

Reactive Designs in Isabelle/UTP

Simon Foster James Baxter Ana Cavalcanti Jim Woodcock
Samuel Canham

September 11, 2019

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	4
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	14
3.1	Reactive design forms	15
3.2	Auxiliary healthiness conditions	17
3.3	Composition laws	18
3.4	Refinement introduction laws	24
3.5	Distribution laws	25
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	32
4.4	Formation laws	34
4.4.1	Order laws	36
4.5	Composition laws	36
4.6	Refinement introduction laws	41
4.7	Closure laws	43
4.8	Distribution laws	44
4.9	Algebraic laws	45
4.10	Recursion laws	46
5	Normal Reactive Designs	47
5.1	UTP theory	61
6	Syntax for reactive design contracts	62
7	Reactive design tactics	63
8	Reactive design parallel-by-merge	65
8.1	Example basic merge	80
8.2	Simple parallel composition	81
9	Productive Reactive Designs	82
9.1	Healthiness condition	82
9.2	Reactive design calculations	83
9.3	Closure laws	84
10	Guarded Recursion	87
10.1	Traces with a size measure	87
10.2	Guardedness	88
10.3	Tail recursive fixed-point calculations	92
11	Reactive Design Programs	93
11.1	State substitution	93
11.2	Assignment	94
11.3	Conditional	95
11.4	Assumptions	96
11.5	Guarded commands	97
11.6	Generalised Alternation	98
11.7	Choose	100
11.8	Divergence Freedom	100
11.9	State Abstraction	101
11.10	Reactive Frames	102
11.11	While Loop	103
11.12	Iteration Construction	108
11.13	Substitution Laws	110
11.14	Algebraic Laws	111
11.15	Lifting designs to reactive designs	112
11.16	State Invariants	114

1 Introduction

This document contains a mechanisation in Isabelle/UTP [2] of our theory of reactive designs. Reactive designs form an important semantic foundation for reactive modelling languages such as Circus [3]. For more details of this work, please see our recent paper [1].

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
```

```
declare des-vars.splits [alpha-splits del]
declare rp-vars.splits [alpha-splits del]
declare rp-vars.splits [alpha-splits]
declare des-vars.splits [alpha-splits]
```

```
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 for 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 $skip\text{-}rea :: ('t::trace, 'α) hrel\text{-}rp (II_C)$ **where**
 $skip\text{-}rea\text{-}def \text{ [urel-defs]}: II_C = (II \vee (\neg \$ok \wedge \$tr \leq_u \$tr'))$

definition $skip\text{-}srea :: ('s, 't::trace, 'α) hrel\text{-}rsp (II_R)$ **where**
 $skip\text{-}srea\text{-}def \text{ [urel-defs]}: II_R = ((\exists \$st \cdot II_C) \triangleleft \$wait \triangleright II_C)$

lemma $skip\text{-}rea\text{-}R1\text{-}lemma: II_C = R1(\$ok \Rightarrow II)$
by $(rel\text{-}auto)$

lemma $skip\text{-}rea\text{-}form: II_C = (II \triangleleft \$ok \triangleright R1(true))$
by $(rel\text{-}auto)$

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

lemma $R1\text{-}skip\text{-}rea: R1(II_C) = II_C$
by $(rel\text{-}auto)$

lemma $R2c\text{-}skip\text{-}rea: R2c\ II_C = II_C$
by $(simp\ add: skip\text{-}rea\text{-}def\ R2c\text{-}and\ R2c\text{-}disj\ R2c\text{-}skip\text{-}r\ R2c\text{-}not\ R2c\text{-}ok\ R2c\text{-}tr\text{-}le\text{-}tr')$

lemma $R2\text{-}skip\text{-}rea: R2(II_C) = II_C$
by $(metis\ R1\text{-}R2c\text{-}is\text{-}R2\ R1\text{-}skip\text{-}rea\ R2c\text{-}skip\text{-}rea)$

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

lemma $skip\text{-}srea\text{-}R1 \text{ [closure]}: II_R \text{ is } R1$
by $(rel\text{-}auto)$

lemma $skip\text{-}srea\text{-}R2c \text{ [closure]}: II_R \text{ is } R2c$
by $(simp\ add: Healthy\text{-}def\ R2c\text{-}skip\text{-}srea)$

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

2.3 RD1: Divergence yields arbitrary traces

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

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

lemma $RD1\text{-}idem: RD1(RD1(P)) = RD1(P)$
by $(rel\text{-}auto)$

lemma $RD1\text{-}Idempotent: Idempotent\ RD1$
by $(simp\ add: Idempotent\text{-}def\ RD1\text{-}idem)$

lemma $RD1\text{-}mono: P \sqsubseteq Q \Longrightarrow RD1(P) \sqsubseteq RD1(Q)$

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 false/\$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 \$ok \triangleright R1(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
 $P \text{ is R1 } (R1(true_h) ;; P) = R1(true_h) (II_C ;; P) = P$
 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 $\dots = (((\neg \$ok \wedge R1(true)) ;; P) \vee P)$
 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 $\dots = ((R1(\neg \$ok) ;; (R1(true_h) ;; P)) \vee P)$
 using dual-order.trans by (rel-blast)

also have ... = $((R1(\neg \$ok) ;; R1(true_h)) \vee P)$
by (*simp add: assms(2)*)
also have ... = $(R1(\neg \$ok) \vee P)$
by (*rel-auto*)
also have ... = $RD1(P)$
by (*rel-auto*)
finally show *?thesis*
by (*simp add: Healthy-def*)
qed

theorem *RD1-left-zero*:
assumes *P is R1 P 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 $RD1(P) = (P \triangleleft \$ok \triangleright R1(true))$
proof –
have $RD1(R1(P)) = (R1(P) \triangleleft \$ok \triangleright R1(true))$
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 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 $(P ;; Q) \text{ is } R3c$
by (*metis Healthy-def' R3c-semir-form assms*)

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

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

assumes *P is R3 Q is R3c*

shows $(P ;; Q) \text{ is } R3c$

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*)

$vwb\text{-}lens\text{-}mwb\ wait\text{-}vwb\text{-}lens)$
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*: $R3c(P) = ((II \triangleleft \$ok \triangleright R1(true)) \triangleleft \$wait \triangleright P)$
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
 $[upred\text{-}defs]: RD2(P) = H2(P)$

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* :: $(t::trace, \alpha) \text{ hrel-rp} \Rightarrow (t, \alpha) \text{ hrel-rp}$ (**R**)
where *[upred-defs]:* $RH(P) = R1(R2c(R3c(P)))$

definition *RHS* :: $(s, t::trace, \alpha) \text{ hrel-rsp} \Rightarrow (s, t, \alpha) \text{ hrel-rsp}$ (**R_s**)
where *[upred-defs]:* $RHS(P) = R1(R2c(R3h(P)))$

definition *RD* :: $(t::trace, \alpha) \text{ hrel-rp} \Rightarrow (t, \alpha) \text{ hrel-rp}$
where *[upred-defs]:* $RD(P) = RD1(RD2(RP(P)))$

definition *SRD* :: $(s, t::trace, \alpha) \text{ hrel-rsp} \Rightarrow (s, t, \alpha) \text{ hrel-rsp}$
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* \mathbf{R}_s
 by (simp add: Idempotent-def RHS-idem)

lemma *RHS-Monotonic*: *Monotonic* \mathbf{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* \mathbf{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-UINF 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]*: $P \text{ is } SRD \implies P \llbracket \text{false}/\$ok \rrbracket = R1(true)$
 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 is *SRD* $\implies R1(true) ;; P = R1(true)$
 by (*simp add: RD1-left-zero SRD-healths(1) SRD-healths(4)*)

lemma *SRD-left-zero-2*:

assumes P is *SRD*

shows $(\exists \$st \cdot II)[[true, true/\$ok, \$wait]] ;; P = (\exists \$st \cdot II)[[true, true/\$ok, \$wait]]$

proof –

have $(\exists \$st \cdot II)[[true, true/\$ok, \$wait]] ;; R3h(P) = (\exists \$st \cdot II)[[true, true/\$ok, \$wait]]$

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.

interpretation *rdes-theory*: *utp-theory-continuous RD*

rewrites $P \in \text{carrier } rdes\text{-theory.thy-order} \longleftrightarrow P \text{ is } RD$

and $\text{carrier } rdes\text{-theory.thy-order} \rightarrow \text{carrier } rdes\text{-theory.thy-order} \equiv \llbracket RD \rrbracket_H \rightarrow \llbracket RD \rrbracket_H$

and $le\ rdes\text{-theory.thy-order} = (\sqsubseteq)$

and $eq\ rdes\text{-theory.thy-order} = (=)$

proof –

show *utp-theory-continuous RD*

by (*unfold-locales, simp-all add: RD-idem RD-Continuous*)

qed (*simp-all*)

interpretation *srdes-theory*: *utp-theory-continuous SRD*

rewrites $P \in \text{carrier } srdes\text{-theory.thy-order} \longleftrightarrow P \text{ is } SRD$

and $\text{carrier } srdes\text{-theory.thy-order} \rightarrow \text{carrier } srdes\text{-theory.thy-order} \equiv \llbracket SRD \rrbracket_H \rightarrow \llbracket SRD \rrbracket_H$

and $le\ srdes\text{-theory.thy-order} = (\sqsubseteq)$

and $eq\ srdes\text{-theory.thy-order} = (=)$

proof –

show *utp-theory-continuous SRD*

by (*unfold-locales, simp-all add: SRD-idem SRD-Continuous*)

qed (*simp-all*)

interpretation *rdes-rea-galois*:

galois-connection $(RD \Leftarrow (RD1 \circ RD2, R3) \Rightarrow RP)$

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

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 $P :: ('a, 'b) \text{ hrel-rp}$

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 $P :: ('a, 'b) \text{ hrel-rp}$

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 ( $RD \Leftarrow \langle RD1 \circ RD2, R3 \rangle \Rightarrow RP$ )
  by (unfold-locales, simp-all add: mk-conn-def utp-partial-order)
    (metis Healthy-if R3-RD-RP RD-def RP-idem eq-refl)

```

```

abbreviation Chaos :: ('s, 't::trace, 'α) hrel-rsp where
Chaos  $\equiv$  srdes-theory.utp-bottom

```

```

abbreviation Miracle :: ('s, 't::trace, 'α) hrel-rsp where
Miracle  $\equiv$  srdes-theory.utp-top

```

```

thm srdes-theory.weak.bottom-lower
thm srdes-theory.weak.top-higher
thm srdes-theory.meet-bottom
thm srdes-theory.meet-top

```

```

abbreviation srd-lfp ( $\mu_R$ ) where  $\mu_R F \equiv$  srdes-theory.utp-lfp F

```

```

abbreviation srd-gfp ( $\nu_R$ ) where  $\nu_R F \equiv$  srdes-theory.utp-gfp 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-theory.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 $\text{Chaos} = \text{SRD}(\text{true})$
by (*metis srdes-theory.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 $\text{Miracle} = \text{SRD}(\text{false})$
by (*metis srdes-theory.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*: $\text{RD1}(\mathbf{R}(P \vdash Q)) = \mathbf{R}(P \vdash Q)$
by (*rel-auto*)

lemma *RD2-reactive-design*:
assumes $\$ok' \nmid P \ \$ok' \nmid Q$
shows $\text{RD2}(\mathbf{R}(P \vdash Q)) = \mathbf{R}(P \vdash Q)$
using *assms*
by (*metis H2-design RD2-RH-commute RD2-def*)

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

lemma *RD2-st-reactive-design*:
assumes $\$ok' \nmid P \ \$ok' \nmid Q$
shows $\text{RD2}(\mathbf{R}_s(P \vdash Q)) = \mathbf{R}_s(P \vdash Q)$
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*:
 $\text{RD}(P) = \mathbf{R}((\neg P_f) \vdash P^t_f)$
proof –
have $\text{RD}(P) = \text{RD1}(\text{RD2}(\text{R1}(\text{R2c}(\text{R3c}(P))))$
by (*simp add: RD-alt-def RH-def*)
also have $\dots = \text{RD1}(\text{H2}(\text{R1}(\text{R2s}(\text{R3c}(P))))$

by (simp add: R1-R2s-R2c RD2-def)
 also have ... = $RD1(R1(H2(R2s(R3c(P)))))$
 by (simp add: R1-H2-commute)
 also have ... = $R1(H1(R1(H2(R2s(R3c(P))))))$
 by (simp add: R1-idem RD1-via-R1)
 also have ... = $R1(H1(H2(R2s(R3c(R1(P))))))$
 by (simp add: R1-H2-commute R1-R2c-commute R1-R2s-R2c R1-R3c-commute RD1-via-R1)
 also have ... = $R1(R2s(H1(H2(R3c(R1(P))))))$
 by (simp add: R2s-H1-commute R2s-H2-commute)
 also have ... = $R1(R2s(H1(R3c(H2(R1(P))))))$
 by (metis RD2-R3c-commute RD2-def)
 also have ... = $R2(R1(H1(R3c(H2(R1(P))))))$
 by (metis R1-R2-commute R1-idem R2-def)
 also have ... = $R2(R3c(R1(H(R1(P)))))$
 by (simp add: R1-R3c-commute RD1-R3c-commute RD1-via-R1)
 also have ... = $RH(H(R1(P)))$
 by (metis R1-R2s-R2c R1-R3c-commute R2-R1-form RH-def)
 also have ... = $RH(H(P))$
 by (simp add: R1-H2-commute R1-R2c-commute R1-R3c-commute R1-idem RD1-via-R1 RH-def)
 also have ... = $RH((\neg P^f) \vdash P^t)$
 by (simp add: H1-H2-eq-design)
 also have ... = $\mathbf{R}((\neg P^f_f) \vdash P^t_f)$
 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 $\mathbf{R}((\neg P^f_f) \vdash P^t_f) = P$
 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*:

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

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 ... = $R1(R2s(R3h(H(P))))$
 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 ... = $\mathbf{R}_s(H(P))$
 by (simp add: R1-R2s-R2c RHS-def)
 also have ... = $\mathbf{R}_s((\neg P^f) \vdash P^t)$
 by (simp add: H1-H2-eq-design)
 also have ... = $\mathbf{R}_s((\neg P^f_f) \vdash P^t_f)$

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 $(R1(P \vdash Q) ;; R1(R \vdash S)) = ((R1(P \vdash Q))^t ;; R1(R \vdash S)[\text{true}/\$ok] \vee (R1(P \vdash Q))^f ;; R1(R \vdash S)[\text{false}/\$ok])$

by (*rule seqr-bool-split[of ok], simp*)

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: design-def usubst R1-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 $((R1(\neg \$ok) :: ('t, 'α, 'β)\ rel\text{-}rp) ;; R1(true)) = (R1(\neg \$ok) :: ('t, 'α, 'γ)\ rel\text{-}rp)$

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 (*simp add: R1-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 (simp add: segr-or-distr utp-pred-laws.sup.assoc)
 also have ... = $R1((R1(Q) ;; (R1(\neg R)))$
 $\vee (R1(Q) ;; (R1(S \wedge \$ok'))$
 $\vee (\neg \$ok)$
 $\vee (R1(\neg P) ;; R1(true))$
 by (simp add: R1-disj R1-segr)
 also have ... = $R1((R1(Q) ;; (R1(\neg R)))$
 $\vee ((R1(Q) ;; R1(S)) \wedge \$ok')$
 $\vee (\neg \$ok)$
 $\vee (R1(\neg P) ;; R1(true))$
 by (rel-blast)
 also have ... = $R1(\neg(\$ok \wedge \neg (R1(\neg P) ;; R1(true)) \wedge \neg (R1(Q) ;; (R1(\neg R))))$
 $\vee ((R1(Q) ;; R1(S)) \wedge \$ok')$
 by (rel-blast)
 also have ... = $R1((\$ok \wedge \neg (R1(\neg P) ;; R1(true)) \wedge \neg (R1(Q) ;; (R1(\neg R))))$
 $\Rightarrow (\$ok' \wedge (R1(Q) ;; R1(S))))$
 by (simp add: impl-alt-def utp-pred-laws.inf-commute)
 also have ... = $R1((\neg (R1(\neg P) ;; R1(true)) \wedge \neg (R1(Q) ;; R1(\neg R))) \vdash (R1(Q) ;; R1(S)))$
 by (simp add: design-def)
 finally show ?thesis .
 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)

lemma *R2s-design:* $R2s(P \vdash Q) = (R2s(P) \vdash R2s(Q))$

by (simp add: R2s-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 $(R3c(R1(P \vdash Q)) ;; R3c(R1(R \vdash S))) =$

$$R3c(R1((\neg (R1(\neg P) ;; R1(true)) \wedge \neg ((R1(Q) \wedge \neg \$wait') ;; R1(\neg R)))$$

$$\vdash (R1(Q) ;; (\lceil II \rceil_D \triangleleft \$wait \triangleright R1(S))))$$

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: $(R1(R3c\text{-}post\ Q) ;; R1(R3c\text{-}post\ S)) = R3c\text{-}post(R1\ Q ;; (\lceil II \rceil_D \triangleleft \$wait \triangleright R1\ S))$
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 $(R3h(R1(P \vdash Q)) ;; R3h(R1(R \vdash S))) =$
 $R3h(R1((\neg (R1(\neg P) ;; R1(true)) \wedge \neg ((R1(Q) \wedge \neg \$wait') ;; R1(\neg R)))$
 $\vdash (R1(Q) ;; ((\exists \$st \cdot \lceil II \rceil_D) \triangleleft \$wait \triangleright R1(S))))$

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: $(R1(R3h\text{-}post\ Q) ;; R1(R3h\text{-}post\ S)) = R3h\text{-}post(R1\ Q ;; ((\exists \$st \cdot \lceil II \rceil_D) \triangleleft \$wait \triangleright R1\ S))$
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 (¬ P) ;; R2 true)) = R2c(R1(R2 (¬ P) ;; R2 true))
  using R2c-not by blast
also have ... = R2(R2 (¬ P) ;; R2 true)
  by (metis R1-R2c-commute R1-R2c-is-R2)
also have ... = (R2 (¬ P) ;; R2 true)
  by (simp add: R2-seqr-distribute)
also have ... = (R1 (¬ 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) ∧ ¬ $wait') ;; R1 (¬ R2s R)) = ((R1 (R2s Q) ∧ ¬ $wait') ;; R1 (¬ R2s
R))
proof -
  have ((R1 (R2s Q) ∧ ¬ $wait') ;; R1 (¬ R2s R)) = R1 (R2 (Q ∧ ¬ $wait') ;; R2 (¬ R))
    by (rel-auto)
  hence R2c ((R1 (R2s Q) ∧ ¬ $wait') ;; R1 (¬ R2s R)) = (R2 (Q ∧ ¬ $wait') ;; R2 (¬ R))
    by (metis R1-R2c-commute R1-R2c-is-R2 R2-seqr-distribute)
  also have ... = ((R1 (R2s Q) ∧ ¬ $wait') ;; R1 (¬ R2s R))
    by (rel-auto)
  finally show ?thesis .
qed

have 3:R2c((R1 (R2s Q) ;; (⌈II⌋D ◁ $wait ▷ R1 (R2s S)))) = (R1 (R2s Q) ;; (⌈II⌋D ◁ $wait ▷ R1
(R2s S)))
proof -
  have R2c(((R1 (R2s Q))⌈true/$wait'⌋ ;; (⌈II⌋D ◁ $wait ▷ R1 (R2s S))⌈true/$wait⌋))
    = ((R1 (R2s Q))⌈true/$wait'⌋ ;; (⌈II⌋D ◁ $wait ▷ R1 (R2s S))⌈true/$wait⌋)
  proof -
    have R2c(((R1 (R2s Q))⌈true/$wait'⌋ ;; (⌈II⌋D ◁ $wait ▷ R1 (R2s S))⌈true/$wait⌋)) =
      R2c(R1 (R2s (Q⌈true/$wait'⌋)) ;; ⌈II⌋D⌈true/$wait⌋)
      by (simp add: usubst cond-unit-T R1-def R2s-def)
    also have ... = R2c(R2(Q⌈true/$wait'⌋) ;; R2(⌈II⌋D⌈true/$wait⌋))
      by (metis R2-def R2-des-lift-skip R2-subst-wait-true)
    also have ... = (R2(Q⌈true/$wait'⌋) ;; R2(⌈II⌋D⌈true/$wait⌋))
      using R2c-seq by blast
    also have ... = ((R1 (R2s Q))⌈true/$wait'⌋ ;; (⌈II⌋D ◁ $wait ▷ R1 (R2s S))⌈true/$wait⌋)
      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))⌈false/$wait'⌋ ;; (⌈II⌋D ◁ $wait ▷ R1 (R2s S))⌈false/$wait⌋))
    = ((R1 (R2s Q))⌈false/$wait'⌋ ;; (⌈II⌋D ◁ $wait ▷ R1 (R2s S))⌈false/$wait⌋)
    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 ◁ $wait ▷ R1 (R2s S) = R2 (⌈II⌋D ◁ $wait ▷ 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 $\dots = R3c(R1((\neg(R1(\neg R2s P) \;; R1 true) \wedge \neg((R1(R2s Q) \wedge \neg \$wait') \;; R1(\neg R2s R)))) \vdash$
 $(R1(R2s Q) \;; ([II]_D \triangleleft \$wait \triangleright R1(R2s S))))$
 by *(simp add: R3c-R1-design-composition assms unrest)*
 also have $\dots = R3c(R1(R2c((\neg(R1(\neg R2s P) \;; R1 true) \wedge \neg((R1(R2s Q) \wedge \neg \$wait') \;; R1(\neg R2s R)))) \vdash$
 $(R1(R2s Q) \;; ([II]_D \triangleleft \$wait \triangleright R1(R2s S))))$
 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 $\dots = R2(R2(\neg P) \;; R2 true)$

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, 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 $R2c(((R1(R2s Q))\llbracket true/\$wait' \rrbracket \;; ((\exists \$st \cdot [II]_D \triangleleft \$wait \triangleright R1(R2s S))\llbracket true/\$wait' \rrbracket))$

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

proof –

have $R2c(((R1 (R2s Q))\llbracket true/\$wait' \rrbracket ;; ((\exists \$st \cdot \lceil II \rceil_D) \triangleleft \$wait \triangleright R1 (R2s S))\llbracket true/\$wait \rrbracket)) =$
 $R2c(R1 (R2s (Q\llbracket true/\$wait' \rrbracket)) ;; (\exists \$st \cdot \lceil II \rceil_D)\llbracket true/\$wait \rrbracket)$
by (*simp add: usubst cond-unit-T R1-def R2s-def*)
also have $\dots = R2c(R2(Q\llbracket true/\$wait' \rrbracket) ;; R2((\exists \$st \cdot \lceil II \rceil_D)\llbracket true/\$wait \rrbracket))$
by (*metis (no-types, lifting) R2-def R2-des-lift-skip R2-subst-wait-true R2-st-ex*)
also have $\dots = (R2(Q\llbracket true/\$wait' \rrbracket) ;; R2((\exists \$st \cdot \lceil II \rceil_D)\llbracket true/\$wait \rrbracket))$
using *R2c-seq* **by** *blast*
also have $\dots = ((R1 (R2s Q))\llbracket true/\$wait' \rrbracket ;; ((\exists \$st \cdot \lceil II \rceil_D) \triangleleft \$wait \triangleright R1 (R2s S))\llbracket true/\$wait \rrbracket)$
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 $R2c(((R1 (R2s Q))\llbracket false/\$wait' \rrbracket ;; ((\exists \$st \cdot \lceil II \rceil_D) \triangleleft \$wait \triangleright R1 (R2s S))\llbracket false/\$wait \rrbracket)) =$
 $((R1 (R2s Q))\llbracket false/\$wait' \rrbracket ;; ((\exists \$st \cdot \lceil II \rceil_D) \triangleleft \$wait \triangleright R1 (R2s S))\llbracket false/\$wait \rrbracket)$
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 $\dots = R3h(R1((\neg (R1(\neg R2s P) ;; R1 true) \wedge \neg ((R1(R2s Q) \wedge \neg \$wait') ;; R1(\neg$
 $R2s R))) \vdash$
 $(R1(R2s Q) ;; ((\exists \$st \cdot \lceil II \rceil_D) \triangleleft \$wait \triangleright R1(R2s S))))$
by (*simp add: R3h-R1-design-composition assms unrest*)
also have $\dots = R3h(R1(R2c((\neg (R1(\neg R2s P) ;; R1 true) \wedge \neg ((R1(R2s Q) \wedge \neg \$wait') ;; R1(\neg$
 $R2s R))) \vdash$
 $(R1(R2s Q) ;; ((\exists \$st \cdot \lceil II \rceil_D) \triangleleft \$wait \triangleright R1(R2s S))))$
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-design-export-R1*: $\mathbf{R}(P \vdash Q) = \mathbf{R}(P \vdash R1(Q))$
 by (*rel-auto*)

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

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

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

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

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

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

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

lemma *RHS-design-ok-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-auto*)

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

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

lemma *RHS-pre-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
 $P_1 \text{ is } R1 \ P_2 \text{ is } R1 \ Q_1 \text{ is } R1 \ Q_2 \text{ is } R1$
 $\$ok \ \# \ P_1 \ \$ok' \ \# \ P_1 \ \$ok \ \# \ P_2 \ \$ok' \ \# \ P_2$
 $\$ok \ \# \ Q_1 \ \$ok' \ \# \ Q_1 \ \$ok \ \# \ Q_2 \ \$ok' \ \# \ Q_2$
 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 $(\prod i \in A \cdot \mathbf{R}_s(P(i) \vdash Q(i))) = \mathbf{R}_s((\prod i \in A \cdot P(i)) \vdash (\prod i \in A \cdot Q(i)))$
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 60) **where**
 $[\text{upred-defs}]: P \diamond Q = (P \triangleleft \$\text{wait}' \triangleright Q)$

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

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

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

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

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

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

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

4.3.3 Substitution laws

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

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

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

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

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

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

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

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

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

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

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

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

lemma *pre-wait-false*:
 $pre_R(P[\$false/\$wait]) = pre_R(P)$
 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*: $pre_R(\mathbf{R}_s(P \vdash Q)) = R1(R2c(pre_s \dagger P))$
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*: $cmt_R(\mathbf{R}_s(P \vdash Q)) = R1(R2c(cmt_s \dagger (P \Rightarrow Q)))$
by (*simp add: RHS-def usubst R3h-def cmt_R-def cmt_s-design R1-idem*)

