Reactive Designs in Isabelle/UTP

Simon Foster    James Baxter    Ana Cavalcanti    Jim Woodcock
Samuel Canham

April 6, 2018

Abstract

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

Contents

1 Introduction 3

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

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

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

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

2 Reactive Designs Healthiness Conditions

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

2.1 Preliminaries

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma R2c-skip-rea: $R2c \text{II}_c = \text{II}_c$
  by (simp add: skip-srea-def R2c-and R2c-disj R2c-skip-r R2c-not R2c-ak R2c-tr′-ge-tr)

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

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

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

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

lemma skip-srea-R2 [closure]: $\text{II}_R$ 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,"β") rel-rp ⇒ (′t,′a,"β") rel-rp where
  [upred-defs]: $RD1(P) = (P \lor (\neg \text{ok} \land \text{str} \leq_{\text{u}} \text{str}'))$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma \textit{skip-srea-RD1} [\textit{closure}]: \( \text{II}_R \text{ is RD1} \\
by \text{(rel-auto)}

lemma \textit{RD1-algebraic-intro}:
\begin{align*}
\text{assumes} & \quad P \text{ is R1} (\text{R1}(\text{true}_{\text{h}}) ;; P) = \text{R1}(\text{true}_{\text{h}}) (\text{II}_c ;; P) = P \\
\text{shows} & \quad P \text{ is RD1} \\
\text{proof} & \quad - \\
\text{have} & \quad P = (\text{II}_c ;; P) \\
\quad \text{by} \quad \text{(simp add: assms(3))} \\
\text{also have} & \quad \ldots = (\text{R1}(\texttt{ok} \Rightarrow \text{II}) ;; P) \\
\quad \text{by} \quad \text{(simp add: skip-rea-R1-lemma)} \\
\text{also have} & \quad \ldots = ((\neg \text{ok} \land \text{R1}(\text{true})) ;; P) \lor P \\
\quad \text{by} \quad \text{(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)} \\
\text{also have} & \quad \ldots = ((\text{R1}(\neg \texttt{ok}) ;; (\text{R1}(\text{true}_{\text{h}}) ;; P)) \lor P) \\
\quad \text{using dual-order.trans by} \quad \text{(rel-blast)} \\
\text{also have} & \quad \ldots = ((\text{R1}(\neg \texttt{ok}) ;; \text{R1}(\text{true}_{\text{h}})) \lor P) \\
\quad \text{by} \quad \text{(simp add: assms(2))} \\
\text{also have} & \quad \ldots = (\text{R1}(\neg \texttt{ok}) \lor P) \\
\quad \text{by} \quad \text{(rel-auto)} \\
\text{also have} & \quad \ldots = \text{RD1}(P) \\
\quad \text{by} \quad \text{(rel-auto)} \\
\text{finally show} & \quad \text{\textit{thesis}} \\
\quad \text{by} \quad \text{(simp add: Healthy-def)}
\end{align*}
\text{qed}
theorem RD1-left-zero:
assumes P is R1 P is RD1
shows \((R1(true) ;; P) = R1(true)\)
proof –
have \((R1(true) ;; R1(RD1(P))) = R1(true)\)
  by (rel-auto)
thus \(?thesis\)
  by (simp add: Healthy-if assms(1) assms(2))
qed

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

lemma RD1-alt-def:
assumes P is R1
shows \(RD1(P) = (P \triangleq \$ok \triangleright R1(true))\)
proof –
have \(RD1(R1(P)) = (R1(P) \triangleq \$ok \triangleright R1(true))\)
  by (rel-auto)
thus \(?thesis\)
  by (simp add: Healthy-if assms)
qed

theorem RD1-algebraic:
assumes P is R1
shows P is RD1 \(\iff\) \((R1(true_h) ;; P) = R1(true)\)
using RD1-algebraic-intro RD1-left-unit RD1-left-zero assms by blast

2.4 R3c and R3h: Reactive design versions of R3

definition R3c :: ('t::trace, 'a) hrel-rp ⇒ ('t, 'a) hrel-rp where
  [upred-defs]: \(R3c(P) = (\II_c \triangleq \$wait \triangleright P)\)

definition R3h :: ('s, 't::trace, 'a) hrel-rsp ⇒ ('s, 't, 'a) hrel-rsp where
  [upred-defs]: \(R3h(P) = ((\exists \$st \cdot H_c) \triangleq \$wait \triangleright P)\)

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

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

lemma R3c-mono: P \subseteq Q \implies R3c(P) \subseteq R3c(Q)
  by (rel-auto)

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

lemma R3h-idem: R3h(\text{R3h}(P)) = R3h(P)
by (rel-auto)

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

lemma R3h-mono: P \sqsubseteq Q \implies R3h(P) \sqsubseteq R3h(Q)
by (rel-auto)

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

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

lemma R3h-seq-closure [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-uvvar wait-vwb-lens)
  also have ... = $((\Pi[true/\$wait\] ; ; Q)[true/\$wait\] < $wait \triangleright (P ; ; Q))$
  by (metis Healthy-def' R3c-def assms(1))
  also have ... = $((\Pi[true/\$wait\] ; ; Q) < $wait \triangleright (P ; ; Q))$
  by (subst-tac)
  also have ... = $((\Pi \land \$wait\') ; ; Q) < $wait \triangleright (P ; ; Q))$
  by (metis no-types, lifting) cond-def conj-pos-var-subst seqr-pre-var-out skip-var utp-pred-laws.inf-left-idem
  also have ... = $((\Pi[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-mub wait-vwb-lens)
  also have ... = $((\Pi[true/\$wait\'] ; ; (H_c < $wait \triangleright Q)[true/\$wait\']) < $wait \triangleright (P ; ; Q))$
  by (metis Healthy-def' R3c-def assms(2))
  also have ... = $((\Pi[true/\$wait\'] ; ; H_c[true/\$wait\']) < $wait \triangleright (P ; ; Q))$
  by (subst-tac)
  also have ... = $((\Pi \land \$wait\') ; ; H_c < $wait \triangleright (P ; ; Q))$
  by (metis seqr-pre-transfer seqr-right-one-point true-alt-def uovar-cover upred-eq-true utp-rel.unrest-ouvar
  vwb-lens-mub wait-vwb-lens)
  also have ... = $((\Pi ; ; H_c) < $wait \triangleright (P ; ; Q))$
  by (simp add: cond-def seqr-pre-transfer utp-rel.unrest-ouvar)
  also have ... = $(H_c < $wait \triangleright (P ; ; Q))$
  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 ⊳ R1(true)) ⊲ $wait ⊳ 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 ⊳ $wait ⊲ P
  by (rel-auto)

lemma R3c-subst-wait: R3c(P) = R3c(P1)
  by (simp add: R3c-def cond-var-subst-right)

lemma R3h-subst-wait: R3h(P) = R3h(P1)
  by (simp add: R3h-cases cond-var-subst-right)

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

lemma R3h-wait-true:
  assumes P is R3h
  shows P1 = II R
proof -
  have P1 = (II R ⊲ $wait ⊳ P)1
    by (metis Healthy-if R3h-form assms)
  also have ... = II R1
    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 segr-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)
2.6 Major healthiness conditions

**Definition** $RH :: (′t::trace,′α) hrel-rp ⇒ (′t,′α) hrel-rp (R)

**Definition** $RHS :: (′s,′t::trace,′α) hrel-rsp ⇒ (′s,′t,′α) hrel-rsp (R_s)

**Definition** $RD :: (′t::trace,′α) hrel-rp ⇒ (′t,′α) hrel-rp

**Definition** $SRD :: (′s,′t::trace,′α) hrel-rsp ⇒ (′s,′t,′α) hrel-rsp

**Lemma** $RH-comp: RH = R1 \circ R2c \circ R3c

**Lemma** $RH-idem: RH(P) = R(P)

**Lemma** $RH-Idempotent: Idempotent R

**Lemma** $RH-Monotonic: Monotonic R

**Lemma** $RH-Continuous: Continuous R

**Lemma** $RHS-comp: RHS = R1 \circ R2c \circ R3h

**Lemma** $RHS-idem: RHS(P) = R1(R2c(R3h(P)))

**Lemma** $RHS-Idempotent: Idempotent R_s

**Lemma** $RHS-Monotonic: Monotonic R_s

**Lemma** $RHS-mono: P \sqsubseteq Q \implies R_s(P) \sqsubseteq R_s(Q)
using mono-def RHS-Monotonic by blast

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

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

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

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

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

lemma RHS-cond:
  R_s(P ⊳ b ⋵ Q) = (R_s(P) ⊳ R2c b ⋵ 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-R2c-commute RD1-R3c-commute RD1-RD2-commute RH-def RD-def RP-def)

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

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

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

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

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

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

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

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

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

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

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

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

lemma SRD-healths [closure]:
  assumes P is SRD
  shows P is R1 P is R2 P is R3h P is RD1 P is RD2
  apply (metis Healthy-def R1-R2c-is-R2 RHS-def SRD-def assms)
  apply (metis Healthy-def RD1-RHS-commute RD2-RHS-commute R3h-idem RD1-R3h-commute RD2-R3h-commute
         SRD-def assms)
  apply (metis Healthy-def RD1-RD2-commute RD2-idem SRD-def assms)
  done

lemma SRD-intro:
  assumes P is R1 P is R2 P is R3h P is RD1 P is RD2
  shows P is SRD
  by (metis Healthy-def R1-R2c-is-R2 RHS-def SRD-def assms)

lemma SRD-ok-false [usubst]: P is SRD \Rightarrow P[\false/ok] = R1(true)
  by (metis (no-types, hide-lams) H1-H2-eq-design Healthy-def R1-ok-false RD1-R1-commute
       RD1-via-R1 RD2-def SRD-def SRD-healths)

lemma SRD-ok-true-wait-true [usubst]:
  assumes P is SRD
  shows P[true, true/ok, wait] = (\exists st \cdot II)[true, true/ok, wait]
  proof
    have P = (\exists st \cdot II) \land \ok \land R1 \land \wait \land P
      by (metis Healthy-def RD1-R3h-cases SRD-healths)
    moreover have (\exists st \cdot II) \land \ok \land R1 \land \wait \land P)[true, true/ok, wait] = (\exists st \cdot II)[true, true/ok, wait]
      by (simp add: usubst)
    ultimately show \thesis
      by (simp)
  qed

lemma SRD-left-zero-1: P is SRD \Rightarrow R1(true) ;; P = R1(true)
  by (simp add: RD1-left-zero SRD-healths)
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 ::= utp-hcond :: (RDES, ('t::trace,'α) rp) uthy ⇒ (('t,'α) rp × ('t,'α) rp) health

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

begin

definition rdes-hcond :: (RDES, ('t::trace,'α) rp) uthy ⇒ (('t,'α) rp × ('t,'α) rp) health where

[upred-defs]: rdes-hcond T = RD

definition srdes-hcond :: (SRDES, ('s,'t::trace,'α) rsp) uthy ⇒ (('s,'t,'α) rsp × ('s,'t,'α) rsp) health where

[upred-defs]: srdes-hcond T = SRD
end

interpretation rdes-theory: utp-theory UTHY(RDES, ('t::trace,'α) rp)

by (unfold-locales, simp-all add: rdes-hcond-def RD-idem)

interpretation rdes-theory-continuous: utp-theory-continuous UTHY(RDES, ('t::trace,'α) rp)

rewrites \( \forall P. P \in \text{carrier} \ (\text{uthy-order} \ RDES) \iff P \text{ is RD} \)

and carrier (uthy-order RDES) → carrier (uthy-order RDES) ≡ \([RD]_H \to [RD]_H\)

and le (uthy-order RDES) = op \subseteq

and eq (uthy-order RDES) = op =

by (unfold-locales, simp-all add: rdes-hcond-def RD-Continuous)

interpretation rdes-rea-galois:

galois-connection (RDES ⇔ (RD1 ○ RD2,R3) → REA)

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

show R3 ∈ \([RD]_H \to [RP]_H\)

by (metis (no-types, lifting) Healthy-def' Pi-I R3-RD-RP RP-idem mem-Collect-eq)

show RD1 ○ RD2 ∈ \([RP]_H \to [RD]_H\)

by (simp add: Pi-iff Healthy-def, metis RD-def RD-idem)

show isotone (utp-order RD) (utp-order RP) R3

by (simp add: R3-Monotonic isotone-utp-orderI)

show isotone (utp-order RP) (utp-order RD) (RD1 ○ RD2)

by (simp add: Monotonic-comp RD1-Monotonic RD2-Monotonic isotone-utp-orderI)

fix P :: ('a, 'b) hrel-rp

assume P is RD

thus P ⊆ RD1 (RD2 (R3 P))

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

next

fix P :: ('a, 'b) hrel-rp

assume a: P is RP
thus $R_3 \ (R_1 \ (R_2 \ P)) \subseteq P$

proof

**have** $R_3 \ (R_1 \ (R_2 \ P)) = R_P \ (R_1 \ (R_2 \ P))$

by (metis Healthy-if $R_3$-$R_D$-$R_P$ RD-def a)

**moreover have** $R_1 \ (R_2 \ P) \subseteq P$

by (rel-auto)

ultimately show '?'thesis

by (metis Healthy-if RP-mono a)

qed

qed

interpretation rdes-rea-retract:

retract $(RDES \leftarrow (R_1 \circ R_2, R_3) \rightarrow REA)$

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

(by metis Healthy-if $R_3$-$R_D$-$R_P$ RD-def RP-idem eq-refl)

interpretation srdes-theory: utp-theory UTHY(SRDES, ('s', 't::trace, 'a) rsp)

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

interpretation srdes-theory-continuous: utp-theory-continuous UTHY(SRDES, ('s', 't::trace, 'a) rsp)

rewrites $P, P \in \text{carrier} (\text{uthy-order SRDES}) \iff P \text{ is SRD}$

and $P \text{ is } SRDES \iff P \text{ is SRD}$

and $(\mu X \cdot F (\text{SRDES}_H X)) = (\mu X \cdot F \ (SRD X))$

and carrier (uthy-order SRDES) $\rightarrow$ carrier (uthy-order SRDES) $\equiv [SRD]_H \rightarrow [SRD]_H$

and $[\text{SRDES}_H]_H \equiv [\text{SRD}]_H \rightarrow [\text{SRD}]_H$

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, 'a) hrel-rsp where

Chaos $\equiv \bot_{SRDES}$

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

Miracle $\equiv \top_{SRDES}$

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

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

thm srdes-theory-continuous.meet-bottom

thm srdes-theory-continuous.meet-top

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

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

syntax

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

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

translations

$\mu_R X \cdot P \Rightarrow \mu_R (\lambda X. P)$

$\nu_R X \cdot P \Rightarrow \mu_R (\lambda X. P)$

The reactive design weakest fixed-point can be defined in terms of relational calculus one.
lemma srd-mu-equiv:
  assumes Monotonic F F ∈ [SRD]H → [SRD]H
  shows (µR X ∙ F(X)) = (µ X ∙ F(SRD(X)))
  by (metis assms srdes-hcond-def srdes-theory-continuous.upt-lfp-def)
end

3 Reactive Design Specifications

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

3.1 Reactive design forms

lemma srdes-skip-def: II_R = R_s(true ⊢ (tr' = u $tr' ∧ ¬ wait' ∧ [II]_R))
apply (rel-auto) using minus-zero-eq by blast+

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

lemma Miracle-def: Miracle = R_s(true ⊢ false)
proof −
  have Miracle = SRD(false)
    by (metis srdes-hcond-def srdes-theory-continuous.healthy-top)
  also have ... = R_s(H(false))
    by (simp add: SRD-RHS-H1-H2)
  also have ... = R_s(true ⊢ false)
  finally show thesis .
qed

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

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

lemma RD1-st-reactive-design: RD1(R_s(P ⊨ Q)) = R_s(P ⊨ Q)
  by (rel-auto)

lemma RD2-st-reactive-design:
  assumes $ok' ∉ P $ok' ∉ Q
shows \( RD2(R_s(P \vdash Q)) = R_s(P \vdash Q) \)
using assms
by (metis H2-design RD2-RHS-commute RD2-def)

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

lemma RD-RH-design-form:
\( RD(P) = R((\neg P_f) \vdash P^t_f) \)
proof 
  have \( RD(P) = RD1(RD2(R1(R2c(R3c(P))))) \)
    by (simp add: RD-alt-def RH-def)
  also have \( ... = RD1(H2(R1(R2s(R3c(P))))) \)
    by (simp add: R1-R2s-R2c RD2-def)
  also have \( ... = RD1(R1(H2(R2s(R3c(P))))) \)
    by (simp add: R1-H2-commute)
  also have \( ... = R1(R1(H2(R2s(R3c(P))))) \)
    by (simp add: R1-idem RD1-via-R1)
  also have \( ... = R2(R1(H2(R3c(H2(R1(P)))))) \)
    by (simp add: R1-R2c-commute RD1-R3c-commute RD1-via-R1)
  also have \( ... = RH(R(H1(P))) \)
    by (simp add: H1-H2-eq-design)
  also have \( ... = RH((\neg P_f) \vdash P^t_f) \)
    by (simp add: H1-H2-eq-design)
  also have \( ... = R((\neg P_f) \vdash P^t_f) \)
    by (metis RD2-R3c-commute RD2-def)
  finally show \( \text{thesis} \).
qed

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

lemma RD-RH-design:
assumes \( \$ok \ \sharp P \ \$ok \ \sharp Q \)
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 \( \$ok \ \sharp P \ \$ok \ \sharp Q \)
shows \( R(P \vdash Q) \) is RD
by (simp add: RD-RH-design Healthy-def assms(1) assms(2))
lemma SRD-RH-design-form:
$$SRD(P) = R_s((\neg P)^f)$$

proof
have $$SRD(P) = R_1(R_2c(R_3h(RD1(RD2(R1(P))))))$$
by (metis (no-types, lifting) R1-H2-commute R1-R2c-commute R1-R3h-commute R1-idem R2c-H2-commute RD1-R1-commute RD1-R2c-commute RD1-R3h-commute RD2-R3h-commute RD2-def RHS-def SRD-def)
also have ...
by (metis (no-types, lifting) R1-H2-commute R1-R2c-commute R1-R3h-commute R1-idem R2c-H2-commute RD1-R2c-commute RD1-R3h-commute RD2-R3h-commute RD2-def RHS-def SRD-def)
also have ...
by
also have ...
by
also have ...
by
finally show ?thesis .
qed

lemma SRD-reactive-design:
assumes P is SRD
shows $$R_s((\neg P)^f) = P$$
by (metis SRD-RH-design-form Healthy-def’ assms)

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

lemma RHS-design-is-SRD:
assumes $ok^{\#}P$ $ok^{\#}Q$
shows $$R_s(P \vdash Q)$$ is SRD
by (simp add: Healthy-def’ SRD-RH-design assms(1) assms(2))

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

3.2 Auxiliary healthiness conditions

definition [upred-defs]: $$R_3c-pre(P) = (true \land \$wait \triangleright P)$$

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

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

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

lemma R3c-pre-seq:
$$(true ;; Q) = true \Longrightarrow R_3c-pre(P ;; Q) = (R_3c-pre(P) ;; Q)$$
by (rel-auto)

lemma unrest-ok-R3c-pre [unrest]: $$\$ok \not\triangleright P \Longrightarrow \$ok \not\triangleright R_3c-pre(P)$$
by (simp add: R3c-pre-def cond-def unrest)

lemma unrest-ok’-R3c-pre [unrest]: $$\$ok^\prime \not\triangleright P \Longrightarrow \$ok^\prime \not\triangleright R_3c-pre(P)$$
lemma unrest-ok-R3c-post [unrest]: $\$ok \not\in P \implies \$ok \not\in R3c-post(P)$

by (simp add: R3c-pre-def cond-def unrest)

lemma unrest-ok-R3c-post' [unrest]: $\$ok' \not\in P \implies \$ok' \not\in R3c-post(P)$

by (simp add: R3c-post-def cond-def unrest)

lemma unrest-ok-R3h-post [unrest]: $\$ok \not\in P \implies \$ok \not\in R3h-post(P)$

by (simp add: R3h-post-def cond-def unrest)

lemma unrest-ok-R3h-post' [unrest]: $\$ok' \not\in P \implies \$ok' \not\in R3h-post(P)$

by (simp add: R3h-post-def cond-def unrest)

3.3 Composition laws

theorem R1-design-composition:

fixes P Q :: (′t::trace,′a,′b) rel-rp

and R S :: (′t,′a,′b,′c) rel-rp

assumes $\$ok' \not\in P \$ok' \not\in Q \$ok' \not\in R \$ok' \not\in S$

deprecated

shows $(R1(P \triangleright Q); R1(R \triangleright S)) = R1(\neg (R1(P) \triangleright R1(S))) 

proof

have $(R1(P \triangleright Q); R1(R \triangleright S)) = (\exists \ ok_0 \cdot (R1(P \triangleright Q)[\$ok_0/\$ok'] \triangleright (R1(R \triangleright S))[\$ok_0/\$ok'])$

using seqr-middle ok-vwb-lens by blast

also from assms have ... = $(\exists \ ok_0 \cdot R1((\$ok \land P) \triangleright (\$ok \land Q)) \triangleright R1((\$ok \land R) \triangleright (\$ok' \land S)))$

by (simp add: design-def R1-design-composition)

also from assms have ... = $(\exists \ ok_0 \cdot R1(\neg (\$ok \land P) \land R \land (\$ok' \land S)))$

by (simp add: impl-alt-def utp-pred-laws.sup.assoc)

also from assms have ... = $(\exists \ ok_0 \cdot R1(\neg (\$ok \land P) \land R \land (\$ok' \land S)))$

by (simp add: R1-disj utp-pred-laws.disj-assoc)

also from assms have ... = $(\exists \ ok_0 \cdot R1(\neg (\$ok \land P) \land R \land (\$ok' \land S)))$

by (simp add: seqr-or-dist sup.assoc)

also from assms have ... = $(\exists \ ok_0 \cdot R1(\neg (\$ok \land P) \land R \land (\$ok' \land S)))$

by (simp add: rel-blast)

also from assms have ... = $(\exists \ ok_0 \cdot R1(\neg (\$ok \land P) \land R \land (\$ok' \land S)))$

by (simp add: R1-disj R1-extend-conj utp-pred-laws.inf-commute)

also have ... = $(\exists \ ok_0 \cdot R1(\neg (\$ok \land S) \land (\$ok' \land S)))$

by (simp add: R1-disj seqr-or-dist)

also have ... = $(\exists \ ok_0 \cdot R1(\neg (\$ok \land S) \land (\$ok' \land S)))$

by (simp add: R1-disj seqr-or-dist)
\[
\begin{align*}
&\lor (R1(\neg \$ok)) \\
&\lor (R1(\neg P) \implies R1(\true))
\end{align*}
\]

proof -
  have \(((R1(\neg \$ok) :: (\'t,\alpha,\beta) \rel-rp) :: R1(\true)) =
  \begin{align*}
  & (R1(\neg \$ok) :: (\'t,\alpha,\gamma) \rel-rp) \\
  & \text{by (rel-auto)}
  \end{align*}
  thus \presubstitute{\text{thesis}}
  by simp
qed

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

theorem R1-design-composition-RR:
  assumes P is RR Q is RR R is RR S is RR
  shows \((R1(P \implies Q) :: R1(R \implies S)) = R1(\neg P \land Q \implies R \implies S))
  apply (rel-blast)
  apply (simp add: subst R1-design-composition)
  apply (simp 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 \implies Q) :: R1(R \implies S)) = R1((P \land Q \implies R) \implies (Q :: S))
  by (simp add: R1-design-composition-RR assms unrest Healthy-if closure wp)

lemma R2s-design: R2s(P \implies Q) = (R2s(P) \implies 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 \( \script{ok}' \not\in P \not\in Q \not\in R \not\in S \)
shows \( \script{R3c}(R1(P \vdash Q)) \vdash \script{R3c}(R1(R \vdash S)) = R3c(R1(\neg R1(\neg P) \vdash R1(true)) \land \neg \((R1(Q) \land \neg \$\text{wait'})) \vdash R1(\neg R)))\)
by (rel-auto)

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

have 2: \(\neg \(R1(R3c-post Q) \vdash R1(\neg R3c-pre R))\) = R3c-pre(\(\neg \((R1 Q \land \neg \$\text{wait'})) \vdash R1(\neg R)))
by (rel-auto, blast+)

have 3: \(R1(R3c-post Q) \vdash R1(R3c-pre S) = R3c-post(R1 Q) \vdash (\exists st \cdot [II]D \not\in \$\text{wait} \triangleright R1(S))))\)
by (rel-auto)

show \(\script{thesis}\)
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 \( \script{ok}' \not\in P \not\in Q \not\in R \not\in S \)
shows \( \script{R3h}(R1(P \vdash Q)) \vdash \script{R3h}(R1(R \vdash S)) = R3h(R1(\neg R1(\neg P) \vdash R1(true)) \land \neg \((R1(Q) \land \neg \$\text{wait'})) \vdash R1(\neg R)))\)
by (rel-auto)

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

have 2: \(\neg \(R1(R3h-post Q) \vdash R1(\neg R3c-pre R))\) = R3c-pre(\(\neg \((R1 Q \land \neg \$\text{wait'}) \vdash R1(\neg R)))\)
by (rel-auto, blast+)

have 3: \(R1(R3h-post Q) \vdash R1(R3h-post S) = R3h-post(R1 Q) \vdash (\exists st \cdot [II]D \not\in \$\text{wait} \triangleright R1(S))))\)
by (rel-auto, blast+)

show \(\script{thesis}\)
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 \( \script{ok}' \not\in P \not\in Q \not\in R \not\in S \)
shows \( \script{R2}(P \vdash Q) \vdash \script{R2}(R \vdash S) = R2(\neg R1(\neg R2c P) \vdash R1 true) \land \neg \((R1(R2c Q) \vdash R1(\neg R2c R))) \vdash (R1(R2c Q) \vdash R1)\)

20
proof -

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

proof -

have 1: R2c (R1 (¬ R2s P) ;; R1 true) = (R1 (¬ R2s P) ;; R2 true)

by (rel-auto)

have R2c (R1 (¬ R2s P) ;; R2 true) = R2c (R1 (¬ R2s P) ;; R2 true)

using R2c-not by blast

also have ... = R2c (¬ 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 R2-not R2s-true)

finally show thesis

by (simp add: 1)

qed

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

proof -

have ((R1 (R2s Q) ∧ ¬ $wait') ;; R1 (¬ R2s R)) = R1 (¬ R2s R)

by (rel-auto)

hence R2c ((R1 (R2s Q) ∧ ¬ $wait') ;; R1 (¬ R2s R)) = (R2 (¬ P) ;; R2 true)

by (simp add: R2-seqr-distribute)

also have ... = (R1 (¬ R2s R) ;; R1 (¬ R2s R))

by (rel-auto)

finally show thesis.

qed

have 3: R2c (((R1 (R2s Q)) ;; (♯ I D < $wait R1 (R2s S)))) = (R1 (R2s Q) ;; (♯ I D < $wait R1 (R2s S)))

proof -

have R2c (((R1 (R2s Q)) (true/$wait')) ;; (♯ I D < $wait R1 (R2s S))(true/$wait')) = (R1 (R2s Q))(true/$wait') ;; (♯ I D < $wait R1 (R2s S))(true/$wait')

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

also have ... = R2c (R2 (Q (true/$wait')) ;; R2 ([♯ I D (true/$wait)])

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

also have ... = (R2 (Q (true/$wait')) ;; R2 ([♯ I D (true/$wait)])

using R2c-seq by blast

also have ... = ((R1 (R2s Q)) (true/$wait')) ;; (♯ I D < $wait R1 (R2s S))(true/$wait')

apply (simp add: unsubst R2-des-lift-skip)
apply (metis R2-def R2-des-lift-skip R2-subst-wait'-true R2-subst-wait-true)
done
finally show ?thesis.
qed
moreover have $R2c(((R1 (R2s Q))\{false/\$wait\} :: ([|D|] D < $wait \triangleright R1 (R2s S))\{false/\$wait\}))$
  = $$((R1 (R2s Q))\{false/\$wait\} :: ([|D|] D < $wait \triangleright R1 (R2s S))\{false/\$wait\})$$
by (simp add: usubst cond-unit-F)
(metais (no-types, hide-lams) R1-wait'-false R1-wait-false R2-def R2-subst-wait'-false R2-subst-wait-false)
ultimately show ?thesis.
proof –
  have $[|D|] D < $wait \triangleright R1 (R2s S) = R2 (([|D|] D < $wait \triangleright S)$
by (simp add: R1-R2c-is-R2 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 $(R1(R2s(R3c(P \triangleright Q))) :: R1(R2s(R3c(R \triangleright S)))) =$
  $(R3c(R1(R2s(P) \triangleright R2s(Q))) :: R3c(R1(R2s(R) \triangleright R2s(S))))$
by (metais (no-types, hide-lams) R1-R2s-R2c R1-R3c-commute R2c-R3c-commute R2-design)
also have ... = $R3c(R1(\neg (R1 (\neg R2s P) :: R1 true) \land \neg ((R1 (R2s Q) \land \neg $wait') :: R1 (\neg R2s R)))$
  \triangleright $(R1 (R2s Q) :: ([|D|] D < $wait \triangleright R1 (R2s S))))$
by (simp add: R3c-R1-design-composition assms unrest)
also have ... = $R3c(R1(R2c(\neg (R1 (\neg R2s P) :: R1 true) \land \neg ((R1 (R2s Q) \land \neg $wait') :: R1 (\neg R2s R)))$
  \triangleright $(R1 (R2s Q) :: ([|D|] D < $wait \triangleright R1 (R2s S))))$
by (simp add: R2c-design R2c-and R2c-not 1 2 3)
finally show ?thesis.
by (simp add: R1-R2s-R2c R1-R3c-commute R2c-R3c-commute RH-def)

lemma RHS-design-composition:
  assumes $\$ok' \notin P \$ok' \notin Q \$ok \notin R \$ok \notin S$
  shows $(R_c(P \triangleright Q)) :: R_c(R \triangleright S) =$
  $(R_c(\neg (R1 (\neg R2s P) :: R1 true) \land \neg ((R1 (R2s Q) \land \neg $wait') :: R1 (\neg R2s R)))$
  \triangleright $(R1 (R2s Q) :: ([|D|] D < $wait \triangleright R1 (R2s S))))$

proof –
  have 1: $R2c(R1 (\neg R2s P) :: R1 true) = (R1 (\neg R2s P) :: R1 true)$
by (rel-auto, blast)
proof –
  have 1: $(R1 (\neg R2s P) :: R1 true) = (R1(R2 (\neg P) :: R2 true))$
by (rel-auto, blast)
have $R2c(R1(R2 (\neg P) :: R2 true)) = R2c(R1(R2 (\neg P) :: R2 true))$
using R2c-not by blast
also have ... = $R2(R2 (\neg P) :: R2 true)$
by (metais R1-R2c-commute R1-R2c-is-R2)
also have ... = $(R2 (\neg P) :: R2 true)$
by (simp add: R2-seq-distribute)
also have ... = $(R1 (\neg R2s P) :: R1 true)$
by (simp add: R2-def R2s-not R2s-true)
finally show ?thesis.
by (simp add: 1)
qed
have 2: R2c (((R1 (R2s Q) ∧ ¬ $\$wait$′) ; R1 (¬ R2s R)) = ((R1 (R2s Q) ∧ ¬ $\$wait$′) ; R1 (¬ R2s R)))

proof 
have ((R1 (R2s Q) ∧ ¬ $\$wait$′) ; R1 (¬ R2s R)) = R1 (R2 (Q ∧ ¬ $\$wait$′) ; R2 (¬ R))
by (rel-auto, blast+)
hence R2c ((R1 (R2s Q) ∧ ¬ $\$wait$′) ; R1 (¬ R2s R)) = (R2 (Q ∧ ¬ $\$wait$′) ; R2 (¬ R))
by (metis (no-types, lifting) R1-R2c-commute R1-R2c-is-R2 R2-seq-distribute)
also have ... = ((R1 (R2s Q) ∧ ¬ $\$wait$′) ; R1 (¬ R2s R))
by (rel-auto, blast+)
finally show ?thesis .

qed

have 3: R2c((R1 (R2s Q) ; ((∃ $st$ · [III]D) $\triangleleft$ $\$wait$ ⊃ R1 (R2s S))) =
(R1 (R2s Q) ; ((∃ $st$ · [III]D) $\triangleleft$ $\$wait$ ⊃ R1 (R2s S)))

proof 
have R2c(((R1 (R2s Q))[true/$\$wait$′] ; ((∃ $st$ · [III]D) $\triangleleft$ $\$wait$ ⊃ R1 (R2s S))[true/$\$wait$]) =
((R1 (R2s Q))[true/$\$wait$′] ; ((∃ $st$ · [III]D) $\triangleleft$ $\$wait$ ⊃ R1 (R2s S))[true/$\$wait$])
by (simp add: usubst cond-unit-T R1-def R2s-def)
also have ... = R2c(R2(Q[true/$\$wait$′]) ; R2((∃ $st$ · [III]D)[true/$\$wait$])
by (metis (no-types, lifting) R2-def R2-des-lift-skip R2-subst-wait-true R2-st-ex)
also have ... = (R2(Q[true/$\$wait$′]) ; R2((∃ $st$ · [III]D)[true/$\$wait$])
using R2c-seq by blast
also have ... = ((R1 (R2s Q))[true/$\$wait$′] ; ((∃ $st$ · [III]D) $\triangleleft$ $\$wait$ ⊃ R1 (R2s S))[true/$\$wait$])
apply (simp add: usubst R2-des-lift-skip R2-st-ex R2-subst-wait-true R2-subst-wait-true)
done
finally show ?thesis .

