok,$wait⟧ ◁ $ok ▷ (P ||M Q)⟦false,true/$ok,$wait⟧) ◁ $wait ▷ (P ||M Q))

```

```

    by (rel-auto)

```

```

  also have ... = ((( $\exists \$st \cdot II$ )⟦true,true/$ok,$wait⟧ ◁ $ok ▷ (P ||M Q)⟦false,true/$ok,$wait⟧) ◁ $wait ▷ (P ||M Q))

```

proof –

```

  have (P ||M Q)⟦true,true/$ok,$wait⟧ = (( $\lceil P \rceil_0 \wedge \lceil Q \rceil_1 \wedge \$v_{<}' =_u \$v$ ) ;; R3hm(M))⟦true,true/$ok,$wait⟧

```

```

    by (simp add: par-by-merge-def U0-as-alpha U1-as-alpha assms Healthy-if)

```

```

  also have ... = (( $\lceil P \rceil_0 \wedge \lceil Q \rceil_1 \wedge \$v_{<}' =_u \$v$ ) ;; ( $\exists \$st_{<} \cdot \$v' =_u \$v_{<}$ ))⟦true,true/$ok,$wait⟧

```

```

    by (rel-blast)

```

```

  also have ... = (( $\lceil R3h(P) \rceil_0 \wedge \lceil R3h(Q) \rceil_1 \wedge \$v_{<}' =_u \$v$ ) ;; ( $\exists \$st_{<} \cdot \$v' =_u \$v_{<}$ ))⟦true,true/$ok,$wait⟧

```

```

    by (simp add: assms Healthy-if)

```

```

  also have ... = ( $\exists \$st \cdot II$ )⟦true,true/$ok,$wait⟧

```

```

    by (rel-auto)

```

```

  finally show ?thesis by (simp add: closure assms unrest)

```

qed

```

  also have ... = ((( $\exists \$st \cdot II$ )⟦true,true/$ok,$wait⟧ ◁ $ok ▷ (R1(true))⟦false,true/$ok,$wait⟧) ◁ $wait ▷ (P ||M Q))

```

proof –

```

  have (P ||M Q)⟦false,true/$ok,$wait⟧ = (( $\lceil P \rceil_0 \wedge \lceil Q \rceil_1 \wedge \$v_{<}' =_u \$v$ ) ;; R3hm(M))⟦false,true/$ok,$wait⟧

```

```

    by (simp add: par-by-merge-def U0-as-alpha U1-as-alpha assms Healthy-if)

```

```

  also have ... = (( $\lceil P \rceil_0 \wedge \lceil Q \rceil_1 \wedge \$v_{<}' =_u \$v$ ) ;; ( $\$tr_{<} \leq_u \$tr'$ ))⟦false,true/$ok,$wait⟧

```

```

    by (rel-blast)

```

```

  also have ... = (( $\lceil R3h(P) \rceil_0 \wedge \lceil R3h(Q) \rceil_1 \wedge \$v_{<}' =_u \$v$ ) ;; ( $\$tr_{<} \leq_u \$tr'$ ))⟦false,true/$ok,$wait⟧

```

```

    by (simp add: assms Healthy-if)

```


also have ... = $(R1(true))\llbracket false, true / \$ok, \$wait \rrbracket$
 by (*rel-blast*)
 finally show ?thesis by simp
 qed
 also have ... = $((\exists \$st \cdot II) \triangleleft \$ok \triangleright R1(true)) \triangleleft \$wait \triangleright (P \parallel_M Q)$
 by (*rel-auto*)
 also have ... = $R3h(P \parallel_M Q)$
 by (*simp add: R3h-cases*)
 finally show ?thesis
 by (*simp add: Healthy-def*)
 qed

definition [*upred-defs*]: $RD1m(M) = (M \vee \neg \$ok_{<} \wedge \$tr_{<} \leq_u \$tr')$

lemma *RD1-par-by-merge* [*closure*]:

assumes P is $R1$ Q is $R1$ M is $R1m$ P is $RD1$ Q is $RD1$ M is $RD1m$
 shows $(P \parallel_M Q)$ is $RD1$

proof –

have 1: $(RD1(R1(P)) \parallel_{RD1m(R1m(M))} RD1(R1(Q)))\llbracket false / \$ok \rrbracket = R1(true)$
 by (*rel-blast*)
 have $(P \parallel_M Q) = (P \parallel_M Q)\llbracket true / \$ok \rrbracket \triangleleft \$ok \triangleright (P \parallel_M Q)\llbracket false / \$ok \rrbracket$
 by (*simp add: cond-var-split*)
 also have ... = $R1(P \parallel_M Q) \triangleleft \$ok \triangleright R1(true)$
 by (*metis 1 Healthy-if R1-par-by-merge assms calculation cond-idem cond-var-subst-right in-var-uvar ok-vwb-lens*)
 also have ... = $RD1(P \parallel_M Q)$
 by (*simp add: Healthy-if R1-par-by-merge RD1-alt-def assms(3)*)
 finally show ?thesis
 by (*simp add: Healthy-def*)

qed

lemma *RD2-par-by-merge* [*closure*]:

assumes M is $RD2$
 shows $(P \parallel_M Q)$ is $RD2$

proof –

have $(P \parallel_M Q) = ((P \parallel_s Q) ;; M)$
 by (*simp add: par-by-merge-def*)
 also from *assms* have ... = $((P \parallel_s Q) ;; (M ;; J))$
 by (*simp add: Healthy-def' RD2-def H2-def*)
 also from *assms* have ... = $((P \parallel_s Q) ;; M) ;; J$
 by (*simp add: seqr-assoc*)
 also from *assms* have ... = $RD2(P \parallel_M Q)$
 by (*simp add: RD2-def H2-def par-by-merge-def*)
 finally show ?thesis
 by (*simp add: Healthy-def'*)

qed

lemma *SRD-par-by-merge*:

assumes P is SRD Q is SRD M is $R1m$ M is $R2m$ M is $R3hm$ M is $RD1m$ M is $RD2$
 shows $(P \parallel_M Q)$ is SRD
 by (*rule SRD-intro, simp-all add: assms closure SRD-healths*)

definition *nmerge-rd0* (N_0) **where**

[*upred-defs*]: $N_0(M) = (\$wait' =_u (\$0 - wait \vee \$1 - wait) \wedge \$tr_{<} \leq_u \$tr' \wedge (\exists \$0 - ok; \$1 - ok; \$ok_{<}; \$ok'; \$0 - wait; \$1 - wait; \$wait_{<}; \$wait' \cdot M))$

definition *nmerge-rd1* (N_1) **where**

[*upred-defs*]: $N_1(M) = (\$ok' =_u (\$0-ok \wedge \$1-ok) \wedge N_0(M))$

definition *nmerge-rd* (N_R) **where**

[*upred-defs*]: $N_R(M) = ((\exists \$st_< \cdot \$v' =_u \$v_<) \triangleleft \$wait_< \triangleright N_1(M)) \triangleleft \$ok_< \triangleright (\$tr_< \leq_u \$tr')$

definition *merge-rd1* (M_1) **where**

[*upred-defs*]: $M_1(M) = (N_1(M) ;; II_R)$

definition *merge-rd* (M_R) **where**

[*upred-defs*]: $M_R(M) = N_R(M) ;; II_R$

abbreviation *rdes-par* ($- \parallel_R -$ - [85,0,86] 85) **where**

$P \parallel_{RM} Q \equiv P \parallel_{M_R(M)} Q$

Healthiness condition for reactive design merge predicates

definition [*upred-defs*]: $RDM(M) = R2m(\exists \$0-ok; \$1-ok; \$ok_<; \$ok'; \$0-wait; \$1-wait; \$wait_<; \$wait' \cdot M)$

lemma *nmerge-rd-is-R1m* [*closure*]:

$N_R(M)$ *is* *R1m*

by (*rel-blast*)

lemma *R2m-nmerge-rd*: $R2m(N_R(R2m(M))) = N_R(R2m(M))$

apply (*rel-auto*) **using** *minus-zero-eq* **by** *blast+*

lemma *nmerge-rd-is-R2m* [*closure*]:

M *is* *R2m* $\implies N_R(M)$ *is* *R2m*

by (*metis Healthy-def' R2m-nmerge-rd*)

lemma *nmerge-rd-is-R3hm* [*closure*]: $N_R(M)$ *is* *R3hm*

by (*rel-blast*)

lemma *nmerge-rd-is-RD1m* [*closure*]: $N_R(M)$ *is* *RD1m*

by (*rel-blast*)

lemma *merge-rd-is-RD3*: $M_R(M)$ *is* *RD3*

by (*metis Healthy-Idempotent RD3-Idempotent RD3-def merge-rd-def*)

lemma *merge-rd-is-RD2*: $M_R(M)$ *is* *RD2*

by (*simp add: RD3-implies-RD2 merge-rd-is-RD3*)

lemma *par-rdes-NSRD* [*closure*]:

assumes P *is* *SRD* Q *is* *SRD* M *is* *RDM*

shows $P \parallel_{RM} Q$ *is* *NSRD*

proof –

have $(P \parallel_{N_R} M Q ;; II_R)$ *is* *NSRD*

by (*rule NSRD-intro', simp-all add: SRD-healths closure assms*)

(*metis (no-types, lifting) Healthy-def R2-par-by-merge R2-seqr-closure R2m-nmerge-rd RDM-def SRD-healths(2) assms skip-srea-R2*

, *metis Healthy-Idempotent RD3-Idempotent RD3-def*)

thus *?thesis*

by (*simp add: merge-rd-def par-by-merge-def seqr-assoc*)

qed

lemma *RDM-intro*:

assumes M is *R2m* $\$0-ok \# M \$1-ok \# M \$ok_{<} \# M \$ok' \# M$
 $\$0-wait \# M \$1-wait \# M \$wait_{<} \# M \$wait' \# M$
shows M is *RDM*
using *assms*
by (*simp add: Healthy-def RDM-def ex-unrest unrest*)

lemma *RDM-unrests* [*unrest*]:

assumes M is *RDM*
shows $\$0-ok \# M \$1-ok \# M \$ok_{<} \# M \$ok' \# M$
 $\$0-wait \# M \$1-wait \# M \$wait_{<} \# M \$wait' \# M$
by (*subst Healthy-if[OF assms, THEN sym], simp-all add: RDM-def unrest, rel-auto*) $+$

lemma *RDM-R1m* [*closure*]: M is *RDM* $\implies M$ is *R1m*

by (*metis (no-types, hide-lams) Healthy-def R1m-idem R2m-def RDM-def*)

lemma *RDM-R2m* [*closure*]: M is *RDM* $\implies M$ is *R2m*

by (*metis (no-types, hide-lams) Healthy-def R2m-idem RDM-def*)

lemma *ex-st'-R2m-closed* [*closure*]:

assumes P is *R2m*
shows $(\exists \$st' \cdot P)$ is *R2m*

proof –

have $R2m(\exists \$st' \cdot R2m P) = (\exists \$st' \cdot R2m P)$
by (*rel-auto*)
thus *?thesis*
by (*metis Healthy-def' assms*)

qed

lemma *parallel-RR-closed*:

assumes P is *RR* Q is *RR* M is *R2m*
 $\$ok_{<} \# M \$wait_{<} \# M \$ok' \# M \$wait' \# M$
shows $P \parallel_M Q$ is *RR*
by (*rule RR-R2-intro, simp-all add: unrest assms RR-implies-R2 closure*)

lemma *parallel-ok-cases*:

$((P \parallel_s Q) ;; M) = ($
 $((P^t \parallel_s Q^t) ;; (M \llbracket true, true / \$0-ok, \$1-ok \rrbracket)) \vee$
 $((P^f \parallel_s Q^t) ;; (M \llbracket false, true / \$0-ok, \$1-ok \rrbracket)) \vee$
 $((P^t \parallel_s Q^f) ;; (M \llbracket true, false / \$0-ok, \$1-ok \rrbracket)) \vee$
 $((P^f \parallel_s Q^f) ;; (M \llbracket false, false / \$0-ok, \$1-ok \rrbracket)))$

proof –

have $((P \parallel_s Q) ;; M) = (\exists ok_0 \cdot (P \parallel_s Q) \llbracket \llbracket ok_0 \rrbracket / \$0-ok \rrbracket ;; M \llbracket \llbracket ok_0 \rrbracket / \$0-ok \rrbracket)$
by (*subst seqr-middle[of left-uvar ok], simp-all*)
also have $\dots = (\exists ok_0 \cdot \exists ok_1 \cdot ((P \parallel_s Q) \llbracket \llbracket ok_0 \rrbracket / \$0-ok \rrbracket \llbracket \llbracket ok_1 \rrbracket / \$1-ok \rrbracket ;; (M \llbracket \llbracket ok_0 \rrbracket / \$0-ok \rrbracket \llbracket \llbracket ok_1 \rrbracket / \$1-ok \rrbracket))$
by (*subst seqr-middle[of right-uvar ok], simp-all*)
also have $\dots = (\exists ok_0 \cdot \exists ok_1 \cdot (P \llbracket \llbracket ok_0 \rrbracket / \$ok' \rrbracket \parallel_s Q \llbracket \llbracket ok_1 \rrbracket / \$ok' \rrbracket) ;; (M \llbracket \llbracket ok_0 \rrbracket, \llbracket ok_1 \rrbracket / \$0-ok, \$1-ok \rrbracket))$
by (*rel-auto robust*)
also have $\dots = ($
 $((P^t \parallel_s Q^t) ;; (M \llbracket true, true / \$0-ok, \$1-ok \rrbracket)) \vee$
 $((P^f \parallel_s Q^t) ;; (M \llbracket false, true / \$0-ok, \$1-ok \rrbracket)) \vee$
 $((P^t \parallel_s Q^f) ;; (M \llbracket true, false / \$0-ok, \$1-ok \rrbracket)) \vee$
 $((P^f \parallel_s Q^f) ;; (M \llbracket false, false / \$0-ok, \$1-ok \rrbracket)))$
by (*simp add: true-alt-def[THEN sym] false-alt-def[THEN sym] disj-assoc*)

```

    utp-pred-laws.sup.left-commute utp-pred-laws.sup-commute usubst)
  finally show ?thesis .
qed

lemma skip-srea-ok-f [usubst]:
  
$$II_R^f = R1(\neg \$ok)$$

  by (rel-auto)

lemma nmerge0-rd-unrest [unrest]:
  