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

lemma *rea-post-RHS-design*: $post_R(\mathbf{R}_s(P \vdash Q \diamond R)) = R1(R2c(post_s \dagger (P \Rightarrow_r R)))$
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*: $\mathbf{R}_s(P \vdash cmt_s \dagger Q) = \mathbf{R}_s(P \vdash Q)$
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*]:
 $\$wait' \nmid pre_R(P) \implies \$wait' \nmid pre_R(SRD\ P)$
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 $R1(R2s(peri_R(P))) = peri_R(P)$
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 $R1(peri_R(P)) = peri_R(P)$
proof –
have $R1(peri_R(P)) = R1(R1(R2s(peri_R(P))))$
by (*simp add: R1-R2s-peri-SRD assms*)
also have $\dots = peri_R(P)$
by (*simp add: R1-idem, simp add: R1-R2s-peri-SRD assms*)
finally show *?thesis* .

qed

lemma *periR-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 *postR-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 $R1(R2c(\text{post}_R(P))) = \text{post}_R(P)$
 by (metis R1-R2s-R2c R1-R2s-post-SRD assms)

lemma *R2-cmt-conj-wait'*:
 P is SRD $\implies R2(\text{cmt}_R P \wedge \neg \$\text{wait}') = (\text{cmt}_R P \wedge \neg \$\text{wait}')$
 by (simp add: R2-def R2s-conj R2s-not R2s-wait' R1-extend-conj R1-R2s-cmt-SRD)

lemma *R2c-preR*:
 P is SRD $\implies R2c(\text{pre}_R(P)) = \text{pre}_R(P)$
 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 $\text{pre}_R(P)$ is R2c
 by (simp add: Healthy-def' R2c-preR)

lemma *R2c-periR*:

$P \text{ is } SRD \implies R2c(per_i_R(P)) = per_i_R(P)$

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

lemma *periR-R2c-closed* [*closure*]: $P \text{ is } SRD \implies per_i_R(P) \text{ is } R2c$

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

lemma *R2c-postR*:

$P \text{ is } SRD \implies R2c(post_R(P)) = post_R(P)$

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*]: $P \text{ is } SRD \implies post_R(P) \text{ is } R2c$

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

lemma *periR-RR* [*closure*]: $P \text{ is } SRD \implies per_i_R(P) \text{ is } RR$

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

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

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

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

$(\$tr' =_u \$tr \wedge [II]_R) \text{ wp}_r \text{ pre}_R P = \text{pre}_R P$

by (*rel-auto*)

lemma *R1-preR* [*closure*]:

$pre_R(P) \text{ is } R1$

by (*rel-auto*)

lemma *trace-ident-left-periR*:

$(\$tr' =_u \$tr \wedge [II]_R) ;; per_i_R(P) = per_i_R(P)$

by (*rel-auto*)

lemma *trace-ident-left-postR*:

$(\$tr' =_u \$tr \wedge [II]_R) ;; post_R(P) = post_R(P)$

by (*rel-auto*)

lemma *trace-ident-right-postR*:

$post_R(P) ;; (\$tr' =_u \$tr \wedge [II]_R) = post_R(P)$

by (*rel-auto*)

lemma *preR-R2-closed* [*closure*]: $P \text{ is } SRD \implies pre_R(P) \text{ is } R2$

by (*simp* *add*: *R2-comp-def* *Healthy-comp* *closure*)

lemma *periR-R2-closed* [*closure*]: $P \text{ is } SRD \implies per_i_R(P) \text{ is } R2$

by (*simp* *add*: *Healthy-def'* *R1-R2c-peri-RHS* *R2-R2c-def*)

lemma *postR-R2-closed* [*closure*]: $P \text{ is } SRD \implies post_R(P) \text{ is } R2$

by (*simp* *add*: *Healthy-def'* *R1-R2c-post-RHS* *R2-R2c-def*)

4.3.5 Calculation laws

lemma *wait'-cond-peri-post-cmt* [*rdes*]:

$cmt_R P = per_i_R P \diamond post_R P$

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*]: $A \neq \{\} \implies \text{pre}_R(\bigcap A) = (\bigwedge P \in A \cdot \text{pre}_R(P))$
by (*rel-auto*)

lemma *periR-INF* [*rdes*]: $\text{peri}_R(\bigcap A) = (\bigvee P \in A \cdot \text{peri}_R(P))$
by (*rel-auto*)

lemma *postR-INF* [*rdes*]: $\text{post}_R(\bigcap A) = (\bigvee P \in A \cdot \text{post}_R(P))$
by (*rel-auto*)

lemma *preR-UINF* [*rdes*]: $\text{pre}_R(\bigcap i \cdot P(i)) = (\bigcap i \cdot \text{pre}_R(P(i)))$
by (*rel-auto*)

lemma *periR-UINF* [*rdes*]: $\text{peri}_R(\bigcap i \cdot P(i)) = (\bigcap i \cdot \text{peri}_R(P(i)))$

by (rel-auto)

lemma *postR-UINF* [rdes]: $\text{post}_R(\sqcap i \cdot P(i)) = (\sqcap i \cdot \text{post}_R(P(i)))$
by (rel-auto)

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

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

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

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

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

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

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

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

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

lemma *preR-inf* [rdes]: $\text{pre}_R(P \sqcap Q) = (\text{pre}_R(P) \wedge \text{pre}_R(Q))$
by (rel-auto)

lemma *periR-inf* [rdes]: $\text{peri}_R(P \sqcap Q) = (\text{peri}_R(P) \vee \text{peri}_R(Q))$
by (rel-auto)

lemma *postR-inf* [rdes]: $\text{post}_R(P \sqcap Q) = (\text{post}_R(P) \vee \text{post}_R(Q))$
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 assms)

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

by (simp add: design-subst-ok)

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

by (simp add: pre_R-def cmt_R-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 assms)

qed

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

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

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

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

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

proof –

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

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

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

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 $\mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \diamond \text{post}_R(P)) = P$
 by (*metis Healthy-if SRD-as-reactive-tri-design assms*)

lemma *SRD-elim* [*RD-elim*]: $\llbracket P \text{ is SRD}; Q(\mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \diamond \text{post}_R(P))) \rrbracket \implies Q(P)$
 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 $A \subseteq \llbracket \text{SRD} \rrbracket_H$
 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 $A \subseteq \llbracket \text{SRD} \rrbracket_H$
 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*: $P \sqsubseteq Q \implies \text{pre}_R(Q) \sqsubseteq \text{pre}_R(P)$
 by (*rel-auto*)

lemma *periR-monotone*: $P \sqsubseteq Q \implies \text{peri}_R(P) \sqsubseteq \text{peri}_R(Q)$
 by (*rel-auto*)

lemma *postR-monotone*: $P \sqsubseteq Q \implies \text{post}_R(P) \sqsubseteq \text{post}_R(Q)$
 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 $(RH(P \vdash Q_1 \diamond Q_2) ;; RH(R \vdash S_1 \diamond S_2)) =$
 $RH((\neg (R1 (\neg R2s P) ;; R1 \text{ true}) \wedge \neg ((R1 (R2s Q_2) \wedge \neg \$wait') ;; R1 (\neg R2s R))) \vdash$
 $((Q_1 \vee (R1 (R2s Q_2) ;; R1 (R2s S_1))) \diamond ((R1 (R2s Q_2) ;; R1 (R2s S_2))))$

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 ... $= ((R1 \ (R2s \ Q_1) ;; [II]_D) \wedge \$wait')$
 by *(rel-auto)*
 also from *assms(2)* have ... $= ((R1 \ (R2s \ Q_1)) \wedge \$wait')$
 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 ... $= ((R1 \ (R2s \ Q_2))\llbracket false/\$wait' \rrbracket ;; (R1 \ (R2s \ S_1) \diamond R1 \ (R2s \ S_2))\llbracket false/\$wait' \rrbracket)$
 by *(metis false-alt-def seqr-right-one-point upred-eq-false wait-vwb-lens)*

 also have ... $= ((R1 \ (R2s \ Q_2)) ;; (R1 \ (R2s \ S_1) \diamond R1 \ (R2s \ S_2)))$
 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 $\dots = ((R1 (R2s Q_2)) \llbracket \text{false} / \$wait' \rrbracket ;; (R1 (R2s S_1) \diamond R1 (R2s S_2)) \llbracket \text{false} / \$wait \rrbracket)$

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 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  $\mathbf{R}_s(P \vdash Q_1 \diamond Q_2) \ ;\ ; \mathbf{R}_s(R \vdash S_1 \diamond S_2) =$ 
   $\mathbf{R}_s(((\neg_r P) \text{ wp}_r \text{ false} \wedge Q_2 \text{ wp}_r R) \vdash (((\exists \$st' \cdot Q_1) \sqcap (Q_2 \ ;\ ; S_1)) \diamond (Q_2 \ ;\ ; S_2)))$  (is ?lhs =
?rhs)
proof –
  have ?lhs =  $\mathbf{R}_s((\neg R1 \ (\neg P) \ ;\ ; R1 \ \text{true} \wedge \neg Q_2 \ ;\ ; R1 \ (\neg R)) \vdash ((\exists \$st' \cdot Q_1) \sqcap (Q_2 \ ;\ ; S_1)) \diamond (Q_2 \ ;\ ; S_2))$ 
  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 ... = ?rhs
  by (rel-auto)
  finally show ?thesis .
qed

theorem RHS-tri-design-composition-RR-wp:
assumes  $P \text{ is } RR \ Q_1 \text{ is } RR \ Q_2 \text{ is } RR$ 
   $R \text{ is } RR \ S_1 \text{ is } RR \ S_2 \text{ is } RR$ 
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
   $\$ok' \# P \ \$ok' \# Q_1 \ \$ok' \# Q_2 \ \$ok \# R \ \$ok \# S_1 \ \$ok \# S_2$ 

```

$\$wait \# R \$wait' \# Q_2 \$wait \# S_1 \$wait \# S_2$
 $P \text{ is } R2c \ Q_1 \text{ is } R1 \ Q_1 \text{ is } R2c \ Q_2 \text{ is } R1 \ Q_2 \text{ is } R2c$
 $R \text{ is } R2c \ S_1 \text{ is } R1 \ S_1 \text{ is } R2c \ S_2 \text{ is } R1 \ S_2 \text{ is } R2c$
 $R1 (\neg P) ;; R1(true) = R1(\neg P) \$st' \# Q_1$
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 $P \text{ is } RC \ Q_1 \text{ is } RR \ \$st' \# Q_1 \ Q_2 \text{ is } RR \ R \text{ is } RR \ S_1 \text{ is } RR \ S_2 \text{ is } RR$
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

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 $\dots = \mathbf{R}_s((\neg R1 (\neg R2s P) ;; R1 \text{ true}) \vdash (\exists \$st' \cdot Q) \diamond (R1 (R2s R) ;; R1 (R2s (\$tr' =_u \$tr \wedge [II]_R))))$
by (*simp-all add: RHS-tri-design-composition assms unrest R2s-true R1-false R2s-false*)
also have $\dots = \mathbf{R}_s((\neg R1 (\neg R2s P) ;; R1 \text{ true}) \vdash (\exists \$st' \cdot Q) \diamond R1 (R2s R))$
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 $\dots = \mathbf{R}_s((\neg (\neg P) ;; R1 \text{ true}) \vdash ((\exists \$st' \cdot Q) \diamond R))$
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 $\dots = \mathbf{R}_s((\neg_r (\neg_r P) ;; \text{true}_r) \vdash ((\exists \$st' \cdot Q) \diamond R))$
by (*rel-auto*)
finally show *?thesis* .
qed

lemma *SRD-composition-wp*:
assumes $P \text{ is } SRD \ Q \text{ is } SRD$
shows $(P ;; Q) = \mathbf{R}_s(((\neg_r \text{pre}_R P) \text{ wp}_r \text{ false} \wedge \text{post}_R P \text{ wp}_r \text{pre}_R Q) \vdash$

$((\exists \$st' \cdot \text{peri}_R P) \vee (\text{post}_R P ;; \text{peri}_R Q)) \diamond (\text{post}_R P ;; \text{post}_R Q))$
 (is ?lhs = ?rhs)
proof –
 have $(P ;; Q) = (\mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \diamond \text{post}_R(P)) ;; \mathbf{R}_s(\text{pre}_R(Q) \vdash \text{peri}_R(Q) \diamond \text{post}_R(Q)))$
 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 $?lhs \longleftrightarrow 'P_1 \Rightarrow Q_1' \wedge 'P_1 \wedge Q_2 \Rightarrow P_2 \diamond P_3'$
 by (simp add: RHS-design-refine assms closure RR-implies-R2c unrest ex-unrest)
 also have $\dots \longleftrightarrow 'P_1 \Rightarrow Q_1' \wedge '(P_1 \wedge Q_2) \diamond (P_1 \wedge Q_3) \Rightarrow P_2 \diamond P_3'$
 by (rel-auto)
 also have $\dots \longleftrightarrow 'P_1 \Rightarrow Q_1' \wedge '((P_1 \wedge Q_2) \diamond (P_1 \wedge Q_3) \Rightarrow P_2 \diamond P_3)[\text{true}/\$wait']' \wedge '((P_1 \wedge Q_2) \diamond (P_1 \wedge Q_3) \Rightarrow P_2 \diamond P_3)[\text{false}/\$wait']'$
 by (rel-auto, metis)
 also have $\dots \longleftrightarrow ?rhs$
 by (simp add: usubst unrest assms)
 finally show ?thesis .
qed

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 $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) \sqsubseteq \mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3) \longleftrightarrow (Q_1 \sqsubseteq P_1) \wedge (P_2 \sqsubseteq (P_1 \wedge Q_2)) \wedge (P_3 \sqsubseteq (P_1 \wedge Q_3))$
 by (simp add: RHS-tri-design-refine assms, rel-auto)

lemma *srdes-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 srdes-refine-intro, simp-all, rel-auto)

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

assumes $P_2 \sqsubseteq P_1 \wedge Q_1 \sqsubseteq (P_1 \wedge Q_2) \wedge R_1 \sqsubseteq (P_1 \wedge R_2)$
 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 $\$wait' \nmid \text{pre}_R(P)$
 shows $(\text{pre}_R(P) \Rightarrow_r \text{peri}_R(P)) = \text{peri}_R(P)$
proof –
 have $\text{peri}_R(P) =$
 $\text{peri}_R(\mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \diamond \text{post}_R(P)))$
 by (simp add: SRD-reactive-tri-design assms)

also have ... = ($pre_R P \Rightarrow_r peri_R P$)
 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 ... = ($pre_R P \Rightarrow_r post_R P$)
 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*)

lemma *srdes-tri-eq-iff*:

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 $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) = \mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3) \longleftrightarrow (P_1 = Q_1 \wedge (P_1 \wedge Q_2) = (Q_1 \wedge P_2) \wedge (P_1 \wedge Q_3) = (Q_1 \wedge P_3))$
 proof –
 have $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) = \mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3) \longleftrightarrow$
 $(\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) \sqsubseteq \mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3) \wedge \mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3) \sqsubseteq \mathbf{R}_s(P_1 \vdash P_2 \diamond P_3))$
 by *fastforce*
 also have ... = $(Q_1 \sqsubseteq P_1 \wedge P_2 \sqsubseteq (P_1 \wedge Q_2) \wedge P_3 \sqsubseteq (P_1 \wedge Q_3) \wedge P_1 \sqsubseteq Q_1 \wedge Q_2 \sqsubseteq (Q_1 \wedge P_2) \wedge Q_3 \sqsubseteq (Q_1 \wedge P_3))$
 by (*simp add: RHS-tri-design-refine' assms*)
 also have ... = $(P_1 = Q_1 \wedge P_2 \sqsubseteq (P_1 \wedge Q_2) \wedge P_3 \sqsubseteq (P_1 \wedge Q_3) \wedge Q_2 \sqsubseteq (Q_1 \wedge P_2) \wedge Q_3 \sqsubseteq (Q_1 \wedge P_3))$
 by *fastforce*
 also have ... = $(P_1 = Q_1 \wedge (P_1 \wedge Q_2) = (Q_1 \wedge P_2) \wedge (P_1 \wedge Q_3) = (Q_1 \wedge P_3))$
 apply (*safe, simp-all*)

apply (*meson eq-iff utp-pred-laws.inf-greatest utp-pred-laws.inf-le1*) +
apply (*metis utp-pred-laws.inf-le2*) +
done
finally show *?thesis* .
qed

lemma *srdes-tri-eq-intro*:

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

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

by (*metis (no-types, hide-lams) assms(1) assms(2) assms(3) design-export-pre wait'-cond-conj-exchange wait'-cond-idem*)

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

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

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*)

4.7 Closure laws

lemma *SRD-srdes-skip [closure]*: Π_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 ... = (\prod 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 $A \neq \{\}$ $A \subseteq \llbracket SRD \rrbracket_H$
 shows $\prod A = \mathbf{R}_s((\bigwedge P \in A \cdot pre_R(P)) \vdash (\bigvee P \in A \cdot peri_R(P)) \diamond (\bigvee P \in A \cdot post_R(P)))$

proof –

have $\prod A = \mathbf{R}_s(pre_R(\prod A) \vdash peri_R(\prod A) \diamond post_R(\prod A))$
 by (metis SRD-as-reactive-tri-design assms srdes-theory.healthy-inf srdes-theory.healthy-inf-def)
 also have ... = $\mathbf{R}_s((\bigwedge P \in A \cdot pre_R(P)) \vdash (\bigvee P \in A \cdot peri_R(P)) \diamond (\bigvee P \in A \cdot post_R(P)))$
 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 $(\prod i \in A \cdot \mathbf{R}_s(P(i) \vdash Q(i) \diamond R(i))) = \mathbf{R}_s((\bigwedge i \in A \cdot P(i)) \vdash (\prod i \in A \cdot Q(i)) \diamond (\prod i \in A \cdot R(i)))$
 by (subst RHS-INF[OF assms, THEN sym], simp add: design-UINF-mem assms, rel-auto)

lemma *SRD-UINF-mem*:

assumes $A \neq \{\}$ $\bigwedge i. P\ i$ is SRD
 shows $(\prod i \in A \cdot P\ i) = \mathbf{R}_s((\bigwedge i \in A \cdot pre_R(P\ i)) \vdash (\bigvee i \in A \cdot peri_R(P\ i)) \diamond (\bigvee i \in A \cdot post_R(P\ i)))$
 (is ?lhs = ?rhs)

proof –

have ?lhs = $(\prod (P\ ' A))$
 by (rel-auto)
 also have ... = $\mathbf{R}_s((\bigwedge Pa \in P\ ' A \cdot pre_R\ Pa) \vdash (\prod Pa \in P\ ' A \cdot peri_R\ Pa) \diamond (\prod Pa \in P\ ' A \cdot post_R\ Pa))$

by (*subst rdes-def*, *simp-all add: assms image-subsetI*)
also have ... = ?*rhs*
by (*rel-auto*)
finally show ?*thesis* .
qed

lemma *RHS-tri-design-UINF-ind* [*rdes-def*]:
 $(\prod i \cdot \mathbf{R}_s(P_1(i) \vdash P_2(i) \diamond P_3(i))) = \mathbf{R}_s((\bigwedge i \cdot P_1 i) \vdash (\bigvee i \cdot P_2(i)) \diamond (\bigvee i \cdot P_3(i)))$
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 ... = $\mathbf{R}_s(P \vdash Q_1 \diamond Q_2 \triangleleft b \triangleright_R R \vdash S_1 \diamond S_2)$
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 $Miracle \;; P = Miracle$

proof –

have $Miracle \;; P = \mathbf{R}_s(true \vdash false) \;; \mathbf{R}_s(pre_R(P) \vdash cmt_R(P))$

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

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

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

also have $\dots = Miracle$

by (*simp add: Miracle-def*)

finally show *?thesis* .

qed

lemma *Chaos-left-zero*:

assumes P is *SRD*

shows $(Chaos \;; P) = Chaos$

proof –

have $Chaos \;; P = \mathbf{R}_s(false \vdash true) \;; \mathbf{R}_s(pre_R(P) \vdash cmt_R(P))$

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

also have $\dots = \mathbf{R}_s((\neg R1 \ true \ \wedge \neg (R1 \ true \ \wedge \neg \$wait') \;; R1 \ (\neg R2s \ (pre_R \ P))) \vdash$
 $R1 \ true \;; ((\exists \$st \cdot [II]_D) \triangleleft \$wait \triangleright R1 \ (R2s \ (cmt_R \ P))))$

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

also have $\dots = \mathbf{R}_s((false \ \wedge \neg (R1 \ true \ \wedge \neg \$wait') \;; R1 \ (\neg R2s \ (pre_R \ P))) \vdash$
 $R1 \ true \;; ((\exists \$st \cdot [II]_D) \triangleleft \$wait \triangleright R1 \ (R2s \ (cmt_R \ P))))$

by (*simp add: RHS-design-conj-neg-R1-pre*)

also have $\dots = \mathbf{R}_s(true)$

by (*simp add: design-false-pre*)

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

by (*simp add: design-def*)

also have $\dots = Chaos$

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

interpretation *srdes-left-unital*: *utp-theory-left-unital SRD II_R*

by (*unfold-locales, simp-all add: closure 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* $F \in \llbracket \text{SRD} \rrbracket_H \rightarrow \llbracket \text{SRD} \rrbracket_H$
shows $(\mu X \cdot \mathbf{R}_s(\text{pre}_R(F X) \vdash \text{peri}_R(F X) \diamond \text{post}_R(F X)))$ *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]*)
apply (*simp add: Healthy-apply-closed SRD-as-reactive-design SRD-reactive-design-alt assms(1) assms(2) mu-srd-SRD*)
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_{utp-order} SRD F*
by (*simp add: assms(1) mono-Monotone-utp-order*)
hence 3: $\mu_R F = F(\mu_R F)$
by (*simp add: srdes-theory.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.weak.LFP-lemma3 gfp-upperbound assms*)
thus *?thesis*
using *assms 1 3 srdes-theory.weak.LFP-lowerbound eq-iff mu-srd-iter*
by (*metis (mono-tags, lifting)*)
qed

lemma *Monotonic-SRD-comp [closure]*: *Monotonic ((;;) P \circ 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*]: $RD3(P) = P \;; \; II_R$

lemma *RD3-idem*: $RD3(RD3(P)) = RD3(P)$

proof –

have $a: II_R \;; \; II_R = II_R$

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*: $RD3(\sqcap A) = (\sqcap P \in A. RD3(P))$

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*: $P \text{ is } RD3 \implies P \text{ is } RD2$

by (*metis Healthy-def RD3-right-subsumes-RD2*)

lemma *RD3-intro-pre*:

assumes $P \text{ is } SRD \ (\neg_r \text{ pre}_R(P)) \;; \; true_r = (\neg_r \text{ pre}_R(P)) \ \$st' \ \# \ \text{peri}_R(P)$

shows $P \text{ is } RD3$

proof –

have $RD3(P) = \mathbf{R}_s \ ((\neg_r \text{ pre}_R P) \text{ wp}_r \text{ false} \vdash (\exists \ \$st' \cdot \text{peri}_R P) \diamond \text{post}_R P)$

by (*simp add: RD3-def SRD-right-unit-tri-lemma assms*)

also have $\dots = \mathbf{R}_s \ ((\neg_r \text{ pre}_R P) \text{ wp}_r \text{ false} \vdash \text{peri}_R P \diamond \text{post}_R P)$

by (*simp add: assms(3) ex-unrest*)

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

by (*simp add: wait'-cond-peri-post-cmt*)

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

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 $\dots = \mathbf{R}_s((\neg R1 (\neg R2s P) ;; R1 true) \vdash (\exists \$st' \cdot Q) \diamond R1 (R2s R))$
 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 $\dots = \mathbf{R}_s((\neg (\neg P) ;; R1 true) \vdash ((\exists \$st' \cdot Q) \diamond R))$
 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 $\dots = \mathbf{R}_s((\neg_r (\neg_r P) ;; true_r) \vdash ((\exists \$st' \cdot Q) \diamond R))$
 by (rel-auto)
 finally show ?thesis .
 qed

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

assumes
 $\$ok' \# P \$ok' \# Q \$ok' \# R \$st' \# Q \$wait' \# R$
 $P \text{ is } R1 (\neg_r P) ;; true_r = (\neg_r P)$
 shows $\mathbf{R}_s(P \vdash Q \diamond R) \text{ is } RD3$
 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*:

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

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

assumes
 $out\alpha \# b \text{ out}\alpha \# e \$ok' \# Q \$st' \# Q \$ok' \# R \$wait' \# R$
 shows $\mathbf{R}_s((b \wedge \neg_r \$tr \hat{^}_u e \leq_u \$tr') \vdash Q \diamond R) \text{ is } RD3$
 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* is *NSRD* \Longrightarrow *P* 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*]:
 $\llbracket P \text{ is } NSRD; Q(\mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \diamond \text{post}_R(P))) \rrbracket \Longrightarrow Q(P)$
by (*simp add: RD-elim closure*)

lemma *NSRD-idem*: *NSRD*(*NSRD*(*P*)) = *NSRD*(*P*)
by (*metis (no-types, hide-lams) Healthy-def NSRD-def RD1-RD2-commute RD1-RD3-commute RD1-RHS-commute RD1-idem RD2-RHS-commute RD2-idem RD3-def RD3-idem RD3-left-subsumes-RD2 RHS-idem SRD-def comp-apply fun.map-comp srdes-left-unital.Healthy-Sequence srdes-left-unital.Healthy-Unit*)

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

lemma *NSRD-Continuous* [*closure*]: *Continuous NSRD*
by (*simp add: Continuous-comp NSRD-def RD1-Continuous RD3-Continuous RHS-Continuous*)

lemma *NSRD-form*:
 $NSRD(P) = \mathbf{R}_s((\neg_r (\neg_r \text{pre}_R(P)) ;; R1 \text{ true}) \vdash ((\exists \$st' \cdot \text{peri}_R(P)) \diamond \text{post}_R(P)))$
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 ... = *RD3*($\mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \diamond \text{post}_R(P))$)
by (*simp add: SRD-as-reactive-tri-design*)
also have ... = $\mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \diamond \text{post}_R(P)) ;; II_R$
by (*simp add: RD3-def*)
also have ... = $\mathbf{R}_s((\neg_r (\neg_r \text{pre}_R(P)) ;; R1 \text{ true}) \vdash ((\exists \$st' \cdot \text{peri}_R(P)) \diamond \text{post}_R(P)))$
by (*simp add: RHS-tri-design-right-unit-lemma unrest*)
finally show ?thesis .
qed

lemma *NSRD-healthy-form*:
assumes *P* is *NSRD*
shows $\mathbf{R}_s((\neg_r (\neg_r \text{pre}_R(P)) ;; R1 \text{ true}) \vdash ((\exists \$st' \cdot \text{peri}_R(P)) \diamond \text{post}_R(P))) = P$
by (*metis Healthy-def NSRD-form assms*)

lemma *NSRD-Sup-closure* [*closure*]:
assumes $A \subseteq \llbracket NSRD \rrbracket_H$ $A \neq \{\}$
shows $\bigcap A$ is *NSRD*
proof –
have *NSRD* ($\bigcap A$) = (\bigcap (*NSRD* '*A*'))
by (*simp add: ContinuousD NSRD-Continuous assms(2)*)
also have ... = ($\bigcap A$)
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 *NSRD-SUP-closure* [closure]:

$\llbracket \bigwedge i. i \in A \implies P(i) \text{ is NSRD}; A \neq \{\} \rrbracket \implies (\bigwedge i \in A. P(i)) \text{ is NSRD}$

by (rule NSRD-Sup-closure, auto)

lemma *NSRD-neg-pre-unit*:

assumes P is NSRD

shows $(\neg_r \text{pre}_R(P)) ;; \text{true}_r = (\neg_r \text{pre}_R(P))$

proof –

have $(\neg_r \text{pre}_R(P)) = (\neg_r \text{pre}_R(\mathbf{R}_s((\neg_r (\neg_r \text{pre}_R(P)) ;; R1 \text{ true}) \vdash ((\exists \$st' \cdot \text{peri}_R(P)) \diamond \text{post}_R(P))))$
by (simp add: NSRD-healthy-form assms)

also have $\dots = R1 (R2c ((\neg_r \text{pre}_R P) ;; R1 \text{ true}))$

by (simp add: rea-pre-RHS-design R1-negate-R1 R1-idem R1-rea-not' R2c-rea-not usubst rpred unrest closure)

also have $\dots = (\neg_r \text{pre}_R P) ;; R1 \text{ true}$

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 $\dots = R1 (R2c (\neg_r (\neg_r \text{pre}_R P) ;; R1 \text{ true} \Rightarrow_r (\exists \$st' \cdot \text{peri}_R P)))$

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 $\dots = (R1 (R2c (\neg_r (\neg_r \text{pre}_R P) ;; R1 \text{ true})))$

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 $\$st' \# pre_R(P)$

proof –

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

by (*simp add: NSRD-healthy-form assms*)

also have $\dots = R1\ (R2c\ (\neg_r (\neg_r pre_R\ P) ;; R1\ true))$

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-peri-under-pre* [*rpred*]:

P is NSRD $\implies (pre_R\ P \Rightarrow_r\ peri_R\ P) = peri_R\ P$

by (*simp add: SRD-peri-under-pre unrest closure*)

lemma *NSRD-post-under-pre* [*rpred*]:

P is NSRD $\implies (pre_R\ P \Rightarrow_r\ post_R\ P) = post_R\ P$

by (*simp add: SRD-post-under-pre unrest closure*)

lemma *NSRD-peri-seq-under-pre*:

assumes *P* is NSRD *Q* is NSRD

shows $(pre_R\ P \Rightarrow_r\ peri_R\ P \vee post_R\ P ;; peri_R\ Q) = (peri_R\ P \vee post_R\ P ;; peri_R\ Q)$

by (*metis NSRD-peri-under-pre assms(1) rea-impl-def utp-pred-laws.disj-assoc*)

lemma *NSRD-postR-seq-periR-impl*:

assumes *P* is NSRD *Q* is NSRD

shows $(post_R\ P\ wp_r\ pre_R\ Q \Rightarrow_r\ (post_R\ P ;; peri_R\ Q)) = (post_R\ P ;; peri_R\ Q)$

by (*metis NSRD-is-SRD NSRD-peri-under-pre assms postR-RR wpR-impl-post-spec*)

lemma *NSRD-postR-seq-postR-impl*:

assumes *P* is NSRD *Q* is NSRD

shows $(post_R\ P\ wp_r\ pre_R\ Q \Rightarrow_r\ (post_R\ P ;; post_R\ Q)) = (post_R\ P ;; post_R\ Q)$

by (*metis NSRD-is-SRD NSRD-post-under-pre assms postR-RR wpR-impl-post-spec*)

lemma *NSRD-peri-under-assms*:

assumes *P* is NSRD *Q* is NSRD

shows $(pre_R\ P \wedge post_R\ P\ wp_r\ pre_R\ Q \Rightarrow_r\ peri_R\ P \vee post_R\ P ;; peri_R\ Q) = (peri_R\ P \vee post_R\ P ;; peri_R\ Q)$

by (*metis (no-types, lifting) NSRD-peri-seq-under-pre assms NSRD-postR-seq-periR-impl rea-impl-conj rea-impl-disj*)

lemma *NSRD-peri-under-assms'*:

assumes *P* is NSRD *Q* is NSRD

shows $(post_R\ P\ wp_r\ pre_R\ Q \Rightarrow_r\ peri_R\ P \vee post_R\ P ;; peri_R\ Q) = (peri_R\ P \vee post_R\ P ;; peri_R\ Q)$

by (*simp add: NSRD-postR-seq-periR-impl assms rea-impl-disj*)

lemma *NSRD-post-under-assms*:

assumes *P* is NSRD *Q* is NSRD

shows $(pre_R\ P \wedge post_R\ P\ wp_r\ pre_R\ Q \Rightarrow_r\ post_R\ P ;; post_R\ Q) = (pre_R\ P \Rightarrow_r\ (post_R\ P ;; post_R\ Q))$

by (*metis NSRD-postR-seq-postR-impl assms(1) assms(2) rea-impl-conj*)

lemma *NSRD-alt-def*: $NSRD(P) = RD3(SRD(P))$

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*: $P \text{ is NSRD} \Rightarrow P ;; II_R = P$

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 $\text{post}_R(P ;; Q) = ((\text{pre}_R P \wedge \text{post}_R P \text{ wp}_r \text{pre}_R Q) \Rightarrow_r (\text{post}_R P ;; \text{post}_R Q))$
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 ;; Miracle = R_s (pre_R P ⊢ peri_R P ◊ false)*
by (*simp add: NSRD-composition-wp closure assms rdes wp rpred*)

lemma *Miracle-right-anhil-iff*:
assumes *P is NSRD*
shows *P ;; Miracle = Miracle ⟷ pre_R P = true_r ∧ (∃ \$st' · peri_R P) = false*
by (*simp add: SRD-right-Miracle-tri-lemma assms closure, simp add: rdes-def sdes-tri-eq-iff closure assms wp rpred, fastforce*)

The set of non-terminating behaviours is a subset

lemma *NSRD-right-Miracle-refines*:
assumes *P is NSRD*
shows *P ⊆ P ;; Miracle*
proof –
have *R_s (pre_R P ⊢ peri_R P ◊ post_R P) ⊆ R_s (pre_R P ⊢ peri_R P ◊ false)*
by (*rule sdes-tri-refine-intro, rel-auto+*)
thus *?thesis*
by (*simp add: NSRD-elim NSRD-right-Miracle-tri-lemma assms*)
qed

Chaos is a right zero of *P* precisely when the conjunction of the precondition of *P* with the weakest precondition under which *post_R P* yields *U(false)* reduces to *U(false)*. This is effectively a feasibility check for reactive designs.

lemma *Chaos-right-anhil-iff*:
assumes *P is NSRD*
shows *P ;; Chaos = Chaos ⟷ (pre_R P ∧ post_R P wp_r false) = false*
by (*simp add: SRD-right-Chaos-tri-lemma assms closure, simp add: rdes-def sdes-tri-eq-iff closure assms wp*)

lemma *upower-Suc-NSRD-closed* [*closure*]:
P is NSRD ⟹ P ^ Suc n 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 is NSRD $\implies P \;; P \wedge n$ is NSRD

by (*metis upower-Suc-NSRD-closed upred-semiring.power-Suc*)

lemma *uplus-NSRD-closed* [closure]: *P is NSRD $\implies P^+$ is NSRD*

by (*simp add: uplus-power-def closure*)

lemma *preR-power*:

assumes *P is NSRD*

shows $\text{pre}_R(P \;; P \wedge n) = (\bigsqcup_{i \in \{0..n\}} (\text{post}_R(P) \wedge i) \text{wp}_r (\text{pre}_R(P)))$

proof (*induct n*)

case 0

then show ?case

by (*simp add: wp closure*)

next

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

have $\text{pre}_R(P \wedge (\text{Suc } n + 1)) = \text{pre}_R(P \;; P \wedge (n+1))$

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

also have ... = $(\text{pre}_R P \wedge \text{post}_R P \text{wp}_r \text{pre}_R(P \wedge (\text{Suc } n)))$

using *NSRD-iff assms preR-NSRD-seq upower-Suc-NSRD-closed* **by** *fastforce*

also have ... = $(\text{pre}_R P \wedge \text{post}_R P \text{wp}_r (\bigsqcup_{i \in \{0..n\}} \text{post}_R P \wedge i \text{wp}_r \text{pre}_R P))$

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

also have ... = $(\text{pre}_R P \wedge (\bigsqcup_{i \in \{0..n\}} \text{post}_R P \text{wp}_r (\text{post}_R P \wedge i \text{wp}_r \text{pre}_R P)))$

by (*simp add: wp*)

also have ... = $(\text{pre}_R P \wedge (\bigsqcup_{i \in \{0..n\}} (\text{post}_R P \wedge (i+1) \text{wp}_r \text{pre}_R P)))$

proof –

have $\bigwedge i. R1 (\text{post}_R P \wedge i \;; (\neg_r \text{pre}_R P)) = (\text{post}_R P \wedge i \;; (\neg_r \text{pre}_R P))$

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 ... = $(\text{post}_R P \wedge 0 \text{wp}_r \text{pre}_R P \wedge (\bigsqcup_{i \in \{0..n\}} (\text{post}_R P \wedge (i+1) \text{wp}_r \text{pre}_R P)))$

by (*simp add: wp assms closure*)

also have ... = $(\text{post}_R P \wedge 0 \text{wp}_r \text{pre}_R P \wedge (\bigsqcup_{i \in \{1..\text{Suc } n\}} (\text{post}_R P \wedge i \text{wp}_r \text{pre}_R P)))$

proof –

have $(\bigsqcup_{i \in \{0..n\}} (\text{post}_R P \wedge (i+1) \text{wp}_r \text{pre}_R P)) = (\bigsqcup_{i \in \{1..\text{Suc } n\}} (\text{post}_R P \wedge i \text{wp}_r \text{pre}_R P))$

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 ... = $(\bigsqcup_{i \in \text{insert } 0 \{1..\text{Suc } n\}} (\text{post}_R P \wedge i \text{wp}_r \text{pre}_R P))$

by (*simp add: conj-upred-def*)

also have ... = $(\bigsqcup_{i \in \{0..\text{Suc } n\}} \text{post}_R P \wedge i \text{wp}_r \text{pre}_R P)$

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 $\text{pre}_R(P \;; P \wedge n) = (\bigsqcup_{i \in \{0..n\}} (\text{post}_R(P) \wedge i) \text{wp}_r (\text{pre}_R(P)))$

by (*simp add: preR-power assms USUP-as-Inf[THEN sym]*)

lemma *preR-power-Suc* [rdes]:

assumes *P is NSRD*

