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

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

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 (Unpublished)

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

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

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

Abstract

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

Contents

1 Introduction .................................................. 3

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

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

4 Reactive Design Triples ........................................... 26
  4.1 Diamond notation .......................................... 26
  4.2 Export laws ............................................... 27
  4.3 Pre-, peri-, and postconditions ................................ 28
    4.3.1 Definitions ......................................... 28
    4.3.2 Unrestriction laws .................................. 28
4.3.3 Substitution laws ........................................... 29
4.3.4 Healthiness laws ........................................... 30
4.3.5 Calculation laws ........................................... 33
4.4 Formation laws ............................................... 35
  4.4.1 Order laws ............................................... 36
4.5 Composition laws ........................................... 37
4.6 Refinement introduction laws .............................. 41
4.7 Closure laws ................................................ 43
4.8 Distribution laws ............................................ 43
4.9 Algebraic laws ............................................... 45
4.10 Recursion laws .............................................. 46

5 Normal Reactive Designs 47
  5.1 UTP theory .................................................. 60

6 Syntax for reactive design contracts 61

7 Reactive design tactics 62

8 Reactive design parallel-by-merge 63
  8.1 Example basic merge ....................................... 78
  8.2 Simple parallel composition ............................ 78

9 Productive Reactive Designs 79
  9.1 Healthiness condition ..................................... 79
  9.2 Reactive design calculations ............................ 81
  9.3 Closure laws ............................................... 82

10 Guarded Recursion 84
  10.1 Traces with a size measure ............................ 84
  10.2 Guardedness ............................................. 85
  10.3 Tail recursive fixed-point calculations ............... 89

11 Reactive Design Programs 90
  11.1 State substitution .......................................... 90
  11.2 Assignment ................................................ 91
  11.3 Conditional ............................................... 92
  11.4 Assumptions ............................................... 92
  11.5 Guarded commands ........................................ 93

12 Generalised Alternation 94
  12.1 Choose .................................................... 96
  12.2 State Abstraction ......................................... 96
  12.3 While Loop ............................................... 97
  12.4 Iteration Construction .................................. 101
  12.5 Substitution Laws ........................................ 101
  12.6 Algebraic Laws ........................................... 102
  12.7 Lifting designs to reactive designs ................... 103

13 Instantaneous Reactive Designs 105
1 Introduction

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

2 Reactive Designs Healthiness Conditions

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

2.1 Preliminaries

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

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

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

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

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

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

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

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

lemma R1-R2c-ex-st: R1 (R2c (∃ $st' · Q1)) = (∃ $st' · R1 (R2c Q1))
  by (rel-auto)

2.2 Identities

We define two identities for reactive designs, which correspond to the regular and state-sensitive versions of reactive designs, respectively. The former is the one used in the UTP book and related publications for CSP.

definition skip-reas :: ('t::trace, 'a) hrel-rp (IIc) where
  skip-reas-def [urel-defs]: IIc = (II ∨ (¬ $ok ∧ $tr ≤u $tr'))

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

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

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

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

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

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

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

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

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

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

2.3 RD1: Divergence yields arbitrary traces

definition RD1 :: ("t::trace,"'a,'a,'β) rel-rp ⇒ ("t,"'a,'a,'β) rel-rp where
  [upred-defs]: RD1(P) = (P ∨ (¬ $ok ∧ $tr ≤ₜ $tr'))

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

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

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

lemma RD1-mono: \( \text{P ⊆ Q} \Longrightarrow \text{RD1}(P) ⊆ \text{RD1}(Q) \)
  by (rel-auto)

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

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

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

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

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

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

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

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

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

lemma RD1-R1-cases: RD1(R1(P)) = (R1(true) ;; P) = P
by (rel-auto)

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

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

lemma RD1-algebraic-intro:
assumes \( P \) is R1 (R1(true) ;; P) = R1(true) (II_c ;; P) = P
shows \( P \) is RD1
proof −

have \( P = (II_c ;; P) \)
by (simp add: assms(3))

also have \( \ldots = (R1($ok \Rightarrow II) ;; P) \)
by (simp add: skip-rea-R1-lemma)

also have \( \ldots = (((\neg $ok \land R1(true)) ;; P) \lor P) \)
by (metis (no-types, lifting) R1-def seqr-left-unit seqr-or-distl skip-rea-R1-lemma skip-rea-def utp-pred-laws.inf-top-left utp-pred-laws.sup-commute)

also have \( \ldots = ((R1($ok ;; (R1(true) ;; P)) \lor P) \)
using dual-order.trans by (rel-blast)

also have \( \ldots = ((R1($ok ;; R1(true)) \lor P) \)
by (simp add: assms(2))

also have \( \ldots = (R1($ok) \lor P) \)
by (rel-auto)

also have \( \ldots = RD1(P) \)
by (rel-auto)

finally show \( \text{thesis} \)
by (simp add: Healthy-def)

qed
**Theorem RD1-left-zero:**

**Assumptions:**
- 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:**

- By (simp add: Healthy-if assms(1) assms(2))

qed

**Theorem RD1-left-unit:**

**Assumptions:**
- P is R1
- P is RD1

**Shows:**
\( IIc ;; P = P \)

**Proof:***

- Have \( IIc ;; R1(RD1(P)) = R1(RD1(P)) \)
  - By (rel-auto)

**Thus:**

- By (simp add: Healthy-if assms(1) assms(2))

qed

**Lemma RD1-alt-def:**

**Assumptions:**
- P is R1

**Shows:**
\[ RD1(P) = (P ⊢ ok ⊲ R1(true)) \]

**Proof:***

- Have \( RD1(R1(P)) = (R1(P) ⊢ ok ⊲ R1(true)) \)
  - By (rel-auto)

**Thus:**

- By (simp add: Healthy-if assms)

qed

**Theorem RD1-algebraic:**

**Assumptions:**
- P is R1

**Shows:**
\( P \iff (R1(true_h) ;; P) = R1(true_h) \land (IIc ;; P) = P \)

**Using:**
- RD1-algebraic-intro RD1-left-unit RD1-left-zero assms by blast

### 2.4 R3c and R3h: Reactive design versions of R3

**Definition R3c:**

\[ t::\text{trace}, \alpha \krel-rp \Rightarrow ('t', \alpha) \krel-rp \]

**[upred-defs]:**
\[ R3c(P) = (IIc \alpha \krel-rp \Rightarrow P) \]

**Definition R3h:**

\[ s::\text{trace}, \alpha \krel-rsp \Rightarrow ('s', \alpha) \krel-rsp \]

**[upred-defs]:**
\[ R3h(P) = ((\exists \$st \cdot IIc) \alpha \krel-rsp \Rightarrow P) \]

**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:**

\( P \subseteq Q \Rightarrow R3c(P) \subseteq R3c(Q) \)

**By (rel-auto)**

**Lemma R3c-Monotonic:**

Monotonic R3c

**By (simp add: mono-def R3c-mono)**
lemma \textit{R3c-Continuous}: Continuous R3c
by (rel-auto)

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

lemma \textit{R3h-Idempotent}: Idempotent R3h
by (simp add: Idempotent-def R3h-idem)

lemma \textit{R3h-mono}: P \subseteq Q \implies R3h(P) \subseteq R3h(Q)
by (rel-auto)

lemma \textit{R3h-Monotonic}: Monotonic R3h
by (simp add: mono-def R3h-mono)

lemma \textit{R3h-Continuous}: Continuous R3h
by (rel-auto)

lemma \textit{R3h-inf}: R3h(P \cap Q) = R3h(P) \cap R3h(Q)
by (rel-auto)

lemma \textit{R3h-UINF}:
\begin{align*}
A \neq \{} \Rightarrow R3h(\bigcap_{i \in A} \cdot P(i)) = (\bigcap_{i \in A} \cdot R3h(P(i)))
\end{align*}
by (rel-auto)

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

lemma \textit{R3c-via-RD1-R3}: RD1(R3c(P)) = R3c(RD1(P))
by (rel-auto)

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

lemma \textit{RD1-R3c-commute}: RD1(R3c(P)) = R3c(RD1(P))
by (rel-auto)

lemma \textit{R1-R3c-commute}: R1(R3c(P)) = R3c(R1(P))
by (rel-auto)

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

lemma \textit{R1-R3h-commute}: R1(R3h(P)) = R3h(R1(P))
by (rel-auto)

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

lemma \textit{RD1-R3h-commute}: RD1(R3h(P)) = R3h(RD1(P))
by (rel-auto)

lemma \textit{R3c-cancels-R3}: R3c(R3(P)) = R3c(P)
by (rel-auto)
lemma R3-cancel-R3c: R3(R3c(P)) = R3(P)
  by (rel-auto)

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

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

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

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

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

lemma 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 >> (P ;; Q))
    by (metis cond-var-split cond-var-subst-right in-var-ivar wait-vwb-lens)
  also have ... = (((II << wait >> P) ;; Q)[true/wait] < wait >> (P ;; Q))
    by (metis Healthy-def′ R3-def assms(1))
  also have ... = (((II[true/wait] ;; Q) < wait >> (P ;; Q))
    by (subst-tac)
  also have ... = (((II ∩ $wait′) ;; Q) < wait >> (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
    by (metis seqr-pre-transfer seqr-right-one-point true-alt-def uovar-cover upred-eq-true utp-rel.unrest-ivar
    vwb-lens-mub wait-vwb-lens)
  also have ... = (((II[true/wait]′) ;; Q,true/wait[wait]′) < wait >> (P ;; Q))
    by (metis Healthy-def′ R3c-def assms(2))
  also have ... = (((II[true/wait]′) ;; IIc[true/wait][wait]′) < wait >> (P ;; Q))
    by (subst-tac)
  also have ... = (((II ∩ $wait′) ;; IIc) < wait >> (P ;; Q))
    by (metis seqr-pre-transfer seqr-right-one-point true-alt-def uovar-cover upred-eq-true utp-rel.unrest-ivar
    vwb-lens-mub wait-vwb-lens)
  also have ... = (((II ;; IIc) < wait >> (P ;; Q))
    by (simp add: cond-def seqr-pre-transfer uovar.unrest-var)
  also have ... = (IIc < wait >> (P ;; Q))
    by simp
  also have ... = R3c(P ;; Q)
    by (simp add: R3c-def)
finally show ?thesis
  by (simp add: Healthy-def′)
lemma \( R3c\text{-cases} \): \( R3c(P) = ((\mathcal{H} \triangleleft \mathsf{ok} \triangleright R1(true)) \triangleleft \mathsf{wait} \triangleright P) \)
by (rel-auto)

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

lemma \( R3h\text{-form} \): \( R3h(P) = \mathcal{H} \triangleleft \mathsf{wait} \triangleright P \)
by (rel-auto)

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

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

lemma skip-srea-R3h \([\text{closure}]\): \( \mathcal{H} \) is \( R3h \)
by (rel-auto)

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

2.5 RD2: A reactive specification cannot require non-termination

definition RD2 where
\([\text{upred-defs}]\): \( RD2(P) = H2(P) \)

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

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

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

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

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

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

lemma RD1-RD2-commute: \( RD1(RD2(P)) = RD2(RD1(P)) \)
by (rel-auto)
lemma RD2-R3c-commute: RD2(R3c(P)) = R3c(RD2(P))
  by (rel-auto)

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

2.6 Major healthiness conditions

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

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

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

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

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

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

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

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

lemma RH-idem: R(R(P)) = R(P)
  by (simp add: RH-idem)

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

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

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

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

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

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

lemma RHS-mono: P ⊑ Q ⇒ Rs(P) ⊑ Rs(Q)
using mono-def RHS-Monotonic by blast

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

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

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

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

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

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

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

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

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

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

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

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

lemma R3-RD-RP:
  R3(R(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 RP-def)

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

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

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

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

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

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

lemma SRD-healths [closure]:
  assumes P is SRD
  shows P is R1 P is R2 P is R3h P is RD1 P is RD2
  apply (metis Healthy-def R1-R2c-is-R2 RHS-def SRD-def assms)
  apply (metis Healthy-def R1-R3h-commute RD2-RHS-commute RD2-R3h-commute
           RHS-def 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 =⇒ 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-healths(1) design-ok-false)

lemma SRD-ok-true-wait-true [usubst]:
  assumes P is SRD
  shows P[true,true/$ok,$wait] = (∃ st · II)[true,true/$ok,$wait]
  proof –
    have P = (∃ st · II) << $ok ◂ R1 true ◄ $wait ◃ P
      by (metis Healthy-def R3h-cases SRD-healths(3) assms)
    moreover have ((∃ st · II) ◄ $ok ◃ R1 true ◄ $wait ◃ P)[true,true/$ok,$wait] = (∃ st · II)[true,true/$ok,$wait]
      by (simp add: usubst)
    ultimately show ?thesis
      by (simp)
  qed

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

lemma SRD-left-zero-2:
  assumes P is SRD
  shows (∃ st · II)[true,true/$ok,$wait] ;; P = (∃ st · II)[true,true/$ok,$wait]
  proof –
    have (∃ st · II)[true,true/$ok,$wait] ;; Rs(P) = (∃ st · II)[true,true/$ok,$wait]
We create two theory objects: one for reactive designs and one for stateful reactive designs.

**typedef** \texttt{RDES}

**abbreviation** \texttt{RDES} $\equiv$ \texttt{UTHY}(\texttt{RDES}, ('t::trace,'a) \texttt{rp})

**abbreviation** \texttt{SRDES} $\equiv$ \texttt{UTHY}(\texttt{SRDES}, ('s,'t::trace,'a) \texttt{rsp})

**overloading**

\texttt{rdes-hcond} $== u\texttt{tp-hcond} :: (\texttt{RDES}, ('t::trace,'a) \texttt{rp}) \text{ uthy} \Rightarrow (('t,'a) \texttt{rp} \times ('t,'a) \texttt{rp}) \text{ health}

\texttt{srdes-hcond} $== u\texttt{tp-hcond} :: (\texttt{SRDES}, ('s,'t::trace,'a) \texttt{rsp}) \text{ uthy} \Rightarrow (('s,'t,'a) \texttt{rsp} \times ('s,'t,'a) \texttt{rsp}) \text{ health}

**begin**

**definition** \texttt{rdes-hcond} :: (\texttt{RDES}, ('t::trace,'a) \texttt{rp}) \text{ uthy} \Rightarrow (('t,'a) \texttt{rp} \times ('t,'a) \texttt{rp}) \text{ health where}

\texttt{[upred-defs]: rdes-hcond} $\text{ T = RD}$

**definition** \texttt{srdes-hcond} :: (\texttt{SRDES}, ('s,'t::trace,'a) \texttt{rsp}) \text{ uthy} \Rightarrow (('s,'t,'a) \texttt{rsp} \times ('s,'t,'a) \texttt{rsp}) \text{ health where}

\texttt{[upred-defs]: srdes-hcond} $\text{ T = SRD}$

**end**

**interpretation** \texttt{rdes-theory; u\texttt{tp-theory}} \texttt{UTHY}(\texttt{RDES}, ('t::trace,'a) \texttt{rp})

**by** (\texttt{unfold-locales, simp-all add: rdes-hcond-def RD-idem})

**interpretation** \texttt{rdes-theory-continuous; u\texttt{tp-theory-continuous}} \texttt{UTHY}(\texttt{RDES}, ('t::trace,'a) \texttt{rp})

**rewrites** \texttt{\bigwedge P. P \in carrier (uthy-order RDES) \leftrightarrow P \text{ is RD}}

**and** \texttt{carrier (uthy-order RDES) \rightarrow carrier (uthy-order RDES) \equiv [RD]_H \rightarrow [RD]_H}

**and** \texttt{le (uthy-order RDES) = op \subseteq}

**and** \texttt{eq (uthy-order RDES) = op =}

**by** (\texttt{unfold-locales, simp-all add: rdes-hcond-def RD-Continuous})

**interpretation** \texttt{rdes-rea-galois:}

\texttt{galois-connection} (\texttt{RDES} $\leftarrow$ (\texttt{RD1} $\circ$ \texttt{RD2}.$\texttt{R3}$) $\rightarrow$ \texttt{REA})

**proof** (\texttt{simp add: mk-conn-def, rule galois-connectionI', simp-all add: u\texttt{tp-partial-order rdes-hcond-def rea-hcond-def})

**show** \texttt{R3} $\in [RD]_H \rightarrow [\texttt{RP}]_H$

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

**show** \texttt{RD1} $\circ$ \texttt{RD2} $\in [\texttt{RP}]_H \rightarrow [RD]_H$

**by** (\texttt{simp add: Pi-iff Healthy-def, metis RD-def RD-idem})

**show** \texttt{isotone} (\texttt{utp-order RD}) (\texttt{utp-order RP}) \texttt{R3}

**by** (\texttt{simp add: R3-Monotonic isotone-utp-orderI})

**show** \texttt{isotone} (\texttt{utp-order RP}) (\texttt{utp-order RD}) (\texttt{RD1} $\circ$ \texttt{RD2})

**by** (\texttt{simp add: Monotonic-comp RD1-Monotonic RD2-Monotonic isotone-utp-orderI})

**fix** \texttt{P} :: ('a, 'b) \texttt{hrel-rp}

**assume** \texttt{P is RD}

**thus** \texttt{P $\subseteq$ RD1 (RD2 (R3 P))}

**by** (\texttt{metis Healthy-if R3-RD-RP RD-def RP-idem eq-iff})

**next**

**fix** \texttt{P} :: ('a, 'b) \texttt{hrel-rp}

**assume** \texttt{a: P is RP}

13
thus \( R3 (RD1 (RD2 P)) \subseteq P \)

proof

- have \( R3 (RD1 (RD2 P)) = RP (RD1 (RD2 P)) \)
  by (metis Healthy-if R3-RD-RP RD-def a)

moreover have \( RD1 (RD2 P) \subseteq P \)
by (rel-auto)

ultimately show ?thesis
by (metis Healthy-if RP-mono a)
qued

qed

interpretation rdes-rea-retract:
retract \((RDES \leftarrow \langle RD1 \circ RD2, R3 \rangle \rightarrow REA)\)
by (unfold-locales, simp-all add: mk-conn-def utp-partial-order rdes-hcond-def rea-hcond-def)
(metis Healthy-if R3-RD-RP RD-def RP-idem eq-refl)

interpretation srdes-theory: utp-theory UTHY \((SRDES, \langle s', t :: \text{trace}, \alpha \rangle \rangle \text{ rsp})\)
by (unfold-locales, simp-all add: srdes-hcond-def SRD-Continuous)

interpretation srdes-theory-continuous: utp-theory-continuous UTHY \((SRDES, \langle s', t :: \text{trace}, \alpha \rangle \rangle \text{ rsp})\)
rewrites \( \forall P. P \in \text{carrier} \ (\text{uthy-order SRDES}) \iff P \text{ is SRD} \)
and \( P \text{ is } H_{SRDES} \iff P \text{ is SRD} \)
and \( (\mu X \cdot F (H_{SRDES} X)) = (\mu X \cdot F (SRD X)) \)
and carrier (uthy-order SRDES) \( \rightarrow \) carrier (uthy-order SRDES) \( \equiv [SRD]_H \rightarrow [SRD]_H \)
and \( \text{le } (\text{uthy-order SRDES}) = \text{op } \subseteq \)
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 :: \( \langle s', t :: \text{trace}, \alpha \rangle \text{ hrel-rsp where} \)
Chaos \( \equiv \bot_{SRDES} \)

abbreviation Miracle :: \( \langle s', t :: \text{trace}, \alpha \rangle \text{ hrel-rsp where} \)
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 \( (\mu_R) \text{ where } \mu_R F \equiv \mu_{SRDES} F \)

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

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

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

The reactive design weakest fixed-point can be defined in terms of relational calculus one.
lemma srd-mu-equiv:
assumes Monotonic \( F F \in [\text{SRD}]_H \rightarrow [\text{SRD}]_H \)
shows \((\mu_R X \cdot F(X)) = (\mu X \cdot F(\text{SRD}(X)))\)
by (metis assms srdes-hcond-def srdes-theory-continuous.utp-lfp-def)

end

3 Reactive Design Specifications

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

3.1 Reactive design forms

lemma srdes-skip-def: \( II_R = R_s(\text{true} \vdash (\$tr' = u \$tr \land \neg \$wait' \land [II]_R)) \)
apply (rel-auto) using minus-zero-eq by blast+

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

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

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

lemma RD2-reactive-design:
  assumes \( \$ok' \not\in P \$ok' \not\in Q \)
  shows \( RD2(R(P \vdash Q)) = R(P \vdash Q) \)
  using assms
by (metis H2-design RD2-RH-commute RD2-def)

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

lemma RD2-st-reactive-design:
  assumes \( \$ok' \not\in P \$ok' \not\in Q \)
shows \( RD2(R_a(P \vdash Q)) = R_a(P \vdash Q) \)
using assms
by (metis H2-design RD2-RHS-commute RD2-def)

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

lemma RD-RH-design-form:
\( RD(P) = R((\neg P f) \vdash P f) \)
proof –
have \( RD(P) = RD1(RD2(R1(R2c(R3c(P)))))) \)
by (simp add: RD-alt-def RH-def)
also have \( ... = RD1(H2(R1(R2s(R3c(P)))))) \)
by (simp add: R1-R2s-R2c RD2-def)
also have \( ... = RD1(R1(H2(R2s(R3c(P)))))) \)
by (simp add: R1-H2-commute)
also have \( ... = R1(R1(R2s(R3c(P)))))))) \)
by (simp add: R1-idem RD1-via-R1)
also have \( ... = R1(R1(H2(R2s(R3c(R1(P))))))) \)
by (simp add: R1-R2c-commute R1-R2-s R2c R1-R3c-commute RD1-via-R1)
also have \( ... = R1(R2s(H1(H2(R3c(R1(P))))))) \)
by (simp add: R2s-R1-commute R2s-H2-commute)
also have \( ... = R2s(R1(R1(H2(R3c(R1(P))))))) \)
by (metis RD2-R3c-commute RD2-def)
also have \( ... = R2(R1(H1(R3c(R2s(R1(P))))))) \)
by (metis R1-R2c-commute R1-idem R2-def)
also have \( ... = R2(R3c(R1(H(R1(P))))))) \)
by (simp add: R1-R3c-commute RD1-R3c-commute RD1-via-R1)
also have \( ... = RH(R1(P))) \)
by (metis R1-R2c-commute R1-R3c-commute R2-R1-form RH-def)
also have \( ... = RH(H(R1(P)))) \)
by (simp add: R1-R2c-commute R1-R3c-commute R1-idem RD1-via-R1 RH-def)
also have \( ... = RH((\neg P f) \vdash P f) \)
by (simp add: H1-H2-eq-design)
also have \( ... = R((\neg P f) \vdash P f) \)
by (metis no-types, lifting R3c-subst-wait RH-def subst-not wait-\( f\)-\( f\)-design)
finally show \(?\)thesis .

qed

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

lemma RD-RH-design:
assumes \$ok’ \( \not \vdash P \$ok’ \( \not \vdash Q \)
shows \( RD(R(P \vdash Q)) = R(P \vdash Q) \)
by (simp add: RD1-reactive-design RD2-reactive-design RD-alt-def RH-idem assms(1) assms(2))

lemma RH-design-is-RD:
assumes \$ok’ \( \not \vdash P \$ok’ \( \not \vdash Q \)
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_\sigma((\neg P_{f_1}) \vdash P_{f_1})

proof
  have SRD(P) = R_1(R_2 c(R_3 h(R_1(R_2 d(R_1(P))))))
    by (metis (no-types, lifting) R_1-H2-commute R_1-R2c-commute R_1-R3h-commute R_1-idem R_2c-H2-commute R_1-R2s-commute R_1-R3h-subst-wait RHS-def subst-not wait-false-design)
  also have ... = R_1(R_2 c(H(P)))
    by (simp add: R_1-R2s-R2c RHS-def)
  also have ...
    by (metis (no-types, lifting) H_1-H2-eq-design)
  finally show ?thesis.
qed

lemma SRD-reactive-design:
assumes P is SRD
shows R_\sigma((\neg P_{f_1}) \vdash P_{f_1}) = P
by (metis SRD-RH-design-form Healthy-def)

lemma SRD-RH-design:
assumes $ok\sharp P $ok\sharp Q
shows SRD(R_\sigma(P \vdash Q)) = R_\sigma(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\sharp P $ok\sharp Q
shows R_\sigma(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_\sigma(H(P))
by (metis (no-types, lifting) H_1-H2-eq-design)

3.2 Auxiliary healthiness conditions

definition [upred-defs]: R_3c-pre(P) = (true \& \& $wait \triangleright P)
definition [upred-defs]: R_3c-post(P) = ([I]_D \& $wait \triangleright P)
definition [upred-defs]: R_3h-post(P) = ((\exists st \cdot [I]_D) \& $wait \triangleright P)
lemma R_3c-pre-conj: R_3c-pre(P \land Q) = (R_3c-pre(P) \land R_3c-pre(Q))
by (rel-auto)
lemma R_3c-pre-seq:
  (true ;; Q) = true \implies R_3c-pre(P ;; Q) = (R_3c-pre(P) ;; Q)
by (rel-auto)
lemma unrest-ok-R_3c-pre [unrest]: $ok \n P \implies $ok \n R_3c-pre(P)
by (simp add: R_3c-pre-cond-def unrest)
lemma unrest-ok'-R_3c-pre [unrest]: $ok' \n P \implies $ok' \n R_3c-pre(P)
3.3 Composition laws

**Theorem R1-design-composition:**

- **Fixes** $P$ $Q$ :: ('t',trace,'α','β') rel-rp
- **And** $R$ $S$ :: ('t','β','γ') rel-rp
- **Assumes** $\$ok' $\not\in$ $P$ $\$ok' $\not\in$ $Q$ $\$ok $\not\in$ $R$ $\$ok $\not\in$ $S$

**Shows**

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

**Proof**

- **Have** $(R1(P \triangleright Q) ; R1(R \triangleright S)) = (\exists ok0 \cdot (R1(P \triangleright Q)[<ok0>/$\$ok'] ; (R1(R \triangleright S))[<ok0>/$\$ok])$

  **Using** seqr-middle ok-vub-lens **by** blast

- **Also from** assms have ...

