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 (Unpublished)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 19, 2018

Abstract

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

Contents

1 Introduction 3

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

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

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

1
1 Introduction

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

2 Reactive Designs Healthiness Conditions

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

2.1 Preliminaries

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

type-synonym (′s,′t) rdes = (′s,t,unit) hrel-rp

translations (type) (′s,′t) rdes <= (type) (′s, t, unit) hrel-rp

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

lemma R2s-st′-eq-st:
  R2s($st′ =_{u} $st) = ($st′ =_{u} $st)
  by (rel-auto)

lemma R2c-st′-eq-st:
  R2c($st′ =_{u} $st) = ($st′ =_{u} $st)
  by (rel-auto)

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

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

lemma R1-R2c-ex-st: R1 (R2c (∃ $st · 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, α) hrel-rp (II_c) where
  skip-rea-def [urel-defs]: II_c = (II ∨ (¬ $ok ∧ $tr ≤_{u} $tr′))

definition skip-srea :: (′s, t::trace, α) hrel-rp (II_R) where
  skip-srea-def [urel-defs]: II_R = ((∃ $st · II_c) ∨ $wait ∨ II_c)

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

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

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

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

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

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

by (simp add: Healthy-def R2c-skip-srea)

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

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

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

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

2.3 RD1: Divergence yields arbitrary traces

definition RD1 :: ('t::trace,'a,'b) rel-rp ⇒ ('t,'a,'b) rel-rp where
[upred-defs]: \( RD1(P) = (P \lor (\neg \$ok \land \$tr \leq u \$tr')) \)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

theorem RD1-left-zero:
assumes P is R1 P is RD1
shows \((R_1(true) :: P) = R_1(true)\)
proof –
have \((R_1(true) :: R_1(RD_1(P))) = R_1(true)\)
  by (rel-auto)
thus \(?thesis\)
  by (simp add: Healthy-if assms(1) assms(2))
qed

theorem RD_1-left-unit:
assumes \(P \vdash R_1\)
shows \((II_c :: P) = P\)
proof –
have \((II_c :: R_1(RD_1(P))) = R_1(RD_1(P))\)
  by (rel-auto)
thus \(?thesis\)
  by (simp add: Healthy-if assms(1) assms(2))
qed

lemma RD_1-alt-def:
assumes \(P \vdash R_1\)
shows \(RD_1(P) = (P \sqsubseteq \$\text{wait} \sqsubseteq P)\)
proof
have \(RD_1(R_1(P)) = (R_1(P) \sqsubseteq \$\text{wait} \sqsubseteq R_1(true))\)
  by (rel-auto)
thus \(?thesis\)
  by (simp add: Healthy-if assms)
qed

theorem RD_1-algebraic:
assumes \(P \vdash R_1\)
shows \(P \vdash RD_1 \iff (R_1(true_h) :: P) = R_1(true) \land (II_c :: P) = P\)
using RD_1-algebraic-intro RD_1-left-unit RD_1-left-zero assms
by blast

2.4 R3c and R3h: Reactive design versions of R3

definition R3c :: ('t::trace, 'a) hrel-\(r\)p \(\Rightarrow\) ('t, 'a) hrel-\(r\)p where
[upred-defs]: \(R_3c(P) = (II_c < \$\text{wait} \triangleright P)\)

definition R3h :: ('s, 't::trace, 'a) hrel-\(r\)sp \(\Rightarrow\) ('s, 't, 'a) hrel-\(r\)sp where
R3h-def [upred-defs]: \(R_3h(P) = ((\exists s t :: II_c < \$\text{wait} \triangleright P)\)

lemma R3c-idem: \(R_3c(R_3c(P)) = R_3c(P)\)
by (rel-auto)

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

lemma R3c-mono: \(P \subseteq Q \Rightarrow R_3c(P) \subseteq R_3c(Q)\)
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: $P \subseteq Q \implies R3h(P) \subseteq R3h(Q)$
by (rel-auto)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma $R3h$-seq-closure:
assumes $P$ is $R3c$ $Q$ is $R3c$ $Q$ is $R1$
shows ($P$ ; $Q$) is $R3c$
by (metis Healthy-def' $R3c$-semir-form assms)

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

lemma $R3c$-$R3$-left-seq-closure:
assumes $P$ is $R3$ $Q$ is $R3c$
shows ($P$ ; $Q$) is $R3c$
proof –
  have ($P$ ; $Q$) = (($P$ ; $Q$)[true/$\$wait] < $\$wait $\triangleright$ ($P$ ; $Q$))
  by (metis cond-var-split cond-var-subst-right in-var-avar wait-vwb-lens)
  also have ... = (((II < $\$wait $\triangleright$ $P$) ; $Q$)[true/$\$wait] < $\$wait $\triangleright$ ($P$ ; $Q$))
  by (metis Healthy-def' $R3c$-def assms(1))
  also have ... = (((II[true/$\$wait$]$ ; $Q$) < $\$wait $\triangleright$ ($P$ ; $Q$))
  by (subst-tac)
  also have ... = (((II ∧ $\$wait$`) ; $Q$) < $\$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 ... = (((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-ouvar vwb-lens-mwb wait-vwb-lens)
  also have ... = (((II[true/$\$wait$`]$ ; IIc < $\$wait $\triangleright$ $Q$)[true/$\$wait$]$) < $\$wait $\triangleright$ ($P$ ; $Q$))
  by (metis Healthy-def' $R3c$-def assms(2))
  also have ... = (((II[true/$\$wait$`]$ ; $Qc[true/$\$wait$]$) < $\$wait $\triangleright$ ($P$ ; $Q$))
  by (subst-tac)
  also have ... = (((II ∧ $\$wait$`) ; $IIc$) < $\$wait $\triangleright$ ($P$ ; $Q$))
  by (metis seqr-pre-transfer seqr-right-one-point true-alt-def uovar-cover upred-eq-true utp-rel.unrest-ouvar vwb-lens-mwb wait-vwb-lens)
  also have ... = (((II ; $IIc$) < $\$wait $\triangleright$ ($P$ ; $Q$))
  by (simp add: cond-def seqr-pre-transfer utp-rel.unrest-ouvar)
  also have ... = ($IIc$ < $\$wait $\triangleright$ ($P$ ; $Q$))
  by simp
  also have ... = $R3c$($P$ ; $Q$)
  by (simp add: $R3c$-def)
finally show ?thesis
by (simp add: Healthy-def')

qed

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

lemma R3h-cases: R3h(P) = (((∃ $st ⋵ II) ⋵ $ok ⋵ R1(true)) ⋵ $wait ⋵ P)
  by (rel-auto)

lemma R3h-form: R3h(P) = II_R ⋵ $wait ⋵ P
  by (rel-auto)

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

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

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

lemma R3h-wait-true:
  assumes P is R3h
  shows P t = II_R t
proof -
  have P t = (II_R ⋵ $wait ⋵ P) t
    by (metis Healthy-if R3h-form assms)
  also have ... = II_R t
    by (simp add: usubst)
  finally show ?thesis .
qed

2.5 RD2: A reactive specification cannot require non-termination

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

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

lemma RD2-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 ⊆ Q ⊃ RD2(P) ⊆ 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)
2.6 Major healthiness conditions

definition RH :: (′t::trace,′α) hrel-rp ⇒ (′t,′α) 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(R(P))) = R(P)
by (simp add: R1-R2c-is-R2 R2c-R3c-commute R3c-idem
R1-R3h-commute R1-idem R2c-R3c-commute R2c-idem
RH-Idempotent)

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 R_s
lemma RHS-inf: $R_s(P \cap Q) = R_s(P) \cap R_s(Q)$
using Continuous-Disjunctuous Disjunctuous-def RHS-Continuous by auto

lemma RHS-INF: 
$\forall A \neq \emptyset \Rightarrow R_s(\bigcap_i i \in A \cdot P(i)) = (\bigcap_i i \in A \cdot R_s(P(i)))$
by (simp add: RHS-def R3h-UINF R2c-USUP R1-USUP)

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

lemma RHS-SUP: 
$\forall A \neq \emptyset \Rightarrow R_s(\bigsqcup_i i \in A \cdot P(i)) = (\bigsqcup_i i \in A \cdot R_s(P(i)))$
by (rel-auto)

lemma RHS-cond: $R_s(P \triangleright b \triangleright Q) = (R_s(P) \triangleright R2c b \triangleright R_s(Q))$
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-R1-commute 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-R3h-commute RHS-def)

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

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

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

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

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

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

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

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

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

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

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

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

lemma SRD-RHS-H1-H2: \( SRD(P) = R_s(H(P)) \)
  by (rel-auto)

lemma SRD-healths [closure]:
  assumes \( P \) is SRD
  shows \( P \) is R1 \( P \) is R2 \( P \) is R3h \( P \) is RD1 \( P \) is RD2
  apply (metis Healthy-def R1-R2c-is-R2 RHS-def SRD-def assms)
  apply (metis Healthy-def R1-R2c-is-R2 RHS-def SRD-def assms)
  apply (metis Healthy-def R1-R2c-is-R2 RHS-def SRD-def assms)
  apply (metis Healthy-def R1-R2c-is-R2 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-false [usubst]: \( P \) is SRD \( \Rightarrow \) \( R_1(\text{true}) \)
  by (metis (no-types, hide-lams) H1-H2-eq-design Healthy-def R1-ok-false RD1-R1-commute RD1-via-R1 RD2-def SRD-healths)

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

lemma SRD-left-zero-1: \( P \) is SRD \( \Rightarrow \) \( R_1(\text{true}) \); \( P = R_1(\text{true}) \)
  by (simp add: RD1-left-zero SRD-healths assms)

lemma SRD-left-zero-2:
  assumes \( P \) is SRD
  shows \( (\exists \, \text{st} \cdot \text{II})[[true, true/$\text{ok}$, $\text{wait}$]/] ; P = (\exists \, \text{st} \cdot \text{II})[[true, true/$\text{ok}$, $\text{wait}$]/
  proof
    have \( (\exists \, \text{st} \cdot \text{II})[[true, true/$\text{ok}$, $\text{wait}$]/ ; R_3h(P) = (\exists \, \text{st} \cdot \text{II})[[true, true/$\text{ok}$, $\text{wait}$]/
      by (rel-auto)
    thus \( \ell \)thesis
    by (simp add: Healthy-if SRD-healths assms)

12
2.7 UTP theories

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

typedecl RDES

abbreviation RDES ≡ UTHY(RDES, (t::trace, α) rp)

abbreviation SRDES ≡ UTHY(SRDES, (s,t::trace, α) rsp)

overloading
rdes-hcond == upr-hcond :: (RDES, (t::trace, α) rp) uph ⇒ ((t,α) rp × (t,α) rp) health
srdes-hcond == upr-hcond :: (SRDES, (s,t::trace, α) rsp) uph ⇒ ((s,t,α) rsp × (s,t,α) rsp) health

definition rdes-hcond :: (RDES, (t::trace, α) rp) uph ⇒ ((t,α) rp × (t,α) rp) health where
begin
  definition rdes-hcond :: (RDES, (t::trace, α) rp) uph ⇒ ((t,α) rp × (t,α) rp) health where
  [upred-defs]: rdes-hcond T = RD
end

interpretation rdes-theory: upr-theory UTHY(RDES, (t::trace, α) rp)
by (unfold-locales, simp-all add: rdes-hcond-def RD-idem)

interpretation rdes-theory-continuous: upr-theory-continuous UTHY(RDES, (t::trace, α) rp)
rewrites ∧. P. P ∈ carrier (uthy-order RDES) ⟷ P is RD
and carrier (uthy-order RDES) → carrier (uthy-order RDES) ≡ [RD]_H → [RD]_H
and le (uthy-order RDES) = op ⊆
and eq (uthy-order RDES) = op =
by (unfold-locales, simp-all add: rdes-hcond-def RD-Continuous)

interpretation rdes-rea-galois:
galois-connection (RDES ← (RD1 ∪ RD2,R3) → REA)

proof (simp add: mk-conn-def, rule galois-connection′, simp-all add: upr-partial-order rdes-hcond-def rea-hcond-def)

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

  show RD1 ∪ RD2 ∈ [RP]_H → [RD]_H
  by (simp add: Pi-iff Healthy-def′, metis RD-def RD-idem)

  show isotone (upr-order RD) (upr-order RP) R3
  by (simp add: R3-Monotonic isotone-upr-order1)

  show isotone (upr-order RP) (upr-order RD) (RD1 ∪ RD2)
  by (simp add: Monotonic-comp RD1-Monotonic RD2-Monotonic isotone-upr-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-iff)

next

  fix P :: ('a, 'b) hrel-rp
  assume a: P is RP
  thus R3 (RD1 (RD2 P)) ⊆ P
  proof
    have R3 (RD1 (RD2 P)) = RP (RD1 (RD2(P)))

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

interpretation rdes-rea-retract:
retract (RDES ← (RD1 ⪯ RD2, R3) → REA)
by (unfold-locales, simp-all add: mk-conn-def utp-partial-order rdes-hcond-def rea-hcond-def)
(metis Healthy-if R3-RD-RP RD-def RP-idem eq-refl)

interpretation srdes-theory: utp-theory UTHY (SRDES, (s, t::trace, α) rsp)
by (unfold-locales, simp-all add: srdes-hcond-def SRD-Continuous)

interpretation srdes-theory-continuous: utp-theory-continuous UTHY (SRDES, (s, t::trace, α) rsp)
rewrites \( \bigwedge P. P \in \text{carrier} (\text{uthy-order SRDES}) \iff P \text{ is SRD} \)
and \( P \text{ is } H_{SRDES} \iff P \text{ is SRD} \)
and \( (\mu X \cdot F (H_{SRDES} X)) = (\mu X \cdot F (SRD X)) \)
and carrier (uthy-order SRDES) \( \cong \) carrier (uthy-order SRDES) \( \equiv [SRD]_H \rightarrow [SRD]_H \)
and \( H_{SRDES} \rightarrow H_{SRDES} \equiv [SRD]_H \rightarrow [SRD]_H \)
and le (uthy-order SRDES) \( = \) op \( \subseteq \)
and eq (uthy-order SRDES) \( = \) op \( = \)
by (unfold-locales, simp-all add: srdes-hcond-def SRD-Continuous)

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

abbreviation Chaos :: (s, t::trace, α) hrel-rsp where
Chaos \equiv SV_{SRDES}

abbreviation Miracle :: (s, t::trace, α) hrel-rsp where
Miracle \equiv TV_{SRDES}

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

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

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

syntax
- srd-mu :: pttrn \Rightarrow \text{logic} \Rightarrow \text{logic} (\mu_R \cdot \cdot \cdot [0, 10] 10)
- srd-mu :: pttrn \Rightarrow \text{logic} \Rightarrow \text{logic} (\nu_R \cdot \cdot \cdot [0, 10] 10)

translations
\( \mu_R X \cdot P \equiv \mu_R (\lambda X. P) \)
\( \nu_R X \cdot P \equiv \mu_R (\lambda X. P) \)
The reactive design weakest fixed-point can be defined in terms of relational calculus one.

lemma srd-mu-equiv:
assumes Monotonic F F \in [SRD]_H \rightarrow [SRD]_H
shows \((\mu_R X \cdot F(X)) = (\mu X \cdot F(SRD(X)))\)
by \((\text{metis assms srdes-hcond-def srdes-theory-continuous.utp-lfp-def})\)

end

3 Reactive Design Specifications

definitions

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

3.1 Reactive design forms

lemma srdes-skip-def: \(\Pi_R = \mathsf{RS}_s(\text{true} \triangleright (\mathsf{str} =_u \mathsf{str} \land \neg \mathsf{wait} \land [\Pi]_R))\)
apply (rel-auto) using minus-zero-eq by blast

lemma Chaos-def: \(\text{Chaos} = \mathsf{RS}_s(\text{false} \triangleright \text{true})\)
proof
  have \(\text{Chaos} = \mathsf{SRD}(\text{true})\)
  by (metis srdes-hcond-def srdes-theory-continuous.healthy-bottom)
  also have \(\ldots = \mathsf{R}_s(\mathsf{H}(\text{true}))\)
  by (simp add: SRD-RHS-H1-H2)
  also have \(\ldots = \mathsf{R}_s(\text{false} \triangleright \text{true})\)
  by (metis H1-design H2-true design-false-pre)
finally show ?thesis .
qed

lemma Miracle-def: \(\text{Miracle} = \mathsf{RS}_s(\text{true} \triangleright \text{false})\)
proof
  have \(\text{Miracle} = \mathsf{SRD}(\text{false})\)
  by (metis srdes-hcond-def srdes-theory-continuous.healthy-top)
  also have \(\ldots = \mathsf{R}_s(\mathsf{H}(\text{false}))\)
  by (simp add: SRD-RHS-H1-H2)
  also have \(\ldots = \mathsf{R}_s(\text{true} \triangleright \text{false})\)
finally show ?thesis .
qed

lemma RD1-reactive-design: \(\text{RD1}(\mathsf{R}(P \triangleright Q)) = \mathsf{R}(P \triangleright Q)\)
by (rel-auto)

lemma RD2-reactive-design:
  assumes \(\text{ok} \not\triangleright P \not\triangleright Q\)
  shows \(\text{RD2}(\mathsf{R}(P \triangleright Q)) = \mathsf{R}(P \triangleright Q)\)
  using assms
  by (metis H2-design RD2-RH-commute RD2-def)

lemma RD1-st-reactive-design: \(\text{RD1}(\mathsf{RS}_s(P \triangleright Q)) = \mathsf{RS}_s(P \triangleright Q)\)
by (rel-auto)

lemma RD2-st-reactive-design:
  assumes \(\text{ok} \not\triangleright P \not\triangleright Q\)
  shows \(\text{RD2}(\mathsf{RS}_s(P \triangleright Q)) = \mathsf{RS}_s(P \triangleright Q)\)
  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:
  \(RD(P) = R((\neg P f) f \vdash P f)\)
proof –
  have \(RD(P) = RD1(RD2(R1(R2c(R3c(P)))))\)
    by (simp add: RD-alt-def RH-def)
  also have \(... = RD1(H2(R1(R2s(R3c(P)))))\)
    by (simp add: R1-R2s-R2c RD2-def)
  also have \(... = RD1(R1(H2(R2s(R3c(P)))))\)
    by (simp add: R1-R2-commute)
  also have \(... = R1((H1(H2(R2s(R3c(P))))))\)
    by (simp add: R1-idem RD1-via-R1)
  also have \(... = R2((R(R1(H2(R3c(H2(R1(P)))))))\)
    by (metis RD2-R3c-commute RD2-def)
  also have \(... = R2((R1(H1(H2(R3c(H2(R1(P)))))))\)
    by (metis R1-R2-commute R1-idem R2-def)
  also have \(... = R2((R3c(R1(H1(P)))))\)
    by (simp add: R1-R3c-commute RD1-R3c-commute RD1-via-R1)
  also have \(... = RH(R1(P)))\)
    by (metis R1-R2s-R2c R1-R3c-commute R2-R1-form RH-def)
  also have \(... = RH(R(P))\)
    by (simp add: R1-R2-commute R1-R3c-commute R1-idem RD1-via-R1 RH-def)
  also have \(... = RH((\neg P f) f \vdash P f)\)
    by (simp add: H1-R2-eq-design)
  also have \(... = R((\neg P f) f \vdash P f)\)
    by (metis no-types, lifting) R3c-subst-wait RH-def subst-not wait-false-design
finally show \(?thesis\)

qed

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

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

lemma SRD-RH-design-form:
  \(SRD(P) = R_\ast((\neg P f) f \vdash P f)\)
proof

have \( \text{SRD}(P) = R_1(R_2c(R_3h(RD_1(RD_2(R_1(P)))))) \)
  by (metis (no-types, lifting) R_1-H2-commute R_1-R2c-commute R_1-R3h-commute R_1-idem R_2c-H2-commute RD_1-R1-commute RD_1-R2c-commute RD_1-R3h-commute RD_2-R3h-commute RD_2-def RHS-def SRD-def)
also have \( \ldots = R_1(R_2s(R_3h(H(P)))) \)
  by (metis (no-types, lifting) R_1-H2-commute R_1-R2c-commute R_1-R3h-commute R_2c-H2-commute RD_1-R2c-commute RD_1-R3h-commute RD_2-R3h-commute RD_2-def RHS-def SRD-def)
also have \( \ldots = R_1(R_2s(R_3h(H(P)))) \)
  by (simp add: H1-H2-eq-design)
finally show \( \text{thesis} \).
qed

lemma \( \text{SRD-reactive-design} \):
assumes \( P \text{ is SRD} \)
shows \( R_s((\neg P f f) \vdash P t f) = P \)
by (metis SRD-RH-design-form Healthy-def assms)

lemma \( \text{SRD-RH-design} \):
assumes \( \$ok' \uparrow P \$ok' \uparrow Q \)
shows \( \text{SRD}(R_s(P \vdash Q)) = R_s(P \vdash Q) \)
by (simp add: RD_1-st-reactive-design RD_2-st-reactive-design RHS-idem SRD-def assms)

lemma \( \text{RHS-design-is-SRD} \):
assumes \( \$ok' \uparrow P \$ok' \uparrow Q \)
shows \( R_s(P \vdash Q) \text{ is SRD} \)
by (metis (no-types, lifting) Healthy-def' SRD-RH-design assms)

3.2 Auxiliary healthiness conditions

definition [upred-defs]: \( R_3c-pre(P) = (\text{true} \& \$wait \triangleright P) \)

definition [upred-defs]: \( R_3c-post(P) = ([I]D \& \$wait \triangleright P) \)

definition [upred-defs]: \( R_3h-post(P) = ((\exists st \cdot [I]D) \& \$wait \triangleright P) \)

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

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

lemma unrest-ok-R_3c-pre [unrest]: \( \$ok \not\triangleright P \Longrightarrow \$ok \not\triangleright R_3c-pre(P) \)
by (simp add: R_3c-pre-def cond-def unrest)

lemma unrest-ok'-R_3c-pre [unrest]: \( \$ok' \not\triangleright P \Longrightarrow \$ok' \not\triangleright R_3c-pre(P) \)
by (simp add: R_3c-pre-def cond-def unrest)
lemma unrest-ok-R3c-post [unrest]: $\text{ok} \not\in P \implies \text{ok} \not\in R3c$-post($P$)
by (simp add: R3c-post-def cond-def unrest)

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

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

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

3.3 Composition laws

**Theorem R1-design-composition:**
fixes $P$ $Q$ :: '(t::trace,α,β) rel-rp
and $R$ $S$ :: '(t,β,γ) rel-rp
assumes $\text{ok}' \not\in P$ $\text{ok}' \not\in Q$ $\text{ok} \not\in R$ $\text{ok} \not\in S$
shows
$\text{R1}(P \equiv Q) \land \text{R1}(R \equiv S)$
$\text{R1}((\neg (\text{R1}(\neg P) \land \text{R1}(\neg Q)) \land \neg (\text{R1}(\neg Q) \land \text{R1}(\neg R)) \land (\text{R1}(Q) \land \text{R1}(S)))$
proof
have $\text{R1}(P \equiv Q) \land \text{R1}(R \equiv S) = (\exists \text{ok} \cdot (\text{R1}(P \equiv Q)[<\text{ok}>]/\text{ok}') \land (\text{R1}(R \equiv S)[<\text{ok}>]/\text{ok}])$
using seqr-middle ok-ovl-lens by blast
also from assms have ... $= (\exists \text{ok} \cdot \text{R1}((\text{ok} \land P) \Rightarrow (<\text{ok}> \land Q)) \land \text{R1}((<\text{ok}> \land R) \Rightarrow (\text{ok}' \land S)))$
by (simp add: design-def R1-def usubst unrest)
also from assms have ...
$= ((\text{R1}((\text{ok} \land P) \Rightarrow (\text{true} \land Q))) \land \text{R1}((\text{true} \land R) \Rightarrow (\text{ok}' \land S)))$
$\lor (\text{R1}((\text{ok} \land P) \Rightarrow (\text{false} \land Q))) \land \text{R1}((\text{false} \land R) \Rightarrow (\text{ok}' \land S)))$
by (simp add: false-alt-def true-alt-def)
also from assms have ...
$= ((\text{R1}((\text{ok} \land P) \Rightarrow Q)) \land \text{R1}((R \Rightarrow (\text{ok}' \land S)))$
$\lor (\text{R1}(\neg (\text{ok} \land P)) \land \text{R1}(true)))$
by simp
also from assms have ...
$= ((\text{R1}(\neg \text{ok} \lor \neg P \lor Q)) \land \text{R1}(\neg R \lor (\text{ok}' \land S))$
$\lor (\text{R1}(\neg \text{ok} \lor \neg P) \land \text{R1}(true)))$
by (simp add: impl-alt-def utp-pred-laws.sup.assoc)
also from assms have ...
$= (((\text{R1}(\neg \text{ok} \lor \neg P) \lor \text{R1}(Q))) \land \text{R1}(\neg R \lor (\text{ok}' \land S))$
$\lor (\text{R1}(\neg \text{ok} \lor \neg P) \land \text{R1}(true)))$
by (simp add: R1-disj utp-pred-laws.disj-assoc)
also from assms have ...
$= ((\text{R1}(\neg \text{ok} \lor \neg P) \lor \text{R1}(\neg R \lor (\text{ok}' \land S))$
$\lor (\text{R1}(\neg \text{ok} \lor \neg P) \land \text{R1}(true)))$
by (simp add: seqr-or-distl utp-pred-laws.sup.assoc)
also from assms have ...
$= ((\text{R1}(Q) \lor \text{R1}(\neg R \lor (\text{ok}' \land S))$
$\lor (\text{R1}(\neg \text{ok} \lor \neg P) \land \text{R1}(true)))$
by (rel-blast)
also from assms have ...
$= ((\text{R1}(Q) \lor (\text{R1}(\neg R) \lor \text{R1}(S) \land \text{ok}'))$
$\lor (\text{R1}(\neg \text{ok} \lor \neg P) \land \text{R1}(true)))$
by (simp add: R1-disj R1-extend-conj utp-pred-laws.inf-commute)
also have ...
$= ((\text{R1}(Q) \lor (\text{R1}(\neg R) \lor \text{R1}(S) \land \text{ok}'))$
$\lor (\text{R1}(\neg \text{ok}) \lor (\text{R1}(\neg P) \land \text{R1}(true)))$
by (simp add: R1-disj seqr-or-distl)
also have ...
$= ((\text{R1}(Q) \lor (\text{R1}(\neg R) \lor \text{R1}(S) \land \text{ok}'))$
$\lor (\text{R1}(\neg \text{ok}))$
$\lor (\text{R1}(\neg P) \land \text{R1}(true)))$
proof
  have \(((R1(\neg \$ok) :: ('t,'a',\beta) rel-rp) ;; R1(true)) =
    (R1(\neg \$ok) :: ('t,'a',\gamma) rel-rp)
  by (rel-auto)
  thus \?thesis
  by simp
qed
also have \ldots = (((R1(Q) ;; (R1(\neg R) \lor (R1(S \& \$ok')))))
  \lor R1(\neg \$ok)
  \lor (R1(\neg P) ;; R1(true)))
  by (simp add: R1-extend-conj)
also have \ldots = (((R1(Q) ;; (R1(\neg R))))
  \lor (R1(Q) ;; (R1(S \& \$ok')))
  \lor (\neg \$ok)
  \lor (R1(\neg P) ;; R1(true)))
  by (simp add: R1-disj R1-seqr)
also have \ldots = R1(((R1(Q) ;; (R1(\neg R))))
  \lor ((R1(Q) ;; (R1(S) \& \$ok')))
  \lor (\neg \$ok)
  \lor (R1(\neg P) ;; R1(true)))
  by (rel-blast)
also have \ldots = R1((\neg(\$ok \& \neg (R1(\neg P) ;; R1(true)) \& \neg (R1(Q) ;; (R1(\neg R)))))))
  \lor ((R1(Q) ;; R1(S)) \& \$ok'))))
  by (rel-blast)
also have \ldots = R1((\neg(\$ok \& \neg (R1(\neg P) ;; R1(true)) \& \neg (R1(Q) ;; (R1(\neg R))))))
  \lor ((R1(Q) ;; R1(S))))
  by (simp add: impl-alt-def utp-pred-laws.inf-commute)
also have \ldots = R1(((\neg (R1(\neg P) ;; R1(true)) \& \neg (R1(Q) ;; R1(\neg R))) \vdash (R1(Q) ;; R1(S))))
  by (simp add: design-def)
finally show \?thesis.
qed

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

theorem R1-design-composition-RC:
  assumes \(P\) is RC \(Q\) is RR \(R\) is RR \(S\) is RR
  shows \(((R1(P \vdash Q) ;; R1(R \vdash S)) = R1((P \& Q wp_r R) \vdash (Q ;; S))\)
  by (simp add: R1-design-composition-RR assms unrest Healthy-if closure wp)

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

lemma R1-R3c-design:
$R1(R3c(P \vdash Q)) = R1(R3c-pre(P) \vdash R3c-post(Q))$
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 $\$ok' \notin P \$ok' \notin Q \$ok \notin R \$ok \notin S
shows $(R3c(R1(P \vdash Q)) ;; R3c(R1(R \vdash S))) = R3c(R1((\neg (R1(\neg P)) ;; R1(true)) \land \neg (((R1(Q) \land \neg \$wait') ;; R1(\neg R)))
\vdash (R1(Q) ;; ([I]_D \land \$wait \land R1(S))))$)
proof
have 1:$(\neg (R1 (\neg R3c-pre P) ;; R1 true)) = (R3c-pre (\neg (R1 (\neg P) ;; R1 true)))$ by (rel-auto)
have 2:$(\neg (R1 (R3c-post Q) ;; R1 (\neg R3c-pre R))) = R3c-pre(\neg ((R1 Q \land \neg \$wait') ;; R1 (\neg R)))$ by (rel-auto, blast+)
have 3:$(R1 (R3c-post Q) ;; R1 (R3c-post S)) = R3c-post(R1 Q ;; ([I]_D \land \$wait \land R1 S))$ by (rel-auto)
show ?thesis
apply (simp add: R3c-semir-form R1-R3c-commute THEN sym) R1-R3c-design unrest)
apply (subst R1-design-composition)
apply (simp-all add: unrest assms R3c-pre-conj 1 2 3)
done

qed

lemma R3h-R1-design-composition:
assumes $\$ok' \notin P \$ok' \notin Q \$ok \notin R \$ok \notin S
shows $(R3h(R1(P \vdash Q)) ;; R3h(R1(R \vdash S))) = R3h(R1((\neg (R1(\neg P)) ;; R1(true)) \land \neg (((R1(Q) \land \neg \$wait') ;; R1(\neg R)))
\vdash (R1(Q) ;; (\exists \$st \cdot [I]_D) \land \$wait \land R1(S))))$)
proof
have 1:$(\neg (R1 (\neg R3c-pre P) ;; R1 true)) = (R3c-pre (\neg (R1 (\neg P) ;; R1 true)))$ by (rel-auto)
have 2:$(\neg (R1 (R3h-post Q) ;; R1 (\neg R3c-pre R))) = R3c-pre(\neg ((R1 Q \land \neg \$wait') ;; R1 (\neg R)))$ by (rel-auto, blast+)
have 3:$(R1 (R3h-post Q) ;; R1 (R3h-post S)) = R3h-post(R1 Q ;; (\exists \$st \cdot [I]_D) \land \$wait \land R1 S))$ by (rel-auto, blast+)
show ?thesis
apply (simp add: R3h-semir-form R1-R3h-commute THEN sym) R1-R3h-design unrest)
apply (subst R1-design-composition)
apply (simp-all add: unrest assms R3c-pre-conj 1 2 3)
done

qed

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

20
proof

lemma RH-design-composition:

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

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

$$RH((\neg (R1 (\neg R2s P) ; R1 true) \land \neg ((R1 (R2s Q) \land (\neg \$wait'))) ; R1 (\neg R2s R))) \vdash (R1 (R2s Q) ; (\neg R1 (R2s S))))$$

proof –

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

proof –

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

by (rel-auto)

have $R2c(R1(R2s (\neg P) ; R2 true)) = R2c(R1(R2s (\neg P) ; R2 true))$

using $R2c$-not by blast

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

by $R1$-$R2c$-commute $R1$-$R2c$-is-$R2$

also have $R2c = (R2s (\neg P) ; R2 true)$

by (simp add: $R2c$-segr-distracte)

also have $R2(R2s (\neg P) ; R2 true)$

by (simp add: $R2$-$def$ $R2$s-$not$ $R2$s-$true$)

finally show $\$thesis$

by (simp add: 1)

qed

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

proof –

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

by (rel-auto)