```

shows  $pre_R(P^\wedge(Suc\ n)) = (\bigsqcup\ i \in \{0..n\} \cdot (post_R(P) \wedge i) \text{ } wp_r \text{ } (pre_R(P)))$ 
by (simp add: upred-semiring.power-Suc rdes assms)

declare upred-semiring.power-Suc [simp]

lemma periR-power:
  assumes P is NSRD
  shows  $peri_R(P ;; P^\wedge n) = (pre_R(P^\wedge(Suc\ n)) \Rightarrow_r (\bigsqcup\ i \in \{0..n\}. post_R(P) \wedge i) ;; peri_R(P))$ 
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  $peri_R(P^\wedge(Suc\ n + 1)) = peri_R(P ;; P^\wedge(n+1))$ 
    by (simp)
  also have  $\dots = (pre_R(P^\wedge(Suc\ n + 1)) \Rightarrow_r (peri_R\ P \vee post_R\ P ;; peri_R(P ;; P^\wedge n)))$ 
    by (simp add: closure assms rdes)
  also have  $\dots = (pre_R(P^\wedge(Suc\ n + 1)) \Rightarrow_r (peri_R\ P \vee post_R\ P ;; (pre_R(P^\wedge(Suc\ n)) \Rightarrow_r (\bigsqcup\ i \in \{0..n\}. post_R\ P^\wedge i) ;; peri_R\ P)))$ 
    by (simp only: hyp)
  also
  have  $\dots = (pre_R\ P \Rightarrow_r peri_R\ P \vee (post_R\ P \text{ } wp_r \text{ } pre_R(P ;; P^\wedge n) \Rightarrow_r post_R\ P ;; (pre_R(P ;; P^\wedge n) \Rightarrow_r (\bigsqcup\ i \in \{0..n\}. post_R\ P^\wedge i) ;; peri_R\ P)))$ 
    by (simp add: rdes closure assms, rel-blast)
  also
  have  $\dots = (pre_R\ P \Rightarrow_r peri_R\ P \vee (post_R\ P \text{ } wp_r \text{ } pre_R(P ;; P^\wedge n) \Rightarrow_r post_R\ P ;; ((\bigsqcup\ i \in \{0..n\}. post_R\ P^\wedge i) ;; peri_R\ P)))$ 
    proof –
    have  $(\bigsqcup\ i \in \{0..n\}. post_R\ P^\wedge i)$  is R1
      by (simp add: NSRD-is-SRD R1-Continuous R1-power Sup-Continuous-closed assms postR-SRD-R1)
    hence  $1:((\bigsqcup\ i \in \{0..n\}. post_R\ P^\wedge i) ;; peri_R\ P)$  is R1
      by (simp add: closure assms)
    hence  $(pre_R(P ;; P^\wedge n) \Rightarrow_r (\bigsqcup\ i \in \{0..n\}. post_R\ P^\wedge i) ;; peri_R\ P)$  is R1
      by (simp add: closure)
    hence  $(post_R\ P \text{ } wp_r \text{ } pre_R(P ;; P^\wedge n) \Rightarrow_r post_R\ P ;; (pre_R(P ;; P^\wedge n) \Rightarrow_r (\bigsqcup\ i \in \{0..n\}. post_R\ P^\wedge i) ;; peri_R\ P))$ 
       $= (post_R\ P \text{ } wp_r \text{ } pre_R(P ;; P^\wedge n) \Rightarrow_r R1(post_R\ P) ;; R1(pre_R(P ;; P^\wedge n) \Rightarrow_r (\bigsqcup\ i \in \{0..n\}. post_R\ P^\wedge i) ;; peri_R\ P))$ 
      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  $\dots = (pre_R\ P \wedge post_R\ P \text{ } wp_r \text{ } pre_R(P ;; P^\wedge n) \Rightarrow_r peri_R\ P \vee post_R\ P ;; ((\bigsqcup\ i \in \{0..n\}. post_R\ P^\wedge i) ;; peri_R\ P))$ 
    by (pred-auto)
  also
  have  $\dots = (pre_R\ P \wedge post_R\ P \text{ } wp_r \text{ } pre_R(P ;; P^\wedge n) \Rightarrow_r peri_R\ P \vee ((\bigsqcup\ i \in \{0..n\}. post_R\ P^\wedge(Suc\ i)) ;; peri_R\ P))$ 
    by (simp add: seq-Sup-distl seqr-assoc[THEN sym])
  also
  have  $\dots = (pre_R\ P \wedge post_R\ P \text{ } wp_r \text{ } pre_R(P ;; P^\wedge n) \Rightarrow_r peri_R\ P \vee ((\bigsqcup\ i \in \{1..Suc\ n\}. post_R\ P^\wedge i) ;; peri_R\ P))$ 
    proof –

```

```

have ( $\prod i \in \{0..n\}. post_R P \wedge Suc i = (\prod i \in \{1..Suc n\}. post_R P \wedge i)$ )
  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 ... = ( $pre_R P \wedge post_R P wp_r pre_R (P ;; P \wedge n) \Rightarrow_r ((\prod i \in \{0..Suc n\}. post_R P \wedge i) ;; peri_R P)$ )
  by (simp add: SUP-atLeastAtMost-first uinf-or segr-or-distl segr-or-distr)
also
have ... = ( $pre_R(P \wedge (Suc (Suc n))) \Rightarrow_r ((\prod i \in \{0..Suc n\}. post_R P \wedge i) ;; peri_R P)$ )
  by (simp add: rdes closure assms)
finally show ?case by (simp)
qed

lemma periR-power' [rdes]:
  assumes  $P$  is NSRD
  shows  $peri_R(P ;; P \wedge n) = (pre_R(P \wedge (Suc n)) \Rightarrow_r (\prod i \in \{0..n\} \cdot post_R(P) \wedge i) ;; peri_R(P))$ 
  by (simp add: periR-power assms UINF-as-Sup[THEN sym])

lemma periR-power-Suc [rdes]:
  assumes  $P$  is NSRD
  shows  $peri_R(P \wedge (Suc n)) = (pre_R(P \wedge (Suc n)) \Rightarrow_r (\prod i \in \{0..n\} \cdot post_R(P) \wedge i) ;; peri_R(P))$ 
  by (simp add: rdes assms)

lemma postR-power [rdes]:
  assumes  $P$  is NSRD
  shows  $post_R(P ;; P \wedge n) = (pre_R(P \wedge (Suc n)) \Rightarrow_r post_R(P) \wedge Suc n)$ 
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 (Suc n) note hyp = this
  have  $post_R (P \wedge (Suc n + 1)) = post_R (P ;; P \wedge (n+1))$ 
    by (simp)
  also have ... = ( $pre_R(P \wedge (Suc n + 1)) \Rightarrow_r (post_R P ;; post_R (P ;; P \wedge n))$ )
    by (simp add: closure assms rdes)
  also have ... = ( $pre_R(P \wedge (Suc n + 1)) \Rightarrow_r (post_R P ;; (pre_R (P \wedge Suc n) \Rightarrow_r post_R P \wedge Suc n))$ )
    by (simp only: hyp)
  also
  have ... = ( $pre_R P \Rightarrow_r (post_R P wp_r pre_R (P \wedge Suc n) \Rightarrow_r post_R P ;; (pre_R (P \wedge Suc n) \Rightarrow_r post_R P \wedge Suc n))$ )
    by (simp add: rdes closure assms, pred-auto)
  also
  have ... = ( $pre_R P \Rightarrow_r (post_R P wp_r pre_R (P \wedge Suc n) \Rightarrow_r post_R P ;; post_R P \wedge Suc n)$ )
    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 ... = ( $pre_R P \wedge post_R P wp_r pre_R (P \wedge Suc n) \Rightarrow_r post_R P \wedge Suc (Suc n)$ )
    by (pred-auto)
  also have ... = ( $pre_R(P \wedge (Suc (Suc n))) \Rightarrow_r post_R P \wedge Suc (Suc n)$ )
    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 $(\mathbf{R}_s(P \vdash Q \diamond R))^\wedge(\text{Suc } n)$

$= \mathbf{R}_s((\bigsqcup_{i \in \{0..n\}} \cdot (R \wedge i) \text{ wp}_r P) \vdash ((\bigsqcap_{i \in \{0..n\}} \cdot R \wedge i) ;; Q) \diamond (R \wedge \text{Suc } n))$

proof (*induct n*)

case 0

then show ?*case*

by (*simp add: wp assms closure*)

next

case (*Suc n*)

have 1: $(P \wedge (\bigsqcup_{i \in \{0..n\}} \cdot R \text{ wp}_r (R \wedge i \text{ wp}_r P))) = (\bigsqcup_{i \in \{0..Suc\ } n\}} \cdot R \wedge i \text{ wp}_r P)$
(is ?*lhs* = ?*rhs*)

proof –

have ?*lhs* = $(P \wedge (\bigsqcup_{i \in \{0..n\}} \cdot (R \wedge \text{Suc } i \text{ wp}_r P)))$

by (*simp add: wp closure assms*)

also have ... = $(P \wedge (\bigsqcup_{i \in \{0..n\}} \cdot (R \wedge \text{Suc } i \text{ wp}_r P)))$

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

also have ... = $(P \wedge (\bigsqcup_{i \in \{1..Suc\ } n\}} \cdot (R \wedge i \text{ wp}_r P)))$

by (*metis (no-types, lifting) INF-cong One-nat-def image-Suc-atLeastAtMost image-image*)

also have ... = $(\bigsqcup_{i \in \text{insert } 0 \{1..Suc\ } n\}} \cdot (R \wedge i \text{ wp}_r P))$

by (*simp add: wp assms closure conj-upred-def*)

also have ... = $(\bigsqcup_{i \in \{0..Suc\ } n\}} \cdot (R \wedge i \text{ wp}_r P))$

by (*simp add: atLeastAtMost-insertL*)

finally show ?*thesis*

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

qed

have 2: $(Q \vee R ;; (\bigsqcap_{i \in \{0..n\}} \cdot R \wedge i) ;; Q) = (\bigsqcap_{i \in \{0..Suc\ } n\}} \cdot R \wedge i) ;; Q$
(is ?*lhs* = ?*rhs*)

proof –

have ?*lhs* = $(Q \vee (\bigsqcap_{i \in \{0..n\}} \cdot R \wedge \text{Suc } i) ;; Q)$

by (*simp add: seqr-assoc[THEN sym] seq-UINF-distl*)

also have ... = $(Q \vee (\bigsqcap_{i \in \{0..n\}} \cdot R \wedge \text{Suc } i) ;; Q)$

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

also have ... = $(Q \vee (\bigsqcap_{i \in \{1..Suc\ } n\}} \cdot R \wedge i) ;; Q)$

by (*metis One-nat-def image-Suc-atLeastAtMost image-image*)

also have ... = $((\bigsqcap_{i \in \text{insert } 0 \{1..Suc\ } n\}} \cdot R \wedge i) ;; Q)$

by (*simp add: disj-upred-def[THEN sym] seqr-or-distl*)

also have ... = $((\bigsqcap_{i \in \{0..Suc\ } n\}} \cdot R \wedge i) ;; Q)$

by (*simp add: atLeastAtMost-insertL*)