  - $(R1((\neg ok \land \Rightarrow (true \land Q)) ; R1((\neg ok \land \Rightarrow (true \land Q)))$

  **By simp add: design-def R1-def usubst unrest

- **Also from** assms have ...

  - $(R1(\neg (ok \lor \Rightarrow (\neg P \land Q)) ; R1(\neg (ok \lor \Rightarrow (\neg P \land Q)))$

  **By simp add: impl-alt-def utp-pred-laws.sup.assoc

- **Also from** assms have ...

  - $(R1((\neg ok \lor \Rightarrow (\neg P \land Q)) \lor R1(Q)) ; R1(\neg (ok \lor \Rightarrow (\neg P \land Q)))$

  **By simp add: R1-disj utp-pred-laws.disj-assoc

- **Also from** assms have ...

  - $(R1(\neg ok \lor \Rightarrow (\neg P \land Q)) \lor R1(\neg (ok \lor \Rightarrow (\neg P \land Q)))$

  **By simp add: seqr-or-distl utp-pred-laws.sup.assoc

- **Also from** assms have ...

  - $(R1(\neg ok \lor \Rightarrow (\neg P \land Q)) \lor R1(\neg (ok \lor \Rightarrow (\neg P \land Q)))$

  **By (rel-blast

- **Also from** assms have ...

  - $(R1(Q)) ; R1(\neg (ok \lor \Rightarrow (\neg P \land Q)))$

  **By simp add: R1-disj R1-extend-conj utp-pred-laws.inf-commute

- **Also have** ...

  - $(R1(Q)) ; R1(\neg (ok \lor \Rightarrow (\neg P \land Q)))$

  **By simp add: R1-disj seqr-or-distl

- **Also have** ...

  - $(R1(Q)) ; R1(\neg (ok \lor \Rightarrow (\neg P \land Q)))$

  **By simp add: R1-disj R1-extend-conj utp-pred-laws.inf-commute
\[ \begin{align*}
\& \lor (R1(\neg \$ok)) \\
\& \lor (R1(\neg P) ; R1(true))) \\
\end{align*} \]

**proof**

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

**qed**

also have \(\ldots \) = \([(R1(Q) :: (R1(\neg R) \lor (R1(S \land \$ok')))\)]

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

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

also have \(\ldots \) = \((R1(Q) :: (R1(\neg R)))\]

\[ \begin{align*}
\& \lor (R1(Q) :: (R1(\neg R))) \\
\& \lor (R1(S \land \$ok')) \\
\& \lor (\neg \$ok) \\
\& \lor (R1(\neg P) ; R1(true))) \\
\end{align*} \]

by \((simp add: R1-disj R1-seqr)\)

also have \(\ldots \) = \((R1(Q) :: (R1(\neg R)))\]

\[ \begin{align*}
\& \lor (R1(Q) :: (R1(\neg R))) \\
\& \lor (\neg \$ok) \\
\& \lor (R1(\neg P) ; R1(true))) \\
\end{align*} \]

by \((rel-blast)\)

also have \(\ldots \) = \((R1(\neg \$ok \land \neg (R1(\neg P) ; R1(true)) \land \neg (R1(Q) :: (R1(\neg R)))))\]

\[ \begin{align*}
\& \lor (R1(Q) :: (R1(\neg R))) \\
\& \lor (\neg \$ok) \\
\& \lor (R1(\neg P) ; R1(true))) \\
\end{align*} \]

by \((rel-blast)\)

also have \(\ldots \) = \((R1(\neg \$ok \land \neg (R1(\neg P) ; R1(true)) \land \neg (R1(Q) :: (R1(\neg R)) \lor (R1(Q) :: R1(S))))\]

\[ \begin{align*}
\& \lor (R1(\neg \$ok \land \neg (R1(\neg P) ; R1(true)) \land \neg (R1(Q) :: (R1(\neg R)) \lor (R1(Q) :: R1(S)))) \\
\end{align*} \]

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

also have \(\ldots \) = \((R1(\neg \$ok \land \neg (R1(\neg P) ; R1(true)) \land \neg (R1(Q) :: (R1(\neg R)) \lor (R1(Q) :: R1(S))))\]

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

finally show \(?thesis\).

**qed**

**theorem** \(R1-design-composition-RR:\)

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

shows \((R1(P \vdash Q) ; R1(R \vdash S)) = R1(((\neg r \ P) wp_r false \land Q wp_r R) \vdash (Q :: S))\)

apply \((subst R1-design-composition)\)
apply \((simp-all add: assms unrest wp-rea-def Healthy-if closure)\)
apply \((rel-auto)\)

**done**

**theorem** \(R1-design-composition-RC:\)

assumes \(P\) is \(RC\) \(Q\) is \(RR\) \(R\) is \(RR\) \(S\) is \(RR\)

shows \((R1(P \vdash Q) ; R1(R \vdash S)) = R1((P \land Q wp_r R) \vdash (Q :: S))\)

by \((simp add: R1-design-composition-RR assms unrest Healthy-if closure wp)\)

**lemma** \(R2s-design: R2s(P \vdash Q) = (R2s(P) \vdash R2s(Q))\)

19
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 \( \text{ok}' \notin P \text{ ok}' \notin Q \text{ ok} \notin R \text{ ok} \notin S \)
shows \( R3c(R1(P \vdash Q)) :: R3c(R1(R \vdash S)) = R3c(R1((\neg R1(P) :: R1(true)) \land \neg ((R1(Q) \land \neg \text{wait}') :: R1(\neg R))) \)
\( \vdash (R1(Q) :: ([II]_D < \text{wait} \triangleright R1(S)))) \)
proof -
  have 1: \( (\neg (R1 (\neg R3c-pre P) :: R1 true)) = (R3c-pre (\neg (R1 (\neg P) :: R1 true))) \)
    by (rel-auto)
  have 2: \( (\neg (R1 (R3c-post Q) :: R1 (\neg R3c-pre R))) = R3c-pre(\neg ((R1 Q \land \neg \text{wait}') :: R1 (\neg R))) \)
    by (rel-auto, blast+)
  have 3: \( (R1 (R3c-post Q) :: R1 (R3c-post S)) = R3c-post (R1 Q :: ([II]_D < \text{wait} \triangleright R1 S)) \)
    by (rel-auto)
  show \text{thesis}
    apply (simp add: R3c-semir-form R1-R3c-commute THEN sym R1-R3c-design unrest)
    apply (subst R1-design-composition)
    apply (simp-all add: unrest assms R3c-pre-conj 1 2 3)
  done

qed

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

qed

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

20
lemma RH-design-composition:
  assumes $\$ok \notin P \$ok \notin Q \$ok \notin R \$ok \notin S$
  shows $(RH(P \vdash Q) \vdash RH(R \vdash S)) =
  RH((\neg (R \neg R2s P) \vdash R1 true) \land \neg ((R1 (R2s Q) \land (\neg \$wait')) \vdash R1 \neg R2s R)) \vdash
  (R1 (R2s Q) \vdash ([\!\![I]_{D} \neg \$wait \triangleright R1 (R2s S)]))$
proof –
  have 1: $R2c (R1 \neg R2s P) \vdash R1 true) = (R1 \neg R2s P) \vdash R1 true)$
proof –
  have 1:(R1 \neg R2s P) \vdash R1 true) = (R1(R2 \neg P) \vdash R2 true))
    by (rel-auto)
  have $R2c(R1(R2 \neg P) \vdash R2 true)) = R2c(R1(R2 \neg P) \vdash R2 true))$
    using $R2c$-not by blast
  also have ... = $R2c(R2 \neg P) \vdash R2 true)
    by (metis $R1$-$R2c$-commute $R1$-$R2c$-is-$R2$)
  also have ... = $(R2 \neg P) \vdash R2 true)
    by (simp add: $R2$-segr-distribute)
  also have ... = $(R1 \neg R2s P) \vdash R1 true)
    by (simp add: $R2$-def $R2c$-not $R2$s-true)
  finally show $\$thesis$
    by (simp add: 1)
qed

have 2:$R2c ((R1 (R2s Q) \land \neg \$wait') \vdash R1 \neg R2s R)) = ((R1 (R2s Q) \land \neg \$wait') \vdash R1 \neg R2s R))$
proof –
  have $((R1 (R2s Q) \land \neg \$wait') \vdash R1 \neg R2s R)) = R1 (R2 (Q \land \neg \$wait') \vdash R2 \neg R))$
    by (rel-auto)
  hence $R2c ((R1 (R2s Q) \land \neg \$wait') \vdash R1 \neg R2s R)) = (R2 (Q \land \neg \$wait') \vdash R2 \neg R))$
    by (metis $R1$-$R2c$-commute $R1$-$R2c$-is-$R2$ $R2$-segr-distribute)
  also have ... = $(R1 (R2s Q) \land \neg \$wait') \vdash R1 \neg R2s R))$
    by (rel-auto)
  finally show $\$thesis$
    .
qed

have 3:$R2c((R1 (R2s Q) \vdash ([\!\![I]_{D} \neg \$wait \triangleright R1 (R2s S)])) = (R1 (R2s Q) \vdash ([\!\![I]_{D} \neg \$wait \triangleright R1 (R2s S)]))$
proof –
  have $R2c((R1 (R2s Q))$true/$\$wait"]) \vdash ([\!\![I]_{D} \neg \$wait \triangleright R1 (R2s S)][true/$\$wait")]) =
    $([R1 (R2s Q))true/$\$wait") \vdash ([\!\![I]_{D} \neg \$wait \triangleright R1 (R2s S)][true/$\$wait")])$
    by (simp add: $\$wait$-def $\$wait$)\_\$wait\_map \_\$wait$\_map)$
  also have ...
    $R2c(R2(Q$true/$\$wait")]) \vdash ([I]_{D}true/$\$wait")])$
    by (metis $R2$-def $R2$-des-$\$wait$\_skip$ $R2$-subst-$\$wait$\_true$
  also have ...
    $(R2(Q$true/$\$wait")]) \vdash ([I]_{D}true/$\$wait")])$
    using $R2c$-seq by blast
  also have ...
    $(R2(Q$true/$\$wait")]) \vdash ([I]_{D}true/$\$wait")])$
    by (metis $\$wait$-def $\$wait$)\_\$wait\_map \_\$wait$\_map)$
  apply (simp add: $\$wait$-def $\$wait$)\_\$wait\_map \_\$wait$\_map)

21
apply \((\text{metis } R2\text{-def } R2\text{-des-lift-skip } R2\text{-subst-wait'\text{-true } R2\text{-subst-wait-true})}\)
done
finally show \?thesis .
qed
moreover have \(R2c((\{R1 (R2s \text{ Q})\} [false/\$wait']) : (\{\lfloor I\rfloor_D \triangleleft \$wait \triangleright R1 (R2s S)\} [false/\$wait'])\)  
\(= ((\{R1 (R2s \text{ Q})\} [false/\$wait']) : (\{\lfloor I\rfloor_D \triangleleft \$wait \triangleright R1 (R2s S)\} [false/\$wait'])\)
by (simp add: usubst cond-unit-F)
(metis (no-types, hide-lams) \(R1\text{-wait'\text{-false } R1\text{-wait-true } R2\text{-def } R2\text{-subst-wait'\text{-false } R2\text{-subst-wait-false})}\)
ultimately show \?thesis
proof –
  have \([I]_D \triangleleft \$wait \triangleright R1 (R2s S) = R2 ([I]_D \triangleleft \$wait \triangleright S)\)
  by (simp add: R1-R2c-is-R2 R1-R3c-commute R2c-R3c-commute RH-def)
then show \?thesis
  by (simp add: R1-R2c-is-R2 R1-R2s-R2c R2c-seq)
qed
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 (metis (no-types, hide-lams) R1-R2s-R2c R1-R3c-commute R2c-R3c-commute R2s-design)
also have \(\ldots = R3c(R1(\sim (R1 (\sim R2s \text{ P}) ; R1 \text{ true}) \land \sim ((R1 (R2s Q) \land \sim \$\text{wait'})) ; R1 (\sim R2s R)))))\) ⊢  
\((R1 (R2s Q) ; ([I]_D \triangleleft \$\text{wait} \triangleright R1 (R2s S))))\)
by (simp add: R3c-R1-design-composition assms unrest)
also have \(\ldots = R3c(R1(R2c(\sim (R1 (\sim R2s \text{ P}) ; R1 \text{ true}) \land \sim ((R1 (R2s Q) \land \sim \$\text{wait'})) ; R1 (\sim R2s R)))))\) ⊢  
\((R1 (R2s Q) ; ([I]_D \triangleleft \$\text{wait} \triangleright R1 (R2s S))))\)
by (simp add: R2c-design R2c-and R2c-not 1 2 3)
finally show \?thesis
by (simp add: R1-R2s-R2c R1-R3c-commute R2c-R3c-commute RH-def)
qed

lemma \(RHS\text{-design-composition:}\)
assumes \$\text{ok' \notin P } \$\text{ok' \notin Q } \$\text{ok \notin R } \$\text{ok \notin S}\)
shows \(\text{R}_s((\sim (R1 (\sim R2s P) ; R1 \text{ true}) \land \sim ((R1 (R2s Q) \land \sim \$\text{wait'})) ; R1 (\sim R2s R)))))\) ⊢  
\((R1 (R2s Q) ; ([I]_D \triangleleft \$\text{wait} \triangleright R1 (R2s S))))\)
proof –
  have \(1: R2c(R1 (\sim R2s P) ; R1 \text{ true}) = (R1 (\sim R2s P) ; R1 \text{ true})\)
  by (rel-auto, blast)
  have \(R2c(R1(R2 (\sim P) ; R2 \text{ true})) = R2c(R1(R2 (\sim P) ; R2 \text{ true}))\)
  using \(R2c\text{-not by blast}\)
  also have \(\ldots = R2(R2 (\sim P) ; R2 \text{ true})\)
  by (metis R1-R2c-commute R1-R2c-is-R2)
  also have \(\ldots = (R2 (\sim P) ; R2 \text{ true})\)
  by (simp add: R2-seqr-distribute)
  also have \(\ldots = (R1 (\sim R2s P) ; R1 \text{ true})\)
  by (simp add: R2-def R2s-not R2s-true)
finally show \?thesis
by (simp add: 1)
qed
have 2: R2c ((R1 (R2s Q) ∧ ¬ $wait \prime$) ; R1 (¬ R2s R)) = ((R1 (R2s Q) ∧ ¬ $wait \prime$) ; R1 (¬ R2s R))
proof
  have (((R1 (R2s Q) ∧ ¬ $wait \prime$) ; R1 (¬ R2s R)) = R1 (R2 (Q ∧ ¬ $wait \prime$) ; R2 (¬ R)))
  by (rel-auto, blast+)
  hence R2c (((R1 (R2s Q) ∧ ¬ $wait \prime$) ; R1 (¬ R2s R)) = (R2 (Q ∧ ¬ $wait \prime$) ; R2 (¬ R)))
  by (metis (no-types, lifting) R1-R2c-commute R1-R2c-is-R2 R2-seq-distribute)
  also have ... = (((R1 (R2s Q) ∧ ¬ $wait \prime$) ; R1 (¬ R2s R))
  by (rel-auto, blast+)
  finally show ?thesis .
qed

have 3: R2c((R1 (R2s Q) ;; (∃ $st \cdot [I]_D) < $wait > R1 (R2s S)))) = (R1 (R2s Q) ;; (∃ $st \cdot [I]_D) < $wait > R1 (R2s S)))
proof
  have R2c(((R1 (R2s Q))[true/$wait \prime$] ;; (∃ $st \cdot [I]_D) < $wait > R1 (R2s S))[true/$wait])
  = ((R1 (R2s Q))[true/$wait \prime$] ;; (∃ $st \cdot [I]_D) < $wait > R1 (R2s S))[true/$wait])
  by (simp add: usubst cond-unit-T R1-def R2-def)
  also have ... = R2c(R2(Q[true/$wait \prime$]) ;; R2((∃ $st \cdot [I]_D)[true/$wait])
  by (metis (no-types, lifting) R2-def R2-des-lift-skip R2-subst-wait-true R2-st-ex)
  also have ... = (R2(Q[true/$wait \prime$]) ;; R2((∃ $st \cdot [I]_D)[true/$wait])
  using R2c-seq by blast
  also have ... = ((R1 (R2s Q))[true/$wait \prime$]) ;; (∃ $st \cdot [I]_D) < $wait > R1 (R2s S))[true/$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/$wait \prime$] ;; (∃ $st \cdot [I]_D) < $wait > R1 (R2s S))[false/$wait])
  = ((R1 (R2s Q))[false/$wait \prime$] ;; (∃ $st \cdot [I]_D) < $wait > R1 (R2s S))[false/$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) ∧ ¬ $wait \prime$)) ; R1 (¬ R2s R))
  by (simp add: R3h-R1-design-composition assms unrest)
  also have ... = R3h(R1(R2c((¬ (R1 (¬ R2s P)) ;; R1 true) ∧ ¬ ((R1 (R2s Q) ∧ ¬ $wait \prime$)) ; R1 (¬ R2s R)))
  by (simp add: R2c-design R2c-and R2c-not 1 2 3)
  finally show ?thesis
  by (simp add: R1-R2s-R2c R1-R3h-commute R2c-R3h-commute RHS-def)
qed
lemma RHS-R2s-design-composition:
  assumes
  \(\text{\$ok} \not\in P \quad \text{\$ok} \not\in Q \quad \text{\$ok} \not\in R \quad \text{\$ok} \not\in S\)
  \(P\) is R2s \(Q\) is R2s \(R\) is R2s \(S\) is R2s
  shows \((R_s(P \vdash Q) ; R_s(R \vdash S)) = \)
  \(R_s((\neg (R1 (\neg P) ; R1 \text{ true}) \wedge \neg ((R1 Q \wedge \neg \text{\$wait'}) ; R1 (\neg R))) \vdash (R1 Q ; ((\exists st \cdot [H|P] < \text{\$wait} v R1 S))))\)

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 \(?thesis\)
    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: \(R_s(P[\text{true, false}/\text{\$ok, \$wait}] \vdash Q[\text{true, false}/\text{\$ok, \$wait}]) = R_s(P \vdash Q)\)
  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:
  \(R_s((\neg R1 P) \wedge Q) \vdash R) = R_s(((\neg P) \wedge Q) \vdash R)\)
  by (rel-auto)
lemma \( \text{RHS-pre-lemma}: (R_s P)^f_f = R_I (R_{2c}(P^f_f)) \)
by (rel-auto)

lemma \( \text{RHS-design-R2c-pre}: R_s(R_{2c}(P) \triangleright Q) = R_s(P \triangleright Q) \)
by (rel-auto)

3.4 Refinement introduction laws

lemma \( \text{R1-design-refine}: \)
\begin{align*}
& \text{assumes} \\
& P_1 \text{ is } R_1 P_2 \text{ is } R_1 Q_1 \text{ is } R_1 Q_2 \text{ is } R_1 \\
& \text{shows } R_1(P_1 + P_2) \subseteq R_1(Q_1 + Q_2) \iff 'P_1 \Rightarrow Q_1' \land 'P_1 \land Q_2 \Rightarrow P_2' \\
& \text{proof} -- \text{have } R_1((\exists \text{ ok};\text{ok} \cdot P_1) + (\exists \text{ ok};\text{ok} \cdot P_2)) \subseteq R_1((\exists \text{ ok};\text{ok} \cdot Q_1) + (\exists \text{ ok};\text{ok} \cdot Q_2)) \\
& \quad \iff 'R_1(\exists \text{ ok};\text{ok} \cdot P_1) \Rightarrow R_1(\exists \text{ ok};\text{ok} \cdot Q_1) \land R_1(\exists \text{ ok};\text{ok} \cdot Q_2) \land R_1(\exists \text{ ok};\text{ok} \cdot P_2)' \\
& \quad \text{by (rel-auto, meson+)} \\
& \text{thus } ?\text{thesis} \\
& \quad \text{by (simp-all add: ex-unrest ex-plus Healthy-if assms)} \\
& \text{qed} \\
\end{align*}

lemma \( \text{R1-design-refine-RR}: \)
\begin{align*}
& \text{assumes } P_1 \text{ is } RR P_2 \text{ is } RR Q_1 \text{ is } RR Q_2 \text{ is } RR \\
& \text{shows } R_1(P_1 + P_2) \subseteq R_1(Q_1 + Q_2) \iff 'P_1 \Rightarrow Q_1' \land 'P_1 \land Q_2 \Rightarrow P_2' \\
& \text{by (simp add: R1-design-refine assms unrest closure)} \\
\end{align*}

lemma \( \text{RHS-design-refine}: \)
\begin{align*}
& \text{assumes} \\
& P_1 \text{ is } R_2 c P_2 \text{ is } R_2 c Q_1 \text{ is } R_2 c Q_2 \text{ is } R_2 c \\
& \text{shows } R_s(P_1 + P_2) \subseteq R_s(Q_1 + Q_2) \iff 'P_1 \Rightarrow Q_1' \land 'P_1 \land Q_2 \Rightarrow P_2' \\
& \text{proof} -- \text{have } R_s(R_2 c(P_1 + P_2)) \subseteq R_s(R_3 h(R_2 c(Q_1 + Q_2))) \\
& \quad \text{by (simp add: R2c-R3h-commute RHS-def)} \\
& \quad \text{also have } ... \iff R_1(R_3 h(P_1 + P_2)) \subseteq R_1(R_3 h(Q_1 + Q_2)) \\
&\quad \text{by (simp add: Healthy-if R2c-design assms)} \\
&\quad \text{also have } ... \iff R_1(R_3 h(P_1 + P_2))[false/swait] \subseteq R_1(R_3 h(Q_1 + Q_2))[false/swait] \\
&\quad \text{by (rel-auto, meson+)} \\
&\quad \text{also have } ... \iff R_1(P_1 + P_2)[false/swait] \subseteq R_1(Q_1 + Q_2)[false/swait] \\
&\quad \text{by (rel-auto)} \\
&\quad \text{also have } ... \iff R_1(P_1 + P_2) \subseteq R_1(Q_1 + Q_2) \\
&\quad \text{by (simp add: usubst assms unrest closure)} \\
&\quad \text{also have } ... \iff 'P_1 \Rightarrow Q_1' \land 'P_1 \land Q_2 \Rightarrow P_2' \\
&\quad \text{by (simp add: R1-design-refine assms)} \\
&\text{finally show } ?\text{thesis} . \\
&\text{qed} \\
\end{align*}

lemma \( \text{srdes-refine-intro}: \)
\begin{align*}
& \text{assumes } 'P_1 \Rightarrow P_2', 'P_1 \land Q_2 \Rightarrow Q_1' \\
&\text{25} \\
\end{align*}
3.5 Distribution laws

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

**by** (metis RHS-inf design-choice)

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

**by** (metis RHS-sup design-inf)

**lemma** RHS-design-USUP:

assumes $A \neq \emptyset$

shows $(\bigsqcap_i i \in A \cdot R_s(P(i) \vdash Q(i))) = R_s((\bigsqcap_i i \in A \cdot P(i)) \vdash (\bigsqcap_i i \in A \cdot Q(i)))$

**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 ° 65) where

  [upred-defs]: $P \circ Q = (P \circ \$wait' \circ Q)$

**lemma** wait’-cond-unrest [unrest]:

  $[\text{out-var wait} \triangleright \alpha \cdot x : x \not\in P; x \not\in Q ] \Longrightarrow x \not\in (P \circ Q)$

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

**lemma** wait’-cond-subst [subsubst]:

  $(\$wait' \triangleright \sigma \Rightarrow (P \circ Q) = (\sigma \triangleright P) \circ (\sigma \triangleright Q)$

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

**lemma** wait’-cond-left-false: false $\circ P = (\neg \$wait' \land P)$

**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 seq-or-disj, rel-blast)

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

**by** (rel-auto)

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

**by** (rel-auto)

**lemma** wait’-cond-idem: $P \circ P = P$

**by** (rel-auto)

**lemma** wait’-cond-conj-exchange:

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

end
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: \( (P \circ Q)[\text{true}]/\text{wait} \circ Q \) = \( P \circ Q \)
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]:
\[ [ P \text{ is } R1 \text{; } Q \text{ is } R1 \implies P \circ Q \text{ is } R1 \]
by (simp add: Healthy-def R1-wait'-cond)

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

4.2 Export laws

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

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

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

**lemma** RH-design-post-R2s: \( \mathbf{R}(P \rightarrow Q \circ R2s(R)) = \mathbf{R}(P \rightarrow Q \circ R) \)
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 \( \text{npre}_s \equiv [\text{ok} \mapsto_s \text{true}, \text{ok}' \mapsto_s \text{true}, \text{wait} \mapsto_s \text{false}, \text{wait}' \mapsto_s \text{false}] \)

definition \( \text{upred-defs} \): \( \text{pre}_R(P) \equiv \text{pre}_s \uparrow P \)
definition \( \text{upred-defs} \): \( \text{cmt}_R(P) = R1(\text{cmt}_s \uparrow P) \)
definition \( \text{upred-defs} \): \( \text{peri}_R(P) = R1(\text{peri}_s \uparrow P) \)
definition \( \text{upred-defs} \): \( \text{post}_R(P) = R1(\text{post}_s \uparrow P) \)

4.3.2 Unrestriction laws

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

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

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

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

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

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

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

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

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

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

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

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

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

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

4.3.3 Substitution laws

lemma pres-design: pres \dagger (P \vdash Q) = (\neg pres \dagger P)
by (simp add: design_def pres_R_def usubst)

lemma peris-design: peris \dagger (P \vdash Q \circ R) = peris \dagger (P \Rightarrow Q)
by (simp add: design_def usubst wait'-cond-def)

lemma posts-design: posts \dagger (P \vdash Q \circ R) = posts \dagger (P \Rightarrow R)
by (simp add: design_def usubst wait'-cond-def)

lemma cmtss-design: cmtss \dagger (P \vdash Q) = cmtss \dagger (P \Rightarrow Q)
by (simp add: design_def usubst wait'-cond-def)

lemma pres-R1 [usubst]: pres \dagger R1(P) = R1(pres \dagger P)
by (simp add: R1_def usubst)

lemma pres-R2c [usubst]: pres \dagger R2c(P) = R2c(pres \dagger P)
by (simp add: R2c_def R2s_def usubst)

lemma peris-R1 [usubst]: peris \dagger R1(P) = R1(peris \dagger P)
by (simp add: R1_def usubst)

lemma peris-R2c [usubst]: peris \dagger R2c(P) = R2c(peris \dagger 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-true:
pre_R(P[false/$wait]) = pre_R(P)
by (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]:
$\wait' \subseteq pre_R(P) \implies$ $\wait' \subseteq pre_R (SRD P)$
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 rea-cmt-RHS-design)

