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:
Foster, Simon David orcid.org/0000-0002-9889-9514, Baxter, James Edward, Cavalcanti, Ana Lucia Caneca orcid.org/0000-0002-0831-1976 et al. (2 more authors) (2018) Reactive Designs in Isabelle/UTP. Working Paper. (Unpublished)

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

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

Simon Foster    James Baxter    Ana Cavalcanti    Jim Woodcock
Samuel Canham
April 6, 2018

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

Contents

1 Introduction 3

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

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

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

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

2 Reactive Designs Healthiness Conditions

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

2.1 Preliminaries

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

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

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

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

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

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

lemma R1-des-lift-skip: R1([II]_D) = [II]_D
  by (rel-auto)

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

lemma R1-R2c-ex-st: R1 (R2c (∃ $st' · Q_1)) = (∃ $st' · R1 (R2c Q_1))
  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-rsp (II_R) where
skip-srea-def [urel-defs]: II_\text{R} = ((\exists \ \text{st} \cdot II_c) \triangleright \text{wait} \triangleright II_c)

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

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

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

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

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

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

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

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

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

lemma skip-srea-R2 [closure]: II_\text{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 :: (\text{t::trace},'\alpha','\beta) \ rel-rp \Rightarrow (\text{t},'\alpha','\beta) \ rel-rp where
  [upred-defs]: RD1(P) = (P \lor (\neg \text{ok} \land \text{str} \leq_{\text{u}} \text{str}'))

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

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

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

lemma RD1-mono: P \sqsubseteq Q \Rightarrow RD1(P) \sqsubseteq RD1(Q)
  by (rel-auto)

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

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

lemma R1-true-RD1-closed [closure]: R1(true) is RD1
lemma RD1-wait-false [closure]: $P \text{ is RD1} \implies P[\text{false}/\text{wait}] \text{ is RD1}$
by (rel-auto)

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

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

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

lemma RD1-R1-commute: $\text{RD1}(\text{R1}(P)) = \text{R1}(\text{RD1}(P))$
by (rel-auto)

lemma RD1-R2c-commute: $\text{RD1}(\text{R2c}(P)) = \text{R2c(\text{RD1}(P))}$
by (rel-auto)

lemma RD1-via-R1: $\text{R1}(\text{H1}(P)) = \text{RD1}(\text{R1}(P))$
by (rel-auto)

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

lemma skip-rea-RD1-skip: $II_c = \text{RD1}(II)$
by (rel-auto)

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

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

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

**Theorem** RD1-left-unit:
assumes $P$ is $R1$ $P$ is $RD1$
shows $(Ie ;; P) = P$

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

**Lemma** RD1-alt-def:
assumes $P$ is $R1$
shows $RD1(P) = (P ⊳$ok$ ⊲ R1(true))$

**Proof** –
have $RD1(R1(P)) = (R1(P) ⊳$ok$ ⊲ R1(true))$
  by (rel-auto)
thus ?thesis
  by (simp add: Healthy-if assms)
qed

**Theorem** RD1-algebraic:
assumes $P$ is $R1$
shows $P$ is $RD1$ ←→ $(R1(true)h ;; P) = R1(true)$

**Proof** 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 ⇒ (∀t, 'a) hrel-rp where
[upred-defs]: $R3c(P) = (Ie$ok$ A P)$

**Definition** R3h :: (∀s, 't::trace, 'a) hrel-rsp ⇒ (∀s, 't, 'a) hrel-rsp where
$R3h-def$ [upred-defs]: $R3h(P) = (∃ s$wait$ A 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: $P ⊆ Q$ ⇒ $R3c(P) ⊆ R3c(Q)$
  by (rel-auto)

**Lemma** R3c-Monotonic: Monotonic $R3c$
  by (simp add: mono-def R3c-mono)
lemma R3c-Continuous: Continuous R3c
by (rel-auto)

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

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

lemma R3h-mono: P ⊆ Q ⇒ R3h(P) ⊆ R3h(Q)
by (rel-auto)

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

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

lemma R3h-inf: R3h(P ∩ Q) = R3h(P) ∩ R3h(Q)
by (rel-auto)

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

lemma R3h-cond: R3h(P ⊲ b ⊳ Q) = (R3h(P) ⊲ b ⊳ R3h(Q))
by (rel-auto)

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

lemma R3c-RD1-def: P is RD1 ⇒ R3c(P) = RD1(R3(P))
by (simp add: Healthy-if R3c-via-RD1-R3)

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

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

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

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

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

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

lemma R3c-cancels-R3: R3c(R3(P)) = R3c(P)
by (rel-auto)
lemma R3-cancels-R3c: $R3c(R3c(P)) = R3(R3c(P))$
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 $R3c$ R1
shows $(P ;; Q)$ is $R3c$
by (metis Healthy-def' R3c-semir-form assms)

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

lemma R3c-R3-left-seq-closure:
assumes P is $R3c$ Q is $R3c$
s shows $(P ;; Q)$ is $R3c$
proof –
  have $(P ;; Q) = ((P ;; Q)[true/\$wait\] < \$wait \triangleright (P ;; Q))$
    by (metis cond-var-split cond-var-subst-right in-var-uvvar wait-vwb-lens)
  also have $\ldots = (((II < \$wait \triangleright P) ;; Q)[true/\$wait\] < \$wait \triangleright (P ;; Q))$
    by (metis Healthy-def' R3-def assms(1))
  also have $\ldots = (((II[true/\$wait\] ;; Q) < \$wait \triangleright (P ;; Q))$
    by (subst-tac)
  also have $\ldots = (((II \land \$wait') ;; Q) < \$wait \triangleright (P ;; Q))$
    by (metis (no-types, lifting) cond-def conj-pos-var-subst seqr-pre-var-out skip-var utp-pred-laws.inf-left-idem wait-vwb-lens)
  also have $\ldots = (((II[true/\$wait'] ;; Q[true/\$wait\]) < \$wait \triangleright (P ;; Q))$
    by (metis seqr-pre-transfer seqr-right-one-point true-alt-def uovar-cover upred-eq-true utp-rel.unrest-uvvar vwb-lens-mub wait-vwb-lens)
  also have $\ldots = (((II[true/\$wait'] ;; IIc < \$wait \triangleright Q)[true/\$wait\] < \$wait \triangleright (P ;; Q))$
    by (metis Healthy-def' R3c-def assms(2))
  also have $\ldots = (((II[true/\$wait'] ;; IIc[true/\$wait\]) < \$wait \triangleright (P ;; Q))$
    by (subst-tac)
  also have $\ldots = (((II \land \$wait') ;; IIc < \$wait \triangleright (P ;; Q))$
    by (metis seqr-pre-transfer seqr-right-one-point true-alt-def uovar-cover upred-eq-true utp-rel.unrest-uvvar vwb-lens-mub wait-vwb-lens)
  also have $\ldots = ((II ;; IIc) < \$wait \triangleright (P ;; Q))$
    by (simp add: cond-def seqr-pre-transfer utp-rel.unrest-uvvar)
  also have $\ldots = (IIc < \$wait \triangleright (P ;; Q))$
    by simp
  also have $\ldots = R3c(P ;; Q)$
    by (simp add: R3c-def)
finally show ?thesis
  by (simp add: Healthy-def')
lemma R3c-cases: \( R3c(P) = ((II \triangleleft ok \triangleright R1(true)) \triangleleft \$wait \triangleright P) \)
by (rel-auto)

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

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

lemma R3h-wait-true:
assumes \( P \) is \( R3h \)
shows \( P \ t = II \triangleright \$\text{wait} \triangleright P \ t \)
proof -
  have \( P \ t = (II_R \triangleleft \$\text{wait} \triangleright P) \ t \)
    by (metis Healthy-if R3h-form assms)
  also have \( \ldots = II_R \ t \)
    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 ⇒ ('t,'a) hrel-rp (R)
where [upred-defs]: RH(P) = R1(R2c(R3c(P)))

definition RHS :: ('s,'t::trace,'α) hrel-rsp ⇒ ('s,'t,'α) hrel-rsp (R_s)
where [upred-defs]: RHS(P) = R1(R2c(R3h(P)))

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

definition SRD :: ('s,'t::trace,'α) hrel-rsp ⇒ ('s,'t,'α) 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(R(P)) = R(P)
by (simp add: R1-R2c-commute R1-R3c-commute R1-idem R2c-R3c-commute R2c-idem R3c-idem RH-def)

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

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

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

lemma RHS-idem: R_s(R_s(P)) = R_s(P)
by (simp add: R1-R2c-is-R2 R1-R3h-commute R2-idem R2c-R3h-commute R3h-idem RHS-def)

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

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

lemma RHS-mono: P ⊆ Q ⇒ R_s(P) ⊆ R_s(Q)
using mono-def RHS-Monotonic by blast

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

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

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

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

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

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

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

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

lemma RD2-RH-commute: RD2(R(P)) = R(RD2(P))
by (metis R1-H2-commute R2c-H2-commute RD2-R3c-commute RD2-def RH-def)

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

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

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

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

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

lemma RD2-RHS-commute: RD2(Rs(P)) = Rs(RD2(P))
by (metis R1-H2-commute R2c-H2-commute RD2-R3h-commute RD2-def RHS-def)

lemma SRD-idem: SRD(SRD(P)) = SRD(P)
by (simp add: RD1-RD2-commute RD1-RHS-commute RD1-idem RD2-RHS-commute RD2-idem RHS-idem SRD-def)
lemma SRD-Idempotent [closure]: Idempotent SRD
  by (simp add: Idempotent-def SRD-idem)

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

lemma SRD-Continuous [closure]: Continuous SRD
  by (simp add: Continuous-comp RD1-Continuous RD2-Continuous RHS-Continuous SRD-comp)

lemma SRD-RHS-H1-H2: SRD(P) = Rs(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 P is RD2
  apply (metis Healthy-def R1-R2c-is-R2 RHS-def SRD-def assms)
  apply (metis Healthy-def R1-R2c-is-R2 RD1-RHS-commute RD2-RHS-commute RHS-def SRD-def assms)
  apply (metis Healthy-def RD1-RHS-commute RD2-RHS-commute RHS-def SRD-def assms)
  apply (metis Healthy-def RD2-RHS-commute RD2-RHS-commute RHS-def SRD-def assms)
  apply (metis Healthy-def RD1-RHS-commute RD2-RHS-commute RHS-def SRD-def assms)
  apply (metis Healthy-def RD2-RHS-commute RD2-RHS-commute RHS-def 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-true-wait-true [usubst]:
  assumes P is SRD
  shows P[true, true/$ok$/wait] = (∃ $st · II)[true, true/$ok$/wait]
  proof
    have P = (∃ $st · II) $ok $ok $ok $wait $wait
      by (metis Healthy-def R3h-cases SRD-healths assms)
    moreover have ((∃ $st · II) $ok $ok $ok $wait $wait)[true, true/$ok$/wait] = (∃ $st · II)[true, true/$ok$/wait]
      by (simp add: usubst)
    ultimately show ?thesis
      by (simp)
  qed

lemma SRD-left-zero-1: P is SRD =⇒ R1(true) (; P = R1(true)
  by (simp add: RD1-left-zero SRD-healths(1) SRD-healths(4))

lemma SRD-left-zero-2:
  assumes P is SRD
  shows (∃ $st · II)[true, true/$ok$/wait] (; P = (∃ $st · II)[true, true/$ok$/wait]
  proof
    have (∃ $st · II)[true, true/$ok$/wait] (; R3h(P) = (∃ $st · II)[true, true/$ok$/wait]
2.7 UTP theories

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

typedecl RDES
typedecl SRDES

abbreviation RDES ≡ UTHY(RDES, ('t::trace,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-reagalois:
galois-connection (RDES ↩(RD1 o RD2, R3) → REA)
proof (simp add: mk-conn-def, rule galois-connection1', simp-all add: utp-partial-order rdes-hcond-def rda-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-ifff Healthy-def, metis RD-def RD-idem)
  show isolate (utp-order RD) (utp-order RP) R3
    by (simp add: R3-Monotonic isolate-utp-order1)
  show isolate (utp-order RP) (utp-order RD) (RD1 o RD2)
    by (simp add: Monotonic-comp RD1-Monotonic RD2-Monotonic isolate-utp-order1)
  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-ifff)
next
  fix P :: ('a, 'b) hrel-rp
  assume a: P is RP
thus $R_3 \left( R D_1 \left( R D_2 \right) \right) \subseteq P$

proof

have $R_3 \left( R D_1 \left( R D_2 \right) \right) = R P \left( R D_1 \left( R D_2 \right) \right)$

by (metis Healthy-if R3-RD-RP RD-def a)

moreover have $R D_1 \left( R D_2 \right) \subseteq P$

by (rel-auto)

ultimately show ?thesis

by (metis Healthy-if RP-mono a)

qed

qed

interpretation rdes-rea-retract:

retract $(R D E S \leftarrow (R D_1 \circ R D_2, R_3) \rightarrow R E A)$

by (unfold-locales, simp-all add: mk-conn-def utp-partial-order rdes-hcond-def rea-hcond-def)

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

interpretation srdes-theory: utp-theory $U T H Y \left( S R D E S, \left( 's, 't::trace, 'a \right) \text{ rsp} \right)$

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

interpretation srdes-theory-continuous: utp-theory-continuous $U T H Y \left( S R D E S, \left( 's, 't::trace, 'a \right) \text{ rsp} \right)$

rewrites $\forall P. P \in \text{carrier} \left( u t h y \text{-order } S R D E S \right) \leftrightarrow P \text{ is SRD}$

and $P \text{ is } H_{S R D E S} \leftarrow \text{carrier} \left( u t h y \text{-order } S R D E S \right)$

and $(\mu X \cdot F (H_{S R D E S} X)) = (\mu X \cdot F (S R D X))$

and carrier (uthy-order SRDES) $\rightarrow$ carrier (uthy-order SRDES) $\equiv [S R D]_H \rightarrow [S R D]_H$

and le (uthy-order SRDES) $\equiv [S R D]_H \rightarrow [S R D]_H$

and eq (uthy-order SRDES) $\equiv op \subseteq$

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

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

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

abbreviation Chaos :: \((s, 't::trace, 'a)\) hrel-rsp where

Chaos $\equiv \bot_{SRDES}$

abbreviation Miracle :: \((s, 't::trace, 'a)\) hrel-rsp where

Miracle $\equiv \top_{SRDES}$

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

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

thm srdes-theory-continuous.meet-bottom

thm srdes-theory-continuous.meet-top

abbreviation srd-lfp ($\mu_R$) where $\mu_R F \equiv \mu_{S R D E S} F$

abbreviation srd-gfp ($\nu_R$) where $\nu_R F \equiv \nu_{S R D E S} F$

syntax

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

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

translations

$\mu_R X \cdot P \equiv \mu_R \left( \lambda X. P \right)$

$\nu_R X \cdot P \equiv \nu_R \left( \lambda X. P \right)$

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 (µR X · F(X)) = (µ X · F(SRD(X))
  by (metis assms srdes-hcond-def srdes-theory-continuous.utp-lfp-def)
end

3 Reactive Design Specifications

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

3.1 Reactive design forms

lemma srdes-skip-def: II R = R ≅ (true ⊢ (str = u str ∧ ¬ wait’ ∧ [II] R))
  apply (rel-auto) using minus-zero-eq by blast+

lemma Chaos-def: Chaos = R s (false ⊢ true)
  proof
    have Chaos = SRD(true) by (metis srdes-hcond-def srdes-theory-continuous.healthy-bottom)
    also have ... = R s (H(true)) by (simp add: SRD-RHS-H1-H2)
    also have ... = R s (false ⊢ false) by (metis H1-design H2-true design-false-pre)
    finally show ?thesis .
  qed

lemma Miracle-def: Miracle = R s (true ⊢ false)
  proof
    have Miracle = SRD(false) by (metis srdes-hcond-def srdes-theory-continuous.healthy-top)
    also have ... = R s (H(false)) by (simp add: SRD-RHS-H1-H2)
    also have ... = R s (true ⊢ true) by (metis no-types, lifting) H1-H2-eq-design p-imp-p subst-impl subst-not utp-pred-laws.compl-bot-eq utp-pred-laws.compl-top-eq
    finally show ?thesis .
  qed

lemma RD1-reactive-design: RD1(R(P ⊢ Q)) = R(P ⊢ Q)
  by (rel-auto)

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

lemma RD1-st-reactive-design: RD1(R s (P ⊢ Q)) = R s (P ⊢ Q)
  by (rel-auto)

lemma RD2-st-reactive-design:
  assumes $ok’ P $ok’ Q

15
shows $RD2(R_s(P \vdash Q)) = R_s(P \vdash Q)$
using assms
by (metis H2-design RD2-RHS-commute RD2-def)

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

lemma RD-RH-design-form:
$RD(P) = R((\neg P f) \vdash P' f)$
proof –
have $RD(P) = RD1(RD2(R1(R2c(R3c(P)))))$
  by (simp add: RD-alt-def RH-def)
also have ...
also have ...
also have ...
also have ...
also have ...
also have ...
also have ...
also have ...
also have ...
also have ...
also have ...
also have ...
also have ...
also have ...
also have ...
also have ...
also have ...
by (simp add: R1-R2-commute R1-idem R2-def)
finally show ?thesis.
qed

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

lemma RD-RH-design:
assumes $\ok' \not\in P \not\in Q$
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 $\ok' \not\in P \not\in Q$
shows $R(P \vdash Q)$ is RD
by (simp add: RD-RH-design Healthy-def' assms(1) assms(2))
lemma SRD-RH-design-form:  
\[ \text{SRD}(P) = R_s((\neg P f f) \vdash P^t_f) \]

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
       RD1-R1-commute RD1-R2c-commute RD1-R3h-commute RD2-R3h-commute RD2-def RHS-def SRD-def)
  also have \( ... = R_1(R_2c(H(P)))) \)
  by (metis (no-types, lifting) R1-H2-commute R1-R2c-is-R2 R1-R3h-commute R2-R1-form RD1-via-R1
       RD2-def)
  also have \( ... = R_1(R_3h(P))) \)
  by (simp add: R1-R2s-R2c RHS-def)
  also have \( ... = R_1((\neg P f f) \vdash P^t_f) \)
  by (metis SRD-RH-design-form)
  finally show \( \text{thesis} \).
qed

lemma SRD-reactive-design:  
assumes \( P \) is SRD
shows \( R_s((\neg P f f) \vdash P^t_f) = P \)
by (metis SRD-RH-design-form)

lemma SRD-RH-design:  
assumes \( \text{ok} \sharp P \)$\text{ok} \sharp Q$
shows \( \text{SRD}(R_s(P \vdash Q)) = R_s(P \vdash Q) \)
by (simp add: RD1-st-reactive-design RD2-st-reactive-design RHS-idem SRD-def)

lemma RHS-design-is-SRD:  
assumes \( \text{ok} \sharp P \)$\text{ok} \sharp Q$
shows \( R_s(P \vdash Q) \) is SRD
by (simp add: Healthy-def SRD-RH-design)

lemma SRD-RHS-H1-H2: \( \text{SRD}(P) = R_s(H(P)) \)
by (metis (no-types, lifting) H1-H2-eq-design)

3.2 Auxiliary healthiness conditions

definition \( \text{upred-defs} \): \( R_3c-pre(P) = (\text{true} \triangle \text{wait} \triangleright P) \)

definition \( \text{upred-defs} \): \( R_3c-post(P) = ([I]_D \triangle \text{wait} \triangleright P) \)

definition \( \text{upred-defs} \): \( R_3h-post(P) = ((\exists \text{st} \cdot [I]_D) \triangle \text{wait} \triangleright P) \)

lemma R3c-pre-conj: \( R_3c-pre(P \land Q) = (R_3c-pre(P) \land R_3c-pre(Q)) \)
by (rel-auto)

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

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

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

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

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

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

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

3.3 Composition laws

theorem R1-design-composition:
  fixes P Q :: ('t::trace,'a,'b) rel-rp
  and R S :: ('t,'b,'c) rel-rp
  assumes $ok' \not\in P $ok \not\in Q $ok \not\in R $ok S
  shows
    (R1(P ⊢ Q) ; R1(R ⊢ S)) =
    R1(¬ (R1(¬ P) ; R1(true)) ∧ ¬ (R1(Q) ; R1(¬ R))) ⊢ (R1(Q) ; R1(S))
proof –
  have (R1(P ⊢ Q) ; R1(R ⊢ S)) = (∃ ok₀ · (R1(P ⊢ Q)[<ok₀>/$ok'] ; (R1(R ⊢ S))[<ok₀>/$ok])
    using seqr-middle ok-vob-lens by blast
  also from assms have ... = (∃ ok₀ · R1((<ok₀> ∧ P) ⇒ (<ok₀> ∧ Q)) ; R1((<ok₀> ∧ R) ⇒ ($ok' ∧ S)))
    by (simp add: design-def R1-def usubst unrest)
  also from assms have ... = ((R1((<ok₀> ∧ P) ⇒ (true ∧ Q)) ; R1((true ∧ R) ⇒ ($ok' ∧ S)))
    ∨ (R1((<ok₀> ∧ P) ⇒ (false ∧ Q)) ; R1((false ∧ R) ⇒ ($ok' ∧ S))))
    by (simp add: false-alt-def true-alt-def)
  also from assms have ... = ((R1((<ok₀> ∧ P) ⇒ Q) ; R1(R ⇒ ($ok' ∧ S)))
    ∨ (R1((<ok₀> ∧ P) ; R1(true)))
    by simp
  also from assms have ... = ((R1(¬ $ok ∨ ¬ P ⊢ Q) ; R1(¬ R ∨ ($ok' ∧ S)))
    ∨ (R1(¬ $ok ∨ ¬ P) ; R1(true)))
    by (simp add: impl-alt-def utp-pred-laws.sup.assoc)
  also from assms have ... = (((R1(¬ $ok ∨ ¬ P) ; R1(¬ R ∨ ($ok' ∧ S)))
    ∨ (R1(Q) ; R1(¬ R ∨ ($ok' ∧ S))))
    ∨ (R1(¬ $ok ∨ ¬ P) ; R1(true)))
    by (simp add: R1-disj utp-pred-laws.disj.assoc)
  also from assms have ... = ((R1(Q) ; R1(¬ R ∨ ($ok' ∧ S)))
    ∨ (R1(¬ $ok ∨ ¬ P) ; R1(true)))
    by (rel-blast)
  also from assms have ... = ((R1(Q) ; (R1(¬ R) ∨ R1(S) ∧ $ok'))
    ∨ (R1(¬ $ok ∨ ¬ P) ; R1(true)))
    by (simp add: R1-disj R1-extend-conj utp-pred-laws.inf-commute)
  also have ... = ((R1(Q) ; R1(¬ R) ∨ R1(S) ∧ $ok'))
    ∨ (R1(¬ $ok) :: ('t,'a,'b) rel-rp) ; R1(true))
    ∨ (R1(¬ P) ; R1(true)))
    by (simp add: R1-disj seqr-or-distl)
  also have ... = ((R1(Q) ; R1(¬ R) ∨ R1(S) ∧ $ok'))

\[ \begin{align*}
&\lor (R1(\neg \$ok)) \\
&\lor (R1(\neg P) \iff R1(\text{true}))
\end{align*} \]

proof –

have \((R1(\neg \$ok) :: \left('t, \alpha, \beta\right) \text{ rel-rp}) :: R1(\text{true})) = \\
(R1(\neg \$ok) :: \left('t, \alpha, \gamma\right) \text{ 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(\text{true}))
by (simp add: \text{ R1-extend-conj})

also have \(... = ( (R1(Q) :: (R1(\neg R)))
\lor (R1(Q) :: (R1(S \land \$ok^'))) \\
\lor (\neg \$ok) \\
\lor (R1(\neg P) :: R1(\text{true}))
by (simp add: \text{ R1-disj R1-seq}))

also have \(... = R1( (R1(Q) :: (R1(\neg R)))
\lor ((R1(Q) :: (R1(S)) \land \$ok^')) \\
\lor (\neg \$ok) \\
\lor (R1(\neg P) :: R1(\text{true}))
by (rel-blast)

also have \(... = R1(\neg(\$ok \land \neg (R1(\neg P) :: R1(\text{true})) \land \neg (R1(Q) :: (R1(\neg R))))
\lor ((R1(Q) :: (R1(S)) \land \$ok^'))
by (rel-blast)

also have \(... = R1(\neg(\$ok \land (R1(\neg P) :: R1(\text{true})) \land \neg (R1(Q) :: (R1(\neg R))))
\lor (\neg \$ok) \\
\lor (R1(\neg P) :: R1(\text{true}))
by (simp add: \text{ impl-alt-def utp-pred-laws.inf-commute})

also have \(... = R1(\neg (R1(\neg P) :: R1(\text{true})) \land \neg (R1(Q) :: (R1(\neg R))) \iff (R1(Q) :: R1(S)))
by (simp add: \text{ design-def})

finally show \(?thesis
qed

\begin{align*}
\text{theorem R1-design-composition-RR:} \\
\text{ assumes } P \text{ is RR Q is RR R is RR S is RR} \\
\text{ shows } (R1(P \rightarrow Q) :: R1(R \rightarrow S)) = R1((\neg P \land \text{ wp}_r, \text{ false} \land Q \land \text{ wp}_r, R) \rightarrow (Q :: S)) \\
\text{ apply (subst R1-design-composition)} \\
\text{ apply (simp-all add: assms unrest wp-rea-def Healthy-if closure)} \\
\text{ apply (rel-auto)}
\end{align*} \]
done

\begin{align*}
\text{theorem R1-design-composition-RC:} \\
\text{ assumes } P \text{ is RC Q is RR R is RR S is RR} \\
\text{ shows } (R1(P \rightarrow Q) :: R1(R \rightarrow S)) = R1((P \land Q \land \text{ wp}_r, R) \rightarrow (Q :: S)) \\
\text{ by (simp add: R1-design-composition-RR assms unrest Healthy-if closure wp)}
\end{align*} \]

\text{lemma R2s-design: R2s(P \rightarrow Q) = (R2s(P) \rightarrow R2s(Q))}
lemma R2c-design: $R2c(P \vdash Q) = (R2c(P) \vdash R2c(Q))$
by (simp add: R2c-def design-def usubst)

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

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

lemma R3c-R1-design-composition:
assumes $\text{thesis} : (\exists P \ \text{sok} \ P \ \text{sok} \ Q \ \text{sok} \ R \ \text{sok} \ S)$
sows $R3c(R1(P \vdash Q))$ ;; $R3c(R1(R \vdash S)) = R3c(R1(\neg (R1(\neg P) ;; R1(true)) \land \neg (\neg (R1(Q) \land \neg \text{wait}') ;; R1(\neg R)))$
$\vdash (R1(Q) ;; (\exists \text{st} \cdot \lfloor II \rfloor_D \text{sd} \text{wait} \triangleright R1(S))))$
proof
have 1:$(\neg (R1(\neg R3c-pre P) ;; R1 true)) = (R3c-pre(\neg (R1(\neg P) ;; R1 true)))$
by (rel-auto)

have 3:$(R1(R3c-post Q) ;; R1(R3c-pre S)) = R3c-post(R1(Q) ;; (\exists \text{st} \cdot \lfloor II \rfloor_D \text{sd} \text{wait} \triangleright R1(S)))$
by (rel-auto)

show ?thesis
apply (simp add: R3c-semir-form R1-R3c-commute THEN sym) R1-R3c-design unrest)
apply (subst R1-design-composition)
apply (simp-all add: unrest assms R3c-pre-conj 1 2 3)
done
qed

lemma R3h-R1-design-composition:
assumes $\text{thesis} : (\exists P \ \text{sok} \ P \ \text{sok} \ Q \ \text{sok} \ R \ \text{sok} \ S)$
sows $R3h(R1(P \vdash Q))$ ;; $R3h(R1(R \vdash S)) = R3h(R1(\neg (R1(\neg P) ;; R1(true)) \land \neg (\neg (R1(Q) \land \neg \text{wait}') ;; R1(\neg R)))$
$\vdash (R1(Q) ;; (\exists \text{st} \cdot \lfloor II \rfloor_D \text{sd} \text{wait} \triangleright R1(S))))$
proof
have 1:$(\neg (R1(\neg R3c-pre P) ;; R1 true)) = (R3c-pre(\neg (R1(\neg P) ;; R1 true)))$
by (rel-auto)

have 3:$(R1(R3h-post Q) ;; R1(R3h-pre S)) = R3h-post(R1(Q) ;; (\exists \text{st} \cdot \lfloor II \rfloor_D \text{sd} \text{wait} \triangleright R1(S)))$
by (rel-auto, blast+)

show ?thesis
apply (simp add: R3h-semir-form R1-R3h-commute THEN sym) R1-R3h-design unrest)
apply (subst R1-design-composition)
apply (simp-all add: unrest assms R3c-pre-conj 1 2 3)
done
qed

lemma R2-design-composition:
assumes $\text{thesis} : (\exists P \ \text{sok} \ P \ \text{sok} \ Q \ \text{sok} \ R \ \text{sok} \ S)$
sows $R2(P \vdash Q) ;; R2(R \vdash S) = R2((\neg (R1(\neg R2c P) ;; R1 true) \land \neg (R1(R2c Q) ;; R1(\neg R2c R))) \vdash (R1(R2c Q) ;; R1
proof (simp add: R2-R2c-def R2c-design R1-design-composition assms unrest R2c-not R2c-and R2c-disj R1-R2c-commute THEN assms R2c-idem R2c-R1-seq)
  apply (metis (no-types, lifting) R2c-R1-seq R2c-not R2c-true)
done

lemma RH-design-composition:
  assumes $\$ok \not \not P \$ok \not \not Q \$ok \not \not R \$ok \not \not S$
  shows $(RH(P \leftarrow Q) :: RH(R \leftarrow S)) =$
  $RH(\neg (R1 (\neg R2s P) :: R1 true) \land \neg ((R1 (R2s Q) \land \neg \$wait \$') :: R1 (\neg R2s R))) \leftarrow$
  $(R1 (R2s Q) :: ([|D|] D \$wait \triangleright R1 (R2s S))))$
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)
  finally show ?thesis
  by (simp add: R2-def R2s-not R2s-true) 
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 $((R1 (R2s Q) \land \neg \$wait \$') :: R1 (\neg R2s R)) = R1 (R2s (Q \land \neg \$wait \$') :: R2 (\neg R))$
  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-seqr-distribute)
also have ... = $((R1 (R2s Q) \land \neg \$wait \$') :: R1 (\neg R2s R))$
  by (rel-auto)
finally show ?thesis . 
qed

have 3:$R2c[(R1 (R2s Q) :: ([|D|] D \$wait \triangleright R1 (R2s S)))) = (R1 (R2s Q) :: ([|D|] D \$wait \triangleright R1 (R2s S)))]$
proof  
  have $R2c(((R1 (R2s Q)][true/$\$wait /]) :: ([|D|] D \$wait \triangleright R1 (R2s S)))[true/$\$wait ])) =$
  $((R1 (R2s Q)][true/$\$wait /]) :: ([|D|] D \$wait \triangleright R1 (R2s S)))[true/$\$wait ])
  by (simp add: usubst cond-unit-T R1-def R2s-def)
  also have ... = $R2c(R2(Q[true/$\$wait /]) :: R2([|D|] D[true/$\$wait ]))$
  by (metis R2-def R2-des-lift-skip R2-subst-wait-true)
  also have ... = $(R2(Q[true/$\$wait /]) :: R2([|D|] D[true/$\$wait ]))$
  using R2c-seq by blast
  also have ... = $((R1 (R2s Q)][true/$\$wait /]) :: ([|D|] D \$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-true)
do
finally show ?thesis.
qed
moreover have \( R2c(((R1 (R2s Q))[false/\$wait']) ;; ([\{I\}]D < \$wait \triangleright R1 (R2s S))[false/\$wait']) = (((R1 (R2s Q))[false/\$wait']) ;; ([\{I\}]D < \$wait \triangleright R1 (R2s S))[false/\$wait']) \)
by (simp add: usubst cond-unit-F)
(metis (no-types, hide-lams) R1-wait'¬false R1-wait-true R2-def R2s-not R2s-true R2c-seq)
ultimately show ?thesis
proof –

have \( [[I]]_D < \$wait \triangleright R1 (R2s S) = R2 ([[I]]_D < \$wait \triangleright S) \)
by (simp add: R1-R2c-is-R2 R1-R2s-R2c R2-condr)
then show ?thesis
by (simp add: R1-R2c-is-R2 R1-R2s-R2c R2c-seq)
qed

have (\( R1\) (\( R2s (R3c (P \vdash Q)) \) ;; \( R1\) (\( R2s (R3c (R \vdash S))) \)) = (\( R3c (R1 \vdash R2s (P \vdash Q)) \) ;; \( R3c (R1 \vdash R2s (R \vdash S)) \)) )
by (metis (no-types, hide-lams) R1-R2c-R2s-R2c R1-R3c-commute R2c-R3c-commute R2s-design)
also have \( \ldots = R3c (\begin{array}{l}
R1 (\neg (R1 (\neg R2s P) ;; R1 true) \land \neg (R1 (R2s Q) \land \neg \$wait') ;; R1 (\neg R2s R))
\end{array}) \vdash
(\begin{array}{l}
R1 (R2s Q) ;; ([\{I\}]_D < \$wait \triangleright R1 (R2s S)))
\end{array})\)
by (simp add: R3c-R1-design-composition assms unrest)
also have \( \ldots = R3c (\begin{array}{l}
R1 (\neg (R1 (\neg R2s P) ;; R1 true) \land \neg (R1 (R2s Q) \land \neg \$wait') ;; R1 (\neg R2s R))
\end{array}) \vdash
(\begin{array}{l}
R1 (R2s Q) ;; ([\{I\}]_D < \$wait \triangleright R1 (R2s S)))
\end{array})\)
by (simp add: R2c-design R2c-and R2c-not 1 2 3)
finally show ?thesis
by (simp add: R1-R2s-R2c R1-R3c-commute R2c-R3c-commute RH-def)
qed


lemma RHS-design-composition:
assumes \( \not \exists P \not \exists Q \not \exists R \not \exists S \)
shows \( R_s (\neg (R1 (\neg R2s P) ;; R1 true) \land \neg (R1 (R2s Q) \land \neg \$wait') ;; R1 (\neg R2s R)) \vdash
(\begin{array}{l}
R1 (R2s Q) ;; ([\exists \not \exists \{I\}]_D < \$wait \triangleright R1 (R2s S))
\end{array})\)
proof –