finally show ?*thesis*

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

qed

have 3: $(\bigsqcap_{i \in \{0..n\}} \cdot R \wedge i) ;; Q$ is *RR*

proof –

have $(\bigsqcap_{i \in \{0..n\}} \cdot R \wedge i) ;; Q = (\bigsqcap_{i \in \{0..n\}} \cdot R \wedge i) ;; Q$

```

    by (simp add: UINF-as-Sup-collect)
  also have ... = ( $\bigcap i \in \text{insert } 0 \{1..n\}. R \hat{~} i$ ) ;; Q
    by (simp add: atLeastAtMost-insertL)
  also have ... = ( $Q \vee (\bigcap i \in \{1..n\}. R \hat{~} i)$ ) ;; Q
  by (metis (no-types, lifting) SUP-insert disj-upred-def segr-left-unit segr-or-distl upred-semiring.power-0)
  also have ... = ( $Q \vee (\bigcap i \in \{0..<n\}. R \hat{~} \text{Suc } i)$ ) ;; Q
  by (metis One-nat-def atLeastLessThanSuc-atLeastAtMost image-Suc-atLeastLessThan image-image)
  also have ... = ( $Q \vee (\bigcap i \in \{0..<n\} \cdot R \hat{~} \text{Suc } i)$ ) ;; 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 $(\mathbf{R}_s(P \vdash Q \diamond R))^+ = \mathbf{R}_s(R^{\star r} \text{wp}_r P \vdash (R^{\star r} ;; Q) \diamond R^+)$

proof –

have $1:(\bigcap i \cdot R \hat{~} i) ;; Q = R^{\star r} ;; Q$
 by (metis (no-types) RA1 assms(2) rea-skip-unit(2) rrel-theory.Star-def ular-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

lemma NSRD-false: NSRD false = Miracle

by (metis Healthy-if NSRD-Miracle NSRD-alt-def NSRD-is-RD3 srdes-theory.healthy-top)

lemma NSRD-true: NSRD true = Chaos

by (metis Healthy-if NSRD-Chaos NSRD-alt-def NSRD-is-RD3 srdes-theory.healthy-bottom)

interpretation nsrdes-theory: utp-theory-kleene NSRD II_R

rewrites $P \in \text{carrier nsrdes-theory.thy-order} \longleftrightarrow P$ is NSRD

and $\text{carrier nsrdes-theory.thy-order} \rightarrow \text{carrier nsrdes-theory.thy-order} \equiv \llbracket \text{NSRD} \rrbracket_H \rightarrow \llbracket \text{NSRD} \rrbracket_H$

and $\text{le nsrdes-theory.thy-order} = (\sqsubseteq)$

and $\text{eq nsrdes-theory.thy-order} = (=)$

and $\text{nsrdes-top: nsrdes-theory.utp-top} = \text{Miracle}$

and $\text{nsrdes-bottom: nsrdes-theory.utp-bottom} = \text{Chaos}$

proof –

have utp-theory-continuous NSRD

by (unfold-locales, simp-all add: NSRD-idem NSRD-Continuous)

then interpret utp-theory-continuous NSRD

by simp

show $t: \text{utp-top} = \text{Miracle}$ and $b: \text{utp-bottom} = \text{Chaos}$

by (simp-all add: healthy-top healthy-bottom NSRD-false NSRD-true)

show utp-theory-kleene NSRD II_R

by (unfold-locales, simp-all add: closure srdes-left-unital.Unit-Left NSRD-right-unit Miracle-left-zero

t)

qed (simp-all)

abbreviation $TestR$ ($test_R$) **where**
 $test_R P \equiv nsrdes-theory.utp-test P$

definition $StarR :: ('s, 't::trace, 'α) hrel-rsp \Rightarrow ('s, 't, 'α) hrel-rsp (-^{*R} [999] 999)$ **where**
 $StarR \equiv nsrdes-theory.utp-star$

We also show how to calculate the Kleene closure of a reactive design.

thm $rdes-def$

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: $StarR-def$ $rrel-theory.Star-alt-def$ $nsrdes-theory.Star-alt-def$ $closure$ $assms$)

($simp$ add: $rrel-theory.Star-alt-def$ $assms$ $closure$ $rdes-def$ $unrest$ $rpred$ $disj-upred-def$)

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 $U(\&tt)$.

definition $mk-RD :: 's upred \Rightarrow ('t::trace \Rightarrow 's upred) \Rightarrow ('t \Rightarrow 's hrel) \Rightarrow ('s, 't, 'a) hrel-rsp$ **where**
 $mk-RD P Q R = \mathbf{R}_s(\lceil P \rceil_{S<} \vdash \lceil Q(x) \rceil_{S<} \llbracket x \rightarrow \&tt \rrbracket \diamond \lceil R(x) \rceil_S \llbracket x \rightarrow \&tt \rrbracket)$

definition $trace-pred :: ('t::trace \Rightarrow 's upred) \Rightarrow ('s, 't, 'α) hrel-rsp$ **where**
 $[upred-defs]: trace-pred P = \lceil (P x) \rceil_{S<} \llbracket x \rightarrow \&tt \rrbracket$

syntax

$-trace-var :: logic$

$-mk-RD :: logic \Rightarrow logic \Rightarrow logic \Rightarrow logic ([\cdot / \vdash \cdot / \mid \cdot]_R)$

$-trace-pred :: logic \Rightarrow logic ([\cdot]_t)$

parse-translation \langle

let

$fun trace-var-tr [] = Syntax.free trace$

$\mid trace-var-tr - = raise Match;$

in

$[(\@ \{syntax-const \text{ } -trace-var\}, K trace-var-tr)]$

end

\rangle

translations

$[P \vdash Q \mid R]_R \Rightarrow CONST mk-RD P (\lambda \text{ } -trace-var. Q) (\lambda \text{ } -trace-var. R)$

$[P \vdash Q \mid R]_R \Leftarrow CONST mk-RD P (\lambda x. Q) (\lambda y. R)$

$[P]_t \Rightarrow CONST trace-pred (\lambda \text{ } -trace-var. P)$

$[P]_t \Leftarrow CONST trace-pred (\lambda t. P)$

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]:
 $[P \text{ trace}]_t$ is *RR*
 by (rel-auto)

lemma *unrest-trace-pred-st'* [unrest]:
 $\$st' \# [P \text{ trace}]_t$
 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 sdes-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

utp-pred-laws.distrib(3,4)

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* = (*ueexpr-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 *frame* *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'*)

Tactics 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*))

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*)

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  $\triangleleft$   $\$<:wait \triangleright$  M)
```

```
lemma R3hm-idem: R3hm(R3hm(P)) = R3hm(P)
  by (rel-auto)
```

```
abbreviation copytype :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a (CPTYPE'(-, -)) where CPTYPE(A, B)  $\equiv$  B
```

```
lemma R3h-par-by-merge [closure]:
```

```
  assumes P is R3h Q is R3h M is R3hm
```

```
  shows (P  $\parallel_M$  Q) is R3h
```

```
proof -
```

```
  have (P  $\parallel_M$  Q) = (((P  $\parallel_M$  Q)  $\llbracket true/\$ok \rrbracket \triangleleft \$ok \triangleright$  (P  $\parallel_M$  Q)  $\llbracket false/\$ok \rrbracket \llbracket true/\$wait \rrbracket \triangleleft \$wait \triangleright$  (P  $\parallel_M$  Q)))
```

```
    by (simp add: cond-var-subst-left cond-var-subst-right)
```

```
  also have ... = (((P  $\parallel_M$  Q)  $\llbracket true, true/\$ok, \$wait \rrbracket \triangleleft \$ok \triangleright$  (P  $\parallel_M$  Q)  $\llbracket false, true/\$ok, \$wait \rrbracket \triangleleft \$wait \triangleright$  (P  $\parallel_M$  Q)))
```

```
    by (rel-auto)
```

```
  also have ... = ((( $\exists \$st \cdot II$ )  $\llbracket true, true/\$ok, \$wait \rrbracket \triangleleft \$ok \triangleright$  (P  $\parallel_M$  Q)  $\llbracket false, true/\$ok, \$wait \rrbracket \triangleleft \$wait \triangleright$  (P  $\parallel_M$  Q)))
```

```
  proof -
```

```
    have (P  $\parallel_M$  Q)  $\llbracket true, true/\$ok, \$wait \rrbracket = ((\lceil P \rceil_0 \wedge \lceil Q \rceil_1 \wedge \$<:\mathbf{v}' =_u \$\mathbf{v}) ;; R3hm(M)) \llbracket true, true/\$ok, \$wait \rrbracket$ 
```

```
      by (simp add: par-by-merge-def U0-as-alpha U1-as-alpha assms Healthy-if)
```

```
    also have ... = ((( $\lceil P \rceil_0 \wedge \lceil Q \rceil_1 \wedge \$<' =_u \$\mathbf{v}$ ) ;; ( $\exists \$<:st \cdot CPTYPE(M, \$\mathbf{v}' =_u \$<))$ )  $\llbracket true, true/\$ok, \$wait \rrbracket$ )
```

```
      by (rel-blast)
```

```
    also have ... = ((( $\lceil R3h(P) \rceil_0 \wedge \lceil R3h(Q) \rceil_1 \wedge \$<' =_u \$\mathbf{v}$ ) ;; ( $\exists \$<:st \cdot CPTYPE(M, \$\mathbf{v}' =_u \$<)$ ))  $\llbracket true, true/\$ok, \$wait \rrbracket$ )
```

```
      by (simp add: assms Healthy-if)
```

```
    also have ... = ( $\exists \$st \cdot II$ )  $\llbracket true, true/\$ok, \$wait \rrbracket$ 
```

```
      by (rel-auto)
```