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

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

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

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

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

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

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

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

lemma R1-R2c-post-RHS:
assumes $P$ is SRD
shows $R1(R2c(\text{post}_R(P))) = \text{post}_R(P)$
by (metis R1-R2s-R2c R1-R2s-R2c 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 $\implies \text{pre}_R(P)$ is R2c
by (simp add: Healthy-def R2c-idem SRD-reactive-design rea-pre-RHS-design)

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

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

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

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

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

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

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

lemma trace-ident-left-periR:
($\text{str}^\prime = u \text{str} \land [H]_R$) ;; $\text{peri}_R(P) = \text{peri}_R(P)$
by (rel-auto)

lemma trace-ident-left-postR:
($\text{str}^\prime = u \text{str} \land [H]_R$) ;; $\text{post}_R(P) = \text{post}_R(P)$
by (rel-auto)
lemma \( \text{trace-ident-right-postR}: \)
\( \text{post}_R(P) \vdash (\$tr' = _u \$tr \land [\text{II}]_R) = \text{post}_R(P) \)
by (rel-auto)

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

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

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

4.3.5 Calculation laws

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

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

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

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

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

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

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

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

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

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

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

33
lemma periR-srdes-skip [rdes]: periR(II_R) = false
by (rel-auto)

lemma postR-srdes-skip [rdes]: post_R(II_R) = (\$tr' =_u \$tr \land [II]_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 \cdot j) = (\bigcup (i,j) \in A \cdot pre_R(P i \cdot j))
by (rel-auto)

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

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

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

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

lemma periR-inf [rdes]: peri\(_R\)(P ∩ Q) = (peri\(_R\)(P) ∨ peri\(_R\)(Q))
  by (rel-auto)

lemma postR-inf [rdes]: post\(_R\)(P ∩ Q) = (post\(_R\)(P) ∨ post\(_R\)(Q))
  by (rel-auto)

lemma preR-SUP [rdes]: pre\(_R\)(\bigcup A) = (∀ P∈A · pre\(_R\)(P))
  by (rel-auto)

lemma periR-SUP [rdes]: A ≠ {} ⇒ peri\(_R\)(\bigcup A) = (∃ P∈A · peri\(_R\)(P))
  by (rel-auto)

lemma postR-SUP [rdes]: A ≠ {} ⇒ post\(_R\)(\bigcup A) = (∃ P∈A · post\(_R\)(P))
  by (rel-auto)

4.4 Formation laws

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

lemma Chaos-tri-def [rdes-def]: Chaos = R\(_s\)(false ⊥ false)
  by (simp add: Chaos-def design-false-pre)

lemma Miracle-tri-def [rdes-def]: Miracle = R\(_s\)(true\(_r\), false ⊥ false)
  by (simp add: Miracle-def R1-design-R1-pre wait'-cond-idem)

lemma RHS-tri-design-form:
  assumes P\(_1\) is RR P\(_2\) is RR P\(_3\) is RR
  shows R\(_s\)(P\(_1\) ⊢ P\(_2\) ⊓ P\(_3\)) = (II\(_R\) ↛ $wait ⇔ ((\$ok ∧ P\(_1\)) ⇒ (\$ok ∧ (P\(_2\) ⊓ P\(_3\))))
proof –
  have R\(_s\)(RR(P\(_1\)) ⊢ RR(P\(_2\)) ⊓ RR(P\(_3\))) = (II\(_R\) ↛ $wait ⇔ ((\$ok ∧ RR(P\(_1\))) ⇒ (\$ok ∧ RR(P\(_2\)) ⊓ RR(P\(_3\)))))
  apply (rel-auto) using minus-zero-eq by blast
  thus ?thesis
  by (simp add: Healthy-if assms)
qed

lemma RHS-design-pre-post-form:
  R\(_s\)((¬ P\(_f\), f) ⊢ P\(_f\), f) = R\(_s\)(pre\(_R\)(P) ⊢ cmt\(_R\)(P))
proof –
  have R\(_s\)((¬ P\(_f\), f) ⊢ P\(_f\), f) = R\(_s\)((¬ P\(_f\), f)[true/\$ok] ⊢ P\(_f\), f)[true/\$ok]
  by (simp add: design-subst-ok)
  also have ... = R\(_s\)(pre\(_R\)(P) ⊢ cmt\(_R\)(P))
  by (simp add: pre\(_R\)-def cmt\(_R\)-def asubst, rel-auto)
  finally show ?thesis .
qed

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

lemma SRD-reactive-design-alt:
  assumes P is SRD
  shows R\(_s\)(pre\(_R\)(P) ⊢ cmt\(_R\)(P)) = P
proof –
  have R\(_s\)(pre\(_R\)(P) ⊢ cmt\(_R\)(P)) = R\(_s\)((¬ P\(_f\), f) ⊢ P\(_f\), f)
by (simp add: RHS-design-pre-post-form)
thus ?thesis
by (simp add: SRD-reactive-design assms)
qed

lemma SRD-reactive-tri-design-lemma:
SRD(P) = Rs((¬ P[T]T ) ⊨ P[T][true/\wait\ ’] ⊨ P[T][false/\wait\ ’])
by (simp add: SRD-RH-design-form wait'-cond-split)

lemma SRD-as-reactive-tri-design:
SRD(P) = Rs(pre R(P) ⊨ peri R(P) ◦ post R(P))
proof –
have SRD(P) = Rs((¬ P[T]T ) ⊨ P[T][true/\wait\ ’] ⊨ P[T][false/\wait\ ’])
by (simp add: SRD-RH-design-form wait'-cond-split)
also have ... = Rs(pre R(P) ⊨ peri R(P) ◦ post R(P))
apply (simp add: usubst)
apply (subst design-subst-ok-ok [THEN sym])
apply (simp add: pre R-def peri R-def post R-def usubst unrest)
apply (rel-auto)
done
finally show ?thesis .
qed

lemma SRD-reactive-tri-design:
assumes P is SRD
shows Rs(pre R(P) ⊨ peri R(P) ◦ post R(P)) = P
by (metis Healthy-if SRD-as-reactive-tri-design assms)

lemma SRD-elimi [RD-elim]: [ P is SRD; Q(Rs(pre R(P) ⊨ peri R(P) ◦ post R(P))) ] ⇒ Q(P)
by (simp add: SRD-reactive-tri-design)

lemma RHS-tri-design-is-SRD [closure]:
assumes $ok‘ ∉ P $ok‘ ∉ Q $ok‘ ∉ R
shows Rs(P ⊨ Q ◦ R) is SRD
by (rule RHS-design-is-SRD, simp-all add: unrest assms)

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

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

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

4.4.1 Order laws

lemma preR-antitone: P ⊆ Q ⇒ pre R(Q) ⊆ pre R(P)
by (rel-auto)
lemma periR-monotone: $P \subseteq Q \implies \peri_R(P) \subseteq \peri_R(Q)$
by (rel-auto)

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

4.5 Composition laws

theorem RH-tri-design-composition
assumes $\leq P \leq Q \leq R \leq S \leq T \leq U$
shows $\leq P \leq Q \leq R \leq S \leq T \leq U$

proof

have 1: $(\leq P \leq Q \leq R \leq S \leq T \leq U)$
by (metis no-types, hide-lams)

have 2: $(\leq P \leq Q \leq R \leq S \leq T \leq U)$
by (metis false-alt-def)

finally show \$thesis 

qed

moreover have $(\leq P \leq Q \leq R \leq S \leq T \leq U)$
by (metis no-types, lifting)

also have $(\leq P \leq Q \leq R \leq S \leq T \leq U)$
by (simp add: lift-des-skip-dr-unit-unrest unrest)

finally show \$thesis 

qed
ultimately show \texttt{thesis}
by (simp add: R2s-wait'-cond R1-wait'-cond wait'-cond-seq)

qed

show \texttt{thesis}
apply (subst RH-design-composition)
apply (simp-all add: assms)
apply (simp add: assoc RS-conj RS-simp RS-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 \( (R1(P \vdash Q) ;; R1(R \vdash S)) = R1((\neg P) wp \; false \land Q \; wp \; R) \vdash (Q ;; S) \)
apply (subst R1-design-composition)
apply (simp-all add: assms unrest Healthy-if closure wp)

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

thesis RHS-tri-design-composition:
assumes \( \$ok' \in P \) \( \$ok' \in Q1 \) \( \$ok \in Q \) \( \$ok \in S1 \) \( \$ok \in S2 \)
\( \$wait \in R \) \( \$wait \in Q2 \) \( \$wait \in S1 \) \( \$wait \in S2 \)
shows \( (R_n(P \vdash Q_1 \circ Q_2) ;; R_n(R \vdash S_1 \circ S_2)) = \)
\( R_n((\neg (R1 \; \neg R2s \; P) ;; R1 \; true) \land \neg (R1(R2s \; Q_2) ;; R1 \; \neg R2s \; R)) \vdash \)
\( ((\exists \; st' \cdot Q_1) \lor (R1 \; (R2s \; Q_2) ;; R1 \; (R2s \; S_1)) \circ ((R1 \; (R2s \; Q_2) ;; R1 \; (R2s \; S_2)))) \)
proof
have 1: \( (R1 \; (R2s \; (Q_1 \circ Q_2)) \land \neg \$wait') ;; R1 \; \neg R2s \; R) \)
\( (\neg ((R1 \; (R2s \; Q_2) \land \neg \$wait') ;; R1 \; \neg R2s \; R)) \)
by (metis (no-types, hide-lams) R1-extend-cond R2s-conj R2s-not R2s-wait' wait'-cond-false)

have 2: \( (R1 \; (R2s \; (Q_1 \circ Q_2)) ;; ((\exists \; st \cdot [H]D) \circ \$wait \circ R1 \; (R2s \; (S_1 \circ S_2))) = \)
\( ((\exists \; st' \cdot R1 \; (R2s \; Q_1)) \lor (R1 \; (R2s \; Q_2) ;; R1 \; (R2s \; S_1)) \circ (R1 \; (R2s \; Q_2) ;; R1 \; (R2s \; S_2))) \)
proof
have \( (R1 \; (R2s \; Q_1)) ;; \$wait \land ((\exists \; st \cdot [H]D) \circ \$wait \circ R1 \; (R2s \; S_1) \circ R1 \; (R2s \; S_2))) = \)
\( (\exists \; st' \cdot ((R1 \; (R2s \; Q_1)) \land \$wait')) \)

proof
have \( (R1 \; (R2s \; Q_1)) ;; \$wait \land ((\exists \; st \cdot [H]D) \circ \$wait \circ R1 \; (R2s \; S_1) \circ R1 \; (R2s \; S_2))) = \)
\( (R1 \; (R2s \; Q_1)) ;; \$wait \land ((\exists \; st \cdot [H]D)) \)
by (rel-auto, blast+)

also have ... = \( (R1 \; (R2s \; Q_1)) ;; (\exists \; st \cdot [H]D) \land \$wait' \)
by (rel-auto)

also from \( \text{assms(2)} \) have ... = \( (\exists \; st' \cdot ((R1 \; (R2s \; Q_1)) \land \$wait')) \)
by (rel-auto, blast)

finally show \texttt{thesis} .

qed
moreover have \((R1 \ (R2s \ Q2) ;; (\neg \ $wait \wedge ((\exists \ $st \cdot [II]_D) \wedge \ $wait \Rightarrow R1 \ (R2s \ S1) \wedge R1 \ (R2s \ S2)))\))

\[= ((R1 \ (R2s \ Q2)) ;; (R1 \ (R2s \ S1) \wedge R1 \ (R2s \ S2)))\]

proof -

have \((R1 \ (R2s \ Q2) ;; (\neg \ $wait \wedge ((\exists \ $st \cdot [II]_D) \wedge \ $wait \Rightarrow R1 \ (R2s \ S1) \wedge R1 \ (R2s \ S2)))\))

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

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

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

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

also have \(\ldots = ((R1 \ (R2s \ Q2)) ;; (R1 \ (R2s \ S1) \wedge R1 \ (R2s \ S2)))\)

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

finally show \(?thesis\).

qed

moreover

have \((\langle R1 \ (R2s \ Q1) \wedge \ $wait' \rangle \vee ((R1 \ (R2s \ Q2)) ;; (R1 \ (R2s \ S1) \wedge R1 \ (R2s \ S2)))\))

\[= (R1 \ (R2s \ Q1) \vee (R1 \ (R2s \ Q2) ;; R1 \ (R2s \ S1)) \wedge ((R1 \ (R2s \ Q2)) ;; R1 \ (R2s \ S2)))\]

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

ultimately show \(?thesis\)

by (simp add: R2s-wait'-cond R1-wait'-cond wait'-cond-seq ex-cond-constr-right unrest)

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

qed

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

by (rel-auto, blast, meson)

show \(?thesis\)

apply (subst RHS-design-composition)

apply (simp-all add: assms)

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

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

apply (simp add: 1 2 3)

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

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

done

qed

theorem \(\text{RHS-tri-design-composition-wp:}\)

assumes \(\text{ Sok'} \notin P \ $ok' \notin Q1 \ $ok' \notin Q2 \ $ok \notin R \ $ok \notin S1 \ $ok \notin S2 \)

\(\text{ $wait \notin R \ $wait' \notin Q2 \ $wait \notin S1 \ $wait \notin S2 \)

\(P \text{ is } R2c \ Q1 \text{ is } R1 \ Q1 \text{ is } R2c \ Q2 \text{ is } R1 \ Q2 \text{ is } R2c \)

\(R \text{ is } R2c \ S1 \text{ is } R1 \ S1 \text{ is } R2c \ S2 \text{ is } R1 \ S2 \text{ is } R2c \)

shows \(\text{ R}_4(P \vdash Q1 \circ Q2) ;; \text{ R}_4(R \vdash S1 \circ S2) =

\text{ R}_4(((\neg R) \ wp_r \ false \wedge Q2 \ wp_r \ R) \vdash ((\exists \ $st' \cdot Q1) \Rightarrow (Q2 :: S1)) \circ (Q2 :: S2)))\) \(\text{is } \ ?lhs = \ ?rhs\)

proof -

have \(?lhs = \text{ R}_4((\neg R1 \ (\neg P) ;; R1 \ true \wedge \neg Q2 ;; R1 \ (\neg R)) \vdash ((\exists \ $st' \cdot Q1) \Rightarrow (Q2 :: S1) \circ Q2 :: S2))\)

\(39\)
by (simp add: RHS-tri-design-composition assms Healthy-if R2c-true R2s-assms assms(11,16))
also have \ldots = \textit{ths}
by (rel-auto)
finally show \textit{thesis}
qed

\textbf{theorem} RHS-tri-design-composition-RR-up:
\textbf{assumes} \(P \text{ is RR } Q_1 \text{ is RR } Q_2 \text{ is RR}
R \text{ is RR } S_1 \text{ is RR } S_2 \text{ is RR}
\textbf{shows} \(R_s((\neg_r P) w_p, false \land Q_2 w_p, R) \vdash ((\exists \textit{st} \cdot Q_1) \cap (Q_2 :: S_1)) \circ (Q_2 :: S_2)) \textit{(is \ ?ths = \ ?rhs)}
by (simp add: RHS-tri-design-composition-up add: closure assms unrest RR-implies-R2c)

\textbf{lemma} RHS-tri-normal-design-composition:
\textbf{assumes} \(\textit{ok' \ not \ Q} \text{ is RR } Q_1 \text{ is RR } Q_2 \text{ is RR}
\textbf{shows} \(R_s((\neg_r P) w_p, false \land Q_2 w_p, R) \vdash ((\exists \textit{st} \cdot Q_1) \cap (Q_2 :: S_1)) \circ (Q_2 :: S_2)) \textit{(is \ ?ths = \ ?lhs)}
by (simp add: RHS-tri-design-composition-wp add: assms unrest R2s-true R1-false R2s-false)

\textbf{lemma} RHS-tri-normal-design-composition' [rdes-def]:
\textbf{assumes} \(P \text{ is RC } Q_1 \text{ is RR } \textit{st} \cdot Q_1 \text{ is RR } R \text{ is RR } S_1 \text{ is RR } S_2 \text{ is RR}
\textbf{shows} \(R_s((P \lor Q_2 w_p, R) \vdash (Q_1 \lor (Q_2 :: S_1)) \circ (Q_2 :: S_2)) \textit{(is \ ?thesis)}
by (simp add: R1-negate-R1 R1-sequ-tri-design, rel-auto)

\textbf{lemma} RHS-tri-design-right-unit-lemma:
\textbf{assumes} \(\textit{ok' \ not \ Q} \text{ is RR } Q_1 \text{ is RR } Q_2 \text{ is RR}
\textbf{shows} \(R_s(P \lor Q \circ R) \vdash (R_1 \vdash \text{true} \vdash (\exists \textit{st} \cdot Q) \vdash (R_1 \vdash \text{false} \vdash ([I] R))) \textit{(is \ ?thesis)}
by (simp add: Rdes-skip-tri-design, rel-auto)

\textbf{lemma} RHS-tri-design-right-unit-lemma:
\textbf{assumes} \(\textit{ok' \ not \ Q} \text{ is RR } Q_1 \text{ is RR } Q_2 \text{ is RR}
\textbf{shows} \(R_s(P \land Q \circ R) \vdash (R_1 \vdash \text{true} \vdash (\exists \textit{st} \cdot Q) \vdash (R_1 \vdash \text{false} \vdash ([I] R))) \textit{(is \ ?thesis)}
by (simp add: Rdes-skip-tri-design, rel-auto)

\textbf{lemma} RHS-tri-design-right-unit-lemma:
\textbf{assumes} \(\textit{ok' \ not \ Q} \text{ is RR } Q_1 \text{ is RR } Q_2 \text{ is RR}
\textbf{shows} \(R_s(P \land Q \circ R) \vdash (R_1 \vdash \text{true} \vdash (\exists \textit{st} \cdot Q) \vdash (R_1 \vdash \text{false} \vdash ([I] R))) \textit{(is \ ?thesis)}
by (simp add: Rdes-skip-tri-design, rel-auto)
also have ... = R_s ((¬ R1 (¬ R2s P) ;; R1 true) ⊬ (∃ $st' ⋄ Q) ∩ R1 (R2s R))

proof –
  from assms(3,4) have (R1 (R2s R) ;; R1 (R2s ($tr' = u $tr ⋄ [I]\_R))) = R1 (R2s R)
  by (rel-auto, metis (no-types, lifting) minus-zero-eq, meson order-reflect-trace-class.diff-cancel)
thus ?thesis 
  by simp
qed

also have ... = Rs((¬ P ;; R1 true) ⊬ ((∃ $st' ⋄ Q) ∩ R))
  by (metis (no-types, lifting) R1-R2s-R1-true-lemma R1-R2s-R2c R2c-not RHS-design-R2c-pre RHS-design-neg-R1-pre RHS-design-post-R1 RHS-design-post-R2s)
also have ... = Rs((¬r (¬ P ;; true_r)) ⊬ ((∃ $st' ⋄ Q) ∩ R))
  by (rel-auto)
finally show ?thesis .
qed

lemma SRD-composition-up:
assumes P is SRD Q is SRD
shows (P ;; Q) = Rs (((¬ r prec_R P) wp_r false ∧ post_R P wp_r prec_R Q) ⊬
(∃ $st' ⋄ peri_R P) ∨ (post_R P ;; peri_R Q)) ∩ (post_R P ;; post_R Q))
(is ?lhs = ?rhs)
proof –
  have (P ;; Q) = (Rs(perc_R(P) ⊬ peri_R(P) ⋄ post_R(P)) ;; Rs(perc_R(Q) ⊬ peri_R(Q) ⋄ post_R(Q)))
  by (simp add: SRD-reactive-tri-design assms(1) assms(2))
also from assms
have ... = ?rhs
  by (simp add: RHS-likely-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 Rs(P_1 ⊬ P_2 ⋄ P_3) ⊆ Rs(Q_1 ⊬ Q_2 ⋄ Q_3) ↔ 'P_1 ⇒ Q_1' ⋄ 'P_1 ∧ Q_2 ⇒ P_2 ⋄ 'P_1 ∧ Q_3 ⇒ P_3'  
(is ?lhs = ?rhs)
proof –
  have ?lhs ↔ 'P_1 ⇒ Q_1' ⋄ 'P_1 ∧ Q_2 ⇒ P_2 ⋄ P_3'
  by (simp add: RHS-design-refine assms closure RR-implies-R2c unrest ex-unrest)
also have ... ↔ 'P_1 ⇒ Q_1' ⋄ '(P_1 ∧ Q_2) ⋄ (P_1 ∧ Q_3) ⇒ P_2 ⋄ P_3' ⋄ 'P_1 ∧ Q_3 ⇒ P_2 ⋄ P_3'[true/$\$wait'] ⋄ '((P_1 ∧ Q_2) ⋄ (P_1 ∧ Q_3) ⇒ P_2 ⋄ P_3)[false/$\$wait']';
  by (rel-auto, metis)
also have ... ↔ ?rhs
  by (simp add: usubst unrest assms)
finally show ?thesis .
qed

lemma srdes-tri-refine-intro:
assumes 'P_1 ⇒ P_2'; 'P_1 ∧ Q_2 ⇒ Q_1' ⋄ 'P_1 ∧ R_2 ⇒ R_1'
shows Rs(P_1 ⊬ Q_1 ⋄ R_1) ⊆ Rs(P_2 ⊬ Q_2 ⋄ R_2)
using assms
by (rule-tac srdes-refine-intro, simp-all, rel-auto)

lemma srdes-tri-eq-intro:
assumes \( P_1 = Q_1 \ P_2 = Q_2 \ P_3 = Q_3 \)
shows \( \mathbf{R_s}(P_1 \triangledown P_2 \triangledown P_3) = \mathbf{R_s}(Q_1 \triangledown Q_2 \triangledown Q_3) \)
using assms by (simp)

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

lemma \( \text{SRD-peri-under-pre} \):
assumes \( P \) is SRD \$wait\$ \# \( \pre_P(P) \)
shows \( \pre_P(P) \Rightarrow \peri_P(P) = \peri_P(P) \)
proof -
  have \( \peri_P(P) = \peri_P(\mathbf{R_s}(\pre_P(P) \triangledown \peri_P(P) \land \post_P(P))) \)
    by (simp add: SRD-reactive-tri-design assms)
  also have \( \ldots = (\pre_P P \Rightarrow \peri_P P) \)
    by (simp add: rea-pre-RHS-design rea-peri-RHS-design assms)
  finally show \( ?thesis \).
qed

lemma \( \text{SRD-post-under-pre} \):
assumes \( P \) is SRD \$wait\$ \# \( \pre_P(P) \)
shows \( \pre_P(P) \Rightarrow \post_P(P) = \post_P(P) \)
proof -
  have \( \post_P(P) = \post_P(\mathbf{R_s}(\pre_P(P) \triangledown \peri_P(P) \land \post_P(P))) \)
    by (simp add: SRD-reactive-tri-design assms)
  also have \( \ldots = (\pre_P P \Rightarrow \post_P P) \)
    by (simp add: rea-post-RHS-design rea-post-RHS-design assms)
  finally show \( ?thesis \).
qed

lemma \( \text{SRD-refine-intro} \):
assumes \( P \) is SRD \( Q \) is SRD
'\( \pre_P(P) \Rightarrow \pre_Q(Q) \)''\( \pre_P(P) \land \peri_P(Q) \Rightarrow \peri_Q(Q) \)'\( \pre_P(P) \land \post_P(Q) \Rightarrow \post_Q(Q) \)'
shows \( P \subseteq Q \)
by (metis SRD-reactive-tri-design assms(1) assms(2) assms(3) assms(4) assms(5) srdes-tri-refine-intro)

lemma \( \text{SRD-refine-intro} \):
assumes \( P \) is SRD \( Q \) is SRD
'\( \pre_P(P) \Rightarrow \pre_Q(Q) \)''\( \pre_P(P) \land \peri_P(Q) \subseteq (\pre_P(P) \land \peri_Q(Q)) \)\( \post_P(P) \subseteq (\pre_P(P) \land \post_Q(Q)) \)
shows \( P \subseteq Q \)
using assms by (rule-tac SRD-refine-intro, simp-all add: refBy-order)

lemma \( \text{SRD-eq-intro} \):
assumes \( P \) is SRD \( Q \) is SRD
\( \pre_P(P) = \pre_Q(Q) \)\( \peri_P(P) = \peri_Q(Q) \)\( \post_P(P) = \post_Q(Q) \)
shows \( P = Q \)
by (metis SRD-reactive-tri-design assms)
4.7 Closure laws

**lemma** SRD-srdes-skip [closure]: \( II_R \) is SRD

by (simp add: srdes-skip-def RHS-design-is-SRD unrest)

**lemma** SRD-seqr-closure [closure]:

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

shows \( (P ;; Q) \) is SRD

proof

have \( (P ;; Q) = R_s ((\neg \ pre_R P) \ wp false \land post_R P \ wp pre_R Q) \vdash \)

\((\exists \ $st' \cdot peri_R P) \lor post_R P ;; peri_R Q) \circ post_R P ;; post_R Q) \)

by (simp add: SRD-composition-wp assms (1) assms (2))

also have \... is SRD

by (rule RHS-design-is-SRD, simp-all add: wp-rea-def unrest)

finally show \?thesis .

qed

**lemma** SRD-power-Suc [closure]: \( P \) is SRD \(=\Rightarrow\) \( P^\ast (Suc \ n) \) 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 \( A \subseteq [\text{SRD}]_H A \neq \{\} \)

shows \( (\bigsqcap A) \) is SRD

proof

have SRD \( (\bigsqcap A) = (\bigsqcap (\text{SRD} 'A)) \)

by (simp add: ContinuousD SRD-Continuous assms (2))

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

by (simp only: Healthy-carrier-image assms)

