Abstract

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

Contents

1 Introduction 3

2 Reactive Designs Healthiness Conditions 3
  2.1 Preliminaries ................................................................. 3
  2.2 Identities ................................................................. 3
  2.3 RD1: Divergence yields arbitrary traces ................................. 4
  2.4 R3c and R3h: Reactive design versions of R3 .......................... 6
  2.5 RD2: A reactive specification cannot require non-termination .......... 9
  2.6 Major healthiness conditions ................................................ 10
  2.7 UTP theories ............................................................. 13

3 Reactive Design Specifications 15
  3.1 Reactive design forms ...................................................... 15
  3.2 Auxiliary healthiness conditions .......................................... 17
  3.3 Composition laws .......................................................... 18
  3.4 Refinement introduction laws ............................................. 25
  3.5 Distribution laws .......................................................... 26

4 Reactive Design Triples 26
  4.1 Diamond notation ........................................................... 26
  4.2 Export laws ................................................................. 27
  4.3 Pre-, peri-, and postconditions .......................................... 28
    4.3.1 Definitions ............................................................ 28
    4.3.2 Unrestriction laws ................................................... 28
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

lemma R2-st-ex: R2 (∃ $st · P) = (∃ $st · 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([II]D) = [II]D
by (rel-auto)

lemma R2-des-lift-skip:
R2([II]D) = [II]D
apply (rel-auto) using minus-zero-eq by blast

lemma R1-R2c-ex-st: R1 (R2c (∃ $st' · Q1)) = (∃ $st' · 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-rea :: ('t::trace, 'a) hrel-rp (IIc) where
skip-rea-def [urel-defs]: IIc = (II ∨ (¬ ok ∧ $tr ≤_u $tr'))

definition skip-srea :: ('s, 't::trace, 'a) hrel-rsp (II_R) where
skip-srea-def [urel-defs]: \( II_R = (\exists \, \text{st} \cdot II_c) \triangleq \text{wait} \triangleright II_c \)

lemma skip-rea-R1-lemma: \( II_c = R1(\$ok \Rightarrow II) \)
by (rel-auto)

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

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

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

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

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

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

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

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

lemma skip-srea-R2 [closure]: \( II_R \) is \( R2 \)
by (metis Healthy-def′ R1-R2c-is-R2 R2c-skip-srea skip-srea-R1)

2.3 RD1: Divergence yields arbitrary traces

definition RD1 :: \( ('t::\text{trace},'\alpha,'\beta) \rel-rp \Rightarrow ('t,'\alpha,'\beta) \rel-rp \) where
[upred-defs]: \( RD1(P) = (P \lor (\neg \$ok \land \$tr \leq \$tr')) \)

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

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

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

lemma RD1-mono: \( P \sqsubseteq Q \Rightarrow RD1(P) \sqsubseteq RD1(Q) \)
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]: \( R1(\text{true}) \) is RD1
lemma RD1-wait-\text{false} [\text{closure}]: P \ is \ RD1 \implies P[false/\$wait\] \ is \ RD1

by (rel-auto)

lemma RD1-wait'-\text{false} [\text{closure}]: P \ is \ RD1 \implies P[false/\$\text{wait'}] \ is \ RD1

by (rel-auto)

lemma RD1-seq: RD1(RD1(P) ;; RD1(Q)) = RD1(P) ;; RD1(Q)

by (rel-auto)

lemma RD1-seq-closure [\text{closure}]: [P \ is \ RD1; Q \ is \ RD1] \implies P ;; Q \ 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) \land \$ok \Rightarrow R1(true))

by (rel-auto)

lemma skip-rea-RD1-skip: II_c = RD1(II)

by (rel-auto)

lemma skip-srea-RD1 [\text{closure}]: II_R \ is \ RD1

by (rel-auto)

lemma RD1-algebraic-intro:

assumes
\[ P \ is \ R1 \ (R1(true_h) ;; P) = R1(true_h) \ (II_c ;; P) = P \]

shows \( P \ is \ RD1 \)

proof
have \( P = (II_c ;; P) \)
  by (simp add: assms(3))
also have \( ... = (R1(\$ok \Rightarrow II) ;; P) \)
  by (simp add: skip-rea-R1-lemma)
also have \( ... = ((\neg \$ok \land R1(true)) ;; P) \lor P) \)
  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 \( ... = ((R1(\neg \$ok) ;; (R1(true_h) ;; P)) \lor P) \)
  using dual-order.trans by (rel-blast)
also have \( ... = ((R1(\neg \$ok) ;; R1(true_h)) \lor P) \)
  by (simp add: assms(2))
also have \( ... = (R1(\neg \$ok) \lor P) \)
  by (rel-auto)
also have \( ... = RD1(P) \)
  by (rel-auto)
finally show \( \lnot \text{thesis} \)
  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 ⊲ ok ⊢ R1(true))
proof
  have RD1(R1(P)) = (R1(P) ⊲ ok ⊢ R1(true))
    by (rel-auto)
  thus \?thesis
    by (simp add: Healthy-if assms)
qed

theorem RD1-algebraic:
assumes P is R1
shows P is RD1 ←→ \((R1(true_h) ;; P) = R1(true_h) ∧ (II_c ;; P) = P\)
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 :: \( \langle \text{t::trace}, \alpha \rangle \text{ hrel-rp} \Rightarrow \langle \text{t}, \alpha \rangle \text{ hrel-rp} \text{ where} \)
\[ \begin{array}{l}
\text{[upred-defs]:} \ R3c(P) = (II_c ◁ \$wait ▷ P) \\
\end{array} \]
definition R3h :: \( \langle \text{s::trace}, \alpha \rangle \text{ hrel-rsp} \Rightarrow \langle \text{s}, \alpha \rangle \text{ hrel-rsp} \text{ where} \)
\[ \begin{array}{l}
\text{[upred-defs]:} \ R3h(P) = ((∃ \text{st} · II_c ◁ \$wait ▷ P) \\
\end{array} \]

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 ⊆ Q ⇒ R3c(P) ⊆ 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 ⊑ Q ⇒ R3h(P) ⊑ 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 ∩ Q) = R3h(P) ∩ R3h(Q)
  by (rel-auto)

lemma R3h-UINF:
  A ≠ {} ⇒ R3h(⨆ i ∈ A · P(i)) = (⨆ i ∈ A · R3h(P(i)))
  by (rel-auto)

lemma R3h-cond: R3h(P ⊲ b ⊳ Q) = (R3h(P) ⊲ b ⊳ 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 ⇒ 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-cancel-R3c: \( R3(R3c(P)) = R3(P) \)
by (rel-auto)

lemma R3h-cancel-R3c: \( R3h(R3c(P)) = R3h(P) \)
by (rel-auto)

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

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

lemma R3c-seq-closure:
assumes \( P \) is R3c \( Q \) is R3c \( Q \) is R1
shows \( (P ;; \ Q) \) is R3c
by (metis Healthy-def', R3c-semi-form assms)

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

lemma R3-h-left-seq-closure:
assumes \( P \) is R3 \( Q \) is R3c
shows \( (P ;; \ Q) \) is R3c
proof
have \( (P ;; \ Q) = ((P ;; \ Q) \ [\text{true} / \text{wait}] < \text{wait} \triangleright (P ;; \ Q)) \)
by (metis cond-var-split cond-var-subst-right in-var-uvar wait-vwb-lens)
also have \( \ldots = (((II \ < \text{wait} \triangleright P) ;; \ Q) \ [\text{true} / \text{wait}] < \text{wait} \triangleright (P ;; \ Q)) \)
by (metis Healthy-def' R3-def assms(1))
also have \( \ldots = (((II[\text{true} / \text{wait}] ;; \ Q) < \text{wait} \triangleright (P ;; \ Q)) \)
by (subst-tac)
also have \( \ldots = (((\text{II \wedge \text{wait}'} ;; \ Q) < \text{wait} \triangleright (P ;; \ Q)) \)
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 \( \ldots = (((II[\text{true} / \text{wait}'] ;; \ Q[\text{true} / \text{wait}]]) < \text{wait} \triangleright (P ;; \ Q)) \)
by (metis seqr-pre-transfer seqr-right-one-point true-alt-def uovar-cover upred-eg-true utp-rel.unrest-uovar
vwb-lens-mub wait-vwb-lens)
also have \( \ldots = (((II[\text{true} / \text{wait}]' ;; \ II \wedge \text{wait} \triangleright Q)[\text{true} / \text{wait}]]) < \text{wait} \triangleright (P ;; \ Q)) \)
by (metis Healthy-def' R3c-def assms(2))
also have \( \ldots = (((II[\text{true} / \text{wait}]' ;; \ IIc[\text{true} / \text{wait}]) < \text{wait} \triangleright (P ;; \ Q)) \)
by (subst-tac)
also have \( \ldots = (((II \wedge \text{wait}' ;; \ IIc[\text{true} / \text{wait}]) < \text{wait} \triangleright (P ;; \ Q)) \)
by (metis seqr-pre-transfer seqr-right-one-point true-alt-def uovar-cover upred-eg-true utp-rel.unrest-uovar
vwb-lens-mub wait-vwb-lens)
also have \( \ldots = (((II ;; \ IIc) < \text{wait} \triangleright (P ;; \ Q)) \)
by (simp add: cond-def seqr-pre-transfer utp-rel.unrest-uovar)
also have \( \ldots = (IIc < \text{wait} \triangleright (P ;; \ Q)) \)
by simp
also have \( \ldots = R3c(P ;; \ Q) \)
by (simp add: R3c-def)
finally show \( \text{thesis} \)
by (simp add: Healthy-def')
lemma $R3c\text{-cases}$: $R3c(P) = ((II \triangleleft \$ok \triangleright R1(\text{true})) \triangleleft \$wait \triangleright P)$
by (rel-auto)

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

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

lemma $R3c\text{-subst-wait}$: $R3c(P) = R3c(P_f)$
by (simp add: $R3c\text{-def}$ cond-var-subst-right)

lemma $R3h\text{-subst-wait}$: $R3h(P) = R3h(P_f)$
by (simp add: $R3h\text{-cases}$ cond-var-subst-right)

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

lemma $R3h\text{-wait-true}$:
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 $R3h\text{-form}$ assms)
also have ... = $II_R t$
by (simp add: usubst)
finally show ?thesis.
qed

2.5 RD2: A reactive specification cannot require non-termination

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

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

lemma $RD2\text{-idem}$: $RD2(RD2(P)) = RD2(P)$
by (simp add: $H2\text{-idem}$ $RD2\text{-def}$)

lemma $RD2\text{-Idempotent}$: Idempotent $RD2$
by (simp add: Idempotent-def $RD2\text{-idem}$)

lemma $RD2\text{-mono}$: $P \triangleleft Q \implies RD2(P) \triangleleft RD2(Q)$
by (simp add: $H2\text{-def}$ $RD2\text{-def}$ seqr-mono)

lemma $RD2\text{-Monotonic}$: Monotonic $RD2$
using mono-def $RD2\text{-mono}$ by blast

lemma $RD2\text{-Continuous}$: Continuous $RD2$
by (rel-auto)

lemma $RD1\text{-RD2}\text{-commute}$: $RD1(RD2(P)) = RD2(RD1(P))$
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 \( RH \) :: \((t::trace,\alpha)\) hrel-rp \( \Rightarrow \) \((t,\alpha)\) hrel-rp (\((R)\))

where [upred-defs]: \( RH(P) = R1(R2c(R3c(P))) \)

definition \( RHS \) :: \((s,t::trace,\alpha)\) hrel-rsp \( \Rightarrow \) \((s,t,\alpha)\) hrel-rsp (\((R_s)\))

where [upred-defs]: \( RHS(P) = R1(R2c(R3h(P))) \)

definition \( RD \) :: \((t::trace,\alpha)\) hrel-rp \( \Rightarrow \) \((t,\alpha)\) hrel-rp

where [upred-defs]: \( RD(P) = RD1(RD2(RP(P))) \)

definition \( SRD \) :: \((s,t::trace,\alpha)\) hrel-rsp \( \Rightarrow \) \((s,t,\alpha)\) 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: \( R(R(P)) = R(P) \)

by (simp add: Idempotent-def RH-idem)

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: \( R_s(R_s(P)) = R_s(P) \)

by (simp add: R1-R2c-is-R2 R1-R3h-commute R2-idem R2c-R3c-commute R2c-idem R3c-idem RH-def)

lemma \( RHS\)-Idempotent [closure]: Idempotent \( R_s \)

by (simp add: Idempotent-def RHS-idem)

lemma \( RHS\)-Monotonic: Monotonic \( R_s \)

by (simp add: mono-def R1-R2c-is-R2 R2-mono R3h-mono RHS-def)

lemma \( RHS\)-mono: \( P \subseteq Q \Rightarrow R_s(P) \subseteq R_s(Q) \)
using mono-def RHS-Monotonic by blast

lemma RHS-Continuous [closure]: Continuous Rs
  by (simp add: Continuous-comp R1-Continuous R2c-Continuous R3h-Continuous RHS-comp)

lemma RHS-inf: Rs(P ∩ Q) = Rs(P) ∩ Rs(Q)
  using Continuous-Disjunctuous Disjunctuous-def RHS-Continuous
  by auto

lemma RHS-INF:
  A ≠ {} ⇒ Rs(⨅ i ∈ A · P(i)) = (∅ i ∈ A · Rs(P(i))
  by (simp add: RHS-def R3h-UINF R2c-USUP R1-USUP)

lemma RHS-sup: Rs(P ⊔ Q) = Rs(P) ⊔ Rs(Q)
  by (rel-auto)

lemma RHS-SUP:
  A ≠ {} ⇒ Rs(⨆ i ∈ A · P(i)) = (∅ i ∈ A · Rs(P(i))
  by (rel-auto)

lemma RHS-cond: Rs(P ⊳ b ⊲ Q) = (Rs(P) ⊳ R2c b ⊲ Rs(Q))
  by (simp add: RHS-def R3h-cond R2c-condr R1-cond)

lemma RD-alt-def:
  RD(P) = RD1(RD2(R(P)))
  by (simp add: R3c-via-RD1-R3 RD1-R2c-commute RD1-R3c-commute RD1-RD2-commute RH-def RD-def RP-def)

lemma RD1-RH-commute: RD1(R(P)) = R(RD1(P))
  by (simp add: RD1-R1-commute RD1-R2c-commute RD1-R3c-commute RH-def)

lemma RD2-RH-commute: RD2(R(P)) = 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 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 RD-RHS-commute: RD1(Rs(P)) = Rs(RD1(P))
  by (simp add: RD1-R1-commute RD1-R2c-commute RD1-R3h-commute RHS-def)

lemma RD2-RHS-commute: RD2(Rs(P)) = Rs(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\(P\) = \(R_s(H(P))\)
  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-R2c-is-R2 RHS-def SRD-def assms)
  apply (metis Healthy-def RD1-RHS-commute RD2-RHS-commute 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)

lemma SRD-ok-false [usubst]:
  assumes P is SRD
  shows \(P\) is \(\{_\}\) \(\{false\}\) is \(\{ok\}\) \(\{false\}\)
  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)

lemma SRD-ok-true-wait-true [usubst]:
  assumes P is SRD
  shows \(\exists st \cdot II\) \(\{true\}, \{true\}\) \(\{ok\}, \{wait\}\)
  proof
    have \(\exists st \cdot II\) \(\{true\} \{true\}\) \(\{false\} \{ok\}\) \(\{false\}\) \(\{ok\}\) \(\{false\}\)
      by (metis Healthy-def RD1-R3h-commute RD1-R3h-commute RD1-R3h-commute RD2-R3h-commute RHS-def SRD-def assms)
    moreover have \(\exists st \cdot II\) \(\{true\} \{true\}\) \(\{false\} \{ok\}\) \(\{true\}\) \(\{false\}\)
      by (simp add: usubst)
    ultimately show \?thesis
    by (simp)
  qed

lemma SRD-left-zero-1: P is SRD \(\Rightarrow\) 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 \(\exists st \cdot II\) \(\{true\} \{true\}\) \(\{false\} \{ok\}\) \(\{false\}\)
  proof
    have \(\exists st \cdot II\) \(\{true\} \{true\}\) \(\{false\} \{ok\}\) \(\{false\}\)
      by (simp add: RD1-left-zero SRD-healths(1) SRD-healths(4))
2.7 UTP theories

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

typedecl RDES

typedecl SRDES

abbreviation RDES ≡ UTHY(RDES, ('t::trace,'a) rp)
abbreviation SRDES ≡ UTHY(SRDES, ('s,'t::trace,'a) rsp)

overloading
rdes-hcond ≡ utp-hcond :: (RDES, ('t::trace,'a) rp) uthy ⇒ (('t,'a) rp × ('t,'a) rp) health sf
srdes-hcond ≡ utp-hcond :: (SRDES, ('s,'t::trace,'a) rsp) uthy ⇒ (('s,'t,'a) rsp × ('s,'t,'a) rsp) health sf

begin

definition rdes-hcond :: (RDES, ('t::trace,'a) rp) uthy ⇒ (('t,'a) rp × ('t,'a) rp) health where
[upred-defs]: rdes-hcond T = RD

definition srdes-hcond :: (SRDES, ('s,'t::trace,'a) rsp) uthy ⇒ (('s,'t,'a) rsp × ('s,'t,'a) rsp) health where
[upred-defs]: srdes-hcond T = SRD

end

interpretation rdes-theory: utp-theory UTHY(RDES, ('t::trace,'a) rp)
by (unfold-locales, simp-all add: rdes-hcond-def RD-idem)

interpretation rdes-theory-continuous: utp-theory-continuous UTHY(RDES, ('t::trace,'a) rp)
rewrites /
\ P. P ∈ carrier (uthy-order RDES) ⟷ P is RD
and carrier (uthy-order RDES) → carrier (uthy-order RDES) ≡ [RD]_H → [RD]_H
and le (uthy-order RDES) = op ⊆
and eq (uthy-order RDES) = op =
by (unfold-locales, simp-all add: rdes-hcond-def RD-Continuous)

interpretation rdes-rea-galois:
galois-connection (RDES ≡(RD1 ∘ RD2,R3) → REA)
proof (simp add: mk-conn-def, rule galois-connectionI', simp-all add: utp-partial-order rdes-hcond-def rea-hcond-def)

show R3 ∈ [RD]_H → [RP]_H
by (metis (no-types, lifting) Healthy-def Pi-I R3-RD-RP RP-idem mem-Collect-eq)

show RD1 ∘ RD2 ∈ [RP]_H → [RD]_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 ∘ RD2)
by (simp add: Monotonic-comp RD1-Monotonic RD2-Monotonic isotone-utp-orderI)

fix P :: ('a, 'b) hrel-rp
assume P is RD
thus P ⊆ RD1 (RD2 (R3 P))
by (metis Healthy-if R3-RD-RP RD-def RP-idem eq-iff)

next
fix P :: ('a, 'b) hrel-rp
assume a: P is RP
thus $R_3 (RD_1 (RD_2 P)) \subseteq P$

proof

have $R_3 (RD_1 (RD_2 P)) = RP (RD_1 (RD_2 P))$
  by (metis Healthy-if R3-RD-RP RD-def a)
moreover have $RD_1(RD_2 P) \subseteq P$
  by (rel-auto)
ultimately show ?thesis
  by (metis Healthy-if RP-mono a)
qed

qed

interpretation rdes-rea-retract:
retract $(RDES \leftarrow (RD_1 \circ RD_2, R_3) \rightarrow REA)$
by (unfold-locales, simp-all add: mk-conn-def utp-partial-order rdes-hcond-def rea-hcond-def)
  (metis Healthy-if R3-RD-RP RD-def RP-idem eq-refl)

interpretation srdes-theory: utp-theory UTHY (SRDES, ('s,'t::trace,'a) rsp)
by (unfold-locales, simp-all add: srdes-hcond-def SRD-idem)

interpretation srdes-theory-continuous: utp-theory-continuous UTHY (SRDES, ('s,'t::trace,'a) rsp)
rewrites $\bigwedge P. P \in \text{carrier (uthy-order SRDES)} \iff P \text{ is SRD}$
and $P$ is $\mathbb{H}_{SRDES} \iff P \text{ is SRD}$
and $(\mu X \cdot F (\mathbb{H}_{SRDES} X)) = (\mu X \cdot F (SRD X))$
and carrier (uthy-order SRDES) $\rightarrow$ carrier (uthy-order SRDES) $\equiv [SRD]_H \rightarrow [SRD]_H$
and $[\mathbb{H}_{SRDES}]_H \rightarrow [\mathbb{H}_{SRDES}]_H \equiv [SRD]_H \rightarrow [SRD]_H$
and le (uthy-order SRDES) $\equiv$ op $\subseteq$
and eq (uthy-order SRDES) $\equiv$ op $=$
by (unfold-locales, simp-all add: srdes-hcond-def SRD-Continuous)

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

abbreviation Chaos :: ('s,'t::trace,'a) hrel-rsp where
Chaos $\equiv \bot_{SRDES}$

abbreviation Miracle :: ('s,'t::trace,'a) hrel-rsp where
Miracle $\equiv \top_{SRDES}$

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

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

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

syntax
- srd-mu ::= pttrn $\Rightarrow$ logic $\Rightarrow$ logic ($\mu_R \cdot \cdot \cdot [0, 10] 10$
- srd-nu ::= pttrn $\Rightarrow$ logic $\Rightarrow$ logic ($\nu_R \cdot \cdot \cdot [0, 10] 10$

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

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

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

3.1 Reactive design forms

lemma srdes-skip-def: II \equiv \mu X. F(X) = \mu X. F(SRD(X))
by (metis assms srdes-hcond-def srdes-theory-continuous.utp-lfp-def)
end

3 Reactive Design Specifications

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

3.1 Reactive design forms

lemma srdes-skip-def: II \equiv \mu X. F(X) = \mu X. F(SRD(X))
by (metis assms srdes-hcond-def srdes-theory-continuous.utp-lfp-def)
end
shows $RD2(R_s(P \vdash Q)) = R_s(P \vdash Q)$
using assms
by (metis H2-design RD2-RHS-commute RD2-def)

lemma wait-false-design:
$(P \vdash Q) \_f = ((P \_f) \vdash (Q \_f))$
by (rel-auto)

lemma RD-RH-design-form:
$RD(P) = R((\neg P\_f) \vdash P\_f)$
proof -
have $RD(P) = RD1(RD2(R1(R2c(R3c(P)))))$
  by (simp add: RD-alt-def RH-def)
also have ... = $RD1(H2(R1(R2s(R3c(P)))))$
  by (simp add: R1-R2s-R2c RD2-def)
also have ... = $RD1(R1(H2(R2s(R3c(P)))))$
  by (simp add: R1-H2-commute)
also have ... = $R1(R1(H2(R2s(R3c(P))))))$
  by (simp add: R1-R2s-commute)
also have ... = $R1(R2s(H1(R3c(H2(R1(P)))))$
  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 ... = $R2(R1(H1(R3c(R2s(H1(P))))))$
  by (metis R1-R3c-commute RD1-via-R1)
also have ... = $R2(R1(R2s(H1(R3c(R1(P))))))$
  by (simp add: R1-R3c-commute RD1-via-R1)
also have ... = $RH(H(R1(P)))$
  by (metis R1-R2s-R2c RH-idem assms(1) assms(2))
also have ... = $RH(H(P))$
  by (simp add: R1-H2-commute R1-R3c-commute RD1-via-R1)
also have ... = $RH((\neg P\_f) \vdash P\_f)$
  by (simp add: H1-H2-eq-design)
also have ... = $R((\neg P\_f) \vdash P\_f)$
  by (metis RD2-R3c-subst-wait RH-def subst-not wait-false-design)
finally show ?thesis .
qed

lemma RD-reactive-design:
assumes P is RD
shows $R((\neg P\_f) \vdash P\_f) = P$
by (metis RD-RH-design-form Healthy-def' assms)

lemma RD-RH-design:
assumes $\$ok \_f \$ P $\$ok \_f \$ Q
shows $RD(R(P \vdash Q)) = R(P \vdash Q)$
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 \_f \$ P $\$ok \_f \$ Q
shows $R(P \vdash Q)$ is RD
by (simp add: RD-RH-design Healthy-def' assms(1) assms(2))
lemma SRD-RH-design-form:
SRD(P) = Rₜ((¬ Pᶠ) ⊢ Pᵗ)

proof 
  have SRD(P) = R₁(R₂c(R₃h(R₁(R₂d(R₃idem(R₂c-H₂-commute R₁-R₂c-commute R₁-R₃h-commute R₂-R₃h-commute R₂⁻def RHS⁻def SRD⁻def)))))
  also have ... = R₁(R₂c(R₃h(H(P))))
  also have ...
  also have ...
finally show ?thesis .
qed

lemma SRD-reactive-design:
  assumes P is SRD
  shows Rs((¬ Pᶠ) ⊢ Pᵗ) = P
  by (metis SRD-RH-design-form Healthy⁻def assms)

lemma SRD-RH-design:
  assumes $ok´♯ P$ $ok´♯ Q$
  shows SRD(Rₜ(P ⊢ Q)) = Rₜ(P ⊢ 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 Rₜ(P ⊢ Q) is SRD
  by (simp add: Healthy⁻def’ SRD-RH-design assms(1) assms(2))

lemma SRD-RHS-H1-H2: SRD(P) = Rₜ(H(P))
  by (metis (no-types, lifting) H₁-H₂-eq-design R₃h-subst-wait RHS-def subst-not wait-false-design)

3.2 Auxiliary healthiness conditions

definition [upred-defs]: R₃c-pre(P) = (true ◁ $wait ▷ P)
definition [upred-defs]: R₃c-post(P) = ([H]D ◁ $wait ▷ P)
definition [upred-defs]: R₃h-post(P) = (∃ st · [H]D ◁ $wait ▷ P)
lemma R₃c-pre-conj: R₃c-pre(P ∧ Q) = (R₃c-pre(P) ∧ R₃c-pre(Q))
  by (rel-auto)
lemma R₃c-pre-seq:
  (true ;; Q) = true ⇒ R₃c-pre(P ;; Q) = (R₃c-pre(P) ;; Q)
  by (rel-auto)
lemma unrest-ok-R₃c-pre [unrest]: $ok´♯ P → $ok´♯ R₃c-pre(P)
  by (simp add: R₃c-pre-cond-def unrest)
lemma unrest-ok’-R₃c-pre [unrest]: $ok♯ P → $ok´♯ R₃c-pre(P)
by (simp add: R3c-pre-def cond-def unrest)

lemma unrest-ok-R3c-post [unrest]: \( \text{\$ok} \not\in P \implies \text{\$ok} \not\in R3c\text{-post}(P) \)
  by (simp add: R3c-post-def cond-def unrest)

lemma unrest-ok-R3c-post' [unrest]: \( \text{\$ok'} \not\in P \implies \text{\$ok'} \not\in R3c\text{-post}(P) \)
  by (simp add: R3c-post-def cond-def unrest)

lemma unrest-ok-R3h-post [unrest]: \( \text{\$ok} \not\in P \implies \text{\$ok} \not\in R3h\text{-post}(P) \)
  by (simp add: R3h-post-def cond-def unrest)

lemma unrest-ok-R3h-post' [unrest]: \( \text{\$ok'} \not\in P \implies \text{\$ok'} \not\in R3h\text{-post}(P) \)
  by (simp add: R3h-post-def cond-def unrest)

3.3 Composition laws

theorem R1-design-composition:
  fixes P Q :: ('t::trace,'a,'b) rel-rp
  and R S :: ('t,'b,'c) rel-rp
  assumes \( \text{\$ok'} \not\in P \) \( \text{\$ok} \not\in Q \) \( \text{\$ok} \not\in R \) \( \text{\$ok} \not\in S \)
  shows
  \( R1(P \supset Q) ; R1(R \supset S) = (\exists \, \text{\$ok}_0 \cdot (R1(P \supset Q)[<\text{\$ok}>/\text{\$ok'}] ; (R1(R \supset S))[<\text{\$ok}>/\text{\$ok}]) \)
  using seqr-middle ok-vwb-lens by blast
  also from assms have \( \ldots = (\exists \, \text{\$ok}_0 \cdot R1((\text{\$ok} \land P) \Rightarrow (<\text{\$ok}> \land Q)) ; R1((<\text{\$ok}> \land R) \Rightarrow (\text{\$ok'} \land S))) \)
    by (simp add: design-def R1-def usubst unrest)
  also from assms have \( \ldots = (((R1((\text{\$ok} \land P) \Rightarrow (true \land Q)) ; R1((true \land R) \Rightarrow (\text{\$ok'} \land S))) \)
    \lor (R1((\text{\$ok} \land P) \Rightarrow (false \land Q)) ; R1((false \land R) \Rightarrow (\text{\$ok'} \land S)))) \)
    by (simp add: false-alt-def true-alt-def)
  also from assms have \( \ldots = (((R1((\text{\$ok} \land P) \Rightarrow Q)) ; R1(R \Rightarrow (\text{\$ok'} \land S))) \)
    \lor (R1(\neg (\text{\$ok} \land P)) ; R1(true)) \)
    by simp
  also from assms have \( \ldots = ((R1(\neg \text{\$ok} \lor \neg P \lor Q) ; R1(\neg R \lor (\text{\$ok'} \land S))) \)
    \lor (R1(\neg \text{\$ok} \lor \neg P) ; R1(true)) \)
    by (simp add: impl-alt-def utp-pred-laws.sup.assoc)
  also from assms have \( \ldots = (((R1(\neg \text{\$ok} \lor \neg P) ; R1(\neg R \lor (\text{\$ok'} \land S))) \)
    \lor (R1(\neg \text{\$ok} \lor \neg P) ; R1(true)) \)
    by (simp add: R1-disj utp-pred-laws.disj-assoc)
  also from assms have \( \ldots = (((R1(Q) ; R1(\neg R \lor (\text{\$ok'} \land S))) \)
    \lor (R1(\neg \text{\$ok} \lor \neg P) ; R1(true)) \)
    by (rel-blast)
  also from assms have \( \ldots = (((R1(Q) ; (R1(\neg R) \lor R1(S) \land \text{\$ok'})) \)
    \lor (R1(\neg \text{\$ok} \lor \neg P) ; R1(true)) \)
    by (simp add: R1-disj R1-extend-cond utp-pred-laws.inf-commute)
  also have \( \ldots = (((R1(Q) ; (R1(\neg R) \lor R1(S) \land \text{\$ok'})) \)
    \lor (R1(\neg \text{\$ok}) :: ('t,'a,'b) rel-rp) ; R1(true)) \)
    by (simp add: R1-disj seqr-or-distl)
  also have \( \ldots = (((R1(Q) ; (R1(\neg R) \lor R1(S) \land \text{\$ok'})) \)
  by (rel-blast)
\[ \forall (R1(\neg \$ok)) \\
\vee (R1(\neg P) ; R1(true)) \]

**proof** –

**have** \(((R1(\neg \$ok) :: (t',a',\beta) \text{ rel-rp}) ; R1(true)) = \\
(R1(\neg \$ok) :: (t',a',\gamma) \text{ rel-rp})\)

**by** (rel-auto)

**thus** \(?thesis\)

**by** simp

**qed**

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

**by** (simp add: R1-extend-conj)

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

**by** (simp add: segr-or-distr utp-pred-laws.sup_assoc)

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

**by** (rel-blast)

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

**by** (rel-blast)

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

**by** (simp add: impl-alt-def utp-pred-laws.inf-commute)

**also have** \(\ldots = R1( (\neg (R1(\neg P) ; R1(true)) \wedge \neg (R1(Q) ; R1(\neg R)) \vdash (R1(Q) ; R1(S))) \)

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

**finally show** \(?thesis\).

**qed**

**theorem** R1-design-composition-RR:

**assumes** \(P \text{ is RR} \text{ Q is RR} \text{ R is RR} \text{ S is RR} \)

**shows** \((R1(P \vdash Q) ; R1(R \vdash S)) = R1(((\neg \neg R) \text{ wp}_r \text{ false} \wedge Q \text{ wp}_r R) \vdash (Q \vdash S)) \)

**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** \(P \text{ is RC} \text{ Q is RR} \text{ R is RR} \text{ S is RR} \)

**shows** \((R1(P \vdash Q) ; R1(R \vdash S)) = R1((P \wedge Q \text{ wp}_r R) \vdash (Q \vdash S)) \)

**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: R2c-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:
\( R1(R3c(P \vdash Q)) = R1(R3c-pre(P) \vdash R3c-post(Q)) \)
by (rel-auto)

lemma R1-R3h-design:
\( R1(R3h(P \vdash Q)) = R1(R3c-pre(P) \vdash R3h-post(Q)) \)
by (rel-auto)

lemma R3c-R1-design-composition:
assumes \( \text{sok'} \notin P \text{ sok'} \notin Q \text{ sok} \notin R \text{ sok} \notin S \)
shows \( R3c(R1(P \vdash Q)) \vdash R3c(R1(R \vdash S)) \)
\( \vdash R3c(R1(\neg R1(\neg P)) \vdash R1(\neg R)) \)
\( \vdash \neg (R1(Q) \vdash \neg (R1(Q) \vdash \neg \text{wait'})) \)

proof
have \( 1: \neg (R1(\neg R3c-pre P) \vdash R1 true)) \)
by (rel-auto)

have \( 2: \neg (R1(R3c-post Q) \vdash R1(\neg R)) \)
by (rel-auto, blast+)

have \( 3: \neg (R1(R3c-post Q) \vdash R1 R3c-post(S)) \)
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

qd

lemma R3h-R1-design-composition:
assumes \( \text{sok'} \notin P \text{ sok'} \notin Q \text{ sok} \notin R \text{ sok} \notin S \)
shows \( R3h(R1(R \vdash P)) \vdash R3h(R1(R \vdash S)) \)
\( \vdash R3h(R1(R \vdash P)) \vdash R1(\neg R) \)
\( \vdash \neg (R1(Q) \vdash \neg (R1(Q) \vdash \neg \text{wait'})) \)

proof
have \( 1: \neg (R1(\neg R3c-pre P) \vdash R1 true)) \)
by (rel-auto)

have \( 2: \neg (R1(R3h-post Q) \vdash R1(\neg R)) \)
by (rel-auto, blast+)

have \( 3: \neg (R1(R3h-post Q) \vdash R1 R3h-post(S)) \)
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

qd

lemma R2-design-composition:
assumes \( \text{sok'} \notin P \text{ sok'} \notin Q \text{ sok} \notin R \text{ sok} \notin S \)
shows \( R2(P \vdash Q) \vdash R2(R \vdash S) \)
\( \vdash R2(\neg R1(\neg R2c P) \vdash R1 true) \)
\( \vdash R2(\neg R1(R2c Q) \vdash R1(\neg R2c R)) \)

qd
proof
\(-\)
\[ (R1-R2c-commute) \]
done

lemma \(RH\)-design-composition:
assumes \(\$ok \not\in P \; \$ok \not\in Q \; \$ok \not\in R \; \$ok \not\in S\)
shows \((RH(P \vdash Q) ; ; RH(R \vdash S)) =
\[ RH(((\neg (R1 (\neg R2s P) ; ; R1 true) \land \neg ((R1 (R2s Q) \land (\neg \$wait'))) ; ; R1 (\neg R2s R))) \vdash
\[ (R1 (R2s Q) ; ; ([II]D \triangleleft \$wait \triangleright R1 (R2s S))))\]
proof –
have 1: \(R2c \; ((R1 (\neg R2s P) ; ; R1 true) = (R1 (\neg R2s P) ; ; R1 true))
by \(rel\)-auto
have 2: \(R2c \; ((R1 (R2s Q) \land (\neg \$wait')) ; ; R1 (\neg R2s R)) = ((R1 (R2s Q) \land (\neg \$wait')) ; ; R1 (\neg R2s R))
proof –
have \(((R1 (R2s Q) \land (\neg \$wait')) ; ; R1 (\neg R2s R)) = R1 (R2s Q ; ; R2 true)
by \(rel\)-auto
have \(((R1 (R2s Q) \land (\neg \$wait')) ; ; R1 (\neg R2s R)) = R2c \; ((R1 (\neg P) ; ; R2 true)) \rightleftharpoons
using \(R2c\)-not by blast
also have \((R1 (\neg R2s P) ; ; R1 true) = (R1 (\neg R2s P) ; ; R1 true)
by \(rel\)-auto
finally show \(\$thesis\)
by \(simp\) add: 1
qed

have 3: \(R2c \; ((R1 (R2s Q) ; ; ([II]D \triangleleft \$wait \triangleright R1 (R2s S)))) = (R1 (R2s Q) ; ; ([II]D \triangleleft \$wait \triangleright R1 (R2s S)))
proof –
have \(((R1 (R2s Q) ; ; ([II]D \triangleleft \$wait \triangleright R1 (R2s S))) = ((R1 (R2s Q) ; ; ([II]D \triangleleft \$wait \triangleright R1 (R2s S)))
by \(simp\) add: unsubst cond-unit-T \(R1\)-def \(R2s\)-def
also have \((R2c \; ((R1 (R2s Q) ; ; ([II]D \triangleleft \$wait \triangleright R1 (R2s S))) = ((R2c \; ((R1 (R2s Q) ; ; ([II]D \triangleleft \$wait \triangleright R1 (R2s S)))
by \(metis\) \(R2c\)-seq \(R2c\)-des-lift-skip
also have \((R2c \; ((R1 (R2s Q) ; ; ([II]D \triangleleft \$wait \triangleright R1 (R2s S)))
by \(simp\) add: unsubst \(R2c\)-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/$wait'$] :: ([I]D < $wait$ R1 (R2s S))[false/$wait'$]))
    = (((R1 (R2s Q)) [false/$wait'$] :: ([I]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-des-lift-skip R2-subst-wait