qed

moreover have R2c(((R1 (R2s Q))[false/$\$wait$′] ; ((∃ $st$ · [III]D) $\triangleleft$ $\$wait$ ⊃ R1 (R2s S))[false/$\$wait$]) =
((R1 (R2s Q))[false/$\$wait$′] ; ((∃ $st$ · [III]D) $\triangleleft$ $\$wait$ ⊃ R1 (R2s S))[false/$\$wait$])
by (simp add: usubst)
(metis (no-types, lifting) R1-wait-&-false R1-wait-false R2-R1-form R2-subst-wait-&-false R2-subst-wait-false R2c-seq)

ultimately show ?thesis 
by (smt R2-R1-form R2-condr' R2-des-lift-skip R2-st-ex R2c-seq R2s-wait)

qed

have (R1(R2s(R3h(P ⊃ Q))) ; R1(R2s(R3h(R ⊃ S)))) =
((R3h(R1(R2s(P) ⊃ R2s(Q)))) ; R3h(R1(R2s(R) ⊃ R2s(S))))
by (metis (no-types, hide-lams) R1-R2s-R2c R1-R3h-commute R2c-R3h-commute R2s-design)
also have ... = R3h(R1 ((¬ (R1 (¬ R2s P) ; R1 true) ∧ ¬ ((R1 (R2s Q) ∧ ¬ $\$wait$′) ; R1 (¬ R2s R)))) ⊃
(R1 (R2s Q) ; ((∃ $st$ · [III]D) $\triangleleft$ $\$wait$ ⊃ R1 (R2s S))))
by (simp add: R3h-R1-design-composition assms unrest)
also have ... = R3h(R1(R2c(¬ (R1 (¬ R2s P) ; R1 true) ∧ ¬ ((R1 (R2s Q) ∧ ¬ $\$wait$′) ; R1 (¬ R2s R)))) ⊃
(R1 (R2s Q) ; ((∃ $st$ · [III]D) $\triangleleft$ $\$wait$ ⊃ 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-R3h-commute R2c-R3h-commute RHS-def)

qed
lemma RHS-R2s-design-composition:

assumes
\( \text{\textit{ok'} P \text{\textit{ok'} Q \text{\textit{ok'} R \text{\textit{ok'} S}}} \)
\( P \text{ is R2s Q is R2s R is R2s S is R2s} \)

shows \( (R_s(P \vdash Q) :: R_s(R \vdash S)) = R_s((\neg (R1 (\neg P) :: R1 \text{ true}) \land (\neg (R1 Q \land \neg \$\text{wait'})) :: R1 (\neg R))) \vdash \)
\( (R1 Q :: (\exists \text{ st } \cdot [II] \text{ P } \text{ derives} \neg \text{ st } \vdash \neg \text{ st } \text{ derives} R1 S))) \)

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 \( \text{thesis} \)
using \( f3 f2 f1 \) by (simp add: RHS-design-composition assms(1) assms(2) assms(3) assms(4))

qed

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

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

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

lemma RHS-design-export-R1: \( R_s(P \vdash Q) = R_s(P \vdash R1(Q)) \)
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: \( R_s(P[\text{true,false}/\$\text{ok},\$\text{wait}] :: Q[\text{true,false}/\$\text{ok},\$\text{wait}]) = R_s(P \vdash Q) \)
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 \textit{RHS-pre-lemma}: $(R_s \; P)^f_f = R_1(R_2c(P^f_f))$
\begin{itemize}
  \item by (rel-auto)
\end{itemize}

lemma \textit{RHS-design-R2c-pre}:

\begin{itemize}
  \item $R_s(R_2c(P) \vdash Q) = R_s(P \vdash Q)$
  \item by (rel-auto)
\end{itemize}

3.4 Refinement introduction laws

lemma \textit{R1-design-refine}: assumes \begin{itemize}
  \item $P_1$ is $R_1 \; P_2$ is $R_1 \; Q_1$ is $R_1 \; Q_2$ is $R_1$
  \item $\$ok \not\supset P_1 \; \$ok' \not\supset P_1 \; \$ok \not\supset P_2 \; \$ok' \not\supset P_2$
  \item $\$ok \not\supset Q_1 \; \$ok' \not\supset Q_1 \; \$ok \not\supset Q_2 \; \$ok' \not\supset Q_2$
\end{itemize}

\begin{itemize}
  \item shows $R_1(P_1 \vdash P_2) \subseteq R_1(Q_1 \vdash Q_2) \iff \langle P_1 \Rightarrow Q_1 \rangle \wedge \langle P_1 \land Q_2 \Rightarrow P_2 \rangle$
\end{itemize}

\begin{itemize}
  \item proof –
  \item have $R_1((\exists \; \$ok; \$ok' \cdot P_1) \vdash (\exists \; \$ok; \$ok' \cdot P_2)) \subseteq R_1((\exists \; \$ok; \$ok' \cdot Q_1) \vdash (\exists \; \$ok; \$ok' \cdot Q_2))$
  \item $\iff \langle R_1(\exists \; \$ok; \$ok' \cdot P_1) \Rightarrow R_1(\exists \; \$ok; \$ok' \cdot Q_1) \rangle \wedge \langle R_1(\exists \; \$ok; \$ok' \cdot P_1) \land R_1(\exists \; \$ok; \$ok' \cdot Q_2) \rangle$
\end{itemize}

\begin{itemize}
  \item by (rel-auto, meson+)
  \item thus \(\langle \text{thesis} \rangle\)
  \item by (simp-all add: ex-unrest ex-plus \text{Healthy-if assms})
\end{itemize}

qed

lemma \textit{R1-design-refine-RR}:

\begin{itemize}
  \item assumes $P_1$ is $RR \; P_2$ is $RR \; Q_1$ is $RR \; Q_2$ is $RR$
  \item shows $R_1(P_1 \vdash P_2) \subseteq R_1(Q_1 \vdash Q_2) \iff \langle P_1 \Rightarrow Q_1 \rangle \land \langle P_1 \land Q_2 \Rightarrow P_2 \rangle$
  \item by (simp add: \textit{R1-design-refine assms unrest closure})
\end{itemize}

lemma \textit{RHS-design-refine}:

\begin{itemize}
  \item assumes \begin{itemize}
    \item $P_1$ is $R_1 \; P_2$ is $R_1 \; Q_1$ is $R_1 \; Q_2$ is $R_1$
    \item $P_1$ is $R_2c \; P_2$ is $R_2c \; Q_1$ is $R_2c \; Q_2$ is $R_2c$
    \item $\$ok \not\supset P_1 \; \$ok' \not\supset P_1 \; \$ok \not\supset P_2 \; \$ok' \not\supset P_2$
    \item $\$ok \not\supset Q_1 \; \$ok' \not\supset Q_1 \; \$ok \not\supset Q_2 \; \$ok' \not\supset Q_2$
    \item $\$wait \not\supset P_1 \; \$wait \not\supset P_2 \; \$wait \not\supset Q_1 \; \$wait \not\supset Q_2$
  \end{itemize}
  \item shows $R_s(P_1 \vdash P_2) \subseteq R_s(Q_1 \vdash Q_2) \iff \langle P_1 \Rightarrow Q_1 \rangle \wedge \langle P_1 \land Q_2 \Rightarrow P_2 \rangle$
\end{itemize}

\begin{itemize}
  \item proof –
  \item have $R_s(P_1 \vdash P_2) \subseteq R_s(Q_1 \vdash Q_2) \iff R_1(R_3h(R_2c(P_1 \vdash P_2))) \subseteq R_1(R_3h(R_2c(Q_1 \vdash Q_2)))$
  \item by (simp add: \textit{R2c-R3h-commute RHS-def})
  \item also have \(\langle \text{thesis} \rangle\)
  \item by (simp add: \text{Healthy-if R2c-design assms})
  \item also have \(\langle \text{thesis} \rangle\)
  \item by (rel-auto, meson+)
  \item also have \(\langle \text{thesis} \rangle\)
  \item by (rel-auto)
\end{itemize}

\begin{itemize}
  \item also have \(\langle \text{thesis} \rangle\)
  \item by (simp add: \text{usubst assms closure unrest})
  \item also have \(\langle \text{thesis} \rangle\)
  \item by (simp add: \textit{R1-design-refine assms})
\end{itemize}

finally show \(\langle \text{thesis} \rangle\).

qed

lemma \textit{srdes-refine-intro}:

\begin{itemize}
  \item assumes $\langle P_1 \Rightarrow P_2 \rangle \land P_1 \land Q_2 \Rightarrow Q_1$
\end{itemize}
shows \( R_s(P_1 \vdash Q_1) \subseteq R_s(P_2 \vdash Q_2) \)
by (simp add: RHS-mono assms design-refine-intro)

### 3.5 Distribution laws

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

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

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

end

### 4 Reactive Design Triples

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

#### 4.1 Diamond notation

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

**lemma** wait'-'-cond-unrest [unrest]:
\([\text{out-var} \, \text{wait} \circ \sigma \, \text{P} \, \text{x} \, \text{Q}] \implies x \uparrow (P \circ Q)\)
by (simp add: wait'-'-cond-def unrest)

**lemma** wait'-'-cond-subst [subst]:
\(\text{\$wait}' \uparrow \sigma \implies \sigma \uparrow (P \circ Q) = (\sigma \uparrow P) \circ (\sigma \uparrow Q)\)
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: \((P \circ Q) \\ R) = ((P \\ (\$wait \land R)) \lor (Q \\ (\neg \$wait \land R)))\)
by (simp add: wait'-'-cond-def cond-def seqr-or-disil, 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:
\((P \circ Q) \land (R \circ S) = (P \land R) \circ (Q \land S)\)

26
by (rel-auto)

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

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

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

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

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

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

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

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

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

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

lemma R2-wait'-cond: \(R2(P \circ Q) = R2(P) \circ R2(Q)
by (simp add: R2-def R2s-wait'-cond R1-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: \(R(P \vdash R1(Q) \circ R) = R(P \vdash Q \circ R)
by (metis (no-types, lifting) R1-idem R1-wait'-cond RH-design-export-R1)

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

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

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

4.3.1 Definitions

abbreviation \( \text{pre}_s \equiv [\text{ok} \mapsto s \text{ true}, \text{ok}' \mapsto s \text{ false}, \text{wait} \mapsto s \text{ false}] \)

abbreviation \( \text{cmt}_s \equiv [\text{ok} \mapsto s \text{ true}, \text{ok}' \mapsto s \text{ true}, \text{wait} \mapsto s \text{ false}] \)

abbreviation \( \text{peri}_s \equiv [\text{ok} \mapsto s \text{ true}, \text{ok}' \mapsto s \text{ true}, \text{wait} \mapsto s \text{ false}, \text{wait}' \mapsto s \text{ true}] \)

abbreviation \( \text{post}_s \equiv [\text{ok} \mapsto s \text{ true}, \text{ok}' \mapsto s \text{ true}, \text{wait} \mapsto s \text{ false}, \text{wait}' \mapsto s \text{ false}] \)

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

definition [upred-defs]: \( \text{pre}_R(P) = (\neg \text{ npre}_R(P)) \)

definition [upred-defs]: \( \text{cmt}_R(P) = R1(\text{cmt}_s \uparrow P) \)

definition [upred-defs]: \( \text{peri}_R(P) = R1(\text{peri}_s \uparrow P) \)

definition [upred-defs]: \( \text{post}_R(P) = R1(\text{post}_s \uparrow P) \)

4.3.2 Unrestriction laws

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

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

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

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

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

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

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

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

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

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

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

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

4.3.3 Substitution laws

lemma pres-design: $pre_s \uparrow (P \vdash Q) = (\neg pre_s \uparrow P)$
by (simp add: design-def presR-def usubst)

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

lemma post-design: $post_s \uparrow (P \vdash Q \land R) = post_s \uparrow (P \Rightarrow R)$
by (simp add: design-def usubst wait'-cond-def)

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

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

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

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

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

lemma post-s-R1 [usubst]: post_s † R1(P) = R1(post_s † 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(R_s(P ⊢ Q)) = R1(R2c(pre_s † P))
by (simp add: RHS-def usubst R3h-def pre_R-def pre_s-design R1-negate-R1 R2c-not rea-not-def)

lemma rea-cmt-RHS-design: cmt_R(R_s(P ⊢ Q)) = R1(R2c(cmt_s † (P ⇒ R)))
by (simp add: RHS-def usubst cmt_R-def cmt_s-design R1-idem)

lemma rea-peri-RHS-design: peri_R(R_s(P ⊢ Q ⋄ R)) = R1(R2c(peri_s † (P ⇒ r R)))
by (simp add: RHS-def usubst peri_R-def R3h-def peri_s-design, rel-auto)

lemma rea-post-RHS-design: post_R(R_s(P ⊢ Q ⋄ R)) = R1(R2c(post_s † (P ⇒ r R)))
by (simp add: RHS-def usubst post_R-def R3h-def post_s-design, rel-auto)

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

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

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

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

4.3.4 Healthiness laws

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

lemma R1-R2s-cmt-SRD:
assumes P is SRD
shows \( R1(R2s(cmt_R(P))) = cmt_R(P) \)
by (metis (no-types, lifting) R1-R2c-commute R1-R2s-R2c R1-idem R2c-idem SRD-reactive-design assms rca-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 \( \text{thesis} \).
qed

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

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

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

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

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

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

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

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

lemma R2c-preR:
  $P$ is SRD $\implies$ $R2c(pre$_R(P)) = pre$_R(P)$
  by (simp add: Healthy-def R2c-idem SRD-reactive-design rea-pre-RHS-design)

lemma R2c-postR:
  $P$ is SRD $\implies$ $R2c(post$_R(P)) = post$_R(P)$
  by (metis no-types, lifting R1-R2c-commute R2-def R2s-idem)

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

lemma wpR-trace-ident-pre [wp]:
  ($\langle trˋ =_u tr \mid [I]_R \rangle$ wp$_c$ 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:
  ($\langle trˋ =_u tr \mid [I]_R \rangle ;; peri$_R(P) = peri$_R(P)$
  by (rel-auto)

lemma trace-ident-left-postR:
  ($\langle trˋ =_u tr \mid [I]_R \rangle ;; post$_R(P) = post$_R(P)$
  by (rel-auto)
lemma trace-ident-right-postR:
\[ \text{post}_R(P) \, ; \, (\$tr' = u \, \& \, [\text{II}] R) = \text{post}_R(P) \]
by (rel-auto)

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

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

lemma postR-R2-closed [closure]: P is SRD \implies \text{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]:
\[ \text{cmt}_R P = \text{peri}_R P \hat{\diamond} \text{post}_R P \]
by (rel-auto)

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

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

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

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

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

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

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

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

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

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

lemma postR-srdes-skip [rdes]: post_R(\Pi_R) = (\$tr' =_u \$tr \land [\Pi]_R)
  by (rel-auto)

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

lemma periR-INF [rdes]: peri_R(\bigcap A) = (\bigvee P \in A \cdot peri_R(P))
  by (rel-auto)

lemma postR-INF [rdes]: post_R(\bigcap A) = (\bigvee P \in A \cdot post_R(P))
  by (rel-auto)

lemma preR-UINF [rdes]: pre_R(\bigcap i \cdot P(i)) = (\bigcup i \cdot pre_R(P(i)))
  by (rel-auto)

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

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

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

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

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

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

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

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

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

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

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

lemma preR-inf [rdes]: pre_R(P \cap Q) = (pre_R(P) \land pre_R(Q))
  by (rel-auto)
lemma periR-inf [rdes]: \( \peri_R(P \cap Q) = (\peri_R(P) \lor \peri_R(Q)) \)
by (rel-auto)

lemma postR-inf [rdes]: \( \post_R(P \cap Q) = (\post_R(P) \lor \post_R(Q)) \)
by (rel-auto)

lemma preR-SUP [rdes]: \( \pre_R(\bigcup A) = (\forall P \in A \cdot \pre_R(P)) \)
by (rel-auto)

lemma periR-SUP [rdes]: \( A \neq \{\} \implies \peri_R(\bigcup A) = (\bigwedge P \in A \cdot \peri_R(P)) \)
by (rel-auto)

lemma postR-SUP [rdes]: \( A \neq \{\} \implies \post_R(\bigcup A) = (\bigwedge P \in A \cdot \post_R(P)) \)
by (rel-auto)

4.4 Formation laws

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

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

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

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

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

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

lemma SRD-reactive-design-alt:
assumes \( P = SRD \)
shows \( \R_s(\pre_R(P) \vdash cmt_R(P)) = P \)
proof
have \( \R_s(\pre_R(P) \vdash cmt_R(P)) = \R_s((\neg P_f^j) \vdash P_f^j) \)

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

lemma SRD-reactive-tri-design-lemma:
    \text{SRD}(P) = \text{R}_s((\neg P f) \vdash P f[\text{true}/$\text{wait}$] \circ P f[\text{false}/$\text{wait}$])
    by (simp add: SRD-RH-design-form wait'-cond-split)

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

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

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

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

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

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

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

4.4.1 Order laws

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

theorem RH-tri-design-composition:
assumes $\text{ok}' \not\subseteq P$ $\exists \text{ok} \ subseteq Q$ $\exists \text{ok} \not\subseteq R$ $\exists \text{ok} \subseteq S_1$ $\exists \text{ok} \subseteq S_2$
shows $(\text{RH}(\vdash Q_1 \circ Q_2) \land \text{RH}(\vdash S_1 \circ S_2) = RH(\neg (R_1 \land \neg R_2s P) \circ R_1 \land \neg R_2s Q) \land \neg $ $\neg \text{ok}' \not\subseteq R_1 \land \neg R_2s R_1 \circ R_1 \land \neg R_2s Q_2) = (R_1 \land \neg R_2s Q_1) \circ (R_1 \land \neg R_2s Q_2)$
proof
have 1:(\neg ((R_1 \land \neg R_2s Q_1) \land \neg \text{ok}' \not\subseteq R_1 \land \neg R_2s R_1)) =
(\neg ((R_1 \land \neg R_2s Q_2) \land \neg \text{ok}' \not\subseteq R_1 \land \neg R_2s R_1))
by (metis (no-types, hide-lams) R1-extend-cond R2s-conj R2s-not R2s-wait ' wait'-cond-false)
have 2: (R1 \land \neg R_2s Q_1) ;; (\neg \text{ok}' \not\subseteq R_1 \land \neg R_2s Q_1) =
(R_1 \land \neg R_2s Q_1) \circ (R_1 \land \neg R_2s Q_2) 
by (rel-auto)
also have ... = (R1 \land \neg R_2s Q_1) \circ (R_1 \land \neg R_2s Q_2)
by (simp add: lift-des-skip-dr-unit-unrest unrest)
finally show \?thesis .
qed

moreover have (R1 \land \neg R_2s Q_1) ;; (\neg \text{ok}' \not\subseteq R_1 \land \neg R_2s R_1) \circ (R_1 \land \neg R_2s Q_2)
= ((R1 \land \neg R_2s Q_1) \circ (R1 \land \neg R_2s Q_2))
by (rel-auto)
also have ... = (R1 \land \neg R_2s Q_1) \circ (R1 \land \neg R_2s Q_2)
by (simp add: /cond-def usubst unrest assms)
finally show \?thesis .
qed

moreover have (R1 \land \neg R_2s Q_1) \land \neg \text{ok}' \not\subseteq R_1 \land \neg R_2s Q_1) \circ (R1 \land \neg R_2s Q_2)
= (R1 \land \neg R_2s Q_1) \circ (R1 \land \neg R_2s Q_2)
by (simp add: /cond-def usubst unrest assms)

finally show \?thesis .
qed
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 P) wp, false ∧ Q wp_r, R) ⊢ (Q ;; ;)
apply (subst RH-design-composition)
apply (simp-all add: assms unrest Healthy-if closure wp)
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 ;; ;))
by (simp add: R1-design-composition-RR assms unrest Healthy-if closure wp)

theorem RHS-tri-design-composition:
  assumes $ok’$ $P$ $ok’$ $Q_1$ $ok’$ $Q_2$ $ok’$ $R$ $ok’$ $S_1$ $ok’$ $S_2$
$wait’$ $R$ $wait’$ $Q_2$ $wait’$ $S_1$ $wait’$ $S_2$
shows $(R_1(P ⊢ Q) ;; R_1(S ⊢ S_1) =
R_1(¬ (R_1 (¬ R_2s P) ;; R_1 true) ∧ ¬ (R_1(R_2s Q_2) ;; R_1 (¬ R_2s R))/)
((∃ $st’$ · $Q_1) ∨ (R_1 (R_2s Q_2);; R_1 (R_2s S_1)) ⊆ ((R_1 (R_2s Q_2);; R_1 (R_2s S_2))))))$
proof –
  have 1:¬ ((R1 (R2s (Q1 ∪ Q2)) ∧ ¬ $wait’$ ;; R1 (¬ R2s R)) =
(¬ ((R1 (R2s Q_2) ∧ ¬ $wait’$);; R1 (¬ R2s R)))
  by (metis (no-types, hide-lams) R1-extend-cond R2s-conj R2s-not R2s-wait’ wait’-cond-false)
  have 2: (R1 (R2s (Q1 ∪ Q2));; (∃ $st$ · ([I]D) < $wait$ > R1 (R2s (S1 ∪ S2)))) =
((∃ $st’$ · R1 (R2s Q_1)) ∨ (R1 (R2s Q_2);; R1 (R2s S_1)) ⊆ (R1 (R2s Q_2);; R1 (R2s S_2)))
  proof –
    have (R1 (R2s Q_1);; ($wait$ ∧ (∃ $st$ · [I]D) < $wait$ > R1 (R2s S_1) ;; R1 (R2s S_2))) =
(∃ $st’$ · ((R1 (R2s Q_1)) ∧ $wait’$))
    proof –
      have (R1 (R2s Q_1);; ($wait$ ∧ (∃ $st$ · [I]D) < $wait$ > R1 (R2s S_1) ;; R1 (R2s S_2))) =
(R1 (R2s Q_1);; ($wait$ ∧ (∃ $st$ · [I]D)))
      by (rel-auto, blast+)
also have ... = (R1 (R2s Q_1);; (∃ $st$ · [I]D)) ∧ $wait’$
      by (rel-auto)
also from assms(2) have ... = (∃ $st’$ · ((R1 (R2s Q_1)) ∧ $wait’$))
      by (rel-auto, blast)
    finally show ?thesis .
  qed

qed
moreover have \( (R_1 (R_2s Q_2) :: (\neg \$ \text{wait} \land (\exists \$ \text{st} \cdot [\Pi]^D) \circ \$ \text{wait} \triangleright R_1 (R_2s S_1) \circ R_1 (R_2s S_2))) \)
\[ = ((R_1 (R_2s Q_2) :: (R_1 (R_2s S_1) \circ R_1 (R_2s S_2))) \]

proof 

have \( (R_1 (R_2s Q_2) :: (\neg \$ \text{wait} \land (\exists \$ \text{st} \cdot [\Pi]^D) \circ \$ \text{wait} \triangleright R_1 (R_2s S_1) \circ R_1 (R_2s S_2))) \)
\[ = ((R_1 (R_2s Q_2) :: (\neg \$ \text{wait} \land (\exists \$ \text{st} \cdot [\Pi]^D) \circ \$ \text{wait} \triangleright R_1 (R_2s S_1) \circ R_1 (R_2s S_2))) \]
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 \( \ldots = ((R_1 (R_2s Q_2))[false/\$ \text{wait}] :: (R_1 (R_2s S_1) \circ R_1 (R_2s S_2))[false/\$ \text{wait}] \)
by (metis false-alt-def seqr-right-one-point upred-eq-false wait-vw b-lens)

also have \( \ldots = ((R_1 (R_2s Q_2)) :: (R_1 (R_2s S_1) \circ R_1 (R_2s S_2)) \)
by (simp add: wait'-cond-def usubst unrest assms)

finally show ?thesis.

qed

moreover

have \( ((R_1 (R_2s Q_1) \land \$ \text{wait}^\dagger) \lor ((R_1 (R_2s Q_2)) :: (R_1 (R_2s S_1) \circ R_1 (R_2s S_2))) \)
\[ = (R_1 (R_2s Q_1) \lor (R_1 (R_2s Q_2) :: R_1 (R_2s S_1)) \circ ((R_1 (R_2s Q_2) :: R_1 (R_2s S_2))) \]
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 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 \( 3: (R_1 (R_2s Q_2) \land \neg \$ \text{wait}^\dagger) :: R_1 (\neg R_2s R) = R_1 (R_2s Q_2) :: R_1 (\neg R_2s R) \)
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 \( \$ \text{ok}^\dagger \not\in P \$ \text{ok}^\dagger \not\in Q_1 \$ \text{ok}^\dagger \not\in Q_2 \$ \text{ok}^\dagger \not\in R \$ \text{ok}^\dagger \not\in S_1 \$ \text{ok}^\dagger \not\in S_2 \)
\( \$ \text{wait}^\dagger \not\in R \$ \text{wait}^\dagger \not\in Q_2 \$ \text{wait}^\dagger \not\in S_1 \$ \text{wait}^\dagger \not\in S_2 \)
\( P \text{ is } R_2c \quad Q_1 \text{ is } R_1 \quad Q_2 \text{ is } R_2c \quad Q_2 \text{ is } R_1 \quad Q_2 \text{ is } R_2c \)
\( R \text{ is } R_2c \quad S_1 \text{ is } R_1 \quad S_1 \text{ is } R_2c \quad S_2 \text{ is } R_2c \quad S_2 \text{ is } R_2c \)

shows \( R_\dagger (P \triangleright Q_1 \circ Q_2) :: R_\dagger (R \triangleright S_1 \circ S_2) = \)
\( R_\dagger (((\neg r) \text{ wp}_r \text{ false} \land Q_2 \text{ wp}_r R) \triangleright (((\exists \$ \text{st}^\dagger \cdot Q_1) \cap (Q_2 :: S_1)) \circ (Q_2 :: S_2))) \)
is ?lhs = \(?rhs\)

proof

have \( ?lhs = R_\dagger (((\neg r) (\triangleright P) :: R_1 \text{ true} \land \neg Q_2 :: R_1 (\neg R) \triangleright (((\exists \$ \text{st}^\dagger \cdot Q_1) \cap Q_2 :: S_1) \circ Q_2 :: S_2)) \)

39
lemma RHS-tri-design-composition: assumes $\$ok \not \in P$ $\$ok \not \in Q$ $\$ok \not \in Q_2$ $\$ok \not \in S_1$ $\$ok \not \in S_2$ $\$wait \not \in R$ $\$wait \not \in Q_2$ $\$wait \not \in S_1$ $\$wait \not \in S_2$ $P$ is $\$rc$ $Q_1$ is $\$rr$ $Q_2$ is $\$rr$ $R$ is $\$rr$ $S_1$ is $\$rr$ $S_2$ is $\$rr$ shows $\text{R}_s((\neg_p P) \land Q_2) \Rightarrow ((\exists \$st' \cdot Q_1) \land (Q_2 :: S_1)) \Rightarrow (Q_2 :: S_2))$ proof have $\text{R}_s((\neg_p P) \land Q_2) \Rightarrow ((\exists \$st' \cdot Q_1) \land (Q_2 :: S_1)) \Rightarrow (Q_2 :: S_2))$ by (simp all add: RHS-tri-design-composition-up def assms unrest) finally have $(P \land Q_2, R) \Rightarrow ((Q_1 \land (Q_2 :: S_1)) \Rightarrow (Q_2 :: S_2))$ by (simp add: assms wp def ex unrest rel auto) finally show $?thesis$. qed

lemma RHS-tri-normal-design-composition: assumes $(R \rightarrow Q_1) \Rightarrow (Q_2 \Rightarrow Q_2)$ $(R \rightarrow S_1) \Rightarrow (S_2 \Rightarrow S_2)$ $(R \rightarrow (\neg_p P)) = R \neg (\neg P)$ $(P \land Q_2, R) \Rightarrow ((Q_1 \land (Q_2 :: S_1)) \Rightarrow (Q_2 :: S_2))$ proof have $(P \land Q_2, R) \Rightarrow ((Q_1 \land (Q_2 :: S_1)) \Rightarrow (Q_2 :: S_2))$ by (simp add: assms unrest) finally show $?thesis$. qed

lemma RHS-tri-design-right-unit-lemma: assumes $(\neg_p R1) \Rightarrow (R1) \Rightarrow (\neg_p R1)$ $(\neg_p R1 \land Q_2) \Rightarrow ((\exists \$st' \cdot Q_1) \Rightarrow (R1 (R2s R) \Rightarrow (R1 (R2s R) (R \land [I]_R}))$ proof have $(\neg_p R1 \land Q_2) \Rightarrow ((\exists \$st' \cdot Q_1) \Rightarrow (R1 (R2s R) \Rightarrow (R1 (R2s R) (R \land [I]_R))))$ by (simp add: assms unrest) finally show $?thesis$. qed

by (simp add: RHS-tri-design-composition assms Healthy-if $\$rc$-healthy-$\$rc$ disj-upred-def) (metis (no-types, hide-lams) R1-negate-R1 $\$rc$-healthy-$\$rc$ assms(11,16)) also have $?thesis$. by (rel auto) finally show $?thesis$. qed

theorem RHS-tri-design-composition-RR-up: assumes $P$ is $\$rr$ $Q_1$ is $\$rr$ $Q_2$ is $\$rr$ $R$ is $\$rr$ $S_1$ is $\$rr$ $S_2$ is $\$rr$ shows $\text{R}_s((\neg_p P) \land Q_2) \Rightarrow ((\exists \$st' \cdot Q_1) \land (Q_2 :: S_1)) \Rightarrow (Q_2 :: S_2))$ (is $?ths = ?rhs$) by (simp add: RHS-tri-design-composition-up add: closure assms unrest $\$rr$-implies-$\$rc$)
also have ... = R_u ((¬ R1 (¬ R2s P) ;; R1 true) ⊔ (∃ $st^\prime \cdot Q) \circ R1 (R2s R))

proof -
from assms(3,4) have (R1 (R2s R) ;; R1 (R2s ($tr^\prime =_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 ... = R_u((¬ (¬ P) ;; R1 true) ⊔ (∃ $st^\prime \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 ... = R_u((¬ _r (¬ _r P) ;; true_r) ⊔ (∃ $st^\prime \cdot Q) \circ R))
by (rel-auto)
finally show ?thesis .

qed

lemma SRD-composition-up:
assumes P is SRD Q is SRD
shows (P ;; Q) = R_u ((¬ R R R P) wp_r false \land post_R P wp_r pre_R Q) ⊔
  ((∃ $st^\prime \cdot peri_R P) \lor (post_R P ; peri_R Q)) \circ (post_R P ;; post_R Q))
(is ?lhs = ?rhs)

proof -
  have (P ;; Q) = (R_u (pre_R (P) ⊔ peri_R (P) \circ post_R (P)) ;; R_u (pre_R (Q) \circ peri_R (Q) \circ post_R (Q)))
    by (simp add: SRD-reactive-tri-design assms(1) assms(2))
  also from assms
  have ... = ?rhs
    by (simp add: RHS-tri-design-composition-up disj-upred-def unrest assms closure)
  finally show ?thesis .

qed

4.6 Refinement introduction laws

lemma RHS-tri-design-refine:
assumes P_1 is RR P_2 is RR P_3 is RR Q_1 is RR Q_2 is RR Q_3 is RR
shows R_u(P_1 \circ P_2 \circ P_3) ⊆ R_u(Q_1 \circ Q_2 \circ Q_3) \iff 'P_1 \Rightarrow Q_1'; \land 'P_1 \land Q_2 \Rightarrow P_2'; \land 'P_1 \land Q_3 \Rightarrow P_3';
(is ?lhs = ?rhs)

proof -
  have ?lhs \iff 'P_1 \Rightarrow Q_1'; \land 'P_1 \land Q_2 \Rightarrow P_2'; \land 'P_1 \land Q_3 \Rightarrow P_3';
    by (simp add: RHS-design-refine assms closure RR-implies-R2c unrest ex-unrest)
  also have ... \iff 'P_1 \Rightarrow Q_1'; \land '((P_1 \land Q_2) \circ (P_1 \land Q_3) \Rightarrow P_2 \circ P_3)' \land '(P_1 \land Q_2) \circ (P_1 \land Q_3) \Rightarrow P_2 \circ P_3''
    by (rel-auto)
  also have ... \iff 'P_1 \Rightarrow Q_1'; \land '((P_1 \land Q_2) \circ (P_1 \land Q_3) \Rightarrow P_2 \circ P_3)[true/$\cdot wait']'' \land '((P_1 \land Q_2) \circ (P_1 \land Q_3) \Rightarrow P_2 \circ P_3)''
    by (rel-auto, metis)
  also have ... \iff ?rhs
    by (simp add: usubst unrest assms)
  finally show ?thesis .