$$\$0-ok \# N_0 \ M \ \$1-ok \# N_0 \ M$$

  by (pred-auto)+

lemma parallel-assm-lemma:
  assumes  $P$  is RD2
  shows  $pre_s \uparrow (P \parallel_{M_R(M)} Q) = ((pre_s \uparrow P) \parallel_{N_0(M)} ;; R1(true) (cmt_s \uparrow Q))$ 
     $\vee ((cmt_s \uparrow P) \parallel_{N_0(M)} ;; R1(true) (pre_s \uparrow Q))$ 
proof -
  have  $pre_s \uparrow (P \parallel_{M_R(M)} Q) = pre_s \uparrow ((P \parallel_s Q) ;; M_R(M))$ 
    by (simp add: par-by-merge-def)
  also have ... =  $((P \parallel_s Q) \llbracket true, false / \$ok, \$wait \rrbracket ;; N_R \ M ;; R1(\neg \$ok))$ 
    by (simp add: merge-rd-def usubst, rel-auto)
  also have ... =  $((P \llbracket true, false / \$ok, \$wait \rrbracket \parallel_s Q \llbracket true, false / \$ok, \$wait \rrbracket) ;; N_1(M) ;; R1(\neg \$ok))$ 
    by (rel-auto robust, (metis)+)
  also have ... = (
     $((P \llbracket true, false / \$ok, \$wait \rrbracket)^t \parallel_s (Q \llbracket true, false / \$ok, \$wait \rrbracket)^t) ;; ((N_1 \ M) \llbracket true, true / \$0-ok, \$1-ok \rrbracket$ 
     $;; R1(\neg \$ok)) \vee$ 
     $((P \llbracket true, false / \$ok, \$wait \rrbracket)^f \parallel_s (Q \llbracket true, false / \$ok, \$wait \rrbracket)^t) ;; ((N_1 \ M) \llbracket false, true / \$0-ok, \$1-ok \rrbracket$ 
     $;; R1(\neg \$ok)) \vee$ 
     $((P \llbracket true, false / \$ok, \$wait \rrbracket)^t \parallel_s (Q \llbracket true, false / \$ok, \$wait \rrbracket)^f) ;; ((N_1 \ M) \llbracket true, false / \$0-ok, \$1-ok \rrbracket$ 
     $;; R1(\neg \$ok)) \vee$ 
     $((P \llbracket true, false / \$ok, \$wait \rrbracket)^f \parallel_s (Q \llbracket true, false / \$ok, \$wait \rrbracket)^f) ;; ((N_1 \ M) \llbracket false, false / \$0-ok, \$1-ok \rrbracket$ 
     $;; R1(\neg \$ok))$ 
  )
  (is - =  $(?C1 \vee_p ?C2 \vee_p ?C3 \vee_p ?C4)$ )
  by (subst parallel-ok-cases, subst-tac)
  also have ... =  $(?C2 \vee ?C3)$ 
proof -
  have  $?C1 = false$ 
    by (rel-auto)
  moreover have  $'?C4 \Rightarrow ?C3'$  (is  $'(?A ;; ?B) \Rightarrow (?C ;; ?D)'$ )
proof -
  from assms have  $'P^f \Rightarrow P^t'$ 
    by (metis RD2-def H2-equivalence Healthy-def)
  hence  $P: 'P_f^f \Rightarrow P_f^t'$ 
    by (rel-auto)
  have  $'?A \Rightarrow ?C'$ 
    using  $P$  by (rel-auto)
  moreover have  $'?B \Rightarrow ?D'$ 
    by (rel-auto)
  ultimately show ?thesis
    by (simp add: impl-seqr-mono)
qed
ultimately show ?thesis
  by (simp add: subsumption2)
qed
also have ... = (

```

$((pre_s \uparrow P) \parallel_s (cmt_s \uparrow Q)) \parallel (N_0 M \parallel R1(true)) \vee$
 $((cmt_s \uparrow P) \parallel_s (pre_s \uparrow Q)) \parallel (N_0 M \parallel R1(true))$
by (*rel-auto, metis+*)
also have ... = (
 $((pre_s \uparrow P) \parallel_{N_0 M} \parallel R1(true) (cmt_s \uparrow Q)) \vee$
 $((cmt_s \uparrow P) \parallel_{N_0 M} \parallel R1(true) (pre_s \uparrow Q))$
by (*simp add: par-by-merge-def*)
finally show *?thesis* .
qed

lemma *pre_s-SRD*:
assumes *P is SRD*
shows $pre_s \uparrow P = (\neg_r pre_R(P))$
proof –
have $pre_s \uparrow P = pre_s \uparrow \mathbf{R}_s(pre_R P \vdash peri_R P \diamond post_R P)$
by (*simp add: SRD-reactive-tri-design assms*)
also have ... = $R1(R2c(\neg pre_s \uparrow pre_R P))$
by (*simp add: RHS-def usubst R3h-def pre_s-design*)
also have ... = $R1(R2c(\neg pre_R P))$
by (*rel-auto*)
also have ... = $(\neg_r pre_R P)$
by (*simp add: R2c-not R2c-preR assms rea-not-def*)
finally show *?thesis* .
qed

lemma *parallel-assm*:
assumes *P is SRD Q is SRD*
shows $pre_R(P \parallel_{M_R(M)} Q) = (\neg_r ((\neg_r pre_R(P)) \parallel_{N_0(M)} \parallel R1(true) cmt_R(Q)) \wedge$
 $\neg_r (cmt_R(P) \parallel_{N_0(M)} \parallel R1(true) (\neg_r pre_R(Q))))$
(is ?lhs = ?rhs)
proof –
have $pre_R(P \parallel_{M_R(M)} Q) = (\neg_r (pre_s \uparrow P) \parallel_{N_0 M} \parallel R1 true (cmt_s \uparrow Q) \wedge$
 $\neg_r (cmt_s \uparrow P) \parallel_{N_0 M} \parallel R1 true (pre_s \uparrow Q))$
by (*simp add: pre_R-def parallel-assm-lemma assms SRD-healths R1-conj rea-not-def [THEN sym]*)
also have ... = *?rhs*
by (*simp add: pre_s-SRD assms cmt_R-def Healthy-if closure unrest*)
finally show *?thesis* .
qed

lemma *parallel-assm-unrest-wait' [unrest]*:
 $\llbracket P \text{ is SRD}; Q \text{ is SRD} \rrbracket \implies \$wait' \# pre_R(P \parallel_{M_R(M)} Q)$
by (*simp add: parallel-assm, simp add: par-by-merge-def unrest*)

lemma *JL1*: $(M_1 M)^t \llbracket false, true / \$0-ok, \$1-ok \rrbracket = N_0(M) \parallel R1(true)$
by (*rel-blast*)

lemma *JL2*: $(M_1 M)^t \llbracket true, false / \$0-ok, \$1-ok \rrbracket = N_0(M) \parallel R1(true)$
by (*rel-blast*)

lemma *JL3*: $(M_1 M)^t \llbracket false, false / \$0-ok, \$1-ok \rrbracket = N_0(M) \parallel R1(true)$
by (*rel-blast*)

lemma *JL4*: $(M_1 \ M)^t \llbracket \text{true}, \text{true} / \$0\text{--ok}, \$1\text{--ok} \rrbracket = (\$ok' \wedge N_0 \ M) ;; II_R^t$
 by (*simp add: merge-rd1-def usubst nmerge-rd1-def unrest*)

lemma *parallel-commitment-lemma-1*:

assumes *P* is *RD2*

shows $\text{cmt}_s \uparrow (P \parallel_{M_R(M)} Q) = ($

$((\text{cmt}_s \uparrow P) \parallel_{(\$ok' \wedge N_0 \ M) ;; II_R^t (\text{cmt}_s \uparrow Q)}) \vee$

$((\text{pre}_s \uparrow P) \parallel_{N_0(M) ;; R1(\text{true}) (\text{cmt}_s \uparrow Q)}) \vee$

$((\text{cmt}_s \uparrow P) \parallel_{N_0(M) ;; R1(\text{true}) (\text{pre}_s \uparrow Q)}))$

proof –

have $\text{cmt}_s \uparrow (P \parallel_{M_R(M)} Q) = (P \llbracket \text{true}, \text{false} / \$ok, \$wait \rrbracket \parallel_{(M_1(M))^t} Q \llbracket \text{true}, \text{false} / \$ok, \$wait \rrbracket)$

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

also have $\dots = ((P \llbracket \text{true}, \text{false} / \$ok, \$wait \rrbracket \parallel_s Q \llbracket \text{true}, \text{false} / \$ok, \$wait \rrbracket) ;; (M_1 \ M)^t)$

by (*simp add: par-by-merge-def*)

also have $\dots = ($

$((\text{cmt}_s \uparrow P) \parallel_s (\text{cmt}_s \uparrow Q)) ;; ((M_1 \ M)^t \llbracket \text{true}, \text{true} / \$0\text{--ok}, \$1\text{--ok} \rrbracket)) \vee$

$((\text{pre}_s \uparrow P) \parallel_s (\text{cmt}_s \uparrow Q)) ;; ((M_1 \ M)^t \llbracket \text{false}, \text{true} / \$0\text{--ok}, \$1\text{--ok} \rrbracket)) \vee$

$((\text{cmt}_s \uparrow P) \parallel_s (\text{pre}_s \uparrow Q)) ;; ((M_1 \ M)^t \llbracket \text{true}, \text{false} / \$0\text{--ok}, \$1\text{--ok} \rrbracket)) \vee$

$((\text{pre}_s \uparrow P) \parallel_s (\text{pre}_s \uparrow Q)) ;; ((M_1 \ M)^t \llbracket \text{false}, \text{false} / \$0\text{--ok}, \$1\text{--ok} \rrbracket))$

by (*subst parallel-ok-cases, subst-tac*)

also have $\dots = ($

$((\text{cmt}_s \uparrow P) \parallel_s (\text{cmt}_s \uparrow Q)) ;; ((M_1 \ M)^t \llbracket \text{true}, \text{true} / \$0\text{--ok}, \$1\text{--ok} \rrbracket)) \vee$

$((\text{pre}_s \uparrow P) \parallel_s (\text{cmt}_s \uparrow Q)) ;; (N_0(M) ;; R1(\text{true})) \vee$

$((\text{cmt}_s \uparrow P) \parallel_s (\text{pre}_s \uparrow Q)) ;; (N_0(M) ;; R1(\text{true})) \vee$

$((\text{pre}_s \uparrow P) \parallel_s (\text{pre}_s \uparrow Q)) ;; (N_0(M) ;; R1(\text{true}))$

(**is** – $(?C1 \vee_p ?C2 \vee_p ?C3 \vee_p ?C4)$)

by (*simp add: JL1 JL2 JL3*)

also have $\dots = ($

$((\text{cmt}_s \uparrow P) \parallel_s (\text{cmt}_s \uparrow Q)) ;; ((M_1(M))^t \llbracket \text{true}, \text{true} / \$0\text{--ok}, \$1\text{--ok} \rrbracket)) \vee$

$((\text{pre}_s \uparrow P) \parallel_s (\text{cmt}_s \uparrow Q)) ;; (N_0(M) ;; R1(\text{true})) \vee$

$((\text{cmt}_s \uparrow P) \parallel_s (\text{pre}_s \uparrow Q)) ;; (N_0(M) ;; R1(\text{true}))$

proof –

from *assms* **have** $P^f \Rightarrow P^t$

by (*metis RD2-def H2-equivalence Healthy-def*)

hence $P: P^f_f \Rightarrow P^t_f$

by (*rel-auto*)

have $?C4 \Rightarrow ?C3$ (**is** $(?A ;; ?B) \Rightarrow (?C ;; ?D)$)

proof –

have $?A \Rightarrow ?C$

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

thus *?thesis*

by (*simp add: impl-seqr-mono*)

qed

thus *?thesis*

by (*simp add: subsumption2*)

qed

finally show *?thesis*

by (*simp add: par-by-merge-def JL4*)

qed

lemma *parallel-commitment-lemma-2*:

assumes *P* is *RD2*

shows $\text{cmt}_s \uparrow (P \parallel_{M_R(M)} Q) =$

$((\text{cmt}_s \uparrow P) \parallel_{(\$ok' \wedge N_0 \ M) ;; II_R^t (\text{cmt}_s \uparrow Q)}) \vee \text{pre}_s \uparrow (P \parallel_{M_R(M)} Q)$

by (simp add: parallel-commitment-lemma-1 assms parallel-assm-lemma)

lemma parallel-commitment-lemma-3:

M is $R1m \implies (\$ok' \wedge N_0 M) ;; II_{R^t}$ is $R1m$

by (rel-simp, safe, metis+)

lemma parallel-commitment:

assumes P is SRD Q is SRD M is RDM

shows $cmt_R(P \parallel_{M_R(M)} Q) = (pre_R(P \parallel_{M_R(M)} Q) \Rightarrow_r cmt_R(P) \parallel_{(\$ok' \wedge N_0 M) ;; II_{R^t}} cmt_R(Q))$

by (simp add: parallel-commitment-lemma-2 parallel-commitment-lemma-3 Healthy-if assms cmt_R-def pre_s-SRD closure rea-impl-def disj-comm unrest)

theorem parallel-reactive-design:

assumes P is SRD Q is SRD M is RDM

shows $(P \parallel_{M_R(M)} Q) = \mathbf{R}_s($

$(\neg_r ((\neg_r pre_R(P)) \parallel_{N_0(M) ;; R1(true)} cmt_R(Q)) \wedge$

$\neg_r (cmt_R(P) \parallel_{N_0(M) ;; R1(true)} (\neg_r pre_R(Q)))) \vdash$

$(cmt_R(P) \parallel_{(\$ok' \wedge N_0 M) ;; II_{R^t}} cmt_R(Q)))$ (is ?lhs = ?rhs)

proof –

have $(P \parallel_{M_R(M)} Q) = \mathbf{R}_s(pre_R(P \parallel_{M_R(M)} Q) \vdash cmt_R(P \parallel_{M_R(M)} Q))$

by (metis Healthy-def NSRD-is-SRD SRD-as-reactive-design assms(1) assms(2) assms(3) par-rdes-NSRD)

also have ... = ?rhs

by (simp add: parallel-assm parallel-commitment design-export-spec assms, rel-auto)

finally show ?thesis .

qed

lemma parallel-periccondition-lemma1:

$(\$ok' \wedge P) ;; II_R \llbracket true, true / \$ok', \$wait' \rrbracket = (\exists \$st' \cdot P) \llbracket true, true / \$ok', \$wait' \rrbracket$

(is ?lhs = ?rhs)

proof –

have ?lhs = $(\$ok' \wedge P) ;; (\exists \$st \cdot II) \llbracket true, true / \$ok', \$wait' \rrbracket$

by (rel-blast)

also have ... = ?rhs

by (rel-auto)

finally show ?thesis .

qed

lemma parallel-periccondition-lemma2:

assumes M is RDM

shows $(\exists \$st' \cdot N_0(M)) \llbracket true, true / \$ok', \$wait' \rrbracket = ((\$0\text{-wait} \vee \$1\text{-wait}) \wedge (\exists \$st' \cdot M))$

proof –

have $(\exists \$st' \cdot N_0(M)) \llbracket true, true / \$ok', \$wait' \rrbracket = (\exists \$st' \cdot (\$0\text{-wait} \vee \$1\text{-wait}) \wedge \$tr' \geq_u \$tr_{<} \wedge M)$

by (simp add: usubst unrest nmerge-rd0-def ex-unrest Healthy-if R1m-def assms)

also have ... = $(\exists \$st' \cdot (\$0\text{-wait} \vee \$1\text{-wait}) \wedge M)$

by (metis (no-types, hide-lams) Healthy-if R1m-def R1m-idem R2m-def RDM-def assms utp-pred-laws.inf-commute)

also have ... = $(\$0\text{-wait} \vee \$1\text{-wait}) \wedge (\exists \$st' \cdot M)$

by (rel-auto)

finally show ?thesis .

qed

lemma parallel-periccondition-lemma3:

$((\$0\text{-wait} \vee \$1\text{-wait}) \wedge (\exists \$st' \cdot M)) = ((\$0\text{-wait} \wedge \$1\text{-wait} \wedge (\exists \$st' \cdot M)) \vee (\neg \$0\text{-wait} \wedge \$1\text{-wait} \wedge (\exists \$st' \cdot M)) \vee (\$0\text{-wait} \wedge \neg \$1\text{-wait} \wedge (\exists \$st' \cdot M)))$

by (rel-auto)

lemma *parallel-pericondition* [rdes]:

fixes $M :: ('s, 't :: \text{trace}, 'a) \text{rsp merge}$

assumes P is SRD Q is SRD M is RDM

shows $\text{peri}_R(P \parallel_{M_R(M)} Q) = (\text{pre}_R(P \parallel_{M_R} M Q) \Rightarrow_r \text{peri}_R(P) \parallel_{\exists \$st' . M} \text{peri}_R(Q)$
 $\vee \text{post}_R(P) \parallel_{\exists \$st' . M} \text{peri}_R(Q)$
 $\vee \text{peri}_R(P) \parallel_{\exists \$st' . M} \text{post}_R(Q))$

proof –

have $\text{peri}_R(P \parallel_{M_R(M)} Q) =$

$(\text{pre}_R(P \parallel_{M_R} M Q) \Rightarrow_r \text{cmt}_R P \parallel_{(\$ok' \wedge N_0 M) ;; II_R[\text{true}, \text{true}/\$ok', \$wait']} \text{cmt}_R Q)$

by (simp add: peri-cmt-def parallel-commitment SRD-healths assms usubst unrest assms)

also have $\dots = (\text{pre}_R(P \parallel_{M_R} M Q) \Rightarrow_r \text{cmt}_R P \parallel_{(\exists \$st' . N_0 M)[\text{true}, \text{true}/\$ok', \$wait']} \text{cmt}_R Q)$

by (simp add: parallel-pericondition-lemma1)

also have $\dots = (\text{pre}_R(P \parallel_{M_R} M Q) \Rightarrow_r \text{cmt}_R P \parallel_{(\$0\text{-wait} \vee \$1\text{-wait}) \wedge (\exists \$st' . M)} \text{cmt}_R Q)$

by (simp add: parallel-pericondition-lemma2 assms)

also have $\dots = (\text{pre}_R(P \parallel_{M_R} M Q) \Rightarrow_r ((\lceil \text{cmt}_R P \rceil_0 \wedge \lceil \text{cmt}_R Q \rceil_1 \wedge \$\mathbf{v}_{<}' =_u \$\mathbf{v}) ;; (\$0\text{-wait} \wedge$
 $\$1\text{-wait} \wedge (\exists \$st' . M)))$
 $\vee (\lceil \text{cmt}_R P \rceil_0 \wedge \lceil \text{cmt}_R Q \rceil_1 \wedge \$\mathbf{v}_{<}' =_u \$\mathbf{v}) ;; (\neg \$0\text{-wait} \wedge \$1\text{-wait}$
 $\wedge (\exists \$st' . M)))$

$\vee (\lceil \text{cmt}_R P \rceil_0 \wedge \lceil \text{cmt}_R Q \rceil_1 \wedge \$\mathbf{v}_{<}' =_u \$\mathbf{v}) ;; (\$0\text{-wait} \wedge \neg \1-wait
 $\wedge (\exists \$st' . M)))$

by (simp add: par-by-merge-alt-def parallel-pericondition-lemma3 seqr-or-distr)

also have $\dots = (\text{pre}_R(P \parallel_{M_R} M Q) \Rightarrow_r ((\lceil \text{peri}_R P \rceil_0 \wedge \lceil \text{peri}_R Q \rceil_1 \wedge \$\mathbf{v}_{<}' =_u \$\mathbf{v}) ;; (\exists \$st' . M)$

$\vee (\lceil \text{post}_R P \rceil_0 \wedge \lceil \text{peri}_R Q \rceil_1 \wedge \$\mathbf{v}_{<}' =_u \$\mathbf{v}) ;; (\exists \$st' . M)$

$\vee (\lceil \text{peri}_R P \rceil_0 \wedge \lceil \text{post}_R Q \rceil_1 \wedge \$\mathbf{v}_{<}' =_u \$\mathbf{v}) ;; (\exists \$st' . M)))$

by (simp add: seqr-right-one-point-true seqr-right-one-point-false cmt_R-def post_R-def peri_R-def usubst unrest assms)

also have $\dots = (\text{pre}_R(P \parallel_{M_R} M Q) \Rightarrow_r \text{peri}_R(P) \parallel_{\exists \$st' . M} \text{peri}_R(Q)$

$\vee \text{post}_R(P) \parallel_{\exists \$st' . M} \text{peri}_R(Q)$

$\vee \text{peri}_R(P) \parallel_{\exists \$st' . M} \text{post}_R(Q))$

by (simp add: par-by-merge-alt-def)

finally show ?thesis .

qed

lemma *parallel-postcondition-lemma1*:

$(\$ok' \wedge P) ;; II_R[\text{true}, \text{false}/\$ok', \$wait'] = P[\text{true}, \text{false}/\$ok', \$wait']$

(is ?lhs = ?rhs)

proof –

have ?lhs = $(\$ok' \wedge P) ;; II[\text{true}, \text{false}/\$ok', \$wait']$

by (rel-blast)

also have $\dots = ?rhs$

by (rel-auto)

finally show ?thesis .

qed

lemma *parallel-postcondition-lemma2*:

assumes M is RDM

shows $(N_0(M))[\text{true}, \text{false}/\$ok', \$wait'] = ((\neg \$0\text{-wait} \wedge \neg \$1\text{-wait}) \wedge M)$

proof –

have $(N_0(M))[\text{true}, \text{false}/\$ok', \$wait'] = ((\neg \$0\text{-wait} \wedge \neg \$1\text{-wait}) \wedge \$tr' \geq_u \$tr_{<} \wedge M)$

by (simp add: usubst unrest nmerge-rd0-def ex-unrest Healthy-if R1m-def assms)

also have $\dots = ((\neg \$0\text{-wait} \wedge \neg \$1\text{-wait}) \wedge M)$

by (metis Healthy-if R1m-def RDM-R1m assms utp-pred-laws.inf-commute)

finally show ?thesis .

qed

lemma *parallel-postcondition* [rdes]:

fixes $M :: ('s, 't :: \text{trace}, 'a) \text{ rsp merge}$

assumes $P \text{ is SRD } Q \text{ is SRD } M \text{ is RDM}$

shows $\text{post}_R(P \parallel_{M_R(M)} Q) = (\text{pre}_R(P \parallel_{M_R(M)} Q) \Rightarrow_r \text{post}_R(P) \parallel_M \text{post}_R(Q))$

proof –

have $\text{post}_R(P \parallel_{M_R(M)} Q) =$

$(\text{pre}_R(P \parallel_{M_R(M)} Q) \Rightarrow_r \text{cmt}_R P \parallel_{(\$ok' \wedge N_0 M)} ;; II_R[\text{true}, \text{false}/\$ok', \$wait'] \text{cmt}_R Q)$

by (*simp add: post-cmt-def parallel-commitment assms usubst unrest SRD-healths*)

also have $\dots = (\text{pre}_R(P \parallel_{M_R(M)} Q) \Rightarrow_r \text{cmt}_R P \parallel_{(\neg \$0\text{-wait} \wedge \neg \$1\text{-wait} \wedge M)} \text{cmt}_R Q)$

by (*simp add: parallel-postcondition-lemma1 parallel-postcondition-lemma2 assms,*
simp add: utp-pred-laws.inf-commute utp-pred-laws.inf-left-commute)

also have $\dots = (\text{pre}_R(P \parallel_{M_R(M)} Q) \Rightarrow_r \text{post}_R P \parallel_M \text{post}_R Q)$

by (*simp add: par-by-merge-alt-def seqr-right-one-point-false usubst unrest cmt_R-def post_R-def assms*)

finally show *?thesis* .

qed

lemma *parallel-precondition-lemma*:

fixes $M :: ('s, 't :: \text{trace}, 'a) \text{ rsp merge}$

assumes $P \text{ is NSRD } Q \text{ is NSRD } M \text{ is RDM}$

shows $(\neg_r \text{pre}_R(P)) \parallel_{N_0(M)} ;; R1(\text{true}) \text{cmt}_R(Q) =$

$((\neg_r \text{pre}_R P) \parallel_M ;; R1(\text{true}) \text{peri}_R Q \vee (\neg_r \text{pre}_R P) \parallel_M ;; R1(\text{true}) \text{post}_R Q)$

proof –

have $((\neg_r \text{pre}_R(P)) \parallel_{N_0(M)} ;; R1(\text{true}) \text{cmt}_R(Q)) =$

$((\neg_r \text{pre}_R(P)) \parallel_{N_0(M)} ;; R1(\text{true}) (\text{peri}_R(Q) \diamond \text{post}_R(Q)))$

by (*simp add: wait'-cond-peri-post-cmt*)

also have $\dots = ((\neg_r \text{pre}_R(P))_0 \wedge [\text{peri}_R(Q) \diamond \text{post}_R(Q)]_1 \wedge \$\mathbf{v}_{<} =_u \$\mathbf{v}) ;; N_0(M) ;; R1(\text{true}))$

by (*simp add: par-by-merge-alt-def*)

also have $\dots = ((\neg_r \text{pre}_R(P))_0 \wedge [\text{peri}_R(Q)]_1 \triangleleft \$1\text{-wait}' \triangleright [\text{post}_R(Q)]_1 \wedge \$\mathbf{v}_{<} =_u \$\mathbf{v}) ;; N_0(M) ;; R1(\text{true}))$

by (*simp add: wait'-cond-def alpha*)

also have $\dots = (((\neg_r \text{pre}_R(P))_0 \wedge [\text{peri}_R(Q)]_1 \wedge \$\mathbf{v}_{<} =_u \$\mathbf{v}) \triangleleft \$1\text{-wait}' \triangleright ((\neg_r \text{pre}_R(P))_0 \wedge [\text{post}_R(Q)]_1 \wedge \$\mathbf{v}_{<} =_u \$\mathbf{v})) ;; N_0(M) ;; R1(\text{true}))$

(is $(?P ;; -) = (?Q ;; -)$ **)**

proof –

have $?P = ?Q$

by (*rel-auto*)

thus *?thesis* **by** *simp*

qed

also have $\dots = ((\neg_r \text{pre}_R P)_0 \wedge [\text{peri}_R Q]_1 \wedge \$\mathbf{v}_{<} =_u \$\mathbf{v})[\text{true}/\$1\text{-wait}'] ;; (N_0 M ;; R1 \text{true})[\text{true}/\$1\text{-wait}] \vee$

$(\neg_r \text{pre}_R P)_0 \wedge [\text{post}_R Q]_1 \wedge \$\mathbf{v}_{<} =_u \$\mathbf{v})[\text{false}/\$1\text{-wait}'] ;; (N_0 M ;; R1 \text{true})[\text{false}/\$1\text{-wait}]$

by (*simp add: cond-inter-var-split*)

also have $\dots = ((\neg_r \text{pre}_R P)_0 \wedge [\text{peri}_R Q]_1 \wedge \$\mathbf{v}_{<} =_u \$\mathbf{v}) ;; N_0 M[\text{true}/\$1\text{-wait}] ;; R1 \text{true} \vee$
 $(\neg_r \text{pre}_R P)_0 \wedge [\text{post}_R Q]_1 \wedge \$\mathbf{v}_{<} =_u \$\mathbf{v}) ;; N_0 M[\text{false}/\$1\text{-wait}] ;; R1 \text{true}$

by (*simp add: usubst unrest*)

also have $\dots = ((\neg_r \text{pre}_R P)_0 \wedge [\text{peri}_R Q]_1 \wedge \$\mathbf{v}_{<} =_u \$\mathbf{v}) ;; (\$wait' \wedge M) ;; R1 \text{true} \vee$
 $(\neg_r \text{pre}_R P)_0 \wedge [\text{post}_R Q]_1 \wedge \$\mathbf{v}_{<} =_u \$\mathbf{v}) ;; (\$wait' =_u \$0\text{-wait} \wedge M) ;; R1 \text{true}$

proof –

have $(\$tr' \geq_u \$tr_{<} \wedge M) = M$

using *RDM-R1m[OF assms(3)]*

by (*simp add: Healthy-def R1m-def conj-comm*)

thus ?thesis
 by (simp add: nmerge-rd0-def unrest assms closure ex-unrest usubst)
 qed
 also have ... = (($\lceil \neg_r \text{pre}_R P \rceil_0 \wedge \lceil \text{peri}_R Q \rceil_1 \wedge \$\mathbf{v}_{<}' =_u \$\mathbf{v}$) ;; M ;; R1 true \vee
 ($\lceil \neg_r \text{pre}_R P \rceil_0 \wedge \lceil \text{post}_R Q \rceil_1 \wedge \$\mathbf{v}_{<}' =_u \$\mathbf{v}$) ;; M ;; R1 true)
 (is ($?P_1 \vee_p ?P_2$) = ($?Q_1 \vee ?Q_2$))
 proof –
 have $?P_1 = (\lceil \neg_r \text{pre}_R P \rceil_0 \wedge \lceil \text{peri}_R Q \rceil_1 \wedge \$\mathbf{v}_{<}' =_u \$\mathbf{v}) ;; (M \wedge \$\text{wait}')$;; R1 true
 by (simp add: conj-comm)
 hence 1: $?P_1 = ?Q_1$
 by (simp add: segr-left-one-point-true segr-left-one-point-false add: unrest usubst closure assms)
 have $?P_2 = ((\lceil \neg_r \text{pre}_R P \rceil_0 \wedge \lceil \text{post}_R Q \rceil_1 \wedge \$\mathbf{v}_{<}' =_u \$\mathbf{v}) ;; (M \wedge \$\text{wait}')) ;; R1 \text{ true } \vee$
 ($\lceil \neg_r \text{pre}_R P \rceil_0 \wedge \lceil \text{post}_R Q \rceil_1 \wedge \$\mathbf{v}_{<}' =_u \$\mathbf{v}$) ;; (M $\wedge \neg \$\text{wait}'$) ;; R1 true)
 by (subst segr-bool-split[of left-uvar wait], simp-all add: usubst unrest assms closure conj-comm)
 hence 2: $?P_2 = ?Q_2$
 by (simp add: segr-left-one-point-true segr-left-one-point-false unrest usubst closure assms)
 from 1 2 show ?thesis by simp
 qed
 also have ... = ($(\neg_r \text{pre}_R P) \parallel_M ;; R1(\text{true}) \text{peri}_R Q \vee (\neg_r \text{pre}_R P) \parallel_M ;; R1(\text{true}) \text{post}_R Q$)
 by (simp add: par-by-merge-alt-def)
 finally show ?thesis .
 qed

lemma swap-nmerge-rd0:
 $\text{swap}_m ;; N_0(M) = N_0(\text{swap}_m ;; M)$
 by (rel-auto, meson+)

lemma SymMerge-nmerge-rd0 [closure]:
 $M \text{ is SymMerge} \implies N_0(M) \text{ is SymMerge}$
 by (rel-auto, meson+)

lemma swap-merge-rd':
 $\text{swap}_m ;; N_R(M) = N_R(\text{swap}_m ;; M)$
 by (rel-blast)

lemma swap-merge-rd:
 $\text{swap}_m ;; M_R(M) = M_R(\text{swap}_m ;; M)$
 by (simp add: merge-rd-def segr-assoc[THEN sym] swap-merge-rd')

lemma SymMerge-merge-rd [closure]:
 $M \text{ is SymMerge} \implies M_R(M) \text{ is SymMerge}$
 by (simp add: Healthy-def swap-merge-rd)

lemma nmerge-rd1-merge3:
 assumes $M \text{ is RDM}$
 shows $\mathbf{M3}(N_1(M)) = (\$ok' =_u (\$0-ok \wedge \$1-0-ok \wedge \$1-1-ok) \wedge$
 $\$wait' =_u (\$0-wait \vee \$1-0-wait \vee \$1-1-wait)) \wedge$
 $\mathbf{M3}(M))$

proof –
 have $\mathbf{M3}(N_1(M)) = \mathbf{M3}(\$ok' =_u (\$0-ok \wedge \$1-ok) \wedge$
 $\$wait' =_u (\$0-wait \vee \$1-wait) \wedge$
 $\$tr_{<} \leq_u \$tr' \wedge$
 $(\exists \{ \$0-ok, \$1-ok, \$ok_{<}, \$ok', \$0-wait, \$1-wait, \$wait_{<}, \$wait' \} \cdot \text{RDM}(M)))$
 by (simp add: nmerge-rd1-def nmerge-rd0-def assms Healthy-if)
 also have ... = $\mathbf{M3}(\$ok' =_u (\$0-ok \wedge \$1-ok) \wedge \$wait' =_u (\$0-wait \vee \$1-wait) \wedge \text{RDM}(M))$

by (*rel-blast*)
 also have ... = ($\$ok' =_u (\$0-ok \wedge \$1-0-ok \wedge \$1-1-ok) \wedge \$wait' =_u (\$0-wait \vee \$1-0-wait \vee \$1-1-wait) \wedge \mathbf{M3}(RDM(M))$)
 by (*rel-blast*)
 also have ... = ($\$ok' =_u (\$0-ok \wedge \$1-0-ok \wedge \$1-1-ok) \wedge \$wait' =_u (\$0-wait \vee \$1-0-wait \vee \$1-1-wait) \wedge \mathbf{M3}(M)$)
 by (*simp add: assms Healthy-if*)
 finally show *?thesis* .
 qed

lemma *nmerge-rd-merge3*:

$\mathbf{M3}(N_R(M)) = (\exists \$st_{<} \cdot \$\mathbf{v}' =_u \$\mathbf{v}_{<}) \triangleleft \$wait_{<} \triangleright \mathbf{M3}(N_1 M) \triangleleft \$ok_{<} \triangleright (\$tr_{<} \leq_u \$tr')$
 by (*rel-blast*)

lemma *swap-merge-RDM-closed* [*closure*]:

assumes *M is RDM*
 shows *swap_m ;; M is RDM*

proof –

have $RDM(\text{swap}_m ;; RDM(M)) = (\text{swap}_m ;; RDM(M))$
 by (*rel-auto*)
 thus *?thesis*
 by (*metis Healthy-def' assms*)

qed

lemma *parallel-precondition*:

fixes *M :: ('s, 't::trace, 'α) rsp merge*
 assumes *P is NSRD Q is NSRD M is RDM*
 shows $\text{pre}_R(P \parallel_{M_R(M)} Q) =$

$(\neg_r ((\neg_r \text{pre}_R P) \parallel_M ;; R1(\text{true}) \text{peri}_R Q) \wedge$
 $\neg_r ((\neg_r \text{pre}_R P) \parallel_M ;; R1(\text{true}) \text{post}_R Q) \wedge$
 $\neg_r ((\neg_r \text{pre}_R Q) \parallel_{(\text{swap}_m ;; M)} ;; R1(\text{true}) \text{peri}_R P) \wedge$
 $\neg_r ((\neg_r \text{pre}_R Q) \parallel_{(\text{swap}_m ;; M)} ;; R1(\text{true}) \text{post}_R P))$

proof –

have *a*: $(\neg_r \text{pre}_R(P)) \parallel_{N_0(M)} ;; R1(\text{true}) \text{cmt}_R(Q) =$
 $((\neg_r \text{pre}_R P) \parallel_M ;; R1(\text{true}) \text{peri}_R Q \vee (\neg_r \text{pre}_R P) \parallel_M ;; R1(\text{true}) \text{post}_R Q)$
 by (*simp add: parallel-precondition-lemma assms*)

have *b*: $(\neg_r \text{cmt}_R P \parallel_{N_0 M} ;; R1 \text{true} (\neg_r \text{pre}_R Q)) =$
 $(\neg_r (\neg_r \text{pre}_R(Q)) \parallel_{N_0(\text{swap}_m ;; M)} ;; R1(\text{true}) \text{cmt}_R(P))$

by (*simp add: swap-nmerge-rd0[THEN sym] seqr-assoc[THEN sym] par-by-merge-def par-sep-swap*)

have *c*: $(\neg_r \text{pre}_R(Q)) \parallel_{N_0(\text{swap}_m ;; M)} ;; R1(\text{true}) \text{cmt}_R(P) =$
 $((\neg_r \text{pre}_R Q) \parallel_{(\text{swap}_m ;; M)} ;; R1(\text{true}) \text{peri}_R P \vee (\neg_r \text{pre}_R Q) \parallel_{(\text{swap}_m ;; M)} ;; R1(\text{true}) \text{post}_R P)$

by (*simp add: parallel-precondition-lemma closure assms*)

show *?thesis*

by (*simp add: parallel-assm closure assms a b c, rel-auto*)

qed

Weakest Parallel Precondition

definition *wrR* ::

$(t::\text{trace}, 'α) \text{hrel-rp} \Rightarrow$
 $(t :: \text{trace}, 'α) \text{rp merge} \Rightarrow$
 $(t, 'α) \text{hrel-rp} \Rightarrow$

(*t*, *'α*) *hrel-rp* (- *wr_R'(-)* - [60,0,61] 61)
where [*upred-defs*]: *Q wr_R(M) P* = (\neg_r ((\neg_r *P*) \parallel_M ;; *R1(true) Q*))

lemma *wrR-R1 [closure]*:
M is R1m \implies *Q wr_R(M) P is R1*
by (*simp add: wrR-def closure*)

lemma *R2-rea-not*: *R2(\neg_r P)* = (\neg_r *R2(P)*)
by (*rel-auto*)

lemma *wrR-R2-lemma*:
assumes *P is R2 Q is R2 M is R2m*
shows ((\neg_r *P*) \parallel_M *Q*) ;; *R1(true_h) is R2*

proof –
have (\neg_r *P*) \parallel_M *Q is R2*
by (*simp add: closure assms*)
thus ?thesis
by (*simp add: closure*)

qed

lemma *wrR-R2 [closure]*:
assumes *P is R2 Q is R2 M is R2m*
shows *Q wr_R(M) P is R2*

proof –
have ((\neg_r *P*) \parallel_M *Q*) ;; *R1(true_h) is R2*
by (*simp add: wrR-R2-lemma assms*)
thus ?thesis
by (*simp add: wrR-def wrR-R2-lemma par-by-merge-seq-add closure*)

qed

lemma *wrR-RR [closure]*:
assumes *P is RR Q is RR M is RDM*
shows *Q wr_R(M) P is RR*
apply (*rule RR-intro*)
apply (*simp-all add: unrest assms closure wrR-def rpred*)
apply (*metis (no-types, lifting) Healthy-def' R1-R2c-commute R1-R2c-is-R2 R1-rea-not RDM-R2m*
RR-implies-R2 assms(1) assms(2) assms(3) par-by-merge-seq-add rea-not-R2-closed
wrR-R2-lemma)

done

lemma *wrR-RC [closure]*:
assumes *P is RR Q is RR M is RDM*
shows (*Q wr_R(M) P*) *is RC*
apply (*rule RC-intro*)
apply (*simp add: closure assms*)
apply (*simp add: wrR-def rpred closure assms*)
apply (*simp add: par-by-merge-def seqr-assoc*)

done

lemma *wppR-choice [wp]*: (*P* \vee *Q*) *wr_R(M) R* = (*P wr_R(M) R* \wedge *Q wr_R(M) R*)

proof –

have (*P* \vee *Q*) *wr_R(M) R* =
(\neg_r ((\neg_r *R*) ;; *U0* \wedge (*P* ;; *U1* \vee *Q* ;; *U1*) \wedge $\$v_{<}' =_u \v) ;; *M* ;; *true_r*)
by (*simp add: wrR-def par-by-merge-def seqr-or-distl*)
also have ... = (\neg_r ((\neg_r *R*) ;; *U0* \wedge *P* ;; *U1* \wedge $\$v_{<}' =_u \$v \vee$ (\neg_r *R*) ;; *U0* \wedge *Q* ;; *U1* \wedge $\$v_{<}' =_u$

$\$v) ;; M ;; true_r)$
 by (simp add: conj-disj-distr utp-pred-laws.inf-sup-distrib2)
 also have ... = $(\neg_r (((\neg_r R) ;; U0 \wedge P ;; U1 \wedge \$v_{<} =_u \$v) ;; M ;; true_r \vee$
 $((\neg_r R) ;; U0 \wedge Q ;; U1 \wedge \$v_{<} =_u \$v) ;; M ;; true_r))$
 by (simp add: segr-or-distl)
 also have ... = $(P wr_R(M) R \wedge Q wr_R(M) R)$
 by (simp add: wrR-def par-by-merge-def)
 finally show ?thesis .
 qed

lemma wppR-miracle [wp]: $false wr_R(M) P = true_r$
 by (simp add: wrR-def)

lemma wppR-true [wp]: $P wr_R(M) true_r = true_r$
 by (simp add: wrR-def)

lemma parallel-precondition-wr [rdes]:
 assumes P is NSRD Q is NSRD M is RDM
 shows $pre_R(P \parallel_{M_R(M)} Q) = (peri_R(Q) wr_R(M) pre_R(P) \wedge post_R(Q) wr_R(M) pre_R(P) \wedge$
 $peri_R(P) wr_R(swap_m ;; M) pre_R(Q) \wedge post_R(P) wr_R(swap_m ;; M) pre_R(Q))$
 by (simp add: assms parallel-precondition wrR-def)

lemma parallel-rdes-def [rdes-def]:
 assumes P_1 is RC P_2 is RR P_3 is RR Q_1 is RC Q_2 is RR Q_3 is RR
 $\$st' \# P_2 \$st' \# Q_2$
 M is RDM
 shows $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) \parallel_{M_R(M)} \mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3) =$
 $\mathbf{R}_s(((Q_1 \Rightarrow_r Q_2) wr_R(M) P_1 \wedge (Q_1 \Rightarrow_r Q_3) wr_R(M) P_1 \wedge$
 $(P_1 \Rightarrow_r P_2) wr_R(swap_m ;; M) Q_1 \wedge (P_1 \Rightarrow_r P_3) wr_R(swap_m ;; M) Q_1) \vdash$
 $((P_1 \Rightarrow_r P_2) \parallel_{\exists \$st'} M (Q_1 \Rightarrow_r Q_2) \vee$
 $(P_1 \Rightarrow_r P_3) \parallel_{\exists \$st'} M (Q_1 \Rightarrow_r Q_2) \vee (P_1 \Rightarrow_r P_2) \parallel_{\exists \$st'} M (Q_1 \Rightarrow_r Q_3)) \diamond$
 $((P_1 \Rightarrow_r P_3) \parallel_M (Q_1 \Rightarrow_r Q_3)))$ (is ?lhs = ?rhs)

proof –
 have ?lhs = $\mathbf{R}_s(pre_R ?lhs \vdash peri_R ?lhs \diamond post_R ?lhs)$
 by (simp add: SRD-reactive-tri-design assms closure)
 also have ... = ?rhs
 by (simp add: rdes closure unrest assms, rel-auto)
 finally show ?thesis .
 qed

lemma Miracle-parallel-left-zero:
 assumes P is SRD M is RDM
 shows $Miracle \parallel_{RM} P = Miracle$

proof –
 have $pre_R(Miracle \parallel_{RM} P) = true_r$
 by (simp add: parallel-assm wait'-cond-idem rdes closure assms)
moreover hence $cmt_R(Miracle \parallel_{RM} P) = false$
 by (simp add: rdes closure wait'-cond-idem SRD-healths assms)
ultimately have $Miracle \parallel_{RM} P = \mathbf{R}_s(true_r \vdash false)$
 by (metis NSRD-iff SRD-reactive-design-alt assms par-rdes-NSRD srdes-theory-continuous.weak.top-closed)
 thus ?thesis
 by (simp add: Miracle-def R1-design-R1-pre)
 qed

lemma Miracle-parallel-right-zero:

assumes P is SRD M is RDM
shows $P \parallel_{RM} \text{Miracle} = \text{Miracle}$
proof –
 have $\text{pre}_R(P \parallel_{RM} \text{Miracle}) = \text{true}_r$
 by (simp add: wait'-cond-idem parallel-assm rdes closure assms)
moreover hence $\text{cmt}_R(P \parallel_{RM} \text{Miracle}) = \text{false}$
 by (simp add: wait'-cond-idem rdes closure SRD-healths assms)
ultimately have $P \parallel_{RM} \text{Miracle} = \mathbf{R}_s(\text{true}_r \vdash \text{false})$
 by (metis NSRD-iff SRD-reactive-design-alt assms par-rdes-NSRD srdes-theory-continuous.weak.top-closed)
thus ?thesis
 by (simp add: Miracle-def R1-design-R1-pre)
qed

8.1 Example basic merge

definition $\text{BasicMerge} :: ('s, 't::\text{trace}, \text{unit}) \text{rsp} \text{ merge } (N_B)$ **where**
 $[\text{upred-defs}]: \text{BasicMerge} = (\$tr_{<} \leq_u \$tr' \wedge \$tr' - \$tr_{<} =_u \$0 - tr - \$tr_{<} \wedge \$tr' - \$tr_{<} =_u \$1 - tr - \$tr_{<} \wedge \$st' =_u \$st_{<})$

abbreviation $\text{rbasic-par } (- \parallel_B - [85,86] \ 85)$ **where**
 $P \parallel_B Q \equiv P \parallel_{M_R(N_B)} Q$

lemma BasicMerge-RDM [closure]: N_B is RDM
 by (rule RDM-intro, (rel-auto)+)

lemma $\text{BasicMerge-SymMerge}$ [closure]:
 N_B is SymMerge
 by (rel-auto)

lemma $\text{BasicMerge}'\text{-calc}$:
assumes $\$ok' \# P \ \$wait' \# P \ \$ok' \# Q \ \$wait' \# Q$ P is R2 Q is R2
shows $P \parallel_{N_B} Q = ((\exists \$st' \cdot P) \wedge (\exists \$st' \cdot Q) \wedge \$st' =_u \$st)$
using assms

proof –
 have $P: (\exists \{\$ok', \$wait'\} \cdot R2(P)) = P$ (is ?P' = -)
 by (simp add: ex-unrest ex-plus Healthy-if assms)
 have $Q: (\exists \{\$ok', \$wait'\} \cdot R2(Q)) = Q$ (is ?Q' = -)
 by (simp add: ex-unrest ex-plus Healthy-if assms)
 have $?P' \parallel_{N_B} ?Q' = ((\exists \$st' \cdot ?P') \wedge (\exists \$st' \cdot ?Q') \wedge \$st' =_u \$st)$
 by (simp add: par-by-merge-alt-def, rel-auto, blast+)
thus ?thesis
 by (simp add: P Q)
qed

8.2 Simple parallel composition

definition $\text{rea-design-par} ::$
 $('s, 't::\text{trace}, 'a) \text{hrel-rsp} \Rightarrow ('s, 't, 'a) \text{hrel-rsp} \Rightarrow ('s, 't, 'a) \text{hrel-rsp}$ (**infixr** \parallel_R 85)
where $[\text{upred-defs}]: P \parallel_R Q = \mathbf{R}_s((\text{pre}_R(P) \wedge \text{pre}_R(Q)) \vdash (\text{cmt}_R(P) \wedge \text{cmt}_R(Q)))$

lemma RHS-design-par :
assumes
 $\$ok' \# P_1 \ \$ok' \# P_2$
shows $\mathbf{R}_s(P_1 \vdash Q_1) \parallel_R \mathbf{R}_s(P_2 \vdash Q_2) = \mathbf{R}_s((P_1 \wedge P_2) \vdash (Q_1 \wedge Q_2))$
proof –
 have $\mathbf{R}_s(P_1 \vdash Q_1) \parallel_R \mathbf{R}_s(P_2 \vdash Q_2) =$