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

qed

4.8 Distribution laws

**lemma** RHS-tri-design-choice [rdes-def]:

\( R_s (P_1 \vdash P_2 \circ P_3) \cap R_s (Q_1 \vdash Q_2 \circ Q_3) = R_s ((P_1 \land Q_1) \vdash (P_2 \lor Q_2) \circ (P_3 \lor Q_3)) \)

apply (simp add: RHS-design-choice)

apply (rule cong[of \( R_s, R_s \)])

apply (simp)

apply (rel-auto)

done

**lemma** RHS-tri-design-sup [rdes-def]:

43
\[ R_s(P_1 \vdash P_2 \circ P_3) \cup R_s(Q_1 \vdash Q_2 \circ Q_3) = R_s((P_1 \lor Q_1) \vdash ((P_1 \Rightarrow P_2) \land (Q_1 \Rightarrow Q_2)) \circ ((P_1 \Rightarrow P_3) \land (Q_1 \Rightarrow Q_3))) \]

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

**lemma** RHS-tri-design-conj [rdes-def]:

\[ (R_s(P_1 \vdash P_2 \circ P_3) \land R_s(Q_1 \vdash Q_2 \circ Q_3)) = R_s((P_1 \lor Q_1) \vdash ((P_1 \Rightarrow P_2) \land (Q_1 \Rightarrow Q_2)) \circ ((P_1 \Rightarrow P_3) \land (Q_1 \Rightarrow Q_3))) \]

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

**lemma** SRD-UINF [rdes-def]:

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

shows \( \prod A = \bigwedge P \in A \cdot \text{pre}_R(P) \vdash (\bigvee P \in A \cdot \text{peri}_R(P)) \circ (\bigvee P \in A \cdot \text{post}_R(P)) \)

**proof**

\[ \begin{aligned} &\text{have } \prod A = R_s((\bigwedge P \in A \cdot \text{pre}_R(P)) \vdash \text{peri}_R(\prod A) \circ \text{post}_R(\prod A)) \\
&\quad \text{by (metis SRD-as-reactive-tri-design assms srdes-hcond-def)} \\
&\quad \text{srdes-theory-continuous,healthy-inf srdes-theory-continuous,healthy-inf-def)} \\
&\text{also have } ... = R_s((\bigwedge P \in A \cdot \text{pre}_R(P)) \vdash (\bigvee P \in A \cdot \text{peri}_R(P)) \circ (\bigvee P \in A \cdot \text{post}_R(P))) \\
&\quad \text{by (simp add: preR-INF periR-INF postR-INF assms)} \\
&\text{finally show } \text{thesis} . \\
\end{aligned} \]

qed

**lemma** RHS-tri-design-USUP [rdes-def]:

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

shows \( \prod i \in A \cdot R_s(P(i) \circ Q(i) \circ R(i)) = R_s((\bigwedge i \in A \cdot P(i)) \vdash (\prod i \in A \cdot Q(i)) \circ (\prod i \in A \cdot R(i))) \)

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

**lemma** SRD-UINF-mem:

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

shows \( \prod i \in A \cdot \text{P}(i) = R_s((\bigwedge i \in A \cdot \text{pre}_R(P(i)) \vdash (\bigvee i \in A \cdot \text{peri}_R(P(i)) \circ (\bigvee i \in A \cdot \text{post}_R(P(i)))) \)

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

**proof**

\[ \begin{aligned} &\text{have } ?\text{lhs} = (\prod (P \cdot A)) \\
&\quad \text{by (rel-auto)} \\
&\text{also have } ... = R_s((\bigwedge Pa \in P \cdot A \cdot \text{pre}_R(Pa)) \vdash (\prod Pa \in P \cdot A \cdot \text{peri}_R(Pa)) \circ (\prod Pa \in P \cdot A \cdot \text{post}_R(Pa)) \\
&\quad \text{by (subst rdes-def, simp-all add: assms image-subsetI)} \\
&\text{also have } ... = ?\text{rhs} \\
&\quad \text{by (rel-auto)} \\
&\text{finally show } \text{thesis} . \\
\end{aligned} \]

qed

**lemma** RHS-tri-design-UINF-ind [rdes-def]:

\( (\prod i \cdot R_s(P_1(i) \circ P_2(i) \circ P_3(i))) = R_s((\bigwedge i \cdot P_1(i)) \vdash (\bigvee i \cdot P_2(i)) \circ (\bigvee i \cdot P_3(i))) \)

by (rel-auto)

**lemma** cond-srea-form [rdes-def]:

\( R_s(P \vdash Q_1 \circ Q_2) \circ \bigcirc R \circ R_s(R \vdash S_1 \circ S_2) = R_s((P \circ \bigcirc R) \vdash Q_1 \circ \bigcirc S_1) \circ (Q_2 \circ \bigcirc S_2) \)

**proof**

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

lemma SRD-cond-srea [closure]:
  assumes \( P \) is SRD \( Q \) is SRD
  shows \( P \triangleleft b \triangleright R Q \) is SRD
proof –
  have \( P \triangleleft b \triangleright R Q = R_s((\mathit{pre}_R(P) \vdash \mathit{peri}_R(P) \circ \mathit{post}_R(P)) \triangleleft b \triangleright R_s(\mathit{peri}_R(Q) \circ \mathit{post}_R(Q))) \)
    by (simp add: SRD-reactive-tri-design assms)
  also have \( ... = R_s((\mathit{pre}_R P \triangleleft b \triangleright R S_1) \triangleright (\mathit{peri}_R P \triangleleft b \triangleright R S_2)) \)
    by (simp add: cond-srea-form)
  also have \( ... \) is SRD
    by (simp add: RHS-tri-design-is-SRD lift-cond-srea-def unrest)
finally show \(?thesis\).
qed

4.9 Algebraic laws

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

lemma skip-srea-self-unit [simp]:
\( \mathit{II} \triangleleft R \triangleright \mathit{II} \triangleleft R \triangleright = \mathit{II} \triangleleft R \triangleright \)
by (simp add: SRD-left-unit closure)

lemma SRD-right-unit-tri-lemma:
  assumes \( P \) is SRD
  shows \( P \triangleright \mathit{II} \triangleleft R \triangleright = R_s(\mathit{true} \vdash \mathit{false}) \triangleright (\exists s \cdot \mathit{peri}_R P) \circ \mathit{post}_R P) \)
by (simp add: SRD-composition-wp closure rdes wp rpred trace-ident-right-postR SRD-reactive-tri-design assms)

lemma Miracle-left-zero:
  assumes \( P \) is SRD
  shows \( \mathit{Miracle} \triangleleft \mathit{II} \triangleright P = \mathit{Miracle} \)
proof –
  have \( \mathit{Miracle} \triangleleft \mathit{II} \triangleright P = R_s(\mathit{true} \vdash \mathit{false}) \triangleright R_s(\mathit{peri}_R(P) \vdash \mathit{cmt}_R(P)) \)
    by (simp add: Miracle-def SRD-reactive-design-alt assms)
  also have \( ... = R_s(\mathit{true} \vdash \mathit{false}) \)
    by (simp add: RHS-design-composition unrest R1-false R2s-false R2s-true)
  also have \( ... = \mathit{Miracle} \)
    by (simp add: Miracle-def)
finally show \(?thesis\).
qed

lemma Chaos-left-zero:
  assumes \( P \) is SRD
  shows \( \mathit{Chaos} \triangleright \mathit{II} \triangleleft R \triangleright P = \mathit{Chaos} \)
proof –
  have \( \mathit{Chaos} \triangleright \mathit{II} \triangleright P = R_s(\mathit{false} \vdash \mathit{true}) \triangleright R_s(\mathit{peri}_R(P) \vdash \mathit{cmt}_R(P)) \)

45
\begin{verbatim}
by (simp add: Chaos-def SRD-reactive-design-alt assms)
also have \ldots = \text{R}_a ((\neg R1 \true \land \neg (R1 \true \land \neg \text{wait}^\prime)) :: R1 (\neg R2s (\text{pre}_R P))) \vdash R1 \true :: (\exists \text{st} \cdot ([R1]_D) \triangle \text{wait} \triangleright R1 (R2s (\text{cm}_{\text{R}} P)))
by (simp add: RHS-design-composition unrest R2s-false R2s-true R1-false)
also have \ldots = \text{R}_a ((\text{false} \land \neg (R1 \true \land \neg \text{wait}^\prime)) :: R1 (\neg R2s (\text{pre}_R P))) \vdash R1 \true :: (\exists \text{st} \cdot ([R1]_D) \triangle \text{wait} \triangleright R1 (R2s (\text{cm}_{\text{R}} P)))
by (simp add: RHS-design-conj-neg-R1-pre)
also have \ldots = Chaos
by (simp add: design-false-pre)
also have \ldots = \text{R}_a (\text{false} \vdash \true)
by (simp add: design-def)
also have \ldots = Chaos
finally show \?thesis.
qed

lemma SRD-right-Chaos-tri-lemma:
assumes P is SRD
shows P :: Chaos = \text{R}_a ((\neg \text{pre}_R P) \text{ wp}_R \false \land \text{post}_R P \text{ wp}_R \false) \vdash (\exists \text{st}^\prime \cdot \text{peri}_R P) \circ \false
by (simp add: SRD-composition-wp closure rdes assms wp, rel-auto)

lemma SRD-right-Miracle-tri-lemma:
assumes P is SRD
shows P :: Miracle = \text{R}_a ((\neg \text{pre}_R P) \text{ wp}_R \false \vdash (\exists \text{st}^\prime \cdot \text{peri}_R P) \circ \false)
by (simp add: SRD-composition-wp closure rdes assms wp, rel-auto)

Stateful reactive designs are left unital

overloading
srdes-unit :: (SRDES, (\text{'s, \text{'t::trace, \text{'a}}) rthy) \text{uthy} \Rightarrow (\text{'s, \text{'t, \text{'a}}) hrel-rsp

begin
definition srdes-unit :: (SRDES, (\text{'s, \text{'t::trace, \text{'a}}) rthy) \text{uthy} \Rightarrow (\text{'s, \text{'t, \text{'a}}) hrel-rsp where
srdes-unit T = \text{H}_R

end

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

4.10 Recursion laws

lemma mono-srd-iter:
assumes mono \lambda X. \text{R}_a (\text{pre}_R (F X) \vdash \text{peri}_R (F X) \circ \text{post}_R (F X))
shows (\lambda X. \text{R}_a (\text{pre}_R (F X) \vdash \text{peri}_R (F X) \circ \text{post}_R (F X)))
is SRD
apply (rule monoI)
apply (rule srdes-right-refine-intro')
apply (meson assms(1) monoE preR-antitone utp-pred-laws.le-infI2)
apply (meson assms(1) monoE periR-monotone utp-pred-laws.le-infI2)
apply (meson assms(1) monoE postR-monotone utp-pred-laws.le-infI2)
done

lemma mu-srd-SRD:
assumes mono \mu X \cdot \text{R}_a (\text{pre}_R (F X) \vdash \text{peri}_R (F X) \circ \text{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)
\end{verbatim}
lemma mu-srd-iter:
  assumes mono F F ∈ [SRD]H → [SRD]H
  shows (µ X · Rs(preR(F(X)) ⊆ periR(F(X)) ⊆ postR(F(X)))) = F(µ X · Rs(preR(F(X)) ⊆ periR(F(X)) ⊆ postR(F(X))))
  apply (subst gfp-unfold)
  apply (simp add: mono-srd-iter assms)
  apply (subst SRD-as-reactive-tri-design[THEN sym])
  using Healthy-func assms (1) assms (2) mu-srd-SRD
  apply blast
done

lemma mu-srd-form:
  assumes mono F F ∈ [SRD]H → [SRD]H
  shows µ F = (µ X · Rs(preR(F(X)) ⊆ periR(F(X)) ⊆ postR(F(X))))
proof –
  have 1: F(µ X · Rs(preR(F(X)) ⊆ periR(F(X)) ⊆ postR(F(X)))) is SRD
    by (simp add: Healthy-apply-closed assms (1) assms (2) mu-srd-SRD)
  have 2: Mono monoton-order SRDES F
    by (simp add: assms (1) mono-Monotone-utp-order)
  hence 3: µ F = F(µ F)
    by (simp add: srdes-theory-continuous.LFP-unfold[THEN sym] assms)
  hence Rs(preR(F(F(µ F)))) ⊆ periR(F(F(µ F))) ⊆ postR(F(F(F(µ F)))) = µ F
    using SRD-reactive-tri-design by force
  hence (µ X · Rs(preR(F(X)) ⊆ periR(F(X)) ⊆ postR(F(X))) ⊆ F(µ F))
    by (simp add: 2 srdes-theory-continuous.weak.LFP-lemma3 gfp-upperbound assms)
  thus ?thesis
    using assms 1 3 srdes-theory-continuous.weak.LFP-lowerbound eq-iff mu-srd-iter
    by (metis (mono-tags, lifting))
qed

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

end

5 Normal Reactive Designs

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

This additional healthiness condition is analogous to H3

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

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

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

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

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

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

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

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

lemma RD3-intro-pre:
  assumes P is SRD (\neg_r \pre_r(P)) :: true_r = (\neg_r \pre_r(P)) \$st' \notin \peri_r(P)
  shows P is RD3
  proof -
    have RD3(P) = R_s (\neg_r \pre_r(P) \wp_r false \vdash (\exists \$st' \cdot \peri_r(P) \circ \post_r P)
      by (simp add: RD3-def SRD-right-unit-tri-lemma assms)
    also have ... = R_s (\neg_r \pre_r(P) \wp_r false \vdash \peri_r P \circ \post_r P)
      by (simp add: assms(3) ex-unrest)
    also have ... = R_s (\neg_r \pre_r(P) \wp_r false \vdash \cmt_r P)
      by (simp add: wait-cond-peri-post-cmt)
    also have ... = R_s (\pre_r 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 $ok' \notin P \$ok' \notin Q \$ok' \notin R \$wait' \notin R
  shows R_s(P \vdash Q \circ R) :: II_R = R_s((\neg_r (\neg_r P) :: true_r) \vdash ((\exists \$st' \cdot Q) \circ R))
  proof -
    have R_s(P \vdash Q \circ R) :: II_R = R_s(P \vdash Q \circ R) :: R_s(true_r \vdash false \circ ($tr' = u \$tr \land [II]_R))
      by (simp add: srdes-skip-tri-design, rel-auto)
    also have ... = R_s ((\neg_r R1 (\neg_r R2s P) :: R1 true) \vdash (\exists \$st' \cdot Q) \circ (R1 (R2s R) :: R1 (R2s ($tr' = u \$tr \land [II]_R))))
      by (simp-all add: RHS-tri-design-composition assms unrest R2s-true R1-false R2s-false)
also have \( R_u (\neg\neg\neg R1 \ (\neg\n R2s \ P) ; ;\ R1\ true) \vdash (\exists\ jst \cdot \ Q) \circ R1 \ (R2s \ R) \)

proof

\[ (rel-auto, \text{metis (no-types, lifting) minus-zero-eq, meson order-refl trace-class.diff-cancel}) \]
thus \( \neg \text{thesis} \)
by \( \text{simp} \)

qed

also have \( R_u (\neg\neg\neg P) ; ;\ R1\ true) \vdash ((\exists\ jst \cdot \ Q) \circ R) \)
by \( \text{metis (no-types, lifting)} \)

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

qed

lemma RHS-tri-design-RD3-intro:
assumes
\[ \begin{align*}
  & \text{shows } R_u ((P \vdash Q \circ R) \text{ is RD3}) \\
  & \text{(simp add: Healthy-def RD3-def)} \\
  & \text{(simp add:subst RHS-tri-design-right-unit-lemma)} \\
  & \text{(simp-all add:assms unrest closure rpred)}
\end{align*} \]
done

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

lemma R1-right-unit-lemma:
\[ \begin{align*}
  & \text{shows } R_u ((b \land \neg_r (\neg_r P) ; ;\ true_r) \vdash ((\exists\ jst \cdot \ Q) \circ R)) \\
  & \text{(rel-auto)} \\
  & \text{(simp add:assms unrest closure rpred)} \\
  & \text{(simp add:assms unrest)}
\end{align*} \]
done

definition NSRD :: (\'(s, t::trace, a) hrel-rsp \Rightarrow (\'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 \( \text{(rel-auto, blast+)} \)

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

lemma NSRD-elim [RD-elim]:
\[ \begin{align*}
  & \text{shows } Q(R_u (\text{pre}_r(P) \vdash \text{peri}_r(P) \circ \text{post}_r(P)))) \Rightarrow Q(P)
\end{align*} \]
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:
  \( \text{NSRD}(P) = R_s(\neg_r (\neg_r \text{pre}_R(P)) :: R1 \text{ true}) \vdash ((\exists \, \text{st}^* \cdot \text{peri}_R(P)) \circ \text{post}_R(P)) \)
proof –
  have \( \text{NSRD}(P) = RD3(\text{SRD}(P)) \)
  by (metis (no-types, lifting) NSRD-def RD1-RD3-commute RD3-left-subsumes-RD2 SRD-def comp-def)
also have \( \ldots = RD3(R_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \circ \text{post}_R(P))) \)
  by (simp add: SRD-as-reactive-tri-design)
also have \( \ldots = R_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \circ \text{post}_R(P)) :: H_R \)
  by (simp add: RD3-def)
also have \( \ldots = R_s((\neg_r (\neg_r \text{pre}_R(P)) :: R1 \text{ true}) \vdash ((\exists \, \text{st}^* \cdot \text{peri}_R(P)) \circ \text{post}_R(P))) \)
  by (simp add: RHS-tri-design-right-unit-lemma unrest)
finally show \( ?\text{thesis} \).
qed

lemma NSRD-healthy-form:
  assumes \( P \) is NSRD
  shows \( R_s((\neg_r (\neg_r \text{pre}_R(P)) :: R1 \text{ true}) \vdash ((\exists \, \text{st}^* \cdot \text{peri}_R(P)) \circ \text{post}_R(P))) = P \)
  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 \( \text{NSRD} (\bigcap A) = (\bigcap (\text{NSRD} 'A)) \)
  by (simp add: ContinuousD NSRD-Continuous assms(2))
also have \( \ldots = (\bigcap A) \)
  by (simp only: Healthy-carrier-image assms)
finally show \( ?\text{thesis} \) 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]:
  \( \forall i. \, i \in A \Rightarrow P(i) \text{ is NSRD} \) \( A \neq \{\} \) \( \Rightarrow (\bigcap i \in A. \, P(i)) \) is NSRD
  by (rule NSRD-Sup-closure, auto)

lemma NSRD-neg-pre-unit:
  assumes \( P \) is NSRD
  shows \( (\neg_r \text{pre}_R(P)) :: \text{true}_r = (\neg_r \text{pre}_R(P)) \)
proof –
  have \( (\neg_r \text{pre}_R(P)) = (\neg_r \text{pre}_R(R_s((\neg_r (\neg_r \text{pre}_R(P)) :: R1 \text{ true}) \vdash ((\exists \, \text{st}^* \cdot \text{peri}_R(P)) \circ \text{post}_R(P)))))) \)
  by (simp add: NSRD-healthy-form assms)
also have \( \ldots = R1 (R2c ((\neg_r \text{pre}_R(P)) :: R1 \text{ true})) \)

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

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

lemma NSRD-st'-unrest-peri [unrest]:
  assumes P is NSRD
  shows $st' ♯ peri_R(P)
proof
  have peri_R(P) = peri_R(R])))
  by (simp add: NSRD-healthy-form assms)
also have ...
  by (simp add: rea-pre-RHS-design usubst unrest)
also have ...
  by (simp add: R1-def R2c-def unrest)
finally show ?thesis .
qed

lemma NSRD-wait'-unrest-pre [unrest]:
  assumes P is NSRD
  shows $wait' ♯ pre_R(P)
proof
  have pre_R(P) = pre_R(Rs(¬r pre_R(P )))
  by (simp add: NSRD-healthy-form assms)
also have ...
  by (simp add: rea-pre-RHS-design usubst unrest)
also have ...
  by (simp add: R1-def R2c-def unrest)
finally show ?thesis .
qed

lemma NSRD-st'-unrest-pre [unrest]:
  assumes P is NSRD
  shows $st' ♯ pre_R(P)
proof
  have pre_R(P) = pre_R(Rs(¬r pre_R(P )))
  by (simp add: NSRD-healthy-form assms)
also have ...
  by (simp add: rea-pre-RHS-design usubst unrest)
also have ...
  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]: \( P \text{ is NSRD} \implies \text{pre}_R(P) \text{ is RR} \)
by (rule RR-intro, simp-all add: closure unrest)

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

lemma NSRD-intro:
  assumes \( P \text{ is SRD} \) \( \neg \text{pre}_R(P) \) \( \true \)
  true \( \neg \text{pre}_R(P) \)
shows \( P \text{ is NSRD} \)
proof -
  have \( \text{NSRD}(P) = \mathcal{R}_s((\neg \text{pre}_R(P)) \implies \true) \vdash ((\exists \mathcal{S} \cdot \text{peri}_R(P)) \circ \text{post}_R(P)) \)
    by (simp add: NSRD-form)
  also have \( \cdots = \mathcal{R}_s(\text{pre}_R P \vdash \text{peri}_R P \circ \text{post}_R P) \)
    by (simp add: assms ex-unrest rpred closure)
  also have \( \cdots = P \)
    by (simp add: SRD-reactive-tri-design assms)
  finally show \( \text{thesis} \)
    using \( \text{Healthy-def} \) by blast
qed

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

lemma NSRD-RC-intro:
  assumes \( P \text{ is SRD} \) \( \text{pre}_R(P) \text{ is RC} \) \( \mathcal{S} \text{ peri}_R(P) \)
shows \( P \text{ is NSRD} \)
by (metis Healthy-def NSRD-form SRD-reactive-tri-design assms)

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

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

lemma NSRD-iff:
  \( P \text{ is NSRD} \iff ((P \text{ is SRD}) \land (\neg \text{pre}_R(P))) \land \text{R1(true)} = (\neg \text{pre}_R(P)) \land (\mathcal{S} \text{ peri}_R(P)) \)
by (meson NSRD-intro NSRD-is-SRD NSRD-neg-pre-unit NSRD-st'-unrest-peri)

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

lemma NSRD-refine-elim:
  assumes

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

\[ \text{lemma} \quad \text{NSRD-right-unit}: P \text{ is NSRD} \implies P = I_R = P \]

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

\[ \text{lemma} \quad \text{NSRD-composition-wp}: \]

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

\[ \text{shows} \quad P \implies (P \& \& Q) \]

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

\[ \text{proof} \quad \]

\[ \text{have} \quad R \quad \text{by simp add: R\_seq add NSRD-is-SRD wp-rea-def NSRD-neg-pre-unit NSRD-st^\prime-unrest-peri R1-idem R2c-not closure Healthy-def Healthy-if NSRD-is-SRD R1-R2c-commute R1-R2c-closed rea-not-R1 rea-not-R2c) \]

\[ \text{then show} \quad \text{thesis} \quad \]

\[ \text{qed} \]

\[ \text{lemma} \quad \text{pre-R-NSRD-seq-lemma}: \]

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

\[ \text{shows} \quad R \quad \text{by simp add: NSRD-comp-wp assms rea-pre-RHS-design usubst unrest wp-rea-def R2c-disj R1-disj R2c-and R2c-preR R1-R2c-commute[THEN sym] R1-extend-conj' R1-idem R2c-not closure (metis (no_types, lifting) Healthy-def Healthy-if NSRD-is-SRD R1-R2c-commute R1-R2c-seq-distribute R1-seq-closure assms(1 \& \& 2) postR-R2c-closed postR-SRD-R1 preR-R2c-seq-killed rea-not-R1 rea-not-R2c) \]

\[ \text{lemma} \quad \text{peri-R-NSRD-seq [rdes]}: \]

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

\[ \text{shows} \quad P \implies (P \& \& Q) \]

\[ \text{by} \text{ (simp add: NSRD-comp-wp assms rea-pre-RHS-design usubst unrest wp-rea-def R2c-disj R1-disj R2c-and R2c-preR R1-R2c-commute[THEN sym] R1-extend-conj' R1-idem R2c-not closure (metis (no_types, lifting) Healthy-def Healthy-if NSRD-is-SRD R1-R2c-commute R1-R2c-seq-distribute R1-seq-closure assms(1 \& \& 2) postR-R2c-closed postR-SRD-R1 preR-R2c-seq-killed rea-not-R1 rea-not-R2c) \]

\[ \text{lemma} \quad \text{post-R-NSRD-seq [rdes]:} \]

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

\[ \text{shows} \quad P \implies (P \& \& Q) \]

\[ \text{proof} \quad \]

\[ \text{have} \quad R \quad \text{by simp add: NSRD-comp-wp assms rea-pre-RHS-design usubst unrest wp-rea-def R2c-disj R1-disj R2c-and R2c-preR R1-R2c-commute[THEN sym] R1-extend-conj' R1-idem R2c-not closure (metis (no_types, lifting) Healthy-def Healthy-if NSRD-is-SRD R1-R2c-commute R1-R2c-seq-distribute R1-seq-closure assms(1 \& \& 2) postR-R2c-closed postR-SRD-R1 preR-R2c-seq-killed rea-not-R1 rea-not-R2c) \]

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

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

lemma NSRD-seqr-closure [closure]:
assumes P is NSRD Q is NSRD
shows (P ;; Q) is NSRD
proof
  have (∼r post_R P wp_r pre_R Q) ;; true_r = (∼r post_R P wp_r pre_R Q)
    by (simp add: wp-rea-def rpred assms seqr-assoc NSRD-neg-pre-unit)
  moreover have $st' ;; pre_R P ∧ post_R P wp_r pre_R Q ⇒r peri_R P ∨ post_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' ;; P $ok' ;; Q_1 $ok' ;; Q_2 $ok' ;; S $ok' ;; S_2
$wait ;; R $wait' ;; Q_2 $wait ;; S_1 $wait ;; S_2
P is R2c Q_1 is R1 Q_1 is R2c Q_2 is R1 Q_2 is R2c
R is R2c S_1 is R1 S_1 is R2c S_2 is R1 S_2 is R2c
R1 (∼ P) ;; R1(true) = R1(∼ P) $st' ;; Q_1
shows R_n((P ⊓ Q_2 wp_R R) ⊓ (Q_1 ∨ (Q_2 ;; S_1)) ∨ (Q_2 ;; S_2)) = R_n((P ∧ Q_2 wp_R R) ⊓ (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_n((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 Q_is RR R is RR S_1 is RR S_2 is RR
shows R_n((P ⊓ Q_2 wp_R R) ⊓ (Q_1 ∨ (Q_2 ;; S_1)) ∨ (Q_2 ;; S_2)) = R_n((P ∧ Q_2 wp_R R) ⊓ (Q_1 ∨ (Q_2 ;; S_1)) ∨ (Q_2 ;; S_2))
         by (simp add: Healthy-def RC1-def rea-not-def)
         using R1-negate-R1 R1-seqr wp-pred-laws.double-compl
         (metis R1-negate-R1 R1-seqr wp-pred-laws.double-compl)
thus ?thesis
         by (simp add: RHS-tri-normal-design-composition assms closure unrest RR-implies-R2c)
qed