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

have \( 1: (R1 (\neg R2s P) ;; R1 true) = (R1 (R2 (\neg P) ;; R2 true)) \)
by (rel-auto, blast)
and \( R2c (R1 (R2 (\neg P) ;; R2 true)) = R2c (R1 (R2 (\neg P) ;; R2 true)) \)
using R2c-not by blast
also have \( \ldots = R2 (R2 (\neg P) ;; R2 true) \)
by (metis R1-R2c-commute R1-R2c-is-R2)
also have \( \ldots = (R2 (\neg P) ;; R2 true) \)
by (simp add: R2-seqr-distribute)
also have \( \ldots = (R1 (\neg R2s P) ;; R1 true) \)
by (simp add: R2-def R2s-not R2s-true)
finally show ?thesis
by (simp add: 1)
qed
have 2:R2c (((R1 (R2s Q) ∧ ¬ $\text{wait}'$) ; R1 (¬ R2s R)) = ((R1 (R2s Q) ∧ ¬ $\text{wait}'$) ; R1 (¬ R2s R)))
proof −
  have ((R1 (R2s Q) ∧ ¬ $\text{wait}'$) ; R1 (¬ R2s R)) = R1 (R2 (Q ∧ ¬ $\text{wait}'$) ; R2 (¬ R))
  by (rel-auto, blast+)
  hence R2c (((R1 (R2s Q) ∧ ¬ $\text{wait}'$) ; R1 (¬ R2s R)) = (R2 (Q ∧ ¬ $\text{wait}'$) ; R2 (¬ R))
  by (metis (no-types, lifting) R1-R2c-commute R1-R2c-is-R2 R2-seqr-distribute)
  also have ... = ((R1 (R2s Q) ∧ ¬ $\text{wait}'$) ; R1 (¬ R2s R))
  by (rel-auto, blast+)
finally show ?thesis.
qed

have 3:R2c((R1 (R2s Q) ; (∃ $st$ · [II]_D) < $\text{wait}$ ▷ R1 (R2s S))) =
(R1 (R2s Q) ; (∃ $st$ · [II]_D) < $\text{wait}$ ▷ R1 (R2s S))
proof −
  have R2c(((R1 (R2s Q))[true/$\text{wait}'$] ; (∃ $st$ · [II]_D) < $\text{wait}$ ▷ R1 (R2s S))[true/$\text{wait}$]) =
(R2c(R1 (R2s (Q)[true/$\text{wait}'$]) ; (∃ $st$ · [II]_D][true/$\text{wait}$])
by (simp add: usubst cond-unit-T R1-def R2s-def)
  also have ... = R2c(R2(Q)[true/$\text{wait}'$] ; R2((∃ $st$ · [II]_D)[true/$\text{wait}$])
by (metis (no-types, lifting) R2-def R2-des-lift-skip R2-subst-wait-true R2-st-ex)
  also have ... = (R2(Q[true/$\text{wait}'$]) ; R2((∃ $st$ · [II]_D)[true/$\text{wait}$])
using R2c-seq by blast
  apply (simp add: usubst R2-des-lift-skip R2-st-ex R2-subst-wait-true R2-subst-wait-true)
done
finally show ?thesis.
qed
moreover have R2c(((R1 (R2s Q))[false/$\text{wait}'$] ; (∃ $st$ · [II]_D) < $\text{wait}$ ▷ R1 (R2s S))[false/$\text{wait}$]) =
((R1 (R2s Q))[false/$\text{wait}'$] ; (∃ $st$ · [II]_D) < $\text{wait}$ ▷ R1 (R2s S))[false/$\text{wait}$])
by (simp add: usubst
  (metis (no-types, lifting) R1-wait′-false R1-wait-false R2-R1-form R2-subst-wait′-false R2-subst-wait-false
R2c-seq)
ultimately show ?thesis
  by (smt R2-R1-form R2-condr′ R2-des-lift-skip R2-st-ex R2c-seq R2s-wait)
qed

have (R1(R2s(R3h(P ⊨ Q))) ; R1(R2s(R3h(R ⊨ S)))) =
((R3h(R1(R2s(P) ⊨ R2s(Q))) ; R3h(R1(R2s(R) ⊨ R2s(S))))
by (metis (no-types, hide-lams) R1-R2s-R2c R1-R3h-commute R2c-R3h-commute R2s-design)
  also have ... = R3h(R1(R2c((¬ (R1 (¬ R2s P) ; R1 true) ∧ ¬ ((R1 (R2s Q) ∧ ¬ $\text{wait}'$) ; R1 (¬ R2s R)))))
  by (simp add: R3h-R1-design-composition assms unrest)
  also have ... = R3h(R1(R2c((¬ (R1 (¬ R2s P) ; R1 true) ∧ ¬ ((R1 (R2s Q) ∧ ¬ $\text{wait}'$) ; R1 (¬ R2s R)))))
  by (simp add: R2c-design R2c-and R2c-not 1 2 3)
finally show ?thesis
  by (simp add: R1-R2s-R2c R1-R3h-commute R2c-R3h-commute RHS-def)
qed
lemma RHS-R2s-design-composition:

assumes
$\text{ok} \not\preceq P \not\preceq Q \not\preceq R \not\preceq S$
P is R2s Q is R2s R is R2s S is R2s

shows \((R_\text{s}(P \vdash Q) ; R_\text{s}(R \vdash S)) =\)
\(R_\text{s}((\neg (R_1 (\neg P) \vdash R_1 \text{ true}) \land \neg ((R_1 Q \land \neg \$\text{wait} \not\preceq) \vdash R_1 (\neg R))) \vdash (R_1 Q \vdash (\exists \$\text{st} \cdot [I|P|d] \not\preceq \$\text{wait} \vdash R_1 S)))\)

proof –

have \(f1: R2s P = P\)
  by (meson Healthy-def assms(5))

have \(f2: R2s Q = Q\)
  by (meson Healthy-def assms(6))

have \(f3: R2s R = R\)
  by (meson Healthy-def assms(7))

have \(R2s S = S\)
  by (meson Healthy-def assms(8))

then show \(?\)thesis
  using \(f3 f2 f1\) by (simp add: RHS-design-composition assms(1) assms(2) assms(3) assms(4))

qed

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

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

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

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

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

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

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

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

24
3.4 Refinement introduction laws

**lemma** RHS-pre-lemma: \((\mathbf{R}_s P)_{pf} = RI(R2c(P_{pf}))\)
by (rel-auto)

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

**proof**

- **assumes**
  \(P_1\) is \(RI\) \(P_2\) is \(RI\) \(Q_1\) is \(RI\) \(Q_2\) is \(RI\)
  \(\$ok \nleq P_1 \$ok' \nleq P_1 \$ok \nleq P_2 \$ok' \nleq P_2\)
  \(\$ok \nleq Q_1 \$ok' \nleq Q_1 \$ok \nleq Q_2 \$ok' \nleq Q_2\)

- **shows** \(RI(P_1 \vdash P_2) \subseteq RI(Q_1 \vdash Q_2) \iff \langle P_1 \Rightarrow Q_1 \rangle \land \langle P_1 \land Q_2 \Rightarrow P_2 \rangle\)
- **proof**
  - **have** \(RI((\exists \$ok;\$ok' \cdot P_1) \vdash (\exists \$ok;\$ok' \cdot P_2)) \subseteq RI((\exists \$ok;\$ok' \cdot Q_1) \vdash (\exists \$ok;\$ok' \cdot Q_2))\)
  \(\iff \langle RI(\exists \$ok;\$ok' \cdot P_1) \Rightarrow RI(\exists \$ok;\$ok' \cdot Q_1) \land RI(\exists \$ok;\$ok' \cdot Q_2)\rangle\)
  - **by** (rel-auto, meson+)
  - **thus** ?thesis
  - **by** (simp-all add: ex-unrest ex-plus Healthy-if assms)

**qed**

**lemma** R1-design-refine-RR:
**assumes** \(P_1\) is \(RR\) \(P_2\) is \(RR\) \(Q_1\) is \(RR\) \(Q_2\) is \(RR\)
**shows** \(RI(P_1 \vdash P_2) \subseteq RI(Q_1 \vdash Q_2) \iff \langle P_1 \Rightarrow Q_1 \rangle \land \langle P_1 \land Q_2 \Rightarrow P_2 \rangle\)
**by** (simp add: R1-design-refine assms unrest closure)

**lemma** RHS-design-refine:
**assumes**
\(P_1\) is \(RI\) \(P_2\) is \(RI\) \(Q_1\) is \(RI\) \(Q_2\) is \(RI\)
\(P_1\) is \(R2c\) \(P_2\) is \(R2c\) \(Q_1\) is \(R2c\) \(Q_2\) is \(R2c\)
\(\$ok \nleq P_1 \$ok' \nleq P_1 \$ok \nleq P_2 \$ok' \nleq P_2\)
\(\$ok \nleq Q_1 \$ok' \nleq Q_1 \$ok \nleq Q_2 \$ok' \nleq Q_2\)
\(\$wait \nleq P_1 \$wait \nleq P_2 \$wait \nleq Q_1 \$wait \nleq Q_2\)
**shows** \(\mathbf{R}_s(P_1 \vdash P_2) \subseteq \mathbf{R}_s(Q_1 \vdash Q_2) \iff \langle P_1 \Rightarrow Q_1 \rangle \land \langle P_1 \land Q_2 \Rightarrow P_2 \rangle\)
**proof**
- **have** \(\mathbf{R}_s(P_1 \vdash P_2) \subseteq \mathbf{R}_s(Q_1 \vdash Q_2) \iff RI(R3h(R2c(P_1 \vdash P_2))) \subseteq RI(R3h(R2c(Q_1 \vdash Q_2)))\)
  - **by** (simp add: R2c-R3h-commute RHS-def)
- **also have** ... \(\iff RI(R3h(P_1 \vdash P_2)) \subseteq RI(R3h(Q_1 \vdash Q_2))\)
  - **by** (simp add: Healthy-if R2c-design assms)
- **also have** ... \(\iff RI(R3h(P_1 \vdash P_2))[\text{false/\$wait}] \subseteq RI(R3h(Q_1 \vdash Q_2))[\text{false/\$wait}]\)
  - **by** (rel-auto, meson+)
- **also have** ... \(\iff RI(P_1 \vdash P_2)[\text{false/\$wait}] \subseteq RI(Q_1 \vdash Q_2)[\text{false/\$wait}]\)
  - **by** (rel-auto)
- **also have** ... \(\iff RI(P_1 \vdash P_2) \subseteq RI(Q_1 \vdash Q_2)\)
  - **by** (simp add: usubst assms closure unrest)
- **also have** ... \(\iff \langle P_1 \Rightarrow Q_1 \rangle \land \langle P_1 \land Q_2 \Rightarrow P_2 \rangle\)
  - **by** (simp add: R1-design-refine assms)
  - **finally show** ?thesis
**qed**

**lemma** srdes-refine-intro:
**assumes** \(\langle P_1 \Rightarrow P_2 \rangle, \langle P_1 \land Q_2 \Rightarrow Q_1 \rangle\)

25
shows $R_s(P_1 \vdash Q_1) \subseteq R_s(P_2 \vdash Q_2)$
by (simp add: RHS-mono assms design-refine-intro)

3.5 Distribution laws

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

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

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

end

4 Reactive Design Triples

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

4.1 Diamond notation

definition wait’-cond :: 
  \(\langle t::\text{trace},\alpha,\beta\rangle\) 
  rel-rp 
  \(\Rightarrow \langle t,\alpha,\beta\rangle\) 
  rel-rp 
  \(\Rightarrow \langle t,\alpha,\beta\rangle\) 
  rel-rp \ (\text{infixr} \circ 65)\) where
[upred-defs]: $P \circ Q = (P \triangleleft \text{wait'} \circ Q)$

lemma wait’-cond-unrest [unrest]:
  \([ \text{out-var wait} \triangleright a \cdot x \isold P ; x \isold Q \] \Longrightarrow x \isold (P \circ Q)
by (simp add: wait’-cond-def unrest)

lemma wait’-cond-subst [asubst]:
  \(\text{\$wait'} \isold \sigma \Rightarrow \sigma \uparrow (P \circ Q) = (\sigma \uparrow P) \circ (\sigma \uparrow Q)\)
by (simp add: wait’-cond-def usubst unrest)

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

lemma wait’-cond-seq: \((P \circ Q) ;; R) = ((P ;; (\$wait \land R)) \lor (Q ;; (\neg \$wait \land R)))
by (simp add: wait’-cond-def cond-def seqr-or-distl, rel-blast)

lemma wait’-cond-true: \((P \circ Q \land \$wait') = (P \land \$wait')\)
by (rel-auto)

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

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

lemma wait’-cond-conj-exchange:
(\(\circ (P \land (R \circ S)) = (P \land R) \circ (Q \land S)\)
by (rel-auto)
by (rel-auto)

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

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

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

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

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

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

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

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

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

lemma R2-wait'cond: \(R2s(P \circ Q) = R2s(P) \circ R2s(Q)\)
by (simp add: wait'cond-def R2s-def R2s-def usubst)

lemma R2-wait'cond: \(R2(P \circ Q) = R2(P) \circ R2(Q)\)
by (simp add: R2-def R2-def wait'cond R1-wait'cond)

lemma wait'cond-R1closed [closure]:
\[
\{P \text{ is } R1; \ Q \text{ is } R1\} \implies P \circ Q \text{ is } R1
\]
by (simp add: Healthy-def R1-wait'cond)

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

4.2 Export laws

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

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

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

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

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

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

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

lemma RHS-design-post-R2s: $\textbf{R}_s(P \vdash Q \circ R2s(R)) = \textbf{R}_s(P \vdash Q \circ R)$
by (metis R1-R2s-R2c RHS-design-peri-R1 RHS-design-peri-R2s)

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

lemma RHS-design-lemma1:
$RHS(P \vdash (R1(R2c(Q)) \vee R) \circ S) = RHS(P \vdash (Q \vee R) \circ S)$
by (metis (no-types, lifting) R1-R2c-is-R2 R1-R2s-R2c R2-R1-form R2-disj R2c-idem RHS-design-peri-R1 RHS-design-peri-R2s)

4.3 Pre-, peri-, and postconditions

4.3.1 Definitions

abbreviation pre_s ≡ [sock → s true, sock’ → s false, wait → s false]
abbreviation cmt_s ≡ [sock → s true, sock’ → s true, wait → s false]
abbreviation peri_s ≡ [sock → s true, sock’ → s true, wait → s false, wait’ → s true]
abbreviation post_s ≡ [sock → s true, sock’ → s true, wait → s false, wait’ → s false]

abbreviation npre_R(P) ≡ pre_s † P

definition [upred-defs]: pre_R(P) = (¬e npre_R(P))
definition [upred-defs]: cmt_R(P) = R1(cmt_s † P)
definition [upred-defs]: peri_R(P) = R1(peri_s † P)
definition [upred-defs]: post_R(P) = R1(post_s † P)

4.3.2 Unrestriction laws

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

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

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

lemma ok-cmt-unrest [unrest]: $\text{ok} \not\in \text{cmt}_R P$

28
by \((\text{simp add: } \text{cmt}_R\text{-def unrest usubst})\)

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

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

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

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

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

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

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

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

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

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

4.3.3 Substitution laws

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

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

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

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

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

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

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

lemma \(\text{peri}_s\text{-R2c [usubst]}: \text{peri}_s \uparrow R2c(P) = R2c(\text{peri}_s \uparrow P)\)

29
by (simp add: R2c-def R2s-def usubst)

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

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

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

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

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

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

lemma rea-pre-RHS-design: \( \text{pre}_R(R_s(P \vdash Q)) = R1(R2c(\text{pre}_s \uparrow P)) \)
by (simp add: RHS-def usubst R3h-def pre_s-design R1-negate-R1 R2c-not rea-not-def)

lemma rea-cmt-RHS-design: \( \text{cmt}_R(R_s(P \vdash Q)) = R1(R2c(\text{cmt}_s \uparrow P \Rightarrow Q)) \)
by (simp add: RHS-def usubst cmt_R-def cmt_s-design R1-idem)

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

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

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

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

lemma rdes-export-cmt: \( R_s(P \vdash \text{cmt}_s \uparrow Q) = R_s(P \vdash Q) \)
by (rel-auto)

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

4.3.4 Healthiness laws

lemma wait\-'-unrest-pre-SRD [unrest]:
\( $\text{wait}^\dagger \text{pre}_R(P) \Rightarrow $\text{wait}^\dagger \text{pre}_R (\text{SRD} P) \)
apply (rel-auto)
using least-zero apply blast+
done

lemma R1-R2s-cmt-SRD:
assumes \( P \) is \( \text{SRD} \)
shows $R_1(R_2s(cmt_R(P))) = cmt_R(P)$
  by (metis (no-types, lifting) R1-R2c-commute R1-R2s-R2c R1-idem R2c-idem SRD-reactive-design
  assms rea-cmt-RHS-design)

lemma R1-R2s-peri-SRD:
  assumes $P$ is SRD
  shows $R_1(R_2s(peri_R(P))) = peri_R(P)$
  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_R'-R1 peri_R'-R2c)

lemma R1-peri-SRD:
  assumes $P$ is SRD
  shows $R_1(peri_R(P)) = peri_R(P)$
  proof
    have $R_1(peri_R(P)) = R_1(R_1(R_2s(peri_R(P))))$
      by (simp add: R1-R2s-peri-SRD assms)
    also have ... = peri_R(P)
      by (simp add: R1-idem, simp add: R1-R2s-peri-SRD assms)
    finally show \(?thesis\).
  qed

lemma periR-SRD-R1 [closure]: $P$ is SRD $\Rightarrow$ peri_R(P) is R1
  by (simp add: Healthy-def' R1-peri-SRD)

lemma R1-R2c-peri-RHS:
  assumes $P$ is SRD
  shows $R_1(R_2c(peri_R(P))) = peri_R(P)$
  by (metis R1-R2s-commute R1-R2c-R2c R1-R2s-R2c R1-R2s-peri-SRD assms)

lemma R1-R2s-post-SRD:
  assumes $P$ is SRD
  shows $R_1(R_2s(post_R(P))) = post_R(P)$
  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_R'-R1 post_R'-R2c)

lemma R2c-peri-SRD:
  assumes $P$ is SRD
  shows $R_2c(peri_R(P)) = peri_R(P)$
  by (metis R1-R2c-commute R1-R2c-R2c R1-R2s-R2c R1-R2s-post-SRD R1-post-SRD assms)

lemma R1-post-SRD:
  assumes $P$ is SRD
  shows $R_1(post_R(P)) = post_R(P)$
  proof
    have $R_1(post_R(P)) = R_1(R_1(R_2s(post_R(P))))$
      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 $R_2c(post_R(P)) = post_R(P)$
  by (metis R1-R2c-commute R1-R2s-R2c R1-R2s-post-SRD R1-post-SRD assms)
lemma postR-SRD-R1 [closure]: $P$ is SRD $\implies$ $\text{post}_R(P)$ is R1
  by (simp add: Healthy-def’ R1-post-SRD)

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

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

lemma R2c-preR [closure]: $P$ is SRD $\implies$ $\text{pre}_R(P)$ is R2c
  by (simp add: Healthy-def R2c-preR)

lemma R2c-postR [closure]: $P$ is SRD $\implies$ $\text{post}_R(P)$ is R2c
  by (metis R1-R2c-commute R1-R2s-R2c R1-R2s-post-SRD R2c-idem)

lemma preR-R2c-closed [closure]: $P$ is SRD $\implies$ $\text{pre}_R(P)$ is R2c
  by (simp add: Healthy-def R2c-preR)

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

lemma postR-RR [closure]: $P$ is SRD $\implies$ $\text{post}_R(P)$ is RR
  by (rule RR-intro, simp-all add: closure unrest)

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

lemma R1-preR [closure]:
  $\text{pre}_R(P)$ is R1
  by (rel-auto)

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

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

32
lemma trace-ident-right-postR:
\[ post_R(P) \triangleq (\$tr' =_u \$tr \land [II]_R) = post_R(P) \]
by (rel-auto)

lemma preR-R2-closed [closure]: P is SRD \implies pre_R(P) is R2
by (simp add: R2-comp-def Healthy-comp closure)

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

lemma postR-R2-closed [closure]: P is SRD \implies 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(Rs(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(Rs(P \vdash Q \circ R)) = (P \Rightarrow R Q)
by (simp add: rea-peri-RHS-design unrest usubst assms Healthy-if RR-implies-R2c closure)

lemma postR-rdes [rdes]:
assumes P is RR R is RR
shows post_R(Rs(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) = true_r
by (rel-auto)
\begin{align*}
\text{lemma} & \hspace{1em} \text{periR-srdes-skip \ [rdes]}: \ peri_R(H_R) = \text{false} \\
& \hspace{1em} \text{by} \ (\text{rel-auto})
\end{align*}

\begin{align*}
\text{lemma} & \hspace{1em} \text{postR-srdes-skip \ [rdes]}: \ post_R(H_R) = (\$tr' = u \ S \ t \ w) \\
& \hspace{1em} \text{by} \ (\text{rel-auto})
\end{align*}

\begin{align*}
\text{lemma} & \hspace{1em} \text{preR-INF \ [rdes]}: A \neq \{\} \implies pre_R(\bigcap A) = (\bigwedge P \in A \cdot pre_R(P)) \\
& \hspace{1em} \text{by} \ (\text{rel-auto})
\end{align*}

\begin{align*}
\text{lemma} & \hspace{1em} \text{periR-INF \ [rdes]}: peri_R(\bigcap A) = (\bigvee P \in A \cdot peri_R(P)) \\
& \hspace{1em} \text{by} \ (\text{rel-auto})
\end{align*}

\begin{align*}
\text{lemma} & \hspace{1em} \text{postR-INF \ [rdes]}: post_R(\bigcap A) = (\bigvee P \in A \cdot post_R(P)) \\
& \hspace{1em} \text{by} \ (\text{rel-auto})
\end{align*}

\begin{align*}
\text{lemma} & \hspace{1em} \text{preR-UINF \ [rdes]}: pre_R(\bigcap i \cdot P(i)) = (\bigcup i \cdot pre_R(P(i))) \\
& \hspace{1em} \text{by} \ (\text{rel-auto})
\end{align*}

\begin{align*}
\text{lemma} & \hspace{1em} \text{periR-UINF \ [rdes]}: peri_R(\bigcap i \cdot P(i)) = (\bigcap i \cdot peri_R(P(i))) \\
& \hspace{1em} \text{by} \ (\text{rel-auto})
\end{align*}

\begin{align*}
\text{lemma} & \hspace{1em} \text{postR-UINF \ [rdes]}: post_R(\bigcap i \cdot P(i)) = (\bigcap i \cdot post_R(P(i))) \\
& \hspace{1em} \text{by} \ (\text{rel-auto})
\end{align*}

\begin{align*}
\text{lemma} & \hspace{1em} \text{preR-UINF-member \ [rdes]}: A \neq \{\} \implies pre_R(\bigcap i \in A \cdot P(i)) = (\bigcup i \in A \cdot pre_R(P(i))) \\
& \hspace{1em} \text{by} \ (\text{rel-auto})
\end{align*}

\begin{align*}
\text{lemma} & \hspace{1em} \text{preR-UINF-member-2 \ [rdes]}: A \neq \{\} \implies pre_R(\bigcap (i,j) \in A \cdot P \ i \ j) = (\bigcup (i,j) \in A \cdot pre_R(P \ i \ j)) \\
& \hspace{1em} \text{by} \ (\text{rel-auto})
\end{align*}

\begin{align*}
\text{lemma} & \hspace{1em} \text{preR-UINF-member-3 \ [rdes]}: A \neq \{\} \implies pre_R(\bigcap (i,j,k) \in A \cdot P \ i \ j \ k) = (\bigcup (i,j,k) \in A \cdot pre_R(P \ i \ j \ k)) \\
& \hspace{1em} \text{by} \ (\text{rel-auto})
\end{align*}

\begin{align*}
\text{lemma} & \hspace{1em} \text{periR-UINF-member \ [rdes]}: peri_R(\bigcap i \in A \cdot P(i)) = (\bigcap i \in A \cdot peri_R(P(i))) \\
& \hspace{1em} \text{by} \ (\text{rel-auto})
\end{align*}

\begin{align*}
\text{lemma} & \hspace{1em} \text{periR-UINF-member-2 \ [rdes]}: peri_R(\bigcap (i,j) \in A \cdot P \ i \ j) = (\bigcap (i,j) \in A \cdot peri_R(P \ i \ j)) \\
& \hspace{1em} \text{by} \ (\text{rel-auto})
\end{align*}

\begin{align*}
\text{lemma} & \hspace{1em} \text{periR-UINF-member-3 \ [rdes]}: peri_R(\bigcap (i,j,k) \in A \cdot P \ i \ j \ k) = (\bigcap (i,j,k) \in A \cdot peri_R(P \ i \ j \ k)) \\
& \hspace{1em} \text{by} \ (\text{rel-auto})
\end{align*}

\begin{align*}
\text{lemma} & \hspace{1em} \text{postR-UINF-member \ [rdes]}: post_R(\bigcap i \in A \cdot P(i)) = (\bigcap i \in A \cdot post_R(P(i))) \\
& \hspace{1em} \text{by} \ (\text{rel-auto})
\end{align*}

\begin{align*}
\text{lemma} & \hspace{1em} \text{postR-UINF-member-2 \ [rdes]}: post_R(\bigcap (i,j) \in A \cdot P \ i \ j) = (\bigcap (i,j) \in A \cdot post_R(P \ i \ j)) \\
& \hspace{1em} \text{by} \ (\text{rel-auto})
\end{align*}

\begin{align*}
\text{lemma} & \hspace{1em} \text{postR-UINF-member-3 \ [rdes]}: post_R(\bigcap (i,j,k) \in A \cdot P \ i \ j \ k) = (\bigcap (i,j,k) \in A \cdot post_R(P \ i \ j \ k)) \\
& \hspace{1em} \text{by} \ (\text{rel-auto})
\end{align*}

\begin{align*}
\text{lemma} & \hspace{1em} \text{preR-inf \ [rdes]}: pre_R(P \cap Q) = (pre_R(P) \land pre_R(Q)) \\
& \hspace{1em} \text{by} \ (\text{rel-auto})
\end{align*}
4.4 Formation laws

lemma srdes-skip-tri-design [rdes-def]: $I_R = R_s(true_r \vdash false \circ II_r)$
by (simp add: srdes-skip-def, rel-auto)

lemma Chaos-tri-def [rdes-def]: $Chaos = R_s(false \vdash false \circ false)$
by (simp add: Chaos-def design-false-pre)

lemma Miracle-tri-def [rdes-def]: $Miracle = R_s(true_r \vdash false \circ false)$
by (simp add: Miracle-def R1-design-R1-pre wait'-cond-idem)

lemma RHS-tri-design-form:
assumes $P_1$ is RR $P_2$ is RR $P_3$ is RR
shows $R_s(P_1 \vdash P_2 \circ P_3) = (I_R \circ \$wait \triangleright ((\$ok \land P_1) \Rightarrow_r (\$ok' \land (P_2 \circ P_3))))$
proof
  - have $R_s(RR(P_1) \vdash RR(P_2) \circ RR(P_3)) = (I_R \circ \$wait \triangleright ((\$ok \land RR(P_1)) \Rightarrow_r (\$ok' \land (RR(P_2) \circ RR(P_3))))))$
    apply (rel-auto) using minus-zero-eq by blast
  thus $?thesis$
    by (simp add: Healthy-if assms)
qed

lemma RHS-design-pre-post-form:
$R_s((\neg P_f) \vdash P^*_f) = R_s(pre_R(P) \vdash cmt_R(P))$
proof
  - have $R_s((\neg P_f) \vdash P^*_f) = R_s((\neg P_f)[true/\$ok] \vdash P^*_f[true/\$ok])$
    by (simp add: design-subst-ok)
  also have $... = R_s(pre_R(P) \vdash cmt_R(P))$
    by (simp add: pre_R-def cmt_R-def usubst, rel-auto)
finally show $?thesis$.
qed

lemma SRD-as-reactive-design:
$SRD(P) = R_s(pre_R(P) \vdash cmt_R(P))$
by (simp add: RHS-design-pre-post-form SRD-RH-design-form)

lemma SRD-reactive-design-alt:
assumes $P$ is SRD
shows $R_s(pre_R(P) \vdash cmt_R(P)) = P$
proof
  - have $R_s(pre_R(P) \vdash cmt_R(P)) = R_s((\neg P_f) \vdash P^*_f)$
lemma SRD-reactive-tri-design-lemma:
SRD(P) = \textstyle R_s(\neg P^f \triangleright P^f [\text{true}] \circ P^f [\text{false}])
by (simp add: SRD-RH-design-form wait'-cond-split)

lemma SRD-as-reactive-tri-design:
SRD(P) = \textstyle R_s(pre_R(P) \triangleright peri_R(P) \circ post_R(P))
proof –
have SRD(P) = \textstyle R_s(\neg P^f \triangleright P^f [\text{true}] \circ P^f [\text{false}])
by (simp add: SRD-RH-design-form wait'-cond-split)
also have ... = \textstyle R_s(pre_R(P) \triangleright peri_R(P) \circ 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 \textit{thesis}.
qed

lemma SRD-reactive-tri-design:
assumes \text{P is SRD}
shows \textstyle R_s(pre_R(P) \triangleright peri_R(P) \circ post_R(P)) = P
by (metis Healthy-if SRD-as-reactive-tri-design assms)

lemma SRD-elim [RD-elim]: [ P is SRD; Q(R_s(pre_R(P) \triangleright peri_R(P) \circ post_R(P))) ] \implies Q(P)
by (simp add: SRD-reactive-tri-design)

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

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

lemma USUP-R1-R2s-cmt-SRD:
assumes \text{A \subseteq \{SRD\}_H}
shows (\bigsqcup P \in A \cdot R_1(R_2s(cmt_R P))) = (\bigsqcup P \in A \cdot cmt_R P)
by (rule USUP-cong[of A], metis (mono-tags, lifting) Ball-Collect R1-R2s-cmt-SRD assms)

lemma UINF-R1-R2s-cmt-SRD:
assumes \text{A \subseteq \{SRD\}_H}
shows (\bigsqcap P \in A \cdot R_1(R_2s(cmt_R P))) = (\bigsqcap P \in A \cdot cmt_R P)
by (rule UINF-cong[of A], metis (mono-tags, lifting) Ball-Collect R1-R2s-cmt-SRD assms)

4.4.1 Order laws

lemma preR-antitone: \text{P \subseteq Q \implies pre_R(Q) \subseteq pre_R(P)}
by (rel-auto)

36
lemma periR-monotone: $P \subseteq Q \Rightarrow \peri_R(P) \subseteq \peri_R(Q)$
by (rel-auto)

lemma postR-monotone: $P \subseteq Q \Rightarrow \post_R(P) \subseteq \post_R(Q)$
by (rel-auto)

