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

White Rose Research Online URL for this paper: http://eprints.whiterose.ac.uk/129386/

---

**Monograph:**

---

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

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

Simon Foster    James Baxter    Ana Cavalcanti    Jim Woodcock
Samuel Canham
April 19, 2018

Abstract

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

Contents

1 Introduction

2 Reactive Designs Healthiness Conditions
  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
  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
  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 [3] of our theory of reactive designs. Reactive designs form an important semantic foundation for reactive modelling languages such as Circus [5]. For more details of this work, please see our recent paper [2].

2 Reactive Designs Healthiness Conditions

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

2.1 Preliminaries

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

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

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

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 (II_c) where
skip-rea-def [urel-defs]: II_c = (II ∨ (¬ $ok ∧ $tr ≤u $tr'))

definition skip-srea :: ('s, 't::trace, 'a) hrel-rp (II_R) where
skip-srea-def [urel-defs]: II_R = ((∃ $st · II_c) < wait II_c)

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

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

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

lemma R2c-skip-rea: \( R2c(\Pi_c) = \Pi_c \)
  by (rel-auto)

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

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

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

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

2.3 RD1: Divergence yields arbitrary traces

definition RD1 :: ('t::trace,'α,'β) rel-rp ⇒ ('t,'α,'β) rel-rp where
[upred-defs]: \( RD1(P) = (P \lor (\neg \$ok \land \$tr \leq_{u} \$tr')) \)

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

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

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

lemma RD1-mono: \( P \subseteq Q \implies RD1(P) \subseteq RD1(Q) \)
  by (rel-auto)

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

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

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

lemma RD1-wait-false [closure]: \( P \) is \( RD1 \) \( \implies P[false/$wait] \) is \( RD1 \)

by (rel-auto)

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

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

lemma RD1-seq-closure [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: $RD1(R1(P)) = (R1(P) <@ \text{ok} \triangleright R1(true))$
by (rel-auto)

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

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

lemma RD1-algebraic-intro:
assumes $P$ is $R1 (R1(true)) ;; P) = R1(true) (II_c ;; P) = P$
shows $P$ is RD1
proof
–
  have $P = (II_c ;; P)$
    by (simp add: assms(3))
  also have $\ldots = (R1(\text{ok} \Rightarrow II) ;; P)$
    by (simp add: skip-rea-R1-lemma)
  also have $\ldots = (((\neg \text{ok} \land R1(true)) ;; P)) \lor P$
    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 $\ldots = ((R1(\neg \text{ok}) ;; (R1(true) ;; P)) \lor P)$
    using dual-order.trans by (rel-blast)
  also have $\ldots = ((R1(\neg \text{ok}) ;; R1(true))) \lor P$
    by (simp add: assms(2))
  also have $\ldots = (R1(\neg \text{ok}) \lor P)$
    by (rel-auto)
  also have $\ldots = RD1(P)$
    by (rel-auto)
  finally show ?thesis
    by (simp add: Healthy-def)
qed

theorem RD1-left-zero:
assumes $P$ is $R1 P$ is RD1
shows \( (R1(true) ;; P) = R1(true) \)

proof –

have \( (R1(true) ;; R1(RD1(P))) = R1(true) \)
  by (rel-auto)

thus \(?thesis\)
  by (simp add: Healthy-if assms(1) assms(2))

qed

theorem RD1-left-unit:
  assumes \( P \) is \( R1 \)
  shows \( (II_c ;; P) = P \)

proof –

have \( (II_c ;; R1(RD1(P))) = R1(RD1(P)) \)
  by (rel-auto)

thus \(?thesis\)
  by (simp add: Healthy-if assms(1) assms(2))

qed

lemma RD1-alt-def:
  assumes \( P \) is \( R1 \)
  shows \( RD1(P) = (P \triangleleft \$\text{ok} \triangleright R1(true)) \)

proof –

have \( RD1(R1(P)) = (R1(P) \triangleleft \$\text{ok} \triangleright R1(true)) \)
  by (rel-auto)

thus \(?thesis\)
  by (simp add: Healthy-if assms)

qed


theorem RD1-algebraic:
  assumes \( P \) is \( R1 \)
  shows \( P \) is \( RD1 \) ←→ \( R1(true_h) ;; P \) = \( R1(true_h) \land (II_c ;; P) = P \)

using RD1-algebraic-intro RD1-left-unit RD1-left-zero assms
by blast

2.4 R3c and R3h: Reactive design versions of R3

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

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

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

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

lemma R3c-mono: \( P \subseteq Q \Longrightarrow R3c(P) \subseteq R3c(Q) \)
  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 \cap Q) = R3h(P) \cap R3h(Q)
  by (rel-auto)

lemma R3h-UINF:
  A ≠ {} ⇒ R3h(\bigsqcup_{i \in A} P(i)) = (\bigsqcup_{i \in 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-R3c: RD1(R3(R3c(P))) = R3c(RD1(P))
  by (rel-auto)

lemma R3c-RD1-def: P is RD1 ⇒ R3c(P) = RD1(R3c(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-R3c: R3c(R3c(P)) = R3c(P)
  by (rel-auto)

lemma R3-cancels-R3c: R3(R3c(P)) = R3c(P)
  by (rel-auto)
lemma \( R3h\)-cancels-\( R3c \): \( R3h(R3c(P)) = R3h(P) \)

by (rel-auto)

lemma \( R3c\)-semir-form: \( (R3c(P) :: R3c(R1(Q))) = R3c(P :: R3c(R1(Q))) \)

by (rel-simp, safe, auto intro: order-trans)

lemma \( R3h\)-semir-form: \( (R3h(P) :: R3h(R1(Q))) = R3h(P :: R3h(R1(Q))) \)

by (rel-simp, safe, auto intro: order-trans, blast+)

lemma \( R3c\)-seq-closure:

assumes \( P \) is \( R3c \) \( Q \) is \( R3c \) \( Q \) is \( R1 \)

shows \( (P :: Q) \) is \( R3c \)

by (metis Healthy-def \( R3c\)-semir-form assms)

lemma \( R3h\)-seq-closure [\( \text{closure} \):]

assumes \( P \) is \( R3h \) \( Q \) is \( R3h \) \( Q \) is \( R1 \)

shows \( (P :: Q) \) is \( R3h \)

by (metis Healthy-def \( R3h\)-semir-form assms)

lemma \( R3c\)-\( R3\)-left-seq-closure:

assumes \( P \) is \( R3 \) \( Q \) is \( R3c \)

shows \( (P :: Q) \) is \( R3c \)

proof

have \( (P :: Q) = ((P :: Q)[true/\$wait\] \prec \$wait \triangleright (P :: Q)) \)

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

also have ... = (((II \prec \$wait \triangleright P) :: Q)[true/\$wait\] \prec \$wait \triangleright (P :: Q))

by (metis Healthy-def \( R3c\)-def assms(1))

also have ... = (((II[true/\$wait\] :: Q) \prec \$wait \triangleright (P :: Q))

by (subst-tac)

also have ... = (((II \& \$wait \true) :: Q) \prec \$wait \triangleright (P :: Q))

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

also have ... = (((II[true/\$wait\] :: Q) \prec \$wait \triangleright (P :: Q))

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

also have ... = (((II[true/\$wait\] :: IIc \prec \$wait \triangleright Q)[true/\$wait\] \prec \$wait \triangleright (P :: Q))

by (metis Healthy-def \( R3c\)-def assms(2))

also have ... = (((II[true/\$wait\] :: IIc[true/\$wait\]) \prec \$wait \triangleright (P :: Q))

by (subst-tac)

also have ... = (((II \& \$wait \true) :: IIc) \prec \$wait \triangleright (P :: Q))

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

also have ... = (((II :: IIc) \prec \$wait \triangleright (P :: Q))

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

also have ... = (IIc \prec \$wait \triangleright (P :: Q))

by simp

also have ... = \( R3c(P :: Q) \)

by (simp add: \( R3c\)-def)

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

by (simp add: Healthy-def)

qed

lemma \( R3c\)-cases: \( R3c(P) = ((II \prec \$ok \triangleright R1(true)) \prec \$wait \triangleright P) \)
by (rel-auto)

lemma $R3h$-cases: $R3h(P) = (((\exists \; \text{st} : \text{II}) \prec \text{Ok} \triangleright R1(\text{true})) \prec \text{Wait} \triangleright P$
    by (rel-auto)

lemma $R3h$-form: $R3h(P) = \text{II}_R \prec \text{Wait} \triangleright P$
    by (rel-auto)

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

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

lemma skip-srea-$R3h$ [closure]: $\text{II}_R$ is $R3h$
    by (rel-auto)

lemma $R3h$-wait-true:
    assumes $P$ is $R3h$
    shows $P_t = \text{II}_R t$
    proof -
      have $P_t = (\text{II}_R \prec \text{Wait} \triangleright P)_t$
        by (metis Healthy-if $R3h$-form assms)
      also have $\ldots = \text{II}_R t$
        by (simp add: usubst)
      finally show $?thesis$ .
    qed

2.5 RD2: A reactive specification cannot require non-termination

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

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

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

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

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

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

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

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

lemma $RD2$-$R3c$-commute: $RD2(R3c(P)) = R3c(RD2(P))$
    by (rel-auto)
lemma RD2-R3h-commute: \( RD2(R3h(P)) = R3h(RD2(P)) \)
  by (rel-auto)

2.6 Major healthiness conditions

definition RH :: '(t::trace,'a) hrel-rp \( \Rightarrow \) (t,'a) hrel-rp (R)
where [upred-defs]: RH(P) = R1(R2c(R3c(P)))

definition RHS :: (s,t::trace,'a) hrel-rsp \( \Rightarrow \) (s,t,'a) hrel-rsp (Rs)
where [upred-defs]: RHS(P) = R1(R2c(R3h(P)))

definition RD :: (t::trace,'a) hrel-rp \( \Rightarrow \) (t,'a) hrel-rp
where [upred-defs]: RD(P) = RD1(RD2(RP(P)))

definition SRD :: (s,t::trace,'a) hrel-rsp \( \Rightarrow \) (s,t,'a) hrel-rsp
where [upred-defs]: SRD(P) = RD1(RD2(RHS(P)))

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

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

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

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

lemma RH-idem: R(P) = R(P)
  by (simp add: 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: Rs(Rs(P)) = Rs(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 Rs
  by (simp add: Idempotent-def RHS-idem)

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

lemma RHS-mono: P \subseteq Q \Rightarrow Rs(P) \subseteq Rs(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: \( R_s(P \cap Q) = R_s(P) \cap R_s(Q) \)
using Continuous-Disjunctuous Disjunctuous-def RHS-Continuous by auto

lemma RHS-INF: 
\( A \neq \{ \} \Rightarrow R_s(\bigcap\{ i \in A : P(i) \}) = (\bigcap\{ i \in A : R_s(P(i)) \}) \)
by (simp add: RHS-def R3h-UINF R2c-USUP R1-USUP)

lemma RHS-sup: \( R_s(P \cup Q) = R_s(P) \cup R_s(Q) \)
by (rel-auto)

lemma RHS-SUP: 
\( A \neq \{ \} \Rightarrow R_s(\bigcup\{ i \in A : P(i) \}) = (\bigcup\{ i \in A : R_s(P(i)) \}) \)
by (rel-auto)

lemma RHS-cond: \( R_s(P \triangleright b \triangleright Q) = (R_s(P) \triangleright b \triangleright R_s(Q)) \)
by (simp add: RHS-def R3h-cond R2c-condr R1-cond)

lemma RD-alt-def: \( RD(P) = RD_1(RD_2(R(P))) \)
by (simp add: R3c-via-RD1-R3 RD1-R1-commute RD1-R2c-commute RD1-R3c-commute RD1-RD2-commute RH-def RD-def RP-def)

lemma RD1-RH-commute: \( RD_1(R(P)) = R(RD_1(P)) \)
by (simp add: RD1-R1-commute RD1-R2c-commute RD1-R3h-commute RHS-def)

lemma RD2-RH-commute: \( RD_2(R(P)) = R(RD_2(P)) \)
by (metis R1-H2-commute R2c-H2-commute RD2-R3h-commute RD2-def RH-def)

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

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

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

lemma R3-RD-RP: \( R_3(RD(P)) = RP(RD_1(RD_2(P))) \)
by (metis (no-types, lifting) R1-R2c-is-R2 R2-R3-commute R3-cancels-R3c RD1-RH-commute RD2-RH-commute RD-alt-def RH-def RP-def)

lemma RD1-RHS-commute: \( RD_1(R_s(P)) = R_s(RD_1(P)) \)
by (simp add: RD1-R1-commute RD1-R2c-commute RD1-R3h-commute RHS-def)

lemma RD2-RHS-commute: \( RD_2(R_s(P)) = R_s(RD_2(P)) \)
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 (simp add: usubst)
  apply (metis Healthy-def R1-R3h-commute R2c-R3h-commute R3h-idem RD1-R3h-commute RD2-R3h-commute RHS-def SRD-def assms)
  apply (simp add: usubst)
  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]: P is SRD \implies P[false/$ok] = R1(true)
  by (metis (no-types, hide-lams) H1-H2-eq-design Healthy-def R1-ok-false RD1-R1-commute RD1-via-R1 RD2-def SRD-def SRD-healths)

lemma SRD-ok-true-wait-true [usubst]:
  assumes P is SRD
  shows P[true,true/$ok$,wait] = (\exists \$st \cdot II)[true,true/$ok$,wait]
  proof
  have P = (\exists \$st \cdot II) \& $ok \& R1 \& $wait \& P
    by (metis Healthy-def R3h-cases SRD-healths assms)
  moreover have ((\exists \$st \cdot II) \& $ok \& R1 \& $wait \& P)[true,true/$ok$,wait] = (\exists \$st \cdot II)[true,true/$ok$,wait]
    by (simp add: usubst)
  ultimately show \?thesis
    by (simp)
  qed

lemma SRD-left-zero-1: P is SRD \implies R1(true); P = R1(true)
  by (simp add: RD1-left-zero SRD-healths assms)

lemma SRD-left-zero-2:
  assumes P is SRD
  shows (\exists \$st \cdot II)[true,true/$ok$,wait]; P = (\exists \$st \cdot II)[true,true/$ok$,wait]
  proof
  have (\exists \$st \cdot II)[true,true/$ok$,wait]; R3h(P) = (\exists \$st \cdot II)[true,true/$ok$,wait]
    by (rel-auto)
  thus \?thesis
    by (simp add: Healthy-if SRD-healths assms)
2.7 UTP theories

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

```plaintext
typedecl RDES
typedecl SRDES

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

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

begin
  definition rdes-hcond :: (RDES, ('t::trace,'α) rp) uthy ⇒ (('t,'α) rp × ('t,'α) rp) health where
    [upred-defs]: rdes-hcond T = RD
  definition srdes-hcond :: (SRDES, ('s,'t::trace,'α) rsp) uthy ⇒ (('s,'t,'α) rsp × ('s,'t,'α) rsp) health
    where
      [upred-defs]: srdes-hcond T = SRD
  end

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

interpretation rdes-theory-continuous: utp-theory-continuous UTHY(RDES, ('t::trace,'α) rp)
  rewrites \ 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 o 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 o 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 o 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 R3 (RD1 (RD2 P)) ⊆ P
  proof
    have R3 (RD1 (RD2 P)) = RP (RD1 (RD2(P)))
```

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

interpretation rdes-rea-retract:
retract (RDES ← (RD1 ⋙ RD2,R3) → 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 ⋀ P. P ∈ carrier (uthy-order SRDES) ←→ P is SRD
and P is HSRDES ←→ P is SRD
and (µ X · F (HSRDES X)) = (µ X · F (SRD X))
and carrier (uthy-order SRDES) → carrier (uthy-order SRDES) ≡ [SRD]H → [SRD]H
and le (uthy-order SRDES) = op ⊆
and eq (uthy-order SRDES) = op =
by (unfold-locales, simp-all add: srdes-hcond-def SRD-Continuous)

declare srdes-theory-continuous.top-healthy [simp del]
declare srdes-theory-continuous.bottom-healthy [simp del]
abbreviation Chaos :: (′s,′t::trace,′a) hrel-rsp where
Chaos ≡ ⊥SRDES
abbreviation Miracle :: (′s,′t::trace,′a) hrel-rsp where
Miracle ≡ ⊤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 (µR) where µR F ≡ µSRDES F
abbreviation srd-gfp (νR) where νR F ≡ νSRDES F

syntax
-srd-mu :: pttrn ⇒ logic ⇒ logic (µR · · · [0, 10] 10)
-srd-nu :: pttrn ⇒ logic ⇒ logic (νR · · · [0, 10] 10)

translations
µR X · P == µR (λ X. P)
νR X · P == µR (λ X. P)

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

lemma srd-mu-equiv:
assumes Monotonic F F ∈ [SRD]H → [SRD]H
shows $(\mu R X \cdot F(X)) = (\mu X \cdot F(SRD(X)))$

by (metis assms srdes-hcond-def srdes-theory-continuous.utp-lfp-def)

end

3 Reactive Design Specifications

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

3.1 Reactive design forms

lemma srdes-skip-def: $II_{R} = R_{u}(true \vdash (str' = _u str \land \neg wait' \land [II]_{R}))$
  apply (rel-auto) using minus-zero-eq by blast+

lemma Chaos-def: $Chaos = R_{s}(false \vdash true)$
proof
  have $Chaos = SRD(true)$
    by (metis srdes-hcond-def srdes-theory-continuous.healthy-bottom)
  also have $... = R_{u}(H(true))$
    by (simp add: SRD-RHS-H1-H2)
  also have $... = R_{u}(false \vdash true)$
    by (metis H1-design H2-true design-false-pre)
  finally show $\text{thesis}$. qed

lemma Miracle-def: $Miracle = R_{s}(true \vdash false)$
proof
  have $Miracle = SRD(false)$
    by (metis srdes-hcond-def srdes-theory-continuous.healthy-top)
  also have $... = R_{u}(H(false))$
    by (simp add: SRD-RHS-H1-H2)
  also have $... = R_{u}(true \vdash false)$
  finally show $\text{thesis}$. qed

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

lemma RD2-reactive-design:
  assumes $\#ok \ni P \#ok \ni Q$
  shows $RD2(R(P \vdash Q)) = R(P \vdash Q)$
using assms
by (metis H2-design RD2-RH-commute RD2-def)

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

lemma RD2-st-reactive-design:
  assumes $\#ok \ni P \#ok \ni Q$
  shows $RD2(R_{u}(P \vdash Q)) = R_{u}(P \vdash Q)$
using assms
lemma wait-false-design:
\((P \vdash Q) \rightarrow ((P^\prime \vdash Q^\prime))\)
by (rel-auto)

lemma RD-RH-design-form:
\(RD(P) = R((\neg P^\prime \vdash P^\prime))\)
proof 
  have \(RD(P) = RD1(RD2(R1(R2c(R3c(R3c(P)))))))\)
    by (simp add: RD-alt-def RH-def)
  also have \(\ldots = RD1(H1(R2(R2s(R3c(R3c(P)))))))\)
    by (simp add: R2s-H2-commute R2s-H1-commute)
  also have \(\ldots = R2(R1(R1(R1(R2(R2s(R3c(R3c(P))))))))\)
    by (simp add: R1-R2c-commute R1-R2s-commute R1-R3c-commute RD1 via-R1)
  also have \(\ldots = R2(R1(R1(R1(H2(R1(P)))))))\)
    by (simp add: R1-R2s-commute R1-R3c-commute RD1 via-R1)
  also have \(\ldots = R2(R1(H1(R2(R2s(R3c(R3c(P))))))))\)
    by (simp add: R2s-H1-commute R2s-H2-commute)
  also have \(\ldots = RH(H(R1(P))))\)
    by (simp add: R1-R3c-commute RD1-R3c-commute RD1 via-R1)
  also have \(\ldots = RH(H(P)))\)
    by (simp add: R1-R2s-commute R1-R3c-commute RD1-R3c-commute RD1 via-R1)
  also have \(\ldots = RH(H(P)))\)
    by (simp add: R2-R1-form RH-def)
  also have \(\ldots = RH(((\neg P^\prime) \vdash P^\prime))\)
    by (simp add: H1-H2-eq-design)
  also have \(\ldots = R((\neg P^\prime \vdash P^\prime))\)
    by (simp add: no-types subst-RH-def subst-not wait-false-design)
  finally show ?thesis.
qed

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

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

lemma SRD-RH-design-form:
\(SRD(P) = R_\ast((\neg P^\prime \vdash P^\prime))\)
proof
  have \( \text{SRD}(P) = R_1(R_2c(R_3h(R_1((R_2(R_1(P))))))) \)
  by (metis (no-types, lifting) R1-H2-commute R1-R2c-commute R1-R3h-commute R1-idem R2c-H2-commute R1-RD1-commute R1-RD2-commute R1-R3h-commute RD2-R3h-commute RD2-def RHS-def SRD-def)
  also have \( \ldots = R_1(R_2(R_3h(H(P)))) \)
  by (metis (no-types, lifting) R1-H2-commute R1-R2c-commute R1-R3h-commute R1-idem R2c-H2-commute RD1-R1-commute RD1-R2c-commute RD1-R3h-commute RD2-R3h-commute RD2-def RHS-def SRD-def)
  also have \( \ldots = R_1(R_2s(R_3h(H(P)))) \)
  by (metis (no-types, lifting) R1-H2-commute R1-R2s-commute R1-R3h-commute R2-R1-form RD1-via-R1 RD2-def)
  also have \( \ldots = R_\#(H(P)) \)
  by (simp add: R1-R2s-R2c RHS-def)
  also have \( \ldots = R_\#((\neg P \#) \# P) \)
  by (simp add: Healthy-def’ SRD-RH-design assms (1) assms (2))
  finally show \(?thesis\).
qed

lemma \( \text{SRD-reactive-design} \):
  assumes \( P \) is SRD
  shows \( R_\#((\neg P \#) \# P) = P \)
  by (metis SRD-RH-design-form Healthy-def’ assms)

lemma \( \text{SRD-RH-design} \):
  assumes \( \# P \# Q \)
  shows \( \text{SRD}(R_\#(P \# Q)) = R_\#(P \# Q) \)
  by (simp add: RD1-st-reactive-design RD2-st-reactive-design RHS-idem SRD-def assms (1) assms (2))

lemma \( \text{RHS-design-is-SRD} \):
  assumes \( \# P \# Q \)
  shows \( R_\#(P \# Q) \) is SRD
  by (metis (no-types, lifting) SRD-RH-design assms (1) assms (2))

lemma \( \text{SRD-RHS-H1-H2} \): \( \text{SRD}(P) = R_\#(H(P)) \)
  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 \( \text{upred-defs} \):
  \( R_3c\text{-pre}(P) = (\text{true} \& \# \text{wait} \# P) \)

definition \( \text{upred-defs} \):
  \( R_3c\text{-post}(P) = ([H]D \& \# \text{wait} \# P) \)

definition \( \text{upred-defs} \):
  \( R_3h\text{-post}(P) = (\exists st . [H]D \& \# \text{wait} \# P) \)

lemma \( \text{R3c-pre-conj} \):
  \( R_3c\text{-pre}(P \& Q) = (R_3c\text{-pre}(P) \& R_3c\text{-pre}(Q)) \)
  by (rel-auto)

lemma \( \text{R3c-pre-seq} \):
  \( (\text{true} ;; Q) = \text{true} \implies R_3c\text{-pre}(P ;; Q) = (R_3c\text{-pre}(P) ;; Q) \)
  by (rel-auto)

lemma \( \text{unrest-ok-R3c-pre} \):
  \( \# P \# P \implies \# P \# R_3c\text{-pre}(P) \)
  by (simp add: R3c-pre-def cond-def unrest)

lemma \( \text{unrest-ok-R3c-pre} \):
  \( \# P \# P \implies \# P \# R_3c\text{-pre}(P) \)
  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,’α,’β) rel-rp
and R S :: (‘t,’β,’γ) rel-rp
assumes $\text{ok}’ \not\in P \land \text{ok}’ \not\in Q \land \text{ok} \not\in R \land \text{ok} \not\in S$
shows
$(R1(P \vdash Q) ;; R1(R \vdash S)) = R1(\neg (R1(\neg P) ;; R1(\text{true})) \land \neg (R1(Q) ;; R1(\neg R))) \vdash (R1(Q) ;; R1(S))$
proof –
have $(R1(P \vdash Q) ;; R1(R \vdash S)) = (\exists \text{ok}_0 \cdot (R1(P \vdash Q)[\text{ok}_0]/\text{ok}’) ;; (R1(R \vdash S)[\text{ok}_0]/\text{ok}’))$
using segr-middle ok-vwb-lens by blast
also from assms have $... = (\exists \text{ok}_0 \cdot R1(((\text{ok} \land P) \Rightarrow (\text{ok}_0 \land Q)) ;; R1(((\text{ok}_0 \land R) \Rightarrow (\text{ok}_0 \land S))))$
by (simp add: design-def R1-def usubst unrest)
also from assms have $... = (R1(((\text{ok} \land P) \Rightarrow (\text{true} \land Q)) ;; R1(((\text{true} \land R) \Rightarrow (\text{ok}’ \land S))))$
by (simp add: false-alt-def true-alt-def)
also from assms have $... = (R1(((\text{ok} \land P) \Rightarrow Q)) ;; R1(R \Rightarrow (\text{ok}’ \land S)))$
by simp
also from assms have $... = (R1(\neg (\text{ok} \land P)) ;; R1(\text{true}))$
by (simp add: impl-alt-def utp-pred-laws.sup.assoc)
also from assms have $... = (R1(\neg (\text{ok} \lor \neg P \lor Q)) ;; R1(\neg (R \lor (\text{ok}’ \land S))))$
by (simp add: R1-disj utp-pred-laws.disj-assoc)
also from assms have $... = (R1(\neg (\text{ok} \lor \neg P \lor Q)) ;; R1(\neg (R \lor (\text{ok}’ \land S))))$
by (simp add: segr-or-distl utp-pred-laws.sup.assoc)
also from assms have $... = (R1(Q) ;; R1(\neg (R \lor (\text{ok}’ \land S))))$
by (rel-blast)
also from assms have $... = (R1(Q) ;; (R1(\neg R) \lor R1(S) \land \text{ok}’))$
by (simp add: R1-disj R1-extend-conj utp-pred-laws.inf-commute)
also have $... = (R1(Q) ;; (R1(\neg R) \lor R1(S) \land \text{ok}’))$
by (simp add: R1-disj segr-or-distl)
also have $... = (R1(Q) ;; (R1(\neg R) \lor R1(S) \land \text{ok}’))$
by (simp add: R1-disj segr-or-distl)
proof

have \([(R1(\neg \$ok) :: (\langle t, \alpha, \beta \rangle \ rel-rp) ;; R1(true)) = \]
\((R1(\neg \$ok) :: (\langle t, \alpha, \gamma \rangle \ rel-rp) \)
by (rel-auto)
thus ?thesis
by simp
qed

also have ... = \(((R1(Q) :: (R1(\neg R) \lor (R1(S \land \$ok'))) ) \)
\lor \((R1(\neg \$ok) \)
\lor \((R1(\neg P) ;; R1(true)) \)
by (simp add: R1-extend-conj)
also have ... = \((R1(Q) :: (R1(\neg R))) \)
\lor \((R1(Q) :: (R1(S \land \$ok'))) \)
\lor \((R1(\neg \$ok) \)
\lor \((R1(\neg P) ;; R1(true)) \)
by (simp add: seqr-or-distr utp-pred-laws.sup_assoc)
also have ... = \(((R1(Q) :: (R1(\neg R))) \)
\lor \((R1(Q) :: (R1(S \land \$ok'))) \)
\lor \((R1(\neg \$ok) \)
\lor \((R1(\neg P) ;; R1(true)) \)
by (simp add: R1-disj R1-seqr)
also have ... = \(((R1(Q) :: (R1(\neg R))) \)
\lor \((R1(Q) :: (R1(S) \land \$ok')) \)
\lor \((R1(\neg P) ;; R1(true)) \)
by (rel-blast)
also have ... = \(((R1(Q) :: (R1(\neg R))) \)
\lor \((R1(Q) :: (R1(S) \land \$ok')) \)
\lor \((R1(\neg P) ;; R1(true)) \)
by (simp add: impl-alt-def utp-pred-laws.inf-commute)
also have ... = \(((R1(Q) :: (R1(\neg R))) \)
\lor \((R1(Q) :: (R1(S))) \)
\lor \((R1(\neg P) ;; R1(true)) \)
by (simp add: design-def)
finally show ?thesis .

qed

theorem R1-design-composition-RR:
assumes P is RR Q is RR R is RR S is RR
shows
\((R1(P \triangleright Q) :: R1(R \triangleright S)) = R1(((\neg r) P \ wp_r, false \land Q) wp_r R) \triangleright (Q :: S) \)
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
\((R1(P \triangleright Q) :: R1(R \triangleright S)) = R1((P \land Q) wp_r, R) \triangleright (Q :: S) \)
by (simp add: R1-design-composition-RR assms unrest Healthy-if closure wp)

lemma R2s-design: R2s(P \triangleright Q) = (R2s(P) \triangleright R2s(Q))
by (simp add: R2s-def design-def usubst)
lemma \( R2c\text{-design} \): \( R2c(P \vdash Q) = (R2c(P) \vdash R2c(Q)) \)
by \( \text{(simp add: design-def impl-alt-def R2c-disj R2c-not R2c-ok R2c-and R2c-ok')} \)

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

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

lemma \( R3c\text{-R1-design-composition} \):
\begin{align*}
\text{assumes } & \exists Q \; \exists P \; \exists Q \; \exists R \; \exists Q \; \exists S \\
\text{shows } & \exists R3c(R1(P \vdash Q)); \exists R3h(R1(R \vdash S)) = \\
& R3c(R1(\neg (R1(\neg P)); R1(true)) \land \neg ((R1(Q) \land \neg $\text{wait'}$); R1(\neg R))) \\
& \vdash (R1(Q) ;; ([\mathcal{I}]_D < $\text{wait' \triangleright} R1(S)))))
\end{align*}
proof –
\begin{align*}
\text{have } & 1:(\neg (R1(\neg R3c-pre P); R1 true)) = (R3c-pre(\neg (R1(\neg P); R1 true))) \\
& \text{by \( \text{(rel-auto)} \)} \\
\text{have } & 2:(\neg (R1(R3c-post Q); R1(\neg R3c-pre R))) = R3c-pre(\neg ((R1 Q \land \neg $\text{wait'}$); R1(\neg R))) \\
& \text{by \( \text{(rel-auto, blast+)} \)} \\
\text{have } & 3:(R1(R3c-post Q); R1 (R3c-post S)) = R3c-post (R1 Q ;; ([\mathcal{I}]_D < $\text{wait' \triangleright} R1 S)) \\
& \text{by \( \text{(rel-auto)} \)} \\
\text{show } & \text{thesis} \\
& \text{apply \( \text{(simp add: R3c-semir-form R1-R3c-commute[THEN sym] R1-R3c-design unrest)} \)} \\
& \text{apply \( \text{(sub R1-design-composition)} \)} \\
& \text{apply \( \text{(simp-all add: unrest assms R3c-pre-conj} \; 1 \; 2 \; 3) \)}
\end{align*}
done

qed

lemma \( R3h\text{-R1-design-composition} \):
\begin{align*}
\text{assumes } & \exists Q \; \exists P \; \exists Q \; \exists R \; \exists Q \; \exists S \\
\text{shows } & \exists R3h(R1(P \vdash Q)); \exists R3c(R1(R \vdash S)) = \\
& R3h(R1(\neg (R1(\neg P); R1(true)) \land \neg ((R1(Q) \land \neg $\text{wait'}$); R1(\neg R))) \\
& \vdash (R1(Q) ;; ([\mathcal{I}]_D < $\text{wait' \triangleright} R1(S)))))
\end{align*}
proof –
\begin{align*}
\text{have } & 1:(\neg (R1(\neg R3c-pre P); R1 true)) = (R3c-pre(\neg (R1(\neg P); R1 true))) \\
& \text{by \( \text{(rel-auto)} \)} \\
\text{have } & 2:(\neg (R1(R3h-post Q); R1(\neg R3c-pre R))) = R3c-pre(\neg ((R1 Q \land \neg $\text{wait'}$); R1(\neg R))) \\
& \text{by \( \text{(rel-auto, blast+)} \)} \\
\text{have } & 3:(R1(R3h-post Q); R1 (R3h-post S)) = R3h-post (R1 Q ;; ([\mathcal{I}]_D < $\text{wait' \triangleright} R1 S)) \\
& \text{by \( \text{(rel-auto, blast+)} \)} \\
\text{show } & \text{thesis} \\
& \text{apply \( \text{(simp add: R3h-semir-form R1-R3h-commute[THEN sym] R1-R3h-design unrest)} \)} \\
& \text{apply \( \text{(sub R1-design-composition)} \)} \\
& \text{apply \( \text{(simp-all add: unrest assms R3c-pre-conj} \; 1 \; 2 \; 3) \)}
\end{align*}
done

qed

lemma \( R2\text{-design-composition} \):
\begin{align*}
\text{assumes } & \exists Q \; \exists P \; \exists Q \; \exists R \; \exists Q \; \exists S \\
\text{shows } & \exists R2(P \vdash Q); \exists R2(R \vdash S) = \\
& R2((\neg (R1(\neg R2c P); R1 true)) \land \neg (R1 (R2c Q); R1 (\neg R2c R))); (R1 (R2c Q); R1 (R2c S)))
\end{align*}
apply \( \text{(simp add: R2-R2c-def R2c-design R1-design-composition assms unrest R2c-not R2c-and R2c-disj)} \)
proof -

lemma RH-design-composition:

assumes $\$ok ' z P $\$ok ' z Q $\$ok z R $\$ok z 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 \circ $\$wait \triangleright R1 (R2s S))))$

proof -

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

proof -

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

by (rel-auto)

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

using R2c-not by blast

also have ... = R2(R2 (\neg P) ; R2 true)

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

also have ... = (R2 (\neg P) ; R2 true)

by (simp add: R2-segr-distribute)

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

by (simp add: R2-def R2s-not R2s-tr)

finally show \$thesis

by (simp add: 1)

qed

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 (R2 (Q \land \neg $\$wait ' ) ; R2 (\neg R))

by (rel-auto)

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

by (metis R1-R2c-commute R1-R2c-is-R2 R2-segr-distribute)

also have ... = ((R1 (R2s Q) \land \neg $\$wait ' ) ; R1 (\neg R2s R))

by (rel-auto)

finally show \$thesis

qed

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

proof -

have R2c(((R1 (R2s Q))[true/$\$wait')] ; ([II]_D \circ $\$wait \triangleright R1 (R2s S))[true/$\$wait']) =

(R1 (R2s Q))[true/$\$wait'] ; ([II]_D \circ $\$wait \triangleright R1 (R2s S))[true/$\$wait']

proof -

have R2c(((R1 (R2s Q))[true/$\$wait')] ; ([II]_D \circ $\$wait \triangleright R1 (R2s S))[true/$\$wait']) =

R2c(R1 (R2s Q)[true/$\$wait')] ; ([II]_D[true/$\$wait])

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

also have ... = R2c(R2(Q[true/$\$wait']) ; [II]_D[true/$\$wait])

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

also have ... = (R2(Q[true/$\$wait']) ; [II]_D[true/$\$wait])

using R2c-seq by blast

also have ... = ((R1 (R2s Q))[true/$\$wait') ; ([II]_D \circ $\$wait \triangleright R1 (R2s S))[true/$\$wait']

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

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

done
finally show \texttt{thesis}.