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

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

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

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

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

lemma NSRD-right-Miracle-tri-lemma:
assumes \( P \) is NSRD
shows \( P ;; \text{Miracle} = R_s(\text{pre}_R P \vdash \text{peri}_R P \circ \text{false}) \)
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 \( P \subseteq P ;; \text{Miracle} \)
proof –
  have \( R_s(\text{pre}_R P \vdash \text{peri}_R P \circ \text{post}_R P) \subseteq R_s(\text{pre}_R P \vdash \text{peri}_R P \circ \text{false}) \)
    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 \( \Rightarrow \) \( P \uplus \text{Suc} n \) is NSRD
proof (induct n)
  case 0
  then show \(?case\)
    by (simp)
next
  case (Suc n)
  then show \(?case\)
    by (simp add: NSRD-seqr-closure upred-semiring.power-Suc)
qed

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

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

lemma preR-power:
  assumes \( P \) is NSRD
  shows \( \text{pre}_R(P ;; P^* \uplus n) = (\bigcup i \in \{0..n\}, (\text{post}_R(P)^* \circ i) \text{ wp}_r(\text{pre}_R(P))) \)
proof (induct n)
  case 0
  then show \(?case\)
    by (simp add: wp closure)
next
  case (Suc n) note hyp = this
have \( \text{pre}_R (P \cdot (\text{Suc } n + 1)) = \text{pre}_R (P \cdot (n+1)) \)

by (simp add: upred-semiring-power-Suc)

also have \( \ldots = (\text{pre}_R P \land \text{post}_R P \cdot \text{wp}_R \cdot \text{pre}_R (P \cdot (\text{Suc } n))) \)

using NSRD-iff assms preR-NSRD-seq power-Suc-NSRD-closed by fastforce

also have \( \ldots = (\text{pre}_R P \land \text{post}_R P \cdot \text{wp}_R (\bigcup i\in\{0..n\}. \text{post}_R P \cdot i \cdot \text{wp}_R \cdot \text{pre}_R P)) \)

by (simp add: hyp upred-semiring.power-Suc)

also have \( \ldots = (\text{pre}_R P \land \bigcup i\in\{0..n\}. \text{post}_R P \cdot \text{wp}_R (\text{post}_R P \cdot i \cdot \text{wp}_R \cdot \text{pre}_R P)) \)

by (simp add: wp)

also have \( \ldots = (\text{pre}_R P \land \bigcup i\in\{0..n\}. \text{post}_R P \cdot (i+1) \cdot \text{wp}_R \cdot \text{pre}_R P)) \)

proof –

have \( \bigwedge i. R1 (\text{post}_R P \cdot i ;; (\neg R \cdot \text{pre}_R P)) = (\text{post}_R P \cdot i ;; (\neg R \cdot \text{pre}_R P)) \)

by (induct-tac i, simp-all add: closure Healthy-if assms)

thus \?thesis

by (simp add: wp-rea-def upred-semiring.power-Suc seqr-assoc rpred closure assms)

qed

also have \( \ldots = (\text{post}_R P \cdot 0 \cdot \text{wp}_R \cdot \text{pre}_R P \land \bigcup i\in\{0..n\}. \text{post}_R P \cdot (i+1) \cdot \text{wp}_R \cdot \text{pre}_R P)) \)

by (simp add: wp assms closure)

also have \( \ldots = (\text{post}_R P \cdot 0 \cdot \text{wp}_R \cdot \text{pre}_R P \land \bigcup i\in\{1..\text{Suc } n\}. \text{post}_R P \cdot i \cdot \text{wp}_R \cdot \text{pre}_R P)) \)

proof –

have \( (\bigcup i\in\{0..n\}. \text{post}_R P \cdot (i+1) \cdot \text{wp}_R \cdot \text{pre}_R P)) = (\bigcup i\in\{1..\text{Suc } n\}. \text{post}_R P \cdot i \cdot \text{wp}_R \cdot \text{pre}_R P)) \)

by (rule cong[of Inf], simp-all add: fun-eq-iff)

(metis (no-types, lifting) image-Suc-atLeastAtMost image-cong image-image)

thus \?thesis by simp

qed

also have \( \ldots = (\bigcup i\in\text{insert } 0 \{1..\text{Suc } n\}. \text{post}_R P \cdot i \cdot \text{wp}_R \cdot \text{pre}_R P) \)

by (simp add: conj-upred-def)

also have \( \ldots = (\bigcup i\in\{0..\text{Suc } n\}. \text{post}_R P \cdot i \cdot \text{wp}_R \cdot \text{pre}_R P) \)

by (simp add: atLeast0-atMost-Suc-eq-insert-0)

finally show \?case by (simp add: upred-semiring.power-Suc)

qed

lemma preR-power'[rdes]:

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

shows \( \text{pre}_R(P ;; P^* n) = (\bigcup i\in\{0..n\} \cdot (\text{post}_R(P) \cdot i) \cdot \text{wp}_R \cdot \text{pre}_R(P)) \)

by (simp add: preR-power assms USUP-as-Inf[THEN sym])

lemma preR-power-Suc [rdes]:

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

shows \( \text{pre}_R(P^*(\text{Suc } n)) = (\bigcup i\in\{0..n\} \cdot (\text{post}_R(P) \cdot i) \cdot \text{wp}_R \cdot \text{pre}_R(P)) \)

by (simp add: upred-semiring.power-Suc rdes assms)

declare upred-semiring.power-Suc [simp]

lemma periR-power:

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

shows \( \text{peri}_R(P ;; P^* n) = (\text{pre}_R(P^*(\text{Suc } n)) \Rightarrow (\prod i\in\{0..n\}. \text{post}_R(P) \cdot i) ;; \text{peri}_R(P)) \)

proof (induct n)

case 0

then show \?case

by (simp add: NSRD-is-SRD NSRD-wait'-unrest-pre SRD-peri-under-pre assms)

next

case (Suc n) note \( \text{hyp} = \text{this} \)

have \( \text{peri}_R (P \cdot (\text{Suc } n + 1)) = \text{peri}_R (P ;; P \cdot (n+1)) \)

by (simp)

also have \( \ldots = (\text{pre}_R(P \cdot (\text{Suc } n + 1)) \Rightarrow \text{peri}_R P \lor \text{post}_R P ;; \text{peri}_R (P ;; P \cdot n)) \)

56
by (simp add: closure assms rdes)
also have ... = (pre R(P) (Suc n)) ⇒ 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 P pre R (P) (Suc n)) ⇒ r post R P ; (pre R (P) (Suc 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 P pre R (P) (Suc 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 post R-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) (Suc n) ⇒ r (∏ i ∈ {0..n}. post R P ∨ i) ; peri R P ) is R1
  by (simp add: closure)
hence (post R P wp P pre R (P) (Suc n) ⇒ r post R P ; (pre R (P) (Suc n) ⇒ r (∏ i ∈ {0..n}. post R P ∨ i) ; peri R P ))
  = (post R P wp P pre R (P) (Suc n) ⇒ r R1 (post R P) ; R1 (pre R (P) (Suc 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-reas-impl-lemma, simp add: Healthy-if I, simp add: R1-post-SRD assms closure)
qed
also have ... = (pre R P ∧ post R P wp P pre R (P) (Suc 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 P pre R (P) (Suc n) ⇒ r peri R P ∨ (∏ i ∈ {0..n}. post R P (Suc i)) ; peri R P ))
  by (simp add: seq-Sup-distl seqr-assoc[THEN sym])
also have ... = (pre R P ∧ post R P wp P pre R (P) (Suc 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)
  using Suc-le-D apply fastforce
done
thus ?thesis by simp
qed
also have ... = (pre R P ∧ post R P wp P pre R (P) (Suc n) ⇒ r (∏ i ∈ {0..Suc n}. post R P ∨ i) ; peri R P )
  by (simp add: SUP-atLeastAtMost-first uinf-or seqr-or-distl seqr-or-distr)
also have ... = (pre R (P) (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]:
assumes $P$ is NSRD
shows $\varpi_R(P; P^\ast n) = (\varpi_R(P^\ast (Suc n)) \Rightarrow_r (\prod i \in \{0..n\} \cdot post_R(P) \cdot i)) ;; \varpi_R(P)$
by (simp add: $\varpi_R$-power assms UINF-as-Sup THEN sym)

lemma $\varpi_R$-power-Suc [rdes]:
assumes $P$ is NSRD
shows $\varpi_R(P^\ast (Suc n)) = (\varpi_R(P^\ast (Suc n)) \Rightarrow_r (\prod i \in \{0..n\} \cdot post_R(P) \cdot i)) ;; \varpi_R(P)$
by (simp add: rdes assms)

lemma postR-power [rdes]:
assumes $P$ is NSRD
shows postR($P; P^\ast n) = (\varpi_R(P^\ast (Suc n)) \Rightarrow_r post_R(P) \cdot Suc n$
proof (induct n)
  case 0
  then show ?case
  by (simp add: NSRD-is-SRD NSRD-wait-unrest-pre SRD-post-under-pre assms)
next
  case (Suc $n$) note hyp = this
  have postR($P \cdot Suc n + 1) = post_R (P; P \cdot (n+1))$
   by (simp)
  also have ... = (\varpi_R(P \cdot (Suc n + 1)) \Rightarrow_r (post_R P; post_R (P; P^\ast n))$
   by (simp add: closure assms rdes)
  also have ... = (\varpi_R(P \cdot (Suc n + 1)) \Rightarrow_r (post_R P; (\varpi_R (P \cdot Suc n) \Rightarrow_r post_R P \cdot Suc n))$
   by (simp only: hyp)
  also have ... = (\varpi_R P \Rightarrow_r (post_R WP \varpi_R (P \cdot Suc n) \Rightarrow_r post_R P; (\varpi_R (P \cdot Suc n) \Rightarrow_r post_R P \cdot Suc n))$
   by (simp add: rdes closure assms)
finally show ?case by (simp)
qed

lemma postR-power-Suc [rdes]:
assumes $P$ is NSRD
shows postR($P^\ast (Suc n)) = (\varpi_R(P^\ast (Suc n)) \Rightarrow_r post_R(P) \cdot Suc n$
by (simp add: rdes assms)

lemma power-rdes-def [rdes-def]:
assumes $P$ is RC $Q$ is RR $R$ is RR $\$st \#\ Q$
shows $(R_\ast(P \cdot Q \circ R))^\ast (Suc n)$
$= R_\ast((\prod i \in \{0..n\} \cdot (R \cdot i) \cdot WP \cdot P) \cdot (\prod i \in \{0..n\} \cdot R \cdot i)) ;; Q) \circ (R \cdot Suc n))$
proof (induct n)
  case 0
  then show ?case
  by (simp add: WP assms closure)
next
  case (Suc $n$)
have 1: \((P \land (\bigsqcup i \in \{0..n\} \cdot R \wp_r (R \cdot i \wp_r P))) = (\bigsqcup i \in \{0..Suc n\} \cdot R \cdot i \wp_r P)\)
(is \(\textsf{lhs} = \textsf{rhs}\)

proof –

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

finally show \(\text{thesis}\)
  by (simp add: USUP-as-Inf-collect)
qed

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

proof –

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

finally show \(\text{thesis}\)
  by (simp add: UINF-as-Sup-collect)
qed

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

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

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

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

59
declare upred-semiring.power-Suc [simp del]

theorem uplus-rdes-def [rdes-def]:
  assumes P is RC Q is RR R is RR $st' * Q
  shows $(R, (P \oplus Q \odot R))^{*} = R_{s}(R_{s}^{*} \uplus_{p} P \oplus R_{s}^{*} \odot Q \odot R^{*})$
proof
  have 1:($\Pi i \cdot R^{-i}) \odot Q = R_{s}^{*} \odot Q
    by (metis (no-types) RA1 assms 2 rea-unit-unit2 rrel-thy Star-def ust-alt-def)
  show \?
  proof
    by (simp add: uplus-power-def seq-UINF-distr wp closure assms rdes-def)
  qed

5.1 UTP theory

typedec NSRDES

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

overloading
  nsrcdes-hcond ≡ upred-hcond :: (NSRDES, ('s', 't::trace', 'a) rsp) uthy ⇒ ((('s', 't', 'a) rsp × ('s', 't', 'a) rsp)
  nsrcdes-unit ≡ upred-unit :: (NSRDES, ('s', 't::trace', 'a) rsp) uthy ⇒ ('s', 't', 'a) hrel-rsp

begin
  definition nsrcdes-hcond :: (NSRDES, ('s', 't::trace', 'a) rsp) uthy ⇒ ((('s', 't', 'a) rsp × ('s', 't', 'a) rsp)
    health where
    [upred-defs]: nsrcdes-hcond $T = NSRD
  definition nsrcdes-unit :: (NSRDES, ('s', 't::trace', 'a) rsp) uthy ⇒ ('s', 't', 'a) hrel-rsp where
  [upred-defs]: nsrcdes-unit $T = I_{R}
end

interpretation nsrc-thy: upred-theory-kleene UTHY(NSRDES, ('s', 't::trace', 'a) rsp)
rewrites $\land P, P \in \text{carrier} (uthy-order NSRDES) \iff P$ is NSRD
and P is $H_{NSRDES} \iff P$ is NSRD
and $(\mu X \cdot F (H_{NSRDES} X)) = (\mu X \cdot F (NSRD X))$
and carrier (uthy-order NSRDES) → carrier (uthy-order NSRDES) ≡ [NSRD]_{H} → [NSRD]_{H}
and $H_{NSRDES} \odot H_{NSRDES} \equiv [NSRD]_{H} \odot [NSRD]_{H}$
and $\bot_{NSRDES} = \text{Miracle}$
and $\top_{NSRDES} = I_{R}$
and $le (uthy-order NSRDES) = op \subseteq$
proof
  interpret lat: uptr-theory-continuous UTHY(NSRDES, ('s', 't', 'a) rsp)
  by (unfold-locales, simp-all add: nsrcdes-hcond-def nsrcdes-unit-def closure Healthy-if)
  show 1: $\top_{NSRDES} = (\text{Miracle} :: ('s', 't', 'a) hrel-rsp)$
  by (metis NSRD-Miracle NSRD-is-SRD lat.top-healthy lat.upt-theory-continuous-axioms nsrcdes-hcond-def
    nsrcdes-theory-continuous.meet-top upred-semiring.add-commute upr-theory-continuous.meet-top)
thus uptr-theory-kleene UTHY(NSRDES, ('s', 't', 'a) rsp)
  by (unfold-locales, simp-all add: nsrcdes-hcond-def nsrcdes-unit-def closure Healthy-if Miracle-left-zero
    SRD-left-unit NSRD-right-unit)
qed (simp-all add: nsrcdes-hcond-def nsrcdes-unit-def closure Healthy-if)

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

abbreviation TestR (test R) where
test \_ R \ P \equiv \ u t e s t \ NSRDES \ P

abbreviation StarR \ :: \ (s \next trace, \ 'a) \ h r e l - r s p \ \Rightarrow \ (s \next, \ 'a) \ h r e l - r s p (\_ \next R [999] 999) \ where
StarR \ P \equiv P \cdot NSRDES

lemma StarR-rdes-def [rdes-def]:
  \textbf{assumes} \ P \ is \ \mathrm{RC} \ Q \ is \ RR \ R \ is \ RR \ $ st' \ not Q \\
  \textbf{shows} \ (R \cdot (P \triangleright Q \odot R))^R = R \cdot ((R^{*\star} wp_r P) \triangleright R^{*\star} \odot Q \odot R^{*\star})
by (simp add: rrel-thy.Star-alt-def assms closure rdes-def unrest rpred disj-upred-def)

end

6 Syntax for reactive design contracts

theory utp-rdes-contracts
  \textbf{imports} utp-rdes-normal
begin

We give an experimental syntax for reactive design contracts \([P \triangleright Q]_R\), 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 \(trace\) which is bound to \&tt.

definition mk-RD \ :: \ 's \ upred \ \Rightarrow \ ('t::trace \Rightarrow 's \ upred) \Rightarrow ('t \Rightarrow 's \ hrel) \Rightarrow ('s, 't, 'a) \ hrel-rsp \ where
mk-RD \ P \ Q \ R = R \cdot ([P]_S \triangleright ([Q(x)]_S \odot [x \rightarrow \& tt] \odot [R(x)]_S \odot [x \rightarrow \& tt])

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

syntax
- \_trace-var :: logic
- \_mk-RD \ :: \ logic \Rightarrow logic \Rightarrow logic \Rightarrow logic \ (\_\_ \ \triangleright \_ \_ \_ \_ \_ \_ \_ \_)
- \_trace-pred \ :: \ logic \Rightarrow logic \ (\_\_ \_\_\_\_\_\_\_)

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

translations
\[ [P \triangleright Q]_R \Rightarrow CONST \ \text{mk-RD} \ P \ (\lambda \_trace-var. \ Q) \ (\lambda \_trace-var. \ R) \]
\[ [P \triangleright Q]_R \ \triangleleft \Rightarrow CONST \ \text{mk-RD} \ P \ (\lambda \ x. \ Q) \ (\lambda \ y. \ R) \]
\[ [P]_t \Rightarrow CONST \ trace-pred \ (\lambda \_trace-var. \ P) \]
\[ [P]_t \ \triangleleft \Rightarrow CONST \ trace-pred \ (\lambda \ t. \ P) \]

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

lemma preR-mk-RD [rdes]: \preR([P \triangleright Q(trace) \mid R(trace) \mid R]_R) = R1([P]_S \triangleleft)
by (simp add: mk-RD-def rea-pre-RHS-design unrest unrest R2c-not R2c-lift-state-pre)
lemma trace-pred-RR-closed [closure]:
\[ \text{if } [P \text{ trace}]_t \text{ is RR} \]
by (rel-auto)

lemma unrest-trace-pred-st' [unrest]:
\[ S' \text{ }\not\subseteq [P \text{ trace}]_t \]
by (rel-auto)

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

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

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

Refinement introduction law for contracts

lemma RD-contract-refine:
assumes \[ Q \text{ is SRD}' [P_1]_{S<} [\text{trace}] | R \]
shows \[ [P_1]_{S<} \Rightarrow R1 [[Q]_{S<}][\text{trace}\rightarrow\&\text{tt}]] \]
proof –
have \[ [P_1]_{S<} \text{ peri}_R Q \]
using assms
by (simp add: mk-RD-def, rule-tac srdes-tri-refine-intro, simp-all)
thus ?thesis
by (simp add: SRD-reactive-tri-design assms(1))
qed

end

7 Reactive design tactics

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

Theorems for normalisation

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

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

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

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

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

Tactic to simplify the definition of a reactive design

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

Tactic to split a refinement conjecture into three POs

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

Tactic to split an equality conjecture into three POs

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

Tactic to prove a refinement

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

Tactics to prove an equality

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

Via antisymmetry

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

Tactic to calculate pre/peri/postconditions from reactive designs

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

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

**method** `rdsp-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`
R3h implicitly depends on RD1, and therefore it requires that both sides be RD1. We also require that both sides are R3c, and that wait also has implicit dependencies on RD1, and therefore it requires that both sides be RD1.

**Lemma st-U1-alpha:** \( \exists \ st \cdot [II]_0 = (\exists \ st \cdot [II]_1) \)

**Proof** by (rel-auto)

**Lemma st-U1-alpha:** \( \exists \ st \cdot [II]_1 = (\exists \ st \cdot [II]_1) \)

**Proof** by (rel-auto)

**Definition skip_rm :: \( ('s, 't::trace, 'α) \) r specifies \( \text{merge} (II_RM) \) where**

\[ \text{upred-defs}: II_RM = (\exists \ st_0 \cdot \text{skip}_m \lor (\neg \$ok_0 \land \$tr_0 \leq \$tr_0)) \]

**Definition** \( \text{upred-defs}: R3hm(M) = (II_RM \land \text{wait} \land M) \)

**Lemma R3hm-idem:** \( R3hm(R3hm(P)) = R3hm(P) \)

**Proof** by (rel-auto)

**Lemma R3h-par-by-merge [closure]:**

**Proof**

- **Have** \( (P \parallel M) = ((P \parallel M) [true/\$ok] \land \$ok \lor (P \parallel M) [false/\$ok] [true/\text{wait}] \land \text{wait} \lor (P \parallel M)) \)
  - **By** (simp add: cond-var-subst-left cond-var-subst-right)

- **Also have** \( (P \parallel M) = ((P \parallel M) [true/\$ok,\text{wait}] \land \$ok \lor (P \parallel M) [false/\$ok,\text{wait}] \land \text{wait} \lor (P \parallel M)) \)
  - **By** (rel-auto)

- **Also have** \( (P \parallel M) = ((P \parallel M) [true/\$ok,\text{wait}] \land \$ok \lor (P \parallel M) [false/\$ok,\text{wait}] \land \text{wait} \lor (P \parallel M)) \)
  - **By** (simp add: cond-var-subst-right)

- **Also have** \( (P \parallel M) = ((P \parallel M) [false/\$ok,\text{wait}] \land \$ok \lor (P \parallel M) [true/\$ok,\text{wait}] \land \text{wait} \lor (P \parallel M)) \)
  - **By** (simp add: assms Healthy-if)

- **Also have** \( (P \parallel M) = ((P \parallel M) [true/\$ok,\text{wait}] \land \$ok \lor (P \parallel M) [false/\$ok,\text{wait}] \land \text{wait} \lor (P \parallel M)) \)
  - **By** (simp add: close assms unrest)

- **Finally show** **thesis by** (simp add: closure assms unrest)

- **QED**

**Also have** \( (P \parallel M) = ((P \parallel M) [true/\$ok,\text{wait}] \land \$ok \lor (P \parallel M) [false/\$ok,\text{wait}] \land \text{wait} \lor (P \parallel M)) \)

**Proof**

- **Have** \( (P \parallel M) [false/\$ok,\text{wait}] = ((P \parallel M) [true/\$ok,\text{wait}] \land \$ok \lor (P \parallel M) [false/\$ok,\text{wait}] \land \text{wait} \lor (P \parallel M)) \)
  - **By** (simp add: par-by-merge-def U0-as-alpha U1-as-alpha assms Healthy-if)

**QED**
also have \( ... = (((\exists \ $st \cdot II) \triangleleft \ $ok \triangleright R1(true)) \triangleleft \ $wait \triangleright (P \parallel_M Q)) \)
  \[ \text{by (rel-auto)} \]
also have \( ... = R3h(P \parallel_M Q) \)
  \[ \text{by (simp add: R3h-cases)} \]
finally show \( ?\text{thesis} \)
  \[ \text{by (simp add: Healthy-def)} \]
qed

definition \([upred-defs]: RD1m(M) = (M \lor \neg \ $ok < \neg \ $tr < \leq_u \ $tr \') \)

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

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

lemma \(SRD\text{-par-by-merge}: \)
  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 \( SRD \)
  \[ \text{by (rule SRD-intro, simp-all add: assms closure SRD-healths)} \]

definition \(nmerge-rd0 \ (N_0) \) where
  \([upred-defs]: N_0(M) = (\$wait' =_u (\$0-wait \lor \$1-wait) \land \text{\$tr} < \leq_u \ text{\$tr \'} \land (\exists \text{\$0-ok;\$1-ok;\$ok <;\$ok \';\$0-wait;\$1-wait;\$wait <;\$wait \'} \cdot M)) \)

definition \(nmerge-rd1 \ (N_1) \) where
  \([upred-defs]: N_1(M) = (\text{\$ok \'} =_u (\$0-ok \land \$1-ok) \land N_0(M)) \)
definition nmerge-rd \((N_R)\) where
\[ N_R(M) = ((\exists s_{t <} \cdot s_{v'} = u \cdot s_{v} < \cdot N_1(M)) \land s_{ok} \land (s_{tr} < u \cdot s_{tr'})) \]
definition merge-rd1 \((M_1)\) where
\[ M_1(M) = (N_1(M) \cdot \cdot II_R) \]
definition merge-rd \((M_R)\) where
\[ M_R(M) = N_R(M) \cdot \cdot II_R \]

abbreviation rdes-par \((\cdot \cdot \cdot R \cdot \cdot \cdot)\) where
\[ P \parallel R M Q \equiv P \parallel M_R(M) Q \]

Healthiness condition for reactive design merge predicates

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

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

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

lemma nmerge-rd-is-R2m [closure]:
\[ M \text{ is } R2m \Longrightarrow N_R(M) \text{ is } R2m \]
by (metis Healthy-def R2-par-by-merge R2-seqr-closure R2m-nmerge-rd RDM-def SRD-healths)

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

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

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

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

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

lemma RDM-intro:
assumes \(M \text{ is } R2m\)
\(s_{0} \cdot s_{1 \cdot} \cdot M \cdot s_{ok} \cdot s_{ok'} \cdot M \cdot s_{ok} \cdot s_{ok'} \cdot M \cdot s_{0 \cdot} \cdot s_{wait} \cdot s_{wait} \cdot s_{wait'} \cdot M \)
\(s_{0 \cdot} \cdot s_{wait} \cdot s_{wait} \cdot s_{wait} \cdot s_{wait'} \cdot M \)

66
shows $M$ is RDM
using assms
by (simp add: Healthy-def RDM-def ex-unrest unrest)

lemma RDM-unrests [unrest]:
assumes $M$ is RDM
shows $0-ok \parallel M \triangleright M \triangleright M \triangleright M \triangleright M$
by (subst Healthy-if[OF assms, THEN sym], simp-all add: RDM-def unrest, rel-auto)+