qed

lemma srdes-tri-refine-intro:
assumes 'P_1 \Rightarrow P_2'; 'P_1 \land Q_2 \Rightarrow Q_1'; 'P_1 \land R_2 \Rightarrow R_1'
shows R_u(P_1 \circ Q_1 \circ R_1) \subseteq R_u(P_2 \circ Q_2 \circ R_2)
using assms
by (rule-tac srdes-refine-intro, simp-all, rel-auto)

lemma srdes-tri-eq-intro:
assumes $P_1 = Q_1 \land P_2 = Q_2 \land P_3 = Q_3$
shows $R_s(P_1 \equiv R_2 \equiv R_3) = R_s(Q_1 \equiv Q_2 \equiv Q_3)$
using assms by (simp)

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

lemma SRD-peri-under-pre:
assumes $P \in SRD \iff \#wait' \not\equiv prec(R)$
shows $(prec(R) \Rightarrow \peri R(P)) = peri R(P)$
proof -

have $peri R(P) =$
peri $R_s(prec(R(P) \equiv peri R(P) \equiv post(R(P))))$
by (simp add: SRD-reactive-tri-design assms)

also have ... = $(prec R \Rightarrow peri R P)$
by (simp add: rea-pre-RHS-design rea-peri-RHS-design assms)
unrest usubst R1-peri-SRD R2c-preR R1-rea-impl R2c-rea-impl R2c-periR)

finally show ?thesis ..

qed

lemma SRD-post-under-pre:
assumes $P \in SRD \iff \#wait' \not\equiv prec(R)$
shows $(prec(R) \Rightarrow \peri R(P)) = peri R(P)$
proof -

have $peri R(P) =$
peri $R_s(prec(R(P) \equiv peri R(P) \equiv post(R(P))))$
by (simp add: SRD-reactive-tri-design assms)

also have ... = $(prec R \Rightarrow peri R P)$
by (simp add: rea-pre-RHS-design rea-post-RHS-design assms)
unrest usubst R1-post-SRD R2c-preR R1-rea-impl R2c-rea-impl R2c-postR)

finally show ?thesis ..

qed

lemma SRD-refine-intro:
assumes $P \in SRD Q \iff \#wait' \not\equiv prec(R)$
shows $\prec prec(R)(P) \Rightarrow \peri R(P) \land peri R(Q) \Rightarrow peri R(P \land peri R(Q)) \Rightarrow peri R(P \land post(R(Q)) \Rightarrow peri R(P)\equiv$
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 $P \in SRD Q \iff \#wait' \not\equiv prec(R)$
shows $\prec prec(R)(P) \Rightarrow \peri R(P) \subseteq \peri R(Q) \Rightarrow \peri R(P) \subseteq \peri R(Q)$
using assms by (rule-tac SRD-refine-intro, simp-all add: refBy-order)

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

lemma SRD-srdes-skip [closure]: $I_R$ is SRD
   by (simp add: srdes-skip-def RHS-design-is-SRD unrest)

lemma SRD-seqr-closure [closure]:
  assumes $P$ is SRD $Q$ is SRD
  shows $(P ;; Q)$ is SRD
proof
  have $(P ;; Q) = R_s((\neg r \text{ pre}_R P \wedge \text{ post}_R P \wedge r \text{ pre}_R Q) \vdash
     ((\exists \text{ stmt} \cdot \text{ peri}_R P) \lor \text{ post}_R P ;; \text{ peri}_R Q) \circ \text{ post}_R P ;; \text{ post}_R Q)$
    by (simp add: SRD-composition-wp assms (1) assms (2))
  also have ... is SRD
    by (rule RHS-design-is-SRD, simp-all add: wp-rea-def unrest)
  finally show ?thesis .
qed

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

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

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

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

4.8 Distribution laws

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

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

by \( (\text{simp add: RHS-design-sup, rel-auto}) \)

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

by \( (\text{simp add: RHS-tri-design-sup conj-upred-def}) \)

lemma \textit{SRD-UINF} [rdes-def]:
\[
\text{assumes } A \neq \emptyset \subseteq [\text{SRD}_i] \\
\text{shows } \bigwedge A = R_s((\bigwedge P \in A \cdot \perin_R(P)) \vdash (\bigwedge P \in A \cdot \post_R(P)) \land (\bigwedge P \in A \cdot \pre_R(P)))
\]

proof –
\[
\text{have } \bigwedge A = R_s((\pre_R(\bigwedge P \in A \cdot \perin_R(P)) \vdash \perin_R(\bigwedge P \in A \cdot \pre_R(P)))
\]

by \( (\text{metis SRD-as-reactive-tri-design assms srdes-hcond-def}) \)

also have \( \ldots = R_s((\bigwedge P \in A \cdot \perin_R(P)) \vdash (\bigwedge P \in A \cdot \post_R(P))) \)

by \( (\text{simp add: preR-INF periR-INF postR-INF assms}) \)

finally show \ ?thesis .

qed

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

by \( (\text{simp add: design-UINF-mem assms, rel-auto}) \)

lemma \textit{SRD-UINF-mem}: 
\[
\text{assumes } A \neq \emptyset \subseteq [\text{SRD}_i] \\
\text{shows } \bigwedge i \in A \cdot P i = R_s(\bigwedge P(i) \vdash (\bigwedge i \in A \cdot \pre_R(P i)) \land (\bigwedge i \in A \cdot \post_R(P i)))
\]

(is \ ?lhs = ?rhs)

proof –
\[
\text{have } \ldots = (\bigwedge P \in A)
\]

by \( (\text{rel-auto}) \)

also have \( \ldots = R_s(\bigwedge P(i) \vdash (\bigwedge P(i)) \land (\bigwedge P(i)) \land (\bigwedge P(i)) \land (\bigwedge P(i))) \)

by \( (\text{rel-auto}) \)

also have \( \ldots = \text{\textit{thesis}} \).

qed

lemma \textit{RHS-tri-design-UINF-ind} [rdes-def]:
\[
(\bigwedge i \cdot R_s(P_1(i) \vdash P_2(i) \land P_3(i))) = R_s((\bigwedge i \cdot P_1(i) \vdash (\bigwedge i \cdot \perin_R(P i)) \land (\bigwedge i \cdot \post_R(P i))))
\]

by \( (\text{rel-auto}) \)

lemma \textit{cond-srea-form} [rdes-def]:
\[
R_s(P \vdash Q_1 \land Q_2) \land R_s(Q_1 \land Q_2) \land R_s(Q_1 \land Q_2)
\]

by \( (\text{rel-auto}) \)

also have \( \ldots = R_s(P \vdash Q_1 \land Q_2) \land R_s(Q_1 \land Q_2) \land R_s(Q_1 \land Q_2) \)

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

\textbf{4.9 Algebraic laws}

\textbf{lemma} \textit{SRD-cond-srea [closure]}:
\quad assumes \textit{P} is \textit{SRD} \textit{Q} is \textit{SRD}
\quad shows \textit{P} \triangleleft b \triangleright \textit{Q} is \textit{SRD}
\quad \text{proof} --
\quad have \textit{P} \triangleleft b \triangleright \textit{Q} = \text{R}_a:\ ((\text{pre}_R(\textit{P}) \circ \text{peri}_R(\textit{P}) \circ \text{post}_R(\textit{P})) \triangleleft b \triangleright \text{R}_a:\ ((\text{pre}_R(\textit{Q}) \circ \text{peri}_R(\textit{Q}) \circ \text{post}_R(\textit{Q}))\))
\quad \quad \text{by (simp add: SRD-reactive-tri-design assms)}
\quad also have \ldots = \text{R}_a:\ ((\text{pre}_R \textit{P} \triangleleft b \triangleright \text{pre}_R \textit{Q}) \vdash (\text{peri}_R \textit{P} \triangleleft b \triangleright \text{peri}_R \textit{Q}) \circ (\text{post}_R \textit{P} \triangleleft b \triangleright \text{post}_R \textit{Q}))
\quad \quad \text{by (simp add: cond-srea-form)}
\quad also have \ldots \text{is} \textit{SRD}
\quad \quad \text{by (simp add: RHS-tri-design-is-SRD lift-cond-srea-def unrest)}
\quad finally show \textit{thesis}.
qed

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

\textbf{lemma} \textit{SRD-right-unit-tri-lemma}:
\quad assumes \textit{P} is \textit{SRD}
\quad shows \textit{P} \vdash \text{II}_R = \text{R}_a\: (\neg, \text{pre}_R \textit{P}) \vdash \textit{wp} \circ \textit{false} \vdash (\exists \textit{st}^\prime \cdot \text{peri}_R \textit{P}) \circ \text{post}_R \textit{P})
\quad \text{by (simp add: SRD-composition-up closure rdes wp C1 R1-negate-R1 R1-false rpred trace-ident-left-periR trace-ident-left-postR SRD-reactive-tri-design assms)}

\textbf{lemma} \textit{Miracle-left-zero}:
\quad assumes \textit{P} is \textit{SRD}
\quad shows \textit{Miracle} \vdash \textit{P} = \textit{Miracle}
\quad \text{proof} --
\quad have \textit{Miracle} \vdash \textit{P} = \text{R}_a\: (\text{true} \vdash \textit{false}) \vdash \text{R}_a\: (\text{pre}_R(\textit{P}) \vdash \text{cmt}_R(\textit{P}))
\quad \quad \text{by (simp add: Miracle-def SRD-reactive-design-alt assms)}
\quad also have \ldots = \text{R}_a\: (\text{true} \vdash \textit{false})
\quad \quad \text{by (simp add: RHS-design-composition unrest R1-false R2s-false R2s-true)}
\quad also have \ldots = \text{Miracle}
\quad \quad \text{by (simp add: Miracle-def)}
\quad finally show \textit{thesis}.
qed

\textbf{lemma} \textit{Chaos-left-zero}:
\quad assumes \textit{P} is \textit{SRD}
\quad shows (\textit{Chaos} \vdash \textit{P}) = \textit{Chaos}
\quad \text{proof} --
\quad have \textit{Chaos} \vdash \textit{P} = \text{R}_a\: (\text{false} \vdash \textit{true}) \vdash \text{R}_a\: (\text{pre}_R(\textit{P}) \vdash \text{cmt}_R(\textit{P}))

45
by (simp add: Chaos-def SRD-reactive-design-alt assms)
also have ... = Rₜ(\true \land \neg (\true \land \neg \wait)) ;; R₁ (\neg R₂s (\prec R P)) \vdash R₁ true ;; (\exists \st \cdot [H]D \land \wait \triangleright R₁ (R₂s (\cm R P)))
  by (simp add: RHS-design-conj-impl-R1
also have ... = Rₜ(\false \land \neg (\true \land \neg \wait)) ;; R₁ (\neg R₂s (\prec R P)) \vdash R₁ true ;; (\exists \st \cdot [H]D \land \wait \triangleright R₁ (R₂s (\cm R P)))
  by (simp add: RHS-design-conj-neg-R1-pre)
also have ... = Rₜ(\false \vdash \true)
  by (simp add: design-false-pre)
also have ... = Rₜ(\false \vdash \true)
  by (simp add: design-def)
also have ... = Chaose
  by (simp add: Chaos-def)
finally show \?thesis .
qed

lemma SRD-right-Chaos-tri-lemma:
  assumes P is SRD
  shows P ;; Chaos = Rₜ(\false \land \post R P \land \false) \vdash (\exists \st \cdot \peri R P) \land \false)
  by (simp add: SRD-composition-up closure rdes assms wp, rel-auto)

lemma SRD-right-Miracle-tri-lemma:
  assumes P is SRD
  shows P ;; Miracle = Rₜ(\false \land \post R P \land \false) \vdash (\exists \st \cdot \peri R P) \land \false)
  by (simp add: SRD-composition-up closure rdes assms wp, rel-auto)

Stateful reactive designs are left unital

overloading
  srdes-unit :: (SRDES, ('s, 't::trace, 'α) rthys) uthy ⇒ ('s, 't, 'α) hrel-rsp

begin
  definition srdes-unit :: (SRDES, ('s, 't::trace, 'α) rthys) uthy ⇒ ('s, 't, 'α) hrel-rsp where
  srdes-unit ⋆ = II R
end

interpretation srdes-left-unital: utp-theory-left-unital SRDES
by (unfold-locales, simp-all add: srdes-hcond-def rdes-unit-def SRD-seq-refinement SRD-seq-rs RDES SRD-left-unit)

4.10 Recursion laws

lemma mono-srd-iter:
  assumes mono \lambda X. Rₜ(\prec R (F X) \vdash \peri R (F X) \land \post R (F X)))
  shows mono \lambda X. Rₜ(\prec R (F X) \vdash \peri R (F X) \land \post R (F X)))
  (rule mono)
  apply (rule rdes-left-unital)
  apply (meson assms(1) monoE preR-antitone utp-pred-laws.le-infl)
  apply (meson assms(1) monoE periR-monotone utp-pred-laws.le-infl)
  apply (meson assms(1) monoE postR-monotone utp-pred-laws.le-infl)
  done

done

lemma mu-srd-SRD:
  assumes mono \lambda X. Rₜ(\prec R (F X) \vdash \peri R (F X) \land \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 \textit{mu-srd-iter}:

assumes \( \mu X \cdot R_a(p_{\text{pre}}(F(X)) \vdash \text{peri}_R(F(X)) \circ \text{post}_R(F(X))) = F(\mu X \cdot R_a(p_{\text{pre}}(F(X)) \vdash \text{peri}_R(F(X)) \circ \text{post}_R(F(X))) \)

apply (subst gfp-unfold)
apply (simp add: mono-srd-iter assms)
apply (subst SRD-as-reactive-tri-design \[\text{THEN sym}\])
using Healthy-func assms (1) assms (2) mu-srd-SRD
apply blast

done

lemma \textit{mu-srd-form}:

assumes \( \mu X \cdot R_a(p_{\text{pre}}(F(X)) \vdash \text{peri}_R(F(X)) \circ \text{post}_R(F(X))) \)

shows \( \mu R F = (\mu X \cdot R_a(p_{\text{pre}}(F(X)) \vdash \text{peri}_R(F(X)) \circ \text{post}_R(F(X))) \)

proof

have 1: \( F(\mu X \cdot R_a(p_{\text{pre}}(F(X)) \vdash \text{peri}_R(F(X)) \circ \text{post}_R(F(X))) \) is SRD
by (simp add: Healthy-apply-closed assms (1) assms (2) mu-srd-SRD)

have 2: Mono \( _{\text{uthy-order SRDES}} F \)
by (simp add: assms (1) mono-Monotone-utp-order)

hence 3: \( \mu R F = F(\mu R F) \)
by (simp add: srdes-theory-continuous.LFP-unfold \[\text{THEN sym}\] assms)

using SRD-reactive-tri-design by force

hence \( (\mu X \cdot R_a(p_{\text{pre}}(F(X)) \vdash \text{peri}_R(F(X)) \circ \text{post}_R(F(X))) \subseteq F(\mu R F) \)
by (simp add: 2 srdes-theory-continuous.weak.LFP-lemma3 gfp-upperbound assms)

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

qed

lemma \textit{Monotonic-SRD-comp} \[\text{closure}\]: \textit{Monotonic} \( (op ;; P \circ \text{SRD}) \)
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
  UTP-KAT.utp-kleene

begin

This additional healthiness condition is analogous to H3

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

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

proof

have a: \( II_R ;; II_R = II_R \)
by (simp add: SRD-left-unit SRD-srdes-skip)

show \textit{?thesis}
by (simp add: RD3-def seqr-assoc a)
lemma RD3-Idempotent [closure]: Idempotent RD3
  by (simp add: Idempotent-def RD3-idem)

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

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

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

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

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

lemma RD3-intro-pre:
  assumes \( P \ is \ SRD \ (\neg_r \ pre_R(P)) ; \ true_r = (\neg_r \ pre_R(P)) \ $st' \ $ perir_R(P) \)
  shows \( P \ is \ RD3 \)
proof
  have \( RD3(P) = R_a ((\neg_r \ pre_R(P)) \ wp_r \ false \vdash (\exists \ $st' \ \cdot \ perir_R(P) \circ post_R(P)) \)
    by (simp add: RD3-def SRD-right-unit-tri-lemma assms)
  also have \( ... = R_a ((\neg_r \ pre_R(P)) \ wp_r \ false \vdash perir_R(P) \circ post_R(P)) \)
    by (simp add: assms(3) ex-unrest)
  also have \( ... = R_a ((\neg_r \ pre_R(P)) \ wp_r \ false \vdash cmt_R(P)) \)
    by (simp add: wait'-cond-peri-post-cmt)
  also have \( ... = R_a (\ pre_R(P) \vdash \ cmt_R(P) \)
    by (simp add: assms(2) rpred wp-rea-def R1-preR)
  finally show ?thesis
    by (metis Healthy-def SRD-as-reactive-design assms(1))
qed

lemma RHS-tri-design-right-unit-lemma:
  assumes \$sk' \ $ P \ $ok' \ $ Q \ $ok' \ $ R \ $wait' \ $ R \)
  shows \( R_a(P \vdash Q \circ R) ; II_R = R_a((\neg_r \ (\neg_r \ P)) ; \ true_r) \vdash ((\exists \ $st' \ \cdot \ Q) \circ R)) \)
proof
  have \( R_a(P \vdash Q \circ R) ; II_R = R_a(P \vdash Q \circ R) ; R_a(true \vdash (\neg \ (\neg_r \ R2s(P)) ; R1 true) \vdash (\exists \ $st' \ \cdot \ Q) \circ (R1 \ R2s(R) ; R1 \ R2s(\neg \ (\neg_r \ R2s(R))) \)) \)
    by (simp add: rhs-des-skip-tri-design, rel-auto)
  also have \( ... = R_a ((\neg_r \ R1 \ (\neg_r \ R2s(P)) ; R1 true) \vdash (\exists \ $st' \ \cdot \ Q) \circ (R1 \ R2s(R) ; R1 \ R2s(\neg \ (\neg_r \ (\neg_r \ R2s(P)))) \)) \)
    by (simp add: rhs-tri-design-composition assms unrest R2s-true R1-false R2s-false)
also have \( R_s ((\neg R1 \ (\neg R2s \ P)) \supset R1 \ true) \vdash (\exists \ \$st' \cdot Q) \circ R1 \ (R2s \ R)) \)

proof -
from assms(3,4) have \( (R1 \ (R2s \ R)) \supset \ (R1 \ (R2s \ (\$tr' = \_u \ \$tr \ \circ [\_I \ _R]))) = R1 \ (R2s \ R) \)
by (rel-auto, metis (no-types, lifting) minus-zero-eq, meson order-refl trace-class.diff-cancel)
thus \(?thesis\)
by simp
qed

also have \( R_s ((\neg P) \supset R1 \ true) \vdash ((\exists \ \$st' \cdot Q) \circ R) \)
by (metis (no-types, lifting) R1-R2s-R1-true-lemma R1-R2s-R2c R2c-not RHS-design-R2c-pre RHS-design-neg-R1-pre
RHS-design-post-R1 RHS-design-post-R2s)

also have \( R_s ((\neg_r (\neg_r \ P)) \supset true_r) \vdash ((\exists \ \$st' \cdot Q) \circ R) \)
by (rel-auto)
finally show \(?thesis\).
qed

lemma \( RHS\)-tri-design-RD3-intro:
assumes \( \$ok' \ # P \ \$ok' \ # Q \ \$ok' \ # R \ \$st' \ # Q \ \$wait' \ # R \)
P is \( R1 \ (\neg_r \ P) \supset true_r = (\neg_r \ P) \)
shows \( R_s (P \vdash Q \circ R) \) is RD3
apply (simp add: Healthy-def RD3-def)
apply (subst RHS-tri-design-right-unit-lemma)
apply (simp-all add: assms ex-unrest rpred)
done

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

lemma \( R1\)-right-unit-lemma:
\[
\begin{array}{ll}
\text{oaut}( \ P \# B; \ oaut \ \# \ E) & \Rightarrow (\neg_r \ b \lor \$tr \ \#_u \ e \leq_u \$tr') \supset R1(true) = (\neg_r b \lor \$tr \ \#_u \ e \leq_u \$tr')
\end{array}
\]
by (rel-auto, blast, metis (no-types, lifting) dual-order.trans)

lemma \( RHS\)-tri-design-RD3-intro-form:
assumes \( \text{oaut}( \ P \# B; \ oaut \ \# E) \equiv (\neg_r \ b \lor \$tr \ \#_u \ e \leq_u \$tr \cdot) \supset R1(true) = (\neg_r b \lor \$tr \ \#_u \ e \leq_u \$tr) \)
shows \( R_s ((\neg_r (\neg_r \ P) \supset true_r) \vdash \ (\exists \ \$st' \cdot Q) \circ R) \)
apply (rule RHS-tri-design-RD3-intro)
apply (simp-all add: assms unrest closure rpred)
apply (subt R1-right-unit-lemma)
apply (simp-all add: assms unrest)
done

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

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

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

lemma NSRD-form:
  \( NSRD(P) = R_\alpha((\neg_r (\neg_r \text{pre}_R(P))) :: R1 \text{true}) \vdash ((\exists \text{st}' \cdot \text{peri}_R(P)) \circ \text{post}_R(P)) \)

proof
  have \( NSRD(P) = RD3(SRD(P)) \)
    by (metis RD3-def RD3-idem RD3-left-subsumes-RD2 SRD-def SRD-idem SRD-seqr-closure SRD-srdes-skip)
  also have \( \cdots = RD3(R_\alpha(\text{pre}_R(P) \Rightarrow \text{peri}_R(P)) \circ \text{post}_R(P)) :: II_R \)
    by (simp add: SRD-as-reactive-tri-design)
  also have \( \cdots = R_\alpha((\neg_r (\neg_r \text{pre}_R(P))) :: R1 \text{true}) \vdash ((\exists \text{st}' \cdot \text{peri}_R(P)) \circ \text{post}_R(P)) \)
    by (simp add: RD3-def)
  also have \( \cdots = R_\alpha((\neg_r (\neg_r \text{pre}_R(P))) :: R1 \text{true}) \vdash ((\exists \text{st}' \cdot \text{peri}_R(P)) \circ \text{post}_R(P)) \)
    by (clarsimp simp add: RD-elim closure)
  finally show \( \neg \exists \text{thesis} \)

qed

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

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

proof
  have \( NSRD \ (\bigcap A) = (\bigcap (\text{NSRD} \ 'A)) \)
    by (simp add: ContinuousD NSRD-Continuous assms(2))
  also have \( \cdots = (\bigcap A) \)
    by (simp only: Healthy-carrier-image assms)
  finally show \( \exists \text{thesis} \) by (simp add: Healthy-def)

qed

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

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

lemma NSRD-neg-pre-unit:
  assumes \( P \) is NSRD
  shows \( (\neg_r \text{pre}_R(P)) :: \text{true}_r = (\neg_r \text{pre}_R(P)) \)

proof
  have \( (\neg_r \text{pre}_R(P)) = (\neg_r \text{pre}_R(R_\alpha((\neg_r (\neg_r \text{pre}_R(P))) :: R1 \text{true})) \vdash ((\exists \text{st}' \cdot \text{peri}_R(P)) \circ \text{post}_R(P)))) \)
    by (clarsimp simp only: Healthy-def assms)
  also have \( \cdots = R1 \cdot (R2c ((\neg_r \text{pre}_R P) :: R1 \text{true})) \)

50
by (simp add: rea-pre-RHS-design R1-negate-R1 R1-idem R1-rea-not' R2c-rea-not usubst rpred unrest closure)
also have ... = (¬r pre R P) ;; R1 true
  by (simp add: R1-R2c-seqr-distribute closure assms)
finally show ?thesis
  by (simp add: rea-not-def)
qed

lemma NSRD-neg-pre-left-zero:
  assumes P is NSRD Q is R1 Q is RD1
  shows (¬ r pre (¬ r pre R P)) ;; Q = (¬ r pre R (¬ r pre R P))
  by (metis (no-types, hide-lams) NSRD-neg-pre-unit RD1-left-zero assms)

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

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

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

lemma NSRD-alt-def: NSRD(P) = RD3(SRD(P))
by (metis NSRD-def RD1-RD3-commute RD3-left-subsumes-RD2 SRD-def comp-eq-dest-lhs)
lemma preR-RR [closure]: $P$ is NSRD $\Rightarrow$ $pre_R(P)$ is RR
by (rule RR-intro, simp-all add: closure unrest)

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

lemma NSRD-intro:
assumes $P$ is SRD $(\neg_r pre_R(P)) :: true_r = (\neg_r pre_R(P)) \triangleright \$st' \n peri_R(P)$
shows $P$ is NSRD
proof
  have NSRD($P$) = $R_s((\neg_r (\neg_r pre_R(P))) :: R1 true) \vdash ((\exists st' \cdot peri_R(P)) \circ post_R(P))$
  by (simp add: NSRD-form)
  also have ... = $R_s(pre_R P \vdash peri_R P \circ post_R P)$
  by (simp add: assms ex-unrest rpred closure)
  also have ... = $P$
  by (simp add: SRD-reactive-tri-design assms (1))
  finally show $?thesis$
  using Healthy-def by blast
qed

lemma NSRD-intro':
assumes $P$ is R2 $P$ is R3h $P$ is RD1 $P$ is RD3
shows $P$ is NSRD
by (metis (no-types, hide-lams) Healthy-def NSRD-def R1-R2c-is-R2 RHS-def assms comp-apply)

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

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

lemma SRD-RD3-implies-NSRD:
$\langle [ P is SRD; P is RD3 ] \rangle \Rightarrow P$ 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 $\iff (\langle P$ is SRD $\rangle \land (\neg_r pre_R(P)) :: R1(true) = (\neg_r pre_R(P)) \land (\$st' \n peri_R(P)))$
by (meson NSRD-intro NSRD-is-SRD NSRD-neg-pre-unit NSRD-st'-unrest-peri)

lemma NSRD-is-RD3 [closure]:
assumes $P$ is NSRD
shows $P$ is RD3
by (simp add: NSRD-is-SRD NSRD-neg-pre-unit NSRD-st'-unrest-peri RD3-intro-pre assms)

lemma NSRD-refine-elim:
assumes
\[ P \subseteq Q \text{ is NSRD} \]
\[ \begin{align*}
\text{[}& \, \text{pre}_R(P) \Rightarrow \text{pre}_R(Q) ; \, \text{pre}_R(P) \land \text{peri}_R(Q) \Rightarrow \text{peri}_R(P) ; \, \text{pre}_R(P) \land \text{post}_R(Q) \Rightarrow \text{post}_R(P) \text{]} \quad \Rightarrow R \\
\text{shows} \ R \\
\text{proof} & \quad \text{have} \ R'_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \circ \text{post}_R(P)) \subseteq R'_s(\text{pre}_R(Q) \vdash \text{peri}_R(Q) \circ \text{post}_R(Q)) \\
& \quad \text{by (simp add: NSRD-is-SRD SRD-reactive-tri-design assms(1) assms(2) assms(3))} \\
\text{hence} \ 1: \text{pre}_P \Rightarrow \text{pre}_R Q \text{ and} 2: \text{pre}_P \land \text{peri}_R Q \Rightarrow \text{peri}_R P \text{ and} 3: \text{pre}_R P \land \text{post}_R Q \Rightarrow \text{post}_R P \\
& \quad \text{by (simp-all add: RHS-tri-design-refine assms closure)} \\
& \quad \text{with assms(4) show ?thesis} \\
& \quad \text{by simp} \\
\text{qed}
\end{align*} \]

**Lemma NSRD-right-unit:** \( P \text{ is NSRD } \Rightarrow P ; \ I_R = P \)
by (metis Healthy-if NSRD-is-RD3 RD3-def)

**Lemma NSRD-composition-wp:**
\[ \text{assumes} \ P \text{ is NSRD} \ Q \text{ is SRD} \\
\text{shows} \ P ; Q = R_s((\text{pre}_R P \land \text{post}_R P \ \text{wp}_r \ \text{pre}_R Q) \vdash (\text{peri}_R P \lor (\text{post}_R P ; \ \text{peri}_R Q)) \circ (\text{post}_R P ; ; \ \text{post}_R Q)) \]
by (simp add: SRD-composition-wp assms NSRD-is-SRD wp-rea-def NSRD-neg-pre-unit NSRD-st'-unrest-peri R1-extend-conj R1-rea-not R2c-preR ex-unrest rpred)

**Lemma preR-NSRD-seq-lemma:**
\[ \text{assumes} \ P \text{ is NSRD} \ Q \text{ is SRD} \\
\text{shows} \ R1 (R2c (postR P ; ; (\neg_r \ \text{pre}_R Q))) = postR P ; ; (\neg_r \ \text{pre}_R Q) \]
proof –
\[ \text{have} \ \text{post}_R P ; ; (\neg_r \ \text{pre}_R Q) = R1(R2c(postR P)) ; ; R1(R2c(\neg_r \ \text{pre}_R Q)) \]
by (simp add: NSRD-is-SRD R1-R2c-post-RHS R1-rea-not R2c-preR R2c-rea-not assms(1) assms(2))
\[ \text{also have} \ ... = R1 (R2c (postR P ; ; (\neg_r \ \text{pre}_R Q))) \]
by (simp add: R1-seqr R2c-R1-seq calculation)
\[ \text{finally show} \ ?\text{thesis} \] ..
\[ \text{qed} \]

**Lemma preR-NSRD-seq [rdes]:**
\[ \text{assumes} \ P \text{ is NSRD} \ Q \text{ is SRD} \\
\text{shows} \ \text{pre}_R(P ; ; Q) = (\text{pre}_R P \land \text{post}_R P \ \text{wp}_r \ \text{pre}_R Q) \]
by (simp add: NSRD-composition-wp assms rea-pre-RHS-design usubst unrest wp-rea-def R2c-disj R1-disj R2c-and R2c-preR R1-R2c-commute[THEN sym] R1-extend-conj' R1-idem R2c-not closure)
(metis (no-types, lifting) Healthy iff NSRD-is-SRD R1-R2c-commute R1-R2c-seqR-distribute R1-seqR-closure assms(1) assms(2) postR-R2c-closed postR-SRD-R1 preR-R2c-closed rea-not-R1 rea-not-R2c)

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

**Lemma postR-NSRD-seq [rdes]:**
\[ \text{assumes} \ P \text{ is NSRD} \ Q \text{ is NSRD} \\
\text{shows} \ \text{post}_R(P ; ; Q) = ((\text{pre}_R P \land \text{post}_R P \ \text{wp}_r \ \text{pre}_R Q) \Rightarrow (\text{post}_R P ; ; \ \text{post}_R Q)) \]

53
by (simp add: NSRD-composition-up assms closure rea-post-RHS-design usbst unrest wp-rea-def
R1-extend-conj' R1-disj R1-R2c-seqr-distribute R2c-disj R2c-and R2c-rea-impl R1-rea-impl'
R2c-preR R2c-periR R1-rea-not' R2c-rea-not)

lemma NSRD-seqr-closure [closure]:
assumes P is NSRD Q is NSRD
shows (P ;; Q) is NSRD
proof –
  have (∼r post R wp_r preR Q) ;; true_r = (∼r post R wp_r preR Q)
  by (simp add: wp-rea-def rpred assms closure seqr-assoc NSRD-neg-pre-unit)
  moreover have $st' ∼r pre R wp_r preR Q ⇒ r peri R P ⊃ peri R Q
  by (simp add: unrest assms wp-rea-def)
ultimately show ?thesis
  by (rule-tac NSRD-intro, simp-all add: seqr-or-distl NSRD-neg-pre-unit assms closure rdes unrest)
qed

lemma RHS-tri-normal-design-composition:
assumes $ok' ∼r Q $ok' Q_2 $ok' R $ok' $S_1 $ok' $S_2
$wait' R $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' ∼r Q_1
shows R_s((P ∨ Q_1 ∨ Q_2) ;; R_s(R ∨ S_1 ∨ S_2)
= R_s((P ∨ Q_2 wp_r R) ⊢ (Q_1 ∨ (Q_2 ;; S_1)) ⊂ (Q_2 ;; S_2))
proof –
  have R_s((P ∨ Q_1 ∨ Q_2) ;; R_s(R ∨ S_1 ∨ S_2) =
    R_s((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_s((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_s((P ∨ Q_1 ∨ Q_2) ;; R_s(R ∨ S_1 ∨ S_2)
= R_s((P ∨ Q_2 wp_r R) ⊢ (Q_1 ∨ (Q_2 ;; S_1)) ⊂ (Q_2 ;; S_2))
proof –
  have R1 (¬ P) ;; R1 true_r = R1(¬ P)
  using RC-implies-RC1[OF assms(1)]
  by (simp add: Healthy-def RC1-def rea-not-def)
      (metis R1-negate-R1 R1-seqr wpt-pred-laws.double-compl)
  thus ?thesis
  by (simp add: RHS-tri-normal-design-composition assms closure unrest RR-implies-R2c)
qed

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

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

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

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

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

lemma NSRD-right-Miracle-tri-lemma:
assumes \( P \) is NSRD
shows \( P ; ; \) Miracle = \( R_\ast (pre_R P \vdash peri_R P \circ \text{false}) \)
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 \( R_\ast (pre_R P \vdash peri_R P \circ \text{false}) \subseteq R_\ast (pre_R P \vdash peri_R P \circ \text{false}) \)
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 =\( \Rightarrow P \upower n \) is NSRD
by (metis upower-Suc-NSRD-closed upred-semiring.power-Suc)