R2c-seq)

ultimately show ?thesis
proof –
    have [I]D < $wait$ R1 (R2s S) = R2 ([I]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

have (R1(R2s(R3c(P => Q))) :: R1(R2s(R3c(R => S)))) =
    (R3c(R1(R2s(P) => R2s(Q))) :: R3c(R1(R2s(R) => R2s(S))))
    by (metis (no-types, hide-lams) R1-R2s-R2c R1-R3c-commute R2c-R3c-commute R2-design)
also have ... = R3c(R1 (~ (R1 (~ R2s P) ;; R1 true) ∧ ~ ((R1 (R2s Q) ∧ ~ $wait$') ;; R1 (~ R2s R))) ⊢
    (R1 (R2s Q) ;; ([I]D < $wait$ R1 (R2s S))))
    by (simp add: R3c-R1-design-composition assms unrest)
also have ... = R3c(R1(R2c(~ (~ R1 (~ R2s P) ;; R1 true) ∧ ~ ((R1 (R2s Q) ∧ ~ $wait$') ;; R1 (~ R2s R))) ⊢
    (R1 (R2s Q) ;; ([I]D < $wait$ R1 (R2s S))))
    by (simp add: R2c-design R2c-and R2c-not I 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 (R1(P => Q)) :: R1(R => S) =
    R1(~ (R1 (~ R2s P) ;; R1 true) ∧ ~ ((R1 (R2s Q) ∧ ~ $wait$') ;; R1 (~ R2s R))) ⊢
    (R1 (R2s Q) ;; ([I]D < $wait$ R1 (R2s S)))
proof –
    have 1: R2c(R1 (~ R2s P) ;; R1 true) = (R1 (~ R2s P) ;; R1 true)
    proof –
    have 1:(R1 (~ R2s P) ;; R1 true) = (R1(R2 (~ P) ;; R2 true))
    by (rel-auto, blast)
    have R2c(R1(R2 (~ P) ;; R2 true)) = R2c(R1(R1 (~ 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 \(2: R2c (((R1 (R2s Q) \land \lnot \text{wait}')) ;; R1 (\lnot R2s R)) = ((R1 (R2s Q) \land \lnot \text{wait}')) ;; R1 (\lnot R2s R))\)

\begin{proof}
\begin{align*}
\text{have} \quad & ((R1 (R2s Q) \land \lnot \text{wait}')) ;; R1 (\lnot R2s R) = R1 (R2 (Q \land \lnot \text{wait}')) ;; R2 (\lnot R)) \\
\text{by} \quad & \text{(rel-auto, blast+)} \\
\text{hence} \quad & R2c (((R1 (R2s Q) \land \lnot \text{wait}')) ;; R1 (\lnot R2s R)) = (R2 (Q \land \lnot \text{wait}')) ;; R2 (\lnot R)) \\
\text{by} \quad & \text{(metis (no-types, lifting) R1-R2c-commute R1-R2c-is-R2 R2-seqr-distribute)} \\
\text{also have} \quad & \ldots = ((R1 (R2s Q) \land \lnot \text{wait}')) ;; R1 (\lnot R2s R)) \\
\text{by} \quad & \text{(rel-auto, blast+)} \\
\text{finally show} \quad & \text{thesis .}
\end{align*}
\end{proof}

\(\text{qed}\)

have \(3: R2c((R1 (R2s Q) ;; (\exists \ s t \cdot [I]_{D} < \text{wait} \triangleright R1 (R2s S))] = (R1 (R2s Q) ;; (\exists \ s t \cdot [I]_{D} < \text{wait} \triangleright R1 (R2s S))\))\)

\begin{proof}
\begin{align*}
\text{have} \quad & R2c(((R1 (R2s Q)) [\text{true}\text{/\text{wait}'}] ;; (\exists \ s t \cdot [I]_{D} < \text{wait} \triangleright R1 (R2s S)) [\text{true}\text{/\text{wait}'}]) \\
& = ((R1 (R2s Q)) [\text{true}\text{/\text{wait}'}] ;; (\exists \ s t \cdot [I]_{D} < \text{wait} \triangleright R1 (R2s S)) [\text{true}\text{/\text{wait}'}]) \\
\text{by} \quad & \text{(simp add: usubst cond-unit-T R1-def R2-def)} \\
\text{also have} \quad & \ldots = R2c(R2(Q[\text{true}\text{/\text{wait}'}]) ;; R2((\exists \ s t \cdot [I]_{D}) [\text{true}\text{/\text{wait}'}]) \\
\text{by} \quad & \text{(metis (no-types, lifting) R2-def R2-des-lift-skip R2-subst-wait-true R2-st-ex)} \\
\text{also have} \quad & \ldots = (R2(Q[\text{true}\text{/\text{wait}'}]) ;; R2((\exists \ s t \cdot [I]_{D}) [\text{true}\text{/\text{wait}'}]) \\
\text{using} \quad & \text{R2c-seq by blast} \\
\text{also have} \quad & \ldots = ((R1 (R2s Q)) [\text{true}\text{/\text{wait}'}] ;; (\exists \ s t \cdot [I]_{D}) < \text{wait} \triangleright R1 (R2s S)) [\text{true}\text{/\text{wait}'}]) \\
\text{apply} \quad & \text{(simp add: usubst R2-des-lift-skip)} \\
\text{apply} \quad & \text{(metis (no-types) R2-def R2-des-lift-skip R2-st-ex R2-subst-wait'-true R2-subst-wait-true)} \\
\text{done} \\
\text{finally show} \quad & \text{thesis .}
\end{align*}
\end{proof}

\(\text{qed}\)

moreover have \(R2c(((R1 (R2s Q)) [\text{false}\text{/\text{wait}'}] ;; (\exists \ s t \cdot [I]_{D}) < \text{wait} \triangleright R1 (R2s S)) [\text{false}\text{/\text{wait}'}]) \\
= ((R1 (R2s Q)) [\text{false}\text{/\text{wait}'}] ;; (\exists \ s t \cdot [I]_{D}) < \text{wait} \triangleright R1 (R2s S)) [\text{false}\text{/\text{wait}'}]) \\
\text{by} \quad & \text{(simp add: usubst)} \\
\text{(metis (no-types, lifting) R1-wait'-false R1-wait-false R2-R1-form R2-subst-wait'-false R2-subst-wait-false R2c-seq)} \\
\text{ultimately show} \quad & \text{thesis} \\
\text{by} \quad & \text{(smt R2-R1-form R2-condr' R2-des-lift-skip R2-st-ex R2c-seq R2s-wait)}
\end{proof}

\(\text{qed}\)

have \((R1(R2s(R3h(P \triangleright Q))) ;; R1(R2s(R3h(R \triangleright S)))) = \)

\begin{align*}
\text{by} \quad & \text{(metis (no-types, lift-lams) R1-R2c R2-R3h-commute R2-R3h-commute R2s-design)} \\
\text{also have} \quad & \ldots = R3h(R1 \lnot(R1 \lnot(R2 P)) ;; R1 \text{true} \land \lnot((R1 (R2s Q) \land \lnot \text{wait}')) ;; R1 (\lnot R2s R)) \triangleright \)

\begin{align*}
\text{by} \quad & \text{(simp add: R3h-R1-design-composition assms unrest)} \\
\text{also have} \quad & \ldots = R3h(R1(R2c(\lnot(R1 \lnot(R2 P)) ;; R1 \text{true} \land \lnot((R1 (R2s Q) \land \lnot \text{wait}')) ;; R1 (\lnot R2s R)) \triangleright \)

\begin{align*}
\text{by} \quad & \text{(simp add: R2c-design R2c-and R2c-not 1 2 3)} \\
\text{finally show} \quad & \text{thesis} \\
\text{by} \quad & \text{(simp add: R1-R2s-R2c R1-R3h-commute R2c-R3h-commute RHS-def)}
\end{align*}
\end{align*}
\(\text{qed}\)
lemma RHS-R2s-design-composition:

assumes
$\text{ok} \not\in P \; \text{ok} \not\in Q \; \text{ok} \not\in R \; \text{ok} \not\in S$

$P$ is R2s $Q$ is R2s $R$ is R2s $S$ is R2s

shows $(R_\text{s}(P \vdash Q) :: R_\text{s}(R \vdash S)) =$

$R_\text{s}((\neg (R_1 (\neg P) :: R_1 \text{ true}) \; \wedge \neg ((R_1 Q \; \wedge \neg \text{wait} ') :: R_1 (\neg R))) \vdash$

$(R_1 Q :: ((\exists \; st \cdot [H|D] < \text{wait} \triangleright R_1 S)))$

proof

have $f_1$: $R_\text{s} P = P$

by (meson Healthy-def assms(5))

have $f_2$: $R_\text{s} Q = Q$

by (meson Healthy-def assms(6))

have $f_3$: $R_\text{s} R = R$

by (meson Healthy-def assms(7))

have $R_\text{s} S = S$

by (meson Healthy-def assms(8))

then show $?\text{thesis}$

using $f_3 \; f_2 \; f_1$ by (simp add: RHS-design-composition assms(1) assms(2) assms(3) assms(4))

qed

lemma RH-design-export-R1: $R(P \vdash Q) = R(P \vdash R_1(Q))$

by (rel-auto)

lemma RH-design-export-R2s: $R(P \vdash Q) = R(P \vdash R_2s(Q))$

by (rel-auto)

lemma RH-design-export-R2c: $R(P \vdash Q) = R(P \vdash R_2c(Q))$

by (rel-auto)

lemma RHS-design-export-R1: $R_\text{s}(P \vdash Q) = R_\text{s}(P \vdash R_1(Q))$

by (rel-auto)

lemma RHS-design-export-R2s: $R_\text{s}(P \vdash Q) = R_\text{s}(P \vdash R_2s(Q))$

by (rel-auto)

lemma RHS-design-export-R2c: $R_\text{s}(P \vdash Q) = R_\text{s}(P \vdash R_2c(Q))$

by (rel-auto)

lemma RHS-design-export-R2: $R_\text{s}(P \vdash Q) = R_\text{s}(P \vdash R_2(Q))$

by (rel-auto)

lemma R1-design-R1-pre:

$R_\text{s}(R_1(P) \vdash Q) = R_\text{s}(P \vdash Q)$

by (rel-auto)

lemma RHS-design-ok-wait: $R_\text{s}(P[\text{true,false}/\text{ok},\text{\$wait}] \vdash Q[\text{true,false}/\text{ok},\text{\$wait}]) = R_\text{s}(P \vdash Q)$

by (rel-auto)

lemma RHS-design-neg-R1-pre:

$R_\text{s}((\neg R_1 P) \vdash R) = R_\text{s}((\neg P) \vdash R)$

by (rel-auto)

lemma RHS-design-conj-neg-R1-pre:

$R_\text{s}((\neg R_1 P) \wedge Q) \vdash R) = R_\text{s}(((\neg P) \wedge Q) \vdash R)$

by (rel-auto)
lemma RHS-pre-lemma: $(R_s P)^f_f = RI(R2c(P^f_f))$
  by (rel-auto)

lemma RHS-design-R2c-pre:
  $R_s(R2c(P) \vdash Q) = R_s(P \vdash Q)$
  by (rel-auto)

### 3.4 Refinement introduction laws

**lemma R1-design-refine:**

assumes

- $P_1$ is $R1 P_2$ is $R1 Q_1$ is $R1 Q_2$ is $R1$
- $\$ok \not\in P_1 \$ok' \not\in P_1 \$ok \not\in P_2 \$ok' \not\in P_2$
- $\$ok \not\in Q_1 \$ok' \not\in Q_1 \$ok \not\in Q_2 \$ok' \not\in Q_2$

shows

$R1(P_1 \vdash P_2) \sqsubseteq R1(Q_1 \vdash Q_2) \iff 'P_1 \Rightarrow Q_1' \land 'P_1 \land Q_2 \Rightarrow P_2'$

**proof**

have

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

$\iff 'R1((\exists \$ok;\$ok' \cdot P_1) \Rightarrow R1((\exists \$ok;\$ok' \cdot Q_1) \land R1((\exists \$ok;\$ok' \cdot Q_2)\cdot Q_2) \Rightarrow R1((\exists \$ok;\$ok' \cdot P_2)'$

by

**(rel-auto, meson+)**

thus

**(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

$R1(P_1 \vdash P_2) \sqsubseteq R1(Q_1 \vdash Q_2) \iff 'P_1 \Rightarrow Q_1' \land 'P_1 \land Q_2 \Rightarrow P_2'$

by

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

**lemma RHS-design-refine:**

assumes

- $P_1$ is $R1 P_2$ is $R2c Q_1$ is $R2c Q_2$ is $R2c$
- $\$ok \not\in P_1 \$ok' \not\in P_1 \$ok \not\in P_2 \$ok' \not\in P_2$
- $\$ok \not\in Q_1 \$ok' \not\in Q_1 \$ok \not\in Q_2 \$ok' \not\in Q_2$
- $\$wait \not\in P_1 \$wait \not\in P_2 \$wait \not\in Q_1 \$wait \not\in Q_2$

shows

$R_s(P_1 \vdash P_2) \sqsubseteq R_s(Q_1 \vdash Q_2) \iff 'P_1 \Rightarrow Q_1' \land 'P_1 \land Q_2 \Rightarrow P_2'$

**proof**

have

$R_s(P_1 \vdash P_2) \sqsubseteq R_s(Q_1 \vdash Q_2) \iff R1(R3h(R2c(P_1 \vdash P_2))) \sqsubseteq R1(R3h(R2c(Q_1 \vdash Q_2)))$

by

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

also have

... $\iff R1(R3h(P_1 \vdash P_2)) \sqsubseteq R1(R3h(Q_1 \vdash Q_2))$

by

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

also have

... $\iff R1(R3h(P_1 \vdash P_2))[\$false/$\$wait] \sqsubseteq R1(R3h(Q_1 \vdash Q_2))[\$false/$\$wait]$

by

**(rel-auto, meson+)**

also have

... $\iff R1(P_1 \vdash P_2)[\$false/$\$wait] \sqsubseteq R1(Q_1 \vdash Q_2)[\$false/$\$wait]$

by

**(rel-auto)**

also have

... $\iff R1(P_1 \vdash P_2) \sqsubseteq R1(Q_1 \vdash Q_2)$

by

**(simp add: usubst assms closure unrest)**

also have

... $\iff 'P_1 \Rightarrow Q_1' \land 'P_1 \land Q_2 \Rightarrow P_2'$

by

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

finally show

**(？thesis）**

qed

**lemma srdes-refine-intro:**

assumes

‘$P_1 \Rightarrow P_2$’ ‘$P_1 \land Q_2 \Rightarrow Q_1$’
shows $R_s(P_1 \vdash Q_1) \subseteq R_s(P_2 \vdash Q_2)$
by (simp add: RHS-mono assms design-refine-intro)

3.5 Distribution laws

lemma RHS-design-choice: $R_s(P_1 \vdash Q_1) \cap R_s(P_2 \vdash Q_2) = R_s((P_1 \land P_2) \vdash (Q_1 \lor Q_2))$
by (metis RHS-inf design-choice)

lemma RHS-design-sup: $R_s(P_1 \vdash Q_1) \cup R_s(P_2 \vdash Q_2) = R_s((P_1 \lor P_2) \vdash ((P_1 \Rightarrow Q_1) \land (P_2 \Rightarrow Q_2)))$
by (metis RHS-sup design-inf)

lemma RHS-design-USUP:
  assumes $A \neq \{\}$
  shows $(\bigsqcap i \in A \cdot R_s(P(i)) \vdash Q(i))) = R_s((\bigsqcap i \in A \cdot P(i)) \vdash (\bigsqcap i \in A \cdot Q(i)))$
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 ::
  $(t::trace, \alpha, \beta)$ rel-rp $\Rightarrow$ $(t', \alpha', \beta)$ rel-rp $\Rightarrow$ $(t', \alpha, \beta)$ rel-rp (infixr $\odot$ 65) where
[upred-defs]: $P \circ Q = (P \triangleleft \text{ wait'} \circ Q)$

lemma wait' -cond-unrest [unrest]:
  $x \not\in \text{ out-var}$ wait $\Rightarrow x \not\in P; x \notin Q$ $\Longrightarrow x \notin (P \circ Q)$
by (simp add: wait' -cond-def unrest)

lemma wait' -cond-subst [esubst]:
$\text{wait'} \not\in \sigma \Rightarrow \sigma \uparrow (P \circ Q) = (\sigma \uparrow P) \circ (\sigma \uparrow Q)$
by (simp add: wait' -cond-def usubst unrest)

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

lemma wait' -cond-seq: $(P \circ Q) :: R = ((P :: (\neg \text{ wait'} \land R)) \mathbin{\lor} (Q :: (\neg \text{ wait'} \land R)))$
by (simp add: wait' -cond-def cond-def seq-or-disil, rel-blast)

lemma wait' -cond-true: $(P \circ Q \land \neg \text{ wait'}) = (P \land \neg \text{ wait'})$
by (rel-auto)

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

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

lemma wait' -cond-conj-exchange:
$(P \circ Q) \land (R \circ S) = (P \land R) \circ (Q \land S)$

26
\begin{align*}
\text{by (rel-auto)}
\end{align*}

\textbf{lemma subst-wait'}-cond-true [usubst]: \((P \circ Q)[true/\text{wait}'] = P[true/\text{wait}']\)
\text{by (rel-auto)}

\textbf{lemma subst-wait'}-cond-false [usubst]: \((P \circ Q)[false/\text{wait}'] = Q[false/\text{wait}']\)
\text{by (rel-auto)}

\textbf{lemma subst-wait'}-left-subst: \((P)[true/\text{wait}'] \circ Q = (P \circ Q)\)
\text{by (rel-auto)}

\textbf{lemma subst-wait'}-right-subst: \((P \circ Q)[false/\text{wait}'] = (P \circ Q)\)
\text{by (rel-auto)}

\textbf{lemma wait'}-cond-split: \(P[true/\text{wait}'] \circ P[false/\text{wait}'] = P\)
\text{by (simp add: wait'-cond-def cond-var-split)}

\textbf{lemma wait}-cond-assoc [simp]: \(P \circ Q \circ R = P \circ R\)
\text{by (rel-auto)}

\textbf{lemma wait}-cond-shadow: \((P \circ Q) \circ R = P \circ Q \circ R\)
\text{by (rel-auto)}

\textbf{lemma wait}-cond-conj [simp]: \(P \circ (Q \land (R \circ S)) = P \circ (Q \land S)\)
\text{by (rel-auto)}

\textbf{lemma R1-wait'}-cond: \(R1(P \circ Q) = R1(P) \circ R1(Q)\)
\text{by (rel-auto)}

\textbf{lemma R2s-wait'}-cond: \(R2s(P \circ Q) = R2s(P) \circ R2s(Q)\)
\text{by (simp add: wait'-cond-def R2s-def R2s-def usubst)}

\textbf{lemma R2-wait'}-cond: \(R2(P \circ Q) = R2(P) \circ R2(Q)\)
\text{by (simp add: R2-def R2s-wait'-cond R1-wait'-cond)}

\textbf{lemma wait'}-cond-R1-closed [closure]: \([P \text{ is R1; } Q \text{ is R1 }] \implies P \circ Q \text{ is R1}\)
\text{by (simp add: Healthy-def R1-wait'-cond)}

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

\subsection{Export laws}

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

\textbf{lemma RH-design-post-R1}: \(\text{R}(P \vdash Q \circ R1(R)) = \text{R}(P \vdash Q \circ R)\)
\text{by (metis R1-wait'-cond RH-design-export-R1 RH-design-peri-R1)}

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

\textbf{lemma RH-design-post-R2s}: \(\text{R}(P \vdash Q \circ R2s(R)) = \text{R}(P \vdash Q \circ R)\)
\text{by (metis (no-types, lifting) R2s-idem R2s-wait'-cond RH-design-export-R2s)}
lemma RH-design-peri-R2c: \( R(P \vdash R2c(Q) \circ R) = R(P \vdash Q \circ R) \)
\[ \text{by (metis } R1-R2s-R2c \text{ RH-design-peri-R1 RH-design-peri-R2s) } \]

lemma RHS-design-peri-R1: \( R_s(P \vdash R1(Q) \circ R) = R_s(P \vdash Q \circ R) \)
\[ \text{by (metis } \text{(no-types, lifting) } R1-idem \text{ R1-wait'}-cond \text{ RHS-design-export-R1) } \]

lemma RHS-design-post-R1: \( R_s(P \vdash Q \circ R1(R)) = R_s(P \vdash Q \circ R) \)
\[ \text{by (metis } R1-wait'-cond \text{ RHS-design-export-R1 RHS-design-peri-R1) } \]

lemma RHS-design-peri-R2s: \( R_s(P \vdash R2s(Q) \circ R) = R_s(P \vdash Q \circ R) \)
\[ \text{by (metis } \text{(no-types, lifting) } R2s-idem \text{ R2s-wait'}-cond \text{ RHS-design-export-R2s) } \]

lemma RHS-design-post-R2s: \( R_s(P \vdash Q \circ R2s(R)) = R_s(P \vdash Q \circ R) \)
\[ \text{by (metis } R2s-wait'-cond \text{ RHS-design-export-R2s RHS-design-peri-R2s) } \]

lemma RHS-design-peri-R2c: \( R_s(P \vdash R2c(Q) \circ R) = R_s(P \vdash Q \circ R) \)
\[ \text{by (metis } R1-R2s-R2c \text{ RHS-design-peri-R1 RHS-design-peri-R2s) } \]

lemma RH-design-lemma1:
\[ RH(P \vdash (R1(R2c(Q)) \circ R) \circ S) = RH(P \vdash (Q \circ R) \circ S) \]
\[ \text{by (metis } \text{(no-types, lifting) } R1-R2c-is-R2 R1-R2s-R2c R2-R1-form R2-disj R2c-idem RH-design-peri-R1 RH-design-peri-R2s) \]

lemma RHS-design-lemma1:
\[ RH(P \vdash (R1(R2c(Q)) \circ R) \circ S) = RH(P \vdash (Q \circ R) \circ S) \]
\[ \text{by (metis } \text{(no-types, lifting) } R1-R2c-is-R2 R1-R2s-R2c R2-R1-form R2-disj R2c-idem RHS-design-peri-R1 RHS-design-peri-R2s) \]

4.3 Pre-, peri-, and postconditions

4.3.1 Definitions

abbreviation \( \text{pre}_s \equiv [\text{ok} \mapsto_s \text{true}, \text{ok}' \mapsto_s \text{false}, \text{wait} \mapsto_s \text{false}] \)
abbreviation \( \text{cmt}_s \equiv [\text{ok} \mapsto_s \text{true}, \text{ok}' \mapsto_s \text{true}, \text{wait} \mapsto_s \text{false}] \)
abbreviation \( \text{peri}_s \equiv [\text{ok} \mapsto_s \text{true}, \text{ok}' \mapsto_s \text{true}, \text{wait} \mapsto_s \text{false}, \text{wait}' \mapsto_s \text{true}] \)
abbreviation \( \text{post}_s \equiv [\text{ok} \mapsto_s \text{true}, \text{ok}' \mapsto_s \text{true}, \text{wait} \mapsto_s \text{false}, \text{wait}' \mapsto_s \text{false}] \)

abbreviation \( \text{npre}_R(P) \equiv \text{pre}_s \vdash P \)

definition \( [\text{upred-defs}]: \text{pre}_R(P) = (\neg \text{ npre}_R(P)) \)
definition \( [\text{upred-defs}]: \text{cmt}_R(P) = R1(\text{cmt}_s \vdash P) \)
definition \( [\text{upred-defs}]: \text{peri}_R(P) = R1(\text{peri}_s \vdash P) \)
definition \( [\text{upred-defs}]: \text{post}_R(P) = R1(\text{post}_s \vdash P) \)

4.3.2 Unrestriction laws

lemma ok-pre-unrest [unrest]: \( \text{ok} \nsubseteq \text{pre}_R P \)
\[ \text{by (simp } \text{add: } \text{pre}_R\text{-def unrest usubst) } \]

lemma ok-peri-unrest [unrest]: \( \text{ok} \nsubseteq \text{peri}_R P \)
\[ \text{by (simp } \text{add: } \text{peri}_R\text{-def unrest usubst) } \]

lemma ok-post-unrest [unrest]: \( \text{ok} \nsubseteq \text{post}_R P \)
\[ \text{by (simp } \text{add: } \text{post}_R\text{-def unrest usubst) } \]

lemma ok-cmt-unrest [unrest]: \( \text{ok} \nsubseteq \text{cmt}_R P \)
lemma ok′-pre-unrest [unrest]: $ok' \not\in pre_R P$
by (simp add: pre_R-def unrest usubst)

lemma ok′-peri-unrest [unrest]: $ok' \not\in peri_R P$
by (simp add: peri_R-def unrest usubst)

lemma ok′-post-unrest [unrest]: $ok' \not\in post_R P$
by (simp add: post_R-def unrest usubst)

lemma ok′-cmt-unrest [unrest]: $ok' \not\in cmt_R P$
by (simp add: cmt_R-def unrest usubst)

lemma wait-pre-unrest [unrest]: $wait \not\in pre_R P$
by (simp add: pre_R-def unrest usubst)

lemma wait-peri-unrest [unrest]: $wait \not\in peri_R P$
by (simp add: peri_R-def unrest usubst)

lemma wait-post-unrest [unrest]: $wait \not\in post_R P$
by (simp add: post_R-def unrest usubst)

lemma wait-cmt-unrest [unrest]: $wait \not\in cmt_R P$
by (simp add: cmt_R-def unrest usubst)

4.3.3 Substitution laws

lemma prevs-design: $prevs \uparrow (P \vdash Q) = (\neg prevs \uparrow P)$
by (simp add: design-def prev_R-def usubst)

lemma peris-design: $peris \uparrow (P \vdash Q \circ R) = peris \uparrow (P \Rightarrow Q)$
by (simp add: design-def usubst wait′-cond-def)

lemma postvs-design: $postvs \uparrow (P \vdash Q \circ R) = postvs \uparrow (P \Rightarrow R)$
by (simp add: design-def usubst wait′-cond-def)

lemma cmts-design: $cmts \uparrow (P \vdash Q) = cmts \uparrow (P \Rightarrow Q)$
by (simp add: design-def usubst wait′-cond-def)

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

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

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

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

lemma post₂-R₁ [usubst]: \( post₂ \uparrow R₁(P) = R₁(post₂ \uparrow P) \)
by (simp add: R₁-def usubst)

lemma post₂-R₂c [usubst]: \( post₂ \uparrow R₂c(P) = R₂c(post₂ \uparrow P) \)
by (simp add: R₂c-def R2s-def usubst)

lemma cnt₂-R₁ [usubst]: \( cnt₂ \uparrow R₁(P) = R₁(cnt₂ \uparrow P) \)
by (simp add: R₁-def usubst)

lemma cnt₂-R₂c [usubst]: \( cnt₂ \uparrow R₂c(P) = R₂c(cnt₂ \uparrow P) \)
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:
\( cmt_R(P[false/\$wait]) = cmt_R(P) \)
by (rel-auto)

lemma rea-pre-RHS-design: \( pre_R(R₁(P ⊢ Q)) = R₁(R₂c(pre₂ ⊢ P)) \)
by (simp add: RHS-def usubst R₃h-def pre₂-def pre₂-design R₁-negate-R₁ R₂c-not rea-not-def)

lemma rea-cmt-RHS-design: \( cmt_R(R₁(P ⊢ Q)) = R₁(R₂c(cmt₂ ⊢ P ⇒ r Q)) \)
by (simp add: RHS-def usubst R₃h-def cmt₂-def R₃h-design, rel-auto)

lemma peri-cmt-def: \( peri_R(P) = (cmt_R(P))[true/\$wait'] \)
by (rel-auto)

lemma post-cmt-def: \( post_R(P) = (cmt_R(P))[false/\$wait'] \)
by (rel-auto)

lemma rdes-export-cmt: \( Rₐ(P \vdash cmt₂ ⊢ Q) = Rₐ(P \vdash Q) \)
by (rel-auto)

lemma rdes-export-pre: \( Rₐ((P[true,false/ok,\$wait]) \vdash Q) = Rₐ(P \vdash Q) \)
by (rel-auto)

4.3.4 Healthiness laws

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

lemma R₁-R₂s-cmt-SRD:
assumes \( P \) is SRD

30
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 ... = 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 $\Rightarrow$ peri_R(P) is R1
  by (simp add: Healthy-def’ R1-peri-SRD)

lemma R1-R2c-peri-RHS:
  assumes $P$ is SRD
  shows $R1(R2c(peri_R(P))) = peri_R(P)$
  by (metis R1-R2c-commute R1-R2s-R2c R1-R2s-peri-SRD assms)

lemma R1-R2s-post-SRD:
  assumes $P$ is SRD
  shows $R1(R2s(post_R(P))) = post_R(P)$
  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 $R2c(peri_R(P)) = peri_R(P)$
  by (metis R1-R2c-commute R1-R2c-peri-RHS R1-peri-SRD assms)

lemma R1-post-SRD:
  assumes $P$ is SRD
  shows $R1(post_R(P)) = post_R(P)$
proof –
  have $R1(post_R(P)) = R1(R1(R2s(post_R(P))))$
    by (simp add: R1-R2s-post-SRD assms)
  also have ... = 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 $R2c(post_R(P)) = post_R(P)$
  by (metis R1-R2c-commute R1-R2s-R2c R1-R2s-post-SRD R1-post-SRD assms)
lemma postR-SRD-R1 [closure]: $P$ is SRD $\implies$ $post_R(P)$ is R1
by (simp add: Healthy-def' R1-post-SRD)

lemma R1-R2c-post-RHS:
assumes $P$ is SRD
shows $R1(R2c(post_R(P))) = post_R(P)$
by (metis R1-R2s-R2c R1-R2s-post-SRD assms)

lemma R2c-cmt-conj-wait':
$P$ is SRD $\implies$ $R2c(cmt_R P \land \neg$ wait $') = (cmt_R P \land \neg$ wait $')$
by (simp add: R2-def R2s-conj R2s-not R2s-wait')

lemma R2-cmt-conj-wait:
$P$ is SRD $\implies$ $R2(cmt_R P \land \neg$ wait $') = (cmt_R P \land \neg$ wait $')$
by (simp add: Healthy-def' R2c-cmt-SRD)

lemma R2c-preR:
$P$ is SRD $\implies$ $R2c(pre_R(P)) = pre_R(P)$
by (metis R1-R2c-commute R2c-idem SRD-reactive-design rea-pre-RHS-design)

lemma R2c-postR:
$P$ is SRD $\implies$ $R2c(post_R(P)) = post_R(P)$
by (metis no-types, lifting) R1-R2c-commute R2c-idem SRD-reactive-design rea-pre-RHS-design

lemma periR-RR [closure]: $P$ is SRD $\implies$ peri_R(P) is RR
by (rule RR-intro, simp-all add: closure unrest)

lemma wpR-trace-ident-pre [wp]:
($str' = u str \land [H]_R$) wpc $pre_R P = pre_R P$
by (rel-auto)

lemma R1-preR [closure]:
$pre_R(P)$ is R1
by (rel-auto)

lemma trace-ident-left-periR:
($str' = u str \land [H]_R$) ;; peri_R(P) = peri_R(P)
by (rel-auto)

lemma trace-ident-left-postR:
($str' = u str \land [H]_R$) ;; $post_R(P) = post_R(P)$
by (rel-auto)

32
lemma \textit{trace-ident-right-postR}: 
\textit{post}_R(P) \triangleright; (\$tr' = u \times tr \land [H]_R) = \textit{post}_R(P) 
\text{by (rel-auto)}

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

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

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

4.3.5 Calculation laws

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

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

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

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

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

lemma \textit{periR-Chaos [rdes]}: \textit{peri}_R(\text{Chaos}) = true_r 
\text{by (simp add: Chaos-def, rel-simp)}

lemma \textit{postR-Chaos [rdes]}: \textit{post}_R(\text{Chaos}) = true_r 
\text{by (simp add: Chaos-def, rel-simp)}

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

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

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

lemma \textit{preR-srdes-skip [rdes]}: \textit{pre}_R(H_R) = true_r 
\text{by (rel-auto)}
lemma \( \text{periR-srdes-skip} \): \( \text{peri}_R(I_R) = \text{false} \)
by (rel-auto)

lemma \( \text{postR-srdes-skip} \): \( \text{post}_R(I_R) = (\text{str}' = u \land [I]_R) \)
by (rel-auto)

lemma \( \text{preR-INF} \): \( A \neq \{\} \implies \text{pre}_R(\prod A) = (\prod P \in A \cdot \text{pre}_R(P)) \)
by (rel-auto)

lemma \( \text{periR-INF} \): \( \text{peri}_R(\prod A) = (\bigvee P \in A \cdot \text{peri}_R(P)) \)
by (rel-auto)

lemma \( \text{postR-INF} \): \( \text{post}_R(\prod A) = (\bigvee P \in A \cdot \text{post}_R(P)) \)
by (rel-auto)

lemma \( \text{preR-UINF} \): \( \text{pre}_R(\prod i \cdot P(i)) = (\bigsqcup i \cdot \text{pre}_R(P(i))) \)
by (rel-auto)

lemma \( \text{periR-UINF} \): \( \text{peri}_R(\prod i \cdot P(i)) = (\prod i \cdot \text{peri}_R(P(i))) \)
by (rel-auto)

lemma \( \text{postR-UINF} \): \( \text{post}_R(\prod i \cdot P(i)) = (\prod i \cdot \text{post}_R(P(i))) \)
by (rel-auto)

lemma \( \text{preR-UINF-member} \): \( A \neq \{\} \implies \text{pre}_R(\prod i \in A \cdot P(i)) = (\bigsqcup i \in A \cdot \text{pre}_R(P(i))) \)
by (rel-auto)

lemma \( \text{preR-UINF-member-2} \): \( A \neq \{\} \implies \text{pre}_R(\prod (i,j) \in A \cdot P i j) = (\bigsqcup (i,j) \in A \cdot \text{pre}_R(P i j)) \)
by (rel-auto)

lemma \( \text{preR-UINF-member-3} \): \( A \neq \{\} \implies \text{pre}_R(\prod (i,j,k) \in A \cdot P i j k) = (\bigsqcup (i,j,k) \in A \cdot \text{pre}_R(P i j k)) \)
by (rel-auto)

lemma \( \text{periR-UINF-member} \): \( \text{peri}_R(\prod i \in A \cdot P(i)) = (\prod i \in A \cdot \text{peri}_R(P(i))) \)
by (rel-auto)

lemma \( \text{periR-UINF-member-2} \): \( \text{peri}_R(\prod (i,j) \in A \cdot P i j) = (\prod (i,j) \in A \cdot \text{peri}_R(P i j)) \)
by (rel-auto)

lemma \( \text{periR-UINF-member-3} \): \( \text{peri}_R(\prod (i,j,k) \in A \cdot P i j k) = (\prod (i,j,k) \in A \cdot \text{peri}_R(P i j k)) \)
by (rel-auto)

lemma \( \text{postR-UINF-member} \): \( \text{post}_R(\prod i \in A \cdot P(i)) = (\prod i \in A \cdot \text{post}_R(P(i))) \)
by (rel-auto)

lemma \( \text{postR-UINF-member-2} \): \( \text{post}_R(\prod (i,j) \in A \cdot P i j) = (\prod (i,j) \in A \cdot \text{post}_R(P i j)) \)
by (rel-auto)

lemma \( \text{postR-UINF-member-3} \): \( \text{post}_R(\prod (i,j,k) \in A \cdot P i j k) = (\prod (i,j,k) \in A \cdot \text{post}_R(P i j k)) \)
by (rel-auto)

lemma \( \text{preR-inf} \): \( \text{pre}_R(P \sqcap Q) = (\text{pre}_R(P) \land \text{pre}_R(Q)) \)
by (rel-auto)

34
lemma periR-inf [rdes]: \( \text{peri}_R(P \cap Q) = (\text{peri}_R(P) \lor \text{peri}_R(Q)) \)
by (rel-auto)

lemma postR-inf [rdes]: \( \text{post}_R(P \cap Q) = (\text{post}_R(P) \lor \text{post}_R(Q)) \)
by (rel-auto)

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

lemma periR-SUP [rdes]: \( A \neq \emptyset \implies \text{peri}_R(\bigsqcup A) = (\forall P \in A \cdot \text{peri}_R(P)) \)
by (rel-auto)

lemma postR-SUP [rdes]: \( A \neq \emptyset \implies \text{post}_R(\bigsqcup A) = (\forall P \in A \cdot \text{post}_R(P)) \)
by (rel-auto)

4.4 Formation laws

lemma srdes-skip-tri-design [rdes-def]: \( \text{II}_R = \text{R}_s(\text{true} \vdash \text{false} \circ \text{II}_r) \)
by (simp add: srdes-skip-def, rel-auto)