4.5 Composition laws

theorem RH-tri-design-composition:
assumes $\$ok ` \notin P \$ok ` \notin Q \$ok ` \notin Q_2 \$ok ` \notin R \$ok ` \notin S_1 \$ok ` \notin S_2$
$\$wait ` \notin Q_2 \$wait ` \notin S_1 \$wait ` \notin S_2$
shows $(RH(P \triangleright Q_1 \circ Q_2) ; RH(R \triangleright S_1 \circ S_2)) =$
$RH((\neg (R1 \neg R2s P) ; R1 true) \land \neg ((R1 (R2s Q_2) \land \neg \$wait ` ) ; R1 (\neg R2s R))) \triangleright$
$((Q_1 \triangleright (R1 (R2s Q_2) \circ R1 (R2s S_1))) \circ ((R1 (R2s Q_2) ; R1 (R2s S_2))))$
proof -
have 1:$(\neg ((R1 (R2s (Q_1 \circ Q_2) \land \neg \$wait ` ) ; R1 (\neg R2s R))) =$
$(\neg ((R1 (R2s Q_2) \land \neg \$wait ` ) ; R1 (\neg R2s R)))$
by (metis (no-types, hide-lams) R1-extend-conj R2s-conj R2s-not R2s-wait ` wait `-cond-false)

have 2: $(R1 (R2s (Q_1 \circ Q_2)) ; ([II]D \& \$wait \triangleright R1 (R2s (S_1 \circ S_2)))) =$
$(R1 (R2s Q_1) \circ (R1 (R2s Q_2) ; R1 (R2s S_1))) \circ (R1 (R2s Q_2) ; R1 (R2s S_2))$

proof -
have $(R1 (R2s Q_1)) ; ((\$wait \land ([II]D \& \$wait \triangleright R1 (R2s S_1) \circ R1 (R2s S_2)))) =$
$(R1 (R2s Q_1)) ; ((\$wait \land ([II]D))$
by (rel-auto)
also have \ldots = $(R1 (R2s Q_1)) ; ([II]D) \land \$wait `)$
by (rel-auto)
also from assms(2) have \ldots = $(R1 (R2s Q_1)) \land \$wait `)$
by (simp add: lift-des-skip-dr-unit-rest unrest)
finally show \ldots thesis .

qed

moreover have $(R1 (R2s Q_2)) ; ((\neg \$wait \land ([II]D \& \$wait \triangleright R1 (R2s S_1) \circ R1 (R2s S_2)))) =$
$(R1 (R2s Q_2)) ; ((\neg \$wait \land ([II]D) \circ R1 (R2s S_1) \circ R1 (R2s S_2))\))$
by (metis (no-types, lifting) cond-def conj-disj-not-abs utp-pred-laws.double-compl utp-pred-laws.inf.left-idem utp-pred-laws.sup-assoc utp-pred-laws.sup-inf-absorb)

also have \ldots = $(R1 (R2s Q_2))[false/$\$wait `] ; (R1 (R2s S_1) \circ R1 (R2s S_2))[false/$\$wait `]$
by (metis false-alt-def seq-right-one-point upred-eq-false wait-vwb-lens)

also have \ldots = $(R1 (R2s Q_2)) ; (R1 (R2s S_1) \circ R1 (R2s S_2))$
by (simp add: wait `-cond-def subst unrest assms)
finally show \ldots thesis .

qed

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


ultimately show ?thesis
  by (simp add: R2s-wait'-cond R1-wait'-cond wait'-cond-seq)
qed

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

theorem R1-design-composition-RR:
  assumes P is RR Q is RR R is RR S is RR
  shows (R1(P ⊃ Q) ⊃ R1(R ⊃ S)) = R1(P ∧ Q wp_r R) ⊃ (Q ;; S))
  apply (subst R1-design-composition)
  apply (simp-all add: assms unrest wp-rea-def Healthy-if closure)
  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
  $wait ⋁ R $wait ⋁ Q2 $wait ⋁ S1 $wait ⋁ S2
  shows (R_r(P ⊃ Q_r) ⊃ R_r(R ⊃ S)) =
    R_r(¬ (R1 (¬ R2s P) ⊃ R1 (¬ R2s Q) ⋀ ¬ (R1(R2s Q_r) ⊃ R1 (¬ R2s R))) ⊃
      ((∃ $st' ⋁ R1 (R2s Q_r) ⋀ R1 (R2s S_r) ⋀ R1 (R2s S_r)) ⊃
        (R1 (R2s Q_r) ⋀ R1 (R2s S_r) ⋀ R1 (R2s S_r))))
proof –
  have 1:¬ ((R1 (R2s (Q1 ⋀ Q2)) ⋀ ¬ $wait') ⋀ R1 (¬ R2s R)) =
    (¬ ((R1 (R2s Q_r) ⋀ ¬ $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)))) =
    ((∃ $st' ⋁ R1 (R2s Q_r) ⋀ R1 (R2s S_r)) ⋀ (R1 (R2s Q_r) ⋀ R1 (R2s S_r) ⋀ R1 (R2s S_r)))
    by (rel-auto, blast+)
proof –
  have (R1 (R2s Q_r) ⋀ ($wait ∧ ((∃ $st ⋁ [IH]D) < $wait > R1 (R2s S_r) ⋀ R1 (R2s S_r))))
    = (∃ $st' ⋁ R1 (R2s Q_r) ⋀ $wait')
    by (rel-auto, blast+)
  also have ... = (R1 (R2s Q_r) ⋀ ($wait ∧ ((∃ $st ⋁ [IH]D) < $wait > R1 (R2s S_r) ⋀ R1 (R2s S_r))))
  also have ... = (R1 (R2s Q_r) ⋀ ($wait ∧ ((∃ $st ⋁ [IH]D) < $wait > R1 (R2s S_r) ⋀ R1 (R2s S_r))))
  also from assms(2) have ... = (∃ $st' ⋁ R1 (R2s Q_r) ⋀ $wait')
    by (rel-auto, blast+)
finally show ?thesis .
qed
moreover have \((R1 \ (R2s \ Q2) :: (\neg \ $wait \wedge (\exists \ $st \cdot [I]_D) \wedge \ $wait \gg R1 \ (R2s \ S1) \gg R1 \ (R2s \ S2)))\)

\[ = ((R1 \ (R2s \ Q2)) :: (R1 \ (R2s \ S1) \gg R1 \ (R2s \ S2))) \]

proof

have \((R1 \ (R2s \ Q2) :: (\neg \ $wait \wedge (\exists \ $st \cdot [I]_D) \wedge \ $wait \gg R1 \ (R2s \ S1) \gg R1 \ (R2s \ S2)))\)

\[ = (R1 \ (R2s \ Q2)) :: (\neg \ $wait \wedge (R1 \ (R2s \ S1) \gg R1 \ (R2s \ S2))) \]

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

also have \(\ldots = ((R1 \ (R2s \ Q2))[false/$wait]\ :: (R1 \ (R2s \ S1) \gg R1 \ (R2s \ S2))[false/$wait])\)

by (metis false-alt-def seqp-right-one-point upred-eq-false wait-vw-lens)

also have \(\ldots = ((R1 \ (R2s \ Q2)) :: (R1 \ (R2s \ S1) \gg R1 \ (R2s \ S2)))\)

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

finally show \(?thesis \).

qed

moreover have \(((R1 \ (R2s \ Q1) \wedge \ $wait') \vee ((R1 \ (R2s \ Q2)) :: (R1 \ (R2s \ S1) \gg R1 \ (R2s \ S2)))\)

\[ = (R1 \ (R2s \ Q1) \vee (R1 \ (R2s \ Q2) :: R1 \ (R2s \ S1)) \gg ((R1 \ (R2s \ Q2) :: R1 \ (R2s \ S2))) \]

by (simp add: wait-cond-def cond-seq-right-distr cond-and-T-integrate unrest)

ultimately show \(?thesis\)

by (simp add: R2s-wait-cond R1-wait-cond wait-cond-seq ex-conj-contr-right unrest)

(simp add: cond-and-T-integrate cond-seq-right-distr unrest-var wait-cond-def)

qed

from assms(7,8) have \(3: (R1 \ (R2s \ Q2) \wedge \neg \ $wait') :: R1 \ (\neg R2s \ R) = R1 \ (R2s \ Q2) :: R1 \ (\neg R2s \ R)\)

by (rel-auto, blast, meson)

show \(?thesis\)

apply (subst RHS-design-composition)

apply (simp-all add: assms)

apply (simp add: assms wait-cond-def unrest)

apply (simp add: assms wait-cond-def unrest)

apply (simp add: 1 2 3)

apply (simp add: R1-R2s-R2c RHS-design-lemma1)

apply (metis R1-R2c-ex-conj RHS-design-lemma1)

done

qed

theorem RHS-tri-design-composition-wp:

assumes $\text{sok}' \equiv P \, $ok' \equiv Q1 \, $ok \equiv Q2 \, $ok \equiv R \, $ok \equiv S1 \, $ok \equiv S2

\$wait' \equiv R \, $wait' \equiv Q2 \, $wait' \equiv S1 \, $wait' \equiv S2

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

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

shows $R, (P \vdash Q1 \circ Q2) :: R, (R \vdash S1 \circ S2) =$

$R, ((\neg \ P) \ wp_r \ false \wedge Q2 \ wp_r \ R) \vdash ((\exists \ $st' \cdot Q1) \cap (Q2 :: S1)) \circ (Q2 :: S2)) \ (is \ ?lhs = \ ?rhs)\)

proof

have \(?lhs = R, ((\neg \ R1 \ (\neg P)) :: R1 true \wedge \neg Q2 :: R1 (\neg R) ) \vdash ((\exists \ $st' \cdot Q1) \cap Q2 :: S1) \circ Q2 :: S2)\)
by (simp add: RHS-tri-design-composition assms Healthy-if R2c-healthy-R2s disb-upred-def) (metis (no-types, hide-lams) R1-negate-R1 R2c-healthy-R2s assms(11,16))
also have ... = \?rhs
  by (rel-auto)
finally show \?thesis .
qed

lemma RHS-tri-normal-design-composition:
assumes
  \$ok' \notin P \$ok' \notin Q_1 \$ok' \notin Q_2 \$ok' \notin R \$ok' \notin S_1 \$ok' \notin S_2
  \$wait' \notin R \$wait' \notin Q_2 \$wait' \notin S_1 \$wait' \notin S_2
P is R2c Q_1 is R1 Q_1 is R2c Q_2 is R1 Q_2 is R2c
R is R2c S_1 is R1 S_1 is R2c S_2 is R1 S_2 is R2c
R1 (¬ P) ;; R1(true) = R1 (¬ P) $st' \notin Q_1
shows R_s(P \vdash Q_1 \circ Q_2) ;; R_s(R \vdash S_1 \circ S_2) = R_s((R1 (¬ P) wp, false \land Q_2 wp, R) \vdash (\exists$st' \cdot Q_1) \cap (Q_2 ;; S_1)) \circ (Q_2 ;; S_2))
proof
  have R_s(P \vdash Q_1 \circ Q_2) ;; R_s(R \vdash S_1 \circ S_2) = R_s(R1 (¬ P) wp, R) \vdash (Q_1 \lor (Q_2 ;; S_1)) \circ (Q_2 ;; S_2))
  by (simp-all add: RHS-tri-design-composition-RR-up ad conj assms unrest RR-implies-R2c)
also have ... = R_s((P \land Q_2 wp, R) \vdash (Q_1 \lor (Q_2 ;; S_1)) \circ (Q_2 ;; S_2))
  by (simp add: assms wp-re-def ex-unrest, rel-auto)
finally show \?thesis .
qed

lemma RHS-tri-normal-design-composition'[rdes-def]:
assumes P is RC Q_1 is RR $st' \notin Q_1 Q_2 is RR R is RR S_1 is RR S_2 is RR
shows R_s(P \vdash Q_1 \circ Q_2) ;; R_s(R \vdash S_1 \circ S_2) = R_s((P \land Q_2 wp, R) \vdash (Q_1 \lor (Q_2 ;; S_1)) \circ (Q_2 ;; S_2))
proof
  have R1 (¬ P) ;; R1 true = R1(¬ P)
    using RC-implies-RC1[OF assms(1)]
  by (simp add: Healthy-def RC1-when-def rea-not-def)
    (metis R1-negate-R1 R1-seqr wp-pred-laws.double-compl)
thus \?thesis
  by (simp add: RHS-tri-normal-design-composition assms closure unrest RR-implies-R2c)
qed

lemma RHS-tri-design-right-unit-lemma:
assumes $ok' \notin P $ok' \notin Q $ok' \notin R $wait' \notin R
shows R_s(P \vdash Q \circ R) ;; H_R = R_s((\neg_P (\neg_P ;; true_r) \vdash (\exists$st' \cdot Q) \cap R))
proof
  have R_s(P \vdash Q \circ R) ;; H_R = R_s(P \vdash Q \circ R) ;; R_s(true_r \vdash false \circ ($tr' = u$tr \land [H_R])
    by (simp add: srdes-skp-tri-design, rel-auto)
also have ... = R_s((\neg(R1 (¬ R2s P) ;; R1(true) \vdash (\exists$st' \cdot Q) \cap (R1 (R2s R) ;; R1 (R2s ($tr' = u$tr \land [H_R])))))
    by (simp-all add: RHS-tri-design-composition assms unrest R2s-true R1-false R2s-false)
also have ... = \text{R}_s((\neg R_1 (\neg R_2s P) ;; R_1 \text{true}) \vdash (\exists \text{st} \cdot Q) \circ R_1 (R_2s R))

proof –
from \text{assms}(3,4) have \((R_1 (R_2s R) ;; R_1 (R_2s (\text{st} = u \circ (\text{IH} R_1))) = R_1 (R_2s R))
by (rel-auto, \text{metis (no-types, lifting)} \text{minus-zero-eq, meson order-refl trace-class.diff-cancel})

thus \text{thesis}.
by \text{simp}

qed

also have ... = \text{R}_s((\neg (\neg P) ;; R_1 \text{true}) \vdash (\exists \text{st} \cdot Q) \circ R))
by (\text{metis (no-types, lifting)} \text{R1-R2s-R1-true-lemma R1-R2s-R2c R2c-not RHS-design-R2c-pre RHS-design-post-R1 RHS-design-post-R2s)

also have ... = \text{R}_s((\neg_r (\neg_r P) ;; \text{true}_r) \vdash (\exists \text{st} \cdot Q) \circ R))
by (rel-auto)

finally show \text{thesis}.

qed

\text{lemma SRD-composition-up:}
\text{assumes P is SRD Q is SRD}
\text{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{pre}_r Q) \vdash}
\((\exists \text{st} \cdot \text{peri}_r P) \lor (\text{post}_r P \circ \text{peri}_r Q)) \circ (\text{post}_r P ;; \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 (\text{simp add: SRD-reactive-tri-design \text{assms}(1) \text{assms}(2)})

also from \text{assms}

have ... = \text{rhs}
by (\text{simp add: \text{RHS-tri-design-composition-up disj-upred-def unrest \text{assms closure}})

finally show \text{thesis}.

qed

4.6 Refinement introduction laws

\text{lemma RHS-tri-design-refine:}
\text{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}
\text{shows \text{R}_s(P_1 \vdash P_2 \circ P_3) \subseteq \text{R}_s(Q_1 \vdash Q_2 \circ Q_3) \longleftrightarrow \text{r_1} \circ Q_1 \cdot \text{r_2} \circ Q_2 \Rightarrow P_2 \circ P_3 \cdot \text{r_1} \circ Q_3 \Rightarrow P_3}
(is \text{lhs} = \text{rhs})

proof –

have \text{lhs} \longleftrightarrow \text{r_1} \circ Q_1 \cdot \text{r_2} \circ Q_2 \Rightarrow P_2 \circ P_3 \cdot \text{r_1} \circ Q_3 \Rightarrow P_3
by (\text{simp add: RHS-design-refine \text{assms closure RR-implies-R2c unrest ex-unrest})

also have ... \longleftrightarrow \text{r_1} \circ Q_1 \cdot (\text{r_1} \circ Q_2 \circ (P_1 \circ Q_3)) \Rightarrow P_2 \circ P_3
by (rel-auto)

also have ... \longleftrightarrow \text{r_1} \circ Q_1 \cdot ((P_1 \circ Q_2) \circ (P_1 \circ Q_3) \Rightarrow P_2 \circ P_3)[\text{true}/\text{\$wait}] \cdot \text{r_1} \circ Q_2 \circ (P_1 \circ Q_3) \Rightarrow P_2 \circ P_3)[\text{false}/\text{\$wait}];
by (rel-auto, \text{metis})

also have ... \longleftrightarrow \text{r_1}
by (\text{simp add: \text{usubst unrest \text{assms})}

finally show \text{thesis}.

qed

\text{lemma srdes-tri-refine-intro:}
\text{assumes \text{r_1} \Rightarrow P_2 \cdot \text{r_1} \circ Q_2 \Rightarrow Q_1 \cdot \text{r_1} \circ R_2 \Rightarrow R_1}
\text{shows \text{R}_s(P_1 \vdash Q_1 \circ R_1) \subseteq \text{R}_s(P_2 \vdash Q_2 \circ R_2)}
\text{using \text{assms}}
by (\text{rule-tac srdes-refine-intro, simp-all, rel-auto})

\text{lemma srdes-tri-equ-intro:}
assumes $P_1 = Q_1\ P_2 = Q_2\ P_3 = Q_3$
shows $R_{\mathcal{s}}(P_1 \vdash P_2 \circ P_3) = R_{\mathcal{s}}(Q_1 \vdash Q_2 \circ Q_3)$
using assms by (simp)

lemma $srdes$-tri-refine-intro' :
assumes $P_2 \subseteq P_1\ Q_1 \subseteq (P_1 \land Q_2)\ R_1 \subseteq (P_1 \land R_2)$
shows $R_{\mathcal{s}}(P_1 \vdash Q_1 \circ R_1) \subseteq R_{\mathcal{s}}(P_2 \vdash Q_2 \circ R_2)$
using assms
by (rule-tac $srdes$-tri-refine-intro, simp-all add: refBy-order)

lemma $SRD$-peri-under-pre :
assumes $P$ is $SRD$ $\mathcal{\$wait}' \not\pre R(P)$
shows $(\pre R(P) \Rightarrow\ peri R(P)) = peri R(P)$
proof –
  have $peri R(P) =$
    $peri R(R_{\mathcal{s}}(\pre R(P) \vdash peri R(P) \circ post R(P)))$
    by (simp add: $SRD$-reactive-tri-design assms)
  also have $... = (\pre R P \Rightarrow peri R P)$
    by (simp add: rea-pre-RHS-design rea-peri-RHS-design assms)
  finally show $?thesis ..$
qed

lemma $SRD$-post-under-pre :
assumes $P$ is $SRD$ $\mathcal{\$wait}' \not\pre R(P)$
shows $(\pre R(P) \Rightarrow post R(P)) = post R(P)$
proof –
  have $post R(P) =$
    $post R(R_{\mathcal{s}}(\pre R(P) \vdash peri R(P) \circ post R(P)))$
    by (simp add: $SRD$-reactive-tri-design assms)
  also have $... = (\pre R P \Rightarrow post R P)$
    by (simp add: rea-post-RHS-design rea-post-RHS-design assms)
  finally show $?thesis ..$
qed

lemma $SRD$-refine-intro :
assumes $P$ is $SRD$ $Q$ is $SRD$
  $\'pre R(P) \Rightarrow pre R(Q)'\ 'pre R(P) \land peri R(Q) \Rightarrow peri R(P)'\ 'pre R(P) \land post R(Q) \Rightarrow post R(P)'$
shows $P \subseteq Q$
by (metis $SRD$-reactive-tri-design assms(1) assms(2) assms(3) assms(4) assms(5) $srdes$-tri-refine-intro)

lemma $SRD$-refine-intro' :
assumes $P$ is $SRD$ $Q$ is $SRD$
  $\'pre R(P) \Rightarrow pre R(Q)'\ peri R(P) \subseteq (\pre R(P) \land peri R(Q))\ post R(P) \subseteq (\pre R(P) \land post R(Q))$
shows $P \subseteq Q$
using assms by (rule-tac $SRD$-refine-intro, simp-all add: refBy-order)

lemma $SRD$-eq-intro :
assumes $P$ is $SRD$ $Q$ is $SRD$ $pre R(P) = pre R(Q)$
  $peri R(P) = peri R(Q)$ $post R(P) = post R(Q)$
shows $P = Q$
by (metis $SRD$-reactive-tri-design assms)
4.7 Closure laws

**lemma SRD-srdes-skip [closure]:** 
\[ \Pi_R \text{ is SRD by } (\text{simp add: srdes-skip-def RHS-design-is-SRD unrest}) \]

**lemma SRD-seqr-closure [closure]:**
assumes \( P \text{ is SRD} \) \( Q \text{ is SRD} \)
shows \( (P \sqcap Q) \text{ is SRD} \)
proof –
  have \( (P \sqcap Q) = R_s (\Pi_R P) \sqsubseteq \Pi_R Q) \)
  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 \( ?\text{thesis} \).
qed

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

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

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

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

4.8 Distribution laws

**lemma RHS-tri-design-choice [rdes-def]:**
\[ R_s (P_1 \vdash P_2 \circ Q_3) \sqsubseteq R_s (Q_1 \vdash Q_2 \circ Q_3) = R_s ((P_1 \land Q_1) \vdash (P_2 \lor Q_2) \circ (P_3 \lor Q_3)) \]
apply (simp add: RHS-design-choice)
apply (rule cong[of \( R_s, R_s \)])
apply (simp)
don
\[ \text{R} \cdot (P_1 \supset P_2 \land P_3) \cup \text{R} \cdot ((Q_1 \supset Q_2 \land Q_3) = \text{R} \cdot ((P_1 \lor Q_1) \supset ((P_1 \Rightarrow P_2) \land (Q_1 \Rightarrow Q_2)) \land ((P_1 \Rightarrow P_3) \land (Q_1 \Rightarrow Q_3))) \]
by (simp add: RHS-design-sup, rel-auto)

**lemma** RHS-tri-design-conj [rdes-def]:
\[ (\text{R} \cdot (P_1 \supset P_2 \land P_3) \cup \text{R} \cdot ((Q_1 \supset Q_2 \land Q_3) = \text{R} \cdot ((P_1 \lor Q_1) \supset ((P_1 \Rightarrow P_2) \land (Q_1 \Rightarrow Q_2)) \land ((P_1 \Rightarrow P_3) \land (Q_1 \Rightarrow Q_3))) \]
by (simp add: RHS-design-sup conj-upred-def)

**lemma** SRD-UINF [rdes-def]:
\[ \text{assumes } A \neq \{\} \land A \subseteq [S_R] \]
\[ \text{shows } \bigwedge A = \text{R} \cdot (\bigwedge \text{P} \cdot \text{pre}_R(\bigwedge A) \supset \text{peri}_R(\bigwedge A) \land \text{post}_R(\bigwedge A)) \]
\[ \text{by (metis SRD-as-reactive-tri-design assms srdes-hcond-def srdes-theory-continuous,healthy-inf srdes-theory-continuous,healthy-inf-def)} \]
\[ \text{also have } \bigwedge = \text{R} \cdot (\bigwedge \text{P} \cdot \text{pre}_R(\bigwedge A) \supset \text{peri}_R(\bigwedge A) \land \text{post}_R(\bigwedge A)) \]
\[ \text{by (simp add: preR-INF periR-INF postR-INF assms)} \]
\[ \text{finally show } \text{thesis} \]

**qed**

**lemma** RHS-tri-design-USUP [rdes-def]:
\[ \text{assumes } A \neq \{\} \land \text{P} i \text{ is SRD} \]
\[ \text{shows } \bigwedge A = \text{R} \cdot (\bigwedge \text{P} \cdot \text{pre}_R(\bigwedge A) \supset \text{peri}_R(\bigwedge A) \land \text{post}_R(\bigwedge A)) \]
\[ \text{by (subst RHS-INF[OF assms, THEN sym], simp add: design-UINF-mem assms, rel-auto)} \]

**lemma** SRD-UINF-mem:
\[ \text{assumes } A \neq \{\} \land \text{P} i \text{ is SRD} \]
\[ \text{shows } \bigwedge A = \text{R} \cdot (\bigwedge \text{P} \cdot \text{pre}_R(\bigwedge A) \supset \text{peri}_R(\bigwedge A) \land \text{post}_R(\bigwedge A)) \]
\[ \text{(is } \text{lhs} = \text{rhs}) \]
\[ \text{by (subst rdes-def, simp-all add: assms image-subsetI)} \]
\[ \text{also have } \bigwedge = \text{R} \cdot (\bigwedge \text{P} \cdot \text{pre}_R(\bigwedge A) \supset \text{peri}_R(\bigwedge A) \land \text{post}_R(\bigwedge A)) \]
\[ \text{by (rel-auto)} \]
\[ \text{finally show } \text{thesis} \]

**qed**

**lemma** RHS-tri-design-UINF-ind [rdes-def]:
\[ (\bigwedge i \cdot \text{R} \cdot (P_1(i) \supset P_2(i) \land P_3(i)) = \text{R} \cdot ((\bigwedge i \cdot P_1(i) \supset (\bigwedge i \cdot P_2(i)) \land (\bigwedge i \cdot P_3(i)) \]
\[ \text{by (rel-auto)} \]

**lemma** cond-srea-form [rdes-def]:
\[ \text{R} \cdot (P \supset Q_1 \land Q_2) \otimes b \land R \cdot (R \supset S_1 \land S_2) = \]
\[ \text{R} \cdot ((P \land b \land R) \supset ((Q_1 \land b \land R) \supset (Q_2 \land b \land R))) \]
\[ \text{by (pred-auto)} \]
\[ \text{also have } \bigwedge = \text{R} \cdot (P \supset Q_1 \land Q_2) \otimes b \land R \supset S_1 \supset S_2 \]
\[ \text{by (simp add: RHS-cond lift-cond-srea-def)} \]

44
also have \( \ldots = R_s((P \triangleleft b \triangleright R) \vdash (Q_1 \triangleright b \triangleleft R S_1 \triangleright S_2)) \)
by (simp add: design-condr lift-cond-srea-def)
also have \( \ldots = R_s((P \triangleleft b \triangleright R) \vdash (Q_1 \triangleright b \triangleleft R S_1) \triangleright (Q_2 \triangleleft b \triangleright R S_2)) \)
by (rule cong[of \( R_s \) \( \vdash R_s \)], simp, rel-auto)
finally show \(?thesis\).

**4.9 Algebraic laws**

**lemma SRD-left-unit:**
assumes \( P \) is SRD
shows \( \mathtt{I}_R \triangleright P \)
by (simp add: SRD-composition-up closure rdsl wp \( C1 \) \( R1\text{-negate-R1} \) \( R1\text{-false} \) rpred trace-ident-left-periR trace-ident-left-postR SRD-reactive-tri-design assms)

**lemma skip-srea-self-unit [simp]:**
\( \mathtt{I}_R \triangleright \mathtt{I}_R = \mathtt{I}_R \)
by (simp add: SRD-left-unit closure)

**lemma SRD-right-unit-tri-lemma:**
assumes \( P \) is SRD
shows \( \mathtt{I}_R \triangleright R_s((\neg \text{pre}_R P) \vdash \text{peri}_R P \triangleright \text{post}_R P) \)
by (simp add: SRD-reactive-tri-design assms)

**lemma Miracle-left-zero:**
assumes \( P \) is SRD
shows \( \text{Miracle} \triangleright P \)
by (simp add: Miracle-def SRD-reactive-design-alt assms)

**lemma Chaos-left-zero:**
assumes \( P \) is SRD
shows \( \text{Chaos} \triangleright P \)
by (simp add: RHS-design-composition unrest R1-false R2s-false R2s-true)

**qed**
lemma SRD-right-Chaos tri-lemma:
  assumes \( P \) is SRD
  shows \( P \);; Chaos = \( R_s (\neg \gamma \pre_R P \wp r \false \and \post_R P \wp r \false) \vdash (\exists \cdot \peri_R P) \circ \false \)
  by (simp add: SRD-composition-wp closure rdes assms wp, rel-auto)

lemma SRD-right-Miracle-tri-lemma:
  assumes \( P \) is SRD
  shows \( P \);; Miracle = \( R_s (\neg \gamma \pre_R P \wp r \false \and \post_R P \wp r \false) \vdash (\exists \cdot \peri_R P) \circ \false \)
  by (simp add: SRD-composition-wp closure rdes assms wp, rel-auto)

Stateful reactive designs are left unital

overloading
  srdes-unit :: (SRDES, ('s,'t::trace,'a) uthy) => ('s,'t,'a) hrel-rsp

begin
definition srdes-unit :: (SRDES, ('s,'t::trace,'a) uthy) => ('s,'t,'a) hrel-rsp where
  srdes-unit = II
end

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

4.10 Recursion laws

lemma mono-srd-iter:
  assumes mono \( F \cdot F \in [\text{SRD}]_H \to [\text{SRD}]_H \)
  shows mono \( \lambda X. R_s (\pre_R (F X) \vdash \peri_R (F X) \circ \post_R (F X)) \)
  (rule monoI)
  apply (rule srdes-refine-intro')
  apply (meson assms(1) monoE preR-antitone uthp-pred-laws.le-infI2)
  apply (meson assms(1) monoE periR-monotone uthp-pred-laws.le-infI2)
  apply (meson assms(1) monoE postR-monotone uthp-pred-laws.le-infI2)
done

lemma mu-srd-SRD:
  assumes mono \( F \cdot F \in [\text{SRD}]_H \to [\text{SRD}]_H \)
  shows \( \mu X \cdot R_s (\pre_R (F X) \vdash \peri_R (F X) \circ \post_R (F X)) \) is SRD
  apply (subst gfp-unfold)
  apply (simp add: mono-srd-iter assms)
  apply (rule RHS-tri-design-is-SRD)
  apply (simp-all add: unrest)

46
done

lemma mu-srd-iter:
  assumes mono F F ∈ [SRD]H → [SRD]H
  shows (µ X · Rₐ(preₐ(F(X)) ⊨ periₐ(F(X)) ⊗ postₐ(F(X)))) = F(µ X · Rₐ(preₐ(F(X)) ⊨ periₐ(F(X)) ⊗ postₐ(F(X))))
    apply (subst gfp-unfold)
    apply (simp add: mono-srd-iter assms)
    apply (subst SRD-as-reactive-tri-design[THEN sym])
    using Healthy-func assms
    using SRD-reactive-tri-design
    by blast
done

lemma mu-srd-form:
  assumes mono F F ∈ [SRD]H → [SRD]H
  shows µₐ F = (µ X · Rₐ(preₐ(F(X)) ⊨ periₐ(F(X)) ⊗ postₐ(F(X))))
proof −
  have 1: F(µ X · Rₐ(preₐ(F(X)) ⊨ periₐ(F(X)) ⊗ postₐ(F(X)))) is SRD
    by (simp add: Healthy-apply-closed assms(1) assms(2) mu-srd-SRD)
  have 2: Monoᵢᵣ-order SRDES F
    by (simp add: assms(1) mono-Monotone-utp-order)
  hence 3: µₐ Rₐ (F (µₐ F)) ⊨ periₐ (F (µₐ F)) ⊗ postₐ (F (µₐ F))) = µₐ F
    using SRD-reactive-tri-design by force
  hence (µ X · Rₐ(preₐ(F(X)) ⊨ periₐ(F(X)) ⊗ postₐ(F(X)))) ⊑ F(µₐ 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 ∘ SRD)
  by (simp add: mono-def R1-R2c-is-R2 R2-mono Rₐh-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 ;; IIₐ

lemma RD₃-idem: RD₃(RD₃(P)) = RD₃(P)
proof −
  have a: IIₐ ;; IIₐ = IIₐ
    by (simp add: SRD-left-unit SRD-srdes-skip)
  show ?thesis
    by (simp add: RD₃-def seqr-assoc a)

47
**lemma RD3-Idempotent [closure]:** Idempotent RD3
by (simp add: Idempotent-def RD3-idem)

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

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

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

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

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

**lemma RD3-intro-pre:**
  assumes $P$ is SRD $((\neg_r pre_{R}(P)) ; true_r = (\neg_r pre_{R}(P))$ $\exists st' \in per_{R}(P)$
  shows $P$ is RD3
proof –
  have $RD3(P) = R_a((\neg_r pre_{R}(P)) wp_r false \vdash (\exists st' \in per_{R}(P) \circ post_{R} P)$
    by (simp add: RD3-def SRD-right-unit-tri-lemma assms)
  also have $\ldots = R_a((\neg_r pre_{R}(P)) wp_r false \vdash per_{R} P \circ post_{R} P)$
    by (simp add: assms(3) ex-unrest)
  also have $\ldots = R_a((\neg_r pre_{R}(P)) wp_r false \vdash cmt_{R} P)$
    by (simp add: wp-reasoning rd-reasoning RD3-def R1-preR)
  finally show $?thesis$
    by (metis Healthy-def SRD-as-reactive-design assms(1))
qed