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

lemma preR-power:
assumes \( P \) is NSRD
shows \( pre_R(P ; ; P \uplus^n) \) = \( \bigsqcup_{i \in 0..n}. \) \( (post_R(P) \uplus \text{false}) \) wp_r \( (pre_R(P)) \)
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 \( \Rightarrow P \uplus^n \) is NSRD
by (metis upower-Suc-NSRD-closed upred-semiring.power-Suc)

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

lemma preR-power:
assumes \( P \) is NSRD
shows \( pre_R(P ; ; P \uplus^n) \) = \( \bigsqcup_{i \in 0..n}. \) \( (post_R(P) \uplus \text{false}) \) wp_r \( (pre_R(P)) \)
proof (induct n)
  case 0
  then show \( \) case
  by (simp add: wp closure)
next
case (Suc n) note hyp = this
have \( \text{pre}_R (P \cdot (\text{Suc} \ n + 1)) = \text{pre}_R (P \cdot P ^ {\cdot (n+1)}) \)
by (simp add: upred-semiring-power-Suc)
also have \( \ldots = (\text{pre}_R P \land \text{post}_R P \ \text{wp}_r \ \text{pre}_R (P \cdot (\text{Suc} \ n))) \)
using \(\text{NSRD-iff assms preR-NSRD-seq upower-Suc-NSRD-closed} \) by fastforce
also have \( \ldots = (\text{pre}_R P \land \text{post}_R P \ \text{wp}_r (\bigsqcup \ i \in \{0\ldots n\}. \ \text{post}_R P ^ {\cdot i} \ \text{wp}_r \ \text{pre}_R P)) \)
by (simp add: hyp upred-semiring-power-Suc)
also have \( \ldots = (\text{pre}_R P \land (\bigsqcup \ i \in \{0\ldots n\}. \ \text{post}_R P \ \text{wp}_r (\text{post}_R P ^ {\cdot i} \ \text{wp}_r \ \text{pre}_R P))) \)
by (simp add: wp)
also have \( \ldots = (\text{pre}_R P \land (\bigsqcup \ i \in \{0\ldots n\}. \ \text{post}_R P ^ {\cdot (i+1)} \ \text{wp}_r \ \text{pre}_R P)) \)
proof
– have \( \bigwedge \ i. \ R1 (\text{post}_R P ^ {\cdot i} \ ; ; (\neg r \ \text{pre}_R P)) = (\text{post}_R P ^ {\cdot i} \ ; ; (\neg r \ \text{pre}_R P)) \)
by (induct-tac \ i, simp-all add: closure Healthy-if assms)
thus \( ?\text{thesis} \)
by (simp add: wp-rea-def upred-semiring-power-Suc seqr-assoc rpred closure assms)
qed
also have \( \ldots = (\text{post}_R P ^ {\cdot 0} \ \text{wp}_r \ \text{pre}_R P \land (\bigsqcup \ i \in \{0\ldots n\}. \ \text{post}_R P ^ {\cdot (i+1)} \ \text{wp}_r \ \text{pre}_R P)) \)
by (simp add: wp assms closure)
also have \( \ldots = (\text{post}_R P ^ {\cdot 0} \ \text{wp}_r \ \text{pre}_R P \land (\bigsqcup \ i \in \{1\ldots \text{Suc} \ n\}. \ \text{post}_R P ^ {\cdot i} \ \text{wp}_r \ \text{pre}_R P)) \)
proof
– have \( (\bigsqcup \ i \in \{0\ldots n\}. \ \text{post}_R P ^ {\cdot (i+1)} \ \text{wp}_r \ \text{pre}_R P)) = (\bigsqcup \ i \in \{1\ldots \text{Suc} \ n\}. \ \text{post}_R P ^ {\cdot i} \ \text{wp}_r \ \text{pre}_R P) \)
by (rule cong[\alpha 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 \( \ldots = (\bigsqcup \ i \in \text{insert} \ 0 \ \{1\ldots \text{Suc} \ n\}. \ \text{post}_R P ^ {\cdot i} \ \text{wp}_r \ \text{pre}_R P) \)
by (simp add: conj-upred-def)
also have \( \ldots = (\bigsqcup \ i \in \{0\ldots \text{Suc} \ n\}. \ \text{post}_R P ^ {\cdot i} \ \text{wp}_r \ \text{pre}_R P) \)
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 \( P \) \( \text{is NSRD} \)
shows \( \text{pre}_R (P ; ; P ^ {\cdot n}) = (\bigsqcup \ i \in \{0\ldots n\} \cdot (\text{post}_R (P ^ {\cdot i}) \ ; ; \ \text{wp}_r (\text{pre}_R (P)))) \)
by (simp add: \text{preR-power assms USUP-as-Inf[THEN sym]})

lemma \( \text{preR-power-Suc} \) [rdes]:
assumes \( P \) \( \text{is NSRD} \)
shows \( \text{pre}_R (P ^ {\cdot (\text{Suc} \ n)}) = (\bigsqcup \ i \in \{0\ldots n\} \cdot (\text{post}_R (P ^ {\cdot i}) \ ; ; \ \text{wp}_r (\text{pre}_R (P)))) \)
by (simp add: upred-semiring-power-Suc rdes assms)
declare \( \text{upred-semiring-power-Suc} \) [simp]

lemma \( \text{periR-power} \):
assumes \( P \) \( \text{is NSRD} \)
shows \( \text{peri}_R (P ; ; P ^ {\cdot n}) = (\text{pre}_R (P ^ {\cdot (\text{Suc} \ n)}) \Rightarrow_r (\bigsqcup \ i \in \{0\ldots n\}, \ \text{post}_R (P ^ {\cdot i}) ; ; \ \text{peri}_R (P))) \)
proof (induct \ n)
  case \( 0 \)
  then show \( ?\text{case} \)
  by (simp add: \text{NSRD-is-SRD NSRD-wait'-unrest-pre SRD-peri-under-pre assms})
next
  case \( (\text{Suc} \ n) \) \text{ note hyp = this} 
  have \( \text{peri}_R (P ^ {\cdot (\text{Suc} \ n + 1)}) = \text{peri}_R (P ; ; P ^ {\cdot (n+1)}) \)
  by (simp)
  also have \( \ldots = (\text{pre}_R (P ^ {\cdot (\text{Suc} \ n + 1)}) \Rightarrow_r (\text{peri}_R P \lor \text{post}_R P ; ; \ \text{peri}_R (P ; ; P ^ {\cdot n}))) \)
by (simp add: closure assms rdes)
also have ... = (pre_r (P ` (Suc n + 1)) ⇒ r (peri_R P ∨ post_R P ;; (pre_R (P ` (Suc n)) ⇒ r (∏ i ∈ {0..n}. post_R P ` i) ;; peri_R P)))
  by (simp only: hyp)
also have ... = (pre_R P ⇒ r peri_R P ∨ (post_R P wp_r pre_R (P ;; P ` n) ⇒ r, post_R P ;; (pre_R (P ;; P ` n) ⇒ r (∏ i ∈ {0..n}. post_R P ` i) ;; peri_R P)))
  by (simp add: rdes closure assms, rel-blast)
also have ... = (pre_R P ⇒ r peri_R P ∨ (post_R P wp_r pre_R (P ;; P ` n) ⇒ r, post_R P ;; (∏ i ∈ {0..n}. post_R P ` i) ;; peri_R P)))
proof –
  have (∏ i ∈ {0..n}. post_R P ` i) is R1
  by (simp add: NSRD-is-SRD R1-Continuous R1-power Sup-Continuous-closed assms postR-SRD-R1)
  hence 1:(∏ i ∈ {0..n}. post_R P ` i) ;; peri_R P) is R1
  by (simp add: closure assms)
  hence (pre_R (P ;; P ` n) ⇒ r (∏ i ∈ {0..n}. post_R P ` i) ;; peri_R P) is R1
  by (simp add: closure)
  hence (post_R P wp_r pre_R (P ;; P ` n) ⇒ r post_R P ;; (pre_R (P ;; P ` n) ⇒ r (∏ i ∈ {0..n}. post_R P ` i) ;; peri_R P))
    = (post_R P wp_r pre_R (P ;; P ` n) ⇒ r R1(post_R P) ;; R1(pre_R (P ;; P ` n) ⇒ r (∏ i ∈ {0..n}. post_R P ` i) ;; peri_R P))
    by (simp add: Healthy-if R1-post-SRD assms closure)
    thus ?thesis
    by (simp only: wp-rea-impl-lemma, simp add: Healthy-if 1, simp add: R1-post-SRD assms closure)
qed
also have ... = (pre_R P ∧ post_R P wp_r pre_R (P ;; P ` n) ⇒ r, peri_R P ∨ post_R P ;; (∏ i ∈ {0..n}. post_R P ` i) ;; peri_R P))
  by (pred-auto)
also have ... = (pre_R P ∧ post_R P wp_r pre_R (P ;; P ` n) ⇒ r, peri_R P ∨ (∏ i ∈ {0..n}. post_R P ` (Suc i)) ;; peri_R P))
  by (simp add: seq-Sup-distr seqr-assoc[THEN sym])
also have ... = (pre_R P ∧ post_R P wp_r pre_R (P ;; P ` n) ⇒ r, peri_R P ∨ (∏ i ∈ {1..Suc n}. post_R P ` i) ;; peri_R P))
proof –
  have (∏ i ∈ {0..n}. post_R P ` Suc i) = (∏ i ∈ {1..Suc n}. post_R P ` i)
    apply (rule cong[of Sup], auto)
    apply (metis atLeast0AtMost atMost-iff image-Suc-atLeastAtMost rev-image-eqI upred-semiring.power-Suc)
    usingSuc-le-D apply fastforce
  done
  thus ?thesis by simp
qed
also have ... = (pre_R P ∧ post_R P wp_r pre_R (P ;; P ` n) ⇒ r (∏ i ∈ {0..Suc n}. post_R P ` i) ;; peri_R P)
  by (simp add: SUP-atLeastAtMost-first unif-or seqr-or-distr seq-or-distr)
also have ... = (pre_R (P * (Suc (Suc n))) ⇒ r (∏ i ∈ {0..Suc n}. post_R P ` i) ;; peri_R P))
  by (simp add: rdes closure assms)
finally show ?case by (simp)
qed

lemma periR-power' [rdes]:

57
assumes $P$ is NSRD
shows $\forall n\leq\alpha (\text{pre}_R(P\cdot\alpha) \Rightarrow \forall i\in\{0..n\} \cdot \text{post}_R(P \cdot \alpha) \Rightarrow \text{peri}_R(P))$
by (simp add: periR-power assms UINF-as-Sup[THEN sym])

\textbf{lemma} periR-power-Suc [rdes]:
\begin{itemize}
  \item assumes $P$ is NSRD
  \item shows $\forall n\leq\alpha (\text{pre}_R(P\cdot\alpha) \Rightarrow \forall i\in\{0..n\} \cdot \text{post}_R(P \cdot \alpha) \Rightarrow \text{peri}_R(P))$
\end{itemize}
by (simp add: rdes assms)

\textbf{lemma} postR-power [rdes]:
\begin{itemize}
  \item assumes $P$ is NSRD
  \item shows $\forall n\leq\alpha (\text{pre}_R(P\cdot\alpha) \Rightarrow \forall i\in\{0..n\} \cdot \text{post}_R(P \cdot \alpha) \Rightarrow \text{peri}_R(P))$
\end{itemize}
by (simp add: rdes assms)

\textbf{next}
\begin{itemize}
  \item case $(\text{Suc } n)$ \textbf{note} hyp = this
  \item have $\forall n\leq\alpha (\text{pre}_R(P\cdot\alpha) \Rightarrow \forall i\in\{0..n\} \cdot \text{post}_R(P \cdot \alpha) \Rightarrow \text{peri}_R(P))$
  \item by (simp)
\end{itemize}
\begin{itemize}
  \item also have $\forall n\leq\alpha (\text{pre}_R(P\cdot\alpha) \Rightarrow \forall i\in\{0..n\} \cdot \text{post}_R(P \cdot \alpha) \Rightarrow \text{peri}_R(P))$
  \item by (simp add: closure assms rdes)
\end{itemize}
\begin{itemize}
  \item also have $\forall n\leq\alpha (\text{pre}_R(P\cdot\alpha) \Rightarrow \forall i\in\{0..n\} \cdot \text{post}_R(P \cdot \alpha) \Rightarrow \text{peri}_R(P))$
  \item by (simp only: hyp)
\end{itemize}
\begin{itemize}
  \item also have $\forall n\leq\alpha (\text{pre}_R(P\cdot\alpha) \Rightarrow \forall i\in\{0..n\} \cdot \text{post}_R(P \cdot \alpha) \Rightarrow \text{peri}_R(P))$
  \item 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)
\end{itemize}
\begin{itemize}
  \item also have $\forall n\leq\alpha (\text{pre}_R(P\cdot\alpha) \Rightarrow \forall i\in\{0..n\} \cdot \text{post}_R(P \cdot \alpha) \Rightarrow \text{peri}_R(P))$
  \item by (pred-auto)
\end{itemize}
\begin{itemize}
  \item also have $\forall n\leq\alpha (\text{pre}_R(P\cdot\alpha) \Rightarrow \forall i\in\{0..n\} \cdot \text{post}_R(P \cdot \alpha) \Rightarrow \text{peri}_R(P))$
  \item by (simp add: rdes closure assms)
\end{itemize}
finally show $\forall case$ by (simp)
\textbf{qed}

\textbf{lemma} postR-power-Suc [rdes]:
\begin{itemize}
  \item assumes $P$ is NSRD
  \item shows $\forall n\leq\alpha (\text{pre}_R(P\cdot\alpha) \Rightarrow \forall i\in\{0..n\} \cdot \text{post}_R(P \cdot \alpha) \Rightarrow \text{peri}_R(P))$
\end{itemize}
by (simp add: rdes assms)

\textbf{lemma} power-rdes-def [rdes-def]:
\begin{itemize}
  \item assumes $P$ is RC $Q$ is RR $R$ is RR $\exists i \neq Q$
  \item shows $\forall n\leq\alpha (\text{pre}_R(P\cdot\alpha) \Rightarrow \forall i\in\{0..n\} \cdot \text{post}_R(P \cdot \alpha) \Rightarrow \text{peri}_R(P))$
\end{itemize}
by (simp add: rdes assms)

\textbf{proof} (induct $n$
\begin{itemize}
  \item case 0
  \item then show $\forall case$
  \item by (simp add: wp assms closure)
\end{itemize}
\begin{itemize}
  \item next
  \item case $(\text{Suc } n)$
\end{itemize}
have 1: \((P \land (\bigsqcup i \in \{0..n\} \cdot R \wp_{p_e} (R ^\ast i \wp_{p_e} P))) = (\bigsqcup i \in \{0..\text{Suc} n\} \cdot R ^\ast i \wp_{p_e} P)\)
(is \(\text{lhs} = \text{rhs}\))

proof –

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

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

proof –

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

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

proof –

have \(\prod i \in \{0..n\} \cdot R ^\ast i) ;; Q = (\prod i \in \{0..n\}. R ^\ast i) ;; Q\)
  by (simp add: UINF-as-Sup-collect)
also have \(\ldots = (\prod i \in \{0..n\}. R ^\ast i) ;; Q\)
  by (simp add: atLeastAtMost-insertL)
also have \(\ldots = (Q \lor (\prod i \in \{1..n\}. R ^\ast i) ;; Q)\)
  by (metis (no-types, lifting) SUP-insert disj-upred-def seqr-left-unit seq-or-distl upred-seming.power-0)
also have \(\ldots = (Q \lor (\prod i \in \{0..<n\}. R ^\ast \text{Suc} i) ;; Q)\)
  by (simp add: UINF-as-Sup-collect)
also have \(\ldots\) is RR
  by (simp add: UINF-as-Sup-collect)
finally show \(\text{thesis}\)
qed

from 1 2 3 Suc show \(\text{?case}\)
  by (simp add: Suc RHS-tri-normal-design-composition' closure assms \(\wp\))
qed
declare upred-semiring.power-Suc [simp del]

theorem uplus-rdes-def [rdes-def]:
  assumes P is RC Q is RR R is RR $st \in Q$
  shows \( (R \uplus (P \circ Q))^+ = R \uplus (R^+ \circ P) \)
proof - 
  have 1: \( (\prod i \cdot R^i) \vdash Q = R^+ \vdash Q \)
  by (metis (no-types) RA1 assms(2) rea-unit-unit rrrel-thy X st-healthy I)
  show ?thesis 
  by (simp add: uplus-power-def seq-UINF-distr wp closure assms rdes-def)
  (by simp add: uplus-power-def seq-UINF-distr wp closure assms rdes-def)
qed

5.1 UTP theory

typedcl NSRDES

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

overloading
  nsrdes-hcond \equiv \upred-hcond :: (NSRDES, ('s', 't::trace', 'α') rsp) uthy \Rightarrow ((s, t, α) \upred \times (s, t, α) \upred)
health
  nsrdes-unit \equiv \upred-unit :: (NSRDES, ('s', 't::trace', 'α') rsp) uthy \Rightarrow ('s', 't', 'α') hrel-rsp
begin 
  definition nsrdes-hcond :: (NSRDES, ('s', 't::trace', 'α') rsp) uthy \Rightarrow ((s, t, α) \upred \times (s, t, α) \upred)
health where
  [upred-defs]: nsrdes-hcond T = NSRD
  definition nsrdes-unit :: (NSRDES, ('s', 't::trace', 'α') rsp) uthy \Rightarrow ('s', 't', 'α') hrel-rsp where
  [upred-defs]: nsrdes-unit T = NSRD
end

interpretation nsr-thy: utp-theory-kleene UTHY(NSRDES, ('s', 't::trace', 'α') rsp)
rewrites \( \prod P. P \in \text{carrier} \text{ (uthy-order NSRDES)} \iff P \text{ is NSRD} \)
and P is \( \text{NSRDES} \iff P \text{ is NSRD} \)
and \( (\mu X \cdot F (H_{\text{NSRDES}} X)) = (\mu X \cdot F (\text{NSRD} X)) \)
and carrier (uthy-order NSRDES) \rightarrow carrier (uthy-order NSRDES) \equiv [\text{NSRD}]_H \rightarrow [\text{NSRD}]_H
and \( [H_{\text{NSRDES}}]_H \rightarrow [H_{\text{NSRDES}}]_H \equiv [\text{NSRD}]_H \rightarrow [\text{NSRD}]_H 
and \( \top_{\text{NSRDES}} = \text{Miracle} \)
and \( \bot_{\text{NSRDES}} = H_R \)
and le (uthy-order NSRDES) \equiv \upred \subseteq 
proof - 
  interpret lat: utp-theory-continuous UTHY(NSRDES, ('s', 't', 'α') rsp)
  by (unfold-locale, simp-all add: nsrdes-hcond-def nsrdes-unit-def closure Healthy-if)
  show 1: \( \top_{\text{NSRDES}} = (\text{Miracle} :: ('s', 't', 'α') hrel-rsp) \)
  by (metis NSRD-Miracle NSRD-is-SRD lat.top-healthy lat.utp-theory-continuous-axioms nsrdes-hcond-def srdes-theory-continuous.meet-top apred-semiring.add-commute utp-theory-continuous.meet-top)
  thus utp-theory-kleene UTHY(NSRDES, ('s', 't', 'α') rsp)
  by (unfold-locale, simp-all add: nsrdes-hcond-def nsrdes-unit-def closure Healthy-if Miracle-left-zero SRD-left-unit NSRD-right-unit)
qed (simp-all add: nsrdes-hcond-def nsrdes-unit-def closure Healthy-if)

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

abbreviation TestR \equiv (test_R) where
\[ \text{test}_R P \equiv \text{u test} \text{NSRDES } P \]

**abbreviation** \( \text{StarR} :: (s, \text{'}t:\text{trace, 'a}) \text{ hrel-rsp } \Rightarrow (s, \text{'}t, \text{'}a) \text{ hrel-rsp } (\cdot^R [999] 999) \)  

where  
\[ \text{StarR} P \equiv P^\ast \text{NSRDES} \]

**lemma** \( \text{StarR-rdes-def [rdes-def]}:\)

- **assumes** \( P \text{ is RC } Q \text{ is RR } R \text{ is RR } \$st' \not\in Q \)
- **shows**  
  \[ (R_\ast(P \vdash Q \circ R))^{*R} = R_\ast((R^{*r} \text{ wp}_r P) \vdash R^{*r} :: Q \circ R^r) \]

by (simp add: rrel-thy.Star-alt-def nsrd-thy.\text{pre-RHS-design unst subst unrest rpred disj-upred-def})

end

### 6 Syntax for reactive design contracts

**theory** \( \text{utp-rdes-contracts} \)

- **imports** \( \text{utp-rdes-normal} \)

**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 pericondition that can refer to the trace and before state but not the after state, and \( R \) is a postcondition. Both \( Q \) and \( R \) can refer only to the trace contribution through a HOL variable \( \text{trace} \) which is bound to \( \&tt \).

**definition** \( \text{mk-RD} :: 's \text{ upred } \Rightarrow ('t:\text{trace }\Rightarrow 's \text{ upred }) \Rightarrow ('t \Rightarrow 's \text{ hrel } \Rightarrow ('s, 't, 'a) \text{ hrel-rsp where} \)

\[
\text{mk-RD} P Q R = R_\ast([P]|S < [Q(x)]x<|[x\rightarrow&tt]| \circ [R(x)]x<|[x\rightarrow&tt]|) 
\]

**definition** \( \text{trace-pred} :: ('t:\text{trace }\Rightarrow 's \text{ upred }) \Rightarrow ('s, 't, 'a) \text{ hrel-rsp where} \)

upred-defs: \( \text{trace-pred } P = ([P x]|S < [x\rightarrow&tt]|) \)

**syntax**

- \( \text{-trace-var} :: \text{logic} \)
- \( \text{-mk-RD} :: \text{logic }\Rightarrow \text{logic }\Rightarrow \text{logic }\Rightarrow \text{logic }([/-] \vdash [/-]|\cdot)_R) \)
- \( \text{-trace-var} :: \text{logic }\Rightarrow \text{logic }\Rightarrow \text{logic }(([-]_R) \)

**parse-translation** \( \langle \langle \)

**let**

- fun \( \text{trace-var-tr} [] = \text{Syntax.free trace} \)
- \( \mid \text{trace-var-tr} _- = \text{raise Match;} \)

**in**

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

end

**translations**

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

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

by (simp add: mk-RD-def closure unrest)

**lemma** \( \text{preR-mk-RD [rdes]}: \text{preR}([P \vdash Q(\text{trace}) | R(\text{trace})]|_R) = R1([P]|S <) \)

by (simp add: mk-RD-def rea-pre-RHS-design usubst unrest R2c-not R2c-lift-state-pre)
lemma trace-pred-RR-closed [closure]:
[P trace] t is RR
by (rel-auto)

lemma unrest-trace-pred-st' [unrest]:
$st' \triangleright [P trace] t$
by (rel-auto)

lemma R2c-msubst-tt: R2c (msubst (λx. ⌈Q x⌉ S) & tt) = (msubst (λx. ⌈Q x⌉ S) & tt)
by (rel-auto)

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

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

Refinement introduction law for contracts

lemma RD-contract-refine:
assumes
'Q is SRD' [P1] S < ⇒ preR Q'
'P1 S S S < ∧ periR Q ⇒ [P2 x S S x → & tt] S < '
'P1 S S S < ∧ postR Q ⇒ [P3 x S S x → & tt] S < '
shows [P1 ⊢ P2(trace) | P3(trace)] R ⊑ Q
proof –
have [P1 ⊢ P2(trace) | P3(trace)] R ⊑ R (preR(Q) ⊢ periR(Q) ⊢ postR(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
disj-assoc
conjun-UNIF-dist
conjun-UNIF-ind-dist
seq-or-distl
seq-or-distr
seq-UNIF-distl
seq-UNIF-distl'
seq-UNIF-distr
The following tactic can be used to simply and evaluate reactive predicates.

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

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

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

Tactic to simply the definition of a reactive design

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

Tactic to split a refinement conjecture into three POs

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

Tactic to split an equality conjecture into three POs

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

Tactic to prove a refinement

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

Tactics to prove an equality

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

Via antisymmetry

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

Tactic to calculate pre/peri/postconditions from reactive designs

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

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

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

**end**

8 Reactive design parallel-by-merge

**theory** utp-rdes-parallel

**imports**
  utp-rdes-normal
require that both sides are R3c, and that
begin

lemma skip-rm: \[ \exists \exists (\exists \exists \exists) \]

proof

also have ...

by (rel-auto)

lemma st-U0-alpha: \[ (\exists \exists \exists) \]

by (rel-auto)

definition skip-rm :: (s,t::trace,t,α) rna merge (H_{RM}) where

[upred-defs]: H_{RM} = (\exists \exists \exists)

-definition [upred-defs]: R3hm(M) = (H_{RM} \triangleright \exists \exists \exists \triangleright M)

lemma R3hm-idem: R3hm(R3hm(P)) = R3hm(P)

by (rel-auto)

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

assumes P is R3h Q is R3h M is R3hm

shows (P,\parallel M, Q) is R3h

proof –

have (P,\parallel M, Q) = ((\exists \exists \exists \parallel M, Q)[true/sok, wait] \triangleright (P,\parallel M, Q)\parallel true/sok, wait] \triangleright (P,\parallel M, Q))

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

also have ...

by (rel-auto)

also have ... = ((\exists \exists \exists \parallel M, Q)[true,\parallel wait/sok, wait] \triangleright (P,\parallel M, Q)\parallel true,\parallel wait/sok, wait] \triangleright (P,\parallel M, Q))

by (rel-auto)

finally show \thesis by (simp add: closure assms unrest)

qed

also have ...

by (simp add: closure assms unrest)

proof –

have (P,\parallel M, Q)[true,\parallel wait/sok, wait] = ((\exists \exists \exists \parallel M, Q)[true/sok, wait] \triangleright (P,\parallel M, Q))

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

also have ...

by (rel-blast)

also have ... = ((\exists \exists \exists \parallel M, Q)[true,\parallel wait/sok, wait] \triangleright (P,\parallel M, Q))

by (rel-auto)

finally show ?thesis by simp

qed
also have \( ... = (((\exists \, \text{st} \cdot II) < \text{sk} \circ R1(\text{true})) < \text{sk} \circ (P \parallel_M Q)) \)
  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** 
\( \text{udpred-defs} \): \( RD1m(M) = (M \lor \neg \text{sk} < \land \text{sk} \leq_u \text{sk}' \) \)

**Lemma** \( RD1-par-by-merge \): \[ \text{closure} \]
assumes \( P \) is \( R1 \) \( Q \) is \( R1 \) \( M \) is \( R1m \) \( P \) is \( RD1 \) \( Q \) is \( RD1 \) \( M \) is \( RD1m \)
shows \( (P \parallel_M Q) \) is \( RD1 \)
proof –
  have 1: \( (RD1(R1(P)) \parallel_{RD1m(R1m(M))} RD1(R1(Q)))[false/\text{sk}] = R1(\text{true}) \)
    by (rel-blast)
  have \( (P \parallel_M Q) = (P \parallel_M Q)[true/\text{sk}] < \text{sk} \circ (P \parallel_M Q)[false/\text{sk}] \)
    by (simp add: cond-var-split)
also have \( ... = R1(P \parallel_M Q) < \text{sk} \circ R1(\text{true}) \)
    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 RD1-alt-def assms(3))
finally show \( \text{thesis} \)
    by (simp add: Healthy-def)
qed

**Lemma** \( RD2-par-by-merge \): \[ \text{closure} \]
assumes \( M \) is \( RD2 \)
shows \( (P \parallel_M Q) \) is \( RD2 \)
proof –
  have \( (P \parallel_M Q) = ((P \parallel\textbf{Q}) :: M) \)
    by (simp add: par-by-merge-def)
  also from \( \text{assms} \) have \( ... = ((P \parallel\textbf{Q}) :: (M :: J)) \)
    by (simp add: Healthy-def\' RD2-def H2-def)
  also from \( \text{assms} \) have \( ... = (((P \parallel\textbf{Q}) :: M) :: J) \)
    by (simp add: seq-assoc)
also from \( \text{assms} \) 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: \( \text{assms} \) \( \text{closure} \) \( \text{SRD-healths} \) )

**Definition** \( nmerge-rd0 \) \( (N_0) \) where
\[ \text{upred-defs}: N_0(M) = (\text{sk} \cdot \text{wait} =_u (\text{sk} \cdot \text{wait} \lor \text{sk} \cdot \text{wait}) \land \text{sk} \cdot \leq_u \text{sk} \cdot \leq_u \text{sk}' \land (\exists \, \text{sk} \cdot \text{ok} \lor \text{sk} \cdot \text{ok} ; \text{sk} \cdot \text{ok} ; \text{sk} \cdot \text{ok} ; \text{sk} \cdot \text{ok} ; \text{sk} \cdot \text{ok} ; \text{sk} \cdot \text{ok} \cdot \text{wait} ; \text{sk} \cdot \text{wait} ; \text{sk} \cdot \text{wait} ; \text{sk} \cdot \text{wait} ; \text{sk} \cdot \text{wait} ; \text{sk} \cdot \text{wait} ; \text{sk} \cdot \text{wait} \cdot M) \) \]

**Definition** \( nmerge-rd1 \) \( (N_1) \) where
\[ \text{upred-defs}: N_1(M) = (\text{sk} \cdot \text{ok} =_u (\text{sk} \cdot \text{ok} \lor \text{sk} \cdot \text{ok}) \land N_0(M) ) \]

65
definition nmerge-rd \((N_R)\) where
\[
\text{[upred-defs]: } N_R(M) = (\exists \$st < \cdot \$v' = u \cdot \$v < \cdot \$tr < \cdot N_1(M)) < \cdot \$ok < \cdot \$tr < \cdot (\$tr < \cdot u \cdot \$tr ')
\]
definition merge-rd1 \((M_1)\) where
\[
\text{[upred-defs]: } M_1(M) = (N_1(M) :: N_R)
\]
definition merge-rd \((M_R)\) where
\[
\text{[upred-defs]: } M_R(M) = N_R(M) :: N_R
\]
abbreviation rdes-par \((- \cdot [85,0,86] \cdot -\) where
\[P \parallel R M Q \equiv P \parallel R R2m(M) Q\]

Healthiness condition for reactive design merge predicates
definition \[\text{[upred-defs]: } RDM(M) = R2m(\exists \$0 - ok; \$0 - ok; \$0 - wait; \$1 \cdot M)\]
lemma nmerge-rd-is-R1m \[\text{[closure]: } N_R(M) \text{ is R1m}\]
\text{by (rel-blast)}
lemma R2m-nmerge-rd \[\text{[closure]: } M \text{ is R2m } \Rightarrow N_R(M) \text{ is R2m}\]
\text{by (metis Healthy-def' R2m-nmerge-rd)}
lemma nmerge-rd-is-R3hm \[\text{[closure]: } N_R(M) \text{ is R3hm}\]
\text{by (rel-blast)}
lemma nmerge-rd-is-RD1m \[\text{[closure]: } N_R(M) \text{ is RD1m}\]
\text{by (rel-blast)}
lemma merge-rd-is-RD3 \[\text{[closure]: } M_R(M) \text{ is RD3}\]
\text{by (metis Healthy-Idempotent RD3-Idempotent RD3-def merge-rd-def)}
lemma merge-rd-is-RD2 \[\text{[closure]: } M_R(M) \text{ is RD2}\]
\text{by (simp add: RD3-implies-RD2 merge-rd-is-RD3)}
lemma par-rdes-NSRD \[\text{[closure]: }\]
\text{assumes } P \text{ is SRD } Q \text{ is SRD } M \text{ is RDM}
\text{shows } P \parallel R M Q \text{ is NSRD}
\text{proof –}
\text{have } (P \parallel R M Q :: H_R) \text{ is NSRD}
\text{by (rule NSRD-intro', simp-all add: SRD-healths closure assms)}
\text{(metis (no-types, lifting) Healthy-def R2-par-by-merge R2-seqr-closure R2m-nmerge-rd RDM-def SRD-healths(2) assms skip-srea-R2 )}
\text{thus } ?\text{thesis}
\text{by (simp thesis: merge-rd-def par-by-merge-def seqr-assoc) qed}

lemma RDM-intro:
\text{assumes } M \text{ is R2m } \$0 - ok ; \$0 - ok ; \$1 - ok ; \$0 - wait \cdot M \text{ is RDM}
\$0 - wait ; \$0 - wait ; \$0 - wait ; \$0 - wait \cdot M \text{ is RDM}
\$0 - wait ; \$0 - wait ; \$0 - wait ; \$0 - wait \cdot M \text{ is RDM}
proof

lemma \textit{RDM-unrests} \{unrest\}:
assumes \( \text{M is RDM} \)
shows \( \exists \, \text{M} \text{ is RDM} \)
by \( (\text{simp add: \text{Healthy-def RDM-def ex-unrest unrest}}) \)