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

lemma Miracle-tri-def [rdes-def]: \( \text{Miracle} = \text{R}_s(\text{true} \vdash \text{false} \circ \text{false}) \)
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 \( \text{R}_s(\text{RR}(P_1) \vdash \text{RR}(P_2) \circ \text{RR}(P_3)) = (\text{II}_R \circ \$\text{wait} \circ ((\$\text{ok} \land P_1) \Rightarrow_r (\$\text{ok} \land P_2 \circ P_3))) \)

proof
  have \( \text{R}_s(\text{RR}(P_1) \vdash \text{RR}(P_2) \circ \text{RR}(P_3)) = (\text{II}_R \circ \$\text{wait} \circ ((\$\text{ok} \land \text{RR}(P_1)) \Rightarrow_r (\$\text{ok} \land (\text{RR}(P_2) \circ \text{RR}(P_3)))) \)
  by (rel-auto) using minus-zero-eq by blast
  thus \( \text{\text{thesis}} \)
  by (simp add: Healthy-if assms)
qed

lemma RHS-design-pre-post-form:
\( \text{R}_s((\neg P^f_1) \vdash P^f_1) = \text{R}_s(\text{pre}_R(P) \vdash \text{cmt}_R(P)) \)

proof
  have \( \text{R}_s((\neg P^f_1) \vdash P^f_1) = \text{R}_s((\neg P^f_1)[\text{true}/\$\text{ok}] \vdash P^f_1[\text{true}/\$\text{ok}]) \)
  by (simp add: design-subst-ok)
  also have \( \ldots = \text{R}_s(\text{pre}_R(P) \vdash \text{cmt}_R(P)) \)
  by (simp add: pre_R-def cmt_R-def usubst, rel-auto)
  finally show \( \text{\text{thesis}} \).
qed

lemma SRD-as-reactive-design:
\( \text{SRD}(P) = \text{R}_s(\text{pre}_R(P) \vdash \text{cmt}_R(P)) \)
by (simp add: RHS-design-pre-post-form SRD-RH-design-form)

lemma SRD-reactive-design-alt:
assumes \( P \) is SRD
shows \( \text{R}_s(\text{pre}_R(P) \vdash \text{cmt}_R(P)) = P \)

proof
  have \( \text{R}_s(\text{pre}_R(P) \vdash \text{cmt}_R(P)) = \text{R}_s((\neg P^f_1) \vdash P^f_1) \)

by (simp add: RHS-design-pre-post-form)
thus \(?thesis\)
by (simp add: SRD-reactive-design assms)
qed

lemma SRD-reactive-tri-design-lemma:
\(SRD(P) = R_s((\neg P_f f) \vdash P_f f[true/\$wait.] \circ P_f f[false/\$wait.])\)
by (simp add: SRD-RH-design-form wait'-cond-split)

lemma SRD-as-reactive-tri-design:
\(SRD(P) = R_s\langle \text{pre}R(P) \vdash \text{peri}_R(P) \circ \text{post}R(P) \rangle\)
proof
\(\begin{array}{l}
\text{have } \text{SRD}(P) = R_s((\neg P_f f) \vdash P_f f[true/\$wait.] \circ P_f f[false/\$wait.]) \\
\text{by (simp add: SRD-RH-design-form wait'-cond-split)}
\end{array}\)
also have \(\ldots = R_s\langle \text{pre}R(P) \vdash \text{peri}_R(P) \circ \text{post}R(P) \rangle\)
apply (simp add: usubst)
apply (subst design-subst-ok-ok [THEN sym])
apply (simp add: preR-def periR-def postR-def usubst unrest)
apply (rel-auto)
done
finally show \(?thesis\).
qed

lemma SRD-reactive-tri-design:
assumes \(P \text{ is SRD}\)
shows \(R_s\langle \text{pre}R(P) \vdash \text{peri}_R(P) \circ \text{post}R(P) \rangle = P\)
by (metis Healthy-if SRD-as-reactive-tri-design assms)

lemma SRD-elim [RD-elim]: \([P \text{ is SRD}; Q(R_s\langle \text{pre}R(P) \vdash \text{peri}_R(P) \circ \text{post}R(P) \rangle)] \implies Q(P)\)
by (simp add: SRD-reactive-tri-design)

lemma RHS-tri-design-is-SRD [closure]:
assumes \(\$ok' \notin P \quad \$ok' \notin Q \quad \$ok' \notin R\)
shows \(R_s\langle P \vdash Q \circ R \rangle \text{ is SRD}\)
by (rule RHS-design-is-SRD, simp-all add: unrest assms)

lemma SRD-rdes-intro [closure]:
assumes \(P \text{ is RR} \quad Q \text{ is RR} \quad R \text{ is RR}\)
shows \(R_s\langle P \vdash Q \circ R \rangle \text{ is SRD}\)
by (rule RHS-tri-design-is-SRD, simp-all add: unrest closure assms)

lemma USUP-R1-R2s-cmt-SRD:
assumes \(A \subseteq [\text{SRD}]_H\)
shows \((\bigcup P \in A \cdot R1(R2s(cmt_R P))) = (\bigcup P \in A \cdot \text{cmt}_R P)\)
by (rule USUP-cong[of A],metis (mono-tags, lifting) Ball-Collect R1-R2s-cmt-SRD assms)

lemma UINF-R1-R2s-cmt-SRD:
assumes \(A \subseteq [\text{SRD}]_H\)
shows \((\bigcap P \in A \cdot R1(R2s(cmt_R P))) = (\bigcap P \in A \cdot \text{cmt}_R P)\)
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: \(P \subseteq Q \implies \text{pre}R(Q) \subseteq \text{pre}R(P)\)
by (rel-auto)
lemma periR-monotone: $P \subseteq Q \Rightarrow \text{peri}_R(P) \subseteq \text{peri}_R(Q)$
by (rel-auto)

lemma postR-monotone: $P \subseteq Q \Rightarrow \text{post}_R(P) \subseteq \text{post}_R(Q)$
by (rel-auto)

4.5 Composition laws

theorem RH-tri-design-composition:
assumes $\$ok \not\in P \ \$ok \not\in Q \ \$ok \not\in Q_2 \ \$ok \not\in R \ \$ok \not\in S_1 \ \$ok \not\in S_2$
$\$wait \not\in Q_2 \ \$wait \not\in S_1 \ \$wait \not\in S_2$
shows $(RH(P \vdash Q_1 \circ Q_2) \implies RH(R \vdash S_1 \circ S_2)) =\nRH(\neg (R1 \neg R2s P) \land \neg ((R1 (R2s Q_2) \land \neg \$wait) \implies R1 (\neg R2s R)) \vdash ((Q_1 \lor (R1 (R2s Q_2) \circ R1 (R2s S_1))) \circ ((R1 (R2s Q_2) \circ R1 (R2s S_2))))$

proof –

have 1:$(\neg ((R1 (R2s Q_1 \circ Q_2) \land \neg \$wait) \implies R1 (\neg R2s R)) =\n(\neg ((R1 (R2s Q_2) \land \neg \$wait) \implies R1 (\neg R2s R)))$
by (metis no-types, hide-lams) RH-extend-conj

have 2: $(R1 (R2s Q_1 \circ Q_2) \implies (\neg \$wait \lor R1 (R2s S_1) \circ R1 (R2s S_2))) =\n(R1 (R2s Q_1) \lor (R1 (R2s Q_2) \circ R1 (R2s S_1))) \circ (R1 (R2s Q_2) \circ R1 (R2s S_2))$

proof –

have $(R1 (R2s Q_1) \circ (\neg \$wait \land (\neg \$wait \lor R1 (R2s S_1) \circ R1 (R2s S_2))))$
= $(R1 (R2s Q_1) \circ (\neg \$wait \land \neg \$wait))$
by (rel-auto)

also have $(\neg \$wait \land \neg \$wait) \implies R1 (R2s S_1) \circ R1 (R2s S_2)$
by (rel-auto)

also from RH-tri-design-composition have $(\neg \$wait \land \neg \$wait) \implies R1 (R2s S_1) \circ R1 (R2s S_2)$
by (simp add: lift-des-skipe-dr-unit-unrest unrest)

finally show ?thesis .

qed

moreover have $(R1 (R2s Q_2) \implies (\neg \$wait \land (\neg \$wait \lor R1 (R2s S_1) \circ R1 (R2s S_2))))$
= $(R1 (R2s Q_2) \circ (\neg \$wait \land \neg \$wait))$
by (metis no-types, lifting)

also have $(\neg \$wait \land \neg \$wait) \implies R1 (R2s S_1) \circ R1 (R2s S_2)$
by (simp add: wait-cond-def usubst unrest assms)

finally show ?thesis .

qed

moreover have $(R1 (R2s Q_1 \circ \$wait) \lor (R1 (R2s Q_2) \circ R1 (R2s S_1) \circ R1 (R2s S_2)))$
= $(R1 (R2s Q_1) \lor (R1 (R2s Q_2) \circ R1 (R2s S_1) \circ R1 (R2s S_2)))$
by (simp add: wait-cond-def cond-seq-right-distr cond-and-T-integrate unrest)

37
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 ⊨ Q) :: R1(R ⊨ S)) = R1(((¬ P) wp R false ∧ Q wp R) ⊨ (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 ⊨ Q) :: R1(R ⊨ S)) = R1((P ∧ Q wp R) ⊨ (Q ;; S))
  by (simp add: R1-design-composition-RR assms unrest Healthy-if closure wp)

theorem RHS-tri-design-composition:
  assumes $ok' $ P $ok' $ Q1 $ok' $ Q2 $ok' $ R $ok' $ S1 $S $ok' $ S2
  $wait $ R $wait' $ Q2 $wait $ S1 $wait $ S2
  shows (Rn(P ⊨ Q1 ∨ Q2) :: Rn(R ⊨ S1 ∨ S2)) =
    Rn(¬ (R1 (¬ R2s P) ;; R1 true) ∧ ¬ (R1(R2s Q2) ;; R1 (¬ R2s R))) ⊨
    (((∃ $st' · Q1) ∨ (R1 (R2s Q2) ;; R1 (R2s S1))) ∪
    (R1 (R2s Q2) ;; R1 (R2s S1))) ∪
    (R1 (R2s Q2) ;; R1 (R2s S2)))
proof –
  have 1:¬ (((R1 (R2s (Q1 ∨ Q2)) ∧ ¬ $wait')) ;; R1 (¬ R2s R)) =
    (¬ (((R1 (R2s Q2) ∧ ¬ $wait') ;; R1 (¬ R2s R)))
  by (metis (no-types, hide-lams) R1-extend-conj R2s-conj R2s-not R2s-wait' wait'-cond-false)
  have 2: (R1 (R2s (Q1 ∨ Q2)) ;; ((∃ $st · [I]D) < $wait ∨ R1 (R2s (S1 ∨ S2)))) =
    (((∃ $st' · R1 (R2s Q1)) ∨ (R1 (R2s Q2) ;; R1 (R2s S1))) ∪ (R1 (R2s Q2) ;; R1 (R2s S2)))
proof –
  have (R1 (R2s Q1) ;; ($wait ∧ ((∃ $st · [I]D) < $wait ∨ R1 (R2s S1) ∧ R1 (R2s S2)))
    = (∃ $st' · ((R1 (R2s Q1)) ∧ $wait'))
proof –
  have (R1 (R2s Q1) ;; ($wait ∧ ((∃ $st · [I]D) < $wait ∨ R1 (R2s S1) ∧ R1 (R2s S2)))
    = (R1 (R2s Q1) ;; ($wait ∧ (∃ $st · [I]D)))
  by (rel-auto, blast+)
  also have ... = (((R1 (R2s Q1) ;; (∃ $st · [I]D)) ∧ $wait'))
  by (rel-auto)
  also from assms(2) have ... = (∃ $st' · ((R1 (R2s Q1)) ∧ $wait'))
  by (rel-auto, blast)
  finally show ?thesis .
qed

38
moreover have \((R1 (R2s Q_2)) \implies (\neg \text{wait} \land ((\exists \mathit{st} \cdot [I]_P) \land \text{wait} 
 R1 (R2s S_1) \circ R1 (R2s S_2)))\)
\[(\neg \text{wait} \land (\exists \mathit{st} \cdot [I]_P) \land \text{wait} \n R1 (R2s S_1) \circ R1 (R2s S_2)))\)

proof

have \((R1 (R2s Q_2)) \implies (\neg \text{wait} \land ((\exists \mathit{st} \cdot [I]_P) \land \text{wait} 
 R1 (R2s S_1) \circ R1 (R2s S_2)))\)
\[(\neg \text{wait} \land (\exists \mathit{st} \cdot [I]_P) \land \text{wait} \n R1 (R2s S_1) \circ 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 \(\ldots = ((R1 (R2s Q_2)) [false / \text{wait}^*] \implies (R1 (R2s S_1) \circ R1 (R2s S_2))[false / \text{wait}])\)

by (metis false-alt-def seq-right-one-point upred-eq-false wait-vwb-lens)

also have \(\ldots = ((R1 (R2s Q_2)) \implies (R1 (R2s S_1) \circ R1 (R2s S_2)))\)

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

finally show \(?\text{thesis} \).

qed

moreover

have \(\ldots = ((R1 (R2s Q_1) \land \text{wait}^* \lor ((R1 (R2s Q_2)) \implies (R1 (R2s S_1) \circ R1 (R2s S_2))))\)
\[(R1 (R2s Q_1) \lor (R1 (R2s Q_2) \circ R1 (R2s S_1))) \circ ((R1 (R2s Q_2) \circ R1 (R2s S_2)))\)

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

ultimately show \(?\text{thesis} \)

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 \(3 \implies (R1 (R2s Q_2) \land \neg \text{wait}^* \lor (R1 (R2s Q_2) \circ R1 (R2s S_2)))\)

by (rel-auto, blast, meson)

show \(?\text{thesis} \)

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-wait'-cond-ex RHS-design-lemma)
apply (metis R1-R2s-R3c RHS-design-lemma1)
done

qed

theorem RHS-tri-design-composition-wp:

assumes \(Sok' \implies P \implies Q_1 \implies Q_2 \implies S\)
\(\text{wait} \implies R \implies S\)
\(P \implies R \implies Q_1 \implies R \implies Q_2 \implies R \implies S\)
\(R \implies Q_2 \implies S\)
\(R \implies Q_2 \implies S\)
\(S\)

shows \(R_a(P \equiv Q_1 \circ Q_2) \equiv R_a(R \equiv S_1 \circ S_2)\)

is \(?\text{lhs} = \text{?rhs}\)

proof

have \(?\text{lhs} = R_a(P \equiv Q_1 \circ Q_2) \equiv R_a(R \equiv S_1 \circ S_2)\)

39
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-upw:
assumes  P is RR Q1 is RR Q2 is RR
R is RR S1 is RR S2 is RR
shows  Rn(P ⊨ Q1 o Q2) ;; Rn(R ⊨ S1 o S2) =
 Rn((¬r P) wp_r false ∧ Q2 wp_r R) ⊨ ((∃ $st' · Q1) ∩ (Q2 ∪ S1)) o (Q2 ∪ S2)) (is  ?lhs = ?rhs)
by (simp add: RHS-tri-design-composition-upw add: closure assms unrest RR-implies-R2c)

lemma RHS-tri-normal-design-composition:
assumes  $ok' ≏ P $ok' ≏ Q1 $ok' ≏ Q2 $ok R $ok S1 $ok S2
$wait' ≏ R $wait' ≏ Q2 $wait' ≏ S1 $wait S2
P is R2c Q1 is R1 Q1 is R2c Q2 is R1 Q2 is R2c
R is R2c S1 is R1 S1 is R2c S2 is R1 S2 is R2c
R1 (¬ P) :: R1(true) = R1(¬ P) $st' ≏ Q1
shows  Rn(P ⊨ Q1 o Q2) ;; Rn(R ⊨ S1 o S2)
 = Rn((P o Q2 wp_r R) ⊨ (Q1 ∪ (Q2 ∪ S1)) o (Q2 ∪ S2))
proof -
have  Rn(P ⊨ Q1 o Q2) ;; Rn(R ⊨ S1 o S2) =
 Rn((R1 (¬ P) wp_r false ∧ Q2 wp_r R) ⊨ ((∃ $st' · Q1) ∩ (Q2 ∪ S1)) o (Q2 ∪ S2))
by (simp-all add: RHS-tri-design-composition-upw rea-not-def assms unrest)
also have  ... = Rn((P o Q2 wp_r R) ⊨ (Q1 ∪ (Q2 ∪ S1)) o (Q2 ∪ S2))
by (simp add: assms wp-rea-def ex-unrest, rel-auto)
finally show  ?thesis  .
qed

lemma RHS-tri-normal-design-composition' [rdes-def]:
assumes  P is RC Q1 is RR $st' ≏ Q1 Q2 is RR R is RR S1 is RR S2 is RR
shows  Rn(P ⊨ Q1 o Q2) ;; Rn(R ⊨ S1 o S2)
 = Rn((P ∧ Q2 wp_r R) ⊨ (Q1 ∪ (Q2 ∪ S1)) o (Q2 ∪ S2))
proof -
have  R1 (¬ P) :: R1 true = R1 (¬ P)
using RC-implies-RC1[OF assms(1)]
by (simp add: Healthy-def RC1-def rea-not-def)
(metis R1-negate-R1 R1-seqr wp-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-lemmaa:
assumes  $ok' ≏ P $ok' ≏ Q $ok' ≏ R $wait' ≏ R
shows  Rn(P ⊨ Q o R) ;; H_R = Rn((¬r (¬r P) :: true_r) ⊨ (∃ $st' · Q) o R)
proof -
have  Rn(P ⊨ Q o R) ;; H_R = Rn(P ⊨ Q o R) ;; Rn(true ⊨ false o ($tr' =u $tr ∧ [H]_R))
by (simp add: rdes SKIP-tri-design, rel-auto)
also have  ... = Rn ((¬ R1 (¬ R2s P) :: R1 true) ⊨ (∃ $st' · Q) o (R1 (R2s R) :: R1 (R2s ($tr' =u $tr ∧ [H]_R))))
by (simp-all add: RHS-tri-design-composition assms unrest R2s-true R1-False R2s-false)
also have \ldots = R_s ((\neg R_1 \leftarrow R_2 \rightarrow P) ;; R_1 true) \vdash (\exists s^{t'} \cdot Q) \circ R_1 (R_2 \circ R)

proof –
  from assms(3,4) have \(R_1 (R_2 \circ R) ;; R_1 (R_2 \circ (\$tr^{t'} = _u \circ tr \leftarrow [I]) \circ R) = R_1 (R_2 \circ R)\)
    by (rel-auto, metis (no-types, lifting) minus-zero-eq, meson order-refl trc-class.diff-cancel)
  thus \(?thesis\)
    by (simp)
qed

also have \ldots = R_s ((\neg R) ;; R_1 true) \vdash ((\exists s^{t'} \cdot Q) \circ R)
by (metis (no-types, lifting) R1-R2s-R1-true-lemma R1-R2s-R2c R2c-not R2s-refine assms R2c-pre R2s-refine assms)

\subsection{4.6 Reﬁnement introduction laws}

\begin{center}
\begin{tabular}{l}
\textbf{lemma \textit{SRD-composition-up}}: \\
\textbf{assumes} \(P \in SRD\) \(Q\) is \(SRD\)
\textbf{shows} \(P ;; Q = R_s ((\neg P) ;; P \circ R \circ R) \vdash ((\exists s^{t'} \cdot P) \circ (\exists s^{t'} \cdot Q) \circ (\exists s^{t'} \cdot P) \circ (\exists s^{t'} \cdot Q) \circ (\exists s^{t'} \cdot P) \circ (\exists s^{t'} \cdot Q))\)
\(\vdash (\exists s^{t'} \cdot P) \circ (\exists s^{t'} \cdot Q) \circ (\exists s^{t'} \cdot P) \circ (\exists s^{t'} \cdot Q) \circ (\exists s^{t'} \cdot P) \circ (\exists s^{t'} \cdot Q)\)
\textbf{proof} –
  have \(P ;; Q = (R_s (\exists s^{t'} \cdot P) \vdash \exists s^{t'} \cdot Q) ;; R_s (\{s^{t'} \cdot P\} \vdash \exists s^{t'} \cdot Q)\)
    by (simp add: SRD-reactive-tri-design assms(1) assms(2))
also from assms
  have \ldots = \(?rhs\)
    by (simp add: \(\text{RHS-tri-design-composition-up}\) disj-upred-def unrest assms closure)
finally show \(?thesis\).
\end{tabular}
\end{center}

\end{document}
assumes $P_1 = Q_1$, $P_2 = Q_2$, $P_3 = Q_3$
shows $R_s(P_1 \triangleright P_2 \triangleright P_3) = R_s(Q_1 \triangleright Q_2 \triangleright Q_3)$
using assms by (simp)

lemma srdes-tri-refine-intro':
  assumes $P_2 \subseteq P_1$, $Q_1 \subseteq (P_1 \land Q_2)$, $R_1 \subseteq (P_1 \land R_2)$
shows $R_s(P_1 \triangleright Q_1 \triangleright R_1) \subseteq R_s(P_2 \triangleright Q_2 \triangleright R_2)$
using assms
by (rule-tac srdes-tri-refine-intro, simp-all add: refBy-order)

lemma SRD-peri-under-pre:
  assumes $P$ is SRD $\$wait' $\neq$ $\$pre$R$($P$)
shows ($\$pre$R$($P$) $\Rightarrow$ $\$peri$R$($P$)) = $\$peri$R$($P$)
proof –
  have $\$peri$R$($P$) =
      $\$peri$R$($R_s$($\$pre$R$($P$) $\triangleright$ $\$peri$R$($P$) $\land$ $\$post$R$($P$)))
  by (simp add: SRD-reactive-tri-design assms)
also have ... = ($\$pre$R$ $P$ $\Rightarrow$ $\$peri$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' $\neq$ $\$pre$R$($P$)
shows ($\$pre$R$($P$) $\Rightarrow$ $\$post$R$($P$)) = $\$post$R$($P$)
proof –
  have $\$post$R$($P$) =
      $\$post$R$($R_s$($\$pre$R$($P$) $\triangleright$ $\$peri$R$($P$) $\land$ $\$post$R$($P$)))
  by (simp add: SRD-reactive-tri-design assms)
also have ... = ($\$pre$R$ $P$ $\Rightarrow$ $\$post$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
    ' conspir $P$ $\Rightarrow$ conspir $Q$' $\$pre$R$($P$) $\land$ $\$peri$R$($Q$) $\Rightarrow$ $\$peri$R$($P$)'
    $\$pre$R$($P$) $\land$ $\$post$R$($Q$) $\Rightarrow$ $\$post$R$($P$)'
shows $P$ $\subseteq$ $Q$
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
    ' conspir $P$ $\Rightarrow$ conspir $Q$' $\$pre$R$($P$) $\land$ $\$peri$R$($P$) $\subseteq$ ($\$pre$R$($P$) $\land$ $\$peri$R$($Q$)) $\$post$R$($P$) $\subseteq$ ($\$pre$R$($P$) $\land$ $\$post$R$($Q$))
shows $P$ $\subseteq$ $Q$
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$) $\equiv$ $\$pre$R$($Q$) $\$peri$R$($P$) $\equiv$ $\$peri$R$($Q$) $\$post$R$($P$) $\equiv$ $\$post$R$($Q$)
shows $P = Q$
by (metis SRD-reactive-tri-design assms)
4.7 Closure laws

lemma SRD-srdes-skip [closure]: $I_R$ 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 $(P;; Q) = R_s((\neg \text{pre}_R P) \text{ wp} false \land \text{post}_R P \text{ wp} \text{ pre}_R Q) \vdash$
    $(\exists \; st' \cdot \text{peri}_R P) \lor \text{pre}_R P \lor post_R P \lor post_R Q)$
  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 $\Rightarrow P^* (\text{Suc } n)$ 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 $\Rightarrow P ;; P^* n$ is SRD
by (metis SRD-power-Suc upred-semiring power-Suc)

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

lemma SRD-Sup-closure [closure]:
  assumes $A \subseteq [\text{SRD}]_H A \neq \{}$
  shows $(\bigsqcap A)$ is SRD
proof
  have $\text{SRD} (\bigsqcap A) = (\bigsqcap (\text{SRD } 'A))$
    by (simp add: ContinuousD SRD-Continuous assms (2))
  also have ... = $(\bigsqcap A)$
    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]:
  $R_s((P_1 \lor P_2 \lor P_3) \land R_s(Q_1 \lor Q_2 \lor Q_3)) = R_s((P_1 \land Q_1) \lor (P_2 \lor Q_2) \lor (P_3 \lor Q_3))$
apply (simp add: RHS-design-choice)
apply (rule cong[of $R_s$, $R_s$])
apply (simp)
apply (rel-auto)
done

lemma RHS-tri-design-sup [rdes-def]:
\( R_s(P_1 \vdash P_2 \land P_3) \cup R_s(Q_1 \vdash Q_2 \land Q_3) = R_s((P_1 \lor Q_1) \vdash ((P_1 \Rightarrow P_2) \land (Q_1 \Rightarrow Q_2)) \land ((P_1 \Rightarrow P_3) \land (Q_1 \Rightarrow Q_3))) \)

by (simp add: RHS-design-sup, rel-auto)

lemma RHS-tri-design-conj [rdes-def]:
\( (R_s(P_1 \vdash P_2 \land P_3) \land R_s(Q_1 \vdash Q_2 \land Q_3)) = R_s((P_1 \lor Q_1) \vdash ((P_1 \Rightarrow P_2) \land (Q_1 \Rightarrow Q_2)) \land ((P_1 \Rightarrow P_3) \land (Q_1 \Rightarrow Q_3))) \)

by (simp add: RHS-tri-design-sup conj-upred-def)

lemma SRD-UINF [rdes-def]:
assumes \( A \neq \{\} \subseteq \{\text{SRD}\} \)
shows \( \prod A = R_s((\bigwedge P \in A \cdot \text{pre}(R(P))) \vdash (\bigvee P \in A \cdot \text{peri}(R(P))) \land (\bigvee P \in A \cdot \text{post}(R(P))) \)\)

proof –
have \( \prod A = R_s((\bigwedge P \in A \cdot \text{pre}(R(P))) \vdash \text{peri}(R(P)) \land \text{post}(R(P))) \)
  by (metis SRD-as-reactive-tri-design assms srdes-hcond-def 
  srdes-theory-continuous.healthy-inf srdes-theory-continuous.healthy-inf-def)
also have \( \cdots = R_s((\bigwedge P \in A \cdot \text{pre}(R(P))) \vdash (\bigvee P \in A \cdot \text{peri}(R(P))) \land (\bigvee P \in A \cdot \text{post}(R(P))) \)\)
  by (simp add: preR-INF periR-INF postR-INF assms)
finally show \( \text{thesis} \).
qed

lemma RHS-tri-design-USUP [rdes-def]:
assumes \( A \neq \{\} \)
shows \( \prod i \in A \cdot R_s(P(i) \vdash Q(i) \land R(i)) = R_s((\bigwedge i \in A \cdot P(i)) \vdash (\bigwedge i \in A \cdot Q(i)) \land (\bigwedge i \in A \cdot R(i))) \)

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

lemma SRD-UINF-mem:
assumes \( A \neq \{\} \subseteq \text{SRD} \) \( i \cdot P \in \text{SRD} \)
shows \( \prod i \in A \cdot P(i) = R_s((\bigwedge i \in A \cdot \text{pre}(R(P(i)))) \vdash (\bigvee i \in A \cdot \text{peri}(R(P(i))) \land (\bigvee i \in A \cdot \text{post}(R(P(i)))) \)\)
(is \( \text{lhs} = \text{rhs} \))

proof –
have \( \text{lhs} = (\prod (P \cdot A)) \)
  by (rel-auto)
also have \( \cdots = R_s((\bigwedge Pa \in P \cdot A \cdot \text{pre}(R(Pa))) \vdash (\prod Pa \in P \cdot A \cdot \text{peri}(R(Pa))) \land (\prod Pa \in P \cdot A \cdot \text{post}(R(Pa)))) \)
  by (subst rdes-def, simp-all add: assms image-subsetI)
also have \( \cdots = \text{rhs} \)
  by (rel-auto)
finally show \( \text{thesis} \).
qed

lemma RHS-tri-design-UINF-ind [rdes-def]:
(\( \prod i \cdot R_s(P_1(i) \vdash P_2(i) \land P_3(i)) = R_s((\bigwedge i \cdot P_1(i) \vdash (\bigvee i \cdot P_2(i))) \land (\bigvee i \cdot P_3(i))) \)\)
by (rel-auto)

lemma cond-srea-form [rdes-def]:
\( R_s((P \vdash Q_1 \land Q_2) \lor b \triangleright R) = R_s((R \vdash S_1 \lor S_2) \lor (Q_1 \land b \triangleright R S_1) \lor (Q_2 \land b \triangleright R S_2)) \)

proof –
have \( R_s((P \vdash Q_1 \land Q_2) \lor b \triangleright R) = R_s((P \vdash Q_1 \lor Q_2) \land \text{R2c}(\{b\}_{S \subset}) \land R_s(R \vdash S_1 \lor S_2) \)\)
  by (pred-auto)
also have \( \cdots = R_s((P \vdash Q_1 \land Q_2) \lor b \triangleright R) \)
  by (simp add: RHS-cond lift-cond-srea-def)
also have \( (P \triangleleft b \triangleright R) \vdash (Q_1 \triangleleft b \triangleright R \ S_1 \triangleleft b \triangleright R) \)
by (simp add: design-condr lift-cond-srea-def)
also have \( (P \triangleleft b \triangleright R) \vdash (Q_2 \triangleleft b \triangleright R \ S_2) \)
by (rule cong[of \( R_s \)], 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 \( P \triangleleft b \triangleright Q = R_s \ (\ (\ \triangleright \ P \triangleleft \ R \ P \triangleright \ R \ Q \) \triangleright \ P \triangleleft \ R \ Q \) \)
by (simp add: \( SRD\)-reactive-tri-design assms)
also have \( \ldots = R_s \ (\ (\ \triangleright \ P \triangleleft \ R \ P \triangleright \ R \ Q \) \triangleright \ P \triangleleft \ R \ Q \) \)
by (simp add: \( cond\)-srea-from\)
also have \( \ldots \) 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 \( \ \triangleright \) \( trace\)-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 \( P ;\ \Pi_R = \Pi_R \)
by (simp add: \( SRD\)-composition-wp closure \( rdes \) \( wp \) \( \ \triangleright \) \( trace\)-ident-right-postR \( SRD\)-reactive-tri-design assms)\)

lemma \( Miracle\)-left-zero:
assumes \( P \) is SRD
shows \( Miracle ;\ P = Miracle \)
proof –
have \( Miracle ;\ P = R_s (true \triangleright \ false) ;\ R_s (\triangleright \ P \triangleright \ cmt_R(P)) \)
by (simp add: \( Miracle\)-def \( SRD\)-reactive-design-alt assms)
also have \( \ldots = R_s (true \triangleright \ false) \)
by (simp add: \( RHS\)-design-composition unrest \( R1\)-false \( R2\)-false \( R2\)-true)\)
also have \( \ldots = Miracle \)
by (simp add: \( Miracle\)-def)\)
finally show \(?thesis\).
qed

lemma \( Chaos\)-left-zero:
assumes \( P \) is SRD
shows \( (Chaos ;\ P) = Chaos \)
proof –
have \( Chaos ;\ P = R_s (false \triangleright \ true) ;\ R_s (\triangleright \ P \triangleright \ cmt_R(P)) \)

45
by (simp add: Chaos-def SRD-reactive-design-alt assms)
also have ... = Rₙ ((¬ R₁ true ∧ ¬ (R₁ true ∧ ¬ $\$w$ait$^\prime$)) ;; R₁ (¬ R₂s (preₚR P))) ⊁
R₁ true ;; ((∃ $\$st • [H]_D) ≪ $\$w$ait$^\prime$ > R₁ (R₂s (cmtₚR P))))
by (simp add: RHS-design-composition unrest R₂s-false R₂s-true R₁-false)
also have ... = Rₙ (false ∧ ¬ (R₁ true ∧ ¬ $\$w$ait$^\prime$)) ;; R₁ (¬ R₂s (preₚR P))) ⊁
R₁ true ;; ((∃ $\$st • [H]_D) ≪ $\$w$ait$^\prime$ > R₁ (R₂s (cmtₚR P))))
by (simp add: RHS-design-conj-neg-R1-pre)
also have ... = Chaos
by (simp add: design-false-pre)
also have ... = Rₙ (false ⊁ true)
by (simp add: design-def)
also have ... = Chaos
by (simp add: Chaos-def)
finally show ?thesis .
qed

lemma SRD-right-Chaos-tri-lemma:
assumes P is SRD
shows P ::; Chaos = Rₙ (((¬ $\$p$reR P) wpₚ R false ∧ postₚ R P wpₚ false) ⊁ (∃ $\$st' • periₚR P) ◁ false)
by (simp add: SRD-composition-up closure rdes assms wp, rel-auto)

lemma SRD-right-Miracle-tri-lemma:
assumes P is SRD
shows P ::; Miracle = Rₙ (((¬ $\$p$reR P) wpₚ R false ⊁ (∃ $\$st' • periₚR P) ◁ false)
by (simp add: SRD-composition-up closure rdes assms wp, rel-auto)

Stateful reactive designs are left unital

overloading
srdes-unit := utp-unit :: (SRDES, (’s,’t::trace,’α) ruty) uthy ⇒ (’s,’t,’α) hrel-rsp

begin

definition srdes-unit :: (SRDES, (’s,’t::trace,’α) rpsy) uthy ⇒ (’s,’t,’α) hrel-rsp
where
srdes-unit T = H_R

end

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

4.10 Recursion laws

lemma mono-srd-iter:
assumes mono F F ∈ [SRD]_H → [SRD]_H
shows mono (λX. Rₙ (preₚR (F X) ⊁ periₚ (F X) ◁ postₚ (F X)))
apply (rule monoI)
apply (rule srdes-tri-refine-intro’)
apply (meson assms(1) monoE preₚR-antitone utp-pred-laws.le-infI2)
apply (meson assms(1) monoE periₚR-monotone utp-pred-laws.le-infI2)
apply (meson assms(1) monoE postₚR-monotone utp-pred-laws.le-infI2)
done

lemma mu-srd-SRD:
assumes mono F F ∈ [SRD]_H → [SRD]_H
shows (μ X • Rₙ (preₚR (F X) ⊁ periₚ (F X) ◁ postₚ (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)

46
lemma \textit{mu-srd-iter}:\newline
\begin{itemize}
  \item \textbf{assumes} \( \text{mono } F \in [\text{SRD}]_H \rightarrow [\text{SRD}]_H \)
  \item \textbf{shows} \( (\mu X \cdot R_s(\text{pre}_R(F(X)) \vdash \text{peri}_R(F(X)) \circ \text{post}_R(F(X)))) = F(\mu X \cdot R_s(\text{pre}_R(F(X)) \vdash \text{peri}_R(F(X)) \circ \text{post}_R(F(X)))) \)
\end{itemize}
\begin{itemize}
  \item \textbf{apply} (\text{subst} gfp-unfold)
  \item \textbf{apply} (\text{simp add:} monosrd-iter \text{assms})
  \item \textbf{using} \text{Healthy-func} \text{assms}(1) \text{assms}(2) \text{mu-srd-SRD} \text{apply} blast
\end{itemize}
done

lemma \textit{mu-srd-form}:\newline
\begin{itemize}
  \item \textbf{assumes} \( \text{mono } F \in [\text{SRD}]_H \rightarrow [\text{SRD}]_H \)
  \item \textbf{shows} \( \mu F = (\mu X \cdot R_s(\text{pre}_R(F(X)) \vdash \text{peri}_R(F(X)) \circ \text{post}_R(F(X)))) \)
\end{itemize}
\begin{itemize}
  \item \textbf{proof} –
    \item \textbf{have} \(1: (\mu X \cdot R_s(\text{pre}_R(F(X)) \vdash \text{peri}_R(F(X)) \circ \text{post}_R(F(X)))) \text{ is SRD} \)
      \item \textbf{by} (\text{simp add:} Healthy-apply-closed \text{assms}(1) \text{assms}(2) \text{mu-srd-SRD})
    \item \textbf{have} \(2: \text{Mono}_H \text{utp-order SRDES} F \)
      \item \textbf{by} (\text{simp add:} \text{assms}(1) \text{mono-Monotone-utp-order})
    \item \textbf{hence} \(3: \mu F = F (\mu F) \)
      \item \textbf{by} (\text{simp add:} \text{srdes-theory-continuous.}LFP-unfold[THEN sym] \text{assms})
    \item \textbf{hence} \(R_s(\text{pre}_R(F(F(\mu F))) \vdash \text{peri}_R(F(F(\mu F))) \circ \text{post}_R(F(F(\mu F)))) = \mu F \)
      \item \textbf{using} \text{SRD-reactive-tri-design by} \text{force}
    \item \textbf{hence} \( (\mu X \cdot R_s(\text{pre}_R(F(X)) \vdash \text{peri}_R(F(X)) \circ \text{post}_R(F(X)))) \subseteq F(\mu F) \)
      \item \textbf{by} (\text{simp add:} 2 \text{srdes-theory-continuous.weak.LFP-lemma3} \text{gfp-upperbound} \text{assms})
    \item \textbf{thus} \text{thesis}
      \item \textbf{using} \text{assms} \(1 \ 3 \text{srdes-theory-continuous.weak.LFP-lowerbound eq-iff} \text{mu-srd-iter} \)
      \item \textbf{by} (\text{metis} (\text{mono-tags, lifting}))
\end{itemize}
qed

lemma \textit{Monotonic-SRD-comp} [\text{closure}]: \textit{Monotonic} \((\text{op} ; ; P \circ \text{SRD})\)
\begin{itemize}
  \item \textbf{by} (\text{simp add:} \text{mono-def} R1-R2c-is-R2 R2-mono R3h-mono RD1-mono RD2-mono RHS-def SRD-def seqr-mono)
\end{itemize}
end

5 \textbf{Normal Reactive Designs}

theory \textit{utp-rdes-normal}
\begin{itemize}
  \item \textbf{imports}
    \begin{itemize}
      \item \textit{utp-rdes-triples}
      \item \textit{UTP-KAT.utp-kleene}
    \end{itemize}
  \item \textbf{begin}
\end{itemize}

This additional healthiness condition is analogous to H3

definition \textit{RD3} where
\begin{itemize}
  \item \textit{upred-defs}: \text{RD3}(P) = P ;; H_R
\end{itemize}

lemma \textit{RD3-idem}: \text{RD3}(\text{RD3}(P)) = \text{RD3}(P)
\begin{itemize}
  \item \textbf{proof} –
    \item \textbf{have} \(a: H_R ;; H_R = H_R \)
      \item \textbf{by} (\text{simp add:} \text{SRD-left-unit} \text{SRD-srdes-skip})
    \item \textbf{show} \text{thesis}
      \item \textbf{by} (\text{simp add:} \text{RD3-def seqr-assoc} a)
\end{itemize}