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

by $R1$-$R2c$-commute $R1$-$R2c$-is-$R2$ $R2$-$segr$-distracte

also have $R2(R2s (\neg P) ; R1 true)$

by (rel-auto)

finally show $\$thesis$

qed

have 3: $R2c((R1 (R2s Q) ; (\neg R1 (R2s S)))) = (R1 (R2s Q) ; (\neg R1 (R2s S)))$

proof –

have $R2c(((R1 (R2s Q))[true/\$wait'] ; ([\neg R1 (R2s S)][true/\$wait]))

= ((R1 (R2s Q))[true/\$wait'] ; ([\neg R1 (R2s S)][true/\$wait]))$

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

also have $R2c(R2s(Q[true/\$wait']) ; R2[\neg R1 (R2s S)][true/\$wait])$

by (metis $R2c$-$def$ $R2$-$des$-$lift$-$skip$ $R2$-$subst$-$wait$-$true$)

also have $R2s(Q[true/\$wait']) ; R2[[\neg R1 (R2s S)][true/\$wait]]$

using $\neg$-$R2c$-$seq$ by blast

also have $R2c = (R1 (R2s Q))[true/\$wait'] ; ([\neg R1 (R2s S)][true/\$wait])$

apply (simp add: subst $R2c$-$des$-$lift$-$skip$)

apply (metis $R2c$-$def$ $R2$-$des$-$lift$-$skip$ $R2$-$subst$-$wait$-$true$-$true$ $R2$-$subst$-$wait$-$true$)

done
finally show ?thesis.

qed

moreover have R2c(((R1 (R2s Q)) [false/wait']) ;; ([I]_D < $wait R1 (R2s S)) [false/wait]))

= ((R1 (R2s Q)) [false/wait'] ;; ([I]_D < $wait R1 (R2s S)) [false/wait])

by (simp add: usubst cond-unit-F)

(metis (no-types, hide-lams) R1-wait-false R1-wait-false R2-def R2-subst-wait-false R2c-seq)

ultimately show ?thesis

proof –

have [I]_D < $wait R1 (R2s S) = R2 ([I]_D < $wait R1 (R2s S))

by (simp add: R1-R2c-is-R2 R1-R2s-R2c R2-condr' R2-des-lift-skip R2s-wait)

then show ?thesis

by (simp add: R1-R2c-is-R2 R1-R2s-R2c R2c-seq)

qed

have (R1(R2s(R3c(P ⊓ Q)))) ;; R1(R2s(R3c(R ⊓ S))))

= ((R3c(R1(R2s(P) ⊓ R2s(Q)))) ;; R3c(R1(R2s(R) ⊓ R2s(S))))

by (metis (no-types, hide-lams) R1-R2s-R2c R1-R3c-commute R2c-R3c-commute R2s-design)

also have ... = R3c(R1 (~ (~ R1 (~ R2s P) ;; R1 true) ∧ ~ ((R1 (R2s Q) ∧ ~ $wait') ;; R1 (~ R2s R))) ⊢

(R1 (R2s Q) ;; ([I]_D < $wait R1 (R2s S))))

by (simp add: R2c-design R2c-and R2c-not 1 2 3)

finally show ?thesis

by (simp add: R1-R2s-R2c R1-R3c-commute R2c-R3c-commute RH-def)

qed

lemma RHS-design-composition:

assumes $ok' ≠ P $ok' ≠ Q $ok ≠ R $ok ≠ S

shows (R_n(P ⊓ Q)) ;; R_n(R ⊓ S) =

R_n((~ (R1 (~ R2s P) ;; R1 true) ∧ ~ ((R1 (R2s Q) ∧ ~ $wait')) ;; R1 (~ R2s R))) ⊢

(R1 (R2s Q) ;; ([I] D < $wait R1 (R2s S))))

proof –

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

proof –

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

by (rel-auto, blast)

have R2c(R1(R2 (~ P) ;; R2 true)) = R2c(R1(R2 (~ P) ;; R2 true))

using R2c-not by blast

also have ... = R2(R2 (~ P) ;; R2 true)

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

also have ... = (R2 (~ P) ;; R2 true)

by (simp add: R2-seqr-distribute)

also have ... = (R1 (~ R2s P) ;; R1 true)

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

finally show ?thesis

by (simp add: 1)

qed

have 2:R2c ((R1 (R2s Q) ∧ ~ $wait') ;; R1 (~ R2s R)) = ((R1 (R2s Q) ∧ ~ $wait') ;; R1 (~ R2s R))
proof

have \((\langle R1 \ (R2s \ Q) \wedge \neg \ $\text{wait}' \rangle :; R1 \ (\neg R2s \ R)) = R1 \ (R2s (Q \wedge \neg \ $\text{wait}' )) ;; R2 \ (\neg R))\)
by (rel-auto, blast+)

hence \(R2c: ((R1 \ (R2s \ Q) \wedge \neg \ $\text{wait}' ));; R1 \ (\neg R2s \ R)) = (R2 (Q \wedge \neg \ $\text{wait}' ));; R2 \ (\neg R))\)
by (metis (no-types, lifting) R1-R2c-commute R1-R2c-is-R2 R2-seqr-distribute)
also have \(= ((R1 \ (R2s \ Q) \wedge \neg \ $\text{wait}' ));; R1 \ (\neg R2s \ R))\)
by (rel-auto, blast+)

finally show \(?thesis \).
qed

have \(3:R2c((R1 \ (R2s \ Q) ;; ((\exists \ $st \cdot [\langle I \rangle ]_D) \triangleleft \ $\text{wait} \triangleright R1 \ (R2s \ S))) = (R1 \ (R2s \ Q) ;; ((\exists \ $st \cdot [\langle I \rangle ]_D) \triangleleft \ $\text{wait} \triangleright R1 \ (R2s \ S)))\)
proof

have \(R2c(((R1 \ (R2s \ Q))(true/$\text{wait}') ;; ((\exists \ $st \cdot [\langle I \rangle ]_D) \triangleleft \ $\text{wait} \triangleright R1 \ (R2s \ S))(true/$\text{wait}'))) = ((R1 \ (R2s \ Q))(true/$\text{wait}') ;; ((\exists \ $st \cdot [\langle I \rangle ]_D) \triangleleft \ $\text{wait} \triangleright R1 \ (R2s \ S))(true/$\text{wait}'))\)
by (simp add: usubst cond-unit-T R1-def R2s-def)
also have \(= R2c(R2(Q(true/$\text{wait}') ;; R2((\exists \ $st \cdot [\langle I \rangle ]_D)(true/$\text{wait}')) = R2c(R1 \ (R2s \ Q(true/$\text{wait}')) ;; ((\exists \ $st \cdot [\langle I \rangle ]_D)(true/$\text{wait}'))\)
using R2c-seq by blast
also have \(= (R1 \ (R2s \ Q))(true/$\text{wait}') ;; ((\exists \ $st \cdot [\langle I \rangle ]_D) \triangleleft \ $\text{wait} \triangleright R1 \ (R2s \ S))(true/$\text{wait}'))\)
apply (simp add: usubst R2-des-lift-skip)
apply (metis (no-types) R2-def R2-des-lift-skip R2-st-ex R2-subst-wait'-true R2-subst-wait-true)
done
finally show \(?thesis \).
qed

moreover have \(R2c(((R1 \ (R2s \ Q))(false/$\text{wait}') ;; ((\exists \ $st \cdot [\langle I \rangle ]_D) \triangleleft \ $\text{wait} \triangleright R1 \ (R2s \ S))(false/$\text{wait}'))) = ((R1 \ (R2s \ Q))(false/$\text{wait}') ;; ((\exists \ $st \cdot [\langle I \rangle ]_D) \triangleleft \ $\text{wait} \triangleright R1 \ (R2s \ S))(false/$\text{wait}'))\)
by (simp add: usubst)
(metis (no-types, lifting) R1-wait'-false R1-wait-wait-wait-wait-true' R2-R1-form R2-subst-wait'-false R2-subst-wait-true)

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

lemma RHS-R2s-design-composition:
assumes

\(\)
show \( R_s(P \vdash Q) ; R_s(R \vdash S) \) =
\( R_s(\langle \neg R1 \ (\neg P) ; R1 \ true \rangle \land \neg ((R1 \ Q \land \neg \$\text{wait'}) ; R1 \ (\neg R)) \rangle \vdash (R1 \ Q ; ((\exists \$st \cdot [I\_]\ P) \land \$\text{wait} \triangleright R1 \ S))) \)

proof –
have \( f1 \): R2s \( P = P \)
  by (meson Healthy-def assms(5))
have \( f2 \): R2s \( Q = Q \)
  by (meson Healthy-def assms(6))
have \( f3 \): R2s \( R = R \)
  by (meson Healthy-def assms(7))
have \( R2s \ S = S \)
  by (meson Healthy-def assms(8))
then show \( \sim\text{thesis} \)
using \( f3 \ f2 \ f1 \) by (simp add: RHS-design-composition assms(5) assms(6) assms(7) assms(8))
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: \( R_s(P \vdash Q) = R_s(P \vdash R1(Q)) \)
  by (rel-auto)
lemma RHS-design-export-R2s: \( R_s(P \vdash Q) = R_s(P \vdash R2s(Q)) \)
  by (rel-auto)
lemma RHS-design-export-R2c: \( R_s(P \vdash Q) = R_s(P \vdash R2c(Q)) \)
  by (rel-auto)
lemma RHS-design-export-R2: \( R_s(P \vdash Q) = R_s(P \vdash R2(Q)) \)
  by (rel-auto)
lemma R1-design-R1-pre: \( R_s(R1(P) \vdash Q) = R_s(P \vdash Q) \)
  by (rel-auto)
lemma RHS-design-ok-wait: \( R_s(P[true,\false/\$\text{ok,}\$\text{wait}] \vdash Q[true,\false/\$\text{ok,}\$\text{wait}]) = R_s(P \vdash Q) \)
  by (rel-auto)
lemma RHS-design-neg-R1-pre: \( R_s((\neg R1 \ P) \vdash R) = R_s((\neg P) \vdash R) \)
  by (rel-auto)
lemma RHS-design-conj-neg-R1-pre: \( R_s((\neg R1 \ P) \land Q) \vdash R) = R_s(((\neg P) \land Q) \vdash R) \)
  by (rel-auto)
lemma RHS-pre-lemma: \( (R_s \ P)^f f = R1(R2c(P^f f)) \)
3.4 Refinement introduction laws

lemma RHS-design-refine:
  assumes
  \( P_1 \) is \( R_1 \) \( P_2 \) is \( R_1 \) \( Q_1 \) is \( R_1 \) \( Q_2 \) is \( R_1 \)
  \( \$ok \not\in P_1 \$ok' \not\in P_1 \$ok \not\in P_2 \$ok' \not\in P_2 \)
  \( \$ok \not= Q_1 \$ok' \not= Q_1 \$ok \not= Q_2 \$ok' \not= Q_2 \)
  shows \( R_1(P_1 \vdash P_2) \subseteq R_1(Q_1 \vdash Q_2) \iff 'P_1 \Rightarrow Q_1' \land 'P_1 \land Q_2 \Rightarrow P_2' \)

proof
  have \( R_1((\exists \$ok: \$ok' \cdot P_1) \vdash (\exists \$ok: \$ok' \cdot Q_1)) \subseteq R_1((\exists \$ok: \$ok' \cdot P_1) \Rightarrow (\exists \$ok: \$ok' \cdot Q_1)) \land R_1((\exists \$ok: \$ok' \cdot P_1) \land R_1((\exists \$ok: \$ok' \cdot P_1)) \Rightarrow 'P_1 \Rightarrow Q_1' \land 'P_1 \land Q_2 \Rightarrow P_2' \)

  thus \( ?thesis \)
  by (simp-all add: ex-unrest ex-plus Healthy-if assms)

qed

lemma RHS-design-refine-RR:
  assumes \( P_1 \) is \( RR \) \( P_2 \) is \( RR \) \( Q_1 \) is \( RR \) \( Q_2 \) is \( RR \)
  shows \( R_1(P_1 \vdash P_2) \subseteq R_1(Q_1 \vdash Q_2) \iff 'P_1 \Rightarrow Q_1' \land 'P_1 \land Q_2 \Rightarrow P_2' \)

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

lemma RHS-design-refine:
  assumes \( P_1 \) is \( R_1 \) \( P_2 \) is \( R_2c \) \( Q_1 \) is \( R_2c \) \( Q_2 \) is \( R_2c \)
  \( \$ok \not\in P_1 \$ok' \not\in P_1 \$ok \not\in P_2 \$ok' \not\in P_2 \)
  \( \$ok \not= Q_1 \$ok' \not= Q_1 \$ok \not= Q_2 \$ok' \not= Q_2 \)
  \( \$wait \not\in P_1 \$wait \not\in P_2 \$wait \not\in Q_1 \$wait \not\in Q_2 \)
  shows \( R_2c(P_1 \vdash P_2) \subseteq R_2c(Q_1 \vdash Q_2) \iff 'P_1 \Rightarrow Q_1' \land 'P_1 \land Q_2 \Rightarrow P_2' \)

proof
  have \( R_2c(P_1 \vdash P_2) \subseteq R_2c(Q_1 \vdash Q_2) \iff R_1(R_3h(R_2c(P_1 \vdash P_2)) \supseteq R_1(R_3h(R_2c(Q_1 \vdash Q_2))) \)
    by (simp add: R2c-R3h-commute RHS-def)
  also have \( ... \iff R_1(R_3h(P_1 \vdash P_2)) \supseteq R_1(R_3h(Q_1 \vdash Q_2)) \)
    by (simp add: Healthy-if R2c-design assms)
  also have \( ... \iff R_1(R_3h(P_1 \vdash P_2))[false/$\text{\text{\$wait}} \supseteq R_1(R_3h(Q_1 \vdash Q_2))[false/$\text{\text{\$wait}} \)
    by (rel-auto, meson+)
  also have \( ... \iff R_1(P_1 \vdash P_2)[false/$\text{\text{\$wait}} \supseteq R_1(Q_1 \vdash Q_2)[false/$\text{\text{\$wait}} \)
    by (rel-auto)
  also have \( ... \iff R_1(P_1 \vdash P_2) \subseteq R_1(Q_1 \vdash Q_2) \)
    by (simp add: usubst assms closure unrest)
  also have \( ... \iff 'P_1 \Rightarrow Q_1' \land 'P_1 \land Q_2 \Rightarrow P_2' \)
    by (simp add: R1-design-refine assms)
  finally show \( ?thesis \)

qed
3.5 Distribution laws

lemma **RHS-design-choice**: \( \mathbf{R}_s(P_1 \vdash Q_1) \cap \mathbf{R}_s(P_2 \vdash Q_2) = \mathbf{R}_s((P_1 \land P_2) \vdash (Q_1 \lor Q_2)) \)
by (metis RHS-inf design-choice)

lemma **RHS-design-sup**: \( \mathbf{R}_s(P_1 \vdash Q_1) \cup \mathbf{R}_s(P_2 \vdash Q_2) = \mathbf{R}_s((P_1 \lor P_2) \vdash ((P_1 \Rightarrow Q_1) \land (P_2 \Rightarrow Q_2))) \)
by (metis RHS-sup design-inf)

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

end

4 Reactive Design Triples

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

4.1 Diamond notation

definition **wait’-cond** :: 
\('t::trace, \alpha, \beta) rel-rp \Rightarrow ('t,\alpha,\beta) rel-rp \Rightarrow ('t,\alpha,\beta) rel-rp \ (\text{infixr} \ 65) \ \text{where} \n\ [\text{upred-defs}]: P \circ Q = (P \circ \text{\$wait'} \circ Q) 
lemma **wait’-cond-unrest [unrest]**: 
\[ [ out-var wait \alpha \cdot x; x \notin P; x \notin Q ] \] \Rightarrow x \notin (P \circ Q)  
by (simp add: wait’-cond-def unrest)

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

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

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

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

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

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

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

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

26
lemma subst-wait'-cond-false [usubst]: \((P \circ Q)[\text{false/\$wait'}] = Q[\text{false/\$wait'}]\)
by (rel-auto)

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

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

lemma wait'-cond-split: \(P[\text{true/\$wait'}] \circ P[\text{false/\$wait'}] = P\)
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 R2s-wait'-cond: \(R2s(P \circ Q) = R2s(P) \circ R2s(Q)\)
by (simp add: wait'-cond-def R2s-def usubst)

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

lemma wait'-cond-R1-closed [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-condr wait'-cond-def Healthy-def, rel-auto)

4.2 Export laws

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

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

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

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

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

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

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

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

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

lemma RHS-design-peri-R2c: \( R_s(P \vdash R2c(Q) \circ R) = R_s(P \vdash Q \circ R) \)
by (metis R1-R2s-wait'-cond RHS-design-export-R1 RHS-design-peri-R1)

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

lemma RH-design-lemma1:
\( RHS(P \vdash (R1(R2c(Q)) \lor R) \circ S) = RHS(P \vdash (Q \lor R) \circ S) \)
by (metis (no-types, lifting) R1-R2c-is-R2 R1-R2s-R2c RHS-design-export-R1)

4.3 Pre-, peri-, and postconditions

4.3.1 Definitions

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

abbreviation \( \text{np}_{\text{pre}}(P) \equiv \text{pre}_{s} \vdash P \)
definition [upred-defs]: \( \text{pre}_{R}(P) = (\neg_{v} \text{np}_{\text{pre}}(P)) \)
definition [upred-defs]: \( \text{cmt}_{R}(P) = R1(\text{cmt}_{s} \vdash P) \)
definition [upred-defs]: \( \text{peri}_{R}(P) = R1(\text{peri}_{s} \vdash P) \)
definition [upred-defs]: \( \text{post}_{R}(P) = R1(\text{post}_{s} \vdash P) \)

4.3.2 Unrestriction laws

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

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

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

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

lemma \( \text{ok'-pre-unrest} \) [unrest]: \( \text{ok}' \not\in \text{pre}_{R} P \)
by (simp add: pre_{R}-def unrest usubst)
lemma $\text{ok'}\text{-peri-unrest} [\text{unrest}]: \text{ok'} \# \text{peri}_R P$
by (simp add: peri$_R$-def unrest usubst)

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

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

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

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

lemma $\text{wait-cmt-unrest} [\text{unrest}]: \text{wait} \# \text{cmt}_R P$
by (simp add: cmt$_R$-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 (simp add: design-def pre$_R$-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 (simp add: design-def usubst wait'-'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 (simp add: design-def usubst wait'-'cond-def)

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

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

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

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

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

lemma $\text{post}_s\text{-R1 [usubst]}: \text{post}_s \uparrow R1(P) = R1(\text{post}_s \uparrow P)$
by (simp add: R1-def usubst)
lemma post_s-R2c [usubst]: post_s † R2c(P) = R2c(post_s † P)
by (simp add: R2c-def R2s-def usubst)

lemma cmt_s-R1 [usubst]: cmt_s † R1(P) = R1(cmt_s † P)
by (simp add: R1-def usubst)

lemma cmt_s-R2c [usubst]: cmt_s † R2c(P) = R2c(cmt_s † P)
by (simp add: R2c-def R2s-def usubst)

lemma pre-wait-false:
pre_R(P[false/$wait]) = pre_R(P)
by (rel-auto)

lemma cmt-wait-false:
cmt_R(P[false/$wait]) = cmt_R(P)
by (rel-auto)

lemma rea-pre-RHS-design: pre_R(Rs(P[true/false/ok/wait]) ⊢ Q) = R1(R2c(pre_s † P))
by (simp add: R3h-def pre_R-pre_s-design)

lemma rea-cmt-RHS-design: cmt_R(Rs(P[true/false/ok/wait]) ⊢ Q) = R1(R2c(cmt_s † P[true/false/ok/wait] ⇒ r Q))
by (simp add: R3h-def)

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

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

lemma rdes-export-cmt: Rs(P ⊢ cmt_s † Q) = Rs(P ⊢ Q)
by (rel-auto)

lemma rdes-export-pre: Rs((P[true,false/ok/wait] ⊢ Q) = Rs(P ⊢ Q)
by (rel-auto)

4.3.4 Healthiness laws

lemma wait′-unrest-pre-SRD [unrest]:
$wait′ ∗ pre_R(P) \Rightarrow \exists \overline{\text{pre}_R}(P \rightarrow \overline{\text{SRD}} P)
apply (rel-auto)
using least-zero apply blast+
done

lemma R1-R2s-cmt-SRD:
assumes P is SRD
shows R1(R2s(cmt_R(P))) = cmt_R(P)
by (metis (no-types, lifting) R1-R2c-commute R1-R2s-R2c R1-idem R2c-idem SRD-reactive-design
assms rea-cmt-RHS-design)
lemma R1-R2s-peri-SRD:
  assumes P is SRD
  shows R1(R2s(peri_R(P))) = peri_R(P)
  by (metis (no-types, hide-lams) Healthy-def R1-R2s-R2c R2-def R2-idem RHS-def SRD-RH-design-form assms R1-idem peri_R-def peri_s-R1 peri_s-R2c)

lemma R1-peri-SRD:
  assumes P is SRD
  shows R1(peri_R(P)) = peri_R(P)
proof –
  have R1(peri_R(P)) = R1(R1(R2s(peri_R(P))))
    by (simp add: R1-R2s-peri-SRD assms)
  also have ... = 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 =⇒ peri_R(P) is R1
by (simp add: Healthy-def’ R1-peri-SRD)

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

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

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

lemma R1-post-SRD:
  assumes P is SRD
  shows R1(post_R(P)) = post_R(P)
proof –
  have R1(post_R(P)) = R1(R1(R2s(post_R(P))))
    by (simp add: R1-R2s-post-SRD assms)
  also have ... = post_R(P)
    by (simp add: R1-idem, simp add: R1-R2s-post-SRD assms)
  finally show ?thesis.
qed

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

lemma postR-SRD-R1 [closure]: P is SRD =⇒ post_R(P) is R1
by (simp add: Healthy-def’ R1-post-SRD)
lemma R1-R2c-post-RHS:
  assumes P is SRD
  shows R1(R2c(post_R(P))) = post_R(P)
  by (metis R1-R2s-R2c R1-R2s-post-SRD assms)

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

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

lemma preR-R2c-closed [closure]: P is SRD \implies pre_R(P) is R2c
  by (simp add: Healthy-def)

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

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

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

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

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

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

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

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

4.3.5 Calculation laws

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

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

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

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

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

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

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

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

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

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

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

lemma postR-srdes-skip [rdes]: post$_R(\text{II}_R) = (\text{str}' =_u \text{str} \land [\text{II}]_R)$
33
by (rel-auto)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**lemma postR-inf [rdes]:** $\post_R(P \sqcap Q) = (\post_R(P) \lor \post_R(Q))$
lemma \text{preR-SUP [rdes]: } \text{pre}_R(\bigcup A) = (\bigvee P \in A \cdot \text{pre}_R(P))

by \text{ (rel-auto)}

lemma \text{periR-SUP [rdes]: } A \neq \{\} \implies \text{peri}_R(\bigcup A) = (\bigwedge P \in A \cdot \text{peri}_R(P))

by \text{ (rel-auto)}

lemma \text{postR-SUP [rdes]: } A \neq \{\} \implies \text{post}_R(\bigcup A) = (\bigwedge P \in A \cdot \text{post}_R(P))

by \text{ (rel-auto)}

4.4 Formation laws

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

by \text{ (simp add: srdes-skip-def, rel-auto)}

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

by \text{ (simp add: Chaos-def design-false-pre)}

lemma \text{Miracle-tri-def [rdes-def]: } \text{Miracle} = \text{R}_s(\text{true}_r \vdash \text{false} \circ \text{false})

by \text{ (simp add: Miracle-def R1-design-R1-pre wait’-cond-idem)}

lemma \text{RHS-tri-design-form:}

assumes \( P_1 \) is \text{RR} \( P_2 \) is \text{RR} \( P_3 \) is \text{RR}

shows \( \text{R}_s(P_1 \vdash P_2 \circ P_3) = (\text{II}_R \circ \text{\$wait} \triangleright ((\text{\$ok} \land P_1) \Rightarrow_r (\text{\$ok} \land (P_2 \circ P_3)))) \)

proof

have \( \text{R}_s(\text{II}_R(P_1) \vdash \text{RR}(P_2) \circ \text{RR}(P_3)) = (\text{II}_R \circ \text{\$wait} \triangleright ((\text{\$ok} \land \text{RR}(P_1)) \Rightarrow_r (\text{\$ok} \land (\text{RR}(P_2) \circ \text{RR}(P_3)))) \)

apply \text{ (rel-auto) using minus-zero-eq by blast}

thus \ Technician

by \text{ (simp add: Healthy-if assms)}

qed

lemma \text{RHS-design-pre-post-form:}

\( \text{R}_s(\neg P^f_j \vdash P^t_j) = \text{R}_s(\text{pre}_R(P) \vdash \text{cmt}_R(P)) \)

proof

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

by \text{ (simp add: design-subst-ok)}

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

by \text{ (simp add: preR-def cmtR-def usubst, rel-auto)}

finally show \ Technician

qed

lemma \text{SRD-as-reactive-design:}

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

by \text{ (simp add: RHS-design-pre-post-form SRD-RH-design-form)}

lemma \text{SRD-reactive-design-alt:}

assumes \( P \) is \text{SRD}

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

proof

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

by \text{ (simp add: RHS-design-pre-post-form)}

thus \ Technician

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

qed
lemma SRD-reactive-tri-design-lemma:
SRD(P) = Rₜ(¬ P[t] ⊢ P[t][true/\$wait`;] ∨ P[t][false/\$wait`;])
by (simp add: SRD-RH-design-form wait'-cond-split)

lemma SRD-as-reactive-tri-design:
SRD(P) = Rₜ(preₐ(P) ⊢ periₐ(P) ∨ postₐ(P))
proof –
have SRD(P) = Rₜ(¬ P[t] ⊢ P[t][true/\$wait`;] ∨ P[t][false/\$wait`;])
  by (simp add: SRD-RH-design-form wait'-cond-split)
also have ... = Rₜ(preₐ(P) ⊢ periₐ(P) ∨ postₐ(P))
  apply (simp add: usubst)
  apply (subst design-subst-ok-ok[THEN sym])
  apply (simp add: preₐ-def periₐ-def postₐ-def usubst unrest)
  apply (rel-auto)
done
finally show ?thesis .
qed

lemma SRD-reactive-tri-design:
assumes P is SRD
shows Rₜ(preₐ(P) ⊢ periₐ(P) ∨ postₐ(P)) = P
by (metis Healthy-if SRD-as-reactive-tri-design assms)

lemma SRD-elim [RD-elim]: [ P is SRD; Q(Rₜ(preₐ(P) ⊢ periₐ(P) ∨ postₐ(P))) ] \implies Q(P)
by (simp add: SRD-reactive-tri-design)

lemma RHS-tri-design-is-SRD [closure]:
assumes $ok` \notin P $ok` \notin Q $ok` \notin R
shows Rₜ(P ⊢ Q ⊢ R) is SRD
by (rule RHS-design-is-SRD, simp-all add: unrest assms)

lemma SRD-rdes-intro [closure]:
assumes P is RR Q is RR R is RR
shows Rₜ(P ⊢ Q ○ R) is SRD
by (rule RHS-tri-design-is-SRD, simp-all add: unrest closure assms)

lemma USUP-R1-R2s-cmt-SRD:
assumes A ⊆ [SRD]
shows (\bigsqcup P ∈ A · R1 (R2s (cmtₐ P))) = (\bigsqcup P ∈ A · cmtₐ P)
by (rule USUP-cong[of A], metis (mono-tags, lifting) Ball-Collect R1-R2s-cmt-SRD assms)

lemma UINF-R1-R2s-cmt-SRD:
assumes A ⊆ [SRD]
shows (\bigsqcap P ∈ A · R1 (R2s (cmtₐ P))) = (\bigsqcap P ∈ A · cmtₐ P)
by (rule UINF-cong[of A], metis (mono-tags, lifting) Ball-Collect R1-R2s-cmt-SRD assms)

4.4.1 Order laws

lemma preR-antitone: P ⊆ Q \implies preₐ(Q) ⊆ preₐ(P)
by (rel-auto)

lemma periR-monotone: P ⊆ Q \implies periₐ(P) ⊆ periₐ(Q)
by (rel-auto)

lemma postR-monotone: P ⊆ Q \implies postₐ(P) ⊆ postₐ(Q)
4.5 Composition laws

**theorem RH-tri-design-composition:**

**assumes** $\exists sok \in P \ s.t. Q_1 \ s.o$ $\exists Q_2 \ s.o$ $\exists R \ s.o$ $S_1 \ s.o$ $S_2$

**shows** $(\text{RH}(P \vdash Q_1 \circ Q_2) ; \text{RH}(R \vdash S_1 \circ S_2)) =$

$$\text{RH}((\neg (R_1 (\neg R_2s P) ; R_1 true) \land \neg ((R_1 (R_2s Q_2) \land \neg \text{wait}')) ; R_1 (\neg R_2s R))) \vdash$

$$((Q_1 \lor (R_1 (R_2s Q_2) ; R_1 (R_2s S_1))) \lor ((R_1 (R_2s Q_2) ; R_1 (R_2s S_2))))$$

**proof**

**have** 1: $(\neg ((R_1 (R_2s (Q_1 \circ Q_2)) \land \neg \text{wait}')) ; R_1 (\neg R_2s R)) =$

$$\neg ((R_1 (R_2s Q_2) \land \neg \text{wait}')) ; R_1 (\neg R_2s R))$$

by (metis (no-types, hide-lams) R1-extend-conj R2s-cond R2s-not R3s-wait’ wait-cond-false)

**have** 2: $(R_1 (R_2s (Q_1 \circ Q_2)); ([II]_D \land \text{wait} \lor R_1 (R_2s (S_1 \circ S_2)))) =$

$$((R_1 (R_2s Q_1) \lor (R_1 (R_2s Q_2); R_1 (R_2s S_1))) \lor (R_1 (R_2s Q_2); R_1 (R_2s S_2)))$$

**proof**

**have** $(R_1 (R_2s Q_1); (\text{wait} \land ([II]_D \land \text{wait} \lor R_1 (R_2s S_1) \land R_1 (R_2s S_2))))$

$$= (R_1 (R_2s Q_1); (\text{wait} \land ([II]_D))$$

by (rel-auto)

**also have** ... $= ((R_1 (R_2s Q_1); [II]_D) \land \text{wait}'))$

by (rel-auto)

**also from** assms(2) **have** ... $= ((R_1 (R_2s Q_1)) \land \text{wait}’)$

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

**finally show** ?thesis .

**qed**

**moreover have** $(R_1 (R_2s Q_2); (\neg \text{wait} \land ([II]_D \land \text{wait} \lor R_1 (R_2s S_1) \land R_1 (R_2s S_2))))$

$$= ((R_1 (R_2s Q_2)); (R_1 (R_2s S_1) \land R_1 (R_2s S_2)))$$

**proof**

**have** $(R_1 (R_2s Q_2); (\neg \text{wait} \land ([II]_D \land \text{wait} \lor R_1 (R_2s S_1) \land R_1 (R_2s S_2))))$

$$= (R_1 (R_2s Q_2); (\neg \text{wait} \land (R_1 (R_2s S_1) \land R_1 (R_2s S_2))))$$

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** ... $= ((R_1 (R_2s Q_2)) \land \text{false/\text{wait}’}); (R_1 (R_2s S_1) \land R_1 (R_2s S_2));[\text{false/\text{wait}’})$

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

**also have** ... $= ((R_1 (R_2s Q_2)); (R_1 (R_2s S_1) \land R_1 (R_2s S_2)))$

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

**finally show** ?thesis .

**qed**

**moreover**

**have** $((R_1 (R_2s Q_1) \land \text{wait}’) \lor ((R_1 (R_2s Q_2)); (R_1 (R_2s S_1) \land R_1 (R_2s S_2))))$

$$= (R_1 (R_2s Q_1) \lor (R_1 (R_2s Q_2); R_1 (R_2s S_1)) \lor ((R_1 (R_2s Q_2); R_1 (R_2s S_2)))$$

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(((¬ r ) wp r false ∧ Q wp r R) ⊢ (Q ;; S))
apply (subst R1-design-composition)
apply (simp-all add: assms unrest wp-rea-def Healthy-if closure)
apply (rel-auto)
done

theorem R1-design-composition-RC:
assumes P is RC Q is RR R is RR S is RR
shows 
(R1 (P ⊢ Q) ;; R1 (R ⊢ S)) = R1((P ∧ Q wp r R) ⊢ (Q ;; S))
by (simp add: R1-design-composition-RR assms unrest Healthy-if closure wp)

theorem RHS-tri-design-composition:
assumes $ok’ ≠ P $ok’ ⊢ Q1 $ok’ ⊢ Q2 $ok ≠ R $ok ≠ S1 $ok ≠ S2
$wait ≠ R $wait ≠ Q2 $wait ≠ S1 $wait ≠ S2
shows (Rn (P ⊢ Q1 o Q2) ;; Rn (R ⊢ S1 o S2)) =
Rn(¬ (R1 (¬ R2s P) ;; R1 true) ∧ ¬ (R1 (R2s Q2) ;; R1 (¬ R2s R))) ⊢
(((⊢ st’ ∨ R1 (R2s Q1) ; R1 (R2s S1))) ⊢ (R1 (R2s Q2) ;; R1 (R2s S2))))

proof –
have 1:¬ ((R1 (R2s (Q1 o Q2)) ∧ ¬ $wait’) ;; R1 (¬ R2s R)) =
(¬ ((R1 (R2s Q2) ∧ ¬ $wait’) ;; R1 (¬ R2s R))
by (metis (no-types, hide-lams) R1-extend-conj R2s-conj R2s-not R2s-wait’ wait’-cond-false)
have 2: (R1 (R2s (Q1 o Q2)) ;; (⊢ st’ ; [II]D) ∨ $wait ⊢ R1 (R2s (S1 o S2))) =
(((⊢ st’ ∨ R1 (R2s Q1)) ∨ (R1 (R2s Q2) ;; R1 (R2s S1))) ⊢ (R1 (R2s Q2) ;; R1 (R2s S2)))))

proof –
have (R1 (R2s Q1) ;; (⊢ $wait ∨ (⊢ st’ ; [II]D) ∨ $wait ⊢ R1 (R2s S1) o R1 (R2s S2)))
= (⊢ st’ ∨ ((R1 (R2s Q1)) ∧ $wait’))

proof –
have (R1 (R2s Q1) ;; (⊢ $wait ∨ (⊢ st’ ; [II]D) ∨ $wait ⊢ R1 (R2s S1) o R1 (R2s S2)))
= (R1 (R2s Q1) ;; (⊢ $wait ∨ (⊢ st’ ; [II]D)))
by (rel-auto, blast+)
also have ... = ((R1 (R2s Q1) ;; (⊢ st’ ; [II]D)) ∧ $wait’)
by (rel-auto)
also from assms(2) have ... = (⊢ st’ : (R1 (R2s Q1)) ∧ $wait’)
by (rel-auto, blast)
finally show thesis .

qed

moreover have (R1 (R2s Q2) ;; (¬ $wait ∨ (⊢ st’ ; [II]D) ∨ $wait ⊢ R1 (R2s S1) o R1 (R2s S2)))))
= ((R1 (R2s Q2)) ;; (R1 (R2s S1) o R1 (R2s S2)))
proof
  have \( (R_1 (R_2s Q_2) \supset (\neg \$wait \land ((\exists s \cdot [H_{P}]_{S}) \land \$wait \supset R_1 (R_2s S_1) \diamond R_1 (R_2s S_2))) \)
  \( = (R_1 (R_2s Q_2) \supset (\neg \$wait \land (R_1 (R_2s S_1) \diamond R_1 (R_2s S_2))) \)
  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 \( \ldots = (\neg \$wait) \supset \ldots \)
  by (metis false-alt-def seq-right-one-point upred-eq-false wait-vw b-lens)

  also have \( \ldots = (\neg \$wait) \supset (R_1 (R_2s S_1) \diamond R_1 (R_2s S_2)) \)
  by (simp add: wait'-cond-def usesub unrest assms)

  finally show \(?thesis\).

  qed

moreover
  have \( ((\neg \$wait) \supset (R_1 (R_2s Q_2) \land \$wait) : (R_1 (R_2s S_1) \diamond R_1 (R_2s S_2))) \)
  \( = (R_1 (R_2s Q_1) \land (R_1 (R_2s Q_2) \supset R_1 (R_2s S_1))) \supset ((R_1 (R_2s Q_2) \diamond R_1 (R_2s S_2))) \)
  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'-seq ex-conj-contr-right unrest
  (simp add: cond-and-T-integrate cond-seq-right-distr unrest-var wait'-cond-def)

  qed

from assms(7,8) have \( 3: (R_1 (R_2s Q_2) \land \neg \$wait) : R_1 (\neg R_2s R) = R_1 (R_2s Q_2) \supset R_1 (\neg R_2s R) \)
  by (rel-auto, blast, meson)

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

  qed

  theorem RHS-tri-design-composition-wp:
  assumes \( \neg s \cdot [H_{P}]_{S} \land \neg s \cdot [H_{Q}]_{S} \land \neg s \cdot [H_{R}]_{S} \)
  \( = (R_1 (\neg P \supset Q_1) \supset (R_2s S_1) \supset (R_2s S_2)) \)
  by (simp add: RHS-tri-design-composition assms Healthy-if R2c-healthy-R2s disj-upred-def
  (metis (no-types, hide-lams) R1-negate-R1 R2c-healthy-R2s assms(11,16))

  also have \( \ldots = ?rhs\)
  by (rel-auto)
finally show \( \text{thesis} \).