**lemma RHS-tri-design-right-unit-lemma:**
  assumes $\exists sok' : P$ $\exists Q : \exists R : wait' \not\in R$
  shows $R_a(P \vdash Q \circ R) ; II_R = R_a(P \vdash Q \circ R) ; R_a(true \vdash false \circ (\exists st' \in Q) \circ (R1 (R2s R) \circ R1 (R2s false) \circ ([II]_R)))$
proof –
  have $R_a(P \vdash Q \circ R) ; II_R = R_a(P \vdash Q \circ R) ; R_a(true \vdash false \circ (\exists st' \in Q) \circ (R1 (R2s R) \circ R1 (R2s false) \circ ([II]_R)))$
    by (simp add: RHSt-tri-design-composition assms unrest R2-false R1-false R2s-false)

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

proof –
from assms(3,4) have (R1 (R2s R) ;; R1 (R2s ($st' = u \circ \varnothing [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 ... = Rs ((\neg R) ;; R1 true) \vdash ((\exists \$st' \cdot Q) \circ R)
by (metis (no-types, lifting) R1-R2s-R1-true-lemma R1-R2s-R2c R2c-not RHS-design-R2c-pre RHS-design-neg-R1-pre RHS-design-post-R1 RHS-design-post-R2s)

also have ... = Rs ((\neg R) ;; true_r) \vdash ((\exists \$st' \cdot Q) \circ R)
by (rel-auto)
finally show \thesis.

qed

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

**assumes**

\(\$ok' \not\in P \circ \$ok' \not\in Q \circ \$ok' \not\in R \circ \$st' \not\in Q \circ \$wait' \not\in R\)

**shows** Rs(P || 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 unrest closure rpred)

**done**

RD3 reactive designs are those whose assumption can be written as a conjunction of a precon-
dition 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:

\[\text{outa } \not\in b; \text{ outa } \not\in e \implies (\neg_r b \lor \$tr' \cup u \circ \le u \circ \$tr') ;; R1(true) = (\neg_r b \lor \$tr' \cup u \circ \le u \circ \$tr')\]

**by** (rel-auto, blast, metis (no-types, lifting) dual-order.trans)

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

**assumes**

\(\text{outa } \not\in b \circ \text{ outa } \not\in e \circ \$ok' \not\in Q \circ \$st' \not\in Q \circ \$ok' \not\in R \circ \$wait' \not\in R\)

**shows** Rs((b \land \neg_r \$tr' \cup u \circ \le u \circ \$tr') \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 :: ('s,t::trace,'a) hrel-rsp ⇒ ('s,'t,'a) hrel-rsp

where [upred-defs]: NSRD = RD1 o RD3 o RHS

**lemma** RD1-RD3-commute: RD1(RD3(P)) = RD3(RD1(P))

**by** (rel-auto, blast+)

**lemma** NSRD-is-SRD [closure]: P is NSRD ⇒ 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]:

\[P is NSRD; Q(Rs(pre_R(P) \vdash peri_R(P) \circ post_R(P))) \implies Q(P)\]
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:
  \( NSRD(P) = R_s((\sim_r (\sim_r \pre_R(P)) :: R1 true) \vdash ((\exists \$st' \cdot \peri_R(P)) \circ \post_R(P))) \)

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 \( ... = RD3(R_s(\pre_R(P) \vdash \peri_R(P) \circ \post_R(P))) \)
    by (simp add: SRD-as-reactive-tri-design)
  also have \( ... = R_s(\pre_R(P) \vdash \peri_R(P) \circ \post_R(P)) :: II_R \)
    by (simp add: RD3-def)
  also have \( ... = R_s((\sim_r (\sim_r \pre_R(P)) :: R1 true) \vdash ((\exists \$st' \cdot \peri_R(P)) \circ \post_R(P))) \)
    by (simp add: RHS-tri-design-right-unit-lemma unrest)
  finally show \(?thesis\) by (simp add: Healthy-def)

qed

lemma NSRD-healthy-form:
  assumes \( P \) is NSRD
  shows \( R_s((\sim_r (\sim_r \pre_R(P)) :: R1 true) \vdash ((\exists \$st' \cdot \peri_R(P)) \circ \post_R(P))) = P \)
  by (metis Healthy-def NSRD-form assms)

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

proof –
  have \( NSRD(\bigsqcup A) = (\bigsqcup (NSRD \ i)) \)
    by (clarsimp simp add: ContinuousD NSRD-Continuous assms)
  also have \( ... = (\bigsqcup A) \)
    by (simp only: Healthy-carrier-image assms)
  finally show \(?thesis\) by (simp add: Healthy-def)

qed

lemma intChoice-NSRD-closed [closure]:
  assumes \( P \) is NSRD \( Q \) is NSRD
  shows \( P \cap Q \) is NSRD
  using NSRD-Sup-closure[of \{ \( P \), \( Q \)\}] by (simp add: assms)

lemma NRSD-SUP-closure [closure]:
  \[ \bigwedge \ i. \ i \in A \Rightarrow P(i) \ is \ NSRD; \ A \neq \{\} \] \( \Rightarrow (\bigsqcup i \in A. \ P(i)) \) is NSRD
  by (rule NRSD-Sup-closure, auto)

lemma NSRD-neg-pre-unit:
  assumes \( P \) is NSRD
  shows \( (\sim_r \pre_R(P)) :: true_r = (\sim_r \pre_R(P)) \)

proof –
  have \( (\sim \pre_R(P)) = (\sim_r \pre_R(R_s((\sim_r (\sim_r \pre_R(P)) :: R1 true) \vdash ((\exists \$st' \cdot \peri_R(P)) \circ \post_R(P))))) \)
    by (simp add: NSRD-healthy-form assms)
  also have \( ... = R1 (R2c ((\sim_r \pre_R P) :: R1 true)) \)

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 ... = (¬预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 (¬预P) ;; Q = (¬预P)
by (metis (no-types, hide-lams) NSRD-neg-pre-unit RD1-left-zero assms (1) assms (2) seqr-assoc)

lemma NSRD-st'-unrest-peri [unrest]:
  assumes P is NSRD
  shows $gst' ♯ peri(P)
proof –
  have peri(P) = peri(Rs((¬预P) ;; R1 true) ⊢ (∃gst' · peri(P) ◦ post(P)))
    by (simp add: NSRD-healthy-form assms)
also have ... = R1 (R2c (¬预P) ;; R1 true ⇒ (∃gst' · peri P))
    by (simp add: rea-peri-RHS-design usubst unrest)
also have $gst' ♯ ...
    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(P)
proof –
  have pre(P) = pre(Rs((¬预P) ;; R1 true) ⊢ (∃gst' · peri(P) ◦ post(P)))
    by (simp add: NSRD-healthy-form assms)
also have ... = (R1 (R2c (¬预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(P)
proof –
  have pre(P) = pre(Rs((¬预P) ;; R1 true) ⊢ (∃gst' · peri(P) ◦ post(P)))
    by (simp add: NSRD-healthy-form assms)
also have ... = R1 (R2c (¬预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 pre-R-RR [closure]: $P$ is NSRD $\implies$ $\text{pre}_R(P)$ is RR

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

lemma NSRD-neg-pre-RC [closure]:

assumes $P$ is NSRD

shows $\text{pre}_R(P)$ is RC

by (rule RC-intro, simp-all add: closure assms NSRD-neg-pre-unit rpred)

lemma NSRD-intro:

assumes $P$ is SRD $(\neg_r \text{pre}_R(P)) ;; \text{true}_r = (\neg_r \text{pre}_R(P)) \# \text{peri}_R(P)$

shows $P$ is NSRD

proof –

have $\text{NSRD}(P) = \text{R}_s((\neg_r (\neg_r \text{pre}_R(P)) ;; \text{R1 true}) \vdash ((\exists \# \text{st} \cdot \text{peri}_R(P)) \circ \text{post}_R(P)))$

by (simp add: NSRD-form)

also have $\ldots = \text{R}_s(\text{pre}_R P \vdash \text{peri}_R P \circ \text{post}_R P)$

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 $\text{pre}_R(P)$ is RC $\# \text{peri}_R(P)$

shows $P$ is NSRD

by (metis Healthy-def NSRD-form SRD-reactive-tri-design assms (1) assms (2) assms (3) ex-unrest rea-not-false wp-rea-RC-false wp-rea-def)

lemma NSRD-rdes-intro [closure]:

assumes $P$ is RC $Q$ is RR $R$ is RR $\# \text{peri}_R(P)$

shows $\text{R}_s(P \vdash Q \circ R)$ 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 $\wedge (\neg_r \text{pre}_R(P)) ;; \text{R1}((\text{true}) = (\neg_r \text{pre}_R(P)) \wedge (\$st \# \text{peri}_R(P))))$

by (meson NSRD-intro NSRD-is-SRD NSRD-neg-pre-unit NSRD-st'-'unrest-peri)

lemma NSRD-is-RD3 [closure]:

assumes $P$ is NSRD

shows $P$ is RD3

by (simp add: NSRD-is-SRD NSRD-neg-pre-unit NSRD-st'-'unrest-peri RD3-intro-pre assms)

lemma NSRD-refine-elim:

assumes
\[ P \subseteq Q \text{ is NSRD} \]

\[ [ \, \text{pre}_R(P) \Rightarrow \text{pre}_R(Q) \, ; \, \text{pre}_R(P) \land \text{peri}_R(Q) \Rightarrow \text{peri}_R(P) \, ; \, \text{pre}_R(P) \land \text{post}_R(Q) \Rightarrow \text{post}_R(P) \, ] \]
\[ \Rightarrow R \]

\[ \text{shows } R \]

\[ \text{proof} \]

\[ \text{have } R_s(\text{pre}_R(P) \Rightarrow \text{peri}_R(P) \land \text{post}_R(P)) \subseteq R_s(\text{pre}_R(P) \Rightarrow \text{peri}_R(Q) \land \text{post}_R(Q)) \]

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

\[ \text{hence } 1: \text{pre}_R(P) \Rightarrow \text{pre}_R(Q) \text{ and } 2: \text{pre}_R(P) \land \text{peri}_R(Q) \Rightarrow \text{peri}_R(P) \text{ and } 3: \text{pre}_R(P) \land \text{post}_R(Q) \Rightarrow \text{post}_R(P) \]

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

\[ \text{with assms(4) show } ?\text{thesis} \]

\[ \text{by } \text{simp} \]

\[ \text{qed} \]

**lemma** NSRD-right-unit: \( P \text{ is NSRD} \Rightarrow P ; II_R = P \)

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

**lemma** NSRD-composition-wp:

\[ \text{assumes } P \text{ is NSRD} \text{ Q is SRD} \]

\[ \text{shows } P ; Q = \]

\[ R_s(\text{pre}_R(P) \land \text{post}_R(P) \text{ wp}_r \text{ pre}_R(Q)) \land (\text{peri}_R(P) \lor (\text{post}_R(P) ; \text{peri}_R(Q)) \land (\text{post}_R(P) ; \text{post}_R(Q)) \]

\[ \text{by } (\text{simp add: SRD-composition-wp assms NSRD-is-SRD wp-rea-def NSRD-neg-pre-unit NSRD-st'-unrest-peri R1-negate-R1 R1-preR ex-unrest rpred}) \]

**lemma** preR-NSRD-seq-lemma:

\[ \text{assumes } P \text{ is NSRD} \text{ Q is SRD} \]

\[ \text{shows } R1 (R2c (post_R P ; (\neg_r \text{ peri}_R Q))) = \text{post}_R(P ; (\neg_r \text{ peri}_R(Q)) \]

\[ \text{proof} \]

\[ \text{have } \text{post}_R(P ; (\neg_r \text{ peri}_R Q) = R1(R2c(post_R P)) ; R1(R2c(\neg_r \text{ peri}_R Q)) \]

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

\[ \text{also have } ... = R1 (R2c (post_R P ; (\neg_r \text{ peri}_R(Q))) \]

\[ \text{by } (\text{simp add: R1-seq}\_R2c-R1-seq calculation) \]

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

\[ \text{qed} \]

**lemma** preR-NSRD-seq [rdes]:

\[ \text{assumes } P \text{ is NSRD} \text{ Q is SRD} \]

\[ \text{shows } \text{pre}_R(P ; Q) = (\text{pre}_R P \land \text{post}_R P \text{ wp}_r \text{ pre}_R(Q) \]

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

\[ (\text{metis (no-types, lifting) Healthy-def Healthy-if NSRD-is-SRD R1-R2c-commute R1-R2c-seq-distribute R1-rea-not' R2c-seq-closure assms(1) assms(2) postR-R2c-closed postR-SRD-R1 preR-R2c-closed rea-not-R1 rea-not-R2c}) \]

**lemma** periR-NSRD-seq [rdes]:

\[ \text{assumes } P \text{ is NSRD} \text{ Q is NSRD} \]

\[ \text{shows } \text{peri}_R(P ; Q) = (\text{peri}_R P \land \text{post}_R P \text{ wp}_r \text{ peri}_R(Q) \Rightarrow_r (\text{peri}_R P \lor (\text{post}_R(P ; \text{peri}_R(Q))) \]

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

**lemma** postR-NSRD-seq [rdes]:

\[ \text{assumes } P \text{ is NSRD} \text{ Q is NSRD} \]

\[ \text{shows } \text{post}_R(P ; Q) = (\text{pre}_R P \land \text{post}_R P \text{ wp}_r \text{ pre}_R(Q) \Rightarrow_r (\text{post}_R P ; \text{post}_R Q)) \]

53
If a normal reactive design has postcondition false, then it is a left zero for sequential composition.

**lemma** NSRD-seq-post-false:

- **assumes** \( P \) is NSRD \( Q \) is SRD \( post_R(P) = false \)
- **shows** \( P ;; Q = P \)
- **apply** (simp add: NSRD-composition-up assms wp rpred closure)
- **using** NSRD-is-SRD SRD-reactive-tri-design assms(1,3) **apply** fastforce

**qed**

If a normal reactive design has postcondition false, then it is a left zero for sequential composition.
lemma NSRD-srd-skip [closure]: II \_R is NSRD  
  by (rule NSRD-intro, simp-all add: rdes closure unrest)

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

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

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

lemma NSRD-right-Miracle-tri-lemma:  
  assumes P is NSRD  
  shows P \_\_ Miracle = R \_s (pre \_R P \_\_ peri \_R P \_\_ false)  
  by (simp add: NSRD-composition-wp closure assms rdes wp rpred)

The set of non-terminating behaviours is a subset

lemma NSRD-right-Miracle-refines:  
  assumes P is NSRD  
  shows P \_\_ Miracle  
  proof -  
    have R \_s (pre \_R P \_\_ peri \_R P \_\_ false) \_\_ R \_s (pre \_R P \_\_ peri \_R P \_\_ false)  
      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 \_\_ P \_\_ \_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 \_\_ P \_\_ \_n is NSRD  
  by (metis upower-Suc-NSRD-closed upred-semiring.power-Suc)

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

lemma preR-power:  
  assumes P is NSRD  
  shows pre \_R (P \_\_ \_n) = (\_\_ i \in \{0..n\}. (post \_R \_P \_\_ i) wp \_r (pre \_R \_P))  
  proof (induct n)  
    case 0  
    then show ?case  
      by (simp add: wp closure)  
  next  
    case (Suc n) note hyp = this

55
have \( \text{pre}_R (P \cdot (\text{Suc } n + 1)) = \text{pre}_R (P \cdot P \cdot (n+1)) \)
  by (simp add: upred-semiring-power-Suc)
also have \( \ldots = (\text{pre}_R P \land \text{post}_R P \wp_r \text{pre}_R (P \cdot (\text{Suc } n))) \)
  using \( \text{NSRD-iff} \) \text{assms} \( \text{preR-NSRD-seq} \) \text{power-Suc-NSRD-closed} by fastforce
also have \( \ldots = (\text{pre}_R P \land \text{post}_R P \wp_r (\bigcup i \in \{0..n\}. \text{post}_R P \cdot i \wp_r \text{pre}_R P)) \)
  by (simp add: hyp upred-semiring.power-Suc)
also have \( \ldots = (\text{pre}_R P \land (\bigcup i \in \{0..n\}. \text{post}_R P \wp_r (\text{post}_R P \cdot i \wp_r \text{pre}_R P))) \)
  by (simp add: wp)
also have \( \ldots = (\text{pre}_R P \land (\bigcup i \in \{0..n\}. (\text{post}_R P \cdot (i+1) \wp_r \text{pre}_R P))) \)
proof –
  have \( \bigwedge i. R1 (\text{post}_R P \cdot i ; ; (\neg_r \text{pre}_R P)) = (\text{post}_R P \cdot i ; ; (\neg_r \text{pre}_R P)) \)
  by (induct-tac i, simp-all add: closure \text{Healthy-if assms})
  thus \( \neg \text{thesis} \)
  by (simp add: wp-rea-def upred-semiring.power-Suc segr-assoc rpred closure assms)
qed
also have \( \ldots = (\text{post}_R P \cdot 0 \wp_r \text{pre}_R P \land (\bigcup i \in \{0..n\}. (\text{post}_R P \cdot (i+1) \wp_r \text{pre}_R P))) \)
  by (simp add: wp assms closure)
also have \( \ldots = (\text{post}_R P \cdot 0 \wp_r \text{pre}_R P \land (\bigcup i \in \{1..\text{Suc } n\}. (\text{post}_R P \cdot i \wp_r \text{pre}_R P))) \)
proof –
  have \( (\bigcup i \in \{0..n\}. (\text{post}_R P \cdot (i+1) \wp_r \text{pre}_R P)) = (\bigcup i \in \{1..\text{Suc } n\}. (\text{post}_R P \cdot i \wp_r \text{pre}_R P)) \)
  by (rule cong[of \text{Inf}], simp-all add: fun-eq-iff)
  (metis \text{no-types}, lifting \text{image-Suc-atLeastAtMost} \text{image-cong} \text{image-image})
  thus \( \neg \text{thesis} \)
  by simp
qed
also have \( \ldots = (\bigcup i \in \text{insert } 0 \{1..\text{Suc } n\}. (\text{post}_R P \cdot i \wp_r \text{pre}_R P)) \)
  by (simp add: conj-upred-def)
also have \( \ldots = (\bigcup i \in \{0..n\}. \text{post}_R P \cdot i \wp_r \text{pre}_R P) \)
  by (simp add: atLeast0-atMost-Suc-eq-insert-0)
finally show \( \neg \text{case} \)
  by (simp add: upred-semiring.power-Suc)
qed

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

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

\text{declare} \text{upred-semiring.power-Suc [simp]}

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

56
by (simp add: closure assms rdes)
also have ... = (preR(P ♦ (Suc n + 1)) ⇒r (periR P ∨ postR P ;; (preR (P ♦ (Suc n)) ⇒r (∏ i∈{0..n}. postR P ♦ i) ;; periR P)))
  by (simp only: hyp)
also have ... = (preR P ⇒r periR P ∨ (postR P wpr preR (P ♦ P ♦ n) ⇒r postR P ;; (preR (P ;; P ♦ n) ⇒r (∏ i∈{0..n}. postR P ♦ i) ;; periR P)))
  by (simp add: rdes closure assms, rel-blast)
also have ... = (preR P ⇒r periR P ∨ (postR P wpr preR (P ;; P ♦ n) ⇒r postR P ;; (∏ i∈{0..n}. postR P ♦ i) ;; periR P)))
proof –
  have (∏ i∈{0..n}. postR P ♦ i) is R1
    by (simp add: NSRD-is-SRD R1-Continuous R1-power Sup-Continuous-closed assms postR-SRD-R1)
  hence 1:(∏ i∈{0..n}. postR P ♦ i) ;; periR P) is R1
    by (simp add: closure assms)
  hence (preR (P ;; P ♦ n) ⇒r (∏ i∈{0..n}. postR P ♦ i) ;; periR P) is R1
    by (simp add: closure)
  hence (postR P wpr preR (P ;; P ♦ n) ⇒r postR P ;; (preR (P ;; P ♦ n) ⇒r (∏ i∈{0..n}. postR P ♦ i) ;; periR P))
    = (postR P wpr preR (P ;; P ♦ n) ⇒r R1(postR P) ;; R1(preR (P ;; P ♦ n) ⇒r (∏ i∈{0..n}. postR P ♦ i) ;; periR P))
    by (simp add: Healthy-if R1-post-SRD assms closure)
thus thesis
  by (simp only: wp-rea-impl-lemma, simp add: Healthy-if 1, simp add: R1-post-SRD assms closure)
qed
also have ... = (preR P ∧ postR P wpwr preR (P ;; P ♦ n) ⇒r periR P ∨ postR P ;; (∏ i∈{0..n}. postR P ♦ i) ;; periR P))
  by (pred-auto)
also have ... = (preR P ∧ postR P wpwr preR (P ;; P ♦ n) ⇒r periR P ∨ (∏ i∈{0..n}. postR P ♦ (Suc i)) ;; periR P))
  by (simp add: seq-Sup-distl seqr-assoc[THEN sym])
also have ... = (preR P ∧ postR P wpwr preR (P ;; P ♦ n) ⇒r periR P ∨ ((∏ i∈{1..Suc n}. postR P ♦ i)) ;; periR P))
proof –
  have (∏ i∈{0..n}. postR P ♦ Suc i) = (∏ i∈{1..Suc n}. postR P ♦ i)
    apply (rule cong[of SUP], auto)
    apply (metis atLeast0AtMost atMost-iff image-Suc-atLeastAtMost rev-image-eql upred-semiring.power-Suc)
    using Suc-le-D apply fastforce
    done
thus thesis by simp
qed
also have ... = (preR P ∧ postR P wpwr preR (P ;; P ♦ n) ⇒r (∏ i∈{0..Suc n}. postR P ♦ i) ;; periR P)
  by (simp add: SUP-atLeastAtMost-first winf-or seqr-or-distl seqr-or-distr)
also have ... = (preR(P ♦ (Suc (Suc n)))) ⇒r (∏ i∈{0..Suc n}. postR P ♦ i) ;; periR P))
  by (simp add: rdes closure assms)
finally show ?case by (simp)
qed

lemma periR-power′ [rdes]:

57
proof (next power-rdes-def lemma upred-semiring P \hat{periR}\text{-power-Suc} n)
  assume P is NSRD
  have \(\text{post}_{R}(P^{\bullet} (\text{Suc } n)) = (\text{pre}_{R}(P^{*} (\text{Suc } n)) \Rightarrow \prod_{i \in \{0..n\}} \cdot \text{post}_{R}(P)^{\bullet} i \Rightarrow \text{peri}_{R}(P))\)
  by (simp add: periR-power assms UINF-as-Sup[THEN sym])

lemma \(\text{periR-power-Suc}\) [rdes]:
  assumes P is NSRD
  shows \(\text{peri}_{R}(P^{*} (\text{Suc } n)) = (\text{pre}_{R}(P^{*} (\text{Suc } n)) \Rightarrow \prod_{i \in \{0..n\}} \cdot \text{post}_{R}(P)^{\bullet} i \Rightarrow \text{peri}_{R}(P))\)
  by (simp add: rdes assms)

lemma \(\text{postR-power}\) [rdes]:
  assumes P is NSRD
  shows \(\text{post}_{R}(P^{\bullet} (\text{Suc } n)) = (\text{pre}_{R}(P^{*} (\text{Suc } n)) \Rightarrow \text{post}_{R}(P)^{\bullet} \text{Suc } n)\)
  proof (induct n)
    case 0
    then show \(\text{case } \emptyset\)
      by (simp add: NSRD-is-SRD NSRD-wait\text{-unrest-pre SRD-post-under-pre assms)
  next
    case \(\text{Suc } n\)
    note hyp = this
    have \(\text{post}_{R}(P^{\bullet} (\text{Suc } n + 1)) = \text{post}_{R}(P^{\bullet} (n+1))\)
      by (simp)
    also have \(\ldots = (\text{pre}_{R}(P^{\bullet} (\text{Suc } n + 1)) \Rightarrow \text{post}_{R}(P^{\bullet} (\text{Suc } n)) \Rightarrow \text{pre}_{R}(P^{\bullet} (\text{Suc } n)))\)
      by (simp add: closure assms rdes)
    also have \(\ldots = (\text{pre}_{R}(P^{\bullet} (\text{Suc } n + 1)) \Rightarrow \text{post}_{R}(P^{\bullet} (\text{Suc } n)) \Rightarrow \text{pre}_{R}(P^{\bullet} (\text{Suc } n)))\)
      by (simp only: hyp)
    also have \(\ldots = (\text{pre}_{R}(P \Rightarrow \text{post}_{R}(P \text{ wp } P) \text{ pre}_{R}(P^{\bullet} (\text{Suc } n)) \Rightarrow \text{post}_{R}(P^{\bullet} (\text{Suc } n)))\)
      by (simp add: rdes closure assms)
    finally show \(\text{case } \emptyset\)
      by (simp)
  qed

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

lemma \(\text{power-rdes-def}\) [rdes-def]:
  assumes P is RC Q is RR R is RR \(\$st' \not\in Q\)
  shows \(\text{R}_{1}(P \Rightarrow Q \circ R)^{\bullet} (\text{Suc } n)\)
    = \(\text{R}_{1}(\prod_{i \in \{0..n\}} \cdot (R^{\bullet} i \text{ wp } P) \Rightarrow (\prod_{i \in \{0..n\}} \cdot R^{\bullet} i) \Rightarrow (Q) \circ (R^{\bullet} (\text{Suc } n))\)
  proof (induct n)
    case 0
    then show \(\text{case } \emptyset\)
      by (simp add: wp assms closure)
  next
    case \(\text{Suc } n\)
have 1: \((P \land (\bigsqcup_i \in \{0..n\} \cdot R \wp_r (R \cdot i \wp_r P))) = (\bigsqcup_i \in \{0..Suc n\} \cdot R \cdot i \wp_r P)\)
(is \(\text{lhs} = \text{rhs}\))
proof -
have \(\text{lhs} = (P \land (\bigsqcup_i \in \{0..n\} \cdot (R \cdot Suc i \wp_r P)))\)
  by (simp add: wp closure assms)
also have \(\ldots = (P \land (\bigsqcup_i \in \{0..n\} \cdot (R \cdot Suc i \wp_r P)))\)
  by (simp only: USUP-as-Inf-collect)
also have \(\ldots = (P \land (\bigsqcup_i \in \{0..Suc n\}. (R \cdot i \wp_r P)))\)
  by (simp add: wp assms closure conj-upred-def)
also have \(\ldots = (\bigsqcup_i \in \{0..Suc n\}. (R \cdot i \wp_r P))\)
  by (simp add: atLeastAtMost-insertL)
finally show \(\text{thesis}\)
  by (simp add: USUP-as-Inf-collect)
qed

have 2: \((Q \lor R ;; (\prod_i \in \{0..n\} \cdot R \cdot i) ;; Q) = (\prod_i \in \{0..Suc n\} \cdot R \cdot i) ;; Q\)
(is \(\text{lhs} = \text{rhs}\))
proof -
have \(\text{lhs} = (Q \lor (\prod_i \in \{0..n\} \cdot R \cdot Suc i) ;; Q)\)
  by (simp add: seqr-assoc THEN sym seq-UINF-distl)
also have \(\ldots = (Q \lor (\prod_i \in \{0..n\}. R \cdot Suc i) ;; Q)\)
  by (simp only: UINF-as-Sup-collect)
also have \(\ldots = (Q \lor (\prod_i \in \{0..Suc n\}. R \cdot i) ;; Q)\)
  by (simp add: disj-upred-def THEN sym seqr-or-distl)
also have \(\ldots = ((\prod_i \in \{0..Suc n\}. R \cdot i) ;; Q)\)
  by (simp add: atLeastAtMost-insertL)
finally show \(\text{thesis}\)
  by (simp add: UINF-as-Sup-collect)
qed

have 3: \((\prod_i \in \{0..n\} \cdot R \cdot i) ;; Q\) is RR
proof -
have \(\prod_i \in \{0..n\} \cdot R \cdot i) ;; Q = (\prod_i \in \{0..n\}. R \cdot i) ;; Q\)
  by (simp add: UINF-as-Sup-collect)
also have \(\ldots = (\prod_i \in \{0..Suc n\}. R \cdot i) ;; Q\)
  by (simp add: atLeastAtMost-insertL)
also have \(\ldots = (Q \lor (\prod_i \in \{0..n\}. R \cdot Suc i) ;; Q)\)
  by (metis (no-types, lifting) SUP-insert disj-upred-def seqr-left-unit seqr-or-distl upred-semiring.power-0)
also have \(\ldots = (Q \lor (\prod_i \in \{0..<n\}. R \cdot Suc i) ;; Q)\)
  by (simp add: UINF-as-Sup-collect)
also have \(\ldots\) is RR
  by (simp-all add: closure assms)
finally show \(\text{thesis}\)
qed
from 1 2 3 Suc show \(\text{case}\)
  by (simp add: Suc RHS-tri-normal-design-composition' closure assms wp)
qed
declare upred-semiring.power-Suc [simp del]

theorem uplus-rdes-def [rdes-def]:
assumes P is RC Q is RR R is RR $st' \dashv Q$
shows (R, (P \vdash Q \circ R))' = R, (R'' \ wp_r, P \vdash R'' \ ; Q \circ R')
proof
  have 1:\(\Pi\ i \cdot R \cdot i\); Q = R'' ; Q
   by (metis (no-types) RA1 assms(2) rea-unit-unit(2) rrel-thy.Star-def ustar-alt-def)
show ?thesis
   by (simp add: uplus-power-def seq-UINF-distr wp closure assms rdes-def)
qed

5.1 UTP theory

typedec NSRDES

abbreviation NSRDES ≡ UTHY(NSRDES, (s', t::trace, 'α) rsp)

overloading
nsrdes-hcond == upred-hcond :: (NSRDES, (s', t::trace, 'α) rsp) utpy ⇒ ((s', t, 'α) rsp × (s', t, 'α) rsp)
nsrdes-unit == upred-unit :: (NSRDES, (s', t::trace, 'α) rsp) utpy ⇒ (s', t, 'α) hrel-rsp

begin
  definition nsrdes-hcond :: (NSRDES, (s', t::trace, 'α) rsp) utpy ⇒ ((s', t, 'α) rsp × (s', t, 'α) rsp)
definition nsrdes-unit :: (NSRDES, (s', t::trace, 'α) rsp) utpy ⇒ (s', t, 'α) hrel-rsp where

interpretation nsrd-thy: utp-theory-kleene UTHY(NSRDES, (s', t::trace, 'α) rsp)
rewrites \(\bigwedge\) P. P ∈ carrier (uthy-order NSRDES) ⟷ P is NSRD
and P is \(\mathbb{H}_{NSRDES}\) ⟷ P is NSRD
and \((\mu X \cdot F (\mathbb{H}_{NSRDES} X)) = (\mu X \cdot F (\mathbb{H}_{NSRD} X))\)
and carrier (uthy-order NSRDES) → carrier (uthy-order NSRDES) ≡ [NSRDES]_H → [NSRDES]_H
and \(\mathbb{H}_{NSRDES}\) = \(\mathbb{H}_{NSRDES}\) 
and \(\mathbb{H}_{NSRDES}\) = \(\mathbb{H}_{NSRDES}\)
and le (uthy-order NSRDES) = op ⊆
proof -
  interpret lat: utp-theory-continuous UTHY(NSRDES, (s', t, 'α) rsp)
  by (unfold-locale, simp-all add: nsrdes-hcond-def nsrdes-unit-def closure Healthy-if)
  show 1: \(\mathbb{H}_{NSRDES}\) = (Miracle :: (s', t, 'α) hrel-rsp)
  by (metis NSRD-Miracle NSRD-is-SRD lat.top-healthy lat.uthy-continuous-axioms nsrdes-hcond-def srdes-theory-continuous.meet-top upred-semiring.add-commute utp-theory-continuous.meet-top)
thus utp-theory-kleene UTHY(NSRDES, (s', t, 'α) rsp)
  by (unfold-locale, simp-all add: nsrdes-hcond-def nsrdes-unit-def closure Healthy-if Miracle-left-zero SRD-left-unit NSR-right-unit)
qed (simp-all add: nsrdes-hcond-def nsrdes-unit-def closure Healthy-if)

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

abbreviation TestR (test_R) where
abbreviation StarR :: (s, 't::trace, 'a) hrel-rsp ⇒ (s, 't, 'a) hrel-rsp (.*R [999] 999) where
StarR P ≡ P*NSRDES

lemma StarR-rdes-def [rdes-def]:
assumes P is RC Q is RR R is RR
shows (R_1((P ⊢ Q ∩ R)))*R = R_1((R** wp_r P) ⊢ R* ∩ Q ∩ R*)
by (simp add: rrel-thy. Star-alt-def assms closure unrest 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]|_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 :: 's upred ⇒ ('t::trace ⇒ 's upred) ⇒ ('t ⇒ 's hrel) ⇒ ('s, 't, 'a) hrel-rsp where
mk-RD P Q R = R_1((P|S< ⊢ (Q|x)|S<[x→&tt] ∩ R(x)|S[x→&tt])
deﬁnition trace-pred :: ('t::trace ⇒ 's upred) ⇒ ('s, 't, 'a) hrel-rsp where
[upred-defs]: trace-pred P = [(P x)|S<[x→&tt]
syntax
  -trace-var :: logic
  -mk-RD :: logic ⇒ logic ⇒ logic ⇒ logic ([/- ⊢ -/ | -]_R)
  -trace-pred :: logic ⇒ logic ([/-]_R)
parse-translation ⟮
let
  fun trace-var-tr [] = Syntax.free trace
         | trace-var-tr t = raise Match;
in
  [@[syntax-const -trace-var], K trace-var-tr]]
end

translations
[P ⊢ Q | R]|_R ⇒ CONST mk-RD P (λ -trace-var. Q) (λ -trace-var. R)
[P ⊢ Q | R]|_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]: preR([P ⊢ Q(trace) | R(trace)]|_R) = R1([P]|_S<)
by (simp add: mk-RD-def rea-pre-RHS-design usubst unrest R2c-not R2c-lift-state-pre)
lemma trace-pred-RR-closed [closure]:
  \([P \;\text{trace}]\) is RR
by (rel-auto)

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

lemma R2c-msubst-tt: R2c (msubst (\(\lambda x. [Q x]_s\) &tt)) = (msubst (\(\lambda x. [Q x]_s\) &tt))
by (rel-auto)

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

Refinement introduction law for contracts

lemma RD-contract-refine:
assumes
  \(Q\) is SRD
  \(\{P_1\}_S< \Rightarrow \text{pre}_R Q\)
  \(\{P_1\}_S< \wedge \text{peri}_R Q \Rightarrow [P_2 x]_S< [x\rightarrow\&tt]\)\)
  \(\{P_1\}_S< \wedge \text{post}_R Q \Rightarrow [P_3 x]_S[\text{trace}\rightarrow\&tt]\)\)
sows
\([P_1 \;\text{trace}] | P_3(\text{trace})]_R \subseteq Q\)
proof
  have \([P_1 \;\text{trace}] | P_3(\text{trace})]_R \subseteq \text{R}_s(\text{pre}_R(Q) \Rightarrow \text{peri}_R(Q) \odot \text{post}_R(Q))\)
  using \(\text{assms}\)
  by (simp add: mk-RD-def, rule_tac srdes-tri-refine-intro, simp-all)
  thus ?thesis
by (simp add: SRD-reactive-tri-design \(\text{assms}(1)\))
qed

end

7 Reactive design tactics

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

Theorems for normalisation

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

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

**method** rpred-simp = (uxexpr-simp simps: rpred usubst closure unrest)

Tactic to expand out healthy reactive design predicates into the syntactic triple form.