```

     $\mathbf{R}_s(P_1 \llbracket true, false / \$ok, \$wait \rrbracket \vdash Q_1 \llbracket true, false / \$ok, \$wait \rrbracket) \parallel_R \mathbf{R}_s(P_2 \llbracket true, false / \$ok, \$wait \rrbracket \vdash Q_2 \llbracket true, false / \$ok, \$wait \rrbracket)$ 
    by (simp add: RHS-design-ok-wait)

  also from assms
  have ... =
     $\mathbf{R}_s((R1 (R2c (P_1)) \wedge R1 (R2c (P_2))) \llbracket true, false / \$ok, \$wait \rrbracket \vdash$ 
     $(R1 (R2c (P_1 \Rightarrow Q_1)) \wedge R1 (R2c (P_2 \Rightarrow Q_2))) \llbracket true, false / \$ok, \$wait \rrbracket)$ 
    apply (simp add: rea-design-par-def rea-pre-RHS-design rea-cmt-RHS-design usubst unrest assms)
    apply (rule cong[of  $\mathbf{R}_s \mathbf{R}_s$ ], simp)
    using assms apply (rel-auto)
  done
  also have ... =
     $\mathbf{R}_s((R2c(P_1) \wedge R2c(P_2)) \vdash$ 
     $(R1 (R2s (P_1 \Rightarrow Q_1)) \wedge R1 (R2s (P_2 \Rightarrow Q_2))))$ 
    by (metis (no-types, hide-lams) R1-R2s-R2c R1-conj R1-design-R1-pre RHS-design-ok-wait)
  also have ... =
     $\mathbf{R}_s((P_1 \wedge P_2) \vdash (R1 (R2s (P_1 \Rightarrow Q_1)) \wedge R1 (R2s (P_2 \Rightarrow Q_2))))$ 
    by (simp add: R2c-R3h-commute R2c-and R2c-design R2c-idem R2c-not RHS-def)
  also have ... =  $\mathbf{R}_s((P_1 \wedge P_2) \vdash ((P_1 \Rightarrow Q_1) \wedge (P_2 \Rightarrow Q_2)))$ 
    by (metis (no-types, lifting) R1-conj R2s-conj RHS-design-export-R1 RHS-design-export-R2s)
  also have ... =  $\mathbf{R}_s((P_1 \wedge P_2) \vdash (Q_1 \wedge Q_2))$ 
    by (rule cong[of  $\mathbf{R}_s \mathbf{R}_s$ ], simp, rel-auto)
  finally show ?thesis .
qed

```

lemma *RHS-tri-design-par*:

```

  assumes  $\$ok' \nmid P_1 \ \$ok' \nmid P_2$ 
  shows  $\mathbf{R}_s(P_1 \vdash Q_1 \diamond R_1) \parallel_R \mathbf{R}_s(P_2 \vdash Q_2 \diamond R_2) = \mathbf{R}_s((P_1 \wedge P_2) \vdash (Q_1 \wedge Q_2) \diamond (R_1 \wedge R_2))$ 
  by (simp add: RHS-design-par assms unrest wait'-cond-conj-exchange)

```

lemma *RHS-tri-design-par-RR* [rdes-def]:

```

  assumes  $P_1$  is RR  $P_2$  is RR
  shows  $\mathbf{R}_s(P_1 \vdash Q_1 \diamond R_1) \parallel_R \mathbf{R}_s(P_2 \vdash Q_2 \diamond R_2) = \mathbf{R}_s((P_1 \wedge P_2) \vdash (Q_1 \wedge Q_2) \diamond (R_1 \wedge R_2))$ 
  by (simp add: RHS-tri-design-par unrest assms)

```

lemma *RHS-comp-assoc*:

```

  assumes  $P$  is NSRD  $Q$  is NSRD  $R$  is NSRD
  shows  $(P \parallel_R Q) \parallel_R R = P \parallel_R Q \parallel_R R$ 
  by (rdes-eq cls: assms)

```

end

9 Productive Reactive Designs

```

theory utp-rdes-productive
  imports utp-rdes-parallel
begin

```

9.1 Healthiness condition

A reactive design is productive if it strictly increases the trace, whenever it terminates. If it does not terminate, it is also classed as productive.

definition *Productive* :: $(s, t :: \text{trace}, \alpha)$ hrel-rsp $\Rightarrow (s, t, \alpha)$ hrel-rsp **where**

[upred-defs]: $Productive(P) = P \parallel_R \mathbf{R}_s(true \vdash true \diamond (\$tr <_u \$tr'))$

lemma *Productive-RHS-design-form:*

assumes $\$ok' \# P \ \$ok' \# Q \ \$ok' \# R$
shows $Productive(\mathbf{R}_s(P \vdash Q \diamond R)) = \mathbf{R}_s(P \vdash Q \diamond (R \wedge \$tr <_u \$tr'))$
using *assms by (simp add: Productive-def RHS-tri-design-par unrest)*

lemma *Productive-form:*

$Productive(SRD(P)) = \mathbf{R}_s(pre_R(P) \vdash peri_R(P) \diamond (post_R(P) \wedge \$tr <_u \$tr'))$

proof –

have $Productive(SRD(P)) = \mathbf{R}_s(pre_R(P) \vdash peri_R(P) \diamond post_R(P)) \parallel_R \mathbf{R}_s(true \vdash true \diamond (\$tr <_u \$tr'))$
by (*simp add: Productive-def SRD-as-reactive-tri-design*)
also have $\dots = \mathbf{R}_s(pre_R(P) \vdash peri_R(P) \diamond (post_R(P) \wedge \$tr <_u \$tr'))$
by (*simp add: RHS-tri-design-par unrest*)
finally show *?thesis* .

qed

A reactive design is productive provided that the postcondition, under the precondition, strictly increases the trace.

lemma *Productive-intro:*

assumes $P \text{ is } SRD \ (\$tr <_u \$tr') \sqsubseteq (pre_R(P) \wedge post_R(P)) \ \$wait' \# pre_R(P)$
shows $P \text{ is } Productive$

proof –

have $P : \mathbf{R}_s(pre_R(P) \vdash peri_R(P) \diamond (post_R(P) \wedge \$tr <_u \$tr')) = P$

proof –

have $\mathbf{R}_s(pre_R(P) \vdash peri_R(P) \diamond post_R(P)) = \mathbf{R}_s(pre_R(P) \vdash (pre_R(P) \wedge peri_R(P)) \diamond (pre_R(P) \wedge post_R(P)))$
by (*metis (no-types, hide-lams) design-export-pre wait'-cond-conj-exchange wait'-cond-idem*)
also have $\dots = \mathbf{R}_s(pre_R(P) \vdash (pre_R(P) \wedge peri_R(P)) \diamond (pre_R(P) \wedge (post_R(P) \wedge \$tr <_u \$tr')))$
by (*metis assms(2) utp-pred-laws.inf.absorb1 utp-pred-laws.inf.assoc*)
also have $\dots = \mathbf{R}_s(pre_R(P) \vdash peri_R(P) \diamond (post_R(P) \wedge \$tr <_u \$tr'))$
by (*metis (no-types, hide-lams) design-export-pre wait'-cond-conj-exchange wait'-cond-idem*)
finally show *?thesis*
by (*simp add: SRD-reactive-tri-design assms(1)*)

qed

thus *?thesis*

by (*metis Healthy-def RHS-tri-design-par Productive-def ok'-pre-unrest unrest-true utp-pred-laws.inf-right-idem utp-pred-laws.inf-top-right*)

qed

lemma *Productive-post-refines-tr-increase:*

assumes $P \text{ is } SRD \ P \text{ is } Productive \ \$wait' \# pre_R(P)$
shows $(\$tr <_u \$tr') \sqsubseteq (pre_R(P) \wedge post_R(P))$

proof –

have $post_R(P) = post_R(\mathbf{R}_s(pre_R(P) \vdash peri_R(P) \diamond (post_R(P) \wedge \$tr <_u \$tr')))$
by (*metis Healthy-def Productive-form assms(1) assms(2)*)
also have $\dots = R1(R2c(pre_R(P) \Rightarrow (post_R(P) \wedge \$tr <_u \$tr')))$
by (*simp add: rea-post-RHS-design unrest usubst assms, rel-auto*)
also have $\dots = R1((pre_R(P) \Rightarrow (post_R(P) \wedge \$tr <_u \$tr')))$
by (*simp add: R2c-impl R2c-preR R2c-postR R2c-and R2c-tr-less-tr' assms*)
also have $(\$tr <_u \$tr') \sqsubseteq (pre_R(P) \wedge \dots)$
by (*rel-auto*)

finally show *?thesis* .

qed

lemma *Continuous-Productive* [closure]: *Continuous Productive*
 by (simp add: Continuous-def Productive-def, rel-auto)

9.2 Reactive design calculations

lemma *preR-Productive* [rdes]:
 assumes *P is SRD*
 shows $\text{pre}_R(\text{Productive}(P)) = \text{pre}_R(P)$
proof –
 have $\text{pre}_R(\text{Productive}(P)) = \text{pre}_R(\mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \diamond (\text{post}_R(P) \wedge \$tr <_u \$tr')))$
 by (metis Healthy-def Productive-form assms)
 thus ?thesis
 by (simp add: rea-pre-RHS-design usubst unrest R2c-not R2c-preR R1-preR Healthy-if assms)
qed

lemma *periR-Productive* [rdes]:
 assumes *P is NSRD*
 shows $\text{peri}_R(\text{Productive}(P)) = \text{peri}_R(P)$
proof –
 have $\text{peri}_R(\text{Productive}(P)) = \text{peri}_R(\mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \diamond (\text{post}_R(P) \wedge \$tr <_u \$tr')))$
 by (metis Healthy-def NSRD-is-SRD Productive-form assms)
 also have $\dots = R1 (R2c (\text{pre}_R P \Rightarrow_r \text{peri}_R P))$
 by (simp add: rea-peri-RHS-design usubst unrest R2c-not assms closure)
 also have $\dots = (\text{pre}_R P \Rightarrow_r \text{peri}_R P)$
 by (simp add: R1-rea-impl R2c-rea-impl R2c-preR R2c-peri-SRD
 R1-peri-SRD assms closure R1-tr-less-tr' R2c-tr-less-tr')
 finally show ?thesis
 by (simp add: SRD-peri-under-pre assms unrest closure)
qed

lemma *postR-Productive* [rdes]:
 assumes *P is NSRD*
 shows $\text{post}_R(\text{Productive}(P)) = (\text{pre}_R(P) \Rightarrow_r \text{post}_R(P) \wedge \$tr <_u \$tr')$
proof –
 have $\text{post}_R(\text{Productive}(P)) = \text{post}_R(\mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \diamond (\text{post}_R(P) \wedge \$tr <_u \$tr')))$
 by (metis Healthy-def NSRD-is-SRD Productive-form assms)
 also have $\dots = R1 (R2c (\text{pre}_R P \Rightarrow_r \text{post}_R P \wedge \$tr' >_u \$tr))$
 by (simp add: rea-post-RHS-design usubst unrest assms closure)
 also have $\dots = (\text{pre}_R P \Rightarrow_r \text{post}_R P \wedge \$tr' >_u \$tr)$
 by (simp add: R1-rea-impl R2c-rea-impl R2c-preR R2c-and R1-extend-conj' R2c-post-SRD
 R1-post-SRD assms closure R1-tr-less-tr' R2c-tr-less-tr')
 finally show ?thesis .
qed

lemma *preR-frame-seq-export*:
 assumes *P is NSRD P is Productive Q is NSRD*
 shows $(\text{pre}_R P \wedge (\text{pre}_R P \wedge \text{post}_R P) ;; Q) = (\text{pre}_R P \wedge (\text{post}_R P ;; Q))$
proof –
 have $(\text{pre}_R P \wedge (\text{post}_R P ;; Q)) = (\text{pre}_R P \wedge ((\text{pre}_R P \Rightarrow_r \text{post}_R P) ;; Q))$
 by (simp add: SRD-post-under-pre assms closure unrest)
 also have $\dots = (\text{pre}_R P \wedge (((\neg_r \text{pre}_R P) ;; Q \vee (\text{pre}_R P \Rightarrow_r R1(\text{post}_R P)) ;; Q)))$
 by (simp add: NSRD-is-SRD R1-post-SRD assms(1) rea-impl-def seqr-or-distl R1-preR Healthy-if)
 also have $\dots = (\text{pre}_R P \wedge (((\neg_r \text{pre}_R P) ;; Q \vee (\text{pre}_R P \wedge \text{post}_R P) ;; Q)))$
proof –
 have $(\text{pre}_R P \vee \neg_r \text{pre}_R P) = R1 \text{ true}$
 by (simp add: R1-preR rea-not-or)

```

then show ?thesis
  by (metis (no-types, lifting) R1-def conj-comm disj-comm disj-conj-distr rea-impl-def seqr-or-distl
    utp-pred-laws.inf-top-left utp-pred-laws.sup.left-idem)
qed
also have ... = (preR P ∧ (((¬r preR P) ∨ (preR P ∧ postR P) ;; Q)))
  by (simp add: NSRD-neg-pre-left-zero assms closure SRD-healths)
also have ... = (preR P ∧ (preR P ∧ postR P) ;; Q)
  by (rel-blast)
finally show ?thesis ..
qed

```

9.3 Closure laws

lemma *Productive-rdes-intro*:

```

assumes ($tr <_u $tr') ⊆ R $ok' # P $ok' # Q $ok' # R $wait # P $wait' # P
shows (Rs(P ⊢ Q ◊ R)) is Productive
proof (rule Productive-intro)
show Rs (P ⊢ Q ◊ R) is SRD
  by (simp add: RHS-tri-design-is-SRD assms)

from assms(1) show ($tr' >_u $tr) ⊆ (preR (Rs (P ⊢ Q ◊ R)) ∧ postR (Rs (P ⊢ Q ◊ R)))
apply (simp add: rea-pre-RHS-design rea-post-RHS-design usubst assms unrest)
using assms(1) apply (rel-auto)
apply fastforce
done

show $wait' # preR (Rs (P ⊢ Q ◊ R))
  by (simp add: rea-pre-RHS-design rea-post-RHS-design usubst R1-def R2c-def R2s-def assms unrest)
qed

```

We use the *R4* healthiness condition to characterise that the postcondition must extend the trace for a reactive design to be productive.

lemma *Productive-rdes-RR-intro*:

```

assumes P is RR Q is RR R is RR R is R4
shows (Rs(P ⊢ Q ◊ R)) is Productive
using assms by (simp add: Productive-rdes-intro R4-iff-refine unrest)

```

lemma *Productive-Miracle [closure]: Miracle is Productive*

```

unfolding Miracle-tri-def Healthy-def
by (subst Productive-RHS-design-form, simp-all add: unrest)

```

lemma *Productive-Chaos [closure]: Chaos is Productive*

```

unfolding Chaos-tri-def Healthy-def
by (subst Productive-RHS-design-form, simp-all add: unrest)

```

lemma *Productive-intChoice [closure]*:

```

assumes P is SRD P is Productive Q is SRD Q is Productive
shows P ⊓ Q is Productive
proof –
  have P ⊓ Q =
    Rs(preR(P) ⊢ periR(P) ◊ (postR(P) ∧ $tr <_u $tr')) ⊓ Rs(preR(Q) ⊢ periR(Q) ◊ (postR(Q) ∧
    $tr <_u $tr'))
  by (metis Healthy-if Productive-form assms)
  also have ... = Rs ((preR P ∧ preR Q) ⊢ (periR P ∨ periR Q) ◊ ((postR P ∧ $tr' >_u $tr) ∨ (postR
  Q ∧ $tr' >_u $tr)))

```

by (simp add: RHS-tri-design-choice)
 also have ... = $\mathbf{R}_s((pre_R P \wedge pre_R Q) \vdash (peri_R P \vee peri_R Q) \diamond (((post_R P) \vee (post_R Q)) \wedge \$tr' >_u \$tr))$
 by (rule cong[of $\mathbf{R}_s \mathbf{R}_s$], simp, rel-auto)
 also have ... is Productive
 by (simp add: Healthy-def Productive-RHS-design-form unrest)
 finally show ?thesis .
 qed

lemma *Productive-cond-rea* [closure]:

assumes *P is SRD P is Productive Q is SRD Q is Productive*

shows $P \triangleleft b \triangleright_R Q$ is Productive

proof –

have $P \triangleleft b \triangleright_R Q =$

$\mathbf{R}_s(pre_R(P) \vdash peri_R(P) \diamond (post_R(P) \wedge \$tr <_u \$tr')) \triangleleft b \triangleright_R \mathbf{R}_s(pre_R(Q) \vdash peri_R(Q) \diamond (post_R(Q) \wedge \$tr <_u \$tr'))$

by (metis Healthy-if Productive-form assms)

also have ... = $\mathbf{R}_s((pre_R P \triangleleft b \triangleright_R pre_R Q) \vdash (peri_R P \triangleleft b \triangleright_R peri_R Q) \diamond ((post_R P \wedge \$tr' >_u \$tr) \triangleleft b \triangleright_R (post_R Q \wedge \$tr' >_u \$tr)))$

by (simp add: cond-srea-form)

also have ... = $\mathbf{R}_s((pre_R P \triangleleft b \triangleright_R pre_R Q) \vdash (peri_R P \triangleleft b \triangleright_R peri_R Q) \diamond (((post_R P) \triangleleft b \triangleright_R (post_R Q)) \wedge \$tr' >_u \$tr))$

by (rule cong[of $\mathbf{R}_s \mathbf{R}_s$], simp, rel-auto)

also have ... is Productive

by (simp add: Healthy-def Productive-RHS-design-form unrest)

finally show ?thesis .

qed

lemma *Productive-seq-1* [closure]:

assumes *P is NSRD P is Productive Q is NSRD*

shows $P ;; Q$ is Productive

proof –

have $P ;; Q = \mathbf{R}_s(pre_R(P) \vdash peri_R(P) \diamond (post_R(P) \wedge \$tr <_u \$tr')) ;; \mathbf{R}_s(pre_R(Q) \vdash peri_R(Q) \diamond (post_R(Q)))$

by (metis Healthy-def NSRD-is-SRD SRD-reactive-tri-design Productive-form assms(1) assms(2) assms(3))

also have ... = $\mathbf{R}_s((pre_R P \wedge (post_R P \wedge \$tr' >_u \$tr) wp_r pre_R Q) \vdash (peri_R P \vee ((post_R P \wedge \$tr' >_u \$tr) ;; peri_R Q)) \diamond ((post_R P \wedge \$tr' >_u \$tr) ;; post_R Q))$

by (simp add: RHS-tri-design-composition-wp rpred unrest closure assms wp NSRD-neg-pre-left-zero SRD-healths ex-unrest wp-rea-def disj-upred-def)

also have ... = $\mathbf{R}_s((pre_R P \wedge (post_R P \wedge \$tr' >_u \$tr) wp_r pre_R Q) \vdash (peri_R P \vee ((post_R P \wedge \$tr' >_u \$tr) ;; peri_R Q)) \diamond ((post_R P \wedge \$tr' >_u \$tr) ;; post_R Q \wedge \$tr' >_u \$tr))$

proof –

have $((post_R P \wedge \$tr' >_u \$tr) ;; R1(post_R Q)) = ((post_R P \wedge \$tr' >_u \$tr) ;; R1(post_R Q) \wedge \$tr' >_u \$tr)$

by (rel-auto)

thus ?thesis

by (simp add: NSRD-is-SRD R1-post-SRD assms)

qed

also have ... is Productive

by (rule Productive-rdes-intro, simp-all add: unrest assms closure wp-rea-def)

finally show ?thesis .

qed

lemma *Productive-seq-2 [closure]*:
assumes *P is NSRD Q is NSRD Q is Productive*
shows *P ;; Q is Productive*
proof –
have $P ;; Q = \mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \diamond (\text{post}_R(P))) ;; \mathbf{R}_s(\text{pre}_R(Q) \vdash \text{peri}_R(Q) \diamond (\text{post}_R(Q) \wedge \$tr <_u \$tr'))$
by (*metis Healthy-def NSRD-is-SRD SRD-reactive-tri-design Productive-form assms*)
also have $\dots = \mathbf{R}_s((\text{pre}_R P \wedge \text{post}_R P \text{ wp}_r \text{pre}_R Q) \vdash (\text{peri}_R P \vee (\text{post}_R P ;; \text{peri}_R Q)) \diamond (\text{post}_R P ;; (\text{post}_R Q \wedge \$tr' >_u \$tr)))$
by (*simp add: RHS-tri-design-composition-wp rpred unrest closure assms wp NSRD-neg-pre-left-zero SRD-healths ex-unrest wp-rea-def disj-upred-def*)
also have $\dots = \mathbf{R}_s((\text{pre}_R P \wedge \text{post}_R P \text{ wp}_r \text{pre}_R Q) \vdash (\text{peri}_R P \vee (\text{post}_R P ;; \text{peri}_R Q)) \diamond (\text{post}_R P ;; (\text{post}_R Q \wedge \$tr' >_u \$tr) \wedge \$tr' >_u \$tr))$
proof –
have $(R1(\text{post}_R P) ;; (\text{post}_R Q \wedge \$tr' >_u \$tr) \wedge \$tr' >_u \$tr) = (R1(\text{post}_R P) ;; (\text{post}_R Q \wedge \$tr' >_u \$tr))$
by (*rel-auto*)
thus *?thesis*
by (*simp add: NSRD-is-SRD R1-post-SRD assms*)
qed
also have \dots *is Productive*
by (*rule Productive-rdes-intro, simp-all add: unrest assms closure wp-rea-def*)
finally show *?thesis* .
qed
end

10 Guarded Recursion

theory *utp-rdes-guarded*
imports *utp-rdes-productive*
begin

10.1 Traces with a size measure

Guarded recursion relies on our ability to measure the trace's size, in order to see if it is decreasing on each iteration. Thus, we here equip the trace algebra with the *ucard* function that provides this.

class *size-trace* = *trace* + *size* +
assumes

size-zero: $\text{size } 0 = 0$ **and**
size-nzero: $s > 0 \implies \text{size}(s) > 0$ **and**
size-plus: $\text{size } (s + t) = \text{size}(s) + \text{size}(t)$

— These axioms may be stronger than necessary. In particular, $0 < ?s \implies 0 < \#_u(?s)$ requires that a non-empty trace have a positive size. But this may not be the case with all trace models and is possibly more restrictive than necessary. In future we will explore weakening.

begin

lemma *size-mono*: $s \leq t \implies \text{size}(s) \leq \text{size}(t)$
by (*metis le-add1 local.diff-add-cancel-left' local.size-plus*)

lemma *size-strict-mono*: $s < t \implies \text{size}(s) < \text{size}(t)$

by (metis cancel-ab-semigroup-add-class.add-diff-cancel-left' local.diff-add-cancel-left' local.less-iff local.minus-gr-zero-iff local.size-nzero local.size-plus zero-less-diff)

lemma trace-strict-prefixE: $xs < ys \implies (\bigwedge zs. \llbracket ys = xs + zs; \text{size}(zs) > 0 \rrbracket \implies \text{thesis}) \implies \text{thesis}$
 by (metis local.diff-add-cancel-left' local.less-iff local.minus-gr-zero-iff local.size-nzero)

lemma size-minus-trace: $y \leq x \implies \text{size}(x - y) = \text{size}(x) - \text{size}(y)$
 by (metis diff-add-inverse local.diff-add-cancel-left' local.size-plus)

end

Both natural numbers and lists are measurable trace algebras.

instance nat :: size-trace
 by (intro-classes, simp-all)

instance list :: (type) size-trace
 by (intro-classes, simp-all add: zero-list-def less-list-def' plus-list-def prefix-length-less)

syntax
 -usize :: logic \Rightarrow logic (size_u'(-'))

translations
 size_u(t) == CONST uop CONST size t

10.2 Guardedness

definition gvirt :: (('t::size-trace, 'α) rp × ('t, 'α) rp) chain **where**
 [upred-defs]: gvirt(n) ≡ (\$tr ≤_u \$tr' ∧ size_u(&tt) <_u <n>)

lemma gvirt-chain: chain gvirt
 apply (simp add: chain-def, safe)
 apply (rel-simp)
 apply (rel-simp)+
 done

lemma gvirt-limit: $\sqcap (\text{range gvirt}) = (\$tr \leq_u \$tr')$
 by (rel-auto)

definition Guarded :: (('t::size-trace, 'α) hrel-rp \Rightarrow ('t, 'α) hrel-rp) \Rightarrow bool **where**
 [upred-defs]: Guarded(F) = (∀ X n. (F(X) ∧ gvirt(n+1)) = (F(X ∧ gvirt(n)) ∧ gvirt(n+1)))

lemma GuardedI: $\llbracket \bigwedge X n. (F(X) \wedge \text{gvirt}(n+1)) = (F(X \wedge \text{gvirt}(n)) \wedge \text{gvirt}(n+1)) \rrbracket \implies \text{Guarded } F$
 by (simp add: Guarded-def)

Guarded reactive designs yield unique fixed-points.

theorem guarded-fp-uniq:
 assumes mono F F ∈ $\llbracket id \rrbracket_H \rightarrow \llbracket SRD \rrbracket_H$ Guarded F
 shows $\mu F = \nu F$
proof –
 have constr F gvirt
 using assms
 by (auto simp add: constr-def gvirt-chain Guarded-def tcontr-alt-def')
 hence (\$tr ≤_u \$tr' ∧ μF) = (\$tr ≤_u \$tr' ∧ νF)
 apply (rule constr-fp-uniq)
 apply (simp add: assms)