\text{qed}

\text{theorem} RHS-tri-design-composition-RR-wp:
\text{assumes} \ P \text{ is RR} \ Q_1 \text{ is RR} \ Q_2 \text{ is RR}
\text{R is RR} \ S_1 \text{ is RR} \ S_2 \text{ is RR}
\text{shows} \ R_\sigma(P \vdash Q_1 \circ Q_2) ; R_\sigma(R \vdash S_1 \circ S_2) =
R_\sigma((\neg_r P) \ wp, false \wedge Q_2 \ wp_r R) \vdash((\exists \text{ $\ast$st' } \cdot \ Q_1) \cap (Q_2 ;; S_1)) \circ (Q_2 ;; S_2)) \ (\text{is} \ ?\text{lhs} = \ ?\text{rhs})
\text{by (simp add: RHS-tri-design-composition-wp add: closure assms unrest RR-implies-R2c)}

\text{lemma} RHS-tri-normal-design-composition:
\text{assumes} \ $\$ok' \not\in P $\$ok' \not\in Q_1 $\$ok' \not\in Q_2 $\$ok \not\in R $\$ok \not\in S_1 $\$ok \not\in S_2
\$wait \not\in R $\$wait \not\in Q_2 $\$wait \not\in S_1 $\$wait \not\in S_2
P \text{ is R}\_c Q_1 \text{ is R1 Q_1 is R2c Q_2 is R1 Q_2 is R2c}
R \text{ is R}\_c S_1 \text{ is R1 S_1 is R2c S_2 is R1 S_2 is R2c}
R_1 (\neg P) :: R_1 (\text{true}) = R_1 (\neg P) $\$st' \not\in Q_1
\text{shows} \ R_\sigma(P \vdash Q_1 \circ Q_2) ; R_\sigma(R \vdash S_1 \circ S_2)
= R_\sigma((P \wedge Q_2 \ wp_r R) \vdash (Q_1 \wedge (Q_2 ;; S_1)) \circ (Q_2 ;; S_2))
\text{proof --}
\text{have} \ R_\sigma(P \vdash Q_1 \circ Q_2) ; R_\sigma(R \vdash S_1 \circ S_2)
= R_\sigma((P \wedge Q_2 \ wp_r R) \vdash (Q_1 \wedge (Q_2 ;; S_1)) \circ (Q_2 ;; S_2))
\text{by (simp-all add: RHS-tri-design-composition-wp rea-not-def assms unrest)}
\text{also have} \ \ldots = R_\sigma((P \wedge Q_2 \ wp_r R) \vdash (Q_1 \wedge (Q_2 ;; S_1)) \circ (Q_2 ;; S_2))
\text{by (simp add: assms wp-rea-def ex-unrest, rel-auto)}
\text{finally show} \ \text{thesis} \.
\text{qed}

\text{lemma} RHS-tri-normal-design-composition'[rdes-def]:
\text{assumes} \ P \text{ is RC} Q_1 \text{ is RR} $\$st' \not\in Q_1 Q_2 \text{ is RR} R \text{ is RR} S_1 \text{ is RR} S_2 \text{ is RR}
\text{shows} \ R_\sigma(P \vdash Q_1 \circ Q_2) ; R_\sigma(R \vdash S_1 \circ S_2)
= R_\sigma((P \wedge Q_2 \ wp_r R) \vdash (Q_1 \wedge (Q_2 ;; S_1)) \circ (Q_2 ;; S_2))
\text{proof --}
\text{have} \ R_1 (\neg P) :: R_1 (\text{true}) = R_1 (\neg P)
\text{using} \ RC\text{-implies-RC1[OF assms(1)]}
\text{by (metis R1-negate-R1 R1-seqr utp-pred-laws.double-compl)}
\text{thus} \ \text{thesis}
\text{by (simp add: RHS-tri-normal-design-composition assms closure unrest RR-implies-R2c)}
\text{qed}

\text{lemma} RHS-tri-design-right-unit-lemma:
\text{assumes} \ $\$ok' \not\in P $\$ok' \not\in Q $\$ok' \not\in R $\$ok \not\in S
\$wait \not\in R $\$wait \not\in Q $\$wait \not\in S
\text{shows} \ R_\sigma(P \vdash Q \circ R) ; II_R = R_\sigma((\neg_r (\neg_r P) ;; true_r) \vdash ((\exists \text{ $\ast$st' } \cdot \ Q) \circ R))
\text{proof --}
\text{have} \ R_\sigma(P \vdash Q \circ R) ; II_R = R_\sigma(P \vdash Q \circ R) ; R_\sigma(true \vdash false \circ (\$str' = u \$str \wedge [II\_R]))
\text{by (simp add: srdes-skip-tri-design, rel-auto)}
\text{also have} \ \ldots = R_\sigma((\neg R_1 (\neg R_2 P) ;; R_1 (\text{true}) \vdash ((\exists \text{ $\ast$st' } \cdot Q) \circ (R_1 (R_2 S R) ;; R_1 (R_2 S (\$str' = u \$str \wedge [II\_R])))\))
\text{by (simp-all add: RHS-tri-design-composition assms unrest R2s-true R1-false R2s-false)}
\text{also have} \ \ldots = R_\sigma((\neg R_1 (\neg R_2 S P) ;; R_1 (\text{true}) \vdash ((\exists \text{ $\ast$st' } \cdot Q) \circ R_1 (R_2 S R)))\)
\text{proof --}
\text{from assms(3,4) have} \ (R_1 (R_2 S R) ;; R_1 (R_2 S (\$str' = u \$str \wedge [II\_R]))) = R_1 (R_2 S R)
\text{by (rel-auto, metis (no-types, lifting) minus-zero-eq, meson order-refl trace-class.diff-cancel)}
thus \( ?\text{thesis} \)
by simp

qed

also have \( \ldots = R_s((\neg (\neg P) ;; R1 \text{ true}) \vdash ((\exists \ $st' \cdot Q) \circ R)) \)
by (metis (no-types, lifting) R1-R2s-R1-true-lemma R1-R2s-R2c R2c-not RHS-design-R2c-pre RHS-design-neg-R1-pre RHS-design-post-R1 RHS-design-post-R2s)
also have \( \ldots = R_s((\neg (\neg P) ;; \text{ true}_r) \vdash ((\exists \ $st' \cdot Q) \circ R)) \)
by (rel-auto)

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

qed

lemma \text{SRD-composition-wp}:
assumes \( P \text{ is SRD} \)
shows \( (P ;; Q) = R_s(((\neg \text{ pre}_P P) \vdash \text{ peri}_R(P) \circ \text{ peri}_R(Q)) ;; R_s(\text{ peri}_P(Q) \circ \text{ peri}_R(Q))) \)
(is \( ?\text{lhs} = ?\text{rhs} \))

proof

have \( (P ;; Q) = R_s((\text{ peri}_P(P) \vdash \text{ peri}_R(P) \circ \text{ peri}_R(Q)) ;; R_s(\text{ peri}_P(Q) \circ \text{ peri}_R(Q))) \)
by (simp add: SRD-reactive-tri-design assms(1) assms(2))

also from assms

have \( \ldots = ?\text{rhs} \)
by (simp add: RHS-tri-composition-wp disj-upred-def unrest assms closure)

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

qed

4.6 Refinement introduction laws

lemma \text{RHS-tri-design-refine}:
assumes \( P_1 \text{ is RR} P_2 \text{ is RR} P_3 \text{ is RR} Q_1 \text{ is RR} Q_2 \text{ is RR} Q_3 \text{ is RR} \)
shows \( R_s(P_1 \vdash P_2 \circ Q_3) \subseteq R_s(Q_1 \vdash Q_2 \circ Q_3) \iff \symbol{331} P_1 \vdash Q_1' \land P_1' \land Q_2 \Rightarrow P_2' \land P_1' \land Q_3 \Rightarrow P_3' \)
(is \( ?\text{lhs} = ?\text{rhs} \))

proof

have \( ?\text{lhs} \iff \symbol{331} P_1 \vdash Q_1' \land \symbol{331} P_1 \land Q_2 \Rightarrow P_2 \land P_3' \)
by (simp add: RHS-design-refine assms closure RR-implies-R2c unrest ex-unrest)

also have \( \ldots \iff \symbol{331} P_1 \vdash Q_1' \land \symbol{331} P_1 \land Q_2 \circ (P_1 \land Q_3) \Rightarrow P_2 \land P_3 ' \)
by (rel-auto)

also have \( \ldots \iff \symbol{331} P_1 \vdash Q_1' \land \symbol{331} (P_1 \land Q_2) \circ (P_1 \land Q_3) \Rightarrow P_2 \land P_3 [\text{true}/\text{wait}] ' \land \symbol{331} (P_1 \land Q_2) \circ (P_1 \land Q_3) \Rightarrow P_2 \land P_3 [\text{false}/\text{wait}] ' \)
by (rel-auto, metis)

also have \( \ldots \iff \symbol{331} \)
by (simp add: usubst unrest assms)

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

qed

lemma \text{srdes-tri-refine-intro}:
assumes \( \symbol{331} P_1 \vdash P_2 \equiv P_2 \land Q_2 \Rightarrow Q_1 ' \equiv P_1 \land R_2 \Rightarrow R_1 ' \)
shows \( R_s(P_1 \vdash Q_1 \circ R_1) \subseteq R_s(P_2 \vdash Q_2 \circ R_2) \)
using assms
by (rule-tac srdes-refine-intro, simp-all, rel-auto)

lemma \text{srdes-tri-eq-intro}:
assumes \( P_1 = Q_1 P_2 = Q_2 P_3 = Q_3 \)
shows \( R_s(P_1 \vdash P_2 \circ P_3) = R_s(Q_1 \vdash Q_2 \circ Q_3) \)
using assms by (simp)
lemma srdes-tri-refine-intro':
  assumes P_2 ⊆ P_1 Q_1 ⊆ (P_1 ∧ Q_2) R_1 ⊆ (P_1 ∧ R_2)
  shows R_*(P_1 ⊨ Q_1 ⊨ R_1) ⊆ R_*(P_2 ⊨ Q_2 ⊨ R_2)
  using assms
  by (rule-tac srdes-tri-refine-intro, simp-all add: refBy-order)

lemma SRD-peri-under-pre:
  assumes P is SRD $wait' ⊑ do R
  shows (pre_R(P) ⇒ peri R(P)) = peri R(P)
  proof –
  have peri R(P) = peri R(* R(P))
  proof —
  by (simp add: SRD-reactive-tri-design assms)
  also have ... = (pre_R(P) ⇒ peri R(P))
  proof —
  by (simp add: refBy-order assms)
  finally show ?thesis ..
  qed

lemma SRD-post-under-pre:
  assumes P is SRD $wait' ⊑ do R
  shows (pre_R(P) ⇒ peri R(P)) = peri R(P)
  proof –
  have peri R(P) = peri R(* R(P))
  proof —
  by (simp add: SRD-reactive-tri-design assms)
  also have ... = (pre_R(P) ⇒ peri R(P))
  proof —
  by (simp add: refBy-order assms)
  finally show ?thesis ..
  qed

lemma SRD-refine-intro:
  assumes P is SRD Q is SRD
  `pre R(P) ⇒ pre R(Q)` `pre R(P) ∧ peri R(Q) ⇒ peri R(P)`
  `pre R(P) ⇒ peri R(Q)`
  `pre R(P) ∧ peri R(Q) ⇒ peri R(Q)`
  shows P ⊆ 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) ⇒ peri R(P) ⊆ (pre R(P) ∧ peri R(Q))`
  shows P ⊆ Q
  using assms by (rule-tac SRD-refine-intro, simp-all add: refBy-order)

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

4.7 Closure laws

lemma SRD-srdes-skip [closure]: II R is SRD
  by (simp add: srdes-skip-def RHS-design-is-SRD unrest)
lemma SRD-seqr-closure [closure]:
  assumes P is SRD Q is SRD
  shows \((P;;Q)\) is SRD
proof
  have \((P;;Q) = R_s((\lnot \text{pre}_R P) \wedge \text{post}_R P \wedge \text{pre}_R Q)\)
    by (simp add: SRD-composition-wp assms(1) assms(2))
also have \(\ldots\) is SRD
  by (rule RHS-design-is-SRD, simp-all add: wp-rea-def unrest)
finally show \(?thesis\).
qed

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

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

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

lemma SRD-Sup-closure [closure]:
  assumes \(A \subseteq [SRD]_H\) \(A \neq \{\}\)
  shows \((\bigsqcap A)\) is SRD
proof
  have \(\text{SRD } (\bigsqcap A) = (\bigsqcap (\text{SRD } 'A))\)
    by (simp add: ContinuousD SRD-Continuous assms(2))
also have \(\ldots = (\bigsqcap A)\)
    by (simp only: Healthy-carrier-image assms)
finally show \(?thesis\) by (simp add: Healthy-def)
qed

4.8 Distribution laws

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

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

lemma RHS-tri-design-sap [rdes-def]:
43
\[ \text{R}_s(P_1 \vdash P_2 \circ P_3) \cup \text{R}_s(Q_1 \vdash Q_2 \circ Q_3) = \text{R}_s((P_1 \lor Q_1) \vdash ((P_1 \Rightarrow P_2) \land (Q_1 \Rightarrow Q_2)) \circ ((P_1 \Rightarrow P_3) \land (Q_1 \Rightarrow Q_3))) \]

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

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

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

**Lemma** SRD-UINF [rdes-def]:
assumes \( A \neq \{ \} \)
shows \( \exists A = \text{R}_s((\bigwedge_{P \in A} \pre_{R}(P)) \vdash (\bigvee P \in A \cdot \peri_{R}(P)) \circ (\bigvee P \in A \cdot \post_{R}(P)) ) \)

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

also have \( \exists \cdot \text{pre}_{R-INF} \peri_{R-INF} \post_{R-INF} \exists \cdot \text{assms} \)
finally show \( \exists \cdot \text{thesis} \).

qed

**Lemma** RHS-tri-design-USUP [rdes-def]:
assumes \( A \neq \{ \} \)
shows \( (\bigcap i \in A \cdot \text{R}_s(P(i) \vdash Q(i) \circ R(i))) \vdash (\bigcap i \in A \cdot P(i)) \circ (\bigcap i \in A \cdot Q(i)) \circ (\bigcap i \in A \cdot R(i)) \)

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

**Lemma** SRD-UINF-mem:
assumes \( A \neq \{ \} \)
shows \( (\bigcap i \in A \cdot \text{R}_s(P \cdot i) = \text{R}_s((\bigwedge i \in A \cdot \pre_{R}(P \cdot i)) \vdash (\bigvee i \in A \cdot \peri_{R}(P \cdot i)) \circ (\bigvee i \in A \cdot \post_{R}(P \cdot i)) ) \)

(is \( \exists \cdot \text{lhs} = \exists \cdot \text{rhs} \)

proof –

have \( \exists \cdot \text{lhs} = (\bigcap (P \cdot A)) \)

by (rel-auto)

also have \( \exists \cdot = \text{R}_s ( (\bigwedge P \in A \cdot \pre_{R} Pa) \vdash (\bigcap P \in A \cdot \peri_{R} Pa) \circ (\bigcap P \in A \cdot \post_{R} Pa) ) \)

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

also have \( \exists \cdot = \exists \cdot \text{thesis} \).

qed

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

by (rel-auto)

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

proof –

have \( \text{R}_s((P \circ b \tri_{R} R) \vdash (Q_1 \circ b \tri_{R} S_1) \circ (Q_2 \circ b \tri_{R} S_2)) = \text{R}_s(P \vdash Q_1 \circ Q_2) \circ R2c([b]_{S \circ} \circ \text{R}_s(R \vdash S_1 \circ S_2)) \)

by (pred-auto)

also have \( \exists \cdot (P \vdash Q_1 \circ Q_2) \circ b \tri_{R} R \vdash S_1 \circ S_2) \)

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

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

lemma SRD-cond-srea [closure]:
  assumes P is SRD Q is SRD
  shows P \bowtie b \triangleright R Q is SRD
proof
  have P \bowtie b \triangleright R Q = \R_s((\pre_R(P) \vdash \peri_R(P) \circ \post_R(P)) \bowtie b \triangleright R \R_s(\pre_R(Q) \vdash \peri_R(Q) \circ \post_R(Q)))
    by (simp add: SRD-reactive-tri-design assms)
  also have ... = \R_s((\pre_R P \bowtie b \triangleright R \pre_R Q) \vdash (\peri_R P \bowtie b \triangleright R \peri_R Q) \circ (\post_R P \bowtie b \triangleright R \post_R Q))
    by (simp add: cond-srea-form)
  also have ... is SRD
    by (simp add: RHS-tri-design-is-SRD unrest)
finally show \"thesis\".
qed

4.9 Algebraic laws

lemma SRD-left-unit:
  assumes P is SRD
  shows I_R ;; P = P
proof
  have I_R ;; P = \R_s(true \vdash false) ;; \R_s(\pre_R(P) \vdash \cm_R(P))
    by (simp add: Miracle-def SRD-reactive-design-alt assms)
  also have ...
    by (simp add: RHS-design-composition unrest R1-false R2s-false R2s-true)
  also have ...
    by (simp add: Miracle-def)
finally show \"thesis\".
qed

lemma Chaos-left-zero:
  assumes P is SRD
  shows (Chaos ;; P) = Chaos
proof
  have Chaos ;; P = \R_s(false \vdash true) ;; \R_s(\pre_R(P) \vdash \cm_R(P))
Stateful reactive designs are left unital

overloading

srdes-unit == utp-unit :: (SRDES, (s,t::trace,α) rsp) uthy ⇒ (s,t,α) hrel-rsp

begin
definition srdes-unit :: (SRDES, (s,t::trace,α) rsp) uthy ⇒ (s,t,α) hrel-rsp where
srdes-unit T = H_R

end

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

4.10 Recursion laws

lemma mono-srd-iter:
assumes mono F F ∈ [SRD]H → [SRD]H
shows mono (λX. R₄(pre_R(F X) ⊢ peri_R(F X) o post_R (F X)))
apply (rule monoI)
apply (rule srdes-refine-intro')
apply (meson assms(1) monoE preR-antitone utp-pred-laws.le-infI2)
apply (meson assms(1) monoE periR-monotone utp-pred-laws.le-infI2)
apply (meson assms(1) monoE postR-monotone utp-pred-laws.le-infI2)
done

lemma mu-srd-SRD:
assumes mono F F ∈ [SRD]H → [SRD]H
shows (µ X · R₄ (pre_R (F X) ⊢ peri_R (F X) o 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)
lemma mu-srd-iter:
  assumes mono F F ∈ [SRD]H → [SRD]H
  shows (µ X ∈ R₄(preR(F(X)) ⊢ periR(F(X))) ⊢ postR(F(X)))) = F(µ X ∈ R₄(preR(F(X)) ⊢ periR(F(X))) ⊢ postR(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 mu-srd-SRD
  apply blast
done

lemma mu-srd-form:
  assumes mono F F ∈ [SRD]H → [SRD]H
  shows µF X ∈ R₄(preR(F(X)) ⊢ periR(F(X))) ⊢ postR(F(X))) = F(µ F X ∈ R₄(preR(F(X)) ⊢ periR(F(X))) ⊢ postR(F(X))))
proof
  have 1: F(µ F X ∈ R₄(preR(F(X)) ⊢ periR(F(X))) ⊢ postR(F(X)))) is SRD
    by (simp add: Healthy-apply-closed assms(1) assms(2) mu-srd-SRD)
  have 2: Mono SRDES F
    by (simp add: assms(1) mono-Monotone-utp-order)
  hence 3: µF X ∈ R₄(preR(F(X)) ⊢ periR(F(X))) ⊢ postR(F(X))) = µF X ∈ R₄(preR(F(X)) ⊢ periR(F(X))) ⊢ postR(F(X)))
    by (simp add: srdes-theory-continuous.LFP-unfold[THEN sym] assms)
  thus ?thesis
    using assms 1 3 srdes-theory-continuous.weak.LFP-lowerbound eq-iff mu-srd-iter
    by (metis (mono-tags, lifting))
qed

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

5 Normal Reactive Designs

theory utp-rdes-normal
import utp-rdes-triples
 beginning
  UTP−KAT.utp-kleene
end

This additional healthiness condition is analogous to H3

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

lemma RD3-idem: RD3(RD3(P)) = RD3(P)
proof
  have a: Hₚ ;; Hₚ = Hₚ
    by (simp add: SRD-left-unit SRD-srdes-skip)
  show ?thesis
    by (simp add: RD3-def seqr-assoc a)
\[\text{lemma } \text{RD3-Idempotent} \ [\text{closure}]: \text{Idempotent } \text{RD3}\]
\> by (simp add: Idempotent-def RD3-idem)

\[\text{lemma } \text{RD3-continuous}: \text{RD3}(\bigcap A) = (\bigcap P \in A. \text{RD3}(P))\]
\> by (simp add: RD3-def seq-Sup-distr)

\[\text{lemma } \text{RD3-Continuous} \ [\text{closure}]: \text{Continuous } \text{RD3}\]
\> by (simp add: Continuous-def RD3-continuous)

\[\text{lemma } \text{RD3-right-subsumes-RD2}: \text{RD2}(\text{RD3}(P)) = \text{RD3}(P)\]
\> proof –
\> have \(a:\Pi I \Rightarrow J = I\)
\> >> by (rel-auto)
\> show \(?\)thesis
\> >> by (metis (no-types, hide-lams) H2-def RD2-def RD3-def a seqr-assoc)

\[\text{lemma } \text{RD3-left-subsumes-RD2}: \text{RD3}(\text{RD2}(P)) = \text{RD3}(P)\]
\> proof –
\> have \(a:J \Rightarrow I = I\)
\> >> by (rel-simp, safe, blast+)
\> show \(?\)thesis
\> >> by (metis (no-types, hide-lams) H2-def RD2-def RD3-def a seqr-assoc)

\[\text{lemma } \text{RD3-implies-RD2}: \text{P is RD3} \implies \text{P is RD2}\]
\> by (metis Healthy-def RD3-right-subsumes-RD2)

\[\text{lemma } \text{RD3-intro-pre}:\]
\> assumes \(\text{P is SRD} \ (\neg_r \pre_R(P)) \Rightarrow \text{true}_r = (\neg_r \pre_R(P)) \# \text{peri}_R(P)\)
\> shows \(\text{P is RD3}\)
\> proof –
\> have \(\text{RD3}(P) = \text{R}_a((\neg_r \pre_R P) \\# \text{peri}_R P) \odot \text{post}_R P\)
\> >> by (simp add: RD3-def SRD-right-unit-tri-lemma assms)
\> also have \(\text{...} = \text{R}_a((\neg_r \pre_R P) \# \text{peri}_R P) \odot \text{post}_R P\)
\> >> by (simp add: assms(3) ex-unrest)
\> also have \(\text{...} = \text{R}_a((\neg_r \pre_R P) \# \text{peri}_R P) \odot \text{post}_R P\)
\> >> by (simp add: wait-cond-peri-post-cmt)
\> also have \(\text{...} = \text{R}_a(\pre_R P) \\# \text{cmt}_R P\)
\> >> by (simp add: assms(2) rpred wp-rea-def R1-preR)
\> finally show \(?\)thesis
\> >> by (metis Healthy-def SRD-as-reactive-design assms(1))

\[\text{lemma } \text{RHS-tri-design-right-unit-lemma}:\]
\> assumes \(\# \text{ok}_r \# P \# \text{ok}_r \# Q \# \text{ok}_r \# R \# \text{wait}_r \# R\)
\> shows \(\text{R}_a(\pre_R Q \odot R) \Rightarrow I = \text{R}_a((\neg_r (\neg_r P)) \Rightarrow (\exists \# \text{peri}_R P) \odot \text{post}_R P)\)
\> proof –
\> have \(\text{R}_a(\pre_R Q \odot R) \Rightarrow I = \text{R}_a((\neg_r (\neg_r P)) \Rightarrow (\exists \# \text{peri}_R P) \odot \text{post}_R P)\)
\> >> by (simp add: assms(1) ex-unrest)
\> also have \(\text{...} \Rightarrow \text{R}_a((\neg_r (\neg_r P)) \Rightarrow (\exists \# \text{peri}_R P) \odot \text{post}_R P)\)
\> >> by (simp add: assms(2) ex-unrest)
\> finally show \(?\)thesis
\> >> by (simp add: RHS-tri-design-composition assms unrest R2s-true R1-false R2s-false)
also have \( \ldots = R_u (\neg R1 (\neg R2s P) ;; R1 \text{ true}) \vdash (\exists \$st' \cdot Q) \circ R1 (R2s R) \)

proof –

from \( \text{assms}(3,4) \) have \( R1 (R2s R) ;; R1 (R2s (\$tr' =_u \$tr \land [I]_R))) = R1 (R2s R) \)

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

thus ?thesis 

by simp

qed

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

by (metis (no-types, lifting) R1-R2s-R1-true-lemma R1-R2s-R2c R2c-not RHS-design-R2c-pre RHS-design-neg-R1-pre RHS-design-post-R1 RHS-design-post-R2s)

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

by (rel-auto)

finally show ?thesis .

qed

lemma RHS-tri-design-RD3-intro:

assumes
\[ \text{where} \quad R1 \equiv \text{Healthy-def RD3-def} \]

shows \( R_u (P \vdash Q \circ R) \) is RD3

apply (simp add: Healthy-def RD3-def)

apply (subst RHS-tri-design-right-unit-lemma)

apply (simp-all add: asms unrest closure rpred)

done

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

lemma R1-right-unit-lemma:
\[ [\text{outa} \not \in \ldots , \text{outa} \not \in \ldots] \implies (\neg_r (b \lor \$tr \leq_u \$tr') ;; R1(\text{true}) = (\neg_r b \lor \$tr \leq_u \$tr') \]

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

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

assumes
\[ \text{where} \quad R1 \equiv \text{Healthy-def RD3-def} \]

shows \( R_u ((b \land \neg_r \$tr \leq_u \$tr') \vdash Q \circ R) \) is RD3

apply (rule RHS-tri-design-RD3-intro)

apply (simp-all add: asms unrest closure rpred)

apply (subst R1-right-unit-lemma)

apply (simp-all add: asms unrest)

done

definition NSRD :: \( ('s',t::trace,'a) \) hrel-rsp \( \Rightarrow ('s',t,'a) \) hrel-rsp

where \( \text{[upred-defs]:} \quad \text{NSRD} = R1 \circ RD3 \circ RHS \)

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

by (rel-auto, blast+)

lemma NSRD-is-SRD [closure]: \( P \) is NSRD \( \Rightarrow \) \( P \) is SRD

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

lemma NSRD-clim [RD-clim]:
\[ [ P \text{ is NSRD}; Q(R_u (\text{pre}_R(P) \vdash \text{peri}_R(P) \circ \text{post}_R(P))))] \implies Q(P) \]
by (simp add: RD-elim closure)

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)

proof

lemma NSRD-form:

NSRD(P) = R₄(⌜¬r (⌜¬r preR(P)) ; R1 true⌝ ⊨ ((∃ $st' · periR(P)) ◦ postR(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(R₄(preR(P) ⊨ periR(P) ◦ postR(P)))
  by (simp add: SRD-as-reactive-tri-design)

also have ... = R₄(preR(P) ⊨ periR(P) ◦ postR(P)) ; II R
  by (simp add: RD₃-def)

also have ... = R₄(⌜¬r (⌜¬r preR(P)) ; R1 true⌝ ⊨ ((∃ $st' · periR(P)) ◦ postR(P)))
  by (simp add: RHS-tri-design-right-unit-lemma unrest)

finally show ?thesis .

done

lemma NSRD-healthy-form:

assumes P is NSRD

shows R₄(⌜¬r (⌜¬r preR(P)) ; R1 true⌝ ⊨ ((∃ $st' · periR(P)) ◦ postR(P))) = P

by (metis Healthy-def NSRD-form assms)

lemma NSRD-Sup-closure [closure]:

assumes A ⊆ [NSRD]₇ A ≠ {} 

shows ⋃ A is NSRD

proof –

have NSRD(⋃ A) = (⋃ (NSRD 'A))
  by (simp add: ContinuousD NSRD-Continuous assms)

also have ... = (⋃ A) 
  by (simp only: Healthy-carrier-image assms)

finally show ?thesis by (simp add: Healthy-def)

done

lemma intChoice-NSRD-closed [closure]:

assumes P is NSRD Q is NSRD

shows P ∩ Q is NSRD

using NSRD-Sup-closure[of {P, Q}] by (simp add: assms)

lemma NSRD-SUP-closure [closure]:

Π i. i ∈ A ⇒ P(i) is NSRD; A ≠ {} \Π i∈A. P(i) is NSRD

by (rule NSRD-Sup-closure, auto)

lemma NSRD-neg-pre-unit:

assumes P is NSRD

shows (⌜¬r preR(P)⌝ ; true, = (⌜¬r preR(P)⌝)

proof –

have (⌜¬r preR(P)⌝ = (⌜¬r preR(R₄(⌜¬r (⌜¬r preR(P)) ; R1 true⌝ ⊨ ((∃ $st' · periR(P)) ◦ postR(P))))))
  by (simp add: NSRD-healthy-form assms)

also have ... = R1 (R₂c (⌜¬r preR P⌝ ; R1 true))
by (simp add: rea-pre-RHS-design R1-negate-R1 R1-idem R1-rea-not R2c-rea-not usubst rpred unrest closure)
also have \( \ldots = (\neg_r \text{ pre}_R \text{ P}) \ ; ; \ R1 \text{ true} \)
  by (simp add: R1-R2c-seqr-distribute closure assms)
finally show \(?\text{thesis}\)
  by (simp add: rea-not-def)
qed

lemma \(\text{NSRD-neg-pre-left-zero}\):
  assumes \(P \text{ is NSRD} \) \(Q \text{ is RD1} \)
  shows \((\neg_r \text{ pre}_R \text{ P}) ; ; Q = (\neg_r \text{ pre}_R \text{ P})\)
by (metis (no-types, hide-lams) NSRD-neg-pre-unit RD1-left-zero assms)

lemma \(\text{NSRD-st’-unrest-peri [unrest]}\):
  assumes \(P \text{ is NSRD} \)
  shows \(\$\text{st’} \sharp \text{ peri}_R \text{ P}\)
proof –
  have \(\text{peri}_R \text{ P} = \text{peri}_R \text{ R}_s((\neg_r (\neg_r \text{ pre}_R \text{ P}) ; ; R1 \text{ true}) \vdash ((\exists \$\text{st’} \cdot \text{peri}_R \text{ P}) \circ \text{post}_R \text{ P}))\)
    by (simp add: NSRD-healthy-form assms)
also have \(\ldots = (R1 (R2c (\neg_r (\neg_r \text{ pre}_R \text{ P}) ; ; R1 \text{ true} \Rightarrow_r (\exists \$\text{st’} \cdot \text{peri}_R \text{ P})))\)
    by (simp add: rea-peri-RHS-design usubst unrest)
also have \($\text{st’} \sharp \ldots\)
    by (simp add: R1-def R2c-def unrest)
finally show \(?\text{thesis}\).
qed

lemma \(\text{NSRD-wait’-unrest-pre [unrest]}\):
  assumes \(P \text{ is NSRD} \)
  shows \(\$\text{wait’} \sharp \text{ pre}_R \text{ P}\)
proof –
  have \(\text{pre}_R \text{ P} = \text{pre}_R \text{ R}_s((\neg_r (\neg_r \text{ pre}_R \text{ P}) ; ; R1 \text{ true}) \vdash ((\exists \$\text{st’} \cdot \text{peri}_R \text{ P}) \circ \text{post}_R \text{ P}))\)
    by (simp add: NSRD-healthy-form assms)
also have \(\ldots = (R1 (R2c (\neg_r (\neg_r \text{ pre}_R \text{ P}) ; ; R1 \text{ true} \Rightarrow_r (\exists \$\text{st’} \cdot \text{peri}_R \text{ P})))\)
    by (simp add: rea-pre-RHS-design usubst unrest)
also have \($\text{wait’} \sharp \ldots\)
    by (simp add: R1-def R2c-def unrest)
finally show \(?\text{thesis}\).
qed

lemma \(\text{NSRD-st’-unrest-pre [unrest]}\):
  assumes \(P \text{ is NSRD} \)
  shows \(\$\text{st’} \sharp \text{ pre}_R \text{ P}\)
proof –
  have \(\text{pre}_R \text{ P} = \text{pre}_R \text{ R}_s((\neg_r (\neg_r \text{ pre}_R \text{ P}) ; ; R1 \text{ true}) \vdash ((\exists \$\text{st’} \cdot \text{peri}_R \text{ P}) \circ \text{post}_R \text{ P}))\)
    by (simp add: NSRD-healthy-form assms)
also have \(\ldots = (R1 (R2c (\neg_r (\neg_r \text{ pre}_R \text{ P}) ; ; R1 \text{ true} \Rightarrow_r (\exists \$\text{st’} \cdot \text{peri}_R \text{ P})))\)
    by (simp add: rea-pre-RHS-design usubst unrest)
also have \($\text{st’} \sharp \ldots\)
    by (simp add: R1-def R2c-def unrest)
finally show \(?\text{thesis}\).
qed

lemma \(\text{NSRD-alt-def}\): \(\text{NSRD} \text{ P} = \text{RD3(SRD} \text{ P})\)
by (metis NSRD-def RD1-RD3-commute RD3-left-subsumes-RD2 SRD-def comp-eq-dest-lhs)

51
lemma pre-R-RR [closure]: \(P\) is NSRD \(\implies\) pre\(_R\)(\(P\)) is RR
by (rule RR-intro, simp-all add: closure unrest)

lemma NSRD-neg-pre-RC [closure]:
assumes \(P\) is NSRD
shows pre\(_R\)(\(P\)) is RC
by (rule RC-intro, simp-all add: closure assms NSRD-neg-pre-unit rpred)

lemma NSRD-intro:
assumes \(P\) is SRD (\(\neg\) pre\(_R\)(\(P\))) \(\land\) true
shows \(P\) is NSRD
proof
have NSRD\((P) = R_s\((\neg\) pre\(_R\)(\(P))) \(\land\) true \(\imp\) \((\exists\) st’ \(\land\) peri\(_R\)(\(P))) \(\o\) post\(_R\)(\(P)))
by (simp add: NSRD-form)
also have \(\ldots = R_s\((pre\(_R\) \(P\) \(\imp\) peri\(_R\) \(P\) \(\o\) post\(_R\) \(P\)))
by (simp add: assms ex-unrest rpred closure)
also have \(\ldots = P\)
by (simp add: SRD-reactive-tri-design assms comp-apply)
finally show \(\ldots\)
using Healthy-def by blast
qed

lemma NSRD-intro’:
assumes \(P\) is \(R2\) \(P\) is \(R3\h\) \(P\) is \(RD1\) \(P\) is \(RD3\)
shows \(P\) is NSRD
by (metis (no-types, hide-lams) Healthy-def NSRD-def \(R1\)-\(R2\)-is-\(R2\) RHS-def assms comp-apply)

lemma NSRD-RC-intro:
assumes \(P\) is SRD pre\(_R\)(\(P\)) is RC
shows \(P\) is NSRD
by (metis Healthy-def NSRD-form SRD-reactive-tri-design assms (1) assms (2) assms (3) ex-unrest rea-not-false wp-rea-RC-false wp-rea-def)

lemma NSRD-rdes-intro [closure]:
assumes \(P\) is RC \(Q\) is RR \(R\) is RR \$st’ \(\neq\) Q
shows \(R \(P\) \(\o\) \(Q\) \(\o\) \(R\)) is NSRD
by (rule NSRD-RC-intro, simp-all add: rdes closure assms unrest)

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

lemma NSRD-iff:
\(P\) is NSRD \(\iff\) (\(P\) is SRD) \(\land\) (\(\neg\) pre\(_R\)(\(P\))) \(\land\) \(R1\)(true) = (\(\neg\) pre\(_R\)(\(P\))) \(\land\) (\$st’ \(\neq\) 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
\[ P \subseteq Q \] is NSRD

\[ \begin{align*}
\text{lemma} & \quad \text{NRD-right-unit}: P \text{ is NSRD} \implies P ; \quad \text{II}_R = P \\
& \quad \text{by (metis Healthy-if NRD-is-RD-RD-def)}
\end{align*} \]

\textbf{lemma \textit{NRD-composition-wp}}:

\[ \quad \text{assumes} \quad P \text{ is NSRD} \quad \text{Q is SRD} \\
\quad \text{shows} \quad P ; \quad Q = \quad R \\
\quad \text{by (simp add: NRD-is-SRD SRD-reactive-tri-design \textbf{assms(1)} \textbf{assms(2)} \textbf{assms(3)})} \]

\textbf{lemma \textit{pre-R-NRD-seq-lemma}}:

\[ \quad \text{assumes} \quad P \text{ is NRD} \quad Q \text{ is SRD} \\
\quad \text{shows} \quad R1 (R2c (post\_R P ; \quad (\neg, \text{pre}_R Q))) = post\_R P ; \quad (\neg, \text{pre}_R Q) \\
\quad \text{proof} \quad - \\
\quad \quad \text{have} \quad post\_R P ; \quad (\neg, \text{pre}_R Q) = R1(R2c(post\_R P)) ; \quad R1(R2c(\neg, \text{pre}_R Q)) \\
\quad \quad \text{by (simp add: NRD-is-SRD R1-R2-post-RHS R1-reap-not R2c-preR R2c-reap-not \textbf{assms(1)} \textbf{assms(2)})} \]

\textbf{lemma \textit{per-R-NRD-seq |rdes|}}:

\[ \quad \text{assumes} \quad P \text{ is NRD} \quad Q \text{ is SRD} \\
\quad \text{shows} \quad \text{pre}_R(P ; \quad Q) = (\text{pre}_R P \quad \text{post}_R P \quad \text{wp}, \quad \text{pre}_R Q) \\
\quad \text{by (simp add: NRD-composition-wp \textbf{assms reap-pre-RHS-design usubst unrest wp-rea-def R2c-disj R1-disj R2c-and R2c-preR R1-R2c-commute[THEM sym] R1-extend-conj' R1-idem R2c-not closure})} \]

\[ \quad \text{(metis \textbf{no-types, lifting}) Healthy-def Healthy-if NRD-is-SRD R1-R2c-commute R1-R2c-seq-end \textbf{assms(1)} \textbf{assms(2)} post\_R-R2c-closed post\_R-R2c-end} \]

\textbf{lemma \textit{post-R-NRD-seq |rdes|}}:

\[ \quad \text{assumes} \quad P \text{ is NRD} \quad Q \text{ is SRD} \\
\quad \text{shows} \quad \text{post}_R(P ; \quad Q) = (\text{pre}_R P \quad \text{post}_R P \quad \text{wp}, \quad \text{pre}_R Q) \implies (\text{post}_R P \quad (\text{pre}_R Q)) \\
\quad \text{by (simp add: NRD-composition-wp \textbf{assms reap-peri-RHS-design usubst unrest wp-rea-def R1-extend-conj' R1-disj R1-R2c-seq-end R2c-disj R2c-and R2c-reap-impl R1-reap-impl')} \]

\textbf{lemma \textit{peri-R-NRD-seq |rdes|}}:

\[ \quad \text{assumes} \quad P \text{ is NRD} \quad Q \text{ is SRD} \\
\quad \text{shows} \quad \text{peri}_R(P ; \quad Q) = ((\text{pre}_R P \quad \text{post}_R P \quad \text{wp}, \quad \text{pre}_R Q) \implies (\text{peri}_R P \quad (\text{pre}_R P \quad (\text{peri}_R Q))) \\
\quad \text{by (simp add: NRD-composition-wp \textbf{assms reap-peri-RHS-design usubst unrest wp-rea-def R1-extend-conj' R1-disj R1-R2c-seq-end R2c-disj R2c-and R2c-reap-impl R1-reap-impl')} \]

\textbf{lemma \textit{peri-R-NRD-seq |rdes|}}:

\[ \quad \text{assumes} \quad P \text{ is NRD} \quad Q \text{ is SRD} \\
\quad \text{shows} \quad \text{peri}_R(P ; \quad Q) = (\text{peri}_R P \quad \text{peri}_R P \quad \text{peri}_R Q) \implies (\text{peri}_R P \quad (\text{peri}_R Q)) \\
\quad \text{by (simp add: NRD-composition-wp \textbf{assms reap-peri-RHS-design usubst unrest wp-rea-def R1-extend-conj' R1-disj R1-R2c-seq-end R2c-disj R2c-and R2c-reap-impl R1-reap-impl')} \]

\textbf{lemma \textit{post-R-NRD-seq |rdes|}}:

\[ \quad \text{assumes} \quad P \text{ is NRD} \quad Q \text{ is SRD} \\
\quad \text{shows} \quad \text{post}_R(P ; \quad Q) = ((\text{pre}_R P \quad \text{post}_R P \quad \text{wp}, \quad \text{pre}_R Q) \implies (\text{post}_R P \quad (\text{post}_R Q))) \\
\quad \text{by (simp add: NRD-composition-wp \textbf{assms reap-peri-RHS-design usubst unrest wp-rea-def R1-extend-conj' R1-disj R1-R2c-seq-end R2c-disj R2c-and R2c-reap-impl R1-reap-impl')} \]

\textbf{lemma \textit{peri-R-NRD-seq |rdes|}}:

\[ \quad \text{assumes} \quad P \text{ is NRD} \quad Q \text{ is SRD} \\
\quad \text{shows} \quad \text{peri}_R(P ; \quad Q) = ((\text{peri}_R P \quad \text{peri}_R P \quad \text{peri}_R Q) \implies (\text{peri}_R P \quad (\text{peri}_R Q)) \\
\quad \text{by (simp add: NRD-composition-wp \textbf{assms reap-peri-RHS-design usubst unrest wp-rea-def R1-extend-conj' R1-disj R1-R2c-seq-end R2c-disj R2c-and R2c-reap-impl R1-reap-impl')} \]

\textbf{lemma \textit{post-R-NRD-seq |rdes|}}:

\[ \quad \text{assumes} \quad P \text{ is NRD} \quad Q \text{ is SRD} \\
\quad \text{shows} \quad \text{post}_R(P ; \quad Q) = ((\text{pre}_R P \quad \text{pre}_R P \quad \text{pre}_R Q) \implies (\text{post}_R P \quad (\text{post}_R Q))) \\
\quad \text{by (simp add: NRD-composition-wp \textbf{assms reap-peri-RHS-design usubst unrest wp-rea-def R1-extend-conj' R1-disj R1-R2c-seq-end R2c-disj R2c-and R2c-reap-impl R1-reap-impl')} \]
by (simp add: NSRD-composition-up assms closure rea-post-RHS-design usubst unrest wp-rea-def
R1-extend-conj' R1-disj R1-R2c-seqr-distribute R2c-disj R2c-and R2c-rea-impl R1-rea-impl'
R2c-preR R2c-periR R1-rea-not' R2c-rea-not)

lemma NSRD-seq-r-closure [closure]:
assumes P is NSRD Q is NSRD
shows (P ;; Q) is NSRD
proof –
  have (∀r. postR P wp r presR Q) ;: true_r = (∀r. postR P wp r presR Q)
  by (simp add: wp-rea-def rpred assms seqr-assoc NSRD-neg-pre-unit)
moreover have $st' ;: presR P ∧ postR P wp r presR Q ⇒ periR P ∨ postR P #: periR Q
  by (simp add: unrest assms wp-rea-def)
ultimately show ?thesis
  by (rule-tac NSRD-intro, simp-all add: seqr-or-distl NSRD-neg-pre-unit assms closure rdes unrest)
qed

lemma RHS-tri-normal-design-composition:
assumes $ok' ;: Q, $ok' ;: Q_2, $ok' ;: Q S $ok ;: S_1 $ok ;: S_2
$wait ;: R $wait' ;: Q_2 $wait ;: S_1 $wait ;: S_2
P is R2c Q_1 is R1 Q_1 is R2c Q_2 is R1 Q_2 is R2c
R is R2c S_1 is R1 S_1 is R2c S_2 is R1 S_2 is R2c
R1 (¬ P) :: R1 (true) = R1 (¬ P) $st' ;: Q_1
shows $R_4(P ⊢ Q_1 ∨ Q_2) ;; $R_4(R ⊢ S_1 ∨ S_2) =
$R_4((P ∧ Q_2 wp r, R) ⊢ (Q_1 ∨ (Q_2 ;; S_1)) ∨ (Q_2 ;; S_2))
proof –
  have $R_4(P ⊢ Q_1 ∨ Q_2) ;; $R_4(R ⊢ S_1 ∨ S_2) =
      $R_4(((R1 (¬ P) wp r, false ∧ Q_2 wp r, R) ⊢ ((∃ $st' · Q_1) ∨ (Q_2 ;; S_1)) ∨ (Q_2 ;; S_2))
  by (simp-all add: RHS-tri-design-composition-up rea-not-def assms unrest)
also have … = $R_4((P ∧ Q_2 wp r, R) ⊢ (Q_1 ∨ (Q_2 ;; S_1)) ∨ (Q_2 ;; S_2))
  by (simp add: assms wp-rea-def ex-unrest, rel-auto)
finally show ?thesis .
qed

lemma RHS-tri-normal-design-composition' [rdes-def]:
assumes P is RC Q_1 is RR $st' ;: Q_2 is RR R is RR S_1 is RR S_2 is RR
shows $R_4(P ⊢ Q_1 ∨ Q_2) ;; $R_4(R ⊢ S_1 ∨ S_2) =
$R_4((P ∧ Q_2 wp r, R) ⊢ (Q_1 ∨ (Q_2 ;; S_1)) ∨ (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-def rea-not-def)
thus ?thesis
  by (simp add: RHS-tri-normal-design-composition assms closure unrest RR-implies-R2c)
qed

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

lemma NSRD-seq-post-false:
assumes P is NSRD Q is SRD postR(P) = false
shows P ;; Q = P
apply (simp add: NSRD-composition-up assms wp rpred closure)
using NSRD-is-SRD SRD-reactive-tri-design assms(1,3) apply fastforce
done

54
lemma NSRD-srd-skip [closure]: \( \Pi_R \) 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 = \( R_s (\text{pre} R P \vdash \text{peri} R P \odot \text{false}) \)
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 \( P \subseteq P ;; \) Miracle
proof
  have \( R_s (\text{pre} R P \vdash \text{peri} R P \odot \text{post} R P) \subseteq R_s (\text{pre} R P \vdash \text{peri} R P \odot \text{false}) \)
    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 \( \Rightarrow \) \( P \) \( ^\text{Suc} \) \( n \) is NSRD
proof (induct \( n \))
  case 0
  then show \(?case \)
    by (simp)
next
  case (Suc \( n \))
  then show \(?case \)
    by (simp add: NSRD-seqr-closure upred-semiring.power-Suc)
qed

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

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

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

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

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

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

declare upred-semiring.power-Suc [simp]

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

by (simp add: closure assms rdes)
also have \( \vdash (pre_R (P \cdot (Suc n + 1)) \Rightarrow_r (\forall i \in \{0..n\}. \text{post}_R P \land \text{post}_R P \cdot i) ;; \text{peri}_R P)) \)
  by (simp only: hyp)
also have \( \vdash (\forall P \cdot i \in \{0..n\}. \text{post}_R P \Rightarrow_r \text{peri}_R P \lor \text{post}_R P \lor \text{pre}_R (P \cdot P \cdot n) \Rightarrow_r (\forall i \in \{0..n\}. \text{post}_R P \cdot i) ;; \text{peri}_R P)) \)
  by (simp add: NSRD-is-NSRD R1-Continuous R1-power Sup-Continuous-closed assms post-RSRD-R1)
  hence \( 1 : ((\forall i \in \{0..n\}. \text{post}_R P \cdot i) ;; \text{peri}_R P) \Rightarrow R1 \)
  by (simp add: closure assms)
  hence \( (\forall P \cdot i \in \{0..n\}. \text{post}_R P \Rightarrow_r (\forall i \in \{0..n\}. \text{post}_R P \cdot i) ;; \text{peri}_R P) \Rightarrow R1 \)
  by (simp add: closure)
  hence \( (\forall P \cdot \text{post}_R P \cdot P \cdot n) \Rightarrow_r (\forall i \in \{0..n\}. \text{post}_R P \cdot i) ;; \text{peri}_R P) \Rightarrow R1 \)
  by (simp add: Healthy-if R1-post-SRD assms closure)
  thus \( \text{thesis} \)
  by (simp only: wp-rea-impl-lemma, simp add: Healthy-if 1, simp add: R1-post-SRD assms closure)
qed
also have \( \vdash (\forall P \cdot \text{post}_R P \cdot P \cdot n) \Rightarrow_r (\forall i \in \{0..n\}. \text{post}_R P \cdot i) ;; \text{peri}_R P) \)
  by (pred-auto)
also have \( \vdash (\forall P \cdot \text{post}_R P \cdot P \cdot n) \Rightarrow_r (\forall i \in \{0..n\}. \text{post}_R P \cdot (Suc i)) ;; \text{peri}_R P) \)
  by (simp add: seq-Sup-distl seqr-assoc[THEN sym])
also have \( \vdash (\forall P \cdot \text{post}_R P \cdot P \cdot n) \Rightarrow_r \text{peri}_R P \lor ((\forall i \in \{1..Suc n\}. \text{post}_R P \cdot i) ;; \text{peri}_R P) \)
  proof --
    have \( (\forall i \in \{0..n\}. post_R P \cdot Suc i) = (\forall i \in \{1..Suc n\}. post_R P \cdot i) \)
    apply (rule cong[of Suc], auto)
    apply (metis atLeast0AtMost atMost-iff image-Suc-atLeastAtMost rev-image-eqI upred-semiring.power-Suc)
    using Suc-le-D apply fastforce
done
  thus \( \text{thesis} \) by simp
qed
also have \( \vdash (\forall P \cdot \text{post}_R P \cdot P \cdot n) \Rightarrow_r ((\forall i \in \{0..Suc n\}. post_R P \cdot i) ;; \text{peri}_R P) \)
  by (simp add: SUP-atLeastAtMost-first winf-or seq-or-distl seq-or-distr)
also have \( \vdash (\forall P \cdot (Suc (Suc n))) \Rightarrow_r ((\forall i \in \{0..Suc n\}. post_R P \cdot i) ;; \text{peri}_R P) \)
  by (simp add: rdes closure assms)
finally show \( \text{?case} \) by (simp)
qed

lemma periR-power' [rdes]:
assumes $P$ is NSRD
shows $\text{peri}_R (P' \text{Suc} n) = (\text{pre}_R (P' \text{Suc} n)) \Rightarrow_r (\prod i \in \{0..n\} \cdot \text{post}_R (P') \supset i) ;; \text{peri}_R (P))$
by (simp add: perir-power assms UINF-as-Sup[THEN sym])

lemma perir-power-Suc [rdes]:
assumes $P$ is NSRD
shows $\text{peri}_R (P' \text{Suc} n) = (\text{pre}_R (P' \text{Suc} n)) \Rightarrow_r \text{post}_R (P') \supset \text{Suc} n$
proof (induct n)
  case 0
  then show $\text{case}$
  by (simp add: NSRD-is-SRD NSRD-wait-unrest-pre SRD-post-under-pre assms)
next
  case (Suc n) note hyp = this
  have $\text{post}_R (P' \text{Suc} (n+1)) = \text{post}_R (P' (n+1))$
  by (simp)
  also have ... = $(\text{pre}_R (P' \text{Suc} (n+1))) \Rightarrow_r (\text{post}_R (P' \text{Suc} (n+1)))$
  by (simp add: closure assms rdes)
  also have ... = $(\text{pre}_R (P' \text{Suc} (n+1))) \Rightarrow_r (\text{post}_R (P' \text{Suc} n))$
  by (simp only: hyp)
  also have ... = $(\text{pre}_R (P' \text{Suc} n)) \Rightarrow_r \text{post}_R (P' \text{Suc} n))$
  by (metis (no-types, lifting) Healthy-if NSRD-is-SRD NSRD-power-Suc R1-power assms hyp postR-SRD-R1 upred-semiring.power-Suc wp-rea-impl-lemma)
  also have ... = $(\text{pre}_R (P' \text{Suc} n)) \Rightarrow_r \text{post}_R (P' \text{Suc} n))$
  by (pred-auto)
  also have ... = $(\text{pre}_R (P' (\text{Suc} n))) \Rightarrow_r \text{post}_R (P' (\text{Suc} n))$
  by (simp add: rdes closure assms)
finally show $\text{case}$ by (simp)
qed

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

lemma power-rdes-def [rdes-def]:
assumes $P$ is RC $Q$ is RR $R$ is RR $\$st' $ R Q$
shows $(R_{\text{rdes}} (P' R Q)) \Rightarrow_r (\text{Suc} n)$
proof (induct n)
  case 0
  then show $\text{case}$
  by (simp add: wp assms closure)
next
  case (Suc n)

58
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 \(\ldots = (P \land (\bigcup i \in \{1..Suc \; n\} \cdot (R \; ^\cdot \; i \; wp_r \; P)))\)
by (metis (no-types, lifting) INF-cong One-nat-def image-Suc-atLeastAtMost image-image)
also have \(\ldots = (\bigcup i \in insert \; 0 \{1..Suc \; n\} \cdot (R \; ^\cdot \; i \; wp_r \; P))\)
by (simp add: wp assms closure conj-upred-def)
also have \(\ldots = (\bigcup i \in \{0..Suc \; n\} \cdot (R \; ^\cdot \; i \; wp_r \; P))\)
by (simp add: atLeastAtMost-insertL)
finally show \(?thesis\)
by (simp add: USUP-as-Inf-collect)
qed

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

proof –

have \(\text{lhs} = (Q \lor (\bigcap i \in \{0..n\} \cdot R \; ^\cdot \; Suc \; i) \;; \; Q)\)
by (simp add: seqr-assoc THEN sym seq-UNF-distl)
also have \(\ldots = (Q \lor (\bigcap i \in \{0..n\} \cdot R \; ^\cdot \; Suc \; i) \;; \; Q)\)
by (simp only: UNF-as-Sup-collect)
also have \(\ldots = (Q \lor (\bigcap i \in \{1..Suc \; n\} \cdot R \; ^\cdot \; i) \;; \; Q)\)
by (metis One-nat-def image-Suc-atLeastAtMost image-image)
also have \(\ldots = (\bigcap i \in insert \; 0 \{1..Suc \; n\} \cdot R \; ^\cdot \; i) \;; \; Q\)
by (simp add: disj-upred-def THEN sym seqr-or-distl)
also have \(\ldots = (\bigcap i \in \{0..Suc \; n\} \cdot R \; ^\cdot \; i) \;; \; Q\)
by (simp add: atLeastAtMost-insertL)
finally show \(?thesis\)
by (simp add: UNINF-as-Inf-collect)
qed

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

proof –

have \((\bigcap i \in \{0..n\} \cdot R \; ^\cdot \; i) \;; \; Q = (\bigcap i \in \{0..n\} \cdot R \; ^\cdot \; i) \;; \; Q\)
by (simp add: UNINF-as-Inf-collect)
also have \(\ldots = (\bigcap i \in insert \; 0 \{1..n\} \cdot R \; ^\cdot \; i) \;; \; Q\)
by (simp add: atLeastAtMost-insertL)
also have \(\ldots = (Q \lor (\bigcap i \in \{1..n\} \cdot R \; ^\cdot \; i) \;; \; Q)\)
by (metis (no-types, lifting) SUP-insert disj-upred-def seqr-left-unit seqr-or-distl upred-semiring.power-0)
also have \(\ldots = (Q \lor (\bigcap i \in \{0..<n\} \cdot R \; ^\cdot \; Suc \; i) \;; \; Q)\)
by (metis One-nat-def atLeastLessThanSuc-atLeastAtMost image-Suc-atLeastLessThan image-image)
also have \(\ldots = (Q \lor (\bigcap i \in \{0..<n\} \cdot R \; ^\cdot \; Suc \; i) \;; \; Q)\)
by (simp add: UNINF-as-Inf-collect)
also have \(\ldots \; \text{is} \; RR\)
by (simp-all add: closure assms)
finally show \(?thesis\)
qed

from 1 2 3 Suc show \(?case\)
by (simp add: Suc RHS-tri-normal-design-composition' closure assms wp)

qed
interpretation

theorem uplus-rdes-def [rdes-def];
assumes P is RC Q is RR R is RR $st' \not\in Q$
shows (R, (P ⊢ Q ⊖ R))^+ = R_o (R^* wpr P ⊦ R^* ;; Q ⊖ R^+)
proof –
  have 1:(i ∙ R ^ i) ;; Q = R^* ;; Q
  by (metis (no-types) RA1 assms(2) rea-skip-unit(2) rrel-thy,Star-def ustar-def)
show ?thesis
  by (simp add: uplus-power-def seq-UINF-distr wp closure assms rdes-def)
qed

5.1 UTP theory

typedef NSRDES

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

overloading
nsrdes-hcond == upthcond :: (NSRDES, ('s,'t::trace,'a) rsp) uthy ⇒ ( ('s,'t,'a) rsp × ('s,'t,'a) rsp)
health
nsrdes-unit == upth-unit :: (NSRDES, ('s,'t::trace,'a) rsp) uthy ⇒ ('s,'t,'a) hrel-rsp
begin
  definition nsrdes-hcond :: (NSRDES, ('s,'t::trace,'a) rsp) uthy ⇒ ( ('s,'t,'a) rsp × ('s,'t,'a) rsp)
  health where
    [upred-defs]: nsrdes-hcond T = NSRD
  definition nsrdes-unit :: (NSRDES, ('s,'t::trace,'a) rsp) uthy ⇒ ( 's,'l,'a) hrel-rsp where
    [upred-defs]: nsrdes-unit T = II_R
end

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

interpretation nsrd-thy: utp-theory-kleene UTHY(NSRDES, ('s,'t::trace,'a) rsp)
rewrites \ A P. P ∈ carrier (uthy-order NSRDES) ⇔ P is NSRD
and P is H_{NSRDES} ⇔ P is NSRD
and (μ X · F (H_{NSRDES} X)) = (μ X · F (NSRD X))
and carrier (uthy-order NSRDES) → carrier (uthy-order NSRDES) ≡ [NSRD]_H → [NSRD]_H
and [H_{NSRDES}]_H → [H_{NSRDES}]_H ≡ [NSRD]_H → [NSRD]_H
and 1_{NSRDES} = Miracle
and 1_{NSRDES} = II_R
and le (uthy-order NSRDES) = op ⊆
proof –
interpret lat: utp-theory-continuous UTHY(NSRDES, ('s,'t,'a) rsp)
  by (unfold-locales, simp-all add: nsrdes-hcond-def nsrdes-unit-def closure Healthy-if)
show 1: T_{NSRDES} = (Miracle :: ('s,'t,'a) 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,'a) rsp)
  by (unfold-locales, simp-all add: nsrdes-hcond-def nsrdes-unit-def closure Healthy-if Miracle-left-zero
      SRD-left-unit NSRD-right-unit)
qed (simp-all add: nsrdes-hcond-def nsrdes-unit-def closure Healthy-if)

declare nsrd-thy.top-healthy [simp del]
We also show how to calculate the Kleene closure of a reactive design.