finally show *?thesis* **by** (*simp add: closure assms unrest*)
qed
also have ... = (((\exists $\$st \cdot II$) $\llbracket true, true/\$ok, \$wait \rrbracket \triangleleft \$ok \triangleright (R1(true))\llbracket false, true/\$ok, \$wait \rrbracket \triangleleft \$wait$
 $\triangleright (P \parallel_M Q)$)
proof –
have ($P \parallel_M Q$) $\llbracket false, true/\$ok, \$wait \rrbracket = (([P]_0 \wedge [Q]_1 \wedge \$<:\mathbf{v}' =_u \$\mathbf{v}) ;; R3hm(M))\llbracket false, true/\$ok, \$wait \rrbracket$
by (*simp add: par-by-merge-def U0-as-alpha U1-as-alpha assms Healthy-if*)
also have ... = ((($[P]_0 \wedge [Q]_1 \wedge \$<:\mathbf{v}' =_u \$\mathbf{v}$) ;; (*CPTYPE*($M, \$<:tr \leq_u \tr'))) $\llbracket false, true/\$ok, \$wait \rrbracket$
by (*rel-blast*)
also have ... = ((($[R3h(P)]_0 \wedge [R3h(Q)]_1 \wedge \$<:\mathbf{v}' =_u \$\mathbf{v}$) ;; (*CPTYPE*($M, \$<:tr \leq_u \tr'))) $\llbracket false, true/\$ok, \$wait \rrbracket$
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: asms 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 $(P \parallel_{NR} M \ Q \ ; \ II_R)$ 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 segr-middle[of ok ;_L mrg-left], simp-all)
 also have ... = (\exists ok₀ · \exists ok₁ · (($P \parallel_s Q$) $\llbracket \llcorner ok_0 \gg / \$0:ok' \rrbracket \llcorner ok_1 \gg / \$1:ok' \rrbracket$) ;; ($M \llbracket \llcorner ok_0 \gg / \$0:ok \rrbracket \llcorner ok_1 \gg / \$1:ok \rrbracket$))
 by (subst segr-middle[of ok ;_L mrg-right], simp-all)
 also have ... = (\exists ok₀ · \exists ok₁ · ($P \llbracket \llcorner ok_0 \gg / \$ok' \rrbracket \parallel_s Q \llbracket \llcorner ok_1 \gg / \$ok' \rrbracket$) ;; ($M \llbracket \llcorner ok_0 \gg, \llcorner ok_1 \gg / \$0:ok, \$1:ok \rrbracket$))
 by (rel-auto robust)
 also have ... = (
 ($(P^t \parallel_s Q^t)$;; ($M \llbracket true, true / \$0:ok, \$1:ok \rrbracket$)) \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 $P: P^f_f \Rightarrow P^t_f$
 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 ... = (
  ((( $pre_s \dagger P$ )  $\parallel_s$  ( $cmt_s \dagger Q$ )) ;; ( $N_0 M$  ;;  $R1(true)$ )))  $\vee$ 
  ((( $cmt_s \dagger P$ )  $\parallel_s$  ( $pre_s \dagger Q$ )) ;; ( $N_0 M$  ;;  $R1(true)$ )))
  by (rel-auto, metis+)
also have ... = (
  (( $pre_s \dagger P$ )  $\parallel_{N_0 M}$  ;;  $R1(true)$  ( $cmt_s \dagger Q$ ))  $\vee$ 
  (( $cmt_s \dagger P$ )  $\parallel_{N_0 M}$  ;;  $R1(true)$  ( $pre_s \dagger Q$ )))
  by (simp add: par-by-merge-def)
finally show  $?thesis$  .
qed

```

lemma pre_s -SRD:

assumes P is SRD

shows $pre_s \dagger P = (\neg_r pre_R(P))$

proof –

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

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

also have ... = $R1(R2c(\neg pre_s \dagger pre_R P))$

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)} ;; $R1(true)$ $cmt_R(Q)$) \wedge $\neg_r (cmt_R(P) \parallel_{N_0(M)} ;; $R1(true)$ ($\neg_r pre_R(Q)$)))$$

(is $?lhs = ?rhs$)

proof –

have $pre_R(P \parallel_{M_R(M)} Q) = (\neg_r (pre_s \dagger P) \parallel_{N_0 M} ;; $R1 true$ ($cmt_s \dagger Q$) \wedge $\neg_r (cmt_s \dagger P) \parallel_{N_0 M} ;; $R1 true$ ($pre_s \dagger Q$))$$

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]:

$\llbracket P \text{ is SRD}; Q \text{ is SRD} \rrbracket \Longrightarrow \$wait' \# pre_R(P \parallel_{M_R(M)} Q)$

by (simp add: parallel-assm, simp add: par-by-merge-def unrest)

lemma *JL1*: $(M_1 \ M)^t \llbracket \text{false}, \text{true} / \$0:\text{ok}, \$1:\text{ok} \rrbracket = N_0(M) \ ; \ ; \ R1(\text{true})$
 by (rel-blast)

lemma *JL2*: $(M_1 \ M)^t \llbracket \text{true}, \text{false} / \$0:\text{ok}, \$1:\text{ok} \rrbracket = N_0(M) \ ; \ ; \ R1(\text{true})$
 by (rel-blast)

lemma *JL3*: $(M_1 \ M)^t \llbracket \text{false}, \text{false} / \$0:\text{ok}, \$1:\text{ok} \rrbracket = N_0(M) \ ; \ ; \ R1(\text{true})$
 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 $\text{cmt}_s \dagger (P \parallel_{M_R(M)} Q) =$
 $((\text{cmt}_s \dagger P) \parallel_{(\$ok' \wedge N_0 \ M)} \ ; \ ; \ II_R^t (\text{cmt}_s \dagger Q)) \vee$
 $((\text{pre}_s \dagger P) \parallel_{N_0(M)} \ ; \ ; \ R1(\text{true}) (\text{cmt}_s \dagger Q)) \vee$
 $((\text{cmt}_s \dagger P) \parallel_{N_0(M)} \ ; \ ; \ R1(\text{true}) (\text{pre}_s \dagger Q))$
 proof –
 have $\text{cmt}_s \dagger (P \parallel_{M_R(M)} Q) = (P \llbracket \text{true}, \text{false} / \$ok, \$wait \rrbracket \parallel_{(M_1(M))^t} Q \llbracket \text{true}, \text{false} / \$ok, \$wait \rrbracket)$
 by (simp add: usubst, rel-auto)
 also have ... = $((P \llbracket \text{true}, \text{false} / \$ok, \$wait \rrbracket \parallel_s Q \llbracket \text{true}, \text{false} / \$ok, \$wait \rrbracket) \ ; \ ; \ (M_1 \ M)^t)$
 by (simp add: par-by-merge-def)
 also have ... =
 $((\text{cmt}_s \dagger P) \parallel_s (\text{cmt}_s \dagger Q)) \ ; \ ; \ ((M_1 \ M)^t \llbracket \text{true}, \text{true} / \$0:\text{ok}, \$1:\text{ok} \rrbracket) \vee$
 $((\text{pre}_s \dagger P) \parallel_s (\text{cmt}_s \dagger Q)) \ ; \ ; \ ((M_1 \ M)^t \llbracket \text{false}, \text{true} / \$0:\text{ok}, \$1:\text{ok} \rrbracket) \vee$
 $((\text{cmt}_s \dagger P) \parallel_s (\text{pre}_s \dagger Q)) \ ; \ ; \ ((M_1 \ M)^t \llbracket \text{true}, \text{false} / \$0:\text{ok}, \$1:\text{ok} \rrbracket) \vee$
 $((\text{pre}_s \dagger P) \parallel_s (\text{pre}_s \dagger Q)) \ ; \ ; \ ((M_1 \ M)^t \llbracket \text{false}, \text{false} / \$0:\text{ok}, \$1:\text{ok} \rrbracket)$
 by (subst parallel-ok-cases, subst-tac)
 also have ... =
 $((\text{cmt}_s \dagger P) \parallel_s (\text{cmt}_s \dagger Q)) \ ; \ ; \ ((M_1 \ M)^t \llbracket \text{true}, \text{true} / \$0:\text{ok}, \$1:\text{ok} \rrbracket) \vee$
 $((\text{pre}_s \dagger P) \parallel_s (\text{cmt}_s \dagger Q)) \ ; \ ; \ (N_0(M) \ ; \ ; \ R1(\text{true})) \vee$
 $((\text{cmt}_s \dagger P) \parallel_s (\text{pre}_s \dagger Q)) \ ; \ ; \ (N_0(M) \ ; \ ; \ R1(\text{true})) \vee$
 $((\text{pre}_s \dagger P) \parallel_s (\text{pre}_s \dagger Q)) \ ; \ ; \ (N_0(M) \ ; \ ; \ R1(\text{true}))$
 (is - = $(?C1 \vee_p ?C2 \vee_p ?C3 \vee_p ?C4)$)
 by (simp add: JL1 JL2 JL3)
 also have ... =
 $((\text{cmt}_s \dagger P) \parallel_s (\text{cmt}_s \dagger Q)) \ ; \ ; \ ((M_1(M))^t \llbracket \text{true}, \text{true} / \$0:\text{ok}, \$1:\text{ok} \rrbracket) \vee$
 $((\text{pre}_s \dagger P) \parallel_s (\text{cmt}_s \dagger Q)) \ ; \ ; \ (N_0(M) \ ; \ ; \ R1(\text{true})) \vee$
 $((\text{cmt}_s \dagger P) \parallel_s (\text{pre}_s \dagger Q)) \ ; \ ; \ (N_0(M) \ ; \ ; \ R1(\text{true}))$
 proof –
 from assms have $P^f \Rightarrow P^t$
 by (metis RD2-def H2-equivalence Healthy-def)
 hence $P: P_f^f \Rightarrow P_f^t$
 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 $cmt_s \dagger (P \parallel_{M_R(M)} Q) =$
 $((cmt_s \dagger P) \parallel_{(\$ok' \wedge N_0 M)} ;; II_R^t (cmt_s \dagger Q)) \vee pre_s \dagger (P \parallel_{M_R(M)} Q)$
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-pericondition-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-pericondition-lemma2:

assumes M is RDM
shows $(\exists \$st' \cdot N_0(M)) \llbracket true, true / \$ok', \$wait' \rrbracket = ((\$0:wait \vee \$1:wait) \wedge (\exists \$st' \cdot M))$

proof –

have $(\exists \$st' \cdot N_0(M)) \llbracket true, true / \$ok', \$wait' \rrbracket = (\exists \$st' \cdot (\$0:wait \vee \$1:wait) \wedge \$tr' \geq_u \$<:tr \wedge M)$
by (simp add: usubst unrest nmerge-rd0-def ex-unrest Healthy-if R1m-def assms)

also have ... = $(\exists \$st' \cdot (\$0:wait \vee \$1:wait) \wedge M)$
by (*metis (no-types, hide-lams) Healthy-if R1m-def R1m-idem R2m-def RDM-def assms utp-pred-laws.inf-commute*)
also have ... = $(\$0:wait \vee \$1:wait) \wedge (\exists \$st' \cdot M)$
by (*rel-auto*)
finally show ?thesis .
qed

lemma parallel-pericondition-lemma3:

$((\$0:wait \vee \$1:wait) \wedge (\exists \$st' \cdot M)) = ((\$0:wait \wedge \$1:wait \wedge (\exists \$st' \cdot M)) \vee (\neg \$0:wait \wedge \$1:wait \wedge (\exists \$st' \cdot M)))$
by (*rel-auto*)

lemma parallel-pericondition [rdes]:

fixes $M :: ('s, 't :: trace, 'a) \text{rsp merge}$

assumes $P \text{ is SRD } Q \text{ is SRD } M \text{ is RDM}$

shows $peri_R(P \parallel_{M_R(M)} Q) = (pre_R(P \parallel_{M_R} M Q) \Rightarrow_r peri_R(P) \parallel_{\exists \$st' \cdot M} peri_R(Q) \vee post_R(P) \parallel_{\exists \$st' \cdot M} peri_R(Q) \vee peri_R(P) \parallel_{\exists \$st' \cdot M} post_R(Q))$

proof –

have $peri_R(P \parallel_{M_R(M)} Q) =$
 $(pre_R(P \parallel_{M_R} M Q) \Rightarrow_r cmt_R P \parallel_{(\$ok' \wedge N_0 M) ;; II_R[true, true/\$ok', \$wait']} cmt_R Q)$
by (*simp add: peri-cmt-def parallel-commitment SRD-healths assms usubst unrest assms*)
also have ... = $(pre_R(P \parallel_{M_R} M Q) \Rightarrow_r cmt_R P \parallel_{(\exists \$st' \cdot N_0 M)[true, true/\$ok', \$wait']} cmt_R Q)$
by (*simp add: parallel-pericondition-lemma1*)
also have ... = $(pre_R(P \parallel_{M_R} M Q) \Rightarrow_r cmt_R P \parallel_{(\$0:wait \vee \$1:wait) \wedge (\exists \$st' \cdot M)} cmt_R Q)$
by (*simp add: parallel-pericondition-lemma2 assms*)
also have ... = $(pre_R(P \parallel_{M_R} M Q) \Rightarrow_r (([cmt_R P]_0 \wedge [cmt_R Q]_1 \wedge \$<:\mathbf{v}' =_u \$\mathbf{v}) ;; (\$0:wait \wedge \$1:wait \wedge (\exists \$st' \cdot M))) \vee ([cmt_R P]_0 \wedge [cmt_R Q]_1 \wedge \$<:\mathbf{v}' =_u \$\mathbf{v}) ;; (\neg \$0:wait \wedge \$1:wait \wedge (\exists \$st' \cdot M))) \vee ([cmt_R P]_0 \wedge [cmt_R Q]_1 \wedge \$<:\mathbf{v}' =_u \$\mathbf{v}) ;; (\$0:wait \wedge \neg \$1:wait \wedge (\exists \$st' \cdot M))))$
by (*simp add: par-by-merge-alt-def parallel-pericondition-lemma3 seqr-or-distr*)
also have ... = $(pre_R(P \parallel_{M_R} M Q) \Rightarrow_r (([peri_R P]_0 \wedge [peri_R Q]_1 \wedge \$<:\mathbf{v}' =_u \$\mathbf{v}) ;; (\exists \$st' \cdot M)) \vee ([post_R P]_0 \wedge [peri_R Q]_1 \wedge \$<:\mathbf{v}' =_u \$\mathbf{v}) ;; (\exists \$st' \cdot M)) \vee ([peri_R P]_0 \wedge [post_R Q]_1 \wedge \$<:\mathbf{v}' =_u \$\mathbf{v}) ;; (\exists \$st' \cdot M)))$
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 ... = $(pre_R(P \parallel_{M_R} M Q) \Rightarrow_r peri_R(P) \parallel_{\exists \$st' \cdot M} peri_R(Q) \vee post_R(P) \parallel_{\exists \$st' \cdot M} peri_R(Q) \vee peri_R(P) \parallel_{\exists \$st' \cdot M} post_R(Q))$
by (*simp add: par-by-merge-alt-def*)
finally show ?thesis .
qed

lemma parallel-postcondition-lemma1:

$(\$ok' \wedge P) ;; II_R[true, false/\$ok', \$wait'] = P[true, false/\$ok', \$wait']$
(is ?lhs = ?rhs)

proof –

have ?lhs = $(\$ok' \wedge P) ;; II[true, false/\$ok', \$wait']$
by (*rel-blast*)
also have ... = ?rhs
by (*rel-auto*)
finally show ?thesis .
qed

lemma *parallel-postcondition-lemma2*:

assumes M is RDM

shows $(N_0(M)) \llbracket \text{true}, \text{false} / \$ok', \$wait' \rrbracket = ((\neg \$0:\text{wait} \wedge \neg \$1:\text{wait}) \wedge M)$

proof –

have $(N_0(M)) \llbracket \text{true}, \text{false} / \$ok', \$wait' \rrbracket = ((\neg \$0:\text{wait} \wedge \neg \$1:\text{wait}) \wedge \$tr' \geq_u \$<:tr \wedge M)$

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 is SRD Q is SRD M 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 $\text{post}_R(P \parallel_{M_R(M)} Q) =$

$(\text{pre}_R(P \parallel_{M_R(M)} Q) \Rightarrow_r \text{cmt}_R P \parallel (\$ok' \wedge N_0 M) ;; H_R \llbracket \text{true}, \text{false} / \$ok', \$wait' \rrbracket \text{cmt}_R Q)$

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 segr-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 is NSRD Q is NSRD M 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 $\dots = ((\neg_r \text{pre}_R P)_0 \wedge [\text{peri}_R Q]_1 \wedge \$<:\mathbf{v}' =_u \$\mathbf{v}) \llbracket \text{true} / \$1:\text{wait}' \rrbracket ;; (N_0 M ;; R1 \text{true}) \llbracket \text{true} / \$1:\text{wait}' \rrbracket \vee$
 $(\neg_r \text{pre}_R P)_0 \wedge [\text{post}_R Q]_1 \wedge \$<:\mathbf{v}' =_u \$\mathbf{v}) \llbracket \text{false} / \$1:\text{wait}' \rrbracket ;; (N_0 M ;; R1 \text{true}) \llbracket \text{false} / \$1:\text{wait}' \rrbracket)$

by (simp add: cond-inter-var-split)
 also have ... = ($(\neg_r \text{pre}_R P]_0 \wedge [\text{peri}_R Q]_1 \wedge \$<:\mathbf{v}' =_u \$\mathbf{v}) ; N_0 M[\text{true}/\$1:\text{wait}] ; R1 \text{ true} \vee$
 $(\neg_r \text{pre}_R P]_0 \wedge [\text{post}_R Q]_1 \wedge \$<:\mathbf{v}' =_u \$\mathbf{v}) ; N_0 M[\text{false}/\$1:\text{wait}] ; R1 \text{ true})$
 by (simp add: usubst unrest)
 also have ... = ($(\neg_r \text{pre}_R P]_0 \wedge [\text{peri}_R Q]_1 \wedge \$<:\mathbf{v}' =_u \$\mathbf{v}) ; (\$wait' \wedge M) ; R1 \text{ true} \vee$
 $(\neg_r \text{pre}_R P]_0 \wedge [\text{post}_R Q]_1 \wedge \$<:\mathbf{v}' =_u \$\mathbf{v}) ; (\$wait' =_u \$0:\text{wait} \wedge M) ; R1 \text{ true})$
 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 ... = ($(\neg_r \text{pre}_R P]_0 \wedge [\text{peri}_R Q]_1 \wedge \$<:\mathbf{v}' =_u \$\mathbf{v}) ; M ; R1 \text{ true} \vee$
 $(\neg_r \text{pre}_R P]_0 \wedge [\text{post}_R Q]_1 \wedge \$<:\mathbf{v}' =_u \$\mathbf{v}) ; M ; R1 \text{ true})$
 (is $(?P_1 \vee_p ?P_2) = (?Q_1 \vee ?Q_2)$)
 proof –
 have $?P_1 = (\neg_r \text{pre}_R P]_0 \wedge [\text{peri}_R Q]_1 \wedge \$<:\mathbf{v}' =_u \$\mathbf{v}) ; (M \wedge \$wait') ; R1 \text{ true}$
 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 $?P_2 = ((\neg_r \text{pre}_R P]_0 \wedge [\text{post}_R Q]_1 \wedge \$<:\mathbf{v}' =_u \$\mathbf{v}) ; (M \wedge \$wait') ; R1 \text{ true} \vee$
 $(\neg_r \text{pre}_R P]_0 \wedge [\text{post}_R Q]_1 \wedge \$<:\mathbf{v}' =_u \$\mathbf{v}) ; (M \wedge \neg \$wait') ; R1 \text{ true})$
 by (subst segr-bool-split[of wait ;L mrg-left], 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 $\mathbf{M3}(N_1(M)) = \mathbf{M3}(\$ok' =_u (\$0:ok \wedge \$1:ok) \wedge$
 $\$wait' =_u (\$0:wait \vee \$1:wait) \wedge$
 $\$<:tr \leq_u \$tr' \wedge$
 $(\exists \{ \$0:ok, \$1:ok, \$<:ok, \$ok', \$0:wait, \$1:wait, \$<:wait, \$wait' \} \cdot RDM(M)))$
by (simp add: nmerge-rd1-def nmerge-rd0-def assms Healthy-if)
also have ... = $\mathbf{M3}(\$ok' =_u (\$0:ok \wedge \$1:ok) \wedge \$wait' =_u (\$0:wait \vee \$1:wait) \wedge RDM(M))$
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:

$\mathbf{M3}(N_R(M)) = (\exists \$<:st \cdot \$v' =_u \$<:v) \triangleleft \$<:wait \triangleright \mathbf{M3}(N_1 M) \triangleleft \$<:ok \triangleright (\$<:tr \leq_u \$tr')$
by (rel-blast)

lemma AssocMerge-nmerge-rd:

assumes M is RDM AssocMerge M
shows AssocMerge($N_R(M)$)

proof –

have $1:\mathbf{M3}(M) = rotate_m ;; \mathbf{M3}(M)$
using assms by (simp add: AssocMerge-def)
have $rotate_m ;; (\mathbf{M3}(N_R(M))) =$
 $rotate_m ;;$
 $((\exists \$<:st \cdot \$v' =_u \$<:v) \triangleleft \$<:wait \triangleright$
 $(\$ok' =_u (\$0:ok \wedge \$1:0:ok \wedge \$1:1:ok) \wedge \$wait' =_u (\$0:wait \vee \$1:0:wait \vee \$1:1:wait) \wedge$
 $\mathbf{M3}(M)) \triangleleft \$<:ok \triangleright$
 $(\$<:tr \leq_u \$tr'))$
by (simp add: AssocMerge-def nmerge-rd-merge3 nmerge-rd1-merge3 assms)
also have ... =
 $((\exists \$<:st \cdot \$v' =_u \$<:v) \triangleleft \$<:wait \triangleright$
 $(\$ok' =_u (\$0:ok \wedge \$1:0:ok \wedge \$1:1:ok) \wedge \$wait' =_u (\$0:wait \vee \$1:0:wait \vee \$1:1:wait) \wedge$
 $(rotate_m ;; \mathbf{M3}(M))) \triangleleft \$<:ok \triangleright$
 $(\$<:tr \leq_u \$tr'))$
by (rel-blast)
also have ... =
 $((\exists \$<:st \cdot \$v' =_u \$<:v) \triangleleft \$<:wait \triangleright$
 $(\$ok' =_u (\$0:ok \wedge \$1:0:ok \wedge \$1:1:ok) \wedge \$wait' =_u (\$0:wait \vee \$1:0:wait \vee \$1:1:wait) \wedge$
 $\mathbf{M3}(M)) \triangleleft \$<:ok \triangleright$
 $(\$<:tr \leq_u \$tr'))$
using 1 by auto
also have ... = $\mathbf{M3}(N_R(M))$
by (simp add: AssocMerge-def nmerge-rd-merge3 nmerge-rd1-merge3 assms)
finally show ?thesis
using AssocMerge-def by blast

qed

lemma *swap-merge-RDM-closed* [closure]:

assumes *M is RDM*

shows $\text{swap}_m \;; M \text{ is RDM}$

proof –

have $\text{RDM}(\text{swap}_m \;; \text{RDM}(M)) = (\text{swap}_m \;; \text{RDM}(M))$

by (*rel-auto*)

thus *?thesis*

by (*metis Healthy-def' assms*)

qed

lemma *parallel-precondition*:

fixes $M :: ('s, 't :: \text{trace}, 'a) \text{ rsp merge}$

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* ::

$('t :: \text{trace}, 'a) \text{ hrel-rp} \Rightarrow$

$('t :: \text{trace}, 'a) \text{ rp merge} \Rightarrow$

$('t, 'a) \text{ hrel-rp} \Rightarrow$

$('t, 'a) \text{ hrel-rp } (- \text{ wr}_R'(-) - [60, 0, 61] \ 61)$

where [*upred-defs*]: $Q \text{ wr}_R(M) P = (\neg_r ((\neg_r P) \parallel_M \;; R1(\text{true}) Q))$

lemma *wrR-R1* [closure]:

$M \text{ is R1m} \Longrightarrow Q \text{ wr}_R(M) P \text{ is R1}$

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 $((\neg_r P) \parallel_M Q) \;; R1(\text{true}_h) \text{ is R2}$

proof –

have $(\neg_r P) \parallel_M Q \text{ is } R2$
by (*simp add: closure assms*)
thus *?thesis*
by (*simp add: closure*)
qed

lemma *wrR-R2 [closure]*:
assumes $P \text{ is } R2 \ Q \text{ is } R2 \ M \text{ is } R2m$
shows $Q \text{ wr}_R(M) \ P \text{ is } R2$
proof –
have $((\neg_r P) \parallel_M Q) ;; R1(true_h) \text{ is } R2$
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 $P \text{ is } RR \ Q \text{ is } RR \ M \text{ is } RDM$
shows $Q \text{ wr}_R(M) \ P \text{ is } RR$
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 $P \text{ is } RR \ Q \text{ is } RR \ M \text{ is } RDM$
shows $(Q \text{ wr}_R(M) \ P) \text{ is } RC$
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]*: $(P \vee Q) \text{ wr}_R(M) \ R = (P \text{ wr}_R(M) \ R \wedge Q \text{ wr}_R(M) \ R)$
proof –
have $(P \vee Q) \text{ wr}_R(M) \ R =$
 $(\neg_r ((\neg_r R) ;; U0 \wedge (P ;; U1 \vee Q ;; U1) \wedge \$<:\mathbf{v}' =_u \$\mathbf{v}) ;; M ;; true_r)$
by (*simp add: wrR-def par-by-merge-def seqr-or-distl*)
also have $\dots = (\neg_r ((\neg_r R) ;; U0 \wedge P ;; U1 \wedge \$<:\mathbf{v}' =_u \$\mathbf{v} \vee (\neg_r R) ;; U0 \wedge Q ;; U1 \wedge \$<:\mathbf{v}' =_u$
 $\$ \mathbf{v}) ;; M ;; true_r)$
by (*simp add: conj-disj-distr utp-pred-laws.inf-sup-distrib2*)
also have $\dots = (\neg_r (((\neg_r R) ;; U0 \wedge P ;; U1 \wedge \$<:\mathbf{v}' =_u \$\mathbf{v}) ;; M ;; true_r \vee$
 $((\neg_r R) ;; U0 \wedge Q ;; U1 \wedge \$<:\mathbf{v}' =_u \$\mathbf{v}) ;; M ;; true_r))$
by (*simp add: seqr-or-distl*)
also have $\dots = (P \text{ wr}_R(M) \ R \wedge Q \text{ wr}_R(M) \ R)$
by (*simp add: wrR-def par-by-merge-def*)
finally show *?thesis* .
qed

lemma *wppR-impl [wp]*: $(P \Rightarrow_r Q) \text{ wr}_R(M) \ R = ((\neg_r P) \text{ wr}_R(M) \ R \wedge Q \text{ wr}_R(M) \ R)$
by (*simp add: rea-impl-def wp*)

lemma *wppR-miracle [wp]*: $false \text{ wr}_R(M) \ P = true_r$

by (simp add: wrR-def)

lemma wppR-true [wp]: $P \text{ wr}_R(M) \text{ true}_r = \text{true}_r$
 by (simp add: wrR-def)

lemma parallel-precondition-wr [rdes]:

assumes P is NSRD Q is NSRD M is RDM

shows $\text{pre}_R(P \parallel_{M_R(M)} Q) = (\text{peri}_R(Q) \text{ wr}_R(M) \text{ pre}_R(P) \wedge \text{post}_R(Q) \text{ wr}_R(M) \text{ pre}_R(P) \wedge \text{peri}_R(P) \text{ wr}_R(\text{swap}_m ;; M) \text{ pre}_R(Q) \wedge \text{post}_R(P) \text{ wr}_R(\text{swap}_m ;; M) \text{ pre}_R(Q))$

by (simp add: assms parallel-precondition wrR-def)

lemma rea-impl-merge-left: $(P \Rightarrow_r Q) \parallel_M R = ((\neg_r P) \parallel_M R) \vee (Q \parallel_M R)$

by (simp add: rea-impl-def par-by-merge-or-left)

lemma rea-impl-merge-right: $P \parallel_M (Q \Rightarrow_r R) = (P \parallel_M (\neg_r Q) \vee P \parallel_M R)$

by (simp add: rea-impl-def par-by-merge-or-right)

lemma parallel-pre-lemma:

$((Q_1 \Rightarrow_r Q_2) \text{ wr}_R(M) P_1 \wedge (P_1 \Rightarrow_r P_2) \parallel_{\exists \$st' . M} (Q_1 \Rightarrow_r Q_2))$
 $= ((Q_1 \Rightarrow_r Q_2) \text{ wr}_R(M) P_1 \wedge P_2 \parallel_{\exists \$st' . M} (Q_1 \Rightarrow_r Q_2))$

apply (simp add: par-by-merge-alt-def)

apply (rel-auto)

apply (meson order-refl)

apply (meson order-refl)

apply blast

apply blast

apply blast

apply blast

done

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

$\mathbf{R}_s(((Q_1 \Rightarrow_r Q_2) \text{ wr}_R(M) P_1 \wedge (Q_1 \Rightarrow_r Q_3) \text{ wr}_R(M) P_1 \wedge$
 $(P_1 \Rightarrow_r P_2) \text{ wr}_R(\text{swap}_m ;; M) Q_1 \wedge (P_1 \Rightarrow_r P_3) \text{ wr}_R(\text{swap}_m ;; M) Q_1) \vdash$
 $(P_2 \parallel_{\exists \$st' . M} Q_2 \vee$
 $P_3 \parallel_{\exists \$st' . M} Q_2 \vee P_2 \parallel_{\exists \$st' . M} Q_3) \diamond$
 $(P_3 \parallel_M Q_3)) \text{ (is ?lhs = ?rhs)})$

proof –

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

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

also have ... =

$\mathbf{R}_s(((Q_1 \Rightarrow_r Q_2) \text{ wr}_R(M) P_1 \wedge (Q_1 \Rightarrow_r Q_3) \text{ wr}_R(M) P_1 \wedge$
 $(P_1 \Rightarrow_r P_2) \text{ wr}_R(\text{swap}_m ;; M) Q_1 \wedge (P_1 \Rightarrow_r P_3) \text{ wr}_R(\text{swap}_m ;; M) Q_1) \vdash$
 $((P_1 \Rightarrow_r P_2) \parallel_{\exists \$st' . M} (Q_1 \Rightarrow_r Q_2) \vee$
 $(P_1 \Rightarrow_r P_3) \parallel_{\exists \$st' . M} (Q_1 \Rightarrow_r Q_2) \vee (P_1 \Rightarrow_r P_2) \parallel_{\exists \$st' . M} (Q_1 \Rightarrow_r Q_3)) \diamond$
 $((P_1 \Rightarrow_r P_3) \parallel_M (Q_1 \Rightarrow_r Q_3)))$

(is - = $\mathbf{R}_s(?X$
 $\vdash (?Y_1 \vee ?Y_2 \vee ?Y_3)$
 $\diamond ?Z))$

by (simp add: rdes closure unrest assms, rel-auto)

also have ... = $\mathbf{R}_s(?X \vdash ((?X \wedge ?Y_1) \vee (?X \wedge ?Y_2) \vee (?X \wedge ?Y_3)) \diamond (?X \wedge ?Z))$

by (rel-auto)

also have ... = $\mathbf{R}_s(?X \vdash ((?X \wedge P_2 \parallel_{\exists} \$st' . M Q_2) \vee (?X \wedge P_3 \parallel_{\exists} \$st' . M Q_2) \vee (?X \wedge P_2 \parallel_{\exists} \$st' . M Q_3)) \diamond (?X \wedge P_3 \parallel_M Q_3))$
proof –
 have 1: $(?X \wedge ?Y_1) = (?X \wedge P_2 \parallel_{\exists} \$st' . M Q_2)$
 by (rel-auto, meson order-refl, meson order-refl, meson order-refl, blast+)
 have 2: $(?X \wedge ?Y_2) = (?X \wedge P_3 \parallel_{\exists} \$st' . M Q_2)$
 by (rel-auto, meson order-refl, meson order-refl, meson order-refl, blast+)
 have 3: $(?X \wedge ?Y_3) = (?X \wedge P_2 \parallel_{\exists} \$st' . M Q_3)$
 by (rel-auto, meson order-refl, meson order-refl, meson order-refl, blast+)
 have 4: $(?X \wedge ?Z) = (?X \wedge P_3 \parallel_M Q_3)$
 by (rel-auto, meson order-refl, meson order-refl, meson order-refl, blast+)
 show ?thesis
 by (simp add: 1 2 3 4)
 qed
also have ... = ?rhs
 by (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 $pre_R(\text{Miracle} \parallel_{RM} P) = true_r$

by (simp add: parallel-assm wait'-cond-idem rdes closure assms)

moreover hence $cmt_R(\text{Miracle} \parallel_{RM} P) = false$

by (simp add: rdes closure wait'-cond-idem SRD-healths assms)

ultimately have $\text{Miracle} \parallel_{RM} P = \mathbf{R}_s(true_r \vdash false)$

by (metis NSRD-iff SRD-reactive-design-alt assms par-rdes-NSRD sdes-theory.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 $pre_R(P \parallel_{RM} \text{Miracle}) = true_r$

by (simp add: wait'-cond-idem parallel-assm rdes closure assms)

moreover hence $cmt_R(P \parallel_{RM} \text{Miracle}) = false$

by (simp add: wait'-cond-idem rdes closure SRD-healths assms)

ultimately have $P \parallel_{RM} \text{Miracle} = \mathbf{R}_s(true_r \vdash false)$

by (metis NSRD-iff SRD-reactive-design-alt assms par-rdes-NSRD sdes-theory.weak.top-closed)

thus ?thesis

by (simp add: Miracle-def R1-design-R1-pre)

qed

8.1 Example basic merge

definition *BasicMerge* :: $((s, t :: \text{trace}, \text{unit}) \text{ rsp}) \text{ merge } (N_B)$ **where**

$[upred-defs]: \text{BasicMerge} = (\$<:tr \leq_u \$tr' \wedge \$tr' - \$<:tr =_u \$0:tr - \$<:tr \wedge \$tr' - \$<:tr =_u \$1:tr - \$<:tr \wedge \$st' =_u \$<:st)$

abbreviation *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 *BasicMerge-SymMerge* [closure]:
 N_B is SymMerge
 by (rel-auto)

lemma *BasicMerge'-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 *rea-design-par* ::

$(\text{'s}, \text{'t}::\text{trace}, \text{'}\alpha) \text{ hrel-rsp} \Rightarrow (\text{'s}, \text{'t}, \text{'}\alpha) \text{ hrel-rsp} \Rightarrow (\text{'s}, \text{'t}, \text{'}\alpha) \text{ hrel-rsp}$ (infixr \parallel_R 85)
 where [upred-defs]: $P \parallel_R Q = \mathbf{R}_s((pre_R(P) \wedge pre_R(Q)) \vdash (cmt_R(P) \wedge cmt_R(Q)))$

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 $\$ok' \# P_1 \$ok' \# P_2$
 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* :: ($'s, 't :: \text{trace}, 'a$) *hrel-rsp* \Rightarrow ($'s, 't, 'a$) *hrel-rsp* **where**
 [*upred-defs*]: *Productive*(P) = $P \parallel_R \mathbf{R}_s(\text{true} \vdash \text{true} \diamond (\$tr <_u \$tr'))$

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($\text{SRD}(P)$) = $\mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \diamond (\text{post}_R(P) \wedge \$tr <_u \$tr'))$

proof –

have *Productive*($\text{SRD}(P)$) = $\mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \diamond \text{post}_R(P)) \parallel_R \mathbf{R}_s(\text{true} \vdash \text{true} \diamond (\$tr <_u \$tr'))$
 by (*simp add: Productive-def SRD-as-reactive-tri-design*)
also have ... = $\mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \diamond (\text{post}_R(P) \wedge \$tr <_u \$tr'))$
 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 is *SRD* $(\$tr <_u \$tr') \sqsubseteq (\text{pre}_R(P) \wedge \text{post}_R(P)) \$wait' \# \text{pre}_R(P)$

shows P is *Productive*
proof –
 have $P : \mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \diamond (\text{post}_R(P) \wedge \$tr <_u \$tr')) = P$
proof –
 have $\mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \diamond \text{post}_R(P)) = \mathbf{R}_s(\text{pre}_R(P) \vdash (\text{pre}_R(P) \wedge \text{peri}_R(P)) \diamond (\text{pre}_R(P) \wedge \text{post}_R(P)))$
 by (metis (no-types, hide-lams) design-export-pre wait'-cond-conj-exchange wait'-cond-idem)
 also have $\dots = \mathbf{R}_s(\text{pre}_R(P) \vdash (\text{pre}_R(P) \wedge \text{peri}_R(P)) \diamond (\text{pre}_R(P) \wedge (\text{post}_R(P) \wedge \$tr <_u \$tr')))$
 by (metis assms(2) utp-pred-laws.inf.absorb1 utp-pred-laws.inf.assoc)
 also have $\dots = \mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \diamond (\text{post}_R(P) \wedge \$tr <_u \$tr'))$
 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 is *SRD* P is *Productive* $\$wait' \nmid \text{pre}_R(P)$
 shows $(\$tr <_u \$tr') \sqsubseteq (\text{pre}_R(P) \wedge \text{post}_R(P))$
proof –
 have $\text{post}_R(P) = \text{post}_R(\mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \diamond (\text{post}_R(P) \wedge \$tr <_u \$tr')))$
 by (metis Healthy-def Productive-form assms(1) assms(2))
 also have $\dots = R1(R2c(\text{pre}_R(P) \Rightarrow (\text{post}_R(P) \wedge \$tr <_u \$tr')))$
 by (simp add: rea-post-RHS-design unrest usubst assms, rel-auto)
 also have $\dots = R1((\text{pre}_R(P) \Rightarrow (\text{post}_R(P) \wedge \$tr <_u \$tr')))$
 by (simp add: R2c-impl R2c-preR R2c-postR R2c-and R2c-tr-less-tr' assms)
 also have $(\$tr <_u \$tr') \sqsubseteq (\text{pre}_R(P) \wedge \dots)$
 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 ... = (pre_R P ⇒_r 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 post_R(Productive(P)) = (pre_R(P) ⇒_r post_R(P) ∧ \$tr <_u \$tr')
proof –
 have post_R(Productive(P)) = post_R(R_s(pre_R(P) ⊢ peri_R(P) ◇ (post_R(P) ∧ \$tr <_u \$tr'))
 by (metis Healthy-def NSRD-is-SRD Productive-form assms)
 also have ... = R1 (R2c (pre_R P ⇒_r post_R P ∧ \$tr' >_u \$tr))
 by (simp add: rea-post-RHS-design usubst unrest assms closure)
 also have ... = (pre_R P ⇒_r post_R P ∧ \$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 (pre_R P ∧ (pre_R P ∧ post_R P) ;; Q) = (pre_R P ∧ (post_R P ;; Q))
proof –
 have (pre_R P ∧ (post_R P ;; Q)) = (pre_R P ∧ ((pre_R P ⇒_r post_R P) ;; Q))
 by (simp add: SRD-post-under-pre assms closure unrest)
 also have ... = (pre_R P ∧ (((¬_r pre_R P) ;; Q ∨ (pre_R P ⇒_r R1(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 ... = (pre_R P ∧ (((¬_r pre_R P) ;; Q ∨ (pre_R P ∧ post_R P) ;; Q)))
proof –
 have (pre_R P ∨ ¬_r pre_R P) = R1 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 ... = (pre_R P ∧ (((¬_r pre_R P) ∨ (pre_R P ∧ post_R P) ;; Q)))
 by (simp add: NSRD-neg-pre-left-zero assms closure SRD-healths)
 also have ... = (pre_R P ∧ (pre_R P ∧ post_R 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 (R_s(P ⊢ Q ◇ R)) is Productive
proof (rule Productive-intro)
 show R_s (P ⊢ Q ◇ R) is SRD
 by (simp add: RHS-tri-design-is-SRD assms)

 from assms(1) show (\$tr' >_u \$tr) ⊆ (pre_R (R_s (P ⊢ Q ◇ R)) ∧ post_R (R_s (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' # pre_R (R_s (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 [closure]:
  assumes P is RR Q is RR R is RR R is R4
  shows (R_s(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 =
    R_s(pre_R(P) ⊢ peri_R(P) ◊ (post_R(P) ∧ $tr <_u $tr')) ⊓ R_s(pre_R(Q) ⊢ peri_R(Q) ◊ (post_R(Q) ∧
    $tr <_u $tr'))
  by (metis Healthy-if Productive-form assms)
  also have ... = R_s((pre_R P ∧ pre_R Q) ⊢ (peri_R P ∨ peri_R Q) ◊ ((post_R P ∧ $tr' >_u $tr) ∨ (post_R
    Q ∧ $tr' >_u $tr)))
  by (simp add: RHS-tri-design-choice)
  also have ... = R_s((pre_R P ∧ pre_R Q) ⊢ (peri_R P ∨ peri_R Q) ◊ (((post_R P) ∨ (post_R Q)) ∧ $tr'
    >_u $tr))
  by (rule cong[of R_s 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 ◁ b ▷_R Q is Productive
proof -
  have P ◁ b ▷_R Q =
    R_s(pre_R(P) ⊢ peri_R(P) ◊ (post_R(P) ∧ $tr <_u $tr')) ◁ b ▷_R R_s(pre_R(Q) ⊢ peri_R(Q) ◊ (post_R(Q)
    ∧ $tr <_u $tr'))
  by (metis Healthy-if Productive-form assms)
  also have ... = R_s((pre_R P ◁ b ▷_R pre_R Q) ⊢ (peri_R P ◁ b ▷_R peri_R Q) ◊ ((post_R P ∧ $tr' >_u
    $tr) ◁ b ▷_R (post_R Q ∧ $tr' >_u $tr)))
  by (simp add: cond-srea-form)
  also have ... = R_s((pre_R P ◁ b ▷_R pre_R Q) ⊢ (peri_R P ◁ b ▷_R peri_R Q) ◊ (((post_R P) ◁ b ▷_R (post_R
    Q)) ∧ $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 $P \;; Q = \mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \diamond (\text{post}_R(P) \wedge \$tr <_u \$tr')) \;; \mathbf{R}_s(\text{pre}_R(Q) \vdash \text{peri}_R(Q) \diamond (\text{post}_R(Q)))$

by (metis Healthy-def NSRD-is-SRD SRD-reactive-tri-design Productive-form assms(1) assms(2) assms(3))

also have ... = $\mathbf{R}_s((\text{pre}_R P \wedge (\text{post}_R P \wedge \$tr' >_u \$tr) \text{wp}_r \text{pre}_R Q) \vdash (\text{peri}_R P \vee ((\text{post}_R P \wedge \$tr' >_u \$tr) \;; \text{peri}_R Q)) \diamond ((\text{post}_R P \wedge \$tr' >_u \$tr) \;; \text{post}_R Q))$

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 ... = $\mathbf{R}_s((\text{pre}_R P \wedge (\text{post}_R P \wedge \$tr' >_u \$tr) \text{wp}_r \text{pre}_R Q) \vdash (\text{peri}_R P \vee ((\text{post}_R P \wedge \$tr' >_u \$tr) \;; \text{peri}_R Q)) \diamond ((\text{post}_R P \wedge \$tr' >_u \$tr) \;; \text{post}_R Q \wedge \$tr' >_u \$tr))$

proof –

have $((\text{post}_R P \wedge \$tr' >_u \$tr) \;; R1(\text{post}_R Q)) = ((\text{post}_R P \wedge \$tr' >_u \$tr) \;; R1(\text{post}_R Q) \wedge \$tr' >_u \$tr)$

by (rel-auto)

thus ?thesis

by (simp add: NSRD-is-SRD R1-post-SRD assms)

qed

also have ... 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 ... = $\mathbf{R}_s((\text{pre}_R P \wedge \text{post}_R P \text{wp}_r \text{pre}_R Q) \vdash (\text{peri}_R P \vee (\text{post}_R P \;; \text{peri}_R Q)) \diamond (\text{post}_R P \;; (\text{post}_R Q \wedge \$tr' >_u \$tr)))$

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 ... = $\mathbf{R}_s((\text{pre}_R P \wedge \text{post}_R P \text{wp}_r \text{pre}_R Q) \vdash (\text{peri}_R P \vee (\text{post}_R P \;; \text{peri}_R Q)) \diamond (\text{post}_R P \;; (\text{post}_R Q \wedge \$tr' >_u \$tr) \wedge \$tr' >_u \$tr))$

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 ... 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 *size* function that provides this.