\texttt{qed}

\texttt{moreover have R2c(((R1 (R2s Q)) [false/\$wait'] :: ([II] D < \$wait \triangleright R1 (R2s S)) [false/\$wait]))}
\texttt{= ((R1 (R2s Q)) [false/\$wait'] :: ([II] D < \$wait \triangleright R1 (R2s S)) [false/\$wait])}
\texttt{by (simp: usubst cond-unit-F)}
\texttt{(metis (no-types, hide-lams) R1-wait'\texttt{-false} R1-wait\texttt{-false} R2-def R2-subst-wait'\texttt{-false} R2-subst-wait\texttt{-false}}

\texttt{R2c-seq)}

\texttt{ultimately show \texttt{thesis}}

\texttt{proof} --
\texttt{have [II] D < \$wait \triangleright R1 (R2s S) = R2 ([II] D < \$wait \triangleright S)}
\texttt{by (simp add: R1-R2c-is-R2 R1-R2s-R2c R2-condr'}
\texttt{R2s-lift-skip R2s-wait)}

\texttt{then show \texttt{thesis}}
\texttt{by (simp add: R1-R2c-is-R2 R1-R2s-R2c R2c-seq)}

\texttt{qed}

\texttt{have (R1(R2s(R3c(P \triangleright Q)))) :: R1(R2s(R3c(R(T S)))) =}
\texttt{((R3c(R1(R2s(P \triangleright R2s(Q)))) :: R3c(R1(R2s(R)) \triangleright R2s(S))))}
\texttt{by (metis (no-types, hide-lams) R1-R2s-R2c R1-R3c-commute R2c-R3c-commute R2s-design)}

\texttt{also have \ldots = R3c(R1 (\sim (R1 (\sim R2s P) ;; R1 true) \land \sim ((R1 (R2s Q) \land \sim \$wait')) ;; R1 (\sim R2s R) )))} 
\texttt{by (simp add: R2c-design R2c-and R2c-not 1 2 3)}

\texttt{finally show \texttt{thesis}}
\texttt{by (simp add: R1-R2s-R2c R1-R3c-commute R2c-R3c-commute RH-def)}

\texttt{qed}

\textbf{lemma RHS-design-composition:}
\texttt{assumes \$ok' \notin P \$ok' \notin Q \$ok \notin R \$ok \notin S}
\texttt{shows \texttt{R}_s(P \triangleright Q) :: \texttt{R}_s(R \triangleright S) =}
\texttt{\texttt{R}_s((\sim (R1 (\sim R2s P) ;; R1 true) \land \sim ((R1 (R2s Q) \land \sim \$wait')) ;; R1 (\sim R2s R) )))} 
\texttt{by (rel-auto, blast)}

\texttt{proof} --
\texttt{have 1: (R2c (R1 (\sim R2s P) ;; R1 true) = (R1 (\sim R2s P) ;; R1 true) }
\texttt{by (rel-auto, blast)}

\texttt{have R2c(R1(R2s (\sim P) ;; R2 true)) = R2c(R1(R2s (\sim P) ;; R2 true) }
\texttt{using R2c-not by blast}

\texttt{also have \ldots = R2(R2s (\sim P) ;; R2 true) }
\texttt{by (metis R1-R2s-R2c-commute R1-R2c-is-R2)}

\texttt{also have \ldots = (R2 (\sim P) ;; R2 true) }
\texttt{by (simp add: R2-seqr-distribute)}

\texttt{also have \ldots = (R1 (\sim R2s P) ;; R1 true) }
\texttt{by (simp add: R2-def R2s-not R2s-true)}

\texttt{finally show \texttt{thesis}}
\texttt{by (simp add: 1)}

\texttt{qed}

\texttt{have 2:R2c ((R1 (R2s Q) \land \sim \$wait') ;; R1 (\sim R2s R)) = ((R1 (R2s Q) \land \sim \$wait') ;; R1 (\sim R2s R))}

\texttt{22}
proof
  have \( ((R1 \ (R2s \ Q) \land \neg \ $wait') \ ;\ R1 \ (\neg \ R2s \ R) = R1 \ (R2 \ (Q \land \neg \ $wait') \ ;\ R2 \ (\neg \ R)) \)  
    by (rel-auto, blast+)
  hence \( R2c \ ((R1 \ (R2s \ Q) \land \neg \ $wait') \ ;\ R1 \ (\neg \ R2s \ R) = (R2 \ (Q \land \neg \ $wait') \ ;\ R2 \ (\neg \ R)) \)  
    by (metis (no-types, lifting) R1-R2c-commute R1-R2c-is-R2 R2-seqr-distribute)
  also have \( \ldots = ((R1 \ (R2s \ Q) \land \neg \ $wait') \ ;\ R1 \ (\neg \ R2s \ R)) \)  
    by (rel-auto, blast+)
  finally show \( \vdots \).
qed

have 3: \( R2c(((R1 \ (R2s \ Q) \ ;\ (\exists \ $st \cdot [I]_D) < $wait \triangleright R1 \ (R2s \ S))) = (R1 \ (R2s \ Q) \ ;\ (\exists \ $st \cdot [I]_D) < $wait \triangleright R1 \ (R2s \ S))) \)  
proof
  have \( R2c(((R1 \ (R2s \ Q))[true/$wait'] \ ;\ (\exists \ $st \cdot [I]_D)[true/$wait]) = ((R1 \ (R2s \ Q))[true/$wait'] \ ;\ (\exists \ $st \cdot [I]_D)[true/$wait]) \)  
    by (simp add: usubst cond-unit-T R1-def R2s-def)
  also have \( \ldots = R2c(R2(Q[true/$wait']) \ ;\ R2((\exists \ $st \cdot [I]_D)[true/$wait])) \)  
    by (metis (no-types, lifting) R2-def R2-des-lift-skip R2-subst-wait-true R2-st-ex)
  using \( R2c-seq \) by blast
  also have \( \ldots = (R1 \ (R2s \ Q))[true/$wait'] \ ;\ R2((\exists \ $st \cdot [I]_D)[true/$wait]) \)  
    apply (simp add: usubst R2-des-lift-skip)
    apply (metis (no-types) R2-def R2-des-lift-skip R2-st-ex R2-subst-wait-true R2-subst-wait-true)
  done
  finally show \( \vdots \).
qed

moreover have \( R2c(((R1 \ (R2s \ Q))[false/$wait'] \ ;\ (\exists \ $st \cdot [I]_D)[false/$wait]) = ((R1 \ (R2s \ Q))[false/$wait'] \ ;\ (\exists \ $st \cdot [I]_D)[false/$wait]) \)  
by (simp add: usubst)
(metis (no-types, lifting) R1-wait-true R1-wait-true R2-R1-form R2-subst-wait-true R2-subst-wait-true)
ultimately show \( \vdots \)  
by (smt R2-R1-form R2-condr' R2-des-lift-skip R2-st-ex R2c-seq R2s-wait)
qed

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

lemma RHS-R2s-design-composition:
  assumes
\[ P \iff Q \iff R \iff S \]
\[ P \text{ is } R2s \quad Q \text{ is } R2s \quad R \text{ is } R2s \quad S \text{ is } R2s \]

**shows** \( \text{Re}((\neg R1 (\neg P) ; R1 \text{ true}) \land \neg ((R1 Q \land \neg \$\text{wait'}) ; R1 (\neg R))) \vdash (R1 Q ; (\exists st \cdot [H]^D \otimes \$\text{wait} \triangleright R1 S)) \)

**proof**
- have f1: \( R2s P = P \)
  - by (meson Healthy-def assms(5))
- have f2: \( R2s Q = Q \)
  - by (meson Healthy-def assms(6))
- have f3: \( R2s R = R \)
  - by (meson Healthy-def assms(7))
- have \( R2s S = S \)
  - by (meson Healthy-def assms(8))
- then show \( \neg \text{thesis} \)
  using f3 f2 f1 by (simp add: RHS-design-composition assms(1) assms(2) assms(3) assms(4))

qed

**lemma** RH-design-export-R1: \( R(P \vdash Q) = R(P \vdash R1(Q)) \)
- by (rel-auto)

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

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

**lemma** RHS-design-export-R1: \( \text{Re}(P \vdash Q) = \text{Re}(P \vdash R1(Q)) \)
- by (rel-auto)

**lemma** RHS-design-export-R2s: \( \text{Re}(P \vdash Q) = \text{Re}(P \vdash R2s(Q)) \)
- by (rel-auto)

**lemma** RHS-design-export-R2c: \( \text{Re}(P \vdash Q) = \text{Re}(P \vdash R2c(Q)) \)
- by (rel-auto)

**lemma** RHS-design-export-R2: \( \text{Re}(P \vdash Q) = \text{Re}(P \vdash R2(Q)) \)
- by (rel-auto)

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

**lemma** RHS-design-ok-wait: \( \text{Re}(P[true,false/$\$ok,$\$\text{wait}] \vdash Q[true,false/$\$ok,$\$\text{wait}]) = \text{Re}(P \vdash Q) \)
- by (rel-auto)

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

**lemma** RHS-design-conj-neg-R1-pre: \( \text{Re}(\neg (R1 P) \land Q) \vdash R = \text{Re}((\neg P) \land Q) \vdash R) \)
- by (rel-auto)

**lemma** RHS-pre-lemma: \( (\text{Re} P)^f f = R1(R2c(P^f f)) \)
3.4 Refinement introduction laws

**Lemma** \(R_1\)-design-refine:

**Assumes**
- \(P_1 \Rightarrow R_1\)
- \(P_1 \Rightarrow Q_1\)
- \(Q_1 \Rightarrow Q\)

**Proof**

- \(R_1(P_1 \Rightarrow P_2) \subseteq R_1(Q_1 \Rightarrow Q_2) \iff \exists P_1 \Rightarrow Q_1 \land P_1 \land Q_2 \Rightarrow P_2\)

**QED**

**Lemma** \(R_1\)-design-refine-RR:

**Assumes**
- \(P_1 \Rightarrow RR\)
- \(P_2 \Rightarrow RR\)
- \(Q_1 \Rightarrow RR\)
- \(Q_2 \Rightarrow RR\)

**Shows** \(R_1(P_1 \Rightarrow P_2) \subseteq R_1(Q_1 \Rightarrow Q_2) \iff \exists P_1 \Rightarrow Q_1 \land P_1 \land Q_2 \Rightarrow P_2\)

**QED**

**Lemma** RHS-design-refine:

**Assumes**
- \(P_1 \Rightarrow R_1\)
- \(P_2 \Rightarrow R_2c\)
- \(Q_1 \Rightarrow R_2c\)
- \(Q_2 \Rightarrow R_2c\)

**Proof**

- \(R_s(P_1 \Rightarrow P_2) \subseteq R_s(Q_1 \Rightarrow Q_2) \iff \exists P_1 \Rightarrow Q_1 \land P_1 \land Q_2 \Rightarrow P_2\)

**QED**

**Lemma** srdes-refine-intro:

**Assumes**
- \(P_1 \Rightarrow P_2\)
- \(P_1 \land Q_2 \Rightarrow Q_1\)

**Shows** \(R_s(P_1 \Rightarrow Q_1) \subseteq R_s(P_2 \Rightarrow Q_2)\)

**QED**
3.5 Distribution laws

**Lemma** \texttt{RHS-design-choice}: \( R_s(P_1 \triangleright Q_1) \land R_s(P_2 \triangleright Q_2) = R_s((P_1 \land P_2) \triangleright (Q_1 \lor Q_2)) \)

by (metis \texttt{RHS-inf design-choice})

**Lemma** \texttt{RHS-design-sup}: \( R_s(P_1 \triangleright Q_1) \lor R_s(P_2 \triangleright Q_2) = R_s((P_1 \lor P_2) \triangleright ((P_1 \Rightarrow Q_1) \land (P_2 \Rightarrow Q_2))) \)

by (metis \texttt{RHS-sup design-inf})

**Lemma** \texttt{RHS-design-USUP}:

assumes \( A \neq \{\} \)

shows \( (\prod_i \in A \cdot R_s(P(i) \triangleright Q(i))) = R_s((\prod_i \in A \cdot P(i)) \triangleright (\prod_i \in A \cdot Q(i))) \)

by (subst \texttt{RHS-INF[OF assms, THEN slm], simp add: design-UINF-mem assms})

end

4 Reactive Design Triples

theory \texttt{utp-rdes-triples}

imports \texttt{utp-rdes-designs}

begin

4.1 Diamond notation

**Definition** \texttt{wait’-cond} ::

\((’t’,\triangleright,\alpha,\beta)\) \texttt{rel-rp} \(\Rightarrow (’t’,\alpha,\beta)\) \texttt{rel-rp} \(\Rightarrow (’t’,\alpha,\beta)\) \texttt{rel-rp} (\texttt{infixr \tvar{65}}) where

[\texttt{upred-defs}] : \( P \circ Q = (P \triangleleft \texttt{wait’} \circ Q) \)

**Lemma** \texttt{wait’-cond-unrest [unrest]}:

\[ \texttt{out-var wait} \circ x; x \notin P; x \notin Q \] \Rightarrow \( x \notin (P \circ Q) \)

by (simp add: \texttt{wait’-cond-def unrest})

**Lemma** \texttt{wait’-cond-subst [usubst]}:

\( P \circ \sigma = (\sigma \circ P) \triangleright (\sigma \circ Q) \)

by (simp add: \texttt{wait’-cond-def usubst unrest})

**Lemma** \texttt{wait’-cond-left-false}: \( \triangleright false \circ P = (\neg \texttt{wait’} \land P) \)

by (rel-auto)

**Lemma** \texttt{wait’-cond-seq}: \( (P \circ Q) \triangleright R = ((P \triangleright (\texttt{wait’} \land R)) \lor (Q \triangleright (\neg \texttt{wait’} \land R))) \)

by (simp add: \texttt{wait’-cond-def cond-def seqr-or-distl, rel-blast})

**Lemma** \texttt{wait’-cond-true}: \( P \circ Q \land \texttt{wait’} = (P \land \texttt{wait’}) \)

by (rel-auto)

**Lemma** \texttt{wait’-cond-false}: \( P \circ Q \land (\neg \texttt{wait’}) = (Q \land (\neg \texttt{wait’})) \)

by (rel-auto)

**Lemma** \texttt{wait’-cond-idem}: \( P \circ P = P \)

by (rel-auto)

**Lemma** \texttt{wait’-cond-conj-exchange}:

\( ((P \circ Q) \land (R \circ S)) = (P \land R) \circ (Q \land S) \)

by (rel-auto)

**Lemma** \texttt{subt-wait’-cond-true [usubst]}: \( (P \circ Q)[true/\texttt{wait’}] = P[true/\texttt{wait’}] \)

by (rel-auto)

26
4.2 Export laws

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

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

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

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

**Lemma** \texttt{RH-design-peri-R2c}: \( R(P \vdash R2c(Q) \circ R) = R(P \vdash Q \circ R) \)
by \( \text{(metis R1-R2s-R2c RH-design-peri-R1 RH-design-peri-R2s)} \)

**Lemma** \texttt{RHS-design-peri-R1}: \( R_s(P \vdash R1(Q) \circ R) = R_s(P \vdash Q \circ R) \)
4.3.2 Unrestriction laws

**Lemma**: \( \text{ok-pre-unrest [unrest]} \) \( \text{\$ok \not\in\{pre\} P} \)

\by (simp add: pre\_def unrest usubst)

**Lemma**: \( \text{ok-peri-unrest [unrest]} \) \( \text{\$ok \not\in\{peri\} P} \)

\by (simp add: peri\_def unrest usubst)

**Lemma**: \( \text{ok-post-unrest [unrest]} \) \( \text{\$ok \not\in\{post\} P} \)

\by (simp add: post\_def unrest usubst)

**Lemma**: \( \text{ok\_cmt-unrest [unrest]} \) \( \text{\$ok \not\in\{cmt\} P} \)

\by (simp add: cmt\_def unrest usubst)

**Lemma**: \( \text{ok\_pre-unrest [unrest]} \) \( \text{\$ok \not\in\{pre\} P} \)

\by (simp add: pre\_def unrest usubst)
lemma \( \text{ok}’\)-peri-unrest [unrest]: $\text{ok}’ \not\in \text{peri}_R \ P$
by (simp add: peri\(_R\)-def unrest usubst)

lemma \( \text{ok}’\)-post-unrest [unrest]: $\text{ok}’ \not\in \text{post}_R \ P$
by (simp add: post\(_R\)-def unrest usubst)

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

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

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

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

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

4.3.3 Substitution laws

lemma \( \text{pre}_s\)-design: \( \text{pre}_s \vdash (P \Rightarrow Q) = (\neg \text{pre}_s \vdash P) \)
by (simp add: design-def \( \text{pre}_R\)-def usubst)

lemma \( \text{peri}_s\)-design: \( \text{peri}_s \vdash (P \Rightarrow Q \circ R) = \text{peri}_s \vdash (P \Rightarrow Q) \)
by (simp add: design-def usubst wait\(’\)-cond-def)

lemma \( \text{post}_s\)-design: \( \text{post}_s \vdash (P \Rightarrow Q \circ R) = \text{post}_s \vdash (P \Rightarrow R) \)
by (simp add: design-def usubst wait\(’\)-cond-def)

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

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

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

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

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

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

29
lemma \( \text{post}_{s} \cdot R2c \ [\text{subst}] \): \( \text{post}_{s} \uparrow R2c(P) = R2c(\text{post}_{s} \uparrow P) \)
by (simp add: \( R2c\text{-def} \) \( R2s\text{-def} \) \( \text{subst} \))

lemma \( \text{cmt}_{s} \cdot R1 \ [\text{subst}] \): \( \text{cmt}_{s} \uparrow R1(P) = R1(\text{cmt}_{s} \uparrow P) \)
by (simp add: \( R1\text{-def} \) \( \text{subst} \))

lemma \( \text{cmt}_{s} \cdot R2c \ [\text{subst}] \): \( \text{cmt}_{s} \uparrow R2c(P) = R2c(\text{cmt}_{s} \uparrow P) \)
by (simp add: \( R2c\text{-def} \) \( R2s\text{-def} \) \( \text{subst} \))

lemma \( \text{pre} \cdot \text{wait} \cdot \text{false} \):
\( \text{pre}(P[\text{false} / \text{wait}]) = \text{pre}(P) \)
by (rel-auto)

lemma \( \text{cmt} \cdot \text{wait} \cdot \text{false} \):
\( \text{cmt}(P[\text{false} / \text{wait}]) = \text{cmt}(P) \)
by (rel-auto)

lemma \( \text{rea} \cdot \text{pre} \cdot \text{RHS-design} \):
\( \text{pre}_{s}(P \vdash Q) = R1(R2c(\text{pre}_{s} \uparrow P)) \)
by (simp add: \( \text{RHS-def} \) \( \text{subst} \) \( \text{pre}_{s} \cdot \text{design} \) \( \text{R1-negate} \) \( R1 \cdot \text{not} \) \( R1 \cdot \text{idem} \) \( \text{R2c-not} \) \( \text{rea-not-def} \))

lemma \( \text{rea} \cdot \text{cmt} \cdot \text{RHS-design} \):
\( \text{cmt}_{s}(P \vdash Q) = R1(R2c(\text{cmt}_{s} \uparrow P \Rightarrow r Q)) \)
by (simp add: \( \text{RHS-def} \) \( \text{subst} \) \( \text{cmt}_{s} \cdot \text{design} \) \( \text{R1-idem} \))

lemma \( \text{rea} \cdot \text{peri} \cdot \text{RHS-design} \):
\( \text{peri}_{s}(P \vdash Q \circ R) = R1(R2c(\text{peri}_{s} \uparrow P \Rightarrow r Q)) \)
by (simp add: \( \text{RHS-def} \) \( \text{subst} \) \( \text{peri}_{s} \cdot \text{design} \) \( \text{rel-auto} \))

lemma \( \text{rea} \cdot \text{post} \cdot \text{RHS-design} \):
\( \text{post}_{s}(P \vdash Q \circ R) = R1(R2c(\text{post}_{s} \uparrow P \Rightarrow r R)) \)
by (simp add: \( \text{RHS-def} \) \( \text{subst} \) \( \text{post}_{s} \cdot \text{design} \) \( \text{rel-auto} \))

lemma \( \text{peri} \cdot \text{cmt} \cdot \text{def} \):
\( \text{peri}(P) = (\text{cmt}(P))[\text{true} / \text{wait}] \)
by (rel-auto)

lemma \( \text{post} \cdot \text{cmt} \cdot \text{def} \):
\( \text{post}(P) = (\text{cmt}(P))[\text{false} / \text{wait}] \)
by (rel-auto)

lemma \( \text{rdes} \cdot \text{export} \cdot \text{cmt} \):
\( \text{R}_{s}(P \vdash \text{cmt}_{s} \uparrow Q) = \text{R}_{s}(P \vdash Q) \)
by (rel-auto)

lemma \( \text{rdes} \cdot \text{export} \cdot \text{pre} \):
\( \text{R}_{s}((P[\text{true} \cdot \text{false} / \text{ok} \cdot \text{wait}]) \vdash Q) = \text{R}_{s}(P \vdash Q) \)
by (rel-auto)

4.3.4 Healthiness laws

lemma \( \text{wait} \cdot \text{unrest} \cdot \text{pre} \cdot \text{SRD} \ [\text{unrest}] \):
\( \text{\$wait} \cdot \uparrow \text{pre}_{s}(P) \Rightarrow \text{\$wait} \cdot \uparrow \text{pre}_{s}(\text{SRD} P) \)
apply (rel-auto)
using least-zero apply blast

4.3.4 Healthiness laws

lemma \( \text{wait} \cdot \text{unrest} \cdot \text{pre} \cdot \text{SRD} \ [\text{unrest}] \):
\( \text{\$wait} \cdot \uparrow \text{pre}_{s}(P) \Rightarrow \text{\$wait} \cdot \uparrow \text{pre}_{s}(\text{SRD} P) \)
apply (rel-auto)
using least-zero apply blast

done

lemma \( \text{R1} \cdot \text{R2s} \cdot \text{cmt} \cdot \text{SRD} \):
assumes \( P \) is \( \text{SRD} \)
shows \( \text{R1}(R2s(\text{cmt}_{s}(P))) = \text{cmt}_{s}(P) \)
by (metis (no-types, lifting) \( \text{R1} \cdot \text{R2c-commute} \) \( \text{R1} \cdot \text{R2s} \cdot \text{R2c} \) \( \text{R1-idem} \) \( \text{R2c-idem} \) \( \text{SRD-reactive} \cdot \text{design} \) \( \text{assms} \) \( \text{rea} \cdot \text{cmt} \cdot \text{RHS-def} \) \( \text{design} \))
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 peri_R-SRD-R1 [closure]: P is SRD ⇒ 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-R2c-peri-RHS R1-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 post_R-SRD-R1 [closure]: P is SRD ⇒ post_R(P) is R1
  by (simp add: Healthy-def' R1-post-SRD)
lemma \( R1-R2c\)-post-RHS:
  assumes \( P \) is SRD
  shows \( R1(R2c(post_R(P))) = post_R(P) \)
  by (metis \( R1-R2s\)-R2c \( R1-R2s\)-post-SRD assms)

lemma \( R2\)-cmt-conj-wait':
  \( P \) is SRD \( \Rightarrow \) \( R2(cmt_R(P) \land \neg\$wait') = (cmt_R(P) \land \neg\$wait') \)
  by (simp add: \( R2\)-def \( R2\)-s-conj \( R2\)-s-not \( R2\)-s-wait'
  \( R1\)-extend-conj \( R1\)-R2s-cmt-SRD)

lemma \( R2\)-preR:
  \( P \) is SRD \( \Rightarrow \) \( R2c(pre_R(P)) = pre_R(P) \)
  by (metis \( no\)-types \( lifting \) \( R1\)-R2c-commute \( R1\)-R2s-idem \( SRD\)-reactive-design \( rea\)-pre-RHS-design)

lemma \( R2\)-postR:
  \( P \) is SRD \( \Rightarrow \) \( R2c(post_R(P)) = post_R(P) \)
  by (metis \( no\)-types \( hide\)-lams \( R1\)-R2c-commute \( R1\)-R2-cmd-R2 R1-R2s-post-SRD \( R2\)-def \( R2\)-s-idem)

lemma \( peri\)-R2c-closed [closure]: \( P \) is SRD \( \Rightarrow \) \( peri_R(P) \) is \( R2\)
  by (simp add: \( Healthy\)-def \( R2\)-peri-SRD)

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

lemma \( post\)-RR [closure]: \( P \) is SRD \( \Rightarrow \) \( post_R(P) \) is \( RR \)
  by (rule \( RR\)-intro, simp-all add: closure \( unrest \))

lemma wp-R-trace-ident-pre [wp]:
  \( (\$tr' = u \$tr \land [II]_R) wp \ pre_R(P) = pre_R(P) \)
  by (rel-auto)

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

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

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

lemma trace-ident-right-postR:
  \( post_R(P) ;; (\$tr' = u \$tr \land [II]_R) = post_R(P) \)
  by (rel-auto)
lemma preR-R2-closed [closure]: $P$ is SRD $\Rightarrow$ pre$_R(P)$ is R2
by (simp add: R2-comp-def Healthy-comp closure)

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

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

4.3.5 Calculation laws

lemma wait'-cond-peri-post-cmt [rdes]:
cmt$_R P = peri_R P \circ post_R P$
by (rel-auto)

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

lemma periR-rdes [rdes]:
assumes $P$ is RR $Q$ is RR
shows peri$_R(R_s(P \vdash Q \circ R)) = (P \Rightarrow Q)$
by (simp add: rea-peri-RHS-design unrest usubst assms Healthy-if RR-implies-R2c closure)

lemma postR-rdes [rdes]:
assumes $P$ is RR $R$ is RR
shows post$_R(R_s(P \vdash Q \circ R)) = (P \Rightarrow R)$
by (simp add: rea-post-RHS-design unrest usubst assms Healthy-if RR-implies-R2c closure)

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

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

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

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

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

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

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

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

lemma postR-srdes-skip [rdes]: post$_R$(II$_R$) = ($\exists tr' = u \exists tr \land [II]_R$)
\[
\text{by (rel-auto)}
\]

\textbf{Lemma preR-INF [rdes]:} \( A \neq \{ \} \implies \text{pre}_R(\bigcap A) = (\bigwedge P \in A \cdot \text{pre}_R(P)) \)

\text{by (rel-auto)}

\textbf{Lemma periR-INF [rdes]:} \( \text{peri}_R(\bigcap A) = (\bigvee P \in A \cdot \text{peri}_R(P)) \)

\text{by (rel-auto)}

\textbf{Lemma postR-INF [rdes]:} \( \text{post}_R(\bigcap A) = (\bigvee P \in A \cdot \text{post}_R(P)) \)

\text{by (rel-auto)}

\textbf{Lemma preR-UINF [rdes]:} \( \text{pre}_R(\bigcap i \cdot P(i)) = (\bigcup i \cdot \text{pre}_R(P(i))) \)

\text{by (rel-auto)}

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

\text{by (rel-auto)}

\textbf{Lemma postR-UINF [rdes]:} \( \text{post}_R(\bigcap i \cdot P(i)) = (\bigcap i \cdot \text{post}_R(P(i))) \)

\text{by (rel-auto)}

\textbf{Lemma preR-UINF-member [rdes]:} \( A \neq \{ \} \implies \text{pre}_R(\bigcap i \in A \cdot P(i)) = (\bigcup i \in A \cdot \text{pre}_R(P(i))) \)

\text{by (rel-auto)}

\textbf{Lemma preR-UINF-member-2 [rdes]:} \( A \neq \{ \} \implies \text{pre}_R(\bigcap (i,j) \in A \cdot P(i,j)) = (\bigcup (i,j) \in A \cdot \text{pre}_R(P(i,j))) \)

\text{by (rel-auto)}

\textbf{Lemma preR-UINF-member-3 [rdes]:} \( A \neq \{ \} \implies \text{pre}_R(\bigcap (i,j,k) \in A \cdot P(i,j,k)) = (\bigcup (i,j,k) \in A \cdot \text{pre}_R(P(i,j,k))) \)

\text{by (rel-auto)}

\textbf{Lemma periR-UINF-member [rdes]:} \( \text{peri}_R(\bigcap i \in A \cdot P(i)) = (\bigcap i \in A \cdot \text{peri}_R(P(i))) \)

\text{by (rel-auto)}

\textbf{Lemma periR-UINF-member-2 [rdes]:} \( \text{peri}_R(\bigcap (i,j) \in A \cdot P(i,j)) = (\bigcap (i,j) \in A \cdot \text{peri}_R(P(i,j))) \)

\text{by (rel-auto)}

\textbf{Lemma periR-UINF-member-3 [rdes]:} \( \text{peri}_R(\bigcap (i,j,k) \in A \cdot P(i,j,k)) = (\bigcap (i,j,k) \in A \cdot \text{peri}_R(P(i,j,k))) \)

\text{by (rel-auto)}

\textbf{Lemma postR-UINF-member [rdes]:} \( \text{post}_R(\bigcap i \in A \cdot P(i)) = (\bigcap i \in A \cdot \text{post}_R(P(i))) \)

\text{by (rel-auto)}

\textbf{Lemma postR-UINF-member-2 [rdes]:} \( \text{post}_R(\bigcap (i,j) \in A \cdot P(i,j)) = (\bigcap (i,j) \in A \cdot \text{post}_R(P(i,j))) \)

\text{by (rel-auto)}

\textbf{Lemma postR-UINF-member-3 [rdes]:} \( \text{post}_R(\bigcap (i,j,k) \in A \cdot P(i,j,k)) = (\bigcap (i,j,k) \in A \cdot \text{post}_R(P(i,j,k))) \)

\text{by (rel-auto)}

\textbf{Lemma preR-inf [rdes]:} \( \text{pre}_R(P \cap Q) = (\text{pre}_R(P) \land \text{pre}_R(Q)) \)

\text{by (rel-auto)}

\textbf{Lemma periR-inf [rdes]:} \( \text{peri}_R(P \cap Q) = (\text{peri}_R(P) \lor \text{peri}_R(Q)) \)

\text{by (rel-auto)}

\textbf{Lemma postR-inf [rdes]:} \( \text{post}_R(P \cap Q) = (\text{post}_R(P) \lor \text{post}_R(Q)) \)
4.4 Formation laws

lemma perIR-SUP [rdes]: $A \neq \emptyset \Rightarrow \peri_R(\bigcup \, A) = (\bigwedge \, P \in A \cdot \peri_R(P))$
  by (rel-auto)

lemma postIR-SUP [rdes]: $A \neq \emptyset \Rightarrow \post_R(\bigcup \, A) = (\bigwedge \, P \in A \cdot \post_R(P))$
  by (rel-auto)

4.4 Formation laws

lemma Chaos-tri-def [rdes-def]: $\text{Chaos} = R_4(\text{false} \triangleright\triangleright\triangleright \text{false})$
  by (simp add: Chaos-def design-false-pre)

lemma Miracle-tri-def [rdes-def]: $\text{Miracle} = R_4(\text{true} \triangleright\triangleright\triangleright \text{false})$
  by (simp add: Miracle-def R1-design-R1-pre-true-cond-idem)

lemma RHS-tri-design-form:
  assumes $P_1$ is RR $P_2$ is RR $P_3$ is RR
  shows $R_4(P_1 \triangleright\triangleright\triangleright P_2) \triangleright\triangleright\triangleright RR(P_3) = (II_R \triangleright\triangleright\triangleright \text{wait} \triangleright\triangleright\triangleright ((\text{ok} \land P_1) \Rightarrow ((\text{ok}) \land (P_2 \triangleright\triangleright\triangleright P_3)))$
  proof -
  have $R_4(RR(P_1) \triangleright\triangleright\triangleright RR(P_2) \triangleright\triangleright\triangleright RR(P_3)) = (II_R \triangleright\triangleright\triangleright \text{wait} \triangleright\triangleright\triangleright ((\text{ok} \land RR(P_1)) \Rightarrow ((\text{ok}) \land (RR(P_2) \triangleright\triangleright\triangleright RR(P_3))))$
  apply (rel-auto) using minus-zero-eq by blast
  thus $?\text{thesis}$
  by (simp add: Healthy-if assms)
qed

lemma RHS-design-pre-post-form:
  $R_4((\neg P^f) \triangleright\triangleright\triangleright P^f) = R_4(\pre_R(P) \triangleright\triangleright\triangleright \text{cmt}_R(P))$
  proof -
  have $R_4((\neg P^f) \triangleright\triangleright\triangleright P^f) = R_4((\neg P^f)[\text{true}/\text{ok}] \triangleright\triangleright\triangleright P^f)[\text{true}/\text{ok}]$
  by (simp add: design-subst-ok)
  also have $.. = R_4(\pre_R(P) \triangleright\triangleright\triangleright \text{cmt}_R(P))$
  by (simp add: pre-R-def cmt-R-def usubst, rel-auto)
  finally show $?\text{thesis}$ .
qed

lemma SRD-as-reactive-design:
  $\text{SRD}(P) = R_4(\pre_R(P) \triangleright\triangleright\triangleright \text{cmt}_R(P))$
  by (simp add: RHS-design-pre-post-form SRD-RH-design-form)

lemma SRD-reactive-design-alt:
  assumes $P$ is SRD
  shows $R_4(\pre_R(P) \triangleright\triangleright\triangleright \text{cmt}_R(P)) = P$
  proof -
  have $R_4(\pre_R(P) \triangleright\triangleright\triangleright \text{cmt}_R(P)) = R_4((\neg P^f) \triangleright\triangleright\triangleright P^f)$
  by (simp add: RHS-design-pre-post-form)
  thus $?\text{thesis}$
  by (simp add: SRD-reactive-design assms)
qed
lemma SRD-reactive-tri-design-lemma:
\[ \text{SRD}(P) = \mathbb{R}_s((\neg P' f) \vdash P' f[true/\$wait\ ] \circ P' f[false/\$wait\ ]) \]
by (simp add: SRD-RH-design-form wait'-cond-split)

lemma SRD-as-reactive-tri-design:
\[ \text{SRD}(P) = \mathbb{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \circ \text{post}_R(P)) \]
proof
  have \[ \text{SRD}(P) = \mathbb{R}_s((\neg P' f) \vdash P' f[true/\$wait\ ] \circ P' f[false/\$wait\ ]) \]
  by (simp add: SRD-RH-design-form wait'-cond-split)
  also have \[ \ldots = \mathbb{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \circ \text{post}_R(P)) \]
  apply (simp add: usubst)
  apply (subst design-subst-ok-ok[THEN sym])
  apply (simp add: pre_R-def peri_R-def post_R-def usubst unrest)
  apply (rel-auto)
  done
finally show \[ ?thesis \].
qed