```

    using gvirt-limit apply blast
  done
moreover have ( $\$tr \leq_u \$tr' \wedge \mu F$ ) =  $\mu F$ 
proof -
  have  $\mu F$  is R1
    by (rule SRD-healths(1), rule Healthy-mu, simp-all add: assms)
  thus ?thesis
    by (metis Healthy-def R1-def conj-comm)
qed
moreover have ( $\$tr \leq_u \$tr' \wedge \nu F$ ) =  $\nu F$ 
proof -
  have  $\nu F$  is R1
    by (rule SRD-healths(1), rule Healthy-nu, simp-all add: assms)
  thus ?thesis
    by (metis Healthy-def R1-def conj-comm)
qed
ultimately show ?thesis
  by (simp)
qed

lemma Guarded-const [closure]: Guarded ( $\lambda X. P$ )
  by (simp add: Guarded-def)

lemma UINF-Guarded [closure]:
  assumes  $\bigwedge P. P \in A \implies \text{Guarded } P$ 
  shows Guarded ( $\lambda X. \bigwedge P \in A. P(X)$ )
proof (rule GuardedI)
  fix  $X n$ 
  have  $\bigwedge Y. ((\bigwedge P \in A. P Y) \wedge \text{gvrt}(n+1)) = ((\bigwedge P \in A. (P Y \wedge \text{gvrt}(n+1))) \wedge \text{gvrt}(n+1))$ 
proof -
  fix  $Y$ 
  let ?lhs =  $((\bigwedge P \in A. P Y) \wedge \text{gvrt}(n+1))$  and ?rhs =  $((\bigwedge P \in A. (P Y \wedge \text{gvrt}(n+1))) \wedge \text{gvrt}(n+1))$ 
  have  $a: ?lhs \llbracket \text{false} / \$ok \rrbracket = ?rhs \llbracket \text{false} / \$ok \rrbracket$ 
    by (rel-auto)
  have  $b: ?lhs \llbracket \text{true} / \$ok \rrbracket \llbracket \text{true} / \$wait \rrbracket = ?rhs \llbracket \text{true} / \$ok \rrbracket \llbracket \text{true} / \$wait \rrbracket$ 
    by (rel-auto)
  have  $c: ?lhs \llbracket \text{true} / \$ok \rrbracket \llbracket \text{false} / \$wait \rrbracket = ?rhs \llbracket \text{true} / \$ok \rrbracket \llbracket \text{false} / \$wait \rrbracket$ 
    by (rel-auto)
  show ?lhs = ?rhs
    using a b c
    by (rule-tac bool-eq-splitI[of in-var ok], simp, rule-tac bool-eq-splitI[of in-var wait], simp-all)
qed
moreover have  $((\bigwedge P \in A. (P X \wedge \text{gvrt}(n+1))) \wedge \text{gvrt}(n+1)) = ((\bigwedge P \in A. (P (X \wedge \text{gvrt}(n))) \wedge \text{gvrt}(n+1))) \wedge \text{gvrt}(n+1))$ 
proof -
  have  $(\bigwedge P \in A. (P X \wedge \text{gvrt}(n+1))) = (\bigwedge P \in A. (P (X \wedge \text{gvrt}(n)) \wedge \text{gvrt}(n+1)))$ 
  proof (rule UINF-cong)
    fix  $P$  assume  $P \in A$ 
    thus  $(P X \wedge \text{gvrt}(n+1)) = (P (X \wedge \text{gvrt}(n)) \wedge \text{gvrt}(n+1))$ 
      using Guarded-def assms by blast
  qed
  thus ?thesis by simp
qed
ultimately show  $((\bigwedge P \in A. P X) \wedge \text{gvrt}(n+1)) = ((\bigwedge P \in A. (P (X \wedge \text{gvrt}(n)))) \wedge \text{gvrt}(n+1))$ 
  by simp

```

qed

lemma *intChoice-Guarded* [closure]:
assumes *Guarded P Guarded Q*
shows *Guarded ($\lambda X. P(X) \sqcap Q(X)$)*

proof –

have *Guarded ($\lambda X. \bigwedge F \in \{P, Q\} \cdot F(X)$)*
by (*rule UINF-Guarded, auto simp add: assms*)
thus *?thesis*
by (*simp*)

qed

lemma *cond-srea-Guarded* [closure]:
assumes *Guarded P Guarded Q*
shows *Guarded ($\lambda X. P(X) \triangleleft b \triangleright_R Q(X)$)*
using *assms* **by** (*rel-auto*)

A tail recursive reactive design with a productive body is guarded.

lemma *Guarded-if-Productive* [closure]:

fixes *P :: ('s, 't::size-trace, 'α) hrel-rsp*
assumes *P is NSRD P is Productive*
shows *Guarded ($\lambda X. P ;; SRD(X)$)*

proof (*clarsimp simp add: Guarded-def*)

— We split the proof into three cases corresponding to valuations for ok, wait, and wait' respectively.

fix *X n*

have *a: (P ;; SRD(X) \wedge gvirt (Suc n)) \llbracket false/\$ok \rrbracket =*
(P ;; SRD(X \wedge gvirt n) \wedge gvirt (Suc n)) \llbracket false/\$ok \rrbracket

by (*simp add: usubst closure SRD-left-zero-1 assms*)

have *b: ((P ;; SRD(X) \wedge gvirt (Suc n)) \llbracket true/\$ok \rrbracket) \llbracket true/\$wait \rrbracket =*
((P ;; SRD(X \wedge gvirt n) \wedge gvirt (Suc n)) \llbracket true/\$ok \rrbracket) \llbracket true/\$wait \rrbracket

by (*simp add: usubst closure SRD-left-zero-2 assms*)

have *c: ((P ;; SRD(X) \wedge gvirt (Suc n)) \llbracket true/\$ok \rrbracket) \llbracket false/\$wait \rrbracket =*
((P ;; SRD(X \wedge gvirt n) \wedge gvirt (Suc n)) \llbracket true/\$ok \rrbracket) \llbracket false/\$wait \rrbracket

proof –

have *1: (P \llbracket true/\$wait' \rrbracket ;; (SRD X) \llbracket true/\$wait \rrbracket \wedge gvirt (Suc n)) \llbracket true,false/\$ok,\$wait \rrbracket =*
(P \llbracket true/\$wait' \rrbracket ;; (SRD (X \wedge gvirt n)) \llbracket true/\$wait \rrbracket \wedge gvirt (Suc n)) \llbracket true,false/\$ok,\$wait \rrbracket

by (*metis (no-types, lifting) Healthy-def R3h-wait-true SRD-healths(3) SRD-idem*)

have *2: (P \llbracket false/\$wait' \rrbracket ;; (SRD X) \llbracket false/\$wait \rrbracket \wedge gvirt (Suc n)) \llbracket true,false/\$ok,\$wait \rrbracket =*
(P \llbracket false/\$wait' \rrbracket ;; (SRD (X \wedge gvirt n)) \llbracket false/\$wait \rrbracket \wedge gvirt (Suc n)) \llbracket true,false/\$ok,\$wait \rrbracket

proof –

have *exp: $\bigwedge Y :: ('s, 't, 'α) hrel-rsp. (P \llbracket false/$wait' \rrbracket ;; (SRD Y) \llbracket false/$wait \rrbracket \wedge gvirt (Suc n)) \llbracket true,false/$ok,$wait $\rrbracket$$*

=

((\neg_r pre_R P) ;; (SRD(Y)) \llbracket false/\$wait \rrbracket \vee (post_R P \wedge \$tr' >_u \$tr) ;; (SRD Y)) \llbracket true,false/\$ok,\$wait \rrbracket)
 \wedge gvirt (Suc n)) \llbracket true,false/\$ok,\$wait \rrbracket

proof –

fix *Y :: ('s, 't, 'α) hrel-rsp*

have *(P \llbracket false/\$wait' \rrbracket ;; (SRD Y) \llbracket false/\$wait \rrbracket \wedge gvirt (Suc n)) \llbracket true,false/\$ok,\$wait \rrbracket =*

((\mathbf{R}_s (pre_R(P) \vdash peri_R(P) \diamond (post_R(P) \wedge \$tr <_u \$tr')) \llbracket false/\$wait' \rrbracket ;; (SRD Y) \llbracket false/\$wait \rrbracket
 \wedge gvirt (Suc n)) \llbracket true,false/\$ok,\$wait \rrbracket

by (*metis (no-types) Healthy-def Productive-form assms(1) assms(2) NSRD-is-SRD*)

also have ... =

((R1(R2c(pre_R(P) \Rightarrow (\$ok' \wedge post_R(P) \wedge \$tr <_u \$tr')))) \llbracket false/\$wait' \rrbracket ;; (SRD Y) \llbracket false/\$wait \rrbracket
 \wedge gvirt (Suc n)) \llbracket true,false/\$ok,\$wait \rrbracket

by (simp add: RHS-def R1-def R2c-def R2s-def R3h-def RD1-def RD2-def usubst unrest assms
 closure design-def)
 also have ... =
 (((\neg_r pre_R(P) \vee (\$ok' \wedge post_R(P) \wedge \$tr <_u \$tr')))[false/\$wait'] ;; (SRD Y)[false/\$wait]
 \wedge gvirt (Suc n)[true,false/\$ok,\$wait]
 by (simp add: impl-alt-def R2c-disj R1-disj R2c-not assms closure R2c-and
 R2c-preR rea-not-def R1-extend-conj' R2c-ok' R2c-post-SRD R1-tr-less-tr' R2c-tr-less-tr')
 also have ... =
 (((\neg_r pre_R P) ;; (SRD(Y))[false/\$wait] \vee (\$ok' \wedge post_R P \wedge \$tr' >_u \$tr) ;; (SRD
 Y)[false/\$wait])) \wedge gvirt (Suc n)[true,false/\$ok,\$wait]
 by (simp add: usubst unrest assms closure seqr-or-distl NSRD-neg-pre-left-zero SRD-healths)
 also have ... =
 (((\neg_r pre_R P) ;; (SRD(Y))[false/\$wait] \vee (post_R P \wedge \$tr' >_u \$tr) ;; (SRD Y)[true,false/\$ok,\$wait]))
 \wedge gvirt (Suc n)[true,false/\$ok,\$wait]
 proof -
 have (\$ok' \wedge post_R P \wedge \$tr' >_u \$tr) ;; (SRD Y)[false/\$wait] =
 ((post_R P \wedge \$tr' >_u \$tr) \wedge \$ok' =_u true) ;; (SRD Y)[false/\$wait]
 by (rel-blast)
 also have ... = (post_R P \wedge \$tr' >_u \$tr)[true/\$ok'] ;; (SRD Y)[false/\$wait][true/\$ok]
 using seqr-left-one-point[of ok (post_R P \wedge \$tr' >_u \$tr) True (SRD Y)[false/\$wait]]
 by (simp add: true-alt-def[THEN sym])
 finally show ?thesis by (simp add: usubst unrest)
 qed
 finally
 show (P[false/\$wait'] ;; (SRD Y)[false/\$wait] \wedge gvirt (Suc n)[true,false/\$ok,\$wait] =
 (((\neg_r pre_R P) ;; (SRD(Y))[false/\$wait] \vee (post_R P \wedge \$tr' >_u \$tr) ;; (SRD
 Y)[true,false/\$ok,\$wait]))
 \wedge gvirt (Suc n)[true,false/\$ok,\$wait] .
 qed
 have 1:((post_R P \wedge \$tr' >_u \$tr) ;; (SRD X)[true,false/\$ok,\$wait] \wedge gvirt (Suc n)) =
 ((post_R P \wedge \$tr' >_u \$tr) ;; (SRD (X \wedge gvirt n))[true,false/\$ok,\$wait] \wedge gvirt (Suc n))
 apply (rel-auto)
 apply (rename-tac tr st more ok wait tr' st' more' tr₀ st₀ more₀ ok')
 apply (rule-tac x=tr₀ in exI, rule-tac x=st₀ in exI, rule-tac x=more₀ in exI)
 apply (simp)
 apply (erule trace-strict-prefixE)
 apply (rename-tac tr st ref ok wait tr' st' ref' tr₀ st₀ ref₀ ok' zs)
 apply (rule-tac x=False in exI)
 apply (simp add: size-minus-trace)
 apply (subgoal-tac size(tr) < size(tr₀))
 apply (simp add: less-diff-conv2 size-mono)
 using size-strict-mono apply blast
 apply (rename-tac tr st more ok wait tr' st' more' tr₀ st₀ more₀ ok')
 apply (rule-tac x=tr₀ in exI, rule-tac x=st₀ in exI, rule-tac x=more₀ in exI)
 apply (simp)
 apply (erule trace-strict-prefixE)
 apply (rename-tac tr st more ok wait tr' st' more' tr₀ st₀ more₀ ok' zs)
 apply (auto simp add: size-minus-trace)
 apply (subgoal-tac size(tr) < size(tr₀))
 apply (simp add: less-diff-conv2 size-mono)
 using size-strict-mono apply blast
 done
 have 2:(\neg_r pre_R P) ;; (SRD X)[false/\$wait] = (\neg_r pre_R P) ;; (SRD(X \wedge gvirt n))[false/\$wait]
 by (simp add: NSRD-neg-pre-left-zero closure assms SRD-healths)


```

show ?thesis
  by (simp add: exp 1 2 utp-pred-laws.inf-sup-distrib2)
qed

show ?thesis
proof –
  have (P ;; (SRD X) ∧ gvirt (n+1))[[true,false/$ok,$wait]] =
    ((P[[true/$wait']] ;; (SRD X)[[true/$wait]] ∧ gvirt (n+1))[[true,false/$ok,$wait]] ∨
     (P[[false/$wait']] ;; (SRD X)[[false/$wait]] ∧ gvirt (n+1))[[true,false/$ok,$wait]])
    by (subst seqr-bool-split[of wait], simp-all add: usubst utp-pred-laws.distrib(4))

  also
  have ... = ((P[[true/$wait']] ;; (SRD (X ∧ gvirt n))[[true/$wait]] ∧ gvirt (n+1))[[true,false/$ok,$wait]]
    ∨
    (P[[false/$wait']] ;; (SRD (X ∧ gvirt n))[[false/$wait]] ∧ gvirt (n+1))[[true,false/$ok,$wait]])
    by (simp add: 1 2)

  also
  have ... = ((P[[true/$wait']] ;; (SRD (X ∧ gvirt n))[[true/$wait]] ∨
    P[[false/$wait']] ;; (SRD (X ∧ gvirt n))[[false/$wait]] ∧ gvirt (n+1))[[true,false/$ok,$wait]]
    by (simp add: usubst utp-pred-laws.distrib(4))

  also have ... = (P ;; (SRD (X ∧ gvirt n) ∧ gvirt (n+1))[[true,false/$ok,$wait]]
    by (subst seqr-bool-split[of wait], simp-all add: usubst)
  finally show ?thesis by (simp add: usubst)
qed

qed
show (P ;; SRD(X) ∧ gvirt (Suc n)) = (P ;; SRD(X ∧ gvirt n) ∧ gvirt (Suc n))
  apply (rule-tac bool-eq-splitI[of in-var ok])
  apply (simp-all add: a)
  apply (rule-tac bool-eq-splitI[of in-var wait])
  apply (simp-all add: b c)
done
qed

```

10.3 Tail recursive fixed-point calculations

```

declare upred-semiring.power-Suc [simp]

lemma mu-csp-form-1 [rdes]:
  fixes P :: ('s, 't::size-trace,'α) hrel-rsp
  assumes P is NSRD P is Productive
  shows (μ X · P ;; SRD(X)) = (⊓ i · P ^ (i+1)) ;; Miracle
proof –
  have 1: Continuous (λX. P ;; SRD X)
    using SRD-Continuous
    by (clarsimp simp add: Continuous-def seq-SUP-distl[THEN sym], drule-tac x=A in spec, simp)
  have 2: (λX. P ;; SRD X) ∈ [[id]]H → [[SRD]]H
    by (blast intro: funcsetI closure assms)
  with 1 2 have (μ X · P ;; SRD(X)) = (ν X · P ;; SRD(X))
    by (simp add: guarded-fp-uniq Guarded-if-Productive[OF assms] funcsetI closure)
  also have ... = (⊓ i. ((λX. P ;; SRD X) ^ i) false)
    by (simp add: sup-continuous-lfp 1 sup-continuous-Continuous false-upred-def)
  also have ... = ((λX. P ;; SRD X) ^ 0) false ⊓ (⊓ i. ((λX. P ;; SRD X) ^ (i+1)) false)
    by (subst Sup-power-expand, simp)

```

```

also have ... = ( $\prod i. ((\lambda X. P ;; SRD X) \hat{\wedge} (i+1)) false$ )
  by (simp)
also have ... = ( $\prod i. P \hat{\wedge} (i+1) ;; Miracle$ )
proof (rule SUP-cong, simp-all)
  fix i
  show  $P ;; SRD ((\lambda X. P ;; SRD X) \hat{\wedge} i) false = (P ;; P \hat{\wedge} i) ;; Miracle$ 
  proof (induct i)
    case 0
    then show ?case
      by (simp, metis srdes-hcond-def srdes-theory-continuous.healthy-top)
  next
    case (Suc i)
    then show ?case
      by (simp add: Healthy-if NSRD-is-SRD SRD-power-comp SRD-seqr-closure assms(1) seqr-assoc[THEN sym] srdes-theory-continuous.weak.top-closed)
  qed
qed
also have ... = ( $\prod i. P \hat{\wedge} (i+1) ;; Miracle$ )
  by (simp add: seq-Sup-distr)
finally show ?thesis
  by (simp add: UINF-as-Sup[THEN sym])
qed

lemma mu-csp-form-NSRD [closure]:
  fixes  $P :: ('s, 't::size-trace, 'α) hrel-rsp$ 
  assumes  $P$  is NSRD  $P$  is Productive
  shows  $(\mu X \cdot P ;; SRD(X))$  is NSRD
  by (simp add: mu-csp-form-1 assms closure)

lemma mu-csp-form-1':
  fixes  $P :: ('s, 't::size-trace, 'α) hrel-rsp$ 
  assumes  $P$  is NSRD  $P$  is Productive
  shows  $(\mu X \cdot P ;; SRD(X)) = (P ;; P^*) ;; Miracle$ 
proof -
  have  $(\mu X \cdot P ;; SRD(X)) = (\prod i \in UNIV \cdot P ;; P \hat{\wedge} i) ;; Miracle$ 
    by (simp add: mu-csp-form-1 assms closure ustar-def)
  also have ... =  $(P ;; P^*) ;; Miracle$ 
    by (simp only: seq-UINF-distl[THEN sym], simp add: ustar-def)
  finally show ?thesis .
qed

declare upred-semiring.power-Suc [simp del]