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

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

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

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

lemma \textit{parallel-ok-cases}:
\[ ((P \parallel_s Q) :: M) = \{ \]
\( ((P^{\top} \parallel_s Q^{\top}) :: (M[\text{true,true}/\text{\$0-ok,\$1-ok}]) \lor \]
\( ((P^{\top} \parallel_s Q^{\top}) :: (M[\text{false,true}/\text{\$0-ok,\$1-ok}]) \lor \]
\( ((P^{\top} \parallel_s Q^{\top}) :: (M[\text{true,false}/\text{\$0-ok,\$1-ok}]) \lor \]
\( ((P^{\top} \parallel_s Q^{\top}) :: (M[\text{false,false}/\text{\$0-ok,\$1-ok}]) \}
proof

have \( ((P \parallel_s Q) :: M) = (\exists \, \text{M} \cdot (P \parallel_s Q)[<\text{\$0-ok,\$1-ok}] :: M[<\text{\$0-ok}]) \)
by \( (\text{subt segr-middle[of left-uar ok], simp-all}) \)
also have \( = (\exists \, \text{M} \cdot (P \parallel_s Q)[<\text{\$0-ok,\$1-ok}] :: M[<\text{\$0-ok}] :: M[<\text{\$0-ok}]) \)
by \( (\text{subt segr-middle[of right-uar ok], simp-all}) \)
also have \( = (\exists \, \text{M} \cdot (P[<\text{\$0-ok}] :: Q[<\text{\$0-ok}] :: M[<\text{\$0-ok}]) :: M[<\text{\$0-ok}]) \)
by \( (\text{rel-auto robust}) \)
also have \( = (\}
by \( (\text{simp add: true-alt-def THEN sym} \{false-alt-def THEN sym\} \text{ disj-assoc}) \)
\( \}
by \( (\text{utp-pred-laws.sup-left-commute utp-pred-laws.sup-right-commute usubst}) \)
finally show \( \text{?thesis} \}
qed
lemma skip-srea-ok-f [usubst]:
$$H_f = R1(\neg\text{ok})$$
by (rel-auto)

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

lemma parallel-assm-lemma:
assumes P is RD2
shows \(\text{pre_s \dagger (P \parallel_{M_R(M)} Q)} = ((\text{pre_s \dagger P}) \parallel_{N_0(M)} ;; R1(\text{true}) (\text{cmt_s \dagger Q})) \lor ((\text{cmt_s \dagger P}) \parallel_{N_0(M)} ;; R1(\text{true}) (\text{pre_s \dagger Q}))\)
proof –
have \(\text{pre_s \dagger (P \parallel_{M_R(M)} Q)} = \text{pre_s \dagger ((P \parallel_{s} Q) ;; M_R(M)}\)
by (simp add: par-by-merge-def)
also have \(\ldots = ((P \parallel_{s} Q)\parallel_{\text{true,false}/\text{ok},\text{wait}} ;; N_R \text{ M} ;; R1(\neg\text{ok}))\)
by (simp add: merge-rd-def usubst, rel-auto)
also have \(\ldots = ((P\parallel_{\text{true},\text{false}/\text{ok},\text{wait}}) \parallel_{s} Q[\text{true},\text{false}/\text{ok},\text{wait}] ;; N_1(M) ;; R1(\neg\text{ok}))\)
by (rel-auto robust, (metis)+)
also have \(\ldots = ((P\parallel_{\text{true},\text{false}/\text{ok},\text{wait}}) \parallel_{s} Q[\text{true},\text{false}/\text{ok},\text{wait}]') ;; (N_1 M)[\text{true},\text{true}/\text{\$0-ok},\text{\$1-ok}]\)
by (simp add: par-by-merge-def)
also have \(\ldots = ((P\parallel_{\text{true},\text{false}/\text{ok},\text{wait}}) \parallel_{s} Q[\text{true},\text{false}/\text{ok},\text{wait}]') ;; (N_1 M)[\text{false},\text{true}/\text{\$0-ok},\text{\$1-ok}]\)
by (simp add: par-by-merge-def)
also have \(\ldots = ((P\parallel_{\text{true},\text{false}/\text{ok},\text{wait}}) \parallel_{s} Q[\text{true},\text{false}/\text{ok},\text{wait}]') ;; (N_1 M)[\text{false},\text{false}/\text{\$0-ok},\text{\$1-ok}]\)
by (simp add: par-by-merge-def)
also have \(\ldots = ((P\parallel_{\text{true},\text{false}/\text{ok},\text{wait}}) \parallel_{s} Q[\text{true},\text{false}/\text{ok},\text{wait}]') ;; (N_1 M)[\text{false},\text{false}/\text{\$0-ok},\text{\$1-ok}]\)
by (simp add: par-by-merge-def)
also have \(\ldots = (\text{\$C1} \lor p \text{\$C2} \lor p \text{\$C3} \lor p \text{\$C4})\)
by (subt parallel-ok-cases, subst-tac)
also have \(\ldots = (\text{\$C2} \lor \text{\$C3})\)
proof –
have \(\text{\$C1 = false}\)
by (rel-auto)
moreover have \("?C4 \Rightarrow \?C3" (is \("?A \Rightarrow \?B \Rightarrow (?C \Rightarrow (?D)\)\))
proof –
from assms have \("?P_f \Rightarrow P^{f}\)\)
by (metis RD2-def H2-equivalence Healthy-def)
hence \(P: \text{"?P^{f} \Rightarrow P^{f}\}\)\)
by (rel-auto)
have \(\?A \Rightarrow \?C\)
using \(P\) by (rel-auto)
moreover have \(\?B \Rightarrow \?D\)\)
by (rel-auto)
ultimately show \(\text{\$thesis}\)
by (simp add: impl-seq-mono)
qed
ultimately show \(\text{\$thesis}\)
by (simp add: subumption2)
qed
also have \(\ldots = (\text{\$C2} \lor (\text{\$C4 \Rightarrow \?C3}) \lor (\text{\$C4 \Rightarrow \?C3})\)
by (simp add: par-by-merge-def)
also have \(\ldots = (\text{\$C4} \Rightarrow \text{\$C3})\)
by (rel-auto, metis-sequ-mono)
qed
lemma $\text{pre}_s$-SRD:
  assumes $P$ is SRD
  shows $\text{pre}_s \vdash P = (\neg_R \text{pre}_R(P))$
proof –
  have $\text{pre}_s \vdash P = \text{pre}_s \vdash R_C(\text{pre}_R P \vdash \text{peri}_R P \circ \text{post}_R P)$
  by (simp add: SRD-reactive-tri-design assms)
also have ... = $R_1(\neg \text{pre}_s \vdash \text{pre}_R(P))$
  by (simp add: RHS-def asubst R3h-def prets-design)
also have ... = $R_1(\neg \text{pre}_R(P))$
  by (rel-auto)
also have ... = $(\neg_R \text{pre}_R(P))$
  by (simp add: $\text{pre}_s$-SRD assms rea-not-def)
finally show $?\text{thesis}$.
qed

lemma parallel-assm:
  assumes $P$ is SRD $Q$ is SRD
  shows $\text{pre}_R(P \parallel M_R(M)) = (\neg_R ((\neg_R \text{pre}_R(P)) \parallel N_0(M)) \circ R_1(\text{true} \circ \text{cmt}_R(Q)) \wedge$
  $\neg_R (\text{cmt}_R(P) \parallel N_0(M) \circ R_1(\text{true} \circ (\neg_R \text{pre}_R(Q))))$
(is $?\text{lhs} = $?\text{rhs})
proof –
  have $\text{pre}_R(P \parallel M_R(M)) = (\neg_R (\text{pre}_s \vdash P) \parallel N_0(M) \circ R_1(\text{true} \circ \text{cmt}_R(Q)) \wedge$
  $\neg_R (\text{cmt}_R(P) \parallel N_0(M) \circ R_1(\text{true} \circ (\text{pre}_s \vdash Q))$
  by (simp add: $\text{pre}_R$-def parallel-assm-lemma assms SRD-healths R1-conj rea-not-def[THEN sym])
also have ... = $?\text{rhs}$
  by (simp add: $\text{pre}_s$-SRD assms cmt$_R$-def Healthy-if closure unrest)
finally show $?\text{thesis}$.
qed

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

lemma JL1: $M \parallel [\text{false}, \text{true}/\$0-\text{ok}, \$1-\text{ok}] = N_0(M) \circ R_1(\text{true})$
by (rel-lust)

lemma JL2: $M \parallel [\text{true}, \text{false}/\$0-\text{ok}, \$1-\text{ok}] = N_0(M) \circ R_1(\text{true})$
by (rel-lust)

lemma JL3: $M \parallel [\text{false}, \text{false}/\$0-\text{ok}, \$1-\text{ok}] = N_0(M) \circ R_1(\text{true})$
by (rel-lust)

lemma JL4: $M \parallel [\text{true}, \text{true}/\$0-\text{ok}, \$1-\text{ok}] = (\$\text{ok} \wedge N_0 M) \circ R_1(\text{true})$
by (simp add: merge-rd1-def asubst nmerge-rd1-def unrest)

lemma parallel-commitment-lemma-1:
assumes $P$ is RD2
shows $cnt_s \vdash (P \parallel_{M_R(M)} Q) = \vDash\vDash$
\begin{align*}
&((cnt_s \vdash P) \parallel (ok' \land N_0 \land M)) \parallel (cnt_s \vdash Q)) \lor \\
&((pre_s \vdash P) \parallel (N_0 \land M) \parallel (R1 \true) \vdash (cnt_s \vdash Q)) \lor \\
&((cnt_s \vdash P) \parallel (N_0 \land M) \parallel (R1 \true) \vdash (pre_s \vdash Q))
\end{align*}
\begin{proof}
\begin{itemize}
\item have $cnt_s \vdash (P \parallel_{M_R(M)} Q) = (P[true, false/ok, wait] \parallel (M_1(M)) \vdash Q[true, false/ok, wait])$
\begin{itemize}
\item by (simp add: usubst, rel-auto)
\end{itemize}
\item also have $\ldots = ((P[true, false/ok, wait] \parallel, Q[true, false/ok, wait]) \parallel (M_1(M))')$
\begin{itemize}
\item by (simp add: par-by-merge-def)
\end{itemize}
\item also have $\ldots = (\ldots )$
\begin{itemize}
\item by (simp add: subst parallel-ok-cases, subst-tac)
\end{itemize}
\item also have $\ldots = (\ldots )$
\begin{itemize}
\item by (simp add: J1L1 J1L2 J2L3)
\end{itemize}
\item proof --
\item from assms have `$P'$ $\Rightarrow$ $P'$
\begin{itemize}
\item by (metis RD2-def H2-equivalence Healthy-def)
\end{itemize}
\item hence $P'$ $\Rightarrow$ $P'$
\begin{itemize}
\item by (rel-auto)
\end{itemize}
\item have `$A' $\Rightarrow$ $?C'$ (is `$A' $\Rightarrow$ $?B' $\Rightarrow$ $?C'$)
\begin{itemize}
\item by (simp add: ?thesis)
\item by (simp add: ?thesis)
\end{itemize}
\item qed
\begin{itemize}
\item by (simp add: subsumption2)
\end{itemize}
\item qed
\begin{itemize}
\item finally show $?thesis$
\begin{itemize}
\item by (simp add: par-by-merge-def J4)
\end{itemize}
\end{itemize}
\end{proof}
\end{itemize}
\begin{proof}
\begin{itemize}
\item lemma parallel-commitment-lemma-2:
\begin{itemize}
\item assumes $P$ is RD2
\item shows $cnt_s \vdash (P \parallel_{M_R(M)} Q) = \vDash\vDash$
\begin{align*}
&(((cnt_s \vdash P) \parallel (ok' \land N_0 \land M)) \parallel (cnt_s \vdash Q)) \lor \\
&((cnt_s \vdash P) \parallel (N_0 \land M) \parallel (R1 \true) \vdash (cnt_s \vdash Q)) \lor \\
&((cnt_s \vdash P) \parallel (N_0 \land M) \parallel (R1 \true) \vdash (pre_s \vdash Q))
\end{align*}
\begin{itemize}
\item by (simp add: parallel-commitment-lemma-1 assms parallel-assm-lemma)
\end{itemize}
\end{itemize}
\end{itemize}
\begin{itemize}
\item lemma parallel-commitment-lemma-3:
\begin{itemize}
\item $M$ is $R1m$ $\Rightarrow$ $(ok' \land N_0 \land M)$ $\vdash R1m$
\end{itemize}
\end{itemize}
\end{proof}
proof

lemma parallel-commitment:
assumes $P$ is SRD $Q$ is SRD $M$ is RDM
shows $\mathit{cmt}_R(P \parallel M_{\mathcal{R}}(M) \parallel Q) \Rightarrow_r \mathit{cmt}_R(P \parallel (\mathsf{ok}^* \land N_0 \parallel M) \parallel \mathcal{H}_R \parallel \mathit{cmt}_R(Q))$
by (simp add: parallel-commitment-lemma-2 parallel-commitment-lemma-3 Healthy-if assms cmtR-def pre-r-SRD closure rea-impl-def disj-comm unrest)

theorem parallel-reactive-design:
assumes $P$ is SRD $Q$ is SRD $M$ is RDM
shows $(P \parallel M_{\mathcal{R}}(M) \parallel Q) = \mathcal{R}_s(\neg_r (\neg_r \mathbf{pre}_R(P)) \parallel N_0(M) \parallel R_1(\text{true}) \mathit{cmt}_R(Q)) \land 
\neg_r (\mathit{cmt}_R(P) \parallel N_0(M) \parallel R_1(\text{true}) (\neg_r \mathbf{pre}_R(Q))) 
\vdash (\mathit{cmt}_R(P) \parallel (\mathsf{ok}^* \land N_0 \parallel M) \parallel \mathcal{H}_R \parallel \mathit{cmt}_R(Q))$ (is $?\text{lhs} = ?\text{rhs}$)
proof –
have $(P \parallel M_{\mathcal{R}}(M) \parallel Q) = \mathcal{R}_s(\mathbf{pre}_R(P) \parallel M_{\mathcal{R}}(M) \parallel \mathit{cmt}_R(P) \parallel M_{\mathcal{R}}(M) \parallel Q)$
by (metis Healthy-def NSRD-is-SRD SRD-as-reactive-design assms(1) assms(2) assms(3) par-rdes-NSRD)
also have $\ldots = ?\text{rhs}$
by (simp add: NSRD-is-SRD SRD-as-reactive-design assms par-rdes-NSRD assms(1) assms(2) assms(3) par-rdes-NSRD)
finally show $?\text{thesis}$.
qed

lemma parallel-pericondition-lemma1:
$(\mathsf{ok}^* \land P) \parallel \mathcal{H}_R(\text{true}, true$/$\mathsf{ok}^*$, $\mathsf{wait}$ $) = (\exists \; \mathsf{st}^* \cdot P)[\text{true}, true / \mathsf{ok}^*$/$\mathsf{wait}$]
(is $?\text{lhs} = ?\text{rhs}$)
proof –
have $?\text{lhs} = (\mathsf{ok}^* \land P) \parallel (\exists \; \mathsf{st} \cdot \mathcal{H})[\text{true}, true / \mathsf{ok}^*$/$\mathsf{wait}$]
by (rel-simp)
also have $\ldots = ?\text{rhs}$
by (rel-auto)
finally show $?\text{thesis}$.
qed

lemma parallel-pericondition-lemma2:
assumes $M$ is RDM
shows $(\exists \; \mathsf{st}^* \cdot N_0(M))[\text{true}, true / \mathsf{ok}^*$/$\mathsf{wait}$] = $((\mathsf{tr} \cdot \text{true} \lor \$1 \cdot \text{wait}) \land (\exists \; \mathsf{st}^* \cdot M))$
proof –
have $(\exists \; \mathsf{st}^* \cdot N_0(M))[\text{true}, true / \mathsf{ok}^*$/$\mathsf{wait}$] = $(\exists \; \mathsf{st}^* \cdot (\mathsf{tr} \cdot \text{true} \lor \$1 \cdot \text{wait}) \land \mathsf{tr}^* \geq_u \mathsf{tr} < \land M)$
by (simp add: usubst unrest nmerge-r0-def ex-unrest Healthy-if R1m-def assms)
also have $\ldots = (\exists \; \mathsf{st}^* \cdot (\mathsf{tr} \cdot \text{true} \lor \$1 \cdot \text{wait}) \land M)$
by (metis (no_types, hide_lams) Healthy-if R1m-def R1m-idem R2m-def RDM-def assms utp-pred-laws.inf-commute)
also have $\ldots = ((\$0 \cdot \text{true} \lor \$1 \cdot \text{wait}) \land (\exists \; \mathsf{st}^* \cdot M))$
by (rel-auto)
finally show $?\text{thesis}$.
qed

lemma parallel-pericondition-lemma3:
$((\mathsf{tr} \cdot \text{true} \lor \$1 \cdot \text{wait}) \land (\exists \; \mathsf{st}^* \cdot M)) = ((\mathsf{tr} \cdot \text{true} \lor \$1 \cdot \text{wait} \land (\exists \; \mathsf{st}^* \cdot M)) \lor (\neg \mathsf{tr} \cdot \text{true} \lor \$1 \cdot \text{wait} \land (\exists \; \mathsf{st}^* \cdot M)) \lor (\mathsf{tr} \cdot \text{true} \lor \$1 \cdot \text{wait} \land (\exists \; \mathsf{st}^* \cdot M))$
by (rel-auto)

lemma parallel-pericondition [rdes]:
fixes $M :: (\mathsf{s}, \mathsf{t}, \mathsf{trace}, \alpha)$ rsp merge
assumes $P$ is SRD $Q$ is SRD $M$ is RDM

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

proof

have $\text{peri}_R(P \parallel M_R(M) Q) = \\
(\text{peri}_R(P \parallel M_R M Q) \Rightarrow \text{cmt}_R P \parallel (\exists \text{st}' \cdot N_0 M)[\text{true,true/sok', \text{wait}'}] \text{cmt}_R Q)$
by (simp add: peri-cmt-def parallel-commitment SRD-healths assms usubst unrest assms)
also have $\ldots = (\text{peri}_R(P \parallel M_R M Q) \Rightarrow \text{cmt}_R P \parallel (\exists \text{st}' \cdot N_0 M)[\text{true,true/sok', \text{wait}'}] \text{cmt}_R Q)$
by (simp add: parallel-pericondition-lemma1)
also have $\ldots = (\text{peri}_R(P \parallel M_R M Q) \Rightarrow \text{cmt}_R P \parallel (\exists \text{st}' \cdot N_0 M)[\text{true,true/sok', \text{wait}'}] \text{cmt}_R Q)$
by (simp add: parallel-pericondition-lemma2 assms)
also have $\ldots = (\text{peri}_R(P \parallel M_R M Q) \Rightarrow \text{cmt}_R P \parallel (\exists \text{st}' \cdot N_0 M)[\text{true,true/sok', \text{wait}'}] \text{cmt}_R Q)$
by (simp add: parallel-pericondition-lemma3 assms)
also have $\ldots = (\text{peri}_R(P \parallel M_R M Q) \Rightarrow \text{cmt}_R P \parallel (\exists \text{st}' \cdot N_0 M)[\text{true,true/sok', \text{wait}'}] \text{cmt}_R Q)$
by (simp add: parallel-pericondition-lemma4 assms)
also have $\ldots = (\text{peri}_R(P \parallel M_R M Q) \Rightarrow \text{cmt}_R P \parallel (\exists \text{st}' \cdot N_0 M)[\text{true,true/sok', \text{wait}'}] \text{cmt}_R Q)$
by (simp add: par-by-merge-alt-def parallel-pericondition-lemma3 seqr-or-distr)
also have $\ldots = (\text{peri}_R(P \parallel M_R M Q) \Rightarrow \text{cmt}_R P \parallel (\exists \text{st}' \cdot N_0 M)[\text{true,true/sok', \text{wait}'}] \text{cmt}_R Q)$
by (simp add: seqr-right-one-point-true seqr-right-one-point-false cmt_R-def post_R-def peri_R-def usubst unrest assms)
also have $\ldots = (\text{peri}_R(P \parallel M_R M Q) \Rightarrow \text{cmt}_R P \parallel (\exists \text{st}' \cdot N_0 M)[\text{true,true/sok', \text{wait}'}] \text{cmt}_R Q)$
by (simp add: par-by-merge-alt-def)

finally show $\exists \text{thesis}$.

qed

lemma parallel-postcondition-lemma1:
$(\exists \text{ok'} \wedge P) :: H_R[\text{true,false}/\text{ok'}, \text{wait}'] = P[\text{true,false}/\text{ok'}, \text{wait}']$
(is $\exists \text{lhs} = \exists \text{rhs}$)

proof

have $\exists \text{lhs} = (\exists \text{ok'} \wedge P) ;; H[\text{true,false}/\text{ok'}, \text{wait}']$
by (rel-blast)
also have $\ldots = \exists \text{rhs}$
by (rel-auto)

finally show $\exists \text{thesis}$.

qed

lemma parallel-postcondition-lemma2:
assumes $M$ is RDM
shows $(\exists \text{ok}\cdot N_0(M))[\text{true,false}/\text{ok'}, \text{wait}'] = ((\exists \text{st} - \text{wait} \wedge \exists \text{st} - \text{wait})) \wedge M$

proof

have $(\exists \text{ok}\cdot N_0(M))[\text{true,false}/\text{ok'}, \text{wait}'] = ((\exists \text{st} - \text{wait} \wedge \exists \text{st} - \text{wait})) \wedge \text{tr} - \text{tr} \geq u [\text{tr} < \text{tr} \wedge M]$
by (simp add: usubst unrest merge-rd0-def ex-unrest Healthy-if R1m-def assms)
also have $\ldots = ((\exists \text{st} - \text{wait} \wedge \exists \text{st} - \text{wait})) \wedge M$
by (metis Healthy-if R1m-def RDM-R1m assms utp-pred-laws.inf-commute)
finally show $\exists \text{thesis}$.

qed

lemma parallel-postcondition $\text{rdes}$:
fixes $M :: (s,t::trace,a) \text{ rsp merge}$
assumes $P$ is SRD $Q$ is SRD $M$ is RDM
shows $\text{post}_R(P \parallel M_R(M) Q) = (\text{pre}_R(P \parallel M_R M) Q) \Rightarrow \text{post}_R(P) \parallel M \text{post}_R(Q)$

proof –

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

$(\text{pre}_R(P \parallel M_R M) Q) \Rightarrow \text{cnt}_R P \parallel \langle \text{sk}_\omega \land \text{N}_0 M \rangle :: \text{IF}_R[\text{true, false, sk}_\omega, \text{wait}]$ $\text{cnt}_R Q$

by (simp add: post-cmt-def parallel-commitment assms usubst unrest SRD-healths)
also have $\ldots = (\text{pre}_R(P \parallel M_R M) Q) \Rightarrow \text{cnt}_R P \parallel \langle \text{not} \text{-} \text{wait} \land \neg \text{sk}_1 \text{-} \text{wait} \land \text{M} \rangle$ $\text{cnt}_R Q$

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

by (simp add: par-by-merge-alt-def seq-right-one-point-false usubst unrest cntR-def postR-def assms)
finally show $?thesis$.

qed

lemma parallel-precondition-lemma:

fixes $M :: (\ell :: 's, t :: 'r, a :: 'a) \text{ rsp merge}$
assumes $P$ is NSRD $Q$ is NSRD $M$ is RDM
shows $(\neg \text{pre}_R(P)) \parallel \text{N}_0(M) :: \text{R}_1(\text{true})$ $\text{cntR}(Q) =$

$(\neg \text{pre}_R(P)) \parallel \text{M} :: \text{R}_1(\text{true})(\text{peri}_R(Q) \land \text{post}_R(Q))$

proof –

have $(\neg \text{pre}_R(P)) \parallel \text{N}_0(M) :: \text{R}_1(\text{true})$ $\text{cntR}(Q) =$

$(\neg \text{pre}_R(P)) \parallel \text{N}_0(M) :: \text{R}_1(\text{true})(\text{peri}_R(Q) \land \text{post}_R(Q))$

by (simp add: wait'-cond-post-cnt)
also have $\ldots = (((\neg \text{pre}_R(P))_0 \land [\text{peri}_R(Q)]_1 \land [\text{peri}_R(Q)]_1 \land \text{true}) \equiv \text{R}_1(\text{true}))$

by (simp add: par-by-merge-alt-def)
also have $\ldots = (((\neg \text{pre}_R(P))_0 \land [\text{peri}_R(Q)]_1 \land \text{true}) \equiv \text{R}_1(\text{true}))$

(is $?P \equiv ?Q$)

proof –

have $?P = ?Q$

by (rel-auto)

thus $?thesis$ by simp

qed

also have $\ldots = (((\neg \text{pre}_R(P))_0 \land [\text{peri}_R(Q)]_1 \land \text{true}) \equiv \text{R}_1(\text{true}))$

by (simp add: cond-inter-var-split)
also have $\ldots = (((\neg \text{pre}_R(P))_0 \land [\text{peri}_R(Q)]_1 \land \text{true}) \equiv \text{R}_1(\text{true})$

by (simp add: usubst unrest)
also have $\ldots = (((\neg \text{pre}_R(P))_0 \land [\text{peri}_R(Q)]_1 \land \text{true}) \equiv \text{R}_1(\text{true})$

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

thus $?thesis$

by (simp add: nmerge-rd0-def unrest assms closure ex-unrest usubst)

qed

also have $\ldots = (((\neg \text{pre}_R(P))_0 \land [\text{peri}_R(Q)]_1 \land \text{true}) \equiv \text{R}_1(\text{true})$

by (RDM-R1m[OF assms(3)])

using RDM-R1m[OF assms(3)]

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

thus $?thesis$

by (simp add: nmerge-rd0-def unrest assms closure ex-unrest usubst)

qed
\[(\lnot \pre_R P)_0 \land [\post_R Q]_1 \land \$v <' = u \$v) \;; \ M \;; R1 \text{ true}\]

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

proof

have \( P_1 = ((\lnot \pre_R P)_0 \land [\post_R Q]_1 \land \$v <' = u \$v) \;; (M \land \$wait') \;; R1 \text{ true}\)

by (simp add: conj-comm)

hence 1: \( P_1 = Q_1\)

by (simp add: segr-left-one-point-true segr-left-one-point-false add: unrest unsubst closure assms)

have \( P_2 = ((\lnot \pre_R P)_0 \land [\post_R Q]_1 \land \$v <' = u \$v) \;; (M \land \$wait') \;; R1 \text{ true} \lor

((\lnot \pre_R P)_0 \land [\post_R Q]_1 \land \$v <' = u \$v) \;; (M \land \sim \$wait') \;; R1 \text{ true}\)

by (subst segr-bool-split[af left-uar wait], simp-all add: unsubst unrest assms closure conj-comm)

hence 2: \( P_2 = Q_2\)

by (simp add: segr-left-one-point-true segr-left-one-point-false unrest unsubst closure assms)

from 1 2 show \( \text{thesis} \).

qed

lemma swap-nmerge-rd0:

\(\text{swap}_m \;; N_0(M) = N_0(\text{swap}_m \;; M)\)

by (rel-auto, meson+)

lemma SymMerge-nmerge-rd0 [closure]:

\(M\) is SymMerge \(\Longrightarrow\) \(N_0(M)\) is SymMerge

by (rel-auto, meson+)

lemma swap-nmerge-rd':

\(\text{swap}_m \;; N_R(M) = N_R(\text{swap}_m \;; M)\)

by (rel-blast)

lemma swap-nmerge-rd:

\(\text{swap}_m \;; M_R(M) = M_R(\text{swap}_m \;; M)\)

by (simp add: merge-rd-def segr-assoc[THEN sym] swap-nmerge-rd')

lemma SymMerge-nmerge-rd [closure]:

\(M\) is SymMerge \(\Longrightarrow\) \(M_R(M)\) is SymMerge

by (simp add: Healthy-def swap-nmerge-rd)

lemma nmerge-rd1-merge3:

assumes \(M\) is RDM

shows \(M3(N_1(M)) = (\$ok' = u \ (\$0-ok \land \$1-0-ok \land \$1-1-ok) \land

\$wait' = u \ (\$0-wait \lor \$1-0-wait \lor \$1-1-wait) \land

M3(M))\)

proof

have \(M3(N_1(M)) = M3(\$ok' = u \ (\$0-ok \land \$1-ok) \land

\$wait' = u \ (\$0-wait \lor \$1-wait) \land

\$tr' \leq u \$tr' \land

(3 \ \{\$0-ok, \$1-ok, \$ok', \$0-wait, \$1-wait, \$wait', \$wait'\} \cdot \text{RDM}(M))\)

by (simp add: nmerge-rd1-def nmerge-rd0-def assms Healthy-if)

also have \( M3(\$ok' = u \ (\$0-ok \land \$1-ok) \land \$wait' = u \ (\$0-wait \lor \$1-wait) \land \text{RDM}(M))\)

by (rel-blast)

also have \( M3(\$ok' = u \ (\$0-ok \land \$1-0-ok \land \$1-1-ok) \land \$wait' = u \ (\$0-wait \lor \$1-0-wait \lor \$1-1-wait) \land M3(\text{RDM}(M))\)

by (rel-blast)
also have \( \cdots = (\$ok' = _u (\$0 -ok \land \$1 -0-ok \land \$1 -1-ok) \land \$wait' = _u (\$0 -wait \lor \$1 -0-wait \lor \$1 -1-wait) \land M3(M)) \)
  by (simp add: assms Healthy-if)
finally show \(?thesis\).
qed

lemma nmerge-rl-merge3:
  \(M3(N_R(M)) = (\exists \$st_\prec \cdot \$v' = _u \$v_\prec) \land \$wait_\prec \land M3(N_1 M) \land \$ok_\prec \land (\$tr_\prec \leq _u \$tr')\)
  by (rel-blast)

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

lemma parallel-precondition:
  fixes \(M\) :: \(\langle s, t :: trace, \alpha \rangle\) resp merge
  assumes \(P\) is \(NSRD\) \(Q\) is \(NSRD\) \(M\) is \(RDM\)
  shows \(\text{pre}_R(P) \parallel M \parallel R1(M) \parallel Q = \text{pre}_R(P) \parallel M \parallel R1(M) \parallel Q\)
proof –
  have \(a: \text{pre}_R(P) \parallel N_0(M) \parallel R1(true) \parallel \text{cmtr}(Q) = \text{pre}_R(P) \parallel M \parallel R1(true) \parallel \text{post}_R Q\)
    by (simp add: parallel-precondition-lemma assms)
  have \(b: \text{cmtr}_R(P) \parallel N_0(M) \parallel R1(true) \text{cmtr}_R(P) = \text{cmtr}_R(P) \parallel M \parallel R1(true) \parallel \text{post}_R Q\)
    by (simp add: swap-nmerge-rl0[THEN sym]\ seqr-assoc[THEN sym] par-by-merge-def par-sep-merge)
  have \(c: \text{cmtr}_R(P) \parallel N_0(swap_m ;; M) \parallel R1(true) \parallel \text{cmtr}_R(P) = \text{cmtr}_R(P) \parallel (swap_m ;; M) \parallel R1(true) \parallel \text{post}_R Q\)
    by (simp add: parallel-precondition-lemma closure assms)
show \(?thesis\)
  by (simp add: parallel-assm closure assms a b c, rel-auto)
qed

Weakest Parallel Precondition

definition \(wrR\) ::
  \(\langle t :: trace, \alpha \rangle\) hrel-rp \Rightarrow
  \(\langle t, \alpha \rangle\) rp merge \Rightarrow
  \(\langle t', \alpha \rangle\) hrel-rp \Rightarrow
  \(\langle t', \alpha \rangle\) hrel-rp \(\langle \text{wr}_R(\cdot) \cdot \rangle - [60,0.61] \parallel 61\)
where \[upred-defs\]: \(Q \ \text{wr}_R(M) \parallel P = \text{pre}_R(P) \parallel M \parallel R1(true) Q\)

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

by (simp add: wrR-def closure)

**Lemma R2-rea-not:** \( R2(\neg_r \; P) = (\neg_r \; R2(P)) \)

by (rel-auto)

**Lemma wrR-R2-lemma:**
- Assumes \( P \text{ is } R2 \) \( Q \text{ is } R2 \) \( M \text{ is } R2m \)
- Shows \((\neg_r \; P) \abbr{M} Q) ; R1(true_h) \text{ is } R2 \)

**Proof**
- Have \((\neg_r \; P) \abbr{M} Q) \text{ is } R2
  - By (simp add: closure assms)
- Thus \(?thesis
  - By (simp add: closure)

qed

**Lemma wrR-R2 [closure]:**
- Assumes \( P \text{ is } R2 \) \( Q \text{ is } R2 \) \( M \text{ is } R2m \)
- Shows \( Q \text{ wr } R(M) \text{ is } R2 \)

**Proof**
- Have \((\neg_r \; P) \abbr{M} Q) ; R1(true_h) \text{ is } R2
  - By (simp add: wrR-R2-lemma assms)
- Thus \(?thesis
  - By (simp add: wrR-def wrR-R2-lemma par-by-merge-seq-add closure)

qed

**Lemma wrR-RR [closure]:**
- Assumes \( P \text{ is } RR \) \( Q \text{ is } RR \) \( M \text{ is } RDM \)
- Shows \( Q \text{ wr } R(M) \text{ is } RR \)

**Proof**
- Apply (rule RR-intro)
  - Apply (simp all add: unrest assms closure wrR-def rpred)
  - Apply (metis (no-types, lifting) Healthy-def R1-R2c-commute R1-R2c-is-R2 R1-rea-not RDM-R2m RR-implies-R2 assms(1) assms(2) assms(3) par-by-merge-seq-add rea-not-R2-closed wrR-R2-lemma)

done

**Lemma wrR-RC [closure]:**
- Assumes \( P \text{ is } RR \) \( Q \text{ is } RR \) \( M \text{ is } RDM \)
- Shows \((Q \text{ wr } R(M) \text{ is } RC \)