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

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

lemma RHS-tri-design-is-SRD [closure]:
assumes \[ \$ok' \not\in P \$ok' \not\in Q \$ok' \not\in R \]
shows \[ \mathbb{R}_s(P \vdash Q \circ R) \text{ is SRD} \]
by (rule RHS-design-is-SRD, simp-all add: unrest assms)

lemma SRD-reds-intro [closure]:
assumes \[ P \text{ is RR } Q \text{ is RR } R \text{ is RR} \]
shows \[ \mathbb{R}_s(P \vdash Q \circ R) \text{ is SRD} \]
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 \[ (\bigsqcup P \in A \cdot R1 (R2s (\text{cmt}_R P))) = (\bigsqcup P \in A \cdot \text{cmt}_R P) \]
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 \[ (\bigsqcap P \in A \cdot R1 (R2s (\text{cmt}_R P))) = (\bigsqcap P \in A \cdot \text{cmt}_R P) \]
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: \[ P \subseteq Q \implies \text{peri}_R(P) \subseteq \text{peri}_R(Q) \]
by (rel-auto)

lemma postR-monotone: \[ P \subseteq Q \implies \text{post}_R(P) \subseteq \text{post}_R(Q) \]
ultimately show thesis

by (amp add: R2s-wait R1-extend-conj R2s-conj R2s-not R2s-wait)

also have ... = ((R1 (R2s Q) :: :: $\text{S} \triangleleft \text{R} \cdot \text{S} \cdot \text{R}) :: :: R2s S) \triangleright \text{R} :: :: \text{S} :: :: R2s S)

finally show thesis

by (amp add: \text{R2s-Sr-step-in-unit-unrest-unrest})

proven by (rel-auto)

\textbf{4.5 Composition laws}

\textbf{shows R1(R2s Q + Q :: :: \text{S} :: :: R2s S) = R1(R2s Q :: :: \text{S} :: :: R2s S)}

\textbf{proof}

by (metis no-types hidelams R1-extend-conj R2s-comp R2s-not R2s-wait)

by (assum(2) have ... = ((R1 (R2s Q) :: :: $\text{S} \triangleleft \text{R} \cdot \text{S} \cdot \text{R}) :: :: \text{S} :: :: R2s S))

by (amp add: \text{R2s-Sr-step-in-unit-unrest-unrest})

finally show thesis

by (amp add: \text{R2s-Sr-step-in-unit-unrest-unrest})

proven by (rel-auto)
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
dqed

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 (((¬r P) wp_r false ∧ Q wp_r 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 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 $ok' ⊢ S2
  shows  
  (R_s ((∀ R1 (¬ R2s P) ;; R1 true) ∧ ¬ (R1(R2s Q2) ;; R1 (¬ R2s R))) ⊢ 
  (((∃ $st' · Q1) ∨ (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 · [IH]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 · [IH]D) ∧ $wait ⊢ R1 (R2s S1) ⊢ R1 (R2s S2)))
  = (∃ $st' · ((R1 (R2s Q1)) ∧ $wait'))

proof –
  have ((R1 (R2s Q1)) ;; ($wait ∧ (∃ $st · [IH]D) ∧ $wait ⊢ R1 (R2s S1) ⊢ R1 (R2s S2)))
  = (R1 (R2s Q1) ;; ($wait ∧ (∃ $st · [IH]D)))
  by (rel-auto, blast+)
  also have ... = ((R1 (R2s Q1) ;; (∃ $st · [IH]D)) ∧ $wait')
  by (rel-auto)
  also from assms(2) have ... = (∃ $st' · ((R1 (R2s Q1)) ∧ $wait'))
  by (rel-auto, blast)
  finally show thesis .
dqed

moreover have
(R1 (R2s Q2) ;; ¬ $wait ∧ (∃ $st · [IH]D) ∧ $wait ⊢ R1 (R2s S1) ⊢ R1 (R2s S2)))
= ((R1 (R2s Q2)) ;; (R1 (R2s S1) ⊢ R1 (R2s S2)))
proof
  have $(R_1 \circ (R_2 \circ Q_2)) \circ \neg \text{wait} \wedge ((\exists \text{st} \cdot [H]P) < \text{wait} \circ R_1 \circ (R_2 \circ S_1) \circ R_1 \circ (R_2 \circ S_2))$
  \begin{align*}
    &= (R_1 \circ (R_2 \circ Q_2)) \circ \neg \text{wait} \wedge (R_1 \circ (R_2 \circ S_1) \circ R_1 \circ (R_2 \circ S_2)))
  \end{align*}
  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 $\ldots = ((R_1 \circ (R_2 \circ Q_2)) [\text{false}/\text{wait}] \circ (R_1 \circ (R_2 \circ S_1) \circ R_1 \circ (R_2 \circ S_2)) [\text{false}/\text{wait}]$
by (metis false-alt-def seq-right-one-point upred-eq-false wait-vub-lens)

also have $\ldots = ((R_1 \circ (R_2 \circ Q_2)) \circ (R_1 \circ (R_2 \circ S_1) \circ R_1 \circ (R_2 \circ S_2)))$
by (simp add: wait'-cond-def unsub unrest assms)

finally show $?\text{thesis}$.

qed

moreover
have $((R_1 \circ (R_2 \circ Q_2) \wedge \text{wait'}) \vee ((R_1 \circ (R_2 \circ Q_2)) \circ (R_1 \circ (R_2 \circ S_1) \circ R_1 \circ (R_2 \circ S_2)))$
\begin{align*}
  &= (R_1 \circ (R_2 \circ Q_1) \circ (R_1 \circ (R_2 \circ Q_2)) \circ (R_1 \circ (R_2 \circ S_1) \circ R_1 \circ (R_2 \circ S_2)))
  \end{align*}
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 $3: (R_1 \circ (R_2 \circ Q_2) \wedge \neg \text{wait'}) \circ R_1 \circ \neg R_2 \circ R = R_1 \circ (R_2 \circ Q_2) \circ R_1 \circ (R_2 \circ R)$
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 unrest unrest unrest unrest)
apply (simp add: 1 2 3)
apply (simp add: R1-R2s-R2c RHS-design-lemma1)
apply (metis R1-R2c-ex-st RHS-design-lemma1)
done

qed

theorem RHS-tri-design-composition-wp:
assumes $\text{ok'} \not\in P \not\in \text{ok'} \not\in Q_1 \not\in \text{ok'} \not\in Q_2 \not\in \text{ok} \not\in S_1 \not\in \text{ok} \not\in S_2$

$\text{wait} \not\in R \not\in \text{wait'} \not\in Q_2 \not\in \text{wait} \not\in S_1 \not\in \text{wait} \not\in S_2$

\begin{align*}
  \text{P is R}\circ C_1 \not\in R_1 \circ Q_1 \not\in R\circ C_2 \not\in R_1 \circ Q_2 \not\in R\circ C_2 \not\in R_1 \circ S_1 \not\in R\circ C_2 \not\in R_1 \circ S_2 \not\in R\circ C_2
  \end{align*}

shows $R_s(P \vdash Q_1 \circ Q_2) :: R_s(R \vdash S_1 \circ S_2) =$
\begin{align*}
  R_s(((\neg R \circ wp \circ \text{false} \wedge Q_2 \circ wp \circ R) \circ ((\exists \text{st} \cdot Q_1) \circ (Q_2 \circ S_1) \circ (Q_2 \circ S_2)))) \text{(is } ?\text{lhs} = ?\text{rhs})
  \end{align*}

proof

have $?\text{lhs} = R_s(((\neg R \circ P) :: R_1 \circ Q_2 :: R \circ \neg R \circ (\exists \text{st} \cdot Q_1) \circ (Q_2 :: S_1) \circ (Q_2 :: S_2))$
by (simp add: RHS-tri-design-composition assms Healthy-if R2c-healthy-R2s disj-upred-def)
(metis (no-types, hide-lams) R1-negate-R1 R2c-healthy-R2s assms(11,16))

also have $\ldots = ?\text{rhs}$
by (rel-auto)

39
finally show thesis.

qed

\textbf{theorem} \( \text{RHS-tri-design-composition-RR-wp:} \)
\textbf{assumes} \( P \) is RR \( Q_1 \) is RR \( Q_2 \) is RR
\( R \) is RR \( S_1 \) is RR \( S_2 \) is RR
\textbf{shows} \( R_s(P \vdash Q_1 \circ Q_2) ; R_s(R \vdash S_1 \circ S_2) = R_s((P \circ Q_2 \circ R) \vdash (((\exists \mathbf{st} \cdot Q_1) \cap (Q_2 ; S_1)) \circ (Q_2 ; S_2))) \) (is thesis = ?rhs)
\textbf{by} (simp add: RHS-tri-design-composition-wp add: closure assms unrest RR-implies-R2c)

\textbf{lemma} \( \text{RHS-tri-normal-design-composition:} \)
\textbf{assumes} \( \$ok' \not\vdash P \$ok' \not\vdash Q_1 \$ok' \not\vdash Q_2 \$ok \not\vdash R \$ok \not\vdash S_1 \$ok \not\vdash S_2 \$
\( \$wait \not\vdash R \$wait \not\vdash Q_1 \$wait \not\vdash Q_2 \$wait \not\vdash S_1 \$wait \not\vdash S_2 \$
\( P \) is RR \( Q_1 \) is RR \( Q_2 \) is RR \( R \) is RR \( S_1 \) is RR \( S_2 \) is RR
\( R \) is RR \( S_1 \) is RR \( S_2 \) is RR \( R \) is RR
\( R \vdash (\neg P) ; R(\vdash (\neg P) \not\vdash Q_1) \)
\textbf{shows} \( R_s(P \vdash Q_1 \circ Q_2) ; R_s(R \vdash S_1 \circ S_2) = R_s((P \circ Q_2 \circ R) \vdash (Q_1 \cap (Q_2 ; S_1)) \circ (Q_2 ; S_2)) \)
\textbf{proof} –
\textbf{have} \( R_s(P \vdash Q_1 \circ Q_2) ; R_s(R \vdash S_1 \circ S_2) = R_s((P \circ Q_2 \circ R) \vdash (Q_1 \cap (Q_2 ; S_1)) \circ (Q_2 ; S_2)) \)
\textbf{by} (simp-all add: RHS-tri-design-composition-wp rea-not-def assms unrest)
\textbf{also have} \( \vdash R_s((P \circ Q_2 \circ R) \vdash (Q_1 \cap (Q_2 ; S_1)) \circ (Q_2 ; S_2)) \)
\textbf{by} (simp add: assms wp-rea-def ex-unrest, rel-auto)
\textbf{finally show} thesis.

qed

\textbf{lemma} \( \text{RHS-tri-normal-design-composition'} [rdes-def]: \)
\textbf{assumes} \( \$ok' \not\vdash P \$ok' \not\vdash Q \$ok' \not\vdash Q_2 \$ok \not\vdash R \$ok \not\vdash S \$
\( \$wait \not\vdash R \$wait \not\vdash Q_2 \$wait \not\vdash S \$
\( P \) is RR \( Q_1 \) is RR \( Q_2 \) is RR \( R \) is RR \( S_1 \) is RR \( S_2 \) is RR
\textbf{shows} \( R_s(P \vdash Q_1 \circ Q_2) ; R_s(R \vdash S_1 \circ S_2) = R_s((P \circ Q_2 \circ R) \vdash (Q_1 \cap (Q_2 ; S_1)) \circ (Q_2 ; S_2)) \)
\textbf{proof} –
\textbf{have} \( R_s(P \vdash Q_1 \circ Q_2) ; R_s(R \vdash S_1 \circ S_2) = R_s((P \circ Q_2 \circ R) \vdash (Q_1 \cap (Q_2 ; S_1)) \circ (Q_2 ; S_2)) \)
\textbf{by} (simp add: Healthy-def RC1-def rea-not-def)
\( \text{(metis R1-negate-R1 R1-seqR utp-pred-laws.double-compl)} \)
\textbf{thus} thesis
\textbf{by} (simp add: RHS-tri-normal-design-composition assms closure unrest RR-implies-R2c)

qed

\textbf{lemma} \( \text{RHS-tri-design-right-unit-lemma:} \)
\textbf{assumes} \( \$ok' \not\vdash P \$ok' \not\vdash Q \$ok' \not\vdash R \$wait' \not\vdash R \$
\( \neg \vdash P \neg \vdash Q \neg \vdash R \neg \vdash S \$
\textbf{shows} \( R_s(P \vdash Q \circ R) ; II_R = R_s((\neg P \neg R) \vdash (\exists \mathbf{st} \cdot Q) \circ R) \)
\textbf{proof} –
\textbf{have} \( R_s(P \vdash Q \circ R) ; II_R = R_s((\neg P \neg R) \vdash (\exists \mathbf{st} \cdot Q) \circ R) \)
\textbf{by} (simp add: srdes-skip-tri-design, rel-auto)
\textbf{also have} \( \vdash R_s((\neg R \neg P \circ R) ; R(\vdash (\exists \mathbf{st} \cdot Q) \circ R) \)
\textbf{by} (simp-all add: RHS-tri-design-composition assms unrest R2s-true R1-false R2s-false)
\textbf{also have} \( \vdash R_s((\neg R \neg P \circ R) ; R(\vdash (\exists \mathbf{st} \cdot Q) \circ R) \)
\textbf{proof} –
\textbf{from} \text{assms(3,4)} \text{have} \( (R(\vdash (\exists \mathbf{st} \cdot Q) \circ R) \)
\textbf{by} (rel-auto, metis (no-types, lifting) minus-zero-eq, meson order-refl trace-class.diff-cancel)

40
Thus \( ?\text{thesis} \)

by simp

qed

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

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)

also have \( ... = \text{R}_s((\neg_r (\neg_r P) ;; \text{true}_r) \vdash ((\exists \$st' \cdot Q) \circ R)) \)

by (rel-auto)

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

qed

lemma \( \text{SRD-composition-wp} \):

assumes \( P \) is SRD \( Q \) is SRD

shows \( (P ;; Q) = \text{R}_s(((\neg_r \text{ pre}_R P) \circ \text{ wp}_r \text{ false} \land \text{ post}_R P \circ \text{ wp}_r \text{ pre}_R Q) \vdash ((\exists \$st' \cdot \text{ peri}_R P) \lor \text{ post}_R P ;; \text{ peri}_R Q)) \circ \text{ post}_R Q) \)

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

proof –

have \( (P ;; Q) = (\text{R}_s(\text{pre}_R(P) \vdash \text{ peri}_R(P) \circ \text{ post}_R(P)) ;; \text{R}_s(\text{pre}_R(Q) \vdash \text{ peri}_R(Q) \circ \text{ post}_R(Q))) \)

by (simp add: SRD-reactive-tri-design \( \text{assms} \)(1) \( \text{assms} \)(2))

also from \( \text{assms} \)

have \( ... = ?\text{rhs} \)

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

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

qed

4.6 Refinement introduction laws

lemma \( \text{RHS-tri-design-refine} \):

assumes \( P_1 \) is RR \( P_2 \) is RR \( P_3 \) is RR \( Q_1 \) is RR \( Q_2 \) is RR \( Q_3 \) is RR

shows \( \text{R}_s(P_1 \vdash P_2 \circ P_3) \subseteq \text{R}_s(Q_1 \vdash Q_2 \circ Q_3) \iff 'P_1 \Rightarrow Q_1' \land 'P_1 \land Q_2 \Rightarrow P_2' \land 'P_1 \land Q_3 \Rightarrow P_3' \)

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

proof –

have \( ?\text{lhs} \iff 'P_1 \Rightarrow Q_1' \land 'P_1 \land Q_2 \Rightarrow P_2' \land 'P_1 \land Q_3 \Rightarrow P_3' \)

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

also have \( ... \iff 'P_1 \Rightarrow Q_1' \land 'P_2 \circ P_3' \land 'P_1 \land Q_2 \Rightarrow P_2' \land 'P_1 \land Q_3 \Rightarrow P_3' \)

by (rel-auto)

also have \( ... \iff 'P_1 \Rightarrow Q_1' \land 'P_2 \circ P_3' \land 'P_1 \land Q_2 \Rightarrow P_2' \land 'P_1 \land Q_3 \Rightarrow P_3' \)

by (rel-auto, metis)

also have \( ... \iff ?\text{rhs} \)

by (simp add: usubst unrest \( \text{assms} \))

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

qed

lemma \( \text{srdes-tri-refine-intro} \):

assumes '\( P_1 \Rightarrow P_2 \)' '\( P_1 \land Q_2 \Rightarrow Q_1' '\( P_1 \land R_2 \Rightarrow R_1' \\

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

using \( \text{assms} \)

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

lemma \( \text{srdes-tri-eq-intro} \):

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

shows \( \text{R}_s(P_1 \vdash P_2 \circ P_3) = \text{R}_s(Q_1 \vdash Q_2 \circ Q_3) \)

using \( \text{assms} \) by (simp)
lemma \emph{srdes-tri-refine-intro}:
\begin{itemize}
\item assumes \( P_2 \subseteq P_1, Q_1 \subseteq (P_1 \land Q_2), R_1 \subseteq (P_1 \land R_2) \)
\item shows \( R_s(P_1 \leftrightarrow Q_1 \circ R_1) \subseteq R_s(P_2 \leftrightarrow Q_2 \circ R_2) \)
\end{itemize}
using \texttt{assms}
by (\texttt{rule-tac srdes-tri-refine-intro}, \texttt{simp-all add: refBy-order})

lemma \emph{SRD-peri-under-pre}:
\begin{itemize}
\item assumes \( P \) is SRD \( \$\text{wait} \$ \not\in \pre_R(P) \)
\item shows \( \pre_R(P) \Rightarrow R \peri_R(P) = \peri_R(P) \)
\end{itemize}
proof
\begin{itemize}
\item have \( \peri_R(P) = \peri_R(R_s(\pre_R(P) \vdash \peri_R(P) \circ \post_R(P))) \)
\item by (\texttt{simp add: SRD-reactive-tri-design \texttt{assms}})
\end{itemize}
also have \( ... = (\pre_R P \Rightarrow \peri_R P) \)
\begin{itemize}
\item by (\texttt{simp add: rea-pre-RHS-design rea-peri-RHS-design \texttt{assms}})
\end{itemize}
unrest \texttt{usubst \texttt{R1-peri-SRD} \texttt{R2c-preR} \texttt{R1-rea-impl} \texttt{R2c-rea-impl} \texttt{R2c-periR}}
\begin{itemize}
\item finally show \( \texttt{?thesis} \)
\end{itemize}
qed

lemma \emph{SRD-post-under-pre}:
\begin{itemize}
\item assumes \( P \) is SRD \( \$\text{wait} \$ \not\in \pre_R(P) \)
\item shows \( \pre_R(P) \Rightarrow R \post_R(P) = \post_R(P) \)
\end{itemize}
proof
\begin{itemize}
\item have \( \post_R(P) = \post_R(R_s(\pre_R(P) \vdash \peri_R(P) \circ \post_R(P))) \)
\item by (\texttt{simp add: SRD-reactive-tri-design \texttt{assms}})
\end{itemize}
also have \( ... = (\pre_R P \Rightarrow \post_R P) \)
\begin{itemize}
\item by (\texttt{simp add: rea-pre-RHS-design rea-post-RHS-design \texttt{assms}})
\end{itemize}
unrest \texttt{usubst \texttt{R1-post-SRD} \texttt{R2c-preR} \texttt{R1-rea-impl} \texttt{R2c-rea-impl} \texttt{R2c-postR}}
\begin{itemize}
\item finally show \( \texttt{?thesis} \)
\end{itemize}
qed

lemma \emph{SRD-refine-intro}:
\begin{itemize}
\item assumes \( P \in \text{SRD} \ Q \in \text{SRD} \)
\item \( \pre_R(P) \Rightarrow \pre_R(Q) \) \( \pre_R(P) \land \peri_R(Q) \Rightarrow \peri_R(P) \land \peri_R(Q) \) \( \pre_R(P) \land \post_R(Q) \Rightarrow \post_R(P) \)
\item shows \( P \subseteq Q \)
\item by (\texttt{metis SRD-reactive-tri-design \texttt{assms(1) assms(2) assms(3) assms(4) assms(5) srdes-tri-refine-intro}})
\end{itemize}

lemma \emph{SRD-refine-intro}:
\begin{itemize}
\item assumes \( P \in \text{SRD} \ Q \in \text{SRD} \)
\item \( \pre_R(P) \Rightarrow \pre_R(Q) \) \( \peri_R(P) \subseteq (\pre_R(P) \land \peri_R(Q)) \) \( \post_R(P) \subseteq (\pre_R(P) \land \post_R(Q)) \)
\item shows \( P \subseteq Q \)
\item using \texttt{assms} by (\texttt{rule-tac SRD-refine-intro}, \texttt{simp-all add: refBy-order})
\end{itemize}

lemma \emph{SRD-eq-intro}:
\begin{itemize}
\item assumes \( P \in \text{SRD} \ Q \in \text{SRD} \)
\item \( \pre_R(P) = \pre_R(Q) \) \( \peri_R(P) = \peri_R(Q) \) \( \post_R(P) = \post_R(Q) \)
\item shows \( P = Q \)
\item by (\texttt{metis SRD-reactive-tri-design \texttt{assms}})
\end{itemize}

4.7 Closure laws

lemma \emph{SRD-srdes-skip [closure]}: \( \Pi_R \) is SRD
\begin{itemize}
\item by (\texttt{simp add: srdes-skip-def RHS-design-is-SRD unrest})
\end{itemize}
lemma SRD-seqr-closure [closure]:
assumes $P$ is SRD $Q$ is SRD
shows $(P ;; Q)$ is SRD
proof –
  have $(P ;; Q) = \mathbf{R_s} (((\neg \text{pre}_R P) wp \text{false} \land post_R P wp \text{pre}_R Q) \vdash
                              ((\exists \text{st}' \cdot \text{peri}_R P) \lor post_R P ;; \text{peri}_R Q) \circ post_R P ;; post_R Q)$
    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 $\Rightarrow P ;; P^\text{n}$ 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 $A \subseteq \{ \text{SRD} \}$ $A \neq \{ \}$
shows $(\bigsqcup A)$ is SRD
proof –
  have $\text{SRD } (\bigsqcup A) = (\bigsqcup (\text{SRD } 'A))$
    by (simp add: ContinuousD SRD-Continuous assms(2))
also have ... = $(\bigsqcup A)$
    by (simp only: Healthy-carrier-image assms)
finally show ?thesis by (simp add: Healthy-def)
qed

4.8 Distribution laws

lemma RHS-tri-design-choice [rdes-def]:
$\mathbf{R_s}((P_1 ;; P_2 \circ P_3) \cap \mathbf{R_s}(Q_1 ;; Q_2 \circ Q_3) = \mathbf{R_s}((P_1 \land Q_1) \vdash (P_2 \lor Q_2) \circ (P_3 \lor Q_3))$
apply (simp add: RHS-design-choice)
apply (rule cong[of $\mathbf{R_s}$ $\mathbf{R_s}$])
apply (simp)
apply (rel-auto)
done

lemma RHS-tri-design-disj [rdes-def]:
$(\mathbf{R_s}((P_1 ;; P_2 \circ P_3) \lor \mathbf{R_s}(Q_1 ;; Q_2 \circ Q_3)) = \mathbf{R_s}((P_1 \lor Q_1) \vdash (P_2 \lor Q_2) \circ (P_3 \lor Q_3))$
by (simp add: RHS-tri-design-choice disj-upred-def)

lemma RHS-tri-design-sup [rdes-def]:
\[ \text{R}_s(P_1 \vdash P_2 \land P_3) \cup \text{R}_s(Q_1 \vdash Q_2 \land Q_3) = \text{R}_s((P_1 \lor Q_1) \vdash ((P_1 \Rightarrow P_2) \land (Q_1 \Rightarrow Q_2)) \circ ((P_1 \Rightarrow P_3) \land (Q_1 \Rightarrow Q_3))) \]

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

lemma RHS-tri-design-conj [rdes-def]:
\[ (\text{R}_s(P_1 \vdash P_2 \land P_3) \land \text{R}_s(Q_1 \vdash Q_2 \land Q_3)) = \text{R}_s(((P_1 \lor Q_1) \vdash ((P_1 \Rightarrow P_2) \land (Q_1 \Rightarrow Q_2)) \circ ((P_1 \Rightarrow P_3) \land (Q_1 \Rightarrow Q_3))) \]

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

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

proof
\[ \text{have } \bigwedge \cdot A = \text{R}_s((\bigwedge \cdot P \in A \cdot \text{pre}_R(P)) \vdash (\bigwedge \cdot P \in A \cdot \text{peri}_R(P)) \circ (\bigwedge \cdot P \in A \cdot \text{post}_R(P))) \]

by (metis SRD-as-reactive-tri-design assms srdes-hcond-def srdes-theory-continuous.healthy-inf srdes-theory-continuous.healthy-inf-def)

also have \[ \ldots = \text{R}_s((\bigwedge \cdot P \in A \cdot \text{pre}_R(P)) \vdash (\bigwedge \cdot P \in A \cdot \text{peri}_R(P)) \circ (\bigwedge \cdot P \in A \cdot \text{post}_R(P))) \]

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

finally show \[ ?\text{thesis} \].
qed

lemma RHS-tri-design-USUP [rdes-def]:
\[ \text{assumes } A \neq {} \land i. P \ i \text{ is SRD} \]
\[ \text{shows } \bigwedge \cdot i \in A \cdot \text{R}_s(P(i) \vdash Q(i) \circ R(i)) = \text{R}_s((\bigwedge \cdot i \in A \cdot P(i)) \vdash (\bigwedge \cdot i \in A \cdot Q(i)) \circ (\bigwedge \cdot i \in A \cdot R(i))) \]

by (simp add: design-USINF[of assms, THEN sym], subst RHS-UINF[OF assms])

lemma SRD-UINF-mem:
\[ \text{assumes } A \neq {} \land i. P \ i \text{ is SRD} \]
\[ \text{shows } \bigwedge \cdot i \in A \cdot P(i) = \text{R}_s((\bigwedge \cdot i \in A \cdot \text{pre}_R(P(i)) \vdash (\bigwedge \cdot i \in A \cdot \text{peri}_R(P(i)) \circ (\bigwedge \cdot i \in A \cdot \text{post}_R(P(i)))) \]

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

proof
\[ \text{have } ?\text{lhs} = (\bigwedge \cdot (P \ i)) \]

by (simp-auto)

also have \[ \ldots = \text{R}_s((\bigwedge \cdot Pa \in P \ i. Pa) = (\bigwedge \cdot Pa \in P \ i. Pa) \circ (\bigwedge \cdot Pa \in P \ i. Pa)) \]

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

also have \[ \ldots = ?\text{rhs} \]

by (simp-auto)

finally show \[ ?\text{thesis} \].
qed

lemma RHS-tri-design-UINF-ind [rdes-def]:
\[ (\bigwedge \cdot i \cdot \text{R}_s(P_1(i) \vdash P_2(i) \circ P_3(i))) = \text{R}_s((\bigwedge \cdot i \cdot P_1 i) \vdash (\bigwedge \cdot i \cdot P_2(i)) \circ (\bigwedge \cdot i \cdot P_3(i))) \]

by (simp-auto)

lemma cond-srea-form [rdes-def]:
\[ \text{R}_s(P \vdash Q_1 \circ Q_2) \circ b \vdash R \text{R}_s(R \vdash S_1 \circ S_2) = \]
\[ \text{R}_s((P \circ b \vdash R) \vdash (Q_1 \circ b \vdash S_1) \circ (Q_2 \circ b \vdash S_2)) \]

proof
\[ \text{have } \text{R}_s(P \vdash Q_1 \circ Q_2) \circ b \vdash R \text{R}_s(R \vdash S_1 \circ S_2) = \text{R}_s(P \vdash Q_1 \circ Q_2) \circ R2c(\cdot b_{S_2} < c) \circ \text{R}_s(R \vdash S_1 \circ S_2) \]

by (pred-auto)

also have \[ \ldots = \text{R}_s(P \vdash Q_1 \circ Q_2 \circ b \vdash R \vdash S_1 \circ S_2) \]

by (simp add: RHS-cond lift-cond-srea-def)
also have \( \ldots = R_s((P \looparrowright b \triangleright R) \vdash (Q_1 \bowtie b \bowtie_R S_1 \bowtie S_2)) \)
by (simp add: \( \text{design-condr lift-cond-srea-def} \))
also have \( \ldots = R_s((P \bowtie b \bowtie_R R) \vdash (Q_1 \bowtie b \bowtie_R S_1 \bowtie (Q_2 \bowtie b \bowtie_R S_2)) \)
by (rule cong[of \( R_s \), simp, rel-auto])
finally show \(?thesis\).
qed

lemma \textit{SRD-cond-srea [closure]}:
assumes \( P \) is SRD \( Q \) is SRD
shows \( P \bowtie b \bowtie_R Q \) is SRD
proof
  have \( P \bowtie b \bowtie_R Q = R_s(\pre_R(P) \vdash \peri_R(P) \circ \post_R(P)) \bowtie b \bowtie_R R_s(\pre_R(Q) \vdash \peri_R(Q) \circ \post_R(Q)) \)
  by (simp add: \( \text{SRD-reactive-tri-design assms} \))
  also have \( \ldots = R_s(\pre_R(P) \bowtie b \bowtie_R \peri_R(P) \bowtie \post_R(P)) \)
  by (simp add: \( \text{cond-srea-form} \))
  also have \( \ldots \) is SRD
  by (simp add: \( \text{RHS-design-composition REST SRD lift-cond-srea-def unrest} \))
finally show \(?thesis\).
qed

4.9 Algebraic laws

lemma \textit{SRD-left-unit}:
assumes \( P \) is SRD
shows \( \mathit{II}_R = P \)
by (simp add: \( \text{SRD-composition-wp closure rdes wp C1 R1-negate-R1 R1-false rpred trace-ident-left-periR trace-ident-left-postR SRD-reactive-tri-design assms} \))

lemma \textit{skip-srea-self-unit [simp]}:
\( \mathit{II}_R = \mathit{II}_R \)
by (simp add: \( \text{SRD-left-unit closure} \))

lemma \textit{SRD-right-unit-tri-lemma}:
assumes \( P \) is SRD
shows \( P = R_s(\per_R(P) \bowtie \false \vdash \true) \bowtie R_s(\pre_R(P) \vdash \peri_R(P)) \)
by (simp add: \( \text{SRD-composition-wp closure rdes wp rpred trace-ident-right-postR SRD-reactive-tri-design assms} \))

lemma \textit{Miracle-left-zero}:
assumes \( P \) is SRD
shows \( \mathit{Miracle} = \mathit{Miracle} \)
proof
  have \( \mathit{Miracle} = P = R_s(\true \vdash \false) \bowtie R_s(\pre_R(P) \vdash \peri_R(P)) \)
  by (simp add: \( \text{Miracle-def SRD-reactive-design-alt assms} \))
  also have \( \ldots = R_s(\true \vdash \false) \)
  by (simp add: \( \text{RHS-design-composition unrest R1-false R2s-false R2s-true} \))
  also have \( \ldots = \mathit{Miracle} \)
  by (simp add: \( \text{Miracle-def} \))
finally show \(?thesis\).
qed

lemma \textit{Chaos-left-zero}:
assumes \( P \) is SRD
shows \( \mathit{Chaos} = \mathit{Chaos} \)
proof
  have \( \mathit{Chaos} = P = R_s(\false \vdash \true) \bowtie R_s(\pre_R(P) \vdash \peri_R(P)) \)
  by (simp add: \( \text{Miracle-def SRD-reactive-design-alt assms} \))
finally show \(?thesis\).
qed
4.10 Recursion laws