end

```

11 Reactive Design Programs

```

theory utp-rdes-prog
imports
  utp-rdes-normal
  utp-rdes-tactics
  utp-rdes-parallel
  utp-rdes-guarded
  UTP-KAT.utp-kleene
begin

```

11.1 State substitution

lemma *srd-subst-RHS-tri-design* [*usubst*]:

$\lceil \sigma \rceil_{S\sigma} \dagger \mathbf{R}_s(P \vdash Q \diamond R) = \mathbf{R}_s(\lceil \sigma \rceil_{S\sigma} \dagger P) \vdash (\lceil \sigma \rceil_{S\sigma} \dagger Q) \diamond (\lceil \sigma \rceil_{S\sigma} \dagger R)$
by (*rel-auto*)

lemma *srd-subst-SRD-closed* [*closure*]:

assumes *P is SRD*

shows $\lceil \sigma \rceil_{S\sigma} \dagger P$ *is SRD*

proof –

have $SRD(\lceil \sigma \rceil_{S\sigma} \dagger (SRD P)) = \lceil \sigma \rceil_{S\sigma} \dagger (SRD P)$

by (*rel-auto*)

thus *?thesis*

by (*metis Healthy-def assms*)

qed

lemma *preR-srd-subst* [*rdes*]:

$pre_R(\lceil \sigma \rceil_{S\sigma} \dagger P) = \lceil \sigma \rceil_{S\sigma} \dagger pre_R(P)$

by (*rel-auto*)

lemma *periR-srd-subst* [*rdes*]:

$peri_R(\lceil \sigma \rceil_{S\sigma} \dagger P) = \lceil \sigma \rceil_{S\sigma} \dagger peri_R(P)$

by (*rel-auto*)

lemma *postR-srd-subst* [*rdes*]:

$post_R(\lceil \sigma \rceil_{S\sigma} \dagger P) = \lceil \sigma \rceil_{S\sigma} \dagger post_R(P)$

by (*rel-auto*)

lemma *srd-subst-NSRD-closed* [*closure*]:

assumes *P is NSRD*

shows $\lceil \sigma \rceil_{S\sigma} \dagger P$ *is NSRD*

by (*rule NSRD-RC-intro, simp-all add: closure rdes assms unrest*)

11.2 Assignment

definition *assigns-srd* :: '*s usubst* \Rightarrow ('*s*, '*t::trace*, '*α*) *hrel-rsp* ($\langle \cdot \rangle_R$) **where**

[*upred-defs*]: *assigns-srd* $\sigma = \mathbf{R}_s(true \vdash (\$tr' =_u \$tr \wedge \neg \$wait' \wedge \lceil \langle \sigma \rangle_a \rceil_S \wedge \$\Sigma_S' =_u \$\Sigma_S))$

syntax

-assign-srd :: *svids* \Rightarrow *uexprs* \Rightarrow *logic* (*'(-) :=_R '(-)*)

-assign-srd :: *svids* \Rightarrow *uexprs* \Rightarrow *logic* (**infixr** :=_R 90)

translations

-assign-srd *xs vs* \Rightarrow *CONST assigns-srd* (*-mk-usubst* (*CONST id*) *xs vs*)

-assign-srd *x v* \Leftarrow *CONST assigns-srd* (*CONST subst-upd* (*CONST id*) *x v*)

-assign-srd *x v* \Leftarrow *-assign-srd* (*-spvar* *x*) *v*

x, y :=_R u, v \Leftarrow *CONST assigns-srd* (*CONST subst-upd* (*CONST subst-upd* (*CONST id*) (*CONST svar* *x*) *u*) (*CONST svar* *y*) *v*)

lemma *assigns-srd-RHS-tri-des* [*rdes-def*]:

$\langle \sigma \rangle_R = \mathbf{R}_s(true_r \vdash false \diamond \langle \sigma \rangle_r)$

by (*rel-auto*)

lemma *assigns-srd-NSRD-closed* [*closure*]: $\langle \sigma \rangle_R$ *is NSRD*

by (*simp add: rdes-def closure unrest*)

lemma *preR-assigns-srd* [rdes]: $pre_R(\langle \sigma \rangle_R) = true_r$
 by (simp add: rdes-def rdes closure)

lemma *periR-assigns-srd* [rdes]: $peri_R(\langle \sigma \rangle_R) = false$
 by (simp add: rdes-def rdes closure)

lemma *postR-assigns-srd* [rdes]: $post_R(\langle \sigma \rangle_R) = \langle \sigma \rangle_r$
 by (simp add: rdes-def rdes closure rpred)

11.3 Conditional

lemma *preR-cond-srea* [rdes]:
 $pre_R(P \triangleleft b \triangleright_R Q) = ([b]_{S<} \wedge pre_R(P) \vee [\neg b]_{S<} \wedge pre_R(Q))$
 by (rel-auto)

lemma *periR-cond-srea* [rdes]:
 assumes *P is SRD Q is SRD*
 shows $peri_R(P \triangleleft b \triangleright_R Q) = ([b]_{S<} \wedge peri_R(P) \vee [\neg b]_{S<} \wedge peri_R(Q))$
proof –
 have $peri_R(P \triangleleft b \triangleright_R Q) = peri_R(R1(P) \triangleleft b \triangleright_R R1(Q))$
 by (simp add: Healthy-if SRD-healths assms)
 thus ?thesis
 by (rel-auto)
qed

lemma *postR-cond-srea* [rdes]:
 assumes *P is SRD Q is SRD*
 shows $post_R(P \triangleleft b \triangleright_R Q) = ([b]_{S<} \wedge post_R(P) \vee [\neg b]_{S<} \wedge post_R(Q))$
proof –
 have $post_R(P \triangleleft b \triangleright_R Q) = post_R(R1(P) \triangleleft b \triangleright_R R1(Q))$
 by (simp add: Healthy-if SRD-healths assms)
 thus ?thesis
 by (rel-auto)
qed

lemma *NSRD-cond-srea* [closure]:
 assumes *P is NSRD Q is NSRD*
 shows $P \triangleleft b \triangleright_R Q$ is NSRD
proof (rule NSRD-RC-intro)
 show $P \triangleleft b \triangleright_R Q$ is SRD
 by (simp add: closure assms)
 show $pre_R(P \triangleleft b \triangleright_R Q)$ is RC
proof –
 have $1:([\neg b]_{S<} \vee \neg_r pre_R P) ;; R1(true) = ([\neg b]_{S<} \vee \neg_r pre_R P)$
 by (metis (no-types, lifting) NSRD-neg-pre-unit aext-not assms(1) seqr-or-distl st-lift-R1-true-right)
 have $2:([b]_{S<} \vee \neg_r pre_R Q) ;; R1(true) = ([b]_{S<} \vee \neg_r pre_R Q)$
 by (simp add: NSRD-neg-pre-unit assms seqr-or-distl st-lift-R1-true-right)
 show ?thesis
 by (simp add: rdes closure assms)
qed
 show $\$st' \# peri_R(P \triangleleft b \triangleright_R Q)$
 by (simp add: rdes assms closure unrest)
qed

11.4 Assumptions

definition $AssumeR :: 's \text{ cond} \Rightarrow ('s, 't::trace, 'a) \text{ hrel-rsp } ([\cdot]^\top_R)$ **where**
 $[upred-defs]: AssumeR \ b = II_R \triangleleft b \triangleright_R \text{ Miracle}$

lemma $AssumeR\text{-rdes-def}$ $[rdes-def]$:
 $[b]^\top_R = \mathbf{R}_s(true_r \vdash false \diamond [b]^\top_r)$
unfolding $AssumeR\text{-def}$ **by** $(rdes\text{-eq})$

lemma $AssumeR\text{-NSRD}$ $[closure]$: $[b]^\top_R$ *is NSRD*
by $(simp \text{ add: } AssumeR\text{-def } closure)$

lemma $AssumeR\text{-false}$: $[false]^\top_R = \text{Miracle}$
by $(rel\text{-auto})$

lemma $AssumeR\text{-true}$: $[true]^\top_R = II_R$
by $(rel\text{-auto})$

lemma $AssumeR\text{-comp}$: $[b]^\top_R ;; [c]^\top_R = [b \wedge c]^\top_R$
by $(rdes\text{-simp})$

lemma $AssumeR\text{-choice}$: $[b]^\top_R \sqcap [c]^\top_R = [b \vee c]^\top_R$
by $(rdes\text{-eq})$

lemma $AssumeR\text{-refine-skip}$: $II_R \sqsubseteq [b]^\top_R$
by $(rdes\text{-refine})$

lemma $AssumeR\text{-test}$ $[closure]$: $test_R [b]^\top_R$
by $(simp \text{ add: } AssumeR\text{-refine-skip } nsrd\text{-thy.} utest\text{-intro})$

lemma $Star\text{-AssumeR}$: $[b]^\top_R^{\star R} = II_R$
by $(simp \text{ add: } AssumeR\text{-NSRD } AssumeR\text{-test } nsrd\text{-thy.} Star\text{-test})$

lemma $AssumeR\text{-choice-skip}$: $II_R \sqcap [b]^\top_R = II_R$
by $(rdes\text{-eq})$

lemma $cond\text{-srea-}AssumeR\text{-form}$:
assumes P *is NSRD* Q *is NSRD*
shows $P \triangleleft b \triangleright_R Q = ([b]^\top_R ;; P \sqcap [\neg b]^\top_R ;; Q)$
by $(rdes\text{-eq } cls: \text{assms})$

lemma $cond\text{-srea-insert-assume}$:
assumes P *is NSRD* Q *is NSRD*
shows $P \triangleleft b \triangleright_R Q = ([b]^\top_R ;; P \triangleleft b \triangleright_R [\neg b]^\top_R ;; Q)$
by $(simp \text{ add: } AssumeR\text{-NSRD } AssumeR\text{-comp } NSRD\text{-segr-closure } RA1 \text{ assms } cond\text{-srea-}AssumeR\text{-form})$

lemma $AssumeR\text{-cond-left}$:
assumes P *is NSRD* Q *is NSRD*
shows $[b]^\top_R ;; (P \triangleleft b \triangleright_R Q) = ([b]^\top_R ;; P)$
by $(rdes\text{-eq } cls: \text{assms})$

lemma $AssumeR\text{-cond-right}$:
assumes P *is NSRD* Q *is NSRD*
shows $[\neg b]^\top_R ;; (P \triangleleft b \triangleright_R Q) = ([\neg b]^\top_R ;; Q)$
by $(rdes\text{-eq } cls: \text{assms})$

11.5 Guarded commands

definition *GuardedCommR* :: 's cond \Rightarrow ('s, 't::trace, 'α) hrel-rsp \Rightarrow ('s, 't, 'α) hrel-rsp (- \rightarrow_R - [85, 86] 85) **where**
gcmd-def[*rdes-def*]: *GuardedCommR* *g* *A* = *A* \triangleleft *g* \triangleright_R *Miracle*

lemma *gcmd-false*[*simp*]: (*false* \rightarrow_R *A*) = *Miracle*
unfolding *gcmd-def* **by** (*pred-auto*)

lemma *gcmd-true*[*simp*]: (*true* \rightarrow_R *A*) = *A*
unfolding *gcmd-def* **by** (*pred-auto*)

lemma *gcmd-SRD*:
assumes *A* is *SRD*
shows (*g* \rightarrow_R *A*) is *SRD*
by (*simp* *add*: *gcmd-def SRD-cond-srea assms srdes-theory-continuous.weak.top-closed*)

lemma *gcmd-NSRD* [*closure*]:
assumes *A* is *NSRD*
shows (*g* \rightarrow_R *A*) is *NSRD*
by (*simp* *add*: *gcmd-def NSRD-cond-srea assms NSRD-Miracle*)

lemma *gcmd-Productive* [*closure*]:
assumes *A* is *NSRD* *A* is *Productive*
shows (*g* \rightarrow_R *A*) is *Productive*
by (*simp* *add*: *gcmd-def closure assms*)

lemma *gcmd-seq-distr*:
assumes *B* is *NSRD*
shows (*g* \rightarrow_R *A*) ;; *B* = (*g* \rightarrow_R (*A* ;; *B*))
by (*simp* *add*: *Miracle-left-zero NSRD-is-SRD assms cond-st-distr gcmd-def*)

lemma *gcmd-nondet-distr*:
assumes *A* is *NSRD* *B* is *NSRD*
shows (*g* \rightarrow_R (*A* \sqcap *B*)) = (*g* \rightarrow_R *A*) \sqcap (*g* \rightarrow_R *B*)
by (*rdes-eq cls: assms*)

lemma *AssumeR-as-gcmd*:
 $[b]^\top_R = b \rightarrow_R II_R$
by (*rdes-eq*)

11.6 Generalised Alternation

definition *AlternateR*
 :: 'a set \Rightarrow ('a \Rightarrow 's upred) \Rightarrow ('a \Rightarrow ('s, 't::trace, 'α) hrel-rsp) \Rightarrow ('s, 't, 'α) hrel-rsp \Rightarrow ('s, 't, 'α) hrel-rsp **where**
 [*upred-defs*, *rdes-def*]: *AlternateR* *I* *g* *A* *B* = ($\bigcap i \in I \cdot ((g\ i) \rightarrow_R (A\ i))$) \sqcap ($(\neg (\bigvee i \in I \cdot g\ i)) \rightarrow_R B$)

definition *AlternateR-list*
 :: ('s upred \times ('s, 't::trace, 'α) hrel-rsp) list \Rightarrow ('s, 't, 'α) hrel-rsp \Rightarrow ('s, 't, 'α) hrel-rsp **where**
 [*upred-defs*, *ndes-simp*]:
AlternateR-list *xs* *P* = *AlternateR* {0..*length* *xs*} ($\lambda i. \text{map } \text{fst } \text{xs} \ ! \ i$) ($\lambda i. \text{map } \text{snd } \text{xs} \ ! \ i$) *P*

syntax
-altindR-els :: *pttrn* \Rightarrow *logic* \Rightarrow *logic* \Rightarrow *logic* \Rightarrow *logic* \Rightarrow *logic* (*if* *R* - \in - \cdot - \rightarrow - *else* - *fi*)

$\text{-altindR} \quad :: \text{pttrn} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic} \ (if_R \text{-}\in\text{-} \cdot \text{-} \rightarrow \text{-} \text{fi})$

$\text{-altgcommR-els} \quad :: \text{gcomms} \Rightarrow \text{logic} \Rightarrow \text{logic} \ (if_R / \text{-} \text{else} \text{-} / \text{fi})$

$\text{-altgcommR} \quad :: \text{gcomms} \Rightarrow \text{logic} \ (if_R / \text{-} / \text{fi})$

translations

$if_R \ i \in I \cdot g \rightarrow A \text{ else } B \text{ fi} \rightarrow \text{CONST AlternateR } I \ (\lambda i. g) \ (\lambda i. A) \ B$

$if_R \ i \in I \cdot g \rightarrow A \text{ fi} \rightarrow \text{CONST AlternateR } I \ (\lambda i. g) \ (\lambda i. A) \ (\text{CONST Chaos})$

$if_R \ i \in I \cdot (g \ i) \rightarrow A \text{ else } B \text{ fi} \leftarrow \text{CONST AlternateR } I \ g \ (\lambda i. A) \ B$

$\text{-altgcommR } cs \rightarrow \text{CONST AlternateR-list } cs \ (\text{CONST Chaos})$

$\text{-altgcommR } (-\text{gcomm-show } cs) \leftarrow \text{CONST AlternateR-list } cs \ (\text{CONST Chaos})$

$\text{-altgcommR-els } cs \ P \rightarrow \text{CONST AlternateR-list } cs \ P$

$\text{-altgcommR-els } (-\text{gcomm-show } cs) \ P \leftarrow \text{CONST AlternateR-list } cs \ P$

lemma *AlternateR-NSRD-closed* [closure]:

assumes $\bigwedge i. i \in I \implies A \ i \text{ is NSRD } B \text{ is NSRD}$

shows $(if_R \ i \in I \cdot g \ i \rightarrow A \ i \text{ else } B \text{ fi}) \text{ is NSRD}$

proof (cases $I = \{\}$)

case *True*

then show ?thesis **by** (simp add: AlternateR-def assms)

next

case *False*

then show ?thesis **by** (simp add: AlternateR-def closure assms)

qed

lemma *AlternateR-empty* [simp]:

$(if_R \ i \in \{\} \cdot g \ i \rightarrow A \ i \text{ else } B \text{ fi}) = B$

by (rdes-simp)

lemma *AlternateR-Productive* [closure]:

assumes

$\bigwedge i. i \in I \implies A \ i \text{ is NSRD } B \text{ is NSRD}$

$\bigwedge i. i \in I \implies A \ i \text{ is Productive } B \text{ is Productive}$

shows $(if_R \ i \in I \cdot g \ i \rightarrow A \ i \text{ else } B \text{ fi}) \text{ is Productive}$

proof (cases $I = \{\}$)

case *True*

then show ?thesis

by (simp add: assms(4))

next

case *False*

then show ?thesis

by (simp add: AlternateR-def closure assms)

qed

lemma *AlternateR-singleton*:

assumes $A \ k \text{ is NSRD } B \text{ is NSRD}$

shows $(if_R \ i \in \{k\} \cdot g \ i \rightarrow A \ i \text{ else } B \text{ fi}) = (A(k) \triangleleft g(k) \triangleright_R B)$

by (simp add: AlternateR-def, rdes-eq cls: assms)

Convert an alternation over disjoint guards into a cascading if-then-else

lemma *AlternateR-insert-cascade*:

assumes

$\bigwedge i. i \in I \implies A \ i \text{ is NSRD}$

$A \ k \text{ is NSRD } B \text{ is NSRD}$

$(g(k) \wedge (\bigvee i \in I \cdot g(i))) = \text{false}$

shows $(if_R i \in insert\ k\ I \cdot g\ i \rightarrow A\ i\ else\ B\ fi) = (A(k) \triangleleft g(k) \triangleright_R (if_R i \in I \cdot g(i) \rightarrow A(i)\ else\ B\ fi))$
proof $(cases\ I = \{\})$
case *True*
then show $?thesis$ **by** $(simp\ add: AlternateR-singleton\ assms)$
next
case *False*
have $1: (\bigcap i \in I \cdot g\ i \rightarrow_R A\ i) = (\bigcap i \in I \cdot g\ i \rightarrow_R \mathbf{R}_s(pre_R(A\ i) \vdash peri_R(A\ i) \diamond post_R(A\ i)))$
by $(simp\ add: NSRD-is-SRD\ SRD-reactive-tri-design\ assms(1)\ cong: UINF-cong)$
from $assms(4)$ **show** $?thesis$
by $(simp\ add: AlternateR-def\ 1\ False)$
 $(rdes-eq\ cls: assms(1-3)\ False\ cong: UINF-cong)$
qed

11.7 Choose

definition $choose_srd :: ('s, 't :: trace, 'a) hrel_rsp\ (choose_R)$ **where**
 $[upred-defs, rdes-def]: choose_R = \mathbf{R}_s(true_r \vdash false \diamond true_r)$

lemma $preR_choose\ [rdes]: pre_R(choose_R) = true_r$
by $(rel-auto)$

lemma $periR_choose\ [rdes]: peri_R(choose_R) = false$
by $(rel-auto)$

lemma $postR_choose\ [rdes]: post_R(choose_R) = true_r$
by $(rel-auto)$

lemma $choose_srd-SRD\ [closure]: choose_R\ is\ SRD$
by $(simp\ add: choose_srd-def\ closure\ unrest)$

lemma $NSRD_choose_srd\ [closure]: choose_R\ is\ NSRD$
by $(rule\ NSRD-intro, simp-all\ add: closure\ unrest\ rdes)$

11.8 State Abstraction

definition $state_srea ::$
 $'s\ itself \Rightarrow ('s, 't :: trace, 'a, 'b) rel_rsp \Rightarrow (unit, 't, 'a, 'b) rel_rsp$ **where**
 $[upred-defs]: state_srea\ t\ P = \langle \exists\ \{\$st, \$st'\} \cdot P \rangle_S$

syntax
 $-state_srea :: type \Rightarrow logic \Rightarrow logic\ (state\ - \cdot -\ [0,200]\ 200)$

translations
 $state\ 'a \cdot P == CONST\ state_srea\ TYPE('a)\ P$

lemma $R1_state_srea: R1(state\ 'a \cdot P) = (state\ 'a \cdot R1(P))$
by $(rel-auto)$

lemma $R2c_state_srea: R2c(state\ 'a \cdot P) = (state\ 'a \cdot R2c(P))$
by $(rel-auto)$

lemma $R3h_state_srea: R3h(state\ 'a \cdot P) = (state\ 'a \cdot R3h(P))$
by $(rel-auto)$

lemma $RD1_state_srea: RD1(state\ 'a \cdot P) = (state\ 'a \cdot RD1(P))$
by $(rel-auto)$

lemma *RD2-state-srea*: $RD2(\text{state } 'a \cdot P) = (\text{state } 'a \cdot RD2(P))$
 by (rel-auto)

lemma *RD3-state-srea*: $RD3(\text{state } 'a \cdot P) = (\text{state } 'a \cdot RD3(P))$
 by (rel-auto, blast+)

lemma *SRD-state-srea [closure]*: $P \text{ is } SRD \implies \text{state } 'a \cdot P \text{ is } SRD$
 by (simp add: Healthy-def R1-state-srea R2c-state-srea R3h-state-srea RD1-state-srea RD2-state-srea
 RHS-def SRD-def)

lemma *NSRD-state-srea [closure]*: $P \text{ is } NSRD \implies \text{state } 'a \cdot P \text{ is } NSRD$
 by (metis Healthy-def NSRD-is-RD3 NSRD-is-SRD RD3-state-srea SRD-RD3-implies-NSRD SRD-state-srea)

lemma *preR-state-srea [rdes]*: $\text{pre}_R(\text{state } 'a \cdot P) = \langle \forall \{ \$st, \$st' \} \cdot \text{pre}_R(P) \rangle_S$
 by (simp add: state-srea-def, rel-auto)

lemma *periR-state-srea [rdes]*: $\text{peri}_R(\text{state } 'a \cdot P) = \text{state } 'a \cdot \text{peri}_R(P)$
 by (rel-auto)

lemma *postR-state-srea [rdes]*: $\text{post}_R(\text{state } 'a \cdot P) = \text{state } 'a \cdot \text{post}_R(P)$
 by (rel-auto)

11.9 While Loop

definition *WhileR* :: $'s \text{ upred} \Rightarrow ('s, 't::\text{size-trace}, 'a) \text{ hrel-rsp} \Rightarrow ('s, 't, 'a) \text{ hrel-rsp} \text{ (while}_R \text{ - do - od)}$
 where
 $\text{WhileR } b \ P = (\mu_R \ X \cdot (P ;; X) \triangleleft b \triangleright_R \text{ II}_R)$

lemma *Sup-power-false*:
 fixes $F :: 'a \text{ upred} \Rightarrow 'a \text{ upred}$
 shows $(\bigcap i. (F \hat{\ } i) \text{ false}) = (\bigcap i. (F \hat{\ } (i+1)) \text{ false})$
proof –
 have $(\bigcap i. (F \hat{\ } i) \text{ false}) = (F \hat{\ } 0) \text{ false} \sqcap (\bigcap i. (F \hat{\ } (i+1)) \text{ false})$
 by (subst Sup-power-expand, simp)
 also have $\dots = (\bigcap i. (F \hat{\ } (i+1)) \text{ false})$
 by (simp)
 finally show ?thesis .
qed

theorem *WhileR-iter-expand*:
 assumes $P \text{ is } NSRD \ P \text{ is Productive}$
 shows $\text{while}_R \ b \ \text{do } P \ \text{od} = (\bigcap i. (P \triangleleft b \triangleright_R \text{ II}_R) \hat{\ } i ;; (P ;; \text{Miracle} \triangleleft b \triangleright_R \text{ II}_R)) \text{ (is ?lhs = ?rhs)}$
proof –
 have $1:\text{Continuous } (\lambda X. P ;; SRD \ X)$
 using *SRD-Continuous*
 by (clarsimp simp add: Continuous-def seq-SUP-distl[THEN sym], drule-tac $x=A$ in spec, simp)
 have $2:\text{Continuous } (\lambda X. P ;; SRD \ X \triangleleft b \triangleright_R \text{ II}_R)$
 by (simp add: 1 closure assms)
 have $?lhs = (\mu_R \ X \cdot P ;; X \triangleleft b \triangleright_R \text{ II}_R)$
 by (simp add: WhileR-def)
 also have $\dots = (\mu \ X \cdot P ;; SRD(X) \triangleleft b \triangleright_R \text{ II}_R)$
 by (auto simp add: srd-mu-equiv closure assms)
 also have $\dots = (\nu \ X \cdot P ;; SRD(X) \triangleleft b \triangleright_R \text{ II}_R)$
 by (auto simp add: guarded-fp-uniq Guarded-if-Productive[OF assms] funcsetI closure assms)
 also have $\dots = (\bigcap i. ((\lambda X. P ;; SRD \ X \triangleleft b \triangleright_R \text{ II}_R) \hat{\ } i) \text{ false})$