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

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

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

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

lemma parallel-ok-cases:
$((P \parallel Q) \parallel M) = $
  ((\forall t \in T \cdot Q t) \parallel (M \parallel Q t) \parallel (M \parallel Q t) \parallel (M \parallel Q t) \parallel (M \parallel Q t))
proof
  have $((P \parallel Q) \parallel M) = (\exists \text{ok}_0 \cdot (P \parallel Q) \parallel \text{ok}_0 \parallel \text{ok}_0 \parallel \text{ok}_0 \parallel \text{ok}_0 \parallel \text{ok}_0)$
by (simp add: true-leaf-def true-alt-def false-leaf-def false-alt-def)
also have $= (\exists \text{ok}_0 \cdot \exists \text{ok}_1 \cdot (P \parallel Q) \parallel \text{ok}_0 \parallel \text{ok}_0 \parallel \text{ok}_0 \parallel \text{ok}_0 \parallel \text{ok}_0) \parallel (M \parallel \text{ok}_0 \parallel \text{ok}_0 \parallel \text{ok}_0 \parallel \text{ok}_0 \parallel \text{ok}_0)$
by (simp add: true-alt-def false-leaf-def false-alt-def)
also have $= (\exists \text{ok}_0 \cdot \exists \text{ok}_1 \cdot (P \parallel \text{ok}_0 \parallel \text{ok}_0 \parallel \text{ok}_0 \parallel \text{ok}_0 \parallel \text{ok}_0))$ \parallel (M \parallel \text{ok}_0 \parallel \text{ok}_0 \parallel \text{ok}_0 \parallel \text{ok}_0 \parallel \text{ok}_0)$
by (rel-auto robust)
also have $= (\exists \text{ok}_0 \cdot \exists \text{ok}_1 \cdot (P \parallel \text{ok}_0 \parallel \text{ok}_0 \parallel \text{ok}_0 \parallel \text{ok}_0 \parallel \text{ok}_0))$
by (simp add: true-leaf-def true-alt-def false-leaf-def false-alt-def)
finally show ?thesis .
qed
lemma skip-srea-ok-f [usubst];
\[ H_{R_f'} = R_1(\neg \$ok) \]
by (rel-auto)

lemma nmerge0-rd-unrest [unrest];
\[ \$0 - \$ok \parallel N_0 \ M \ \$1 - \$ok \parallel N_0 \ M \]
by (pred-auto)+

lemma parallel-assm-lemma:
assumes \( P \) is RD2
shows \( \text{pre}_s \uparrow (P \parallel_{M_R(M)} Q) = (\text{pre}_s \uparrow (P \parallel Q) \parallel M_R(M)) \)
by (simp add: par-by-merge-def)
also have \( \ldots = ((P \parallel Q)[true, false/\$ok, \$wait] \parallel N_R \ M \parallel R_1(\neg \$ok)) \)
by (simp add: merge-rd-def usubst, rel-auto)
also have \( \ldots = ((P[true, false/\$ok, \$wait] \parallel Q[true, false/\$ok, \$wait]) \parallel N_1(M) \parallel R_1(\neg \$ok)) \)
by (rel-auto robust, (metis)+)
also have \( \ldots = (((P[true, false/\$ok, \$wait]) \parallel Q[true, false/\$ok, \$wait]) \parallel (N_1 M)[true, true/\$0 - \$ok, \$1 - \$ok]) \parallel R_1(\neg \$ok))) \)
by (simp add: par-by-merge-def)
also have \( \ldots = (((P[true, false/\$ok, \$wait]) \parallel Q[true, false/\$ok, \$wait]) \parallel (N_1 M)[false, true/\$0 - \$ok, \$1 - \$ok]) \parallel R_1(\neg \$ok))) \)
also have \( \ldots = ((P[true, false/\$ok, \$wait]) \parallel Q[true, false/\$ok, \$wait]) \parallel (N_1 M)[false, false/\$0 - \$ok, \$1 - \$ok]) \parallel R_1(\neg \$ok))) \)
also have \( \ldots = (\text{pre}_s \uparrow (P \parallel_{M_R(M)} Q) = \text{pre}_s \uparrow ((P \parallel Q) \parallel M_R(M)) \)
by (simp add: par-by-merge-def)
also have \( \ldots = ((P \parallel Q)[true, false/\$ok, \$wait] \parallel N_R \ M \parallel R_1(\neg \$ok)) \)
by (simp add: merge-rd-def usubst, rel-auto)
also have \( \ldots = ((P[true, false/\$ok, \$wait] \parallel Q[true, false/\$ok, \$wait]) \parallel N_1(M) \parallel R_1(\neg \$ok)) \)
by (rel-auto robust, (metis)+)
also have \( \ldots = (\text{pre}_s \uparrow (P \parallel_{M_R(M)} Q) = \text{pre}_s \uparrow ((P \parallel Q) \parallel M_R(M)) \)
by (simp add: par-by-merge-def)
also have \( \ldots = ((P \parallel Q)[true, false/\$ok, \$wait] \parallel N_R \ M \parallel R_1(\neg \$ok)) \)
by (simp add: merge-rd-def usubst, rel-auto)
also have \( \ldots = ((P[true, false/\$ok, \$wait] \parallel Q[true, false/\$ok, \$wait]) \parallel N_1(M) \parallel R_1(\neg \$ok)) \)
by (rel-auto robust, (metis)+)
also have \( \ldots = (\text{pre}_s \uparrow (P \parallel_{M_R(M)} Q) = \text{pre}_s \uparrow ((P \parallel Q) \parallel M_R(M)) \)
by (simp add: par-by-merge-def)
also have \( \ldots = ((P \parallel Q)[true, false/\$ok, \$wait] \parallel N_R \ M \parallel R_1(\neg \$ok)) \)
by (simp add: merge-rd-def usubst, rel-auto)
also have \( \ldots = ((P[true, false/\$ok, \$wait] \parallel Q[true, false/\$ok, \$wait]) \parallel N_1(M) \parallel R_1(\neg \$ok)) \)
by (rel-auto robust, (metis)+)
also have \( \ldots = (\text{pre}_s \uparrow (P \parallel_{M_R(M)} Q) = \text{pre}_s \uparrow ((P \parallel Q) \parallel M_R(M)) \)
by (simp add: par-by-merge-def)
also have \( \ldots = ((P \parallel Q)[true, false/\$ok, \$wait] \parallel N_R \ M \parallel R_1(\neg \$ok)) \)
by (simp add: merge-rd-def usubst, rel-auto)
also have \( \ldots = ((P[true, false/\$ok, \$wait] \parallel Q[true, false/\$ok, \$wait]) \parallel N_1(M) \parallel R_1(\neg \$ok)) \)
by (rel-auto robust, (metis)+)
also have \( \ldots = (\text{pre}_s \uparrow (P \parallel_{M_R(M)} Q) = \text{pre}_s \uparrow ((P \parallel Q) \parallel M_R(M)) \)
by (simp add: par-by-merge-def)
also have \( \ldots = ((P \parallel Q)[true, false/\$ok, \$wait] \parallel N_R \ M \parallel R_1(\neg \$ok)) \)
by (simp add: merge-rd-def usubst, rel-auto)
also have \( \ldots = ((P[true, false/\$ok, \$wait] \parallel Q[true, false/\$ok, \$wait]) \parallel N_1(M) \parallel R_1(\neg \$ok)) \)
by (rel-auto robust, (metis)+)}
lemma \( \pre^s \vdash P \) is SRD:

assumes \( P \) is SRD

shows \( \pre^s \vdash P = (\neg_r \pre_R(P)) \)

proof –

have \( \pre^s \vdash P = \pre^s \vdash R_s(\pre_R P \vdash peri_R P \circ \post_R P) \)
  by (simp add: SRD-reactive-tri-design assms)

also have \( \vdash R_1(R_{2c}(\neg \pre^s \vdash \pre_R P)) \)
  by (simp add: RHS-def asubst R3h-def assms)

also have \( \vdash (\neg_r \pre_R P) \)
  by (rel-auto)

also have \( \vdash \neg \pre_R P \)
  by (simp add: parallel-assm assms)

finally show \( \forall \thesis_{\forall \thesis} \)

qed

lemma parallel-assm:

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

shows \( \pre_R(P \parallel_{M_R(M)} Q) = (\neg_r((\neg_r \pre_R(P)) \parallel_{N_0(M)} ; R_1(true) \cmt_R(Q)) \land \neg_r(\cmt_R(P) \parallel_{N_0(M)} ; R_1(true) (\neg_r \pre_R(Q)))) \)

(is \( \forall \thesis_{\forall \thesis} \))

proof –

have \( \pre_R(P \parallel_{M_R(M)} Q) = (\neg_r(\pre^s \vdash P) \parallel_{N_0(M)} ; R_1 true (\cmt^s \vdash Q) \land \neg_r(\cmt^s \vdash P) \parallel_{N_0(M)} ; R_1 true (\pre^s \vdash Q)) \)
  by (simp add: parallel-assm-lemma assms)

also have \( \vdash (\neg_r \pre_R P) \)
  by (simp add: parallel-assm assms)

finally show \( \forall \thesis_{\forall \thesis} \)

qed

lemma parallel-assm-unrest-wait’ [unrest]:

\( \parallel P \) is SRD \( Q \) is SRD \( \Rightarrow \) \( $\wait \vdash \pre_R(P \parallel_{M_R(M)} Q) \)

by (simp add: parallel-assm, simp add: par-by-merge-def unrest)

lemma JL1: \( (M_1 M)^! [false, true /$0-0, \$1-0] = N_0(M) ; R_1(true) \)

by (rel-blast)

lemma JL2: \( (M_1 M)^! [true, false /$0-0, \$1-0] = N_0(M) ; R_1(true) \)

by (rel-blast)

lemma JL3: \( (M_1 M)^! [false, false /$0-0, \$1-0] = N_0(M) ; R_1(true) \)

by (rel-blast)

lemma JL4: \( (M_1 M)^! [true, true /$0-0, \$1-0] = (\$ok \land N_0(M) ; II_R^t \)

by (simp add: merge-rd1-def asubst nmerge-ld1-def unrest)

lemma parallel-commitment-lemma-1:
assumes $P$ is RD2
shows $\text{cnt}_s \vdash (P \parallel_{M_R(M)} Q) = (\text{pre}_s \parallel P) \parallel_{(M_1(M))_s} (\text{cnt}_s \parallel Q) \lor (\text{cnt}_s \parallel P) \parallel_{(M_1(M))_s} (\text{pre}_s \parallel Q))$

proof
have $\text{cnt}_s \vdash (P \parallel_{M_R(M)} Q) = (P[\text{true}, \text{false} \parallel\{\text{ok}, \text{false} \parallel\{\text{wait} \})] \parallel_{(M_1(M))_s} Q[\text{true}, \text{false} \parallel\{\text{ok}, \text{false} \parallel\{\text{wait} \})])$
  by (simp add: subst, rel-auto)
also have $... = ((P[\text{true}, \text{false} \parallel\{\text{ok}, \text{false} \parallel\{\text{wait} \})] \parallel_{(M_1(M))_s} Q[\text{true}, \text{false} \parallel\{\text{ok}, \text{false} \parallel\{\text{wait} \})]) \parallel_{(M_1(M))_s} (\text{pre}_s \parallel Q))$
  by (simp add: par-by-merge-def)
also have $... = (\text{pre}_s \parallel P) \parallel_{(M_1(M))_s} (\text{cnt}_s \parallel Q) \lor (\text{cnt}_s \parallel P) \parallel_{(M_1(M))_s} (\text{pre}_s \parallel Q))$
  by (stim add: subst parallel-ok-cases, subst-tac)
also have $... = (\text{cnt}_s \parallel P) \parallel_{(M_1(M))_s} (\text{cnt}_s \parallel Q) \lor (\text{pre}_s \parallel P) \parallel_{(M_1(M))_s} (\text{cnt}_s \parallel Q) \lor (\text{cnt}_s \parallel P) \parallel_{(M_1(M))_s} (\text{pre}_s \parallel Q) \lor (\text{pre}_s \parallel P) \parallel_{(M_1(M))_s} (\text{cnt}_s \parallel Q) \lor (\text{cnt}_s \parallel P) \parallel_{(M_1(M))_s} (\text{pre}_s \parallel Q))$
  by (simp add: J12 J1 J2 J3)
proof
from assms have $'P' \Rightarrow P'$
  by (metis RD2-def H2-equivalence Healthy-def)
hence $P':P' \Rightarrow P' :'$
  by (rel-auto)
have $'?C_4 \Rightarrow '?C_3' \text{ (is } '(?A ; '?B ) \Rightarrow (?C ; '?D)' )$
proof
have $'?A \Rightarrow '?C'$
  using $P$ by (rel-auto)
thus $?thesis$
    by (simp add: impl-seq-mon)
qed
thus $?thesis$
  by (simp add: substumption2)
qed
finally show $?thesis$
  by (simp add: par-by-merge-def J4)
qed

lemma parallel-commitment-lemma-2:
assumes $P$ is RD2
shows $\text{cnt}_s \vdash (P \parallel_{M_R(M)} Q) = (\text{cnt}_s \parallel P) \parallel_{(M_1(M))_s} (\text{pre}_s \parallel Q))$
  by (simp add: parallel-commitment-lemma-1 assms parallel-assm-lemma)

lemma parallel-commitment-lemma-3:
$M$ is $\text{R1m} \Rightarrow (\text{ok} \wedge \text{N_0 M}) ; H_{R^t}$ is $\text{R1m}$
by (rel-simp, safe, metis+)

lemma parallel-commitment:
  assumes P is SRD Q is SRD M is RDM
  shows \( cmt_R(P \parallel M_R(M) \parallel Q) = \langle \text{pre}_R(P) \parallel M_R(M) \parallel Q \rangle \Rightarrow \ cmt_R(Q) \parallel (\text{sok} \land N_0(M)) ; II_{R'} cmt_R(Q) \rangle \)
  by (simp add: parallel-commitment-lemma-2 parallel-commitment-lemma-3 Healthy-assms cmtR-def pre\_SRD closure rea-impl-def disj-comm unrest)

theorem parallel-reactive-design:
  assumes P is SRD Q is SRD M is RDM
  shows \( P \parallel M_R(M) \parallel Q = \text{R}_R(\text{pre}_R(P) \parallel N_0(M) ; R_I(\text{true}) cmt_R(Q) \rangle) \land \\
  \text{cmt}_R(P) \parallel (\text{sok} \land N_0(M)) ; R_I(\text{true}) (\neg \text{pre}_R(Q)) \langle \text{II}_{R'} cmt_R(Q) \rangle) \)
  (is ?lhs = ?rhs)
  proof
  have \( (P \parallel M_R(M) \parallel Q = \text{R}_R(\text{pre}_R(P) \parallel M_R(M) \parallel Q) \parallel \text{cmt}_R(P) \parallel M_R(M) \parallel Q) \)
  by (metis Healthy-def NSRD-is-SRD SRD-as-reactive-design assms(1) assms(2) assms(3) par\_rdes-NSRD)
  also have \( ... = ?rhs \)
  by (simp add: parallel-assm parallel-commitment design-export-spec assms, rel-auto)
  finally show \(?thesis\).

qed

lemma parallel-pericondition-lemma1:
(\( \text{sok} \land P \rangle ; II_{R'}(\text{true}, \text{true}/\text{sok}', \text{wait}'\rangle = (\exists \text{st}' \cdot P)[\text{true}, \text{true}/\text{sok}', \text{wait}'\rangle \)
(is ?lhs = ?rhs)
proof
  have \( ?lhs = (\text{sok} \land P) ; (\exists \text{st} \cdot II)[\text{true}, \text{true}/\text{sok}', \text{wait}'\rangle \)
  by (rel-auto)
  also have \( ... = ?rhs \)
  by (rel-auto)
  finally show \(?thesis\).

qed

lemma parallel-pericondition-lemma2:
  assumes M is RDM
  shows \( \exists \text{st}' \cdot N_0(M)[\text{true}, \text{true}/\text{sok}', \text{wait}'\rangle = ((\text{false} \land \text{true}) \land (\exists \text{st}' \cdot M)) \)
  proof
  have \( \exists \text{st}' \cdot N_0(M)[\text{true}, \text{true}/\text{sok}', \text{wait}'\rangle = (\exists \text{st}' \cdot (\text{false} \land \text{true}) \land (\exists \text{st}' \cdot M)) \land M \)
  by (simp add: usubst unrest nmerge-rd0-def ex-unrest Healthy-if R1m-def assms)
  also have \( ... = (\exists \text{st}' \cdot (\text{false} \land \text{true}) \land \text{M}) \)
  by (metis (no-types, hide-lams) Healthy-if R1m-def R1m-idem R2m-def RDM-def assms utp-pred-laws.inf-commute)
  also have \( ... = ((\text{false} \land \text{true}) \land (\exists \text{st}' \cdot M)) \)
  by (rel-auto)
  finally show \(?thesis\).

qed

lemma parallel-pericondition-lemma3:
(\( (\text{false} \land \text{true}) \land (\exists \text{st}' \cdot M)) = ((\text{false} \land \text{true}) \land (\exists \text{st}' \cdot M)) \land \neg (\text{false} \land \text{true}) \land \text{false} \land (\exists \text{st}' \cdot M)) \)
  by (rel-auto)

lemma parallel-pericondition [rdes]:
  fixes M :: ('s,'t::trace,α) rsp merge


assumes $P$ is SRD $Q$ is SRD $M$ is RDM
shows $\text{peri}_R(P \parallel M(M) Q) = (\text{peri}_R(P \parallel M_R M Q) \Rightarrow (\exists \text{peri}_R(P) \parallel M \text{peri}_R(Q))$
\begin{align*}
\text{proof} \quad & \\
\text{have} & \quad \text{peri}_R(P \parallel M(M) Q) = \\
& \quad (\text{peri}_R(P \parallel M_R M Q) \Rightarrow (\exists \text{peri}_R(P) \parallel M \text{peri}_R(Q))
\begin{align*}
\text{also have} & \quad \Rightarrow (\exists \text{peri}_R(P) \parallel M \text{peri}_R(Q))
\end{align*}
\begin{align*}
\text{also have} & \quad (\exists \text{peri}_R(P) \parallel M \text{peri}_R(Q))
\end{align*}
\begin{align*}
\text{finally show} & \quad \text{thesis}.
\end{align*}
qed

\text{lemma parallel-postcondition-lemma1:}

$(\exists \text{peri}_R(P) \parallel M(M) Q) = (\exists \text{peri}_R(P) \parallel M \text{peri}_R(Q))$
\begin{align*}
\text{proof} \quad & \\
\text{have} & \quad \Rightarrow (\exists \text{peri}_R(P) \parallel M \text{peri}_R(Q))
\begin{align*}
\text{finally show} & \quad \text{thesis}.
\end{align*}
qed

\text{lemma parallel-postcondition-lemma2:}

assumes $M$ is RDM
shows $(\exists \text{peri}_R(P) \parallel M(M) Q) = (\exists \text{peri}_R(P) \parallel M \text{peri}_R(Q))$
\begin{align*}
\text{proof} \quad & \\
\text{have} & \quad (\exists \text{peri}_R(P) \parallel M \text{peri}_R(Q))
\begin{align*}
\text{finally show} & \quad \text{thesis}.
\end{align*}
qed

\text{lemma parallel-postcondition [rdes]:}

fixes $M :: (s,t::\text{trace,}\alpha)$ rsp merge

72
assumes $P$ is SRD $Q$ is SRD $M$ is RDM
shows $\text{post}_R(P \parallel M_R(M), Q) = (\text{pre}_R(P \parallel M_R(M), Q) \Rightarrow \text{post}_R(P \parallel M \parallel M_R(P)))$
proof
  have $\text{post}_R(P \parallel M_R(M), Q) =$
    $(\text{pre}_R(P \parallel M_R(M), Q) \Rightarrow (\text{cnt}_R P \parallel (\text{ok} \land N_0(M)) :: H_R[\text{true/false/o\_\_k}, \text{wait}\_\_\_k] \parallel \text{cnt}_R Q))$
    by (simp add: post-cnt-def parallel-commitment assms usubst unrest SRD-health)
  also have ... = $(\text{pre}_R(P \parallel M_R(M), Q) \Rightarrow (\text{cnt}_R P \parallel (\text{\_\_k\_\_k}_0 \land \text{\_\_k\_\_k}_1 \land \text{\_\_k\_\_k}_1) \parallel \text{\_\_k\_\_k}_0(M) :: R_1(true) \parallel \text{\_\_k\_\_k}_1(M) :: R_1(true) \parallel \text{\_\_k\_\_k}_1(M) :: R_1(true) \parallel \text{\_\_k\_\_k}_1(M) :: R_1(true) \parallel \text{\_\_k\_\_k}_1(M) :: R_1(true) )$)
    by (simp add: post-cnt-def parallel-commitment assms usubst unrest LRM-def post-RM-def assms)
  finally show ?thesis .
qed

lemma parallel-precondition-lemma:
fixes $M$ :: (\_\_k\_\_k::trace,\_\_k) rsp merge
assumes $P$ is NSRD $Q$ is NSRD $M$ is RDM
shows $((\text{\_\_k\_\_k}_0(M) :: R_1(true) \parallel \text{\_\_k\_\_k}_1(M) :: R_1(true) \parallel \text{\_\_k\_\_k}_1(M) :: R_1(true) \parallel \text{\_\_k\_\_k}_1(M) :: R_1(true) \parallel \text{\_\_k\_\_k}_1(M) :: R_1(true) )$)
proof
  have $((\text{\_\_k\_\_k}_0(M) :: R_1(true) \parallel \text{\_\_k\_\_k}_1(M) :: R_1(true) \parallel \text{\_\_k\_\_k}_1(M) :: R_1(true) \parallel \text{\_\_k\_\_k}_1(M) :: R_1(true) \parallel \text{\_\_k\_\_k}_1(M) :: R_1(true) )$)
    by (simp add: post-cnt-def parallel-commitment assms usubst unrest SRD-health)
  also have ... = $(\text{\_\_k\_\_k}_0(M) :: R_1(true) \parallel \text{\_\_k\_\_k}_1(M) :: R_1(true) \parallel \text{\_\_k\_\_k}_1(M) :: R_1(true) \parallel \text{\_\_k\_\_k}_1(M) :: R_1(true) \parallel \text{\_\_k\_\_k}_1(M) :: R_1(true) )$)
    by (simp add: post-cnt-def parallel-commitment assms usubst unrest SRD-health)
  also have ... = $(\text{\_\_k\_\_k}_0(M) :: R_1(true) \parallel \text{\_\_k\_\_k}_1(M) :: R_1(true) \parallel \text{\_\_k\_\_k}_1(M) :: R_1(true) \parallel \text{\_\_k\_\_k}_1(M) :: R_1(true) \parallel \text{\_\_k\_\_k}_1(M) :: R_1(true) )$)
    by (simp add: post-cnt-def parallel-commitment assms usubst unrest SRD-health)
  finally show ?thesis .
qed

73
proof

have \( \bar P_1 = ([\neg \nu \text{ pre}_R P]_0 \land [\text{peri}_R Q]_1 \land \$v_\prec = u \$v) ;; (M \land \$wait') ;; R1 \text{ true} \)
  by (simp add: conj-comm)

have \( \bar P_2 = ([\neg \nu \text{ pre}_R P]_0 \land [\text{peri}_R Q]_1 \land \$v_\prec = u \$v) ;; (M \land \$wait') ;; R1 \text{ true} \)
  by (simp add: subst seqr-bool-split[of left-uvar wait], simp-all add: unrest unrest assms closure conj-comm)

also have \( \bar P_1 \lor \bar P_2 = (\bar P_1 \lor \bar P_2) \)
  by (simp add: then-sym)

finally show \( \bar P_1 \lor \bar P_2 \)
  by (simp add: then-sym)

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

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

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

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

lemma nmerge-rd1-merge3:
  assumes \( M \text{ is RDM} \)
  shows \( M_3(N_1(M)) = (\$ok' = u \ ($0-\text{ok} \land \$1-\text{ok} \land \$1-1-\text{ok}) \land \$wait' = u \ ($0-\text{wait} \lor \$1-\text{wait} \lor \$1-1-\text{wait}) \land M_3(M) \)
  proof
    have \( M_3(N_1(M)) = M_3(\$ok' = u \ ($0-\text{ok} \land \$1-\text{ok}) \land \$wait' = u \ ($0-\text{wait} \lor \$1-\text{wait}) \land M_3(M) \)
      by (simp add: nmerge-rd1-def nmerge-rd0-def assms Healthy-if)
    also have \( M_3(\$ok' = u \ ($0-\text{ok} \land \$1-\text{ok}) \land \$wait' = u \ ($0-\text{wait} \lor \$1-\text{wait}) \lor M_3(M) \)
      by (rel-blast)
  qed

74
also have \ldots = (\$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 \textit{nmerge-rd-merge}:
M3(N_{\text{R}}(M)) = (\exists \$st_S \cdot \$v’ = u \$v_S < \$wait_S \lor M3(N_{\text{S}} M) < \$ok_S \lor (\$tr_S \leq u \$tr’)
by (rel-blast)

\textbf{lemma swap-merge-RDM-closed [closure]:}
\textbf{assumes} M is RDM
\textbf{shows} swap_{\text{m}} ;; M is RDM
\textbf{proof –}
\textbf{have} RDM\{\text{swap}_{\text{m}} ;; RDM(M) = (\text{swap}_{\text{m}} ;; \text{RDM}(M))
\textbf{by} (rel-auto)
\textbf{thus} \$thesis
\textbf{by} (metis Healthy-def’ assms)
\qed