Stateful reactive designs are left unital

lema SRD-right-Miracle-tri-lemma

proof

by (simp add: Chaos-def) also have也成为 chaos also have chaos else... by (simp add: Chaos-def)

also have... = R, false = true

by (simp add: Chaos-def) also have... = R, false = true

also have... = R, false = true

by (simp add: Chaos-def) also have... = R, false = true

also have... = R, false = true

end

end

end
done

lemma mu-srd-iter:
assumes mono F F ∈ [SRD]H → [SRD]H
shows (μ X · R₄(pre(F(X)) ⊨ peri(F(X)) ⊠ post(F(X)))) = F(μ X · R₄(pre(F(X)) ⊨ peri(F(X)) ⊠ post(F(X))))
apply (subt gfp-unfold)
apply (simp add: mono-srd-iter assms)
apply (subst SRD-as-reactive-tri-design [THEN sym])
using Healthy-func assms (1) assms (2) mu-srd-SRD
apply blast
done

lemma mu-srd-form:
assumes mono F F ∈ [SRD]H → [SRD]H
shows μR F = (μ X · R₄(pre(F(X)) ⊨ peri(F(X)) ⊠ post(F(X))))

proof −
have 1: F (μ X · R₄(pre(F(X)) ⊨ peri(F(X)) ⊠ post(F(X)))) is SRD
  by (simp add: Healthy-apply-closed assms (1) assms (2) mu-srd-SRD)

have 2: Mono_uthy-order SRDES F
  by (simp add: assms (1) mono-Monotone-utp-order)

hence 3: μR F = F (μR F)
  by (simp add: srdes-theory-continuous.LFP-unfold [THEN sym] assms)

using SRD-reactive-tri-design by force
hence (μ X · R₄(pre(F(X)) ⊨ peri(F(X)) ⊠ post(F(X))) ⊆ F (μR F))
  by (simp add: 2 srdes-theory-continuous.weak.LFP-lemma3 gfp-upperbound assms)

thus ?thesis
  using assms 1 3 srdes-theory-continuous.weak.LFP-lowerbound eq-iff mu-srd-iter
  by (metis (mono-tags, lifting))

qed

lemma Monotonic-SRD-comp [closure]: Monotonic (op ;; P o SRD)
  by (simp add: mono-def R1-R2c-is-R2 R2-mono R₃h-mono RD1-mono RD2-mono RHS-def SRD-def seqr-mono)

end

5 Normal Reactive Designs

theory utp-rdes-normal
imports
  utp-rdes-triples
  UTP−KAT.utp-kleene
begin

This additional healthiness condition is analogous to H3

definition RD3 where
  [upred-defs]: RD3(P) = P ;; II_R

lemma RD3-idem: RD3(RD3(P)) = RD3(P)
proof −
  have a: II_R ;; II_R = II_R
    by (simp add: SRD-left-unit SRD-srdes-skip)
  show ?thesis
  by (simp add: RD3-def seqr-assoc a)
\[ \text{Lemma RD3-Idempotent} \text{ [closure]: Idempotent RD3} \]
\begin{itemize}
\item by (simp add: Idempotent-def RD3-idem)
\end{itemize}

\[ \text{Lemma RD3-continuous: RD3}(\bigcap A) = (\bigcap P \in A \cdot RD3(P)) \]
\begin{itemize}
\item by (simp add: RD3-def seq-Sup-distr)
\end{itemize}

\[ \text{Lemma RD3-Continuous} \text{ [closure]: Continuous RD3} \]
\begin{itemize}
\item by (simp add: Continuous-def RD3-continuous)
\end{itemize}

\[ \text{Lemma RD3-right-subsumes-RD2: RD2}(RD3(P)) = RD3(P) \]
\begin{itemize}
\item proof –
\item have \( a: II_R :: J = II_R \)
\item by (rel-auto)
\item show \( \text{thesis} \)
\item by (metis (no-types, hide-lams) H2-def RD2-def RD3-def a seqr-assoc)
\end{itemize}

\[ \text{Lemma RD3-left-subsumes-RD2: RD3}(RD2(P)) = RD3(P) \]
\begin{itemize}
\item proof –
\item have \( a: J :: II_R = II_R \)
\item by (rel-simp, safe, blast+)
\item show \( \text{thesis} \)
\item by (metis (no-types, hide-lams) H2-def RD2-def RD3-def a seqr-assoc)
\end{itemize}

\[ \text{Lemma RD3-implies-RD2: P is RD3 } \longrightarrow \text{ P is RD2} \]
\begin{itemize}
\item by (metis Healthy-def RD3-right-subsumes-RD2)
\end{itemize}

\[ \text{Lemma RD3-intro-pre:} \]
\begin{itemize}
\item assumes \( P \text{ is SRD } \neg_r \text{ pre}_R(P) \); \( \text{true}_r = (\neg_r \text{ pre}_R(P) \cdot \exists st' \ni \text{ peri}_R(P) \) \)
\item shows \( P \text{ is RD3} \)
\item proof –
\item have \( RD3(P) = R_s((\neg_r \text{ pre}_R P) \cdot \text{ wp}_r \ni (\exists st' \cdot \text{ peri}_R P) \ni \text{ peri}_R P \cdot \text{ post}_R P) \)
\item by (simp add: RD3-def SRD-right-unit-tri-lemma assms)
\item also have \( ... = R_s((\neg_r \text{ pre}_R P) \ni \text{ peri}_R P \ni \text{ post}_R P) \)
\item by (simp add: assms(3) ex-unrest)
\item also have \( ... = R_s((\neg_r \text{ pre}_R P) \ni \text{ peri}_R P \ni \text{ cmt}_R P) \)
\item by (simp add: wait-cond-peri-post-cmt)
\item also have \( ... = R_s(\text{ peri}_R P \ni \text{ cmt}_R P) \)
\item by (simp add: assms(2) rpred wp-rea-def R1-preR)
\item finally show \( \text{thesis} \)
\item by (metis Healthy-def SRD-as-reactive-design assms(1))
\end{itemize}

\[ \text{Lemma RHS-tri-design-right-unit-lemma:} \]
\begin{itemize}
\item assumes \( \text{ok'} \ni P \ni Q \ni \text{ok'} \ni R \ni \text{wait'} \ni R \)
\item shows \( R_s(P \ni Q \ni R) :: II_R = R_s((\neg_r \text{ pre}_R P) \ni \text{ true}_r) = (\exists st' \cdot Q) \ni (\text{ peri}_R P \ni \text{ post}_R P) \)
\item proof –
\item have \( R_s(P \ni Q \ni R) :: II_R = R_s((\neg_r \text{ pre}_R P) \ni \text{ true}_r) = (\exists st' \ni Q) \ni (\text{ peri}_R P \ni \text{ post}_R P) \)
\item by (simp add: srdes-skip-tri-design, rel-auto)
\item also have \( ... = R_s((\neg_r R1 \ni (\neg_r 2s P) \ni \text{ R1 true}) = (\exists st' \cdot Q) \ni (\text{ peri}_R P \ni \text{ post}_R P) \ni (\text{ peri}_R P \ni \text{ post}_R P) \)
\item by (simp-all add: RHS-tri-design-composition assms unrest R2s-true R1-false R2s-false)
also have ... = \text{R}_u((\neg R1 (\neg R2s P) :: R1 \text{ true}) \vdash (\exists \text{st'} \cdot Q) \circ R1 (R2s R))

proof 
  from \text{assms}(3,4) have (R1 (R2s R) :: R1 (R2s ($\text{str'} =_u \text{str} \land [\mathcal{I}]_R))) = R1 (R2s R)
  by (rel-auto, metis (no-types, lifting) minus-zero-eq, meson order-refl trace-class.diff-cancel)
  thus ?thesis
  by simp
qed

also have ... = \text{R}_u((\neg \neg P) :: R1 \text{ true}) \vdash ((\exists \text{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)

also have ... = \text{R}_u((\neg_r (\neg_r P) :: \text{true}_r) \vdash ((\exists \text{st'} \cdot Q) \circ R))
by (rel-auto)
finally show ?thesis.
qed

lemma RHS-tri-design-RD3-intro:
assumes $\text{ok'} \not\equiv P \text{ok'} \not\equiv Q \text{ok'} \not\equiv R \text{st'} \not\equiv Q \text{wait'} \not\equiv R$
  \text{P is R1 (\neg P) :: true}_r = (\neg \neg P)
shows \text{R}_u(P \vdash Q \circ R) is RD3
apply (simp add: Healthy-def RD3-def)
apply (subst RHS-tri-design-right-unit-lemma)
apply (simp-all add: assms ex-unrest rpred)
done

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

lemma R1-right-unit-lemma:
\[(\text{oaut } \not\equiv e) \implies (\neg_r b \lor \text{str'} \not\equiv e \leq_\text{u} \text{str'}) :: R1(\text{true}) = (\neg_r b \lor \text{str'} \not\equiv e \leq_\text{u} \text{str'})\]
by (rel-auto, blast, metis (no-types, lifting) dual-order.trans)

lemma RHS-tri-design-RD3-intro-form:
assumes $\text{oaut } \not\equiv b \text{ oaut } \not\equiv e \text{ oaut } \not\equiv Q \text{ oaut } \not\equiv R \text{ oaut } \not\equiv R$
shows \text{R}_u((b \land \neg_r \text{str'} \not\equiv e \leq_\text{u} \text{str'}) \vdash Q \circ R) is RD3
apply (rule RHS-tri-design-RD3-intro)
apply (simp-all add: assms unrest closure rpred)
apply (subst R1-right-unit-lemma)
apply (simp-all add: assms unrest)
done

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

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

lemma NSRD-is-SRD [closure]: P is NSRD \implies P is SRD
by (simp add: Healthy-def NSRD-def SRD-def, metis Healthy-def RD1-RD3-commute RD2-RHS-commute RD3-def RD3-right-subsumes-RD2 SRD-def SRD-idem SRD-seqr-closure SRD-srdes-skip)

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

49
lemma **NSRD-Idempotent** [closure]: Idempotent NSRD  

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

lemma **NSRD-form**:

\[ NSRD(P) = R_\ast ((\neg_r (\neg_r pre_R(P)) ; R1 true) \vdash ((\exists st' \cdot peri_R(P) \circ post_R(P))) \]

proof –

have \[ NSRD(P) = RD3(SRD(P)) \]

by (metis (no-types, lifting) NSRD-def RD1-RD3-commute RD3-left-subsumes-RD2 SRD-def comp-def)

also have \[ ... = RD3(R_\ast (pre_R(P) \vdash peri_R(P) \circ post_R(P))) \]

by (clarsimp simp add: SRD-as-reactive-tri-design)

also have \[ ... = R_\ast (pre_R(P) \vdash peri_R(P) \circ post_R(P)) ; II \]

by (clarsimp simp add: RD3-def)

also have \[ ... = R_\ast ((\neg_r (\neg_r pre_R(P)) ; R1 true) \vdash ((\exists st' \cdot peri_R(P) \circ post_R(P))) \]

by (clarsimp simp add: Healthy-def NRSD-assms)

finally show \[ thesis \]

qed

lemma **NSRD-healthy-form**:

assumes \( P \) is NSRD

shows \( R_\ast ((\neg_r (\neg_r pre_R(P)) ; R1 true) \vdash ((\exists st' \cdot peri_R(P) \circ post_R(P))) = P \)

by (clarsimp simp add: Healthy-def NRSD-assms)

lemma **NSRD-Sup-closure** [closure]:

assumes \( A \subseteq \{NSRD\}_H \) \( A \neq \{\} \)

shows \( \bigsqcap A \) is NSRD

proof –

have \( NSRD (\bigsqcap A) = (\bigsqcap (NSRD \cdot A)) \)

by (clarsimp simp add: ContinuousD NSRD-Continuous assms)

also have \( ... = (\bigsqcap A) \)

by (clarsimp simp add: Healthy-carrier-image assms)

finally show \[ thesis \] by (clarsimp 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 (clarsimp simp add: assms)

lemma **NSRD-SUP-closure** [closure]:

\[ \bigwedge i. i \in A \Longrightarrow P(i) \text{ is NSRD} ; A \neq \{\} \] \( \Longrightarrow (\bigsqcap i \in A. P(i)) \) is NSRD

by (rule NSRD-Sup-closure, auto)

lemma **NSRD-neg-pre-unit**:  

assumes \( P \) is NSRD

shows \( (\neg_r pre_R(P)) ; true_r = (\neg_r pre_R(P)) \)

proof –

have \( (\neg_r pre_R(P)) = (\neg_r pre_R(R_\ast ((\neg_r (\neg_r pre_R(P)) ; R1 true) \vdash ((\exists st' \cdot peri_R(P) \circ post_R(P)))))) \)

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

also have \[ ... = R1 (R2c ((\neg_r p R) ;; R1 true)) \]

50
by (simp add: rea-pre-RHS-design R1-negate-R1 R1-idem R1-rea-not' R2c-rea-not usubst 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)

lemma NSRD-st'-unrest-peri [unrest]:
assumes P is NSRD
shows $st´ ♯ peri_R(P)
proof –
  have peri_R(P) = peri_R(Rs((¬_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 (¬_r pre_R(P)) ;; R1 true ⇒ (∃ $st´ ♯ peri_R(P))))
  by (simp add: rea-peri-RHS-design usubst 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(Rs((¬_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 (¬_r pre_R(P)) ;; R1 true))
  by (simp add: rea-pre-RHS-design usubst 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(Rs((¬_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 (¬_r pre_R(P)) ;; R1 true))
  by (simp add: rea-pre-RHS-design usubst 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 preR-RR [closure]: $P$ is NSRD $\implies$ 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 pre$_R(P)$ is RC 
by (rule RC-intro, simp-all add: closure assms NSRD-neg-pre-unit rpred)

lemma NSRD-intro:
assumes $P$ is SRD $(\neg_r$ pre$_R(P)) :: \text{true} = (\neg_r$ pre$_R(P)) \odot$ peri$_R(P)$ 
shows $P$ is NSRD 
proof –
  have NSRD$(P) = \text{R}_s((\neg_r$ pre$_R(P)) :: R1 true) \vdash ((\exists st' \odot$ peri$_R(P)) \odot$ post$_R(P)) 
    by (simp add: NSRD-form)
  also have $\ldots = \text{R}_s$(pre$_R P \vdash$ peri$_R P \odot$ post$_R P)$ 
    by (simp add: assms ex-unrest rpred closure)
  also have $\ldots = P$ 
    by (simp add: SRD-reactive-tri-design assms (1))
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 pre$_R(P)$ is RC $\odot$ peri$_R(P)$ 
shows $P$ is NSRD 
by (metis Healthy-def NSRD-form SRD-reactive-tri-design assms (1) assms (2) assms (3) ex-unrest rea-not-false wp-rea-RC-false wp-rea-def)

lemma NSRD-rdes-intro [closure]:
assumes $P$ is RC $Q$ is RR $R$ is RR $\odot$ peri$_R(P)$ 
shows $R_s$(P $\vdash$ Q $\odot$ R) is NSRD 
by (rule NSRD-RC-intro, simp-all add: rdes closure assms unrest)

lemma SRD-RD3-implies-NSRD:
[$P$ is SRD; $P$ is RD3] $\implies$ $P$ 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 $\iff$ (($P$ is SRD) $\land$ (\neg_r$ pre$_R(P)) :: R1(\text{true}) = (\neg_r$ pre$_R(P)) \land (\exists st' \odot$ 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
\[ P \subseteq Q \text{ is NSRD} \quad Q \text{ is NSRD} \]

\[
\begin{align*}
\text{\textit{pre}}_R(P) &\Rightarrow \text{\textit{pre}}_R(Q); \\
\text{\textit{peri}}_R(P) &\land \text{\textit{peri}}_R(Q) \Rightarrow \text{\textit{peri}}_R(P); \\
\text{\textit{pre}}_R(P) &\land \text{\textit{post}}_R(Q) \Rightarrow \text{\textit{post}}_R(P)
\end{align*}
\]

\[ \Rightarrow R \]

shows R

proof –

have \(\text{\textit{peri}}_R(P) \land \text{\textit{peri}}_R(P) \land \text{\textit{post}}_R(P)) \subseteq \text{\textit{peri}}_R(Q) \land \text{\textit{peri}}_R(Q) \land \text{\textit{post}}_R(Q))

by \((\text{simp add: NSRD-is-SRD SRD-reactive-tri-design assms(1) assms(2) assms(3)})\)

\text{\textit{pre}}_R(P) \Rightarrow \text{\textit{pre}}_R(Q) \land \text{\textit{peri}}_R(P) \land \text{\textit{peri}}_R(Q) \Rightarrow \text{\textit{peri}}_R(P) \land \text{\textit{peri}}_R(Q) \land \text{\textit{post}}_R(Q) \Rightarrow \text{\textit{post}}_R(P)

by \((\text{simp-all add: RHS-tri-design-refine assms closure})\)

with assms(4) show ?thesis
by simp

qed

\textbf{lemma NSRD-right-unit:} \(P\) is NSRD \(\Rightarrow P; R = P\)

by \((\text{metis Healthy-if NSRD-is-RD3 RD3-def})\)

\textbf{lemma NSRD-composition-wp:}

\textbf{assumes} \(P\) is NSRD \(Q\) is SRD

\textbf{shows} \(P; Q = R\)

by \((\text{simp add: NSRD-is-SRD WP-rea-def NSRD-neg-pre-unit NSRD-st\'-unrest-peri R1-negate-R1 R1-rea-unrest R1-preR cs-unrest rpred})\)

\textbf{lemma preR-NSRD-seq-lemma:}

\textbf{assumes} \(P\) is NSRD \(Q\) is SRD

\textbf{shows} \(R1 (\text{\textit{R2c}} (\text{\textit{post}}_R P ; ; (\neg\text{\textit{pre}}_R Q))) = \text{\textit{post}}_R P ; ; (\neg\text{\textit{pre}}_R Q)\)

\textbf{proof –}

have \(\text{\textit{post}}_R P ; ; (\neg\text{\textit{pre}}_R Q) = R1 (\text{\textit{R2c}} (\text{\textit{post}}_R P)) ; ; R1 (\text{\textit{R2c}} (\neg\text{\textit{pre}}_R Q))\)

by \((\text{simp add: NSRD-is-SRD R1-R2c-post-RHS R1-rea-not R2c-preR R2c-rea-not assms(1) assms(2)})\)

also have \(\ldots = R1 (\text{\textit{R2c}} (\text{\textit{post}}_R P ; ; (\neg\text{\textit{pre}}_R Q)))\)

by \((\text{simp add: R1-seqR R2c-R1-seq calculation})\)

finally show ?thesis ..

qed

\textbf{lemma preR-NSRD-seq \[\text{\textit{rdes}};\]:}

\textbf{assumes} \(P\) is NSRD \(Q\) is SRD

\textbf{shows} \(\text{\textit{pre}}_R(P ; ; Q) = (\text{\textit{pre}}_R P \land \text{\textit{post}}_R P \land \text{\textit{pre}}_R Q)\)

by \((\text{simp add: NSRD-composition-wp assms rea-pre-RHS-design usubst unrest wp-rea-def R2c-disj R1-disj R2c-and R2c-preR R1-R2c-commute[THEN sym] R1-extend-conj' R1-idem R2c-not closure})\)

\textbf{metis (no-types, lifting) Healthy-def Healthy-if NSRD-is-SRD R1-R2c-commute R1-R2c-seqR-distribute R1-seqR-closure assms(1) assms(2) post-R-R2c-closed post-SRD-R1 preR-R2c-closed rea-not-R1 rea-not-R2c)\)

\textbf{lemma periR-NSRD-seq \[\text{\textit{rdes}};\]:}

\textbf{assumes} \(P\) is NSRD \(Q\) is NSRD

\textbf{shows} \(\text{\textit{peri}}_R(P ; ; Q) = ((\text{\textit{peri}}_R P \land \text{\textit{post}}_R P \land \text{\textit{pre}}_R Q) \Rightarrow (\text{\textit{peri}}_R P \lor (\text{\textit{post}}_R P ; ; \text{\textit{peri}}_R Q)))\)

by \((\text{simp add: NSRD-composition-wp assms closure rea-peri-RHS-design usubst unrest wp-rea-def R1-extend-conj' R1-disj R1-R2c-seqR-distribute R2c-disj R2c-and R2c-rea-impl R1-rea-impl' R2c-preR R2c-periR R1-rea-not' R2c-rea-not R1-peri-SRD})\)

\textbf{lemma postR-NSRD-seq \[\text{\textit{rdes}};\]:}

\textbf{assumes} \(P\) is NSRD \(Q\) is NSRD

\textbf{shows} \(\text{\textit{post}}_R(P ; ; Q) = ((\text{\textit{pre}}_R P \land \text{\textit{post}}_R P \land \text{\textit{pre}}_R Q) \Rightarrow (\text{\textit{post}}_R P ; ; \text{\textit{post}}_R Q))\)

53
proof -
  have \( \neg_r \text{ post}_R \ P \ \text{ wp}_R \ \text{ pre}_R \ Q \) ;
    \text{ true}_r = (\neg_r \text{ post}_R \ P \ \text{ wp}_R \ \text{ pre}_R \ Q)
  by (simp add: \text{ wp-rea-def \ rpred \ assms \ closure \ seqr-assoc} \text{ NSRD-neg-pre-unit})
moreover have \$st' \ 
\text{ pre}_R \ P \ \text{ \wedge \ post}_R \ P \ \text{ \wp}_R \ \text{ \pre}_R \ Q \ \Rightarrow_r \ \text{ peri}_R \ P \ \vee \ \text{ post}_R \ P \ ;
  \text{ peri}_R \ Q)
  by (simp add: \text{ unrest \ assms \ wp-rea-def})
ultimately show \$\text{thesis}
  by (rule-tac \text{ NSRD-intro, simp-all add: seqr-or-distl \text{ NSRD-neg-pre-unit \ assms \ closure \ rdes \ unrest})
qed

lemma \text{ NSRD-seq-closure [\text{ closure}]}
  \text{ assumes} \ P \ \text{ is \ NSRD} \ \text{ Q \ is \ NSRD}
  \text{ shows} (P ;; Q) \ \text{ is \ NSRD}
proof -
  have \s
\text{ shows} \ R
\text{ finally show} \ ?\text{thesis}
  by
qed

lemma \text{ NSRD-tri-normal-design-composition [\text{ rdes-def}]}
  \text{ assumes} \ P \ \text{ is \ RC} \ \text{ Q}_1 \ \text{ is} \ \text{ RR} \ \$st' \ 
\text{ Q}_2 \ \text{ is} \ \text{ RR} \ \text{ S}_1 \ \text{ is} \ \text{ RR} \ \text{ S}_2 \ \text{ is} \ \text{ RR}
  \text{ shows} \ \text{ R}_n((P \ \text{ \& \ Q}_2 \ \text{ wp}_R, R) \ \text{ \vdash} \ (Q_1 \ \text{ \vee} \ (Q_2 ;; S_1)) \ \text{ \&} \ (Q_2 ;; S_2))
proof -
  have \text{ R}_n((P \ \text{ \vdash} \ (Q_1 \ \text{ \&} \ Q_2)) \ ;; \text{ R}_n((R \ \text{ \vdash} \ S_1 ;; S_2)) =
    \text{ R}_n((P \ \text{ \&} \ Q_2 \ \text{ wp}_R, R) \ \text{ \vdash} \ (Q_1 \ \text{ \&} \ (Q_2 ;; S_1)) \ \text{ \&} \ (Q_2 ;; S_2))
  by (simp-all add: \text{ NSRD-tri-normal-design-composition-wp \ rea-not-def \ assms \ unrest})
also have \text{ \_} \text{ =} \text{ R}_n((P \ \text{ \&} \ Q_2 \ \text{ wp}_R, R) \ \text{ \vdash} \ (Q_1 \ \text{ \&} \ (Q_2 ;; S_1)) \ \text{ \&} \ (Q_2 ;; S_2))
  by (simp add: \text{ assms \ wp-rea-def \ ex-unrest, rel-auto})
finally show \$\text{thesis} .
qed

If a normal reactive design has postcondition false, then it is a left zero for sequential composition.

lemma \text{ NSRD-seq-post-false}
  \text{ assumes} \ P \ \text{ is \ NSRD} \ Q \ \text{ is \ SRD} \ \text{ post}_R(P) \ = \ \text{ false}
  \text{ shows} P ;; Q = P
  apply (simp add: \text{ NSRD-composition-up \ assms \ wp \ rpred \ closure})
  using \text{ NSRD-is-SRD \ SRD-reactive-tri-design \ assms(1,3) apply \ fastforce}
done
lemma NSRD-srd-skip [closure]: $I_R$ is NSRD
by (rule NSRD-intro, simp-all add: rdes closure unrest)

lemma NSRD-Chaos [closure]: Chaos is NSRD
by (rule NSRD-intro, simp-all add: closure rdes unrest)

lemma NSRD-Miracle [closure]: Miracle is NSRD
by (rule NSRD-intro, simp-all add: closure rdes unrest)

Post-composing a miracle filters out the non-terminating behaviours

lemma NSRD-right-Miracle-tri-lemma:
assumes $P$ is NSRD
shows $P ;\!!;\text{Miracle} = R_s (\text{pre} R P \vdash \text{peri} R P \circ \neg \neg)$
by (simp add: NSRD-composition-wp closure assms rdes wp rpred)

The set of non-terminating behaviours is a subset

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

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

lemma NSRD-power-Suc [closure]:
$P$ is NSRD $\Longrightarrow P ^ n$ is NSRD
by (metis upower-Suc-NSRD-closed upred-semiring.power-Suc)

lemma uplus-NSRD-closed [closure]: $P$ is NSRD $\Longrightarrow P ^ \text{+}$ is NSRD
by (simp add: uplus-power-def closure)

lemma preR-power:
assumes $P$ is NSRD
shows $\text{pre}_R(P ;\!!; P ^ n) = (\bigsqcup_{i \in \{0..n\}} (\text{post}_R(P) ^ \text{+} i) \circ \text{wp}_r (\text{pre}_R(P)))$
proof (induct n)
case 0
then show $?case$
  by (simp add: wp closure)
next
case (Suc n) note hyp = this
have \( \text{pre}_R \ (P \ * \ (\text{Suc} \ n \ + \ 1)) = \text{pre}_R \ (P \ * \ (n+1)) \)
by (simp add: upred-semiring.power-Suc)
also have \( \ldots = (\text{pre}_R \ P \ \land \ \text{post}_R \ P \ w_p \ \text{pre}_R \ (P \ * \ (\text{Suc} \ n))) \)
using NSRD-iff assms preR-NSRD-seq power-Suc-NSRD-closed by fastforce
also have \( \ldots = (\text{pre}_R \ P \ \land \ \text{post}_R \ P \ w_p \ \bigcup_{i \in \{0..n\}} \ \text{post}_R \ P \ * \ i \ w_p \ \text{pre}_R \ P) \)
by (simp add: hyp upred-semiring.power-Suc)
also have \( \ldots = (\text{pre}_R \ P \ \land \ \bigcup_{i \in \{0..n\}} \ \text{post}_R \ P \ w_p \ \text{(post}_R \ P \ * \ i \ w_p \ \text{pre}_R \ P)) \)
by (simp add: wp)
also have \( \ldots = (\text{pre}_R \ P \ \land \ \bigcup_{i \in \{0..n\}} \ \text{post}_R \ P \ w_p \ (\text{post}_R \ P \ * \ i \ w_p \ \text{pre}_R \ P)) \)
proof \( \ldots \)
also have \( \ldots \)
by (induct_tac i, simp-all add: closure Healthy-if assms)
thus \( \exists \text{thesis} \)
by (simp add: wp-reProp def upred-semiring.power-Suc seqr-associ power-Suc rdes assms)
qed
also have \( \ldots = (\text{post}_R \ P \ * \ 0 \ w_p \ \text{pre}_R \ P \ \land \ \bigcup_{i \in \{0..n\}} \ \text{post}_R \ P \ * \ (i+1) \ w_p \ \text{pre}_R \ P) \)
by (simp add: wp assms closure)
also have \( \ldots = (\text{post}_R \ P \ * \ 0 \ w_p \ \text{pre}_R \ P \ \land \ \bigcup_{i \in \{1..Suc \ n\}} \ \text{post}_R \ P \ * \ i \ w_p \ \text{pre}_R \ P) \)
proof \( \ldots \)
have \( \bigcup_{i \in \{0..n\}} \ \text{post}_R \ P \ * \ (i+1) \ w_p \ \text{pre}_R \ P) = \bigcup_{i \in \{1..Suc \ n\}} \ \text{post}_R \ P \ * \ i \ w_p \ \text{pre}_R \ P \)
by (rule cong [of Inf], simp-all add: fun-eq-iff)
(metis (no-types, lifting) image-Suc-atLeastAtMost image-cong image-image)
thus \( \exists \text{thesis} \)
by simp
qed
also have \( \ldots = (\bigcup_{i \in \{0..Suc \ n\}} \ \text{post}_R \ P \ * \ i \ w_p \ \text{pre}_R \ P) \)
by (simp add: conj-upred-def)
also have \( \ldots = (\bigcup_{i \in \{0..Suc \ n\}} \ \text{post}_R \ P \ * \ i \ w_p \ \text{pre}_R \ P) \)
by (simp add: atLeast0-atMost-Suc-eq-insert-0)
finally show \( \exists \text{case} \)
by (simp add: upred-semiring.power-Suc)
qed

lemma preR-power! [rdes]:
assumes \( P \) is NSRD
shows \( \text{pre}_R (P \ * \ n) = \bigcup_{i \in \{0..n\}} \ \text{post}_R (P) \ * \ i \ w_p \ \text{pre}_R (P)) \)
by (simp add: preR-power assms USUP-as-Inf THEN sym)

lemma preR-power-Suc [rdes]:
assumes \( P \) is NSRD
shows \( \text{pre}_R (P \ * \ (\text{Suc} \ n)) = \bigcup_{i \in \{0..n\}} \ \text{post}_R (P) \ * \ i \ w_p \ \text{pre}_R (P) \)
by (simp add: upred-semiring.power-Suc rdes assms)

declare upred-semiring.power-Suc [simp]

lemma periR-power:
assumes \( P \) is NSRD
shows \( \text{peri}_R (P \ * \ n) = \text{peri}_R (P \ * \ (\text{Suc} \ n)) \Rightarrow_r (\bigcap_{i \in \{0..n\}} \ \text{post}_R (P) \ * \ i \ : : \ \text{peri}_R (P)) \)
proof (induct n)
case 0
then show \( \exists \text{case} \)
by (simp add: NSRD-is-SRD NSRD-wait'-unrest-pre SRD-peri-under-pre assms)
next
case \( \text{Suc} \ n \)
note hyp = this
have \( \text{peri}_R (P \ * \ (\text{Suc} \ n \ + \ 1)) = \text{peri}_R (P \ * \ (n+1)) \)
by (simp)
also have \( \ldots = \text{peri}_R (P \ * \ (\text{Suc} \ n \ + \ 1)) \Rightarrow_r (\text{peri}_R P \ \lor \ \text{post}_R P \ ; ; \ \text{peri}_R (P \ ; ; \ P \ * \ n)) \)

by (simp add: closure assms rdes)
also have ... = (pre_R(P ∩ (Suc n + 1)) ⇒ₚ (peri_R P ∨ post_R P ;; (pre_R (P ∩ (Suc n)) ⇒ₚ (∏ i \in \{0..n\}. post_R P ∩ i ;; peri_R P)))
  by (simp only: hyp)
also have ... = (pre_R P ⇒ₚ peri_R P ∨ (post_R P wp_r pre_R (P ∩ n) ⇒ₚ post_R P ;; (pre_R (P ;; P ∩ n) ⇒ₚ (∏ i \in \{0..n\}. post_R P ∩ i ;; peri_R P)))
  by (simp add: rdes closure assms, rel-blast)
also have ... = (pre_R P ⇒ₚ peri_R P ∨ (post_R P wp_r pre_R (P ;; P ∩ n) ⇒ₚ post_R P ;; (∏ i \in \{0..n\}. post_R P ∩ i ;; peri_R P)))
proof –
  have (∏ i \in \{0..n\}. post_R P ∩ i) is R1
  by (simp add: NSRD-is-SRD R1-Continuous R1-power Sup-Continuous-closed assms post-R-SRD-R1)
  hence 1:( (∏ i \in \{0..n\}. post_R P ∩ i) ;; peri_R P) is R1
  by (simp add: closure assms)
  hence (pre_R (P ;; P ∩ n) ⇒ₚ (∏ i \in \{0..n\}. post_R P ∩ i) ;; peri_R P) is R1
  by (simp add: closure)
  hence (post_R P wp_r pre_R (P ;; P ∩ n) ⇒ₚ post_R P ;; (pre_R (P ;; P ∩ n) ⇒ₚ (∏ i \in \{0..n\}. post_R P ∩ i) ;; peri_R P))
= (post_R P wp_r pre_R (P ;; P ∩ n) ⇒ₚ R1(post_R P) ;; R1(pre_R (P ;; P ∩ n) ⇒ₚ (∏ i \in \{0..n\}. post_R P ∩ i) ;; peri_R P))
  by (simp add: Healthy-if R1-post-SRD assms closure)
thus ?thesis
  by (simp only: wp-rea-impl-lemma, simp add: Healthy-if 1, simp add: R1-post-SRD assms closure)
qed
also
have ... = (pre_R P ∧ post_R P wp_r pre_R (P ;; P ∩ n) ⇒ₚ peri_R P ∨ post_R P ;; (∏ i \in \{0..n\}. post_R P ∩ i) ;; peri_R P))
  by (pred-auto)
also
have ... = (pre_R P ∧ post_R P wp_r pre_R (P ;; P ∩ n) ⇒ₚ peri_R P ∨ (∏ i \in \{0..n\}. post_R P ∩ (Suc i)) ;; peri_R P))
  by (simp add: seq-Sup-distl seqr-assoc[THEN sym])
also
have ... = (pre_R P ∧ post_R P wp_r pre_R (P ;; P ∩ n) ⇒ₚ peri_R P ∨ (∏ i \in \{1..Suc n\}. post_R P ∩ i) ;; peri_R P))
proof –
  have (∏ i \in \{0..n\}. post_R P ∩ Suc i) = (∏ i \in \{1..Suc n\}. post_R P ∩ i)
    apply (rule cong[of Sup], auto)
    apply (metis atLeast0AtMost atMost-iff image-Suc-atleastAtMost rev-image-eqI upred-semiring.power-Suc)
    using Suc-le-D apply fastforce
  done
thus ?thesis by simp
qed
also
have ... = (pre_R P ∧ post_R P wp_r pre_R (P ;; P ∩ n) ⇒ₚ (∏ i \in \{0..Suc n\}. post_R P ∩ i) ;; peri_R P)
  by (simp add: SUP-atLeastAtMost-first winf-or seqr-or-distrl seqr-or-distr)
also
have ... = (pre_R(P ∩ (Suc (Suc n))) ⇒ₚ (∏ i \in \{0..Suc n\}. post_R P ∩ i) ;; peri_R P))
  by (simp add: rdes closure assms)
finally show ?case by (simp)
qed

lemma periR-power′ [rdes]:

assumes \( P \) is NSRD
shows \( \text{peri}_R(P \sqcup P^\ast n) = (\text{pre}_R(P^\ast(Suc \ n)) \Rightarrow \bigwedge \{ i \in \{0..n\} \cdot \text{post}_R(P) \land i \} ; ; \text{peri}_R(P) \) 
by (simp add: periR-power assms UINF-as-Sup[THEN sym])

lemma periR-power-Suc [rdes]:
assumes \( P \) is NSRD
shows \( \text{peri}_R(P^\ast(Suc \ n)) = (\text{pre}_R(P^\ast(Suc \ n)) \Rightarrow \bigwedge \{ i \in \{0..n\} \cdot \text{post}_R(P) \land i \} ; ; \text{peri}_R(P) \) 
by (simp add: rdes assms)

lemma postR-power [rdes]:
assumes \( P \) is NSRD
shows \( \text{post}_R(P \sqcup P^\ast n) = (\text{pre}_R(P^\ast(Suc \ n)) \Rightarrow \text{post}_R(P) \land Suc \ n) \)
proof (induct \( n \))
  case 0 
  then show \( \text{case} \) 
  by (simp add: NSRD-is-SRD NSRD-wait'-unrest-pre SRD-post-under-pre assms)
next
  case (Suc \( n \)) note \( hyp = this \)
  have \( \text{post}_R(P \land (Suc \ n + 1)) = \text{post}_R(P \sqcup (P \land (n+1)) \)
  by (simp)
  also have \( \ldots = (\text{pre}_R(P \land (Suc \ n + 1)) \Rightarrow (\text{post}_R P \sqcup \text{post}_R (P \sqcup P^\ast n))) \)
  by (simp add: closure assms rdes)
  also have \( \ldots = (\text{pre}_R(P \land (Suc \ n + 1)) \Rightarrow (\text{post}_R P \sqcup (\text{pre}_R (P^\ast Suc \ n)) \Rightarrow \text{post}_R P \land Suc \ n))) \)
  by (simp only: \( hyp \))
  also have \( \ldots = (\text{pre}_R P \Rightarrow (\text{post}_R P \sqcup \text{pre}_R (P \land Suc \ n)) \Rightarrow \text{post}_R P \land Suc \ (Suc \ n)) \)
  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 \( \ldots = (\text{pre}_R P \land post_R P \sqcup P \land (P \land Suc \ n)) \Rightarrow \text{post}_R P \land Suc \ (Suc \ n)) \)
  by (pred-auto)
  also have \( \ldots = (\text{pre}_R (P^\ast (Suc \ Suc \ n))) \Rightarrow \text{post}_R P \land Suc \ (Suc \ n)) \)
  by (simp add: rdes closure assms)
finally show \( \text{case} \) by (simp)
qed