**method** rdes-expand uses cls = (insert cls, (erule RD-elim)+)

Tactic to simplify the definition of a reactive design

**method** rdes-simp uses cls cong simps =

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

Tactic to split a refinement conjecture into three POs

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

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

Tactic to split an equality conjecture into three POs

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

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

Tactic to prove a refinement

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

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

Tactics to prove an equality

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

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

Via antisymmetry

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

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

Tactic to calculate pre/peri/postconditions from reactive designs

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

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

**method** rdspl-refine =

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

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

**method** rdspl-eq =

(rule-tac antisym, rdes-refine, rdes-refine)

end

8 Reactive design parallel-by-merge

theory utp-rdes-parallel

imports

  utp-rdes-normal
begin

R3h implicitly depends on RD1, and therefore it requires that both sides be RD1. We also require that both sides are R3c, and that wait\(_m\) is a quasi-unit, and \(\div_m\) yields divergence.

lemma \(st\)-U0-alpha: \(\exists \ \$$st \cdot II\)_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, 'α) rsp merge \((II\_RM)\) where
\[ [upred-defs]: \ H\_RM = (\(\exists \ $$st\_< \cdot \skip\_m \lor (\lnot \$$ok\_< \land \$$tr\_< \leq_{\alpha} \$$str\_<))\]

definition [upred-defs]: \(R3hm(M) = (II\_RM \triangleright \$$wait\_< \triangleright M)\)

lemma \(R3h\)-idem: \(R3hm(R3hm(P)) = R3hm(P)\)
by (rel-auto)

lemma \(R3h\)-par-by-merge [closure]:
assumes \(P\) is \(R3h\) \(Q\) is \(R3h\) \(M\) is \(R3hm\)
shows \(\langle P \parallel M \rangle\) is \(R3h\)
proof -
\[
\begin{align*}
& \text{have } (P \parallel M) = ((P \parallel M)\[true/\$$ok\] \langle \$$ok \triangleright (P \parallel M)\[false/\$$ok\]\[true/\$$wait\] \langle \$$wait \triangleright (P \parallel M) (P \parallel M)\]
& \quad \text{by (simp add: cond-var-subst-left cond-var-subst-right)}
& \quad \text{also have } ... = ((P \parallel M)\[true, true/\$$ok, $$\$$wait\] \langle \$$ok \triangleright (P \parallel M)\[false, true/\$$ok, $$\$$wait\] \langle \$$wait \triangleright (P \parallel M) (P \parallel M)\]
& \quad \text{by (rel-auto)}
& \quad \text{also have } ... = ((\exists \ $$st \cdot II)\[true, true/\$$ok, $$\$$wait\] \langle \$$ok \triangleright (P \parallel M)\[false, true/\$$ok, $$\$$wait\] \langle \$$wait \triangleright (P \parallel M) (P \parallel M)\]
\end{align*}
\]
proof -
\[
\begin{align*}
& \text{have } (P \parallel M)\[true, true/\$$ok, $$\$$wait\] = (([P]_0 \land [Q]_1 \land \$$v\_< =_{\alpha} \$$v) ;; R3hm(M))\[true, true/\$$ok, $$\$$wait\]
& \quad \text{by (simp add: par-by-merge-def U0-as-alpha U1-as-alpha assms Healthy-if)}
& \quad \text{also have } ... = (([P]_0 \land [Q]_1 \land \$$v\_< =_{\alpha} \$$v) ;; (\exists \ $$st\_< \cdot \$$v\_< =_{\alpha} \$$v\_<))\[true, true/\$$ok, $$\$$wait\]
& \quad \text{by (rel-blast)}
& \quad \text{also have } ... = ((R3h(P))_0 \land \$$v\_< =_{\alpha} \$$v) ;; (\exists \ $$st\_< \cdot \$$v\_< =_{\alpha} \$$v\_<))\[true, true/\$$ok, $$\$$wait\]
& \quad \text{by (simp add: assms Healthy-if)}
& \quad \text{also have } ... = (\exists \ $$st \cdot II)\[true, true/\$$ok, $$\$$wait\]
& \quad \text{by (rel-auto)}
\end{align*}
\]
finally show \(\forall \text{thesis by (simp add: closure assms unrest)}\)
\[\text{qed}\]
also have ... = (\(\exists \ $$st \cdot II)\[true, true/\$$ok, $$\$$wait\] \langle \$$ok \triangleright (RI(true))\[false, true/\$$ok, $$\$$wait\] \langle \$$wait \triangleright (P \parallel M) (P \parallel M)\]
\[
\begin{align*}
& \text{prove -}
& \text{have } (P \parallel M)\[false, true/\$$ok, $$\$$wait\] = (([P]_0 \land [Q]_1 \land \$$v\_< =_{\alpha} \$$v) ;; R3hm(M))\[false, true/\$$ok, $$\$$wait\]
& \quad \text{by (simp add: par-by-merge-def U0-as-alpha U1-as-alpha assms Healthy-if)}
& \quad \text{also have } ... = (([P]_0 \land [Q]_1 \land \$$v\_< =_{\alpha} \$$v) ;; (\exists \ $$st\_< \cdot \$$v\_< =_{\alpha} \$$v\_<))\[false, true/\$$ok, $$\$$wait\]
& \quad \text{by (simp add: assms Healthy-if)}
& \quad \text{also have } ... = ((R3h(P))_0 \land \$$v\_< =_{\alpha} \$$v) ;; (\exists \ $$st\_< \cdot \$$v\_< =_{\alpha} \$$v\_<))\[false, true/\$$ok, $$\$$wait\]
& \quad \text{by (rel-blast)}
\end{align*}
\]
finally show \(\forall \text{thesis by simp}\)
\[\text{qed}\]
also have ...
  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))} RD1(R1(Q)))[[false/$ok] = R1(true)
    by (rel-blast)
  have (P \parallel_M Q) = (P \parallel_M Q)[true/$ok] ∧ $ok ⇒ (P \parallel_M Q)[false/$ok]
    by (simp add: cond-var-split)
  also have ... = R1(P \parallel_M Q) ∧ $ok ⇒ R1(true)
    by (metis 1 Healthy-if R1-par-by-merge assms calculation
        cond-iden cond-var-subst-right in-var-uvvar ok-evb-lens)
  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_{Q} :: M)
    by (simp add: par-by-merge-def)
  also from assms have ...
    by (simp add: Healthy-def RD2-def H2-def)
  also from assms have ...
    by (simp add: seq-assoc)
  also from assms have ...
    by (simp add: RD2(P \parallel_M Q)
    by (simp add: RD2-def H2-def par-by-merge-def)
finally show ?thesis
    by (simp add: Healthy-def)
qed

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

definition nmerge-rd0 (N_0) where
  [upred-defs]: N_0(M) = ($wait´ =_{u} ($0-wait \lor $1-wait) ∧ $tr_{<} ≤_{u} $tr´
  \lor (\exists $0-ok;$1-ok;$ok_{<};$ok´;$0-wait;$1-wait;$wait_{<};$wait´ · M))

definition nmerge-rd1 (N_1) where
  [upred-defs]: N_1(M) = ($ok´ =_{u} ($0-ok \land $1-ok) \land N_0(M))
definition nmerge-rd \((N_R)\) where
\[\text{[upred-defs]: } N_R(M) = ((\exists \; s< v, v' = u \; s< v < u \; s< v < N_1(M)) < s< v < (s< u \leq s< v'))\]

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

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

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

Healthiness condition for reactive design merge predicates

definition [upred-defs]: \(RDM(M) = R2m(N_R(R2m(M))) = N_R(R2m(M))\)

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

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

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

lemma nmerge-rd-is-R2m [closure]: \(M\) is \(R2m\)
by (metis Healthy-def \(\parallel\) R2m-nmerge-rd)

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

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

lemma merge-rd-is-RD3: \(M_R(M)\) is \(RD3\)
by (metis Healthy-Idempotent RD3-Idempotent RD3-def merge-rd-def)

lemma merge-rd-is-RD2: \(M_R(M)\) is \(RD2\)
by (simp add: RD3-implies-RD2 merge-rd-is-RD3)

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

lemma RDM-intro:
assumes \(M\) is \(R2m\) \(0\) \(\parallel\) \(M\) \(1\) \(\parallel\) \(M\) \(0\) \(\parallel\) \(M\) \(1\) \(\parallel\) \(M\) \(0\) \(\parallel\) \(M\) \(1\) \(\parallel\) \(M\) \(0\) \(\parallel\) \(M\) \(1\) \(\parallel\) \(M\)
shows $M$ is RDM
using assms
by (simp add: Healthy-def RDM-def ex-unrest unrest)

lemma RDM-unrests [unrest]:
assumes $M$ is RDM
shows $\exists \mathit{st} \cdot \mathit{P} \parallel M \parallel 1 - \mathit{ok} \parallel M \parallel \mathit{ok} \cdot \parallel M \parallel 0 - \mathit{ok} \parallel M$
$\exists \mathit{wait} \cdot \parallel M \parallel 1 - \mathit{wait} \parallel M \parallel \mathit{wait} \cdot \parallel M \parallel 0 - \mathit{wait} \parallel M$
by (subst Healthy-if[of assms, THEN sym], simp-all add: RDM-def unrest, rel-auto)+

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

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

lemma ex-st'-R2m-closed [closure]:
assumes $P$ is R2m
shows $\exists \mathit{st}' \cdot P$ is R2m
proof 
  have $R2m(\exists \mathit{st}' \cdot R2m P) = (\exists \mathit{st}' \cdot R2m P)$
    by (rel-auto)
  thus ?thesis
  by (metis Healthy-def' assms)
qed

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

lemma parallel-ok-cases:
$((P \parallel Q) :: M) =$
$((P^t \parallel Q^t) :: (M[true, true/$0 - \mathit{ok}, 1 - \mathit{ok}])) \lor$
$((P^f \parallel Q^f) :: (M[false, true/$0 - \mathit{ok}, 1 - \mathit{ok}])) \lor$
$((P^t \parallel Q^f) :: (M[true, false/$0 - \mathit{ok}, 1 - \mathit{ok}])) \lor$
$((P^f \parallel Q^f) :: (M[false, false/$0 - \mathit{ok}, 1 - \mathit{ok}]))$

proof 
  have $((P \parallel Q) :: M) = (\exists \mathit{ok}_0 \cdot (P \parallel Q)[<\mathit{ok}_0>/0 - \mathit{ok}'] :: M[<\mathit{ok}_0>/0 - \mathit{ok}])$
    by (subst seqr-middle[of left-uar ok], simp-all)
  also have $\ldots = (\exists \mathit{ok}_0 \cdot (P \parallel Q)[<\mathit{ok}_1>/0 - \mathit{ok}'][<\mathit{ok}_1>1 - \mathit{ok}'] :: (M[<\mathit{ok}_0>/0 - \mathit{ok}][<\mathit{ok}_1>/1 - \mathit{ok}]))$
    by (subst seqr-middle[of right-uar ok], simp-all)
  also have $\ldots = (\exists \mathit{ok}_0 \cdot (P[<\mathit{ok}_0>/\mathit{ok}'] \parallel Q[<\mathit{ok}_1>/\mathit{ok}'] :: (M[<\mathit{ok}_0>,<\mathit{ok}_1>/0 - \mathit{ok}, 1 - \mathit{ok}]))$
    by (rel-auto robust)
  also have $\ldots =$
    $((P^t \parallel Q^t) :: (M[true, true/$0 - \mathit{ok}, 1 - \mathit{ok}])) \lor$
    $((P^f \parallel Q^f) :: (M[false, true/$0 - \mathit{ok}, 1 - \mathit{ok}])) \lor$
    $((P^t \parallel Q^f) :: (M[true, false/$0 - \mathit{ok}, 1 - \mathit{ok}])) \lor$
    $((P^f \parallel Q^f) :: (M[false, false/$0 - \mathit{ok}, 1 - \mathit{ok}]))$
    by (simp add: true-alt-def THEN sym false-alt-def THEN sym disj-assoc utp-pred-laws.sup.left-commute utp-pred-laws.sup-commute u subst)
  finally show ?thesis .
qed
lemma skip-srea-ok-f [usubst]:
\[ R_1^{f} = R_1(\neg$ok) \]
by (rel-auto)

lemma nmerge0-rd-unrest [unrest]:
\[ S0-ok \not\in S N_0 M S1-ok \not\in S N_0 M \]
by (pred-auto)+

lemma parallel-assm-lemma:
assumes \( P \) is RD2
shows \( \text{pre}_s \vdash (P \parallel_{R(M)} Q) = ((\text{pre}_s \vdash P) \parallel_{N_0(M)} ; R_1(\text{true}) (\text{cmt}_s \vdash Q)) \)
\varepsilon ((\text{cmt}_s \vdash P) \parallel_{N_0(M)} ; R_1(\text{true}) (\text{pre}_s \vdash Q))

proof –
\begin{itemize}
\item have \( \text{pre}_s \vdash (P \parallel_{R(M)} Q) = \text{pre}_s \vdash ((P \parallel Q) ; M_{R(M)}) \)
\item by (simp add: par-by-merge-def)
\item also have \( \ldots = ((P \parallel Q) [true/false/$ok,$wait]] ; N_R M ; R_1(\neg$ok)) \)
\item by (simp add: merge-rd-def usubst, rel-auto)
\item also have \( \ldots = ((P [true/false/$ok,$wait]] \parallel Q [true/false/$ok,$wait]) ; N_1(M) ; R_1(\neg$ok)) \)
\item by (rel-auto robust, (metis)+)
\item also have \( \ldots = ((P [true/false/$ok,$wait]]) \parallel Q [true/false/$ok,$wait]) ; R_1(\neg$ok) \)
\item by (simp add: par-by-merge-def H2-2equiv Healthy-def)
\item hence \( P ; P_f = P_f \)
\item by (rel-auto)
\item have \( \neg A \Rightarrow \neg C \)
\item using \( P \) by (rel-auto)
\item moreover have \( \neg B \Rightarrow \neg D \)
\item by (rel-auto)
\item ultimately show \( ?thesis \)
\item by (simp add: impl-seq-mon)
\end{itemize}
qed

ultimately show \( ?thesis \)
by (simp add: subumption2)

qed

also have \( \ldots = ( \)
\[ ((\text{pre}_s \vdash P) \parallel_{s} (\text{cmt}_s \vdash Q)) \parallel ((N_0 M ; R_1(\text{true}))) \]
\varepsilon ((\text{cmt}_s \vdash P) \parallel_{s} (\text{pre}_s \vdash Q)) \parallel ((N_0 M ; R_1(\text{true})))
\item by (rel-auto, metis+)
\item also have \( \ldots = ( \)

68
\[(\langle \text{pre}_r \uparrow P \rangle \parallel N_0 \cdot M) ; \text{R1} (\text{true}) (\text{cmt}_R \uparrow Q)) \lor \\
(\langle \text{cmt}_R \uparrow P \rangle \parallel N_0 \cdot M) ; \text{R1} (\text{true}) (\text{pre}_r \uparrow Q))\]

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

finally show \(?thesis .

qed

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

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

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

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

also have \(\cdots = (\neg_r \text{pre}_R P)\)
  by (simp add: \text{R2c-not R2c-preR assms rea-not-def})

finally show \(?thesis .

qed

lemma \text{parallel-assm}:
assumes \(P \text{ is SRD} \text{ Q is SRD}\)
shows \(\text{pre}_R(P \parallel M_R(M) \cdot Q) = (\neg_r (\langle \neg_r \text{pre}_R(P) \rangle) \parallel N_0(M) ; \text{R1}(\text{true}) (\text{cmt}_R(Q)) ) \land \\
\neg_r (\text{cmt}_R(P) \parallel N_0(M) ; \text{R1}(\text{true}) (\neg_r \text{pre}_R(Q))))\)
(is \(?lhs = ?rhs\)
proof –

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

also have \(\cdots = \text{?rhs}\)
  by (simp add: \text{pre}_r-\text{SRD assms cmt}_R-def \text{rea-not-def Healthy-if closure unrest})

finally show \(?thesis .

qed

lemma \text{parallel-assm-unrest-wait}' [unrest]:
\([ \text{P is SRD; Q is SRD } ] \Rightarrow \$\text{wait} \\neg \text{pre}_R(P \parallel M_R(M) \cdot Q)\)
by (simp add: \text{parallel-assm, simp add: par-by-merge-def unrest})

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

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

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

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

lemma \text{parallel-commitment-lemma-1}:
assumes P is RD2
shows \( cmt_s \downarrow (P \parallel_{M_R(M)} Q) = (\) 
\( ((cmt_s \downarrow P) \parallel_{\text{\$ok} \wedge N_0 M} :: I_R^t (cmt_s \downarrow Q))) \lor \) 
\( ((\text{pre}_s \downarrow P) \parallel_{N_0 M} :: R1(\text{true}) (cmt_s \downarrow Q)) \lor \) 
\( ((cmt_s \downarrow P) \parallel_{N_0 M} :: R1(\text{true}) (\text{pre}_s \downarrow Q))) \) 

proof –

have \( cmt_s \downarrow (P \parallel_{M_R(M)} Q) = (P[\text{true, false}/\text{\$ok, \$wait}] \parallel_{(M_1 M)^t} Q[\text{true, false}/\text{\$ok, \$wait}]) \) by (simp add: usubst, rel-auto)
also have \( \ldots = ((P[\text{true, false}/\text{\$ok, \$wait}] \parallel_s Q[\text{true, false}/\text{\$ok, \$wait}]) :: (M_1 M)^t) \)
also have \( \ldots = ( \)
\( (((cmt_s \downarrow P) \parallel_s (cmt_s \downarrow Q)) :: ((M_1 M)^t[\text{true, true}/\text{\$0 - ok, \$1 - ok}]) \lor \)
\( (((\text{pre}_s \downarrow P) \parallel_s (cmt_s \downarrow Q)) :: (M_1 M)^t[\text{false, true}/\text{\$0 - ok, \$1 - ok}]) \lor \)
\( (((cmt_s \downarrow P) \parallel_s (\text{pre}_s \downarrow Q)) :: (M_1 M)^t[\text{true, true}/\text{\$0 - ok, \$1 - ok}]) \lor \)
\( (((\text{pre}_s \downarrow P) \parallel_s (\text{pre}_s \downarrow Q)) :: (M_1 M)^t[\text{false, false}/\text{\$0 - ok, \$1 - ok}]) \))
by (simp add: par-by-merge-def)
also have \( \ldots = ( \)
\( (((cmt_s \downarrow P) \parallel_s (cmt_s \downarrow Q)) :: ((M_1 M)^t[\text{true, true}/\text{\$0 - ok, \$1 - ok}]) \lor \)
\( (((\text{pre}_s \downarrow P) \parallel_s (cmt_s \downarrow Q)) :: (N_0 M) :: R1(\text{true}))) \lor \)
\( (((cmt_s \downarrow P) \parallel_s (\text{pre}_s \downarrow Q)) :: (N_0 M) :: R1(\text{true}))) \lor \)
\( (((\text{pre}_s \downarrow P) \parallel_s (\text{pre}_s \downarrow Q)) :: (N_0 M) :: R1(\text{true}))) \)
by (simp add: subst parallel-ok-cases, subst-tac)
also have \( \ldots = ( \)
\( (((cmt_s \downarrow P) \parallel_s (cmt_s \downarrow Q)) :: ((M_1 M)^t[\text{true, true}/\text{\$0 - ok, \$1 - ok}]) \lor \)
\( (((\text{pre}_s \downarrow P) \parallel_s (cmt_s \downarrow Q)) :: (N_0 M) :: R1(\text{true}))) \lor \)
\( (((cmt_s \downarrow P) \parallel_s (\text{pre}_s \downarrow Q)) :: (N_0 M) :: R1(\text{true}))) \lor \)
\( (((\text{pre}_s \downarrow P) \parallel_s (\text{pre}_s \downarrow Q)) :: (N_0 M) :: R1(\text{true}))) \)
by (simp add: JL1 JL2 JL3)
also have \( \ldots = ( \)
\( (((cmt_s \downarrow P) \parallel_s (cmt_s \downarrow Q)) :: ((M_1 M)^t[\text{true, true}/\text{\$0 - ok, \$1 - ok}]) \lor \)
\( (((\text{pre}_s \downarrow P) \parallel_s (cmt_s \downarrow Q)) :: (N_0 M) :: R1(\text{true}))) \lor \)
\( (((cmt_s \downarrow P) \parallel_s (\text{pre}_s \downarrow Q)) :: (N_0 M) :: R1(\text{true}))) \lor \)
\( (((\text{pre}_s \downarrow P) \parallel_s (\text{pre}_s \downarrow Q)) :: (N_0 M) :: R1(\text{true}))) \)
by (simp add: rel-auto)

proof –
from assms have \( \langle P \rangle \Rightarrow P^t \).
by (metis RD2-def H2-equivalence Healthy-def)

hence \( P' : P^t \Rightarrow P^t \).
by (rel-auto)

have \( (?C_4 \Rightarrow ?C^3) \) (is \( \langle？A :: ？B \Rightarrow ？C :: ？D \rangle \) )

proof –

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

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

qed

thus \( ?\text{thesis} \)
by (simp add: substumption2)

qed

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

qed

lemma parallel-commitment-lemma-2:
assumes P is RD2
shows \( cmt_s \downarrow (P \parallel_{M_R(M)} Q) = (\) 
\( (((cmt_s \downarrow P) \parallel_{\text{\$ok} \wedge N_0 M} :: I_R^t (cmt_s \downarrow Q))) \lor \) 
\( ((\text{pre}_s \downarrow P) \parallel_{N_0 M} :: R1(\text{true}) (cmt_s \downarrow Q)) \lor \) 
\( ((cmt_s \downarrow P) \parallel_{N_0 M} :: R1(\text{true}) (\text{pre}_s \downarrow Q))) \) by (simp add: parallel-commitment-lemma-I assms parallel-assm-lemma)

lemma parallel-commitment-lemma-3:
\( M \text{ is } R1\text{m} \Rightarrow (\text{\$ok} \wedge N_0 M) :: I_R^t \text{ is } R1\text{m} \)
by (rel-simp, safe, metis+)

lemma parallel-commitment:
assumes P is SRD Q is SRD M is RDM
shows \( \text{cmd}_R(P \parallel M_R(M) \parallel Q) =\( \text{pre}_R(P) \parallel \text{cmd}_R(Q) \) \)
by (simp add: parallel-commitment-lemma-2 parallel-commitment-lemma-3 Healthy-if assms cmdR-def pre-s-SRD closure rea-impl-def disj-comm unrest)

theorem parallel-reactive-design:
assumes P is SRD Q is SRD M is RDM
shows \( (P \parallel M_R(M) \parallel Q) = \text{R}_s(\neg_r \text{cmd}_R(P) \parallel N_0(M) \parallel \text{cmd}_R(Q) \) \)
by (metis Healthy-def NSRD-is-SRD SRD-as-reactive-design assms(1) assms(2) assms(3) par-rdes-NSRD)
also have \( \text{lhs} = \text{rhs} \)
by (simp add: parallel-assm parallel-commitment design-export-spec assms, rel-auto)
finally show \( \text{thesis} \).

qed

lemma parallel-percondition-lemma1:
\( \exists s \cdot N_0(M) \parallel \text{cmd}_R(P) \parallel \text{cmd}_R(Q) \parallel \text{cmd}_R(M_R(M) \parallel Q) \)
by (rel-simp, safe, metis+)

proof
have \( \text{thesis} \).

qed

lemma parallel-percondition-lemma2:
assumes M is RDM
shows \( \exists s \cdot N_0(M) \parallel \text{cmd}_R(P) \parallel \text{cmd}_R(Q) \parallel \text{cmd}_R(M_R(M) \parallel Q) \)
by (rel-simp, safe, metis+)

proof
have \( \text{thesis} \).

qed

lemma parallel-percondition-lemma3:
\( (\text{thesis}) \)
by (rel-simp, safe, metis+)

lemma parallel-percondition [rules]:
fixes M :: \( (\text{'s,'t::{trace,α}) \) \)

71
assumes $P$ is SRD $Q$ is SRD $M$ is RDM
shows $\text{peri}_R(P \parallel_{M} (M) \parallel_{Q}) = (\text{peri}_R(P \parallel_{M} M \parallel_{Q}) \Rightarrow \text{peri}_R(P \parallel_{M} \exists s_t \cdot M \text{peri}_R(Q))$

proof –

have $\text{peri}_R(P \parallel_{M} (M) \parallel_{Q}) = (\text{peri}_R(P \parallel_{M} M \parallel_{Q}) \Rightarrow \text{peri}_R(P \parallel_{M} \exists s_t \cdot M \text{peri}_R(Q))$

by (simp add: peri-cmt-def parallel-commitment SRD-healths assms usubst unrest assms)

also have $\ldots = (\text{peri}_R(P \parallel_{M} M \parallel_{Q}) \Rightarrow \text{peri}_R(P \parallel_{M} \exists s_t \cdot M \text{peri}_R(Q))$

by (simp add: parallel-pericondition-lemma1)

also have $\ldots = (\text{peri}_R(P \parallel_{M} M \parallel_{Q}) \Rightarrow \text{peri}_R(P \parallel_{M})$

by (simp add: parallel-pericondition-lemma2 assms)

also have $\ldots = (\text{peri}_R(P \parallel_{M} M \parallel_{Q}) \Rightarrow \text{peri}_R(P \parallel_{M} \exists s_t \cdot M \text{peri}_R(Q))$

by (simp add: par-by-merge-alt-def parallel-pericondition-lemma3 seqr-or-distr)

also have $\ldots = (\text{peri}_R(P \parallel_{M} M \parallel_{Q}) \Rightarrow \text{peri}_R(P \parallel_{M} \exists s_t \cdot M \text{peri}_R(Q))$

by (simp add: seqr-right-one-point-true seqr-right-one-point-false cmt-def post-def peri-def usubst unrest assms)

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

finally show $\text{thesis}$.qed

lemma parallel-postcondition-lemma1:

$(\exists s_t \cdot M) \Rightarrow \text{peri}_R(P \parallel_{M} \exists s_t \cdot M \text{peri}_R(Q))$

(is $?lhs = ?rhs$)

proof –

have $?lhs = (\exists s_t \cdot M) \Rightarrow \text{peri}_R(P \parallel_{M} \exists s_t \cdot M \text{peri}_R(Q))$

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

by (metis Healthy-if R1m-def RDM-R1m assms trl-invt-law inf-commute)

finally show $\text{thesis}$.qed

lemma parallel-postcondition-lemma2:

assumes $M$ is RDM
shows $(N_0(M) \parallel_{true/false/sok',\text{wait}}] = ((\neg s_0-wait \land \neg s_1-wait) \land M)$

proof –

have $(N_0(M) \parallel_{true/false/sok',\text{wait}}] = ((\neg s_0-wait \land \neg s_1-wait) \land s_0 \geq u \land M)$

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

by (metis Healthy-if R1m-def RDM-R1m assms trl-invt-law inf-commute)

finally show $\text{thesis}$.qed

lemma parallel-postcondition [rdes]:

fixes $M :: (s',t::trace',a)$ rsp merge
assumes $P$ is SRD $Q$ is SRD $M$ is RDM
shows $\text{post}_R(P \parallel M_R(M) \parallel Q) = (\text{pre}_R(P \parallel M_R(M) \parallel Q) \Rightarrow \text{post}_R(P \parallel M \parallel \text{post}_R(Q)))$
proof
have $\text{post}_R(P \parallel M_R(M) \parallel Q) = (\text{pre}_R(P \parallel M_R(M) \parallel Q) \Rightarrow C)$
  by (simp add: post-cmt-def parallel-commitment assms usubst unrest SRD-healths)
also have ... = $\text{pre}_R(P \parallel M_R(M) \parallel Q) \Rightarrow C$
also have ... = $(\text{pre}_R(P \parallel M_R(M) \parallel Q) \Rightarrow \text{post}_R(P \parallel M \parallel \text{post}_R(Q))$
  by (simp add: par-by-merge-alt-def sepq-right-one-point-false usubst unrest cmt)
finally show $?thesis$.
qed

lemma parallel-precondition-lemma:
fixes $M : (',t::trace,\alpha) \text{ rsp merge}$
assumes $P$ is NSRD $Q$ is NSRD $M$ is RDM
shows $((\neg \text{pre}_R(P)) \parallel N_0(M) ;; R1(true) \parallel \text{post}_R(Q)) = ((\neg \text{pre}_R(P)) \parallel N_0(M) ;; R1(true) \parallel (\text{pre}_R(Q) \circ \text{post}_R(Q)))$
proof
have $((\neg \text{pre}_R(P)) \parallel N_0(M) ;; R1(true) \parallel \text{post}_R(Q)) = ((\neg \text{pre}_R(P)) \parallel N_0(M) ;; R1(true))$
  by (simp add: wait-cond-post-cmt)
also have ... = $((\neg \text{pre}_R(P)) \parallel N_0(M) ;; R1(true))$
  by (simp add: par-by-merge-alt-def)
also have ... = $((\neg \text{pre}_R(P)) \parallel N_0(M) ;; R1(true))$
  by (simp add: alpha)
also have ... = $((\neg \text{pre}_R(P)) \parallel N_0(M) ;; R1(true))$
  by (simp add: rel-auto)
thus $?thesis$ by simp
qed

lemma parallel-precondition-lemma:
fixes $M : (',t::trace,\alpha) \text{ rsp merge}$
assumes $P$ is NSRD $Q$ is NSRD $M$ is RDM
shows $((\neg \text{pre}_R(P)) \parallel N_0(M) ;; R1(true) \parallel \text{post}_R(Q)) = ((\neg \text{pre}_R(P)) \parallel N_0(M) ;; R1(true))$
proof
have $((\neg \text{pre}_R(P)) \parallel N_0(M) ;; R1(true) \parallel \text{post}_R(Q)) = ((\neg \text{pre}_R(P)) \parallel N_0(M) ;; R1(true))$
  by (simp add: wait-cond-post-cmt)
also have ... = $((\neg \text{pre}_R(P)) \parallel N_0(M) ;; R1(true))$
  by (simp add: par-by-merge-alt-def)
also have ... = $((\neg \text{pre}_R(P)) \parallel N_0(M) ;; R1(true))$
  by (simp add: alpha)
also have ... = $((\neg \text{pre}_R(P)) \parallel N_0(M) ;; R1(true))$
  by (simp add: rel-auto)
thus $?thesis$ by simp
qed
\[\neg R \text{ pre } P \land [\text{post } Q]_1 \land \$v <' = u \$v \};; M \};; R1 \text{ true}\]