\textbf{lemma parallel-precondition:}
\textbf{fixes} M :: (‘s,t::trace, ‘a) resp merge
\textbf{assumes} P is NSRD Q is NSRD M is RDM
\textbf{shows} pre_{\text{R}} P \parallel M_{\text{R}}(M) Q =
\:\begin{align*}
(\sim_r (\sim_r \text{pre}_{\text{R}} P) \parallel M ;; \text{R1(true) peri}_{\text{R}} Q) \land \\
(\sim_r (\sim_r \text{pre}_{\text{R}} P) \parallel M ;; \text{R1(true) post}_{\text{R}} Q) \land \\
(\sim_r (\sim_r \text{pre}_{\text{R}} Q) \parallel (\text{swap}_{\text{m}} ;; M) ;; \text{R1(true) peri}_{\text{R}} P) \land \\
(\sim_r (\sim_r \text{pre}_{\text{R}} Q) \parallel (\text{swap}_{\text{m}} ;; M) ;; \text{R1(true) post}_{\text{R}} P))
\end{align*}
\textbf{proof –}
\textbf{have} a: \sim_r (\sim_r \text{pre}_{\text{R}} P) \parallel N_0(M) ;; \text{R1(true) cmr}_{\text{R}}(Q) =
(\sim_r \text{pre}_{\text{R}} P) \parallel M ;; \text{R1(true) peri}_{\text{R}} Q \lor \sim_r \text{pre}_{\text{R}} P) \parallel M ;; \text{R1(true) post}_{\text{R}} Q)
\textbf{by} (simp add: parallel-precondition-lemma assms)

\textbf{have} b: \sim_r \text{cmr}_{\text{R}} P \parallel N_0 M ;; \text{R1 true} \sim_r \text{pre}_{\text{R}} Q) =
(\sim_r \text{pre}_{\text{R}}(Q)) \parallel N_0(\text{swap}_{\text{m}} ;; M) ;; \text{R1(true) cmr}_{\text{R}}(P))
\textbf{have} c: \sim_r \text{pre}_{\text{R}}(Q)) \parallel N_0(\text{swap}_{\text{m}} ;; M) ;; \text{R1(true) peri}_{\text{R}} P \lor \sim_r \text{pre}_{\text{R}} Q) \parallel (\text{swap}_{\text{m}} ;; M) ;; \text{R1(true) post}_{\text{R}} P)
\textbf{by} (simp add: parallel-precondition-lemma closure assms)

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

\textbf{Weakest Parallel Precondition}

\textbf{definition wr_{R} ::}
(t::trace, ‘a) hrel-rp \\
(t’ :: trace, ‘a) rp merge \\
(t’, ‘a) hrel-rp \\
(t’, ‘a) hrel-rp \sim (wr_{R}(\cdot) [60,0.61] 61)
\textbf{where} [upred-defs]: Q wr_{R}(M) P = (\sim_r ((\sim_r P) \parallel M ;; \text{R1(true) Q}))

\textbf{lemma wr_{R}R1 [closure]:}
\[ M \text{ is } R_{1m} \implies Q \text{ wr } R(M) \text{ P is } R_1 \]
by (simp add: wrR-def closure)

lemma \textit{R2-rea-not}: \[ R_2(\neg_r \ P) = (\neg_r \ R_2(P)) \]
by (rel-auto)

lemma \textit{wrR-R2-lemma}:
  assumes \[ P \text{ is } R_2 \text{ Q is } R_2 \text{ M is } R_{2m} \]
  shows \[ \((\neg_r \ P) \parallel M \ Q) ;; R_1(\text{true}_h) \text{ is } R_2 \]
proof
  have \[ (\neg_r \ P) \parallel M \ Q \text{ is } R_2 \]
  by (simp add: closure assms)
  thus \thesis
  by (simp add: closure)
qed

lemma \textit{wrR-R2} [closure]:
  assumes \[ P \text{ is } R_2 \text{ Q is } R_2 \text{ M is } R_{2m} \]
  shows \[ Q \text{ wr } R(M) \text{ P is } R_2 \]
proof
  have \[((\neg_r \ P) \parallel M \ Q) ;; R_1(\text{true}_h) \text{ is } R_2 \]
  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 \textit{wrR-RR} [closure]:
  assumes \[ P \text{ is } R_{RR} \text{ Q is } R_{RR} \text{ M is } R_{DM} \]
  shows \[ Q \text{ wr } R(M) \text{ P is } R_{RR} \]
apply (rule RR-intro)
apply (simp-all add: unrest assms closure wrR-def rpred)
apply (metis (no-types, lifting) Healthy-def' R1-R2c-commute R1-R2c-is-R2 R1-rea-not RDM-R2m
 RR-implies-R2 assms (1) assms (2) assms (3) par-by-merge-seq-add rea-not-R2-closed
 wrR-R2-lemma)
done

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

lemma \textit{wppR-choice} [wp]: \[ (P \lor Q) \text{ wr } R(M) \text{ R = (P \ wr } R(M) \text{ R \land Q \ wr } R(M) \text{ R) \]
proof
  have \[ (P \lor Q) \text{ wr } R(M) \text{ R =} \]
  \[ (\neg_r (\neg_r R) ;; U_0 \land (P ;; U_1 \lor Q ;; U_1) \land \$v_\lt< =_u \$v) ;; M ;; true_r) \]
  by (simp add: wrR-def par-by-merge-def seqr-or-distl)
  also have \[ ... = (\neg_r (\neg_r R) ;; U_0 \land P ;; U_1 \land \$v_\lt< =_u \$v \lor (\neg_r R) ;; U_0 \land Q ;; U_1 \land \$v_\lt< =_u \$v) ;; M ;; true_r \]
  by (simp add: conj-disj-distr wt-pred-laws.inf-sup-distrib2)
  also have \[ ... = (\neg_r ((\neg_r R) ;; U_0 \land P ;; U_1 \land \$v_\lt< =_u \$v) ;; M ;; true_r \land \]
  \((\neg_r R) ;; U_0 \land Q ;; U_1 \land \$v_\lt< =_u \$v) ;; M ;; true_r) \]
  by (simp add: conj-disj-distr wt-pred-laws.inf-sup-distrib2)

76
proof
also have ... = (P wrR(M) R ∧ Q wrR(M) R)
  by (simp add: wrR-def par-by-merge-def)
finally show thesis.
qed

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

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

lemma parallel-precondition-wr [rdes];
assumes P is NSRD Q is NSRD M is RDM
shows pre_R(P) || M wrR(M) Q) = (peri_R(Q) wrR(M) pre_R(P) ∧ post_R(Q) wrR(M) pre_R(P) ∧
peri_R(P) wrR(swap_m :: M) pre_R(Q) ∧ post_R(P) wrR(swap_m :: M) pre_R(Q))
by (simp add: assms parallel-precondition-wrR-def)

lemma parallel-rdes-def [rdes-def];
assumes P_1 is RC P_2 is RR P_3 is RR Q_1 is RC Q_2 is RR Q_3 is RR
  $st^+ \parallel P_2 \parallel Q_2 \parallel Q_2 \parallel Q_2 \parallel Q_2 M is RDM
shows R_3(P_1 \Rightarrow P_2 \Rightarrow P_3) || M wrR(M) R_3(Q_1 \Rightarrow Q_2 \Rightarrow Q_2)
= R_3(((Q_1 \Rightarrow Q_2) wrR(M) P_1 \Rightarrow (Q_1 \Rightarrow Q_3) wrR(M) P_1 \Rightarrow
  (P_1 \Rightarrow P_2) wrR(swap_m :: M) Q_1 \Rightarrow (P_1 \Rightarrow P_3) wrR(swap_m :: M) Q_1) \\ 
  ((P_1 \Rightarrow P_2) \parallel M (Q_1 \Rightarrow Q_2) \parallel (P_1 \Rightarrow P_2) \parallel M (Q_1 \Rightarrow Q_3)) \parallel
  ((P_1 \Rightarrow P_3) \parallel M (Q_1 \Rightarrow Q_3)) (is ?lhs = ?rhs)
proof –
  have ?lhs = R_3 (pre_R ?lhs \parallel peri_R ?lhs \parallel post_R ?lhs)
    by (simp add: SRD-reactive-tri-design assms closure)
also have ... = ?rhs
  by (simp add: rdes closure unrest assms, rel-auto)
finally show thesis.
qed

lemma Miracle-parallel-left-zero;
assumes P is SRD M is RDM
shows Miracle || M wrM P = Miracle
proof –
  have pre_R(Miracle || M wrM P) = true_r
    by (simp add: parallel-assm wait'-cond-idem rdes closure assms)
moreover hence cmt_R(Miracle || M wrM P) = false
    by (simp add: rdes closure wait'-cond-idem SRD-healths assms)
ultimately have Miracle || M wrM P = R_3(true_r \parallel false)
    by (metis NSRD-iff SRD-reactive-design-alt assms par-rdes-NSRD srdes-theory-continuous.weak.top-closed)
thus thesis
    by (simp add: Miracle-def R1-design-R1-pre)
qed

lemma Miracle-parallel-right-zero;
assumes P is SRD M is RDM
shows P || M Miracle = Miracle
proof –
  have pre_R(P || M Miracle) = true_r
    by (simp add: parallel-assm wait'-cond-idem rdes closure assms)
moreover have ... = (P wrR(M) R \parallel Q wrR(M) R)
  by (simp add: wrR-def par-by-merge-def)
ultimately have ... = (P wrR(M) R \parallel Q wrR(M) R)
    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
by (simp add: wait'·cond-idem parallel-assm rdes closure assms)
moreover hence \( cmt_R(P \parallel_R M) = \text{false} \)
  by (simp add: wait'·cond-idem rdes closure SRD-healths assms)
ultimately have \( P \parallel_R M \text{ Miracle} = R_s(\text{true, } \top, \text{ false}) \)
by (metis NSRD-iff SRD-reactive-design-alt assms par-rdes-NSRD srdes-theory-continuous.weak.top-closed)
thus \( \text{thesis} \)
  by (simp add: Miracle-def R1-design-R1-pre)
qed

8.1 Example basic merge

definition BasicMerge :: \( ((\prime s, \prime t::trace, \prime unit) \text{ rsp}) \text{ merge } (N_B) \) where
\[ \text{upred-defs: BasicMerge} = (\exists \text{str'} < \text{u} \land \text{str'} - \text{str} = \text{u} \land - \text{str} < \text{u} \land \text{str'} - \text{str} = \text{u} \land - \text{str} < \text{u} \land - \text{str} < \text{u} \land 1 - \text{tr} - \text{str} < \text{u} \land \text{str'} = \text{u} \land \text{str} < \text{u}) \]

abbreviation rbasic-par \( (\cdot |B - [85,86] | 85) \) where
\( P |B Q = P | M_R(N_B) Q \)

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

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

lemma BasicMerge'·calc:
  assumes \( \text{sok'} \equiv P \parallel_R \text{wait'} \equiv P \parallel_R \text{ok'} \equiv Q \parallel \text{wait'} \equiv Q \parallel P \) is \( R2 \) \( Q \) is \( R2 \)
  shows \( P |N_B Q = (\exists \text{st'} \cdot P) \land (\exists \text{st'} \cdot Q) \land \text{st'} = \text{u} \land \text{st} \)
  using assms
proof –
  have \( P; (\exists \text{sok'} \cdot \text{wait'}) \cdot R2(P) = P \) \( \text{is} \ ?P' = \prime \cdot \)
    by (simp add: ex-unrest ex-plus Healthy-if assms)
  have \( Q; (\exists \text{sok'} \cdot \text{wait'}) \cdot R2(Q) = Q \) \( \text{is} \ ?Q' = \prime \cdot \)
    by (simp add: ex-unrest ex-plus Healthy-if assms)
  have \( \prime \cdot \parallel_R \text{N}_{N_B} \prime \cdot Q' = ((\exists \text{st'} \cdot \prime \cdot P') \land (\exists \text{st'} \cdot \prime \cdot Q') \land \text{st'} = \text{u} \land \text{st}) \)
    by (simp add: par-by-merge-alt-def, rel-auto, blast+)
  thus \( \text{thesis} \)
    by (simp add: P Q)
qed

8.2 Simple parallel composition

definition rea-design-par ::
(\prime s, \prime t::trace, \prime \alpha) hrel-rsp \Rightarrow (\prime s, \prime t, \prime \alpha) hrel-rsp \Rightarrow (\prime s, \prime t, \prime \alpha) hrel-rsp \text{ (infixr } |R 85)\]
where \( \text{upred-defs: P |}_R Q = R_s(\text{pre}_R(P) \land \text{pre}_R(Q)) \lor (\text{cmt}_R(P) \land \text{cmt}_R(Q)) \)

lemma RHS-design-par:
  assumes \( \text{sok'} \equiv P \parallel_R \text{sok'} \equiv P_2 \)
  shows \( R_s(P_1 \parallel_R Q_1) |R R_s(P_2 \parallel_R Q_2) = R_s((P_1 \land P_2) \parallel_R (Q_1 \land Q_2)) \)
proof –
  have \( R_s(P_1 \parallel_R Q_1) |R R_s(P_2 \parallel_R Q_2) = \)
    \( R_s((P_1 \parallel_R Q_1) |R Q_2) \)
    by (simp add: RHS-design-ok-wait)
from assms
have ... =
  \(R_s \langle R1 \ (R2c \ (P_1) \land R1 \ (R2c \ (P_2)) \rangle \rightarrow\)
  \((R1 \ (R2c \ (P_1) \Rightarrow Q_1)) \land R1 \ (R2c \ (P_2) \Rightarrow Q_2))\)
apply simp add: rea-design-par-def rea-pre-RHS-design rea-cmt-RHS-design unsubst unrest assms
apply (rule cong[of \(R_s \ R_s\), simp])
using assms apply (rel-auto)
done
also have ... =
  \(R_s \langle R1 \ (R2c \ (P_1) \land R2c \ (P_2)) \rangle \rightarrow\)
  \((R1 \ (R2s \ (P_1) \Rightarrow Q_1)) \land R1 \ (R2s \ (P_2) \Rightarrow Q_2))\)
by (metis (no-types, hide-lams) R1-R2s-R2c-R1-conj R1-design-R1-pre RHS-design-ok-wait)
also have ... = \(R_s \langle (P_1 \land P_2) \rightarrow ((P_1 \Rightarrow Q_1) \land (P_2 \Rightarrow Q_2)) \rangle\)
by (metis (no-types, lifting) R1-conj R2s-conj RHS-design-export-R1 RHS-design-export-R2s)
also have ... = \(R_s \langle (P_1 \land P_2) \rightarrow (Q_1 \land Q_2) \rangle\)
by (rule cong[of \(R_s \ R_s\), simp, rel-auto]
finally show \("\theta\) thesis .
qed

lemma RHS-tri-design-par:
  assumes \$\text{ok} \not\equiv P_1 \not\equiv P_2$
  shows \(R_s \langle P_1 \Rightarrow Q_1 \land R_1 \parallel R_s \langle P_2 \Rightarrow Q_2 \rangle \rangle \Rightarrow\)
  \(R_s \parallel (P_1 \land P_2) \Rightarrow (Q_1 \land Q_2) \Rightarrow (R_1 \land R_2)\)
by (simp add: RHS-design-par assms unrest wait\'-cond-conj-exchange)

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

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

end

9 Productive Reactive Designs

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

9.1 Healthiness condition

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

definition Productive :: ('s, 't::trace, 'a) hrel-rsp ⇒ ('s, 't, 'a) hrel-rsp where
  [upred-defs]: Productive(P) = P \parallel_R \text{RR} \parallel_R (true \Rightarrow true \diamond (\text{\$tr} < \text{\$tr}'))

lemma Productive-RHS-design-form:
  assumes \$\text{ok} \not\equiv P \not\equiv Q \not\equiv R$

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

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

lemma Productive-form:
$\text{Productive}(\text{SRD}(P)) = R_u(\text{peri}_R(P) \land \text{post}_R(P) \land \$tr <_u \$tr'))$

proof 
  have $\text{Productive}(\text{SRD}(P)) = R_u(\text{peri}_R(P) \land \text{post}_R(P)) \parallel R_u(\text{true} \land \$tr <_u \$tr'))$
    by (simp add: Productive-def SRD-as-reactive-tri-design)
  also have ... = $R_u(\text{peri}_R(P) \land \text{post}_R(P) \land \$tr <_u \$tr'))$
    by (simp add: RHS-tri-design-par unrest)
  finally show ?thesis .
qed

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

lemma Productive-intro:
assumes $P$ is SRD $(\$tr <_u \$tr') \subseteq (\text{peri}_R(P) \land \text{post}_R(P)) \parallel \text{wait} \land \neg \text{peri}_R(P)$
shows $P$ is Productive

proof 
  have $P.R_u(\text{peri}_R(P) \land \text{post}_R(P)) = P$
  proof 
    have $R_u(\text{peri}_R(P) \land \text{post}_R(P)) = R_u(\text{peri}_R(P) \land \text{peri}_R(P)) \land (\text{peri}_R(P) \land \text{post}_R(P))$
      by (metis (no-types, hide-lams) design-export-pre wait'-cond-conj-exchange wait'-cond-idem)
    also have ... = $R_u(\text{peri}_R(P) \land \text{peri}_R(P)) \land (\text{peri}_R(P) \land \text{post}_R(P) \land \$tr <_u \$tr'))$
      by (metis assms(2) utp-pred-laws.inf.absorb1 utp-pred-laws.inf_assoc)
    also have ... = $R_u(\text{peri}_R(P) \land \text{post}_R(P) \land \$tr <_u \$tr'))$
      by (metis (no-types, hide-lams) design-export-pre wait'-cond-conj-exchange wait'-cond-idem)
  finally show ?thesis 
    by (simp add: SRD-reactive-tri-design assms(1))
  qed
  thus ?thesis 
    by (metis Healthy-def RHS-tri-design-par Productive-def ok'-pre-unrest unrest-true utp-pred-laws.inf-right-idem utp-pred-laws.inf-top-right)
  qed

lemma Productive-refines-tr-increase:
assumes $P$ is SRD $P$ is Productive \$wait' \land \neg \text{peri}_R(P)$
shows $(\$tr <_u \$tr') \subseteq (\text{peri}_R(P) \land \text{post}_R(P))$

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

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

80
9.2 Reactive design calculations

**Lemma preR-Productive [rdes]:**
- **Asumes** $P$ is SRD
- **Shows** $\text{pre}_R(\text{Productive}(P)) = \text{pre}_R(P)$
  - **Proof**
    - **Have** $\text{pre}_R(\text{Productive}(P)) = \text{pre}_R(\text{peri}_R(\text{productive}(P)) \vee \text{post}_R(P) \wedge \text{str} < u \cdot \text{str}'))$
    - **By** (metis Healthy-def Productive-form assms)
    - **Thus** $\neg \text{thesis}$
      - **By** (simp add: rea-RHS-design usubst unrest R2c-not preR R1-preR Healthy-if assms)
  - **Qed**

**Lemma periR-Productive [rdes]:**
- **Asumes** $P$ is NSRD
- **Shows** $\text{peri}_R(\text{Productive}(P)) = \text{peri}_R(P)$
  - **Proof**
    - **Have** $\text{peri}_R(\text{Productive}(P)) = \text{peri}_R(\text{peri}_R(\text{productive}(P)) \vee \text{post}_R(P) \wedge \text{str} < u \cdot \text{str}'))$
    - **By** (metis Healthy-def NSRD-is-SRD Productive-form assms)
    - **Also Have** $\neg \text{thesis}$
      - **By** (simp add: rea-peri-RHS-design usubst unrest R2c-not assms)
  - **Qed**

**Lemma postR-Productive [rdes]:**
- **Asumes** $P$ is NSRD
- **Shows** $\text{post}_R(\text{Productive}(P)) = (\text{pre}_R(P) \Rightarrow \text{post}_R(P) \wedge \text{str} < u \cdot \text{str}'))$
  - **Proof**
    - **Have** $\text{post}_R(\text{Productive}(P)) = \text{post}_R(\text{peri}_R(\text{productive}(P)) \vee \text{post}_R(P) \wedge \text{str} < u \cdot \text{str}'))$
    - **By** (metis Healthy-def NSRD-is-SRD Productive-form assms)
    - **Also Have** $\neg \text{thesis}$
      - **By** (simp add: rea-RHS-design usubst unrest assms)
  - **Qed**

**Lemma preR-frame-seq-export:**
- **Asumes** $P$ is NSRD $P$ is Productive $Q$ is NSRD
- **Shows** $((\text{pre}_R P \land (\text{pre}_R P \land \text{post}_R P) :: Q)) = (\text{pre}_R P \land (\text{post}_R P :: Q))$
  - **Proof**
    - **Have** $((\text{pre}_R P \land (\text{pre}_R P :: Q)) = (\text{pre}_R P \land (\text{pre}_R P \Rightarrow \text{post}_R P :: Q))$
      - **By** (simp add: R1-post-under-pre assms closure unrest)
    - **Also Have** $\neg \text{thesis}$
      - **By** (simp add: NSRD-is-SRD R1-post-SRD assms(1) rea-impl-def seqr-or-distl R1-preR Healthy-if)
  - **Qed**

**proof**
- **Have** $(\text{pre}_R P \Rightarrow \text{pre}_R P) = \text{R1 TRUE}$
  - **By** (simp add: R1-preR rea-not-or)
- **Then Show** $\neg \text{thesis}$
  - **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
9.3 Closure laws

**Lemma: Productive-rdes-intro**
assumes $(\text{str} < u \text{str}^-) \subseteq R \\text{ok}^- \land P \\text{ok}^- \land Q \\text{ok}^- \land R \\text{wait}^- \land P \\text{wait}^- \land P$
shows $(R_n (P \triangleright Q \circ R))$ is Productive
proof (rule Productive-intro)
show $R_n (P \triangleright Q \circ R)$ is SRD
by (simp add: RHS-tri-design-is-SRD assms)

from assms(1) show $(\text{str}^- >_u \text{str}) \subseteq (\text{pre}_R (R_n (P \triangleright Q \circ R)) \land \text{post}_R (R_n (P \triangleright Q \circ R)))$
apply (simp add: rea-pre-RHS-design rea-post-RHS-design subst assms unrest)
apply fastforce
done

show $\text{wait}^- \land \text{pre}_R (R_n (P \triangleright Q \circ R))$
by (simp add: rea-pre-RHS-design rea-post-RHS-design subst R1-def R2c-def R2s-def assms unrest)

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

**Lemma: Productive-rdes-RR-intro**
assumes $P$ is RR $Q$ is RR $R$ is RR $R$ is $R_4$
shows $(R_n (P \triangleright Q \circ R))$ is Productive
using assms by (simp add: Productive-rdes-intro $R_4$-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 $P \land Q$ is Productive
proof -
have $P \land Q =$
  $(R_n (\text{pre}_R (P) \triangleright \text{peri}_R (P) \circ (\text{post}_R (P) \land \text{str} <_u \text{str}^-)) \cap R_n (\text{pre}_R (Q) \triangleright \text{peri}_R (Q) \circ (\text{post}_R (Q) \land \text{str} <_u \text{str}^-))$
  by (metis Healthy-if Productive-form assms)
also have ... = $R_n ((\text{pre}_R P \land \text{pre}_R Q) \triangleright (\text{peri}_R P \lor \text{peri}_R Q) \circ ((\text{post}_R P \land \text{str}^- >_u \text{str}) \lor (\text{post}_R Q \land \text{str}^- >_u \text{str}^-)))$
  by (simp add: RHS-tri-design-choice)
also have ... = $R_n ((\text{pre}_R P \land \text{pre}_R Q) \triangleright (\text{peri}_R P \lor \text{peri}_R Q) \circ (((\text{post}_R P) \lor (\text{post}_R Q)) \land \text{str}^- >_u \text{str}))$

82
by (rule cong[of $R_u$, $R_u$], 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 $P \circ b \triangleright_R Q$ is Productive
proof –
  have $P \circ b \triangleright_R Q =$
    $R_u (\text{pre}_R(P) \vdash \text{peri}_R(P) \circ (\text{post}_R(P) \land \text{str} <_u \text{str}')) \circ b \triangleright_R R_u (\text{pre}_R(Q) \vdash \text{peri}_R(Q) \circ (\text{post}_R(Q) \land \text{str} <_u \text{str}'))$
    by (metis Healthy-if Productive-rdes-intro assms closure unrest assms)
  also have ... = $R_u ((\text{pre}_R P \circ b \triangleright_R \text{pre}_R Q) \vdash (\text{peri}_R P \circ b \triangleright_R \text{peri}_R Q) \circ ((\text{post}_R P \land \text{str} >_u \text{str}) \circ b \triangleright_R (\text{post}_R Q) \land \text{str} >_u \text{str}))$
    by (rule cong[of $R_u$, $R_u$], simp, rel-auto)
  finally show ... is Productive.
qed

lemma Productive-seq-1 [closure]:
  assumes P is NSRD P is Productive Q is NSRD
  shows $P ; Q$ is Productive
proof –
  have $P ; : Q = R_u (\text{pre}_R(P) \vdash \text{peri}_R(P) \circ (\text{post}_R(P) \land \text{str} <_u \text{str}')) ;: R_u (\text{pre}_R(Q) \vdash \text{peri}_R(Q) \circ (\text{post}_R(Q)))$
    by (metis Healthy-def NSRD-is-SRD SRD-reactive-tri-design Productive-form assms)
  also have ... = $R_u ((\text{pre}_R P \land (\text{post}_R P \land \text{str} >_u \text{str}) \text{wp}_R \text{pre}_R Q) \vdash$
    $(\text{peri}_R P \lor ((\text{post}_R P \land \text{str} >_u \text{str}) ;: \text{peri}_R Q)) \circ ((\text{post}_R P \land \text{str} >_u \text{str}) ;: \text{post}_R Q)$
    by (simp add: RHS-tri-design-composition-wp rpred unrest closure assms NSRD-neg-pre-left-zero SRD-healths ex-unrest wp-redef disg-apred-def)
  also have ... = $R_u ((\text{pre}_R P \land (\text{post}_R P \land \text{str} >_u \text{str}) \text{wp}_R \text{pre}_R Q) \vdash$
    $(\text{peri}_R P \lor ((\text{post}_R P \land \text{str} >_u \text{str}) ;: \text{peri}_R Q)) \circ ((\text{post}_R P \land \text{str} >_u \text{str}) ;: \text{post}_R Q \land \text{str} >_u \text{str})$
    by (rel-auto)
  thus ?thesis
    by (simp add: NSRD-is-SRD R1-post-SRD assms)
qed

also have ... is Productive
  by (rule Productive-rdes-intro, simp-all add: unrest assms closure wp-redef)
finally show ?thesis.
qed

lemma Productive-seq-2 [closure]:
  assumes P is NSRD Q is NSRD Q is Productive

shows $P :: Q$ is Productive

proof –
  have $P :: Q = R_s(prem(P) \circ perim(P) \circ (post(P)) :: R_s(prem(Q) \circ perim(Q) \circ (post(Q) \land \exists tr < u \exists tr'))$
    by (metis Healthy-def NSRD-is-SRD SRD-reactive-tri-design Productive-form assms)
  also have ... = $R_s((pre(P) \land post(P) \circ prem(P)) \circ (pre(Q) \lor (post(P) :: perim(Q)) \circ (post(P) :: (post(Q) \land \exists tr < u \exists tr')))$
    by (simp add: RHS-tri-design-composition-wp rpred unrest closure assms wp N SRD-neg-pre-left-zero SRD-healths ex-unrest wp-rea-def disj-upred-def)
  also have ... = $R_s((pre(P) \land post(P) \circ prem(P)) \circ (pre(Q) \lor (post(P) :: perim(Q)) \circ (post(P) :: (post(Q) \land \exists tr < u \exists tr')))$
    by (simp add: NSRD-is-SRD R1-post-SRD assms)
  qed
also have ... is Productive
  by (rule Productive-rdes-intro, simp-add: unrest assms closure wp-rea-def)
finally show ?thesis .
qed

end

10 Guarded Recursion

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

10.1 Traces with a size measure

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

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

begin

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

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

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

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

end

Both natural numbers and lists are measurable trace algebras.