**Proof**
- Apply (rule RC-intro)
  - Apply (simp add: closure assms)
  - Apply (simp add: wrR-def rpred closure assms)
  - Apply (simp add: par-by-merge-def seqr-assoc)

done

**Lemma uppR-choice [wp]:** \( (P \lor Q) \text{ wr } R(M) \text{ is } R = (P \text{ wr } R(M) \text{ is } R \land Q \text{ wr } R(M) \text{ is } R) \)

**Proof**
- Have \((P \lor Q) \text{ wr } R(M) \text{ is } R = \)
  \((\neg_r ((\neg_r \; R) ; U0 \land (P ; U1 \lor Q ; U1) \land \$v_\neg_r = u) \text{ wr } (\neg_r \; R) ; U0 \land Q ; U1 \land \$v_\neg_r = u) \text{ wr } \)
  \((\neg_r \; (R) ; U0 \land P ; U1 \land \$v_\neg_r = u) \text{ wr } (\neg_r \; R) ; U0 \land Q ; U1 \land \$v_\neg_r = u)

\)
  \text{ wr } M \land true_r)

  - By (simp add: conj-disj-distr utp-pred-laws inf-sup-distrib2)

also have \( ... = \neg_r (((\neg_r \; R) ; U0 \land P ; U1 \land \$v_\neg_r = u) \text{ wr } U0 \land Q ; U1 \land \$v_\neg_r = u) \text{ wr } \)

\text{ wr } M \land true_r)

- By (simp add: conj-disj-distr utp-pred-laws inf-sup-distrib2)

also have \( ... = \neg_r (((\neg_r \; R) ; U0 \land P ; U1 \land \$v_\neg_r = u) \text{ wr } M \land true_r \lor \)

\text{ wr } M \land true_r))

76
proof
by (simp add: seq-r-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 \( \wr_R(M) \ P = \text{true}_R \)
by (simp add: wrR-def)

lemma uppR-true [wp]: \( \wr_R(M) \text{ true}_R = \text{true}_R \)
by (simp add: wrR-def)

lemma parallel-precondition-ur [rdes];
assumes \( P \) is SRD \( M \) is RDM
shows \( \text{pre}_R(P \parallel_M R(M) \ Q) = (\text{peri}_R(Q) \wr_R(M) \text{pre}_R(P) \land \text{post}_R(Q) \wr_R(M) \text{pre}_R(P) \land \text{peri}_R(P) \wr_R(\text{swap}_m :: M) \text{pre}_R(Q) \land \text{post}_R(P) \wr_R(\text{swap}_m :: M) \text{pre}_R(Q)) \)
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
\( M \) is RDM
shows \( R_1(P_1 \parallel P_2 \parallel P_3) \parallel_M R_2(Q_1 \parallel Q_2 \parallel Q_3) = R_2(\text{peri}_R(Q_1 \wr_R(M) P_1 \land (Q_1 \Rightarrow_R Q_3) \wr_R(M) P_1 \land (P_1 \Rightarrow_R P_2) \wr_R(\text{swap}_m :: M) Q_1 \land (P_1 \Rightarrow_R P_3) \wr_R(\text{swap}_m :: M) Q_1) \)
 proof -
have \( ?lhs = R_2(\text{peri}_R ?lhs \parallel \text{peri}_R ?lhs) \)
by (simp add: SRD-reactive-tri-design assms closure)
also have \( \ldots = ?rhs \)
by (simp add: rdes closure unrest assms, rel-auto)
finally show ?thesis.
qed

lemma Miracle-parallel-left-zero;
assumes \( P \) is SRD \( M \) is RDM
shows \( \text{Miracle} \parallel_M P = \text{Miracle} \)
 proof -
have \( \text{pre}_R(\text{Miracle} \parallel_M P) = \text{true}_R \)
by (simp add: parallel-assm wait'-cond-idem rdes closure assms)
moreover hence \( \text{cmt}_R(\text{Miracle} \parallel_M P) = \text{false} \)
by (simp add: rdes closure wait'-cond-idem SRD-healths assms)
ultimately have \( \text{Miracle} \parallel_M P = R_1(\text{true}_R \parallel \text{false}) \)
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

lemma Miracle-parallel-right-zero;
assumes \( P \) is SRD \( M \) is RDM
shows \( P \parallel_M \text{Miracle} = \text{Miracle} \)
 proof -
have \( \text{pre}_R(P \parallel_M \text{Miracle}) = \text{true}_R \)

8.1 Example basic merge

**Definition** *BasicMerge* :: ("s", "t::trace, unit") *rsp* merge \((N_B)\) where

```
[upred-defs]: BasicMerge = (\(\text{st}_{\leq} \equiv \text{st}′ - \text{st}_{\leq} = u \text{ $0 - tr - \text{st}_{\leq} \land \text{st}′ - \text{st}_{\leq} = u \text{ $1 - tr}\)
```

**Abbreviation** *rbasic-par* (- \(\parallel\) - [85,86] 85) where

\(P \parallel_B Q \equiv P \parallel_{M_R(N_B)} Q\)

**Lemma** *BasicMerge-RDM* [closure]: \(N_B\) is RDM

```
by (rule RDM-intro, (rel-auto)\+)
```

**Lemma** *BasicMerge-SymMerge* [closure]:

\(N_B\) is SymMerge

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

**Lemma** *BasicMerge\'·calc:*

```
assumes \(\exists \text{ok}\; \parallel\; P\; \exists \text{wait}\; \parallel\; Q\; \parallel\; wait\) \(P\) is \(R2\) \(Q\) is \(R2\)

shows \(P \parallel_{N_B} Q = (\exists \text{st}′ \cdot P) \land (\exists \text{st}′ \cdot Q) \land \text{st}′ = u \text{ st}\)
```

**Proof**

- have \(P; (\exists \{\text{ok}, \text{wait}\}) \cdot R2(P)) = P\) (is \?P\' = -)
  
  by (simp add: ex-unrest ex-plus Healthy-if assms)

- have \(Q; (\exists \{\text{ok}, \text{wait}\}) \cdot R2(Q)) = Q\) (is \?Q\' = -)
  
  by (simp add: ex-unrest ex-plus Healthy-if assms)

- have \(?P\; \parallel_{N_B} ?Q\; = (\exists \{\text{st}′ \cdot ?P\}) \land (\exists \{\text{st}′ \cdot ?Q\}) \land \text{st}′ = u \text{ st}\)
  
  by (simp add: par-by-merge-alt-def, rel-auto, blast\+)

thus \?thesis

by (simp add: P Q)
```

**QED**

8.2 Simple parallel composition

**Definition** *rea-design-par* ::

(\(s\), \(t::trace, \alpha\) hrel-rsp \(\Rightarrow\) (\(s\), \(t\), \(\alpha\)) hrel-rsp \(\Rightarrow\) (\(s\), \(t\), \(\alpha\)) hrel-rsp (infxr \(\parallel\) 85)

where [upred-defs]: \(P \parallel_R Q = R_\alpha((\text{pre}_R(P) \land \text{pre}_R(Q)) \land (\text{cnt}_R(P) \land \text{cnt}_R(Q)))\))

**Lemma** *RHS-design-par:*

```
assumes \(\exists \text{ok}\; \parallel\; P_1 \; \exists \text{ok}\; \parallel\; P_2\)

shows \(R_\alpha(P_1 \parallel Q_1) \parallel_R R_\alpha(P_2 \parallel Q_2) = R_\alpha((P_1 \land P_2) \parallel (Q_1 \land Q_2))\)
```

**Proof**

- have \(R_\alpha(P_1 \parallel Q_1) \parallel_R R_\alpha(P_2 \parallel Q_2) = \)
  
  \(R_\alpha(P_1[[\text{true,false} / \text{ok,wait}] \parallel Q_1[[\text{true,false} / \text{ok,wait}]] \parallel R_\alpha(P_2[[\text{true,false} / \text{ok,wait}] \parallel Q_2[[\text{true,false} / \text{ok,wait}])]\)
  
  by (simp add: RHS-design-ok-wait)
```

**QED**

78
also from `assms`

have ... =
  \( R_1 ((R1 \land R2c (P_1)) \land R1 (R2c (P_2))) \)
  \( (R1 (R2c (P_1 \Rightarrow Q_1)) \land R1 (R2c (P_2 \Rightarrow Q_2))) \)
apply (simp add: rea-design-par-def rea-pre-RHS-design rea-cmt-RHS-design unsubst unrest `assms`)
apply (rule cong[of \( R_1, R_2 \), simp])
using `assms` apply (rel-auto)
done
also have ... =
  \( R_1 ((P_1 \land P_2) \Rightarrow (R1 (R2c (P_1 \Rightarrow Q_1)) \land R1 (R2c (P_2 \Rightarrow Q_2))) \)
by (metis (no-types, hide-lams) R1-R2s-R2c R1-conj R1-design-R1-pre RHS-design-ok-wait)
also have ... =
  \( R_1 ((P_1 \land P_2) \Rightarrow ((P_1 \Rightarrow Q_1) \land (P_2 \Rightarrow Q_2))) \)
by (simp (no-types, lifting) R1-conj R2s-conj RHS-design-export-R1 RHS-design-export-R2s)
also have ... =
  \( R_1 ((P_1 \land P_2) \Rightarrow (Q_1 \land Q_2)) \)
by (rule cong[of \( R_1, R_2 \), simp, rel-auto]
finally show `?thesis`.
qed

\textbf{lemma} \( \text{RHS-tri-design-par;} \)
\textbf{assumes} \( \$ok` \: `P_1, `ok` \: `P_2 \)
\textbf{shows} \( R_1 ((P_1 \Rightarrow Q_1) \land R_1 (P_2 \Rightarrow Q_2)= R_1 ((P_1 \land P_2) \Rightarrow (Q_1 \land Q_2) \land (R_1 \land R_2))) \)
by (simp add: RHS-design-par `assms` unrest `wait`-cond-conj-exchange)

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

\textbf{lemma} \( \text{RHS-comp-assoc;} \)
\textbf{assumes} \( P \: \text{is} \: \text{NSRD} \: Q \: \text{is} \: \text{NSRD} \: R \: \text{is} \: \text{NSRD} \)
\textbf{shows} \( (P \parallel R) \parallel R = P \parallel R \parallel R \)
by (`rdes-eq cls: `assms`)

declare qed

\section{Productive Reactive Designs}

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

\subsection{Healthiness condition}

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

\textbf{definition} \( \text{Productive :: `s, `t::trace, `a) hrel-rsp} \Rightarrow \text{(`s, `t, `a) hrel-rsp} \text{where} \)
\textbf{assumes} \( \text{Productive}(P) = P \parallel_R R_1 \parallel_R (true \parallel true \parallel (\$tr <_u \$tr')) \)
\end{theory}

\section{Productive Reactive Designs}

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

\subsection{Healthiness condition}

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

\textbf{definition} \( \text{Productive :: (`s, `t::trace, `a) hrel-rsp} \Rightarrow \text{(`s, `t, `a) hrel-rsp} \text{where} \)
\textbf{assumes} \( \text{Productive}(P) = P \parallel_R R_1 \parallel_R (true \parallel true \parallel (\$tr <_u \$tr')) \)
\end{theory}
shows \( \text{Productive}(R_u(P \vdash Q \circ R)) = R_u(P \vdash Q \circ (R \land \$tr <_u \$tr')) \)
using \text{assms} by (simp add: Productive-def RHS-tri-design-par unrest)

\begin{align*}
\text{lemma Productive-form:} \\
\text{Productive}(\text{SRD}(P)) &= R_u(\text{pre}_R(P) \vdash \text{peri}_R(P) \circ (\text{post}_R(P) \land \$tr <_u \$tr')) \\
\text{proof} - & \\
\text{have Productive}(\text{SRD}(P)) = R_u(\text{pre}_R(P) \vdash \text{peri}_R(P) \circ \text{post}_R(P)) \parallel R_u(\text{true} \vdash \text{true} \circ (\$tr <_u \$tr')) \\
& \quad \text{by (simp add: Productive-def SRD-as-reactive-tri-design)} \\
\text{also have} \ldots = R_u(\text{pre}_R(P) \vdash \text{peri}_R(P) \circ (\text{post}_R(P) \land \$tr <_u \$tr')) \\
& \quad \text{by (simp add: RHS-tri-design-par unrest)} \\
\text{finally show} \ ?\text{thesis} .
\end{align*}

\text{qed}

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

\begin{align*}
\text{lemma Productive-intro:} \\
\text{assumes} P \text{ is SRD} (\$tr <_u \$tr') \subseteq (\text{pre}_R(P) \land \text{post}_R(P)) \$\text{wait}' \notin \text{pre}_R(P) \\
\text{shows} P \text{ is Productive} \\
\text{proof} - & \\
\text{have} P, R_u(\text{pre}_R(P) \vdash \text{peri}_R(P) \circ (\text{post}_R(P) \land \$tr <_u \$tr')) = P \\
\text{proof} - & \\
\text{have} R_u(\text{pre}_R(P) \vdash \text{peri}_R(P) \circ \text{post}_R(P)) = R_u(\text{pre}_R(P) \vdash (\text{pre}_R(P) \land \text{peri}_R(P)) \circ (\text{pre}_R(P) \land \text{post}_R(P))) \\
& \quad \text{by (metis (no-types, hide-lams) design-export-pre wait’-cond-conj-exchange wait’-cond-idem)} \\
\text{also have} \ldots = R_u(\text{pre}_R(P) \vdash (\text{pre}_R(P) \land \text{peri}_R(P)) \circ (\text{pre}_R(P) \land (\text{post}_R(P) \land \$tr <_u \$tr'))) \\
& \quad \text{by (metis assms(2) utp-pred-laws.inf.absorb1 utp-pred-laws.inf.assoc)} \\
\text{also have} \ldots = R_u(\text{pre}_R(P) \vdash \text{peri}_R(P) \circ (\text{post}_R(P) \land \$tr <_u \$tr')) \\
& \quad \text{by (metis (no-types, hide-lams) design-export-pre wait’-cond-conj-exchange wait’-cond-idem)} \\
\text{finally show} \ ?\text{thesis} \\
& \quad \text{by (simp add: SRD-reactive-tri-design assms(1))} \\
\text{qed} \\
\text{thus} \ ?\text{thesis} \\
& \quad \text{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)} \\
\text{qed}
\end{align*}

\begin{align*}
\text{lemma Productive-refines-tr-increase:} \\
\text{assumes} P \text{ is SRD} P \text{ is Productive } \$\text{wait}' \notin \text{pre}_R(P) \\
\text{shows} (\$tr <_u \$tr') \subseteq (\text{pre}_R(P) \land \text{post}_R(P)) \\
\text{proof} - & \\
\text{have} \text{post}_R(P) = \text{post}_R(R_u(\text{pre}_R(P) \vdash \text{peri}_R(P) \circ (\text{post}_R(P) \land \$tr <_u \$tr'))) \\
& \quad \text{by (metis Healthy-def Productive-form assms(1) assms(2))} \\
\text{also have} \ldots = R1(\text{R2c}(\text{pre}_R(P) \Rightarrow (\text{post}_R(P) \land \$tr <_u \$tr'))) \\
& \quad \text{by (simp add: rea-post-RHS-design unrest uesubst assms, rel-auto)} \\
\text{also have} \ldots = R1((\text{pre}_R(P) \Rightarrow (\text{post}_R(P) \land \$tr <_u \$tr'))) \\
& \quad \text{by (simp add: R2c-impl R2c-preR R2c-postR R2c-and R2c-tr-less-tr' assms)} \\
\text{also have} (\$tr <_u \$tr') \subseteq (\text{pre}_R(P) \land \ldots) \\
& \quad \text{by (rel-auto)} \\
\text{finally show} \ ?\text{thesis} .
\end{align*}

\text{qed}

\begin{align*}
\text{lemma Continuous-Productive [closure]:} \text{ Continuous Productive} \\
& \quad \text{by (simp add: Continuous-def Productive-def, rel-auto)}
\end{align*}
9.2 Reactive design calculations

**Lemma preR-Productive** [rdes]:
- **Assumes** $P$ is SRD
- **Shows** $\text{pre}_R(\text{Productive}(P)) = \text{pre}_R(P)$

**Proof**
- **Have** $\text{pre}_R(\text{Productive}(P)) = \text{pre}_R(\text{peri}_R(P) \circ (\text{post}_R(P) \land \$tr <_u \$tr'))$
  - **By** (metis Healthy-def Productive-form assms)
  - **Thus** \( ?\)thesis
  - **By** (simp add: rea-pre-RHS-design usubst unrest R2c-not R2c-preR R1-preR Healthy-if assms)

**Qed**

**Lemma periR-Productive** [rdes]:
- **Assumes** $P$ is NSRD
- **Shows** $\text{peri}_R(\text{Productive}(P)) = \text{peri}_R(P)$

**Proof**
- **Have** $\text{peri}_R(\text{Productive}(P)) = \text{peri}_R(\text{pre}_R(P) \circ (\text{post}_R(P) \land \$tr <_u \$tr'))$
  - **By** (metis Healthy-def NSRD-is-SRD Productive-form assms)
  - **Also have** \( \ldots = R1 (R2c (\text{pre}_R P \Rightarrow_r \text{peri}_R P)) \)
  - **By** (simp add: rea-peri-RHS-design usubst unrest R2c-not assms closure)
  - **Also have** \( \ldots = (\text{pre}_R P \Rightarrow_r \text{peri}_R P) \)
  - **By** (simp add: R1rea-impl R2c-rea-impl R2c-preR R2c-peri-SRD R1 peri-SRD assms closure R1-tr-less-tr' R2c-tr-less-tr')

**Finally show** \( ?\)thesis
- **By** (simp add: SRD-peri-under-pre assms unrest closure)

**Qed**

**Lemma postR-Productive** [rdes]:
- **Assumes** $P$ is NSRD
- **Shows** $\text{post}_R(\text{Productive}(P)) = (\text{pre}_R(P) \Rightarrow_r \text{post}_R(P) \land \$tr <_u \$tr')$

**Proof**
- **Have** $\text{post}_R(\text{Productive}(P)) = \text{post}_R(\text{peri}_R(P) \circ (\text{post}_R(P) \land \$tr <_u \$tr'))$
  - **By** (metis Healthy-def NSRD-is-SRD Productive-form assms)
  - **Also have** \( \ldots = R1 (R2c (\text{pre}_R P \Rightarrow_r \text{post}_R P \land \$tr'>_u \$tr)) \)
  - **By** (simp add: rea-post-RHS-design usubst unrest assms closure)
  - **Also have** \( \ldots = (\text{pre}_R P \Rightarrow_r \text{post}_R P \land \$tr'>_u \$tr) \)
  - **By** (simp add: R1rea-impl R2c-rea-impl R2c-preR R2c-and R1-extend-conj' R2c-post-SRD R1 post-SRD assms closure R1-tr-less-tr' R2c-tr-less-tr')

**Finally show** \( ?\)thesis
- **By** (simp add: SRD-peri-under-pre assms unrest closure)

**Qed**

**Lemma preR-frame-seq-export:**
- **Assumes** $P$ is NSRD $P$ is Productive $Q$ is NSRD
- **Shows** $(\text{pre}_R P \land (\text{pre}_R P \land \text{post}_R P) \land Q) = (\text{pre}_R P \land (\text{post}_R P \land Q))$

**Proof**
- **Have** $(\text{pre}_R P \land (\text{post}_R P \land Q) = (\text{pre}_R P \land ((\text{pre}_R P \Rightarrow_r \text{post}_R P) \land Q))$
  - **By** (simp add: SRD-post-under-pre assms closure unrest)
  - **Also have** \( \ldots = (\text{pre}_R P \land ((\neg_r \text{pre}_R P) \land Q \lor (\text{pre}_R P \Rightarrow_r R1(\text{post}_R P)) \land Q)) \)
  - **By** (simp add: NSRD-is-SRD R1 post-SRD assms(1) rea-impl-def seqr-or-distl R1-preR Healthy-if)
  - **Also have** \( \ldots = (\text{pre}_R P \land ((\neg_r \text{pre}_R P) \land Q \lor (\text{pre}_R P \land \text{post}_R P) \land Q)) \)
  - **Proof**
    - **Have** $\text{pre}_R P \lor \neg_r \text{pre}_R P = R1 \text{ true}$
    - **By** (simp add: R1-preR rea-not-or)
  - **Then show** \( ?\)thesis
    - **By** (metis (no-types, lifting) R1-def conj-comm disj-comm disj-conj-distr rea-impl-def seqr-or-distl utp-pred-laws.inf-top-left utp-pred-laws.sup.left-idem)

81
also have ... = (\text{pre}_R P \land (\neg \text{pre}_R P) \lor (\text{pre}_R P \land \text{post}_R P) \land Q))
    by (simp add: NSRD-neg-pre-left-zero assms closure SRD-healths)
also have ... = (\text{pre}_R P \land (\text{pre}_R P \land \text{post}_R P) \land Q)
    by (rel-blast)
finally show ?thesis ..

9.3 Closure laws

lemma Productive-rdes-intro:
  assumes ($\text{tr} <_u \text{tr}^\prime$) \sqsubseteq R $\text{ok}^\prime$ $\text{tr}$ $\text{wait}^\prime$ $\text{tr} \quad P$ $\text{wait}^\prime$ $\text{tr} \quad P$ $\text{wait}^\prime$ $\text{tr} \quad P$
  shows (\text{R}_u(P \vdash Q \circ R)) is Productive
proof (rule Productive-intro)
  show R_u (P \vdash Q \circ R) is SRD
    by (simp add: RHS-tri-design-is-SRD assms)

from assms(1) show ($\text{tr} \circ_r >_u \text{tr}$) \sqsubseteq (\text{pre}_R (\text{R}_u (P \vdash Q \circ R)) \land \text{post}_R (\text{R}_u (P \vdash Q \circ R)))
  apply (simp add: rea-pre-RHS-design rea-post-RHS-design usubst assms unrest)
  using assms(1) apply (rel-auto)
  apply fastforce
  done

show $\text{wait}^\prime$ $\text{tr} \quad \text{pre}_R (\text{R}_u (P \vdash Q \circ R))$
  by (simp add: rea-pre-RHS-design rea-post-RHS-design usubst R1-def R2c-def R2s-def assms unrest)

We use the $R_4^\prime$ 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 (\text{R}_u(P \vdash Q \circ R)) is Productive
using assms by (simp add: Productive-rdes-intro R4-iff-refine unrest)

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

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

lemma Productive-intChoice [closure]:
  assumes P is SRD P is Productive Q is SRD Q is Productive
  shows P \sqcap Q is Productive
proof
  have P \sqcap Q =
    \text{R}_u(\text{pre}_R(P) \vdash \text{peri}_R(P) \circ (\text{post}_R(P) \land \text{tr} <_u \text{tr}^\prime)) \sqcap \text{R}_u(\text{pre}_R(Q) \vdash \text{peri}_R(Q) \circ (\text{post}_R(Q) \land \text{tr} <_u \text{tr}^\prime))
    by (metis Healthy-if Productive-form assms)
also have ... = \text{R}_u ((\text{pre}_R P \land \text{pre}_R Q) \vdash (\text{peri}_R P \lor \text{peri}_R Q) \circ (((\text{post}_R P \land \text{tr} >_u \text{tr}) \lor (\text{post}_R Q \land \text{tr} >_u \text{tr})))
    by (simp add: RHS-tri-design-choice)
also have ... = \text{R}_u ((\text{pre}_R P \land \text{pre}_R Q) \vdash (\text{peri}_R P \lor \text{peri}_R Q) \circ (((\text{post}_R P) \lor (\text{post}_R Q) \land \text{tr} >_u \text{tr})))

82
by (rule cong[of R, R], simp, rel-auto)
also have ... is Productive
  by (simp add: Healthy-def Productive-RHS-design-form unrest)
finally show ?thesis.
qed

lemma Productive-cond-rea [closure]:
  assumes P is SRD P is Productive Q is SRD Q is Productive
  shows P ∨ b ∨ R Q is Productive
proof
  have P ∨ b ∨ R Q =
    Rₜ(preₜ(P) ⊨ periₜ(P) ∨ (postₜ(P) ∧ $tr <ₜ $tr') ∨ b ∨ R Rₜ(preₜ(Q) ⊨ periₜ(Q) ∨ (postₜ(Q) ∧ $tr <ₜ $tr'))
  by (metis Healthy-if Productive-form assms)
also have ... = Rₜ ((preₜ(P) ∨ b ∨ R preₜ(Q)) ⊨ (periₜ(P) ∨ b ∨ R periₜ(Q) ∨ ((postₜ(P) ∧ $tr' >ₜ $tr) ∨ b ∨ R (postₜ(Q) ∧ $tr' >ₜ $tr)))
  by (simp add: cond-srea-form)
also have ... = Rₜ ((preₜ(P) ∨ b ∨ R preₜ(Q)) ⊨ (periₜ(P) ∨ b ∨ R periₜ(Q) ∨ ((postₜ(P) ∨ b ∨ R (postₜ(Q)))) ∧ $tr' >ₜ $tr))
  by (rule cong[of R, R], simp, rel-auto)
also have ... is Productive
  by (simp add: Healthy-def Productive-RHS-design-form unrest)
finally show ?thesis.
qed

lemma Productive-seq-1 [closure]:
  assumes P is NSRD P is Productive Q is NSRD
  shows P ;; Q is Productive
proof
  have P ;; Q = Rₜ(preₜ(P) ⊨ periₜ(P) ∨ (postₜ(P) ∧ $tr <ₜ $tr')) ;; Rₜ(preₜ(Q) ⊨ periₜ(Q) ∨ (postₜ(Q)))
  by (metis Healthy-def NSRD-is-SRD SRD-reactive-tri-design Productive-form assms(1) assms(2) assms(3))
also have ... = Rₜ ((preₜ(P) ∧ (postₜ(P) ∧ $tr' >ₜ $tr) wpₑ preₜ(Q)) ⊨
    (periₜ(P) ∨ ((postₜ(P) ∧ $tr' >ₜ $tr) ;; periₜ(Q)) ∨ ((postₜ(P) ∧ $tr' >ₜ $tr) ;; postₜ(Q)))
  by (simp add: Rₜ NSRD-is-SRD Rₜ wpₑ unrest closure assms wp NSRD-neg-pre-left-zero SRD-healths ex-unrest wpₑ-aut-def disj-aut-def)
also have ... = Rₜ ((preₜ(P) ∧ (postₜ(P) ∧ $tr' >ₜ $tr) wpₑ preₜ(Q)) ⊨
    (periₜ(P) ∨ ((postₜ(P) ∧ $tr' >ₜ $tr) ;; periₜ(Q)) ∨ ((postₜ(P) ∧ $tr' >ₜ $tr) ;; postₜ(Q) ∧ $tr' >ₜ $tr))
  by (rel-auto)
  thus ?thesis
  by (simp add: NSRD-is-SRD R₁-post-SRD assms)
qed
also have ... is Productive
  by (rule Productive-rdes-intro, simp-all add: unrest assms closure wpₑ-aut-def)
finally show ?thesis.
qed

lemma Productive-seq-2 [closure]:
  assumes P is NSRD Q is NSRD Q is Productive
Guarded recursion relies on our ability to measure the trace’s size, in order to see if it is non-empty. A trace has a positive size. But this may not be the case with all trace models and is possibly stronger than necessary. In particular, $0 < \#s \Rightarrow 0 < \#u(\forall s)$ requires that a non-empty trace have a positive size. But this may not be the case with all trace models and is possibly more restrictive than necessary. In future we will explore weakening.

begin

lemma size-mono: $s \leq t \Rightarrow size(s) \leq size(t)$
  by (metis le-add1 local.diff-add-cancel-left' local.size-plus)

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

lemma trace-strict-prefixE: $xs < ys \Rightarrow (\exists zs. [ys = xs + zs; size(zs) > 0 ] \Rightarrow thesis) \Rightarrow thesis$

end
by (metis local.diff-add-cancel-left' local.less-iff local.minus-gr-zero-iff local.size-nzero)

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

end

Both natural numbers and lists are measurable trace algebras.

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

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

syntax 
-usize :: logic ⇒ logic (size_u' ('-'))
translations 
size_u(t) == CONST uop CONST size t

10.2 Guardedness

definition gvr_t :: ('t::size-trace,α) rp × ('t,α) rp chain where
[upred-defs]: gvr_t(n) ≡ ($tr \leq_u$ tr' ∧ size_u(αt) <_u <n>)

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

lemma gvr_t-limit: \( \bigcap \) (range gvr_t) = ($tr \leq_u$ tr')
by (rel-auto)

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

lemma GuardedI: \( \bigwedge \) X n. (F(X) ∧ gvr_t(n+1)) = (F(X ∧ gvr_t(n)) ∧ gvr_t(n+1)) \( \implies \) Guarded F
by (simp add: Guarded-def)

Guarded reactive designs yield unique fixed-points.

theorem guarded-fp-uniq:
assumes mono F F ∈ [id]_H ⇒ [SRD]_H Guarded F
shows \( \mu \ F = \nu \ F \)
proof –
  have constr F gvr_t
    using assms
    by (auto simp add: constr-def gvr_t-chain Guarded-def tcontr-alt-def')
  hence ($tr \leq_u$ tr' ∧ \( \mu \ F \)) = ($tr \leq_u$ tr' ∧ \( \nu \ F \))
    apply (rule constr-fp-uniq)
    apply (simp add: assms)
  using gvr_t-limit apply blast
done
moreover have ($tr \leq_u$ tr' ∧ \( \mu \ F \)) = \( \mu \ F \)
proof –
have $F$ is $R1$
  by (rule SRD-healths(1), rule Healthy-mu, simp-all: assms)
thus $\text{thesis}$
  by (metis Healthy-def R1-def conj-comm)

qed

moreover have $(\$tr \leq_u \$tr' \land \nu F) = \nu F$
proof -
  have $\nu F$ is $R1$
  by (rule SRD-healths(1), rule Healthy-nu, simp-all: assms)
  thus $\text{thesis}$
  by (metis Healthy-def R1-def conj-comm)

qed

ultimately show $\text{thesis}$
  by (simp)

qed

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

lemma UINF-Guarded [closure]:
  assumes $\bigwedge P. \ P \in A \implies \text{Guarded } P$
  shows Guarded $(\lambda X. \bigwedge P \in A \cdot P(X))$
proof (rule GuardedI)
  fix $X\ n$
  have $\bigwedge Y. (\bigwedge P \in A \cdot P Y) \land \text{gvrt}(n+1)) = (\bigwedge P \in A \cdot (P Y \land \text{gvrt}(n+1)) \land \text{gvrt}(n+1))$
proof -
  fix $Y$
  let $?lhs = (\bigwedge P \in A \cdot P Y) \land \text{gvrt}(n+1))$ and $?rhs = (\bigwedge P \in A \cdot (P Y \land \text{gvrt}(n+1)) \land \text{gvrt}(n+1))$
  have $a$:?lhs[false/$\text{ok}$] = $?rhs[false/$\text{ok}$]
    using $a\ b\ c$
    by (rule-tac bool-eq-splitI[of in-var ok], simp, rule-tac bool-eq-splitI[of in-var wait], simp-all)

  qed

moreover have $(\bigwedge P \in A \cdot (P X \land \text{gvrt}(n+1))) \land \text{gvrt}(n+1)) = (\bigwedge P \in A \cdot (P (X \land \text{gvrt}(n)) \land \text{gvrt}(n+1))) \land \text{gvrt}(n+1))$
proof -
  have $(\bigwedge P \in A \cdot (P X \land \text{gvrt}(n+1))) = (\bigwedge P \in A \cdot (P (X \land \text{gvrt}(n)) \land \text{gvrt}(n+1)))$
  proof (rule UINF-cong)
    fix $P \ assume P \in A$
    thus $(P X \land \text{gvrt}(n+1)) = (P (X \land \text{gvrt}(n)) \land \text{gvrt}(n+1))$
      using Guarded-def assms by blast
  qed
  thus $\text{thesis}$ by simp

  qed

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

  qed

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

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

proof

  - have Guarded \((\lambda X. \; \bigwedge F\in\{P,Q\} \cdot F(X))\)
    
    by (rule UNF-Guarded, auto simp add: assms)

  thus \(\Box\)thesis
    
    by (simp)

qed

lemma cond-srea-Guarded [closure]:

assumes \(\Box\)Guarded \(P\) Guarded \(Q\)

shows Guarded \((\lambda X. \; P(X) \triangleleft b \triangleright_{R} Q(X))\)

using assms by (rel-auto)

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

lemma Guarded-if-Productive [closure]:

fixes \(P::('s, 't::size-trace,'a) hrel-rsp\)

assumes \(P\) is NSRD \(P\) is Productive

shows Guarded \((\lambda X. \; P :: SRD(X))\)

proof (clar simp simp add: Guarded-def)

— We split the proof into three cases corresponding to valuations for ok, wait, and wait’ respectively.

fix \(X\) \(n\)

have \(a::(P :: SRD(X) \land gvt (Suc n))[false/sok] =\)