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

lemma power-rdes-def [rdes-def]:
assumes \( P \) is RC \( Q \) is RR \( R \) is RR \$st' \& Q
shows \( \textbf{R}_1((\bigwedge \{ i \in \{0..n\} \cdot (R \land i) \sqcup_p P \}) \Rightarrow (\bigwedge \{ i \in \{0..n\} \cdot (R \land i) \} ; ; Q) \Rightarrow (R \land Suc \ n)) \)
proof (induct \( n \))
  case 0 
  then show \( \text{case} \) 
  by (simp add: wp assms closure)
next
  case (Suc \( n \))
have 1: \((P \land (\bigsqcap i \in \{0..n\} \cdot R \ wp_r (R \ ^i \ wp_r P))) = (\bigsqcup i \in \{0..Suc n\} \cdot R \ ^i \ wp_r P)\)
(is \(\text{lhs} = \text{rhs}\))

proof –

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

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

proof –

have \(\text{lhs} = (Q \lor (\bigsqcap i \in \{0..n\} \cdot R \ ^i ;; Q))\)
  by (simp add: seqr-assoc THEN simp seq-UINF-distl)
also have \(\ldots = (Q \lor (\bigsqcap i \in \{0..n\}. R \ ^i ;; Q))\)
  by (simp only: UINF-as-Sup-collect)
also have \(\ldots = (Q \lor (\bigsqcap i \in \{0..Suc n\}. R \ ^i ;; Q))\)
  by (simp add: seq-UINF-distl)
also have \(\ldots = (\bigsqcap i \in \{0..Suc n\}. R \ ^i ;; Q)\)
  by (simp add: atLeastAtMost-insertL)
finally have \(?thesis\)
proof
  by (simp add: UINF-as-Sup-collect)
qed

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

proof –

have \(\bigsqcap i \in \{0..n\} \cdot R \ ^i ;; Q = \bigsqcap i \in \{0..n\}. R \ ^i ;; Q\)
  by (simp add: UINF-as-Sup-collect)
also have \(\ldots = (\bigsqcap i \in \{0..Suc n\}. R \ ^i ;; Q)\)
  by (simp add: atLeastAtMost-insertL)
also have \(\ldots = (Q \lor (\bigsqcap i \in \{0..Suc n\}. R \ ^i ;; Q))\)
  by (metis (no-types, lifting) SUP-insert seqr-left-unit seqr-or-distl power-0)
also have \(\ldots = (Q \lor (\bigsqcap i \in \{0..<n\}. R \ ^ Suc i ;; Q))\)
  by (simp add: UINF-as-Sup-collect)
also have \(\ldots is RR\)
  by (simp-all add: closure assms)
finally have \(?thesis\).

from 1 2 3 Suc show \$case
proof
  by (simp add: Suc RHS-tri-normal-design-composition' closure assms wp)
qed
5.1 UTP theory

Here, we show that normal stateful reactive designs form a Kleene UTP theory, and thus a
Kleene algebra [4, 1]. This provides the basis for reasoning about iterative reactive contracts.

interpretation nsr-thy: utp-theory-kleene UTHY(NSRDES, (s,t::trace,α) rsp)
rewrites ∃ P. P ∈ carrier (uthy-order NSRDES) ←→ P is NSRD
and P is H_{NSRDES} ←→ P is NSRD
and (μ X · F (H_{NSRDES} X)) = (μ X · F (NSRD X))
and carrier (uthy-order NSRDES) → carrier (uthy-order NSRDES) ≡ [NSRD]_H → [NSRD]_H
and H_{NSRDES}_H = Miracle
and T_{NSRDES} = Miracle
and I_{NSRDES} = H_R
and le (uthy-order NSRDES) = op ⊆
proof –
interpret lat: utp-theory-continuous UTHY(NSRDES, (s,t,α) rsp)
by (unfold-locales, simp-all add: nsrdes-hcond-def nsrdes-unit-def closure Healthy-if)
show 1: T_{NSRDES} = (Miracle :: (s,t,α) hrel-rsp)
by (metis NSRD-Miracle NSRD-is-SRD lat.top-healthy lat.utp-theory-continuous-axioms nsrdes-hcond-def
srdes-theory-continuous.meet-top utpred-semiring.add-commute utp-theory-continuous.meet-top)
thus utp-theory-kleene UTHY(NSRDES, (s,t,α) rsp)
by (unfold-locales, simp-all add: nsrdes-hcond-def nsrdes-unit-def closure Healthy-if Miracle-left-zero
SRD-left-unit NSRD-right-unit)
qed (simp-all add: nsrdes-hcond-def nsrdes-unit-def closure Healthy-if)

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

abbreviation TestR (test R) where
  test R P ≡ u test NSRDES P

abbreviation StarR :: (s, t:t:trace, α) hrel-rsp ⇒ (s, t, α) hrel-rsp (s∗R [999] 999) where
  StarR P ≡ P∗NSRDES

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

lemma StarR-rdes-def [rdes-def]:
  assumes P is RC Q is RR R is RR $s t z Q
  shows (R_s(P ⇒ Q)∗R_s = R_s((R∗R wp, P) ⇒ R∗R ))
  by (simp add: rrel-thy.Star-alt-def nsrd-thy.Star-alt-def assms closure rdes-def unrest rpred disj-upred-def)
end

6 Syntax for reactive design contracts

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

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

definition mk-RD :: (t::trace ⇒ 0 apred) ⇒ (t::trace ⇒ 0 apred) ⇒ (t ⇒ 0 hrel) ⇒ (s, t, α) hrel-rsp where
mk-RD P Q R = R_s((P) | s < ο [Q(x)]S_ < [x → &tt] ∩ [R(x)]S_ [x → &tt])

definition trace-pred :: (t::trace ⇒ 0 apred) ⇒ (s, t, α) hrel-rsp where
trace-pred P = [(P x)]S_ [x → &tt]

syntax
  -trace-var :: logic
  -mk-RD :: logic ⇒ logic ⇒ logic ⇒ logic ([-/ |-/ | -]R)
  -trace-var :: logic ⇒ logic ([/- /-]R)

parse-translation !!
  let
    fun trace-var-tr [] = Syntax.free trace
    | trace-var-tr - = raise Match;
    in
      (⌜@[syntax-const -trace-var, K trace-var-tr] ⌜
end

translations
  [P ⊢ Q]R <= CONST mk-RD P (λ x. Q) (λ y. R)
  [P]t ⇒ CONST trace-pred (λ -trace-var. P)
  [P]t <= CONST trace-pred (λ t. P)

lemma SRD-mk-RD [closure]: [P ⊢ Q(trace) | R(trace)]R is SRD
by (simp add: mk-RD-def closure unrest)

lemma preR-mk-RD [rdes]: pre_R([P ⊢ Q(trace) | R(trace)]_R) = R1([P]_S<)
by (simp add: mk-RD-def rea-pre-RHS-design unrest R2c-not R2c-lift-state-pre)

lemma trace-pred-RR-closed [closure]:
[P trace], is RR
by (rel-auto)

lemma unrest-trace-pred-st′ [unrest]:
$st' \sharp [P trace]$
by (rel-auto)

lemma R2c-msubst-tt: R2c (msubst (λx. [Q x]_S) &tt) = (msubst (λx. [Q x]_S) &tt)
by (rel-auto)

lemma periR-mk-RD [rdes]: peri_R([P ⊢ Q(trace) | R(trace)]_R) = R1(([Q(trace)]_S<)[trace→&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 ⊢ Q(trace) | R(trace)]_R) = ([P]_S< ⇒ R1(([R(trace)]_S)[trace→&tt]))
by (simp add: mk-RD-def rea-post-RHS-design unrest R2c-not R2c-lift-state-pre
impl-alt-def R2c-disj R2c-msubst-tt R2c-rea-impl R1-rea-impl)

Refinement introduction law for contracts

lemma RD-contract-refine:
assumes
Q is SRD 'P1.S< ⇒ pre_R Q'
'P1.S< ∧ peri_R Q ⇒ [P2 x]_S<[x→&tt]'
'P1.S< ∧ post_R Q ⇒ [P3 x]_S[x→&tt]'
shows
[P1 ⊢ P2(trace) | P3(trace)]_R ⊑ Q
proof −
  have [P1 ⊢ P2(trace) | P3(trace)]_R ⊑ R_s(pre_R(Q) ⇒ peri_R(Q) ⊨ post_R(Q))
  using assms
  by (simp add: mk-RD-def, rule-tac srdes-tri-refine-intro, simp-all)
  thus ?thesis
  by (simp add: SRD-reactive-tri-design assms(1))
qed

end

7 Reactive design tactics

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

Theorems for normalisation

lemmas rdes-rel-norms =
  prod.case-eq-if
  conj-assoc
  disj-assoc
  conj-disj-distr
  conj-UINF-dist
  conj-UINF-ind-dist

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: closure)?, (simp add: rdes-def rdes-rel-norms rdes rpred cls closure alpha usubst unrest wp simps cong: cong))
```

Tactic to split a refinement conjecture into three POs

**method** `rdes-refine-split` uses `cls cong simps` =

```
((rdes-simp cls: cls cong: cong simps: simps; rule-tac srdes-tri-refine-intro)
```

Tactic to split an equality conjecture into three POs

**method** `rdes-eq-split` uses `cls cong simps` =

```
((rdes-simp cls: cls cong: cong simps: simps; (rule-tac srdes-tri-eq-intro))
```

Tactic to prove a refinement

**method** `rdes-refine` uses `cls cong simps` =

```
((rdes-refine-split cls: cls cong: cong simps: simps; (insert cls; rel-auto))
```

Tactics to prove an equality

**method** `rdes-eq` uses `cls cong simps` =

```
((rdes-eq-split cls: cls cong: cong simps: simps; rel-auto)
```

Via antisymmetry

**method** `rdes-eq-anti` uses `cls cong simps` =

```
((rdes-simp cls: cls cong: cong simps: simps; (rule-tac antisym; (rule-tac srdes-tri-refine-intro; rel-auto)))
```

Tactic to calculate pre/peri/postconditions from reactive designs

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

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

**method** `rdspl-refine` =

```
(rule-tac SRD-refine-intro; (simp add: closure rdes unrest usubst ; rel-blast?))
```

The following tactic combines antisymmetry with the previous tactic to prove an equality.

**method** `rdspl-eq` =