```

    by (simp add: sup-continuous-lfp 2 sup-continuous-Continuous false-upred-def)
  also have ... = ( $\bigcap i. ((\lambda X. P ;; \text{SRD } X \triangleleft b \triangleright_R II_R) \wedge (i+1)) \text{ false}$ )
    by (simp add: Sup-power-false)
  also have ... = ( $\bigcap i. (P \triangleleft b \triangleright_R II_R) \wedge i ;; (P ;; \text{Miracle} \triangleleft b \triangleright_R II_R)$ )
  proof (rule SUP-cong, simp)
    fix i
    show  $((\lambda X. P ;; \text{SRD } X \triangleleft b \triangleright_R II_R) \wedge (i + 1)) \text{ false} = (P \triangleleft b \triangleright_R II_R) \wedge i ;; (P ;; \text{Miracle} \triangleleft b \triangleright_R II_R)$ 
  next
    proof (induct i)
      case 0
      thm if-eq-cancel
      then show ?case
        by (simp, metis srdes-hcond-def srdes-theory-continuous.healthy-top)
    next
      case (Suc i)
      show ?case
      proof -
        have  $((\lambda X. P ;; \text{SRD } X \triangleleft b \triangleright_R II_R) \wedge (\text{Suc } i + 1)) \text{ false} =$ 
           $P ;; \text{SRD } ((\lambda X. P ;; \text{SRD } X \triangleleft b \triangleright_R II_R) \wedge (i + 1)) \text{ false} \triangleleft b \triangleright_R II_R$ 
        by simp
        also have ... =  $P ;; \text{SRD } ((P \triangleleft b \triangleright_R II_R) \wedge i ;; (P ;; \text{Miracle} \triangleleft b \triangleright_R II_R)) \triangleleft b \triangleright_R II_R$ 
        using Suc.hyps by auto
        also have ... =  $P ;; ((P \triangleleft b \triangleright_R II_R) \wedge i ;; (P ;; \text{Miracle} \triangleleft b \triangleright_R II_R)) \triangleleft b \triangleright_R II_R$ 
        by (metis (no-types, lifting) Healthy-if NSRD-cond-srea NSRD-is-SRD NSRD-power-Suc
          NSRD-srd-skip SRD-cond-srea SRD-seqr-closure assms(1) power.power-eq-if seqr-left-unit srdes-theory-continuous.top-closed)
        also have ... =  $(P \triangleleft b \triangleright_R II_R) \wedge \text{Suc } i ;; (P ;; \text{Miracle} \triangleleft b \triangleright_R II_R)$ 
        proof (induct i)
          case 0
          then show ?case
            by (simp add: NSRD-is-SRD SRD-cond-srea SRD-left-unit SRD-seqr-closure SRD-srdes-skip
              assms(1) cond-L6 cond-st-distr srdes-theory-continuous.top-closed)
        next
          case (Suc i)
          have 1:  $II_R ;; ((P \triangleleft b \triangleright_R II_R) ;; (P \triangleleft b \triangleright_R II_R) \wedge i) = ((P \triangleleft b \triangleright_R II_R) ;; (P \triangleleft b \triangleright_R II_R) \wedge i)$ 
            by (simp add: NSRD-is-SRD RA1 SRD-cond-srea SRD-left-unit SRD-srdes-skip assms(1))
          then show ?case
          proof -
            have  $\bigwedge u. (u ;; (P \triangleleft b \triangleright_R II_R) \wedge \text{Suc } i) ;; (P ;; (\text{Miracle} \triangleleft b \triangleright_R (II_R))) \triangleleft b \triangleright_R (II_R) =$ 
               $((u \triangleleft b \triangleright_R II_R) ;; (P \triangleleft b \triangleright_R II_R) \wedge \text{Suc } i) ;; (P ;; (\text{Miracle} \triangleleft b \triangleright_R (II_R)))$ 
            by (metis (no-types) Suc.hyps 1 cond-L6 cond-st-distr power.power-power-Suc)
            then show ?thesis
              by (simp add: RA1 upred-semiring.power-Suc)
          qed
        qed
      qed
    finally show ?thesis .
  qed
  qed
  qed
  also have ... =  $(\bigcap i. (P \triangleleft b \triangleright_R II_R) \wedge i ;; (P ;; \text{Miracle} \triangleleft b \triangleright_R II_R))$ 
    by (simp add: UINF-as-Sup-collect')
  finally show ?thesis .
qed

```

theorem WhileR-star-expand:
 assumes P is NSRD P is Productive

shows $\text{while}_R b \text{ do } P \text{ od} = (P \triangleleft b \triangleright_R II_R)^{*R} ;; (P ;; \text{Miracle} \triangleleft b \triangleright_R II_R)$ (is ?lhs = ?rhs)
proof –
 have ?lhs = $(\prod i \cdot (P \triangleleft b \triangleright_R II_R) \wedge i) ;; (P ;; \text{Miracle} \triangleleft b \triangleright_R II_R)$
 by (simp add: WhileR-iter-expand seq-UINF-distr' assms)
 also have ... = $(P \triangleleft b \triangleright_R II_R)^* ;; (P ;; \text{Miracle} \triangleleft b \triangleright_R II_R)$
 by (simp add: ustar-def)
 also have ... = $((P \triangleleft b \triangleright_R II_R)^* ;; II_R) ;; (P ;; \text{Miracle} \triangleleft b \triangleright_R II_R)$
 by (simp add: seqr-assoc SRD-left-unit closure assms)
 also have ... = $(P \triangleleft b \triangleright_R II_R)^{*R} ;; (P ;; \text{Miracle} \triangleleft b \triangleright_R II_R)$
 by (simp add: nsrd-thy.Star-def)
 finally show ?thesis .
qed

lemma WhileR-NSRD-closed [closure]:

assumes P is NSRD P is Productive

shows $\text{while}_R b \text{ do } P \text{ od}$ is NSRD

by (simp add: WhileR-star-expand assms closure)

theorem WhileR-iter-form-lemma:

assumes P is NSRD

shows $(P \triangleleft b \triangleright_R II_R)^{*R} ;; (P ;; \text{Miracle} \triangleleft b \triangleright_R II_R) = ([b]^\top_R ;; P)^{*R} ;; [\neg b]^\top_R$

proof –

have $(P \triangleleft b \triangleright_R II_R)^{*R} ;; (P ;; \text{Miracle} \triangleleft b \triangleright_R II_R) = ([b]^\top_R ;; P \sqcap [\neg b]^\top_R)^{*R} ;; (P ;; \text{Miracle} \triangleleft b \triangleright_R II_R)$

by (simp add: AssumeR-NSRD NSRD-right-unit NSRD-srd-skip assms(1) cond-srea-AssumeR-form)

also have ... = $(([b]^\top_R ;; P)^{*R} ;; [\neg b]^\top_R)^{*R} ;; (P ;; \text{Miracle} \triangleleft b \triangleright_R II_R)$

by (simp add: AssumeR-NSRD NSRD-seqr-closure nsrd-thy.Star-denest assms(1))

also have ... = $(([b]^\top_R ;; P)^{*R})^{*R} ;; (P ;; \text{Miracle} \triangleleft b \triangleright_R II_R)$

by (metis (no-types, hide-lams) RD3-def RD3-idem Star-AssumeR nsrd-thy.Star-def)

also have ... = $(([b]^\top_R ;; P)^{*R}) ;; (P ;; \text{Miracle} \triangleleft b \triangleright_R II_R)$

by (simp add: AssumeR-NSRD NSRD-seqr-closure nsrd-thy.Star-invol assms(1))

also have ... = $(([b]^\top_R ;; P)^{*R}) ;; ([b]^\top_R ;; P ;; \text{Miracle} \sqcap [\neg b]^\top_R)$

by (simp add: AssumeR-NSRD NSRD-Miracle NSRD-right-unit NSRD-seqr-closure NSRD-srd-skip assms(1) cond-srea-AssumeR-form)

also have ... = $(([b]^\top_R ;; P)^{*R}) ;; [b]^\top_R ;; P ;; \text{Miracle} \sqcap (([b]^\top_R ;; P)^{*R}) ;; [\neg b]^\top_R$

by (simp add: upred-semiring.distrib-left)

also have ... = $[b]^\top_R ;; P)^{*R} ;; [\neg b]^\top_R$

proof –

have $(([b]^\top_R ;; P)^{*R}) ;; [\neg b]^\top_R = (II_R \sqcap ([b]^\top_R ;; P)^{*R} ;; [b]^\top_R ;; P) ;; [\neg b]^\top_R$

by (simp add: AssumeR-NSRD NSRD-seqr-closure nsrd-thy.Star-unfoldr-eq assms(1))

also have ... = $[\neg b]^\top_R \sqcap (([b]^\top_R ;; P)^{*R} ;; [b]^\top_R ;; P) ;; [\neg b]^\top_R$

by (metis (no-types, lifting) AssumeR-NSRD AssumeR-as-gcmd NSRD-srd-skip Star-AssumeR nsrd-thy.Star-slide gcmd-seq-distr skip-srea-self-unit urel-dioid.distrib-right')

also have ... = $[\neg b]^\top_R \sqcap (([b]^\top_R ;; P)^{*R} ;; [b]^\top_R ;; P ;; [b \vee \neg b]^\top_R) ;; [\neg b]^\top_R$

by (simp add: AssumeR-true NSRD-right-unit assms(1))

also have ... = $[\neg b]^\top_R \sqcap (([b]^\top_R ;; P)^{*R} ;; [b]^\top_R ;; P ;; [b]^\top_R) ;; [\neg b]^\top_R$

$\sqcap (([b]^\top_R ;; P)^{*R} ;; [b]^\top_R ;; P ;; [\neg b]^\top_R) ;; [\neg b]^\top_R$

by (metis (no-types, hide-lams) AssumeR-choice upred-semiring.add-assoc upred-semiring.distrib-left upred-semiring.distrib-right)

also have ... = $[\neg b]^\top_R \sqcap ([b]^\top_R ;; P)^{*R} ;; [b]^\top_R ;; P ;; ([b]^\top_R ;; [\neg b]^\top_R)$

$\sqcap ([b]^\top_R ;; P)^{*R} ;; [b]^\top_R ;; P ;; ([\neg b]^\top_R ;; [\neg b]^\top_R)$

by (simp add: RA1)

also have ... = $[\neg b]^\top_R \sqcap (([b]^\top_R ;; P)^{*R} ;; [b]^\top_R ;; P ;; \text{Miracle})$

$\sqcap (([b]^\top_R ;; P)^{*R} ;; [b]^\top_R ;; P ;; [\neg b]^\top_R)$

by (simp add: AssumeR-comp AssumeR-false)

finally have $([b]^\top_R ;; P)^{*R} ;; [\neg b]^\top_R \sqsubseteq (([b]^\top_R ;; P)^{*R}) ;; [b]^\top_R ;; P ;; \text{Miracle}$
by (*simp add: semilattice-sup-class.le-supI1*)
thus *?thesis*
by (*simp add: semilattice-sup-class.le-iff-sup*)
qed
finally show *?thesis* .
qed

theorem *WhileR-iter-form:*

assumes *P is NSRD P is Productive*
shows $\text{while}_R b \text{ do } P \text{ od} = ([b]^\top_R ;; P)^{*R} ;; [\neg b]^\top_R$
by (*simp add: WhileR-iter-form-lemma WhileR-star-expand assms*)

theorem *WhileR-false:*

assumes *P is NSRD*
shows $\text{while}_R \text{false} \text{ do } P \text{ od} = \text{II}_R$
by (*simp add: WhileR-def rpred closure sdes-theory-continuous.LFP-const*)

theorem *WhileR-true:*

assumes *P is NSRD P is Productive*
shows $\text{while}_R \text{true} \text{ do } P \text{ od} = P^{*R} ;; \text{Miracle}$
by (*simp add: WhileR-iter-form AssumeR-true AssumeR-false SRD-left-unit assms closure*)

lemma *WhileR-insert-assume:*

assumes *P is NSRD P is Productive*
shows $\text{while}_R b \text{ do } ([b]^\top_R ;; P) \text{ od} = \text{while}_R b \text{ do } P \text{ od}$
by (*simp add: AssumeR-NSRD AssumeR-comp NSRD-seqr-closure Productive-seq-2 RA1 WhileR-iter-form assms*)

theorem *WhileR-rdes-def [rdes-def]:*

assumes *P is RC Q is RR R is RR \$st' \# Q R is R4*
shows $\text{while}_R b \text{ do } \mathbf{R}_s(P \vdash Q \diamond R) \text{ od} =$
 $\mathbf{R}_s([b]^\top_r ;; R)^{*r} \text{ wp}_r ([b]_{S<} \Rightarrow_r P) \vdash ([b]^\top_r ;; R)^{*r} ;; [b]^\top_r ;; Q \diamond ([b]^\top_r ;; R)^{*r} ;; [\neg b]^\top_r$
(is ?lhs = ?rhs)

proof –

have $?lhs = ([b]^\top_R ;; \mathbf{R}_s(P \vdash Q \diamond R))^{*R} ;; [\neg b]^\top_R$
by (*simp add: WhileR-iter-form Productive-rdes-RR-intro assms closure*)
also have $\dots = ?rhs$
by (*simp add: rdes-def assms closure unrest rpred wp del: rea-star-wp*)
finally show *?thesis* .
qed

Refinement introduction law for reactive while loops

theorem *WhileR-refine-intro:*

assumes
— Closure conditions
 $Q_1 \text{ is RC } Q_2 \text{ is RR } Q_3 \text{ is RR } \$st' \# Q_2 \text{ } Q_3 \text{ is R4}$
— Refinement conditions
 $([b]^\top_r ;; Q_3)^{*r} \text{ wp}_r ([b]_{S<} \Rightarrow_r Q_1) \sqsubseteq P_1$
 $P_2 \sqsubseteq [b]^\top_r ;; Q_2$
 $P_2 \sqsubseteq [b]^\top_r ;; Q_3 ;; P_2$
 $P_3 \sqsubseteq [\neg b]^\top_r$
 $P_3 \sqsubseteq [b]^\top_r ;; Q_3 ;; P_3$
shows $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) \sqsubseteq \text{while}_R b \text{ do } \mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3) \text{ od}$
proof (*simp add: rdes-def assms, rule sdes-tri-refine-intro*)