```

class size-trace = trace + size +
  assumes

```

```

    size-zero: size 0 = 0 and
    size-nzero: s > 0  $\implies$  size(s) > 0 and
    size-plus: size (s + t) = size(s) + size(t)

```

— These axioms may be stronger than necessary. In particular, $(0::'a) < ?s \implies 0 < \text{size } ?s$ 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 ≤ t  $\implies$  size(s) ≤ size(t)
  by (metis le-add1 local.diff-add-cancel-left' local.size-plus)

```

```

lemma size-strict-mono: s < t  $\implies$  size(s) < 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; size(zs) > 0  $\rrbracket \implies$  thesis)  $\implies$  thesis
  by (metis local.diff-add-cancel-left' local.less-iff local.minus-gr-zero-iff local.size-nzero)

```

```

lemma size-minus-trace: y ≤ x  $\implies$  size(x - y) = size(x) - 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 (sizeu'(-))

```

```

translations
  sizeu(t) == CONST uop CONST size t

```

10.2 Guardedness

definition $gvert :: ((t::size\text{-}trace, 'a) \text{ } rp \times (t, 'a) \text{ } rp) \text{ } chain \text{ } \mathbf{where}$
 $[upred\text{-}defs]: gvert(n) \equiv (\$tr \leq_u \$tr' \wedge size_u(\&tt) <_u \ll n \gg)$

lemma $gvert\text{-}chain: chain \text{ } gvert$
apply ($simp \text{ } add: chain\text{-}def, safe$)
apply ($rel\text{-}simp$)
apply ($rel\text{-}simp$)
done

lemma $gvert\text{-}limit: \sqcap (range \text{ } gvert) = (\$tr \leq_u \$tr')$
by ($rel\text{-}auto$)

definition $Guarded :: ((t::size\text{-}trace, 'a) \text{ } hrel\text{-}rp \Rightarrow (t, 'a) \text{ } hrel\text{-}rp) \Rightarrow bool \text{ } \mathbf{where}$
 $[upred\text{-}defs]: Guarded(F) = (\forall X \text{ } n. (F(X) \wedge gvert(n+1)) = (F(X \wedge gvert(n)) \wedge gvert(n+1)))$

lemma $GuardedI: \llbracket \bigwedge X \text{ } n. (F(X) \wedge gvert(n+1)) = (F(X \wedge gvert(n)) \wedge gvert(n+1)) \rrbracket \Longrightarrow Guarded \text{ } F$
by ($simp \text{ } add: Guarded\text{-}def$)

Guarded reactive designs yield unique fixed-points.

theorem $guarded\text{-}fp\text{-}uniq:$
assumes $mono \text{ } F \text{ } F \in \llbracket id \rrbracket_H \rightarrow \llbracket SRD \rrbracket_H \text{ } Guarded \text{ } F$
shows $\mu \text{ } F = \nu \text{ } F$
proof –
have $constr \text{ } F \text{ } gvert$
using $assms$
by ($auto \text{ } simp \text{ } add: constr\text{-}def \text{ } gvert\text{-}chain \text{ } Guarded\text{-}def \text{ } tcontr\text{-}alt\text{-}def'$)
hence $(\$tr \leq_u \$tr' \wedge \mu \text{ } F) = (\$tr \leq_u \$tr' \wedge \nu \text{ } F)$
apply ($rule \text{ } constr\text{-}fp\text{-}uniq$)
apply ($simp \text{ } add: assms$)
using $gvert\text{-}limit$ **apply** $blast$
done
moreover **have** $(\$tr \leq_u \$tr' \wedge \mu \text{ } F) = \mu \text{ } F$
proof –
have $\mu \text{ } F \text{ } is \text{ } R1$
by ($rule \text{ } SRD\text{-}healths(1), rule \text{ } Healthy\text{-}\mu, simp\text{-}all \text{ } add: assms$)
thus $?thesis$
by ($metis \text{ } Healthy\text{-}def \text{ } R1\text{-}def \text{ } conj\text{-}comm$)
qed
moreover **have** $(\$tr \leq_u \$tr' \wedge \nu \text{ } F) = \nu \text{ } F$
proof –
have $\nu \text{ } F \text{ } is \text{ } R1$
by ($rule \text{ } SRD\text{-}healths(1), rule \text{ } Healthy\text{-}\nu, simp\text{-}all \text{ } add: assms$)
thus $?thesis$
by ($metis \text{ } Healthy\text{-}def \text{ } R1\text{-}def \text{ } conj\text{-}comm$)
qed
ultimately **show** $?thesis$
by ($simp$)
qed

lemma $Guarded\text{-}const \text{ } [closure]: Guarded \text{ } (\lambda X. P)$
by ($simp \text{ } add: Guarded\text{-}def$)

lemma $UINF\text{-}Guarded \text{ } [closure]:$
assumes $\bigwedge P. P \in A \Longrightarrow Guarded \text{ } P$


```

shows Guarded ( $\lambda X. \sqcap P \in A \cdot P(X)$ )
proof (rule GuardedI)
  fix  $X\ n$ 
  have  $\bigwedge Y. ((\sqcap P \in A \cdot P\ Y) \wedge \text{gvr}(n+1)) = ((\sqcap P \in A \cdot (P\ Y \wedge \text{gvr}(n+1))) \wedge \text{gvr}(n+1))$ 
  proof –
    fix  $Y$ 
    let  $?lhs = ((\sqcap P \in A \cdot P\ Y) \wedge \text{gvr}(n+1))$  and  $?rhs = ((\sqcap P \in A \cdot (P\ Y \wedge \text{gvr}(n+1))) \wedge \text{gvr}(n+1))$ 
    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  $((\sqcap P \in A \cdot (P\ X \wedge \text{gvr}(n+1))) \wedge \text{gvr}(n+1)) = ((\sqcap P \in A \cdot (P\ (X \wedge \text{gvr}(n)) \wedge \text{gvr}(n+1))) \wedge \text{gvr}(n+1))$ 
  proof –
    have  $(\sqcap P \in A \cdot (P\ X \wedge \text{gvr}(n+1))) = (\sqcap P \in A \cdot (P\ (X \wedge \text{gvr}(n)) \wedge \text{gvr}(n+1)))$ 
    proof (rule UINF-cong)
      fix  $P$  assume  $P \in A$ 
      thus  $(P\ X \wedge \text{gvr}(n+1)) = (P\ (X \wedge \text{gvr}(n)) \wedge \text{gvr}(n+1))$ 
        using Guarded-def assms by blast
    qed
    thus  $?thesis$  by simp
  qed
  ultimately show  $((\sqcap P \in A \cdot P\ X) \wedge \text{gvr}(n+1)) = ((\sqcap P \in A \cdot (P\ (X \wedge \text{gvr}(n)))) \wedge \text{gvr}(n+1))$ 
    by simp
qed