```
(rule-tac antisym, rdes-refine, rdes-refine)
```
8 Reactive design parallel-by-merge

theory utp-rdes-parallel
imports
  utp-rdes-normal
  utp-rdes-tactics
begin

R3h implicitly depends on RD1, and therefore it requires that both sides be RD1. We also
require that both sides are R3c, and that wait\textsubscript{m} is a quasi-unit, and div\textsubscript{m} yields divergence.

lemma st-U0-alpha: \([\exists \; st \cdot II]_0 = (\exists \; st \cdot [II]_0)\)
  by (rel-auto)

lemma st-U1-alpha: \([\exists \; st \cdot II]_1 = (\exists \; st \cdot [II]_1)\)
  by (rel-auto)

definition skip-rm :: \((s, t::trace, \alpha)\) s\textsuperscript{u}p\textsuperscript{r}-r\textsuperscript{m}
\parallel \quad \begin{aligned}
\text{where} & \\
\end{aligned}

\text{upred-defs}: \; H_{RM} = (\exists \; st < \cdot \text{skip\textsubscript{m}} \; \vee \; (\neg \; \text{ok}_< \land \text{tr}_< \leq u \; \text{tr}_<'))

definition [upred-defs]: \(R3hm(M) = (H_{RM} < \text{wait}_< \triangleright M)\)

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

lemma R3h-par-by-merge [closure]:
  assumes \(P \; \text{is } R3h \; Q \; \text{is } R3h \; M \; \text{is } R3hm\)
  shows \((P \; |_M \; Q) \; \text{is } R3h\)

proof -
  have \((P \; |_M \; Q) = (((P \; |_M \; Q)[true/\text{ok}] \; \triangleright \text{ok} \; (P \; |_M \; Q)[false/\text{ok}])[true/\text{wait}] \; \triangleright \text{wait} \\
    \triangleright (P \; |_M \; Q))\)
    by (simp add: cond-var-subst-left cond-var-subst-right)
  also have \(= (((P \; |_M \; Q)[true,\text{true}/\text{ok,\text{wait}}] \; \triangleright \text{ok} \\
    \triangleright (P \; |_M \; Q))\)
    by (rel-auto)
  also have \(= (((P \; |_M \; Q)[true,\text{true}/\text{ok,\text{wait}}] \; \triangleright \text{ok} \; (P \; |_M \; Q)[false,\text{true}/\text{ok,\text{wait}}]) \; \triangleright \text{wait} \\
    \triangleright (P \; |_M \; Q))\)
    by (simp add: cond-var-subst-right)
  also have \(= (((P \; |_M \; Q)[true,\text{true}/\text{ok,\text{wait}}] \; \triangleright \text{ok} \\
    \triangleright (P \; |_M \; Q))\)
    by (rel-auto)
  finally show \text{thesis} by (simp add: closure assms unrest)
qed

also have \(= (((P \; |_M \; Q)[true,\text{true}/\text{ok,\text{wait}}] \; \triangleright \text{ok} \; (R1(true))[false,\text{true}/\text{ok,\text{wait}}]) \; \triangleright \text{wait} \\
    \triangleright (P \; |_M \; Q))\)
    by (simp add: cond-var-subst-left cond-var-subst-right)
  also have \(= (((P \; |_M \; Q)[true,\text{true}/\text{ok,\text{wait}}] \; \triangleright \text{ok} \\
    \triangleright (P \; |_M \; Q))\)
    by (rel-auto)
  also have \(= (((P \; |_M \; Q)[true,\text{true}/\text{ok,\text{wait}}] \; \triangleright \text{ok} \\
    \triangleright (P \; |_M \; Q))\)
    by (simp add: closure assms unrest)
qed
also have ... = (R1(true))[[false,true/$\$ok,$\$wait]]
  by (rel-blast)
finally show ?thesis by simp
qed
also have ... = (((∃ $st \cdot II) < $ok \triangleright R1(true)) < $wait \triangleright (P \parallel_M Q))
  by (rel-auto)
also have ... = R3h(P \parallel_M Q)
  by (simp add: R3h-cases)
finally show ?thesis
  by (simp add: Healthy-def)
qed

definition [upred-defs]: RD1m(M) = (M ∨ ¬ $ok < \$tr_≤u $tr´)

lemma RD1-par-by-merge [closure]:
  assumes P is R1 Q is R1 M is R1m P is RD1 Q is RD1 M is RD1m
  shows (P \parallel_M Q) is RD1
proof –
  have 1: (RD1(R1(P)) \parallel RD1m(R1m(M)) \parallel RD1(Q))[[false/$\$ok] = R1(true)
    by (rel-blast)
  have (P \parallel_M Q) = (P \parallel_M Q)[true/$\$ok] < $ok \triangleright (P \parallel_M Q)[false/$\$ok]
    by (simp add: cond-var-split)
  also have ... = R1(P \parallel_M Q) < $ok \triangleright R1(true)
    by (metis 1 Healthy-if R1-par-by-merge assms calculation
        cond-idem cond-var-subst-right in-var-uvar ok-vwb-lens)
  also have ... = RD1(P \parallel_M Q)
    by (simp add: Healthy-if R1-par-by-merge RD1-alt-def assms(3))
finally show ?thesis
  by (simp add: Healthy-def)
qed

lemma RD2-par-by-merge [closure]:
  assumes M is RD2
  shows (P \parallel_M Q) is RD2
proof –
  have (P \parallel_M Q) = ((P \parallel_s Q) :: M)
    by (simp add: par-by-merge-def)
  also from assms have ... = ((P \parallel_s Q) :: (M :: J))
    by (simp add: Healthy-def RD2-def H2-def)
  also from assms have ... = (((P \parallel_s Q) :: M) :: J)
    by (simp add: seqr-assoc)
  also from assms have ... = RD2(P \parallel_M Q)
    by (simp add: RD2-def H2-def par-by-merge-def)
finally show ?thesis
  by (simp add: Healthy-def)
qed

lemma SRD-par-by-merge:
  assumes P is SRD Q is SRD M is R1m M is R2m M is R3hm M is RD1m M is RD2
  shows (P \parallel_M Q) is SRD
by (rule SRD-intro, simp-all add: assms closure SRD-healths)

definition nmerge-rd0 (N₀) where
[upred-defs]: N₀(M) = ($\$wait´ = u ($\$0\$-wait \lor \$\$1\$-wait) \land \$\$tr_≤u $\$tr´
  \land (∃ $\$0\$\$-\$ok;\$\$1\$-\$ok;\$\$ok´;\$\$ok`\$\$\$0\$-\$wait;\$\$1\$-\$wait;\$\$wait;\$\$\$wait´ \cdot M))
definition \textit{nmerge-rd1} \((N_1)\) where 
\[ \text{[upred-defs]}:\ N_1(M) = (\$ok^' =_u (\$0 - ok \land \$I - ok) \land N_0(M)) \]

definition \textit{nmerge-rd} \((N_R)\) where 
\[ \text{[upred-defs]}:\ N_R(M) = ((\exists \$st_< \cdot \$v^' =_u \$v_<) \triangleleft \$wait_\prec N_1(M)) \triangleleft \$ok_\prec (\$tr_\preceq \leq_u \$tr^') \]

definition \textit{merge-rd1} \((M_1)\) where 
\[ \text{[upred-defs]}:\ M_1(M) = (N_1(M) ;; II_R) \]

definition \textit{merge-rd} \((M_R)\) where 
\[ \text{[upred-defs]}:\ M_R(M) = N_R(M) ;; II_R \]

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

Healthiness condition for reactive design merge predicates 

definition \[ \text{[upred-defs]}:\ RDM(M) = R2m(\exists \$0 - ok; \$1 - ok; \$ok_\prec; \$ok^'; \$0 - wait; \$1 - wait; \$wait_\prec; \$wait_\prec' \cdot M) \]

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

lemma \textit{R2m-nmerge-rd}: 
\[ R2m(N_R(R2m(M))) = N_R(R2m(M)) \]
apply (rel-auto) using minus-zero-eq by blast+

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

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

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

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

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

lemma \textit{merge-rd-is-RD23}: 
\[ M_R(M) \text{ is RD23} \]
by (simp add: RD3-implies-RD2 merge-rd-is-RD3)

lemma \textit{par-rdes-NSRD} \([\text{[closure]}]\): 
\[ \text{assumes } P \text{ is SRD } Q \text{ is SRD } M \text{ is RDM} \]
shows \[ P \parallel_R M Q \text{ is NSRD} \]
proof –
have \[(P \parallel_R M Q ;; II_R) \text{ is NSRD} \]
by (rule NSRD-intro', simp-all add: SRD-healths closure assms)
(metis (no-types, lifting) Healthy-def R2-par-by-merge R2-seqr-closure R2m-nmerge-rd RDM-def SRD-healths(2) assms skip-srea-R2)
thus \[?thesis \]
by (simp add: merge-rd-def par-by-merge-def seqr-assoc)
qed
lemma RDM-intro:
assumes $M$ is R2m $0$ − ok $\triangleright M$ $1$ − ok $\triangleright M$ $\triangleright M$
shows $M$ is RDM
using assms
by (simp add: Healthy-def RDM-def unrest unrest)

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

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

lemma ex-st' R2m-closed [closure]:
assumes P is RR Q is RR M is R2m
shows $0$ − ok $\triangleright M$ $1$ − ok $\triangleright M$ $\triangleright M$ $\triangleright M$
by (rule RR-R2-intro, simp-all add: unrest assms RR-implies-R2-closure)

lemma parallel-ok-cases: ((P || s, Q) :: M) = {
(P' || s, Q') :: (M[true, true/$0$ − ok,$1$ − ok]) ∨
(P'' || s, Q'') :: (M[false, true/$0$ − ok,$1$ − ok]) ∨
(P''' || s, Q''') :: (M[false, false/$0$ − ok,$1$ − ok]) ∨
(P' || s, Q') :: (M[false, false/$0$ − ok,$1$ − ok]))

proof –
have ((P || s, Q) :: M) = (∃ ok0 · (P || s, Q)<ok0>/<$0$ − ok $>$ $>$); M<$0$ − ok $>$)
by (subst segm-middle[of left-uvar ok], simp-all)
also have ... = (∃ ok0 · (P || s, Q)<ok0>/<$0$ − ok $>$ $>$); M<$0$ − ok $>$)
by (subst segm-middle[of right-uvar ok], simp-all)
also have ... = (∃ ok0 · (P || s, Q)<ok0>/<$0$ − ok $>$ $>$); M<$0$ − ok $>$)
by (rel-auto robust)
also have ...
(P' || s, Q') :: (M[true, true/$0$ − ok,$1$ − ok]) ∨
(P'' || s, Q'') :: (M[false, true/$0$ − ok,$1$ − ok]) ∨
(P''' || s, Q''') :: (M[false, false/$0$ − ok,$1$ − ok]) ∨
by (simp add: true-alt-def [THEN sym] false-alt-def [THEN sym] disj-assoc
lemma skip-srva-thesis [usubst]:
\[ H_R \overset{f}{=} R1(\neg \$ok) \]
by (rel-auto)

lemma nmerge0-rd-unrest [unrest]:
\[ \$0 \text{--} \text{ok} \not\in N_0 \quad | \quad \$1 \text{--} \text{ok} \not\in N_0 \quad M \]
by (pred-auto)+

lemma parallel-assm-lemma:
assumes \( P \text{ is RD2} \)
shows \( \text{pre} \overset{\uparrow}{=} (P \parallel M(R(M)) \ Q) = ((\text{pre} \overset{\uparrow}{=} P) \parallel N_0(M) \ ;; \ R1(\text{true}) \ (\text{cmt} \overset{\uparrow}{=} Q)) \)
\[ \lor ((\text{cmt} \overset{\uparrow}{=} P) \parallel N_0(M) \ ;; \ R1(\text{true}) \ (\text{pre} \overset{\uparrow}{=} Q)) \]
proof –
\begin{align*}
& \text{have } \text{pre} \overset{\uparrow}{=} (P \parallel M(R(M)) \ Q) = \text{pre} \overset{\uparrow}{=} ((P \parallel Q) \ ;; \ M(R(M)) ) \\
& \quad \text{by (simp add: par-by-merge-def)} \\
& \text{also have } \ldots = ((P \parallel Q)[\text{true} \text{--} \text{false} / \text{ok}/\text{ok}/\text{wait}] ; N_R M ; R1(\neg \$ok)) \\
& \quad \text{by (simp add: merge-rd-def usubst, rel-auto)} \\
& \text{also have } \ldots = ((P[\text{true} \text{--} \text{false} / \text{ok}/\text{ok}/\text{wait}] \parallel Q[\text{true} \text{--} \text{false} / \text{ok}/\text{ok}/\text{wait}] ) ; N_1(M) ; R1(\neg \$ok)) \\
& \quad \text{by (rel-auto robust, (metis+) )} \\
& \text{also have } \ldots = ((P \parallel Q)[\text{true} \text{--} \text{false} / \text{ok}/\text{ok}/\text{wait}] \parallel (Q[\text{true} \text{--} \text{false} / \text{ok}/\text{ok}/\text{wait}] ) ; (N_1 M) [\text{true} \text{--} \text{false} / \text{ok}/\text{ok}/\text{wait}] ) ; R1(\neg \$ok)) \\
& \quad \text{by (rel-auto parallel-ok-cases, subst-tac)} \\
& \text{also have } \ldots = (\text{?C2} \lor \text{?C3}) \\
proof –
\end{align*}
\begin{align*}
& \text{have } \text{?C1} = \text{false} \\
& \quad \text{by (rel-auto)} \\
& \text{moreover have } ?C4 \Rightarrow ?C3 \text{ (is } (?A \ ;; \ ?B) \Rightarrow (?C \ ;; \ ?D) \text{) } \\
proof –
\end{align*}
\begin{align*}
& \text{from } \text{assms } \text{have } ?Pf \Rightarrow P^f, \\
& \quad \text{by (metis RD2-def H2-equivalence Healthy-def’)} \\
& \text{hence } P ; ?Pf \Rightarrow P^f, \\
& \quad \text{by (rel-auto)} \\
& \text{have } ?A \Rightarrow ?C, \\
& \quad \text{using } P \text{ by (rel-auto)} \\
& \text{moreover have } ?B \Rightarrow ?D, \\
& \quad \text{by (rel-auto)} \\
& \text{ultimately show } ?thesis \\
& \quad \text{by (simp add: impl-seq-mono)} \\
\end{align*}
qed

ultimately show ?thesis 
by (simp add: subst-tac)
qed
also have \ldots = (}
lemma \( \mathit{pre}_s \text{-SRD} \):
assumes \( P \) is SRD
shows \( \mathit{pre}_s \vdash P \equiv (\neg_r \mathit{pre}_R(P)) \)
proof –
  have \( \mathit{pre}_s \vdash P = \mathit{pre}_s \vdash \mathit{R}(\mathit{pre}_R P \vdash \mathit{peri}_R P \circ \mathit{post}_R P) \)
    by (simp add: \( \mathit{SRD}\text{-reactive-tri-design} \) \( \mathit{assms} \))
  also have \( \ldots = \mathit{R}(\mathit{R}2c(\neg \mathit{pre}_s \vdash \mathit{pre}_R P)) \)
    by (simp add: \( \mathit{RHS}\text{-def} \) \( \mathit{usubst} \) \( \mathit{R}3h\text{-def} \) \( \mathit{pre}_s\text{-design} \))
  also have \( \ldots = \mathit{R}(\mathit{R}2c(\neg \mathit{pre}_R P)) \)
    by (rel-auto)
  also have \( \ldots = (\neg_r \mathit{pre}_R P) \)
    by (simp add: \( \mathit{R}2c\text{-not} \) \( \mathit{R}2c\text{-preR} \) \( \mathit{assms} \) \( \mathit{rea}\text{-not-def} \))
  finally show \( ?\mathit{thesis} \).
qed

lemma \( \mathit{parallel-assm} \):
assumes \( P \) is SRD \( Q \) is SRD
shows \( \mathit{pre}_R(P \parallel M_{R(M)} Q) = (\neg_r ((\neg_r \mathit{pre}_R(P)) \parallel N_0(M) \parallel \mathit{R}1(\mathit{true}) \mathit{cmt}_R(Q)) \land
\neg_r (\mathit{cmt}_R(P) \parallel N_0(M) \parallel \mathit{R}1(\mathit{true}) (\neg_r \mathit{pre}_R(Q))))) \)\( \)\( \)
(is \( ?\mathit{lhs} = ?\mathit{rhs} \))
proof –
  have \( \mathit{pre}_R(P \parallel M_{R(M)} Q) = (\neg_r (\mathit{pre}_s \vdash P) \parallel N_0(M) \parallel \mathit{R}1(\mathit{true}) \mathit{cmt}_R(Q) \land
\neg_r (\mathit{cmt}_s \vdash P) \parallel N_0(M) \parallel \mathit{R}1(\mathit{true}) (\mathit{pre}_s \vdash Q)) \)
    by (simp add: \( \mathit{pre}_R\text{-def} \) \( \mathit{parallel-assm-lemma} \) \( \mathit{assms} \) \( \mathit{SRD}\text{-healths} \) \( \mathit{R}1\text{-conj} \) \( \mathit{rea}\text{-not-def} \)[THEN \( \mathit{sym} \)])
  also have \( \ldots = ?\mathit{rhs} \)
    by (simp add: \( \mathit{pre}_s\text{-SRD} \) \( \mathit{assms} \) \( \mathit{cmt}_R\text{-def} \) \( \mathit{Healthy-if} \) \( \mathit{closure} \) \( \mathit{unrest} \))
  finally show \( ?\mathit{thesis} \).
qed

lemma \( \mathit{parallel-assm-unrest-wait}' \) [unrest]:
\( \parallel P \) is SRD \( Q \) is SRD \( \parallel \mathit{wait} \parallel \mathit{pre}_R(P \parallel M_{R(M)} Q) \)
by (simp add: \( \mathit{parallel-assm} \), \( \mathit{simp add} = \mathit{par-by-merge-def} \) \( \mathit{unrest} \))

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

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

lemma \( \mathit{JL3} \): \( (M_1 M)^t[\mathit{false}, \mathit{false} / \mathit{\$0-ok}, \mathit{\$1-ok}] = N_0(M) \parallel R1(\mathit{true}) \)
by (rel-blast)
lemma \textit{JL4}: \((M_1 M)^t [\text{true, true}/s_0-\text{ok},s_1-\text{ok}] = (s_0 \land N_0 M) :: H_R^t\)
by (simp add: merge-rd1-def usubst nmerge-rd1-def anrest)

lemma \textit{parallel-commitment-lemma-1}:
assumes \(P\) is \(RD2\)
shows \(\text{cmt} \vdash (P \parallel M_R(M)) \parallel Q = (\parallel (P [\text{true, false}/s_0, s_1 \lor \text{wait}] \parallel (M_1 M)^t [\text{true, false}/s_0, s_1 \lor \text{wait}])\)
by (simp add: usubst rel-auto)
also have \(\ldots = ((P [\text{true, false}/s_0, s_1 \lor \text{wait}] \parallel Q [\text{true, false}/s_0, s_1 \lor \text{wait}]) :: (M_1 M)^t)\)
by (simp add: par-by-merge-def)
also have \(\ldots = (\parallel (P [\text{true, true}/s_0-\text{ok}, s_1-\text{ok}] \parallel (M_1 M)^t [\text{true, true}/s_0-\text{ok}, s_1-\text{ok}]))\)
by (simp add: parallel-ok-cases subst-tac)
also have \(\ldots = (\parallel (P [\text{true, true}/s_0-\text{ok}, s_1-\text{ok}] \parallel (M_1 M)^t [\text{true, true}/s_0-\text{ok}, s_1-\text{ok}]))\)
by (simp add: JL1 JL2 JL3)

proof\ --
from \textit{assms} have \(\vdash P \Rightarrow P^t\)\textit{'}
by (metis RD2-def H2-equivalence Healthy-def')
hence \(P; P^t \Rightarrow P^t\)\textit{'}
by (rel-auto)
have \(\vdash ?C_4 \Rightarrow ?C_2\) (is \(\vdash ?A \Rightarrow ?B \Rightarrow (?C \Rightarrow ?D)\)')
proof\ --
have \(\vdash ?A \Rightarrow ?C\)
using \(P\) by (rel-auto)
thus \(\vdash ?\text{thesis}\)
by (simp add: impl-seq-nono)
qed
thus \(\vdash ?\text{thesis}\)
by (simp add: subsumption2)
qed
finally show \(\vdash ?\text{thesis}\)
by (simp add: par-by-merge-def JL4)
qed

lemma \textit{parallel-commitment-lemma-2}:
assumes \(P\) is \(RD2\)
shows \(\text{cmt} \vdash (P \parallel M_R(M)) \parallel Q = (\parallel (P [\text{true, true}/s_0-\text{ok}, s_1-\text{ok}] \parallel (M_1 M)^t [\text{true, true}/s_0-\text{ok}, s_1-\text{ok}]))\)
by (simp add: parallel-ok-cases subst-tac)

70
by (simp add: parallel-commitment-lemma-1 assms parallel-assm-lemma)

**lemma** parallel-commitment-lemma-3:

$M$ is $R1m$ $\implies$ $(\text{ok}' \land N_0 M) ;; II_R'$ is $R1m$

by (rel-simp, safe, metis+)

**lemma** parallel-commitment:

assumes $P$ is SRD $Q$ is SRD $M$ is RDM

shows $\text{cmt}_R(P \parallel M_{R(M)}(Q)) = (\text{pre}_{R}(P \parallel M_{R(M)}(Q)) \implies \text{cmt}_R(Q))$

by (simp add: parallel-commitment-lemma-2 parallel-commitment-lemma-3 Healthy-if assms cmt

**proof** parallel-reactive-design:

assumes $P$ is SRD $Q$ is SRD $M$ is RDM

shows $(P \parallel M_{R(M)}(Q)) = R_s(\text{pre}_{R}(P \parallel M_{R(M)}(Q)) \implies \text{cmt}_R(Q)))$

by (simp add: parallel-commitment-lemma-1 assms parallel-assm-lemma)

also have ... $\equiv$ $\text{cmt}_t$

by (rel-auto)

finally show $\text{thesis}$.

qed

**lemma** parallel-pericondition-lemma1:

$(\text{ok}' \land P) ;; II_R[\text{true},true/$ok$' , \text{wait}'] = (\exists st' \cdot P)[true,true/$ok$',\text{wait}']$

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

**proof** parallel-pericondition-lemma2:

assumes $M$ is RDM

shows $(\exists st' \cdot N_0(M))[true,true/$ok$' , \text{wait}'] = ((\text{ok}' \land N_0 M) \land (\exists st' \cdot M))$

**proof** parallel-pericondition-lemma3:

$(\exists st' \cdot M) = ((\text{ok}' \land N_0 M) \land (\exists st' \cdot M))$

by (simp add: unsubst unrest nmerge-rd0-def ex-unrest Healthy-if $R1m$-def assms)

also have ... $\equiv$ $\exists st' \cdot (\text{ok}' \land M) = (\exists st' \cdot (\text{ok}' \land M) \land (\exists st' \cdot M))$

also have ... $\equiv$ $((\text{ok}' \land N_0 M) \land (\exists st' \cdot M))$

by (rel-auto)

finally show $\text{thesis}$.

qed

**lemma** parallel-pericondition-lemma3:

$((\text{ok}' \land N_0 M) \land (\exists st' \cdot M)) = ((\text{ok}' \land N_0 M) \land (\exists st' \cdot M)) \land (\exists st' \cdot M)$

by (rel-auto)

finally show $\text{thesis}$.

qed
by (rel-auto)

lemma parallel-pericondition [rdes]:

fixes P :: ('s, 'a::trace, 'a) rsp merge
assumes P :: SRD Q is SRD M is RDM
shows peri_r(P ⌦_M M Q) = (peri_r(P ⌦_M M Q) ⇒ peri_r(P) ⌦_M peri_r(Q)
∨ post_r(P) ⌦_M peri_r(Q)
∨ peri_r(P) ⌦_M post_r(Q))

proof
  have peri_r(P ⌦_M M Q) =
    (peri_r(P ⌦_M M Q) ⇒ cmt_r P ⌦(_ok' ∧ N_0 M) ⇒ H_R[[true,true/$ok', $wait'']] cmt_r Q)
  by (simp add: peri-cmt-def parallel-commitment SRD-healths assms usubst unrest assms)
also have ... = (peri_r(P ⌦_M M Q) ⇒ cmt_r P ⌦(_∃ $st' · N_0 M)true,true/$ok', $wait''] cmt_r Q)
  by (simp add: parallel-pericondition-lemma1)
also have ... = (peri_r(P ⌦_M M Q) ⇒ cmt_r P ⌦(_∃ $st' · M) cmt_r Q)
  by (simp add: parallel-pericondition-lemma2 assms)
also have ... = (peri_r(P ⌦_M M Q) ⇒ (cmt_r P)_0 ⌦ (cmt_r Q)_1 ⌦ _v_< = _v) (¬ _0−wait ⌦ _1−wait)
∧ (∃ $st' · M))
  by (simp add: par-by-merge-alt-def parallel-pericondition-lemma3 seqr-or-distr)
also have ... = (peri_r(P ⌦_M M Q) ⇒ (peri_r P)_0 ⌦ (peri_r Q)_1 ⌦ _v_< = _v) (∃ $st' · M)
∧ (post_r P)_0 ⌦ (peri_r Q)_1 ⌦ _v_< = _v) (∃ $st' · M)
  by (simp add: seqr-right-one-point-true seqr-right-one-point-false cmt_r-def post_r-def peri_r-def usubst unrest assms)
also have ... = (peri_r(P ⌦_M M Q) ⇒ peri_r(P) ⌦_M peri_r(Q)
  ∨ post_r(P) ⌦_M peri_r(Q)
  ∨ peri_r(P) ⌦_M post_r(Q))
  by (simp add: par-by-merge-alt-def)
finally show ?thesis .

qed

lemma parallel-postcondition-lemma1:

(_ok' ∧ P) :: H_R[[true,false/$ok', $wait'']] = P[[true,false/$ok', $wait'']]
(is ?lhs = ?rhs)

proof
  have ?lhs = (_ok' ∧ P) :: H[[true,false/$ok', $wait'']
  by (rel-blast)
also have ... = ?rhs
  by (rel-auto)
finally show ?thesis .

qed

lemma parallel-postcondition-lemma2:

assumes M is RDM
shows (N_0(M)[[true,false/$ok', $wait'']]) = ((¬ _0−wait ⌦ ¬ _1−wait) ⌦ M)

proof
  have (N_0(M))[[true,false/$ok', $wait'']] = ((¬ _0−wait ⌦ ¬ _1−wait) ⌦ _tr' ≥ _tr < ⌦ M)
  by (simp add: usubst unrest nmerge-rd0-def ex-unrest Healthy-if R1m-def_assms
also have ... = ((¬ _0−wait ⌦ ¬ _1−wait) ⌦ M)
  by (metis Healthy-if R1m-def RDM-R1m_assms utp-pred-laws.inf-commute)
finally show ?thesis .

72
qed

lemma parallel-postcondition [rdes]:
fixes M :: (′s,′t::trace,′a) rsp merge
assumes P is SRD Q is SRD M is RDM
shows post\_R(P \parallel M R(M) Q) = (pre\_R (P \parallel M R M) \Rightarrow post\_R(P) \parallel M post\_R(Q))
proof –
  have post\_R(P \parallel M R(M) Q) =
    (pre\_R (P \parallel M R M) \Rightarrow\text{cnt}_R P || (\$ok' \land \text{N}_0 M);; \text{H}_R[\text{true}\text{-false}/\$ok', \$\text{wait}' ] cnt_R Q)
    by (simp add: post\text{-}cnt\text{-}def parallel\text{-}commitment assms usbst\text{-}unrest SRD\text{-}healths)
  also have ... = (pre\_R (P \parallel M R M Q) \Rightarrow\text{cnt}_R P || (\neg \$0\text{-}wait \land \neg \$1\text{-}wait \land M) cnt_R Q)
    by (simp add: parallel\text{-}postcondition\text{-}lemma1 parallel\text{-}postcondition\text{-}lemma2 assms,
      simp add: utp\text{-}pred\text{-}laws.inf\text{-}commute utp\text{-}pred\text{-}laws.inf\text{-}left\text{-}commute)
  also have ... = (pre\_R (P \parallel M R M Q) \Rightarrow post\_R(P) \parallel M post\_R(Q))
    by (simp add: par\text{-}by\text{-}merge\text{-}all\text{-}def segr\text{-}right\text{-}one\text{-}point\text{-}false usbst\text{-}unrest cnt\text{-}R\text{-}def post\text{-}R\text{-}def assms)
finally show \text{thesis} .

qed

lemma parallel-precondition\text{-}lemma:
fixes M :: (′s,′t::trace,′a) rsp merge
assumes P is NSRD Q is NSRD M is RDM
shows (\neg \text{pre}_R(P)) \parallel \text{N}_0(M) ;; \text{R}_1(true) \text{cnt}_R(Q) =
  ((\neg \text{pre}_R P) \parallel M ;; \text{R}_1(true) \text{peri}_R Q \lor (\neg \text{pre}_R P) \parallel M ;; \text{R}_1(true) \text{post}_R Q)
proof –
  have ((\neg \text{pre}_R(P)) \parallel \text{N}_0(M) ;; \text{R}_1(true) \text{cnt}_R(Q)) =
    ((\neg \text{pre}_R P) \parallel \text{M} ;; \text{R}_1(true) \text{peri}_R Q \lor (\neg \text{pre}_R P) \parallel \text{M} ;; \text{R}_1(true) \text{post}_R(Q))
    by (simp add: wait\text{-}cond\text{-}peri\text{-}post\text{-}cnt)
  also have ... = (((\neg \text{pre}_R(P))_0 \land \text{peri}_R(Q) \lor \text{post}_R(Q))_1 \land \$v' = _u \$v) ;; \text{N}_0(M) ;; \text{R}_1(true))
    by (simp add: par\text{-}by\text{-}merge\text{-}alt\text{-}def)
  also have ... = (((\neg \text{pre}_R(P))_0 \land [\text{peri}_R(Q)]_1 \land \$v' = _u \$v) \lor \$1\text{-}wait' \lor ([\neg \text{pre}_R(P)]_0 \land [\text{post}_R(Q)]_1 \land \$v' = _u \$v)) ;; \text{N}_0(M) ;; \text{R}_1(true))
    (is (if \text{P} ;; \text{Q} = (if \text{Q} ;; \text{P} )))
    by (simp add: wait\text{-}cond\text{-}def alpha)
  also have ... = (((\neg \text{pre}_R P)_0 \land [\text{peri}_R Q]_1 \land \$v' = _u \$v)[true/\$1\text{-}wait'] ;; \text{N}_0(M) ;; \text{R}_1(true))[\text{true}/\$1\text{-}wait]
    \lor ((\neg \text{pre}_R P)_0 \land [\text{post}_R Q]_1 \land \$v' = _u \$v)[false/\$1\text{-}wait'] ;; \text{N}_0(M) ;; \text{R}_1(true))
    by (simp add: cond\text{-}inter\text{-}var\text{-}split)
  also have ... = (((\neg \text{pre}_R P)_0 \land [\text{peri}_R Q]_1 \land \$v' = _u \$v) ;; \text{N}_0(M)[true/\$1\text{-}wait] ;; \text{R}_1 true \lor
    (\neg \text{pre}_R P)_0 \land [\text{post}_R Q]_1 \land \$v' = _u \$v) ;; \text{N}_0(M)[false/\$1\text{-}wait] ;; \text{R}_1 true)
    by (simp add: usbst\text{-}unrest)
  also have ... = (((\neg \text{pre}_R P)_0 \land [\text{peri}_R Q]_1 \land \$v' = _u \$v) ;; \text{R}_1 true \lor
    (\neg \text{pre}_R P)_0 \land [\text{post}_R Q]_1 \land \$v' = _u \$v) ;; (\text{wait'} \land M) ;; \text{R}_1 true)
    by (simp add: RDM-R1m[OF assms(3)])
proof –
  have ($str' \geq _u \$tr < \land M) = M
    using RDM-R1m[of M]
    by (simp add: Healthy\text{-}def R1m\text{-}def conj\text{-}comm)
thus \( \?)thesis
by (simp add: nmerge-rd0-def unrest assms closure ex-unrest usbst)
qed

also have \( \ldots = ((\neg p \ldotp \pre R \ldotp P) \land [\peri R \ldotp Q]_1 \land s v' = u \ldotp s v) ;; M ;; \rd 1 \true \lor \\
(\neg p \ldotp \pre R \ldotp P) \land [\post R \ldotp Q]_1 \land s v' = u \ldotp s v) ;; M ;; \rd 1 \true) \\
(is (?P_1 \lor_p ?P_2) = (?Q_1 \lor ?Q_2))
proof –
have \( ?P_1 = ((\neg p \ldotp \pre R \ldotp P) \land [\peri R \ldotp Q]_1 \land s v' = u \ldotp s v) ;; (M \land s \wait') ;; \rd 1 \true \\
  by (simp add: conj-comm)
hence 1: ?P_1 = ?Q_1
by (simp add: seqr-left-one-point-true seqr-left-one-point-false add: unrest usbst closure assms)
have \( ?P_2 = ((\neg p \ldotp \pre R \ldotp P) \land [\post R \ldotp Q]_1 \land s v' = u \ldotp s v) ;; (M \land s \wait') ;; \rd 1 \true \lor \\
(\neg p \ldotp \pre R \ldotp P) \land [\post R \ldotp Q]_1 \land s v' = u \ldotp s v) ;; (M \land s \wait') ;; \rd 1 \true) \\
by (subst seqr-boot-split[of left-uvar \wait], simp-all add: usbst unrest assms closure conj-comm)
hence 2: ?P_2 = ?Q_2
by (simp add: seqr-left-one-point-true seqr-left-one-point-false unrest usbst closure assms)
from 1 \& 2 show \( ?\)thesis by simp
qed

also have \( \ldots = ((\neg p \ldotp \pre R \ldotp P) \parallel M ;; \rd 1(\true) \peri R \ldotp Q \lor (\neg p \ldotp \pre R \ldotp P) \parallel M ;; \rd 1(\true) \post R \ldotp Q) \\
by (simp add: par-by-merge-alt-def)
finally show \( ?\)thesis .
qed

lemma swap-nmerge-rd0:
\( \ldots = (\swa_m :: \ldots) \\
by (rel-auto, meson+)

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

lemma swap-merge-rd':
\( \ldots = (\swa_m :: \ldots) \\
by (rel-blast)

lemma swap-merge-rd:
\( \ldots = (\swa_m :: \ldots) \\
by (simp add: merge-rd-def seqr-assoc[THEN sym] swap-merge-rd')

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

lemma nmerge-rd1-merge3:
assumes \( M \is \rdm \\
shows \( M_3(N_1(M)) = (sok' = u \ldotp (s_0-\ok \land s_1-\ok) \land \\
\ldots = (s_0-\wait \lor s_1-\wait) \land \\
M_3(M)) \\
proof –
have \( M_3(N_1(M)) = M_3(sok' = u \ldotp (s_0-\ok \land s_1-\ok) \land \\
\ldots = (s_0-\wait \lor s_1-\wait) \land \\
M_3(M)) \\
by (simp add: nmerge-rd1-def nmerge-rd0-def assms Healthy-if)
also have \( \ldots = M_3(sok' = u \ldotp (s_0-\ok \land s_1-\ok) \land \\
\ldots = (s_0-\wait \lor s_1-\wait) \land \rdm(M)) \\
74
lemma nmerge-rd-merge3:
M3(N_R(M)) = (\exists \text{st}_< \cdot \text{sv}_< = u \text{sv}_<) < \text{swap}_< > M3(N_1 \ M) \ < \text{sk}_< > (\text{tr}_< \leq u \text{tr}_<)
by (rel-blast)

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

lemma parallel-precondition:
fixes M :: ('s,'t::\text{trace},'a) \text{rsp merge}
assumes P is NSRD Q is NSRD M is RDM
shows \(\text{pre}_R(P \parallel R(M) \ Q) =\)
\((\neg_r ((\neg_r \text{pre}_R(P) \ \parallel_M ;; \ R1(\text{true}) \ \text{peri}_R \ Q) \wedge\)
\neg_r ((\neg_r \text{pre}_R(P) \ \parallel_M ;; \ R1(\text{true}) \ \text{post}_R \ Q) \wedge\)
\neg_r ((\neg_r \text{pre}_R(Q) \ \parallel_{\text{swap}_m ;; M} ;; \ R1(\text{true}) \ \text{peri}_R \ P) \wedge\)
\neg_r ((\neg_r \text{pre}_R(Q) \ \parallel_{\text{swap}_m ;; M} ;; \ R1(\text{true}) \ \text{post}_R \ P))
proof
have a: (\neg_r \text{pre}_R(P)) \ \parallel_M ;; \ R1(\text{true}) \ \text{cnt}_R(Q) =
((\neg_r \text{pre}_R(P)) \ \parallel_M ;; \ R1(\text{true}) \ \text{peri}_R \ Q \ \vee \ (\neg_r \text{pre}_R(P)) \ \parallel_M ;; \ R1(\text{true}) \ \text{post}_R \ Q)
by (simp add: parallel-precondition-lemma \ text{assms})

have b: (\neg_r \text{cnt}_R \ P \ \parallel_M \ \text{R1 true} \ (\neg_r \text{pre}_R(Q)) =
(\neg_r (\neg_r \text{pre}_R(Q)) \ \parallel_M \ \text{R1 true} \ \text{cnt}_R(P))
by (simp add: swap-nmerge-rd[\text{THEN \ sym}] seqr-assoc[\text{THEN \ sym}] \ \text{par-by-merge-def \ par-sep-swap})

have c: (\neg_r \text{pre}_R(Q)) \ \parallel_M \ \text{R1 true} \ (\text{peri}_R \ P \ \vee \ (\neg_r \text{pre}_R(Q)) \ \parallel_M ;; \ R1(\text{true}) \ \text{post}_R \ P)
by (simp add: parallel-precondition-lemma closure \text{assms})

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

Weakest Parallel Precondition

definition wrR ::
('t::\text{trace},'a) \text{hrel-rp} \Rightarrow
('t :: \text{trace},'a) \text{rp merge} \Rightarrow
('t, 'a) \text{hrel-rp} \Rightarrow
where \( \text{upred-defs} \): \( Q \wr_R(M) P = (\neg_r ((\neg_r P) \parallel_M \\langle R1 \rangle Q)) \)

**Lemma wrR-1** [closure]:
\( M \) is \( R1m \) \( \Rightarrow \) \( Q \wr_R(M) P \) is \( R1 \)
by (simp add: wrR-def closure)

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

**Lemma wrR-R2-lemma**:
assumes \( P \) is \( R2 \) \( Q \) is \( R2 \) \( M \) is \( R2m \)
shows \( (\neg_r P) \parallel_M Q \) is \( R2 \)
proof –
have \( (\neg_r P) \parallel_M Q \) is \( R2 \)
by (simp add: closure assms)
thus \(?thesis\)
by (simp add: closure)
qed

**Lemma wrR-R2** [closure]:
assumes \( P \) is \( R2 \) \( Q \) is \( R2 \) \( M \) is \( R2m \)
shows \( Q \wr_R(M) P \) is \( R2 \)
proof –
have \( (\neg_r P) \parallel_M Q \) is \( R2 \)
by (simp add: wrR-R2-lemma assms)
thus \(?thesis\)
by (simp add: wrR-def wrR-R2-lemma par-by-merge-seq-add closure)
qed

**Lemma wrR-RR** [closure]:
assumes \( P \) is \( RR \) \( Q \) is \( RR \) \( M \) is \( RDM \)
shows \( Q \wr_R(M) P \) is \( RR \)
apply (rule RR-intro)
apply (simp-all add: unrest assms closure wrR-def rpred)
apply (metis (no-types, lifting) Healthy-def' R1-R2c-commute R1-R2c-is-R2 R1-rea-not RDM-R2m
RR-implies-R2 assms(1) assms(2) assms(3) par-by-merge-seq-add rea-not-R2-closed
wrR-R2-lemma)
done

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

**Lemma wprR-choice** [wp]: \( (P \lor Q) \wr_R(M) R = (P \wr_R(M) R \land Q \wr_R(M) R) \)
proof –
have \( (P \lor Q) \wr_R(M) R \)
\( = \)
\( (\neg_r ((\neg_r R) \parallel U0 \land (P \parallel U1 \lor Q \parallel U1) \land \$v < \$v \parallel \parallel M \parallel true_r)) \)
by (simp add: wrR-def par-by-merge-def seqr-or-distl)
also have \( \ldots = (\neg_r ((\neg_r R) \parallel U0 \land P \parallel U1 \land \$v < \$v \parallel (\neg_r R) \parallel U0 \land Q \parallel U1 \land \$v < \$v) \)

76
\[ \text{\$v \implies M \implies \text{true}_\text{r}} \]

by (simp add: conj-disj-distr utp-pred-laws,inf-sup-distrib)

also have ... = (\text{\text{\$v} \implies u \implies M \implies \text{true}_\text{r}})

by (simp add: seq-or-distl)

also have ... = (P \implies M \implies \text{true}_\text{r})

by (simp add: \text{wr-R-def})

finally show \(?thesis\).

qed

lemma uppR-miracle [wp]: false \text{wr}_R(M) P = \text{true}_\text{r}

by (simp add: \text{wr-R-def})

lemma uppR-true [wp]: P \text{wr}_R(M) \text{true}_\text{r} = \text{true}_\text{r}

by (simp add: \text{wr-R-def})

lemma parallel-precondition-wr [rdes]:

assumes P is NSRD Q is NSRD M is RDM

shows \text{pre}_R(P \parallel_M \text{wr}_R(M)) \parallel_M \text{wr}_R(Q) \parallel_M \text{pre}_R(P) \land \text{post}_R(Q) \parallel_M \text{wr}_R(M) \parallel_M \text{pre}_R(P) \land \text{pre}_R(P) \parallel_M \text{wr}_R(\text{swap}_M \parallel_M M) \parallel_M \text{pre}_R(Q) \parallel_M \text{post}_R(P) \parallel_M \text{wr}_R(\text{swap}_M \parallel_M M) \parallel_M \text{pre}_R(Q))

by (simp add: \text{assms parallel-precondition \text{wr-R-def}})

lemma parallel-rdes-def [rdes-def]:

assumes P \parallel_M P_2 is RR P_3 is RR Q_1 is RR Q_2 is RR Q_3 is RR

\$st \parallel_M P_2 \parallel_M \text{swap}_M \parallel_M Q_2 \parallel_M M \parallel_M \text{RDM}

shows \text{R}_s((P_1 \parallel_M P_2 \parallel_M P_3) \parallel_M \text{wr}_R(M)) \parallel_M \text{R}_s(Q_1 \parallel_M Q_2 \parallel_M Q_3) = \text{R}_s((Q_1 \parallel_M Q_2) \parallel_M \text{wr}_R(M) \parallel_M \text{pre}_R(P) \land \text{post}_R(Q) \parallel_M \text{wr}_R(M) \parallel_M \text{pre}_R(P) \parallel_M \text{pre}_R(Q)))