```

show  $([b]^\top_r ;; Q_3)^{*r} wp_r ([b]_{S<} \Rightarrow_r Q_1) \sqsubseteq P_1$ 
  by (simp add: assms)
show  $P_2 \sqsubseteq (P_1 \wedge ([b]^\top_r ;; Q_3)^{*r} ;; [b]^\top_r ;; Q_2)$ 
proof –
  have  $P_2 \sqsubseteq ([b]^\top_r ;; Q_3)^{*r} ;; [b]^\top_r ;; Q_2$ 
    by (simp add: assms rea-assume-RR rrel-thy.Star-inductl seq-RR-closed seqr-assoc)
  thus ?thesis
    by (simp add: utp-pred-laws.le-infI2)
qed
show  $P_3 \sqsubseteq (P_1 \wedge ([b]^\top_r ;; Q_3)^{*r} ;; [\neg b]^\top_r)$ 
proof –
  have  $P_3 \sqsubseteq ([b]^\top_r ;; Q_3)^{*r} ;; [\neg b]^\top_r$ 
    by (simp add: assms rea-assume-RR rrel-thy.Star-inductl seqr-assoc)
  thus ?thesis
    by (simp add: utp-pred-laws.le-infI2)
qed
qed

```

11.10 Iteration Construction

definition *IterateR*

$:: 'a \text{ set} \Rightarrow ('a \Rightarrow 's \text{ upred}) \Rightarrow ('a \Rightarrow ('s, 't::\text{size-trace}, 'a) \text{ hrel-rsp}) \Rightarrow ('s, 't, 'a) \text{ hrel-rsp}$
where *IterateR* $A \ g \ P = \text{while}_R (\bigvee i \in A \cdot g(i)) \text{ do } (if_R i \in A \cdot g(i) \rightarrow P(i) \text{ fi}) \text{ od}$

definition *IterateR-list*

$:: ('s \text{ upred} \times ('s, 't::\text{size-trace}, 'a) \text{ hrel-rsp}) \text{ list} \Rightarrow ('s, 't, 'a) \text{ hrel-rsp}$ **where**
 $[upred-defs, ndes-simp]:$
IterateR-list $xs = \text{IterateR} \{0..<\text{length } xs\} (\lambda i. \text{map fst } xs ! i) (\lambda i. \text{map snd } xs ! i)$

syntax

-iter-srd $:: p\text{trn} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic} \text{ (do}_R \text{ -}\in\text{ -} \cdot \text{ -} \rightarrow \text{ - fi})$
-iter-gcommR $:: g\text{comms} \Rightarrow \text{logic} \text{ (do}_R \text{ / - /od)}$

translations

-iter-srd $x \ A \ g \ P \Rightarrow \text{CONST } \text{IterateR } A \ (\lambda x. g) \ (\lambda x. P)$
-iter-srd $x \ A \ g \ P \Leftarrow \text{CONST } \text{IterateR } A \ (\lambda x. g) \ (\lambda x'. P)$
-iter-gcommR $cs \rightarrow \text{CONST } \text{IterateR-list } cs$
-iter-gcommR $(\text{-gcomm-show } cs) \leftarrow \text{CONST } \text{IterateR-list } cs$

lemma *IterateR-NSRD-closed [closure]:*

assumes
 $\bigwedge i. i \in I \Longrightarrow P(i) \text{ is NSRD}$
 $\bigwedge i. i \in I \Longrightarrow P(i) \text{ is Productive}$
shows $\text{do}_R i \in I \cdot g(i) \rightarrow P(i) \text{ fi is NSRD}$
by (*simp add: IterateR-def closure assms*)

lemma *IterateR-empty:*

$\text{do}_R i \in \{\} \cdot g(i) \rightarrow P(i) \text{ fi} = II_R$
by (*simp add: IterateR-def srd-mu-equiv closure rpred gfp-const WhileR-false*)

lemma *IterateR-singleton:*

assumes $P \ k \text{ is NSRD } P \ k \text{ is Productive}$
shows $\text{do}_R i \in \{k\} \cdot g(i) \rightarrow P(i) \text{ fi} = \text{while}_R g(k) \text{ do } P(k) \text{ od}$ (**is** ?lhs = ?rhs)
proof –
have ?lhs = $\text{while}_R g \ k \text{ do } P \ k \triangleleft g \ k \triangleright_R \text{Chaos}$ **od**
by (*simp add: IterateR-def AlternateR-singleton assms closure*)

also have ... = $\text{while}_R g k \text{ do } [g k]^\top_R ; (P k \triangleleft g k \triangleright_R \text{Chaos}) \text{ od}$
by (*simp add: WhileR-insert-assume closure assms*)
also have ... = $\text{while}_R g k \text{ do } P k \text{ od}$
by (*simp add: AssumeR-cond-left NSRD-Chaos WhileR-insert-assume assms*)
finally show ?thesis .
qed

declare *IterateR-list-def* [rdes-def]
declare *IterateR-def* [rdes-def]

method *unfold-iteration* = *simp add: IterateR-list-def IterateR-def AlternateR-list-def AlternateR-def UINF-upto-expand-first*

11.11 Substitution Laws

lemma *srd-subst-Chaos* [usubst]:
 $\sigma \dagger_S \text{Chaos} = \text{Chaos}$
by (*rdes-simp*)

lemma *srd-subst-Miracle* [usubst]:
 $\sigma \dagger_S \text{Miracle} = \text{Miracle}$
by (*rdes-simp*)

lemma *srd-subst-skip* [usubst]:
 $\sigma \dagger_S II_R = \langle \sigma \rangle_R$
by (*rdes-eq*)

lemma *srd-subst-assigns* [usubst]:
 $\sigma \dagger_S \langle \varrho \rangle_R = \langle \varrho \circ \sigma \rangle_R$
by (*rdes-eq*)

11.12 Algebraic Laws

theorem *assigns-srd-id*: $\langle \text{id} \rangle_R = II_R$
by (*rdes-eq*)

theorem *assigns-srd-comp*: $\langle \sigma \rangle_R ; \langle \varrho \rangle_R = \langle \varrho \circ \sigma \rangle_R$
by (*rdes-eq*)

theorem *assigns-srd-Miracle*: $\langle \sigma \rangle_R ; \text{Miracle} = \text{Miracle}$
by (*rdes-eq*)

theorem *assigns-srd-Chaos*: $\langle \sigma \rangle_R ; \text{Chaos} = \text{Chaos}$
by (*rdes-eq*)

theorem *assigns-srd-cond* : $\langle \sigma \rangle_R \triangleleft b \triangleright_R \langle \varrho \rangle_R = \langle \sigma \triangleleft b \triangleright_s \varrho \rangle_R$
by (*rdes-eq*)

theorem *assigns-srd-left-seq*:
assumes *P is NSRD*
shows $\langle \sigma \rangle_R ; P = \sigma \dagger_S P$
by (*rdes-simp cls: assms*)

lemma *AlternateR-seq-distr*:
assumes $\bigwedge i. A i \text{ is NSRD } B \text{ is NSRD } C \text{ is NSRD}$
shows $(\text{if}_R i \in I \cdot g i \rightarrow A i \text{ else } B \text{ fi}) ; C = (\text{if}_R i \in I \cdot g i \rightarrow A i ; C \text{ else } B ; C \text{ fi})$

proof (*cases* $I = \{\}$)
case *True*
then show *?thesis* **by** (*simp*)
next
case *False*
then show *?thesis*
by (*simp add: AlternateR-def upred-semiring.distrib-right seq-UINF-distr gcmd-seq-distr assms(3)*)
qed

lemma *AlternateR-is-cond-srea*:
assumes A is NSRD B is NSRD
shows $(if_R i \in \{a\} \cdot g \rightarrow A \text{ else } B \text{ fi}) = (A \triangleleft g \triangleright_R B)$
by (*rdes-eq cls: assms*)

lemma *AlternateR-Chaos*:
 $if_R i \in A \cdot g(i) \rightarrow Chaos \text{ fi} = Chaos$
by (*cases* $A = \{\}$, *simp*, *rdes-eq*)

lemma *choose-srd-par*:
 $choose_R \parallel_R choose_R = choose_R$
by (*rdes-eq*)

11.13 Lifting designs to reactive designs

definition *des-rea-lift* $:: 's \text{ hrel-des} \Rightarrow ('s, 't::\text{trace}, 'a) \text{ hrel-rsp } (\mathbf{R}_D)$ **where**
 $[upred-defs]: \mathbf{R}_D(P) = \mathbf{R}_s(\lceil pre_D(P) \rceil_S \vdash (false \diamond (\$tr' =_u \$tr \wedge \lceil post_D(P) \rceil_S)))$

definition *des-rea-drop* $:: ('s, 't::\text{trace}, 'a) \text{ hrel-rsp} \Rightarrow 's \text{ hrel-des } (\mathbf{D}_R)$ **where**
 $[upred-defs]: \mathbf{D}_R(P) = \lfloor (pre_R(P)) \rfloor_{\$tr/\$tr'} \vdash_v \$st \rfloor_{S<} \vdash_n \lfloor (post_R(P)) \rfloor_{\$tr/\$tr'} \vdash_v \{\$st, \$st'\} \rfloor_S$

lemma *ndesign-rea-lift-inverse*: $\mathbf{D}_R(\mathbf{R}_D(p \vdash_n Q)) = p \vdash_n Q$
apply (*simp add: des-rea-lift-def des-rea-drop-def rea-pre-RHS-design rea-post-RHS-design*)
apply (*simp add: R1-def R2c-def R2s-def usubst unrest*)
apply (*rel-auto*)
done

lemma *ndesign-rea-lift-injective*:
assumes P is \mathbf{N} Q is \mathbf{N} $\mathbf{R}_D P = \mathbf{R}_D Q$ (**is** *?RP(P) = ?RQ(Q)*)
shows $P = Q$

proof –
have $?RP(\lfloor pre_D(P) \rfloor_{<} \vdash_n post_D(P)) = ?RQ(\lfloor pre_D(Q) \rfloor_{<} \vdash_n post_D(Q))$
by (*simp add: ndesign-form assms*)
hence $\lfloor pre_D(P) \rfloor_{<} \vdash_n post_D(P) = \lfloor pre_D(Q) \rfloor_{<} \vdash_n post_D(Q)$
by (*metis ndesign-rea-lift-inverse*)
thus *?thesis*
by (*simp add: ndesign-form assms*)
qed

lemma *des-rea-lift-closure* $[closure]: \mathbf{R}_D(P)$ is SRD
by (*simp add: des-rea-lift-def RHS-design-is-SRD unrest*)

lemma *preR-des-rea-lift* $[rdes]$:
 $pre_R(\mathbf{R}_D(P)) = R1(\lceil pre_D(P) \rceil_S)$
by (*rel-auto*)

lemma *periR-des-rea-lift* [*rdes*]:
 $\text{peri}_R(\mathbf{R}_D(P)) = (\text{false} \triangleleft \lceil \text{pre}_D(P) \rceil_S \triangleright (\$tr \leq_u \$tr'))$
by (*rel-auto*)

lemma *postR-des-rea-lift* [*rdes*]:
 $\text{post}_R(\mathbf{R}_D(P)) = ((\text{true} \triangleleft \lceil \text{pre}_D(P) \rceil_S \triangleright (\neg \$tr \leq_u \$tr')) \Rightarrow (\$tr' =_u \$tr \wedge \lceil \text{post}_D(P) \rceil_S))$
apply (*rel-auto*) **using** *minus-zero-eq* **by** *blast*

lemma *ndes-rea-lift-closure* [*closure*]:
assumes *P is N*
shows $\mathbf{R}_D(P)$ *is NSRD*

proof –

obtain *p Q* **where** *P*: $P = (p \vdash_n Q)$
by (*metis H1-H3-commute H1-H3-is-normal-design H1-idem Healthy-def assms*)
show *?thesis*
apply (*rule NSRD-intro*)
apply (*simp-all add: closure rdes unrest P*)
apply (*rel-auto*)
done

qed

lemma *R-D-mono*:

assumes *P is H Q is H* $P \sqsubseteq Q$
shows $\mathbf{R}_D(P) \sqsubseteq \mathbf{R}_D(Q)$
apply (*simp add: des-rea-lift-def*)
apply (*rule srdes-tri-refine-intro'*)
apply (*auto intro: H1-H2-refines assms aext-mono*)
apply (*rel-auto*)
apply (*metis (no-types, hide-lams) aext-mono assms(3) design-post-choice*
semilattice-sup-class.sup.orderE utp-pred-laws.inf.coboundedI1 utp-pred-laws.inf commute utp-pred-laws.sup.order-iff)
done

Homomorphism laws

lemma *R-D-Miracle*:

$\mathbf{R}_D(\top_D) = \text{Miracle}$
by (*simp add: Miracle-def, rel-auto*)

lemma *R-D-Chaos*:

$\mathbf{R}_D(\perp_D) = \text{Chaos}$

proof –

have $\mathbf{R}_D(\perp_D) = \mathbf{R}_D(\text{false} \vdash_r \text{true})$
by (*rel-auto*)
also have $\dots = \mathbf{R}_s(\text{false} \vdash \text{false} \diamond (\$tr' =_u \$tr))$
by (*simp add: Chaos-def des-rea-lift-def alpha*)
also have $\dots = \mathbf{R}_s(\text{true})$
by (*rel-auto*)
also have $\dots = \text{Chaos}$
by (*simp add: Chaos-def design-false-pre*)
finally show *?thesis* .

qed

lemma *R-D-inf*:

$\mathbf{R}_D(P \sqcap Q) = \mathbf{R}_D(P) \sqcap \mathbf{R}_D(Q)$
by (*rule antisym, rel-auto+*)

lemma *R-D-cond*:

$\mathbf{R}_D(P \triangleleft [b]_{D<} \triangleright Q) = \mathbf{R}_D(P) \triangleleft b \triangleright_R \mathbf{R}_D(Q)$
by (*rule antisym, rel-auto+*)

lemma *R-D-seq-ndesign*:

$\mathbf{R}_D(p_1 \vdash_n Q_1) ;; \mathbf{R}_D(p_2 \vdash_n Q_2) = \mathbf{R}_D((p_1 \vdash_n Q_1) ;; (p_2 \vdash_n Q_2))$
apply (*rule antisym*)
apply (*rule SRD-refine-intro*)
apply (*simp-all add: closure rdes ndesign-composition-wp*)
using *dual-order.trans* **apply** (*rel-blast*)
using *dual-order.trans* **apply** (*rel-blast*)
apply (*rel-auto*)
apply (*rule SRD-refine-intro*)
apply (*simp-all add: closure rdes ndesign-composition-wp*)
apply (*rel-auto*)
apply (*rel-auto*)
apply (*rel-auto*)
done

lemma *R-D-seq*:

assumes *P is N Q is N*
shows $\mathbf{R}_D(P) ;; \mathbf{R}_D(Q) = \mathbf{R}_D(P ;; Q)$
by (*metis R-D-seq-ndesign assms ndesign-form*)

These laws are applicable only when there is no further alphabet extension

lemma *R-D-skip*:

$\mathbf{R}_D(\Pi_D) = (\Pi_R :: ('s, 't::trace, unit) hrel-rsp)$
apply (*rel-auto*) **using** *minus-zero-eq* **by** *blast+*

lemma *R-D-assigs*:

$\mathbf{R}_D(\langle \sigma \rangle_D) = (\langle \sigma \rangle_R :: ('s, 't::trace, unit) hrel-rsp)$
by (*simp add: assigns-d-def des-rea-lift-def alpha assigns-srd-RHS-tri-des, rel-auto*)

end

12 Instantaneous Reactive Designs

theory *utp-rdes-instant*

imports *utp-rdes-prog*

begin

definition *ISR D1* :: $('s, 't::trace, 'a) hrel-rsp \Rightarrow ('s, 't, 'a) hrel-rsp$ **where**
 $[upred-defs]: \text{ISR D1}(P) = P \parallel_R \mathbf{R}_s(\text{true}_r \vdash \text{false} \diamond (\$tr' =_u \$tr))$

definition *ISR D* :: $('s, 't::trace, 'a) hrel-rsp \Rightarrow ('s, 't, 'a) hrel-rsp$ **where**
 $[upred-defs]: \text{ISR D} = \text{ISR D1} \circ \text{NSRD}$

lemma *ISR D1-idem*: $\text{ISR D1}(\text{ISR D1}(P)) = \text{ISR D1}(P)$
by (*rel-auto*)

lemma *ISR D1-monotonic*: $P \sqsubseteq Q \Longrightarrow \text{ISR D1}(P) \sqsubseteq \text{ISR D1}(Q)$
by (*rel-auto*)

lemma *ISR D1-RHS-design-form*:

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

shows $ISR D1(\mathbf{R}_s(P \vdash Q \diamond R)) = \mathbf{R}_s(P \vdash \text{false} \diamond (R \wedge \$tr' =_u \$tr))$
using *assms* **by** (*simp add: ISR D1-def choose-srd-def RHS-tri-design-par unrest, rel-auto*)

lemma *ISR D1-form*:

$ISR D1(SRD(P)) = \mathbf{R}_s(\text{pre}_R(P) \vdash \text{false} \diamond (\text{post}_R(P) \wedge \$tr' =_u \$tr))$
by (*simp add: ISR D1-RHS-design-form SRD-as-reactive-tri-design unrest*)

lemma *ISR D1-rdes-def* [*rdes-def*]:

$\llbracket P \text{ is } RR; R \text{ is } RR \rrbracket \implies ISR D1(\mathbf{R}_s(P \vdash Q \diamond R)) = \mathbf{R}_s(P \vdash \text{false} \diamond (R \wedge \$tr' =_u \$tr))$
by (*simp add: ISR D1-def rdes-def closure rpred*)

lemma *ISR D-intro*:

assumes $P \text{ is } NSRD \text{ } \text{peri}_R(P) = (\neg_r \text{pre}_R(P)) (\$tr' =_u \$tr) \sqsubseteq \text{post}_R(P)$
shows $P \text{ is } ISR D$

proof –

have $\mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \diamond \text{post}_R(P)) \text{ is } ISR D1$
apply (*simp add: Healthy-def rdes-def closure assms(1-2)*)
using *assms(3) least-zero* **apply** (*rel-blast*)
done
hence $P \text{ is } ISR D1$
by (*simp add: SRD-reactive-tri-design closure assms(1)*)
thus *?thesis*
by (*simp add: ISR D-def Healthy-comp assms(1)*)

qed

lemma *ISR D1-rdes-intro*:

assumes $P \text{ is } RR \ Q \text{ is } RR \ (\$tr' =_u \$tr) \sqsubseteq Q$
shows $\mathbf{R}_s(P \vdash \text{false} \diamond Q) \text{ is } ISR D1$
unfolding *Healthy-def*
by (*simp add: ISR D1-rdes-def assms closure unrest utp-pred-laws.inf.absorb1*)

lemma *ISR D-rdes-intro* [*closure*]:

assumes $P \text{ is } RC \ Q \text{ is } RR \ (\$tr' =_u \$tr) \sqsubseteq Q$
shows $\mathbf{R}_s(P \vdash \text{false} \diamond Q) \text{ is } ISR D$
unfolding *Healthy-def*
by (*simp add: ISR D-def closure Healthy-if ISR D1-rdes-def assms unrest utp-pred-laws.inf.absorb1*)

lemma *ISR D-implies-ISR D1*:

assumes $P \text{ is } ISR D$
shows $P \text{ is } ISR D1$

proof –

have $ISR D(P) \text{ is } ISR D1$
by (*simp add: ISR D-def Healthy-def ISR D1-idem*)
thus *?thesis*
by (*simp add: assms Healthy-if*)

qed

lemma *ISR D-implies-SRD*:

assumes $P \text{ is } ISR D$
shows $P \text{ is } SRD$

proof –

have $1:ISR D(P) = \mathbf{R}_s((\neg_r (\neg_r \text{pre}_R P) ;; R1 \text{ true} \wedge R1 \text{ true}) \vdash \text{false} \diamond (\text{post}_R P \wedge \$tr' =_u \$tr))$
by (*simp add: NSRD-form ISR D1-def ISR D-def RHS-tri-design-par rdes-def unrest closure*)
moreover **have** ... *is SRD*
by (*simp add: closure unrest*)

ultimately have $ISR D(P)$ is SRD
 by (simp)
 with *assms* show ?thesis
 by (simp add: Healthy-def)
 qed

lemma *ISR D-implies-NSRD* [closure]:

assumes P is $ISR D$
 shows P is $NSRD$

proof –

have $1:ISR D(P) = ISR D1(RD3(SRD(P)))$
 by (simp add: *ISR D-def NSRD-def SRD-def*, *metis RD1-RD3-commute RD3-left-subsumes-RD2*)
 also have $\dots = ISR D1(RD3(P))$
 by (simp add: *assms ISR D-implies-SRD Healthy-if*)
 also have $\dots = ISR D1(\mathbf{R}_s((\neg_r pre_R P) wp_r false_h \vdash (\exists \$st' \cdot peri_R P) \diamond post_R P))$
 by (simp add: *RD3-def*, *subst SRD-right-unit-tri-lemma*, *simp-all add: assms ISR D-implies-SRD*)
 also have $\dots = \mathbf{R}_s((\neg_r pre_R P) wp_r false_h \vdash false \diamond (post_R P \wedge \$tr' =_u \$tr))$
 by (simp add: *RHS-tri-design-par ISR D1-def unrest choose-srd-def rpred closure ISR D-implies-SRD assms*)
 also have $\dots = (\dots ;; II_R)$
 by (*rdes-simp*, *simp add: RHS-tri-normal-design-composition' closure assms unrest ISR D-implies-SRD wp rpred wp-rea-false-RC*)
 also have \dots is $RD3$
 by (simp add: *Healthy-def RD3-def segr-assoc*)
 finally show ?thesis
 by (simp add: *SRD-RD3-implies-NSRD Healthy-if assms ISR D-implies-SRD*)
 qed

lemma *ISR D-form*:

assumes P is $ISR D$
 shows $\mathbf{R}_s(pre_R(P) \vdash false \diamond (post_R(P) \wedge \$tr' =_u \$tr)) = P$

proof –

have $P = ISR D1(P)$
 by (simp add: *ISR D-implies-ISR D1 assms Healthy-if*)
 also have $\dots = ISR D1(\mathbf{R}_s(pre_R(P) \vdash peri_R(P) \diamond post_R(P)))$
 by (simp add: *SRD-reactive-tri-design ISR D-implies-SRD assms*)
 also have $\dots = \mathbf{R}_s(pre_R(P) \vdash false \diamond (post_R(P) \wedge \$tr' =_u \$tr))$
 by (simp add: *ISR D1-rdes-def closure assms*)
 finally show ?thesis ..
 qed

lemma *ISR D-elim* [*RD-elim*]:

$\llbracket P \text{ is } ISR D; Q(\mathbf{R}_s(pre_R(P) \vdash false \diamond (post_R(P) \wedge \$tr' =_u \$tr))) \rrbracket \implies Q(P)$
 by (simp add: *ISR D-form*)

lemma *skip-srd-ISR D* [closure]: II_R is $ISR D$

by (*rule ISR D-intro*, *simp-all add: rdes closure*)

lemma *assigns-srd-ISR D* [closure]: $\langle \sigma \rangle_R$ is $ISR D$

by (*rule ISR D-intro*, *simp-all add: rdes closure, rel-auto*)

lemma *seq-ISR D-closed*:

assumes P is $ISR D$ Q is $ISR D$
 shows $P ;; Q$ is $ISR D$
 apply (*insert assms*)

```

apply (erule ISRD-elim)+
apply (simp add: rdes-def closure assms unrest)
apply (rule ISRD-rdes-intro)
  apply (simp-all add: rdes-def closure assms unrest)
apply (rel-auto)
done

```

```

lemma ISRD-Miracle-right-zero:
  assumes P is ISRD  $\text{pre}_R(P) = \text{true}_r$ 
  shows  $P \;; \text{Miracle} = \text{Miracle}$ 
  by (rdes-simp cls: assms)

```

A recursion whose body does not extend the trace results in divergence

```

lemma ISRD-recurse-Chaos:
  assumes P is ISRD  $\text{post}_R P \;; \text{true}_r = \text{true}_r$ 
  shows  $(\mu_R X \cdot P \;; X) = \text{Chaos}$ 
proof –
  have  $1: (\mu_R X \cdot P \;; X) = (\mu X \cdot P \;; \text{SRD}(X))$ 
    by (auto simp add: srdes-theory-continuous.utp-lfp-def closure assms)
  have  $(\mu X \cdot P \;; \text{SRD}(X)) \sqsubseteq \text{Chaos}$ 
  proof (rule gfp-upperbound)
    have  $P \;; \text{Chaos} \sqsubseteq \text{Chaos}$ 
      apply (rdes-refine-split cls: assms)
      using assms(2) apply (rel-auto, metis (no-types, lifting) dual-order.antisym order-refl)
      apply (rel-auto)+
      done
    thus  $P \;; \text{SRD Chaos} \sqsubseteq \text{Chaos}$ 
      by (simp add: Healthy-if srdes-theory-continuous.bottom-closed)
  qed
thus ?thesis
  by (metis 1 dual-order.antisym srdes-theory-continuous.LFP-closed srdes-theory-continuous.bottom-lower)
qed

```

```

lemma recursive-assign-Chaos:
   $(\mu_R X \cdot \langle \sigma \rangle_R \;; X) = \text{Chaos}$ 
  by (rule ISRD-recurse-Chaos, simp-all add: closure rdes, rel-auto)

```

end

13 Meta-theory for Reactive Designs

```

theory utp-rea-designs
imports
  utp-rdes-healths
  utp-rdes-designs
  utp-rdes-triples
  utp-rdes-normal
  utp-rdes-contracts
  utp-rdes-tactics
  utp-rdes-parallel
  utp-rdes-prog
  utp-rdes-instant
  utp-rdes-guarded
begin end

```

References

- [1] A. Armstrong, V. Gomes, and G. Struth. Building program construction and verification tools from algebraic principles. *Formal Aspects of Computing*, 28(2):265–293, 2015.
- [2] S. Foster, A. Cavalcanti, S. Canham, J. Woodcock, and F. Zeyda. Unifying theories of reactive design contracts. *Submitted to Theoretical Computer Science*, Dec 2017. Preprint: <https://arxiv.org/abs/1712.10233>.
- [3] S. Foster, F. Zeyda, and J. Woodcock. Unifying heterogeneous state-spaces with lenses. In *ICTAC*, LNCS 9965. Springer, 2016.
- [4] D. Kozen. On Kleene algebras and closed semirings. In *Proc. 15th Symp. on Mathematical Foundations of Computer Science (MFCS)*, volume 452 of *LNCS*, pages 26–47. Springer, 1990.
- [5] M. V. M. Oliveira. *Formal Derivation of State-Rich Reactive Programs using Circus*. PhD thesis, Department of Computer Science - University of York, UK, 2006. YCST-2006-02.