(is (?P_1 \lor ?P_2) = (?Q_1 \lor ?Q_2))

\begin{verbatim}
proof -
  have ?P_1 = ([\neg R \text{ pre } P]_0 \land [\text{peri } Q]_1 \land \$v <' = u \$v) \land (M \land \$\text{wait'} = \text{wait}) \land R1 \text{ 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 usubst closure assms)
  have ?P_2 =(((\neg R \text{ pre } P]_0 \land [\text{post } Q]_1 \land \$v <' = u \$v) \land (M \land \neg \$\text{wait'} = \text{wait'}) \land R1 \text{ true})
    by (substitution seqr-bool-split[of left-var \text{wait}], simp-all add: unrest 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 unrest assms closure assms)
  from 1 2 show ?thesis by simp
qed
\end{verbatim}

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

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

**Lemma swap-merge-rd':**
\[\text{swap}_m \land N_R(M) = N_R(\text{swap}_m \land M)\]
by (rel-blast)

**Lemma swap-merge-rd:**
\[\text{swap}_m \land N_R(M) = N_R(\text{swap}_m \land M)\]
by (simp add: merge-rd-def seqr-assoc[THEN sym] swap-merge-rd')

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

**Lemma nmerge-rd1-merge3:**
assumes \(M \text{ is RDM}\)
shows \(M3(N_1(M)) = (\$ok' = u (\$0-ok \land \$1-0-ok \land \$1-1-ok) \land \$\text{wait'} = u (\$0-\text{wait} \lor \$1-0-\text{wait} \land \$1-1-\text{wait}) \land M3(M))\)

\begin{verbatim}
proof -
  have M3(N_1(M)) = M3(\$ok' = u (\$0-ok \land \$1-ok) \land \$\text{wait'} = u (\$0-\text{wait} \lor \$1-\text{wait}) \land \$tr < u \$tr' \land (\{\$0-ok, \$1-ok, \$ok', \$0-\text{wait}, \$1-\text{wait}, \$\text{wait'}, \$\text{wait'}\} \\ RDM(M)))
    by (simp add: nmerge-rd1-def nmerge-rd0-def assms Healthy-if)
  also have ... = M3(\$ok' = u (\$0-ok \land \$1-ok) \land \$\text{wait'} = u (\$0-\text{wait} \lor \$1-\text{wait}) \land RDM(M))
    by (rel-blast)
  also have ... = (\$ok' = u (\$0-ok \land \$1-0-ok \land \$1-1-ok) \land \$\text{wait'} = u (\$0-\text{wait} \lor \$1-0-\text{wait} \land \$1-1-\text{wait}) \land M3(RDM(M)))
    by (rel-blast)
\end{verbatim}

74
also have ... = (\$ok' =_u (\$0-ok \land \$1-0-ok \land \$1-1-ok) \land \$\text{wait'} =_u (\$0-\text{wait} \lor \$1-0-\text{wait} \\
\lor \$1-1-\text{wait}) \land M\beta(M))
  
  by (simp add: assms Healthy-if)

finally show \text{thesis}.

qed

lemma nmerge-\text{rd-merge3}:
\text{M\beta(N\beta(M))} = (3 \$\text{st}_\beta \cdot \$\text{v'} =_u \$\text{v}_\beta) \land \$\text{\text{wait}}_\beta \land M\beta(N\beta(M)) < \$\text{ok}_\beta \land (\$\text{tr}_\beta < _u \$\text{tr'} _u)

by (rel-blast)

lemma swap-\text{merge-RDM-closed} [\text{closure}):

\text{assumes } M \text{ is RDM}
\text{shows } swap_m :: M \text{ is RDM}

proof –

\text{have } \text{RDM}(swap_m :: RDM(M)) = (swap_m :: RDM(M))
  
  by (rel-auto)

thus \text{thesis}

  by (metis \text{Healthy-def'} \text{assms})

qed

lemma \text{parallel-precondition}:

\text{fixes } M :: ('s, 't::trace, 'a) \text{ rsp merge}
\text{assumes } P \text{ is NSRD } Q \text{ is NSRD } M \text{ is RDM}

\text{shows } \text{pre}_R(P \parallel_{M\beta(M)} Q) = 

\neg_r ((\neg_r \text{pre}_R P) \parallel_M \text{R1(true) peri}_R Q) \land 
\neg_r ((\neg_r \text{pre}_R P) \parallel_M \text{R1(true) post}_R Q) \land 
\neg_r ((\neg_r \text{pre}_R Q) \parallel_{\text{swap}_m} \text{M} :: \text{R1(true) peri}_R P) \land 
\neg_r ((\neg_r \text{pre}_R Q) \parallel_{\text{swap}_m} \text{M} :: \text{R1(true) post}_R P))

proof –

\text{have } a: (\neg_r \text{pre}_R(P)) \parallel_{N_0(M)} \text{R1(true) cmtr}_{R}(Q) = 

\neg_r \text{pre}_R(P) \parallel_M \text{R1(true) peri}_R Q \lor (\neg_r \text{pre}_R P) \parallel_M \text{R1(true) post}_R Q)

  by (simp add: \text{parallel-precondition-lemma \text{assms}})

\text{have } b: (\neg_r \text{cmtr}_{R} P \parallel_{N_0} M :: \text{R1 true (\neg_r \text{pre}_R Q)}) = 

(\neg_r \text{cmtr}_{R}(Q)) \parallel_{N_0(\text{swap}_m :: M)} \text{R1(true) cmtr}_{R}(P))

  by (simp add: \text{swap-nmerge-\text{rd0}}[THEN \text{sym}] \text{seqr-\text{assoc}[THEN \text{sym}]} \text{par-by-\text{merge-def \text{par-sep-\text{swap}}})

\text{have } c: (\neg_r \text{pre}_R(Q)) \parallel_{\text{swap}_m} \text{M} :: \text{R1(true) peri}_R P \lor (\neg_r \text{pre}_R Q) \parallel_{\text{swap}_m} \text{M} :: \text{R1(true) post}_R P)

  by (simp add: \text{parallel-precondition-lemma \text{closure \text{assms}})

show \text{thesis}

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

qed

Weakest Parallel Precondition

\text{definition } wrR ::
\langle 't::trace, 'a \rangle \text{ hrel-rp} \Rightarrow 
\langle 't :: \text{trace}, 'a \rangle \text{ \text{merge} } \Rightarrow 
\langle 't, 'a \rangle \text{ hrel-rp} \Rightarrow 
\langle 't, 'a \rangle \text{ hrel-rp (- \text{wr}_{R}('t') \cdot [60,0,61]) 61}

\text{where [upred-defs]}: Q \text{ wr}_{R}(M) P = (\neg_r ((\neg_r P) \parallel_M \text{R1(true) Q}))

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

\textbf{lemma R2-rea-not:} R2(\neg_r \ P) = (\neg_r R2(P))
by \ (\text{rel-auto})

\textbf{lemma wrR-R2-lemma:}
\textbf{assumes} P is R2 Q is R2 M is R2m
\textbf{shows} ((\neg_r P) \parallel_M Q) ;; R1(true_h) is R2
proof –
\hspace{1em}have ((\neg_r P) \parallel_M Q) is R2
by \ (\text{simp add: closure assms})
thus \ ?thesis
by \ (\text{simp add: closure})
Qed

\textbf{lemma wrR-R2 [closure]:}
\textbf{assumes} P is R2 Q is R2 M is R2m
\textbf{shows} Q \text{ wr}_R(M) \ P is R2
proof –
\hspace{1em}have ((\neg_r P) \parallel_M Q) ;; R1(true_h) is R2
by \ (\text{simp add: wrR-R2-lemma assms})
thus \ ?thesis
by \ (\text{simp add: wrR-def wrR-R2-lemma par-by-merge-seq-add closure})
Qed

\textbf{lemma wrR-RR [closure]:}
\textbf{assumes} P is RR Q is RR M is RDM
\textbf{shows} Q \text{ wr}_R(M) \ P is RR
apply \ (\text{rule RR-intro})
apply \ (\text{simp-all add: unrest assms closure wrR-def rpred})
apply \ (\text{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

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

\textbf{lemma wppR-choice [wp]:} (P \lor Q) \text{ wr}_R(M) \ R = (P \text{ wr}_R(M) \ R \land Q \text{ wr}_R(M) \ R)
proof –
\hspace{1em}have (P \lor Q) \text{ wr}_R(M) \ R =
\hspace{2.5em}(\neg_r ((\neg_r R) ;; U0 \lor (P ;; U1 \lor Q ;; U1) \land \$v_{<\cdot} =_u \$v) ;; M ;; true_r)
by \ (\text{simp add: wrR-def par-by-merge-def seqr-or-distl})
also have ... = (\neg_r ((\neg_r R) ;; U0 \lor P ;; U1 \land \$v_{<\cdot} =_u \$v \lor (\neg_r R) ;; U0 \lor Q ;; U1 \land \$v_{<\cdot} =_u \$v) ;; M ;; true_r)
by \ (\text{simp add: conj-disj-distr utp-pred-laws.inf-sup-distrib2})
also have ... = (\neg_r (((\neg_r R) ;; U0 \land P ;; U1 \land \$v_{<\cdot} =_u \$v) ;; M ;; true_r \lor
((\neg_r R) ;; U0 \land Q ;; U1 \land \$v_{<\cdot} =_u \$v) ;; M ;; true_r))

76
by (simp add: segr-or-distl)
also have \ldots = (P \wpp R(M) R \land Q \wpp R(M) R)
  by (simp add: \wpp R-def par-by-merge-def)
finally show \ \thesis .
qed

lemma \wpp R-miracle \ [wp]: false \wpp R(M) P = true_r
  by (simp add: \wpp R-def)

lemma \wpp R-true \ [wp]: P \wpp R(M) true_r = true_r
  by (simp add: \wpp R-def)

lemma parallel-precondition-ur \ [rdes]:
  assumes P is NSRD Q is NSRD M is RDM
  shows \pre R(P \parallel M R(M) Q) = (\peri R(Q) \wpp R(M) \pre R(P) \land
      \peri R(P) \wpp R(swap_m :: M) \pre R(Q) \land \pre R(P) \wpp R(swap_m :: M) \pre R(Q))
  by (simp add: assms parallel-precondition \wpp R-def)

lemma parallel-rdes-def \ [rdes-def]:
  assumes P is RC P_2 is RR P_3 is RR Q_1 is RC Q_2 is RR Q_3 is RR
      \$st^* :: P_2 \$st^* :: Q_2
           M is RDM
  shows \r_1(P_1 \parallel P_2 \parallel P_3) \parallel M R(M) \r_2(Q_1 \parallel Q_2)
      = \r_1 (\{(Q_1 \parallel Q_2) \wpp R(M) P_1 \land (Q_1 \parallel Q_3) \wpp R(M) P_1 \land
              (P_1 \parallel P_2) \wpp R(swap_m :: M) Q_1 \land (P_1 \parallel P_3) \wpp R(swap_m :: M) Q_1) \parallel
      ((P_1 \parallel P_2) \parallel M (Q_1 \parallel Q_2) \parallel (P_1 \parallel P_2) \parallel M (Q_1 \parallel Q_3) \parallel
      ((P_1 \parallel P_3) \parallel M (Q_1 \parallel Q_3))) \parallel M (\peri R(Q_3)) \parallel
      \{ M \mapsto \peri R(Q_3) \}
  proof \ldots
    have \ ?lhs = \r_1 (\pre R \ ?lhs \parallel \peri R ?lhs \parallel \pre R ?lhs)
    by (simp add: \peri R-def rdes closure unrest assms closure)
    also have \ldots = ?rhs
    by (simp add: rdes closure unrest assms, rel-auto)
  finally show \ ?thesis .
qed

lemma Miracle-parallel-left-zero:
  assumes P is SRD M is RDM
  shows Miracle \parallel M P = Miracle
  proof \ldots
    have \pre R(Miracle \parallel M P) = true_r,
      by (simp add: parallel-assm wait'-cond-idem rdes closure assms)
    moreover hence cmt R(Miracle \parallel M P) = false
      by (simp add: rdes closure wait'-cond-idem SRD-healthy assms)
    ultimately have Miracle \parallel M P = \r_1(true_r \parallel false)
      by (metis NSRD-iff SRD-reactive-design-alt assms par-rdes-NSRD srdes-theory-continuous.weak.top-closed)
  thus \ ?thesis
    by (simp add: Miracle-def R1-design-R1-pre)
qed

lemma Miracle-parallel-right-zero:
  assumes P is SRD M is RDM
  shows P \parallel M Miracle = Miracle
  proof \ldots
    have \pre R(P \parallel M Miracle) = true_r,
      by (simp add: Miracle-def R1-design-R1-pre)
  qed
by (simp add: wait’-cond-idem parallel-assm rdes closure assms)
moreover hence \( \text{cnt}_R(P \parallel_M \text{Miracle}) = \text{false} \)
by (simp add: wait’-cond-idem rdes closure SRD-healths assms)
ultimately have \( P \parallel_M \text{Miracle} = \text{R}_s(\text{true}, \vdash \text{false}) \)
by (metis NSRD-iff SRD-reactive-design-alt assms par-rdes-NSRD srdes.thoery-continuous.weak.top-closed)
thus \(?\text{thesis}\)
by (simp add: Miracle-def R1-design-R1-pre)
qed

8.1 Example basic merge

definition BasicMerge :: \(((\text{‘s, ‘t::trace, unit}) \text{rsp}) \text{merge} (N_B)\) where
  [upred-defs]: BasicMerge = \((\text{str}_< \leq_u \text{str’} \land \text{str’} - \text{str}_< =_u \text{0-tr} - \text{str}_< \land \text{str’} - \text{str}_< =_u \text{1-tr} - \text{str}_< \land \text{str’} =_u \text{0-tr})\)
abbreviation rbasic-par \((- \parallel_B - [85,86] 85\) where
  \( P \parallel_B Q \equiv P \parallel_M(N_B) \parallel Q \)

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

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

lemma BasicMerge’-calc:
  assumes \( \text{soc’} \not\equiv P \text{wait’} \not\equiv P \text{sok’} \not\equiv Q \text{wait’} \not\equiv Q \) \(P\) is \(R_2\) \(Q\) is \(R_2\)
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 \(P;((\exists \text{soc’};\text{wait’}) \cdot R_2(P)) = P\) (is \(?P’ = -\)
  by (simp add: ex-unrest ex-plus Healthy-if assms)
  have \(Q;((\exists \text{soc’};\text{wait’}) \cdot R_2(Q)) = Q\) (is \(?Q’ = -\)
  by (simp add: ex-unrest ex-plus Healthy-if assms)
  have \(?P’ \parallel_N_B ?Q’ = ((\exists \text{st’} \cdot ?P’) \land (\exists \text{st’} \cdot ?Q’) \land \text{st’} =_u \text{st})\)
  by (simp add: par-by-merge-alt-def, rel-auto, blast+)
  thus \(?\text{thesis}\)
  by (simp add: \(P \parallel Q\))
qed

8.2 Simple parallel composition

definition rea-design-par ::
  \((\text{‘s, ‘t::trace, ‘a}) \text{hrel-rsp} \Rightarrow (\text{‘s, ‘t’, ‘a}) \text{hrel-rsp} \Rightarrow (\text{‘s, ‘t’, ‘a}) \text{hrel-rsp} (\text{infixr} \parallel_R 85)\)
where [upred-defs]: \(P \parallel_R Q = \text{R}_s((P \text{pre}_R(P) \land P \text{pre}_R(Q)) \vdash (\text{cnt}_R(P) \land \text{cnt}_R(Q)))\)

lemma RHS-design-par:
  assumes \( \text{soc’} \not\equiv P_1 \not\equiv P_2 \)
shows \( \text{R}_s((P_1 \vdash Q_1) \parallel_R \text{R}_s(P_2 \vdash Q_2) = \text{R}_s((P_1 \land P_2) \vdash (Q_1 \land Q_2))\)
proof –
  have \(\text{R}_s((P_1 \vdash Q_1) \parallel_R \text{R}_s(P_2 \vdash Q_2) = \text{R}_s((P_1[\text{true},\text{false}/\text{soc},\text{wait}] \vdash Q_1[\text{true},\text{false}/\text{soc},\text{wait}]) \parallel_R \text{R}_s(P_2[\text{true},\text{false}/\text{soc},\text{wait}] \vdash Q_2[\text{true},\text{false}/\text{soc},\text{wait}]))\)
  by (simp add: RHS-design-ok-wait)
  
78
also from \textit{assms}

\begin{verbatim}
have ... := 
  \text{R} _n( (R1 (R2c (P_1)) \land R1 (R2c (P_2)))[true, false/$\$ok, $\$wait] \vdash 
  \langle R1 (R2c (P_1 \Rightarrow Q_1)) \land R1 (R2c (P_2 \Rightarrow Q_2)) \rangle)
apply \text{simp add: rea-design-par-def rea-pre-RHS-design rea-cmt-RHS-design unsubst unrest \textit{assms})
apply \text{rule cong[of \textit{R}_n, \textit{R}_m], simp}
using \textit{assms} apply \text{(rel-auto)}
done
\end{verbatim}
also have ... =
  \text{R} _n((R2c(P_1) \land R2c(P_2)) \vdash 
  \langle R1 (R2s (P_1 \Rightarrow Q_1)) \land R1 (R2s (P_2 \Rightarrow Q_2)) \rangle)
by \text{(metis (no-types, hide-lams) R1-R2s-R2c-R1-conj R1-design-R1-pre RHS-design-ok-wait)}
also have ... = \text{R} _n((P_1 \land P_2) \vdash (P_1 \Rightarrow Q_1) \land (P_2 \Rightarrow Q_2))
by \text{(simp (no-types, lifting) R1-conj R2s-conj RHS-design-export-R1 RHS-design-export-R2s)}
also have ... = \text{R} _n((P_1 \land P_2) \vdash (Q_1 \land Q_2))
by \text{(rule cong[of \textit{R}_n, \textit{R}_m], simp, rel-auto)}
finally show \text{thesis}.
qed

\textbf{lemma RHS-tri-design-par:}
assumes $\$ok \not\approx P_1 \not\approx P_2$
shows $\text{R} _n(P_1 \vdash Q_1 \land R_1) \parallel_R \text{R} _n(P_2 \vdash Q_2 \land R_2) = \text{R} _n((P_1 \land P_2) \vdash (Q_1 \land Q_2) \land (R_1 \land R_2))$
by \text{(simp add: RHS-design-par \textit{assms} unrest wait'-cond-conj-exchange)}

\textbf{lemma RHS-tri-design-par-RR [rdes-def]:}
assumes $P_1$ is RR $P_2$ is RR
shows $\text{R} _n(P_1 \vdash Q_1 \land R_1) \parallel_R \text{R} _n(P_2 \vdash Q_2 \land R_2) = \text{R} _n((P_1 \land P_2) \vdash (Q_1 \land Q_2) \land (R_1 \land R_2))$
by \text{(simp add: RHS-tri-design-par unrest \textit{assms})}

\textbf{lemma RHS-comp-assoc:}
assumes $P$ is NRD $Q$ is NRD $R$ is NRD
shows $(P \parallel_R Q) \parallel_R R = P \parallel_R Q \parallel_R R$
by \text{(rdes-eq rus cls: \textit{assms})}

\end{verbatim}

\section{Productive Reactive Designs}

\textbf{theory utp-rdes-productive}
\textbf{imports utp-rdes-parallel}
\textbf{begin}

\section{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.

\textbf{definition} \textit{Productive} :: \texttt{('s, 't::trace, 'a) hrel-rsp} \Rightarrow \texttt{('s, 't, 'a) hrel-rsp where}
[upred-defs]: \textit{Productive} (P) = \texttt{P} \parallel_R \texttt{R}_n(true \vdash true \circ (\$tr \prec_u \$str'))

\textbf{lemma} \textit{Productive-RHS-design-form:}
assumes $\$ok \not\approx P \not\approx Q \not\approx R$

\end{verbatim}

79
shows $\text{Productive}(\mathbf{R}_s(P \vdash Q \circ R)) = \mathbf{R}_s(P \vdash Q \circ (R \land \$tr <_u \$tr'))$

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

lemma Productive-form:
$\text{Productive}(\text{SRD}(P)) = \mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \circ (\text{post}_R(P) \land \$tr <_u \$tr'))$

proof –
have $\text{Productive}(\text{SRD}(P)) = \mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \circ \text{post}_R(P)) \parallel_R \mathbf{R}_s(\text{true} \vdash \text{true} \circ (\$tr <_u \$tr'))$
  by (simp add: Productive-def SRD-as-reactive-tri-design)
also have ... = $\mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \circ (\text{post}_R(P) \land \$tr <_u \$tr'))$
  by (simp add: RHS-tri-design-par unrest)
finally show ?thesis .

qed

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

lemma Productive-intro:
assumes $P$ is SRD ($\$tr <_u \$tr' \subseteq (\text{pre}_R(P) \land \text{post}_R(P)) \$wait' \not\vdash \text{pre}_R(P)$
shows $P$ is Productive

proof –
have $P: \mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \circ (\text{post}_R(P) \land \$tr <_u \$tr')) = P$
  by (simp add: SRD-reactive-tri-design assms(1))

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

qed

lemma Productive-refines-tr-increase:
assumes $P$ is SRD $P$ is Productive $\$wait' \not\vdash \text{pre}_R(P)$
shows ($\$tr <_u \$tr' \subseteq (\text{pre}_R(P) \land \text{post}_R(P))$

proof –
have $\text{post}_R(P) = \text{post}_R(\mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \circ (\text{post}_R(P) \land \$tr <_u \$tr')))$
  by (metis Healthy-def Productive-form assms(1) assms(2))
also have ... = $R1(R2c(\text{pre}_R(P) \Rightarrow (\text{post}_R(P) \land \$tr <_u \$tr')))$
  by (simp add: rea-post-RHS-design unrest usubst assms, rel-auto)
also have ... = $R1((\text{pre}_R(P) \Rightarrow (\text{post}_R(P) \land \$tr <_u \$tr')))$
  by (simp add: R2c-impl R2c-preR R2c-postR R2c-and R2c-tr-less-tr' assms)
also have ($\$tr <_u \$tr' \subseteq (\text{pre}_R(P) \land ...)$
  by (rel-auto)
finally show ?thesis .

qed

lemma Continuous-Productive [closure]: Continuous Productive
by (simp add: Continuous-def Productive-def, rel-auto)
9.2 Reactive design calculations

**Lemma** \texttt{preR-Productive [rdes]}:
\begin{itemize}
\item \textbf{Assumptions} \( P \text{ is SRD} \)
\item \textbf{Shows} \( \texttt{pre}_R(\text{Productive}(P)) = \texttt{pre}_R(P) \)
\item \textbf{Proof} –
\begin{itemize}
\item \texttt{have} \( \texttt{pre}_R(\text{Productive}(P)) = \texttt{pre}_R(\texttt{NSRD}(\texttt{Productive}(P)) \heartsuit \text{peri}_R(P) \odot (\text{post}_R(P) \land \$tr <_u \$tr') \)\)
\item \textbf{by} (\texttt{metis Healthy-def Productive-form assms})
\item \textbf{thus} \( \text{thesis} \)
\item \textbf{by} (\texttt{simp add: rea-pre-RHS-design usubst unrest R2c-not R2c-preR R1-preR Healthy-if assms})
\end{itemize}
\end{itemize}
\texttt{qed}

**Lemma** \texttt{periR-Productive [rdes]}:
\begin{itemize}
\item \textbf{Assumptions} \( P \text{ is NSRD} \)
\item \textbf{Shows} \( \texttt{peri}_R(\text{Productive}(P)) = \texttt{peri}_R(P) \)
\item \textbf{Proof} –
\begin{itemize}
\item \texttt{have} \( \texttt{peri}_R(\text{Productive}(P)) = \texttt{peri}_R(\texttt{NSRD}(\texttt{Productive}(P)) \heartsuit \text{peri}_R(P) \odot (\text{post}_R(P) \land \$tr <_u \$tr') \)\)
\item \textbf{by} (\texttt{metis Healthy-def NSRD-is-SRD Productive-form assms})
\item \textbf{also have} \( \ldots = R1 \ (R2c \ (\text{post}_R(P) \Rightarrow \text{peri}_R(P))) \)
\item \textbf{by} (\texttt{simp add: rea-peri-RHS-design usubst unrest R2c-not assms closure})
\item \textbf{also have} \( \ldots = (\texttt{pre}_R(P \Rightarrow \text{peri}_R(P)) \)
\item \textbf{by} (\texttt{simp add: R1-rea-impl R2c-rea-impl R2c-preR R2c-peri-SRD R1-peri-SRD assms closure R1-tr-less-tr' R2c-tr-less-tr'})
\item \textbf{finally show} \( \text{thesis} \)
\item \textbf{by} (\texttt{simp add: SRD-peri-under-pre assms unrest closure})
\end{itemize}
\texttt{qed}

**Lemma** \texttt{postR-Productive [rdes]}:
\begin{itemize}
\item \textbf{Assumptions} \( P \text{ is NSRD} \)
\item \textbf{Shows} \( \texttt{post}_R(\text{Productive}(P)) = (\text{pre}_R(P) \Rightarrow \text{post}_R(P) \land \$tr <_u \$tr' \)\)
\item \textbf{Proof} –
\begin{itemize}
\item \texttt{have} \( \texttt{post}_R(\text{Productive}(P)) = \texttt{post}_R(\texttt{NSRD}(\texttt{Productive}(P)) \heartsuit \text{peri}_R(P) \odot (\text{post}_R(P) \land \$tr <_u \$tr') \)\)
\item \textbf{by} (\texttt{metis Healthy-def NSRD-is-SRD Productive-form assms})
\item \textbf{also have} \( \ldots = R1 \ (R2c \ (\text{post}_R(P) \Rightarrow \text{peri}_R(P))) \)
\item \textbf{by} (\texttt{simp add: rea-post-RHS-design usubst unrest assms closure})
\item \textbf{also have} \( \ldots = (\text{pre}_R(P \Rightarrow \text{peri}_R(P) \land \$tr' >_u \$tr) \)
\item \textbf{by} (\texttt{simp add: R1-rea-impl R2c-rea-impl R2c-preR R2c-and R1-extend-conj' R2c-post-SRD R1-post-SRD assms closure R1-tr-less-tr' R2c-tr-less-tr'})
\item \textbf{finally show} \( \text{thesis} \)
\end{itemize}
\texttt{qed}

**Lemma** \texttt{preR-frame-seq-export}:
\begin{itemize}
\item \textbf{Assumptions} \( P \text{ is NSRD} \ P \text{ is Productive} Q \text{ is NSRD} \)
\item \textbf{Shows} \( (\text{pre}_R \land (\text{post}_R \land \text{post}_R) \odot Q) = (\text{pre}_R \land (\text{post}_R \land Q)) \)
\item \textbf{Proof} –
\begin{itemize}
\item \texttt{have} \( (\text{pre}_R \land (\text{post}_R \land Q) = (\text{pre}_R \land ((\text{pre}_R \Rightarrow \text{post}_R) \land Q)) \)
\item \textbf{by} (\texttt{simp add: SRD-post-under-pre assms closure unrest})
\item \textbf{also have} \( \ldots = (\text{pre}_R \land ((\neg \text{pre}_R \Rightarrow Q) \lor (\text{pre}_R \Rightarrow R1(\text{post}_R(P)) \land Q))) \)
\item \textbf{by} (\texttt{simp add: NSRD-is-SRD R1-post-SRD assms(1) rea-impl-def seqr-or-distl R1-preR Healthy-if})
\item \textbf{also have} \( \ldots = (\text{pre}_R \land ((\neg \text{pre}_R \Rightarrow Q) \lor (\text{pre}_R \land \text{post}_R) \land Q)) \)
\item \textbf{Proof} –
\begin{itemize}
\item \texttt{have} \( (\text{pre}_R \land \neg \text{pre}_R \Rightarrow R1 \text{ true}) \)
\item \textbf{by} (\texttt{simp add: R1-preR rea-not-or})
\item \textbf{then show} \( \text{thesis} \)
\item \textbf{by} (\texttt{metis (no-types, lifting) R1-def conj-comm disj-comm disj-conj-distr rea-impl-def seqr-or-distl utp-pred-laws.inf-top-left utp-pred-laws.sup.left-idem})
\end{itemize}
\end{itemize}
\texttt{qed}
proof

also have \( \vdash (\text{pre}_R \ P \land (\neg \text{pre}_R \ P) \lor (\text{pre}_R \ P \land \text{post}_R \ P) \land \ Q)) \)

by \((\text{simp add: NSRD-neg-pre-left-zero assms closure SRD-healths})\)
also have \( \vdash (\text{pre}_R \ P \land (\text{pre}_R \ P \land \text{post}_R \ P) \land \ Q) \)

by \((\text{rel-blast})\)
finally show \(?thesis\) ..

qed

9.3 Closure laws

lemma Productive-rdes-intro: assumes \((\text{str} <_u \text{str} \prime) \subseteq R \ (\text{ok} \prime \land \text{str} \prime \land \text{ok} \land \text{str} \land \text{wait} \land \text{str} \prime \land \text{wait} \prime \land \text{P}) \)

shows \((\text{R}_s (P \vdash Q \land R)) \) is Productive

proof \((\text{rule Productive-intro})\)
show \(\text{R}_s (P \vdash Q \land R)\) is SRD
by \((\text{simp add: RHS-tri-design-is-SRD assms})\)

from assmss(1) show \((\text{str} >_u \text{str}) \subseteq (\text{pre}_R (\text{R}_s (P \vdash Q \land R)) \land \text{post}_R (\text{R}_s (P \vdash Q \land R)))\)
apply \((\text{simp add: rea-pre-RHS-design rea-post-RHS-design usubst assms unrest})\)
using assmss(1) apply \((\text{rel-auto})\)
apply \fastforce

done

show \(\text{str} \prime \land \text{pre}_R (\text{R}_s (P \vdash Q \land R))\)
by \((\text{simp add: rea-pre-RHS-design rea-post-RHS-design usubst R1-def R2c-def R2s-def assms unrest})\)

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 \(P \vdash \text{R}\ Q \land \text{R} \land \text{R} \land \text{R} \land \text{R}^4\)

shows \((\text{R}_s (P \vdash Q \land R)) \) is Productive

using assms by \((\text{simp add: Productive-rdes-intro R4-iff-refine unrest})\)

lemma Productive-Miracle [closure]: Miracle is Productive unfolding Miracle-tri-def Healthy-def
by \((\text{subst Productive-RHS-design-form, simp-all add: unrest})\)

lemma Productive-Chaos [closure]: Chaos is Productive unfolding Chaos-tri-def Healthy-def
by \((\text{subst Productive-RHS-design-form, simp-all add: unrest})\)

lemma Productive-intChoice [closure]:

assumes \(P \vdash \text{SRD} \ Q \land \text{SRD} \ Q \land \text{SRD} \ Q \land \text{Productive}\)

shows \(P \land Q \land \text{Productive}\)

proof

have \(P \land Q =\)

\(\text{R}_s (\text{pre}_R (P \vdash \text{peri}_R (P) \land \text{post}_R (P) \land \text{str} <_u \text{str} \prime)) \land \text{R}_s (\text{pre}_R (Q) \vdash \text{peri}_R (Q) \land \text{post}_R (Q) \land \text{str} <_u \text{str} \prime))\)

by \((\text{metis Healthy-if Productive-form assms})\)
also have \((\text{pre}_R \ P \land \text{pre}_R \ Q) \vdash (\text{peri}_R \ P \lor \text{peri}_R \ Q) \land (\text{post}_R \ Q \land \text{str} >_u \text{str})\)

by \((\text{simp add: RHS-tri-design-choice})\)
also have \((\text{pre}_R \ P \land \text{pre}_R \ Q) \vdash (\text{peri}_R \ P \lor \text{peri}_R \ Q) \land ((\text{post}_R \ P \lor \text{post}_R \ Q) \land \text{str} >_u \text{str})\)