**6 Syntax for reactive design contracts**

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

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

definition mk-RD :: 's upred $\Rightarrow$ ('t::trace $\Rightarrow$ 's upred) $\Rightarrow$ ('t $\Rightarrow$ 's hrel-rsp) $\Rightarrow$ ('s, 't, 'a) hrel-rsp where
mk-RD P Q R = R.s((P \vdash Q \circ R)) \ast R.s((R^w_{wp}, P) \vdash R^r ;; Q \circ R^r)

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

syntax
    - trace-var :: logic
    - mk-RD :: logic $\Rightarrow$ logic $\Rightarrow$ logic $\Rightarrow$ logic ([/- $\vdash$ -]R)
    - trace-pred :: logic $\Rightarrow$ logic ([/-]R)

parse-translation ⟨⟨
let
  fun trace-var-tr [] = Syntax.free trace
   | trace-var-tr - = raise Match;
in
[(@{syntax-const -trace-var}, K trace-var-tr)]
end ⟩

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

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

lemma preR-mk-RD [rdes]: \( \text{pre}_R([P \vdash Q(\text{trace})] | R(\text{trace})]_R) = R_1([P]_{S<}) \)
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]:
\$ st' \# [P trace]t
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]: peri_R([P \vdash Q(\text{trace})] | R(\text{trace})]_R) = ([P]_{S<} \Rightarrow R_1(([Q(\text{trace})]_{S<})[\text{trace} \rightarrow \& tt]))
by (simp add: mk-RD-def rea-peri-RHS-design usubst unrest R2c-not R2c-lift-state-pre R2c-disj R2c-msubst-tt R1-disj R2c-rea-impl R1-rea-impl)

lemma postR-mk-RD [rdes]: post_R([P \vdash Q(\text{trace})] | R(\text{trace})]_R) = ([P]_{S<} \Rightarrow R_1(([R(\text{trace})]_S)[\text{trace} \rightarrow \& tt]))
by (simp add: mk-RD-def rea-post-RHS-design usubst 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 \( '(P_1)_{S<} \Rightarrow \text{pre}_R Q' \)
\( '(P_1)_{S<} \land \text{peri}_R Q = [P_2 x]_{S<} [x \rightarrow \& tt]' \)
\( '(P_1)_{S<} \land \text{post}_R Q = [P_3 x]_{S<} [x \rightarrow \& tt]' \)
shows \([P_1 \vdash P_2(\text{trace}) | P_3(\text{trace})]_R \subseteq Q\)
proof –
have \([P_1 \vdash P_2(\text{trace}) | P_3(\text{trace})]_R \subseteq R_0(\text{pre}_R(Q) \vdash \text{peri}_R(Q) \circ \text{post}_R(Q))\)
using assms
by (simp add: mk-RD-def, rule-tac srdes-tri-refine-intro, simp-all)
thus \?thesis
by (simp add: SRD-reactive-tri-design assms(1))
qed

end

7 Reactive design tactics

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

Theorems for normalisation

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

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

Tactic to split a refinement conjecture into three POs

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

Tactic to split an equality conjecture into three POs

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

Tactic to prove a refinement

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

Tactics to prove an equality

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

Via antisymmetry

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

Tactic to calculate pre/per/peri/postconditions from reactive designs

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

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

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

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

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

end
8 Reactive design parallel-by-merge

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

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

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

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

definition skip-\(\text{rm}\) :: \((s::\text{trace},'a)\) \text{rsp merge} \((II_{RM})\) \text{where}
  \begin{align*}
  \text{upred-defs:} \quad II_{RM} &= ((\exists \ st < \cdot \ \text{skip_m} \lor (\neg \ \text{so}_{<} < \land \ \text{tr}_{<} \leq u \ \text{tr}'))
  \end{align*}

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

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

lemma R3h-par-by-merge: \[\text{closure}]:
  assumes \(P\) is R3h \(Q\) is R3h \(M\) is R3hm
  shows \((P \parallel_M Q)\) is R3h
proof -
  have \((P \parallel_M Q) = ((P \parallel_M Q)[\text{true}/\text{so}_k] \land \text{so}_k \triangleright (P \parallel_M Q)[\text{false}/\text{so}_k][\text{true}/\text{wait}] < \text{wait} \triangleright (P \parallel_M Q))\)
    by (simp add: cond-var-subst-left cond-var-subst-right)
  also have ... = \(((P \parallel_M Q)[\text{true}/\text{true}/\text{so}_k,\text{wait}] \land \text{so}_k \triangleright (P \parallel_M Q)[\text{false},\text{true}/\text{so}_k,\text{wait}]) < \text{wait} \triangleright (P \parallel_M Q))\)
    by (rel-auto)
  also have ... = \(((\exists \ st \cdot II)[\text{true}/\text{true}/\text{so}_k,\text{wait}] \land \text{so}_k \triangleright (P \parallel_M Q)[\text{false},\text{true}/\text{so}_k,\text{wait}]) < \text{wait} \triangleright (P \parallel_M Q))\)
proof -
  have \((P \parallel_M Q)[\text{true},\text{true}/\text{so}_k,\text{wait}] = ((P \parallel\parallel_M Q) \land [Q]_1 \land \text{so}_k < = u \ \text{so}_k) \land \text{so}_k \triangleright (P \parallel_M Q)[\text{true}/\text{so}_k,\text{wait}])\)
    by (simp add: par-by-merge-def U0-as-alpha U1-as-alpha assms Healthy-if)
  also have ... = \(((P \parallel\parallel_M Q) \land [Q]_1 \land \text{so}_k < = u \ \text{so}_k) \land \exists \ st < \cdot \ \text{so}_k < = u \ \text{so}_k) < \text{true},\text{true}/\text{so}_k,\text{wait}])\)
    by (simp add: assms Healthy-if)
  also have ... = \(((R3h(P))_0 \land [R3h(Q)]_1 \land \text{so}_k < = u \ \text{so}_k) < \text{true},\text{true},\text{so}_k,\text{wait}])\)
    by (simp add: assms Healthy-if)
  also have ... = ((R3h(P))_0 \land [R3h(Q)]_1 \land \text{so}_k < = u \ \text{so}_k) < \text{true},\text{true},\text{so}_k,\text{wait}])\)
    by (simp add: assms Healthy-if)
  finally show \(\text{thesis}\) by (simp add: closure assms unrest)
qed

also have ... = \(((\exists \ st \cdot II)[\text{true},\text{true}/\text{so}_k,\text{wait}] < \text{so}_k \triangleright (R1(\text{true}))[\text{false},\text{true}/\text{so}_k,\text{wait}]) < \text{wait} \triangleright (P \parallel_M Q))\)
proof -
  have \((P \parallel_M Q)[\text{false},\text{true}/\text{so}_k,\text{wait}] = ((P \parallel\parallel_M Q) \land [Q]_1 \land \text{so}_k < = u \ \text{so}_k) \land \text{so}_k \triangleright (P \parallel_M Q)[\text{false},\text{true}/\text{so}_k,\text{wait}])\)
    by (simp add: par-by-merge-def U0-as-alpha U1-as-alpha assms Healthy-if)
  also have ... = \(((P \parallel\parallel_M Q) \land [Q]_1 \land \text{so}_k < = u \ \text{so}_k) \land \text{so}_k < = u \ \text{so}_k) \land \text{so}_k \triangleright (P \parallel_M Q)[\text{false},\text{true}/\text{so}_k,\text{wait}])\)
    by (simp add: assms Healthy-if)
  also have ... = \(((R3h(P))_0 \land [R3h(Q)]_1 \land \text{so}_k < = u \ \text{so}_k) \land \text{so}_k < = u \ \text{so}_k) < \text{false},\text{true}/\text{so}_k,\text{wait}])\)
    by (simp add: assms Healthy-if)
also have ... = \( (R1(true))[false, true/ok, \overline{\text{wait}}] \)
  by (rel-blast)
finally show \( \text{thesis} \) by simp
qed

also have ... = (\( (\exists \, \text{st} \cdot II) < ok \triangleright R1(true) \) \( \triangleleft \text{wait} \triangleright (P \parallel M Q) \))
  by (rel-auto)
also have ... = R3h(P \parallel M Q)
  by (simp add: R3h-cases)
finally show \( \text{thesis} \)
  by (simp add: Healthy-def)
qed

definition [upred-defs]: \( RD1m(M) = (M \lor \neg sok < \land \neg tr < \leq u \neg tr' ) \)

lemma RD1-par-by-merge [closure]:
assumes P is R1 Q is R1 M is RD1 P is RD1 Q is RD1 M is RD1m
shows (P \parallel M Q) is RD1
proof –
  have 1: (RD1(R1(P)) \parallel RD1m(R1m(M)) \parallel RD1(R1(Q)))[false/ok] = R1(true)
    by (rel-blast)
  have (P \parallel M Q) = (P \parallel M Q)[true/ok] < ok \triangleright (P \parallel M Q)[false/ok]
    by (simp add: cond-var-split)
also have ... = R1(P \parallel M Q) < ok \triangleright R1(true)
    by (metis 1 Healthy-if R1-par-by-merge assms calculation
         cond-idem cond-var-subst-right in-var-uvar ok-vwb-lens)
also have ... = RD1(P \parallel M Q)
    by (simp add: Healthy-if R1-par-by-merge RD1-alt-def assms(3))
finally show \( \text{thesis} \)
    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 ... = ((P \parallel Q) :: (M :: J))
    by (simp add: Healthy-def RD2-def H2-def)
  also from assms have ... = ((P \parallel Q) :: M :: J)
    by (simp add: seqr-assoc)
  also from assms have ... = RD2(P \parallel M Q)
    by (simp add: RD2-def H2-def par-by-merge-def)
finally show \( \text{thesis} \)
    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]: \( N_0(M) = (\exists \neg wait' =_u (\neg \text{wait} \lor \text{wait}) \land \neg tr < \leq u \neg tr' \land (\exists \neg \text{ok}; \neg \text{ok}; \neg \text{ok}; \neg \text{wait}; \text{wait}; \text{wait}; \text{wait}; \text{wait}; \text{wait} \cdot M) ) \)

65
**definition nmerge-rd1** \((N_1)\) where  
\(\text{[upred-defs]: } N_1(M) = (\$\text{ok}' = u \, (\$\text{0} \land \$\text{1} \land N_0(M))\)

**definition nmerge-rd** \((N_R)\) where  
\(\text{[upred-defs]: } N_R(M) = (\exists \, s_t < \cdot v' = u \, v <) \circ (\$\text{wait} \circ N_1(M)) \circ (\$\text{ok} \circ (\$\text{tr} \leq u \, \$\text{tr}'))\)

**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** \((-\parallel_R -\parallel\) \([85,0,86]\) \([85]\) where  
\(P \parallel_R M \equiv P \parallel M_R(M)\)

**Healthiness condition for reactive design merge predicates**

**definition** \(\text{[upred-defs]: } RDM(M) = R2m(\exists \, \$\text{0} \land \$\text{1} \land \$\text{ok} < \$\text{ok} < \$\text{ok}' < \$\text{0} < \$\text{1} < \$\text{wait} < \$\text{wait} < \$\text{wait}' \cdot M)\)

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

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

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

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

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

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

**lemma par-rdes-NSRD** [closure]:  
\(\text{assumes } P \text{ is SRD} \quad Q \text{ is SRD} \quad M \text{ is RDM} \quad \text{shows } P \parallel_M Q \text{ is NSRD} \quad \text{proof } \)

\(\text{have } (P \parallel_M Q :: H_R) \text{ is NSRD by } \text{(rule NSRD-intro', simp-all add: SRD-healths closure assms)}\)

\(\text{(metis (no-types, lifting) Healthy-def R2-par-by-merge R2-seqr-closure R2m-nmerge-rd RDM-def SRD-healths(2) assms skip-srea-R2 \,metis Healthy-Idempotent RD3-Idempotent RD3-def)}\)

\(\text{thus } ?\text{thesis by } \text{(simp add: merge-rd-def par-by-merge-def seqr-assoc)}\)

\(\text{qed}\)
lemma \( RDM\text{-intro} \):
assumes \( M \text{ is } R2m \), \( \$0\text{-ok} \cong M \), \( \$1\text{-ok} \cong M \), \( \$0\text{-wait} \cong M \), \( \$1\text{-wait} \cong M \), \( \$\text{ ok} < \cong M \), \( \$\text{ wait} < \cong M \)
shows \( M \text{ is } RDM \)
using assms
by (simp add: Healthy-def RDM-def ex-unrest unrest)

lemma \( RDM\text{-unrests} \) \( [\text{unrest}] \):
assumes \( M \text{ is } RDM \)
shows \( \$0\text{-ok} \cong M \), \( \$1\text{-ok} \cong M \), \( \$\text{ ok} < \cong M \), \( \$\text{ wait} < \cong M \)
by (subst Healthy-if \([OF \text{ assms, THEN sym}], \text{ simp-all add: } RDM\text{-def unrest, rel-auto} \))

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

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

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

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

lemma \( \text{ parallel-ok-cases} \):
\((\parallel x, Q) : M \) \( = \) 
\((\parallel x, Q) : (M[\text{true,true }/\$0\text{-ok},\$1\text{-ok}]) \lor 
\((\parallel x, Q) : (M[\text{false,true }/\$0\text{-ok},\$1\text{-ok}]) \lor 
\((\parallel x, Q) : (M[\text{true,false }/\$0\text{-ok},\$1\text{-ok}]) \lor 
\((\parallel x, Q) : (M[\text{false,false }/\$0\text{-ok},\$1\text{-ok}]) \lor 
proof –
  have \((\parallel x, Q) : M) = (\exists ok \cdot (P \parallel Q)[<ok>/\$0\text{-ok}] \lor M[<ok>/\$0\text{-ok}]) \)
  by (subst segm-middle[of left-uvar ok], simp-all)
  also have \( = (\exists ok \cdot \exists ok \cdot \exists Q \cdot (P \parallel Q)[<ok>/\$0\text{-ok}] [<ok>/\$1\text{-ok}] \lor M[<ok>/\$0\text{-ok}] [<ok>/\$1\text{-ok}] \]) \)
  by (subst segm-middle[of right-uvar ok], simp-all)
  also have \( = (\exists ok \cdot \exists Q \cdot (P[<ok>/\$0\text{-ok}] \parallel Q[<ok>/\$0\text{-ok}] \lor M[<ok>/\$0\text{-ok}] [<ok>/\$1\text{-ok}] \)]) \)
  by (rel-auto robust)
also have \( = (\exists Q \cdot (P[<ok>/\$0\text{-ok}] \parallel Q[<ok>/\$0\text{-ok}] \lor M[<ok>/\$0\text{-ok}] [<ok>/\$1\text{-ok}] \)]) \)
by (simp add: true-alt-def THEN sym false-alt-def THEN sym disj-assoc)
lemma skip-srev-thesis [usubst]:
\[ H_{R^f} = R1(\neg \$ok) \]
by (rel-auto)

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

lemma parallel-assm-lemma:
assumes \( P \) is RD2
shows \( \text{pre}_s \uparrow (P \parallel M_{R(M)} \cdot Q) = (((\text{pre}_s \uparrow P) \parallel M_{n(M)} :: R1(\text{true}) (\text{cmt}_s \uparrow Q)) \)
\[ \lor ((\text{cmt}_s \uparrow P) \parallel M_{n(M)} :: R1(\text{true}) (\text{pre}_s \uparrow Q)) \]
proof –
\begin{itemize}
\item have \( \text{pre}_s \uparrow (P \parallel M_{R(M)} \cdot Q) = \text{pre}_s \uparrow ((P \parallel Q :: M_{R(M)}) \)
  \begin{itemize}
  \item by (simp add: par-by-merge-def)
  \end{itemize}
\item also have \( ... = (((P \parallel Q)[true/false/$\text{ok}$/wait] :: N_R \cdot M :: R1(\neg \$ok)) \)
  \begin{itemize}
  \item by (simp add: merge-rd-def usubst, rel-auto)
  \end{itemize}
\item also have \( ... = (((P[true/false/$\text{ok}$/wait] :: Q[true/false/$\text{ok}$/wait]) :: N_1(M) :: R1(\neg \$ok)) \)
  \begin{itemize}
  \item by (rel-auto robust, (metis)+)
  \end{itemize}
\item also have \(... = (((P[true/false/$\text{ok}$/wait]f :: Q[true/false/$\text{ok}$/wait])f :: (N_1 M)[true,true/$\text{false}$/\$ok$/\$1-ok] \)
  \begin{itemize}
  \item \( R1(\neg \$ok)) \lor \)
  \end{itemize}
\item also have \(... = (((P[true/false/$\text{ok}$/wait]f :: Q[true/false/$\text{ok}$/wait])f :: (N_1 M)[false,true/$\text{false}$/\$1-ok] \)
  \begin{itemize}
  \item \( R1(\neg \$ok)) \lor \)
  \end{itemize}
\item also have \(... = (((P[true/false/$\text{ok}$/wait]f :: Q[true/false/$\text{ok}$/wait])f :: (N_1 M)[false,$\text{false}$/\$1-ok] \)
  \begin{itemize}
  \item \( R1(\neg \$ok)) \lor \)
  \end{itemize}
\item also have \(... = (((P[true/false/$\text{ok}$/wait]f :: Q[true/false/$\text{ok}$/wait])f :: (N_1 M)[false,$\text{false}$/\$1-ok] \)
  \begin{itemize}
  \item \( R1(\neg \$ok)) \lor \)
  \end{itemize}
\item also have \(... = (((P[true/false/$\text{ok}$/wait]f :: Q[true/false/$\text{ok}$/wait])f :: (N_1 M)[true,$\text{false}$/\$1-ok] \)
\end{itemize}
\begin{itemize}
\item (is s = (?C1 v p ?C2 v p ?C3 v p ?C4))
\item by (subst parallel-ok-cases, subst-tac)
\end{itemize}
\item also have \(... = (?C2 v ?C3) \)
proof –
\begin{itemize}
\item have \( ?C1 = \text{false} \)
  \begin{itemize}
  \item by (rel-auto)
  \end{itemize}
\item moreover have \( ?C4 \Rightarrow ?C3' \) (is \( (?A :: ?B) \Rightarrow (?C :: ?D) \))
\item proof –
\begin{itemize}
\item from \text{assms} have \( ?P \Rightarrow \) \text{Pf'}
  \begin{itemize}
  \item by (metis RD2-def H2-equivalence Healthy-def)
  \end{itemize}
\item hence \( P' \Rightarrow \text{Pf'} \)
  \begin{itemize}
  \item by (rel-auto)
  \end{itemize}
\item have \( ?A \Rightarrow ?C' \)
  \begin{itemize}
  \item using \text{P} by (rel-auto)
  \end{itemize}
\item moreover have \( ?B \Rightarrow ?D' \)
  \begin{itemize}
  \item by (rel-auto)
  \end{itemize}
\item ultimately show \( \text{thesis} \)
  \begin{itemize}
  \item by (simp add: impl-seq-mono)
  \end{itemize}
\end{itemize}
\item qed
\item ultimately show \( \text{thesis} \)
  \begin{itemize}
  \item by (simp add: subsumption2)
  \end{itemize}
\item qed
lemma \( \text{pre_s-SRD} \):
assumes \( P \) is SRD
shows \( \text{pre_s} \vdash P = (\neg_r \text{pre}_R(P)) \)
proof
  have \( \text{pre_s} \vdash P = \text{pre_s} \vdash R_s(\text{pre}_R P \vdash \text{peri}_R P \circ \text{post}_R P) \)
    by (simp add: SRD-reactive-tri-design assms)
  also have \( \cdots = R1(R2c(\neg \text{pre_s} \vdash \text{pre}_R P)) \)
    by (simp add: RHS-def usubst R3h-def pre_s-design)
  also have \( \cdots = R1(R2c(\neg \text{pre}_R P)) \)
    by (rel-auto)
  also have \( \cdots = (\neg_r \text{pre}_R P) \)
    by (simp add: R2c-not R2c-preR assms rea-not-def)
finally show ?thesis .
qed

lemma \( \text{parallel-assm} \):
assumes \( P \) is SRD \( Q \) is SRD
shows \( \text{pre}_R(P \parallel_{M(R(M))} Q) = (\neg_r ((\neg_r \text{pre}_R(P)) \parallel_{N_0(M)} ; R1(\text{true}) \text{cmt}_R(Q)) \land \neg_r (\text{cmt}_R(P) \parallel_{N_0(M)} ; R1(\text{true}) (\neg_r \text{pre}_R(Q)))) \)
(is \( ?\text{lhs} = ?\text{rhs} \))
proof
  have \( \text{pre}_R(P \parallel_{M(R(M))} Q) = (\neg_r (\text{pre}_s \vdash P) \parallel_{N_0(M)} ; R1 \text{true} (\text{cmt}_s \vdash Q) \land \neg_r (\text{cmt}_s \vdash P) \parallel_{N_0(M)} ; R1 \text{true} (\text{pre}_s \vdash Q)) \)
    by (simp add: preR-def parallel-assm-lemma assms SRD-healths R1-conj rea-not-def[THEN sym])
  also have \( \cdots = ?\text{rhs} \)
    by (simp add: pre_s-SRD assms cmt_R-def Healthy-if closure unrest)
finally show ?thesis .
qed

lemma \( \text{parallel-assm-unrest-wait'} \text{[unrest]} \):
\( \parallel P \) is SRD; \( Q \) is SRD \( \implies \$\text{wait'} \parallel_{M(R(M))} Q \)
by (simp add: parallel-assm, simp add: par-by-merge-def unrest)

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

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

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

69
lemma \( JL4 \): \((M_1 \ M)^t[true, true/\$0-ok, \$1-ok] = (\$ok' \& N_0 \ M) \vdash II_{R^t}\)
by (simp add: merge-rd1-def usubst nmerge-rd1-def anrest)

lemma parallel-commitment-lemma-1:
assumes \( P \) is \( RD^2 \)
shows \( \text{cmt}_s \vdash (P \parallel R(M) \ Q) = (\parallel (M_1 \ M)^t[true, true/\$0-ok, \$1-ok]) \) \( \lor \)
\( ((\text{pre}_s \vdash P) \parallel N_0(M) \vdash R1(true) (\text{cmt}_s \vdash Q)) \) \( \lor \)
\( ((\text{cmt}_s \vdash P) \parallel N_0(M) \vdash R1(true) (\text{pre}_s \vdash Q)) \)

proof –
have \( \text{cmt}_s \vdash (P \parallel R(M) \ Q) = (P[true, false/\$0-ok, \$1-ok] \parallel (M_1(M))^t[true, false/\$0-ok, \$1-ok]) \)
by (simp add: usubst, rel-auto)
also have \( ... = ((P[true, false/\$0-ok, \$1-ok] \parallel Q[true, false/\$0-ok, \$1-ok]) \vdash (M_1 \ M)^t) \)
by (simp add: par-by-merge-def)
also have \( ... = \)
\( (((\text{cmt}_s \vdash P) \parallel (\text{cmt}_s \vdash Q)) \parallel (M_1 \ M)^t[false, true/\$0-ok, \$1-ok]) \) \( \lor \)
\( (((\text{pre}_s \vdash P) \parallel (\text{cmt}_s \vdash Q)) \parallel (M_1 \ M)^t[false, true/\$0-ok, \$1-ok]) \) \( \lor \)
\( (((\text{cmt}_s \vdash P) \parallel (\text{pre}_s \vdash Q)) \parallel (M_1 \ M)^t[false, true/\$0-ok, \$1-ok]) \) \( \lor \)
\( (((\text{pre}_s \vdash P) \parallel (\text{pre}_s \vdash Q)) \parallel (M_1 \ M)^t[false, false/\$0-ok, \$1-ok]) \)
by (subst parallel-ok-cases, subst-tac)
also have \( ... = \)
\( (((\text{cmt}_s \vdash P) \parallel (\text{cmt}_s \vdash Q)) \parallel (M_1(M))^t[true, true/\$0-ok, \$1-ok]) \) \( \lor \)
\( (((\text{pre}_s \vdash P) \parallel (\text{cmt}_s \vdash Q)) \parallel (N_0(M) \vdash R1(true)) \) \( \lor \)
\( (((\text{cmt}_s \vdash P) \parallel (\text{pre}_s \vdash Q)) \parallel (N_0(M) \vdash R1(true)) \) \( \lor \)
\( (((\text{pre}_s \vdash P) \parallel (\text{pre}_s \vdash Q)) \parallel (N_0(M) \vdash R1(true))) \)
by (simp add: JL1 JL2 JL3)
also have \( ... = \)
\( (((\text{cmt}_s \vdash P) \parallel (\text{cmt}_s \vdash Q)) \parallel (M_1(M))^t[true, true/\$0-ok, \$1-ok]) \) \( \lor \)
\( (((\text{pre}_s \vdash P) \parallel (\text{cmt}_s \vdash Q)) \parallel (N_0(M) \vdash R1(true)) \) \( \lor \)
\( (((\text{cmt}_s \vdash P) \parallel (\text{pre}_s \vdash Q)) \parallel (N_0(M) \vdash R1(true)) \) \( \lor \)
\( (((\text{pre}_s \vdash P) \parallel (\text{pre}_s \vdash Q)) \parallel (N_0(M) \vdash R1(true))) \)
proof –
from assns have \( \vdash P' = P^t \)
by (metis RD2-def H2-equivalence Healthy-def)
hence \( P: P'_f \Rightarrow P^{t_f} \)
by (rel-auto)
have \( \vdash ?C'_4 \Rightarrow \vdash ?C'_6 \)
(is \( \vdash (?A \vdash ?B) \Rightarrow (?C \vdash ?D)' \))
proof –
have \( \vdash ?A \Rightarrow ?C' \)
using \( P \) by (rel-auto)
thus \( \vdash ?\text{thesis} \)
by (simp add: impl-seq-mon)
qed
thus \( \vdash ?\text{thesis} \)
by (simp add: subsumption2)
qed
finally show \( \vdash ?\text{thesis} \)
by (simp add: par-by-merge-def JL4)
qed
by (simp add: parallel-commitment-lemma-1 assms parallel-assm-lemma)

lemma parallel-commitment-lemma-3:
M is R1m =⇒ ($ok' ∧ N_0 M) ; H_R' is R1m
by (rel-simp, safe, metis+)

lemma parallel-commitment:
assumes P is SRD Q is SRD M is RDM
shows cmt_R(P ||_M(M) Q) = (pre_R(P ||_M(M) Q) ⇒ cmt_R(Q))
by (simp add: parallel-commitment-lemma-2 parallel-commitment-lemma-3 Healthy-if assms cmt_R-def par-rdes-SRD closure rea-impl-def disj-comm unrest)

theorem parallel-reactive-design:
assumes P is SRD Q is SRD M is RDM
shows (P ||_M(M) Q) = R_s(
¬ (¬_r (¬_pre_R(P)) ||_M(M) ; R(true) cmt_R(Q)) ∧
¬_r (cmt_R(P) ||_M(M) ; R(true) (¬_pre_R(Q)))) ⇒
(cmt_R(P) ||($ok' ∧ N_0 M) ; H_R' cmt_R(Q))) (is ?lhs = ?rhs)
proof –
have (P ||_M(M) Q) = R_s(pre_R(P ||_M(M) Q) ⇒ cmt_R(P ||_M(M) Q))
by (metis Healthy-def NSRD-is-SRD RDM-as-reactive-design assms(1) assms(2) assms(3) par-rdes-NSRD)
also have ... = ?rhs
by (simp add: parallel-assm parallel-commitment design-export-spec assms, rel-auto)
finally show ?thesis .
qed

lemma parallel-pericondition-lemma1:
($ok' ∧ P) ; H_R[true, true/$ok', $wait'] = (∃ $st' · P)[true, true/$ok', $wait']
(is ?lhs = ?rhs)
proof –
have ?lhs = ($ok' ∧ P) ; (∃ $st · H)[true, true/$ok', $wait']
by (rel-blast)
also have ... = ?rhs
by (rel-auto)
finally show ?thesis .
qed

lemma parallel-pericondition-lemma2:
assumes M is RDM
shows (∃ $st' · N_0(M))[true, true/$ok', $wait'] = (($0−wait ∨ $1−wait) ∧ (∃ $st' · M))
proof –
have (∃ $st' · N_0(M))[true, true/$ok', $wait'] = (∃ $st' · ($0−wait ∨ $1−wait) ∧ $str' ≥_u $str_< ∧ M)
by (simp add: usubst unrest nmerge-rd0-def ex-unrest Healthy-if R1m-def assms)
also have ... = (∃ $st' · ($0−wait ∨ $1−wait) ∧ M)
by (metis (no-types, hide-lams) Healthy-if R1m-def R1m-idem R2m-def RDM-def assms utp-pred-laws.inf-commute)
also have ... = (($0−wait ∨ $1−wait) ∧ (∃ $st' · M))
by (rel-auto)
finally show ?thesis .
qed

lemma parallel-pericondition-lemma3:
(($0−wait ∨ $1−wait) ∧ (∃ $st' · M)) = (($0−wait ∧ $1−wait ∧ (∃ $st' · M)) \lor (¬ $0−wait ∧ $1−wait ∧ (∃ $st' · M)) \lor (∃ $st' · M) ∨ ($0−wait ∧ $1−wait ∧ (∃ $st' · M))))
by (rel-auto)

**lemma** parallel-pericondition [rdes]:

fixes $P$ :: ('s,'t::trace,'a) rsp merge

assumes $P$ : SRD Q is SRD

shows $\text{peri}_R(P \parallel M_R(M) \ Q) = (\text{pre}_R (P \parallel M_R(M) \ Q) \Rightarrow \text{peri}_R(P) \parallel \exists \ \text{st'} . M \ \text{peri}_R(Q)$

\[ \lor \ \text{post}_R(P) \parallel \exists \ \text{st'} . M \ \text{peri}_R(Q) \]

\[ \lor \ \text{peri}_R(P) \parallel \exists \ \text{st'} . M \ \text{post}_R(Q) \]

proof –

have $\text{peri}_R(P \parallel M_R(M) \ Q) =$

$(\text{pre}_R (P \parallel M_R(M) \ Q) \Rightarrow \text{cm}t_R P \parallel (\$ok' \land N_0 \ M) :: H_R[[true,\text{true}/\$ok', \$\text{wait}']] \text{cm}t_R Q) \lor \text{post}_R(P) \parallel (\exists \ \text{st'} . M \ \text{peri}_R(Q) \lor \text{peri}_R(P) \parallel \exists \ \text{st'} . M \ \text{post}_R(Q))$

by (simp add: parallel-commitment SRD-healths assms usurp unrest assms)

also have ... = $(\text{pre}_R (P \parallel M_R(M) \ Q) \Rightarrow \text{cm}t_R P \parallel \exists \ \text{st'} . \exists N_0 \ M)[true,\text{true}/\$ok', \$\text{wait}'] \text{cm}t_R Q) \lor \text{post}_R(P) \parallel (\exists \ \text{st'} . M \ \text{peri}_R(Q) \lor \text{peri}_R(P) \parallel \exists \ \text{st'} . M \ \text{post}_R(Q))$

by (simp add: parallel-pericondition-lemma1)

also have ... = $(\text{pre}_R (P \parallel M_R(M) \ Q) \Rightarrow (\text{cm}t_R P)_{\exists \ \text{st'} . M} \parallel (\text{cm}t_R Q)_{\exists \ \text{st'} . M} \parallel \text{cm}t_R Q) \lor \text{post}_R(P) \parallel (\exists \ \text{st'} . M \ \text{peri}_R(Q) \lor \text{peri}_R(P) \parallel \exists \ \text{st'} . M \ \text{post}_R(Q))$

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

also have ... = $(\text{pre}_R (P \parallel M_R(M) \ Q) \Rightarrow (\text{cm}t_R P)_{\exists \ \text{st'} . M} \parallel (\text{cm}t_R Q)_{\exists \ \text{st'} . M} \parallel \text{cm}t_R Q) \lor \text{post}_R(P) \parallel (\exists \ \text{st'} . M \ \text{peri}_R(Q) \lor \text{peri}_R(P) \parallel \exists \ \text{st'} . M \ \text{post}_R(Q))$

by (simp add: parallel-pericondition-lemma3 seqr-or-distr)

also have ... = $(\text{pre}_R (P \parallel M_R(M) \ Q) \Rightarrow (\text{cm}t_R P)_{\exists \ \text{st'} . M} \parallel \text{cm}t_R Q)_{\exists \ \text{st'} . M} \parallel \text{cm}t_R Q) \lor \text{post}_R(P) \parallel (\exists \ \text{st'} . M \ \text{peri}_R(Q) \lor \text{peri}_R(P) \parallel \exists \ \text{st'} . M \ \text{post}_R(Q))$

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

finally show ?thesis.

**qed**

**lemma** parallel-postcondition-lemma1:

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

(is ?lhs = ?rhs)

proof –

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

by (rel-blast)

also have ... = ?rhs

by (rel-auto)

finally show ?thesis.

**qed**

**lemma** parallel-postcondition-lemma2:

assumes $M$ is RDM

shows $(N_0(M))[[true,false/\$ok', \$\text{wait}']] = ((\neg \$0-wait \land \neg \$1-wait) \land M)$

proof –

have $(N_0(M))[[true,false/\$ok', \$\text{wait}']] = ((\neg \$0-wait \land \neg \$1-wait) \land \$tr' \geq \$tr < \land M)$

by (simp add: unrest usurp unrest merge-rd0-def ex-unrest Healthy-if r1m-def assms)

also have ... = $(\neg \$0-wait \land \neg \$1-wait) \land M)$

by (metis Healthy-if r1m-def RDM-R1m assms utp-pred-laws.inf-commute)

finally show ?thesis.