```

```

lemma intChoice-Guarded [closure]:
  assumes Guarded  $P$  Guarded  $Q$ 
  shows Guarded ( $\lambda X. P(X) \sqcap Q(X)$ )
proof –
  have Guarded ( $\lambda X. \sqcap F \in \{P, Q\} \cdot F(X)$ )
    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  $P :: ('s, 't::\text{size-trace}, 'a) \text{hrel-rsp}$ 
  assumes  $P$  is NSRD  $P$  is Productive
  shows Guarded ( $\lambda X. P ;; \text{SRD}(X)$ )
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 $a:(P ;; \text{SRD}(X) \wedge \text{gvt}(\text{Suc } n))\llbracket \text{false}/\$ok \rrbracket =$
 $(P ;; \text{SRD}(X \wedge \text{gvt } n) \wedge \text{gvt}(\text{Suc } n))\llbracket \text{false}/\$ok \rrbracket$
by (*simp add: usubst closure SRD-left-zero-1 assms*)
have $b:(P ;; \text{SRD}(X) \wedge \text{gvt}(\text{Suc } n))\llbracket \text{true}/\$ok \rrbracket\llbracket \text{true}/\$wait \rrbracket =$
 $((P ;; \text{SRD}(X \wedge \text{gvt } n) \wedge \text{gvt}(\text{Suc } n))\llbracket \text{true}/\$ok \rrbracket)\llbracket \text{true}/\$wait \rrbracket$
by (*simp add: usubst closure SRD-left-zero-2 assms*)
have $c:(P ;; \text{SRD}(X) \wedge \text{gvt}(\text{Suc } n))\llbracket \text{true}/\$ok \rrbracket\llbracket \text{false}/\$wait \rrbracket =$
 $((P ;; \text{SRD}(X \wedge \text{gvt } n) \wedge \text{gvt}(\text{Suc } n))\llbracket \text{true}/\$ok \rrbracket)\llbracket \text{false}/\$wait \rrbracket$
proof –
have $1:(P\llbracket \text{true}/\$wait' \rrbracket ;; (\text{SRD } X)\llbracket \text{true}/\$wait \rrbracket \wedge \text{gvt}(\text{Suc } n))\llbracket \text{true}, \text{false}/\$ok, \$wait \rrbracket =$
 $(P\llbracket \text{true}/\$wait' \rrbracket ;; (\text{SRD } (X \wedge \text{gvt } n))\llbracket \text{true}/\$wait \rrbracket \wedge \text{gvt}(\text{Suc } n))\llbracket \text{true}, \text{false}/\$ok, \$wait \rrbracket$
by (*metis (no-types, lifting) Healthy-def R3h-wait-true SRD-healths(3) SRD-idem*)
have $2:(P\llbracket \text{false}/\$wait' \rrbracket ;; (\text{SRD } X)\llbracket \text{false}/\$wait \rrbracket \wedge \text{gvt}(\text{Suc } n))\llbracket \text{true}, \text{false}/\$ok, \$wait \rrbracket =$
 $(P\llbracket \text{false}/\$wait' \rrbracket ;; (\text{SRD } (X \wedge \text{gvt } n))\llbracket \text{false}/\$wait \rrbracket \wedge \text{gvt}(\text{Suc } n))\llbracket \text{true}, \text{false}/\$ok, \$wait \rrbracket$
proof –
have $\text{exp}:\bigwedge Y::('s, 't, 'a) \text{ hrel-rsp. } (P\llbracket \text{false}/\$wait' \rrbracket ;; (\text{SRD } Y)\llbracket \text{false}/\$wait \rrbracket \wedge \text{gvt}(\text{Suc } n))\llbracket \text{true}, \text{false}/\$ok, \$wait \rrbracket$
 $=$
 $((\neg_r \text{pre}_R P) ;; (\text{SRD}(Y))\llbracket \text{false}/\$wait \rrbracket \vee (\text{post}_R P \wedge \$tr' >_u \$tr) ;; (\text{SRD}$
 $Y)\llbracket \text{true}, \text{false}/\$ok, \$wait \rrbracket))$
 $\wedge \text{gvt}(\text{Suc } n))\llbracket \text{true}, \text{false}/\$ok, \$wait \rrbracket$
proof –
fix $Y :: ('s, 't, 'a) \text{ hrel-rsp}$

have $(P\llbracket \text{false}/\$wait' \rrbracket ;; (\text{SRD } Y)\llbracket \text{false}/\$wait \rrbracket \wedge \text{gvt}(\text{Suc } n))\llbracket \text{true}, \text{false}/\$ok, \$wait \rrbracket =$
 $((\mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \diamond (\text{post}_R(P) \wedge \$tr <_u \$tr')))\llbracket \text{false}/\$wait' \rrbracket ;; (\text{SRD } Y)\llbracket \text{false}/\$wait \rrbracket$
 $\wedge \text{gvt}(\text{Suc } n))\llbracket \text{true}, \text{false}/\$ok, \$wait \rrbracket$
by (*metis (no-types) Healthy-def Productive-form assms(1) assms(2) NSRD-is-SRD*)
also have $\dots =$
 $((R1(R2c(\text{pre}_R(P) \Rightarrow (\$ok' \wedge \text{post}_R(P) \wedge \$tr <_u \$tr'))))\llbracket \text{false}/\$wait' \rrbracket ;; (\text{SRD } Y)\llbracket \text{false}/\$wait \rrbracket$
 $\wedge \text{gvt}(\text{Suc } n))\llbracket \text{true}, \text{false}/\$ok, \$wait \rrbracket$
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 $\dots =$
 $((\neg_r \text{pre}_R(P) \vee (\$ok' \wedge \text{post}_R(P) \wedge \$tr <_u \$tr'))\llbracket \text{false}/\$wait' \rrbracket ;; (\text{SRD } Y)\llbracket \text{false}/\$wait \rrbracket$
 $\wedge \text{gvt}(\text{Suc } n))\llbracket \text{true}, \text{false}/\$ok, \$wait \rrbracket$
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 $\dots =$
 $((\neg_r \text{pre}_R P) ;; (\text{SRD}(Y))\llbracket \text{false}/\$wait \rrbracket \vee (\$ok' \wedge \text{post}_R P \wedge \$tr' >_u \$tr) ;; (\text{SRD}$
 $Y)\llbracket \text{false}/\$wait \rrbracket)) \wedge \text{gvt}(\text{Suc } n))\llbracket \text{true}, \text{false}/\$ok, \$wait \rrbracket$
by (*simp add: usubst unrest assms closure seqr-or-distl NSRD-neg-pre-left-zero*)
also have $\dots =$
 $((\neg_r \text{pre}_R P) ;; (\text{SRD}(Y))\llbracket \text{false}/\$wait \rrbracket \vee (\text{post}_R P \wedge \$tr' >_u \$tr) ;; (\text{SRD } Y)\llbracket \text{true}, \text{false}/\$ok, \$wait \rrbracket))$
 $\wedge \text{gvt}(\text{Suc } n))\llbracket \text{true}, \text{false}/\$ok, \$wait \rrbracket$
proof –
have $(\$ok' \wedge \text{post}_R P \wedge \$tr' >_u \$tr) ;; (\text{SRD } Y)\llbracket \text{false}/\$wait \rrbracket =$
 $((\text{post}_R P \wedge \$tr' >_u \$tr) \wedge \$ok' =_u \text{true}) ;; (\text{SRD } Y)\llbracket \text{false}/\$wait \rrbracket$
by (*rel-blast*)
also have $\dots = (\text{post}_R P \wedge \$tr' >_u \$tr)\llbracket \text{true}/\$ok' \rrbracket ;; (\text{SRD } Y)\llbracket \text{false}/\$wait \rrbracket\llbracket \text{true}/\$ok \rrbracket$
using *seqr-left-one-point*[*of ok (post_R P \wedge \\$tr' >_u \\$tr) True (SRD Y)\llbracket \text{false}/\\$wait \rrbracket*]
by (*simp add: true-alt-def[THEN sym]*)
finally show *?thesis* **by** (*simp add: usubst unrest*)
qed
finally
show $(P\llbracket \text{false}/\$wait' \rrbracket ;; (\text{SRD } Y)\llbracket \text{false}/\$wait \rrbracket \wedge \text{gvt}(\text{Suc } n))\llbracket \text{true}, \text{false}/\$ok, \$wait \rrbracket =$

$$(((\neg_r \text{pre}_R P) ;; (\text{SRD}(Y))\llbracket \text{false}/\$wait \rrbracket \vee (\text{post}_R P \wedge \$tr' >_u \$tr) ;; (\text{SRD}(Y)\llbracket \text{true}, \text{false}/\$ok, \$wait \rrbracket))$$

$$\wedge \text{gvt} (Suc\ n))\llbracket \text{true}, \text{false}/\$ok, \$wait \rrbracket .$$

qed

have 1: $((\text{post}_R P \wedge \$tr' >_u \$tr) ;; (\text{SRD } X)\llbracket \text{true}, \text{false}/\$ok, \$wait \rrbracket \wedge \text{gvt} (Suc\ n)) =$
 $((\text{post}_R P \wedge \$tr' >_u \$tr) ;; (\text{SRD } (X \wedge \text{gvt } n))\llbracket \text{true}, \text{false}/\$ok, \$wait \rrbracket \wedge \text{gvt} (Suc\ n))$

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: $(\neg_r \text{pre}_R P) ;; (\text{SRD } X)\llbracket \text{false}/\$wait \rrbracket = (\neg_r \text{pre}_R P) ;; (\text{SRD}(X \wedge \text{gvt } n))\llbracket \text{false}/\$wait \rrbracket$

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 $(P ;; (\text{SRD } X) \wedge \text{gvt } (n+1))\llbracket \text{true}, \text{false}/\$ok, \$wait \rrbracket =$

$((P\llbracket \text{true}/\$wait' \rrbracket ;; (\text{SRD } X)\llbracket \text{true}/\$wait \rrbracket \wedge \text{gvt } (n+1))\llbracket \text{true}, \text{false}/\$ok, \$wait \rrbracket \vee$

$(P\llbracket \text{false}/\$wait' \rrbracket ;; (\text{SRD } X)\llbracket \text{false}/\$wait \rrbracket \wedge \text{gvt } (n+1))\llbracket \text{true}, \text{false}/\$ok, \$wait \rrbracket)$

by (*subst seqr-bool-split[of wait], simp-all add: usubst utp-pred-laws.distrib(4)*)

also

have $\dots = ((P\llbracket \text{true}/\$wait' \rrbracket ;; (\text{SRD } (X \wedge \text{gvt } n))\llbracket \text{true}/\$wait \rrbracket \wedge \text{gvt } (n+1))\llbracket \text{true}, \text{false}/\$ok, \$wait \rrbracket$

\vee

$(P\llbracket \text{false}/\$wait' \rrbracket ;; (\text{SRD } (X \wedge \text{gvt } n))\llbracket \text{false}/\$wait \rrbracket \wedge \text{gvt } (n+1))\llbracket \text{true}, \text{false}/\$ok, \$wait \rrbracket)$

by (*simp add: 1 2*)

also

have $\dots = ((P\llbracket \text{true}/\$wait' \rrbracket ;; (\text{SRD } (X \wedge \text{gvt } n))\llbracket \text{true}/\$wait \rrbracket \vee$

$P\llbracket \text{false}/\$wait' \rrbracket ;; (\text{SRD } (X \wedge \text{gvt } n))\llbracket \text{false}/\$wait \rrbracket) \wedge \text{gvt } (n+1))\llbracket \text{true}, \text{false}/\$ok, \$wait \rrbracket$

by (*simp add: usubst utp-pred-laws.distrib(4)*)

also have $\dots = (P ;; (\text{SRD } (X \wedge \text{gvt } n)) \wedge \text{gvt } (n+1))\llbracket \text{true}, \text{false}/\$ok, \$wait \rrbracket$

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) ∈  $\llbracket id \rrbracket_H \rightarrow \llbracket SRD \rrbracket_H$ 
    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 ... = (⋀ i. ((λX. P ;; SRD X) ^^ (i+1)) false)
    by (simp)
  also have ... = (⋀ i. P ^ (i+1)) ;; Miracle
proof (rule SUP-cong, simp-all)
  fix i
  show P ;; SRD (((λX. P ;; SRD X) ^^ i) false) = (P ;; P ^ i) ;; Miracle
proof (induct i)
  case 0
  then show ?case
    by (simp, metis srdes-theory.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.top-closed)
qed
qed
also have ... = (⋀ i. P ^ (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 $P :: ('s, 't::size-trace, 'a) hrel-rsp$
assumes P is NSRD P is Productive
shows $(\mu X \cdot P ;; SRD(X)) = (P ;; P^*) ;; Miracle$
proof –
have $(\mu X \cdot P ;; SRD(X)) = (\bigcap i \in UNIV \cdot P ;; P \hat{=} i) ;; Miracle$
by (*simp add: mu-csp-form-1 assms closure ustar-def*)
also have $\dots = (P ;; P^*) ;; Miracle$
by (*simp only: seq-UNIF-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*]:

$$[\sigma]_{S\sigma} \dagger \mathbf{R}_s(P \vdash Q \diamond R) = \mathbf{R}_s([\sigma]_{S\sigma} \dagger P \vdash ([\sigma]_{S\sigma} \dagger Q) \diamond ([\sigma]_{S\sigma} \dagger R))$$
by (*rel-auto*)

lemma *srd-subst-SRD-closed* [*closure*]:
assumes P is SRD
shows $[\sigma]_{S\sigma} \dagger P$ is SRD
proof –
have $SRD([\sigma]_{S\sigma} \dagger (SRD P)) = [\sigma]_{S\sigma} \dagger (SRD P)$
by (*rel-auto*)
thus ?thesis
by (*metis Healthy-def assms*)
qed

lemma *preR-srd-subst* [*rdes*]:

$$pre_R([\sigma]_{S\sigma} \dagger P) = [\sigma]_{S\sigma} \dagger pre_R(P)$$
by (*rel-auto*)

lemma *periR-srd-subst* [*rdes*]:

$$peri_R([\sigma]_{S\sigma} \dagger P) = [\sigma]_{S\sigma} \dagger peri_R(P)$$
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 $('s, 't::trace, 'a)$ *hrel-rsp* $(\langle - \rangle_R)$ **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* $(\langle - \rangle :=_R \langle - \rangle)$
-assign-srd :: *svids* \Rightarrow *uexprs* \Rightarrow *logic* (**infixr** $:=_R$ 62)

translations

-assign-srd xs $vs \Rightarrow$ *CONST assigns-srd* (*-mk-usubst* (id_s) xs vs)
-assign-srd x $v \Leftarrow$ *CONST assigns-srd* (*CONST subst-upd* (id_s) 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* (id_s) (*CONST pr-var* x)
 u) (*CONST pr-var* 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*)

lemma *taut-eq-impl-property*:
 $\llbracket vwb\text{-}lens\ x; \$x \# P \rrbracket \Longrightarrow '(\$x =_u \llbracket v \rrbracket \wedge Q) \Rightarrow P' = 'Q \llbracket \llbracket v \rrbracket / \$x \rrbracket \Rightarrow P'$
by (*rel-auto, meson mwb-lens-weak vwb-lens-mwb weak-lens.put-get*)

lemma *st-subst-taut-impl*:
assumes *vwb-lens* x $\$st:x \# Q$ P *is* *RR* Q *is* *RR*
shows $'[\&x \mapsto_s \llbracket k \rrbracket] \dagger_S P \Rightarrow Q' = '[\&x =_u \llbracket k \rrbracket]_{S<} \wedge P \Rightarrow Q'$ (**is** $?lhs = ?rhs$)

proof –

have $?lhs = 'P \llbracket \llbracket k \rrbracket / \$st:x \rrbracket \Rightarrow Q'$
by (*simp add: usubst-st-lift-def alpha usubst*)
also have $\dots = '(\$st:x =_u \llbracket k \rrbracket) \wedge RR(P) \Rightarrow RR(Q)'$
by (*simp add: Healthy-if assms taut-eq-impl-property*)
also have $\dots = '[\&x =_u \llbracket k \rrbracket]_{S<} \wedge RR(P) \Rightarrow RR(Q)'$
by (*rel-blast*)
finally show $?thesis$ **by** (*simp add: assms Healthy-if*)

qed

The following law explains how to refine a program Q when it is first initialised by an assignment. Would be good if it could be generalised to a more general precondition.

lemma *AssignR-init-refine-intro*:

assumes
 $vwb\text{-}lens\ x\ \$st:x\ \# \ P_2\ \$st:x\ \# \ P_3$
 $P_2\ is\ RR\ P_3\ is\ RR\ Q\ is\ NSRD$
 $\mathbf{R}_s([\&x =_u \ll k \gg]_{S<} \vdash P_2 \diamond P_3) \sqsubseteq Q$
shows $\mathbf{R}_s(true_r \vdash P_2 \diamond P_3) \sqsubseteq (x :=_R \ll k \gg) ;; Q$
proof –
have $\mathbf{R}_s([\&x =_u \ll k \gg]_{S<} \vdash P_2 \diamond P_3) \sqsubseteq \mathbf{R}_s(pre_R(Q) \vdash peri_R(Q) \diamond post_R(Q))$
by (*simp add: NSRD-is-SRD SRD-reactive-tri-design assms*)
hence $\mathbf{R}_s(true_r \vdash P_2 \diamond P_3) \sqsubseteq x :=_R \ll k \gg ;; \mathbf{R}_s(pre_R(Q) \vdash peri_R(Q) \diamond post_R(Q))$
proof (*clarsimp simp add: rdes-def assms closure unrest rpred wp RHS-tri-design-refine, safe*)
assume $a1: '[\&x =_u \ll k \gg]_{S<} \Rightarrow pre_R(Q)'$ **and** $a2: '[\&x =_u \ll k \gg]_{S<} \wedge peri_R(Q) \Rightarrow P_2'$ **and** $a3: '[\&x =_u \ll k \gg]_{S<} \wedge post_R(Q) \Rightarrow P_3'$
from $a1\ assms(1)$ **show** $'R1\ true \Rightarrow [\&x \mapsto_s \ll k \gg] \uparrow_S pre_R(Q)'$
by (*rel-simp*)
show $'[\&x \mapsto_s \ll k \gg] \uparrow_S peri_R(Q) \Rightarrow P_2'$
by (*simp add: a2 assms st-subst-taut-impl closure*)
show $'[\&x \mapsto_s \ll k \gg] \uparrow_S post_R(Q) \Rightarrow P_3'$
by (*simp add: a3 assms st-subst-taut-impl closure*)
qed
thus *?thesis*
by (*simp add: NSRD-is-SRD SRD-reactive-tri-design assms*)
qed

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 $\text{pre}_R (P \triangleleft b \triangleright_R Q)$ is RC
proof –
have 1: $(\lceil \neg b \rceil_{S<} \vee \neg_r \text{pre}_R P) ;; R1(\text{true}) = (\lceil \neg b \rceil_{S<} \vee \neg_r \text{pre}_R P)$
by (metis (no-types, lifting) NSRD-neg-pre-unit aext-not assms(1) seqr-or-distl st-lift-R1-true-right)
have 2: $(\lceil b \rceil_{S<} \vee \neg_r \text{pre}_R Q) ;; R1(\text{true}) = (\lceil b \rceil_{S<} \vee \neg_r \text{pre}_R Q)$
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 $\$st' \# \text{peri}_R (P \triangleleft b \triangleright_R Q)$
by (simp add: rdes assms closure unrest)
qed

11.4 Assumptions

definition $\text{AssumeR} :: 's \text{ cond} \Rightarrow ('s, 't::\text{trace}, 'a) \text{ hrel-rsp } ([\cdot]^\top_R)$ **where**
 $[\text{upred-defs}]: \text{AssumeR } b = II_R \triangleleft b \triangleright_R \text{Miracle}$

lemma AssumeR-rdes-def [rdes-def]:
 $[\text{b}]^\top_R = \mathbf{R}_s(\text{true}_r \vdash \text{false} \diamond [\text{b}]^\top_r)$
unfolding AssumeR-def **by** (rdes-eq)

lemma AssumeR-NSRD [closure]: $[\text{b}]^\top_R$ is NSRD
by (simp add: AssumeR-def closure)

lemma AssumeR-false : $[\text{false}]^\top_R = \text{Miracle}$
by (rel-auto)

lemma AssumeR-true : $[\text{true}]^\top_R = II_R$
by (rel-auto)

lemma AssumeR-comp : $[\text{b}]^\top_R ;; [\text{c}]^\top_R = [\text{b} \wedge \text{c}]^\top_R$
by (rdes-simp)

lemma AssumeR-choice : $[\text{b}]^\top_R \sqcap [\text{c}]^\top_R = [\text{b} \vee \text{c}]^\top_R$
by (rdes-eq)

lemma $\text{AssumeR-refine-skip}$: $II_R \sqsubseteq [\text{b}]^\top_R$
by (rdes-refine)

lemma AssumeR-test [closure]: $\text{test}_R [\text{b}]^\top_R$
by (simp add: AssumeR-refine-skip nsrdes-theory.utest-intro)

lemma Star-AssumeR : $[\text{b}]^\top_R^{\star R} = II_R$
by (simp add: StarR-def AssumeR-NSRD AssumeR-test nsrdes-theory.Star-test)

lemma $\text{AssumeR-choice-skip}$: $II_R \sqcap [\text{b}]^\top_R = II_R$
by (rdes-eq)

lemma $\text{AssumeR-seq-refines}$:
assumes P is NSRD

shows $P \sqsubseteq P ;; [b]^\top_R$
by (*rdes-refine cls: assms*)

lemma *cond-srea-AssumeR-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-eq cls: assms*)

lemma *cond-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 add: AssumeR-NSRD AssumeR-comp NSRD-seqr-closure RA1 assms cond-srea-AssumeR-form*)

lemma *AssumeR-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-eq cls: assms*)

lemma *AssumeR-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-eq cls: assms*)

11.5 Guarded commands

definition *GuardedCommR* :: $'s \text{ cond} \Rightarrow ('s, 't::\text{trace}, 'a) \text{ hrel-rsp} \Rightarrow ('s, 't, 'a) \text{ hrel-rsp} (- \rightarrow_R - [85, 86] 85)$ **where**
gcmd-def[rdes-def]: *GuardedCommR* $g A = A \triangleleft g \triangleright_R \text{Miracle}$

lemma *gcmd-false[simp]*: $(\text{false} \rightarrow_R A) = \text{Miracle}$
unfolding *gcmd-def* **by** (*pred-auto*)

lemma *gcmd-true[simp]*: $(\text{true} \rightarrow_R A) = A$
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.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*)

lemma *AssumeR-gcomm*:

assumes P is NSRD
shows $[b]^\top_R ;; (c \rightarrow_R P) = (b \wedge c) \rightarrow_R P$
by (*rdes-eq cls: assms*)

11.6 Generalised Alternation

definition *AlternateR*

$:: 'a \text{ set} \Rightarrow ('a \Rightarrow 's \text{ upred}) \Rightarrow ('a \Rightarrow ('s, 't::\text{trace}, 'a) \text{ hrel-rsp}) \Rightarrow ('s, 't, 'a) \text{ hrel-rsp} \Rightarrow ('s, 't, 'a) \text{ hrel-rsp}$ **where**
[upred-defs, rdes-def]: $\text{AlternateR } I \ g \ A \ B = (\bigsqcap i \in I \cdot ((g \ i) \rightarrow_R (A \ i))) \sqcap ((\neg (\bigvee i \in I \cdot g \ i)) \rightarrow_R B)$

definition *AlternateR-list*

$:: ('s \text{ upred} \times ('s, 't::\text{trace}, 'a) \text{ hrel-rsp}) \text{ list} \Rightarrow ('s, 't, 'a) \text{ hrel-rsp} \Rightarrow ('s, 't, 'a) \text{ hrel-rsp}$ **where**
[upred-defs, ndes-simp]:
 $\text{AlternateR-list } xs \ P = \text{AlternateR } \{0..<\text{length } xs\} \ (\lambda i. \text{map fst } xs \ ! \ i) \ (\lambda i. \text{map snd } xs \ ! \ i) \ P$

syntax

-altindR-els $:: \text{pttrn} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic} \ (if_R \ - \in \ - \cdot \ - \rightarrow \ - \text{ else } - \text{ fi})$
-altindR $:: \text{pttrn} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic} \ (if_R \ - \in \ - \cdot \ - \rightarrow \ - \text{ fi})$
-altgcommR-els $:: \text{gcomms} \Rightarrow \text{logic} \Rightarrow \text{logic} \ (if_R / \ - \text{ else } - / \text{ fi})$
-altgcommR $:: \text{gcomms} \Rightarrow \text{logic} \ (if_R / \ - / \text{ fi})$

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 $\bigwedge i. i \in I \Rightarrow A \ i \text{ is NSRD } B \text{ is NSRD}$
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 $(\text{if}_R i \in I \cdot g \ i \rightarrow A \ i \text{ else } B \text{ fi}) \text{ is Productive}$
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 $(\text{if}_R i \in \{k\} \cdot g \ i \rightarrow A \ i \text{ else } B \text{ fi}) = (A(k) \triangleleft g(k) \triangleright_R B)$
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 $(\text{if}_R i \in \text{insert } k \ I \cdot g \ i \rightarrow A \ i \text{ else } B \text{ fi}) = (A(k) \triangleleft g(k) \triangleright_R (\text{if}_R i \in I \cdot g(i) \rightarrow A(i) \text{ else } B \text{ fi}))$
proof (cases $I = \{\}$)
case *True*
then show ?thesis **by** (simp add: AlternateR-singleton assms)
next
case *False*
have 1: $(\bigcap i \in I \cdot g \ i \rightarrow_R A \ i) = (\bigcap i \in I \cdot g \ i \rightarrow_R \mathbf{R}_s(\text{pre}_R(A \ i) \vdash \text{peri}_R(A \ i) \diamond \text{post}_R(A \ i)))$
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)
 $(\text{rdes-eq cls: assms}(1-3) \text{ False cong: UINF-cong})$
qed

lemma *AlternateR-assume-branch*:
assumes $I \neq \{\}$ $\bigwedge i. i \in I \implies P \ i \text{ is NSRD } Q \text{ is NSRD}$
shows $(\bigcap i \in I \cdot b \ i]^\top_R ;; \text{AlternateR } I \ b \ P \ Q) = (\bigcap i \in I \cdot b \ i \rightarrow_R P \ i) \ (\text{is } ?lhs = ?rhs)$
proof –
have ?lhs = $\bigcap i \in I \cdot b \ i]^\top_R ;; ((\bigcap i \in I \cdot b \ i \rightarrow_R P \ i) \sqcap (\neg (\bigcap i \in I \cdot b \ i)) \rightarrow_R Q)$
by (simp add: AlternateR-def closure assms)
also have ... = $\bigcap i \in I \cdot b \ i]^\top_R ;; (\bigcap i \in I \cdot b \ i \rightarrow_R P \ i) \sqcap \text{Miracle}$
by (simp add: seqr-inf-distr AssumeR-gcomm closure assms)
also have ... = $(\bigcap i \in I \cdot ((\bigcap i \in I \cdot b \ i) \wedge b \ i) \rightarrow_R P \ i) \sqcap \text{Miracle}$
by (simp add: seq-UINF-distl AssumeR-gcomm closure assms cong: UINF-cong)
also have ... = $(\bigcap i \in I \cdot b \ i \rightarrow_R P \ i) \sqcap \text{Miracle}$
proof –
have $\bigwedge i. i \in I \implies ((\bigcap i \in I \cdot b \ i) \wedge b \ i) = b \ i$
by (rel-auto)
thus ?thesis

by (simp cong: UINF-cong)
 qed
 also have ... = ($\bigcap i \in I \cdot b \ i \rightarrow_R P \ i$)
 by (simp add: closure assms)
 finally show ?thesis .
 qed

11.7 Choose

definition choose-srd :: ('s, 't::trace, 'α) hrel-rsp (choose_R) **where**
 [upred-defs, rdes-def]: choose_R = $\mathbf{R}_s(\text{true}_r \vdash \text{false} \diamond \text{true}_r)$

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 Divergence Freedom

definition ndiv-srd :: ('s, 't::trace, 'α) hrel-rsp (ndiv_R)
where [rdes-def]: ndiv-srd = $\mathbf{R}_s(\text{true}_r \vdash \text{true}_r \diamond \text{true}_r)$

lemma ndiv-NSRD [closure]: ndiv_R is NSRD
 by (simp add: rdes-def closure unrest)

lemma ndiv-srd-refines-preR-true:
 assumes P is SRD
 shows $\text{ndiv}_R \sqsubseteq P \longleftrightarrow \text{pre}_R(P) = \text{true}_r$ (**is** ?lhs \longleftrightarrow ?rhs)

proof

assume ?lhs

thus ?rhs

by (metis R1-preR ndiv-srd-def preR-antitone preR-rdes rea-true-RR rea-true-disj(2) utp-pred-laws.sup.orderE)

next

assume ?rhs

hence $\text{ndiv}_R \sqsubseteq \mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \diamond \text{post}_R(P))$

by (simp add: RHS-tri-design-conj assms ndiv-srd-def periR-SRD-R1 postR-SRD-R1 rea-true-conj(1) rea-true-impl utp-pred-laws.inf.absorb-iff2)

thus ?lhs

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

qed

lemma ndiv-srd-refines-rdes-pre-true:
 assumes P_1 is RR P_2 is RR P_3 is RR
 shows $\text{ndiv}_R \sqsubseteq \mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) \longleftrightarrow P_1 = \text{true}_r$ (**is** ?lhs \longleftrightarrow ?rhs)
 by (simp add: ndiv-srd-refines-preR-true closure assms rdes unrest)

11.9 State Abstraction

definition *state-srea* ::

's itself \Rightarrow (*'s, 't::trace, 'α, 'β*) *rel-rsp* \Rightarrow (*unit, 't, 'α, 'β*) *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* - - - [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*(*state 'a* \cdot *P*) = (*state 'a* \cdot *RD2*(*P*))
by (*rel-auto*)

lemma *RD3-state-srea*: *RD3*(*state 'a* \cdot *P*) = (*state 'a* \cdot *RD3*(*P*))
by (*rel-auto*, *blast+*)

lemma *SRD-state-srea* [*closure*]: *P* is *SRD* \implies *state 'a* \cdot *P* 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* is *NSRD* \implies *state 'a* \cdot *P* 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*]: *pre_R*(*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*]: *peri_R*(*state 'a* \cdot *P*) = *state 'a* \cdot *peri_R*(*P*)
by (*rel-auto*)

lemma *postR-state-srea* [*rdes*]: *post_R*(*state 'a* \cdot *P*) = *state 'a* \cdot *post_R*(*P*)
by (*rel-auto*)

lemma *state-srea-rdes-def* [*rdes-def*]:
assumes *P* is *RC* *Q* is *RR* *R* is *RR*
shows *state 'a* \cdot **R_s**(*P* \vdash *Q* \diamond *R*) = **R_s**($\langle \forall \{ \$st, \$st' \} \cdot P \rangle_S \vdash$ (*state 'a* \cdot *Q*) \diamond (*state 'a* \cdot *R*))
(is ?lhs = ?rhs)

proof –

have *?lhs* = **R_s**(*pre_R*(*?lhs*) \vdash *peri_R*(*?lhs*) \diamond *post_R*(*?lhs*))
by (*simp add: RC-implies-RR SRD-rdes-intro SRD-reactive-tri-design SRD-state-srea assms*)
also have ... = **R_s**($\langle \forall \{ \$st, \$st' \} \cdot P \rangle_S \vdash$ *state 'a* \cdot (*P* \Rightarrow_r *Q*) \diamond *state 'a* \cdot (*P* \Rightarrow_r *R*))
by (*simp add: rdes closure assms*)
also have ... = *?rhs*

by (rel-auto)
 finally show ?thesis .
 qed

lemma ext-st-rdes-dist [rdes-def]:
 $\mathbf{R}_s(P \vdash Q \diamond R) \oplus_p \text{abs-st}_L = \mathbf{R}_s(P \oplus_p \text{abs-st}_L \vdash Q \oplus_p \text{abs-st}_L \diamond R \oplus_p \text{abs-st}_L)$
 by (rel-auto)

lemma state-srea-refine:
 $(P \oplus_p \text{abs-st}_L) \sqsubseteq Q \implies P \sqsubseteq (\text{state-srea } \text{TYPE}('s) \ Q)$
 by (rel-auto)

11.10 Reactive Frames

definition rdes-frame-ext :: $(' \alpha \implies ' \beta) \Rightarrow (' \alpha, 't::\text{trace}, 'r) \text{ hrel-rsp} \Rightarrow (' \beta, 't, 'r) \text{ hrel-rsp}$ **where**
 [upred-defs, rdes-def]: $\text{rdes-frame-ext } a \ P = \mathbf{R}_s(\text{rel-aext } (\text{pre}_R(P)) (\text{map-st}_L \ a) \vdash \text{rel-aext } (\text{peri}_R(P)) (\text{map-st}_L \ a) \diamond a : [\text{post}_R(P)]_r^+)$

syntax
 $\text{-rdes-frame-ext} :: \text{salpha} \Rightarrow \text{logic} \Rightarrow \text{logic } (:-)_{R^+} [99,0] \ 100)$

translations
 $\text{-rdes-frame-ext } x \ P \Rightarrow \text{CONST } \text{rdes-frame-ext } x \ P$
 $\text{-rdes-frame-ext } (-\text{salphaset } (-\text{salphamk } x)) \ P \Leftarrow \text{CONST } \text{rdes-frame-ext } x \ P$

lemma RC-rel-aext-st-closed [closure]:
 assumes P is RC
 shows $\text{rel-aext } P (\text{map-st}_L \ a)$ is RC
proof –
 have $\text{RC}(\text{rel-aext } (\text{RC}(P)) (\text{map-st}_L \ a)) = \text{rel-aext } (\text{RC}(P)) (\text{map-st}_L \ a)$
 by (rel-auto)
 (metis (no-types, hide-lams) diff-add-cancel-left' dual-order.trans le-add trace-class.add-diff-cancel-left trace-class.add-left-mono)
 thus ?thesis
 by (rule-tac Healthy-intro, simp add: assms Healthy-if)
 qed

lemma rdes-frame-ext-SRD-closed:
 $\llbracket P \text{ is SRD}; \$\text{wait}' \ \# \ \text{pre}_R(P) \rrbracket \implies a : [P]_{R^+} \text{ is SRD}$
 unfolding rdes-frame-ext-def
 apply (rule SRD-rdes-intro)
 apply (simp-all add: closure unrest)
 apply (simp add: RR-R2-intro ok'-pre-unrest ok-pre-unrest preR-R2-closed rea-aext-RR wait-pre-unrest)
 done

lemma preR-rdes-frame-ext:
 $P \text{ is NSRD} \implies \text{pre}_R(a : [P]_{R^+}) = \text{rel-aext } (\text{pre}_R(P)) (\text{map-st}_L \ a)$
 by (simp add: preR-RR preR-rdes rdes-frame-ext-def rea-aext-RR)

lemma unrest-rel-aext-st' [unrest]: $\$st' \ \# \ P \implies \$st' \ \# \ \text{rel-aext } P (\text{map-st}_L \ a)$
 by (rel-auto)

lemma rdes-frame-ext-NSRD-closed:
 $P \text{ is NSRD} \implies a : [P]_{R^+} \text{ is NSRD}$
 apply (rule NSRD-RC-intro)
 apply (rule rdes-frame-ext-SRD-closed)

apply (*simp-all add: closure unrest rdes*)
apply (*simp add: NSRD-neg-pre-RC RC-rel-aext-st-closed preR-RR preR-rdes rdes-frame-ext-def rea-aext-RR*)
apply (*simp add: rdes-frame-ext-def*)
apply (*simp add: rdes closure unrest*)
done

lemma *skip-srea-frame* [*frame*]:
vwb-lens a $\implies a:[II_R]_R^+ = II_R$
by (*rdes-eq*)

lemma *seq-srea-frame* [*frame*]:
assumes *vwb-lens a P is NSRD Q is NSRD*
shows *a:[P ;; Q]_R⁺ = a:[P]_R⁺ ;; a:[Q]_R⁺ (is ?lhs = ?rhs)*
proof –
have *?lhs = $\mathbf{R}_s ((pre_R P \wedge post_R P wp_r pre_R Q) \oplus_r map-st_L[a] \vdash$*
 $((pre_R P \wedge post_R P wp_r pre_R Q) \oplus_r map-st_L[a] \Rightarrow_r (peri_R P \vee post_R P ;; peri_R Q)$
 $\oplus_r map-st_L[a]) \diamond$
a:[pre_R P \wedge post_R P wp_r pre_R Q $\Rightarrow_r post_R P ;; post_R Q$]_R⁺
using *assms(1) by (rdes-simp cls: assms(2-3))*
also have *... = $\mathbf{R}_s ((pre_R P \wedge post_R P wp_r pre_R Q) \oplus_r map-st_L[a] \vdash$*
 $((peri_R P \vee post_R P ;; peri_R Q) \oplus_r map-st_L[a]) \diamond$
a:[post_R P ;; post_R Q]_R⁺
by (*rel-auto*)
also from *assms(1) have ... = ?rhs*
apply (*rdes-eq-split cls: assms(2-3)*)
apply (*rel-auto*)
apply (*metis mwb-lens-def vwb-lens-mwb weak-lens.put-get*)
apply (*rel-auto*)
apply (*metis mwb-lens-def vwb-lens-mwb weak-lens.put-get*)
apply (*metis vwb-lens-wb wb-lens-def weak-lens.put-get*)
apply (*metis mwb-lens-def vwb-lens-mwb weak-lens.put-get*)
apply (*rel-auto*)
apply (*metis mwb-lens-def vwb-lens-mwb weak-lens.put-get*)
done
finally show *?thesis .*
qed

lemma *rdes-frame-ext-Productive-closed* [*closure*]:
assumes *P is NSRD P is Productive*
shows *x:[P]_R⁺ is Productive*
proof –
have *x:[Productive(P)]_R⁺ is Productive*
by (*rdes-simp cls: assms, rel-auto*)
thus *?thesis*
by (*simp add: Healthy-if assms*)
qed

11.11 While Loop

definition *WhileR* :: '*s upred* \Rightarrow ('*s*, '*t*::size-trace, '*α*) *hrel-rsp* \Rightarrow ('*s*, '*t*, '*α*) *hrel-rsp* **where**
WhileR b P = ($\mu_R X \cdot (P ;; X) \triangleleft b \triangleright_R II_R$)

syntax *-whileR* :: *uexp* \Rightarrow *logic* \Rightarrow *logic* (*while_R* - *do* - *od*)