47
qed

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

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

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

**lemma** RD3-right-subsumes-RD2: RD2(RD3(P)) = RD3(P)
  
  **proof** –
  
  have \(a: II_R \;; J = II_R\)
    
    by (rel-auto)
  
  show ?thesis
    
    by (metis (no-types, hide-lams) H2-def RD2-def RD3-def a seqr-assoc)
  
  qed

**lemma** RD3-left-subsumes-RD2: RD3(RD2(P)) = RD3(P)
  
  **proof** –
  
  have \(a: J \;; II_R = II_R\)
    
    by (rel-simp, safe, blast+)
  
  show ?thesis
    
    by (metis (no-types, hide-lams) H2-def RD2-def RD3-def a seqr-assoc)
  
  qed

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

**lemma** RD3-intro-pre:
  
  assumes P is SRD \((\neg_r \; pre_R(P)) \;; \; true_r = (\neg_r \; pre_R(P)) \; \$st' \; \; \; \per_i_R(P)\)
  
  shows P is RD3

  **proof** –
  
  have RD3(P) = R_s ((\neg_r \; pre_R(P) \; wp_r \; false \; \vdash (\exists \; st' \; \cdot \; per_i_R(P) \; \circ \; post_R(P))
    
    by (simp add: RD3-def SRD-right-unit-tri-lemma assms)
  
  also have ... = R_s ((\neg_r \; pre_R(P) \; wp_r \; false \; \vdash \; \per_i_R(P) \; \circ \; post_R(P))
    
    by (simp add: assms(3) ex-unrest)
  
  also have ... = R_s ((\neg_r \; pre_R(P) \; wp_r \; false \; \vdash \; cmt_R(P))
    
    by (simp add: wait-cond-peri-post-cmt)
  
  also have ... = R_s (pre_R \; \circ \; 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 R_s((P \; \vdash \; Q \; \circ \; R)) \;; \; II_R = R_s((\neg_r \; (\neg_r \; P) \;; \; true_r) \; \vdash \; ((\exists \; st' \; \cdot \; Q) \; \circ \; R))

  **proof** –
  
  have R_s((P \; \vdash \; Q \; \circ \; R)) \;; \; II_R = R_s((P \; \vdash \; Q \; \circ \; R)) \;; \; R_s(true \; \vdash \; \false \; \circ \; (\$tr' =_u \; \$tr \; \land \; [II]_R))
    
    by (simp add: rhs-desk-tri-design, rel-auto)
  
  also have ... = R_s ((\neg \; R1 \; (\neg \; R2s \; P) \;; \; R1 \; true) \; \vdash \; ((\exists \; st' \; \cdot \; Q) \; \circ \; (R1 \; (R2s \; R) \;; \; R1 \; (R2s \; (\neg \; R2s \; \neg \; R2s \; false))))
    
    by (simp add: NASA-tri-design-composition assms unrest R2s-true R1-false R2s-false)

48
also have \( \ldots = R_u((\neg R1 \ (\neg R2s P) ;; R1 \ true) \vdash (\exists \ $st' \cdot Q) \circ R1 \ (R2s R)) \)

proof

- from \( \text{assms}(3,4) \) have \( (R1 \ (R2s R) ;; R1 \ (R2s (\$tr' =_u \ $tr \ [P]) R)) = R1 \ (R2s R) \)
  - by (rel-auto, metis (no-types, lifting) minus-zero-eq, meson order-refl trace-class.diff-cancel)

thus \( \exists \text{thesis} \)
  - by simp

qed

also have \( \ldots = R_u((\neg (\neg P) ;; R1 \ true) \vdash ((\exists \ $st' \cdot Q) \circ R)) \)
  - 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)

finally show \( \exists \text{thesis} \).

qed

lemma \( \text{RHS-tri-design-RD3-intro} \):
  assumes \( \ldots \)
  shows \( \ldots \)
  apply simp add: Healthy-def RD3-def
  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 \( \text{R1-right-unit-lemma} \):
  \[ \text{oaut} \ R b; \text{oaut} \ R e \implies (\neg_r b \lor \$tr \ u \leq_u \ $tr') ;; R1(true) = (\neg_r b \lor \$tr \ u \leq_u \ $tr') \]
  - by (rel-auto, blast, metis (no-types, lifting) dual-order.trans)

lemma \( \text{RHS-tri-design-RD3-intro-form} \):
  assumes \( \ldots \)
  shows \( \ldots \)
  apply (rule RHS-tri-design-RD3-intro)
  apply (simp-all add: assms unrest closure rpred)
  apply (subt R1-right-unit-lemma)
  apply (simp-all add: assms unrest)

done

definition \( \text{NSRD} :: (\text{"s,t::trace,"a}) hrel-rsp \Rightarrow (\text{"s,t,"a}) hrel-rsp \)
where \( \text{[upred-defs]}: \text{NSRD} = \text{RD1} \circ \text{RD3} \circ \text{RHS} \)

lemma \( \text{RD1-RD3-commute: \text{RD1}}(\text{RD3}(P)) = \text{RD3}(\text{RD1}(P)) \)
  - by (rel-auto, blast+)

lemma \( \text{NSRD-is-SRD \ [closure]: \ P is NSRD \Rightarrow P is SRD} \)
  - 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-seq-closure SRD-srdes-skip)

lemma \( \text{NSRD-elim \ [RD-elim]}: \)
  \[ \text{P is NSRD}; \ Q(R_u(\text{pre}_R(P) \vdash \text{peri}_R(P) \circ \text{post}_R(P))) \] \( \Rightarrow \) \( Q(P) \)

49
by (simp add: RD-elim closure)

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

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

lemma NSRD-form:
  \[ NSRD(P) = R_s((\neg_r (\neg_r \text{pre}_R(P)) :: R1 \text{true}) \Rightarrow ((\exists \text{st'} \cdot \text{peri}_R(P)) \cdot \text{post}_R(P))) \]

proof –
  have \[ \text{NSRD}(P) = RD3(SRD}(P) \]
    by (metis (no-types, lifting) NSRDndef RD1-RD3-commute RD3-left-subsumes-RD2 SRD-def comp-def)
  also have \[ ... = RD3(R_s(\text{pre}_R(P) \cdot \text{peri}_R(P) \cdot \text{post}_R(P))) \]
    by (simp add: SRD-as-reactive-tri-design)
  also have \[ ... = R_s(\text{pre}_R(P) \cdot \text{peri}_R(P) \cdot \text{post}_R(P)) :: I_R \]
    by (simp add: RD3-def)
  also have \[ ... = R_s(\neg_r (\neg_r \text{pre}_R(P)) :: R1 \text{true}) \Rightarrow ((\exists \text{st'} \cdot \text{peri}_R(P)) \cdot \text{post}_R(P)) \]
    by (simp add: RHS-tri-design-right-unit-lemma unrest)
  finally show \?thesis .
qed

lemma NSRD-healthy-form:
  assumes \( P \) is NSRD
  shows \( R_s((\neg_r (\neg_r \text{pre}_R(P)) :: R1 \text{true}) \Rightarrow ((\exists \text{st'} \cdot \text{peri}_R(P)) \cdot \text{post}_R(P))) = P \)
  by (metis Healthy-def NSRD-form assms)

lemma NSRD-Sup-closure [closure]:
  assumes \( A \subseteq \text{NSRD}_H \) \( A \neq \{\} \)
  shows \( \bigcap A \) is NSRD

proof –
  have \( \text{NSRD} (\bigcap A) = (\bigcap (\text{NSRD} \ A)) \)
    by (simp add: ContinuousD NSRD-Continuous assms)
  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 \( P \cap Q \) is NSRD
  using NSRD-Sup-closure[of \{P, Q\}] by (simp add: assms)

lemma NRSD-SUP-closure [closure]:
  \[ \bigwedge i. \ i \in A \Rightarrow P(i) \text{ is NSRD} ; A \neq \{\} \] \( \Rightarrow (\bigcap i \in A. P(i)) \) is NSRD
  by (rule NRSD-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 \( (\neg_r \text{pre}_R(P)) = (\neg_r \text{pre}_R(R_s((\neg_r (\neg_r \text{pre}_R(P)) :: R1 \text{true}) \Rightarrow ((\exists \text{st'} \cdot \text{peri}_R(P)) \cdot \text{post}_R(P)))))) \)
    by (simp add: NSRD-healthy-form assms)
  also have \( ... = R1 (R2c ((\neg_r \text{pre}_R P) :: R1 \text{true})) \)

50
by (simp add: rea-pre-RHS-design R1-negate-R1 R1-idem R1-rea-not R2c-rea-not usbst rpred unrest closure)
also have ... = (¬r pre_R P) ;; R1 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 (¬r pre_R(P)) ;; Q = (¬r 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' ♯ peri_R(P)
proof –
  have peri(R(P) = peri(R,((¬r (¬r pre_R(P))) ;; R1 true) ⊢ ((∃ $st' ♯ peri_R(P)) o post_R(P))))
    by (simp add: NSRD-healthy-form assms)
also have ... = R1 (R2c (¬r pre_R P) ;; R1 true ⇒ r (∃ $st' ♯ peri_R P))
    by (simp add: rea-pre-RHS-design usbst unrest)
also have $st' ♯ ...
    by (simp add: R1-def R2c-def unrest)
finally show ?thesis .
qed

lemma NSRD-wait'-unrest-pre [unrest]:
  assumes P is NSRD
  shows $wait' ♯ pre_R(P)
proof –
  have pre_R(P) = pre_R(R,((¬r (¬r pre_R(P))) ;; R1 true) ⊢ ((∃ $st' ♯ peri_R(P)) o post_R(P))))
    by (simp add: NSRD-healthy-form assms)
also have ... = (R1 (R2c (¬r pre_R P) ;; R1 true))
    by (simp add: rea-pre-RHS-design usbst unrest)
also have $wait' ♯ ...
    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(R,((¬r (¬r pre_R(P))) ;; R1 true) ⊢ ((∃ $st' ♯ peri_R(P)) o post_R(P))))
    by (simp add: NSRD-healthy-form assms)
also have ... = R1 (R2c (¬r pre_R P) ;; R1 true))
    by (simp add: rea-pre-RHS-design usbst unrest)
also have $st' ♯ ...
    by (simp add: R1-def R2c-def unrest)
finally show ?thesis .
qed

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 \( \text{preR-RR} \) [closure]: \( P \) is NSRD \( \Rightarrow \) \( \text{pre}_R(P) \) 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 \( (\neg \ \text{r pre}_R(P)) \odot \text{true} = (\neg \ \text{r pre}_R(P)) \odot \text{peri}_R(P) \)
shows \( P \) is NSRD
proof
  have \( \text{NSRD}(P) = R_s((\neg \ \text{r pre}_R(P)) \odot \text{true} \vdash (\exists \ \text{st}^* \cdot \text{peri}_R(P)) \odot \text{post}_R(P)) \)
    by (simp add: NSRD-form)
  also have \( = R_s(\text{pre}_R P \vdash \text{peri}_R P \odot \text{post}_R P) \)
    by (simp add: assms ex-unrest rpred closure)
  also have \( = P \)
    by (simp add: SRD-reactive-tri-design assms)
  finally show \( ?\text{thesis} \)
    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
\( \text{st}^* \cdot \text{peri}_R(P) \)
shows \( P \) is NSRD
by (metis Healthy-def NSRD-form SRD-reactive-tri-design assms)

lemma NSRD-rdes-intro [closure]:
assumes \( P \) is RC \( Q \) is RR \( R \) is RR \( \text{st}^* \odot Q \)
shows \( R_s(\text{pre}_R P \odot Q \odot R) \) is NSRD
by (rule NSRD-RC-intro, simp-all add: rdes closure assms unrest)

lemma SRD-RD3-implies-NSRD:
\[ \lll P \text{ is SRD}; \ P \text{ is RD3} \rrr \Rightarrow P \text{ is NSRD} \]
by (metis (no-types, lifting) Healthy-def NSRD-def RHS-idem SRD-healths(4) SRD-reactive-design comp-apply)

lemma NSRD-iff:
\( P \) is NSRD \( \iff (P \text{ is SRD}) \land (\neg \text{r pre}_R(P)) \odot \text{R1}(\text{true}) = (\neg \text{r pre}_R(P)) \land (\neg \text{r peri}_R(P)) \)
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

\[ P \subseteq Q \text{ is NSRD} \]
\[ [\text{pre}\_R(P) \Rightarrow \text{pre}\_R(Q)]; \text{pre}\_R(P) \land \text{peri}\_R(Q) \Rightarrow \text{peri}\_R(P)'; \text{pre}\_R(P) \land \text{post}\_R(Q) \Rightarrow \text{post}\_R(P') ] \]
\[ \Rightarrow R \]
\[ \text{shows } R \]
\[ \text{proof} \]
\[ \text{have } R\_s(\text{pre}\_R(P) \land \text{peri}\_R(P) \land \text{post}\_R(P)) \subseteq R\_s(\text{pre}\_R(Q) \land \text{peri}\_R(Q) \land \text{post}\_R(Q)) \]
\[ \text{by (simp add: NSRD-is-SRD SRD-reactive-tri-design assms) (1) assms(2) assms(3))} \]
\[ \text{hence } 1: \text{pre}\_R(P) \Rightarrow \text{pre}\_R(Q) \land \text{peri}\_R(Q) \Rightarrow \text{peri}\_R(P) \text{ and } 2: \text{pre}\_R(P) \land \text{peri}\_R(Q) \Rightarrow \text{peri}\_R(P') \text{ and } 3: \text{pre}\_R(P) \land \text{post}\_R(Q) \Rightarrow \text{post}\_R(P') \]
\[ \text{by (simp-all add: RHS-tri-design-refine assms closure)} \]
\[ \text{with assms(4) show } \text{thesis} \]
\[ \text{by simp} \]
\[ \text{qed} \]

\textbf{lemma} NSRD-right-unit: \( P \text{ is NSRD} \Rightarrow P \land II\_R = P \)
\[ \text{by (metis Healthy-if NSRD-is-RD3 RD3-def)} \]

\textbf{lemma} NSRD-composition-wp:
\[ \text{assumes } P \text{ is NSRD} Q \text{ is SRD}
\[ \text{shows } P \land Q = R\_s((\text{pre}\_R(P) \land \text{post}\_R(P) \land \text{pre}\_R(Q)) \land (\text{pre}\_R(P) \land \text{post}\_R(P) \land \text{peri}\_R(Q)) \land (\text{post}\_R(P) \land \text{peri}\_R(Q))) \]
\[ \text{by (simp add: SRD-composition-wp assms NSRD-is-SRD wp-rea-def NSRD-neg-pre-unit NSRD-st'-unrest-peri R1-negate-R1 R1-preR cx-unrest rpred)} \]

\textbf{lemma} preR-NSRD-seq-lemma:
\[ \text{assumes } P \text{ is NSRD} Q \text{ is SRD}
\[ \text{shows } R\_1 (R\_2c (\text{post}\_R(P) \land \text{peri}\_R(Q))) = \text{post}\_R(P) \land (\text{peri}\_R(Q)) \]
\[ \text{proof} \]
\[ \text{have } \text{post}\_R(P) \land (\text{peri}\_R(Q)) = R\_1(R\_2c(\text{post}\_R(P))) \land R\_1(R\_2c(\text{peri}\_R(Q))) \]
\[ \text{by (simp add: NSRD-is-SRD R1-R2c-post-RHS R1-rea-not R2c-preR R2c-rea-not assms(1) assms(2))} \]
\[ \text{also have } ... = R\_1 (R\_2c (\text{post}\_R(P) \land (\text{peri}\_R(Q)))) \]
\[ \text{by (simp add: R1-seq R2c-R1-seq calculation)} \]
\[ \text{finally show } \text{thesis}. \]
\[ \text{qed} \]

\textbf{lemma} preR-NSRD-seq [rdes]:
\[ \text{assumes } P \text{ is NSRD} Q \text{ is SRD}
\[ \text{shows } \text{pre}\_R(P) \land Q = (\text{pre}\_R(P) \land \text{post}\_R(P) \land \text{pre}\_R(Q)) \]
\[ \text{by (simp add: NSRD-composition-wp assms rea-pre-RHS-design usubst unrest wp-rea-def R2c-disj R1-disj R2c-and R2c-preR R1-R2c-commute[THEN sym] R1-extend-conj' R1-extend-conj' R1-idem R2c-not closure)} \]
\[ \text{(metis (no_types, lifting) Healthy-if NSRD-is-SRD R1-R2c-commute R1-R2c-seqR-distribute R1-seqR-closure assms(1) assms(2) postR-R2c-closed postR-R2c-closed postR-SRD-R1 preR-R2c-closed rea-not-R1 rea-not-R2c)} \]

\textbf{lemma} periR-NSRD-seq [rdes]:
\[ \text{assumes } P \text{ is NSRD} Q \text{ is NSRD}
\[ \text{shows } \text{peri}\_R(P) \land Q = (\text{peri}\_R(P) \land \text{peri}\_R(Q)) \text{ and } (\text{peri}\_R(P) \land (\text{peri}\_R(Q))) \]
\[ \text{by (simp add: NSRD-composition-wp assms rea-peri-RHS-design usubst unrest wp-rea-def R1-extend-conj' R1-extend-conj' R1-disj R1-R2c-seqR-distribute R2c-disj R2c-and R2c-rea-impl R1-rea-impl' R2c-preR R2c-periR R1-rea-not'R2c-rea-not'R1-peri-SRD)} \]

\textbf{lemma} postR-NSRD-seq [rdes]:
\[ \text{assumes } P \text{ is NSRD} Q \text{ is NSRD}
\[ \text{shows } \text{post}\_R(P) \land Q = (\text{pre}\_R(P) \land \text{post}\_R(P) \land \text{pre}\_R(Q) \land \text{post}\_R(Q)) \]
\[ \text{by (simp add: NSRD-composition-wp assms rea-post-RHS-design usubst unrest wp-rea-def R1-extend-conj' R1-extend-conj' R1-disj R1-R2c-seqR-distribute R2c-disj R2c-and R2c-rea-impl R1-rea-impl' R2c-preR R2c-periR R1-rea-not'R2c-rea-not'R1-peri-SRD)} \]

53
by (simp add: NSRD-composition-up 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 R1
  assumes P is NSRD Q is NSRD
  shows R
  by (rule-tac NSRD-intro, simp-all add: seqr-or-distl NSRD-nsrd pre-unit assms closure rdes unrest)
qed

lemma RHS-tri-normal-design-composition:
assumes $\exists \exists' P \exists' Q_1 \exists' Q_2 \exists S \exists S_2$
$s\exists' \exists' S\exists' P \exists' Q_1 \exists' Q_2 \exists S \exists S_2$
$P is R2c Q_1 is R1 Q_1 is R2c Q_2 is R1 Q_2 is R2c$
$R is R2c S_1 is R1 S_1 is R2c S_2 is R1 S_2 is R2c$
$R1 (\neg P) ;; R1 (true) = R1 (\neg P) \exists' \exists' S\exists' Q_1$
sows R_r ((P \equiv Q_1 \equiv Q_2) ;; R_r ((R \equiv S_1 \equiv S_2) =
= R_r ((P \equiv Q_2 \equiv R) \equiv (Q_1 \equiv (Q_2 ;; S_1)) \equiv (Q_2 ;; S_2))
proof
have R_r ((P \equiv Q_1 \equiv Q_2) ;; R_r ((R \equiv S_1 \equiv S_2) =
= R_r ((P \equiv Q_2 \equiv R) \equiv (Q_1 \equiv (Q_2 ;; S_1)) \equiv (Q_2 ;; S_2))
by (simp-all add: RHS-tri-design-composition-up rea-not-def assms unrest)
also have ...
= R_r ((P \equiv Q_2 \equiv R) \equiv (Q_1 \equiv (Q_2 ;; S_1)) \equiv (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 $\exists' S\exists' Q_2 \equiv RR R is RR S_1 is RR S_2 is RR$
sows R_r ((P \equiv Q_1 \equiv Q_2) ;; R_r ((R \equiv S_1 \equiv S_2) =
= R_r ((P \equiv Q_2 \equiv R) \equiv (Q_1 \equiv (Q_2 ;; S_1)) \equiv (Q_2 ;; S_2))
proof
have R1 (\neg P) ;; R1 true = R1 (\neg P)
using RC-implies-RC1[OF assms(1)]
by (simp add: Healthy-def RC1-def rea-not-def)
(metis R1-negate-R1 R1-seqr wp-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 post_R(P) = false
shows P ;; Q = P
apply (simp add: NSRD-composition-up assms wp rpred closure)
using NSRD-is-SRD SRD-reactive-tri-design assms(1,3) apply fastforce
done

54
lemma NSRD-srd-skip [closure]: \( H_R \) 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 \( P \vdash \text{Miracle} \)
by (simp add: NSRD-composition-wp closure assms rdes wp rpred)

The set of non-terminating behaviours is a subset

lemma NSRD-right-Miracle-refines:
assumes \( P \) is NSRD
shows \( P \subseteq P \vdash \text{Miracle} \)
proof –
  have \( R_s (\pre R P \vdash \peri R P \circ \false) \subseteq R_s (\pre R P \vdash \peri R P \circ \false) \)
  by (rule srdes-tri-refine-intro, rel-auto+)
  thus \( \text{thesis} \)
  by (simp add: NSRD-elim NSRD-right-Miracle-tri-lemma assms)
qed

lemma upower-Suc-NSRD-closed [closure]:
\( P \) is NSRD \( 
\Rightarrow \)
\( P^\text{Suc n} \) is NSRD
proof (induct n)
  case 0
  then show \( \text{case} \)
  by (simp)
next
case (Suc n)
  then show \( \text{case} \)
  by (simp add: NSRD-seqr-closure upred-semiring.power-Suc)
qed

lemma NSRD-power-Suc [closure]:
\( P \) is NSRD \( 
\Rightarrow \)
\( P^\text{Suc n} \) is NSRD
by (metis upower-Suc-NSRD-closed upred-semiring.power-Suc)

lemma uplus-NSRD-closed [closure]: \( P \) is NSRD \( 
\Rightarrow \)
\( P^+ \) is NSRD
by (simp add: uplus-power-def closure)

lemma preR-power:
assumes \( P \) is NSRD
shows \( \pre R (P^\text{Suc n}) = (\bigsqcup \{0..n\}, (\post R (P^*) \circ i) \wp_r (\pre R (P))) \)
proof (induct n)
  case 0
  then show \( \text{case} \)
  by (simp add: wp closure)
next
case (Suc n) note hyp = this
have \( \text{pre}_R (P \cdot (\text{Suc} \ n \ + \ 1)) = \text{pre}_R (P \cdot P \cdot (n+1)) \)
by (simp add: upred-semiring.power-Suc)
also have \( \ldots = (\text{pre}_R P \land \text{post}_R P \ \wp_r \ \text{pre}_R (P \cdot (\text{Suc} \ n))) \)
using NSRD-iff \( \text{assms} \) \( \text{preR-NSRD-seq} \) \( \text{power-Suc-NSRD-closed} \) by fastforce
also have \( \ldots = (\text{pre}_R P \land \text{post}_R P \ \wp_r \ (\bigsqcup i \in \{0..n\}. \ \text{post}_R P \cdot i \ \wp_r \ \text{pre}_R P)) \)
by (simp add: hyp upred-semiring.power-Suc)
also have \( \ldots = (\text{pre}_R P \land (\bigsqcup i \in \{0..n\}. \ \text{post}_R P \ \wp_r \ (\text{post}_R P \cdot i \ \wp_r \ \text{pre}_R P))) \)
by (simp add: wp)
also have \( \ldots = (\text{pre}_R P \land (\bigsqcup i \in \{0..n\}. \ (\text{post}_R P \cdot (i+1) \ \wp_r \ \text{pre}_R P))) \)
proof
\[
\begin{align*}
\text{have } \bigwedge i. \ R1 \ (\text{post}_R P \cdot i \ ;\ (-r, \ \text{pre}_R P)) &= (\text{post}_R P \cdot i \ ;\ (-r, \ \text{pre}_R P)) \\
&\quad \text{by (induct-tac i, simp-all add: closure Healthy-if assms)} \\
&\quad \text{thus } \text{?thesis} \\
&\quad \text{by (simp add: wp-rea-def upred-semiring.power-Suc seqr-assoc rpred closure assms)} \\
&\text{qed} \\
\text{also have } \ldots = (\text{post}_R P \cdot 0 \ \wp_r \ \text{pre}_R P \land (\bigsqcup i \in \{0..n\}. \ (\text{post}_R P \cdot (i+1) \ \wp_r \ \text{pre}_R P))) \\
&\quad \text{by (simp add: wp assms closure)} \\
\text{also have } \ldots = (\text{post}_R P \cdot 0 \ \wp_r \ \text{pre}_R P \land (\bigsqcup i \in \{1..\text{Suc} \ n\}. \ (\text{post}_R P \cdot i \ \wp_r \ \text{pre}_R P))) \\
&\quad \text{proof} \\
&\quad \begin{align*}
&\text{have } (\bigsqcup i \in \{0..n\}. \ (\text{post}_R P \cdot (i+1) \ \wp_r \ \text{pre}_R P)) = (\bigsqcup i \in \{1..\text{Suc} \ n\}. \ (\text{post}_R P \cdot i \ \wp_r \ \text{pre}_R P)) \\\n&\quad &\quad \text{by (rule cong[of Inf], simp-all add: fun-eq iff)} \\
&\quad &\quad \text{(metis (no-types, lifting) image-Suc-atLeastAtMost image-cong image-image)} \\
&\quad &\quad \text{thus } \text{?thesis} \text{ by simp} \\
&\quad &\text{qed} \\
&\text{also have } \ldots = (\bigsqcup i \in \text{insert} \ 0 \ \{1..\text{Suc} \ n\}. \ (\text{post}_R P \cdot i \ \wp_r \ \text{pre}_R P)) \\
&\quad &\quad \text{by (simp add: conj-upred-def)} \\
&\text{also have } \ldots = (\bigsqcup i \in \{0..\text{Suc} \ n\}. \ \text{post}_R P \cdot i \ \wp_r \ \text{pre}_R P) \\
&\quad &\quad \text{by (simp add: atLeast0-atMost-Suc-eq-insert-0)} \\
&\text{finally show } \text{?case} \text{ by (simp add: upred-semiring.power-Suc)} \\
&\text{qed} \\
\end{align*}
\end{align*}
\]

Lemma preR-power' \([\text{rdes}]\): \(\text{assumes } P \text{ is NSRD} \)
shows \(\text{preR}(P \cdot P^\ast n) = (\bigsqcup i \in \{0..n\} \cdot \text{post}_R(P) \cdot i) \ \wp_r \ \text{pre}_R(P))\)
by (simp add: preR-power assms USUP-as-Inf[THEN sym])

Lemma preR-power-Suc \([\text{rdes}]\): \(\text{assumes } P \text{ is NSRD} \)
shows \(\text{preR}(P^\ast (\text{Suc} \ n)) = (\bigsqcup i \in \{0..n\} \cdot \text{post}_R(P) \cdot i) \ \wp_r \ \text{pre}_R(P))\)
by (simp add: upred-semiring.power-Suc rdes assms)

Declare upred-semiring.power-Suc \([\text{simp}]\)

Lemma periR-power: \(\text{assumes } P \text{ is NSRD} \)
shows \(\text{periR}(P \cdot P^\ast n) = (\text{preR}(P^\ast (\text{Suc} \ n)) \Rightarrow (\bigsqcup i \in \{0..n\}. \ \text{post}_R(P) \cdot i) :: \text{peri}_R(P))\)
proof (induct \(n\))
\[
\begin{align*}
&\text{case } 0 \\
&\quad \text{then show } \text{?case} \\
&\quad \quad \text{by (simp add: NSRD-is-SRD NSRD-wait'-unrest-pre SRD-peri-under-pre assms)} \\
&\text{next} \\
&\text{case } (\text{Suc} \ n) \text{ note } \text{hyp} = \text{this} \\
&\quad \text{have } \text{periR} (P \cdot (\text{Suc} \ n \ + \ 1)) = \text{periR} (P + P \cdot (n+1)) \\
&\quad \quad \text{by (simp)} \\
&\quad \text{also have } \ldots = (\text{periR}(P \cdot (\text{Suc} \ n \ + \ 1)) \Rightarrow (\text{peri}_R P \lor \text{post}_R P \lor \text{peri}_R (P \cdot P \cdot P \cdot n))) \\
\end{align*}
\]
by (simp add: closure assms rdes)
also have ... = (preR(P ∨ (Suc n + 1)) ⇒ periR P ∨ postR P :: (preR (P ∨ (Suc n)) ⇒ periR (P ∨ i); periR P)))
also have ... = (preR P ⇒ periR P ∨ (postR P wp_r preR (P ; P ∨ n) ⇒ postR P :: (preR (P ; P ∨ n) ⇒ periR (P ∨ i); periR P)))
also have ... = (preR P ⇒ periR P ∨ (postR P wp_r preR (P ; P ∨ n) ⇒ postR P :: ( periR (P ; P ∨ n) ⇒ periR (P ∨ i); periR P)))

proof 
have (∏ i∈{0..n}. postR P ∨ i) is R1
by (simp add: NSRD-is-SRD R1-Continuous R1-power Sup-Continuous-closed assms postR-SRD-R1)

thence 1:( (∏ i∈{0..n}. postR P ∨ i) ; periR P ) is R1
by ( simp add : closure assms)

thesis
by (simp only : wp-rea-impl-lemma, simp add: Healthy-if 1, simp add: R1-post-SRD assms closure)

qed
also
have ... = (preR P ∧ postR P wp_r preR (P ; P ∨ n) ⇒ periR P ∨ postR P :: ( (∏ i∈{0..n}. postR P ∨ i) ; periR P ))
by (pred-auto)
also
have ... = (preR P ∧ postR P wp_r preR (P ; P ∨ n) ⇒ periR P ∨ (∏ i∈{0..n}. postR P ∨ (Suc i)) ; periR P ))
by (simp add: seq-Sup-distr seq-assoc[THEN sym])
also
have ... = (preR P ∧ postR P wp_r preR (P ; P ∨ n) ⇒ periR P ∨ (( ∏ i∈{1..Suc n} . postR P ∨ i) ; periR P ) )

proof 
have (∏ i∈{0..n}. postR P ∨ Suc i) = (∏ i∈{1..Suc n} . postR P ∨ i)
apply (rule cong[of Sup], auto)
apply (metis atLeast0AtMost atMost-iff image-Suc-atLeastAtMost rev-image-eql upred-semiring.power-Suc)
using Suc-le-D apply fastforce

thus ?thesis by simp

also
have ... = (preR P ∧ postR P wp_r preR (P ; P ∨ n) ⇒ periR P ∨ (∏ i∈{0..Suc n} . postR P ∨ i) ; periR P )
by (simp add: SUP-atLeastAtMost-first winf-or seq-or-distr seq-or-distr)
also
have ... = (preR(P^*{Suc (Suc n)})) ⇒ periR ( (∏ i∈{0..Suc n} . postR P ∨ i) ; periR P )
by (simp add: rdes closure assms)

finally show ?case by (simp)

qed

lemma periR-power'[rdes]:
assumes \( P \) is NSRD
shows \( \text{peri}_R(P ;; P^\cdot n) = (\text{pre}_R(P^* (\text{Suc } n)) \Rightarrow_{r} (\prod_{i \in \{0\ldots n\} \cdot \text{post}_R (P) ^\cdot i) ;; \text{peri}_R(P)) \)
by (simp add: periR-power assms UINF-as-Sup THEN sym)

lemma periR-power-Suc [rdes]:
assumes \( P \) is NSRD
shows \( \text{peri}_R(P^* (\text{Suc } n)) = (\text{pre}_R(P^* (\text{Suc } n)) \Rightarrow_{r} \text{post}_R(P) ^\cdot \text{Suc } n) \)
proof (induct n)
  case 0
  then show ?case
  by (simp add: NSRD-is-SRD NSRD-power-Suc R1-power assms hyp post R-SRD-R1 upred-semiring.power-Suc wp-rea-impl-lemma)
also
have \( \ldots = (\text{pre}_R(P) ^\cdot (\text{Suc } n + 1)) \Rightarrow_{r} (\text{post}_R P ;; \text{pre}_R (P ;; P^\cdot n)) \)
  by (simp)
also have \( \ldots = (\text{pre}_R(P^* (\text{Suc } n + 1)) \Rightarrow_{r} (\text{post}_R P ;; \text{pre}_R (P ;; P^* n))) \)
  by (simp add: closure assms rdes)
also
have \( \ldots = (\text{pre}_R(P^* (\text{Suc } n + 1)) \Rightarrow_{r} (\text{post}_R P ;; (\text{pre}_R (P^* \text{Suc } n) \Rightarrow_{r} \text{post}_R P ^\cdot \text{Suc } n))) \)
  by (simp only: hyp)
also
have \( \ldots = (\text{pre}_R P \Rightarrow_{r} (\text{post}_R P wp_{r} \text{pre}_R (P ^\cdot \text{Suc } n)) \Rightarrow_{r} \text{post}_R P ;; (\text{pre}_R (P ^\cdot \text{Suc } n) \Rightarrow_{r} \text{post}_R P ^\cdot \text{Suc } n))) \)
  by (simp add: rdes closure assms, pred-auto)
also
have \( \ldots = (\text{pre}_R P \Rightarrow_{r} (\text{post}_R P wp_{r} \text{pre}_R (P ^\cdot \text{Suc } n)) \Rightarrow_{r} \text{post}_R P wp_{r} P ^\cdot \text{Suc } (\text{Suc } n)) \)
  by (metis (no-types, lifting) Healthy-if NSRD-is-SRD NSRD-power-Suc R1-power assms hyp postR-SRD-R1 upred-semiring.power-Suc wp-rea-impl-lemma)
also
have \( \ldots = (\text{pre}_R (P^* (\text{Suc } n))) \Rightarrow_{r} \text{post}_R P ^\cdot \text{Suc } (\text{Suc } n) \)
  by (pred-auto)
also have \( \ldots = (\text{pre}_R (P^* (\text{Suc } n))) \Rightarrow_{r} \text{post}_R P wp_{r} P ^\cdot \text{Suc } (\text{Suc } n) \)
  by (simp add: rdes closure assms)
finally show ?case by (simp)
qed

lemma postR-power-Suc [rdes]:
assumes \( P \) is NSRD
shows \( \text{post}_R(P^* (\text{Suc } n)) = (\text{pre}_R(P^* (\text{Suc } n)) \Rightarrow_{r} \text{post}_R(P) ^\cdot \text{Suc } n) \)
by (simp add: rdes assms)

lemma power-rdes-def [rdes-def]:
assumes \( R \) is RC \( Q \) is RR \( R \) is RR \$st \( \not\approx Q \)
shows \( \text{R}_{R}(P \vdash Q \circ R)^\cdot (\text{Suc } n) \)
  \( = \text{R}_{R}((\prod_{i \in \{0\ldots n\} \cdot (R ^\cdot i) wp_{r} P) \vdash (\prod_{i \in \{0\ldots n\} \cdot (R ^\cdot i)) ;; Q) \circ (R ^\cdot \text{Suc } n)) \)
proof (induct n)
  case 0
  then show ?case
  by (simp add: wp assms closure)
next
case \( \text{Suc } n \)
have 1: \((P \land (\bigsqcup i \in \{0..n\} \cdot R \wp_i (R \cdot i \wp_i P))) = (\bigsqcup i \in \{0..Suc n\} \cdot R \cdot i \wp_i P)\)
(is \(\text{lhs} = \text{rhs}\))

proof
  have \(\text{lhs} = (P \land (\bigsqcup i \in \{0..n\} \cdot (R \cdot Suc i \wp_i P)))\)
    by (simp add: wp closure assms)
  also have \(\ldots = (P \land (\bigsqcup i \in \{0..n\}. (R \cdot Suc i \wp_i P)))\)
    by (simp only: USUP-as-Inf-collect)
  also have \(\ldots = (P \land (\bigsqcup i \in \{1..Suc n\}. (R \cdot i \wp_i P)))\)
    by (metis (no-types, lifting) INF-cong One-nat-def image-Suc-atLeastAtMost image-image)
  also have \(\ldots = (\bigsqcup i \in insert 0 \{1..Suc n\}. (R \cdot i \wp_i P))\)
    by (simp add: wp assms closure conj-upred-def)
  also have \(\ldots \cdot Suc \cdot \text{insert} 0 \{0..Suc n\}. (R \cdot i \wp_i P)\)
    by (simp add: atLeastAtMost-insertL)
  finally show \(\text{thesis}\)
    by (simp add: USUP-as-Inf-collect)
qed

have 2: \((Q \lor R \cdot (\bigsqcup i \in \{0..n\} \cdot R \cdot i) \cdot Q) = (\bigsqcup i \in \{0..Suc n\} \cdot R \cdot i) \cdot Q\)
(is \(\text{lhs} = \text{rhs}\))

proof
  have \(\text{lhs} = (Q \lor (\bigsqcup i \in \{0..n\} \cdot R \cdot Suc i) \cdot Q)\)
    by (simp add: seqr-assoc THEN sym seq-UINF-distl)
  also have \(\ldots = (Q \lor (\bigsqcup i \in \{0..n\}. R \cdot Suc i) \cdot Q)\)
    by (simp only: UINF-as-Sup-collect)
  also have \(\ldots = (Q \lor (\bigsqcup i \in \{1..Suc n\}. R \cdot Suc i) \cdot Q)\)
    by (metis One-nat-def image-Suc-atLeastAtMost image-image)
  also have \(\ldots = (\bigsqcup i \in insert 0 \{1..Suc n\}. R \cdot Suc i) \cdot Q\)
    by (simp add: atLeastAtMost-insertL)
  finally show \(\text{thesis}\)
    by (simp add: UINF-as-Sup-collect)
qed

have 3: \((\bigsqcup i \in \{0..n\} \cdot R \cdot i) \cdot Q\) is \(RR\)

proof
  have \(\ldots = (\bigsqcup i \in \{0..n\} \cdot R \cdot Suc i) \cdot Q\)
    by (simp add: atLeastAtMost-insertL)
  also have \(\ldots = (Q \lor (\bigsqcup i \in \{0..n\}. R \cdot Suc i) \cdot Q)\)
    by (metis One-nat-def atLeastLessThanSuc-atLeastAtMost image-Suc-atLeastLessThan image-image)
  also have \(\ldots = (Q \lor (\bigsqcup i \in \{\Suc i\} \cdot R \cdot Suc i) \cdot Q)\)
    by (simp add: UINF-as-Sup-collect)
  also have \(\ldots\) is \(RR\)
    by (simp add: closure assms)
  finally show \(\text{thesis}\)
qed

from 1 2 3 Suc show \(\text{case}\)
  by (simp add: Suc RHS-tri-normal-design-composition closure assms wp)
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 $(R, (P + Q + R)) = R, (R^∗, wp, P + R^∗ D Q + R^∗)$
proof –
  have $1: i . R * i D Q = R^∗ D Q$
  by (metis (no-types) RA1 assms(2) rea-unit-unit(2) rrel-thy,Star-def ust-ar-def)
show $?thesis$
  by (simp add: uplus-power-def seq-UINF-distr wp closure assms rdes-def)
qed