**qed**
qed

**lemma** parallel-postcondition [rdes]:

fixes $M :: (\cdot',\cdot::\text{trace},'\cdot)$ rsp merge
assumes $P$ is SRD Q is SRD $M$ is RDM
shows $\text{post}_R(P \parallel_{\text{M}_R(M)} Q) = (\text{pre}_R (P \parallel_{\text{M}_R M} Q) \Rightarrow_r \text{post}_R(P) \parallel_{\text{M}} \text{post}_R(Q))$

**proof** –

have $\text{post}_R(P \parallel_{\text{M}_R(M)} Q) = (\text{pre}_R (P \parallel_{\text{M}_R M} Q) \Rightarrow_r \text{cnt}_R P \parallel_{(\text{false}/\text{false},'\text{wait}')} \text{cnt}_R Q)$

by (simp add: post-cnt-def parallel-commitment assms usubst unrest SRD-healths)

also have ... = $(\text{pre}_R (P \parallel_{\text{M}_R M} Q) \Rightarrow_r \text{cnt}_R P \parallel_{(\text{false}/\text{false},'\text{wait}')} \text{cnt}_R Q)$


also have ... = $(\text{pre}_R (P \parallel_{\text{M}_R M} Q) \Rightarrow_r \text{post}_R P \parallel_{\text{M}} \text{post}_R(Q))$

by (simp add: par-by-merge-all-def sqrr-right-one-point-false usubst unrest cnt-_def post-_def assms)

finally show $\text{thesis}$.

qed

**lemma** parallel-precondition-lemma:

fixes $M :: (\cdot',\cdot::\text{trace},'\cdot)$ rsp merge
assumes $P$ is NSRD Q is NSRD $M$ is RDM
shows $(\neg_r \text{pre}_R(P)) \parallel_{\text{N}_0(M)} \text{R}_1(\text{true}) \text{cnt}_R(Q) = ((\neg_r \text{pre}_R(P)) \parallel_{\text{M}} \text{R}_1(\text{true}) \text{peri}_R Q \lor (\neg_r \text{pre}_R(P)) \parallel_{\text{M}} \text{R}_1(\text{true}) \text{post}_R(Q))$

**proof** –

have $(\neg_r \text{pre}_R(P)) \parallel_{\text{N}_0(M)} \text{R}_1(\text{true}) \text{cnt}_R(Q) = ((\neg_r \text{pre}_R(P)) \parallel_{\text{M}} \text{R}_1(\text{true}) \text{peri}_R(Q) \circ \text{post}_R(Q)))$

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

also have ... = $(([\neg_r \text{pre}_R(P)]_0 \land \text{peri}_R(Q) \circ \text{post}_R(Q)]_1 \land \text{\$v\ < ' = } \text{\$v}) \land \text{N}_0(M) \land \text{R}_1(\text{true}))$

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

also have ... = $(([\neg_r \text{pre}_R(P)]_0 \land \text{peri}_R(Q)]_1 \land \text{\$v\ < ' = } \text{\$v}) \land \text{N}_0(M) \land \text{R}_1(\text{true}))$

(is $(?P \Rightarrow ?Q) = (?Q \Rightarrow ?Q)$ ;)

**proof** –

have $?P = ?Q$

by (rel-auto)

thus $\text{thesis}$ by simp

qed

also have ... = $(([\neg_r \text{pre}_R(P)]_0 \land \text{peri}_R(Q)]_1 \land \text{\$v\ < ' = } \text{\$v}[\text{true}/\text{\$1\ - wait}]) \land \text{\$v} \land \text{\$v}\ < ' = \text{\$v}) \land \text{N}_0(M) \land \text{R}_1(\text{true}))$

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

also have ... = $(([\neg_r \text{pre}_R(P)]_0 \land \text{peri}_R(Q)]_1 \land \text{\$v\ < ' = } \text{\$v}) \land \text{N}_0(M) \land \text{R}_1(\text{true}) \lor (\neg_r \text{pre}_R(P)]_0 \land \text{peri}_R(Q)]_1 \land \text{\$v\ < ' = } \text{\$v}) \land \text{N}_0(M) \land \text{R}_1(\text{true}))$

by (simp add: usubst unrest)

also have ... = $(([\neg_r \text{pre}_R(P)]_0 \land \text{peri}_R(Q)]_1 \land \text{\$v\ < ' = } \text{\$v}) \land (\text{\$\ - wait} \land \text{M}) \land \text{R}_1(\text{true}) \lor (\neg_r \text{pre}_R(P)]_0 \land \text{peri}_R(Q)]_1 \land \text{\$v\ < ' = } \text{\$v}) \land (\text{\$\ - wait} \land \text{M}) \land \text{R}_1(\text{true}))$

**proof** –

have $(\text{\$tr\ \geq ,} \text{\$tr\ \land } \text{\$M}) = M$

using $\text{RDM-R1m}[OF \text{ assms}(3)]$

by (simp add: Healthy-def R1m-def conj-comm)
thus \( \text{thesis} \)
by (simp add: nmerge-rd0-def unrest assms closure ex-unrest usbst)

qed
also have \( \ldots = \ldots \)
by (simp add: conj-comm)

hence 1: \( ?P_1 = ?Q_1 \)
by (simp add: seqr-left-one-point-true seqr-left-one-point-false add: unrest usbst closure assms)

have \( ?P_2 = \ldots \)
by (simp add: subst seqr-bool-split[of left-uvar wait], simp-all add: usbst unrest assms closure conj-comm)

hence 2: \( ?P_2 = ?Q_2 \)
by (simp add: seqr-left-one-point-true seqr-left-one-point-false unrest usbst closure assms)

from 1 2 show \( \text{thesis} \) by simp

qed
also have \( \ldots = \ldots \)
by (simp add: par-by-merge-all-def)

finally show \( \text{thesis} \)

qed

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

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

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

lemma swapp-merge-rd :
\( \text{swap}_m; M_R(M) = M_R(\text{swap}_m; M) \)
by (simp add: symr-merge-def seqr-assoc[THEN sym] swapp-merge-rd')

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

lemma nmerge-rd1-merge3 :
\( \text{assumes } M \text{ is RDM} \)
\( \text{shows } M_3(N_1(M)) = (\$\text{ok'} = _u (\$\text{0-ok} \land \$1-\text{0-ok} \land \$1-\text{1-ok}) \land \$\text{wait'} = _u (\$\text{0-wait} \lor \$\text{1-wait} \lor \$\text{1-wait}) \land M_3(M)) \)

proof
have \( M_3(N_1(M)) = M_3(\$\text{ok'} = _u (\$\text{0-ok} \land \$1-\text{0-ok}) \land \$\text{wait'} = _u (\$\text{0-wait} \lor \$\text{1-wait}) \land \$\text{tr'} \leq _u \$\text{tr'} \land (3 \{\$\text{0-ok}, \$1-\text{0-ok}, \$\text{ok'}, \$\text{0-wait}, \$\text{1-wait}, \$\text{wait'}, \$\text{wait'} \} \cdot \text{RDM}(M)) \) 
by (simp add: nmerge-rd1-def nmerge-rd0-def assms Healthy-if)
also have \( \ldots = M_3(\$\text{ok'} = _u (\$\text{0-ok} \land \$1-\text{0-ok}) \land \$\text{wait'} = _u (\$\text{0-wait} \lor \$\text{1-wait}) \land M_3(M)) \)

lemma swap-merge-RDM-closed:
\[ M \land (N(M) = (\exists \text{st}_N \cdot \text{sv}_N = u \land \text{wait}_N > M) < N(M) < (\text{str}_N \leq u \land \text{str}') \]
by (rel-blast)

also have \[ (\exists \text{st}_N \cdot \text{sv}_N = u \land \text{wait}_N > M) < N(M) < (\text{str}_N \leq u \land \text{str}') \]
by (rel-blast)

lemma nmerge-merge3:
\[ M3(N(M)) = (\exists \text{st}_N \cdot \text{sv}_N = u \land \text{wait}_N > M) < N(M) < (\text{str}_N \leq u \land \text{str}') \]
by (rel-blast)

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

lemma parallel-precondition:
fixes P :: (\text{'s, t::trace}, \alpha) \text{rsp merge}
assumes P is NSRD Q is NSRD M is RDM
shows \[ \text{pre}_R(P \parallel M(M) \parallel Q) = \]
\[ (\neg_{r} (\neg_{r} \text{pre}_R(P) \parallel M(M) \parallel R_{1}(\text{true}) \text{peri}_R(Q) \land \neg_{r} (\neg_{r} \text{pre}_R(P) \parallel M(M) \parallel R_{1}(\text{true}) \text{post}_R(Q) \land \neg_{r} (\neg_{r} \text{pre}_R(Q) \parallel M(M) \parallel R_{1}(\text{true}) \text{peri}_R(P) \land \neg_{r} (\neg_{r} \text{pre}_R(Q) \parallel M(M) \parallel R_{1}(\text{true}) \text{post}_R(Q))) \]
proof –
have \[ a: (\neg_{r} \text{pre}_R(P) \parallel N_0(M) ; R_1(\text{true}) \text{cmt}_R(Q) = \]
\[ (\neg_{r} \text{pre}_R(P) \parallel M(M) ; R_1(\text{true}) \text{peri}_R(Q) \lor (\neg_{r} \text{pre}_R(P) \parallel M(M) ; R_1(\text{true}) \text{post}_R(Q)) \]
by (simp add: parallel-precondition-lemma \text{assms})

have \[ b: (\neg_{r} \text{cmt}_R(P \parallel N_0 M ; R_1 \text{true}) (\neg_{r} \text{pre}_R(Q)) = \]
\[ (\neg_{r} (\neg_{r} \text{pre}_R(Q) \parallel N_0(\text{swap}_m ; M) ; R_1(\text{true}) \text{cmt}_R(P)) \]
by (simp add: swap-nmerge-rd0[THEN \text{sym}] \text{seqr-assoc}[THEN \text{sym}] \text{par-by-merge-def} \text{par-sep-swap})

have \[ c: (\neg_{r} \text{pre}_R(Q) \parallel N_0(\text{swap}_m ; M) ; R_1(\text{true}) \text{peri}_R(P) \lor (\neg_{r} \text{pre}_R(Q) \parallel M(M) ; R_1(\text{true}) \text{cmt}_R(P) = \]
\[ (\neg_{r} \text{pre}_R(Q) \parallel \text{swap}_m ; M ; R_1(\text{true}) \text{peri}_R(P) \lor (\neg_{r} \text{pre}_R(Q) \parallel M(M) ; R_1(\text{true}) \text{post}_R(Q)) \]
by (simp add: parallel-precondition-lemma \text{closure \text{assms}})

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

Weakest Parallel Precondition

definition \text{wr} ::
(\text{'t::trace}, \alpha) \text{hrel-rp} \Rightarrow
(\text{'t} :: \text{trace}, \alpha) \text{rp merge} \Rightarrow
(\text{'t}, \alpha) \text{hrel-rp} \Rightarrow
Lemma \( \text{wrR-R1} \): 
\[
M \text{ is } R1m \implies Q \text{ wr}_R(M) P \text{ is } R1
\]
by (simp add: wrR-def closure)

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

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

Lemma \( \text{wrR-RR} \): 
\[
\text{assumes } P \text{ is } RR Q \text{ is } RR M \text{ is } RDM
\]
shows \( Q \text{ wr}_R(M) P \text{ is } RR
\)
proof (rule RR-intro)
apply (metis (no-types, lifting) Healthy-def \( R1-R2c-commute \) \( R1-R2c-is-R2 \) \( R1-rea-not \) \( RDM-R2m \) \( RR-implies-R2 \) \( assays(1) \) \( assays(2) \) \( assays(3) \) \( par-by-merge-seq-add rea-not-R2-closed \) \( \text{wrR-R2-lemma} \) )
done

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

Lemma \( \text{uppr-choice} \): \( (P \lor Q) \text{ wr}_R(M) R = (P \text{ wr}_R(M) R \land Q \text{ wr}_R(M) R) \)
proof –
\[
\text{have } (P \lor Q) \text{ wr}_R(M) R = \neg_r (\neg_r R) \lor U0 \land (P \land U1 \lor Q \land U1) \land $v <' \_u =_u $v \land M \land true_r)
\]
by (simp add: wrR-def par-by-merge-def seqr-or-distl)
also have \( ... = \neg_r (\neg_r R) \lor U0 \land P \land U1 \land $v <' \_u =_u $v \lor (\neg_r R) \lor U0 \land Q \land U1 \land $v <' \_u =_u \)
\$v\vdash;M\vdash;true_r$

by (simp add: conj-disj-distr utp-pred-laws.inf-sup-distrib2)
also have \(\ldots = (\neg_r ((\neg_r R) \vdash; U0 \land P) \vdash; U1 \land \neg v < =_u \$v) \vdash; M \vdash; true_r \lor (\neg_r R) \vdash; U0 \land Q) \vdash; U1 \land \neg v < =_u \$v) \vdash; M \vdash; true_r\))

by (simp add: seq-or-distl)
also have \(\ldots = (P \ wr_R(M) R \land Q \ wr_R(M) R)\)
by (simp add: wrR-def par-by-merge-def)
finally show \(?thesis\).

qed

lemma uppR-miracle [wp]: false \(\ wr_R(M) P = true_r\)
by (simp add: wrR-def)

lemma uppR-true [wp]: \(P \ wr_R(M) true_r = true_r\)
by (simp add: wrR-def)

lemma parallel-precondition-ur [rdes]:
assumes \(P\) is NSRD \(Q\) is NSRD \(M\) is RDM
shows \(\ pre_R(P \ ||_{M_R(M)}Q) = (\ peri_R(Q) \ wr_R(M) \ pre_R(P) \land \ post_R(Q) \ wr_R(M) \ pre_R(P) \land \ peri_R(P) \ wr_R(swampm :: M) \ pre_R(Q) \land \ post_R(P) \ wr_R(swampm :: M) \ pre_R(Q))\)
by (simp add: assms parallel-precondition wrR-def)

lemma parallel-rdes-def [rdes-def]:
assumes \(P_1\) is RC \(P_2\) is RR \(P_3\) is RR \(Q_1\) is RC \(Q_2\) is RR \(Q_3\) is RR
\(\$st' :: P_2 \ $st' :: Q_2\)
\(M\) is RDM
shows \(\mathbf{R}_4(P_1 \vdash P_2 \circ P_3) \parallel_{M_R(M)} \mathbf{R}_4(Q_1 \vdash Q_2 \circ Q_3) =\)
\(\mathbf{R}_4(((Q_1 \Rightarrow_r Q_3) \ wr_R(M) P_1 \land (Q_1 \Rightarrow_r Q_3) \ wr_R(M) P_1 \land (P_1 \Rightarrow_r P_2) \ wr_R(swampm :: M) \ Q_1 \land (P_1 \Rightarrow_r P_3) \ wr_R(swampm :: M) \ Q_1) \vdash ((P_1 \Rightarrow_r P_2) \ ||_{M} \ Q_1 \Rightarrow_r Q_2) \lor ((P_1 \Rightarrow_r P_3) \ ||_{M} \ Q_1 \Rightarrow_r Q_2) \lor (P_1 \Rightarrow_r P_2) \ ||_{M} \ Q_1 \Rightarrow_r Q_3)) \vdash (P_1 \Rightarrow_r P_3) \ ||_{M} \ (Q_1 \Rightarrow_r Q_3)) (\text{is} \ ?lhs \ = \ ?rhs)\)

proof –

have \(?lhs = \mathbf{R}_4(\ pre_R ?lhs \vdash \ peri_R ?lhs \circ \ post_R ?lhs)\)
by (simp add: SRD-reactive-tri-design assms closure)
also have \(?rhs\)
by (simp add: rdes closure unrest assms, rel-auto)
finally show \(?thesis\).

qed

lemma Miracle-parallel-left-zero:
assumes \(P\) is SRD \(M\) is RDM
shows \(\ Miracle \ ||_{RM} P = \ Miracle\)

proof –

have \(\ pre_R(\ Miracle \ ||_{RM} P) = true_r,\)
by (simp add: parallel-assm wait'-cond-idem rdes closure assms)
moreover hence \(\ cmn_R(\ Miracle \ ||_{RM} P) = false\)
by (simp add: rdes closure wait'-cond-idem SRD-healths assms)
ultimately have \(\ Miracle \ ||_{RM} P = \mathbf{R}_4(true_r \vdash false)\)
by (metis NSRD-iff SRD-reactive-design-alt assms par-rdes-NSRD srdes-theory-continuous.weak.top-closed)
thus \(?thesis\)
by (simp add: Miracle-def R1-design-R1-pre)

qed

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

8.1 Example basic merge

definition BasicMerge :: "((s', t::trace, unit) rsp) merge (N_B) where
[upred-defs]: BasicMerge = (\$(tr<_u \$tr' \& \$tr' - \$tr<_u \$0 - \$tr<_u \& \$tr' - \$tr<_u \$1 - \$tr' \& \$st' =_u \$st<_u)

abbreviation rbasic-par (- ||_B - [85,86] 85) where
$P \parallel_B Q \equiv P \parallel_M(N_B) \parallel_B Q$

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

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

lemma BasicMerge'-calc:
  assumes $\$ok' \notin P \$wait' \notin P \$ok' \notin Q \$wait' \notin Q \ P \parallel Q = R2 \ Q \parallel R2
  shows $P \parallel_N_B \ Q = ((\exists \$st' \cdot P) \& (\exists \$st' \cdot Q) \& \$st' =_u \$st)$
  using assms
proof –
  have $P:(\exists \$ok',\$wait') \cdot R2(P)) = P \ (\$?P' = -)
    by (simp add: ex-unrest ex-plus Healthy-if assms)
  have $Q:(\exists \$ok',\$wait') \cdot R2(Q)) = Q \ (\$?Q' = -)
    by (simp add: ex-unrest ex-plus Healthy-if assms)
  have $?P' \parallel_P Q' = ((\exists \$st' \cdot ?P') \& (\exists \$st' \cdot ?Q') \& \$st' =_u \$st)$
    by (simp add: par-by-merge-alt-def, rel-auto, blast+)
  thus ?thesis
    by (simp add: P Q)
qed

8.2 Simple parallel composition

definition rea-design-par ::
  "(s', t::trace, 'a) hrel-rsp \Rightarrow (s', 't, 'a) hrel-rsp \Rightarrow (s', 't, 'a) hrel-rsp (infixr \parallel R 85)
where [upred-defs]: P \parallel_R Q = R_\alpha((\text{pre}_R(P) \& \text{pre}_R(Q)) \vdash (\text{cmt}_R(P) \& \text{cmt}_R(Q)))$

lemma RHS-design-par:
  assumes $\$ok' \notin P_1 \$ok' \notin P_2$
  shows $R_\alpha(P_1 \vdash Q_1) \parallel_R R_\alpha(P_2 \vdash Q_2) = R_\alpha((P_1 \& P_2) \vdash (Q_1 \& Q_2))$
proof –
  have $R_\alpha(P_1 \vdash Q_1) \parallel_R R_\alpha(P_2 \vdash Q_2) =$
R₅(P₁[true,false/$\$ok,\$\$wait] ⊨ Q₁[true,false/$\$ok,\$\$wait]) \parallel R₅(P₂[true,false/$\$ok,\$\$wait] ⊨ Q₂[true,false/$\$ok,\$\$wait])
by (simp add: RHS-design-ok-wait)

also from assms
have ...
  R₅((R₁ (R₂c (P₁)) ∧ R₁ (R₂c (P₂))))[true,false/$\$ok,\$\$wait] ⊨
  \{(R₁ (R₂c (P₁)) ⇒ Q₁) ∧ R₁ (R₂c (P₂) ⇒ Q₂)\}
by (simp add: R₅ design-par assms unrest assms)

apply (simp add: rule cong[of R₅ R₅], simp)
using assms apply (rel-auto)
done

also have ...
  R₅((P₁ ∧ P₂) ⊨ (R₁ (R₂s (P₁) ⇒ Q₁) ∧ R₁ (R₂s (P₂) ⇒ Q₂)))
by (simp add: R₅ 3lth-commute R₂c-and R₂c-design R₂c-idem R₂c-not RHS-def)

also have ...
  R₅((P₁ ∧ P₂) ⊨ ((P₁ ⇒ Q₁) ∧ (P₂ ⇒ Q₂)))
by (metis (no-types, lifting) R₁-conj R₂s-conj RHS-design-export-R₁ RHS-design-export-R₂)

finally show ?thesis .

qed

lemma RHS-tri-design-par:
  assumes $\$\$ok' ⊨ P₁ $\$\$ok' ⊨ P₂
  shows R₅(P₁ ⊨ Q₁ ⊢ R₁) \parallel R₅(P₂ ⊨ Q₂ ⊢ R₂) = R₅((P₁ ∧ P₂) ⊨ (Q₁ ∧ Q₂) ⊢ (R₁ ∧ R₂))
by (simp add: RHS-design-par assms unrest wait'-cond-conj-exchange)

lemma RHS-tri-design-par-RR [rdes-def]:
  assumes P₁ is RR P₂ is RR
  shows R₅(P₁ ⊨ Q₁ ⊢ R₁) \parallel R₅(P₂ ⊨ Q₂ ⊢ R₂) = R₅((P₁ ∧ P₂) ⊨ (Q₁ ∧ Q₂) ⊢ (R₁ ∧ R₂))
by (simp add: RHS-tri-design-par assms)

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

end

9 Productive Reactive Designs

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

9.1 Healthiness condition

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

definition Productive :: ('s, 't::trace, 'a) hrel-rsp ⇒ ('s, 't, 'a) hrel-rsp where
lemma Productive-RHS-design-form:
assumes $\text{ok} \not\in P \, \text{ok} \not\in Q \, \text{ok} \not\in R$
shows Productive($R_s(P \lor Q \lor R)) = R_s(P \lor Q \lor (R \land \text{tr} < u \land \text{tr}'))
using assms by (simp add: Productive-def RHS-tri-design-par unrest)

lemma Productive-form:
Productive($SRD(P)) = R_s(pre_R(P) \vdash peri_R(P) \circ (post_R(P) \land \text{tr} < u \land \text{tr'}))$
proof
have Productive($SRD(P)) = R_s(pre_R(P) \vdash peri_R(P) \circ (post_R(P) \land \text{tr} < u \land \text{tr'}))$
  by (simp add: Productive-def SRD-as-reactive-tri-design)
also have ... = R_s(pre_R(P) \vdash peri_R(P) \circ (post_R(P) \land \text{tr} < u \land \text{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 $P \, SRD \, (\text{tr} < u \land \text{tr'}) \subseteq (pre_R(P) \land post_R(P))$
shows $P \, Productive$
proof
have $P \cdot R_s(pre_R(P) \vdash peri_R(P) \circ (post_R(P) \land \text{tr} < u \land \text{tr'})) = P$
proof
  have $R_s(pre_R(P) \vdash peri_R(P) \circ (post_R(P))) = R_s(pre_R(P) \vdash (pre_R(P) \land peri_R(P)) \circ (pre_R(P) \land post_R(P)))$
    by (metis (no-types, hide-lams) design-export-pre wait-cond-conj-exchange wait-cond-idem)
  also have ... = R_s(pre_R(P) \vdash (pre_R(P) \land peri_R(P)) \circ (pre_R(P) \land (post_R(P) \land \text{tr} < u \land \text{tr'})))
    by (metis assms(2) utp-pred-laws.inf.absorb1 utp-pred-laws.inf.assoc)
  also have ... = R_s(pre_R(P) \vdash peri_R(P) \circ (post_R(P) \land \text{tr} < u \land \text{tr'}))
    by (metis (no-types, hide-lams) design-export-pre wait-cond-cond-exchange wait-cond-idem)
finally show ?thesis
  by (simp add: SRD-reactive-tri-design assms(1))
qed

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

qed

lemma Productive-post-refines-tr-increase:
assumes $P \, SRD \, P \, Productive \, \text{wait'} \not\in pre_R(P)$
shows $(\text{tr} < u \land \text{tr'}) \subseteq (pre_R(P) \land post_R(P))$
proof
have post_R(P) = post_R(R_s(pre_R(P) \vdash peri_R(P) \circ (post_R(P) \land \text{tr} < u \land \text{tr'})))
  by (metis Healthy-def Productive-form assms(1) assms(2))
also have ... = R1(R2c(pre_R(P) \Rightarrow (post_R(P) \land \text{tr} < u \land \text{tr'})))
  by (simp add: rea-post-RHS-design unrest usubst assms, rel-auto)
also have ... = R1((pre_R(P) \Rightarrow (post_R(P) \land \text{tr} < u \land \text{tr'}))
  by (simp add: R2c-impl R2c-prefR R2c-postR R2c-and R2c-tr-less-tr' assms)
also have $(\text{tr} < u \land \text{tr'}) \subseteq (pre_R(P) \land ...)$
  by (rel-auto)
finally show ?thesis
qed
9.2 Reactive design calculations

**Lemma** \(\text{Continuous-Productive} \ [\text{rdes}]:\) Continuous Productive

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

---

**Lemma** \(\text{preR-Productive} \ [\text{rdes}]:\)

assumes \(P \text{ is SRD}\)

shows \(\text{pre}_R(\text{Productive}(P)) = \text{pre}_R(P)\)

**Proof**

- have \(\text{pre}_R(\text{Productive}(P)) = \text{pre}_R(\text{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \circ (\text{post}_R(P) \land \text{str} <_u \text{str}')))\)
  
  by (metis Healthy-def Productive-form assms)

  **Proof**

  - have \(\text{peri}_R(\text{Productive}(P)) = \text{peri}_R(\text{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \circ (\text{post}_R(P) \land \text{str} <_u \text{str}')))\)
    
    by (metis Healthy-def Productive-form assms)

finally show \(?thesis\).

**QED**

---

**Lemma** \(\text{postR-Productive} \ [\text{rdes}]:\)

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

shows \(\text{post}_R(\text{Productive}(P)) = (\text{pre}_R(P) \Rightarrow r \text{ post}_R(P) \land \text{str} <_u \text{str}'')\)

**Proof**

- have \(\text{post}_R(\text{Productive}(P)) = \text{post}_R(\text{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \circ (\text{post}_R(P) \land \text{str} <_u \text{str}'')))\)
  
  by (metis Healthy-def Productive-form assms)

also have \(?thesis\).

**QED**

---

**Lemma** \(\text{preR-frame-seq-export}:\)

assumes \(P \text{ is NSRD} \land P \text{ is Productive} Q \text{ is NSRD}\)

shows \((\text{pre}_R P \land (\text{pre}_R P \land \text{post}_R P) :: Q) = (\text{pre}_R P \land (\text{post}_R P :: Q))\)

**Proof**

- have \((\text{pre}_R P \land (\text{post}_R P :: Q) = (\text{pre}_R P \land ((\text{pre}_R P \Rightarrow r \text{ post}_R P) :: Q))\)
  
  by (simp add: SRD-post-under-pre assms closure unrest)

also have \(?thesis\).

**QED**

---

**Lemma** \(\text{健康的框架-seq-export}:\)

assumes \(P \text{ is NSRD} \land P \text{ is Productive} Q \text{ is NSRD}\)

shows \((\text{pre}_R P \land ((\text{pre}_R P) :: Q \lor (\text{pre}_R P \Rightarrow r \text{ post}_R P) :: Q))\)

by (simp add: Healthy-form assms closure unrest)

**Proof**

- have \((\text{pre}_R P \land ((\text{pre}_R P) :: Q \lor (\text{pre}_R P \land \text{post}_R P) :: Q))\)
  
  by (simp add: Healthy-form assms closure unrest)

final show \(?thesis\).

**QED**

---

\(81\)
then show ?thesis
  by (metis (no_types, lifting) R1-def conj-comm disj-comm disj-conj-distr rea-impl-def seqr-or-distl utp-pred-laws.inf-top-left utp-pred-laws.sup.left-idem)
qed

also have ... = (preR P ∧ ((¬r preR P) ∨ (preR P ∧ postR P) ;; Q)))
  by (simp add: NSRD-neg-pre-left-zero assms closure SRD-healths)
also have ... = (preR P ∧ (preR P ∧ postR P) ;; Q)
  by (rel-blast)
finally show ?thesis ..
qed

9.3 Closure laws

lemma Productive-rdes-intro:
  assumes (\$tr <_u \$tr') ⊑ R $ok' \# P $ok' \# Q $ok' \# R $wait' \# P $wait' \# P
  shows (R""(P ⊑ Q ∘ R)) is Productive
proof (rule Productive-intro)
  show R"" (P ⊑ Q ∘ R) is SRD
    by (simp add: RHS-tri-design-is-SRD assms)

from assms(1) show (\$tr' >_u \$tr) ⊑ (preR (R"" (P ⊑ Q ∘ R)) ∧ postR (R"" (P ⊑ Q ∘ R)))
  apply (simp add: rea-pre-RHS-design rea-post-RHS-design usubst assms unrest)
  using assms(1) apply (rel-auto)
  apply fastforce
done

show $\$wait' \# preR (R"" (P ⊑ Q ∘ R))
  by (simp add: rea-pre-RHS-design rea-post-RHS-design usubst R1-def R2c-def R2s-def assms unrest)
qed

We use the R4.healthiness condition to characterise that the postcondition must extend the trace for a reactive design to be productive.

lemma Productive-rdes-RR-intro:
  assumes P is RR Q is RR R is RR R is R4
  shows (R""(P ⊑ Q ∘ R)) is Productive
  using assms by (simp add: Productive-rdes-intro R4-iff-refine unrest)

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

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

lemma Productive-intChoice [closure]:
  assumes P is SRD P is Productive Q is SRD Q is Productive
  shows P ⊑ Q is Productive
proof –
  have P ⊑ Q =
    R""(preR(P) ⊑ periR(P) o (postR(P) ∧ $tr <_u $tr')) ∩ R""(preR(Q) ⊑ periR(Q) o (postR(Q) ∧ $tr <_u $tr'))
    by (metis Healthy-if Productive-form assms)
  also have ... = R"" ((preR P ∧ preR Q) ⊑ (periR P ∨ periR Q) o ((postR P ∧ $tr' >_u $tr) ∨ (postR Q ∧ $tr' >_u $tr)))
by (simp add: RHS-tri-design-choice)
also have ... = Rs (\{\text{pre}_R P \land \text{pre}_R Q\} \vdash \text{peri}_R P \lor \text{peri}_R Q) \circ ((\text{post}_R P) \lor (\text{post}_R Q)) \land \text{str'} \succsim_u \text{str})
  by (rule \text{cong}[of Rs, Rs], simp, rel-auto)
also have ... is Productive
  by (simp add: Healthy-def Productive-RHS-design-form unrest)
finally show \text{thesis}.
qed

lemma Productive-cond-rea [closure];
assumes P is SRD P is Productive Q is SRD Q is Productive
shows P \circ b \triangleright_R Q is Productive
proof –
  have P \circ b \triangleright_R Q =
    Rs (\{\text{pre}_R P\} \vdash \text{peri}_R P) \circ (\text{post}_R P) \land \text{str} \precsim_u \text{str'} \circ b \triangleright_R
    Rs (\{\text{pre}_R Q\} \vdash \text{peri}_R Q) \circ (\text{post}_R Q) \land \text{str} \precsim_u \text{str'}
  by (metis Healthy-if Productive-form assms)
also have ... = Rs (\{\text{pre}_R P \circ b \triangleright_R \text{pre}_R Q\} \vdash \text{peri}_R P \circ b \triangleright_R \text{peri}_R Q) \circ ((\text{post}_R P \circ b \triangleright_R (\text{post}_R Q)) \land \text{str'} \succsim_u \text{str})
  by (simp add: cond-srea-form)
also have ... = Rs ((\text{pre}_R P \circ b \triangleright_R \text{pre}_R Q) \vdash \text{peri}_R P \circ b \triangleright_R \text{peri}_R Q) \circ ((\text{post}_R P \circ b \triangleright_R (\text{post}_R Q)))
also have ... is Productive
  by (simp add: Healthy-def Productive-RHS-design-form unrest)
finally show \text{thesis}.
qed

lemma Productive-seq-1 [closure];
assumes P is NSRD P is Productive Q is NSRD
shows P \circ Q is Productive
proof –
  have P \circ Q = Rs (\{\text{pre}_R P\} \vdash \text{peri}_R P) \circ (\text{post}_R P) \land \text{str} \precsim_u \text{str'} \circ Q
    Rs (\{\text{pre}_R Q\} \vdash \text{peri}_R Q) \circ (\text{post}_R Q)
  by (metis Healthy-def NSRD-is-SRD SRD-reactive-tri-design Productive-form assms(1) assms(2) assms(3))
also have ... = Rs ((\text{pre}_R P \land (\text{post}_R P \land \text{str'} \succsim_u \text{str}) \circ \text{pre}_R Q) \vdash
    \text{peri}_R P \lor ((\text{post}_R P \land \text{str'} \succsim_u \text{str} \circ \text{peri}_R Q)) \circ ((\text{post}_R P \land \text{str'} \succsim_u \text{str}) \circ \text{post}_R Q)
  by (simp add: RHS-tri-design-composition-wp rpred unrest closure assms wp NSRD-neg-pre-left-zero
    SRD-healths ex-unrest wp-rea-def disj-upred-def)
also have ... = Rs ((\text{pre}_R P \land (\text{post}_R P \land \text{str'} \succsim_u \text{str}) \circ \text{pre}_R Q) \vdash
    \text{peri}_R P \lor ((\text{post}_R P \land \text{str'} \succsim_u \text{str} \circ \text{peri}_R Q)) \circ ((\text{post}_R P \land \text{str'} \succsim_u \text{str}) \circ \text{post}_R Q)
  by (simp add: RHS-tri-design-composition-wp rpred unrest closure assms wp-rea-def)
proof –
  have ((\text{post}_R P \land \text{str'} \succsim_u \text{str}) \circ R1(\text{post}_R Q)) = ((\text{post}_R P \land \text{str'} \succsim_u \text{str}) \circ R1(\text{post}_R Q) \land \text{str'}
    \succsim_u \text{str})
    by (rel-auto)
  thus \text{thesis}
    by (simp add: NSRD-is-SRD R1-post-SRD assms)
qed
also have ... is Productive
  by (rule Productive-rls-intro, simp-all add: unrest assms closure wp-rea-def)
finally show \text{thesis}.
qed
lemma Productive-seq-2 [closure]:
  assumes P is NSRD Q is NSRD Q is Productive
  shows P ;; Q is Productive
proof –
  have P ;; Q = tr_r (pre_R(P) \vdash peri_R(P) \circ (post_R(P))) ;; R_u (pre_R(Q) \vdash peri_R(Q) \circ (post_R(Q) \land \text{str} <_u (\text{str}')))
    by (metis Healthy-def NSRD-is-SRD SRD-reactive-tri-design Productive-form assms)
  also have \ldots = R_u ((pre_R P \land post_R P \land peri_R Q) \vdash (peri_R P \land \text{peri}_R Q) \circ (post_R P ;; (post_R Q \land \text{str} >_u \text{str}')))
    by (simp add: RHS-tri-design-composition-wpred rpred unrest closure assms wp NSRD-neg-pre-left-zero SRD-healths ex-unrest wp-rea-def disj-upred-def)
  also have \ldots = R_u ((pre_R P \land post_R P \land peri_R Q) \vdash (peri_R P \land \text{peri}_R Q) \circ (post_R P ;; (post_R Q \land \text{str} >_u \text{str}')))
    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 ucard function that provides this.