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

proof

have \(?lhs = \text{R}_s(\text{pre}_R ?lhs \parallel_M \text{peri}_R ?lhs \parallel_M \text{post}_R ?lhs)

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

also have ... = \(?rhs\)

by (simp add: \text{rdes closure unrest \text{assms}, rel-auto})

finally show \(?thesis\).

qed

lemma Miracle-parallel-left-zero:

assumes P is SRD M is RDM

shows Miracle \parallel_M P = Miracle

proof

have \text{pre}_R(Miracle \parallel_M P) = \text{true}_\text{r}

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

moreover hence \text{cmt}_R(Miracle \parallel_M P) = \text{false}

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

ultimately have Miracle \parallel_M P = \text{R}_s(\text{true}_\text{r} \parallel_M \text{false})

by (metis \text{NSRD-iff SRD-reactive-design-alt assms par-rdes-NSRD srdes-theory-continuous.weak.top-clased})

thus \(?thesis\)

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

qed

lemma Miracle-parallel-right-zero:
assumes $P$ is SRD $M$ is RDM
shows $P \parallel_M^r \text{Miracle} = \text{Miracle}$
proof –
  have $\prev_R(P \parallel_M^r \text{Miracle}) = \text{true}$
    by (simp add: wait'-cond-idem parallel-assm rdes closure assms)
moreover hence $\ctc_R(P \parallel_M^r \text{Miracle}) = \text{false}$
  by (simp add: wait'-cond-idem rdes closure SRD-healths assms)
ultimately have $P \parallel_M^r \text{Miracle} = \text{R} \alpha (\text{true} \vdash \text{false})$
  by (metis NSRD-iff SRD-reactive-design-alt assms par-rdes-NSRD srdes-theory-continuous.weak.top-closed)
thus \(?\thesis\)
  by (simp add: Miracle-def R1-design-R1-pre)
qed

8.1 Example basic merge

definition BasicMerge :: \((\text{'s, 't::trace, unit} \text{ rsp}) \text{ merge} (N_B)) \text{ where}
  [upred-defs]: BasicMerge = \((\text{str}_< \leq_u \text{str}' \land \text{str}' - \text{str}_< =_u \text{st}_0 - \text{tr} - \text{str}_< \land \text{str}' - \text{str}_< =_u \text{st}_1 - \text{tr})\)
abbreviation \(r\text{basic-par} - \parallel_B^{[85,86]} \parallel_B 85\) \text{ where}
  \(P \parallel_B Q \equiv P \parallel_M^{(N_B)} Q\)

lemma BasicMerge-RDM \text{ [closure]}: $N_B$ is RDM
  by (rule RDM-intro, (rel-auto)+)

lemma BasicMerge-SymMerge \text{ [closure]}:
  $N_B$ is SymMerge
  by (rel-auto)

lemma BasicMerge'-calc:
  assumes $\text{sok}' \not\in P \text{s\text{\textquoteleft\text{\textquoteleft wait\textquoteleft\textquoteleft}}} \not\in P \text{s\text{\textquoteleft\text{\textquoteleft wait\textquoteleft\textquoteleft}}} Q \not\in P \text{is R2 Q is R2}$
  shows $P \parallel_N^B Q = ((\exists \text{st}' \cdot P) \land (\exists \text{st}' \cdot Q) \land \text{st}' =_u \text{st})$
  using assms
proof –
  have $P : ((\exists \text{sok}' \cdot \text{wait}') \cdot \text{R2}(P)) = P$ \text{is \?P' = -}
    by (simp add: ex-unrest ex-plus Healthy-if assms)
  have $Q : ((\exists \text{sok}' \cdot \text{wait}') \cdot \text{R2}(Q)) = Q$ \text{is \?Q' = -}
    by (simp add: ex-unrest ex-plus Healthy-if assms)
  have \(?P' \parallel_N^B \parallel_B Q' = ((\exists \text{st}' \cdot \?P') \land (\exists \text{st}' \cdot \?Q') \land \text{st}' =_u \text{st})$
    by (simp add: par-by-merge-alt-def, rel-auto, blast+)
  thus \(?\thesis\)
    by (simp add: P Q)
qed

8.2 Simple parallel composition

definition \text{rea-design-par} :: \((\text{'s, 't::trace, \text{'a}) \text{ hrel-rsp} \Rightarrow (\text{'s, 't, \text{'a}) \text{ hrel-rsp} \Rightarrow (\text{'s, 't, \text{'a}) \text{ hrel-rsp}} \text{ (infixr \parallel_R 85)}$
where [upred-defs]: $P \parallel_R Q = \text{R}_a((\prev_R(P) \land \prev_R(Q)) \vdash (\ctc_R(P) \land \ctc_R(Q)))$

lemma RHS-design-par:
  assumes $\text{sok}' \not\in P_1 \text{sok}' \not\in P_2$
  shows $\text{R}_a(P_1 \vdash Q_1) \parallel_R \text{R}_a(P_2 \vdash Q_2) = \text{R}_a((P_1 \land P_2) \vdash (Q_1 \land Q_2))$
proof –
  have $\text{R}_a(P_1 \vdash Q_1) \parallel_R \text{R}_a(P_2 \vdash Q_2) = \ldots$
R(P₁[true,false/ok,wait] ⊨ Q₁[true,false/ok,wait]) ⊨ R(P₂[true,false/ok,wait] ⊨ Q₂[true,false/ok,wait])
by (simp add: RHS-design-ok-wait)

also from assms
have ...
  R₄((R₁(R₂c(P₁)) ∧ R₁(R₂c(P₂)))[true,false/ok,wait]) ⊨
   (R₁(R₂c(P₁ ⇒ Q₁)) ∧ R₁(R₂c(P₂ ⇒ Q₂))[true,false/ok,wait])
apply (simp add: rea-design-par-def rea-pre-RHS-design rea-cmt-RHS-design unsubst unrest assms)
apply (rule cong[of R₄ R₄], simp)
using assms apply (rel-auto)
done
also have ...
  R₄((R₂c(P₁) ∧ R₂c(P₂)) ⊨
   (R₁(R₂s(P₁ ⇒ Q₁)) ∧ R₁(R₂s(P₂ ⇒ Q₂))))
by (simp add: R₂c-R3h-commute R₂c-and R₂c-design R₂c-idem R₂c-not RHS-def)
also have ...
  R₄((P₁ ∧ P₂) ⊨ (P₁ ⇒ Q₁) ∧ R₁(R₂s(P₂ ⇒ Q₂)))
by (metis (no-types, lifting) R₁-conj R₂s-conj RHS-design-export-R₁ RHS-design-export-R₂s)
also have ...
  R₄(((P₁ ∧ P₂) ⊨ (Q₁ ∧ Q₂))
by (rule cong[of R₄ R₄], simp, rel-auto)
finally show ?thesis .
qed

lemma RHS-tri-design-par:
  assumes $ok' ∉ P₁ $ok' ∉ P₂
shows R₄(P₁ ⊨ Q₁ ⊨ R₁) ||ₐ R₄(P₂ ⊨ Q₂ ⊨ R₂) = R₄((P₁ ∧ P₂) ⊨ (Q₁ ∧ Q₂) ⊨ (R₁ ∧ R₂))
by (simp add: RHS-design-par assms unrest wait l⊥-cond-conj-exchange)

lemma RHS-tri-design-par-RR [rdes-def]:
  assumes P₁ is RR P₂ is RR
shows R₄(P₁ ⊨ Q₁ ⊨ R₁) ||ₐ R₄(P₂ ⊨ Q₂ ⊨ R₂) = R₄((P₁ ∧ P₂) ⊨ (Q₁ ∧ Q₂) ⊨ (R₁ ∧ R₂))
by (simp add: RHS-tri-design-par assms)

lemma RHS-comp-assoc:
  assumes P is NSRD Q is NSRD R is NSRD
shows (P ||ₐ Q) ||ₐ R = P ||ₐ Q ||ₐ R
by (rdes-eq cls: assms)

end

9 Productive Reactive Designs

theory utp-rdes-productive
  imports utp-rdes-parallel
begin

9.1 Healthiness condition

A reactive design is productive if it strictly increases the trace, whenever it terminates. If it
does not terminate, it is also classed as productive.

definition Productive :: (′s, ′t:trace, ′a) hrel-rsp ⇒ (′s, ′t, ′a) hrel-rsp where
lemma Productive-RHS-design-form:
assumes Sok ′ ⊢ P Sok ′ ⊢ Q Sok ′ ⊢ R
shows Productive(R_s(P ⊢ Q ⊢ R)) = R_s(P ⊢ Q ⊢ (R ∧ $tr <_u $tr′))
using assms by (simp add: Productive-def RHS-tri-design-par unrest)

lemma Productive-form:
Productive(SRD(P)) = R_s(pre_R(P) ⊢ peri_R(P) ⊢ (post_R(P) ∧ $tr <_u $tr′))
proof –
have Productive(SRD(P)) = R_s(pre_R(P) ⊢ peri_R(P) ⊢ (post_R(P) ∧ $tr <_u $tr′))
  by (simp add: Productive-def SRD-as-reactive-tri-design)
also have ... = R_s(pre_R(P) ⊢ peri_R(P) ⊢ (post_R(P) ∧ $tr <_u $tr′))
  by (simp add: RHS-tri-design-par unrest)
finally show ?thesis .

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

lemma Productive-intro:
assumes P is SRD ($tr <_u $tr′) ⊆ (pre_R(P) ∧ post_R(P)) $wait ′ ∈ pre_R(P)
shows P is Productive
proof –
have P.R_s(pre_R(P) ⊢ peri_R(P) ⊢ (post_R(P) ∧ $tr <_u $tr′)) = P
  by (metis no-types hide-lams design-export-pre wait′-cond-conj-exchange wait′-cond-idem)
also have ... = R_s(pre_R(P) ⊢ (pre_R(P) ∧ peri_R(P)) ⊢ (pre_R(P) ∧ peri_R(P)) ⊢ (pre_R(P) ∧ post_R(P)))
  by (metis assms(2) utp-pred-laws.inf.absorb1 utp-pred-laws.inf.assoc)
also have ... = R_s(pre_R(P) ⊢ peri_R(P) ⊢ (post_R(P) ∧ $tr <_u $tr′))
  by (metis no-types hide-lams design-export-pre wait′-cond-conj-exchange wait′-cond-idem)
finally show ?thesis
  by (simp add: SRD-reactive-tri-design assms(1))

demonstration thus ?thesis
by (metis Healthy-def RHS-tri-design-par Productive-def ok′-pre-unrest unrest-true utp-pred-laws.inf-right-idem utp-pred-laws.inf-top-right)

demonstration

lemma Productive-post-refines-tr-increase:
assumes P is SRD $wait ′ ∈ pre_R(P)
shows ($tr <_u $tr′) ⊆ (pre_R(P) ∧ post_R(P))
proof –
have post_R(P) = post_R(R_s(pre_R(P) ⊢ peri_R(P) ⊢ (post_R(P) ∧ $tr <_u $tr′)))
  by (metis Healthy-def Productive-form assms(1) assms(2))
also have ... = R_l(R_{≤c}(pre_R(P) ⇒ (post_R(P) ∧ $tr <_u $tr′)))
  by (simp add: rea-post-RHS-design unrest usubst assms, rel-auto)
also have ... = R_l((pre_R(P) ⇒ (post_R(P) ∧ $tr <_u $tr′)))
  by (simp add: R_{≤c}-impl R_{≤c}-preR R_{≤c}-postR R_{≤c}-and R_{≤c}-tr-less-tr′ assms)
also have ($tr <_u $tr′) ⊆ (pre_R(P) ∧ ...)
  by (rel-auto)
finally show ?thesis .

demonstration

80
lemma Continuous-Productive [closure]: Continuous Productive
  by (simp add: Continuous-def Productive-def, rel-auto)

9.2 Reactive design calculations

lemma preR-Productive [rdes]:
  assumes P is SRD
  shows \( P \rightarrow_r (P \rightarrow (\textit{post}(P) \land \#tr < u \#tr')) \)
  proof
    have \( P \rightarrow (P \rightarrow (\textit{peri}(P) \rightarrow (\textit{post}(P) \land \#tr < u \#tr')) \)
      by (metis Healthy-def Productive-form assms)
    thus \(?thesis \)
      by (simp add: rea-Pre-RHS-design unsubst unrest R2c-not R2c-preR R1-preR Healthy-if assms)
  qed

lemma periR-Productive [rdes]:
  assumes P is NSRD
  shows \( P \rightarrow (P \rightarrow (\textit{peri}(P) \rightarrow (\textit{post}(P) \land \#tr < u \#tr')) \)
  proof
    have \( P \rightarrow (P \rightarrow (\textit{peri}(P) \rightarrow (\textit{post}(P) \land \#tr < u \#tr')) \)
      by (metis Healthy-def Productive-form assms)
    also have \(?thesis \)
      by (simp add: rea-Peri-RHS-design unsubst unrest R2c-not R2c-preR R1-preR Healthy-if assms)
  qed

lemma postR-Productive [rdes]:
  assumes P is NSRD
  shows \( P \rightarrow (P \rightarrow (\textit{post}(P) \land \#tr < u \#tr')) \)
  proof
    have \( P \rightarrow (P \rightarrow (\textit{post}(P) \land \#tr < u \#tr')) \)
      by (metis Healthy-def Productive-form assms)
    also have \(?thesis \)
      by (simp add: SRD-peri-under-pre assms unrest closure)
  qed

lemma preR-frame-seq-export:
  assumes P is NSRD P is Productive Q is NSRD
  shows \( (P \rightarrow (P \rightarrow (P \rightarrow (Q \rightarrow (P \rightarrow Q)))) \)
  proof
    have \( (P \rightarrow (P \rightarrow (P \rightarrow (Q \rightarrow (P \rightarrow Q)))) \)
      by (simp add: SRD-post-under-pre assms unrest)
    also have \(?thesis \)
      by (simp add: NSRD-is-SRD R1-post-SRD assms(1) rea-impl-def seq-or-distl R1-preR Healthy-if)
  qed

  have \( P \rightarrow (\neg_r P \rightarrow Q) \)
    by (simp add: R1-preR rea-not-or)
then show \( ?thesis \)
by (metis (no-types, lifting) R1-def conj-comm disj-comm disj-conj-distr rea-impl-def seqr-or-distr uts-pred-laws.inf-top-left uts-pred-laws.\supakedown-left-ids)
qed
also have \( ... = (pre_R P \land (\neg \neg pre_R P) \lor (pre_R P \land \varphi_R P) \land Q) \)
by (simp add: NSRD-neg-pre-left-zero assms closure SRD-healths)
also have \( ... = (pre_R P \land (pre_R P \land \varphi_R P) \land Q) \)
by (rel-blast)
finally show \( ?thesis \)
qed

9.3 Closure laws

lemma Productive-rdes-intro:
assumes \((\$tr < u \$tr ) \subseteq R \$ok' \not\in P \$ok' \not\in Q \$ok' \not\in R \$\wait \not\in P \$\wait \not\in P\)
shows \((R_s (P \vdash Q \circ R))\) is Productive
proof (rule Productive-intro)
show \( R_s (P \vdash Q \circ R) \) is SRD
by (simp add: RHS-tri-design-is-SRD assms)

from assms(1) show \((\$tr' > u \$tr) \subseteq (pre_R R_s (P \vdash Q \circ R)) \land \varphi_R R_s (P \vdash Q \circ R))\)
apply (simp add: rea-pre-RHS-design rea-post-RHS-design usubst assms unrest)
apply (rel-auto)
apply fastforce
done
show \( \$\wait \not\in pre_R R_s (P \vdash Q \circ R)\)
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 \( R_4 \)
shows \((R_s (P \vdash Q \circ R))\) 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 \( P \) is Productive \( Q \) is SRD \( Q \) is Productive
shows \( P \sqcap Q \) is Productive
proof
have \( P \sqcap Q = \)
\begin{align*}
R_s (pre_R(P) \vdash peri_R(P) \circ (post_R(P) \land \$tr < u \$tr')) \sqcap R_s (pre_R(Q) \vdash peri_R(Q) \circ (post_R(Q) \land \$tr < u \$tr'))
\end{align*}
by (metis Healthy-if Productive-form assms)
also have \( ... = R_s ((pre_R P \land pre_R Q) \vdash (peri_R P \lor peri_R Q) \circ ((post_R P \land \$tr' > u \$tr) \lor (post_R Q \land \$tr' > u \$tr)))
\end{align*}

by (simp add: RHS-tri-design-choice)
also have $\dots = R_s \ (\pre_R P \land \pre_R Q) \vdash (\peri_R P \lor \peri_R Q) \circ ((\post_R P) \lor (\post_R Q)) \land \str’ >_u \str)$
  by (rule cong[of $R_s$, simp, rel-auto]
also have $\dots$ 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 $P \circ b \tri_R Q$ is Productive
proof –
  have $P \circ b \tri_R Q =$
    $\neg \neg (\peri_R(P) \circ (\post_R(P) \land \str <_u \str’)) \circ b \tri_R \neg \neg (\peri_R(Q) \circ (\post_R(Q) \land \str’ <_u \str))$
    by (metis Healthy-if Productive-form assms)
also have $\dots =$ $\neg \neg \ (\peri_R(P) \circ (\post_R(P) \land \str’ <_u \str)) \circ b \tri_R \neg \neg \ (\peri_R(Q) \circ (\post_R(Q) \land \str’ >_u \str))$
    by (simp add: cond-srea-form)
also have $\dots =$ $\neg \neg \neg \ (\peri_R(P) \circ (\post_R(P) \land \str >_u \str)) \circ b \tri_R \neg \neg \neg \ (\peri_R(Q) \circ (\post_R(Q) \land \str’ <_u \str))$
    by (rule cong[of $R_s$, simp, rel-auto]
also have $\dots$ is Productive
  by (simp add: Healthy-def Productive-RHS-design-form unrest)
finally show $\thesis$.
qed

lemma Productive-seq-1 [closure]:
assumes $P$ is NSRD $P$ is Productive $Q$ is NSRD
shows $P \circ Q$ is Productive
proof –
  have $P \circ Q =$ $R_s \ (\peri_R(P) \circ (\post_R(P) \land \str <_u \str’)) \circ R_s \ (\peri_R(Q) \circ (\post_R(Q)))$
    by (metis Healthy-def NSRD-is-SRD SRD-reactive-tri-design Productive-form assms(1) assms(2) assms(3))
also have $\dots =$ $\neg \neg \ (\peri_R(P) \circ (\post_R(P) \land \str >_u \str)) \circ \neg \neg \ (\peri_R(Q) \circ (\post_R(Q)))$
    by (simp add: RHS-tri-design-composition-wp rpred unrest closure assms wp NSRD-neg-pre-left-zero SRD-healths ex-unrest wp-reas-def disj-upred-def)
also have $\dots =$ $\neg \neg \neg \ (\peri_R(P) \circ (\post_R(P) \land \str >_u \str)) \circ \neg \neg \neg \ (\peri_R(Q) \circ (\post_R(Q)))$
    by (simp add: R1-post-SRD assms)
proof –
  have $\thesis$.
  thus $\thesis$
    by (simp add: NSRD-is-SRD R1-post-SRD assms)
qed
also have $\dots$ is Productive
  by (rule Productive-rdes-intro, simp-all add: unrest assms closure wp-rea-def)
finally show $\thesis$.
qed

83
lemma Productive-seq-2 [closure]:
  assumes P is NSRD Q is NSRD Q is Productive
  shows P ;; Q is Productive
proof
  have P ;; Q = Rs ((pre R (P) v (post R (P))) ;; Rs ((pre R (Q) v (post R (Q)))(post R (Q) ^ $tr <u $tr')))
    by (metis Healthy-def NSRD-is-SRD SRD-reactive-tri-design Productive-form assms)
  also have ... = Rs ((pre R (P) v (post R (P) wp_v pre R Q) v (peri R P v (post R P ;; peri R Q)))(post R P ;; (post R Q ^ $tr' >u $tr))
    by (simp add: RHS-tri-design-composition-wpred unrest closure assms wp NSRD-neg-pre-left-zero)
  also have ... = Rs ((pre R (P) v (post R (R1 (post R P) ;; (post R Q ^ $tr' >u $tr)) ^ $tr' >u $tr))
    by (rel-auto)
  thus ?thesis
    by (simp add: NSRD-is-SRD R1-post-SRD assms)
qed
also have ... is Productive
  by (rule Productive-rdes-intro, simp-all add: unrest assms closure wp-reas-def)
finally show ?thesis.
qed

theory utp-rdes-guarded
  imports utp-rdes-productive
begin

10.1 Traces with a size measure

Guarded recursion relies on our ability to measure the trace’s size, in order to see if it is decreasing on each iteration. Thus, we here equip the trace algebra with the ucard function that provides this.

class size-trace = trace + size +
  assumes
    size-zero: size 0 = 0 and
    size-nzero: s > 0 => size(s) > 0 and
    size-plus: size (s + t) = size(s) + size(t)
  -- These axioms may be stronger than necessary. In particular, 0 < ?s => 0 < #u(?s) requires that a non-empty trace have a positive size. But this may not be the case with all trace models and is possibly more restrictive than necessary. In future we will explore weakening.
begin

lemma size-mono: s <= t => size(s) <= size(t)
  by (metis le-add1 local.diff-add-cancel-left' local.size-plus)

lemma size-strict-mono: s < t => size(s) < size(t)
by (metis cancel-ab-semigroup-add-class.add-diff-cancel-left' local.diff-add-cancel-left' local.less-iff local.minus-gr-zero-iff local.size-nzero local.size-plus zero-less-diff)

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

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

end

Both natural numbers and lists are measurable trace algebras.

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

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

syntax
-usize :: logic \(\Rightarrow\) logic (size\(_u\)('t'))

translations
size\(_u\)(t) == CONST uop CONST size t

10.2 Guardedness

definition gvrт :: (('t::size-trace,α) rp × ('t,α) rp) chain where
[upred-defs]: gvrт(n) \equiv \(\forall tr. \text{tr} \leq u \text{tr} \}.{size\(_u\)(tt) < u < n}\

lemma gvrт-chain: chain gvrт
  apply (simp add: chain-def, safe)
  apply (rel-simp)
  apply (rel-simp)+
done

lemma gvrт-limit: \(\bigcap (\text{range gvrт}) = (\forall tr. \text{tr} \leq u \text{tr} \).\)
by (rel-auto)

definition Guarded :: (('t::size-trace,α) hrel-rp ⇒ ('t,α) hrel-rp) ⇒ bool where
[upred-defs]: Guarded(F) = (\forall X n. (F(X) ∧ gvrт(n+1)) = (F(X) ∧ gvrт(n)) ∧ gvrт(n+1)))

lemma GuardedI: \(\forall X n. (F(X) \land gvrт(n+1)) = (F(X) \land gvrт(n)) \land gvrт(n+1)) \][ \implies Guarded F \)
by (simp add: Guarded-def)

Guarded reactive designs yield unique fixed-points.

theorem guarded-fp-uniq:
  assumes mono F F ∈ [id]_H \rightarrow [SRD]_H Guarded F
  shows \(\mu F = \nu F \)
proof -
  have constr F gvrт
    using assms
    by (auto simp add: constr-def gvrт-chain Guarded-def tcontr-alt-def')
  hence \(\forall tr. \text{tr} \leq u \text{tr} \). \(\text{tr} \leq u \text{tr} \).\)
    apply (rule constr-fp-uniq)
    apply (simp add: assms)
  done
using gvt-limit apply blast
done
moreover have $(\exists tr \leq u \exists tr' \land \mu F) = \mu F
proof
  have $\mu F$ is R1
    by (rule SRD-healths(1), rule Healthy-mu, simp-all add: assms)
  thus $\text{thesis}$
    by (metis Healthy-def R1-def conj-comm)
qed
moreover have $(\exists tr \leq u \exists tr' \land \nu F) = \nu F
proof
  have $\nu F$ is R1
    by (rule SRD-healths(1), rule Healthy-nu, simp-all add: assms)
  thus $\text{thesis}$
    by (metis Healthy-def R1-def conj-comm)
qed
ultimately show $\text{thesis}$
  by (simp)
qed

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

lemma UINF-Guarded [closure]:
  assumes $\bigwedge P. P \in A \implies \text{Guarded} P$
  shows $\text{Guarded} (\lambda X. \bigwedge P \in A \cdot P(X))$
proof (rule GuardedI)
  fix $X n$
  have $\bigwedge Y. (\bigwedge P \in A \cdot P Y) \land gvt(n+1)) = (\bigwedge P \in A \cdot (P Y \land gvt(n+1)) \land gvt(n+1))$
proof
  fix $Y$
  let $?lhs = (\bigwedge P \in A \cdot P Y) \land gvt(n+1))$ and $?rhs = (\bigwedge P \in A \cdot (P Y \land gvt(n+1)) \land gvt(n+1))$
  have $a: ?lhs[[false/$ok]] = ?rhs[[false/$ok]]$
    by (rel-auto)
  have $b: ?lhs[[true/$ok]][true/$wait] = ?rhs[[true/$ok]][true/$wait]$
    by (rel-auto)
  have $c: ?lhs[[true/$ok]][false/$wait] = ?rhs[[true/$ok]][false/$wait]$
    by (rel-auto)
  show $?lhs = ?rhs$
    using $a \ b \ c$
    by (rule-tac bool-eq-splitI[of in-var $ok], simp, rule-tac bool-eq-splitI[of in-var $wait], simp-all)
qed
moreover have $(\bigwedge P \in A \cdot (P X \land gvt(n+1))) \land gvt(n+1)) = (\bigwedge P \in A \cdot (P (X \land gvt(n)) \land gvt(n+1))) \land gvt(n+1))$
proof
  have $(\bigwedge P \in A \cdot (P X \land gvt(n+1))) = (\bigwedge P \in A \cdot (P (X \land gvt(n)) \land gvt(n+1)))$
  proof (rule UINF-cong)
    fix $P$ assume $P \in A$
    thus $(P X \land gvt(n+1)) = (P (X \land gvt(n)) \land gvt(n+1))$
      using Guarded-def assms by blast
  qed
  thus $\text{thesis}$ by simp
qed
ultimately show $(\bigwedge P \in A \cdot P X) \land gvt(n+1)) = (\bigwedge P \in A \cdot (P (X \land gvt(n)) \land gvt(n+1))$
  by simp
A tail recursive reactive design with a productive body is guarded.

**Lemma** Guarded-if-Productive

- **assumes** Guarded P Guarded Q
- **shows** Guarded (λ X. P(X) ∩ Q(X))

**proof**

- **have** Guarded (λ X. ∩ F∈{P,Q} · F(X))
  - by (rule UINF-Guarded, auto simp add: assms)
- **thus** ?thesis
  - by (simp)

**Lemma** cond-srea-Guarded

- **assumes** Guarded P Guarded Q
- **shows** Guarded (λ X. P(X) ∖ b ⊆R Q(X))
  - using assms by (rel-auto)

---

We split the proof into three cases corresponding to valuations for ok, wait, and wait' respectively.

**fix** X

**have** a:(P :: SRD(X) ∧ gvrtn (Suc n))(false/ok) = 
  (P :: SRD(X ∧ gvrtn (Suc n))(false/ok)
  by (simp add: subst closure SRD-left-zero-1 assms)

**have** b:(P :: SRD(X) ∧ gvrtn (Suc n))(true/ok)[true/wait] = 
  ((P :: SRD(X ∧ gvrtn (Suc n))(true/ok)[true/wait]
  by (simp add: subst closure SRD-left-zero-2 assms)

**have** c:(P :: SRD(X) ∧ gvrtn (Suc n))(true/ok)[false/wait] = 
  ((P :: SRD(X ∧ gvrtn n) ∧ gvrtn (Suc n))(true/ok)[false/wait]
  by (meson (no_types, lifting) Healthy-def R3h-wait-true SRD-health(∃) SRD-iden)

**proof**

- **have** 1:(P[true/wait] :: (SRD X)[true/wait] ∧ gvrtn (Suc n))(true/false/ok,wait) = 
  (P[true/wait] :: (SRD (X ∧ gvrtn n))[true/false/ok,wait] ∧ gvrtn (Suc n))(true/false/ok,wait)
  by (metis (no_types, lifting) Healthy-def R3h-wait-true SRD-health(∃) SRD-iden)

- **have** 2:(P[false/wait] :: (SRD X)[false/wait] ∧ gvrtn (Suc n))(true/false/ok,wait) = 
  (P[false/wait] :: (SRD (X ∧ gvrtn n))[false/wait] ∧ gvrtn (Suc n))(true/false/ok,wait)

**proof**

- **have** exp:∀ Y::(′s, ′t,′α) hrel-rsp. (P[false/wait] :: (SRD Y)[false/wait] ∧ gvrtn (Suc n))(true/false/ok,wait) = 
  (((¬r ′pre R P :: (SRD Y))[false/wait] ∨ (post R P ∧ $tr >_u $tr’ :: (SRD Y)[false/wait] ∧ gvrtn (Suc n))(true/false/ok,wait)
  by (meson (no_types) Health-def Productive-form assms(1) assms(2) NSRD-is-SRD)

- **have** ...
  - ([(R1(R2x(pre R P ⇒ ($ok ∧ post R P ∧ $tr <_u $tr’))))[false/wait] :: (SRD Y)[false/wait] ∧ gvrtn (Suc n))(true/false/ok,wait)]

---

87
by (simp add: RHS-def R1-def R2c-def R2s-def R3h-def RD1-def RD2-def usubst unrest assms closure design-def)
also have ... =
  (((∀, pre(P)) ∨ (ok ∧ post(P) ∧ str < u $str′)))[false/1wait] ∨ (SRD Y)[false/1wait]
∧ gvrtr ((Succ n))[true/false/$ok/$wait]
  by (simp add: impl-alt-def R2c-disj R1-disj R2c-not assms closure R2c-and
R2c-preR rea-not-def R1-extend-conv R2c-ok R2c-post-SRD R1-tr-less-tr′ R2c-tr-less-tr′)
also have ... =
  ((((∀, pre(P)) :: (SRD Y))[false/1wait] ∨ (ok ∧ post(P) ∧ str′ > u $str)) :: (SRD Y))[false/1wait]
∧ gvrtr (Succ n))[true/false/$ok/$wait]
  by (simp add: usubst unrest assms closure seq-or-distl NSRD-neg-pre-left-zero SRD-healths)
also have ... =
  (((∀, pre(P)) :: (SRD Y))[false/1wait] ∨ (post(P) ∧ str′ > u $str)) :: (SRD Y)[true/false/$ok/$wait]
∧ gvrtr (Succ n))[true/false/$ok/$wait]
proof –
  have (ok ∧ post(P) ∧ str′ > u $str) :: (SRD Y)[false/1wait] =
    (((post(P) ∧ str′ > u $tr) ∧ ok′ = u true) :: (SRD Y))[false/1wait]
  by (rel-blast)
also have ... = (post(P) ∧ str′ > u $str)[true/$ok′] :: (SRD Y)[false/1wait][true/$ok]
using seqr-left-one-point[of ok (post(P) ∧ str′ > u $tr) True (SRD Y)[false/1wait]]
  by (simp add: true-alt-def THEN sym)
finally show ?thesis by (simp add: usubst unrest)
qed

finally show (P)[false/1wait] :: (SRD Y)[false/1wait] ∧ gvrtr (Succ n))[true/false/$ok/$wait]
  = (((∀, pre(P)) :: (SRD Y))[false/1wait] ∨ (post(P) ∧ str′ > u $str)) :: (SRD Y)[true/false/$ok/$wait]
∧ gvrtr (Succ n))[true/false/$ok/$wait]
.

qed

have 1:(post(P) ∧ str′ > u $str) :: (SRD X)[true/false/$ok/$wait] ∧ gvrtr (Succ n)) =
  ((post(P) ∧ str′ > u $tr)) :: (SRD (X ∧ gvrtr n))[true/false/$ok/$wait] ∧ gvrtr (Succ n))
apply (rel-auto)
  apply (rename-tac tr st more ok wait tr′ st′ more′ tr0 st0 more0 ok′)
  apply (rule-tac x=tr0 in exI, rule-tac x=st0 in exI, rule-tac x=more0 in exI)
  apply (simp)
  apply (erule trace-strict-prefixE)
  apply (rename-tac tr st ref ok wait tr′ st′ ref′ tr0 st0 ref0 ok′ zs)
  apply (rule-tac x=False in exI)
  apply (simp add: size-minus-trace)
  apply (subgoal-tac size(tr) < size(tr0))
  apply (simp add: less-diff-conv2 size-mono)
using size-strict-mono apply blast
apply (rename-tac tr st more ok wait tr′ st′ more′ tr0 st0 more0 ok′)
apply (rule-tac x=tr0 in exI, rule-tac x=st0 in exI, rule-tac x=more0 in exI)
apply (simp)
apply (erule trace-strict-prefixE)
apply (rename-tac tr st more ok wait tr′ st′ more′ tr0 st0 more0 ok′ zs)
apply (auto simp add: size-minus-trace)
apply (subgoal-tac size(tr) < size(tr0))
apply (simp add: less-diff-conv2 size-mono)
using size-strict-mono apply blast
done
have 2:(∀, pre(P)) :: (SRD X)[false/1wait] = (∀, pre(P)) :: (SRD(X ∧ gvrtr n))[false/1wait]
  by (simp add: NSRD-neg-pre-left-zero closure assms SRD-healths)

88
show ?thesis
  by (simp add: exp 1 2 utp-pred-laws.inf-sup-distrib2)
qed

show ?thesis
proof –
  have (P ;; (SRD X) ∨ gvr (Suc n))(true/false/$\$ok,$\$wait]$ =
    (P[true/$\$wait$] ;; (SRD X)[true/$\$wait$] ∨ gvr (Suc n))(true/false/$\$ok,$\$wait$] ∨
    (P[false/$\$wait$] ;; (SRD X)[false/$\$wait$] ∨ gvr (Suc n))(true/false/$\$ok,$\$wait$])
  by (subst seqr-bool-split[of wait], simp-all add: usubst utp-pred-laws.distrib(4))
also
  have ... = ((P[true/$\$wait$] ;; (SRD X)(true/$\$wait$] ∨ gvr (Suc n))(true/false/$\$ok,$\$wait$]
    ∨
    (P[false/$\$wait$] ;; (SRD X)(false/$\$wait$] ∨ gvr (Suc n))(true/false/$\$ok,$\$wait$])
  by (simp add: usubst: simp add: usubst utp-pred-laws.distrib(4))
also
  have ... = (P ;; (SRD X) ∨ gvr (Suc n))(true/false/$\$ok,$\$wait$]
  by (subst seqr-bool-split[of wait], simp-all add: usubst)
finally show ?thesis by (simp add: usubst)
qed

10.3 Tail recursive fixed-point calculations

declare upred-semiring.power-Suc [simp]

lemma mu-csp-form-1 [rdes]:
  fixes P :: ('s, 't::size-trace, 'a) hrel-rsp
  assumes P is NSRD P is Productive
  shows (μ X. P ;; SRD(X)) = (μ i. P * (i+1)) ;; Miracle
proof –
  have 1:Continuous (λX. P ;; SRD X)
    using SRD-Continuous
    by (clarsimp simp add: Continuous-def seq-SUP-distl[THEN sym], drule-tac x=A in spec, simp)
  have 2: (λX. P ;; SRD X) ∈ {id}H → [SRD]H
    by (blast intro: funcsetI closure assms)
  with 1 2 have (μ X. P ;; SRD(X)) = (μ X. P ;; SRD(X))
    by (simp add: guarded-fp-uniq Guared-if-Productive[of assms] funcsetI closure)
  also have ... = (μ i. ((λX. P ;; SRD X) * i) false)
    by (simp add: sup-continuous-lfp 1 sup-continuous-Continuous false-upred-def)
  also have ... = ((λX. P ;; SRD X) * 0) false ∩ (μ i. ((λX. P ;; SRD X) * i) false)
    by (subst Sup-power-expand, simp)
also have ... = (\prod i. ((\lambda X. P :: SRD X) ^^ (i+1)) false)
   by (simp)
also have ... = (\prod i. P ^ (i+1)) ;; Miracle
proof (rule SUP-cong, simp-all)
  fix i
  show P ;; SRD (((\lambda X. P :: SRD X) ^^ i) false) = (P ;; P ^ i) ;; Miracle
  proof (induct i)
    case 0
    then show ?case
      by (simp, metis srdes-hcond-def srdes-theory-continuous.healthy-top)
  next
    case (Suc i)
    then show ?case
      by (simp add: Healthy-if NSRD-is-SRD SRD-power-comp SRD-seqr-closure assms)
  qed
qed

fix i
show P ;; SRD ((\lambda X. P ;; SRD X) ^^ i) = (P ;; P ^ i) ;; 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) hrel-rsp
  assumes P is NSRD P is Productive
  shows (\mu X. P ;; SRD(X)) is NSRD
  by (simp add: mu-csp-form-1 assms closure ustar-def)

lemma mu-csp-form-1':
  fixes P :: ('s, 't::size-trace, 'a) hrel-rsp
  assumes P is NSRD P is Productive
  shows (\mu X. P ;; SRD(X)) = (P :: P^*) ;; Miracle
proof
  have (\mu X. P ;; SRD(X)) = (\prod i\in UNIV. P ;; P ^ i) ;; Miracle
    by (simp add: mu-csp-form-1 assms closure ustar-def)
  also have ... = (P :: P^*) ;; Miracle
    by (simp only: seq-UINF-distl[THEN sym], simp add: ustar-def)
  finally show ?thesis .
qed

declare upred-semiring.power-Suc [simp del]

end

11 Reactive Design Programs

theory utp-rdes-prog
import utp-rdes-normal utp-rdes-tactics utp-rdes-parallel utp-rdes-guarded UTP–KAT.utp-kleene
begin
11.1 State substitution

**lemma** srd-subst-RHS-tri-design [usubst]:
\[ [\sigma]_{S^R} \uparrow R_\sigma (P \vdash Q \circ R) = R_\sigma (([\sigma]_{S^R} \uparrow P) \vdash ([\sigma]_{S^R} \uparrow Q) \circ ([\sigma]_{S^R} \uparrow R)) \]
by (rel-auto)

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

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

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

**lemma** postR-srd-subst [rdes]:
\[ \text{post}_R ([\sigma]_{S^R} \uparrow P) = [\sigma]_{S^R} \uparrow \text{post}_R (P) \]
by (rel-auto)

**lemma** srd-subst-NSRD-closed [closure]:
assumes \( P \) is NSRD
shows \([\sigma]_{S^R} \uparrow P \) is NSRD
by (rule NSRD-RC-intro, simp-all add: closure rdes assms unrest)

11.2 Assignment

definition assigns-srd :: 's usubst ⇒ ('s, 't::trace, 'a) hrel-rsp ((·)R) where
[upred-defs]: assigns-srd \( \sigma \) = \( R_{\sigma} (\text{true} \vdash (\$tr' = u \$tr ∧ \neg \$wait' \land (\langle \sigma \rangle_S \land \$\Sigma_S' = u \$\Sigma_S)) \]

syntax
- assigns-srd :: svids ⇒ uexprs ⇒ logic (infixr :=R 90)
- assigns-srd :: svids ⇒ uexprs ⇒ logic (infixl :=R 90)

translations
- assigns-srd \( \var_1 \var_2 \vars \vars' \Rightarrow \text{CONST assigns-srd} (\text{-mk-usubst} (\text{CONST id}) \var_1 \vars \vars') \)
- assigns-srd \( \var x \vars' \Leftarrow \text{CONST assigns-srd} (\text{CONST subst-upd} (\text{CONST id}) \var \vars) \)
- assigns-srd \( \var x \vars' \Leftarrow \text{-assign-srd} (\text{-spvar} \var) \var \vars' \)

\( x, y := \var_1 \var_2 \vars \vars' \Leftarrow \text{CONST assigns-srd} (\text{CONST subst-upd} (\text{CONST subst-upd} (\text{CONST id}) (\text{CONST svar} \var x) (\text{CONST svar} \var y)) \var) \)

**lemma** assigns-srd-RHS-tri-des [rdes-def]:
\( \langle \sigma \rangle_R = R_{\sigma} (\text{true}_R \vdash \false \circ \langle \sigma \rangle) \)
by (rel-auto)

**lemma** assigns-srd-NSRD-closed [closure]: \( \langle \sigma \rangle_R \) is NSRD
by (simp add: rdes-def closure unrest)
lemma \( \text{preR-assigns-srd [rdes]} \): \( \text{pre}_R(\langle \sigma \rangle_R) = \text{true} \)
by (simp add: rdes-def rdes closure)

lemma \( \text{periR-assigns-srd [rdes]} \): \( \text{peri}_R(\langle \sigma \rangle_R) = \text{false} \)
by (simp add: rdes-def rdes closure)

lemma \( \text{postR-assigns-srd [rdes]} \): \( \text{post}_R(\langle \sigma \rangle_R) = \langle \sigma \rangle_R \)
by (simp add: rdes-def rdes closure rpred)

11.3 Conditional

lemma \( \text{preR-cond-srea [rdes]} \):
\[
\text{pre}_R(P \triangleleft \! \triangleright R Q) = ([b]_S < \wedge \text{pre}_R(P) \lor [\neg b]_S < \wedge \text{pre}_R(Q))
\]
by (rel-auto)

lemma \( \text{periR-cond-srea [rdes]} \):
assumes \( P \text{ is SRD} \) \( Q \text{ is SRD} \)
shows \( \text{peri}_R(P \triangleleft \! \triangleright R Q) = ([b]_S < \wedge \text{peri}_R(P) \lor [\neg b]_S < \wedge \text{peri}_R(Q)) \)
proof -
have \( \text{peri}_R(P \triangleleft \! \triangleright R Q) = \text{peri}_R(R1(P) \triangleleft \! \triangleright R1(Q)) \)
by (simp add: Healthy-if SRD-healths assms)
thus ?thesis
by (rel-auto)
qed