5.1 UTP theory
typedec NSRDES

abbreviation NSRDES ≡ UTHY(NSRDES, (’s,’t::trace,’a) rsp)

overloading
  nsrdes-hcond == upthcond :: (NSRDES, (’s,’t::trace,’a) rsp) uthy ⇒ ((’s,’t,’a) rsp × (’s,’t,’a) rsp)
  nsrdes-unit == utp-unit :: (NSRDES, (’s,’t::trace,’a) rsp) uthy ⇒ (’s,’t,’a) hrel-rsp

begin
  definition nsrdes-hcond :: (NSRDES, (’s,’t::trace,’a) rsp) uthy ⇒ ((’s,’t,’a) rsp × (’s,’t,’a) rsp)
  health where
    [upred-defs]: nsrdes-hcond T = NSR
  definition nsrdes-unit :: (NSRDES, (’s,’t::trace,’a) rsp) uthy ⇒ (’s,’t,’a) hrel-rsp where
    [upred-defs]: nsrdes-unit T = II_R
end

interpretation nsrd-thy: utp-theory-kleene UTHY(NSRDES, (’s,’t::trace,’a) rsp)
  rewrites $\bigwedge P. P ∈ carrier (uthy-order NSRDES) \rightarrow P is NSR$
  and $P is NSRDES \leftrightarrow P is NSR$
  and $(\mu X. F (H_{NSRDES} X)) = (\mu X. F (NSR X))$
  and carrier (uthy-order NSRDES) → carrier (uthy-order NSRDES) ≡ [NSR]_H → [NSR]_H
  and $[H_{NSRDES}]_H \rightarrow [H_{NSRDES}]_H \equiv [NSR]_H \rightarrow [NSR]_H$
  and $T_{NSRDES} = Miracle$
  and $\Pi_{NSRDES} = II_R$
  and $le (uthy-order NSRDES) = op \sqsubseteq$
proof –
  interpret lat: utp-theory-continuous UTHY(NSRDES, (’s,’t,’a) rsp)
  by (unfold-locale, simp-all): nsrdes-hcond-def nsrdes-unit-def closure Healthy-if)
show 1: $T_{NSRDES} = (Miracle :: (’s,’t,’a) hrel-rsp)$
  by (metis NSR-SR-Miracle NSR-is-SRD lat.top-healthy lat.utp-theory-continuous-axioms nsrdes-hcond-def
  srdes-theory-continuous meet-top apred-semiring.add-commute utp-theory-continuous meet-top)

thus utp-theory-kleene UTHY(NSRDES, (’s,’t,’a) rsp)
  by (unfold-locale, simp-all): nsrdes-hcond-def nsrdes-unit-def closure Healthy-if Miracle-left-zero
  SDR-left-unit NSR-right-unit)
qed (simp-all add: nsrdes-hcond-def nsrdes-unit-def closure Healthy-if)

declare nsrd-thy.top-healthy [simp del]
declare nsrd-thy.bottom-healthy [simp del]

abbreviation TestR (test_R) where
test \( R \) \( P \equiv \mu \) test NSRDES \( P \)

**abbreviation** StarR :: \( (s, 't::trace, \alpha) \) hrel-rsp \( \Rightarrow (s, 't, \alpha) \) hrel-rsp \( (*^R \cdot [999] \cdot 999) \) where

StarR \( P \equiv P^*_{NSRDES} \)

**lemma** StarR-rdes-def [rdes-def]:
assumes \( P \) is RC \( Q \) is RR \( R \) is RR \$st \( \not\in \) \( Q \)
shows \( \{ (R_s (P \vdash Q \circ R))^* R = R_s((R^{*r \cdot wp}_r P) \vdash R^{*r \circ Q \circ R}) \}
\)
by (simp add: rrel-thy.Star-alt-def assms closure rdes-def unrest rpred disj-upred-def)

end

6 Syntax for reactive design contracts

**theory** utp-rdes-contracts
imports utp-rdes-normal
begin

We give an experimental syntax for reactive design contracts \( [P \vdash Q | R]_R \), where \( P \) is a pre-condition on undashed state variables only, \( Q \) is a per-condition that can refer to the trace and before state but not the after state, and \( R \) is a post-condition. Both \( Q \) and \( R \) can refer only to the trace contribution through a HOL variable \( trace \) which is bound to \&tt.

**definition** mk-RD :: \( 's \) upred \( \Rightarrow ('t::trace \Rightarrow 's \) upred \( ) \Rightarrow ('t \Rightarrow 's \) hrel \( ) \Rightarrow ('s, 't, \alpha) \) hrel-rsp
where
mk-RD \( P \) \( Q \) \( R \) = \( \lambda ([P]_{S<} \vdash [Q(x)]_{S<}[x \rightarrow \&tt]) \circ [R(x)]_{S<}[x \rightarrow \&tt]] \)

**definition** trace-pred :: \( ('t::trace \Rightarrow 's \) upred \( ) \Rightarrow ('s, 't, \alpha) \) hrel-rsp
where
[upred-defs]: trace-pred \( P \) = \( ([P x]_{S<}[x \rightarrow \&tt]) \)

**syntax**
- trace-var :: logic
- mk-RD :: logic \Rightarrow logic \Rightarrow logic \Rightarrow logic \( ([/-] \vdash [-]/[-]_R) \)
- trace-pred :: logic \Rightarrow logic \( ([/-]_1) \)

**parse-translation** 
\( \langle \langle \)
let
\( \text{fun} \) trace-var-tr \( [] = \text{Syntax.free} \) trace
| trace-var-tr \( x = \text{raise Match}; \)
in
\( ([@\{\text{syntax-const -trace-var}\}, K \text{ trace-var-tr}]) \)
end
\( \rangle \)

**translations**
\( [P \vdash Q | R]_R => \text{CONST} \) mk-RD \( \lambda \) trace-var. \( Q \) \( \lambda \) trace-var. \( R \)
\( [P \vdash Q | R]_R <= \text{CONST} \) mk-RD \( \lambda x. Q \) \( \lambda y. R \)
\( [P]_t \Rightarrow \text{CONST} \) trace-pred \( \lambda \) trace-var. \( P \)
\( [P]_t <= \text{CONST} \) trace-pred \( \lambda t. P \)

**lemma** SRD-mk-RD [closure]: \( [P \vdash Q(trace) | R(trace)]_R \) is SRD
\( \text{by (simp add: mk-RD-def closure unrest)} \)

**lemma** preR-mk-RD [rdes]: \( \text{preR}([P \vdash Q(trace) | R(trace)]_R) \) = \( \text{R1}([P]_{S<}) \)
\( \text{by (simp add: mk-RD-def rea-pre-RHS-design subst unrest R2c-not R2c-lift-state-pre} \)

61
lemma trace-pred-RR-closed [closure]:
\[ [P \text{ trace}]_t \text{ is RR} \]
by (rel-auto)

lemma unrest-trace-pred-st' [unrest]:
\$st' \sharp [P \text{ trace}]_t$
by (rel-auto)

lemma R2c-msubst-tt: $R2c (\text{msubst} (\lambda x. \lceil Q x \rceil_S) \& \text{tt}) = (\text{msubst} (\lambda x. \lceil Q x \rceil_S) \& \text{tt})$
by (rel-auto)

lemma periR-mk-RD [rdes]: $peri_R ([P \vdash Q \text{(trace)} | R(\text{trace})]_R) = ([P]_S \Rightarrow R1(([Q(\text{trace})]_S) [\text{trace} \rightarrow \& \text{tt}]))$
by (simp add: mk-RD-def rea-peri-RHS-design 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]: $post_R ([P \vdash Q \text{(trace)} | R(\text{trace})]_R) = ([P]_S \Rightarrow R1(([R(\text{trace})]_S) [\text{trace} \rightarrow \& \text{tt}]))$
by (simp add: mk-RD-def rea-post-RHS-design unrest R2c-not R2c-lift-state-pre impl-all-def R2c-disj R2c-msubst-tt R2c-rea-impl R1-rea-impl)

Refinement introduction law for contracts

lemma RD-contract-refine:
assumes
\[ Q \text{ is SRD } ' [P_1]_S \Rightarrow \text{pre}_R Q ' \]
\[ ' [P_1]_S \& peri_R Q = [P_2]_S \in,x\& \text{tt}' \]
\[ ' [P_1]_S \& post_R Q = [P_3]_S \in,x\& \text{tt}' \]
shows $[P_1 \vdash P_2(\text{trace}) | P_3(\text{trace})]_R \sqsubseteq Q$
proof
have $[P_1 \vdash P_2(\text{trace}) | P_3(\text{trace})]_R \sqsubseteq R_0 (\text{pre}_R(Q) \Rightarrow peri_R(Q) \circ post_R(Q))$
using \assms
by (simp add: mk-RD-def, rule_tac srdes-tri-refine-intro, simp-all)
thus \?thesis
by (simp add: SRD-reactive-tri-design \assms(1))

qed

end

7 Reactive design tactics

theory utp-rdes-tactics
imports utp-rdes-triples
begin

Theorems for normalisation

lemmas rdes-rel-norms =
prod.case-eq-if
conj-assoc
disj-assoc
conj-UINF-dist
conj-UINF-ind-dist
seqr-or-distl
seqr-or-distr
seq-UINF-distl
seq-UINF-distl'
seq-UINF-distr

62
The following tactic can be used to simply and evaluate reactive predicates.

**method** rpred-simp = (uexpr-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: rdes-def rdes-rel-norms rdes rpred cls closure alpha usubst unrest wp simps cong: cong))

Tactic to split a refinement conjecture into three POs

**method** rdes-refine-split uses cls cong simps =
  (rdes-simp cls: cls cong: cong simps: simps; rule-tac srdes-tri-refine-intro)

Tactic to split an equality conjecture into three POs

**method** rdes-eq-split uses cls cong simps =
  (rdes-simp cls: cls cong: cong simps: simps; (rule-tac srdes-tri-eq-intro))

Tactic to prove a refinement

**method** rdes-refine uses cls cong simps =
  (rdes-refine-split cls: cls cong: cong simps: simps; (insert cls; rel-auto))

Tactics to prove an equality

**method** rdes-eq uses cls cong simps =
  (rdes-eq-split cls: cls cong: cong simps: simps; rel-auto)

Via antisymmetry

**method** rdes-eq-anti uses cls cong simps =
  (rdes-simp cls: cls cong: cong simps: simps; (rule-tac antisym; (rule-tac srdes-tri-refine-intro; rel-auto)))

Tactic to calculate pre/peri/postconditions from reactive designs

**method** rdes-calc = (simp add: rdes rpred closure alpha usubst unrest wp prod.case-eq-if)

The following tactic attempts to prove a reactive design refinement by calculation of the pre-, peri-, and postconditions and then showing three implications between them using rel-blast.

**method** rdsp-rlfine =
  (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** rdsp-eq =
  (rule-tac antisym, rdes-refine, rdes-refine)

end

8 Reactive design parallel-by-merge

**theory** utp-rdes-parallel

**imports**
  utp-rdes-normal
R3h implicitly depends on RD1, and therefore it requires that both sides be RD1. We also require that both sides are R3c, and that $\text{wait}_m$ is a quasi-unit, and $\text{div}_m$ yields divergence.

**Lemma st-U0-alpha:** $\exists \text{ st} \cdot [II]_0 = (\exists \text{ st} \cdot [II]_0)\$
by (rel-auto)

**Lemma st-U1-alpha:** $\exists \text{ st} \cdot [II]_1 = (\exists \text{ st} \cdot [II]_0)$
by (rel-auto)

**Definition skip-rm:** $(s,t::\text{trace,'α}) \text{ resp merge (II_{RM}) where}
[\text{upred-defs}]: II_{RM} = (\exists \text{ st}_< \cdot \text{skip}_m \lor (\neg \text{ok}_< \land \text{str}_< \leq u \text{ str}_<))$

**Definition [upred-defs]:** $R3hm(M) = (II_{RM} \land \text{wait}_< \triangleright M)$

**Lemma R3hm-idem:** $R3hm(R3hm(P)) = R3hm(P)$
by (rel-auto)

**Lemma R3h-par-by-merge [closure]:**
assumes $P$ is R3h Q is R3h M is R3hm
shows $(P \parallel M Q)$ is R3h
proof –
have $(P \parallel M Q) = ((P \parallel M Q)[\text{true}/\text{ok}] < \text{ok} \triangleright (P \parallel M Q)[\text{false}/\text{ok}] < \text{ok} \triangleright (P \parallel M Q))$
by (simp add: cond-var-subst-left cond-var-subst-right)
also have ... = $((P \parallel M Q)[\text{true}/\text{ok},\text{wait}] < \text{ok} \triangleright (P \parallel M Q)[\text{false}/\text{ok},\text{wait}]) < \text{ok} \triangleright (P \parallel M Q))$
by (rel-auto)
also have ... = $((\exists \text{ st} \cdot II)[\text{true}/\text{ok},\text{wait}] < \text{ok} \triangleright (P \parallel M Q)[\text{false}/\text{ok},\text{wait}]) < \text{ok} \triangleright (P \parallel M Q))$
proof –
have $(P \parallel M Q)[\text{true}/\text{ok},\text{wait}] = (((P_0 \lor [Q_0]_1 \land \text{sv}_< = u \text{ sv}) \triangleright R3hm(M)[\text{true}/\text{ok},\text{wait}])$
by (simp add: par-by-merge-def U0-as-alpha U1-as-alpha asms Healthy-if)
also have ... = $((P_0 \lor [Q_0]_1 \land \text{sv}_< = u \text{ sv}) \triangleright (\exists \text{ st}_< \cdot \text{sv}_< = u \text{ sv}_<)))[\text{true}/\text{ok},\text{wait}]
by (rel-blast)
also have ... = $((R3h(P)_0 \lor [R3h(Q)]_1 \land \text{sv}_< = u \text{ sv}) \triangleright (\exists \text{ st}_< \cdot \text{sv}_< = u \text{ sv}_<)))[true,\text{ok},\text{wait}]$
by (simp add: asms Healthy-if)
also have ... = $((\exists \text{ st} \cdot II)[\text{true}/\text{ok},\text{wait}]$
by (rel-auto)
finally show ?thesis by (simp add: closure asms unrest)
qed
also have ... = $((\exists \text{ st} \cdot II)[\text{true}/\text{ok},\text{wait}] < \text{ok} \triangleright (R1(\text{true}))[\text{false}/\text{ok},\text{wait}]) < \text{ok} \triangleright (P \parallel M Q))$
proof –
have $(P \parallel M Q)[\text{false}/\text{ok},\text{wait}] = (((P_0 \lor [Q_0]_1 \land \text{sv}_< = u \text{ sv}) \triangleright R3hm(M))[\text{false}/\text{ok},\text{wait}])$
by (simp add: par-by-merge-def U0-as-alpha U1-as-alpha asms Healthy-if)
also have ... = $((P_0 \lor [Q_0]_1 \land \text{sv}_< = u \text{ sv}) \triangleright (\text{str}_< \leq u \text{ str})[\text{false}/\text{ok},\text{wait}])$
by (rel-blast)
also have ... = $((R3h(P)_0 \lor [R3h(Q)]_1 \land \text{sv}_< = u \text{ sv}) \triangleright (\text{str}_< \leq u \text{ str})[\text{false}/\text{ok},\text{wait}])$
by (simp add: asms Healthy-if)
also have ... = $((R1(\text{true}))[\text{false}/\text{ok},\text{wait}]$
by (rel-blast)
finally show ?thesis by simp
qed
also have ... = \((\exists \, s \cdot II) \land s < ok \land R1(true)\) \land \(\text{wait} < M\)
by (rel-auto)
also have ... = \(R3h(M)\)
by (simp add: R3h-cases)
finally show \(\text{thesis}\)
by (simp add: Healthy-def)
qed
definition [upred-defs]: \(RD1m(M) = (M \lor \neg \, s < ok \land \text{tr} < u \land s < \text{tr'})\)

lemma \(RD1\)-par-by-merge [closure]:
assumes \(P \equiv R1 \land Q \equiv R1 \land M \equiv R1m \land P \equiv RD1 \land Q \equiv RD1 \land M \equiv RD1m\)
shows \((P \equiv M) \equiv RD1\)
proof –
  have 1: \((RD1(R1(P)) \equiv RD1m(R1m(M))) \equiv RD1(R1(Q)))\equiv false/sok = R1(true)
  by (rel-blast)
  have \((P \equiv M) = (P \equiv M)\equiv true/sok = sok \equiv (P \equiv M)\equiv false/sok\)
  by (simp add: cond-var-split)
  also have ... = \(R1(P \equiv M) \equiv sok \lor R1(true)\)
  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 ... = \(RD1(P \equiv M)\)
  by (simp add: Healthy-if \(R1\)-par-by-merge RD1-alt-def assms(3))
finally show \(\text{thesis}\)
by (simp add: Healthy-def)
qed

lemma \(RD2\)-par-by-merge [closure]:
assumes \(M \equiv RD2\)
shows \((P \equiv M) \equiv RD2\)
proof –
  have \((P \equiv M) = ((P \equiv M) ; M)\)
  by (simp add: par-by-merge-def)
  also from assms have ... = \(((P \equiv M) ; M) ; J)\)
  by (simp add: Healthy-def RD2-def H2-def)
  also from assms have ... = \(((P \equiv M) ; M) ; J)\)
  by (simp add: seq-assoc)
  also from assms have ... = \(RD2(P \equiv M)\)
  by (simp add: RD2-def H2-def par-by-merge-def)
finally show \(\text{thesis}\)
by (simp add: Healthy-def)
qed

lemma \(SRD\)-par-by-merge:
assumes \(P \equiv SRD \land Q \equiv SRD \land M \equiv R1m \land M \equiv R2m \land M \equiv R3hm \land M \equiv RD1m \land M \equiv RD2\)
shows \((P \equiv M) \equiv SRD\)
by (rule SRD-intro, simp-all add: assms closure SRD-healths)
definition \(nmerge-rd0(N_0)\) where
[upred-defs]: \(N_0(M) = (\text{wait'} = u (s0-wait \lor s1-wait) \land \text{tr} < u \land \text{tr'})
\land (\exists \, s0-ok; s1-ok; s0-ok'; s0-wait; s1-wait; \text{wait} < s \land \text{wait'} \cdot M)\)
definition \(nmerge-rd1(N_1)\) where
[upred-defs]: \(N_1(M) = (s0-ok \land s1-ok) \land N_0(M)\)
definition \textit{nmerge-rd} \((N_R)\) where
\[ N_R(M) = ((\exists s_{st} < \cdot s_{v'} = u \cdot s_{v'} \triangleleft N_1(M)) \triangleleft s_{ok} \triangleleft (s_{tr} < \cdot s_{tr'})) \]

definition \textit{merge-rd1} \((M_1)\) where
\[ M_1(M) = (N_1(M) \;; \; H_R) \]

definition \textit{merge-rd} \((M_R)\) where
\[ M_R(M) = N_R(M) \;; \; H_R \]

abbreviation \textit{rdes-par} \((-\parallel R -\parallel-\parallel [85,0,86])\) where
\[ P \parallel_R M Q \equiv P \parallel_M R M Q \]

Healthiness condition for reactive design merge predicates

definition \textit{upred-defs}: \(R_{DM}(M) = (N_{R_{2m}}(N_R(M)) \;; \; II_{R_{2m}})\)

lemma \textit{nmerge-rd-is-R1m} [\textit{closure}]: \(N_R(M)\) is R1m
\[
\text{by (rel-blast)}
\]

lemma \textit{nmerge-rd-is-R2m} [\textit{closure}]: \(M\) is R2m \(\implies N_R(M)\) is R2m
\[
\text{by (metis Healthy-def' R2m-nmerge-rd)}
\]

lemma \textit{nmerge-rd-is-R3hm} [\textit{closure}]: \(N_R(M)\) is R3hm
\[
\text{by (rel-blast)}
\]

lemma \textit{nmerge-rd-is-RD1m} [\textit{closure}]: \(N_R(M)\) is RD1m
\[
\text{by (rel-blast)}
\]

lemma \textit{merge-rd-is-RD3}: \(M_R(M)\) is RD3
\[
\text{by (metis Healthy-Idempotent RD3-Idempotent RD3-def merge-rd-def)}
\]

lemma \textit{merge-rd-is-RD2}: \(M_R(M)\) is RD2
\[
\text{by (simp add: RD3-implies-RD2 merge-rd-is-RD3)}
\]

lemma \textit{par-rdes-NSRD} [\textit{closure}]:
\[
\text{assumes } P \text{ is SRD } Q \text{ is SRD } M \text{ is RDM } \text{ shows } P \parallel_R M Q \text{ is NSRD}
\]

proof –
\[
\text{have } (P \parallel_R M Q \;; \ R_R) \text{ is NSRD}
\]
\[
\text{by (rule NSRD-intro', simp-all add: SRD-healths closure assms)}
\]
\[
\text{(metis (no-types, lifting) Healthy-def R2-par-by-merge R2-seqr-closure R2m-nmerge-rd RDM-def SRD-healths(2) assms skip-srea-R2)}
\]
\[
\text{thus } ?\text{thesis}
\]
\[
\text{by (simp thesis: merge-rd-def par-by-merge-def seqr-assoc)}
\]

qed

lemma \textit{RDM-intro}:
\[
\text{assumes } M \text{ is R2m } s_{0-ok} \parallel M s_{1-ok} \parallel M s_{ok} \parallel M s_{ok'} \parallel M
\]
\[
\text{$s_{0-wait} \parallel M s_{1-wait} \parallel M s_{wait} \parallel M s_{wait'} \parallel M}$
\]
shows $M$ is RDM
using assms
by (simp add: Healthy-def RDM-def unrest unrest)

lemma RDM-unrests [unrest]:
assumes $M$ is RDM
shows $(\exists \, \text{ok} \cdot \text{ok}^\prime \cdot M)$ \quad $(\exists \, \text{ok} \cdot \text{ok}^\prime \cdot M)$ 
$\text{wait} < M$ \quad $M$ \quad $\text{wait}^\prime < M$
by (subt Healthy-if [OF assms, THEN sym], simp-all add: RDM-def unrest, rel-auto)+

lemma RDM-R1m [closure]: $M$ is RDM $\Longrightarrow M$ is R1m
by (metis (no-types, hide-lams) Healthy-def R1m-idem R2m-def RDM-def)

lemma RDM-R2m [closure]: $M$ is RDM $\Longrightarrow M$ 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 $(\exists \, \text{st} \cdot P)$ is R2m
proof
have $R2m(\exists \, \text{st} \cdot R2m \, P) = (\exists \, \text{st} \cdot R2m \, P)$
  by (rel-auto)
thus \text{thesis}
by (metis Healthy-def' assms)
qed

lemma parallel-RR-closed:
assumes $P$ is RR $Q$ is RR $M$ is R2m
shows $P \parallel_M Q$ is RR
by (rule RR-R2-closure, simp-all add: unrest assms RR-implies-R2 closure)

lemma parallel-ok-cases:
$\parallel (P, Q) :: M$ = 
$\parallel (P^t, Q^t) :: (M[true,true/$\text{ok}^\prime$/\text{ok}]) \lor$
$\parallel (P^f, Q^f) :: (M[false,true/$\text{ok}^\prime$/\text{ok}]) \lor$
$\parallel (P^f, Q^f) :: (M[true,false/$\text{ok}^\prime$/\text{ok}]) \lor$
$\parallel (P^t, Q^f) :: (M[false,false/$\text{ok}^\prime$/\text{ok}])$
proof
have $\parallel (P, Q) :: M = (\exists \, \text{ok}_0 \cdot (P, Q)[\text{ok}_0/\text{ok}]) \lor (M[\text{ok}_0/\text{ok}])$
  by (subst seqr-middle[of left-uvar ok], simp-all)
also have $\ldots = (\exists \, \text{ok}_0 \cdot \exists \, \text{ok}_1 \cdot (P, Q)[\text{ok}_0/\text{ok}][\text{ok}_1/\text{ok}]) \lor (M[\text{ok}_0/\text{ok}][\text{ok}_1/\text{ok}])$
  by (subst seqr-middle[of right-uvar ok], simp-all)
also have $\ldots = (\exists \, \text{ok}_0 \cdot \exists \, \text{ok}_1 \cdot (P, Q)[\text{ok}_0/\text{ok}][\text{ok}_1/\text{ok}]) \lor (M[\text{ok}_0,\text{ok}_1/\text{ok}])$
by (rel-auto robust)
also have $\ldots = (P^t, Q^t) :: (M[true,true/$\text{ok}^\prime$/\text{ok}]) \lor$
$\ldots = (P^f, Q^f) :: (M[false,true/$\text{ok}^\prime$/\text{ok}]) \lor$
$\ldots = (P^f, Q^f) :: (M[true,false/$\text{ok}^\prime$/\text{ok}]) \lor$
$\ldots = (P^t, Q^f) :: (M[false,false/$\text{ok}^\prime$/\text{ok}])$
by (simp all: true-alt-def THEN sym false-alt-def THEN sym disj-assoc
utp-pred-laws.sup.left-commute utp-pred-laws.sup-commute subst)
finally show \text{thesis} .
qed
lemma skip-srea-ok-f [usubst]:
\[ H_{\ell}^f = R1(\neg \text{sok}) \]
by (rel-auto)

lemma nmerge0-rd-unrest [unrest]:
\[ \text{s0-ok} \not\in N_0 \quad M \quad \text{s1-ok} \not\in N_0 \quad M \]
by (pred-auto)+

lemma parallel-assm-lemma:
assumes P is RD2
shows \[ \text{pre}_s \vdash (P \parallel \text{M}_{R(M)}(Q)) = ((\text{pre}_s \parallel P) \parallel \text{M}_{N_0}(\text{r}1(\text{true})(\text{cmt}_s \parallel Q)) \]
\[ \lor ((\text{cmt}_s \parallel P) \parallel \text{M}_{N_0}(\text{r}1(\text{true})(\text{pre}_s \parallel Q))) \]
proof –
have \[ \text{pre}_s \vdash (P \parallel \text{M}_{R(M)}(Q)) = \text{pre}_s \vdash ((P \parallel Q) \parallel \text{M}_{R(M)}) \]
by (simp add: par-by-merge-def)
also have \[ \ldots = ((P \parallel Q)[\text{true},\text{false}/\text{sok},\text{swait}] \parallel N_R \quad M \quad R1(\neg \text{sok})) \]
by (simp add: merge-rd-def usubst, rel-auto)
also have \[ \ldots = ((P[\text{true},\text{false}/\text{sok},\text{swait}] \parallel Q[\text{true},\text{false}/\text{sok},\text{swait}]) \parallel N_1(M) \quad R1(\neg \text{sok})) \]
by (rel-auto robust, (metis)+)
also have \[ \ldots = (((P[\text{true},\text{false}/\text{sok},\text{swait}]) \parallel Q[\text{true},\text{false}/\text{sok},\text{swait}]) \parallel ((N_1 \quad M)[\text{true},\text{true}/\text{s0-ok},\text{s1-ok}]) \]
\[ \lor (R1(\neg \text{sok})) \]
\[ \lor (((P[\text{true},\text{false}/\text{sok},\text{swait}]) \parallel Q[\text{true},\text{false}/\text{sok},\text{swait}]) \parallel ((N_1 \quad M)[\text{false},\text{true}/\text{s0-ok},\text{s1-ok}]) \]
\[ \lor (R1(\neg \text{sok})) \]
\[ \lor (((P[\text{true},\text{false}/\text{sok},\text{swait}]) \parallel Q[\text{true},\text{false}/\text{sok},\text{swait}]) \parallel ((N_1 \quad M)[\text{false},\text{false}/\text{s0-ok},\text{s1-ok}]) \]
\[ \lor (R1(\neg \text{sok})) \]
(is - = ((?C1 \lor ?C2 \lor ?C3 \lor ?C4))
by (subst parallel-ok-cases, subst-tac)
also have \[ \ldots = (?C2 \lor ?C3) \]
proof –
have \[ ?C1 = \text{false} \]
by (rel-auto)
moreover have \[ (?C4 \Rightarrow ?C3)\]
is \[ (?A \Rightarrow (?B \Rightarrow (?C \Rightarrow (?D))) \)
proof –
from assms have \[ ?P \Rightarrow ?P \]
by (metis RD2-def H2-equivalence Healthy-def)
hence \[ ?P \Rightarrow ?P \]
by (rel-auto)
have \[ ?A \Rightarrow ?C \]
using \[ P \]
by (rel-auto)
multiply have \[ ?B \Rightarrow ?D \]
by (rel-auto)
ultimately show \[ ?\text{thesis} \]
by (simp add:impl-seq-mono)
qed
ultimately show \[ ?\text{thesis} \]
by (simp add: substitution2)
qed
also have \[ \ldots = ( \]
\[ (((\text{pre}_s \parallel P) \parallel (\text{cmt}_s \parallel Q)) \parallel (N_0 \quad M \quad R1(\text{true}))) \]
\[ \lor (((\text{cmt}_s \parallel P) \parallel (\text{pre}_s \parallel Q)) \parallel (N_0 \quad M \quad R1(\text{true}))) \]
by (rel-auto, metis+)
also have \[ \ldots = ( \]

68
\[ ((\text{pre}_s \uparrow P) \parallel_{N_0} M ;; R1(\text{true}) \ (\text{cmt}_s \uparrow Q)) \lor \\
((\text{cmt}_s \uparrow P) \parallel_{N_0} M ;; R1(\text{true}) \ (\text{pre}_s \uparrow Q))) \]

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

finally show ?thesis.

qed

lemma \(\text{pre}_s\)-SRD:
assumes \(P\) is SRD
shows \(\text{pre}_s \uparrow P = (\neg_r \text{pre}_R(P))\)

proof –
  have \(\text{pre}_s \uparrow P = \text{pre}_s \uparrow R_s(\text{pre}_R P \vdash \text{peri}_R P \circ \text{post}_R P)\)
  by (simp add: SRD-reactive-tri-design assms)

also have \(\vdash R1(R2c(\neg \text{pre}_s \uparrow \text{pre}_R P))\)
  by (simp add: RHS-def asubst R3h-def pre)_s-design)

also have \(\vdash R1(R2c(\neg \text{pre}_R P))\)
  by (rel-auto)

also have \(\vdash (\neg_r \text{pre}_R P)\)
  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 \(\text{pre}_R(P \parallel_{M_R(M)} Q) = (\neg_r ((\neg_r \text{pre}_R(P)) \parallel_{N_0(M)} ;; R1(\text{true}) \ (\text{cmt}_R(Q)) \land \\
\neg_r (\text{cmt}_R(P) \parallel_{N_0(M)} ;; R1(\text{true}) \ (\neg_r \text{pre}_R(Q))))\)

(is \(?\text{lhs} = ?\text{rhs}\))

proof –
  have \(\text{pre}_R(P \parallel_{M_R(M)} Q) = (\neg_r (\text{pre}_s \uparrow P) \parallel_{N_0(M)} ;; R1 \text{true} (\text{cmt}_s \uparrow Q) \land \\
\neg_r (\text{cmt}_s \uparrow P) \parallel_{N_0(M)} ;; R1 \text{true} (\text{pre}_s \uparrow Q)))\)
  by (simp add: \text{pre}_R-def parallel-assm-lemma assms SRD-healths R1-conj rea-not-def[THEN sym])

also have \(\vdash ?\text{rhs}\)
  by (simp add: \text{pre}_s\)-SRD assms cmt_R-def Healthy-if closure unrest)

finally show ?thesis.

qed

lemma parallel-assm-unrest-wait’ [unrest]:
\[ [ P \text{ is SRD; } Q \text{ is SRD }] \implies \$\text{wait}’ \not \pre_{\text{pre}_R(P \parallel_{M_R(M)} Q) \quad \\
by (\text{simp add: parallel-assm, simp add: par-by-merge-def unrest})

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

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

lemma JL3: \((M_1 M)^I[false,\text{false}/\$0-ok,\$1-ok] = N_0(M) ;; R1(\text{true})\)
by (rel-blast)

lemma JL4: \((M_1 M)^I[true,\text{true}/\$0-ok,\$1-ok] = (\$\text{ok}’ \land N_0 M) ;; II_R^I\)
by (simp add: merge-rd1-def asubst nmerge-rd1-def unrest)

lemma parallel-commitment-lemma-1:
assumes $P$ is RD2

shows $\text{cnt}_s \vdash (P \parallel M_R(M) Q) = (\text{\texttt{\textbackslash sim add}}: \text{\texttt{\textbackslash subst parallel-ok-cases, subst-tac}})$

proof –

have $\text{cnt}_s \vdash (P \parallel M_R(M) Q) = (P[\text{\texttt{true}}][\text{\texttt{false}}][\text{\texttt{\$ok}}][\text{\texttt{\$wait}}] || (M_1(M))'[\text{\texttt{true}}][\text{\texttt{false}}][\text{\texttt{\$ok}}][\text{\texttt{\$wait}}])$
  by (simp add: rel-auto)

also have $(P[\text{\texttt{true}}][\text{\texttt{false}}][\text{\texttt{\$ok}}][\text{\texttt{\$wait}}] || (M_1(M))'$)
  by (simp add: par-by-merge-def)

also have $(P[\text{\texttt{true}}][\text{\texttt{false}}][\text{\texttt{\$ok}}][\text{\texttt{\$wait}}] || (M_1(M))'[\text{\texttt{true}}][\text{\texttt{false}}][\text{\texttt{\$ok}}][\text{\texttt{\$wait}}])$
  by (subt parallel-ok-cases, subst-tac)

also have $\text{\texttt{-}}$
  from assms have $'P_f' \Rightarrow P'f$
    by (metis RD2-def H2-equivalence Healthy-def)

hence $P; 'P_f' \Rightarrow P'f'$
  by (rel-auto)

have $'?A' \Rightarrow $?C'$ (is $'(?A :: ?B) \Rightarrow (?C :: ?D)'$)

proof –

have $'?A' \Rightarrow $?C'$
  using $P$ by (rel-auto)

thus $?\texttt{thesis}$
  by (simp add: impl-seq-mono)

qed

thus $?\texttt{thesis}$
  by (simp add: subsumption2)

qed

finally show $?\texttt{thesis}$
  by (simp add: par-by-merge-def JL4)

qed

lemma parallel-commitment-lemma-2:

assumes $P$ is RD2

shows $\text{cnt}_s \vdash (P \parallel M_R(M) Q) = (\text{\texttt{\textbackslash sim add}}: \text{\texttt{\textbackslash subst parallel-commitment-lemma-1 assms parallel-assm-lemma}})$

lemma parallel-commitment-lemma-3:

$M$ is $R1m \Rightarrow (\text{\texttt{\textbackslash sim add}}: \text{\texttt{\textbackslash subst parallel-commitment-lemma-1 assms parallel-assm-lemma}})$
Lemma parallel-commitment:
assumes P is SRD Q is SRD M is RDM
shows \( \text{cmd}_R(P \parallel M(R(M)) Q) = (\text{pre}_R(P \parallel M(R(M)) Q) \Rightarrow \text{cmd}_R(Q)) \)
by (simp add: parallel-commitment-lemma-2 parallel-commitment-lemma-3 Healthy-if assms \text{cmd}_R-def pre_c-SRD closure rea-impl-def disj-comm unrest)

Theorem parallel-reactive-design:
assumes P is SRD Q is SRD M is RDM
shows \( (P \parallel M(R(M)) Q) = \text{R}_s(\neg_r ((\neg_r \text{pre}_R(P)) \parallel N_0(M)) \cdot R_1(\text{true} \cdot \text{cmd}_R(Q)) \land \\
\neg_r (\text{cmd}_R(P) \parallel N_0(M)) \cdot R_1(\text{true} \cdot (\neg_r \text{pre}_R(Q)))) \Rightarrow \\
\text{cmd}_R(P) \parallel (\text{ok}_r \land N_0 M) \cdot H_R \cdot \text{cmd}_R(Q)) \) (is \( ?\text{lhs} = ?\text{rhs} \))
proof –
have \( (P \parallel M(R(M)) Q) = \text{R}_s(\text{pre}_R(P \parallel M(R(M)) Q) \parallel \text{cmd}_R(P \parallel M(R(M)) Q)) \)
by (metis Healthy-def NSRD-is-SRD SRD-as-reactive-design assms(1) assms(2) assms(3) par-rdes-NSRD)
also have \( \vdash ?\text{rhs} \)
by (simp add: parallel-assm parallel-commitment design-export-spec assms, rel-auto)
finally show \( ?\text{thesis} \).
qed