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

lemma size-mono: s \leq t \implies \text{size}(s) \leq \text{size}(t)
  by (metis le-add1 local.diff-add-cancel-left1 local.size-plus)

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

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

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

end

Both natural numbers and lists are measurable trace algebras.

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

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

syntax
-usize :: logic \(\Rightarrow\) logic (size_u'('\-'))

translations
size_u(t) \(\mapsto\) CONST uop CONST size t

10.2 Guardedness

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

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

lemma gvt-limit: \(\bigwedge\) (range gvt) = ($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 gvt(n+1)) = (F(X \land gvt(n)) \land gvt(n+1)))

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

Guarded reactive designs yield unique fixed-points.

theorem guarded-fp-uniq:
  assumes mono F F \in [id]_H \rightarrow [SRD]_H Guarded F
  shows \(\mu F = \nu F\)
proof
  have constr F gvt
  using assms
  by (auto simp add: constr-def gvt-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 $(\$tr \leq_\mu \$tr' \land \mu F) = \mu F$
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 $(\$tr \leq_\nu \$tr' \land \nu F) = \nu F$
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 $\land P. P \in A \Longrightarrow \text{Guarded } P$
shows $\text{Guarded } (\lambda X. \prod P \in A \cdot P(X))$
proof (rule GuardedI)
  fix $X$ $n$
  have $\land Y. (\prod P \in A \cdot (P \land gvrt(n+1))) = (\prod P \in A \cdot (P \land \land gvrt(n+1)))$
  proof
    fix $Y$
    let $?lhs = (\prod P \in A \cdot (P \land gvrt(n+1)))$ and $?rhs = (\prod P \in A \cdot (P \land \land gvrt(n+1)))$
    have $a: ?lhs[false/$ok$] = ?rhs[false/$ok$]
      by (rel-auto)
    have $b: ?lhs[true/$ok$][true/$wait$] = ?rhs[true/$ok$][true/$wait$]
      by (rel-auto)
    have $c: ?lhs[true/$ok$][false/$wait$] = ?rhs[true/$ok$][false/$wait$]
      by (rel-auto)
    show $?lhs = ?rhs$
      using $a$ $b$ $c$
      by (rule-tac bool-eq-splitI[of in-var wait], simp, rule-tac bool-eq-splitI[of in-var wait], simp-all)
  qed
moreover have $$(\prod P \in A \cdot (P \land gvrt(n+1))) \land gvrt(n+1) = (\prod P \in A \cdot (P \land \land gvrt(n+1))) \land \land gvrt(n+1))$$
proof
  have $\prod P \in A \cdot (P \land gvrt(n+1))) = (\prod P \in A \cdot (P \land \land gvrt(n+1)))$
  proof (rule UINF-cong)
    fix $P$ assume $P$ $\in A$
    thus $P \land gvrt(n+1)) = (P \land \land gvrt(n+1))$
      using Guarded-def assms by blast
  qed
  thus $?thesis$ by simp
qed
ultimately show $$(\prod P \in A \cdot P \landgvrt(n+1)) = (\prod P \in A \cdot (P \land \land gvrt(n+1))) \land \land gvrt(n+1))$$
  by simp

86
lemma intChoice-Guarded [closure]:
  assumes Guarded P Guarded Q
  shows Guarded (λ X. P(X) ∈ Q(X))
proof –
  have Guarded (λ X. P(X) ∈ Q(X))
  by (rule UNIF-Guarded, auto simp add: assms)
thus ?thesis
  by (simp)
qed

lemma cond-srea-Guarded [closure]:
  assumes Guarded P Guarded Q
  shows Guarded (λ X. P(X) ∈ Q(X))
using assms (rel-auto)

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

lemma Guarded-if-Productive [closure]:
  fixes P :: (‘s, ‘t::size-trace,’a) hrel-rsp
  assumes P is NSRD P is Productive
  shows Guarded (λ X. P :: SRD(X))
proof (clarsimp simp add: Guarded-def)
  — We split the proof into three cases corresponding to valuations for ok, wait, and wait’
  respectively.
  fix X n
  have a: \(P :: SRD(X) \land gvt (Suc n)\) = \(P :: SRD(X) \land gvt (Suc n)\)
  by (simp add: subst closure SRD-left-zero 1 assms)
  have b: \(P :: SRD(X) \land gvt (Suc n)\) = \(P :: SRD(X) \land gvt (Suc n)\)
  by (simp add: subst closure SRD-left-zero 2 assms)
  have c: \(P :: SRD(X) \land gvt (Suc n)\) = \(P :: SRD(X) \land gvt (Suc n)\)
  proof –
  have 1: \(P[true\land wait'] :: SRD(X)\) = \(P[true\land wait'] :: SRD(X)\)
  by (metis (no_types, lifting) Healthy-def R3h-wait-true SRD-healths SRD-idem)
  have 2: \(P[false\land wait'] :: SRD(X)\) = \(P[false\land wait'] :: SRD(X)\)
  proof –
  have \(\exists Y :: (\'s, \'t, \'a) \text{ hrel-rsp. } (P[false\land wait'] :: SRD(Y))\) = \(P[false\land wait'] :: SRD(Y)\)
  proof –
  have \(P[false\land wait'] :: SRD(Y)\) = \(P[false\land wait'] :: SRD(Y)\)
  by (metis (no_types) Healthy-def Productive-form assms 1 assms 2 NSRD-is-SRD)
  also have \(\ldots = \(P[false\land wait'] :: SRD(Y)\)\)
  by (metis (no_types) Healthy-def Productive-form assms 1 assms 2 NSRD-is-SRD)
  fix Y :: (\'s, \'t, \'a) \text{ hrel-rsp}
  have \(P[false\land wait'] :: SRD(Y)\) = \(P[false\land wait'] :: SRD(Y)\)
  by (metis (no_types) Healthy-def Productive-form assms 1 assms 2 NSRD-is-SRD)
  also have \(\ldots = \(P[false\land wait'] :: SRD(Y)\)\)
  by (metis (no_types) Healthy-def Productive-form assms 1 assms 2 NSRD-is-SRD)
  qed

87
by (simp add: RHS-def R1-def R2c-def R2s-def R3h-def RD1-def RD2-def usubst unrest assms closure design-def)

also have ... =

(((¬_其 pre_R(P) \lor (\$ok' \land post_R(P) \land \$tr <_u \$tr')))[false/\$wait'] ;; (SRD Y)[false/\$wait])
\wedge gvt (Suc n))[[true,false/\$ok,\$wait]]

by (simp add: impl-ali-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 ... =

(((¬_其 pre_R(P) ;; (SRD Y)[false/\$wait] \lor (\$ok' \land post_R(P) \land \$tr' >_u \$tr)) ;; (SRD Y)[false/\$wait]) \wedge gvt (Suc n))[[true,false/\$ok,\$wait]]

by (simp add: usubst unrest assms closure seqr-or-distl NSRD-neg-pre-left-zero SRD-healths)

also have ... =

(((¬_其 pre_R(P) ;; (SRD Y)[false/\$wait] \lor (post_R(P) \land \$tr' >_u \$tr')) ;; (SRD Y)[true,false/\$ok,\$wait]))
\wedge gvt (Suc n))[[true,false/\$ok,\$wait]]

proof -

have (\$ok' \land post_R(P) \land \$tr' >_u \$tr) ;; (SRD Y)[false/\$wait]

by (rel-blast)

also have ... = (post_R(P) \land \$tr' >_u \$tr)[true/$ok'] ;; (SRD Y)[false/\$wait][true/$ok]

using seqr-left-one-point[of ok (post_R(P) \land \$tr' >_u \$tr)] True (SRD Y)[false/\$wait]

by (simp add: true-ali-def[THEN sym])

finally show ?thesis by (simp add: usubst unrest)

qed

finally

show (P)[false/\$wait'] ;; (SRD Y)[false/\$wait] \wedge gvt (Suc n))[[true,false/$ok,$\$\text{wait}]] =

(((¬_其 pre_R(P) ;; (SRD Y)[false/\$wait] \lor (post_R(P) \land \$tr' >_u \$tr')) ;; (SRD Y)[true,false/$ok,$\$\text{wait}]))
\wedge gvt (Suc n))[[true,false/$ok,$\$\text{wait}]].

qed

have 1:(post_R(P) \land \$tr' >_u \$tr) ;; (SRD X)[true,false/$ok,$\$\text{wait}] \wedge gvt (Suc n)) =

((post_R(P) \land \$tr' >_u \$tr)) ;; (SRD (X \wedge gvt n))[[true,false/$ok,$\$\text{wait}]] \wedge gvt (Suc n))

apply (rel-auto)

apply (rename-rac tr st more ok wait tr' st' more' tr0 st0 more0 ok')

apply (rule-rac x=x0 in exl, rule-rac x=x0 in exl, rule-rac x=x0 in exl)

apply (simp)

apply (erule trace-strict-prefixE)

apply (rename-rac tr st ref ok wait tr' st' ref' tr0 st0 ref0 ok' zs)

apply (rule-rac x=False in exl)

apply (simp add: size-minus-trace)

apply (subgoal-rac size(tr) < size(tr0))

apply (simp add: less-diff-conv2 size-mono)

using size-strict-mono apply blast

apply (rename-rac tr st more ok wait tr' st' more' tr0 st0 more0 ok')

apply (rule-rac x=x0 in exl, rule-rac x=x0 in exl, rule-rac x=x0 in exl)

apply (simp)

apply (erule trace-strict-prefixE)

apply (rename-rac tr st more ok wait tr' st' more' tr0 st0 more0 ok' zs)

apply (auto simp add: size-minus-trace)

apply (subgoal-rac size(tr) < size(tr0))

apply (simp add: less-diff-conv2 size-mono)

using size-strict-mono apply blast

done

have 2:(¬_其 pre_R(P) ;; (SRD X)[false/\$wait] = (¬_其 pre_R(P) ;; (SRD(X \wedge gvt n))[[false/\$wait]

by (simp add: NSRD-neg-pre-left-zero closure assms SRD-healths)
show \( \text{thesis} \)
by (simp add: exp 1 2 utp-pred-laws.inf-sup-distrib2)
qed

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

also have ... = \( (P[true/\text{wait}] :: (SRD X \land \text{gvr} n))[true/\text{wait}] \land \text{gvr}(n+1))[true/false/\text{ok}/\text{wait}] \lor \)
    \( (P[false/\text{wait}] :: (SRD X \land \text{gvr} n))[false/\text{wait}] \land \text{gvr}(n+1))[true/false/\text{ok}/\text{wait}] \)
by (simp add: 1 2)

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

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

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

10.3 Tail recursive fixed-point calculations

declare upred-semiring.power-Suc [simp]

lemma mu-csp-form-1 [rdes]:
  fixes \( P :: (\text{s} \cdot \text{t} :: \text{size-trace}, \alpha) \) hrel-rsp
  assumes \( P \) is NSRD \( P \) is Productive
  shows \( \mu X \cdot P :: \text{SRD}(X) = (\bigcap i \cdot P \cdot^i (i+1)) :: \text{Miracle} \)
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 \([id]_H \to [\text{SRD}]_H \))
    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 ... = \( (\bigcap i \cdot ((\lambda X. P :: \text{SRD} X) \cdot^i (i+1)) \cdot \text{false}) \)
    by (simp add: sup-continuous-lfp 1 sup-continuous-Continuous false-upred-def)
  also have ... = \( (\bigcap i \cdot ((\lambda X. P :: \text{SRD} X) \cdot^i 0) \cdot \text{false} \cap (\bigcap i \cdot ((\lambda X. P :: \text{SRD} X) \cdot^i (i+1)) \cdot \text{false}) \)
    by (subst Sup-power-expand, simp)
also have ... = (\prod i. ((\lambda X. P ;; SRD X) ^^ (i+1)) false)
  by (simp)
also have ... = (\prod i. P ^ (i+1)) ;; Miracle
proof (rule SUP-cong, simp-all)
  fix i
  show P ;; SRD (((\lambda X. P ;; SRD X) ^^ i) false) = (P ;; P ^ i) ;; Miracle
  proof (induct i)
    case 0
    then show ?case
    by (simp, metis srdes-hcond-def srdes-theory-continuous.healthy-top)
  next
    case (Suc i)
    then show ?case
    by (simp add: Healthy-if NSRD-is-SRD SRD-power-comp SRD-seqr-closure assms
    seqr-assoc seqr-weak seqr-top-closed top-closed)
  qed
qed

also have ... = (\prod i. P ^ (i+1)) ;; Miracle
  by (simp add: seq-Sup-distr)
finally show ?thesis
  by (simp add: UINF-as-Sup THEN sym)
qed

lemma mu-csp-form-NSRD [closure]:
  fixes P :: ('s, 't::size-trace,'alpha) hrel-rsp
  assumes P is NSRD P is Productive
  shows (\mu X · P ;; SRD(X)) is NSRD
  by (simp add: mu-csp-form-1 assms closure)

lemma mu-csp-form-1':
  fixes P :: ('s, 't::size-trace,'alpha) hrel-rsp
  assumes P is NSRD P is Productive
  shows (\mu X · P ;; SRD(X)) = (P ;; P*) ;; Miracle
proof — 
  have (\mu X · P ;; SRD(X)) = (\prod i\in UNIV · P ;; P ^ i) ;; Miracle
  by (simp add: mu-csp-form-1 assms closure ustar-def)
also have ... = (P ;; P*) ;; Miracle
  by (simp only: seq-UINF-distl THEN sym, simp add: ustar-def)
finally show ?thesis.
qed

declare upred-semiring.power-Suc [simp del]

end

11 Reactive Design Programs

theory utp-rdes-prog
  imports
    utp-rdes-normal
    utp-rdes-tactics
    utp-rdes-parallel
    utp-rdes-guarded
    UTP−KAT.utp-kleene
begin
11.1 State substitution

**lemma srd-subst-RHS-tri-design [usubst]:**
\[ [\sigma]_S \uparrow R.(P \vdash Q \circ R) = R.s([\sigma]_S \uparrow P \vdash ([\sigma]_S \uparrow Q) \circ ([\sigma]_S \uparrow R)) \]
by (rel-auto)

**lemma srd-subst-SRD-closed [closure]:**
assumes \( P \) is SRD
shows \([\sigma]_S \uparrow P \) is SRD
proof
− have SRD([\sigma]_S \uparrow (SRD P)) = [\sigma]_S \uparrow (SRD P)
  by (rel-auto)
thus \(?thesis
  by (metis Healthy-def assms)
qed

**lemma preR-srd-subst [rdes]:**
\[ pre_R([\sigma]_S \uparrow P) = [\sigma]_S \uparrow pre_R(P) \]
by (rel-auto)

**lemma periR-srd-subst [rdes]:**
\[ peri_R([\sigma]_S \uparrow P) = [\sigma]_S \uparrow peri_R(P) \]
by (rel-auto)

**lemma postR-srd-subst [rdes]:**
\[ post_R([\sigma]_S \uparrow P) = [\sigma]_S \uparrow post_R(P) \]
by (rel-auto)

**lemma srd-subst-NSRD-closed [closure]:**
assumes \( P \) is NSRD
shows \([\sigma]_S \uparrow P \) 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 ((\cdot)_R) where**

[upred-defs]: assigns-srd \( \sigma \) = \( R.s(true \vdash (\$tr' =_{u} \$tr \wedge \neg \$wait' \wedge [(\sigma)_{a}]_S \wedge \$\Sigma_{S'} =_{u} \$\Sigma_{S}) \)

**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 \( x v \Leftarrow \) CONST assigns-srd (CONST subst-upd (CONST id) \( x v \))
- assigns-srd \( x v \Leftarrow \) -assign-srd (-spvar x) \( v \)
- \( x.y :=_{R} u,v \Leftarrow \) CONST assigns-srd (CONST subst-upd (CONST subst-upd (CONST id) (CONST svar x) u) (CONST svar y) \( v \))

**lemma assigns-srd-RHS-tri-des [rdes-def]:**
\( \langle \sigma \rangle_R = R.s(true_r \vdash false \circ \langle \sigma \rangle_r) \)
by (rel-auto)

**lemma assigns-srd-NSRD-closed [closure]: \( \langle \sigma \rangle_R \) is NSRD**
by (simp add: rdes-def closure unrest)
lemma \textit{preR-assigns-srd} [rdes]: \[
\text{pre}_R(\langle \sigma \rangle_R) = \text{true}_r
\]
by (simp add: rdes-def rdes closure)

lemma \textit{periR-assigns-srd} [rdes]: \[
\text{peri}_R(\langle \sigma \rangle_R) = \text{false}
\]
by (simp add: rdes-def rdes closure)

lemma \textit{postR-assigns-srd} [rdes]: \[
\text{post}_R(\langle \sigma \rangle_R) = \langle \sigma \rangle_r
\]
by (simp add: rdes-def rdes closure rpred)

11.3 Conditional

lemma \textit{preR-cond-srea} [rdes]:
\[
\text{pre}_R(P \triangleright b \triangleleft R Q) = (\langle b \rangle_{S<} \land \text{pre}_R(P) \lor \langle \neg b \rangle_{S<} \land \text{pre}_R(Q))
\]
by (rel-auto)

lemma \textit{periR-cond-srea} [rdes]:
assumes \(P\) is SRD \(Q\) is SRD
shows \(\text{peri}_R(P \triangleright b \triangleleft R Q) = (\langle b \rangle_{S<} \land \text{peri}_R(P) \lor \langle \neg b \rangle_{S<} \land \text{peri}_R(Q))\)
proof –
have \(\text{peri}_R(P \triangleright b \triangleleft R Q) = \text{peri}_R(R1(P) \triangleright b \triangleleft R1(Q))\)
by (simp add: Healthy-if SRD-healths assms)
thus \(?thesis\)
by (rel-auto)
qed

lemma \textit{postR-cond-srea} [rdes]:
assumes \(P\) is SRD \(Q\) is SRD
shows \(\text{post}_R(P \triangleright b \triangleleft R Q) = (\langle b \rangle_{S<} \land \text{post}_R(P) \lor \langle \neg b \rangle_{S<} \land \text{post}_R(Q))\)
proof –
have \(\text{post}_R(P \triangleright b \triangleleft R Q) = \text{post}_R(R1(P) \triangleright b \triangleleft R1(Q))\)
by (simp add: Healthy-if SRD-healths assms)
thus \(?thesis\)
by (rel-auto)
qed

lemma \textit{NSRD-cond-srea} [closure]:
assumes \(P\) is NSRD \(Q\) is NSRD
shows \(P \triangleright b \triangleleft R Q\) is SRD
proof (rule NSRD-RC-intro)
show \(P \triangleright b \triangleleft R Q\) is SRD
by (simp add: closure assms)
show \(\text{pre}_R(P \triangleright b \triangleleft R Q)\) is RC
proof –
have 1:([\neg b]_{S<} \lor \neg_{\text{r pre}} R P) ;; R1(true) = ([\neg b]_{S<} \lor \neg_{\text{r pre}} R P)
by (metis (no-types, lifting) NSRD-neg-pre-unit aext-not assms 1 seqr-or-distl st-lift-R1-true-right)
have 2:([b]_{S<} \lor \neg_{\text{r pre}} R P) ;; R1(true) = ([b]_{S<} \lor \neg_{\text{r pre}} R P)
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 \(\text{st}^* \not\in \text{peri}_R(P \triangleright b \triangleleft R Q)\)
by (simp add: rdes assms closure unrest)
qed

92
11.4 Assumptions

definition AssumeR :: \('s cond \Rightarrow ('s::trace, 't::trace, 'a) hrel-rsp ([|]\cdot R)\) where
[upred-defs]: AssumeR \(b = \Pi_R \triangleq b \triangleright_R \) Miracle

lemma AssumeR-rdes-def [rdes-def]:
[\(b\)]_R = \(R_a(true, \vdash false \triangleq [b]_R)\)
unfolding AssumeR-def by (rdes-eq)

lemma AssumeR-NSRD [closure]: [\(b\)]_R is NSRD
by (simp add: AssumeR-def closure)

lemma AssumeR-false: [\(false\)]_R = Miracle
by (rel-auto)

lemma AssumeR-true: [\(true\)]_R = II_R
by (rel-auto)

lemma AssumeR-comp: [\(b\)]_R ;; [\(c\)]_R = [\(b \land c\)]_R
by (rdes-simp)

lemma AssumeR-choice: [\(b\)]_R \cap [\(c\)]_R = [\(b \lor c\)]_R
by (rdes-eq)

lemma AssumeR-refine-skip: II_R \subseteq [\(b\)]_R
by (rdes-refine)

lemma AssumeR-test [closure]: test_R [\(b\)]_R
by (simp add: AssumeR-refine-skip nsrd-thy.ustest-intro)

lemma Star-AssumeR: [\(b\)]_R* = II_R
by (simp add: AssumeR-NSRD AssumeR-test nsrd-thy.Star-test)

lemma AssumeR-NSRD [closure]: [\(b\)]_R is NSRD
by (simp add: AssumeR-refine-skip nsrd-thy.ustest-intro)

lemma AssumeR-test [closure]: test_R [\(b\)]_R
by (simp add: AssumeR-refine-skip nsrd-thy.ustest-intro)

lemma Star-AssumeR: [\(b\)]_R* = II_R
by (simp add: AssumeR-NSRD AssumeR-test nsrd-thy.Star-test)

lemma AssumeR-choice-skip: II_R \cap [\(b\)]_R = II_R
by (rdes-eq)

lemma cond-srea-AssumeR-form:
assumes \(P\) is NSRD \(Q\) is NSRD
shows \(P \triangleq b \triangleright_R Q = ([\(b\)]_R ;; P \cap [\neg b]_R ;; Q)\)
by (rdes-eq cls: assms)

lemma cond-srea-insert-assume:
assumes \(P\) is NSRD \(Q\) is NSRD
shows \(P \circ b \triangleright_R Q = ([\(b\)]_R ;; P \circ b \triangleright_R [\neg b]_R ;; Q)\)
by (simp add: AssumeR-NSRD AssumeR-comp NSRD-seqr-closure RA1 assms cond-srea-AssumeR-form)

lemma AssumeR-cond-left:
assumes \(P\) is NSRD \(Q\) is NSRD
shows [\(b\)]_R ;; (P \circ b \triangleright_R Q) = ([\(b\)]_R ;; P)
by (rdes-eq cls: assms)

lemma AssumeR-cond-right:
assumes \(P\) is NSRD \(Q\) is NSRD
shows [\(\neg b\)]_R ;; (P \circ b \triangleright_R Q) = ([\neg b]_R ;; Q)
by (rdes-eq cls: assms)
11.5 Guarded commands

definition GuardedCommR :: 's cond ⇒ ('s, 't::trace, 'α) hrel-rsp ⇒ ('s, 't, 'α) hrel-rsp (-→R - [85, 86]) where
gcmd-def[rdes-def]: GuardedCommR g A = A ⊓ g ⊢R Miracle

lemma gcmd-false[simp]: (false →R A) = Miracle
  unfolding gcmd-def by (pred-auto)

lemma gcmd-true[simp]: (true →R A) = A
  unfolding gcmd-def by (pred-auto)

lemma gcmd-SRD: assumes A is SRD shows (g →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 →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 →R A) is Productive
  by (simp add: gcmd-def closure assms)

lemma gcmd-seq-distr: assumes B is NSRD shows (g →R A) ;; B = (g →R A) ;; (g →R B)
  by (simp add: Miracle-left-zero NSRD-is-SRD assms cond-st-distr gcmd-def)

lemma gcmd-nondet-distr: assumes A is NSRD B is NSRD shows (g →R (A ⊓ B)) = (g →R A) ⊓ (g →R B)
  by (rdes-eq cls: assms)

lemma AssumeR-as-gcmd: [b] ⊢R = b →R II
  by (rdes-eq)

11.6 Generalised Alternation

definition AlternateR :: 'a set ⇒ ('a ⇒ 's upred) ⇒ ('a ⇒ ('s, 't::trace, 'α) hrel-rsp) ⇒ ('s, 't, 'α) hrel-rsp ⇒ ('s, 't, 'α) hrel-rsp where
  [upred-defs, rdes-def]: AlternateR I g A B = (⨍ i ∈ I · ((g i) →R (A i))) ⊓ ((¬ (∨ i ∈ I · g i)) →R B)

definition AlternateR-list :: ('s upred × ('s, 't::trace, 'α) hrel-rsp) list ⇒ ('s, 't, 'α) hrel-rsp ⇒ ('s, 't, 'α) hrel-rsp where
  [upred-defs, ndes-simp]: AlternateR-list xs P = AlternateR {0..<length xs} (λ i. map fst xs ! i) (λ i. map snd xs ! i) P

syntax
- altindR-els :: ptn ⇒ logic ⇒ logic ⇒ logic ⇒ logic ⇒ logic (if_R -· · · - else - f)
-altindR :: ptrn ⇒ logic ⇒ logic ⇒ logic ⇒ logic (if _R ∈ · · · → _fi)

-altgcommR-els :: gcomms ⇒ logic ⇒ logic ⇒ logic ⇒ logic (if _R/ else _fi)
-altgcommR :: gcomms ⇒ logic (if _R/ _fi)

translations
if _R i ∈ I · _g → A else _fi
CONST AlternateR I (_λi. _g) (_λi. A) _B
if _R i ∈ I · _g i → A else _fi
CONST AlternateR I _g (_λi. A) _B
-altgcommR cs → CONST AlternateR-list cs (CONST Chaos)
-altgcommR (-gcomm-show cs) ← CONST AlternateR-list cs (CONST Chaos)
-altgcommR-els cs _P → CONST AlternateR-list cs _P
-altgcommR-els (-gcomm-show cs) _P ← CONST AlternateR-list cs _P

lemma AlternateR-NSRD-closed [closure]:
  assumes \( \bigwedge_i. i \in I \implies A i \text{ is NSRD } B \text{ is NSRD} \)
  shows (if _R i ∈ I · _g i → A i else _fi) is NSRD
proof (cases I = {})
  case True
  then show ?thesis by (simp add: AlternateR-def assms)
next
  case False
  then show ?thesis by (simp add: AlternateR-def closure assms)
qed

lemma AlternateR-empty [simp]:
  (if _R i ∈ {} · _g i → A i else _fi) = _B
by (rdes-simp)

lemma AlternateR-Productive [closure]:
  assumes \( \bigwedge_i. i \in I \implies A i \text{ is Productive } B \text{ is Productive} \)
  shows (if _R i ∈ I · _g i → A i else _fi) is Productive
proof (cases I = {})
  case True
  then show ?thesis
    by (simp add: assms(4))
next
  case False
  then show ?thesis
    by (simp add: AlternateR-def closure assms)
qed

lemma AlternateR-singleton:
  assumes _A k is NSRD _B is NSRD
  shows (if _R i ∈ \{k\} · _g i → A i else _fi) = (A(k) ∩ g(k) ⊃ _R _B)
by (simp add: AlternateR-def, rdes-eq cls: assms)

Convert an alternation over disjoint guards into a cascading if-then-else

lemma AlternateR-insert-cascade:
  assumes \( \bigwedge_i. i \in I \implies A i \text{ is NSRD} \)
  \( A k \text{ is NSRD } B \text{ is NSRD} \)
  \( (g(k) \land (\forall i\in I \cdot g(i))) = \text{false} \)
shows \((\text{if } R_i \in \text{insert } k I \cdot g i \to A \text{ i else } B \text{ fi}) = (A(k) \circ g(k) \triangleright_R (\text{if } R_i \in I \cdot g(i) \to A(i) \text{ else } B \text{ fi}))\)

proof (cases \(I = \{\}\))

case True
then show \(\text{thesis} \) by (simp add: AlternateR-singleton assms)

next

case False
have 1: \((\bigcap i \in I \cdot g i \to R A i) = (\bigcap i \in I \cdot g i \to R R s (\text{pre } R(A i) \vdash \text{peri } R(A i) \diamond \text{post } R(A i)))\)
by (simp add: NSRD-is-SRD SRD-reactive-tri-design assms (1) cong: UINF-cong)

from assms (4) show \(?thesis \) by (simp add: AlternateR-def 1 False cong: UINF-cong)

qed

11.7 Choose

definition choose-srd :: \('s, 't::trace, 'α) hrel-rsp (choose \(_R\)) \ where
\([\text{upred-defs}, \ rdes-def]: \text{choose} \(_R\) = R_s (\text{true} \r \vdash \text{false} \circ \text{true}_r)\)

lemma preR-choose [rdes]: \(\text{pre}_R(\text{choose} \(_R\)) = \text{true}_r\)
by (rel-auto)

lemma periR-choose [rdes]: \(\text{peri}_R(\text{choose} \(_R\)) = \text{false}\)
by (rel-auto)

lemma postR-choose [rdes]: \(\text{post}_R(\text{choose} \(_R\)) = \text{true}_r\)
by (rel-auto)

lemma choose-srd-SRD [closure]: \(\text{choose} \(_R\) \text{ is SRD}\)
by (simp add: choose-srd-def closure unrest)

lemma NSRD-choose-srd [closure]: \(\text{choose} \(_R\) \text{ is NSRD}\)
by (rule NSRD-intro, simp-all add: closure unrest rdes)

11.8 State Abstraction

definition state-srea :: \('s itself \Rightarrow \text{type} \Rightarrow \text{logic} \Rightarrow \text{logic} (\text{state} \cdot - \cdot [0,200]) \Rightarrow \text{logic}\)

syntax
state \('a \cdot P == CONST state-srea TYPE('a) P

translations
state \('a \cdot P == CONSTR state-srea TYPE('a) P

lemma R1-state-srea: R1(state \('a \cdot P) = (state \('a \cdot R1(P))
by (rel-auto)

lemma R2c-state-srea: R2c(state \('a \cdot P) = (state \('a \cdot R2c(P))
by (rel-auto)

lemma R3h-state-srea: R3h(state \('a \cdot P) = (state \('a \cdot R3h(P))
by (rel-auto)

lemma RD1-state-srea: RD1(state \('a \cdot P) = (state \('a \cdot RD1(P))
by (rel-auto)

96
lemma \( RD2\text{-}state\text{-}srea\): \( RD2(state \ 'a \cdot P) = (state \ 'a \cdot RD2(P)) \)
by (rel-auto)

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

lemma \( SRD\text{-}state\text{-}srea \) [closure]: \( P \text{ is } SRD \implies state \ 'a \cdot P \text{ is } SRD \)
by (simp add: Healthy-def R1-state-srea R2c-state-srea R3h-state-srea RD1-state-srea RD2-state-srea
RHS-def SRD-def)

lemma \( NSRD\text{-}state\text{-}srea \) [closure]: \( P \text{ is } NSRD \implies state \ 'a \cdot P \text{ is } NSRD \)
by (metis Healthy-def NSRD-is-RD3 NSRD-is-SRD RD3-state-srea RD3-state-srea RD3-implies-NSRD SRD-state-srea)

lemma \( preR\text{-}state\text{-}srea \) [rdes]: \( pre_R(state \ 'a \cdot P) = \langle \forall \{\$st,\$st'\} \cdot pre_R(P)\rangle_S \)
by (simp add: state-def, rel-auto)

lemma \( periR\text{-}state\text{-}srea \) [rdes]: \( peri_R(state \ 'a \cdot P) = state \ 'a \cdot peri_R(P) \)
by (rel-auto)