\((P :: SRD(X \land gvt n) \land gvt (Suc n))[false/sok]\)

by (simp add: usubst closure SRD-left-zero-1 assms)

have \(b::(P :: SRD(X) \land gvt (Suc n))[true/sok])[true/wait] =\)

\(((P :: SRD(X \land gvt n) \land gvt (Suc n))[true/sok])[true/wait]\)

by (simp add: usubst closure SRD-left-zero-2 assms)

have \(c::(P :: SRD(X) \land gvt (Suc n))[false/wait] =\)

\(((P :: SRD(X \land gvt n) \land gvt (Suc n))[true/sok])[false/wait]\)

proof —

  - have \(1::(P[true/wait'] :: SRD X)[true/wait] \land gvt (Suc n))[true/false/sok,$wait] =\)

    \((P[true/wait'] :: SRD (X \land gvt n))\land gvt (Suc n))[true/false/sok,$wait]\)

    by (metis (no-types, lifting) Healthy-def R3h-wait-true SRD-healths(3) SRD-idem)

  - have \(2::(P[false/wait'] :: SRD X)[false/wait] \land gvt (Suc n))[true/false/sok,$wait] =\)

    \((P[false/wait'] :: SRD (X \land gvt n))[false/wait] \land gvt (Suc n))[true/false/sok,$wait]\)

    proof —

    - have \(exp:\forall Y::('s, 't,'a) hrel-rsp.\()

      \(P[false/wait'] :: SRD Y)[false/wait] \land gvt (Suc n))[true/false/sok,$wait] =\)

      \(((\neg r\; pre\; P) :: SRD(Y))[false/wait] \lor (post_{R} \; P \land \; \; str \;' >_{u} \; str) :: SRD Y)[true/false/sok,$wait]\)

    - have \(X::(\forall Y::('s, 't,'a) hrel-rsp.\)

      \(P[false/wait'] :: SRD Y)[false/wait] \land gvt (Suc n))[true/false/sok,$wait] =\)

      \(((R_{S} (\neg r\; pre\; P) \lor \; pre_{R}(P) \land \; \; str <_{u} \; str) :: SRD Y)[false/wait]\)

      \land gvt (Suc n))[true/false/sok,$wait]\)

      by (metis Healthy-def Productive-form assms(1) assms(2) NSRD-is-SRD)

    also have \(=\)

    \(((R_{1}(R_{2} (\neg r\; pre\; P) \lor \; pre_{R}(P) \land \; \; str <_{u} \; str')) :: SRD Y)[false/wait]\)

    \land gvt (Suc n))[true/false/sok,$wait]\)

    by (simp add: RHS-def R1-def R2c-def R2s-def RD1-def RD2-def usubst unrest assms closure design-def)

    also have \(=\)

    \(((\neg r\; pre_{R}(P) \lor \; post_{R}(P) \land \; \; str <_{u} \; str') :: SRD Y)[false/wait]\)
\( \land \text{gert (Suc } n)\)[true,false/$\text{ok}$,$\text{swait}] \\
\text{by (simp add: impl-all-def R2c-dijr R1-disj R2c-not assms closure R2c-and} \\
R2c-preR rea-not-def R1-extend-conj' R2c-ok' R2c-post-SRD R1-tr-less-tr' R2c-tr-less-tr') \\
\text{also have } \ldots = \\
(((\neg \text{pre}_R P) :: (\text{SRD}(Y))[false/$\text{swait}] \lor (\text{ok}' \land \text{post}_R P \land \text{tr}' > u \text{ tr}) :: (\text{SRD} Y)[false/$\text{swait}] \land \text{gert (Suc } n)))[true,false/$\text{ok}$,$\text{swait}] \\
\text{by (simp add: usubst unrest assms closure seq-or-distl NSRD-neg-pre-left-zero SRD-healths} \\
\text{also have } \ldots = \\
(((\neg \text{pre}_R P) :: (\text{SRD}(Y))[false/$\text{swait}] \lor (\text{post}_R P \land \text{tr}' > u \text{ tr}) :: (\text{SRD} Y)[true,false/$\text{ok}$,$\text{swait}]) \\
\land \text{gert (Suc } n))[true,false/$\text{ok}$,$\text{swait}] \\
\text{proof } \\
\text{have } (\text{post}_R P \land \text{tr}' > u \text{ tr}) :: (\text{SRD} Y)[false/$\text{swait}] = \\
((\text{post}_R P \land \text{tr}' > u \text{ tr}) \land \text{ok}' = u \text{ true}) :: (\text{SRD} Y)[false/$\text{swait}] \\
\text{by (rel-blast) } \\
\text{also have } \ldots = (\text{post}_R P \land \text{tr}' > u \text{ tr})[true/$\text{ok}'] :: (\text{SRD} Y)[false/$\text{swait}][true/$\text{ok}] \\
\text{using seq-or-left-one-point[of ok (post}_R P \land \text{tr}' > u \text{ tr}) True (\text{SRD} Y)[false/$\text{swait}]] \\
\text{by (simp add: true-alt-def[THEN sym])} \\
\text{finally show } ?\text{thesis by (simp add: usubst unrest) } \\
\text{qed} \\
\text{finally } \\
\text{show } (P)[false/$\text{swait}'] :: (\text{SRD} Y)[false/$\text{swait}] \land \text{gert (Suc } n))[true,false/$\text{ok}$,$\text{swait}] = \\
(((\neg \text{pre}_R P) :: (\text{SRD}(Y))[false/$\text{swait}] \lor (\text{post}_R P \land \text{tr}' > u \text{ tr}) :: (\text{SRD} Y)[true,false/$\text{ok}$,$\text{swait}]) \\
\land \text{gert (Suc } n))[true,false/$\text{ok}$,$\text{swait}] . \\
\text{qed} \\
\text{have } 1::(\text{post}_R P \land \text{tr}' > u \text{ tr}) :: (\text{SRD} X)[true,false/$\text{ok}$,$\text{swait}] \land \text{gert (Suc } n)) = \\
((\text{post}_R P \land \text{tr}' > u \text{ tr}) :: (\text{SRD} (X \land \text{gert } n))[true,false/$\text{ok}$,$\text{swait}] \land \text{gert (Suc } n)) \\
\text{apply (rel-auto) } \\
\text{apply (rename-tac tr st more ok wait tr' st' more' tr}_0 \text{ st}_0 \text{ more}_0 \text{ ok'}) \\
\text{apply (rule-tac x=tr}_0 \text{ in eIx, rule-tac x=st}_0 \text{ in eIx, rule-tac x=more}_0 \text{ in eIx} \\
\text{apply (simp) } \\
\text{apply (erule trace-strict-prefixE) } \\
\text{apply (rename-tac tr st ref ok wait tr' st' ref' tr}_0 \text{ st}_0 \text{ ref}_0 \text{ ok' zs) } \\
\text{apply (rule-tac x=False in eIx) } \\
\text{apply (simp add: size-minus-trace) } \\
\text{apply (subgoal-tac size(tr) < size(tr}_0)) \\
\text{apply (simp add: less-diff-conv2 size-mono) } \\
\text{using size-strict-mono apply blast } \\
\text{apply (rename-tac tr st more ok wait tr' st' more' tr}_0 \text{ st}_0 \text{ more}_0 \text{ ok'}) \\
\text{apply (rule-tac x=tr}_0 \text{ in eIx, rule-tac x=st}_0 \text{ in eIx, rule-tac x=more}_0 \text{ in eIx) } \\
\text{apply (simp) } \\
\text{apply (erule trace-strict-prefixE) } \\
\text{apply (rename-tac tr st more ok wait tr' st' more' tr}_0 \text{ st}_0 \text{ more}_0 \text{ ok' zs) } \\
\text{apply (auto simp add: size-minus-trace) } \\
\text{apply (subgoal-tac size(tr) < size(tr}_0)) \\
\text{apply (simp add: less-diff-conv2 size-mono) } \\
\text{using size-strict-mono apply blast done } \\
\text{have } 2::(\neg \text{pre}_R P) :: (\text{SRD} X)[false/$\text{swait}] = (\neg \text{pre}_R P) :: (\text{SRD}(X \land \text{gert } n))[false/$\text{swait}] \\
\text{by (simp add: NSRD-neg-pre-left-zero closure assms SRD-healths) } \\
\text{show } ?\text{thesis by (simp add: exp 1 2 utp-pred-laws.inf-sup-distrib2) } \\
\text{qed}
show ?thesis
proof -
  have \((P :: \text{SRD}\ X \land \text{gvr}\ (n+1))[\text{true}/\text{false}/\text{ok}/\text{wait}] = (P[\text{true}/\text{false}/\text{wait}] ;; \text{SRD}\ X)[\text{true}/\text{false}/\text{wait}] \land \text{gvr}\ (n+1))[\text{true}/\text{false}/\text{ok}/\text{wait}] \lor (P[\text{false}/\text{false}/\text{wait}] ;; \text{SRD}\ X)[\text{false}/\text{false}/\text{wait}] \land \text{gvr}\ (n+1))[\text{true}/\text{false}/\text{ok}/\text{wait}]\) by (simp add: subst seqr-bool-split[of wait], simp add: subst utp-pred-laws.distrib(4))
also
  have \(\ldots = ((P[\text{true}/\text{false}/\text{wait}] ;; \text{SRD}\ X)[\text{false}/\text{false}/\text{wait}] \land \text{gvr}\ (n+1))[\text{true}/\text{false}/\text{ok}/\text{wait}] \lor (P[\text{false}/\text{false}/\text{wait}] ;; \text{SRD}\ X)[\text{false}/\text{false}/\text{wait}] \land \text{gvr}\ (n+1))[\text{true}/\text{false}/\text{ok}/\text{wait}]\) by (simp add: subst seqr-bool-split[of wait], simp add: subst utp-pred-laws.distrib(4))
also
  have \(\ldots = (P :: \text{SRD}\ X \land \text{gvr}\ n)[\text{true}/\text{false}/\text{ok}/\text{wait}] \land \text{gvr}\ (n+1))[\text{true}/\text{false}/\text{ok}/\text{wait}]\) by (simp add: subst seqr-bool-split[of wait], simp add: subst utp-pred-laws.distrib(4))
also have \(\ldots = (P :: \text{SRD}\ X \land \text{gvr}\ n)[\text{false}/\text{false}/\text{ok}/\text{wait}] \land \text{gvr}\ (n+1))[\text{false}/\text{false}/\text{ok}/\text{wait}]\) by (simp add: subst seqr-bool-split[of wait], simp add: subst utp-pred-laws.distrib(4))
finally show ?thesis by (simp add: subst)
qed

10.3 Tail recursive fixed-point calculations

declare upred-semiring.power-Suc [simp]

lemma mu-csp-form-1 [rdes]:
  fixes P :: \(\langle s, t::\text{size-trace}, \alpha\rangle\) hrel-rsp
  assumes P is NSRD P is Productive
  shows \((\mu X \cdot P ;; \text{SRD}\ X) = (\langle \prod i \cdot P \rangle \cdot (i+1)) :: \text{Miracle}\)
proof -
  have \(1:\text{Continuous}\ (\lambda X. P :: \text{SRD}\ X)\)
    using SRD-Continuous
    by (clarsimp simp add: Continuous-def seq-SUP-distl[THEN sym], drule-tac x=A in spec, simp)
  have \(2: (\lambda X. P :: \text{SRD}\ X) \in [\text{id}]_H \Rightarrow [\text{SRD}]_H\)
    by (blast intro: funcsetI closure assms)
  with \(1 \ 2\) have \((\mu X \cdot P ;; \text{SRD}\ X) = (\nu X \cdot P ;; \text{SRD}\ X)\)
    by (simp add: guarded-fp-uniq Guarded-if-Productive[of assms] funcsetI closure)
  also have \(\ldots = (\langle \lambda X. P ;; \text{SRD}\ X \rangle \cdot (i+1))\)
    by (simp add: sup-continuous-lfp 1 sup-continuous-Continuous false-upred-def)
  also have \(\ldots = (\langle \lambda X. P ;; \text{SRD}\ X \rangle \cdot (\prod i. ((\lambda X. P ;; \text{SRD}\ X) \cdot i) \cdot false))\)
    by (simp)
  also have \(\ldots = (\langle \prod i. (\lambda X. P ;; \text{SRD}\ X) \cdot i) \cdot false\)
    by (simp)
  also have \(\ldots = (\prod i. P \cdot (i+1))\)
    by (rule SUP-cong, simp-all)
proof
fix i
show P ;; SRD (((λX. P ;; SRD X) ^^ i) false) = (P ;; P ^ i) ;; Miracle
proof (induct i)
case 0
then show ?case
  by (simp, metis srdes-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[THEN sym] srdes-theory-continuous.weak.top-closed)
qed

also have ...
  = (∈ UNIV 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,'a) hrel-rsp
assumes P is NSRD P is Productive
shows (μ 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,'a) hrel-rsp
assumes P is NSRD P is Productive
shows (μ X · P ;; SRD(X)) = (P ;; P*) ;; Miracle
proof –
  have (μ X · P ;; SRD(X)) = (∈ UNIV i. 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-distr[THEN sym], simp add: ustar-def)
  finally show ?thesis .
qed

declare upred-semiring.power-Suc [simp del]
end

11 Reactive Design Programs

theory utp-rdes-prog
imports
  utp-rdes-normal
  utp-rdes-tactics
  utp-rdes-parallel
  utp-rdes-guarded
  UTP−KAT.utp-kleene
begin

11.1 State substitution

lemma srd-subst-RHS-tri-design [usubst]:
\[ \sigma \]_{SR} \vdash R_s(P \vdash Q \diamond R) = R_s(\langle \sigma \rangle_{SR} \vdash P \vdash (\langle \sigma \rangle_{SR} \vdash Q) \diamond (\langle \sigma \rangle_{SR} \vdash R)) \\
\text{by (rel-auto)}

**Lemma srd-subst-SRD-closed [closure]:**

**assumes** \( P \) is SRD  
**shows** \( \langle \sigma \rangle_{SR} \vdash P \) is SRD  

**proof**  
have \( SRD(\langle \sigma \rangle_{SR} \vdash (SRD P)) = \langle \sigma \rangle_{SR} \vdash (SRD P) \)  
\text{by (rel-auto)}  
thus \( ?\text{thesis} \)  
\text{by (metis Healthy-def assms)}  
**qed**

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

\[ \text{preR}(\langle \sigma \rangle_{SR} \vdash P) = \langle \sigma \rangle_{SR} \vdash \text{preR}(P) \]  
\text{by (rel-auto)}

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

\[ \text{periR}(\langle \sigma \rangle_{SR} \vdash P) = \langle \sigma \rangle_{SR} \vdash \text{periR}(P) \]  
\text{by (rel-auto)}

**Lemma postR-srd-subst [rdes]:**

\[ \text{postR}(\langle \sigma \rangle_{SR} \vdash P) = \langle \sigma \rangle_{SR} \vdash \text{postR}(P) \]  
\text{by (rel-auto)}

**Lemma srd-subst-NSRD-closed [closure]:**

**assumes** \( P \) is NSRD  
**shows** \( \langle \sigma \rangle_{SR} \vdash P \) is NSRD  
\text{by (rule NSRD-RC-intro, simp-all add: closure rdes assms unrest)}

**11.2 Assignment**

**Definition assigns-srd :: 's usubst ⇒ ('s, 't::trace, 'α) hrel-rsp (⟨'α⟩R) where**

\[ \text{assigns-srd} \sigma = R_s(\text{true} \vdash (\text{true} \vdash \text{false} \diamond \langle \sigma \rangle_{SR} \vdash \text{true} \vdash \text{true} \diamond \text{true} \vdash \text{true})) \]

**Syntax**

- assigns-srd :: usvids ⇒ uexprs ⇒ logic (infixr :=R 90)

**Translations**

- assigns-srd xs vs => CONST assigns-srd (-mk-usubst (CONST id) xs vs)
- assigns-srd x v <== CONST assigns-srd (CONST subst-upd (CONST id) x v)
- assigns-srd x v <= -assign-srd (-svar x) v

**Lemma assigns-srd-RHS-tri-des [rdes-def]:**

\[ \langle \sigma \rangle_{SR} = R_s(\text{true} \vdash \text{false} \diamond \langle \sigma \rangle_{SR}) \]  
\text{by (rel-auto)}

**Lemma assigns-srd-NSRD-closed [closure]:** \( \langle \sigma \rangle_{SR} \) is NSRD  
\text{by (simp add: rdes-def closure unrest)}

**Lemma preR-assigns-srd [rdes]:**\( \text{preR}(\langle \sigma \rangle_{SR}) = \text{true} \)
\text{by (simp add: rdes-def rdes closure)}
lemma periR-assigns-srd \(rdes\):
\[ peri_R(\langle \sigma \rangle_R) = \text{false} \]
by (simp add: rdes-def rdes closure)

lemma postR-assigns-srd \(rdes\):
\[ post_R(\langle \sigma \rangle_R) = \langle \sigma \rangle_R \]
by (simp add: rdes-def rdes closure rpred)

11.3 Conditional

lemma preR-cond-srea \(rdes\):
\[ pre_R(P \leadsto b \leq R Q) = (\langle b \rangle S < \land pre_R(P) \lor \langle \neg b \rangle S < \land pre_R(Q)) \]
by (rel-auto)

lemma periR-cond-srea \(rdes\):
assumes \(P\) is SRD \(Q\) is SRD
shows peri_R(P \leadsto b \leq R Q) = (\langle b \rangle S < \land peri_R(P) \lor \langle \neg b \rangle S < \land peri_R(Q))
proof –
\[ \text{have } peri_R(P \leadsto b \leq R Q) = peri_R(R1(P) \leadsto b \leq R1(Q)) \]
by (simp add: Healthy-if SRD-healths assms)
thus \(?thesis\)
by (rel-auto)
qed

lemma postR-cond-srea \(rdes\):
assumes \(P\) is SRD \(Q\) is SRD
shows post_R(P \leadsto b \leq R Q) = (\langle b \rangle S < \land post_R(P) \lor \langle \neg b \rangle S < \land post_R(Q))
proof –
\[ \text{have } post_R(P \leadsto b \leq R Q) = post_R(R1(P) \leadsto b \leq R1(Q)) \]
by (simp add: Healthy-if SRD-healths assms)
thus \(?thesis\)
by (rel-auto)
qed

lemma NSRD-cond-srea \(closure\):
assumes \(P\) is NSRD \(Q\) is NSRD
shows \(P \leadsto b \leq R Q\) is SRD
proof (rule NSRD-RC-intro)
\[ \text{show } P \leadsto b \leq R Q\text{ is SRD} \]
\[ \text{by (simp add: closure assms)} \]
\[ \text{show } pre_R(P \leadsto b \leq R Q)\text{ is RC} \]
proof –
\[ \text{have } 1:(\langle \neg b \rangle S < \lor \neg pre_R P) \leq R1(\text{true}) = (\langle \neg b \rangle S < \lor \neg pre_R P) \]
by (metis (no-types, lifting) NSRD-neg-pre-unit acxt-not assms(1) seqor-distl st-lift-R1-true-right)
\[ \text{have } 2:(\langle b \rangle S < \lor \neg pre_R Q) \leq R1(\text{true}) = (\langle b \rangle S < \lor \neg pre_R Q) \]
by (simp add: NSRD-neg-pre-unit assms seqor-distl st-lift-R1-true-right)
\[ \text{show } \?thesis \]
by (simp add: rdes closure assms)
qed
\[ \text{show } \$st' \# peri_R(P \leadsto b \leq R Q) \]
by (simp add: rdes closure unrest)
qed

11.4 Assumptions

definition AssumeR :: \('s cond \Rightarrow (\text{'s \textit{trace}}, 'a) hrel-rsp (\langle \cdot \rangle^T_R)\) where \[\text{upred-defs}]\text{ AssumeR } b = II_R \leq b \leq_R \text{ Miracle} \]
lemma AssumeR-rdes-def [rdes-def]:
\([b]^T_R = R_{\omega(true, \top)} \circ false \circ [b]^T_r]\)
unfolding AssumeR-def by (rdes-eq)

lemma AssumeR-NSRD [closure]: \([b]^T_R\) is NSRD
by (simp add: AssumeR-def closure)

lemma AssumeR-false: \([false]^T_R = Miracle\)
by (rel-auto)

lemma AssumeR-true: \([true]^T_R = II_R\)
by (rel-auto)

lemma AssumeR-comp: \([b]^T_R ;; [c]^T_R = [b \land c]^T_R\)
by (rdes-simp)

lemma AssumeR-choice: \([b]^T_R \sqcap [c]^T_R = [b \lor c]^T_R\)
by (rdes-eq)

lemma AssumeR-refine-skip: \(II_R \sqsubseteq [b]^T_R\)
by (rdes-refine)

lemma AssumeR-test [closure]: \(test_R [b]^T_R\)
by (simp add: AssumeR-refine-skip nsrd-thy.ustest-intro)

lemma Star-AssumeR: \([b]^T_R^*R = II_R\)
by (simp add: AssumeR-NSRD AssumeR-test nsrd-thy.Star-test)

lemma AssumeR-choice-skip: \(II_R \sqcap [b]^T_R = II_R\)
by (rdes-eq)

lemma cond-srea-AssumeR-form:
assumes P is NSRD Q is NSRD
shows \((P \triangleright b \triangleright_R Q = ([b]^T_R ;; P \sqcap \neg[b]^T_R ;; Q)\)
by (rdes-eq cls: assms)

lemma cond-srea-insert-assume:
assumes P is NSRD Q is NSRD
shows \((P \triangleright b \triangleright_R Q = ([b]^T_R ;; P \triangleright \neg[b]^T_R ;; Q)\)
by (simp add: AssumeR-NSRD AssumeR-comp NSRD-seqr-closure RA1 assms cond-srea-AssumeR-form)

lemma AssumeR-cond-left:
assumes P is NSRD Q is NSRD
shows \([b]^T_R ;; (P \triangleright b \triangleright_R Q) = ([b]^T_R ;; P)\)
by (rdes-eq cls: assms)

lemma AssumeR-cond-right:
assumes P is NSRD Q is NSRD
shows \(\neg[b]^T_R ;; (P \triangleright b \triangleright_R Q) = (\neg[b]^T_R ;; Q)\)
by (rdes-eq cls: assms)

11.5 Guarded commands

definition GuardedCommR :: 's cond ⇒ ('s, 't::trace, 'α) hrel-rsp ⇒ ('s, 't, 'α) hrel-rsp (- →R - [85, 86] 85) where
  gcmd-def[rdes-def]: GuardedCommR g A = A ◦ g R Miracle
lemma \text{gcmd-false [simp]}: \text{(false } \rightarrow_R \text{ A)} = \text{Miracle}
 unfolding \text{gcmd-def by (pred-auto)}

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

lemma \text{gcmd-SRD}: 
  assumes \text{A is SRD} 
  shows \text{(g } \rightarrow_R \text{ A)} = \text{Miracle}
 unfolding \text{gcmd-def by (pred-auto)}

lemma \text{gcmd-NSRD [closure]}: 
  assumes \text{A is NSRD} 
  shows \text{(g } \rightarrow_R \text{ A)} = \text{NSRD-Miracle}
 unfolding \text{gcmd-def by (pred-auto)}

lemma \text{gcmd-Productive [closure]}: 
  assumes \text{A is NSRD A is Productive} 
  shows \text{(g } \rightarrow_R \text{ A)} = \text{Productive}
 unfolding \text{gcmd-def closure assms}

lemma \text{gcmd-seq-distr}: 
  assumes \text{B is NSRD} 
  shows \text{(g } \rightarrow_R \text{ A)} = \text{B} = \text{NSRD-is-SRD assms cond-st-distr gcmd-def}
 unfolding \text{Miracle-left-zero NSRD-is-SRD assms cond-st-distr gcmd-def}

lemma \text{AssumeR-as-gcmd}: 
  \text{[b]}^+_R = \text{b } \rightarrow_R \text{ II}_R 
 unfolding \text{Miracle-left-zero NSRD-is-SRD assms cond-st-distr gcmd-def}

12 Generalised Alternation

definition \text{AlternateR} 
 :: \text{a set } \Rightarrow \text{(?a } \Rightarrow \text{'s upred) } \Rightarrow \text{(?a } \Rightarrow \text{('s, 't::trace, 'a) hrel-rsp) } \Rightarrow \text{('s, 't, 'a) hrel-rsp } \Rightarrow \text{('s, 't, 'a) hrel-rsp where}
 [upred-defs, ndes-simp]: \text{AlternateR I g A B} = \text{(\prod i \in I \cdot ((g i) } \rightarrow_R \text{ (A i)) } \cap \text{((\forall i \in I \cdot g i) } \rightarrow_R \text{ B)}

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

syntax 
 -altindR-els :: \text{pttrn } \Rightarrow \text{logic } \Rightarrow \text{logic } \Rightarrow \text{logic } \Rightarrow \text{logic } \Rightarrow \text{logic (if}_R \text{-el} \cdot \text{- } \Rightarrow \text{- else - fi)}
 -altindR :: \text{pttrn } \Rightarrow \text{logic } \Rightarrow \text{logic } \Rightarrow \text{logic } \Rightarrow \text{logic (if}_R \text{-el} \cdot \text{- } \Rightarrow \text{- fi)}

-altgcommR-els :: \text{gcomms } \Rightarrow \text{logic } \Rightarrow \text{logic (if}_R \text{- else - fi)}
 -altgcommR :: \text{gcomms } \Rightarrow \text{logic (if}_R \text{- fi)}
translations
if \( R_i \in I \cdot g \rightarrow A \) else \( B \) fi \( \rightarrow \text{CONST AlternateR} \ (\lambda g. (\lambda i. A) B) \)
if \( R_i \in I \cdot g \rightarrow A \) fi \( \rightarrow \text{CONST AlternateR} \ (\lambda g. (\lambda i. A) \text{ (CONST Chaos)}) \)
if \( R_i \in I \cdot g(i) \rightarrow A \) else \( B \) fi \( \leftarrow \text{CONST AlternateR} \ i g (\lambda i. A) B \)
\text{-altgcommR} cs \( \rightarrow \text{CONST AlternateR-list} \ cs \ (\text{CONST Chaos}) \)
\text{-altgcommR} (-gcomm-show cs) \( \leftarrow \text{CONST AlternateR-list} \ cs \ (\text{CONST Chaos}) \)
\text{-altgcommR-els} cs \( P \rightarrow \text{CONST AlternateR-list} \ cs \ P \)
\text{-altgcommR-els} (-gcomm-show cs) \( P \leftarrow \text{CONST AlternateR-list} \ cs \ P \)

lemma AlternateR-NSRD-closed [closure]:
assumes \( \bigwedge i. i \in I \Rightarrow A \) is NSRD \( B \) is NSRD
shows \( (\text{if} \ R_i \in I \cdot g i \rightarrow A \ \text{else} \ B \) fi) is NSRD
proof (cases \( I = \{\} \))
case True
then show \( \text{thesis} \) by (simp add: AlternateR-def assms)
next
case False
then show \( \text{thesis} \) by (simp add: AlternateR-def closure assms)
qed

lemma AlternateR-empty [simp]:
(\( \text{if} \ R_i \in \{\} \cdot g i \rightarrow A \) i else \( B \) fi) = \( B \)
by (rdes-simp)

lemma AlternateR-Productive [closure]:
assumes \( \bigwedge i. i \in I \Rightarrow A \) i is Productive \( B \) is Productive
shows \( (\text{if} \ R_i \in I \cdot g i \rightarrow A \ i \text{ else} \ B \) fi) is Productive
proof (cases \( I = \{\} \))
case True
then show \( \text{thesis} \) by (simp add: assms(4))
next
case False
then show \( \text{thesis} \) by (simp add: AlternateR-def closure assms)
qed

lemma AlternateR-singleton:
assumes \( A k \) is NSRD \( B \) is NSRD
shows \( (\text{if} \ R_i \in \{k\} \cdot g i \rightarrow A \ i \text{ else} \ B \) fi) = \( (A(k) \circ g(k) \triangleright_R \ B) \)
by (simp add: AlternateR-def, rdes-eq cls: assms)

Convert an alternation over disjoint guards into a cascading if-then-else

lemma AlternateR-insert-cascade:
assumes \( \bigwedge i. i \in I \Rightarrow A \) i is NSRD
\( A k \) is NSRD \( B \) is NSRD
\( (g(k) \land (\bigvee i \in I \cdot g(i))) = \text{false} \)
shows \( (\text{if} \ R_i \in \text{insert } k I \cdot g i \rightarrow A \ i \text{ else} \ B \) fi) = \( (A(k) \circ g(k) \triangleright_R (\text{if} \ R_i \in I \cdot g(i) \rightarrow A(i) \text{ else} \ B \) fi)) \)
proof (cases \( I = \{\} \))
case True
then show \( \text{thesis} \) by (simp add: AlternateR-singleton assms)
qed
next
  case False
  have 1: (⨅ i ∈ I · g i →_R A i) = (⨅ i ∈ I · g i →_R R_s(pre_R(A i) ⊢ peri_R(A i) ⋄ post_R(A i)))
    by (simp add: NSRD-is-SRD SRD-reactive-tri-design assms(1) cong: UINF-cong)
  from assays(4) show ?thesis
    by (simp add: AlternateR-def 1 False cong: UINF-cong)
qed

12.1 Choose

definition choose-srd :: (′s,′t::trace,′α) hrel-rsp (choose_R) where
  [upred-defs, rdes-def]: choose_R = R_s(true_r ⊢ false ⋄ true_r)

lemma preR-choose [rdes]: pre_R(choose_R) = true_r
  by (rel-auto)

lemma periR-choose [rdes]: peri_R(choose_R) = false
  by (rel-auto)

lemma postR-choose [rdes]: post_R(choose_R) = true_r
  by (rel-auto)

lemma choose-srd-SRD [closure]: choose_R is SRD
  by (simp add: choose-srd-def closure unrest)

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

12.2 State Abstraction

definition state-srea :: ∃ s itself ⇒ (′s,′t::trace,′α,′β) rel-rsp ⇒ (unit,′t,′α,′β) rel-rsp where
  [upred-defs]: state-srea t P = (∃ {$st,$st'} · P)\s

syntax
  -state-srea :: type ⇒ logic ⇒ logic (state · · · [0,200] 200)

translations
  state 'a · P == CONST state-srea TYPE('a) P

lemma R1-state-srea: R1(state 'a · P) = (state 'a · R1(P))
  by (rel-auto)

lemma R2c-state-srea: R2c(state 'a · P) = (state 'a · R2c(P))
  by (rel-auto)

lemma R3h-state-srea: R3h(state 'a · P) = (state 'a · R3h(P))
  by (rel-auto)

lemma RD1-state-srea: RD1(state 'a · P) = (state 'a · RD1(P))
  by (rel-auto)

lemma RD2-state-srea: RD2(state 'a · P) = (state 'a · RD2(P))
  by (rel-auto)
lemma **RD3-state-srea**: \( RD3(\text{state } 'a \cdot P) = (\text{state } 'a \cdot RD3(P)) \)
   by (rel-auto, blast+)

lemma **SRD-state-srea [closure]**: \( P \text{ is SRD} \implies \text{state } 'a \cdot P \text{ is SRD} \)
   by (simp add: Healthy-def R1-state-srea R2c-state-srea R3h-state-srea RD1-state-srea RD2-state-srea RHS-def SRD-def)

lemma **NSRD-state-srea [closure]**: \( P \text{ is NSRD} \implies \text{state } 'a \cdot P \text{ is NSRD} \)
   by (metis Healthy-def NSRD-is-RD3 NSRD-is-SRD RD3-state-srea SRD-RD3-implies-NSRD SRD-state-srea)

lemma **preR-state-srea [rdes]**: \( \text{pre}_R(\text{state } 'a \cdot P) = (\\forall \{\text{st}, \$\text{st}\} \cdot \text{pre}_R(P))_S \)
   by (simp add: state-srea-def, rel-auto)

lemma **periR-state-srea [rdes]**: \( \text{peri}_R(\text{state } 'a \cdot P) = \text{state } 'a \cdot \text{peri}_R(P) \)
   by (rel-auto)

lemma **postR-state-srea [rdes]**: \( \text{post}_R(\text{state } 'a \cdot P) = \text{state } 'a \cdot \text{post}_R(P) \)
   by (rel-auto)