Lemma parallel-pericondition-lemma1:
(\( \text{ok}_r \land P \)) \cdot H_R(\text{true} / \text{ok}_r, \text{wait} \cdot) = (\exists \text{st} \cdot P)[\text{true, true} / \text{ok}_r, \text{wait} \cdot]
(is \( ?\text{lhs} = ?\text{rhs} \))
proof –
have \( ?\text{lhs} = (\text{ok}_r \land P) \cdot (\exists \text{st} \cdot H)[\text{true, true} / \text{ok}_r, \text{wait} \cdot] \)
by (rel-simp)
also have \( \vdash ?\text{rhs} \)
by (rel-auto)
finally show \( ?\text{thesis} \).
qed

Lemma parallel-pericondition-lemma2:
assumes M is RDM
shows \( (\exists \text{st} \cdot N_0(M))[\text{true, true} / \text{ok}_r, \text{wait} \cdot] = (\text{ok} - \text{wait} \lor \text{I - wait}) \land (\exists \text{st} \cdot M) \)
proof –
have \( (\exists \text{st} \cdot N_0(M))[\text{true, true} / \text{ok}_r, \text{wait} \cdot] = (\exists \text{st} \cdot (\text{ok} - \text{wait} \lor \text{I - wait}) \land \text{tr} \cdot \geq_u \text{tr} < \\
\land M) \)
by (simp add: usubst unrest nmerge-rd0-def ex-unrest Healthy-if R1m-def assms)
also have \( \vdash (\exists \text{st} \cdot (\text{ok} - \text{wait} \lor \text{I - wait}) \land M) \)
by (metis (no-types, hide-lams) Healthy-if R1m-def R1m-idem R2m-def RDM-def assms utp-pred-laws.inf-commute)
also have \( \vdash ((\text{ok} - \text{wait} \lor \text{I - wait}) \land (\exists \text{st} \cdot M)) \)
by (rel-auto)
finally show \( ?\text{thesis} \).
qed

Lemma parallel-pericondition-lemma3:
(\( (\text{ok} - \text{wait} \lor \text{I - wait}) \land (\exists \text{st} \cdot M) \)) = (((\text{ok} - \text{wait} \land \text{I - wait} \land (\exists \text{st} \cdot M)) \lor (\neg \text{ok} - \text{wait} \land \\
\text{I - wait} \land (\exists \text{st} \cdot M)) \lor (\text{ok} - \text{wait} \land \neg \text{I - wait} \land (\exists \text{st} \cdot M))) \)
by (rel-auto)

Lemma parallel-pericondition [rdes]:
fixes M :: (\( 's, 't::trace, 'a) \text{ rsp merge} \)
assumes $P$ is SRD $Q$ is SRD $M$ is RDM
shows $\text{peri}_R(P \parallel_{M,R} M, Q) = (\text{pre}_R(P \parallel_{M,R} M, Q) \Rightarrow \text{peri}_R(P) \parallel_{M,R} M, \text{peri}_R(Q))$
\begin{align*}
\text{proof} & \quad \text{have } \text{peri}_R(P \parallel_{M,R} M, Q) = \\
& \quad (\text{pre}_R(P \parallel_{M,R} M, Q) \Rightarrow \text{cmt}_R P \parallel_{(\exists \text{st} \cdot M) \parallel \text{peri}_R(Q), \text{cmt}_R Q}) \\
& \quad \text{by (simp add: peri-cmt-def parallel-commitment SRD-healths assms usuabl unrest assms)} \\
& \quad \text{also have } \ldots = (\text{pre}_R(P \parallel_{M,R} M, Q) \Rightarrow \text{cmt}_R P \parallel (\exists \text{st} \cdot N_0 M) \parallel \text{peri}_R(Q), \text{cmt}_R Q) \\
& \quad \text{by (simp add: parallel-pericondition-lemma1)} \\
& \quad \text{also have } \ldots = (\text{pre}_R(P \parallel_{M,R} M, Q) \Rightarrow \text{cmt}_R P \parallel (\exists \text{st} \cdot \text{peri}_R(Q), \text{cmt}_R Q) \\
& \quad \text{by (simp add: parallel-pericondition-lemma2 assms)} \\
& \quad \text{also have } \ldots = (\text{pre}_R(P \parallel_{M,R} M, Q) \Rightarrow (\exists \text{st} \cdot \text{peri}_R(Q), \text{cmt}_R Q) \\
& \quad \text{by (simp add: par-by-merge-all-def parallel-pericondition-lemma3 seqr-or-distr) \\
& \quad \text{also have } \ldots = (\text{pre}_R(P \parallel_{M,R} M, Q) \Rightarrow (\exists \text{st} \cdot \text{peri}_R(Q), \text{cmt}_R Q) \\
& \quad \text{by (simp add: seq-right-one-point-true seqr-right-one-point-false cmtR-def postR-def periR-def usuabl unrest assms)} \\
& \quad \text{also have } \ldots = (\text{pre}_R(P \parallel_{M,R} M, Q) \Rightarrow (\exists \text{st} \cdot \text{peri}_R(Q), \text{cmt}_R Q) \\
& \quad \text{by (simp add: par-by-merge-all-def)} \\
& \quad \text{finally show } \text{thesis} .
\end{align*}
\text{qed}

\text{lemma parallel-postcondition-lemma1:}

($\text{ok} \land \text{P}$) $:: \text{H}_R[\text{true, false}/\text{ok}, \text{wait}'] = \text{P}[\text{true, false}/\text{ok}, \text{wait}']$

\text{is } \text{lhs} = \text{rhs} \\
\text{proof} & \quad \text{have } \text{lhs} = (\text{ok} \land \text{P}) ;; \text{H}[\text{true, false}/\text{ok}, \text{wait}'] \\
& \quad \text{by (req-blast) \\
& \quad \text{also have } \ldots = \text{rhs} \\
& \quad \text{by (req-auto) \\
& \quad \text{finally show } \text{thesis} .
\end{align*}
\text{qed}

\text{lemma parallel-postcondition-lemma2:}

assumes $M$ is RDM
shows $\text{peri}_R(N_0(M)) [\text{true, false}/\text{ok}, \text{wait}'] = ((\neg \text{of}_0 - \text{wait} \land \neg 71 - \text{wait}) \land \text{M})$
\begin{align*}
\text{proof} & \quad \text{have } \ldots = ((\neg \text{of}_0 - \text{wait} \land \neg 71 - \text{wait}) \land \text{M}) \\
& \quad \text{by (simp add: usuabl unrest merge-rid0-def ex-intr lusty-if R1m-def assms)} \\
& \quad \text{also have } \ldots = ((\neg \text{of}_0 - \text{wait} \land \neg 71 - \text{wait}) \land \text{M}) \\
& \quad \text{by (metis Healthy-if R1m-def RDM-R1m assms upt-pred-laws.inf-commute)} \\
& \text{finally show } \text{thesis} .
\end{align*}
\text{qed}

\text{lemma parallel-postcondition} \quad \text{rdes}:

\text{fixes } M :: (s,t::trace',o) \text{ rdp 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 $\text{post}_R(P \parallel M R(M) Q) =$

$(\text{pre}_R(P \parallel M R M Q) \Rightarrow R \text{cnt}_R P \parallel (\text{ok}^\ast \land N_0 M) \parallel \text{II}[\text{true}_R \text{false}_R \text{ask}_R \text{wait}_R] \text{cnt}_R Q)$

by (simp add: post-cnt-def parallel-commitment assms usbst unrst SRD-health)

also have $\ldots = (\text{pre}_R(P \parallel M R M Q) \Rightarrow R \text{cnt}_R P \parallel (\neg S_0 \text{wait} \land \neg S_1 \text{wait} \land M) \text{cnt}_R Q)$

by (simp add: parallel-postcondition-lemma1 parallel-postcondition-lemma2 assms, simp add: up-t-pred-laws.inf-commute up-t-pred-laws.inf-left-commute)

also have $\ldots = (\text{pre}_R(P \parallel M R M Q) \Rightarrow R \text{post}_R(P) \parallel M \text{post}_R(Q))$

by (simp add: par-by-merge-alt-def seqr-right-one-point-false usbst unrst cnt_R-def post_R-def assms)

finally show \?thesis .

qed

lemma parallel-precondition-lemma:

fixes $M :: (\langle s, t :: \text{trace}, a \rangle \text{ tsp merge})$

assumes $P$ is NSRD $Q$ is NSRD $M$ is RDM

shows $(\neg R) \text{pre}_R(P) \parallel N_0(M) \parallel R (\text{true}) \parallel R \text{cnt}_R(Q) =$

$(\langle \neg R \text{pre}_R P \rangle \parallel M \parallel R (\text{true}) \parallel \text{peri}_R Q \parallel (\neg R \text{pre}_R P) \parallel M \parallel R (\text{true}) \parallel \text{post}_R(Q) = R (\text{true})$

proof –

have $(\langle \neg R \text{pre}_R P \rangle \parallel N_0(M) \parallel R (\text{true}) \parallel R \text{cnt}_R(Q) =$

$(\langle \neg R \text{pre}_R P \rangle \parallel N_0(M) \parallel R (\text{true}) \parallel \text{peri}_R(Q) \parallel (\neg R \text{pre}_R P) \parallel M \parallel (\text{true} \parallel \text{post}_R(Q)$)

by (simp add: wait’-cond-post-cnt)

also have $\ldots = (\langle \neg R \text{pre}_R P \rangle \parallel N_0(M) \parallel R (\text{true}) \parallel \text{peri}_R(Q) \parallel (\neg R \text{pre}_R P) \parallel M \parallel (\text{true} \parallel \text{post}_R(Q)$

by (simp add: var-by-merge-alt-def)

also have $\ldots = (\langle \neg R \text{pre}_R P \rangle \parallel N_0(M) \parallel R (\text{true}) \parallel \text{peri}_R(Q) \parallel (\neg R \text{pre}_R P) \parallel M \parallel (\text{true} \parallel \text{post}_R(Q)$

(is \?P \?Q ) \?P \?Q )

by (rel-auto)

thus \?thesis by simp

qed

also have $\ldots = (\langle \neg R \text{pre}_R P \rangle \parallel N_0(M) \parallel R (\text{true}) \parallel \text{peri}_R(Q) \parallel (\neg R \text{pre}_R P) \parallel M \parallel (\text{true} \parallel \text{post}_R(Q)$

by (simp add: cond-inter-var-split)

also have $\ldots = (\langle \neg R \text{pre}_R P \rangle \parallel N_0(M) \parallel R (\text{true}) \parallel \text{peri}_R(Q) \parallel (\neg R \text{pre}_R P) \parallel M \parallel (\text{true} \parallel \text{post}_R(Q)$

by (simp add: usbst unrst)

also have $\ldots = (\langle \neg R \text{pre}_R P \rangle \parallel N_0(M) \parallel R (\text{true}) \parallel \text{peri}_R(Q) \parallel (\neg R \text{pre}_R P) \parallel M \parallel (\text{true} \parallel \text{post}_R(Q)$

proof –

have $(\text{str}^* \geq \text{str}_0 \land M) = M$

using $\text{RDM-Red}(\text{OF assms}(\text{?)})$

by (simp add: Healthy-def R1m-def conj-comm)

thus \?thesis

by (simp add: nmerge-rd0-def unrst assms closure ex-unrst usbst)

qed

also have $\ldots = (\langle \neg R \text{pre}_R P \rangle \parallel N_0(M) \parallel R (\text{true}) \parallel \text{peri}_R(Q) \parallel (\neg R \text{pre}_R P) \parallel M \parallel (\text{true} \parallel \text{post}_R(Q)$

73
proof
  have \( ?P_1 = ([\lnot \rho \text{pre}_R P]_0 \land [\text{pre}_R Q]_1 \land \$v <^\rho = u \$v) ;; \ M ;; R1 \text{ true} \)
  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 \( ?P_2 = ([\lnot \rho \text{pre}_R P]_0 \land [\text{post}_R Q]_1 \land \$v <^\rho = u \$v) ;; \ (M \land \$\lnot \text{wait}') ;; R1 \text{ true} \)
    by (subst segr-bool-split[of left-var wait], simp-all add: usubst unrest assms closure conj-comm)
  hence 2: \( ?P_2 = ?Q_2 \)
    by (simp add: segr-left-one-point-true segr-left-one-point-false unrest usubst closure assms)
from 1 2 show \(?\text{thesis} \).
qed

lemma \text{swap-nmerge-rd0}:
  \( \text{swap}_m ;; N_0(M) = N_0(\text{swap}_m ;; \ M) \)
by (rel-auto, meson+)

lemma \text{SymMerge-nmerge-rd0 [closure]}:
  \( M \text{ is SymMerge} \implies N_0(M) \text{ is SymMerge} \)
by (rel-auto, meson+)

lemma \text{swap-merge-rd'}:
  \( \text{swap}_m ;; N_R(M) = N_R(\text{swap}_m ;; \ M) \)
by (rel-blast)

lemma \text{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 \text{SymMerge-merge-rd [closure]}:
  \( M \text{ is SymMerge} \implies M_R(M) \text{ is SymMerge} \)
by (simp add: Healthy-def swap-merge-rd)

lemma \text{nmerge-rd1-merge3}:
  assumes \( M \text{ is RDM} \)
  shows \( M^3(N_1(M)) = (\$\text{ok}' = u \ ($0 - \text{ok} \land \$1 - 0 - \text{ok} \land \$1 - 1 - \text{ok}) \land \$\text{wait}' = u \ ($0 - \text{wait} \land \$1 - 0 - \text{wait} \lor \$1 - 1 - \text{wait}) \land M^3(M) \)
proof –
  have \( M^3(N_1(M)) = M^3(\$\text{ok}' = u \ ($0 - \text{ok} \land \$1 - \text{ok}) \land \$\text{wait}' = u \ ($0 - \text{wait} \land \$1 - \text{wait}) \land \$\text{tr}' \leq u \$\text{tr}' \land (3 \{\$0 - \text{ok}, \$1 - \text{ok}, \$\text{ok}', \$0 - \text{wait}, \$1 - \text{wait}, \$\text{wait}' \cdot \text{RDM}(M)\}) \)
    by (simp add: nmerge-rd1-def nmerge-rd0-def assms Healthy-if)
  also have \( ... = M^3(\$\text{ok}' = u \ ($0 - \text{ok} \land \$1 - \text{ok}) \land \$\text{wait}' = u \ ($0 - \text{wait} \land \$1 - \text{wait}) \land \text{RDM}(M) \)
    by (rel-blast)
  also have \( ... = (\$\text{ok}' = u \ ($0 - \text{ok} \land \$1 - 0 - \text{ok} \land \$1 - 1 - \text{ok}) \land \$\text{wait}' = u \ ($0 - \text{wait} \land \$1 - 0 - \text{wait} \lor \$1 - 1 - \text{wait}) \land M^3(\text{RDM}(M)) \)
    by (rel-blast)

74
also have ... = (\$ok' = u (\$0-ok \land \$1-0-ok \land \$1-1-ok) \land \$wait' = u (\$0-wait \lor \$1-0-wait \lor \$1-1-wait) \land M3(M)) 
  by (simp add: assms Healthy-if)
finally show \textit{thesis} .
qed

lemma \textit{nmerge-nd-merge3}:
\[
M3(N_R(M)) = (\exists \$st_0 \cdot \$v_0 = u \$v_0) \land (\$wait_0 \lor M3(N, M) \land \$ok_0 \lor (\$tr_0 \leq u \$tr'))
\]
by (rel-blast)

lemma \textit{swap-merge-RDM-closed [closure]}:
assumes \textit{M is RDM}
shows \textit{swap}_m ;\; M \textit{is RDM}
proof
have \textit{RDM} (\textit{swap}_m ;\; \textit{RDM} (M)) = (\textit{swap}_m ;\; \textit{RDM} (M))
  by (rel-auto)
thus \textit{thesis}
  by (metis \textit{Healthy-def' assms})
qed

lemma \textit{parallel-precondition}:
fixes \textit{M} :: \{'s, t::trace, a\} \textit{rsp merge}
assumes \textit{P is NSRD} \textit{Q is NSRD} \textit{M is RDM}
shows \textit{pre}_R (P \parallel_{M(R(M))} Q) =
\[\neg r (\neg r \textit{pre}_R (P) \parallel_{M} ; R1(\textit{true}) \textit{peri}_R Q) \land 
\neg r (\neg r \textit{pre}_R (P) \parallel_{M} ; R1(\textit{true}) \textit{post}_R Q) \land 
\neg r (\neg r \textit{pre}_R Q) \parallel (\textit{swap}_m ; M) ; R1(\textit{true}) \textit{peri}_R P \land 
\neg r (\neg r \textit{pre}_R Q) \parallel (\textit{swap}_m ; M) ; R1(\textit{true}) \textit{post}_R P)\]
proof
have a: (\neg r \textit{pre}_R (P)) \parallel_{N_0(M)} ; R1(\textit{true}) cmt_R (Q) =
(\neg r \textit{pre}_R (P) \parallel_{M} ; R1(\textit{true}) \textit{peri}_R Q \lor (\neg r \textit{pre}_R (P) \parallel_{M} ; R1(\textit{true}) \textit{post}_R Q)\]
  by (simp add: \textit{parallel-precondition-lemma assms})

have b: (\neg r cmt_R P \parallel_{N_0 M} ; R1 true (\neg r \textit{pre}_R Q)) =
(\neg r (\neg r \textit{pre}_R (Q)) \parallel_{N_0(\textit{swap}_m ; M)} ; R1(\textit{true}) cmt_R (P))
  by (simp add: \textit{swap-nmerge-rl0[THEN sym]} \textit{seqr-assoc[THEN sym]} \textit{par-by-merge-def par-sep-swap})

have c: (\neg r \textit{pre}_R (Q)) \parallel_{N_0(\textit{swap}_m ; M)} ; R1(\textit{true}) cmt_R (P) =
((\neg r \textit{pre}_R Q) \parallel (\textit{swap}_m ; M) ; R1(\textit{true}) \textit{peri}_R P \lor (\neg r \textit{pre}_R Q) \parallel (\textit{swap}_m ; M) ; R1(\textit{true}) \textit{post}_R P)
  by (simp add: \textit{parallel-precondition-lemma closure assms})

show \textit{thesis}
  by (simp add: \textit{parallel-assm closure assms a b c, rel-auto})
qed

Weakest Parallel Precondition

definition \textit{wrR} ::
\[\langle t::trace, a\rangle \textit{hrel-rp} \Rightarrow \langle t::trace, a\rangle \textit{rp merge} \Rightarrow \langle t, a\rangle \textit{hrel-rp} \Rightarrow
\langle t', a\rangle \textit{hrel-rp} (- \textit{wr}_R (-) - [60,0.61] 61)\]
where [\textit{upred-defs}]: \textit{Q wr}_R (M) P = (\neg r ((\neg r P) \parallel_{M} ; R1(\textit{true}) Q))

lemma \textit{wrR-R1 [closure]}:
\[ M \text{ is } R1m \implies Q \text{ wr } R(M) \text{ is } R1 \]
by (simp add: \text{wrR-def closure})

**Lemma R2-rea-not:** \[ R2(\neg_r P) = (\neg_r R2(P)) \]
by (rel-auto)

**Lemma \text{wrR-R2-lemma}:**
assumes \[ P \text{ is } R2 \text{ Q is } R2 \text{ M is } R2m \]
shows \[ ((\neg_r P) \parallel M) \text{ is } R2 \]
proof
have \[ (\neg_r P) \parallel M \text{ is } R2 \]
by (simp add: \text{closure assms})
thus \[ ?\text{thesis} \]
by (simp add: \text{closure})
qed

**Lemma \text{wrR-R2} [\text{closure}]:**
assumes \[ P \text{ is } R2 \text{ Q is } R2 \text{ M is } R2m \]
shows \[ Q \text{ wr } R(M) \text{ is } R2 \]
proof
have \[ (\neg_r P) \parallel M \text{ is } R2 \]
by (simp add: \text{wrR-\text{R2-lemma} assms})
thus \[ ?\text{thesis} \]
by (simp add: \text{wrR-def \text{wrR-R2-lemma par-by-merge-seq-add closure}})
qed

**Lemma \text{wrR-RR} [\text{closure}]:**
assumes \[ P \text{ is } RR \text{ Q is } RR \text{ M is } RDM \]
shows \[ Q \text{ wr } R(M) \text{ is } RR \]
apply (rule \text{RR-intro})
apply (simp-all add: \text{unrest assms closure \text{wrR-def rpred})}
apply (metis \text{RR-implies-R2 assms (1) assms (2) assms (3) par-by-merge-seq-add rea-not-R2-closed \text{wrR-\text{R2-lemma}}})
done

**Lemma \text{wrR-RC} [\text{closure}]:**
assumes \[ P \text{ is } RR \text{ Q is } RR \text{ M is } RDM \]
shows \[ (Q \text{ wr } R(M)) \text{ is } RC \]
apply (rule \text{RC-intro})
apply (simp add: \text{closure assms})
apply (simp add: \text{wrR-def \text{rpred closure assms})}
apply (simp add: \text{par-by-merge-def seqr-assoc})
done

**Lemma \text{wpR-choice} [\text{wp}]:** \[ (P \lor Q) \text{ wr } R(M) \text{ is } R = (P \lor Q) \text{ wr } R(M) \text{ is } R \]
proof
have \[ (P \lor Q) \text{ wr } R(M) \text{ is } R \]
also have \[ \ldots = (\neg_r ((\neg_r R) \lor U0 \land P \lor U1 \land S_{\neg r} = u \lor v) \land U1 \land S_{\neg r} = u \lor v \lor (\neg_r R) \lor U0 \land Q \lor U1 \land S_{\neg r} = u \lor v) \land M \land true \]
by (simp add: \text{conj-disj-distr \text{utp-pred-laws} inf-sup-distrib2})
also have \[ \ldots = (\neg_r ((\neg_r R) \lor U0 \land P \lor U1 \land S_{\neg r} = u \lor v) \land M \land true) \lor \]
by (simp add: \text{conj-disj-distr \text{utp-pred-laws} inf-sup-distrib2})
also have \[ \ldots = (\neg_r ((\neg_r R) \lor U0 \land P \lor U1 \land S_{\neg r} = u \lor v) \land M \land true) \lor \]
by (simp add: \text{conj-disj-distr \text{utp-pred-laws} inf-sup-distrib2})
by (simp add: segr-or-distl)
also have \( \ldots = (P \, \text{wr}_R(M) \, R \land Q \, \text{wr}_R(M) \, R) \)
by (simp add: \text{wrR-def} \text{ par-by-merge-def})
finally show \(?\text{thesis}\).
qed

\textbf{lemma} \, \text{uppR-miracle} \, [\text{wp}]: \text{false} \, \text{wr}_R(M) \, P = \text{true}_r
\by (\text{simp add: \text{wrR-def}})

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

\textbf{lemma} \, \text{parallel-precondition-ur} \, [\text{rdes}]:
\textbf{assumes} \, P \, \text{is SRD} \, M = \text{RDM} \\
\textbf{shows} \, \text{pre}_R(P \, \| \, _M{R}(M) \, Q) \, = \, (\text{peri}_R(Q) \, \text{wr}_R(M) \, \text{pre}_R(P) \land \text{post}_R(Q) \, \text{wr}_R(M) \, \text{pre}_R(P) \land \\
\text{peri}_R(P) \, \text{wr}_R(\text{swap}_m \, :: \, M) \, \text{pre}_R(Q) \land \text{post}_R(P) \, \text{wr}_R(\text{swap}_m \, :: \, M) \, \text{pre}_R(Q))
\by (\text{simp add: \text{assms parallel-precondition \text{wrR-def}}})

\textbf{lemma} \, \text{parallel-rdes-def} \, [\text{rdes-def}]:
\textbf{assumes} \, P_1 \, \text{is RC} \, P_2 \, \text{is RR} \, P_3 \, \text{is RR} \, Q_1 \, \text{is RC} \, Q_2 \, \text{is RR} \, Q_3 \, \text{is RR} \\
\$st^+ \cdot \, P_2 \, \$st^+ \cdot \, Q_2 \\
M = \text{RDM} \\
\textbf{shows} \, R_1(P_1 \, \Rightarrow_r \, P_2 \, \& \, P_3) \, \| \, _M{R}(M) \, R_2(Q_1 \, \Rightarrow \, Q_2 \, \& \, Q_3) = \\
R_1(((Q_1 \, \Rightarrow_r \, Q_2) \, \text{wr}_R(M) \, P_1 \, \& \, (Q_1 \, \Rightarrow_r \, Q_3) \, \text{wr}_R(M) \, P_1 \, \& \\
(P_1 \, \Rightarrow_r \, P_2) \, \text{wr}_R(\text{swap}_m \, :: \, M) \, Q_1 \, \& \, (P_1 \, \Rightarrow_r \, P_3) \, \text{wr}_R(\text{swap}_m \, :: \, M) \, Q_1) \, \Rightarrow \\
((P_1 \, \Rightarrow_r \, P_2) \, \| \, M \, (Q_1 \, \Rightarrow_r \, Q_2) \, \& \, (P_1 \, \Rightarrow_r \, P_3) \, \| \, M \, (Q_1 \, \Rightarrow_r \, Q_3)) \, \| \\
((P_1 \, \Rightarrow_r \, P_3) \, \| \, M \, (Q_1 \, \Rightarrow_r \, Q_3))) \, (\text{is \, ?lhs} = \, ?\text{rhs})
\proof
\begin{itemize}
  \item \, \text{have \, ?lhs} = \, R_2 \, (\text{pre}_R \, \text{?lhs} \, \& \, \text{peri}_R \, \text{?lhs} \, \& \, \text{post}_R \, \text{?lhs})
  \by (\text{simp add: \text{SRD-reactive-tri-design} \text{assms} \text{ closure})}
  \item \, \text{also have \, \ldots} = \, ?\text{rhs}
  \by (\text{simp add: \text{rdes closure} \text{ unrest} \text{ assms}, \text{ rel-auto})}
\end{itemize}
\textbf{finally show} \, ?\text{thesis}.
\qed

\textbf{lemma} \, \text{Miracle-parallel-left-zero}:
\textbf{assumes} \, P \, \text{is SRD} \, M = \text{RDM} \\
\textbf{shows} \, \text{Miracle} \, \| \, _R{M} \, P = \, \text{Miracle}
\proof
\begin{itemize}
  \item \, \text{have \, \text{pre}_R(M \, \| \, _R{M} \, P) = \, \text{true}_r}
    \by (\text{simp add: \text{parallel-assm} \text{ wait'-cond-idem} \text{ rdes closure assms})}
  \item \, \text{moreover hence} \, \text{cmt}_R(M \, \| \, _R{M} \, P) = \, \text{false}
    \by (\text{simp add: \text{rdes closure} \text{ wait'-cond-idem} \text{ SRD-healths assms})}
  \item \, \text{ultimately have} \, \text{Miracle} \, \| \, _R{M} \, P = \, R_4(\text{true}_r \, \& \, \text{false})
    \by (\text{metis \text{NSRD-iff} \text{SRD-reactive-design-alt assms} \text{ par-by-merge NSRD} \text{ srdes-theory-continuous.weak.top-closed})}
  \item \, \text{thus \, ?thesis}
    \by (\text{simp add: \text{Miracle-def} \text{ R1-design-R1-pre})}
\end{itemize}
\qed

\textbf{lemma} \, \text{Miracle-parallel-right-zero}:
\textbf{assumes} \, P \, \text{is SRD} \, M = \text{RDM} \\
\textbf{shows} \, P \, \| \, _R{M} \, \text{Miracle} = \, \text{Miracle}
\proof
\begin{itemize}
  \item \, \text{have \, \text{pre}_R(P \, \| \, _R{M} \, \text{Miracle}) = \, \text{true}_r}
\end{itemize}
\qed
8.1 Example basic merge

definition BasicMerge :: \((\ell', 't::trace, unit) \mathit{rsp})\) merge \(N_B\) where

\[\text{upred-defs}: \text{BasicMerge} = (\exists \ell. \ell' \leq \ell' \land \ell' = u \cdot \ell - \ell' \land \ell - \ell' = u \cdot (1 - \ell' - \ell)\]

abbreviation rbasic-par \((\cdot ||B - [85,86] 85)\) where

\[P ||B Q \equiv P ||M_{B}(N_B) Q\]

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 \(\exists k' \equiv P \cdot \text{wait'} \equiv P \cdot \text{ok'} \equiv Q \cdot \text{wait'} \equiv Q \cdot P \) is \(R2\)

shows \(P ||N_B Q = (\exists \ell \cdot \text{P} \land (\exists \ell' \cdot \text{Q}) \land \ell' = u \cdot \text{st}\)

using assms

proof –

have \(P \cdot (\exists \ell \cdot \text{ok'} \cdot \text{wait'}) \cdot \text{R2}(P) = P \cdot (\exists \ell' \equiv -)\)

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

have \(Q \cdot (\exists \ell \cdot \text{ok'} \cdot \text{wait'}) \cdot \text{R2}(Q) = Q \cdot (\exists \ell' \equiv -)\)

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

have \(\exists \ell' \equiv P \cdot Q = \text{R2}()\)

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 ::

\((\ell', 't::trace, 'a) \text{hrel-rsp} \Rightarrow (\ell', 't', 'a) \text{hrel-rsp} \Rightarrow (\ell', 't', 'a) \text{hrel-rsp}\) (infhr ||R 85)

where \(\text{upred-defs}: P ||R Q = R_s((\text{pre}_R(P) \land \text{pre}_R(Q)) = (\text{cnt}_R(P) \land \text{cnt}_R(Q)))\)

lemma RHS-design-par:

assumes \(\exists k' \equiv P_1 \cdot \text{ok'} \equiv P_2\)

shows \(R_s(P_1 \equiv Q_1) ||R R_s(P_2 \equiv Q_2) = R_s((P_1 \equiv P_2) \equiv (Q_1 \equiv Q_2))\)

proof –

have \(R_s(P_1 \equiv Q_1) ||R R_s(P_2 \equiv Q_2) = R_s(P_1 \equiv true . \text{false} . \text{ok} P_2 \equiv Q_1 \equiv Q_2 \equiv true . \text{false} . \text{ok} P_2 \equiv Q_2)\)

by (simp add: RHS-design-ok-wait)

qed
also from **assms**

have ... =

\[ \text{R}_s((R2c(P_1) \land R2c(P_2)) \vdash (R1 (R2c (P_1 \Rightarrow Q_1)) \land R1 (R2c (P_2 \Rightarrow Q_2)))) \]

apply (simp add: rea-design-par-def rea-pre-RHS-design rea-cmt-RHS-design unsubst unrest assms)
apply (rule cong[of \text{R}_s \text{R}_s], simp)
using assms apply (rel-auto)
done

also have ... =

\[ \text{R}_s((P_1 \land P_2) \vdash (R1 (R2s (P_1 \Rightarrow Q_1)) \land R1 (R2s (P_2 \Rightarrow Q_2)))) \]

by (metis (no-types, hide-lams) R1-R2s-R2c R1-conj R1-design-R1-pre RHS-design-ok-wait)

also have ... =

\[ \text{R}_s((P_1 \land P_2) \vdash ((P_1 \Rightarrow Q_1) \land (P_2 \Rightarrow Q_2))) \]

by (simp (no-types, lifting) R1-conj R2s-conj RHS-design-export-R1 RHS-design-export-R2s)

also have ... = \text{R}_s((P_1 \land P_2) \vdash (Q_1 \land Q_2))

by (rule cong[of \text{R}_s \text{R}_s], simp, rel-auto)

finally show \( ? \text{thesis} \).

qed

**lemma RHS-tri-design-par:**

assumes \( \text{Sok}' \notin P_1 \text{Sok}' \notin P_2 \)

shows \( \text{R}_s(P_1 \vdash Q_1 \circ R_1) \parallel_R \text{R}_s(P_2 \vdash Q_2 \circ R_2) = \text{R}_s((P_1 \land P_2) \vdash (Q_1 \land Q_2) \circ (R_1 \land R_2)) \)

by (simp add: RHS-design-par assms unrest wait’t-cond-conj-exchange)

**lemma RHS-tri-design-par-RR [rdes-def]:**

assumes \( P_1 \text{ is RR } P_2 \text{ is RR} \)

shows \( \text{R}_s(P_1 \vdash Q_1 \circ R_1) \parallel_R \text{R}_s(P_2 \vdash Q_2 \circ R_2) = \text{R}_s((P_1 \land P_2) \vdash (Q_1 \land Q_2) \circ (R_1 \land R_2)) \)

by (simp add: RHS-tri-design-par unrest assms)

**lemma RHS-comp-assoc:**

assumes \( P \text{ is NSRD } Q \text{ is NSRD } R \text{ is NSRD} \)

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** Projective :: ('s, 't::trace, 'a) hrel-rsp \( \Rightarrow ('s, 't, 'a) hrel-rsp \) where

[upred-defs]: Projective(P) = P \parallel_R \text{R}_s(true \vdash true \circ (\$tr <_{u} \$tr'))

**lemma** Projective-RHS-design-form:

assumes \( \text{Sok}' \notin P \text{ Sok}' \notin Q \text{ Sok}' \notin R \)
shows \( \text{Productive}(\mathbf{R}_s(P \vdash Q \circ R)) = \mathbf{R}_s(P \vdash Q \circ (R \land \text{\$tr <}_u \text{\$tr}^-)) \)

using \text{assms by (simp add: Productive-def RHS-tri-design-par unrest)}

\textbf{lemma} Productive-form:

\[ \text{Productive} (\text{SRD}(P)) = \mathbf{R}_s(\text{peri}_R(P) \circ (\text{post}_R(P) \land \text{\$tr <}_u \text{\$tr}^-)) \]

\textbf{proof} –

\textbf{have} \[ \text{Productive} (\text{SRD}(P)) = \mathbf{R}_s(\text{peri}_R(P) \circ (\text{post}_R(P) \land \text{\$tr <}_u \text{\$tr}^-)) \]

\text{by (simp add: Productive-def SRD-as-reactive-tri-design)}

\textbf{also have} \[ \ldots = \mathbf{R}_s(\text{peri}_R(P) \circ (\text{post}_R(P) \land \text{\$tr <}_u \text{\$tr}^-)) \]

\text{by (simp add: RHS-tri-design-par unrest)}

\textbf{finally show} \ ?thesis

\textbf{qed}

A reactive design is productive provided that the postcondition, under the precondition, strictly increases the trace.