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

11.9 While Loop

definition \( WhileR :: 's upred \Rightarrow ('s, 't::size-trace, 'a) hrel-rsp \Rightarrow ('s, 't, 'a) hrel-rsp \) (where \( R:\text{-}do\text{-}od \))
where
\( WhileR \ b \ P = (\mu_R X \cdot (P ;; X) \triangleleft b \triangleright_R II_R) \)

lemma \( Sup\text{-}power\text{-}false\):
fixes \( F :: 'a upred \Rightarrow 'a upred \)
shows \( (\prod i \cdot (F ^^ i) \text{ false}) = (\prod i \cdot (F ^^ (i+1)) \text{ false}) \)
proof
  have \( (\prod i \cdot (F ^^ i) \text{ false}) = (F ^^ 0) \text{ false} \cap (\prod i \cdot (F ^^ (i+1)) \text{ false}) \)
  by (subst Sup-power-expand, simp)
  also have \( ... = (\prod i \cdot (F ^^ (i+1)) \text{ false}) \)
  by (simp)
  finally show \( ?thesis \).
qed

theorem \( WhileR\text{-}iter\text{-}expand\):
assumes \( P \text{ is } NSRD P \text{ is } Productive \)
shows \( while_R b \text{ do } P \text{ od} = (\prod i \cdot (P \triangleleft b \triangleright_R II_R) ^ \cdot i ;; (P ;; \text{ Miracle} \triangleleft b \triangleright_R II_R)) \) (is \( ?lhs = ?rhs \))
proof
  have \( 1: \text{Continuous} \quad (\lambda X. \ P ;; SRD \ X) \)
  using SRD-Continuous
  by (clarsimp simp add: Continuous-def seq-SUP-distl[THEN sym], drule-tac x=A in spec, simp)
  have \( 2: \text{Continuous} \quad (\lambda X. \ P ;; SRD \ X \triangleleft b \triangleright_R II_R) \)
  by (simp add: 1 closure assms)
  have \( ?lhs = (\mu_R \ P ;; X \triangleleft b \triangleright_R II_R) \)
  by (simp add: WhileR-def)
  also have \( ... = (\mu X \cdot P ;; SRD(X) \triangleleft b \triangleright_R II_R) \)
  by (auto simp add: srd-mu-equiv closure assms)
  also have \( ... = (\nu X \cdot P ;; SRD(X) \triangleleft b \triangleright_R II_R) \)
  by (auto simp add: guarded-fp-uniq Guarded-if-Productive[OF assms] funcsetI closure assms)
  also have \( ... = (\prod i \cdot ((\lambda X. \ P ;; SRD \ X \triangleleft b \triangleright_R II_R) ^ \cdot i) \text{ false}) \)

97
by (simp add: sup-continuous-lfp 2 sup-continuous-Continuous false-upred-def)
also have \ldots = (\bigcap i \cdot (\lambda X. P \bowtie SRD X \bowtie b \bowtie II_{R} (i + 1)) \text{false})
by (simp add: Sup-power-false)
also have \ldots = (\bigcap i \cdot (P \bowtie b \bowtie II_{R}) \bowtie i ;; (P ;; Miracle \bowtie b \bowtie II_{R}))
proof (rule SUP-cong, simp)
fix \(i\)
show \((\lambda X. P ;; SRD X \bowtie b \bowtie II_{R}) \bowtie (i + 1)\) \text{false} = (P \bowtie b \bowtie II_{R}) \bowtie i ;; (P ;; Miracle \bowtie b \bowtie II_{R})
proof (induct \(i\))
case \(\emptyset\)
thesis
then show \(?\case\)
proof by (simp,metis srdes-hcond-def srdes-theory-continuous.top-closed)
next
case \((\text{Suc } i)\)
show \(?\case\)
proof (induct \(i\))
case \(\emptyset\)
then show \(?\case\)
proof by (simp add: NSRD-is-SRD SRD-cond-srea SRD-left-unit SRD-seqr-closure SRD-srdes-skip assesms(1) power-power-eq-if seqr-left-unit srdes-theory-continuous.top-closed)
also have \ldots = (P \bowtie b \bowtie II_{R}) \bowtie \text{Suc } i ;; (P ;; Miracle \bowtie b \bowtie II_{R})
proof (induct \(i\))
case \(\emptyset\)
then show \(?\case\)
proof by (simp add: RA1 upred-semiring.power-Suc)
qed
qed
finally show \(?\thesis\).
qed
qed
also have \ldots = (\bigcap i \cdot (P \bowtie b \bowtie II_{R}) \bowtie i ;; (P ;; Miracle \bowtie b \bowtie II_{R}))
by (simp add: UNF-as-Sup-collect')
finally show \(?\thesis\).
qed

theorem WhileR-star-expand;
assumes \(P \text{ is NSRD } P \text{ is Productive}\)
shows while$_R$ b do P od = (P $\lhd b \triangleright_R$ H$_R$)$^R$ ;; (P ;; Miracle $\lhd b \triangleright_R$ H$_R$) (is $\exists$lhs = $\exists$rhs)

proof 

have $\exists$hs = ([$\prod$ i * (P $\lhd b \triangleright_R$ H$_R$) $\ell$ i) ;; (P ;; Miracle $\lhd b \triangleright_R$ H$_R$)

by (simp add: WhileR-iter-expand seq-UNF-distr' assms)
also have ... = (P $\lhd b \triangleright_R$ H$_R$)$^R$ ;; (P ;; Miracle $\lhd b \triangleright_R$ H$_R$)
by (simp add: ustar-def)
also have ... = ((P $\lhd b \triangleright_R$ H$_R$)$^R$ ;; H$_R$) ;; (P ;; Miracle $\lhd b \triangleright_R$ H$_R$)
by (simp add: seq-assoc SRD-left-unit closure assms)
also have ... = (P $\lhd b \triangleright_R$ H$_R$)$^R$ ;; (P ;; Miracle $\lhd b \triangleright_R$ H$_R$)
by (simp add: nsrd-thy.Star-def)
finally show $\exists$thesis.

qed

lemma WhileR-NSRD-closed [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 $\lhd b \triangleright_R$ H$_R$)$^R$ ;; (P ;; Miracle $\lhd b \triangleright_R$ H$_R$) = ([$b]^R_R ;; P)$^R$ ;; ($\neg$ b)$^R_R$

proof 

have (P $\lhd b \triangleright_R$ H$_R$)$^R$ ;; (P ;; Miracle $\lhd b \triangleright_R$ H$_R$) = ([$b]^R_R ;; P \sqcap \neg [b]^R_R)$^R$ ;; (P ;; Miracle $\lhd b \triangleright_R$ H$_R$)

by (simp add: AssumeR-NSRD NSRD-right-unit NSRD-srd-skip assms(1) cond-srba-AssumeR-form)
also have ... = ([$b]^R_R ;; P)$^R$ ;; ($\neg$ b)$^R_R)$^R ;; (P ;; Miracle $\lhd b \triangleright_R$ H$_R$)
by (simp add: AssumeR-NSRD NSRD-seq-closure nsrd-thy.Star-distrib assms(1))
also have ... = ([$b]^R_R ;; P)$^R$ ;; ([$b]^R_R ;; P ;; Miracle $\lhd b \triangleright_R$ H$_R$)
by (simp add: AssumeR-NSRD NSRD-seq-closure nsrd-thy.Star-invol assms(1))
also have ... = ([$b]^R_R ;; P)$^R$ ;; ([$b]^R_R ;; P ;; Miracle $\lhd b \triangleright_R$ H$_R$)
by (simp add: AssumeR-NSRD NSRD-Miracle NSRD-right-unit NSRD-seq-closure NSRD-srd-skip assms(1) cond-srba-AssumeR-form)
also have ... = ([$b]^R_R ;; P)$^R$ ;; ([$b]^R_R ;; P ;; Miracle $\lhd b \triangleright_R$ H$_R$) ;; ($\neg$ b)$^R_R$
by (simp add: upred-semiring.distrib-left)
also have ... = ([$b]^R_R ;; P)$^R$ ;; ($\neg$ b)$^R_R$

proof 

have ([$b]^R_R ;; P)$^R$ ;; ($\neg$ b)$^R_R$ = (H$_R$ $\sqcap$ ($[$b]$^R_R ;; P)$^R ;; ($\neg$ b)$^R_R$) ;; (P ;; Miracle $\lhd b \triangleright_R$ H$_R$)
by (simp add: AssumeR-NSRD NSRD-seq-closure nsrd-thy.Star-unfold-r EQ assms(1))
also have ... = ($\neg$ b)$^R_R$ $\sqcap$ ([$b]^R_R ;; P)$^R$ ;; ($\neg$ b)$^R_R$ ;; (P ;; Miracle $\lhd b \triangleright_R$ H$_R$)
by (metis (no-types, lifting) AssumeR-NSRD AssumeR-as-gcmn NSRD-srd-skip Star-AssumeR nsrd-thy.Star-slide gcmn-seq-distr skip-srba-self-unit urel-doiroid.distrib-right)
also have ... = ($\neg$ b)$^R_R$ $\sqcap$ ([$b]^R_R ;; P)$^R$ ;; ([$b]^R_R ;; P ;; [b $\lor \neg$ b)$^R_R$) ;; ($\neg$ b)$^R_R$
by (simp add: AssumeR-true NSRD-right-unit assms(1))
also have ... = ($\neg$ b)$^R_R$ $\sqcap$ ([$b]^R_R ;; P)$^R$ ;; ($\neg$ b)$^R_R$ $\sqcap$ ($[$b]$^R_R ;; P)$^R ;; ($\neg$ b)$^R_R$
by (metis (no-types, hide-lams) AssumeR-choice upred-semiring.add-assoc upred-semiring.distrib-left upred-semiring.distrib-right)
also have ... = ($\neg$ b)$^R_R$ $\sqcap$ ([$b]^R_R ;; P)$^R$ ;; ([$b]^R_R ;; P ;; [b $\lor \neg$ b)$^R_R$) $\sqcap$ ($[$b]$^R_R ;; P)$^R ;; ($\neg$ b)$^R_R$
by (simp add: RAI)
also have ... = ($\neg$ b)$^R_R$ $\sqcap$ ([$b]^R_R ;; P)$^R$ ;; ([$b]^R_R ;; P ;; Miracle)$
by (simp add: AssumeR-comp AssumeR-false)
finally have \((b^T_R :: P)^*R :: [\sim b]^T_R \sqsubseteq ((b^T_R :: P)^*R) :: [b]^T_R :: P :: \text{Miracle})

by (simp add: semilattice-sup-class.le-supI1)

thus ?thesis

by (simp add: semilattice-sup-class.le-iff-sup)

qed

finally show ?thesis.

qed

theorem WhileR-iter-form:
assumes \(P\) is NSRD \(P\) is Productive
shows \(\text{while}_R b \, \text{do} \, P \, \text{od} = ([b]^T_R :: P)^*R :: [\sim b]^T_R\)
by (simp add: WhileR-iter-form-lemma WhileR-star-expand assms)

theorem WhileR-false:
assumes \(P\) is NSRD
shows \(\text{while}_R \text{false} \, \text{do} \, P \, \text{od} = \Pi_R\)
by (simp add: WhileR-def rpred closure srdes-theory-continuous.LFP-const)

theorem WhileR-true:
assumes \(P\) is NSRD \(P\) is Productive
shows \(\text{while}_R \text{true} \, \text{do} \, P \, \text{od} = P^*R :: [\sim b]^T_R\)
by (simp add: WhileR-iter-form AssumeR-true AssumeR-false SRD-left-unit ass ms closure)

lemma WhileR-insert-assume:
assumes \(P\) is NSRD \(P\) is Productive
shows \(\text{while}_R b \, \text{do} \, ((b)^T_R :: P) \, \text{od} = \text{while}_R b \, \text{do} \, P \, \text{od}\)
by (simp add: AssumeR-NSRD AssumeR-comp NSRD-seq-closure Productive-seq-2 RA1 WhileR-iter-form assms)

theorem WhileR-rdes-def [rdes-def]:
assumes \(P\) is RC \(Q\) is RR \(R\) is RR
\(\text{st} \# Q\) \(Q_2\) is RR \((\# Q_2)\) is R4
shows \(\text{while}_R b \, \text{do} \, R_s(P \vdash Q \diamond R) \, \text{od} = \nu R_s \omega\)
\((b)^T_r :: R^*r \wedge_r ((b)^T_r :: R^*r) :: [b]^T_r :: Q \diamond ((b)^T_r :: R^*r) :: [\sim b]^T_r\)
(is ?lhs = ?rhs)

proof
have ?lhs = \((b)^T_R :: R_s(P \vdash Q \diamond R))^*r :: [\sim b]^T_R
  by (simp add: WhileR-iter-form Productive-rdes-RR-intro assms closure)
also have " = ?rhs
  by (simp add: rdes-def assms closure unrest rpred wp del: rea-star-wp)
finally show ?thesis .

qed

Refinement introduction law for reactive while loops

theorem WhileR-refine-intro:
assumes \(Q_1\) is RC \(Q_2\) is RR \(Q_3\) is RR \(\text{st} \# Q_2\) \(Q_3\) is R4
\(\text{Refinement conditions}\)
\(((b)^T_r :: Q_3)^*r \wedge r ((b)^T_S < r, Q_1) \subseteq P_1\)
\(P_2 \sqsubseteq [b]^T_r :: Q_2\)
\(P_2 \sqsubseteq [b]^T_r :: Q_3 :: P_2\)
\(P_3 \sqsubseteq [\sim b]^T_r\)
\(P_3 \sqsubseteq [b]^T_r :: Q_3 :: P_3\)
shows \(R_s(P_1 \vdash P_2 \diamond P_3) \sqsubseteq \text{while}_R b \, \text{do} \, R_s(Q_1 \vdash Q_2 \diamond Q_3) \, \text{od}\)
proof (simp add: rdes-def assms, rule srdes-tri-refine-intro)
show \((b)^+_r; Q_3)^r\ wp_r ((b) s \Rightarrow_r Q_1) \subseteq P_1
by (simp add: assms)

show \(P_2 \subseteq (P_1 \land ((b)^+_r; Q_3)^r; [b]^+_r; Q_2)\)
proof −
have \(P_2 \subseteq ([b]^+_r; Q_3)^r; [b]^+_r; Q_2)\)
by (simp add: assms rea-assume-RR rrel-thy Star-inductl seq-RR-closed seqr-assoc)
thus \(?thesis\)
by (simp add: utp-pred-laws.le-invl2)
qed

11.10 Iteration Construction

definition IterateR
:: 'a set \Rightarrow ('a \Rightarrow 's upred) \Rightarrow ('a \Rightarrow ('s, 't::size-trace, 'α) hrel-rsp) \Rightarrow ('s, 't, 'α) hrel-rsp
where IterateR A g P = while_R (\forall i \in A \cdot g(i)) do (\if_R i \in A \cdot g(i) \Rightarrow P(i) \fi) od

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

syntax
- iter-srd :: ptrn \Rightarrow logic \Rightarrow logic \Rightarrow logic \Rightarrow logic (do_R - \cdots - \_)
- iter-gcommR :: gcomm \Rightarrow logic (do_R/ - / od)

translations
- iter-srd x A g P => CONST IterateR A (\lambda x. g) (\lambda x. P)
- iter-srd x A g P <= CONST IterateR A (\lambda x. g) (\lambda x'. P)
- iter-gcommR cs => CONST IterateR-list cs
- iter-gcommR (-gcomm-show cs) <= CONST IterateR-list cs

lemma IterateR-NSRD-closed [closure]:
assumes
\(\forall i. i \in I \Rightarrow P(i) \text{ is NSRD}\)
\(\forall i. i \in I \Rightarrow P(i) \text{ is Productive}\)
shows \(do_R \ i \in I \cdot g(i) \Rightarrow P(i) \text{ fi is NSRD}\)
by (simp add: IterateR-def closure assms)

lemma IterateR-empty:
\(do_R \ i \in \{\} \cdot g(i) \Rightarrow P(i) \text{ fi = II}_R\)
by (simp add: IterateR-def srd-mu-equiv closure rpred gfp-const WhileR-false)

lemma IterateR-singleton:
assumes \(P \text{ k is NSRD} \ P \text{ k is Productive}\)
shows \(do_R \ i \in \{k\} \cdot g(i) \Rightarrow P(i) \text{ fi = while}_R g(k) \text{ do P(k) od (is } ?lhs = ?rhs)\)
proof −
have \(?lhs = \text{ while}_R g \text{ k do P k < g k } \triangleright_R \text{ Chaos od}\)
by (simp add: IterateR-def AlternateR-singleton assms closure)

101
also have \( \ldots = \text{while} R \ g \ k \ \text{do} \ [g \ k]^\top_R ;; (P \ k < g \ k \triangleright_R \text{Chaos}) \ \text{od} \)

by (simp add: WhileR-insert-assume closure assms)

also have \( \ldots = \text{while} R \ g \ k \ \text{do} \ P \ k \ \text{od} \)

by (simp add: AssumeR-cond-left NSRD-Chaos WhileR-insert-assume assms)

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

qed

|
| declare IterateR-list-def [rdes-def] |
| declare IterateR-def [rdes-def] |
| method unfold-iteration = simp add: IterateR-list-def IterateR-def AlternateR-list-def AlternateR-def UINF-upto-expand-first |

11.11 Substitution Laws

| lemma srd-subst-Chaos [usubst]: |
| \( \sigma \uparrow_S \text{Chaos} = \text{Chaos} \) |
| by (rdes-simp) |

| lemma srd-subst-Miracle [usubst]: |
| \( \sigma \uparrow_S \text{Miracle} = \text{Miracle} \) |
| by (rdes-simp) |

| lemma srd-subst-skip [usubst]: |
| \( \sigma \uparrow_S \text{II} R = \langle \sigma \rangle_R \) |
| by (rdes-eq) |

| lemma srd-subst-assigns [usubst]: |
| \( \sigma \uparrow_S \langle \hat{\nu} \rangle_R = \langle \hat{\nu} \diamond \sigma \rangle_R \) |
| by (rdes-eq) |

11.12 Algebraic Laws

| theorem assigns-srd-id: \( \langle \text{id} \rangle_R = \text{II} R \) |
| by (rdes-eq) |

| theorem assigns-srd-comp: \( \langle \sigma \rangle_R ;; \langle \hat{\nu} \rangle_R = \langle \hat{\nu} \diamond \sigma \rangle_R \) |
| 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: \( \langle \sigma \rangle_R \triangleleft b \triangleright_R \langle \hat{\nu} \rangle_R = \langle \sigma \triangleleft b \triangleright_s \hat{\nu} \rangle_R \) |
| 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 \( \bigwedge i. \; A \; \text{i is NSRD} \; B \; \text{i is NSRD} \; C \; \text{i is NSRD} \) |
| shows \( \langle \text{if} R \ i \in I \cdot g \ i \rightarrow A \ i \; \text{else} \; B \; \text{fi} \rangle ;; C = \langle \text{if} R \ i \in I \cdot g \ i \rightarrow A \ i ;; C \; \text{else} \; B ;; C \; \text{fi} \rangle \) |
proof (cases \( I = \{\} \))
  case True
  then show ?thesis by (simp)
next
  case False
  then show ?thesis
    by (simp add: AlternateR-def upred-semiring.distrib-right seq-UINF-distr gcmd-seq-distr assms (3))
qed

lemma AlternateR-is-cond-srea:
  assumes A is NSRD B is NSRD
  shows (if \( R \in [a] \cdot g \rightarrow A \) else \( B \) fi) = (\( A \triangleright g \triangleleft B \))
  by (rdes-eq cls: assms)

lemma AlternateR-Chaos:
  if \( R \in A \cdot g(i) \rightarrow Chaos \) = Chaos
  by (cases \( A = \{\} \), simp, rdes-eq)

lemma choose-srd-par:
  choose \( \parallel R \parallel \) choose \( R \parallel \) choose \( R \parallel \) choose \( R \parallel \)
  by (rdes-eq)

11.13 Lifting designs to reactive designs

definition des-rea-lift \( 's hrel-des \Rightarrow ('s,'t::trace,'a) hrel-rsp (R_D) \) where
  [upred-defs]: \( R_D(P) = R_s (\lceil \text{pre}_D(P) \rceil_S \vdash (false \circ (\text{str'} =_u \text{str} \land [post_D(P)]_S)) \)

definition des-rea-drop \( ('s,'t::trace,'a) hrel-rsp \Rightarrow 's hrel-des (D_R) \) where
  [upred-defs]: \( D_R(P) = [\langle \text{pre}_R(P) \rangle[\text{str}/\text{str'}] [\nu \text{str},\text{str'}]_S < \nu [\langle \text{post}_R(P) \rangle[\text{str}/\text{str'}] [\nu \text{str},\text{str'}]_S \)

lemma ndesign-rea-lift-inverse: \( D_R(R_D(P) \vdash_n Q) = p \vdash_n Q \)
  apply (simp add: des-rea-lift-def des-rea-drop-def rea-pre-RHS-design rea-post-RHS-design)
  apply (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 \( \?RP([\text{pre}_D(P)]_S \vdash_n \text{post}_D(P)) = \?RQ([\text{pre}_D(Q)]_S \vdash_n \text{post}_D(Q)) \)
    by (simp add: ndesign-form assms)
  hence \( [\text{pre}_D(P)]_S \vdash_n \text{post}_D(P) = [\text{pre}_D(Q)]_S \vdash_n \text{post}_D(Q) \)
    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]:
  \( \text{pre}_R(R_D(P)) = R1([\langle \text{pre}_D(P) \rangle]_S) \)
  by (rel-auto)
lemma periR-des-rea-lift [rdes]:
peri(RD(P)) = (false \land \lceil \text{pre}_D(P) \rceil_S \triangleright (\$tr \leq_u \$tr')
by (rel-auto)

lemma postR-des-rea-lift [rdes]:
post(RD(P)) = ((true \land \lceil \text{pre}_D(P) \rceil_S \triangleright (\neg \$tr \leq_u \$tr') \Rightarrow (\$tr' =_u \$tr \land \lceil \text{post}_D(P) \rceil_S))
apply (rel-auto) using minus-zero-eq by blast

lemma ndes-rea-lift-closure [closure]:
assumes P is N
shows RD(P) is NSRD
proof -
  obtain p Q where P: P = (p \vdash_n Q)
  by (metis H1-H3-commute H1-H3-is-normal-design H1-idem Healthy-def assms)
  show \thesis
  apply (rule NSRD-intro)
  apply (simp-all add: closure rdes unrest P)
  apply (rel-auto)
done
qed

lemma R-D-mono:
assumes P is H Q is H P \sqsubseteq Q
shows RD(P) \sqsubseteq RD(Q)
apply (simp add: des-rea-lift-def)
apply (rule srdes-tri-refine-intro')
apply (auto intro: H1-H2-refines assms aext-mono)
apply (rel-auto)
apply (metis (no-types, hide-lams) aext-mono assms (3) design-post-choice
  semilattice-sup-class.sup.orderE utp-pred-laws.inf.coboundedH1 utp-pred-laws.inf.commute utp-pred-laws.sup.order-iff
  done

Homomorphism laws

lemma R-D-Miracle:
RD(\top_D) = Miracle
by (simp add: Miracle-def, rel-auto)

lemma R-D-Chaos:
RD(\bot_D) = Chaos
proof -
  have RD(\bot_D) = RD(false \vdash true)
  by (rel-auto)
  also have ... = RD(false \vdash false \circ (\$tr' =_u \$tr))
  by (simp add: Chaos-def des-rea-lift-def alpha)
  also have ... = RD(true)
  by (rel-auto)
  also have ... = Chaos
  by (simp add: Chaos-def design-false-pre)
finally show \thesis.
qed

lemma R-D-inf:
RD(P \cap Q) = RD(P) \cap RD(Q)
by (rule antisym, rel-auto+)

104
lemma R-D-cond:
\[ R_D(P \triangleleft [b] D < \triangleright Q) = R_D(P) \triangleleft b \triangleright R_D(Q) \]
by (rule antisym, rel-auto+)

lemma R-D-seq-ndesign:
\[ R_D(p_1 \vdash_n Q_1) \sqcup R_D(p_2 \vdash_n Q_2) = R_D((p_1 \vdash_n (Q_1) \sqcup (p_2 \vdash_n Q_2)) \]
apply (rule antisym)
apply (rule SRD-refine-intro)
apply (simp-all add: closure rdes ndesign-composition-up)
using dual-order. trans apply (rel-blast)
apply (rel-auto)
done

lemma R-D-seq:
assumes P is N Q is N
shows R_D(P \sqcup Q) = R_D(P ;; Q)
by (metis R-D-seq-ndesign assms ndesign-form)

These laws are applicable only when there is no further alphabet extension

lemma R-D-skip:
\[ R_D(II_D) = (II_R ; (s::trace,unit) hrel-rsp) \]
apply (rel-auto) using minus-zero-eq by blast+

lemma R-D-assigns:
\[ R_D(\langle \sigma \rangle_D) = (\langle \sigma \rangle_R ; (s::trace,unit) hrel-rsp) \]
by (simp add: assigns-d-def des-rea-lift-def alpha assigns-srd-RHS-tri-des, rel-auto)

end

12 Instantaneous Reactive Designs

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

definition ISRD1 :: (s::trace,α) hrel-rsp ⇒ (s::trace,α) hrel-rsp where
[upred-defs]: ISRD1(P) = P \parallel R_s(true \vdash false \odot (\$tr' = \$tr))

definition ISRD :: (s::trace,α) hrel-rsp ⇒ (s::trace,α) hrel-rsp where
[upred-defs]: ISRD = ISRD1 \circ NSRD

lemma ISRD1-idem: ISRD1(ISRD1(P)) = ISRD1(P)
  by (rel-auto)

lemma ISRD1-monotonic: P \sqsubseteq Q \implies ISRD1(P) \sqsubseteq ISRD1(Q)
  by (rel-auto)

lemma ISRD1-RHS-design-form:
  assumes \$ok' # P \$ok' # Q \$ok' # R
shows $\text{ISRD1}(R_s(P \vdash Q \circ R)) = R_s(P \vdash \text{false} \circ (R \land \$tr' = u \$tr))$

using assms by (simp add: ISRD1-def choose-srd-def RHS-tri-design-par unrest, rel-auto)

lemma ISRD1-form:
$\text{ISRD1}(\text{SRD}(P)) = R_s(\text{pre}_R(P) \vdash \text{false} \circ (\text{post}_R(P) \land \$tr' = u \$tr))$
by (simp add: ISRD1-RHS-design-form SRD-as-reactive-tri-design unrest)

lemma ISRD1-rdes-def [rdes-def]:
$P \text{ is RR}; R \text{ is RR} \implies ISRD1(R_s(\text{pre}_R(P) \vdash Q \circ R)) = R_s(P \vdash \text{false} \circ (R \land \$tr' = u \$tr))$
by (simp add: ISRD1-def rdes-def closure rpred)

lemma ISRD-intro:
assumes $P \text{ is NSRD peri}_R(P) = (\neg \text{ pre}_R(P)) (\$tr' = u \$tr) \sqsubseteq \text{post}_R(P)$
shows $P \text{ is ISRD}$
proof –
have $R_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \circ \text{post}_R(P)) \text{ is ISRD1}$
apply (simp add: Healthy-def rdes-def closure assms (1-2))
using assms(3) least-zero apply (rel-blast)
done
hence $P \text{ is ISRD1}$
by (simp add: SRD-reactive-tri-design closure assms (1))
thus ?thesis
by (simp add: ISRD-def Healthy-comp assms (1))
qed

lemma ISRD1-rdes-intro:
assumes $P \text{ is RR} Q \text{ is RR} (\$tr' = u \$tr) \sqsubseteq Q$
shows $R_s(\text{pre}_R(P) \vdash \text{false} \circ Q) \text{ is ISRD1}$
unfolding Healthy-def
by (simp add: ISRD1-rdes-def assms closure unrest utp-pred-laws.inf.absorb1)

lemma ISRD-implies-ISRD1:
assumes $P \text{ is ISRD}$
shows $P \text{ is ISRD1}$
proof –
have $\text{ISRD}(P) \text{ is ISRD1}$
by (simp add: ISRD-def Healthy-def ISRD1-idem)
thus ?thesis
by (simp add: assms Healthy-if)
qed

lemma ISRD-implies-SRD:
assumes $P \text{ is ISRD}$
shows $P \text{ is SRD}$
proof –
have $1:\text{ISRD}(P) = R_s((\neg_r (\neg_r \text{ pre}_R(P)) :: R1 \text{ true} \land R1 \text{ true}) \vdash \text{false} \circ (\text{post}_R P \land \$tr' = u \$tr))$
by (simp add: NSRD-form ISRD1-def ISRD-def RHS-tri-design-par rdes-def unrest closure)
moreover have ... is SRD
by (simp add: closure unrest)

106
ultimately have \( ISRD(P) \) is SRD
  by (simp)
with assms show \(?thesis\)
  by (simp add: Healthy-def)
qed

lemma ISRD-implies-NSRD \[\text{[closure]}\]:
  assumes \( P \) is ISRD
  shows \( P \) is NSRD
proof –
  have \( 1: ISRD(P) = ISRD1(RD3(SRD(P))) \)
    by (simp add: ISRD-def NSRD-def SRD-def, metis RD1-RD3-commute RD3-left-subsumes-RD2)
  also have \( ... = ISRD1(RD3(P)) \)
    by (simp add: assms ISRD-implies-SRD)
  also have \( ... = R_s((\neg \pre R P \wp_r false_h \vdash (\exists \, st' \cdot \peri R P \circ post R P)) \)
    by (simp add: RD3-def, subst SRD-right-unit-tri-lemma, simp-all add: assms ISRD-implies-SRD)
  finally show \(?thesis\)
    by (simp add: Rdes-simp, simp add: RHS-tri-normal-design-composition' closure assms unrest ISRD-implies-SRD)
qed

lemma ISRD-form:
  assumes \( P \) is ISRD
  shows \( R_s((\pre R P \vdash false \circ (\post R P \land \str =_u \str)) = P \)
proof –
  have \( P = ISRD1(P) \)
    by (simp add: ISRD-implies-ISRD1 assms Healthy-if)
  also have \( ... = ISRD1(R_s((\pre R P \vdash \peri R P \circ post R P)) \)
    by (simp add: SRD-reactive-tri-design assms)
  finally show \(?thesis\)
    by (simp add: Rdes-simp, simp add: closure assms)
qed

lemma ISRD-elim \[\text{[RD-elim]}\]:
\[
\begin{array}{l}
\tri P \rightarrow Q(R_s((\pre R P \vdash false \circ (\post R P \land \str =_u \str)))) \\
\end{array}
\]
by (simp add: ISRD-form)

lemma skip-srd-ISRD \[\text{[closure]}\]: \( \tri R \) is ISRD
by (rule ISRD-intro, simp-all add: rdes close)

lemma assigns-srd-ISRD \[\text{[closure]}\]: \( \langle \sigma \rangle R \) is ISRD
by (rule ISRD-intro, simp-all add: rdes close, rel-auto)

lemma seq-ISRD-closed:
  assumes \( P \) is ISRD \( Q \) is ISRD
  shows \( P ;; Q \) is ISRD
  apply (insert assms)
apply (erule ISRD-elim)+
apply (simp add: rdes-def closure assms unrest)
apply (rule ISRD-rdes-intro)
  apply (simp-all add: rdes-def closure assms unrest)
apply (rel-auto)
done

lemma ISRD-Miracle-right-zero:
  assumes $P$ is ISRD $\text{pre}_R(P) = true_r$
  shows $P ;; \text{Miracle} = \text{Miracle}$
  by (rdes-simp cls: assms)

A recursion whose body does not extend the trace results in divergence

lemma ISRD-recurse-Chaos:
  assumes $P$ is ISRD $\text{post}_R P ;; true_r = true_r$
  shows $(\mu X \cdot P ;; X) = \text{Chaos}$
proof –
  have 1: $(\mu X \cdot X ;; X) = (\mu X \cdot P ;; \text{SRD}(X))$
    by (auto simp add: srdes-theory-continuous.utp-lfp-def closure assms)
  have $(\mu X \cdot P ;; \text{SRD}(X)) \sqsubseteq \text{Chaos}$
  proof (rule gfp-upperbound)
    have $P ;; \text{Chaos} \sqsubseteq \text{Chaos}$
      by (simp add: Healthy-if srdes-theory-continuous.bottom-closed)
    thus $P ;; \text{SRD} \text{Chaos} \sqsubseteq \text{Chaos}$
      by (metis 1 dual-order.antisym srdes-theory-continuous.LFP-closed srdes-theory-continuous.bottom-lower)
  qed
  thus $(\mu X \cdot \langle \sigma \rangle_R ;; X) = \text{Chaos}$
  by (rule ISRD-recurse-Chaos, simp-all add: closure rdes, rel-auto)
qed

13 Meta-theory for Reactive Designs

theory utp-rea-designs
  imports utp-rdes-parallel utp-rdes-guarded
end
References