### 12.3 While Loop

definition **WhileR**: 's upred \( \Rightarrow \) ('s, 't::size-trace, 'a) hrel-rsp \( \Rightarrow \) ('s, 't, 'a) hrel-rsp (while_R - do - od)
where
WhileR \( b \cdot P = (\mu_R X \cdot (P ;; X) \triangleleft b \triangleright_R H_R) \)

lemma **Sup-power-false**: 
   fixes \( F :: 'a \cdot \text{upred} \Rightarrow 'a \cdot \text{upred} \)
   shows \((\prod i. (F \cdot' i) \cdot \text{false}) = (\prod i. (F \cdot' (i+1)) \cdot \text{false})\)
 proof --
   have \((\prod i. (F \cdot' i) \cdot \text{false}) = (F \cdot' 0) \cdot \text{false} \cap (\prod i. (F \cdot' (i+1)) \cdot \text{false})\)
      also have \((\prod i. (F \cdot' (i+1)) \cdot \text{false})\)
         by (simp)
   finally show \(?thesis\).
 qed

definition **WhileR-iter-expand**: 
   assumes \( P \text{ is NSRD } \cdot \text{P is Productive} \)
   shows \( \text{while}_R \cdot b \text{ do } \cdot \text{P od} = (\prod i. (P \triangleleft b \triangleright_R H_R) \cdot i ;; (P ;; \text{Miracle} \triangleleft b \triangleright_R H_R)) \cdot (\text{is } \text{?lhs} = \text{?rhs})\)
 proof --
   have 1:Continuous \((\lambda X. P ;; \text{SRD } X)\)
      using SRD-Continuous
      by (clarsimp simp add: Continuous-def seq-SUP-distl[THEN sym], drule-tac x=A in spec, simp)
  have 2:Continuous \((\lambda X. P ;; \text{SRD } X \triangleleft b \triangleright_R H_R)\)
      by (simp add: 1 closure assms)
  have \(?lhs = (\mu_R X \cdot P ;; X \triangleleft b \triangleright_R H_R)\)
      by (simp add: WhileR-def)
  also have \(?rhs = (\mu X \cdot P ;; \text{SRD}(X) \triangleleft b \triangleright_R H_R)\)
      by (auto simp add: srdomequiv closure assms)
  also have \(?rhs = (\mu X \cdot P ;; \text{SRD}(X) \triangleleft b \triangleright_R H_R)\)
      by (auto simp add: guarded-fp-univ Guarded-if-Productive[OF assms] funcsetI closure assms)
  also have \((\lambda X. P ;; \text{SRD } X \triangleleft b \triangleright_R H_R)^{\cdot' i} \cdot \text{false}\)
      by (simp add: sup-continuous-lfp 2 sup-continuous-Continuous false-upred-def)
  also have \((\lambda X. P ;; \text{SRD } X \triangleleft b \triangleright_R H_R)^{\cdot' (i+1)} \cdot \text{false}\)
      by (simp add: Sup-power-false)
  also have \((\prod i. (P \triangleleft b \triangleright_R H_R)\cdot i ;; (P ;; \text{Miracle} \triangleleft b \triangleright_R H_R))\)

97
proof (rule SUP-cong, simp)
  fix i
  show ((λX. P ;; SRD X < b ∨R II_R) `" (Suc i + 1)) false = (P < b ∨R II_R) ^ i ;; (P ;; Miracle < b ∨R II_R)
  proof (induct i)
    case 0
    then show ?thesis by (simp, metis srdes-cond-srea srdes-theory-continuous.healthy-top)
  next
    case (Suc i)
    have ((λX. P ;; SRD X < b ∨R II_R) `" (Suc i + 1)) false =
      P ;; SRD ((λX. P ;; SRD X < b ∨R II_R) `" (i + 1)) false < b ∨R II_R
    proof
      by simp
      also have ... = P ;; SRD ((P < b ∨R II_R) ^ i ;; (P ;; Miracle < b ∨R II_R)) < b ∨R II_R
      using Suc.hyps by auto
      also have ... = P ;; ((P < b ∨R II_R) ^ i ;; (P ;; Miracle < b ∨R II_R)) < b ∨R II_R
    proof (induct i)
      case 0
      then show ?thesis by (simp add: NSRD-is-SRD SRD-cond-srea SRD-left-unit SRD-seqr-closure SRD-srdes-skip
      assms(1) cond-L6 cond-st-distr srdes-theory-continuous.top-closed)
    next
      case (Suc i)
      have 1: II_R ;; ((P < b ∨R II_R) ;; (P < b ∨R II_R) ^ i) = ((P < b ∨R II_R) ;; (P < b ∨R II_R) ^ i)
      proof
        by (simp add: NSRD-is-SRD RA1 SRD-cond-srea SRD-left-unit SRD-srdes-skip assms(1))
      then show ?thesis
        by (simp add: RA1 upred-semiring.power-Suc)
    qed
    qed
    finally show ?thesis .
  qed
  qed
  qed
  qed
  also have ... = (Π i · (P < b ∨R II_R) ^ i ;; (P ;; Miracle < b ∨R II_R))
  by (simp add: UNF-as-Sup-collect')
  finally show ?thesis .
  qed

theorem WhileR-star-expand:
  assumes P is NSRD P is Productive
  shows WHILE b do P od = (P < b ∨R II_R) ^ b do P od ;; (P ;; Miracle < b ∨R II_R) (is ?lhs = ?rhs)
proof
  have ?lhs = (Π i · (P < b ∨R II_R) ^ i) ;; (P ;; Miracle < b ∨R II_R)
  by (simp add: WhileR-iter-expand seq-UNF-distr' assms)
also have \( \vdash (P \triangleleft b \triangleright R II_R)^* \); \( \vdash (P \vdash b \triangleright R II_R) \)
by (simp add: ustar-def)
also have \( \vdash (P \triangleleft b \triangleright R II_R)^* \); \( \vdash (P \vdash b \triangleright R II_R) \)
by (simp add: seqr-assoc SRD-left-unit closure assms)
also have \( \vdash (P \triangleleft b \triangleright R II_R)^* \); \( \vdash (P \vdash b \triangleright R II_R) \)
by (simp add: nsrd-thy.Star-def)
finally show \( \vdash \)thesis.

qed

lemma WhileR-NSRD-closed \[\text{[\textit{closure}]}\]:
assumes \( P \) is NSRD \( P \) is Productive
shows \( \text{while}_R b \) do \( P \) od is NSRD
by (simp add: WhileR-star-expand assms closure)

theorem WhileR-iter-form-lemma:
assumes \( P \) is NSRD
shows \( \vdash (P \triangleleft b \triangleright R II_R)^* \); \( \vdash (P \vdash b \triangleright R II_R) \)
by (simp add: cond-srea-AssumeR-form)
also have \( \vdash (P \triangleleft b \triangleright R II_R)^* \); \( \vdash (P \vdash b \triangleright R II_R) \)
by (simp add: AssumeR-NSRD NSRD-right-unit NSRD-srd-skip assms)
also have \( \vdash (P \triangleleft b \triangleright R II_R)^* \); \( \vdash (P \vdash b \triangleright R II_R) \)
by (simp add: AssumeR-NSRD NSRD-seqr-closure nsrd-thy.Star-denest assms)
also have \( \vdash (P \triangleleft b \triangleright R II_R)^* \); \( \vdash (P \vdash b \triangleright R II_R) \)
by (simp add: AssumeR-NSRD NSRD-Miracle NSRD-right-unit NSRD-seqr-closure NSRD-srd-skip assms)
also have \( \vdash (P \triangleleft b \triangleright R II_R)^* \); \( \vdash (P \vdash b \triangleright R II_R) \)
by (simp add: upred-semiring.distrib-left)
also have \( \vdash (P \triangleleft b \triangleright R II_R)^* \); \( \vdash (P \vdash b \triangleright R II_R) \)
by (simp add: upred-semiring.distrib-right)
also have \( \vdash (P \triangleleft b \triangleright R II_R)^* \); \( \vdash (P \vdash b \triangleright R II_R) \)
by (simp add: RA1)
also have \( \vdash (P \triangleleft b \triangleright R II_R)^* \); \( \vdash (P \vdash b \triangleright R II_R) \)
by (simp add: AssumeR-comp AssumeR-false)
finally have \( \vdash (P \triangleleft b \triangleright R II_R)^* \); \( \vdash (P \vdash b \triangleright R II_R) \)
by (simp add: semilattice-sup-class.le-sup II1)
thus \( \vdash \)thesis.
by (simp add: semilattice-sup-class.le-iff-sup)
theorem WhileR-true:
  assumes P is NSRD P is Productive
  shows while_R b do P od = ([b]^T R ;; P)^* R ;; [¬ b]^T R
  by (simp add: WhileR-iter-form-lemma WhileR-star-expand assms)

theorem WhileR-false:
  assumes P is NSRD
  shows while_R false do P od = H_R
  by (simp add: WhileR-def rpred closure srdes-theory-continuous LFP-const)

theorem WhileR-true:
  assumes P is NSRD P is Productive
  shows while_R true do P od = P^* R ;; Miracle
  by (simp add: WhileR-iter-form AssumeR-true AssumeR-false SRD-left-unit assms closure)

lemma WhileR-insert-assume:
  assumes P is NSRD P is Productive
  shows while_R b do (([b]^T R ;; P) od = while_R b do P od
  by (simp add: AssumeR-NSRD AssumeR-comp NSRD-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 $st$ Q R is R4
  shows while_R b do R_s (P ⊢ Q ◁ R) od =
  (is ?lhs = ?rhs)
  proof
    have ?lhs = ([b]^T R ;; R_s (P ⊢ Q ◁ R))^* R ;; [¬ b]^T R
      by (simp add: WhileR-iter-form Productive-rdes-RR-intro assms closure)
    also have ... = ?rhs
      by (simp add: rdes-def assms closure unrest rpred wp del: rea-star-wp)
  finally show ?thesis .
  qed

Refinement introduction law for reactive while loops

theorem WhileR-refine-intro:
  assumes
  — Closure conditions
  Q_1 is RC Q_2 is RR Q_3 is RR $st$ Q_2 Q_3 is R4
  — Refinement conditions
  ([[b]^r R ;; Q_3]^* R wp_r ([b]r S ◁ ⇒ (Q_1) ⊆ P_1)
  P_2 ⊆ [[b]^r R ;; Q_2
  P_2 ⊆ [[b]^r R ;; Q_3 ;; P_2
  P_3 ⊆ [¬ b]^r R
  P_3 ⊆ [[b]^r R ;; Q_3 ;; P_3
  shows R_s (P_1 ⊢ P_2 ◁ P_3) ⊆ while_R b do R_s (Q_1 ⊢ Q_2 ◁ Q_3) od
  proof (simp add: rdes-def assms, rule srdes-tri-refine-intro')
    show ([[b]^r R ;; Q_3]^* R wp_r ([b]r S ◁ ⇒ (Q_1) ⊆ P_1
      by (simp add: assms)
    show P_2 ⊆ (P_1 ∧ ([[b]^r R ;; Q_3]^* R ;; [[b]^r R ;; Q_2)
      proof —

have \( P_2 \subseteq ([b]^T \ddot{r} \dddot{r} ; Q_3)^{**} \dddot{r} \dddot{r} ; [b]^T \ddot{r} ; Q_2 \)
by (simp add: asms rea-assume-RR rrel-thy. Star-inductl seq-RR-closed seqr-assoc)
thus \( \text{thesis} \)
by (simp add: utp-pred-laws.le-infl2)
qed

\[ \text{lemma IterateR-empty:} \]
\[
\begin{array}{l}
\text{do}_R i \in \{ \} \cdot g(i) \rightarrow P(i) \ \text{fi} = \text{II}_R \\
\text{by (simp add: IterateR-def srd-mu-equiv closure rpred gfp-const WhileR-false)}
\end{array}
\]

\[ \text{lemma IterateR-singleton:} \]
\[
\begin{array}{l}
\text{assumes} \\
\wedge i. \ i \in I \implies P(i) \text{ is NSRD} \\
\text{shows} \ \text{do}_R i \in \{ \} \cdot g(i) \rightarrow P(i) \ \text{fi} = \text{while}_R g(k) \text{ do } P(k) \text{ od (is \ ?lhs = \ ?rhs)} \\
\text{proof} – \\
\text{have \ ?lhs = WhileR g k do P k < g k \triangleright_R Chaos od} \\
\text{by (simp add: IterateR-def AlternateR-singleton asms closure)} \\
\text{also have \ ... = WhileR g k do \[ g k \]^T \ddot{r} \dddot{r} \dddot{r} ; (P k < g k \triangleright_R Chaos) \text{ od}} \\
\text{by (simp add: WhileR-insert-assume closure asms)} \\
\text{also have \ ... = WhileR g k do P k od} \\
\text{by (simp add: AssumeR-cond-left NSRD-Chaos WhileR-insert-assume asms)} \\
\text{finally show \ ?thesis .}
\end{array}
\]
qed

12.4 Iteration Construction

\[ \text{definition \ IterateR} \]
\[
\begin{array}{l}
\dddot{r} \dddot{r} \dddot{r} A g P = \text{while}_R (\bigvee i \in A \cdot g(i)) \text{ do } (\text{if}_R i \in A \cdot g(i) \rightarrow P(i) \ \text{fi} ) \text{ od}
\end{array}
\]

\[ \text{translations} \]
\[
\begin{array}{l}
- \text{-iter-srd : ptt \implies \text{logic} \implies \text{logic} \implies \text{logic} \text{ (do}_R - \dddot{r} - - \rightarrow - \ \text{fi})}
\end{array}
\]

\[ \text{lemma \ IterateR-NSRD-closed [closure]:} \]
\[
\begin{array}{l}
\text{assumes} \\
\wedge i. \ i \in I \implies P(i) \text{ is NSRD} \\
\text{shows} \ \text{do}_R i \in \{ \} \cdot g(i) \rightarrow P(i) \ \text{fi} \text{ is NSRD} \\
\text{by (simp add: IterateR-def closure asms)}
\end{array}
\]

\[ \text{lemma \ IterateR-empty:} \]
\[
\begin{array}{l}
\text{do}_R i \in \{ \} \cdot g(i) \rightarrow P(i) \ \text{fi} = \text{II}_R \\
\text{by (simp add: IterateR-def srd-mu-equiv closure rpred gfp-const WhileR-false)}
\end{array}
\]

\[ \text{lemma \ IterateR-singleton:} \]
\[
\begin{array}{l}
\text{assumes} \\
\text{shows} \ \text{do}_R i \in \{ \} \cdot g(i) \rightarrow P(i) \ \text{fi} = \text{while}_R g(k) \text{ do } P(k) \text{ od (is \ ?lhs = \ ?rhs)} \\
\text{proof} – \\
\text{have \ ?lhs = WhileR g k do P k < g k \triangleright_R Chaos od} \\
\text{by (simp add: IterateR-def AlternateR-singleton asms closure)} \\
\text{also have \ ... = WhileR g k do \[ g k \]^T \ddot{r} \dddot{r} \dddot{r} ; (P k < g k \triangleright_R Chaos) \text{ od}} \\
\text{by (simp add: WhileR-insert-assume closure asms)} \\
\text{also have \ ... = WhileR g k do P k od} \\
\text{by (simp add: AssumeR-cond-left NSRD-Chaos WhileR-insert-assume asms)} \\
\text{finally show \ ?thesis .}
\end{array}
\]

12.5 Substitution Laws

\[ \text{lemma \ srd-subst-Chaos \ [asubst]:} \]
\[
\begin{array}{l}
\sigma \uparrow_S \text{Chaos} = \text{Chaos} \\
\text{by (rdes-simp)}
\end{array}
\]
lemma srd-subst-Miracle [usubst]:
\[ \sigma \uparrow_S \text{Miracle} = \text{Miracle} \]
by (rdes-simp)

lemma srd-subst-skip [usubst]:
\[ \sigma \uparrow_S \text{ II} = \langle \sigma \rangle_R \]
by (rdes-eq)

lemma srd-subst-assigns [usubst]:
\[ \sigma \uparrow_S \langle \dot{\nu} \rangle_R = \langle \dot{\nu} \circ \sigma \rangle_R \]
by (rdes-eq)

12.6 Algebraic Laws

theorem assigns-srd-id: \( \langle \text{id} \rangle_R = \text{II} \)
by (rdes-eq)

theorem assigns-srd-comp: \( \langle \sigma \rangle_R ;; \langle \dot{\nu} \rangle_R = \langle \dot{\nu} \circ \sigma \rangle_R \)
by (rdes-eq)

theorem assigns-srd-Miracle: \( \langle \sigma \rangle_R ;; \text{Miracle} = \text{Miracle} \)
by (rdes-eq)

theorem assigns-srd-Chaos: \( \langle \sigma \rangle_R ;; \text{Chaos} = \text{Chaos} \)
by (rdes-eq)

theorem assigns-srd-cond : \( \langle \sigma \rangle_R \triangleleft b \triangleright \langle \dot{\nu} \rangle_R = \langle \sigma \triangleleft b \triangleright s \dot{\nu} \rangle_R \)
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 \( \forall i. \ A \ i \text{ is NSRD} \ B \text{ is NSRD} \ C \text{ is NSRD} \)
shows \( \langle \text{if} R \ i \in I \cdot g \ i \to A \ i \text{ else} B \ f i \rangle ;; C = \langle \text{if} R \ i \in I \cdot g \ i \to A \ i ;; C \text{ else} B ;; C \ f i \rangle \)
proof (cases I = \{}
next
\text{case} True
then show \( ?\text{thesis} \) by (simp)
next
\text{case} False
then show \( ?\text{thesis} \)
by (simp add: AlternateR-def upred-semiring.distrib-right seq-UNInf-distr gcmd-seq-distr assms(3))
qed

lemma AlternateR-is-cond-srea:
assumes A is NSRD B is NSRD
shows \( \langle \text{if} R \ i \in \{a\} \cdot g \ i \to A \text{ else} B \ f i \rangle = (A \triangleleft g \triangleright R \ B) \)
by (rdes-eq cls: assms)

lemma AlternateR-Chaos:
if \( R \ i \in A \cdot g(i) \to \text{Chaos} \ f i = \text{Chaos} \)
by (cases A = \{}, simp, rdes-eq)
lemma choose-srd-par:
  choose_R ‖_R choose_R = choose_R
by (rdes-eq)

12.7 Lifting designs to reactive designs

definition des-rea-lift :: _hrel-des ⇒ _hrel-rsp (R_D) where
[upred-defs]: R_D(P) = R_u([pre_D(P)]_S ‒ (false ° ($tr' =_u $tr ∧ [post_D(P)]_S)))

definition des-rea-drop :: _hrel-rsp ⇒ _hrel-des (D_R) where
[upred-defs]: D_R(P) = [([pre_R(P)][$str/$str'] [fst $st]_S ‒ 
  ‒ ′_n ([post_R(P)][$tr/$tr'] [fst $st,$st']_S)

lemma ndesign-rea-lift-inverse: D_R(R_D(p ‒ n Q)) = p ‒ n Q
apply (simp add: des-rea-lift-def des-rea-drop-def rea-pre-RHS-design rea-post-RHS-design)
apply (simp add: R1-def R2c-def R2s-def usubst unrest)
apply (rel-auto)
done

lemma ndesign-rea-lift-injective:
  assumes P is N Q is N R_D P = R_D Q (is ?RP(P) = ?RQ(Q))
shows P = Q
proof –
  have ?RP([pre_D(P)] ‒ n post_D(P)) = ?RQ([pre_D(Q)] ‒ n post_D(Q))
    by (simp add: ndesign-form assms)
  hence [pre_D(P)] ‒ n post_D(P) = [pre_D(Q)] ‒ n post_D(Q)
    by (metis ndesign-rea-lift-inverse)
  thus ?thesis
    by (simp add: ndesign-form assms)
qed

lemma des-rea-lift-closure [closure]: R_D(P) is SRD
by (simp add: des-rea-lift-def RHS-design-is-SRD unrest)

lemma preR-des-rea-lift [rdes]:
  pre_R(R_D(P)) = R1([pre_D(P)]_S)
by (rel-auto)

lemma periR-des-rea-lift [rdes]:
  peri_R(R_D(P)) = (false ° [pre_D(P)]_S ‒ ($tr ≤_u $tr'))
by (rel-auto)

lemma postR-des-rea-lift [rdes]:
  post_R(R_D(P)) = (true ° [pre_D(P)]_S ‒ ($tr ≤_u $tr')) ⇒ ($tr' =_u $tr ∧ [post_D(P)]_S)
apply (rel-auto) using minus-zero-eq by blast

lemma ndes-rea-lift-closure [closure]:
  assumes P is N
shows R_D(P) is NSRD
proof –
  obtain p Q where P: P = (p ‒ n Q)
    by (metis H1-H3-commute H1-H3-is-normal-design H1-idem Healthy-def assms)
show ?thesis
  apply (rule NSRD-intro)
    apply (simp-all add: closure rdes unrest P)
  apply (rel-auto)
lemma R-D-mono:
assumes P is H Q is H P \sqsubseteq Q
shows R \mathcal{D}(P) \sqsubseteq R \mathcal{D}(Q)
apply (simp add: des-rea-lift-def)
apply (rule srdes-tri-refine-intro')
apply (auto intro: H1-H2-refines assms aext-mono)
apply (rel-auto)
apply (metis (no-types, hide-lams) aext-mono assms (3) design-post-choice semilattice-sup-class.sup.orderE utp-pred-laws.inf.coboundedI1 utp-pred-laws.inf.commute utp-pred-laws.sup.order-iff)
done

Homomorphism laws

lemma R-D-Miracle:
R \mathcal{D}(\top \mathcal{D}) = Miracle
by (simp add: Miracle-def, rel-auto)

lemma R-D-Chaos:
R \mathcal{D}(\bot \mathcal{D}) = Chaos
proof –
  have R \mathcal{D}(\bot \mathcal{D}) = R \mathcal{D}(false \vdash true)
    by (rel-auto)
  also have ... = R_a (false \vdash false \circ (\$tr' =_u \$tr))
    by (simp add: Chaos-def des-rea-lift-def alpha)
  also have ... = Chaos
    by (simp add: Chaos-def design-false-pre)
  finally show \?thesis .
qed

lemma R-D-inf:
R \mathcal{D}(P \sqcap Q) = R \mathcal{D}(P) \sqcap R \mathcal{D}(Q)
by (rule antisym, rel-auto+)

lemma R-D-cond:
R \mathcal{D}(P \sqsubseteq [b]_{D_< \triangleright} Q) = R \mathcal{D}(P) \sqsubseteq b \triangleright_R R \mathcal{D}(Q)
by (rule antisym, rel-auto+)

lemma R-D-seq-ndesign:
R \mathcal{D}(p_1 \vdash_n Q_1) ;; R \mathcal{D}(p_2 \vdash_n Q_2) = R \mathcal{D}((p_1 \vdash_n Q_1) ;; (p_2 \vdash_n Q_2))
apply (rule antisym)
apply (rule SRD-refine-intro)
apply (simp-all add: closure ndesign-composition-wp)
using dual-order.trans apply (rel-blast)
using dual-order.trans apply (rel-blast)
apply (rel-auto)
apply (rule SRD-refine-intro)
apply (simp-all add: closure ndesign-composition-wp)
apply (rel-auto)
apply (rel-auto)
apply (rel-auto)
done
lemma $R$-$D$-seq:
  assumes $P$ is $N$ $Q$ is $N$
  shows $R_D(P) :: R_D(Q) = R_D(P :: Q)$
  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:
  $R_D(\bot_D) = (\bot_R :: (\',t::\text{trace},\text{unit}) \text{ hrel-rsp})$
  apply (rel-auto) using minus-zero-eq by blast+

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

end

13 Instantaneous Reactive Designs

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

definition $ISRD1 :: (\',t::\text{trace},\alpha) \text{ hrel-rsp} \Rightarrow (\',t,\alpha) \text{ hrel-rsp}$ where
[upred-defs]: $ISRD1(P) = P \parallel R_a(\text{true}_r \vdash false \circ (\$tr' = \alpha \$tr))$

definition $ISRD :: (\',t::\text{trace},\alpha) \text{ hrel-rsp} \Rightarrow (\',t,\alpha) \text{ hrel-rsp}$ where
[uppred-defs]: $ISRD = ISRD1 \circ NSRD$

lemma $ISRD1$-idem: $ISRD1(ISRD1(P)) = ISRD1(P)$
  by (rel-auto)

lemma $ISRD1$-monotonic: $P \sqsubseteq Q \rightarrow ISRD1(P) \sqsubseteq ISRD1(Q)$
  by (rel-auto)

lemma $ISRD1$-RHS-design-form:
  assumes $\$ok' \notin P \$ok' \notin Q \$ok' \notin R$
  shows $ISRD1(R_a(P \parallel Q \circ R)) = R_a(P \vdash false \circ (R \land \$tr' = \alpha \$tr))$
  using assms by (simp add: $ISRD1$-def choose-srd-def RHS-tri-design-par unrest, rel-auto)

lemma $ISRD1$-form:
  $ISRD1(SRD(P)) = R_a(\text{pre}_R(P) \vdash false \circ (\text{post}_R(P) \land \$tr' = \alpha \$tr))$
  by (simp add: $ISRD1$-RHS-design-form $SRD$-as-reactive-tri-design unrest)

lemma $ISRD1$-rdes-def [rdes-def]:
  \[ P \text{ is RR; } R \text{ is RR } \] \rightarrow $ISRD1(R_a(P \parallel Q \circ R)) = R_a(P \vdash false \circ (R \land \$tr' = \alpha \$tr))$
  by (simp add: $ISRD1$-def rdes-def closure rpred)

lemma $ISRD$-intro:
  assumes $P$ is $NSRD$ peri$_R(P) = (\neg_\alpha \text{pre}_R(P)) (\$tr' = \alpha \$tr) \sqsubseteq \text{post}_R(P)$
  shows $P$ is $ISRD$

proof –
  have $R_a(\text{pre}_R(P) \parallel \neg_\alpha \text{pre}_R(P) \circ \text{post}_R(P))$ is $ISRD1$
  apply (simp add: Healthy-def rdes-def closure assms(1-2))
  using assms(3) least-zero apply (rel-blast)
done

hence $P$ is ISRD1
  by (simp add: SRD-reactive-tri-design closure assms(1))
thus ?thesis
  by (simp add: ISRD-def Healthy-comp assms(1))

qed

lemma ISRD1-rdes-intro:
  assumes $P$ is RR $Q$ is RR ($\text{tr'} =_u \text{tr}$) $\subseteq Q$
  shows $R_s(P \vdash \text{false} \circ Q)$ is ISRD1
unfolding Healthy-def
by (simp add: ISRD1-rdes-def assms closure unrest utp-pred-laws.inf.absorb1)

lemma ISRD-rdes-intro [closure]:
  assumes $P$ is RC $Q$ is RR ($\text{tr'} =_u \text{tr}$) $\subseteq Q$
  shows $R_s(P \vdash \text{false} \circ Q)$ is ISRD
unfolding Healthy-def
by (simp add: ISRD-def closure Healthy-if ISRD1-rdes-def assms unrest utp-pred-laws.inf.absorb1)

lemma ISRD-implies-ISRD1:
  assumes $P$ is ISRD
  shows $P$ is ISRD1
proof -
  have ISRD($P$) is ISRD1
    by (simp add: ISRD-def Healthy-def ISRD1-idem)
  thus ?thesis
    by (simp add: assms Healthy-if)

qed

lemma ISRD-implies-SRD:
  assumes $P$ is ISRD
  shows $P$ is SRD
proof -
  have $1$:ISRD($P$) = $R_s((\neg r \neg\text{pre}_R P) :: R1 true \land R1 true) \vdash \text{false} \circ (post_R P \land \text{tr'} =_u \text{tr})$
    by (simp add: NSRD-form ISRD1-def ISRD-def RHS-tri-design-par rdes-def unrest closure)
  moreover have ... is SRD
    by (simp add: closure unrest)
  ultimately have ISRD($P$) is SRD
    by (simp)
  with assms show ?thesis
    by (simp: Healthy-def)

qed

lemma ISRD-implies-NSRD [closure]:
  assumes $P$ is ISRD
  shows $P$ is NSRD
proof -
  have $1$:ISRD($P$) = ISRD1($RD3(SRD(P))$)
    by (simp add: ISRD-def NSRD-def SRD-def, metis RD1-RD3-commute RD3-left-subsumes-RD2)
  also have ... = ISRD1($RD3(P)$)
    by (simp add: assms ISRD-implies-SRD Healthy-if)
  also have ... = ISRD1 ($R_s((\neg r \neg\text{pre}_R P) \wp \text{false}_R \vdash (\exists \text{st'} \cdot \text{peri}_R P) \circ post_R P)$)
    by (simp add: RD3-def subst SRD-right-unit-tri-lemma, simp-all add: assms ISRD-implies-SRD)
  also have ... = $R_s((\neg r \neg\text{pre}_R P) \wp \text{false}_R \vdash \text{false} \circ (post_R P \land \text{tr'} =_u \text{tr})$)
    by (simp add: RHS-tri-design-par ISRD1-def unrest choose-srd-def rpred closure ISRD-implies-SRD)

106
lemma \textit{ISRD-form}:
\begin{align*}
\text{assumes } P \text{ is ISRD} \\
\text{shows } R_s(\text{pre}_R(P) \vdash \false \odot (\text{post}_R(P) \land $tr^\prime = u \ $tr)) = P
\end{align*}
\begin{proof}
\begin{enumerate}
\item have \( P = \text{ISRD1}(P) \)
\begin{enumerate}
\item by \( \text{simp add: ISRD-implies-ISRD1 \ \text{assms \ Healthy-if}} \)
\item also have \( ... = \text{ISRD1}(R_s(\text{pre}_R(P) \vdash \false \odot (\text{post}_R(P) \land $tr^\prime = u \ $tr)) \)
\begin{enumerate}
\item by \( \text{simp add: SRD-reactive-tri-design \ ISRD-implies-SRD \ \text{assms}} \)
\item also have \( ... = R_s(\text{pre}_R(P) \vdash \false \odot (\text{post}_R(P) \land $tr^\prime = u \ $tr)) \)
\item by \( \text{simp add: ISRD1-rdes-def \ closure \ \text{assms}} \)
\end{enumerate}
\item finally show \( \?thesis \)
\item qed
\end{enumerate}
\end{enumerate}
\end{proof}

lemma \textit{ISRD-elim \ [RD-elim]}:
\begin{align*}
\sigma \ P \text{ is ISRD} \\
\sigma \ Q \ (R_s(\text{pre}_R(P) \vdash \false \odot (\text{post}_R(P) \land $tr^\prime = u \ $tr))) \Rightarrow \ Q(P)
\end{align*}
\begin{proof}
\begin{enumerate}
\item by \( \text{simp add: ISRD-form} \)
\item qed
\end{enumerate}
\end{proof}

lemma \textit{skip-srd-ISRD \ [closure]}: \( II_R \) is ISRD
\begin{proof}
\begin{enumerate}
\item by \( \text{rule ISRD-intro, simp-all add: rdes \ closure} \)
\item qed
\end{enumerate}
\end{proof}

lemma \textit{assigns-srd-ISRD \ [closure]}: \( \langle \sigma \rangle_R \) is ISRD
\begin{proof}
\begin{enumerate}
\item by \( \text{rule ISRD-intro, simp-all add: rdes \ closure, rel-auto} \)
\item done
\end{enumerate}
\end{proof}

lemma \textit{seq-ISRD-closed}:
\begin{proof}
\begin{enumerate}
\item assumes \( P \text{ is ISRD} \ Q \text{ is ISRD} \)
\item shows \( P ; ; Q \text{ is ISRD} \)
\item apply (insert \text{assms})
\item apply (erule ISRD-elim)+
\item apply (simp add: rdes-def \ closure \text{assms unrest})
\item apply (rule ISRD-rdes-intro)
\item apply (simp-all add: rdes-def closure \text{assms unrest})
\item apply (rel-auto)
\item done
\end{enumerate}
\end{proof}

lemma \textit{ISRD-Miracle-right-zero}:
\begin{proof}
\begin{enumerate}
\item assumes \( P \text{ is ISRD} \ \text{pre}_R(P) = \true_r \)
\item shows \( P ; ; \text{Miracle} = \text{Miracle} \)
\item by \( \text{rdes-simp cls: \text{assms}} \)
\item qed
\end{enumerate}
\end{proof}

A recursion whose body does not extend the trace results in divergence

lemma \textit{ISRD-recurse-Chaos}:
\begin{proof}
\begin{enumerate}
\item assumes \( P \text{ is ISRD} \ \text{post}_R P ; ; \text{true}_r = \false_r \)
\item shows \( (\mu_R X \cdot P ; ; X) = \text{Chaos} \)
\item proof
\begin{enumerate}
\item have \( 1: (\mu_R X \cdot P ; ; X) = (\mu X \cdot P ; ; \text{SRD}(X)) \)
\item qed
\end{enumerate}
\end{enumerate}
\end{proof}
by (auto simp add: srdes-theory-continuous.utp-lfp-def closure assms)
have \( \mu X \cdot P \;;\; \text{SRD}(X) \subseteq \text{Chaos} \)
proof (rule gfp-upperbound)
  have \( P \;;\; \text{Chaos} \subseteq \text{Chaos} \)
  apply (rdes-refine-split cls: assms)
  using assms(2) apply (rel-auto, metis (no-types, lifting) dual-order.antisym order-refl)
  apply (rel-auto)+
  done
  thus \( P \;;\; \text{SRD} \) \( \text{Chaos} \subseteq \text{Chaos} \)
  by (simp add: Healthy-if srdes-theory-continuous.bottom-closed)
qed
thus \(?\text{thesis}\)
  by (metis 1 dual-order.antisym srdes-theory-continuous.LFP-closed srdes-theory-continuous.bottom-lower)
qed

lemma recursive-assign-Chaos:
\( (\mu R \cdot \langle \sigma \rangle R :: X) = \text{Chaos} \)
by (rule ISRD-recurse-Chaos, simp-all add: closure rdes, rel-auto)

end

14 Meta-theory for Reactive Designs

theory utp-rea-designs
  imports
    utp-rdes-healths
    utp-rdes-designs
    utp-rdes-triples
    utp-rdes-normal
    utp-rdes-contracts
    utp-rdes-tactics
    utp-rdes-parallel
    utp-rdes-prag
    utp-rdes-instant
    utp-rdes-guarded
begin end

References