\textbf{lemma} Productive-intro:

\textbf{assumes} \( P \) is SRD \( (\text{\$tr <}_u \text{\$tr}^-) \subseteq (\text{peri}_R(P) \land \text{post}_R(P)) \)

\textbf{shows} \( P \) is Productive

\textbf{proof} –

\textbf{have} \[ P : \mathbf{R}_s(\text{peri}_R(P) \circ (\text{post}_R(P) \land \text{\$tr <}_u \text{\$tr}^-)) = P \]

\textbf{proof} –

\textbf{have} \[ \mathbf{R}_s(\text{peri}_R(P) \circ (\text{post}_R(P) \land \text{\$tr <}_u \text{\$tr}^-)) = \mathbf{R}_s(\text{peri}_R(P) \circ (\text{peri}_R(P) \circ (\text{peri}_R(P) \circ (\text{peri}_R(P)) \circ (\text{peri}_R(P) \circ (\text{peri}_R(P)) \circ (\text{peri}_R(P) \circ (\text{peri}_R(P)))))))) \]

\text{by (metis (no-types, hide-lams) design-export-pre wait'-cond-conj-exchange wait'-cond-ident)}

\textbf{also have} \[ \ldots = \mathbf{R}_s(\text{peri}_R(P) \circ (\text{peri}_R(P) \circ (\text{peri}_R(P) \circ (\text{peri}_R(P) \circ (\text{peri}_R(P)) \circ (\text{peri}_R(P) \circ (\text{peri}_R(P)) \circ (\text{peri}_R(P) \circ (\text{peri}_R(P)))))))) \]

\text{by (metis (no-types, hide-lams) design-export-pre wait'-cond-conj-exchange wait'-cond-ident)}

\textbf{finally show} \ ?thesis

\text{by (simp add: SRD-reactive-tri-design \text{assms(1)})}

\textbf{qed}

\textbf{thus} \ ?thesis

\text{by (metis \text{Healthy-def RHS-tri-design-par Productive-def ok' pre-unrest unrest-true utp-pred-laws.inf-right-ident utp-pred-laws.inf-top-right)}

\textbf{qed}

\textbf{lemma} Productive-refines-tr-increase:

\textbf{assumes} \( P \) is SRD \( P \) is Productive \( \text{\$wait' \notin \text{peri}_R(P)} \)

\textbf{shows} \( (\text{\$tr <}_u \text{\$tr}^-) \subseteq (\text{peri}_R(P) \land \text{post}_R(P)) \)

\textbf{proof} –

\textbf{have} \[ \text{\$tr <}_u \text{\$tr}^- = \text{\$tr <}_u \text{\$tr}^- \]

\text{by (metis \text{Healthy-def Productive-form \text{assms}(1) \text{assms}(2))}

\textbf{also have} \[ \ldots = R1(R2c(\text{peri}_R(P) \Rightarrow (\text{post}_R(P) \land \text{\$tr <}_u \text{\$tr}^-))) \]

\text{by (simp add: rea-post-RHS-design unrest usubst \text{assms, rel-auto})}

\textbf{also have} \[ \ldots = R1((\text{peri}_R(P) \Rightarrow (\text{post}_R(P) \land \text{\$tr <}_u \text{\$tr}^-))) \]

\text{by (simp add: R2c-impl R2c-preR R2c-postR R2c-and R2c-tr-less-tr' \text{assms})}

\textbf{also have} \[ (\text{\$tr <}_u \text{\$tr}^-) \subseteq (\text{peri}_R(P) \land \ldots) \]

\text{by (rel-auto)}

\textbf{finally show} \ ?thesis

\textbf{qed}

\textbf{lemma} Continuous-Productive [closure]: Continuous Productive

\text{by (simp add: Continuous-def Productive-def, rel-auto)}
9.2 Reactive design calculations

**Lemma** `preR-Productive [rdes]`:
assumes P is SRD
shows `preR(Productive(P)) = preR(P)`
proof –
  have `preR(Productive(P)) = preR(R_2c(peri(P) \leadsto P))` 
    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 `periR(Productive(P)) = periR(P)`
proof –
  have `periR(Productive(P)) = periR(R_2c(peri(P) \leadsto P))` 
    by (metis Healthy-def NSRD-is-SRD Productive-form assms)
also have `... = R1 (R2c (preR P \rightarrow periR P))` 
    by (simp add; rea-peri-RHS-design usubst unrest R2c-not assms)
also have `... = (preR P \Rightarrow periR P)` 
    by (simp add; R1-rea-impl R2c-rea-impl R2c-preR R2c-peri-SRD 
        R1-peri-SRD assms)
finally show `\thesis`
  by (simp add; SRD-peri-under-pre assms)
qed

**Lemma** `postR-Productive [rdes]`:
assumes P is NSRD
shows `postR(Productive(P)) = (preR P \Rightarrow postR P \land \str < \str')`
proof –
  have `postR(Productive(P)) = postR(R_2c(peri(P) \leadsto P))` 
    by (metis Healthy-def NSRD-is-SRD Productive-form assms)
also have `... = R1 (R2c (preR P \rightarrow periR P))` 
    by (simp add; rea-post-RHS-design usubst unrest assms)
also have `... = (preR P \Rightarrow periR P)` 
    by (simp add; R1-rea-impl R2c-rea-impl R2c-preR R2c-post-SRD 
        R1-post-SRD assms)
finally show `\thesis`
  by (simp add; SRD-post-under-pre assms)
qed

**Lemma** `preR-frame-seq-export`:
assumes P is NSRD P is Productive Q is NSRD
shows `(preR P \land (preR P \land postR P) \land Q) = (preR P \land (postR P \land Q))`
proof –
  have `(preR P \land (postR P \land Q)) = (preR P \land ((preR P \Rightarrow postR P) \land Q))` 
    by (simp add; SRD-post-under-pre assms)
also have `... = (preR P \land ((\neg (\neg preR P) \land Q) \lor (preR P \Rightarrow R1(postR P) \land Q)))` 
    by (simp add; NSRD-is-SRD R1-post-SRD assms)
also have `... = (preR P \land (((\neg preR P) \land Q) \lor (preR P \land postR P) \land Q))` 
    by (simp add; R1-rea-impl R2c-rea-impl R2c-preR R2c-and R1-extend-conj' R2c-post-SRD 
        R1-post-SRD assms)
finally show `\thesis`
  by (simp add; SRD-post-under-pre assms)
qed

81
9.3 Closure laws

**Lemma** Productive-rdes-intro:

assumes \( \langle \text{str} <_u \text{str} \rangle \subseteq R \text{ok} \vdash P \text{ok} \vdash Q \text{ok} \vdash R \text{wait} \vdash P \text{wait} \vdash P \)

shows \( R_u (P \vdash Q \circ R) \) is Productive

**Proof** (rule Productive-intro)

show \( R_u (P \vdash Q \circ R) \) is SRD

by (simp add: RHS-tri-design-is-SRD assms)

from assms(1) show \( \langle \text{str} \rangle \subseteq (\text{pre}_R (R_u (P \vdash Q \circ R)) \land \text{post}_R (R_u (P \vdash Q \circ R))) \)

apply (simp add: rea-pre-RHS-design rea-post-RHS-design usubst assms unrest)

using assms(1) apply (rel-auto)

apply fastforce

done

show \( \text{wait} \vdash \text{pre}_R (R_u (P \vdash Q \circ R)) \)

by (simp add: rea-pre-RHS-design rea-post-RHS-design usubst R1-def R2c-def R2s-def assms unrest)

qed

We use the \( R_4 \) healthiness condition to characterise that the postcondition must extend the trace for a reactive design to be productive.

**Lemma** Productive-rdes-RR-intro:

assumes \( P \) is \( RR \) \( Q \) is \( RR \) \( R \) is \( RR \) \( R \) is \( R_4 \)

shows \( R_u (P \vdash Q \circ R) \) 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 \( Q \) is SRD \( Q \) is Productive

shows \( P \cap Q \) is Productive

**Proof**

have \( P \cap Q = \)

\[ R_u (\text{pre}_R (P) \vdash \text{peri}_R (P) \circ (\text{post}_R (P) \land \text{str} <_u \text{str}')) \cap R_u (\text{pre}_R (Q) \vdash \text{peri}_R (Q) \circ (\text{post}_R (Q) \land \text{str} <_u \text{str}')) \]

by (metis Healthy-if Productive-form assms)

also have \( \ldots = R_u ((\text{pre}_R P \land \text{pre}_R Q) \vdash (\text{peri}_R P \lor \text{peri}_R Q) \circ ((\text{post}_R P \land \text{str} <_u \text{str}) \lor (\text{post}_R Q \land \text{str} <_u \text{str})) \)

by (simp add: RHS-tri-design-choice)

also have \( \ldots = R_u ((\text{pre}_R P \land \text{pre}_R Q) \vdash (\text{peri}_R P \land \text{peri}_R Q) \circ ((\text{post}_R P) \lor (\text{post}_R Q) \land \text{str} <_u \text{str}) \)

82
by (rule cong[of R, 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 ▷ Q :=
proof –
  have P ◁ b ▷ Q =
    R,(pre_R(P) ⊨ peri_R(P) ◁ (post_R(P) ∧ $tr < u $tr′)) ◁ b ▷ R,(peri_R(Q) ◁ (post_R(Q)
    ∧ $tr < u $tr′))
    by (metis Healthy-if Productive-form assms)
  also have ... :=
    by (simp add: cond-srea-form)
  also have ... :=
    by (rule cong[of R, simp, rel-auto])
  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 :=
    R,(pre_R(P) ⊨ peri_R(P) ◁ (post_R(P) ∧ $tr < u $tr′)) ;
    R,(peri_R(Q) ◁ (post_R(Q)))
    by (metis Healthy-if NSRD-is-SRD SRD-reactive-tri-design Productive-form assms)
  also have ... :=
    by (simp add: cond-srea-form)
  also have ... :=
    by (rule cong[of R, simp, rel-auto])
  finally show ?thesis .
qed

lemma Productive-seq-2 [closure]:
  assumes P is NSRD Q is NSRD Q is Productive
shows \( P \vdash Q \) is Productive

proof –

have \( P \vdash Q = R_s(\text{pre}_R(P) \triangleright \text{peri}_R(P) \triangleright (\text{post}_R(P))) \vdash R_s(\text{pre}_R(Q) \triangleright \text{peri}_R(Q) \triangleright (\text{post}_R(Q) \wedge \#\text{tr} < u \#\text{tr}'))\)

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

also have \( \ldots = R_s((\text{pre}_R P \wedge \text{post}_R P \triangleright \text{peri}_R \ldots \wedge \text{peri}_R Q) \triangleright (\text{post}_R P \vdash (\text{post}_R Q \wedge \#\text{tr} - u \#\text{tr}))\)

by (simp add: RHS-tri-design-composition-wp rpred unrest closure assms wp NSRD-neg-pre-left-zero SRD-healhys ex-unrest wp-rea-def disj-upred-def)

also have \( \ldots = R_s((\text{pre}_R P \wedge \text{post}_R P \triangleright \text{peri}_R P \wedge (\text{post}_R P \vdash \text{peri}_R Q)) \triangleright (\text{post}_R P \vdash (\text{post}_R Q \wedge \#\text{tr} - u \#\text{tr} - u \#\text{tr}))\)

proof –

have \( (R_1(\text{post}_R P) \vdash (\text{post}_R Q \wedge \#\text{tr} - u \#\text{tr} - u \#\text{tr} - u \#\text{tr}) = (R_1(\text{post}_R P) \vdash (\text{post}_R Q \wedge \#\text{tr} - u \#\text{tr} - u \#\text{tr} - u \#\text{tr}))\)

by (rel-auto)

thus \( \#\text{thesis} \)

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

qed

also have \( \ldots \) is Productive

by (rule Productive-rdes-intro, simp-all add: unrest assms closure wp-rea-def)

finally show \( \#\text{thesis} \).

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 \text{ucard} function that provides this.

class size-trace = trace + size +

assumes

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

- These axioms may be stronger than necessary. In particular, \( 0 < ?s \implies 0 < \#\text{u}(?s) \) requires that a non-empty trace have a positive size. But this may not be the case with all trace models and is possibly more restrictive than necessary. In future we will explore weakening.

begin

lemma size-mono: \( s \leq t \implies \#\text{size}(s) \leq \#\text{size}(t) \)

by (metis le-add1 local.diff-add-cancel-left' local.size-plus)

lemma size-strict-mono: \( s < t \implies \#\text{size}(s) < \#\text{size}(t) \)

by (metis cancel-ab-semigroup-add-class.add-diff-cancel-left' local.diff-add-cancel-left' local.less-iff local.minus-gr-zero-iff local.size-nzero local.size-plus-zero-less-diff)

lemma trace-strict-prefixE: \( xs < ys \implies (\forall zs. \[ ys = xs + zs; \#\text{size}(zs) > 0 \] \implies \#\text{thesis} \) \implies \#\text{thesis}
by (metis local.diff-add-cancel-left' local.less_iff local.minus-gr-zero_iff local.size-nzero)

lemma size-minus-trace: \( y \leq x \Rightarrow \text{size}(x - y) = \text{size}(x) - \text{size}(y) \)
by (metis diff-add-inverse local.diff-add-cancel-left' local.size-plus)
end

Both natural numbers and lists are measurable trace algebras.

instance nat :: size-trace
by (intro-classes, simp-all)

instance list :: (type) size-trace
by (intro-classes, simp-all add: zero-list-def less-list-def' plus-list-def prefix-length-less)

syntax
\texttt{-usize} :: logic \Rightarrow logic (size_u('\texttt{-}'))

translations
\texttt{size_u(t)} == \texttt{CONST uop CONST size t}

10.2 Guardedness

definition \texttt{gvrt} :: (('t::size-trace,'a) \texttt{rp} \times ('t,'a) \texttt{rp}) \texttt{chain} where
\texttt{[upred-defs]: \texttt{gvrt}(n) \equiv ($tr \leq_u \texttt{tr'} \land size_u(\&\texttt{tt}) <_u <n$)}

lemma \texttt{gvrt-chain}: \texttt{chain \texttt{gvrt}}
apply (simp add: chain-def, safe)
apply (rel-simp)
apply (rel-simp)+
done

lemma \texttt{gvrt-limit}: \( \bigwedge (\text{range \texttt{gvrt}}) = ($tr \leq_u \texttt{tr'}$) \)
by (rel-auto)

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

lemma \texttt{GuardedI}: \( \bigwedge X \, (F(X) \land \texttt{gvrt}(n+1)) = (F(X \land \texttt{gvrt}(n)) \land \texttt{gvrt}(n+1)) \) \Rightarrow Guarded F
by (simp add: Guarded-def)

Guarded reactive designs yield unique fixed-points.

theorem \texttt{guarded-fp-uniq}: 
assumes \( \texttt{mono F} 
F \in [\texttt{id}]_H \Rightarrow [\texttt{SRD}]_H \texttt{Guarded F} \)
shows \( \mu F = \nu F \)
proof –
have \texttt{constr F \texttt{gvrt}}
using \texttt{assms}
by (auto simp add: constr-def gvirt-chain Guarded-def tcontr-alt-def')
hence \( ($tr \leq_u \texttt{tr'} \land \mu F) = ($tr \leq_u \texttt{tr'} \land \nu F$)
apply (rule constr-fp-uniq)
apply (simp add: \texttt{assms})
using \texttt{gvrt-limit} apply blast
done
moreover have \( ($tr \leq_u \texttt{tr'} \land \mu F) = \mu F \)
proof –
have $\mu F = R1$
  by (rule SRD-healths(1), rule Healthy-mu, simp-all: assms)
thus $\vdash \text{thesis}$
  by (metis Healthy-def R1-def conj-comm)
qed
moreover have ($\text{str} \leq_u \text{str'} \land \nu F) = \nu F$
proof –
  have \nu F = R1
    by (rule SRD-healths(1), rule Healthy-nu, simp-all: assms)
  thus $\vdash \text{thesis}$
  by (metis Healthy-def R1-def conj-comm)
qed
ultimately show $\vdash \text{thesis}$
  by (simp)
qed

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

lemma UINF-Guarded [closure]:
  assumes $\bigwedge P. P \in A \Rightarrow \text{Guarded} P$
  shows $\text{Guarded} (\lambda X. \bigwedge P \in A. P(X))$
proof (rule GuardedI)
  fix $X n$
  have $\bigwedge Y. ((\bigwedge P \in A \cdot P Y) \land \text{gvr}(n+1)) = ((\bigwedge P \in A \cdot (P Y \land \text{gvr}(n+1))) \land \text{gvr}(n+1))$
proof –
  fix $Y$
  let $?lhs = (\bigwedge P \in A \cdot P Y) \land \text{gvr}(n+1)$ \and $?rhs = (\bigwedge P \in A \cdot (P Y \land \text{gvr}(n+1))) \land \text{gvr}(n+1)$
  have \alpha: $?lhs[\text{false}/\text{ok}] = ?rhs[\text{false}/\text{ok}]$
    by (rel-auto)
  have \beta: $?lhs[\text{true}/\text{ok}] [\text{true}/\text{wait}] = ?rhs[\text{true}/\text{ok}] [\text{true}/\text{wait}]$
    by (rel-auto)
  have \gamma: $?lhs[\text{true}/\text{ok}] [\text{false}/\text{wait}] = ?rhs[\text{true}/\text{ok}] [\text{false}/\text{wait}]$
    by (rel-auto)
  show $?lhs = ?rhs$
    using \alpha \beta \gamma
    by (rule-tac bool-eq-splitI[of in-var ok], simp, rule-tac bool-eq-splitI[of in-var wait], simp-all)
qed
moreover have $((\bigwedge P \in A \cdot (P X \land \text{gvr}(n+1))) \land \text{gvr}(n+1)) = ((\bigwedge P \in A \cdot (P X \land \text{gvr}(n)) \land \text{gvr}(n+1))) \land \text{gvr}(n+1))$
proof –
  have $\bigwedge P \in A \cdot (P X \land \text{gvr}(n+1))) = (\bigwedge P \in A \cdot (P X \land \text{gvr}(n)) \land \text{gvr}(n+1))$
proof (rule UINF-cong)
  fix $P$ assume $P \in A$
  thus $(P X \land \text{gvr}(n+1))) = (P X \land \text{gvr}(n)) \land \text{gvr}(n+1))$
    using Guarded-def assms by blast
  qed
  thus $\vdash \text{thesis}$ by simp
qed
ultimately show $((\bigwedge P \in A \cdot P X) \land \text{gvr}(n+1)) = ((\bigwedge P \in A \cdot (P X \land \text{gvr}(n))) \land \text{gvr}(n+1))$
  by simp
qed

lemma intChoice-Guarded [closure]:
  assumes $\text{Guarded} P \text{ Guarded} Q$

86
shows Guarded \((\lambda X. P(X) \sqcap Q(X))\)

proof
  - have Guarded \((\lambda X. \prod F \in \{P,Q\} \cdot F(X))\)
    by (rule UNF-Guarded, auto simp add: assms)
  thus \(?thesis\)
  by (simp)
qed

lemma cond-srea-Guarded [closure]:
  assumes Guarded \(P\) Guarded \(Q\)
  shows Guarded \((\lambda X. P(X) \sqsubset b \vDash_R Q(X))\)
  using assms by (rel-auto)

A tail recursive reactive design with a productive body is guarded.

lemma Guarded-if-Productive [closure]:
  fixes \(P \::\ (s', t::size-trace, a) \ hrel-rsp\)
  assumes \(P\) is NSRD \(P\) is Productive
  shows Guarded \((\lambda X. P :: SRD(X))\)
  proof (clarsimp simp add: Guarded-def)
    -- We split the proof into three cases corresponding to valuations for ok, wait, and wait' respectively.
    fix \(X\) \(n\)
    have \(a: (P :: SRD(X) \land \text{gvt} (\text{Suc} n))[[\text{false$/ok}] = (P :: SRD(X) \land \text{gvt} (\text{Suc} n))[[\text{false}$/ok]\]
    by (simp add: usubst closure SRD-left-zero-1 assms)
    have \(b: (P :: SRD(X) \land \text{gvt} (\text{Suc} n))[[\text{true}$/ok] = (P :: SRD(X) \land \text{gvt} (\text{Suc} n))[[\text{true}$/wait]\]
    by (simp add: usubst closure SRD-left-zero-2 assms)
    have \(c: (P :: SRD(X) \land \text{gvt} (\text{Suc} n))[[\text{false}$/wait] = (P :: SRD(X) \land \text{gvt} (\text{Suc} n))[[\text{false}$/wait]\]
    proof
      have \(1: (P :: SRD(X) \land \text{wait} (\text{Suc} n))[[\text{true}$/wait] = (P :: SRD(X) \land \text{wait} (\text{Suc} n))[[\text{true}$/wait]\]
      by (metis (no-types, lifting) Healthy-def R3h-wait-true SRD-healthls(3) SRD-idem)
      have \(2: (P :: SRD(X) \land \text{wait} (\text{Suc} n))[[\text{false}$/wait] = (P :: SRD(X) \land \text{wait} (\text{Suc} n))[[\text{false}$/wait]\]
      proof
        have \(Y :: (\text{Suc}, \text{t}', \text{a}') \ hrel-rsp. (P :: \text{false}$/wait') :: (SRD Y) :: SRD X \land \text{wait} (\text{Suc} n))[[\text{false}$/wait]\]
          = (((\text{Suc}': \text{pre}_R P) :: (SRD Y) :: \text{false}$/wait) \lor \text{post}_R P \land \text{str} >_u \text{str'}) :: (SRD Y) :: SRD X \land \text{wait} (\text{Suc} n))[[\text{false}$/wait]\]
        proof
          fix \(Y :: (\text{Suc}, \text{t}', \text{a}') \ hrel-rsp\)
          have \((P :: SRD Y) :: SRD X \land \text{wait} (\text{Suc} n))[[\text{false}$/wait] = ((R_a (\text{pre}_R P) \land \text{pre}_R P) \land \text{str} <_u \text{str'})[[\text{false}$/wait'] :: (SRD Y) :: SRD X \land \text{wait} (\text{Suc} n))[[\text{false}$/wait]\]
            by (metis (no-types) Healthy-def Productive-form assms(1) assms(2) NSRD-is-SRD)
          also have \(\ldots = \)
            \(((R_1 (R_2c (\text{pre}_R P) \Rightarrow \text{ok}' \land \text{post}_R P \land \text{str} <_u \text{str'})[[\text{false}$/wait'] :: (SRD Y) :: SRD X \land \text{wait} (\text{Suc} n))[[\text{false}$/wait]\]
            by (simp add: RHS-def R1-def R2c-def R2s-def RD1-def RD2-def usubst unrest assms closure design-def)
          also have \(\ldots = \)
            \(((\text{Suc}' \land \text{post}_R P \land \text{str} <_u \text{str'})[[\text{false}$/wait'] :: (SRD Y) :: SRD X \land \text{wait} (\text{Suc} n))[[\text{false}$/wait]\]
            by (simp add: RHS-def R1-def R2c-def R2s-def RD1-def RD2-def usubst unrest assms closure design-def)
\( \land \text{gert (Suc } n)\)[true, false/\$ok, \$wait] \\
\text{by (simp add: impl-all-def R2c-disj R1-disj R2c-not asms closure R2c-and R2c-preR rea-not-def R1-extend-conj' R2c-ok' R2c-post-SRD R1-tr-less-tr' R2c-tr-less-tr')}

\text{also have } ... = \\
(((\neg _r \text{ pre}_R \ P) \iff ( \text{SRD}(Y)))[false/\$wait] \land (ok' \land \text{post}_R \ P \land \text{str'} > \text{u} \ \text{str}) \iff ( \text{SRD} Y)[false/\$wait]) \land \text{gert (Suc } n)[true, false/\$ok, \$wait]

\text{by (simp add: ubust unrest asms closure seqr-or-distl NSRD-neg-pre-left-zero SRD-healths)}

\text{also have } ... = \\
(((\neg _r \text{ pre}_R \ P) \iff ( \text{SRD}(Y)))[false/\$wait] \land (post_R \ P \land \text{str'} > \text{u} \ \text{str}) \iff ( \text{SRD} Y)[true, false/\$ok, \$wait])

\text{\wedge gert (Suc } n)[true, false/\$ok, \$wait]

\text{proof} --

\text{have } (\neg _r \text{ pre}_R \ P \iff ( \text{SRD} Y)[false/\$wait] =\\
 ((\text{post}_R \ P \land \text{str'} > \text{u} \ \text{str}) \land \text{ok}' = \text{u} \ \text{true}) \iff ( \text{SRD} Y)[false/\$wait]

\text{by (rel-blast)}

\text{also have } ... = (\text{post}_R \ P \land \text{str'} > \text{u} \ \text{str})[true/\$ok'] \iff ( \text{SRD} Y)[false/\$wait][true/\$ok]

\text{using seqr-left-one-point of ok (post_R \ P \land \text{str'} > \text{u} \ \text{str}) True ( \text{SRD} Y)[false/\$wait]}

\text{by (simp add: true-all-def[THEN sym])}

\text{finally show ?thesis by (simp add: ubust unrest)}

\text{qed}

\text{finally show (P)[false/\$wait]] \iff ( \text{SRD} Y)[false/\$wait] \land \text{gert (Suc } n)[true, false/\$ok, \$wait] =\\
(((\neg _r \text{ pre}_R \ P) \iff ( \text{SRD}(Y)))[false/\$wait] \land (post_R \ P \land \text{str'} > \text{u} \ \text{str}) \iff ( \text{SRD} Y)[true, false/\$ok, \$wait])

\text{\wedge gert (Suc } n)[true, false/\$ok, \$wait]) \).

\text{\text{\wedge gert (Suc } n)[true, false/\$ok, \$wait] .}

\text{\text{\wedge gert (Suc } n)[true, false/\$ok, \$wait] .}

\text{have 1:(post_R \ P \land \text{str'} > \text{u} \ \text{str}) \iff ( \text{SRD} X)[true, false/\$ok, \$wait] \land \text{gert (Suc } n) =\\
((\text{post}_R \ P \land \text{str'} > \text{u} \ \text{str}) \iff ( \text{SRD} (X \land \text{gert } n))[true, false/\$ok, \$wait] \land \text{gert (Suc } n))

\text{apply (rel-auto)}

\text{apply (rename-tac \ tr \ st \ more \ ok \ wait \ tr' \ st' \ more' \ tr0 \ st0 \ more_0 \ ok')} \\
\text{apply (rule-tac \ x=tr0 \ in \ exI, \ rule-tac \ x=st0 \ in \ exI, \ rule-tac \ x=more_0 \ in \ exI)} \\
\text{apply (simp)} \\
\text{apply (erule trace-strict-prefixE)} \\
\text{apply (rename-tac \ tr \ st \ ref \ ok \ wait \ tr' \ st' \ ref' \ tr0 \ st0 \ ref_0 \ ok' \ zs)} \\
\text{apply (rule-tac \ x=False \ in \ exI)} \\
\text{apply (simp add: size-minus-trace)} \\
\text{apply (subgoal-tac size(\text{tr}) < size(\text{tr_0})} \\
\text{apply (simp add: less-diff-conv2 size- mono)}

\text{using size-strict-mono apply blast}

\text{apply (rename-tac \ tr \ st \ more \ ok \ wait \ tr' \ st' \ more' \ tr0 \ st0 \ more_0 \ ok')} \\
\text{apply (rule-tac \ x=tr0 \ in \ exI, \ rule-tac \ x=st0 \ in \ exI, \ rule-tac \ x=more_0 \ in \ exI)} \\
\text{apply (simp)} \\
\text{apply (erule trace-strict-prefixE)} \\
\text{apply (rename-tac \ tr \ st \ more \ ok \ wait \ tr' \ st' \ more' \ tr0 \ st0 \ more_0 \ ok')} \\
\text{apply (auto simp add: size-minus-trace)} \\
\text{apply (subgoal-tac size(\text{tr}) < size(\text{tr_0})} \\
\text{apply (simp add: less-diff-conv2 size- mono)}

\text{using size-strict-mono apply blast}

\text{done}

\text{have 2:(\neg _r \text{ pre}_R \ P) \iff ( \text{SRD} X)[false/\$wait] = (\neg _r \text{ pre}_R \ P) \iff ( \text{SRD}(X \land \text{gert } n))[false/\$wait]}

\text{by (simp add: NSRD-neg-pre-left-zero closure asms SRD-healths)}

\text{show ?thesis by (simp add: exp 1 2 utp-pred-laws.inf-sup-distrib2)}

\text{qed}
show \( ? \text{thesis} \)
proof
  have \( (P ;; (\text{SRD } X \land \text{gvrt } n)) [\text{true/false/} \text{ok/} \text{wait}] = \)
  \((P[\text{true/} \text{wait}] ;; (\text{SRD } X) [\text{true/} \text{wait}] \land \text{gvrt } n + 1)) [\text{true/false/} \text{ok/} \text{wait}] \lor \)
  \((P[\text{false/} \text{wait}] ;; (\text{SRD } X) [\text{false/} \text{wait}] \land \text{gvrt } n + 1)) [\text{true/false/} \text{ok/} \text{wait}] \)
  by (subst seqr-bool-split[of wait], simp add: usubst utp-pred-laws.distrib(4))

  also have \(... = ((P[\text{true/} \text{wait}] ;; (\text{SRD } X \land \text{gvrt } n)) [\text{true/} \text{wait}] \land \text{gvrt } n + 1)) [\text{true/false/} \text{ok/} \text{wait}] \lor \)
  \((P[\text{false/} \text{wait}] ;; (\text{SRD } X \land \text{gvrt } n)) [\text{false/} \text{wait}] \land \text{gvrt } n + 1)) [\text{true/false/} \text{ok/} \text{wait}] \)
  by (simp add: usubst utp-pred-laws.distrib(4))

  also have \(... = (P ;; (\text{SRD } X \land \text{gvrt } n)) \land \text{gvrt } n + 1)) [\text{true/false/} \text{ok/} \text{wait}] \)
  by (subst seqr-bool-split[of wait], simp add: usubst)

finally show \( ? \text{thesis} \) by (simp add: usubst)
qed

qed

show \( P ;; \text{SRD}(X) \land \text{gvrt } (\text{Suc } n) = (P ;; \text{SRD}(X \land \text{gvrt } n) \land \text{gvrt } (\text{Suc } n)) \)
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 [rtes]:
  fixes P :: ('s, 't: size-trace, 'a) hrel-rsp
  assumes P is NSRD P is Productive
  shows \( (\mu X \cdot P ;; \text{SRD}(X)) = (\text{Miracle}) \)
proof
  have 1:Continuous \( (\lambda X. P ;; \text{SRD } X) \) using SRD-Continuous
    by (clarsimp simp add: Continuous-def seq-SUP-distr THEN sym, drule_tac x=A in spec, simp)
  have 2: \( (\lambda X. P ;; \text{SRD } X) \in \{id\}_H \rightarrow \{\text{SRD}\}_H \)
    by (blast intro: funcsetI closure assms)
  with 1 2 have \( (\nu X \cdot P ;; \text{SRD}(X)) = (\nu X \cdot P ;; \text{SRD}(X)) \)
    by (simp add: guarded-fp-uniq Guarded-if-Productive[OF assms] funcsetI closure)
  also have \(... = ((\lambda X. P ;; \text{SRD } X) ^^ 0) false \sqinter ((\lambda X. P ;; \text{SRD } X) ^^ (i + 1)) false \)
    by (simp add: sup-continuous-lfp 1 sup-continuous-Continuous false-upred-def)
  also have \(... = ((\lambda X. P ;; \text{SRD } X) ^^ i) false \sqinter ((\lambda X. P ;; \text{SRD } X) ^^ (i + 1)) false \)
    by (simp add: sup-power-expand, simp)
  also have \(... = ((\lambda X. P ;; \text{SRD } X) ^^ (i + 1)) false \)
    by (simp)
  also have \(... = ((\lambda X. P ;; \text{SRD } X) ^^ (i + 1)) false \)
    by (simp add: simp)
proof (rule SUP-cong, simp-all)
fix \(i\)
show \(P :: \text{SRD} ( ((\lambda X. P :: \text{SRD} X) ^^ i) \text{false}) = (P :: P ^ i) :: \text{Miracle} \)
proof (induct \(i\))
  case \(0\)
  then show \(?case\)
    by (simp, metis srdes-hcond-def srdes-theory-continuous.healthy-top)
next
  case (Suc \(i\))
  then show \(?case\)
    by (simp add: Healthy-if NSRD-is-SRD SRD-power-comp SRD-seqr-closure assms sequence-associative assms 1 seqr-assoc [THEN sym] srdes-theory-continuous.weak.top-closed)
qed

also have ...
  = (\(\bigsqcap i \in \text{UNIV} \cdot P :: P ^ i\)) :: \text{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 \(P :: (s, 't::size-trace, 'a) \text{hrel-rsp}\)
  assumes \(P \text{ is NSRD} P \text{ is Productive}\)
  shows \((\mu X \cdot P :: \text{SRD}(X)) \text{ is NSRD}\)
  by (simp add: mu-csp-form-1 assms closure)

lemma mu-csp-form-1':
  fixes \(P :: (s, 't::size-trace, 'a) \text{hrel-rsp}\)
  assumes \(P \text{ is NSRD} P \text{ is Productive}\)
  shows \((\mu X \cdot P :: \text{SRD}(X)) = (P :: P') :: \text{Miracle}\)
proof
  have \((\mu X \cdot P :: \text{SRD}(X)) = (\prod i \in \text{UNIV} \cdot P :: P ^ i) :: \text{Miracle}\)
    by (simp add: mu-csp-form-1 assms closure ustar-def)
  also have ...
    = (P :: P') :: \text{Miracle}
    by (simp only: seq-UINF-distr [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
  \text{UTP-KAT.\text{utp-kleene}}
begin

11.1 State substitution

lemma srd-subst-RHS-tri-design [usubst]:
\[ [\sigma]_{S\sigma} \vdash R_s(P \vdash Q \diamond R) = R_s([\sigma]_{S\sigma} \vdash P) \vdash ([\sigma]_{S\sigma} \vdash Q) \diamond ([\sigma]_{S\sigma} \vdash R) \]

by (rel-auto)

**lemma** srd-subst-SRD-closed [closure]:
assumes \( P \) is SRD
shows \([\sigma]_{S\sigma} \vdash P\) is SRD

**proof** –

have \( \text{SRD}([\sigma]_{S\sigma} \vdash (\text{SRD}P)) = [\sigma]_{S\sigma} \vdash (\text{SRD}P) \)
  by (rel-auto)
thus \( \Box \)thesis
  by (metis Healthy-def assms)
qed

**lemma** preR-srd-subst [rdes]:
\[ \text{pre}_R([\sigma]_{S\sigma} \vdash P) = [\sigma]_{S\sigma} \vdash \text{pre}_R(P) \]
by (rel-auto)

**lemma** periR-srd-subst [rdes]:
\[ \text{peri}_R([\sigma]_{S\sigma} \vdash P) = [\sigma]_{S\sigma} \vdash \text{peri}_R(P) \]
by (rel-auto)

**lemma** postR-srd-subst [rdes]:
\[ \text{post}_R([\sigma]_{S\sigma} \vdash P) = [\sigma]_{S\sigma} \vdash \text{post}_R(P) \]
by (rel-auto)

**lemma** srd-subst-NSRD-closed [closure]:
assumes \( P \) is NSRD
shows \([\sigma]_{S\sigma} \vdash P\) is NSRD
by (rule NSRD-RC-intro, simp-all add: closure rdes assms unrest)

11.2 Assignment

**definition** assigns-srd :: 
  \( \text{`s usubst } \Rightarrow \text{`s::trace } \Rightarrow \text{hrel-rsp (}`R`) \)
where

\[ \text{[upred-defs]: assigns-srd } \sigma = R_s(\text{true } \vdash (\text{`tr} = \text{u } \text{`tr} \land \neg \text{`wait} = \text{u } \text{`Sigma} \text{`Sigma} = \text{u } \text{`Sigma})) \]

**syntax**

\[ -\text{assign-srd } :: \text{svids } \Rightarrow \text{uexprs } \Rightarrow \text{logic } (\text{`
  `'} :=R \text{`'})) \]
\[ -\text{assign-srd } :: \text{svids } \Rightarrow \text{uexprs } \Rightarrow \text{logic } (\text{infixr } :=R 90) \]

**translations**

\[ -\text{assign-srd } \text{xs vs } => \text{CONST assigns-srd } (\text{-mk-usubst (CONST id) xs vs}) \]
\[ -\text{assign-srd } \text{x v } <= \text{CONST assigns-srd } (\text{CONST subst-upd (CONST id) x v}) \]
\[ -\text{assign-srd } \text{x v } <= \text{-assign-srd } (\text{-spvar x}) \text{ v} \]
\[ x,y :=R u,v <= \text{CONST assigns-srd } (\text{CONST subst-upd (CONST subst-upd (CONST id) (CONST svar x) u) (CONST svar y) v}) \]

**lemma** assigns-srd-RHS-tri-des [rdes-def]:
\[ \langle \sigma \rangle_R = R_s(\text{true} = \text{false} \circ \langle \sigma \rangle_R) \]
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]: \( \text{pre}_R(\langle \sigma \rangle_R) = \text{true} \)
by (simp add: rdes-def rdes closure)
11.3 Conditional

**lemma perIR-cond-srea [rdes]:**
\[
\text{peri}_R(⟨σ⟩_R) = \text{false}
\]
by (simp add: rdes-def rdes closure)

**lemma postR-cond-srea [rdes]:**
\[
\text{post}_R(⟨σ⟩_R) = ⟨σ⟩_r
\]
by (simp add: rdes-def rdes closure rpred)

**proof**
\begin{itemize}
\item have periR-cond-srea
\end{itemize}
\end{proof}

**qed**

**lemma NSRD-cond-srea [closure]:**
\[
\text{AssumeR} b = \text{II} R ≪ b ≫ R \text{Miracle}
\]
92
lemma AssumeR-rdes-def [rdes-def]:
\[ [b] R = R_{\text{true}} \cup [\text{false }] [c] R \]
unfolding AssumeR-def by (rdes-eq)

lemma AssumeR-NSRD [closure]: \([b] R \) is NSRD
by (simp add: AssumeR-def closure)

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

lemma AssumeR-true: \([\text{true}] R \) = II
by (rel-auto)

lemma AssumeR-comp: \([b] R \ ;; \ [c] R = [b \land c] R \)
by (rdes-simp)

lemma AssumeR-choice: \([b] R \cap [c] R = [b \lor c] R \)
by (rdes-eq)

lemma AssumeR-refine-skip: II \subseteq [b] R
by (rdes-refine)

lemma AssumeR-test [closure]: test R \([b] R \)
by (simp add: AssumeR-refine-skip nsrd-thy.ustest-intro)

lemma Star-AssumeR: \([b]^{*} R = II \)
by (simp add: AssumeR-NSRD AssumeR-test nsrd-thy.Star-test)

lemma AssumeR-choice-skip: II \subseteq [b] R
by (rel-auto)

lemma cond-srea-AssumeR-form:
assumes \(P\) is NSRD \(Q\) is NSRD
shows \(P \circ b \triangleright R Q = ([b] R \ ;; \ P \cap [\neg b] R \ ;; \ Q)\)
by (rdes-eq cls: assms)

lemma cond-srea-insert-assume:
assumes \(P\) is NSRD \(Q\) is NSRD
shows \(P \circ b \triangleright R Q = ([b] R \ ;; \ P \circ b \triangleright R [\neg b] R \ ;; \ Q)\)
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 \([b] R \ ;; \ (P \circ b \triangleright R Q) = ([b] R \ ;; \ P)\)
by (rdes-eq cls: assms)

lemma AssumeR-cond-right:
assumes \(P\) is NSRD \(Q\) is NSRD
shows \([\neg b] R \ ;; \ (P \circ b \triangleright R Q) = ([\neg b] R \ ;; \ Q)\)
by (rdes-eq cls: assms)

11.5 Guarded commands

definition GuardedCommR :: \('s cond \Rightarrow ('s, 't::trace, 'α) hrel-rsp \Rightarrow ('s, 't, 'α) hrel-rsp (- \rightarrow_R - [85, 86] 85)\) where
gcmd-def[rdes-def]: GuardedCommR g A = A \circ g \triangleright_R Miracle
lemma gcmd-false [simp]: (false \rightarrow_R A) = Miracle
unfolding gcmd-def by (pred-auto)

lemma gcmd-true [simp]: (true \rightarrow_R A) = A
unfolding gcmd-def by (pred-auto)

lemma gcmd-SRD:
  assumes A is SRD
  shows (g \rightarrow_R A) is SRD
  by (simp add: gcmd-def SRD-cond-srea assms srdes-theory-continuous.weak.top-closed)

lemma gcmd-NSRD [closure]:
  assumes A is NSRD
  shows (g \rightarrow_R A) is NSRD
  by (simp add: gcmd-def NSRD-cond-srea assms NSRD-Miracle)

lemma gcmd-Productive [closure]:
  assumes A is NSRD A is Productive
  shows (g \rightarrow_R A) is Productive
  by (simp add: gcmd-def closure assms)

lemma gcmd-seq-distr:
  assumes B is NSRD
  shows (g \rightarrow_R A) \&\& B = (g \rightarrow_R A \&\& B)
  by (simp add: Miracle-left-zero NSRD-is-SRD assms cond-st-distr gcmd-def)

lemma gcmd-nondet-distr:
  assumes A is NSRD B is NSRD
  shows (g \rightarrow_R (A \&\& B)) = (g \rightarrow_R A) \&\& (g \rightarrow_R B)
  by (rdes-eq cls: assms)

lemma AssumeR-as-gcmd:
  [b] = b \rightarrow_R II
  by (rdes-eq)