translations *-whileR b P == CONST WhileR b P*

lemma *Sup-power-false*:

fixes $F :: 'a \text{ upred} \Rightarrow 'a \text{ upred}$

shows $(\bigcap i. (F \text{ ^^ } i) \text{ false}) = (\bigcap i. (F \text{ ^^ } (i+1)) \text{ false})$

proof –

have $(\bigcap i. (F \text{ ^^ } i) \text{ false}) = (F \text{ ^^ } 0) \text{ false} \sqcap (\bigcap i. (F \text{ ^^ } (i+1)) \text{ false})$

by (*subst Sup-power-expand, simp*)

also have $\dots = (\bigcap i. (F \text{ ^^ } (i+1)) \text{ false})$

by (*simp*)

finally show *?thesis* .

qed

theorem *WhileR-unfold*:

assumes P is NSRD

shows $\text{while}_R b \text{ do } P \text{ od} = (P ;; \text{while}_R b \text{ do } P \text{ od}) \triangleleft b \triangleright_R II_R$

apply (*simp add: WhileR-def*)

apply (*subst srdes-theory.LFP-unfold*)

apply (*simp-all add: mono-Monotone-utp-order closure assms*)

done

theorem *WhileR-iter-expand*:

assumes P is NSRD P is Productive

shows $\text{while}_R b \text{ do } P \text{ od} = (\bigcap i. (P \triangleleft b \triangleright_R II_R) \text{ ^^ } i ;; (P ;; \text{Miracle} \triangleleft b \triangleright_R II_R))$ (**is** *?lhs = ?rhs*)

proof –

have $1:\text{Continuous } (\lambda X. P ;; \text{SRD } X)$

using *SRD-Continuous*

by (*clarsimp simp add: Continuous-def seq-SUP-distl[THEN sym], drule-tac x=A in spec, simp*)

have $2:\text{Continuous } (\lambda X. P ;; \text{SRD } X \triangleleft b \triangleright_R II_R)$

by (*simp add: 1 closure assms*)

have $?lhs = (\mu_R X \cdot P ;; X \triangleleft b \triangleright_R II_R)$

by (*simp add: WhileR-def*)

also have $\dots = (\mu X \cdot P ;; \text{SRD}(X) \triangleleft b \triangleright_R II_R)$

by (*auto simp add: srd-mu-equiv closure assms*)

also have $\dots = (\nu X \cdot P ;; \text{SRD}(X) \triangleleft b \triangleright_R II_R)$

by (*auto simp add: guarded-fp-uniq Guarded-if-Productive[OF assms] funcsetI closure assms*)

also have $\dots = (\bigcap i. ((\lambda X. P ;; \text{SRD } X \triangleleft b \triangleright_R II_R) \text{ ^^ } i) \text{ false})$

by (*simp add: sup-continuous-lfp 2 sup-continuous-Continuous false-upred-def*)

also have $\dots = (\bigcap i. ((\lambda X. P ;; \text{SRD } X \triangleleft b \triangleright_R II_R) \text{ ^^ } (i+1)) \text{ false})$

by (*simp add: Sup-power-false*)

also have $\dots = (\bigcap i. (P \triangleleft b \triangleright_R II_R) \text{ ^^ } i ;; (P ;; \text{Miracle} \triangleleft b \triangleright_R II_R))$

proof (*rule SUP-cong, simp*)

fix i

show $((\lambda X. P ;; \text{SRD } X \triangleleft b \triangleright_R II_R) \text{ ^^ } (i + 1)) \text{ false} = (P \triangleleft b \triangleright_R II_R) \text{ ^^ } i ;; (P ;; \text{Miracle} \triangleleft b \triangleright_R II_R)$

proof (*induct i*)

case 0

thm *if-eq-cancel*

then show *?case*

by (*simp, metis srdes-theory.healthy-top*)

next

case (*Suc i*)

show *?case*

proof –

have $((\lambda X. P ;; \text{SRD } X \triangleleft b \triangleright_R II_R) \text{ ^^ } (\text{Suc } i + 1)) \text{ false} =$

$P ;; \text{SRD } ((\lambda X. P ;; \text{SRD } X \triangleleft b \triangleright_R II_R) \text{ ^^ } (i + 1)) \text{ false} \triangleleft b \triangleright_R II_R$

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.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.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
finally show ?*thesis* .
qed
qed
qed
also have ... = $(\bigcap i \cdot (P \triangleleft b \triangleright_R II_R) \wedge i \;; (P \;; \text{Miracle} \triangleleft b \triangleright_R II_R))$
by (*simp add: UINF-as-Sup-collect'*)
finally show ?*thesis* .
qed

theorem *WhileR-star-expand*:

assumes *P is NSRD P is Productive*

shows $\text{while}_R b \text{ do } P \text{ od} = (P \triangleleft b \triangleright_R II_R)^{\star R} \;; (P \;; \text{Miracle} \triangleleft b \triangleright_R II_R)$ (**is** ?*lhs* = ?*rhs*)

proof –

have ?*lhs* = $(\bigcap i \cdot (P \triangleleft b \triangleright_R II_R) \wedge i) \;; (P \;; \text{Miracle} \triangleleft b \triangleright_R II_R)$

by (*simp add: WhileR-iter-expand seq-UINF-distr' assms*)

also have ... = $(P \triangleleft b \triangleright_R II_R)^{\star} \;; (P \;; \text{Miracle} \triangleleft b \triangleright_R II_R)$

by (*simp add: ustar-def*)

also have ... = $((P \triangleleft b \triangleright_R II_R)^{\star} \;; II_R) \;; (P \;; \text{Miracle} \triangleleft b \triangleright_R II_R)$

by (*simp add: seqr-assoc SRD-left-unit closure assms*)

also have ... = $(P \triangleleft b \triangleright_R II_R)^{\star R} \;; (P \;; \text{Miracle} \triangleleft b \triangleright_R II_R)$

by (*simp add: StarR-def nsrdes-theory.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: StarR-def WhileR-star-expand assms closure*)

theorem *WhileR-iter-form-lemma*:

assumes *P is NSRD*

shows $(P \triangleleft b \triangleright_R II_R)^{\star R} ;; (P ;; Miracle \triangleleft b \triangleright_R II_R) = ([b]^\top_R ;; P)^{\star R} ;; [\neg b]^\top_R$

proof –

have $(P \triangleleft b \triangleright_R II_R)^{\star R} ;; (P ;; Miracle \triangleleft b \triangleright_R II_R) = ([b]^\top_R ;; P) \sqcap [\neg b]^\top_R)^{\star R} ;; (P ;; Miracle \triangleleft b \triangleright_R II_R)$

by (*simp add: AssumeR-NSRD NSRD-right-unit NSRD-srd-skip assms(1) cond-srea-AssumeR-form*)

also have $\dots = ([b]^\top_R ;; P)^{\star R} ;; [\neg b]^\top_R)^{\star R} ;; (P ;; Miracle \triangleleft b \triangleright_R II_R)$

by (*simp add: StarR-def AssumeR-NSRD NSRD-seqr-closure nsrdes-theory.Star-denest assms(1)*)

also have $\dots = ([b]^\top_R ;; P)^{\star R} ;; (P ;; Miracle \triangleleft b \triangleright_R II_R)$

by (*metis (no-types, hide-lams) StarR-def RD3-def RD3-idem Star-AssumeR nsrdes-theory.Star-def*)

also have $\dots = ([b]^\top_R ;; P)^{\star R} ;; (P ;; Miracle \triangleleft b \triangleright_R II_R)$

by (*simp add: StarR-def AssumeR-NSRD NSRD-seqr-closure nsrdes-theory.Star-invol assms(1)*)

also have $\dots = ([b]^\top_R ;; P)^{\star R} ;; ([b]^\top_R ;; P ;; Miracle) \sqcap [\neg b]^\top_R$

by (*simp add: AssumeR-NSRD NSRD-Miracle NSRD-right-unit NSRD-seqr-closure NSRD-srd-skip assms(1) cond-srea-AssumeR-form*)

also have $\dots = ([b]^\top_R ;; P)^{\star R} ;; [b]^\top_R ;; P ;; Miracle) \sqcap ([b]^\top_R ;; P)^{\star R} ;; [\neg b]^\top_R$

by (*simp add: upred-semiring.distrib-left*)

also have $\dots = ([b]^\top_R ;; P)^{\star R} ;; [\neg b]^\top_R$

proof –

have $([b]^\top_R ;; P)^{\star R} ;; [\neg b]^\top_R = (II_R \sqcap ([b]^\top_R ;; P)^{\star R} ;; [b]^\top_R ;; P) ;; [\neg b]^\top_R$

by (*simp add: StarR-def AssumeR-NSRD NSRD-seqr-closure nsrdes-theory.Star-unfoldr-eq assms(1)*)

also have $\dots = [\neg b]^\top_R \sqcap ([b]^\top_R ;; P)^{\star R} ;; [b]^\top_R ;; P ;; [\neg b]^\top_R$

by (*metis (no-types, lifting) StarR-def AssumeR-NSRD AssumeR-as-gcmd NSRD-srd-skip Star-AssumeR nsrdes-theory.Star-slide gcmd-seq-distr skip-srea-self-unit urel-dioid.distrib-right*)

also have $\dots = [\neg b]^\top_R \sqcap ([b]^\top_R ;; P)^{\star R} ;; [b]^\top_R ;; P ;; [b \vee \neg b]^\top_R ;; [\neg b]^\top_R$

by (*simp add: AssumeR-true NSRD-right-unit assms(1)*)

also have $\dots = [\neg b]^\top_R \sqcap ([b]^\top_R ;; P)^{\star R} ;; [b]^\top_R ;; P ;; [b]^\top_R ;; [\neg b]^\top_R$

$\sqcap ([b]^\top_R ;; P)^{\star R} ;; [b]^\top_R ;; P ;; [\neg b]^\top_R ;; [\neg b]^\top_R$

by (*metis (no-types, hide-lams) AssumeR-choice upred-semiring.add-assoc upred-semiring.distrib-left upred-semiring.distrib-right*)

also have $\dots = [\neg b]^\top_R \sqcap ([b]^\top_R ;; P)^{\star R} ;; [b]^\top_R ;; P ;; ([b]^\top_R ;; [\neg b]^\top_R)$

$\sqcap ([b]^\top_R ;; P)^{\star R} ;; [b]^\top_R ;; P ;; ([\neg b]^\top_R ;; [\neg b]^\top_R)$

by (*simp add: RA1*)

also have $\dots = [\neg b]^\top_R \sqcap ([b]^\top_R ;; P)^{\star R} ;; [b]^\top_R ;; P ;; Miracle$

$\sqcap ([b]^\top_R ;; P)^{\star R} ;; [b]^\top_R ;; P ;; [\neg b]^\top_R$

by (*simp add: AssumeR-comp AssumeR-false*)

finally have $([b]^\top_R ;; P)^{\star R} ;; [\neg b]^\top_R \sqsubseteq ([b]^\top_R ;; P)^{\star R} ;; [b]^\top_R ;; P ;; Miracle$

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 $\text{while}_R b \text{ do } P \text{ od} = ([b]^\top_R ;; P)^{\star R} ;; [\neg b]^\top_R$

by (*simp add: WhileR-iter-form-lemma WhileR-star-expand assms*)

theorem WhileR-outer-refine-intro:

assumes

P is NSRD P is Productive

$S \sqsubseteq ([b]^\top_R ;; P) ;; S \sqsubseteq [\neg b]^\top_R$

shows $S \sqsubseteq \text{while}_R b \text{ do } P \text{ od}$

apply (*simp add: assms StarR-def WhileR-iter-form*)

apply (*rule nsrdes-theory.Star-inductl*)

apply (*simp-all add: closure assms*)
done

theorem *WhileR-outer-refine-init-intro:*

assumes

P is NSRD I is NSRD P is Productive

$S \sqsubseteq I \;; \; [\neg b]^\top_R$

$S \sqsubseteq S \;; \; [b]^\top_R \;; \; P$

$S \sqsubseteq I \;; \; [b]^\top_R \;; \; P$

shows $S \sqsubseteq I \;; \; \text{while}_R \; b \; \text{do} \; P \; \text{od}$

proof –

have $S \sqsubseteq I \;; \; ([b]^\top_R \;; \; P) \;; \; ([b]^\top_R \;; \; P)^{\star R} \;; \; [\neg b]^\top_R$

proof –

have $S \sqsubseteq I \;; \; ([b]^\top_R \;; \; P) \;; \; ([b]^\top_R \;; \; P)^{\star R}$

by (*metis (no-types, hide-lams) StarR-def AssumeR-NSRD NSRD-seqr-closure RA1 assms(1) assms(2) assms(5) assms(6) nsrdes-theory.Star-inductr semilattice-sup-class.le-sup-iff*)

thus *?thesis*

by (*metis (no-types, lifting) AssumeR-NSRD AssumeR-seq-refines StarR-def assms(1) dual-order.trans nsrdes-theory.Healthy-Sequence nsrdes-theory.utp-theory-kleene-axioms urel-dioid.mult-isol utp-theory-kleene.Star-Healthy*)

qed

moreover have $S \sqsubseteq I \;; \; II_R \;; \; [\neg b]^\top_R$

by (*simp add: AssumeR-NSRD assms nsrdes-theory.Unit-Left*)

ultimately show *?thesis*

apply (*simp add: assms WhileR-iter-form StarR-def*)

apply (*subst nsrdes-theory.Star-unfoldl-eq[THEN sym]*)

apply (*auto simp add: closure assms seqr-inf-distr*)

done

qed

theorem *WhileR-false:*

assumes *P is NSRD*

shows $\text{while}_R \; \text{false} \; \text{do} \; P \; \text{od} = II_R$

by (*simp add: WhileR-def rpred closure srdes-theory.LFP-const*)

theorem *WhileR-true:*

assumes *P is NSRD P is Productive*

shows $\text{while}_R \; \text{true} \; \text{do} \; P \; \text{od} = P^{\star R} \;; \; \text{Miracle}$

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 $\text{while}_R \; b \; \text{do} \; \mathbf{R}_s(P \vdash Q \diamond R) \; \text{od} =$

$\mathbf{R}_s \; ([b]^\top_r \;; \; R)^{\star r} \; \text{wp}_r \; ([b]_{S<} \Rightarrow_r P) \vdash ([b]^\top_r \;; \; R)^{\star r} \;; \; [b]^\top_r \;; \; Q) \diamond ([b]^\top_r \;; \; R)^{\star r} \;; \; [\neg b]^\top_r)$
(is ?lhs = ?rhs)

proof –

have $?lhs = ([b]^\top_R \;; \; \mathbf{R}_s(P \vdash Q \diamond R))^{\star R} \;; \; [\neg b]^\top_R$

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 is RC Q_2 is RR Q_3 is RR $\$st' \# Q_2$ Q_3 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 srdes-tri-refine-intro'*)
show $([b]^\top_r ;; Q_3)^{*r} \text{ wp}_r ([b]_{S<} \Rightarrow_r Q_1) \sqsubseteq P_1$
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-theory.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-theory.Star-inductl seqr-assoc*)
thus *?thesis*
by (*simp add: utp-pred-laws.le-infI2*)
qed
qed

11.12 Iteration Construction

definition *IterateR*

$:: 'a \text{ set} \Rightarrow ('a \Rightarrow 's \text{ upred}) \Rightarrow ('a \Rightarrow ('s, 't::\text{size-trace}, ' \alpha) \text{ hrel-rsp}) \Rightarrow ('s, 't, ' \alpha) \text{ hrel-rsp}$
where $\text{IterateR } A \ g \ P = \text{while}_R (\bigvee i \in A \cdot g(i)) \text{ do } (\text{if}_R i \in A \cdot g(i) \rightarrow P(i) \text{ fi}) \text{ od}$

definition *IterateR-list*

$:: ('s \text{ upred} \times ('s, 't::\text{size-trace}, ' \alpha) \text{ hrel-rsp}) \text{ list} \Rightarrow ('s, 't, ' \alpha) \text{ hrel-rsp}$ **where**
 $[\text{upred-defs}, \text{ndes-simp}]$:
 $\text{IterateR-list } xs = \text{IterateR } \{0..<\text{length } xs\} (\lambda i. \text{map fst } xs ! i) (\lambda i. \text{map snd } xs ! i)$

syntax

-iter-srd $:: \text{pttrn} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic} \ (do_R -\in- \cdot - \rightarrow - \text{ od})$
-iter-gcommR $:: \text{gcomms} \Rightarrow \text{logic} \ (do_R / - / \text{od})$

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
 $\bigwedge i. i \in I \implies P(i) \text{ is NSRD}$
 $\bigwedge i. i \in I \implies P(i) \text{ is Productive}$
shows $do_R i \in I \cdot g(i) \rightarrow P(i) \text{ od is NSRD}$
by (*simp add: IterateR-def closure assms*)

lemma *IterateR-empty*:
 $do_R i \in \{\} \cdot g(i) \rightarrow P(i) \text{ od} = II_R$
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 $do_R i \in \{k\} \cdot g(i) \rightarrow P(i) \text{ od} = \text{while}_R g(k) \text{ do } P(k) \text{ od}$ (*is ?lhs = ?rhs*)
proof –
have $?lhs = \text{while}_R g k \text{ do } P k \triangleleft g k \triangleright_R \text{Chaos} \text{ od}$
by (*simp add: IterateR-def AlternateR-singleton assms closure*)
also have $\dots = \text{while}_R g k \text{ do } [g k]^\top_R ;; (P k \triangleleft g k \triangleright_R \text{Chaos}) \text{ od}$
by (*simp add: WhileR-insert-assume closure assms*)
also have $\dots = \text{while}_R g k \text{ do } P k \text{ od}$
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*]

lemma *R4-Continuous* [*closure*]: *Continuous R4*
by (*rel-auto*)

lemma *cond-rea-R4-closed* [*closure*]:
 $\llbracket P \text{ is } R4; Q \text{ is } R4 \rrbracket \implies P \triangleleft b \triangleright_R Q \text{ is } R4$
by (*simp add: Healthy-def R4-cond*)

lemma *IterateR-outer-refine-intro*:
assumes $I \neq \{\} \bigwedge i. i \in I \implies P i \text{ is NSRD} \bigwedge i. i \in I \implies P i \text{ is Productive}$
 $\bigwedge i. i \in I \implies S \sqsubseteq (b i \rightarrow_R P i ;; S)$
 $S \sqsubseteq [\neg (\bigwedge i \in I \cdot b i)]^\top_R$
shows $S \sqsubseteq do_R i \in I \cdot b(i) \rightarrow P(i) \text{ od}$
apply (*simp add: IterateR-def*)
apply (*rule WhileR-outer-refine-intro*)
apply (*simp-all add: assms closure AlternateR-assume-branch seq-UINF-distr UINF-refines*)
done

lemma *IterateR-outer-refine-init-intro*:
assumes
 $A \neq \{\} \bigwedge i. i \in A \implies P i \text{ is NSRD}$
 $\bigwedge i. i \in A \implies P i \text{ is Productive}$
 $I \text{ is NSRD}$
 $S \sqsubseteq I ;; [\neg (\bigwedge i \in A \cdot b i)]^\top_R$
 $\bigwedge i. i \in A \implies S \sqsubseteq S ;; b i \rightarrow_R P i$
 $\bigwedge i. i \in A \implies S \sqsubseteq I ;; b i \rightarrow_R P i$
shows $S \sqsubseteq I ;; do_R i \in A \cdot b(i) \rightarrow P(i) \text{ od}$
apply (*simp add: IterateR-def*)
apply (*rule-tac WhileR-outer-refine-init-intro*)
apply (*simp-all add: assms closure AlternateR-assume-branch seq-UINF-distl UINF-refines*)

done

lemma *IterateR-lemma1*:

$(\bigsqcap i \in I \cdot b\ i]^\top_r \;; (\bigsqcap i \in I \cdot P\ i \triangleleft b\ i \triangleright_R \text{false}) = (\bigsqcap i \in I \cdot [b\ i]^\top_r \;; P\ i)$
 by (*rel-auto*; *fastforce*)

lemma *IterateR-lemma2*:

assumes $I \neq \{\}$ $\bigwedge i. i \in I \implies P(i)$ *is RR*
shows $(\bigsqcap i \in I \cdot b\ i]_{S<} \Rightarrow_r (\bigsqcap i \in I \cdot (P\ i) \triangleleft b\ i \triangleright_R R1\ \text{true}) \wedge \text{false} \triangleleft (\neg (\bigsqcap i \in I \cdot b\ i)) \triangleright_R R1\ \text{true})$
 $= (\bigsqcap i \in I \cdot (P\ i) \triangleleft b\ i \triangleright_R R1\ \text{true})$

proof –

from *assms*(1)

have $(\bigsqcap i \in I \cdot b\ i]_{S<} \Rightarrow_r (\bigsqcap i \in I \cdot RR(P\ i) \triangleleft b\ i \triangleright_R R1\ \text{true}) \wedge \text{false} \triangleleft (\neg (\bigsqcap i \in I \cdot b\ i)) \triangleright_R R1\ \text{true})$
 $= (\bigsqcap i \in I \cdot RR(P\ i) \triangleleft b\ i \triangleright_R R1\ \text{true})$

by (*rel-auto*)

thus *?thesis*

by (*simp add: assms Healthy-if cong: USUP-cong*)

qed

lemma *IterateR-lemma3*:

assumes $\bigwedge i. i \in I \implies P(i)$ *is RR*
shows $(\bigsqcap i \in I \cdot P\ i \triangleleft b\ i \triangleright_R R1\ \text{true}) = (\bigsqcap i \in I \cdot [b\ i]_{S<} \Rightarrow_r P\ i)$

proof –

have $(\bigsqcap i \in I \cdot RR(P\ i) \triangleleft b\ i \triangleright_R R1\ \text{true}) = (\bigsqcap i \in I \cdot [b\ i]_{S<} \Rightarrow_r RR(P\ i))$
by (*rel-auto*)

thus *?thesis*

by (*simp add: assms Healthy-if cong: USUP-cong*)

qed

theorem *IterateR-refine-intro*:

assumes

– Closure conditions

$\bigwedge i. i \in I \implies Q_1(i)$ *is RC* $\bigwedge i. i \in I \implies Q_2(i)$ *is RR* $\bigwedge i. i \in I \implies Q_3(i)$ *is RR*

$\bigwedge i. i \in I \implies \$st' \nmid Q_2(i)$ $\bigwedge i. i \in I \implies Q_3(i)$ *is R4* $I \neq \{\}$

$(\bigsqcap i \in I \cdot [b\ i]^\top_r \;; Q_3\ i)^{\star r} \text{wp}_r (\bigsqcap i \in I \cdot [b\ i]_{S<} \Rightarrow_r Q_1\ i) \sqsubseteq P_1$

$P_2 \sqsubseteq (\bigsqcap i \in I \cdot [b\ i]^\top_r \;; Q_2\ i)$

$P_2 \sqsubseteq (\bigsqcap i \in I \cdot [b\ i]^\top_r \;; Q_3\ i) \;; P_2$

$P_3 \sqsubseteq [\neg (\bigsqcap i \in I \cdot b\ i)]^\top_r$

$P_3 \sqsubseteq (\bigsqcap i \in I \cdot [b\ i]^\top_r \;; Q_3\ i) \;; P_3$

shows $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) \sqsubseteq \text{do}_R i \in I \cdot b(i) \rightarrow \mathbf{R}_s(Q_1(i) \vdash Q_2(i) \diamond Q_3(i))$ *od*

apply (*simp add: rdes-def closure assms unrest del: WhileR-rdes-def*)

apply (*rule WhileR-refine-intro*)

apply (*simp-all add: closure assms unrest IterateR-lemma1 IterateR-lemma2 segr-assoc[THEN sym]*)

apply (*simp add: IterateR-lemma3 closure assms unrest*)

done

method *unfold-iteration* = *simp add: IterateR-list-def IterateR-def AlternateR-list-def AlternateR-def UINF-upto-expand-first*

11.13 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*]:
 $\sigma \dagger_S \langle \varrho \rangle_R = \langle \varrho \circ_s \sigma \rangle_R$
by (*rdes-eq*)

11.14 Algebraic Laws

theorem *assigns-srd-id*: $\langle id_s \rangle_R = II_R$
by (*rdes-eq*)

theorem *assigns-srd-comp*: $\langle \sigma \rangle_R ;; \langle \varrho \rangle_R = \langle \varrho \circ_s \sigma \rangle_R$
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* : $\langle \sigma \rangle_R \triangleleft b \triangleright_R \langle \varrho \rangle_R = \langle \sigma \triangleleft b \triangleright \varrho \rangle_R$
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 $(if_R\ i \in I \cdot g\ i \rightarrow A\ i \text{ else } B\ fi) ;; C = (if_R\ i \in I \cdot g\ i \rightarrow A\ i ;; C \text{ else } B ;; C\ fi)$
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 $(if_R\ i \in \{a\} \cdot g \rightarrow A \text{ else } B\ fi) = (A \triangleleft g \triangleright_R B)$
by (*rdes-eq cls: assms*)

lemma *AlternateR-Chaos*:
 $if_R\ i \in A \cdot g(i) \rightarrow \text{Chaos}\ fi = \text{Chaos}$
by (*cases* $A = \{\}$, *simp*, *rdes-eq*)

lemma *choose-srd-par*:
 $choose_R \parallel_R choose_R = choose_R$
by (*rdes-eq*)

11.15 Lifting designs to reactive designs

definition *des-rea-lift* :: $'s \text{ hrel-des} \Rightarrow ('s, 't::\text{trace}, 'α) \text{ hrel-rsp } (\mathbf{R}_D)$ **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* :: $('s, 't::\text{trace}, 'α) \text{ hrel-rsp} \Rightarrow 's \text{ hrel-des } (\mathbf{D}_R)$ **where**
 $[upred-defs]: \mathbf{D}_R(P) = \lfloor (pre_R(P)) \llbracket \$tr / \$tr' \rrbracket \vdash_v \$st \rfloor_{S<} \vdash_n \lfloor (post_R(P)) \llbracket \$tr / \$tr' \rrbracket \vdash_v \{ \$st, \$st' \} \rfloor_S$

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 $P \text{ is } \mathbf{N} \ Q \text{ is } \mathbf{N} \ \mathbf{R}_D \ P = \mathbf{R}_D \ Q$ (**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*]:
 $peri_R(\mathbf{R}_D(P)) = (false \triangleleft \lceil pre_D(P) \rceil_S \triangleright (\$tr \leq_u \$tr'))$
by (*rel-auto*)

lemma *postR-des-rea-lift* [*rdes*]:
 $post_R(\mathbf{R}_D(P)) = ((true \triangleleft \lceil pre_D(P) \rceil_S \triangleright (\neg \$tr \leq_u \$tr')) \Rightarrow (\$tr' =_u \$tr \wedge \lceil post_D(P) \rceil_S))$
apply (*rel-auto*) **using** *minus-zero-eq* **by** *blast*

lemma *ndes-rea-lift-closure* [*closure*]:
assumes $P \text{ is } \mathbf{N}$
shows $\mathbf{R}_D(P)$ *is NSRD*

proof –
obtain $p \ Q$ **where** $P = (p \vdash_n Q)$
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 (meson aext-mono assms(3) design-refine-thms(1) refBy-order)
 apply (rel-auto)
 apply (smt aext-and aext-mono assms(1) assms(2) assms(3) rdesign-ref-monos(2) utp-pred-laws.inf.cobounded2
 utp-pred-laws.inf.coboundedI2 utp-pred-laws.inf-left-commute utp-pred-laws.le-inf-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 \mathbf{N} Q is \mathbf{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*:
 $\mathbf{R}_D(\Pi_D) = (\Pi_R :: ('s, 't::trace, unit) \text{ hrel-rsp})$
apply (*rel-auto*) **using** *minus-zero-eq* **by** *blast+*

lemma *R-D-assigns*:
 $\mathbf{R}_D(\langle \sigma \rangle_D) = (\langle \sigma \rangle_R :: ('s, 't::trace, unit) \text{ hrel-rsp})$
by (*simp add: assigns-d-def des-rea-lift-def alpha assigns-srd-RHS-tri-des, rel-auto*)

11.16 State Invariants

definition *StateInvR* :: $'s \text{ upred} \Rightarrow ('s, 't::trace, 'a) \text{ hrel-rsp} \text{ (} \text{sinv}_R'(-) \text{)}$ **where**
 $[rdes-def]: \text{sinv}_R(b) = \mathbf{R}_s([b]_{S<} \vdash \text{true}_r \diamond [b]_{S>})$

lemma *StateInvR-NSRD* [*closure*]: *sinv_R(b) is NSRD*
by (*simp add: StateInvR-def closure unrest*)

lemma *StateInvR-srd-skip-refine*: $\text{sinv}_R(b) \sqsubseteq \Pi_R$
by (*rdes-refine*)

lemma *StateInvR-seq-idem*:
 $\text{sinv}_R(b) ;; \text{sinv}_R(b) = \text{sinv}_R(b)$
by (*rdes-eq*)

lemma *StateInvR-seq-refine*:
assumes $\text{sinv}_R(b) \sqsubseteq P$ $\text{sinv}_R(b) \sqsubseteq Q$
shows $\text{sinv}_R(b) \sqsubseteq P ;; Q$
by (*metis (full-types) StateInvR-seq-idem assms seqr-mono*)

lemma *ndiv-StateInvR*: $\text{ndiv}_R = \text{sinv}_R(\text{true})$
by (*rdes-eq*)

end

12 Instantaneous Reactive Designs

theory *utp-rdes-instant*
imports *utp-rdes-prog*
begin

definition *ISRDI* :: $('s, 't::trace, 'a) \text{ hrel-rsp} \Rightarrow ('s, 't, 'a) \text{ hrel-rsp}$ **where**
 $[upred-defs]: \text{ISRDI}(P) = P \parallel_R \mathbf{R}_s(\text{true}_r \vdash \text{false} \diamond (\$tr' =_u \$tr))$

definition *ISRDI* :: $('s, 't::trace, 'a) \text{ hrel-rsp} \Rightarrow ('s, 't, 'a) \text{ hrel-rsp}$ **where**
 $[upred-defs]: \text{ISRDI} = \text{ISRDI} \circ \text{NSRD}$

lemma *ISRDI-idem*: $\text{ISRDI}(\text{ISRDI}(P)) = \text{ISRDI}(P)$
by (*rel-auto*)

lemma *ISRD1-monotonic*: $P \sqsubseteq Q \implies \text{ISRD1}(P) \sqsubseteq \text{ISRD1}(Q)$
by (*rel-auto*)

lemma *ISRD1-RHS-design-form*:

assumes $\$ok' \# P \$ok' \# Q \$ok' \# R$
shows $\text{ISRD1}(\mathbf{R}_s(P \vdash Q \diamond R)) = \mathbf{R}_s(P \vdash \text{false} \diamond (R \wedge \$tr' =_u \$tr))$
using *assms* **by** (*simp add: ISRD1-def choose-srd-def RHS-tri-design-par unrest, rel-auto*)

lemma *ISRD1-form*:

$\text{ISRD1}(\text{SRD}(P)) = \mathbf{R}_s(\text{pre}_R(P) \vdash \text{false} \diamond (\text{post}_R(P) \wedge \$tr' =_u \$tr))$
by (*simp add: ISRD1-RHS-design-form SRD-as-reactive-tri-design unrest*)

lemma *ISRD1-rdes-def* [*rdes-def*]:

$\llbracket P \text{ is } RR; R \text{ is } RR \rrbracket \implies \text{ISRD1}(\mathbf{R}_s(P \vdash Q \diamond R)) = \mathbf{R}_s(P \vdash \text{false} \diamond (R \wedge \$tr' =_u \$tr))$
by (*simp add: ISRD1-def rdes-def closure rpred*)

lemma *ISRD-intro*:

assumes $P \text{ is } \text{NSRD}$ $\text{peri}_R(P) = (\neg_r \text{pre}_R(P)) (\$tr' =_u \$tr) \sqsubseteq \text{post}_R(P)$
shows $P \text{ is } \text{ISRD}$

proof –

have $\mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \diamond \text{post}_R(P)) \text{ is } \text{ISRD1}$
apply (*simp add: Healthy-def rdes-def closure assms(1-2)*)
using *assms(3)* **least-zero** **apply** (*rel-blast*)
done
hence $P \text{ is } \text{ISRD1}$
by (*simp add: SRD-reactive-tri-design closure assms(1)*)
thus *?thesis*
by (*simp add: ISRD-def Healthy-comp assms(1)*)

qed

lemma *ISRD1-rdes-intro*:

assumes $P \text{ is } RR$ $Q \text{ is } RR$ $(\$tr' =_u \$tr) \sqsubseteq Q$
shows $\mathbf{R}_s(P \vdash \text{false} \diamond Q) \text{ is } \text{ISRD1}$
unfolding *Healthy-def*
by (*simp add: ISRD1-rdes-def assms closure unrest utp-pred-laws.inf.absorb1*)

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

assumes $P \text{ is } RC$ $Q \text{ is } RR$ $(\$tr' =_u \$tr) \sqsubseteq Q$
shows $\mathbf{R}_s(P \vdash \text{false} \diamond Q) \text{ is } \text{ISRD}$
unfolding *Healthy-def*
by (*simp add: ISRD-def closure Healthy-if ISRD1-rdes-def assms unrest utp-pred-laws.inf.absorb1*)

lemma *ISRD-implies-ISRD1*:

assumes $P \text{ is } \text{ISRD}$
shows $P \text{ is } \text{ISRD1}$

proof –

have $\text{ISRD}(P) \text{ is } \text{ISRD1}$
by (*simp add: ISRD-def Healthy-def ISRD1-idem*)
thus *?thesis*
by (*simp add: assms Healthy-if*)

qed

lemma *ISRD-implies-SRD*:

assumes $P \text{ is } \text{ISRD}$
shows $P \text{ is } \text{SRD}$

proof –

have $1: \text{ISRD}(P) = \mathbf{R}_s((\neg_r (\neg_r \text{pre}_R P) ;; R1 \text{ true} \wedge R1 \text{ true}) \vdash \text{false} \diamond (\text{post}_R P \wedge \$tr' =_u \$tr))$
 by (simp add: NSRD-form ISRD1-def ISRD-def RHS-tri-design-par rdes-def unrest closure)
 moreover have ... is SRD
 by (simp add: closure unrest)
 ultimately have $\text{ISRD}(P)$ is SRD
 by (simp)
 with *assms* show ?thesis
 by (simp add: Healthy-def)

qed

lemma *ISRD-implies-NSRD* [closure]:

assumes P is ISRD
 shows P is NSRD

proof –

have $1: \text{ISRD}(P) = \text{ISRD1}(\text{RD3}(\text{SRD}(P)))$
 by (simp add: ISRD-def NSRD-def SRD-def,metis RD1-RD3-commute RD3-left-subsumes-RD2)
 also have ... = $\text{ISRD1}(\text{RD3}(P))$
 by (simp add: assms ISRD-implies-SRD Healthy-if)
 also have ... = $\text{ISRD1}(\mathbf{R}_s((\neg_r \text{pre}_R P) \text{wp}_r \text{false}_h \vdash (\exists \$st' \cdot \text{peri}_R P) \diamond \text{post}_R P))$
 by (simp add: RD3-def, subst SRD-right-unit-tri-lemma, simp-all add: assms ISRD-implies-SRD)
 also have ... = $\mathbf{R}_s((\neg_r \text{pre}_R P) \text{wp}_r \text{false}_h \vdash \text{false} \diamond (\text{post}_R P \wedge \$tr' =_u \$tr))$
 by (simp add: RHS-tri-design-par ISRD1-def unrest choose-srd-def rpred closure ISRD-implies-SRD assms)
 also have ... = (... ;; II_R)
 by (rdes-simp, simp add: RHS-tri-normal-design-composition' closure assms unrest ISRD-implies-SRD wp rpred wp-rea-false-RC)
 also have ... is RD3
 by (simp add: Healthy-def RD3-def segr-assoc)
 finally show ?thesis
 by (simp add: SRD-RD3-implies-NSRD Healthy-if assms ISRD-implies-SRD)

qed

lemma *ISRD-form*:

assumes P is ISRD
 shows $\mathbf{R}_s(\text{pre}_R(P) \vdash \text{false} \diamond (\text{post}_R(P) \wedge \$tr' =_u \$tr)) = P$

proof –

have $P = \text{ISRD1}(P)$
 by (simp add: ISRD-implies-ISRD1 assms Healthy-if)
 also have ... = $\text{ISRD1}(\mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \diamond \text{post}_R(P)))$
 by (simp add: SRD-reactive-tri-design ISRD-implies-SRD assms)
 also have ... = $\mathbf{R}_s(\text{pre}_R(P) \vdash \text{false} \diamond (\text{post}_R(P) \wedge \$tr' =_u \$tr))$
 by (simp add: ISRD1-rdes-def closure assms)
 finally show ?thesis ..

qed

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

$\llbracket P \text{ is ISRD}; Q(\mathbf{R}_s(\text{pre}_R(P) \vdash \text{false} \diamond (\text{post}_R(P) \wedge \$tr' =_u \$tr))) \rrbracket \implies Q(P)$
 by (simp add: ISRD-form)

lemma *skip-srd-ISRD* [closure]: II_R is ISRD

by (rule ISRD-intro, simp-all add: rdes closure)

lemma *assigns-srd-ISRD* [closure]: $\langle \sigma \rangle_R$ is ISRD

by (rule ISRD-intro, simp-all add: rdes closure, rel-auto)

```

lemma seq-ISRD-closed:
  assumes  $P$  is ISRD  $Q$  is ISRD
  shows  $P \;; Q$  is ISRD
  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 ISRDMiracle-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.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  $P \;; \text{SRD } \text{Chaos} \sqsubseteq \text{Chaos}$ 
    by (simp add: Healthy-if srdes-theory.bottom-closed)
  qed
thus ?thesis
  by (metis 1 dual-order.antisym srdes-theory.LFP-closed srdes-theory.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] 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>.
- [2] S. Foster, F. Zeyda, and J. Woodcock. Unifying heterogeneous state-spaces with lenses. In *ICTAC*, LNCS 9965. Springer, 2016.
- [3] 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.