\(\)
by (rule cong[of R, simp, rel-auto])
also have ... is Productive
  by (simp add: Healthy-def Productive-RHS-design-form unrest)
finally show thesis .
qed

lemma Productive-cond-reas [closure]:
assumes P is NSRD Q is SRD Q is Productive
shows P ∨ b ⊢ Q is Productive
proof -
  have P ∨ b ⊢ Q =
    R_s (pre_R(P) ⊢ peri_R(P) ◁ (post_R(P) ∧ $tr < u $tr')) ∨ b ⊢ R_s (pre_R(Q) ⊢ peri_R(Q) ◁ (post_R(Q) ∧ $tr < u $tr'))
    by (metis Healthy-if Productive-RHS-design-form assms)
  also have ... = R_s ((pre_R(P) ∧ b ⊢ R pre_R Q) ⊢ (peri_R(P) ⊢ b ⊢ R peri_R Q) ◁ ((post_R P ∧ $tr' > u $tr) ∨ b ⊢ R (post_R Q) ∧ $tr' > u $tr))
    by (simp add: cond-sread-form)
  also have ... = R_s ((pre_R(P) ∧ b ⊢ R pre_R Q) ⊢ (peri_R(P) ∧ b ⊢ R peri_R Q) ◁ ((post_R P ∧ b ⊢ R (post_R Q)) ∧ $tr' > u $tr))
    by (rule cong[of R_s, simp, rel-auto])
  also have ... is Productive
    by (simp add: Healthy-def Productive-RHS-design-form unrest)
finally show thesis .
qed

lemma Productive-seq-1 [closure]:
assumes P is NSRD Q is NSRD
shows P ;; Q is Productive
proof -
  have P ;; Q = R_s (pre_R(P) ⊢ peri_R(P) ◁ (post_R(P) ∧ $tr < u $tr')) ;; R_s (pre_R(Q) ⊢ peri_R(Q) ◁ (post_R(Q))))
    by (metis Healthy-def NSRD-is-SRD SRD-reactive-tri-design Productive-form assms)
  also have ... = R_s ((pre_R(P) ∧ (post_R P ∧ $tr' > u $tr) wp_p pre_R Q) ⊢ (peri_R(P) ∨ ((post_R P ∧ $tr' > u $tr) ;; peri_R(Q)) ◁ ((post_R P ∧ $tr' > u $tr) ;; post_R(Q)))
    by (simp add: RHS-tri-design-composition-wp rpred unrest closure assms wp NSRD-NEG-pre-left-zero SRD-healths ex-unrest wp-reas-def disj-apred-def)
  also have ... = R_s ((pre_R(P) ∧ (post_R P ∧ $tr' > u $tr) wp_p pre_R Q) ⊢ (peri_R(P) ∨ ((post_R P ∧ $tr' > u $tr) ;; peri_R(Q)) ◁ ((post_R P ∧ $tr' > u $tr) ;; post_R Q ∧ $tr' > u $tr))
    proof -
      have ((post_R P ∧ $tr' > u $tr) ;; R1(post_R Q)) = ((post_R P ∧ $tr' > u $tr) ;; R1(post_R Q) ∧ $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

lemma Productive-seq-2 [closure]:
assumes P is NSRD Q is NSRD Q is Productive
shows $P \Rightarrow Q$ is Productive

proof

have $P \Rightarrow Q = R_s\,(\text{pre}_R(P) \vdash \text{peri}_R(P) \circ (\text{post}_R(P))) \Rightarrow R_s\,(\text{pre}_R(Q) \vdash \text{peri}_R(Q) \circ (\text{post}_R(Q) \land \$tr < u \$\text{tr}'))$

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

also have $R_s\,(\text{pre}_R(P \land \text{post}_R P \circ \text{pre}_R Q) \vdash (\text{peri}_R P \lor (\text{post}_R P \Rightarrow \text{peri}_R Q)) \circ (\text{post}_R P \Rightarrow (\text{post}_R Q \land \$tr' > u \$tr'))$

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

also have $R_s\,(\text{pre}_R(P \land \text{post}_R P \circ \text{pre}_R Q) \vdash (\text{peri}_R P \lor (\text{post}_R P \Rightarrow \text{peri}_R Q)) \circ (\text{post}_R P \Rightarrow (\text{post}_R Q \land \$tr' > u \$tr'))$

proof

have $(R1\,(\text{post}_R P) \Rightarrow (\text{post}_R Q \land \$tr' > u \$tr') \land \$\text{tr}' > u \$\text{tr}) = (R1\,(\text{post}_R P) \Rightarrow (\text{post}_R Q \land \$\text{tr}' > u \$\text{tr})$

by (rel-auto)

thus $?thesis$

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

qed

also have $\ldots$ is Productive

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

finally show $?thesis$.

qed

end

10 Guarded Recursion

theory utp-rdes-guarded

imports utp-rdes-productive

begin

10.1 Traces with a size measure

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

class size-trace = trace + size +

assumes

size-zero: $\text{size}(0) = 0$ and

size-nzero: $s > 0 \Rightarrow \text{size}(s) > 0$ and

size-plus: $\text{size}(s + t) = \text{size}(s) + \text{size}(t)$

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

begin

lemma size-mono: $s \leq t \Rightarrow \text{size}(s) \leq \text{size}(t)$

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

lemma size-strict-mono: $s < t \Rightarrow \text{size}(s) < \text{size}(t)$

by (metis cancel-ab-semitrivial-group.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: $xs < ys \Rightarrow (\forall zs. \left[ ys = xs + zs; \text{size}(zs) > 0 \right] \Rightarrow \text{thesis}) \Rightarrow \text{thesis}$
by (metis local.diff-add-cancel-left' local.less-iff local.minus-gr-zero-iff local.size-nzero)

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

Both natural numbers and lists are measurable trace algebras.

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

instance list :: (type) size-trace
by (intro-classes, simp-all add: zero-list-def less-list-def' plus-list-def prefix-length-less)
syntax -usize :: logic \Rightarrow logic (size_u"(-")
translations size_u(t) == CONST uop CONST size t

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

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

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

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

lemma GuardedI: \( \bigwedge (F(X) \land gVRT(n+1)) = (F(X \land gVRT(n)) \land gVRT(n+1)) \bigwedge \implies \text{Guarded F} \)
by (simp add: Guarded-def)

Guarded reactive designs yield unique fixed-points.

theorem guarded-fp-uniq:
assumes mono F F \in \([id]_H \rightarrow [SRD]_H \text{ Guarded F}
shows \( \mu F = \nu F \)
proof
have constr F gvrt
using assms
by (auto simp add: constr-def gvrt-chain Guarded-def tcontr-alt-def')
hence (\$tr \leq_u \$tr' \land \mu F) = (\$tr \leq_u \$tr' \land \nu F)
apply (rule constr-fp-uniq)
apply (simp add: assms)
using gvrt-limit apply blast
done
moreover have (\$tr \leq_u \$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 $?thesis$
  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 $?thesis$
    by (metis Healthy-def $R1-def$ conj-comm)

qed

ultimately show $?thesis$
  by (simp)

qed

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

lemma UINF-Guarded [closure]:
  assumes $\bigwedge P. P \in A \Longrightarrow \text{Guarded} P$
  shows Guarded $(\lambda X. \bigwedge P\in A \cdot P(X))$
proof (rule GuardedI)
  fix $X \ n$
  have $Y. (P \in A \cdot P \ Y \land \text{gvr}(n+1)) = (P \in A \cdot (P \ Y \land \text{gvr}(n+1)) \land \text{gvr}(n+1))$
proof
  fix $Y$
  let $?lhs = (P \in A \cdot P \ Y \land \text{gvr}(n+1))$ and $?rhs = (P \in A \cdot (P \ Y \land \text{gvr}(n+1)) \land \text{gvr}(n+1))$
  have $a$: $?lhs[\text{false}/\text{ok}] = $?rhs[\text{false}/\text{ok}]
    by (rel-auto)
  have $b$: $?lhs[\text{true}/\text{ok}][\text{true}/\text{wait}] = $?rhs[\text{true}/\text{ok}][\text{true}/\text{wait}]
    by (rel-auto)
  have $c$: $?lhs[\text{true}/\text{ok}][\text{false}/\text{wait}] = $?rhs[\text{true}/\text{ok}][\text{false}/\text{wait}]
    by (rel-auto)
  show $?lhs = 0$\n    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 $(P \in A \cdot (P \ X \land \text{gvr}(n+1))) \land \text{gvr}(n+1)) = (P \in A \cdot (P \ (X \land \text{gvr}(n)) \land \text{gvr}(n+1))) \land \text{gvr}(n+1))$
proof
  have $(P \in A \cdot (P \ X \land \text{gvr}(n+1))) = (P \in A \cdot (P \ (X \land \text{gvr}(n)) \land \text{gvr}(n+1)))$
proof (rule UINF-cong)
  fix $P$ assume $P \in A$
  thus $(P \ X \land \text{gvr}(n+1)) = (P \ (X \land \text{gvr}(n)) \land \text{gvr}(n+1))$
  using Guarded-def assms by blast

qed

thus $?thesis$ by simp

qed

ultimately show $(P \in A \cdot P \ X \land \text{gvr}(n+1)) = (P \in A \cdot (P \ (X \land \text{gvr}(n)))) \land \text{gvr}(n+1))$
  by simp

qed

lemma intChoice-Guarded [closure]:
  assumes Guarded $P$ Guarded $Q$

86
A tail recursive reactive design with a productive body is guarded.

**lemma** cond-srea-Guarded [closure]:
**assumes** Guarded P Guarded Q
**shows** Guarded (λ X. P(X) ∩ 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 n

**have** a:(P :: SRD(X) ∧ gvt (Suc n))[(false/sok) = (P :: SRD(X) ∧ gvt (Suc n))][false/sok]
**by** (simp add: usbst closure SRD-left-zero-1 assms)

**have** b:(P :: SRD(X) ∧ gvt (Suc n))[(true/sok) = (P :: SRD(X) ∧ gvt (Suc n))][true/swait]
**by** (simp add: usbst closure SRD-left-zero-2 assms)

**have** c:(P :: SRD(X) ∧ gvt (Suc n))[(false/swait) = (P :: SRD(X) ∧ gvt (Suc n))][false/swait]

---

**have** 1:(P::SRD X)[true/swait ∧ gvt (Suc n)][true/false/sok,$\$wait] =
(P::SRD X)[true/swait ∧ gvt (Suc n)][true/false/sok,$\$wait]
**by** (metis (no-types, lifting) Healthy-def R3h-wait-true SRD-halts SRD-idem)

**have** 2:(P::SRD X)[false/swait ∧ gvt (Suc n)][true/false/sok,$\$wait] =
(P::SRD X)[false/swait ∧ gvt (Suc n)][true/false/sok,$\$wait]
**by** (simp add: usbst closure SRD-left-zero assms)

---

**have** exp:A Y:(’s, ’t, ’a) hrel-rsp. (P[false/swait’] :: (SRD Y)[false/swait ∧ gvt (Suc n)][true/false/sok,$\$wait]
= (((ν_r pre R P) :: (SRD Y)[false/swait] ∨ (post R P ∧ $tr >, $tr ) :: (SRD Y)[true/false/sok,$\$wait])
∧ gvt (Suc n)))[true/false/sok,$\$wait]

---

**have** (P::SRD Y)[false/swait’ ∧ gvt (Suc n)][true/false/sok,$\$wait] =
((R(ν_r pre R P), | post R P ∧ $tr <, $tr ) :: (SRD Y)[false/swait’]) :: (SRD Y)[false/swait]
∧ gvt (Suc n)))[true/false/sok,$\$wait]
**by** (metis (no-types) Healthy-def Productive-form assms(1) assms(2) NSRD-is-SRD)

**also have** ...

---

**also have** ...

---

87
∧ gvr (Suc n) [true, false/$ok$, $\$\text{wait}$]

by (simp add: impl-all-def R2c-disj R1-disj R2c-not assms closure R2c-and
R2c-preR rea-not-def R1-extend-conj R2c-ok' R2c-post-SRD R1-tr-less-tr' R2c-tr-less-tr')
also have ... =
(((∀ P ( SRD(Y)))[false/$\$\text{wait}$] ∨ (ok' ∧ post R P ∧ $\text{str}' > u$ $\text{str}$)) ;; (SRD Y)[false/$\$\text{wait}$]) ∨ gvr (Suc n)][true,false/$ok$, $\$\text{wait}$]
by (simp add: usubst unrest assms closure seqr-or-distl NSRD-neg-pre-left-zero SRD-healths)
also have ... =
(((∀ P ( SRD(Y)))[false/$\$\text{wait}$] ∨ (post R P ∧ $\text{str}' > u$ $\text{str}$)) ;; (SRD Y)[true,false/$ok$, $\$\text{wait}$])
∧ gvr (Suc n)]true,false/$ok$, $\$\text{wait}$]

proof
have (post R P ∧ $\text{str}' > u$ $\text{str}$) ;; (SRD Y)[false/$\$\text{wait}$] =
((post R P ∧ $\text{str}' > u$ $\text{str}$) ∧ ok' = u true) ;; (SRD Y)[false/$\$\text{wait}$]
by (rel-blast)
also have ... = (post R P ∧ $\text{str}' > u$ $\text{str}$)[true/$\$\text{ok}'$] ;; (SRD Y)[false/$\$\text{wait}$][true/$\$\text{ok}$]
using seqr-left-one-point[of ok (post R P ∧ $\text{str}' > u$ $\text{str}$) True (SRD Y)[false/$\$\text{wait}$]]
by (simp add: true-alt-def)[THEN sym])
finally show ?thesis by (simp add: usubst unrest)

qed

finally show (P)[false/$\$\text{wait}$] ;; (SRD Y)[false/$\$\text{wait}$] ∧ gvr (Suc n)][true,false/$ok$, $\$\text{wait}$] =
(((∀ P ( SRD(Y)))[false/$\$\text{wait}$] ∨ (post R P ∧ $\text{str}' > u$ $\text{str}$)) ;; (SRD Y)[true,false/$ok$, $\$\text{wait}$])
∧ gvr (Suc n)]true,false/$ok$, $\$\text{wait}$]

qed

have 1:((post R P ∧ $\text{str}' > u$ $\text{str}$)) ;; (SRD X)[true,false/$ok$, $\$\text{wait}$] ∧ gvr (Suc n)) =
((post R P ∧ $\text{str}' > u$ $\text{str}$) ;; (SRD (X ∧ gvr n))[true,false/$ok$, $\$\text{wait}$] ∧ gvr (Suc n))
apply (rel-auto)
apply (rename-tac tr st more ok wait tr' st' more' tr0 st0 more0 ok')
apply (rule-tac x=x0 in exI, rule-tac x=x0 in exI, rule-tac x=x0 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=x0 in exI, rule-tac x=x0 in exI, rule-tac x=x0 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:(∀ P ( SRD(Y)))[false/$\$\text{wait}$] = (∀ P ( SRD(Y)))[false/$\$\text{wait}$]
by (simp add: NSRD-neg-pre-left-zero closure assms SRD-healths)
show ?thesis
by (simp add: exp 1 2 utp-pred-laws.inf-sup-distrib2)

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

  also have \(\ldots = ((P[true/\text{wait}] ;; (\text{SRD} \land \text{gvert} \ n))[true/false/\text{ok}/\text{wait}] \lor \)
  \((P[false/\text{wait}] ;; (\text{SRD} \land \text{gvert} \ (n+1))[true/false/\text{ok}/\text{wait}] \))
  by (simp add: 1 2)

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

  also have \(\ldots = (P ;; (\text{SRD} \land \text{gvert} \ n)) \land \text{gvert} \ (n+1))[true/false/\text{ok}/\text{wait}] \)
  by (subst seqr-bool-split[of wait], simp add: subst)
  finally show \(?thesis \) by (simp add: subst)

qed

qed (P ;; \text{SRD}(X) \land \text{gvert} \ (Suc \ n)) = (P ;; \text{SRD}(X) \land \text{gvert} \ (Suc \ n))
apply (rule-tac bool-eq-splitI[of in-var \text{ok}])
apply (simp-all add: a)
apply (rule-tac bool-eq-splitI[of in-var wait])
apply (simp-all add: b c)
done

10.3 Tail recursive fixed-point calculations

declare upred-semiring.power-Suc [simp]

lemma mu-csp-form-1 [rdes]:
  fixes P :: \('s, 't:size-trace,'α\) hrel-rsp
  assumes P is NSRD P is Productive
  shows \((\mu X \cdot P ;; \text{SRD}(X)) = (\prod i \cdot P ^* \ (i+1)) ;; \text{Miracle} \)
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: \((\lambda X. \ P ;; \text{SRD} \ X) \in \prod i \rightarrow [\text{SRD}]_H \)
  by (blast intro: funcsetI closure assms)
  with 1 2 have \((\mu X \cdot P ;; \text{SRD}(X)) = (\nu X \cdot P ;; \text{SRD}(X)) \)
  by (simp add: guarded-fp-uniq Guarded-if-Productive[OF assms] funcsetI closure)
  also have \(\ldots = (\prod i \cdot ((\lambda X. \ P ;; \text{SRD} \ X) ^* \ i)) \)
  by (simp add: sup-continuous-lfp 1 sup-continuous-Continuous false-upred-def)
  also have \(\ldots = ((\lambda X. \ P ;; \text{SRD} \ X) ^* \ 0) \) \(\land (\prod i \cdot ((\lambda X. \ P ;; \text{SRD} \ X) ^* \ (i+1)) \)
  by (subst Sup-power-expand, simp)
  also have \(\ldots = (\prod i \cdot (\lambda X. \ P ;; \text{SRD} \ X) ^* \ (i+1)) \)
  by (simp)
  also have \(\ldots = (\prod i. \ P ^* \ (i+1) ;; \text{Miracle} \)
  proof (rule SUP-cong, simp-all)

89
fix \( i \)

show \( P ;; \text{SRD} (((\lambda X. P ;; \text{SRD} X) ^^ i) \text{false}) = (P ;; P ^ \ast i) ;; \text{Miracle} \)

proof (induct \( i \))
  
  case 0
  then show ?case
    by (simp, metis srdes-cond-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 seqr-assoc [THEN sym], simp add: Healthy-if NSRD-is-SRD SRD-power-comp SRD-seqr-closure assms seqr-assoc [THEN sym], weak top-closed)

qed

also have ... = (\( i \in \text{UNIV} \cdot P ^ \ast i \)) ;; \text{Miracle}

by (simp only: seq-UINF-distl [THEN sym], simp add: ustar-def)

finally show ?thesis.

qed

lemma mu-csp-form-NSRD [closure]:
  
  fixes \( P :: (s, \cdot t::\text{size-trace}, \cdot \alpha) \text{hrel-rsp} \)

  assumes \( P \text{ is NSRD} \ P \text{ is Productive} \)

  shows \( (\mu X \cdot P ;; \text{SRD}(X)) \text{ is NSRD} \)

  by (simp add: mu-csp-form-1 assms closure ustar-def)

proof –
  have \( (\mu X \cdot P ;; \text{SRD}(X)) = (\bigcap i \in \text{UNIV} \cdot P ;; P ^ \ast i) ;; \text{Miracle} \)

    by (simp add: mu-csp-form-1 assms closure ustar-def)

  also have ... = (P ;; P^*) ;; \text{Miracle}

    by (simp only: seq-UINF-distl [THEN sym], simp add: ustar-def)

  finally show ?thesis.

qed

declare upred-semiring.power-Suc [simp del]

end

11 Reactive Design Programs

theory utp-rdes-prog

imports
  utp-rdes-normal
  utp-rdes-tactics
  utp-rdes-parallel
  utp-rdes-guarded
  UTP-KAT.utp-kleene

begin

11.1 State substitution

lemma srd-subst-RHS-tri-design [usubst]:
\[ [\sigma]_{R} \vdash \sigma_{R}(P \vdash Q \circ R) = \sigma_{R}(\langle [\sigma]_{R} \vdash P \rangle \vdash (\langle [\sigma]_{R} \vdash Q \rangle \circ (\langle [\sigma]_{R} \vdash R \rangle)) \]

by (rel-auto)

**Lemma srd-subst-SRD-closed [closure]:**

assumes \( P \) is SRD

shows \( [\sigma]_{R} \vdash P \) is SRD

**Proof**

have \( SRD([\sigma]_{R} \vdash (SRD \ P)) = [\sigma]_{R} \vdash (SRD \ P) \)

by (rel-auto)

thus \(?thesis\)

by (metis Healthy-def assms)

qed

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

\( \langle [\sigma] \rangle_{R} = [\sigma]_{R} \vdash \langle [\sigma] \rangle_{R} \)

by (rel-auto)

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

\( [\sigma]_{R} \vdash \langle [\sigma] \rangle_{R} \)

by (rel-auto)

**Lemma postR-srd-subst [rdes]:**

\( [\sigma]_{R} \vdash \langle [\sigma] \rangle_{R} \)

by (rel-auto)

**Lemma srd-subst-NSRD-closed [closure]:**

assumes \( P \) is NSRD

shows \( [\sigma]_{R} \vdash P \) is NSRD

by (rule NSRD-RC-intro, simp-all add: closure rdes assms unrest)

### 11.2 Assignment

**Definition assigns-srd :: \( 's \ usubst \Rightarrow ('s, 't::trace, 'a) \ hrel-rsp (\langle \cdot \rangle_{R}) \) where**

\[ upred-defs: \assigns-srd \sigma = R_{s}(true \vdash (\$tr' = u \ \$tr' \land \neg \$wait' \land \langle (\sigma)_{u} \rangle_{S} \land \Sigma_{S'} = u \ \Sigma_{S})) \]

**Syntax**

- assigns-srd :: svids \Rightarrow uexprs \Rightarrow logic \ (infixr := R 90)

**Translations**

- assigns-srd xs vs \Rightarrow CONST assigns-srd (-mk-usubst (CONST id) xs vs)
- assigns-srd x v \Rightarrow CONST assigns-srd (CONST subst-upd (CONST id) x v)
- assigns-srd x v \Rightarrow -assign-srd (-spvar x) v
- x,y := R u,v \Rightarrow CONST assigns-srd (CONST subst-upd (CONST subst-upd (CONST subst-upd (CONST id) (CONST spvar x) u) (CONST spvar y) v)

**Lemma assigns-srd-RHS-tri-des [rdes-def]:**

\( \langle \sigma \rangle_{R} = R_{s}(true \vdash false \circ \langle \sigma \rangle_{r}) \)

by (rel-auto)

**Lemma assigns-srd-NSRD-closed [closure]: \( \langle \sigma \rangle_{R} \) is NSRD**

by (simp add: rdes-def closure unrest)

**Lemma preR-assigns-srd [rdes]: preR(\langle \sigma \rangle_{R}) = true_{r}**

by (simp add: rdes-def rdes closure)
lemma periR-assigns-srd [rdes]: peri_\(_R(\sigma)_R) = false
  by (simp add: rdes-def rdes closure)

lemma postR-assigns-srd [rdes]: post_\(_R(\sigma)_R) = (\sigma)_r
  by (simp add: rdes-def rdes closure rpred)

11.3 Conditional

lemma preR-cond-srea [rdes]:
  pre_\(_R(P \triangleleft b \triangleright R Q) = (| b|_S < \land pre_\(_R(P) \lor [\neg b|_S < \land peri_\(_R(Q))
  by (rel-auto)

lemma periR-cond-srea [rdes]:
  assumes P is SRD Q is SRD
  shows peri_\(_R(P \triangleleft b \triangleright R Q) = (| b|_S < \land peri_\(_R(P) \lor [\neg b|_S < \land peri_\(_R(Q))
  proof
    have peri_\(_R(R1(P) \triangleleft b \triangleright R R1(Q)) = (| b|_S < \land peri_\(_R(P) \lor [\neg b|_S < \land peri_\(_R(Q))
      by (simp add: Healthy-if SRD-healths assms)
    thus ?thesis
      by (rel-auto)
  qed

lemma postR-cond-srea [rdes]:
  assumes P is SRD Q is SRD
  shows post_\(_R(P \triangleleft b \triangleright R Q) = (| b|_S < \land post_\(_R(P) \lor [\neg b|_S < \land post_\(_R(Q))
  proof
    have post_\(_R(R1(P) \triangleleft b \triangleright R R1(Q)) = (| b|_S < \land post_\(_R(P) \lor [\neg b|_S < \land post_\(_R(Q))
      by (simp add: Healthy-if SRD-healths assms)
    thus ?thesis
      by (rel-auto)
  qed

lemma NSRD-cond-srea [closure]:
  assumes P is NSRD Q is NSRD
  shows P \triangleleft b \triangleright R Q is SRD
  proof (rule NSRD-RC-intro)
    show P \triangleleft b \triangleright R Q is SRD
      by (simp add: closure assms)
    show pre_\(_R(P \triangleleft b \triangleright R Q) is RC
      proof
        have 1:([\neg b|_S < \lor \neg pre_\(_R(P)) \supset R1(true) = ([\neg b|_S < \lor \neg pre_\(_R(P)
          by (metis (no-types, lifting) NSRD-neg-pre-unit acxt-not assms1 seqr-or-distl st-lift-R1-true-right)
        have 2:([b|_S < \lor pre_\(_R(Q)) \supset R1(true) = ([b|_S < \lor \neg pre_\(_R(Q)
          by (simp add: NSRD-neg-pre-unit assms seqr-or-distl st-lift-R1-true-right)
        show ?thesis
          by (simp add: rdes closure assms)
      qed
    show $st’ \notin peri_\(_R(P \triangleleft b \triangleright R Q)
      by (simp add: rdes closure unrest)
  qed

11.4 Assumptions

definition AssumeR :: ’s cond ⇒ (’s, ’t::trace, ’a) hrel-rsp ([| - ]_R) where
  [upred-defs]: AssumeR b = II_\(R \triangleleft b \triangleright R Miracle
lemma AssumeR-rdes-def [rdes-def]:
\[ b^\top_R = R_s(true, \triangleright false \odot [b]^\top_R) \]
unfolding AssumeR-def by (rdes-eq)

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

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

lemma AssumeR-true: \([true]^\top_R = II \]
by (rel-auto)

lemma AssumeR-comp: \([b]^\top_R ;; [c]^\top_R = [b \land c]^\top_R \]
by (rdes-simp)

lemma AssumeR-choice: \([b]^\top_R \land [c]^\top_R = [b \lor c]^\top_R \]
by (rdes-eq)

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

lemma AssumeR-test [closure]: \(test_R [b]^\top_R \]
by (simp add: AssumeR-refine-skip nsrd-thy.utest-intro)

lemma Star-AssumeR: \([b]^\top_R^{\ast R} = II_R \]
by (simp add: AssumeR-NSRD Star-assume-test)

lemma AssumeR-choice-skip: \(II_R \sqsubseteq [b]^\top_R = II_R \]
by (rdes-eq)

lemma cond-srea-AssumeR-form:
assumes P is NSRD Q is NSRD
shows \(P \triangleright [b]^\top_R = ([b]^\top_R ;; P \sqcap [\neg b]^\top_R ;; Q) \]
by (rdes-eq cls: assms)

lemma cond-srea-insert-assume:
assumes P is NSRD Q is NSRD
shows \(P \triangleright [b]^\top_R \quad Q = ([b]^\top_R ;; P \triangleright [\neg b]^\top_R ;; Q) \)
by (simp add: AssumeR-NSRD AssumeR-comp NSRD-seqr-closure RA1 assms cond-srea-AssumeR-form)

lemma AssumeR-cond-left:
assumes P is NSRD Q is NSRD
shows \([b]^\top_R ;; (P \triangleright [b]^\top_R) = ([b]^\top_R ;; P) \]
by (rdes-eq cls: assms)

lemma AssumeR-cond-right:
assumes P is NSRD Q is NSRD
shows \([\neg b]^\top_R ;; (P \triangleright [\neg b]^\top_R) = ([\neg b]^\top_R ;; Q) \]
by (rdes-eq cls: assms)

11.5 Guarded commands

definition GuardedCommR :: 's cond ⇒ ('s, 't::trace, 'α) hrel-rsp ⇒ ('s, 't, 'α) hrel-rsp (- →_R - [85, 86] 85) where
gcmd-def[rdes-def]: GuardedCommR g A = A g v_R Miracle
lemma gcmd-false [simp]: \((false \rightarrow_R A) = Miracle\)
unfolding gcmd-def by (pred-auto)

lemma gcmd-true [simp]: \((true \rightarrow_R A) = A\)
unfolding gcmd-def by (pred-auto)

lemma gcmd-SRD:
assumes \(A\) is SRD
shows \((g \rightarrow_R A)\) is SRD
by (simp add: gcmd-def SRD-cond-srea assms srdes-theory-continuous.weak.top-closed)

lemma gcmd-NSRD [closure]:
assumes \(A\) is NSRD
shows \((g \rightarrow_R A)\) is NSRD
by (simp add: gcmd-def NSRD-cond-srea assms NSRD-Miracle)

lemma gcmd-Productive [closure]:
assumes \(A\) is NSRD \(A\) is Productive
shows \((g \rightarrow_R A)\) is Productive
by (simp add: gcmd-def closure assms)

lemma gcmd-seq-distr:
assumes \(B\) is NSRD
shows \((g \rightarrow_R A) ; B = (g \rightarrow_R (A ; B))\)
by (simp add: gcmd-def cond-st-distr assms)

lemma gcmd-nondet-distr:
assumes \(A\) is NSRD \(B\) is NSRD
shows \((g \rightarrow_R (A \sqcap B)) = (g \rightarrow_R A) \sqcap (g \rightarrow_R B)\)
by (rdes-eq cls: assms)

lemma AssumeR-as-gcmd:
\([b]^+_R = b \rightarrow_R I_R\)
by (rdes-eq)

12 Generalised Alternation

definition AlternateR
:: 'a set ⇒ ('a ⇒ 's upred) ⇒ ('a ⇒ ('s, 't::trace, 'a) hrel-rsp) ⇒ ('s, 't, 'a) hrel-rsp ⇒ ('s, 't, 'a) hrel-rsp
where
[upred-defs, rdes-def]: AlternateR \(I g A B = (\prod i \in I \cdot ((g i) \rightarrow_R (A i))) \sqcap ((\forall i \in I \cdot g i)) \rightarrow_R B\)

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

syntax
-alternateR-els :: pttrn ⇒ logic ⇒ logic ⇒ logic ⇒ logic ⇒ logic (if \(R \in\cdot\cdot\cdot\rightarrow\cdot\cdot\cdot\rightarrow\) else \(\cdot\cdot\cdot\))
-alternateR :: pttrn ⇒ logic ⇒ logic ⇒ logic ⇒ logic (if \(R \in\cdot\cdot\cdot\rightarrow\cdot\cdot\cdot\rightarrow\) else \(\cdot\cdot\cdot\))
-alternateR-commR-els :: gcomms ⇒ logic ⇒ logic (if \(R \in\cdot\cdot\cdot\rightarrow\cdot\cdot\cdot\rightarrow\) else \(\cdot\cdot\cdot\))
-alternateR-commR :: gcomms ⇒ logic (if \(R \in\cdot\cdot\cdot\rightarrow\cdot\cdot\cdot\rightarrow\) else \(\cdot\cdot\cdot\))
translations

\[
\begin{align*}
& \text{if } R \in I \cdot g \to A \text{ else } B \text{ fi } \to \text{CONST AlternateR } I (\lambda i \cdot g) (\lambda i \cdot A) B \\
& \text{if } R \in I \cdot g \to A \text{ else } B \text{ fi } \to \text{CONST AlternateR } I (\lambda i \cdot g) (\lambda i \cdot A) (\text{CONST Chaos}) \\
& \text{if } R \in I \cdot (g(i) \to A \text{ else } B \text{ fi } \isFalse \to \text{CONST AlternateR } I g (\lambda i \cdot A) B \\
& \text{-altgcommR cs } \to \text{CONST AlternateR-list cs (CONST Chaos)} \\
& \text{-altgcommR } (-\text{gcomm-show cs}) \isFalse \to \text{CONST AlternateR-list cs (CONST Chaos)} \\
& \text{-altgcommR-els cs } P \isFalse \to \text{CONST AlternateR-list cs P} \\
\end{align*}
\]

lemma AlternateR-NSRD-closed [closure]:

assumes \( \bigwedge i \cdot i \in I \implies A \text{ is NSRD} B \text{ is NSRD} \)

shows \( \text{if } R \in I \cdot g \to A \text{ else } B \text{ fi } \text{is NSRD} \)

proof (cases \( I = \{\} \))

case True
then show ?thesis by (simp add: AlternateR-def assms)

case False
then show ?thesis by (simp add: AlternateR-def closure assms)

qed

lemma AlternateR-empty [simp]:

\( (\text{if } R \in \{\} \cdot g \to A \text{ else } B \text{ fi} ) = B \)

proof (rdes-simp)

lemma AlternateR-Productive [closure]:

assumes \( \bigwedge i \cdot i \in I \implies A \text{ is Productive} B \text{ is Productive} \)

shows \( \text{if } R \in I \cdot g \to A \text{ else } B \text{ fi } \text{is Productive} \)

proof (cases \( I = \{\} \))

case True
then show ?thesis
by (simp add: assms(4))

case False
then show ?thesis
by (simp add: AlternateR-def closure assms)

qed

lemma AlternateR-singleton:

assumes \( A k \text{ is NSRD} B \text{ is NSRD} \)

shows \( (\text{if } R \in \{k\} \cdot g \to A \text{ else } B \text{ fi} ) = (A(k) \triangleleft g(k) \triangleright R B) \)

by (simp add: AlternateR-def, rdes-eq cls: assms)

Convert an alternation over disjoint guards into a cascading if-then-else

lemma AlternateR-insert-cascade:

assumes \( \bigwedge i \cdot i \in I \implies A \text{ is NSRD} \)

\[ A \text{ k is NSRD} B \text{ is NSRD} \]

\[ (g(k) \land (\forall i \in I \cdot g(i))) = \text{false} \]

shows \( (\text{if } R \in \text{insert } k \cdot g \to A \text{ else } B \text{ fi} ) = (A(k) \triangleleft g(k) \triangleright R (\text{if } R \in I \cdot g(i) \to A(i) \text{ else } B \text{ fi} )) \)

proof (cases \( I = \{\} \))

case True
then show ?thesis by (simp add: AlternateR-singleton assms)
next

case False
have 1: \((\prod i \in I \cdot g i \rightarrow_R A i) = (\prod i \in I \cdot g i \rightarrow_R R_s(pre_R(A i) \triangleright peri_R(A i) \odot post_R(A i)))\)
  by (simp add: NSRD-is-SRD SRD-reactive-tri-design assms(1) cong: UINF-cong)
from assms(4) show ?thesis
  by (simp add: AlternateR-def 1 False cong: UINF-cong)
qed

12.1 Choose

definition choose-srd :: (′s,′t::trace,′α) hrel-rsp (choose\_R) where
  [upred-defs, rdes-def]: choose\_R = R_s(true \_r \triangleright false \_r false)

lemma preR-choose [rdes]: pre\_R(choose\_R) = true\_r
  by (rel-auto)

lemma periR-choose [rdes]: peri\_R(choose\_R) = false
  by (rel-auto)

lemma postR-choose [rdes]: post\_R(choose\_R) = true\_r
  by (rel-auto)

lemma choose-srd-SRD [closure]: choose\_R is SRD
  by (simp add: choose-srd-def closure unrest)

lemma NSRD-choose-srd [closure]: choose\_R is NSRD
  by (rule NSRD-intro, simp-all add: closure unrest rdes)

12.2 State Abstraction

definition state-srea ::
  ′s itself ⇒ (′s,′t::trace,′α,′β) rel-rsp ⇒ (unit,′t,′α,′β) rel-rsp where
  [upred-defs]: state-srea t P = (∃ \{s\_st, s\_st′\} · P)\_s

syntax
  -state-srea :: type ⇒ logic ⇒ logic (state - · - [0,200] 200)

translations
  state \_t \_a · P == CONST state-srea TYPE(′a) P

lemma R1-state-srea: R1(state \_t \_a · P) = (state \_t \_a · R1(P))
  by (rel-auto)

lemma R2c-state-srea: R2c(state \_t \_a · P) = (state \_t \_a · R2c(P))
  by (rel-auto)

lemma R3h-state-srea: R3h(state \_t \_a · P) = (state \_t \_a · R3h(P))
  by (rel-auto)

lemma RD1-state-srea: RD1(state \_t \_a · P) = (state \_t \_a · RD1(P))
  by (rel-auto)

lemma RD2-state-srea: RD2(state \_t \_a · P) = (state \_t \_a · RD2(P))
  by (rel-auto)
lemma RD3-state-srea: RD3(state 'a · P) = (state 'a · RD3(P))
  by (rel-auto, blast+)

lemma SRD-state-srea [closure]: P is SRD \implies state 'a · P is SRD
  by (simp add: Healthy-def R1-state-srea R2c-state-srea R3h-state-srea RD1-state-srea RD2-state-srea RHS-def SRD-def)

lemma NSRD-state-srea [closure]: P is NSRD \implies state 'a · P is NSRD
  by (metis Healthy-def NSRD-is-RD3 NSRD-is-SRD RD3-state-srea SRD-RD3-implies-NSRD SRD-state-srea)

lemma preR-state-srea [rdes]: preR(state 'a · P) = (\forall \{\$st, st\} \cdot preR(P))_S
  by (simp add: state-srea-def, rel-auto)

lemma periR-state-srea [rdes]: periR(state 'a · P) = state 'a · periR(P)
  by (rel-auto)

lemma postR-state-srea [rdes]: postR(state 'a · P) = state 'a · postR(P)
  by (rel-auto)

12.3 While Loop

definition WhileR :: 's upred ⇒ ('s, 't::size-trace, 'a) hrel-rsp ⇒ ('s, 't, 'a) hrel-rsp (whileR - do - od)
where
  WhileR b P = (μR X ∙ (P ;; X) ≺ b ⪰R H_R)

lemma Sup-power-false:
  fixes F :: 'a upred ⇒ 'a upred
  shows (\prod i. (F ^^ i) false) = (\prod i. (F ^^ (i+1)) false)
proof –
  have (\prod i. (F ^^ i) false) = (F ^^ 0) false ∩ (\prod i. (F ^^ (i+1)) false)
    by (subst Sup-power-expand, simp)
  also have ... = (\prod i. (F ^^ (i+1)) false)
    by (simp)
  finally show ?thesis .
qed

theorem WhileR-iter-expand:
  assumes P is NSRD P is Productive
  shows whileR b do P od = (\prod i ∙ (P ≺ b ⪰R H_R) ≺ i ;; (P ;; Miracle ≺ b ⪰R H_R)) (is ?lhs = ?rhs)
proof –
  have 1:Continuous (λX. P ;; SRD X)
    using SRD-Continuous
    by (clarsimp simp add: Continuous-def seq-SUP-distl[THEN sym], drule-tac x=A in spec, simp)
  have 2: Continuous (λX. P ;; SRD X ≺ b ⪰R H_R)
    by (simp add: 1 closure assms)
  have ?lhs = (μR X ∙ (P ;; X) ≺ b ⪰R H_R)
    by (simp add: WhileR-def)
  also have ... = (μR X ∙ P ;; SRD(X) ≺ b ⪰R H_R)
    by (auto simp add: srd-mu-equi closure assms)
  also have ... = (μR X ∙ P ;; SRD(X) ≺ b ⪰R H_R)
    by (auto simp add: guarded-fp-uniq Guarded-if-Productive[of assms] funcsetI closure assms)
  also have ... = (μR X ∙ P ;; SRD(X) ≺ b ⪰R H_R)
    by (auto simp add: guarded-fp-uniq Guarded-if-Productive[of assms] funcsetI closure assms)
  also have ... = (\prod i. (μX ∙ P ;; SRD X ≺ b ⪰R H_R) ^^ i) false)
    by (simp add: sup-continuous-lfp 2 sup-continuous-Continuous false-upred-def)
  also have ... = (\prod i. (μX ∙ P ;; SRD X ≺ b ⪰R H_R) ^^ (i+1)) false)
    by (simp add: sup-power-false)
  also have ... = (\prod i. (P ≺ b ⪰R H_R) + i ;; (P ;; Miracle ≺ b ⪰R H_R))
  qed
proof (rule SUP-cong, simp)
  fix i
  show \((\lambda X. P :: SRD X \triangleleft b \triangleright R I_R) \triangleright^\bowtie (i + 1)\) \false = \((P \triangleleft a \triangleright R I_R) \triangleright^\bowtie i :: (P :: Miracle \triangleleft a \triangleright R I_R)\)
  proof (induct i)
    case 0
    then show \(?thesis\)
  proof
    case 0
    then show \(?thesis\)
  qed
  qed
next
  case (Suc i)
  show \(?thesis\)
  proof
    case 0
    then show \(?thesis\)
  qed
qed

theorem \textit{WhileR-star-expand}:
  assumes \(P \text{ is NSRD \ P is Productive} \)
  shows \(\text{while}_{R \ b \ \text{do \ P}} \ od = (P \triangleleft a \triangleright R I_R)^{\star \ R} :: (P :: \text{Miracle} \triangleleft a \triangleright R I_R)\) (is \(?lhs = ?rhs\))
proof 
  have \(?lhs = (\prod i \cdot (P \triangleleft a \triangleright R I_R) \triangleright^\bowtie i :: (P :: \text{Miracle} \triangleleft a \triangleright R I_R))\)
    by (simp add: \textit{WhileR-iter-expand seq-UINF-distr'} \assms)
  qed
also have \( \ldots = (P \circ b \triangleright_R II_R)^* \circ (P \circ \text{ Miracle} \circ b \triangleright_R II_R) \)
by (simp add: ustar-def)
also have \( \ldots = (((P \circ b \triangleright_R II_R)^* \circ II_R) \circ (P \circ \text{ Miracle} \circ b \triangleright_R II_R) \)
by (simp add: seqr-assoc SRD-left-unit closure assms)
also have \( \ldots = (P \circ b \triangleright_R II_R)^* \circ R \circ (P \circ \text{ Miracle} \circ b \triangleright_R II_R) \)
by (simp add: nsrd-thy.Star-def)
finally show \(?thesis\).
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 \( (P \circ b \triangleright_R II_R)^* \circ R \circ (P \circ \text{ Miracle} \circ b \triangleright_R II_R) = ([b]^T_R \circ P)^* \circ R \circ ([\neg b]^T_R)^* \circ R \circ (P \circ \text{ Miracle} \circ b \triangleright_R II_R) \)
proof –
have \( (P \circ b \triangleright_R II_R)^* \circ R \circ (P \circ \text{ Miracle} \circ b \triangleright_R II_R) = ([b]^T_R \circ P \cap [\neg b]^T_R)^* \circ R \circ (P \circ \text{ Miracle} \circ b \triangleright_R II_R) \)
by (simp add: AssumeR-NSRD NSRD-right-unit NSRD-srd-skip assms(1) cond-srea- AssumeR-form)
also have \( \ldots = (((P \circ b \triangleright_R II_R)^* \circ R \circ (P \circ \text{ Miracle} \circ b \triangleright_R II_R) \)
by (simp add: AssumeR-NSRD NSRD-seqr-closure nsrd-thy.Star-denest assms(1))
also have \( \ldots = (((P \circ b \triangleright_R II_R)^* \circ R \circ (P \circ \text{ Miracle} \circ b \triangleright_R II_R) \)
by (simp add: AssumeR-NSRD NSRD-Miracle NSRD-right-unit NSRD-seqr-closure NSRD-srd-skip assms(1) cond-srea- AssumeR-form)
also have \( \ldots = (((P \circ b \triangleright_R II_R)^* \circ R \circ (P \circ \text{ Miracle} \circ b \triangleright_R II_R) \)
by (simp add: upred-semiring.distrib-left)
also have \( \ldots = ([b]^T_R \circ P)^* \circ R \circ (P \circ \text{ Miracle} \circ b \triangleright_R II_R) \)
by (simp add: upred-semiring.distrib-left)
proof –
have \( (([b]^T_R \circ P)^* \circ R \circ (P \circ \text{ Miracle} \circ b \triangleright_R II_R) \)
by (simp add: AssumeR-NSRD NSRD-seqr-closure nsrd-thy.Star-unfoldr-eq assms(1))
also have \( \ldots = ([\neg b]^T_R \circ (P \circ \text{ Miracle} \circ b \triangleright_R II_R) \)
by (simp add: AssumeR-true NSRD-right-unit assms(1))
also have \( \ldots = ([\neg b]^T_R \circ (P \circ \text{ Miracle} \circ b \triangleright_R II_R) \)
by (simp add: AssumeR-choice upred-semiring.add-assoc upred-semiring.distrib-left upred-semiring.distrib-right)
also have \( \ldots = ([\neg b]^T_R \circ (P \circ \text{ Miracle} \circ b \triangleright_R II_R) \)
by (simp add: RA1)
also have \( \ldots = ([\neg b]^T_R \circ (P \circ \text{ Miracle} \circ b \triangleright_R II_R) \)
by (simp add: AssumeR-comp AssumeR-false)
finally have \( ([b]^T_R \circ P)^* \circ R \circ (P \circ \text{ Miracle} \circ b \triangleright_R II_R) \)
by (simp add: semilattice-sup-class.le-sup11)
thus \(?thesis\).
by (simp add: semilattice-sup-class.le-iff-sup)
qed  
finally show \( \text{thesis} \).
qed

**theorem** WhileR-iter-form:  
assumes \( P \) is NSRD \( P \) is Productive  
shows while \( b \) do \( P \) od = \( ([b]^{\top}_R :: P)^* R :: [\neg b]^{\top}_R \)  
by (simp add: WhileR-iter-form-lemma WhileR-star-expand assms)

**theorem** WhileR-false:  
assumes \( P \) is NSRD  
shows while \( b \) false do \( P \) od = \( \Pi_R \)  
by (simp add: WhileR-def rpred closure srdes-theory-continuous LFP-const)

**theorem** WhileR-true:  
assumes \( P \) is NSRD \( P \) is Productive  
shows while \( b \) true do \( P \) od = \( P^{* R} :: \text{Miracle} \)  
by (simp add: WhileR-iter-form AssumeR-true AssumeR-false SRD-left-unit assms closure)

**lemma** WhileR-insert-assume:  
assumes \( P \) is NSRD \( P \) is Productive  
shows while \( b \) do \( ([b]^{\top}_R :: P) \) od = while \( b \) do \( P \) od  
by (simp add: AssumeR-NSRD AssumeR-comp NSRD-seqr-closure Productive-seq-2 RA1 WhileR-iter-form assms)

**theorem** WhileR-refine-intro:  
assumes \( P \) is RC \( Q \) is RR \( R \) is RR \( \$st' \not\in Q \) \( R \) is \( R_4 \)  
shows while \( b \) do \( R_s (P \vdash Q \circ R) \) od = \( R_s (([b]^{\top}_r :: R)^* r wp_r (|b|_{\prec} \Rightarrow_r P) \vdash ([b]^{\top}_r :: R)^* r :: [\neg b]^{\top}_r) \)  
(is \( \text{lhs} = \text{rhs} \))  
proof  
  have \( \text{lhs} = ([b]^{\top}_r :: R_s (P \vdash Q \circ R))^{* R} :: [\neg b]^{\top}_r \)  
  by (simp add: WhileR-iter-form Productive-rdes-RR-intro assms closure)  
also have \( \ldots = \text{rhs} \)  
by (simp add: rdes-def assms closure unrest rpred wp del: rea-star-wp)  
finally show \( \text{thesis} \).
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 \( R_4 \)  
  — Refinement conditions  
  \( ([b]^{\top}_r :: Q_3)^* r wp_r ([b]_{\prec} \Rightarrow_r Q_1) \subseteq P_1 \)  
  \( P_2 \sqsubseteq [b]^{\top}_r :: Q_2 \)  
  \( P_2 \sqsubseteq [b]^{\top}_r :: Q_3 :: P_2 \)  
  \( P_3 \sqsubseteq [\neg b]^{\top}_r \)  
  \( P_3 \sqsubseteq [b]^{\top}_r :: Q_3 :: P_3 \)  
shows \( R_s (P_1 \circ P_2 \circ P_3) \sqsubseteq \text{while}_R b \) do \( R_s (Q_1 \vdash Q_2 \circ Q_3) \) od  
proof (simp add: rdes-def assms, rule srdes-tri-refine-intro)  
show \( ([b]^{\top}_r :: Q_3)^* r wp_r ([b]_{\prec} \Rightarrow_r Q_1) \subseteq P_1 \)  
by (simp add: assms)  
show \( P_2 \sqsubseteq (P_1 \land ([b]^{\top}_r :: Q_3)^* r :: [b]^{\top}_r :: Q_2) \)  
proof

100
12.4 Iteration Construction

**definition IterateR**

:: 'a set ⇒ ('a ⇒ 's upred) ⇒ ('a ⇒ ('s, 't::size-trace, 'α) hrel-rsp) ⇒ ('s, 't, 'α) hrel-rsp

**where** IterateR A g P = while_R (∧ i∈A · g(i)) do (if_R i∈A · g(i) ⇒ P(i) fi) od

**syntax**

- `iter-srd`: `pattrn ⇒ logic ⇒ logic ⇒ logic ⇒ logic (do_R ∈- · → - fi)`

**translations**

- `iter-srd x A g P =⇒ CONST IterateR A (λ x. g) (λ x. P)
- `iter-srd x A g P =⇐ CONST IterateR A (λ x. g) (λ x'. P)

**lemma IterateR-NSRD-closed [closure]:**

assumes

\[ \bigwedge i. i \in I \implies P(i) \text{ is NSRD} \]

\[ \bigwedge i. i \in I \implies P(i) \text{ is Productive} \]

shows `do_R i∈I · g(i) ⇒ P(i) fi = II_R`

by (simp add: IterateR-def closure assms)

**lemma IterateR-empty:**

`do_R i∈{} · g(i) ⇒ 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 `do_R i∈{k} · g(i) ⇒ P(i) fi = while_R g(k) do P(k) od (is ?lhs = ?rhs)`

**proof**

have `?lhs = while_R g k do P k ⇒ g k ▷ R Chaos od`

by (simp add: IterateR-def AlternateR-singleton assms closure)

also have `... = while_R g k do [g k]^T_R ;; (P k ⇒ g k ▷ R Chaos) od`

by (simp add: WhileR-insert-assume closure assms)

also have `... = while_R g k do P k od`

by (simp add: AssumeR-cond-left NSRD-Chaos WhileR-insert-assume assms)

finally show `?thesis`.

qed

12.5 Substitution Laws

**lemma srd-subst-Chaos [asubst]:**

\[ σ ↑_S Chaos = Chaos \]

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]:
\[ \sigma \uparrow_S \langle \varrho \rangle_R = \langle \varrho \circ \sigma \rangle_R \]
by (rdes-eq)

12.6 Algebraic Laws

theorem assigns-srd-id: \( \langle \text{id} \rangle_R = \text{II}_R \)
by (rdes-eq)

theorem assigns-srd-comp: \( \langle \sigma \rangle_R ;; \langle \varrho \rangle_R = \langle \varrho \circ \sigma \rangle_R \)
by (rdes-eq)

theorem assigns-srd-Miracle: \( \langle \sigma \rangle_R ;; \text{Miracle} = \text{Miracle} \)
by (rdes-eq)

theorem assigns-srd-Chaos: \( \langle \sigma \rangle_R ;; \text{Chaos} = \text{Chaos} \)
by (rdes-eq)

theorem assigns-srd-cond: \( \langle \sigma \rangle_R \triangleleft b \triangleright_R \langle \varrho \rangle_R = \langle \sigma \triangleleft b \triangleright_R \varrho \rangle_R \)
by (rdes-eq)

theorem assigns-srd-left-seq:
assumes P is NSRD
shows \( \langle \sigma \rangle_R ;; P = \sigma \uparrow_S P \)
by (rdes-simp cls: assms)

lemma AlternateR-seq-distr:
assumes \( \bigwedge_i. \text{A} \text{\ is NSRD} \ B \text{\ is NSRD} \ C \text{\ is NSRD} \)
shows \( (\text{if } R i \in I \cdot g \rightarrow A \text{ else } B \text{ \fi }) ;; C = (\text{if } R i \in I \cdot g \rightarrow A \text{ else } C \text{ else } B ;; C \text{ \fi}) \)
proof (cases I = \{\})
  case True
    then show \?thesis by (simp)
  next
  case False
    then show \?thesis
  
    by (simp add: AlternateR-def upred-semiring,distrib-right seq-UINF-distr gcmd-seq-distr assms(3))
qed

lemma AlternateR-is-cond-srea:
assumes A is NSRD B is NSRD
shows \( (\text{if } R i \in \{a\} \cdot g \rightarrow A \text{ else } B \text{ \fi}) = (A \triangleleft g \triangleright_R B) \)
by (rdes-eq cls: assms)

lemma AlternateR-Chaos:
if \( R i \in A \cdot g(i) \rightarrow \text{Chaos} \text{ \fi} = \text{Chaos} \)
by (cases A = \{\}, simp, rdes-eq)
12.7 Lifting designs to reactive designs

**definition** des-rea-lift :: 's hrel-des ⇒ ('s',t::trace,'α) hrel-rsp (R_D) where
[upred-defs]: R_D(P) = R_u([pre_D(P)]S → (false ∘ ($str' =_u $tr ∧ [post_D(P)]S)))

**definition** des-rea-drop :: ('s',t::trace,'α) hrel-rsp ⇒ 's hrel-des (D_R) where

**lemma** ndesign-rea-lift-inverse: D_R(R_D(p ⊨n Q)) = p ⊨n Q
apply (simp add: des-rea-lift-def des-rea-drop-def rea-pre-RHS-design rea-post-RHS-design)
apply (simp add: R1-def R2c-def R2s-def usubst unrest)
apply (rel-auto)
done

**lemma** ndesign-rea-lift-injective:
assumes P is N Q is N R_D P = R_D Q (is ?RP(P) = ?RQ(Q))
shows P = Q
proof –
have ?RP([pre_D(P)]< ⊨n post_D(P)) = ?RQ([pre_D(Q)]< ⊨n post_D(Q))
  by (simp add: ndesign-form assms)
hence [pre_D(P)]< ⊨n post_D(P) = [pre_D(Q)]< ⊨n post_D(Q)
  by (metis ndesign-rea-lift-inverse)
thus ??thesis
  by (simp add: ndesign-form assms)
qed

**lemma** des-rea-lift-closure [closure]: R_D(P) is SRD
by (simp add: des-rea-lift-def RHS-design-is-SRD unrest)

**lemma** preR-des-rea-lift [rdes]:
pre_R(R_D(P)) = R1([pre_D(P)]S)
by (rel-auto)

**lemma** periR-des-rea-lift [rdes]:
peri_R(R_D(P)) = (false ∘ [pre_D(P)]S ∨ ($str ≤_u $tr'))
by (rel-auto)

**lemma** postR-des-rea-lift [rdes]:
post_R(R_D(P)) = (true ∘ [pre_D(P)]S ∨ (~ $str ≤_u $tr') ⇒ ($str' =_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)
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_n(false ⊢ false o ($tr = u$ tr))
      by (simp add: Chaos-def des-rea-lift-def alpha)
    also have ... = Chaos
      by (simp add: Chaos-def設計false-pre)
  finally show ?thesis .
  qed

lemma R-D-inf:
  ℛ_D(P ⊓ Q) = ℛ_D(P) ⊓ ℛ_D(Q)
  by (rule antisym, rel-auto+)

lemma R-D-cond:
  ℛ_D(P ⊳ ⌈b⌉_D< ⊲ Q) = ℛ_D(P) ⊳ b ⊲ ℛ_D(Q)
  by (rule antisym, rel-auto+)

lemma R-D-seq-ndesign:
  ℛ_D(p_1 ⊳_n Q_1) ;; ℛ_D(p_2 ⊳_n Q_2) = ℛ_D((p_1 ⊳_n Q_1) ;; (p_2 ⊳_n Q_2))
  apply (rule antisym)
  apply (rule SRD-refine-intro)
   apply (simp-all add: closure rdes ndesign-composition-wp)
using dual-order_trans apply (rel-blast)
using dual-order_trans apply (rel-blast)
apply (rel-auto)
apply (rule SRD-refine-intro)
   apply (simp-all add: closure rdes ndesign-composition-wp)
apply (rel-auto)
apply (rel-auto)
apply (rel-auto)
done

104
Thes laws are applicable only when there is no further alphabet extension

lemma R-D-skip:
  assumes P is N Q is N
  shows R_D(P ;: R_D(Q) = R_D(P ;: Q)
  by (metis R-D-seq-nodesign assms ndesign-form)

13 Instantaneous Reactive Designs

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

definition ISRD1 :: ('s,'t::trace,'a) hrel-rsp ⇒ ('s,'t,'a) hrel-rsp where
  [upred-defs]: ISRD1(P) = P || R_a(true_r ⊢ false ⊔ ($str = u $tr))

definition ISRD :: ('s,'t::trace,'a) hrel-rsp ⇒ ('s,'t,'a) 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
  shows ISRD1(R_a(P ⊔ Q ⊓ R)) = R_a(P ⊢ false ◦ (R ∧ $str = u $tr))
  using assms by (simp add: ISRD1-def choose-srd-def RHS-tri-design-par unrest, rel-auto)

lemma ISRD1-form:
  ISRD1(SRD(P)) = R_a(pre_R(P) ⊢ false ◦ (post_R(P) ∧ $str = u $tr))
  by (simp add: ISRD1-RHS-design-form SRD-as-reactive-tri-design unrest)

lemma ISRD1-rdes-def [rdes-def]:
  □ P is RR; R is RR □ ⇒ ISRD1(R_a(P ⊔ Q ⊓ R)) = R_a(P ⊢ false ◦ (R ∧ $str = u $tr))
  by (simp add: ISRD1-def rdes-def closure rpred)

lemma ISRD-intro:
  assumes P is NSRD peri_R(P) = (¬v pre_R(P)) ($str = u $tr) ⊆ post_R(P)
  shows P is ISRD
  proof –
    have R_a(pre_R(P) ⊢ peri_R(P) ◦ post_R(P)) is ISRD1
      apply (simp add: Healthy-def rdes-def closure assms(1-2))
      using assms(3) least-zero apply (rel-blast)
  end
done

hence $P$ is ISRD1
  by (simp add: SRD-reactive-tri-design closure assms)

thus $\text{?thesis}$
  by (simp add: ISRD-def Healthy-comp assms)

qed

lemma ISRD1-rdes-intro:
  assumes $P$ is RR $Q$ is RR ($\text{tr} = u \text{tr}$) ⊑ $Q$
  shows $R_s(P \vdash \text{false} \circ Q)$ is ISRD1

unfolding Healthy-def
  by (simp add: ISRD1-rdes-def assms closure unrest utp-pred-laws.inf.absorb1)

lemma ISRD-rdes-intro [closure]:
  assumes $P$ is RC $Q$ is RR ($\text{tr} = u \text{tr}$) ⊑ $Q$
  shows $R_s(P \vdash \text{false} \circ Q)$ is ISRD

unfolding Healthy-def
  by (simp add: ISRD-def closure Healthy-if ISRD1-rdes-def assms unrest utp-pred-pred laws.inf.absorb1)

lemma ISRD-implies-ISRD1:
  assumes $P$ is ISRD
  shows $P$ is ISRD1

proof
  have ISRD($P$) is ISRD1
    by (simp add: ISRD-def Healthy-def ISRD1-idem)

thus $\text{?thesis}$
  by (simp add: assms Healthy-if)

qed

lemma ISRD-implies-SRD:
  assumes $P$ is ISRD
  shows $P$ is SRD

proof
  have 1:ISRD($P$) = $R_s((\neg_r (\neg_r \text{pre}_R P) :: R1 true \land R1 true) \vdash \text{false} \circ (\text{post}_R P \land \text{tr} = u \text{tr}))$
    by (simp add: NSRD-form ISRD1-def ISRD-def RHS-tri-design-par rdes-def unrest closure)

moreover have ... is SRD
  by (simp add: closure unrest)

ultimately have ISRD($P$) is SRD
  by (simp)

with assms show $\text{?thesis}$
  by (simp: Healthy-def)

qed

lemma ISRD-implies-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 Healthy-if)

also have ... = ISRD1($R_s((\neg_r \text{pre}_R P) \text{wp}_R false_h \vdash (\exists \text{st} \cdot \text{peri}_R P) \circ \text{post}_R P)$)
  by (simp add: RD3-def, subst SRD-right-unit-tri-lemma, simp-all add: assms ISRD-implies-SRD)

also have ... = $R_s((\neg_r \text{pre}_R P) \text{wp}_R false_h \vdash \text{false} \circ (\text{post}_R P \land \text{tr} = u \text{tr}))$
  by (simp add: RHS-tri-design-par ISRD1-def unrest choose-srd-def rpred closure ISRD-implies-SRD)

106
also have \( \ldots = (\ldots ; R) \)
by (rdes-simp, simp add: RHS-tri-normal-design-composition' closure assms unrest ISRD-implies-SRD wp rpred wp-rea-false-RC)
also have \( \ldots \text{ is RD3} \)
by (simp add: Healthy-def RD3-def seqr-assoc)
finally show \( \mathbf{?thesis} \)
by (simp add: SRD-RD3-implies-NSRD Healthy-if assms ISRD-implies-SRD)
qed

lemma \( \text{ISRD-form} \):
assumes \( P \text{ is ISRD} \)
shows \( R_s (\text{pre}_R(P) \vdash false \circ (\text{post}_R(P) \land \$tr^{'} = u \$tr)) = P \)
proof --
  have \( P = \text{ISRD1}(P) \)
    by (simp add: ISRD-implies-ISRD1 assms Healthy-if)
  also have \( \ldots = \text{ISRD1}(R_s (\text{pre}_R(P) \vdash \text{peri}_R(P) \circ \text{post}_R(P))) \)
    by (simp add: SRD-reactive-tri-design ISRD-implies-SRD assms)
  also have \( \ldots = R_s (\text{pre}_R(P) \vdash false \circ (\text{post}_R(P) \land \$tr^{'} = u \$tr)) \)
    by (simp add: ISRD1-rdes-def closure assms)
  finally show \( \mathbf{?thesis} \).
qed

lemma \( \text{ISRD-elim} [\text{RD-elim}] \):
\[ \entry \{ P \text{ is ISRD}; Q(R_s (\text{pre}_R(P) \vdash false \circ (\text{post}_R(P) \land \$tr^{'} = u \$tr))) \entry \} \Rightarrow Q(P) \]
by (simp add: ISRD-form)

lemma \( \text{skip-srd-ISRD} [\text{closure}] \): \( H_R \text{ is ISRD} \)
by (rule ISRD-intro, simp-all add: rdes closure)

lemma \( \text{assigns-srd-ISRD} [\text{closure}] \): \( \langle \sigma \rangle_R \text{ is ISRD} \)
by (rule ISRD-intro, simp-all add: rdes closure, rel-auto)

lemma \( \text{seq-ISRD-closed} \):
assumes \( P \text{ is ISRD} \ Q \text{ is ISRD} \)
shows \( P ; ; Q \text{ is ISRD} \)
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 \( \text{ISRD-Miracle-right-zero} \):
assumes \( P \text{ is ISRD} \ pre_R(P) = true_r \)
shows \( P ; ; \text{Miracle} = \text{Miracle} \)
by (rdes-simp cls: assms)

A recursion whose body does not extend the trace results in divergence

lemma \( \text{ISRD-recurse-Chaos} \):
assumes \( P \text{ is ISRD} \ post_R P ; ; true_r = true_r \)
shows \( (\mu_R X ; P ; ; X) = \text{Chaos} \)
proof --
  have \( 1: (\mu_R X ; P ; ; X) = (\mu X ; P ; ; \text{SRD}(X)) \)
by (auto simp add: srdes-theory-continuous.utp-lfp-def closure assms)

have \( (\mu X \cdot P \cdot SRD(X)) \subseteq Chaos \)

proof (rule gfp-upperbound)

have \( P \cdot Chaos \subseteq Chaos \)

apply (rdes-refine-split cls: assms)

using assms(2) apply (rel-auto, metis (no-types, lifting) dual-order.antisym order-refl)

apply (rel-auto)+

done

thus \( P \cdot SRD Chaos \subseteq Chaos \)

by (simp add: Healthy-if srdes-theory-continuous.bottom-closed)

qed

thus ?thesis

by (metis 1 dual-order.antisym srdes-theory-continuous.LFP-closed srdes-theory-continuous.bottom-lower)

qed

lemma recursive-assign-Chaos:

\[(\mu_R X \cdot \langle \sigma \rangle_R \cdot X) = Chaos\]

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

end

14 Meta-theory for Reactive Designs

theory utp-rea-designs

imports

  utp-rdes-heaths
  utp-rdes-designs
  utp-rdes-triples
  utp-rdes-normal
  utp-rdes-contracts
  utp-rdes-tactics
  utp-rdes-parallel
  utp-rdes-prag
  utp-rdes-instant
  utp-rdes-guarded

begin end

References