lemma \( \text{postR-cond-srea [rdes]} \):
assumes \( P \text{ is SRD} \) \( Q \text{ is SRD} \)
shows \( \text{post}_R(P \triangleleft \! \triangleright R Q) = ([b]_S < \wedge \text{post}_R(P) \lor [\neg b]_S < \wedge \text{post}_R(Q)) \)
proof -
have \( \text{post}_R(P \triangleleft \! \triangleright R Q) = \text{post}_R(R1(P) \triangleleft \! \triangleright R1(Q)) \)
by (simp add: Healthy-if SRD-healths assms)
thus ?thesis
by (rel-auto)
qed

lemma \( \text{NSRD-cond-srea [closure]} \):
assumes \( P \text{ is NSRD} \) \( Q \text{ is NSRD} \)
shows \( P \triangleleft \! \triangleright R Q \text{ is NSRD} \)
proof (rule NSRD-RC-intro)
show \( P \triangleleft \! \triangleright R Q \text{ is SRD} \)
by (simp add: closure assms)
show \( \text{pre}_R(P \triangleleft \! \triangleright R Q) \text{ is RC} \)
proof -
have \( 1:([\neg b]_S < \wedge \neg \text{pre}_R P) ;; R1(\text{true}) = ([\neg b]_S < \wedge \neg \text{pre}_R P) \)
by (metis (no-types, lifting) NSRD-neg-pre-unit assms ext-not assms seqr-or-distl st-lift-R1-true-right)
have \( 2:([b]_S < \wedge \neg \text{pre}_R Q) ;; R1(\text{true}) = ([b]_S < \wedge \neg \text{pre}_R Q) \)
by (simp add: NSRD-neg-pre-unit assms seqr-or-distl st-lift-R1-true-right)
show ?thesis
by (simp add: rdes assms unrest)
qed

case 
have \( \text{st'} \notin \text{peri}_R(P \triangleleft \! \triangleright R Q) \)
by (simp add: rdes assms closure unrest)
qed
11.4 Assumptions

**definition** AssumeR :: 's cond ⇒ ('s::trace, 't::trace) hrel-rsp ([|]-T_R) where
[upred-defs]: AssumeR b = II_R ∘ b ▷ R Miracle

**lemma** AssumeR-rdes-def [rdes-def]:
[b]^T_R = R_s(true, ⊢ false ∘ [b]^T_s)

**unfolding** AssumeR-def by (rdes-eq)

**lemma** AssumeR-NSRD [closure]: [b]^T_R is NSRD
by (simp add: AssumeR-def closure)

**lemma** AssumeR-false: [false]^T_R = Miracle
by (rel-auto)

**lemma** AssumeR-true: [true]^T_R = II_R
by (rel-auto)

**lemma** AssumeR-comp: [b]^T_R ;; [c]^T_R = [b ∧ c]^T_R
by (rdes-simp)

**lemma** AssumeR-choice: [b]^T_R ∩ [c]^T_R = [b ∨ c]^T_R
by (rdes-eq)

**lemma** AssumeR-refine-skip: II_R ⊑ [b]^T_R
by (rdes-refine)

**lemma** AssumeR-test [closure]: test_R [b]^T_R
by (simp add: AssumeR-refine-skip nsrd-thy.utest-intro)

**lemma** Star-AssumeR: [b]^T_R⋆R = II_R
by (simp add: AssumeR-NSRD AssumeR-test nsrd-thy.Star-test)

**lemma** AssumeR-choice-skip: II_R ∩ [b]^T_R = II_R
by (rdes-eq)

**lemma** cond-srea-AssumeR-form:
assumes P is NSRD Q is NSRD
shows P ∘ b ▷ R Q = ([b]^T_R ;; P ∩ ¬[b]^T_R ;; Q)
by (rdes-eq cls: assms)

**lemma** cond-srea-insert-assume:
assumes P is NSRD Q is NSRD
shows P ∘ b ▷ R Q = ([b]^T_R ;; P ∘ b ▷ R ¬[b]^T_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]^T_R ;; (P ∘ b ▷ R Q) = ([b]^T_R ;; P)
by (rdes-eq cls: assms)

**lemma** AssumeR-cond-right:
assumes P is NSRD Q is NSRD
shows [¬b]^T_R ;; (P ∘ b ▷ R Q) = ([¬b]^T_R ;; Q)
by (rdes-eq cls: assms)
11.5 Guarded commands

**definition** GuardedCommR :: \( 's \text{ cond} \Rightarrow ('s, 't::trace, 'a) hrel-rsp \Rightarrow ('s, 't, 'a) hrel-rsp (\rightarrow_R - [85, 86]) \) where

gcmd-def[rdes-def]: GuardedCommR g A = A \triangleleft g \triangleright_R Miracle

**lemma** gcmd-false[simp]: \((\text{false} \rightarrow_R A) = \text{Miracle} \)

**unfolding** gcmd-def by (pred-auto)

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

**unfolding** gcmd-def by (pred-auto)

**lemma** gcmd-SRD:
assumes \( A \) is SRD
shows \((g \rightarrow_R A) \) is SRD
by (simp add: gcmd-def SRD-cond-srea assms srdes-theory-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 \((g \rightarrow_R (A \cap B)) = (g \rightarrow_R A) \cap (g \rightarrow_R B) \)
by (rdes-eq cls: assms)

**lemma** AssumeR-as-gcmd:
\([b]^+_R = b \rightarrow_R II_R \)
by (rdes-eq)

11.6 Generalised Alternation

**definition** AlternateR :: \( 'a \) set \Rightarrow ('a \Rightarrow 's upred) \Rightarrow ('a \Rightarrow ('s, 't::trace, 'a) hrel-rsp) \Rightarrow ('a, 't, 'a) hrel-rsp \Rightarrow ('a, 't, 'a) hrel-rsp where

[upred-defs, rdes-def]: AlternateR I g A B = (\bigcap i \in I \cdot ((g i) \rightarrow_R (A i))) \cap ((\neg (\forall 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 where

[upred-defs, ndes-simp]:
AlternateR-list xs P = AlternateR \{0..<\text{length} xs\} (\lambda i. \text{map} \ f s t \ x s ! i) (\lambda i. \text{map} \ \text{snd} \ x s ! i) P

**syntax**
- altindR-els :: pttrn \Rightarrow logic \Rightarrow logic \Rightarrow logic \Rightarrow logic \Rightarrow logic (if \( R - \cdot - \rightarrow - \text{else} - f)
-altindR :: ptryn ⇒ logic ⇒ logic ⇒ logic ⇒ logic (if R ∈ · · · → · fi)

-altgcommR-els :: gcomms ⇒ logic ⇒ logic ⇒ logic ⇒ logic (if R/ - else - /fi)
-altgcommR :: gcomms ⇒ logic (if R/ - /fi)

translations
if R i∈ I · g → A else B fi → CONST AlternateR I (λi. g) (λi. A) B
if R i∈ I · g → A fi → CONST AlternateR I (λi. g) (λi. A) (CONST Chaos)
if R i∈ I · (g i) → A else B fi ← CONST AlternateR I g (λi. A) B
-altgcommR cs ← CONST AlternateR-list cs (CONST Chaos)
-altgcommR (-gcomm-show cs) ← CONST AlternateR-list cs (CONST Chaos)
-altgcommR-els cs P ← CONST AlternateR-list cs P
-altgcommR-els (-gcomm-show cs) P ← CONST AlternateR-list cs P

lemma AlternateR-NSRD-closed [closure]:
assumes \( \bigwedge i. i \in I \Rightarrow A i \text{ is NSRD} B \text{ is NSRD} \)
shows (if R i∈ I · g i → A i else B fi) is NSRD
proof (cases I = {})
case True
then show \?thesis by (simp add: AlternateR-def assms)
next
case False
then show \?thesis by (simp add: AlternateR-def closure assms)
qed

lemma AlternateR-empty [simp]:
(if R i∈ I · g i → A i else B fi) = B
by (rdes-simp)

lemma AlternateR-Productive [closure]:
assumes \( \bigwedge i. i \in I \Rightarrow A i \text{ is NSRD} B \text{ is NSRD} \)
\( \bigwedge i. i \in I \Rightarrow A i \text{ is Productive} B \text{ is Productive} \)
shows (if R i∈ I · g i → A i else B fi) 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 is NSRD B is NSRD
shows (if R i∈ {k} · g i → A i else B fi) = (A(k) ⊆ g(k) ⊆ 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 is NSRD B is NSRD
\((g(k) \land (\lor i\in I \cdot g(i))) = \) false
shows \((if \ R_i \in I \cdot g \ i \rightarrow A \ i \ else \ B \ fi) = (A(k) \odot g(k) \triangleright_R (if \ R_i \in I \cdot g(i) \rightarrow A(i) \ else \ B \ fi))\)

proof (cases \(I = \{\}\))

   case True
   
   then show ?thesis by (simp add: AlternateR-singleton assms)

next

   case False
   
   have 1: \((\bigsqcap \ i \in I \cdot g \ i \rightarrow R \ A \ i) = (\bigsqcap \ i \in I \cdot g \ i \rightarrow R \ R_\text{pre}_R(A \ i) \triangleright peri_R(A \ i) \odot post_R(A \ i))\)
   
   by (simp add: NSRD-is-SRD SRD-reactive-tri-design assms 1 cong: UINF-cong)

from assms (1) show ?thesis by (simp add: AlternateR-def 1 False cong: UINF-cong)

qed

11.7 Choose

definition choose-srd :: \(\langle s', t::trace, \alpha \rangle \ hrel-rsp \ (\text{choose}_R)\) where
   [upred-defs, rdes-def]: \text{choose}_R = R_\text{true}_r \triangleright false \odot true_r

lemma preR-choose [rdes]: \(\text{pre}_R(\text{choose}_R) = true_r\)
   by (rel-auto)

lemma periR-choose [rdes]: \(\text{peri}_R(\text{choose}_R) = false\)
   by (rel-auto)

lemma postR-choose [rdes]: \(\text{post}_R(\text{choose}_R) = true_r\)
   by (rel-auto)

lemma choose-srd-SRD [closure]: \text{choose}_R is SRD
   by (simp add: choose-srd-def closure unrest)

lemma NSRD-choose-srd [closure]: \text{choose}_R is NSRD
   by (rule NSRD-intro, simp-all add: closure unrest rdes)

11.8 State Abstraction

definition state-srea :: \(\langle s itself \Rightarrow \langle s', t::trace, \alpha, \beta \rangle \ hrel-rsp \Rightarrow \langle \text{unit}, t', \alpha, \beta \rangle \ hrel-rsp \rangle\) where
   [upred-defs]: state-srea t P = \(\exists \ (\$s, \$st') \cdot P\)

syntax
   -state-srea :: type \Rightarrow \text{logic} \Rightarrow \text{logic} (state - - [0,200] 200)

translations
   state 'a \cdot P == \text{CONST state-srea TYPE('a) P}

lemma R1-state-srea: R1(state 'a \cdot P) = (state 'a \cdot R1(P))
   by (rel-auto)

lemma R2c-state-srea: R2c(state 'a \cdot P) = (state 'a \cdot R2c(P))
   by (rel-auto)

lemma R3h-state-srea: R3h(state 'a \cdot P) = (state 'a \cdot R3h(P))
   by (rel-auto)

lemma RD1-state-srea: RD1(state 'a \cdot P) = (state 'a \cdot RD1(P))
   by (rel-auto)

96
lemma \( RD2\)-state-srea: \( RD2(\text{state } 'a \cdot P) = (\text{state } 'a \cdot RD2(P)) \)

by (rel-auto)

lemma \( RD3\)-state-srea: \( RD3(\text{state } 'a \cdot P) = (\text{state } 'a \cdot RD3(P)) \)

by (rel-auto, blast+)

lemma \( SRD\)-state-srea [closure]: \( \text{P is SRD} \implies \text{state } 'a \cdot P \text{ is SRD} \)

by (simp add: Healthy-def R1-state-srea R2c-state-srea R3h-state-srea RD1-state-srea RD2-state-srea RHS-def SRD-def)

lemma \( NSRD\)-state-srea [closure]: \( \text{P is NSRD} \implies \text{state } 'a \cdot P \text{ is NSRD} \)

by (metis Healthy-def NSRD-is-RD3 NSRD-is-SRD RD3-state-srea SRD-RD3-implies-NSRD SRD-state-srea)

lemma \( peri\)-state-srea [rdes]: \( peri_R(\text{state } 'a \cdot P) = \text{state } 'a \cdot peri_R(P) \)

by (rel-auto)

lemma \( post\)-state-srea [rdes]: \( post_R(\text{state } 'a \cdot P) = \text{state } 'a \cdot post_R(P) \)

by (rel-auto)

11.9 While Loop

definition \( \text{WhileR}: \text{ 's upred } \Rightarrow (\text{ 's, } \text{ 't::size-trace, } \alpha) \text{ hrel-rsp } \Rightarrow (\text{ 's, } \text{ 't, } \alpha) \text{ hrel-rsp (whileR - do - od) where} \)

\( \text{WhileR } b \ P = (\mu_R X \cdot (P ;; X) \triangleleft b \triangleright_R II_R) \)

lemma \( \text{Sup-power-false}: \)

fixes \( F :: \alpha \text{ upred } \Rightarrow \alpha \text{ upred} \)

shows \( (\prod i. (F \cdot i) \text{ false}) = (\prod i. (F \cdot (i+1)) \text{ false}) \)

proof -

have \( (\prod i. (F \cdot i) \text{ false}) = (F \cdot 0) \text{ false } \cap (\prod i. (F \cdot (i+1)) \text{ false}) \)

by (subst Sup-power-expand, simp)

also have \( \ldots = (\prod i. (F \cdot (i+1)) \text{ false}) \)

by (simp)

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

qed

theorem \( \text{WhileR-iter-expand}: \)

assumes \( P \text{ is NSRD} \ P \text{ is Productive} \)

shows \( \text{whileR } b \ P \text{ od } = (\prod i. (P \triangleleft b \triangleright_R II_R) ^ i ;; (P ;; \text{ Miracle } \triangleleft b \triangleright_R II_R)) \) (is \( \text{?lhs } = \text{?rhs} \))

proof -

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

using SRD-Continuous

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

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

by (simp add: 1 closure assms)

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

by (simp add: WhileR-def)

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

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

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

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

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

97
by (simp add: sup-continuous-lfp 2 sup-continuous-Continuous false-upred-def)
also have ... = (Π i. (λX. P :: SRD X ⊲ b ⪰R II_R) "" (i+1)) false
  by (simp add: Sup-power-false)
also have ... = (Π i. (P ⊲ b ⪰R II_R) "" i :: (P :: Miracle ⊲ b ⪰R II_R))
proof (rule SUP-cong, simp)
  fix i
  show ((λX. P :: SRD X ⊲ b ⪰R II_R) "" (Suc i)) false = (P ⊲ b ⪰R II_R) "" i :: (P :: Miracle ⊲ b ⪰R II_R)
  proof (induct i)
    case 0
    then show ?case
      by (simp, metis srdes-hcond-def srdes-theory-continuous.healthy-top)
  next
    case (Suc i)
    show ?case
    proof
      have ((λX. P :: SRD X ⊲ b ⪰R II_R) "" (Suc i)) false = (P ⊲ b ⪰R II_R) "" (Suc i)
        by (simp add: Suc.hyps)
      using Suc.hyps by auto
      also have ... = P :: SRD ((P ⊲ b ⪰R II_R) "" i :: (P :: Miracle ⊲ b ⪰R II_R)) ⊲ b ⪰R II_R
        by (metis (no-types, lifting) Healthy-if NSRD-cond-srea NSRD-is-SRD NSRD-power-Suc NSRD-srd-skip SRD-cond-srea SRD-seqr-closure assms(1) power-power-eq-if seqr-left-unit srdes-theory-continuous.top-closed)
      also have ... = (P ⊲ b ⪰R II_R) "" Suc i :: (P :: Miracle ⊲ b ⪰R II_R)
      proof (induct i)
        case 0
        then show ?case
      next
        case (Suc i)
        have 1: II_R :: ((P ⊲ b ⪰R II_R) :: (P ⊲ b ⪰R II_R) "" i) = ((P ⊲ b ⪰R II_R) :: (P ⊲ b ⪰R II_R) "" i)
          by (simp add: NSRD-is-SRD RA1 SRD-cond-srea SRD-left-unit SRD-srd-skip assms(1))
        then show ?case
          proof
            have \ u :: u :: (P ⊲ b ⪰R II_R) "" Suc i :: (P :: Miracle) ⊲ b ⪰R (II_R) = ((u ⊲ b ⪰R II_R) :: (P ⊲ b ⪰R II_R) "" Suc i :: (P :: Miracle) ⊲ b ⪰R (II_R))
              by (metis (no-types) Suc.hyps 1 cond-L6 cond-st-distr power-power-power-Suc)
            then show ?thesis
              by (simp add: RA1 upred-semiring.power-Suc)
          qed
          qed
          finally show ?thesis .
          qed
          qed
          qed
also have ... = (Π i :: (P ⊲ b ⪰R II_R) "" i :: (P :: Miracle ⊲ b ⪰R II_R))
    by (simp add: UNF-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 = \((P \triangleright R \text{ II}_R)^* \); (P ;; Miracle \( \triangleleft R \text{ II}_R \)) (is \( \text{lhs} = \text{rhs} \))

proof –

have \( \text{lhs} = ((\prod i \cdot (P \triangleright R \text{ II}_R)^* i ) ;; (P ;; \text{Miracle} \triangleright R \text{ II}_R) \)
by (simp add: WhileR-iter-expand seq-UNF-distr' assms)

also have \( \ldots = (P \triangleright R \text{ II}_R)^* ;; (P ;; \text{Miracle} \triangleright R \text{ II}_R) \)
by (simp add: ustar-def)

also have \( \ldots = ((P \triangleright R \text{ II}_R)^* ;; \text{II}_R) ;; (P ;; \text{Miracle} \triangleright R \text{ II}_R) \)
by (simp add: seq-assoc SRD-left-unit closure assms)

also have \( \ldots = (P \triangleright R \text{ II}_R)^* R ;; (P ;; \text{Miracle} \triangleright R \text{ II}_R) \)
by (simp add: nsrd-thy.\text{Star-def})

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

qed

lemma WhileR-NSRD-closed [\text{closure}]:

assumes \( P \) is NSRD \( P \) is Productive

shows while \( R \) b do P od is NSRD

by (simp add: WhileR-star-expand assms closure)

theorem WhileR-iter-form-lemma:

assumes \( P \) is NSRD

shows \((P \triangleleft R \text{ II}_R)^* ;; (P ;; \text{Miracle} \triangleleft R \text{ II}_R) = (\triangleright I_R ;; P)^* ;; \text{[\sim b]}^\top_R \)

proof –

have \((P \triangleright R \text{ II}_R)^* ;; (P ;; \text{Miracle} \triangleright R \text{ II}_R) = (\triangleright I_R ;; P) ;; \text{[\sim b]}^\top_R \)
by (simp add: AssumeR-NSRD NSRD-right-unit NSRD-srd-skp assms(1) cond-srea-AssumeR-form)

also have \( \ldots = ((\triangleright I_R ;; P)^* ;; \text{[\sim b]}^\top_R) ;; (P ;; \text{Miracle} \triangleright R \text{ II}_R) \)
by (simp add: AssumeR-NSRD NSRD-seq-closure nsrd-thy.\text{Star-distrib assms}(1))

also have \( \ldots = (((\triangleright I_R ;; P)^* ;; \text{[\sim b]}^\top_R) ;; (\triangleright I_R ;; P) ;; \text{Miracle} \triangleright R \text{ II}_R) \)
by (simp add: AssumeR-NSRD NSRD-seq-closure nsrd-thy.\text{Star-invass assms}(1))

also have \( \ldots = (((\triangleright I_R ;; P)^* ;; \text{[\sim b]}^\top_R) ;; (\triangleright I_R ;; P) ;; \text{Miracle} \triangleright R \text{ II}_R) \)
by (simp add: AssumeR-NSRD NSRD-Miracle NSRD-right-unit NSRD-seq-closure NSRD-srd-skp assms(1) cond-srea-AssumeR-form)

also have \( \ldots = (((\triangleright I_R ;; P)^* ;; \text{[\sim b]}^\top_R) ;; (\triangleright I_R ;; P) ;; \text{Miracle} \triangleright R \text{ II}_R) \)
by (simp add: upred-semiring.distrib-left)

also have \( \ldots = (((\triangleright I_R ;; P)^* ;; \text{[\sim b]}^\top_R) ;; (\triangleright I_R ;; P) ;; \text{Miracle} \triangleright R \text{ II}_R) \)
by (simp add: upred-semiring.distrib-left)

proof –

have \(((\triangleright I_R ;; P)^* ;; \text{[\sim b]}^\top_R) ;; (\triangleright I_R ;; P)^* ;; \text{[\sim b]}^\top_R) = (\text{II}_R ;; (\triangleright I_R ;; P)^* ;; \text{[\sim b]}^\top_R) ;; (\triangleright I_R ;; P) ;; \text{[\sim b]}^\top_R \)
by (simp add: AssumeR-NSRD NSRD-seq-closure nsrd-thy.\text{Star-unfoldr-eq assms}(1))

also have \( \ldots = (\text{[\sim b]}^\top_R) ;; (\triangleright I_R ;; P)^* ;; \text{[\sim b]}^\top_R) ;; (\triangleright I_R ;; P) ;; \text{[\sim b]}^\top_R \)
by (metis (no-types, lifting) AssumeR-NSRD AssumeR-as-gcdn NSRD-srd-skp Star-\text{AssumeR nsrd-thy.\text{Star-srea-seq-distr-skp-srea-self-\text{unit urel}DioD distrib-right})

also have \( \ldots = (\text{[\sim b]}^\top_R) ;; (\triangleright I_R ;; P)^* ;; \text{[\sim b]}^\top_R) ;; (\text{[\sim b]}^\top_R) ;; (\text{[\sim b]}^\top_R) \)
by (metis (no-types, hide-lams) AssumeR-\text{choice} upred-semiring.\text{add-assoc upred-semiring.distrib-left upred-semiring.distrib-right})

also have \( \ldots = (\text{[\sim b]}^\top_R) ;; (\triangleright I_R ;; P)^* ;; \text{[\sim b]}^\top_R) ;; (\triangleright I_R ;; P) ;; \text{[\sim b]}^\top_R \)
by (simp add: RA1)

also have \( \ldots = (\text{[\sim b]}^\top_R) ;; (\triangleright I_R ;; P)^* ;; \text{[\sim b]}^\top_R) ;; (\text{[\sim b]}^\top_R) \)
by (simp add: AssumeR-\text{comp} AssumeR-false)
finally have \((b)^{\uparrow} \vdash P^{*}; P^{*}; \neg b)^{\uparrow} R \subseteq ((b)^{\uparrow} \vdash P^{*}); b)^{\uparrow} \vdash P ; \text{ Miracle}
\)
by (simp add: semilattice-sup-class.le-supH1)
thus ?thesis
by (simp add: semilattice-sup-class.le-iff-sup)
qed 
finally show ?thesis .
qed

theorem WhileR-iter-form:
assumes \(P\) is NSRD \(P\) is Productive
shows while\_\_\_R b do P od = ((b)^{\uparrow} \vdash P^{*}; P)^{\uparrow} \vdash \neg b)^{\uparrow}
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 = II
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^{*}; ; \text{ Miracle}
by (simp add: WhileR-iter-form AssumeR-true AssumeR-false SRD-left-unit assms closure)

lemma WhileR-insert-assume:
assumes \(P\) is NSRD \(P\) is Productive
shows while\_\_\_R b do ((b)^{\uparrow} \vdash 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 \(\$st \not\in Q R\) is R4
shows while\_\_\_R b do \(R\_s(P \vdash Q \circ R)\) od =
\(R\_s ((b)^{\uparrow} \vdash R^{*}; wp_r ((b)^{\uparrow} \vdash R^{*}; Q \circ ((b)^{\uparrow} \vdash R^{*}; Q \circ ((b)^{\uparrow} \vdash R^{*}; \neg b)^{\uparrow})
(is \(?lhs = ?rhs)\)
proof –
have \(?lhs = ((b)^{\uparrow} \vdash \neg b)^{\uparrow} \vdash P^{*} ; \text{ Miracle}
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
\(Q_1\) is RC \(Q_2\) is RR \(Q_3\) is RR \(\$st \not\in Q_2 Q_3\) is R4
– Refinement conditions
\(((b)^{\uparrow} \vdash Q_3)^{*}; wp_r ((b)^{\uparrow} \vdash r, Q_1) \subseteq P_1
P_2 \subseteq (b)^{\uparrow} \vdash Q_2
P_2 \subseteq (b)^{\uparrow} \vdash Q_3 \vdash P_2
P_3 \subseteq (\neg b)^{\uparrow} \vdash
P_3 \subseteq (b)^{\uparrow} \vdash Q_3 \vdash P_3
shows R_s(P_1 \vdash P_2 \circ P_3) \subseteq while\_\_\_R b do R_s (Q_1 \vdash Q_2 \circ Q_3) od
proof (simp add: rdes-def assms, rule srdes-tri-refine-intro’

100
show \([b]_r^{+} \cdot Q_3^{r'} \cdot \wp_r \cdot ([b]_{<r} \Rightarrow_r Q_1) \subseteq P_1\)
by (simp add: assms)
show \(P_2 \subseteq (P_1 \land ([b]_r^{+} \cdot Q_3^{r'} \cdot [b]_r^{+} \cdot Q_2)\)
proof -
  have \(P_2 \subseteq ([b]_r^{+} \cdot Q_3^{r'} \cdot [b]_r^{+} \cdot Q_2)\)
  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-infl2)
qed

proof -
  have \(P_3 \subseteq (P_1 \land ([b]_r^{+} \cdot Q_3^{r'} \cdot [\neg b]_r^{+} \cdot Q_2)\)
  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-infl2)
qed

11.10 Iteration Construction

definition IterateR
  :: 'a set => ('a => 's upred) => ('a => ('s::size-trace, 'α) hrel-rsp) => ('s, 't, 'α) hrel-rsp
where IterateR A g P = whileR (V i\in A \cdot g(i)) do (if R i\in A \cdot g(i) \Rightarrow P(i) \Rightarrow R) od

definition IterateR-list
  :: ('s upred \times ('s::size-trace, 'α) hrel-rsp) list => ('s, 't, 'α) hrel-rsp
where IterateR-list xs = IterateR \{0..<length xs\} \{\lambda i. map fst xs ! i\} \{\lambda i. map snd xs ! i\}

syntax
  -iter-srd :: ptrn => logic => logic => logic => logic (do R -\cdot -\cdot - \Rightarrow - R)
  -iter-gcommR :: gcomm => logic (do R/ - od)

translations
  -iter-srd x A g P => CONST IterateR A (\lambda x. g) (\lambda x. P)
  -iter-srd x A g P <= CONST IterateR A (\lambda x. g) (\lambda x'. P)
  -iter-gcommR cs => CONST IterateR-list cs
  -iter-gcommR (-gcomm-show cs) <= CONST IterateR-list cs

lemma IterateR-NSRD-closed [closure]:
  assumes
  \(\forall i. i \in I \Rightarrow P(i)\) is NSRD
  \(\forall i. i \in I \Rightarrow P(i)\) is Productive
  shows doR i\in I \cdot g(i) \Rightarrow P(i)\ fi is NSRD
by (simp add: IterateR-def closure assms)

lemma IterateR-empty:
  doR i\in\{\} \cdot g(i) \Rightarrow P(i)\ fi = II_R
by (simp add: IterateR-def srd-mu-equiv closure rpred gfp-const WhileR-false)

lemma IterateR-singleton:
  assumes P k is NSRD P k is Productive
  shows doR i\in\{k\} \cdot g(i) \Rightarrow P(i)\ fi = whileR g(k) do P(k) od (is ?lhs = ?rhs)
proof -
  have ?lhs = whileR g k do P k < g k \Rightarrow R Chaos od
  by (simp add: IterateR-def AlternateR-singleton assms closure)

101
also have \( ... = \text{while}_{R} g k \text{ do } [g k]^{\top}_{R} :: (P k < g k \triangleright_{R} \text{ Chaos}) \text{ od} \)
by (simp add: WhileR-insert-assume closure assms)

also have \( ... = \text{while}_{R} g k \text{ do } P k \text{ od} \)
by (simp add: AssumeR-cond-left NSRD-Chaos WhileR-insert-assume assms)

finally show \(?\text{thesis} \).
qed
declare IterateR-list-def [rdes-def]
declare IterateR-def [rdes-def]

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

11.11 Substitution Laws

lemma srd-subst-Chaos [usubst]:
\( \sigma \uparrow_{S} \text{ Chaos} = \text{ Chaos} \)
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)

11.12 Algebraic Laws

theorem assigns-srd-id: \( \langle id \rangle_{R} = \text{ II}_{R} \)
by (rdes-eq)

theorem assigns-srd-comp: \( \langle \sigma \rangle_{R} :: \langle \sigma \rangle_{R} = \langle \sigma \circ \sigma \rangle_{R} \)
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 : \( \langle \sigma \rangle_{R} < b \triangleright_{R} \langle \sigma \rangle_{R} = \langle \sigma < b \triangleright_{S} \sigma \rangle_{R} \)
by (rdes-eq)

theorem assigns-srd-left-seq:
assumes \( P \text{ is NSRD} \)
shows \( \langle \sigma \rangle_{R} :: P = \sigma \uparrow_{S} P \)
by (rdes-simp cls: assms)

lemma AlternateR-seq-distr:
assumes \( \forall i. \text{ A i is NSRD} \text{ B is NSRD} \text{ C is NSRD} \)
shows \( (\text{if}_{R} i \in I \cdot g i \rightarrow A i \text{ else B fi}) :: \langle C \rangle_{R} = (\text{if}_{R} i \in I \cdot g i \rightarrow A i :: C \text{ else B :: C fi}) \)

102
proof (cases I = { })
  case True
  then show ?thesis by (simp)
next
  case False
  then show ?thesis
    by (simp add: AlternateR-def upred-semiring.distrib-right seq-UINF-distr gcmd-seq-distr assms (3))
qed

lemma AlternateR-is-cond-srea:
  assumes A is NSRD B is NSRD
  shows (if $R \in \{ a \cdot g \rightarrow A \} \cdot f_i$ then B else A fi) = (A \bowtie g \bowtie B)
  by (rdes-eq cls: assms)

lemma AlternateR-Chaos:
  if $R \in A \cdot g(i) \rightarrow Chaos$ then Chaos = Chaos
  by (cases A = {}, simp, rdes-eq)

lemma choose-srd-par:
  choose $\parallel R \parallel choose R = choose R$
  by (rdes-eq)

11.13 Lifting designs to reactive designs

definition des-rea-lift :: '@s hrel-des @'s hrel-rsp (R D) where
lemma periR-des-rea-lift [rdes]:
periR(R_D(P)) = (false ⊲ [pre_D(P)]_S ⊩ ($tr \leq u \, $tr'))
by (rel-auto)

lemma postR-des-rea-lift [rdes]:
postR(R_D(P)) = (true ⊲ [pre_D(P)]_S ⊩ (¬ $tr \leq u \, $tr')) \Rightarrow ($tr' = u \, $tr ∧ [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 ⊢ n Q)
by (metis H1-H3-commute H1-H3-is-normal-design H1-idem Healthy-def assms)
show ?thesis
apply (rule NSRD-intro)
apply (simp-all add: closure rdes unrest P)
apply (rel-auto)
done
qed

lemma R-D-mono:
assumes P is H Q is H P ⊑ 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 (3) design-post-choice
     semilattice-sup-class.sup.orderE utp-pred-laws.inf.coboundedH1 utp-pred-laws.inf.commute utp-pred-laws.sup.order-iff)
done

Homomorphism laws

lemma R-D-Miracle:
R_D(⊤_D) = Miracle
by (simp add: Miracle-def, rel-auto)

lemma R-D-Chaos:
R_D(⊥_D) = Chaos
proof –
have R_D(⊥_D) = R_D(false ⊢ true)
by (rel-auto)
also have ... = R_a (false ⊢ false o ($tr' = u \, $tr))
by (simp add: Chaos-def des-rea-lift-def alpha)
also have ... = R_a (true)
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:
\[
R_D(P \triangleleft b \triangleleft Q) = R_D(P) \triangleleft b \triangleright R_D(Q)
\]
by (rule antisym, rel-auto+)

lemma \(R-D\)-seq-ndesign:
\[
R_D(p_1 \vdash_n Q_1) \triangleleft R_D(p_2 \vdash_n Q_2) = R_D((p_1 \vdash_n Q_1) :: (p_2 \vdash_n Q_2))
\]
apply (rule antisym)
apply (rule SRD-refine-intro)
apply (simp-all add: closure rdes ndesign-composition-up)
using dual-order. trans apply (rel-blast)
using dual-order. trans apply (rel-blast)
apply (rel-auto)
apply (rel-auto)
apply (rel-auto)
done

lemma \(R-D\)-seq:
assumes \(P\) is \(N\) \(Q\) is \(N\)
shows \(R_D(P) :: Q\) = \(R_D(P) ;; Q\)
by (metis \(R-D\)-seq-ndesign assms ndesign-form)

These laws are applicable only when there is no further alphabet extension

lemma \(R-D\)-skip:
\(R_D(\text{II}_D) = (\text{II}_R :: (s::trace,\text{unit}) hrel-rsp)\)
apply (rel-auto) using minus-zero-eq by blast+

lemma \(R-D\)-assigns:
\(R_D(\langle \sigma \rangle D) = (\langle \sigma \rangle R :: (s::trace,\text{unit}) hrel-rsp)\)
by (simp add: assigns-d-def des-rea-lift-def alpha assigns-srd-RHS-tri-des, rel-auto)

end

12 Instantaneous Reactive Designs

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

  definition ISRD1 :: '(s::trace,\alpha) hrel-rsp ⇒ (s,\alpha) hrel-rsp where
  [upred-defs]: ISRD1(P) = P ||_R R_s(true_r ⊩ false o (\$tr_r − \$tr))

  definition ISRD :: (s,\alpha) hrel-rsp ⇒ (s,\alpha) hrel-rsp where
  [upred-defs]: ISRD = ISRD1 ◦ NSRD

  lemma ISRD1-idem: ISRD1(ISRD1(P)) = ISRD1(P)
  by (rel-auto)

  lemma ISRD1-monotonic: P ⊆ Q ⇒ ISRD1(P) ⊆ ISRD1(Q)
  by (rel-auto)

  lemma ISRD1-RHS-design-form:
  assumes \$ok' \# P \$ok' \# Q \$ok' \# R

end

105
shows $\text{ISRD1}(R_s(P \triangledown Q \triangledown R)) = R_s(P \triangledown false \triangledown (R \land $tr' =_u $tr))$

using assms by (simp add: ISRD1-def choose-srd-def RHS-tri-design-par unrest, rel-auto)

lemma ISRD1-form:
$\text{ISRD1}(\text{SRD}(P)) = R_s(p_{\triangledown R}(P) \triangledown false \triangledown (p_{\triangledown R}(P) \land $tr' =_u $tr))$
by (simp add: ISRD1-RHS-design-form SRD-as-reactive-tri-design unrest)

lemma ISRD1-rdes-def [rdes-def]:
\[
\begin{aligned}
\text{P is RR; R is RR} & \quad \Rightarrow \quad \text{ISRD1}(R_s(P \triangledown Q \triangledown R)) = R_s(P \triangledown false \triangledown (R \land $tr' =_u $tr)) \\
\end{aligned}
\]
by (simp add: ISRD1-def rdes-def closure rpred)

lemma ISRD-intro:
assumes P is NSRD peri R (P) = (¬ r_{\triangledown R}(P)) ($\triangledown R$ true $\triangledown R$ true) $\triangledown post R (P)$
shows P is ISRD
proof –
have $R_s(p_{\triangledown R}(P) \triangledown peri_{\triangledown R}(P) \triangledown post_{\triangledown R}(P))$ is ISRD1
apply (simp add: Healthy-def rdes-def closure assms (1 – 2))
using assms(3) least-zero apply (rel-blast)
done
hence P is ISRD1
by (simp add: ISRD-def Healthy-comp assms (1))
thus ?thesis
by (simp add: ISRD1-rdes-def assms closure unrest utp-pred-laws.inf.absorb1)
qed

lemma ISRD1-rdes-intro:
assumes P is RR Q is RR ($\triangledown R$ true $\triangledown R$ true) $\triangledown Q$
shows $R_s(P \triangledown false \triangledown Q)$ is ISRD1
unfolding Healthy-def
by (simp add: ISRD1-rdes-def assms closure unrest utp-pred-laws.inf.absorb1)

lemma ISRD-implies-ISRD1:
assumes P is ISRD
shows P is ISRD1
proof –
have $\text{ISRD}(P)$ 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:ISRD(P) = R_s((¬ r_{\triangledown R}(P) ; R true R true) $\triangledown false \triangledown (p_{\triangledown R}(P) \land $tr' =_u $tr))
by (simp add: NSRD-form ISRD1-def ISRD-def RHS-tri-design-par rdes-def unrest closure)
moreover have ... is SRD
by (simp add: closure unrest)

qed
ultimately have ISRD(P) is SRD
  by (simp)
with assms show thesis
  by (simp add: Healthy-def)
qed

lemma ISRD-imples-NSRD [closure]:
  assumes P is ISRD
  shows P is NSRD
proof –
  have 1:ISRD(P) = ISRD1(RD3(SRD(P)))
    by (simp add: ISRD-def NSRD-def SRD-def, metis RD1-RD3-commute RD3-left-subsumes-RD2)
  also have ... = ISRD1(RD3(P))
    by (simp add: assms ISRD-implies-SRD)
  also have ... = Rₗ ((¬ₗ preₗ P) wpₗ falseₗ h ⌷ (∃ $st′ • periₗ P) ⊢ postₗ P)
    by (simp add: RD3-def, subst SRD-right-unit-tri-lemma, simp-all add: assms ISRD-implies-SRD)
  also have ... = (¬ₗ preₗ P) wpₗ falseₗ h ⌷ (postₗ P ∧ $tr′ = $tr)
    by (simp add: RHS-tri-design-par ISRD1-def unrest choose-srd-def rpred closure I SRD-implies-SRD assms)
  finally show thesis ..
qed

lemma ISRD-form:
  assumes P is ISRD
  shows Rₗ (preₗ P) ⊢ (false ○ (postₗ(P) ∧ $tr′ = $tr)) = P
proof –
  have P = ISRD1(P)
    by (simp add: ISRD-implies-ISRD1 assms Healthy-if)
  also have ... = ISRD1(Rₗ (preₗ P ⊢ periₗ(P) ○ postₗ(P)))
    by (simp add: RD3-reactive-tri-design ISRD-implies-SRD assms)
  also have ... = Rₗ (preₗ P ⊢ false ○ (postₗ(P) ∧ $tr′ = $tr))
    by (simp add: RHS-tri-normal-design-composition ISRD1-rdes-def closure assms)
  finally show thesis ..
qed

lemma ISRD-elim [RD-elim]:
  [ P is ISRD; Q(Rₗ (preₗ P) ⊢ false ○ (postₗ(P) ∧ $tr′ = $tr)) ] ⇒ Q(P)
by (simp add: ISRD-form)

lemma skip-srd-ISRD [closure]: IIₗ is ISRD
by (rule ISRD-intro, simp-all add: rdes closure)

lemma assigns-srd-ISRD [closure]: ⟨σ⟩ₗ is ISRD
by (rule ISRD-intro, simp-all add: rdes closure, rel-auto)

lemma seq-ISRD-closed:
  assumes P is ISRD Q is ISRD
  shows P ; Q is ISRD
apply (insert assms)
apply (erule ISRD-elim)+
apply (simp add: rdes-def closure assms unrest)
apply (rule ISRD-rdes-intro)
  apply (simp-all add: rdes-def closure assms unrest)
apply (rel-auto)
done

lemma ISRD-Miracle-right-zero:
  assumes P is ISRD pre R
  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 R
  shows (µ R X · P ;; X) = Chaos
proof –
  have 1: (µ R X · P ;; X) = (µ X · P ;; SRD(X))
    by (auto simp add: srdes-theory-continuous.upt-lfp-def closure assms)
  have (µ X · P ;; SRD(X)) ⊑ Chaos
    proof (rule gfp-upperbound)
    have P ;; Chaos ⊑ Chaos
      by (simp add: Healthy-if srdes-theory-continuous.bottom-closed)
    qed
    thus P ;; SRD Chaos ⊑ 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:
  (µ R X · ⟨σ⟩R ;; X) = Chaos
  by (rule ISRD-recurse-Chaos, simp-all add: closure rdes, rel-auto)

end

13 Meta-theory for Reactive Designs

theory utp-rea-designs
  imports
    utp-rdes-healths
    utp-rdes-designs
    utp-rdes-triples
    utp-rdes-normal
    utp-rdes-contracts
    utp-rdes-tactics
    utp-rdes-parallel
    utp-rdes-prog
    utp-rdes-instant
    utp-rdes-guarded
begin end
References