12 Generalised Alternation

definition AlternateR :: `'a set \Rightarrow ('a \Rightarrow 's upred) \Rightarrow ('a \Rightarrow ('s, 't::trace, 'a) hrel-rsp) \Rightarrow ('s, 't, 'a) hrel-rsp \Rightarrow ('s, 't, 'a) hrel-rsp`
[upred-defs, rdes-def]: AlternateR I g A B = (\prod i \in I \cdot ((g i) \rightarrow_R (A i))) \&\& (\neg (\bigvee i \in I \cdot g i) \rightarrow_R B)

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

syntax
  -altindR-els :: pttrn \Rightarrow logic \Rightarrow logic \Rightarrow logic \Rightarrow logic \Rightarrow logic (if R -\epsilon- \cdot - \rightarrow - else - fi)
  -altindR :: pttrn \Rightarrow logic \Rightarrow logic \Rightarrow logic \Rightarrow logic (if R -\epsilon- \cdot - \rightarrow - fi)
  -altgcommR-els :: gcomms \Rightarrow logic \Rightarrow logic (if R - else - fi)
  -altgcommR :: gcomms \Rightarrow logic (if R - fi)
translations
\[
\begin{align*}
& \text{if } R_i \in I \cdot g \rightarrow A \text{ else } B \text{ fi} \rightarrow \text{CONST AlternateR } I (\lambda i. g) (\lambda i. A) B \\
& \text{if } R_i \in I \cdot g \rightarrow A \text{ else } B \text{ fi} \rightarrow \text{CONST AlternateR } I (\lambda i. g) (\lambda i. A) (\text{CONST Chaos}) \\
& \text{if } R_i \in I \cdot (g i) \rightarrow A \text{ else } B \text{ fi} \rightarrow \text{CONST AlternateR } I g (\lambda i. A) B \\
& \text{-altgcommR cs \rightarrow \text{CONST AlternateR-list cs (CONST Chaos)}} \\
& \text{-altgcommR-els cs P \rightarrow \text{CONST AlternateR-list cs P}} \\
& \text{-altgcommR-els (-gcomm-show cs) P \leftrightarrow \text{CONST AlternateR-list cs P}}
\end{align*}
\]

lemma AlternateR-NSRD-closed [closure]:
assumes \( \bigwedge i. i \in I \Rightarrow A i \text{ is NSRD} B \text{ is NSRD} \)
shows \((\text{if } R_i \in I \cdot g \rightarrow A i \text{ else } B \text{ fi}) \text{ is NSRD}\)
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]:
\((\text{if } R_i \in \{\} \cdot g \rightarrow A i \text{ else } B \text{ fi}) = B\)
by (rdes-simp)

lemma AlternateR-Productive [closure]:
assumes \( \bigwedge i. i \in I \Rightarrow A i \text{ is Productive} B \text{ is Productive} \)
shows \((\text{if } R_i \in I \cdot g \rightarrow A i \text{ else } B \text{ fi}) \text{ is Productive}\)
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 \((\text{if } R_i \in \{k\} \cdot g \rightarrow A i \text{ else } B \text{ fi}) = (A(k) \triangleleft g(k) \triangleright_R B)\)
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 \( \bigwedge i. i \in I \Rightarrow A i \text{ is NSRD} \)
\( A k \text{ is NSRD} B \text{ is NSRD} \)
\( g(k) \land \bigvee i \in I . g(i) \) = false
shows \((\text{if } R_i \in \text{insert } k I \cdot g \rightarrow A i \text{ else } B \text{ fi}) = (A(k) \triangleleft g(k) \triangleright_R (\text{if } R_i \in I \cdot g(i) \rightarrow A(i) \text{ else } B \text{ fi}))\)
proof (cases \( I = \{\}\))
  case True
  then show \(?thesis by (simp add: AlternateR-singleton assms)\)
next
  case False
  have 1: (∏ i ∈ I · g i →R A i) = (∏ i ∈ I · g i →R \(\text{pre}_R(A i) \land \text{peri}_R(A i) \land \text{post}_R(A i)\))
    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 cong: UINF-cong)
qed

12.1 Choose
definition choose-srd :: ('s,'t::trace,'α) hrel-rsp (choose_R) where
  [upred-defs,rdes-def]: choose_R = \(\text{R}_s(\text{true} \land \text{false} \land \text{true})\)

lemma preR-choose [rdes]: pre_R(choose_R) = true
  by (rel-auto)

lemma periR-choose [rdes]: peri_R(choose_R) = false
  by (rel-auto)

lemma postR-choose [rdes]: post_R(choose_R) = true
  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)

12.2 State Abstraction
definition state-srea ::
  's itself ⇒ ('s,'t::trace,'α,'β) rel-rsp ⇒ (unit,'t,'α,'β) rel-rsp where
  [upred-defs]: state-srea t P = (\∃ \{\$st,\$st'\} · P)\s

syntax
  -state-srea :: type ⇒ logic ⇒ logic (state · · · [0,200] 200)

translations
  state 'a · P == CONST state-srea TYPE('a) P

lemma R1-state-srea: R1(state 'a · P) = (state 'a · R1(P))
  by (rel-auto)

lemma R2c-state-srea: R2c(state 'a · P) = (state 'a · R2c(P))
  by (rel-auto)

lemma R3h-state-srea: R3h(state 'a · P) = (state 'a · R3h(P))
  by (rel-auto)

lemma RD1-state-srea: RD1(state 'a · P) = (state 'a · RD1(P))
  by (rel-auto)

lemma RD2-state-srea: RD2(state 'a · P) = (state 'a · RD2(P))
  by (rel-auto)

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

lemma SRD-state-srea [closure]: P is SRD ⇒ state 'a · 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 ⇒ state 'a · 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]: preR(state 'a · P) = (\forall \{st,st\} · pre_R(P)) S
  by (simp add: state-srea-def, rel-auto)

lemma periR-state-srea [rdes]: peri_R(state 'a · P) = state 'a · peri_R(P)
  by (rel-auto)

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

12.3 While Loop

definition WhileR :: 's upred ⇒ ('s, 't::size-trace, 'a) hrel-rsp ⇒ ('s, 't, 'a) hrel-rsp (while_R - do - od)
where
WhileR b P = (μ_R X · (P ;; X) ≪ b ∘_R H_R)

lemma Sup-power-false:
  fixes F :: 'a upred ⇒ 'a upred
  shows (\prod i. (F ^ i false) = (\prod i. (F ^ (i+1)) false)
proof –
  have (\prod i. (F ^ i false) = (F ^ 0 false) \cap (\prod i. (F ^ (i+1)) false)
    by (subst Sup-power-expand, simp)
  also have ... = (\prod i. (F ^ (i+1)) false)
    by (simp)
  finally show ?thesis .
qed

theorem WhileR-iter-expand:
  assumes P is NSRD P is Productive
  shows while_R b do P od = (\prod i · (P ≪ b ∘_R H_R) ^ i ;; (P ;; Miracle ≪ b ∘_R H_R)) (is ?lhs = ?rhs)
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: Continuous (λX. P ;; SRD X ≪ b ∘_R H_R)
    by (simp add: 1 closure assms)
  have ?lhs = (μ_R X · P ;; X ≪ b ∘_R H_R)
    by (simp add: WhileR-def)
  also have ... = (μ X · P ;; SRD(X) ≪ b ∘_R H_R)
    by (auto simp add: srd-mu-equiv closure assms)
  also have ... = (μ X · P ;; SRD(X) ≪ b ∘_R H_R)
    by (auto simp add: guarded-fp-uniq Guarded-if-Productive[OF assms] funcsetI closure assms)
  also have ... = (\prod i. ((λX. P ;; SRD X ≪ b ∘_R H_R) ^ i false)
    by (simp add: sup-continuous-lfp 2 sup-continuous-Continuous false-upred-def)
  also have ... = (\prod i. (λX. P ;; SRD X ≪ b ∘_R H_R) ^ (i+1)) false)
    by (simp add: Sup-power-false)
  also have ... = (\prod i. (P ≪ b ∘_R H_R) ^ i ;; (P ;; Miracle ≪ b ∘_R H_R))
proof (rule SUP-cong, simp)
  fix i
  show \((\lambda X. P :: SRD X \triangleleft b \triangleright_R H_R) \circ \circ (i + 1))\ false = (P \triangleleft b \triangleright_R H_R) \circ \circ i :: (P :: Miracle \triangleleft b \triangleright_R H_R)
proof (induct i)
  case 0
  then show ?case
  using Suc.hyps by auto
  also have \((P \triangleleft b \triangleright_R H_R) \circ \circ i :: (P :: Miracle \triangleleft b \triangleright_R H_R)) \triangleleft b \triangleright_R H_R
  by (metis no-types, lifting) Healthy-if NSRD-cond-srea NSRD-is-SRD NSRD-power-Suc NSRD-srd-skip SRD-cond-srea SRD-seqr-closure assms(1) power.power-eq-if seqr-left-unit srdes-theory-continuous.top-closed)
  have \((P \triangleleft b \triangleright_R H_R) \circ \circ i :: (P :: Miracle \triangleleft b \triangleright_R H_R)) \triangleleft b \triangleright_R H_R
proof (induct i)
  case 0
  then show ?case
  have \((P \triangleleft b \triangleright_R H_R) \circ \circ (\bigcap i \cdot (P \triangleleft b \triangleright_R H_R)^* i :: (P :: Miracle \triangleleft b \triangleright_R H_R))\)
  by (simp add: UNFIN-as-Sup-collect')
  finally show ?thesis .
qed

theorem WhileR-star-expand:
assumes P is NSRD P is Productive
shows while \_R b do P od = (P \triangleleft b \triangleright_R H_R)^* R :: (P :: Miracle \triangleleft b \triangleright_R H_R) (is \_lhs = \_rhs)
proof –
  have \_lhs = \(\bigcap i \cdot (P \triangleleft b \triangleright_R H_R) \circ \circ i :: (P :: Miracle \triangleleft b \triangleright_R H_R))
  by (simp add: WhileR-iter-expand seq-UNFIN-distr' assms)
also have \( \ldots = (P \triangleleft b \triangleright_R II_R)^* \triangleleft (P \triangleright_R \triangleleft b \triangleright_R II_R) \)
by \(\text{simpl add: \text{ustar-def}}\)
also have \( \ldots = ((P \triangleleft b \triangleright_R II_R)^* \triangleleft II_R) \triangleleft (P \triangleright_R \triangleleft b \triangleright_R II_R) \)
by \(\text{simpl add: \text{seqr-assoc SRD-left-unit closure assms}}\)
also have \( \ldots = (P \triangleleft b \triangleright_R II_R)^* \triangleright_R \triangleleft (P \triangleright_R \triangleleft b \triangleright_R II_R) \)
by \(\text{simpl add: \text{nsrd-thy.Star-def}}\)
finally show \( \text{thesis} \).
qed

lemma WhileR-NSRD-closed \([\text{closure}]\):
assumes \( P \) is NSRD \( P \) is Productive
shows \( \text{while}_{R} b \) do \( P \) od is NSRD
by \(\text{simpl add: WhileR-star-expand assms closure}\)

theorem WhileR-iter-form-lemma:
assumes \( P \) is NSRD
shows \( (P \triangleleft b \triangleright_R II_R)^* \triangleleft (P \triangleright_R \triangleleft b \triangleright_R II_R) = (([b]^T_R \triangleleft P)^* \triangleright_R \triangleleft ([\neg b]^T_R)^* \triangleright_R \triangleleft (P \triangleright_R \triangleleft b \triangleright_R II_R)) \)
proof –
  have \( (P \triangleleft b \triangleright_R II_R)^* \triangleright_R \triangleleft (P \triangleright_R \triangleleft b \triangleright_R II_R) = (([b]^T_R \triangleleft P)^* \triangleright_R \triangleleft ([\neg b]^T_R)^* \triangleright_R \triangleleft (P \triangleright_R \triangleleft b \triangleright_R II_R)) \)
  by \(\text{simpl add: AssumeR-NSRD NSRD-right-unit NSRD-srd-skip assms(1) cond-srea-AssumeR-form}\)
also have \( \ldots = (((([b]^T_R \triangleleft P)^* \triangleright_R \triangleleft ([\neg b]^T_R)^* \triangleright_R \triangleleft (P \triangleright_R \triangleleft b \triangleright_R II_R))) \triangleright_R \triangleleft (P \triangleright_R \triangleleft b \triangleright_R II_R))/ (P \triangleright_R \triangleleft b \triangleright_R II_R) \)
by \(\text{simpl add: AssumeR-NSRD NSRD-seqr-closure nsrd-thy.Star-denest assms(1)}\)
also have \( \ldots = (((([b]^T_R \triangleleft P)^* \triangleright_R \triangleleft ([\neg b]^T_R)^* \triangleright_R \triangleleft (P \triangleright_R \triangleleft b \triangleright_R II_R))) \triangleright_R \triangleleft (P \triangleright_R \triangleleft b \triangleright_R II_R))/ (P \triangleright_R \triangleleft b \triangleright_R II_R) \)
by \(\text{simpl add: AssumeR-NSRD NSRD-seqr-closure nsrd-thy.Star-invol assms(1)}\)
also have \( \ldots = (((([b]^T_R \triangleleft P)^* \triangleright_R \triangleleft ([\neg b]^T_R)^* \triangleright_R \triangleleft (P \triangleright_R \triangleleft b \triangleright_R II_R))) \triangleright_R \triangleleft (P \triangleright_R \triangleleft b \triangleright_R II_R))/ (P \triangleright_R \triangleleft b \triangleright_R II_R) \)
by \(\text{simpl add: AssumeR-NSRD NSRD-Miracle NSRD-right-unit NSRD-seqr-closure NSRD-srd-skip assms(1) cond-srea-AssumeR-form}\)
also have \( \ldots = (((([b]^T_R \triangleleft P)^* \triangleright_R \triangleleft ([\neg b]^T_R)^* \triangleright_R \triangleleft (P \triangleright_R \triangleleft b \triangleright_R II_R))) \triangleright_R \triangleleft (P \triangleright_R \triangleleft b \triangleright_R II_R))/ (P \triangleright_R \triangleleft b \triangleright_R II_R) \)
by \(\text{simpl add: upred-semiring.distrib-left}\)
also have \( \ldots = (((([b]^T_R \triangleleft P)^* \triangleright_R \triangleleft ([\neg b]^T_R)^* \triangleright_R \triangleleft (P \triangleright_R \triangleleft b \triangleright_R II_R))) \triangleright_R \triangleleft (P \triangleright_R \triangleleft b \triangleright_R II_R))/ (P \triangleright_R \triangleleft b \triangleright_R II_R) \)
proof –
  have \( (((([b]^T_R \triangleleft P)^* \triangleright_R \triangleleft ([\neg b]^T_R)^* \triangleright_R \triangleleft (P \triangleright_R \triangleleft b \triangleright_R II_R))) \triangleright_R \triangleleft (P \triangleright_R \triangleleft b \triangleright_R II_R))/ (P \triangleright_R \triangleleft b \triangleright_R II_R) = (II_R \triangleright_R (P \triangleright_R \triangleleft b \triangleright_R II_R)) \)
by \(\text{simpl add: AssumeR-NSRD NSRD-seqr-closure nsrd-thy.Star-unfoldr-eq assms(1)}\)
also have \( \ldots = (\neg b)^T_R \triangleright_R (P \triangleright_R \triangleleft b \triangleright_R II_R)) \)
by \(\text{metis \(\text{no-types, lifting}\)}\)
also have \( \ldots = (\neg b)^T_R \triangleright_R (P \triangleright_R \triangleleft b \triangleright_R II_R)) \)
by \(\text{simpr add: AssumeR-as-gcmd NSRD-srd-skip Star-AssumeR nsrd-thy.Star-slide gcmd-seq-distr skip-srea-self-unit wrl-dioioid.distrib-right')\)
also have \( \ldots = (\neg b)^T_R \triangleright_R (P \triangleright_R \triangleleft b \triangleright_R II_R)) \)
by \(\text{simpr add: AssumeR-true NSRD-right-unit assms(1)}\)
also have \( \ldots = (\neg b)^T_R \triangleright_R (P \triangleright_R \triangleleft b \triangleright_R II_R)) \)
by \(\text{simpr add: AssumeR-comp AssumeR-false}\)
finally have \( \text{thesis} \)
by \(\text{simpl add: semilattice-sup-class.le-sup11}\)
thus \( \text{thesis} \)
by \(\text{simpl add: semilattice-sup-class.le-iff-sup} \)
theorem WhileR-iter-form:
assumes P is NSRD P is Productive
shows while R b do P od = ([b]T R ;; P)∗R ;; [¬ b]T R
by (simp add: WhileR-iter-form-lemma WhileR-star-expand assms)

theorem WhileR-false:
assumes P is NSRD
shows while R false do P od = ⊥R
by (simp add: WhileR-def rpred closure srdes-theory-continuous LFP-const)

theorem WhileR-true:
assumes P is NSRD P is Productive
shows while R true do P od = P∗R ;; 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 while R b do (([b]T R ;; P) od = while R b do P 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 while R b do R4(P ⊢ Q o R) od =
(is ?lhs = ?rhs)
proof –
have ?lhs = ([b]T R ;; R4(P ⊢ Q o R))∗R ;; [¬ b]T R
  by (simp add: WhileR-iter-form Productive-rdes-RR-intro assms closure)
also have ... = ?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
Q1 is RC Q2 is RR Q3 is RR $st' $ Q2 Q3 is R4
⎯ — Refinement conditions
([b]T r ;; Q1)∗ r wp_r ([b]|< r Q1) ⊆ P1
P2 ⊆ [b]T r ;; Q2
P2 ⊆ [b]T r ;; Q3 ;; P2
P3 ⊆ [¬ b]T r
P3 ⊆ [b]T r ;; Q3 ;; P3
shows R4(P1 ∩ P2 o P3) ⊆ while R b do R4(Q1 ⊢ Q2 o Q3) od
proof (simp add: rdes-def assms, rule srdes-tri-refine-intro')
show ([b]T r ;; Q3)∗ r wp_r ([b]|< r Q3) ⊆ P1
  by (simp add: assms)
show P2 ⊆ (P1 ∩ ([b]T r ;; Q3)∗ r ;; [b]T r ;; Q2)
proof –
have \( P_2 \subseteq ([b]^T_r ; [\bar{b}]^T_r ; Q_3)^{*r} ; [b]^T_r ; Q_2 \)

by (simp add: assms rea-assume-RR rrel-thy. Star-inductl seq-RR-closed seqr-assoc)

thus \(?thesis\)

by (simp add: utp-pred-laws.le-inf12)

qed

definition \texttt{IterateR} :: \('a set \Rightarrow ('a \Rightarrow 'a) \Rightarrow ('a \Rightarrow ('a, 't::size-trace, 'a) hrel-rsp) \Rightarrow ('a, 't, 'a) hrel-rsp

where \texttt{IterateR} A g P = \texttt{whileR} (\bigvee i \in A \cdot g(i)) \texttt{do} (\texttt{ifR} i \in A \cdot g(i) \Rightarrow P(i) \texttt{fi}) \texttt{od}

syntax

\(-\texttt{iter-srd} : \texttt{pttrn} \Rightarrow \texttt{logic} \Rightarrow \texttt{logic} \Rightarrow \texttt{logic} \Rightarrow \texttt{logic} (\texttt{doR} -\cdot - \Rightarrow - \texttt{fi})\)

translations

\(-\texttt{iter-srd} x A g P = > \texttt{CONST IterateR} A (\lambda x. g) (\lambda x. P)
\texttt{-iter-srd} x A g P = < \texttt{CONST IterateR} A (\lambda x. g) (\lambda x'. P)\)

lemma \texttt{IterateR-NSRD-closed [closure]}:

assumes

\(\land i . \in I \Rightarrow P(i) \text{ is NSRD}\)

\(\land i . \in I \Rightarrow P(i) \text{ is Productive}\)

shows \(\texttt{doR} i \in I \cdot g(i) \Rightarrow P(i) \texttt{fi} \text{ is NSRD}\)

by (simp add: \texttt{IterateR-def closure assms})

lemma \texttt{IterateR-empty}:

\(\texttt{doR} i \in \{\} \cdot g(i) \Rightarrow P(i) \texttt{fi} = \Pi_R\)

by (simp add: \texttt{IterateR-def srd-mu-equiv closure rpred gfp-const WhileR-false})

lemma \texttt{IterateR-singleton}:

assumes \(P k \text{ is NSRD} P \ k \text{ is Productive}\)

shows \(\texttt{doR} i \in \{\} \cdot g(i) \Rightarrow P(i) \texttt{fi} = \texttt{whileR} g(k) \texttt{do} P(k) \texttt{od} \text{ (is \texttt{?lhs = ?rhs})}\)

proof

- have \(\text{lhs = whileR} g k \text{ do} P k \text{ srd \texttt{gfp-const WhileR-false}} \texttt{od}\)

  by (simp add: \texttt{IterateR-def AlternateR-singleton assms closure})

also have ... = \texttt{whileR} g k \texttt{do} g k \texttt{rdes-\texttt{gfp-const WhileR-false}} \texttt{od}\)

  by (simp add: \texttt{AlternateR-insert-assume closure assms})

also have ... = \texttt{whileR} g k \texttt{do} P k \texttt{od}\)

  by (simp add: \texttt{AssumeR-cond-left NSRD-Chaos WhileR-insert-assume assms})

finally show \(?thesis\)

qed

12.4 Iteration Construction

12.5 Substitution Laws

lemma \texttt{srd-subst-Chaos [asubst]}:

\(\sigma \uparrow_S \text{Chaos} = \text{Chaos}\)

by (rdes-simp)
lemma srd-subst-Miracle [usubst]:
\[ \sigma \uparrow_S \text{Miracle} = \text{Miracle} \]
by (rdes-simp)

lemma srd-subst-skip [usubst]:
\[ \sigma \uparrow_S \text{II}_R = \langle \sigma \rangle_R \]
by (rdes-eq)

lemma srd-subst-assigns [usubst]:
\[ \sigma \uparrow_S \langle \sigma \rangle_R = \langle \sigma \circ \sigma \rangle_R \]
by (rdes-eq)

12.6 Algebraic Laws

theorem assigns-srd-id: \[ \langle \text{id} \rangle_R = \text{II}_R \]
by (rdes-eq)

theorem assigns-srd-comp: \[ \langle \sigma \rangle_R ;\; \langle \varrho \rangle_R = \langle \varrho \circ \sigma \rangle_R \]
by (rdes-eq)

theorem assigns-srd-Miracle: \[ \langle \sigma \rangle_R ;\; \text{Miracle} = \text{Miracle} \]
by (rdes-eq)

theorem assigns-srd-Chaos: \[ \langle \sigma \rangle_R ;\; \text{Chaos} = \text{Chaos} \]
by (rdes-eq)

theorem assigns-srd-cond: \[ \langle \sigma \rangle_R \triangleleft b \triangleright_R \langle \varrho \rangle_R = \langle \sigma \triangleleft b \triangleright_s \varrho \rangle_R \]
by (rdes-eq)

theorem assigns-srd-left-seq:
assumes P is NSRD
shows \[ \langle \sigma \rangle_R ;\; P = \sigma \uparrow_S P \]
by (rdes-simp cls: assms)

lemma AlternateR-seq-distr:
assumes \( \forall i. \ A \ i \text{ is NSRD} \ B \text{ is NSRD} \ C \text{ is NSRD} \)
shows \[ (\text{if} \ R \ i \in I \cdot g \rightarrow A \ i \text{ else } B \ f) ;\; C = (\text{if} \ R \ i \in I \cdot g \rightarrow A \ i ;\; C \text{ else } B ;\; C \ f) \]
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 \[ (\text{if} \ R \ i \in \{a\} \cdot g \rightarrow A \ i \text{ else } B \ f) = (A \triangleleft g \triangleright_R B) \]
by (rdes-eq cls: assms)

lemma AlternateR-Chaos:
if \( R \ i \in A \cdot g(i) \rightarrow \text{Chaos} \ f = \text{Chaos} \)
by (cases A = {}, simp, rdes-eq)
lemma choose-srd-par:
choose \( R \parallel R \) choose \( R = \) choose \( R \)
by (rdes-eq)

12.7 Lifting designs to reactive designs

definition des-rea-lift :: \( \text{'s hrel-des} \Rightarrow \text{'s hrel-rsp} (R_D) \) where

\[
\text{upred-defs}: R_D(P) = R_u([\pre_D(P)]_S \vdash (\false \circ (\str \leq u \str') \land [\post_D(P)]_S))
\]

definition des-rea-drop :: \( \text{'s hrel-des} \Rightarrow \text{'s hrel-rsp} (D_R) \) where

\[
\text{upred-defs}: D_R(P) = \left([\pre_R(P)][\str/\str'] \vdash [\text{\$st}_s]_S \leq \right)_n \left([\post_R(P)][\str/\str'] \vdash [\text{\$st}_s\str']_S\right)
\]

lemma ndesign-rea-lift-inverse: \( D_R(R_D(p \vdash_n Q)) = p \vdash_n Q \)
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-inverse:
assumes \( P \) is \( N \)
\( Q \) is \( N \)
\( R_D P = R_D Q \) (is \( ?R(P) = ?RQ(Q) \))
shows \( Q = Q \)
proof
  have \( ?R([\pre_D(P)]_S \vdash_n \post_D(P)) = ?RQ([\pre_D(Q)]_S \vdash_n \post_D(Q)) \)
    by (simp add: ndesign-form assms)
  hence \( [\pre_D(P)]_S \vdash_n \post_D(P) = [\pre_D(Q)]_S \vdash_n \post_D(Q) \)
    by (metis ndesign-rea-lift-inverse)
  thus \( ?thesis \)
    by (simp add: ndesign-form assms)
qed

lemma des-rea-lift-closure [closure]: \( R_D(P) \) is \( SRD \)
by (simp add: des-rea-lift-def RHS-design-is-SRD unrest)

lemma preR-des-rea-lift [rdes]:
\( \pre_R(R_D(P)) = R1([\pre_D(P)]_S) \)
by (rel-auto)

lemma periR-des-rea-lift [rdes]:
\( \peri_R(R_D(P)) = (\false \circ [\pre_D(P)]_S \triangleright (\str \leq u \str')) \)
by (rel-auto)

lemma postR-des-rea-lift [rdes]:
\( \post_R(R_D(P)) = (\true \circ [\pre_D(P)]_S \triangleright (\neg \str \leq u \str')) \Rightarrow (\str' = u \str \land [\post_D(P)]_S) \)
apply (rel-auto) using minus-zero-eq by blast

lemma ndes-rea-lift-closure [closure]:
assumes \( P \) is \( N \)
shows \( R_D(P) \) is \( NSRD \)
proof
  obtain \( p \) \( Q \) where \( P: P = (p \vdash_n Q) \)
  by (metis H1-H3-commute H1-H3-is-normal-design H1-idem Healthy-def assms)
show \( ?thesis \)
  apply (rule NSRD-intro)
    apply (simp-all intro: closure rdes unrest P)
  apply (rel-auto)

103
lemma R-D-mono:
assumes P is H Q is H P ⊑ Q
shows R_D(P) ⊑ R_D(Q)
apply (simp add: des-rea-lift-def)
apply (rule srdes-tri-refine-intro')
apply (auto intro: H1-H2-refines assms aext-mono)
apply (rel-auto)
apply (metis (no-types, hide-lams) aext-mono assms)
done

Homomorphism laws

lemma R-D-Miracle:
R_D(⊤) = Miracle
by (simp add: Miracle-def, rel-auto)

lemma R-D-Chaos:
R_D(⊥) = Chaos
proof –
  have R_D(⊥) = R_D(false ⊢ true)
    by (rel-auto)
also have ... = R_a (false ⊢ false ◦ ($tr' = u$tr))
    by (simp add: Chaos-def des-rea-lift-def alpha)
also have ... = Chaos
    by (rel-auto)
also have ... = Chaos
    by (simp add: Chaos-def design-false-pre)
finally show ?thesis .
qed

lemma R-D-inf:
R_D(P ⊓ Q) = R_D(P) ⊓ R_D(Q)
by (rule antisym, rel-auto+)

lemma R-D-cond:
R_D(P ⊲ [b]D< ⊳ Q) = R_D(P) ⊲ R_D(Q)
by (rule antisym, rel-auto+)

lemma R-D-seq-ndesign:
R_D(p_1 ⊢_n Q_1 ; p_2 ⊢_n Q_2) = R_D((p_1 ⊢_n Q_1 ; (p_2 ⊢_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 \textit{R-D-seq}:
assumes \(P \text{ is } N\) \(Q \text{ is } N\)
shows \(R_D(P) ;: R_D(Q) = R_D(P ;; Q)\)
by (metis \textit{R-D-seq-ndesign} \textit{assms} \textit{rdesign-form})

These laws are applicable only when there is no further alphabet extension

lemma \textit{R-D-skip}:
\(R_D([H_D]) = ([H_R] :: ([s, t]::\text{trace}, \text{unit}) \text{ hrel-rsp})\)
apply (rel-auto) using \textit{minus-zero-eq} by blast+

lemma \textit{R-D-assigns}:
\(R_D([\sigma_D]) = ([\sigma_R] :: ([s, t]::\text{trace}, \text{unit}) \text{ hrel-rsp})\)
by (simp add: \textit{assigns-d-def} \textit{des-rea-lift-def} \textit{alpha} \textit{assigns-srd-RHS-tri-des} \textit{ISRD-RHS-design-form} \textit{SRD-as-reactive-tri-design} unrest)

end

13 Instantaneous Reactive Designs

theory \textit{utp-rdes-instant}
imports \textit{utp-rdes-prog}
begin

definition \textit{ISRD1} :: \('s, \t::\text{trace}, \alpha\) \text{ hrel-rsp} \Rightarrow \('s, \t, \alpha\) \text{ hrel-rsp} where
[upred-defs]: \textit{ISRD1}(P) = P \parallel R_s(\text{true} \vdash \text{false} \circ (\$tr' = _u \$tr))

definition \textit{ISRD} :: \('s, \t::\text{trace}, \alpha\) \text{ hrel-rsp} \Rightarrow \('s, \t, \alpha\) \text{ hrel-rsp} where
[upred-defs]: \textit{ISRD} = \textit{ISRD1} \circ \textit{NSRD}

lemma \textit{ISRD1-idem}: \textit{ISRD1}((\textit{ISRD1}(P))) = \textit{ISRD1}(P)
by (rel-auto)

lemma \textit{ISRD1-monotonic}: \(P \subseteq Q \Rightarrow \textit{ISRD1}(P) \subseteq \textit{ISRD1}(Q)\)
by (rel-auto)

lemma \textit{ISRD1-RHS-design-form}:
assumes \(\textit{ok}' \notin P\) \(\textit{ok}' \notin Q\) \(\textit{ok}' \notin R\)
shows \(\textit{ISRD1}(R_s(P \parallel Q \circ R)) = R_s(P \vdash \text{false} \circ (R \land \$tr' = _u \$tr))\)
using \textit{assms} by (simp add: \textit{ISRD1-def} \textit{choose-srd-def} \textit{RHS-tri-design-par} unrest, rel-auto)

lemma \textit{ISRD1-form}:
\textit{ISRD1}(\textit{SRD}(P)) = R_s(\textit{pre}_R(P) \vdash \text{false} \circ (\textit{post}_R(P) \land \$tr' = _u \$tr))
by (simp add: \textit{ISRD1-RHS-design-form} \textit{SRD-as-reactive-tri-design} unrest)

lemma \textit{ISRD1-rdes-def [rdes-def]}:
\[ P \text{ is } RR; R \text{ is }RR \] \Rightarrow \textit{ISRD1}(R_s(P \parallel Q \circ R)) = R_s(P \vdash \text{false} \circ (R \land \$tr' = _u \$tr))
by (simp add: \textit{ISRD1-def} \textit{rdes-def} closure \textit{rpred})

lemma \textit{ISRD-intro}:
assumes \(P \text{ is } \textit{NSRD} \textit{peri}_R(P) = (\neg_{\textit{r}} \textit{pre}_R(P)) (\$tr' = _u \$tr) \subseteq \textit{post}_R(P)\)
show \(P \text{ is } \textit{ISRD}\)
proof -
have \(R_s((\textit{pre}_R(P) \vdash \textit{peri}_R(P) \circ \textit{post}_R(P))) \text{ is } \textit{ISRD1}\)
apply (simp add: \textit{Healthy-def} \textit{rdes-def} closure \textit{assms}(1–2))
using \textit{assms}(3) \textit{least-zero} apply (rel-blast)

105
done

hence \( P \) is 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 \( P \) is RR \( Q \) is RR \( (\$tr' \equiv_u \$tr) \subseteq Q \)

shows \( R_\times (P \vdash \text{false} \circ Q) \) is ISRD1

unfolding Healthy-def

by (simp add: ISRD1-rdes-def assms unrest utp-pred-laws inf absorb1)

lemma ISRD-rdes-intro [closure]:

assumes \( P \) is RC \( Q \) is RR \( (\$tr' \equiv_u \$tr) \subseteq Q \)

shows \( R_\times (P \vdash \text{false} \circ Q) \) is 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 \( P \) is ISRD

shows \( P \) is ISRD1

proof –

have ISRD\((P)\) is ISRD1

by (simp add: ISRD-def Healthy-def ISRD1-idem)

thus \(?thesis\)

by (simp add: assms Healthy-if)

qed

lemma ISRD-implies-SRD:

assumes \( P \) is ISRD

shows \( P \) is SRD

proof –

have \( 1:\text{ISRD}(P) = R_\times ((\lnot_r (\lnot_r \text{pre}_R \ P) :: R1 \text{true} \land R1 \text{true}) \vdash \text{false} \circ (\text{post}_R \ P \land \$tr' \equiv_u \$tr))\)

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: Healthy-def)

qed

lemma ISRD-implies-NSRD [closure]:

assumes \( P \) is ISRD

shows \( P \) is NSRD

proof –

have \( 1:\text{ISRD}(P) = \text{ISRD1}(R_\times (\text{RD3}(\text{SRD}(P))))\)

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 \( ... = \text{ISRD1}(R_\times ((\lnot_r \text{pre}_R \ P) \text{wp}_r \text{false}_h \vdash (\exists \$st' \cdot \text{peri}_R \ P) \circ \text{post}_R \ P))\)

by (simp add: RD3-def,subst SRD-right-unit-tri-lemma, simp-all add: assms ISRD-implies-SRD)

also have \( ... = R_\times ((\lnot_r \text{pre}_R \ P) \text{wp}_r \text{false}_h \vdash \text{false} \circ (\text{post}_R \ P \land \$tr' \equiv_u \$tr))\)

by (simp add: RHS-tri-design-par ISRD1-def unrest choose-srd-def rpred closure ISRD-implies-SRD)
lemma ISRD-form:
assumes P is ISRD
shows Rs(pre Rs(P) ⊢ false ◦ (post Rs(P) ∧ $tr' = u$tr)) = P

proof -
  have P = ISRD1(P)
  also have ... = ISRD1(Rs(pre Rs(P) ⊢ peri Rs(P) ◦ post Rs(P)))
  also have ... = Rs(pre Rs(P) ⊢ false ◦ (post Rs(P) ∧ $tr' = u$tr))
  finally show ?thesis ..

qed

definition ISRD-elim [RD-elim]:

lemma IRD-form: P is ISRD
shows Rs(pre Rs(P) ⊢ false ◦ (post Rs(P) ∧ $tr' = u$tr)) = P

proof -
  have P = ISRD1(P)
  also have ... = ISRD1(Rs(pre Rs(P) ⊢ peri Rs(P) ◦ post Rs(P)))
  also have ... = Rs(pre Rs(P) ⊢ false ◦ (post Rs(P) ∧ $tr' = u$tr))
  finally show ?thesis ..

qed

lemma ISRD-elim [RD-elim]:

lemma skip-srd-ISRD [closure]: H is ISRD

lemma assigns-srd-ISRD [closure]: ⟨σ⟩ is ISRD

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 ISRD-Miracle-right-zero:
assumes P is ISRD pre Rs(P) = true
shows P ;; Miracle = 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 post Rs P ;; true = true
shows (µ Rs X · P ;; X) = Chaos

proof -
  have 1: (µ Rs X · P ;; X) = (µ X · P ;; SRD(X))
by (auto simp add: srdes-theory-continuous.utp-lfp-def closure assms)
have \((\mu X \cdot P :: SRD(X)) \subseteq Chaos\)
proof (rule gfp-upperbound)
  have \(P :: Chaos \subseteq Chaos\)
  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 \(P :: SRD Chaos \subseteq Chaos\)
by (simp add: Healthy-if srdes-theory-continuous.bottom-closed)
qed
thus \(?thesis\)
by (metis 1 dual-order.antisym srdes-theory-continuous.LFP-closed srdes-theory-continuous.bottom-lower)
qed

lemma recursive-assign-Chaos:
  \((\mu R X \cdot \langle \sigma \rangle_R :: X) = Chaos\)
by (rule ISRD-recurse-Chaos, simp-all add: closure rdes, rel-auto)

end

14 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-prag
  utp-rdes-instant
  utp-rdes-guarded
begin end

References