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

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

syntax -usize :: logic \( \Rightarrow \) logic (sizeu('\_'))
translations sizeu(t) == CONST uop CONST size t

10.2 Guardianedness

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

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

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

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

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

Guarded reactive designs yield unique fixed-points.

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

have $\mu F$ is R1
  by (rule SRD-healths(1), rule Healthy-mu, simp-all add: assms)
thus $\langle$thesis
by (metis Healthy-def R1-def conj-comm)
qed
moreover have ($str \leq_u \langlestr' \land \nu F = \nu F$
proof --
  have $\nu F$ is R1
  by (rule SRD-healths(1), rule Healthy-nu, simp-all add: assms)
thus $\langle$thesis
by (metis Healthy-def R1-def conj-comm)
qed
ultimately show $\langle$thesis
by (simp)
qed

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

lemma UINF-Guarded [closure]:
assumes $\forall P. P \in A \Rightarrow \text{Guarded } P$
shows Guarded ($\lambda X. \forall P \in A \cdot P(X)$)
proof (rule GuardedI)
  fix $X$ $n$
  have $\forall Y. (\forall P \in A \cdot P Y \land \text{gvr}(n+1)) = (\forall P \in A \cdot (P Y \land \text{gvr}(n+1))) \land \text{gvr}(n+1))$
proof --
  fix $Y$
  let $\langlelhs = (\forall P \in A \cdot P Y \land \text{gvr}(n+1))$ and $\langlerhs = (\forall P \in A \cdot (P Y \land \text{gvr}(n+1))) \land \text{gvr}(n+1))$
  have a:$\langlelhs[\text{false}/$ok] = $\langlerhs[\text{false}/$ok]
    by (rel-auto)
  have b:$\langlelhs[\text{true}/$ok][\text{true}/$wait] = $\langlerhs[\text{true}/$ok][\text{true}/$wait]
    by (rel-auto)
  have c:$\langlelhs[\text{true}/$ok][\text{false}/$wait] = $\langlerhs[\text{true}/$ok][\text{false}/$wait]
    by (rel-auto)
  show $\langlelhs = $\langlerhs
    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 $(\forall P \in A \cdot (P X \land \text{gvr}(n+1))) \land \text{gvr}(n+1)) = (\forall P \in A \cdot (P (X \land \text{gvr}(n))) \land \text{gvr}(n+1)))) \land \text{gvr}(n+1))$
proof --
  have $(\forall P \in A \cdot (P X \land \text{gvr}(n+1))) = (\forall P \in A \cdot (P (X \land \text{gvr}(n))) \land \text{gvr}(n+1))))$
  proof (rule UINF-cong)
    fix $P$ assume $P \in A$
    thus $(P X \land \text{gvr}(n+1)) = (P (X \land \text{gvr}(n))) \land \text{gvr}(n+1))$
      using Guarded-def assms by blast
  qed
  thus $\langle$thesis by simp
qed
ultimately show $(\forall P \in A \cdot P X) \land \text{gvr}(n+1)) = (\forall P \in A \cdot (P (X \land \text{gvr}(n)))) \land \text{gvr}(n+1))$
by simp
qed

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

86
Proof: We split the proof into three cases corresponding to valuations for ok, wait, and wait’ respectively.

We prove the guarded property for each case separately:

**Case 1: ok**

1. Assume \( \lambda X. P(X) \sqsubseteq Q(X) \)
2. By (rule UINF-Guarded, auto simp add: assms)
3. Thus \( \text{thesis} \)
4. By (simp)

**Case 2: wait**

1. Assume \( \lambda X. P(X) \sqsubseteq Q(X) \)
2. By (rule UINF-Guarded, auto simp add: assms)
3. Thus \( \text{thesis} \)
4. By (simp)

**Case 3: wait’**

1. Assume \( \lambda X. P(X) \sqsubseteq Q(X) \)
2. By (rule UINF-Guarded, auto simp add: assms)
3. Thus \( \text{thesis} \)
4. By (simp)

Qed
∧ gert (Suc n))(true,false/$ok,$wait)]
  by (simp add: impl-all-def R2c-disj R1-disj R2c-not assms closure R2c-and
       R2c-preR rea-not-def R1-extend-conj' R2c-ok' R2c-post-SRD R1-tr-less-tr' R2c-tr-less-tr')
also have "(...) =
  (((放弃了 pre R P) ;; (SRD(Y))[false/$wait] ∨ ($ok' ∧ post R P ∧ $tr' > u $tr) ;; (SRD Y)[false/$wait]))) ∧ gert (Suc n))][true,false/$ok,$wait]
  by (simp add: usubst unrest assms closure seq-or-distl NSRD-neg-pre-left-zero SRD-healths)
also have "(...) =
  (((放弃了 pre R P) ;; (SRD(Y))[false/$wait] ∨ (post R P ∧ $tr' > u $tr) ;; (SRD Y)[true,false/$ok,$wait]))
∧ gert (Suc n))][true,false/$ok,$wait]
proof –
  have ($ok' ∧ post R P ∧ $tr' > u $tr) ;; (SRD Y)[false/$wait] =
    ((post R P ∧ $tr' > u $tr) ∧ $ok' = u true) ;; (SRD Y)[false/$wait]
  by (rel-blast)
also have "(...) = (post R P ∧ $tr' > u $tr)[true/$ok'] ;; (SRD Y)[false/$wait][true/$ok]
using seqr-left-one-point[of ok (post R P ∧ $tr' > u $tr) True (SRD Y)[false/$wait]]
  by (simp add: true-alt-def)[THEN sym])
finally show ?thesis by (simp add: usubst unrest)
qed
finally
  show (P)[false/$wait`] ;; (SRD Y)[false/$wait] ∧ gert (Suc n)][true,false/$ok,$wait] =
    (((放弃了 pre R P) ;; (SRD(Y))[false/$wait] ∨ (post R P ∧ $tr' > u $tr) ;; (SRD Y)[true,false/$ok,$wait])
∧ gert (Suc n))][true,false/$ok,$wait]].
qed
have 1:((post R P ∧ $tr' > u $tr) ;; (SRD X)[true,false/$ok,$wait] ∧ gert (Suc n)) =
  ((post R P ∧ $tr' > u $tr) ;; (SRD (X ∧ gert n)))[true,false/$ok,$wait] ∧ gert (Suc n))
apply (rel-auto)
apply (rename-tac tr st more ok wait tr' st' more' tr0 st0 more0 ok')
apply (rule-tac x=tr0 in exI, rule-tac x=st0 in exI, rule-tac x=more0 in exI)
apply (simp)
apply (erule trace-strict-prefixE)
apply (rename-tac tr st ref ok wait tr' st' ref' tr0 st0 ref0 ok' zs)
apply (rule-tac x=False in exI)
apply (simp add: size-minus-trace)
apply (subgoal-tac size(tr) < size(tr0))
apply (simp add: less-diff-conv2 size-mono)
using size-strict-mono apply blast
apply (rename-tac tr st more ok wait tr' st' more' tr0 st0 more0 ok')
apply (rule-tac x=tr0 in exI, rule-tac x=st0 in exI, rule-tac x=more0 in exI)
apply (simp)
apply (erule trace-strict-prefixE)
apply (rename-tac tr st more ok wait tr' st' more' tr0 st0 more0 ok' zs)
apply (auto simp add: size-minus-trace)
apply (subgoal-tac size(tr) < size(tr0))
apply (simp add: less-diff-conv2 size-mono)
using size-strict-mono apply blast
done
have 2:(放弃了 pre R P) ;; (SRD(X)[false/$wait] = (放弃了 pre R P) ;; (SRD(X ∧ gert n))[false/$wait]
  by (simp add: NSRD-neg-pre-left-zero closure assms SRD-healths)
show ?thesis
  by (simp add: exp 1 2 utp-pred-laws.inf-sup-distrib2)
qed
lemma mu-csp-form-1 [rdes]:
fixes P :: (\'s, 't::size-trace,'a) hrel-rsp
assumes P is NSRD P is Productive
shows (\(\mu X \cdot P ;; SRD(X)\)) = (\(\prod i \cdot P \sim (i+1)\)) ;; Miracle

proof –

have 1:Continuous (\(\lambda X. P ;; SRD X\))
  using SRD-Continuous
  by (clarsimp simp add: Continuous-def seq-SUP-distl[THEN sym], drule-tac x=A in spec, simp)

have 2: (\(\lambda X. P ;; SRD X\) \in [[id]]_H \rightarrow [[SRD]]_H)
  by (blast intro: funcsetI closure assms)

with 1 2 have (\(\mu X \cdot P ;; SRD(X)\)) = (\(\nu X \cdot P ;; SRD(X)\))
  by (simp add: guarded-fp-uniq Guarded-if-Productive[OF assms] funcsetI closure)

also have ... = (\(\lambda X. P ;; SRD(X) \sim 0\) false \sqinter ((\(i \cdot (\lambda X. P ;; SRD X) \sim (i+1))\) false)
  by (simp add: sup-continuous-lfp 1 sup-continuous-Continuous false-uptp-def)

also have ... = (\(\prod i. ((\lambda X. P ;; SRD X) \sim (i+1))\) false)
  by (subgoal_tac Sup-power-expand, simp)

also have ... = (\(\prod i. P \sim (i+1)\)) ;; Miracle
  by (simp)

proof (rule SUP-cong, simp-all)
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(1) seqr-assoc[THEN sym] srdes-theory-continuous.weak.top-closed)
  qed
also have ...
    = (∏ i ∈ UNIV . P ^ (i + 1)) ;; Miracle
    by (simp add: seq-Sup-distr)
finally show ?thesis
  by (simp only: seq-UINF-distl[THEN sym], simp add: ustar-def)
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)) = (∏ i ∈ UNIV . P ;; P ^ i) ;; Miracle
      by (simp add: mu-csp-form-1 assms closure ustar-def)
    also have ...
      = (P ;; P^*) ;; Miracle
      by (simp only: seq-UINF-distl[THEN sym], simp add: ustar-def)
    finally show ?thesis .
  qed

declare upred-semiring.power-Suc [simp del]
end

11 Reactive Design Programs
theory utp-rdes-prog
imports
tup-rdes-normal
tup-rdes-tactics
tup-rdes-parallel
tup-rdes-guarded
UTP−KAT.utp-kleene
begin

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

by (rel-auto)

**lemma** srd-subst-SRD-closed [closure]:

assumes \( P \) is SRD

shows \([\sigma]_{S\sigma} \vdash P \) is SRD

**proof** –

have SRD \(([\sigma]_{S\sigma} \vdash (SRD P)) = \([\sigma]_{S\sigma} \vdash (SRD P)\)

by (rel-auto)

thus ?thesis

by (metis Healthy-def assms)

qed

**lemma** preR-srd-subst [rdes]:

\( \text{pre}_R([\sigma]_{S\sigma} \vdash P) = [\sigma]_{S\sigma} \vdash \text{pre}_R(P) \)

by (rel-auto)

**definition** assigns-srd :: ‘s usubst => (‘s, ‘t::trace) hrel-rsp ((‘_)_R) where

[upred-defs]: assigns-srd σ = \( \text{R}_s((\text{true} \mathbin{\vdash} (\text{false} \circ \langle [\sigma]_S \rangle) \mathbin{\wedge} \text{wait} \mathbin{\wedge} [\langle(\sigma)]_S \mathbin{\wedge} \Sigma_S \mathbin{=} \Sigma_S)) \)

**syntax**

- assigns-srd :: svids => uexprs => logic (‘(‘_):=\( \text{R} \) ‘(‘_))

- assigns-srd :: svids => uexprs => logic (infixr :=\( \text{R} \) 90)

**translations**

- assigns-srd xs vs => \( \text{CONST} \) assigns-srd (-mk-usubst \( \text{CONST} \) id) xs vs

- assigns-srd x v <= \( \text{CONST} \) assigns-srd \( \text{CONST} \) subst-upd \( \text{CONST} \) id) x v

- assigns-srd x v <= -assign-srd (-svar x) v

x,y :=\( \text{R} \) u,v <= \( \text{CONST} \) assigns-srd \( \text{CONST} \) subst-upd \( \text{CONST} \) subst-upd \( \text{CONST} \) id) (\( \text{CONST} \) svar x) u) (\( \text{CONST} \) svar y) v

**lemma** assigns-srd-RHS-tri-des [rdes-def]:

\( \langle [\sigma] \rangle_R = \text{R}_s((\text{true} \mathbin{\vdash} \text{false} \circ \langle [\sigma]_S \rangle) \)

by (rel-auto)

**lemma** assigns-srd-NSRD-closed [closure]: \( \langle [\sigma] \rangle_R \) is NSRD

by (simp add: rdes-def closure unrest)

**lemma** preR-assigns-srd [rdes]: \( \text{pre}_R(\langle [\sigma] \rangle_R) = \text{true} \)

by (simp add: rdes-def rdes closure)
lemma periR-assigns-srd [rdes]: peri(⟨σ⟩) = false
    by (simp add: rdes-def rdes closure)

lemma postR-assigns-srd [rdes]: post(⟨σ⟩) = ⟨σ⟩r
    by (simp add: rdes-def rdes closure rpred)

11.3 Conditional

lemma preR-cond-srea [rdes]:
    pre(P ⊃ b ⊲ Q) = (⟨b⟩S ∧ pre(P) ∨ [¬b]S ∧ peri(Q))
    by (rel-auto)

lemma periR-cond-srea [rdes]:
    assumes P is SRD Q is SRD
    shows peri(P ⊃ b ⊲ Q) = (⟨b⟩S ∧ peri(P) ∨ [¬b]S ∧ peri(Q))
    proof –
      have peri(P ⊃ b ⊲ Q) = peri(R1(P) ⊃ b ⊲ 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(P ⊃ b ⊲ Q) = (⟨b⟩S ∧ post(P) ∨ [¬b]S ∧ post(Q))
    proof –
      have post(P ⊃ b ⊲ Q) = post(R1(P) ⊃ b ⊲ 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 ⊃ b ⊲ Q is NSRD
    proof (rule NSRD-RC-intro)
      show P ⊃ b ⊲ Q is SRD
        by (simp add: closure assms unrest)
    qed

11.4 Assumptions

definition AssumeR :: 's cond ⇒ ('s::trace, 'a) hrel-rsp (⟨[−]⟩) where
    [upred-defs]: AssumeR b = H ⊃ b ⊲ Miracle
lemma AssumeR-rdes-def [rdes-def]:
\([b]^+ R = R_s(\text{true} \rightarrow false \circ [b]^+\text{true})\]

unfolding AssumeR-def by (rdes-eq)

lemma AssumeR-NSRD [closure]: \([b]^+ R \text{ is } \text{NSRD}\]
by (simp add: AssumeR-def closure)

lemma AssumeR-false: \([false]^+ R = \text{Miracle}\]
by (rel-auto)

lemma AssumeR-true: \([true]^+ R = \text{II} R\]
by (rel-auto)

lemma AssumeR-comp: \([b]^+ R \circ [c]^+ R = [b \land c]^+ R\]
by (rdes-simp)

lemma AssumeR-choice: \([b]^+ R \lor [c]^+ R = [b \lor c]^+ R\]
by (rdes-eq)

lemma AssumeR-refine-skip: \(\text{II} R \sqsubseteq [b]^+ R\)
by (rdes-refine)

lemma AssumeR-test [closure]: \(\text{test}_R [b]^+ R\)
by (simp add: AssumeR-refine-skip nsrd-thy.utest-intro)

lemma Star-AssumeR: \([b]^+ R^\star = \text{II} R\]
by (simp add: AssumeR-NSRD AssumeR-test nsrd-thy.Star-test)

lemma AssumeR-choice-skip: \(\text{II} R \lor [b]^+ R = \text{II} R\)
by (rdes-eq)

lemma cond-srea-AssumeR-form:
assumes P is NSRD Q is NSRD
shows \(P \circ b \triangleright R Q = ([b]^+ R \circ P \cap [\neg b]^+ R \circ Q)\)
by (rdes-eq cls: assms)

lemma cond-srea-insert-assume:
assumes P is NSRD Q is NSRD
shows \(P \circ b \triangleright R_Q = ([b]^+ R \circ P \cap [\neg b]^+ R \circ Q)\)
by (simp add: AssumeR-NSRD AssumeR-comp NSRD-seqr-closure RA1 assms cond-srea-AssumeR-form)

lemma AssumeR-cond-left:
assumes P is NSRD Q is NSRD
shows \([b]^+ R \circ (P \circ b \triangleright R_Q) = ([b]^+ R \circ P)\)
by (rdes-eq cls: assms)

lemma AssumeR-cond-right:
assumes P is NSRD Q is NSRD
shows \([\neg b]^+ R \circ (P \circ b \triangleright R_Q) = ([\neg b]^+ R \circ Q)\)
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 gcmd-false[simp]: (false $\rightarrow_R A) = \text{Miracle}
unfolding gcmd-def by (pred-auto)

lemma gcmd-true[simp]: (true $\rightarrow_R A) = A
unfolding gcmd-def by (pred-auto)

lemma gcmd-SRD:
assumes A is SRD
shows (g $\rightarrow_R A) is SRD
by (simp add: gcmd-def SRD-cond-srea assms srdes-theory-continuous.weak.top-closed)

lemma gcmd-NSRD [closure]:
assumes A is NSRD
shows (g $\rightarrow_R A) is NSRD
by (simp add: gcmd-def NSRD-cond-srea assms NSRD-Miracle)

lemma gcmd-Productive [closure]:
assumes A is NSRD A is Productive
shows (g $\rightarrow_R A) is Productive
by (simp add: gcmd-def closure assms)

lemma gcmd-seq-distr:
assumes B is NSRD
shows (g $\rightarrow_R A ;; B) = (g $\rightarrow_R A ;; B)
by (simp add: Miracle-left-zero NSRD-is-SRD assms cond-st-distr gcmd-def)

lemma gcmd-nondet-distr:
assumes A is NSRD B is NSRD
shows (g $\rightarrow_R (A \sqcap B)) = (g $\rightarrow_R A) \sqcap (g $\rightarrow_R B)
by (rdes-eq cls: assms)

lemma AssumeR-as-gcmd:
[b]$^>_R = b \rightarrow_R II_R
by (rdes-eq)

12 Generalised Alternation

definition AlternateR
:: ('a set $\Rightarrow$ ('a $\Rightarrow$ 's upred) $\Rightarrow$ ('a $\Rightarrow$ ('s, 't::trace, 'a) hrel-rsp) $\Rightarrow$ ('s, 't, 'a) hrel-rsp $\Rightarrow$ ('s, 't, 'a) hrel-rsp where
[upred-defs, rdes-def]
AlternateR I g A B = ($\prod i \in I \cdot ((g i) $\rightarrow_R (A i))) \sqcap ((\forall i \in I \cdot g i) $\rightarrow_R B)

definition AlternateR-list
:: ('s upred $\times$ ('s, 't::trace, 'a) hrel-rsp) list $\Rightarrow$ ('s, 't, 'a) hrel-rsp $\Rightarrow$ ('s, 't, 'a) hrel-rsp where
[upred-defs, rdes-simp]
AlternateR-list xs P = AlternateR \{0..<length xs\} (\lambda i. map fst xs ! i) (\lambda i. map snd xs ! i) P

syntax
-alttindR-els :: pttrn $\Rightarrow$ logic $\Rightarrow$ logic $\Rightarrow$ logic $\Rightarrow$ logic $\Rightarrow$ logic (if $R$ -$\in$- $\cdot$ $\cdot$ $\rightarrow$ - else - fi)
-alttindR :: pttrn $\Rightarrow$ logic $\Rightarrow$ logic $\Rightarrow$ logic $\Rightarrow$ logic (if $R$ -$\in$- $\cdot$ $\cdot$ $\rightarrow$ - fi)

-altgcommR-els :: gcomms $\Rightarrow$ logic $\Rightarrow$ logic (if $R$ - else - fi)
alttgcommR :: gcomms $\Rightarrow$ logic (if $R$ - fi)

94
translations
\[
\begin{align*}
if_R \ i \in I \cdot g & \rightarrow A \ else \ B \ \fi \rightarrow \text{CONST AlternateR } I \ (\lambda i. \ g) \ (\lambda i. \ A) \ B \\
if_R \ i \in I \cdot g & \rightarrow A \ else \ B \ \fi \rightarrow \text{CONST AlternateR } I \ (\lambda i. \ g) \ (\lambda i. \ A) \ (\text{CONST Chaos}) \\
if_R \ i \in I \cdot (g i) & \rightarrow A \ else \ B \ \fi \rightarrow \text{CONST AlternateR } I \ g (\lambda i. \ A) \ B \\
\text{-algcommR } cs & \rightarrow \text{CONST AlternateR-list } cs \ (\text{CONST Chaos}) \\
\text{-algcommR-els } cs P & \rightarrow \text{CONST AlternateR-list } cs \ P \\
\text{-algcommR-els } (-gcomm-show \ cs) & \rightarrow \text{CONST AlternateR-list } cs \ (\text{CONST Chaos}) \\
\text{ lemma AlternateR-NSRD-closed [closure]:} \\
\text{ assumes } \bigwedge \ i. \ i \in I \implies A \ is \ NSRD \ B \ is \ NSRD \\
\text{ shows } (\text{if } R \ i \in I \cdot g \ i \rightarrow A \ i \ else \ B \ i) \ is \ NSRD \\
\text{ proof } (\text{cases } I = \{\}) \\
\text{ case True} \\
\text{ then show } \ ?thesis \ by \ (\text{simp add: AlternateR-def assms}) \\
\text{ next} \\
\text{ case False} \\
\text{ then show } \ ?thesis \ by \ (\text{simp add: AlternateR-def closure assms}) \\
\text{ qed} \\
\text{ lemma AlternateR-empty [simp]:} \\
\text{ (if } R \ i \in I \cdot g \ i \rightarrow A \ i \ else \ B \ i) \ = \ B \\
\text{ by } (\text{rdes-simp}) \\
\text{ lemma AlternateR-Productive [closure]:} \\
\text{ assumes } \bigwedge \ i. \ i \in I \implies A \ i \ is \ Productive \ B \ is \ Productive \\
\text{ shows } (\text{if } R \ i \in I \cdot g \ i \rightarrow A \ i \ else \ B \ i) \ is \ Productive \\
\text{ proof } (\text{cases } I = \{\}) \\
\text{ case True} \\
\text{ then show } \ ?thesis \ by \ (\text{simp add: assms(4)}) \\
\text{ next} \\
\text{ case False} \\
\text{ then show } \ ?thesis \ by \ (\text{simp add: AlternateR-def closure assms}) \\
\text{ qed} \\
\text{ lemma AlternateR-singleton:} \\
\text{ assumes } A \ k \ is \ NSRD \ B \ is \ NSRD \\
\text{ shows } (\text{if } R \ i \in \{k\} \cdot g \ i \rightarrow A \ i \ else \ B \ i) \ = \ (A(k) \triangleleft g(k) \triangleright B) \\
\text{ by } (\text{simp add: AlternateR-def, rdes-eq cls: assms}) \\
\text{ Convert an alternation over disjoint guards into a cascading if-then-else} \\
\text{ lemma AlternateR-insert-cascade:} \\
\text{ assumes } \bigwedge \ i. \ i \in I \implies A \ i \ is \ NSRD \\
\text{ A } k \ is \ NSRD \ B \ is \ NSRD \\
\text{ (g(k) } \land \ (\bigvee \ i \in I \cdot g(i)) \) = false \\
\text{ shows } (\text{if } R \ i \in \text{insert } k I \cdot g \ i \rightarrow A \ i \ else \ B \ i) \ = \ (A(k) \triangleleft g(k) \triangleright (if_R \ i \in I \cdot g(i) \rightarrow A(i) \ else \ B \ i)) \\
\text{ proof } (\text{cases } I = \{\}) \\
\text{ case True} \\
\text{ then show } \ ?thesis \ by \ (\text{simp add: AlternateR-singleton assms})
next  
  case False  
  have 1: (\( \bigcap_{i \in I} g_i \to_R A_i \) = (\( \bigcap_{i \in I} g_i \to_R R_s(\text{pre}_R(A_i) \vdash \text{peri}_R(A_i) \circ \text{post}_R(A_i)) \))  
    by (simp add: NSRD-is-SRD SRD-reactive-tri-design assms(1) cong: UINF-cong)  
  from assms(4) show ?thesis  
    by (simp add: AlternateR-def 1 False cong: UINF-cong)  
qed  

12.1 Choose  

definition choose-srd :: (′s,′t::trace,′α) hrel-rsp (choose_R) where  
  [upred-defs, rdes-def]: choose_R = R_s(true \vdash false \circ true)  

lemma preR-choose [rdes]: pre_R(choose_R) = true  
  by (rel-auto)  

lemma periR-choose [rdes]: peri_R(choose_R) = false  
  by (rel-auto)  

lemma postR-choose [rdes]: post_R(choose_R) = true  
  by (rel-auto)  

lemma choose-srd-SRD [closure]: choose_R is SRD  
  by (simp add: choose-srd-def closure unrest)  

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

12.2 State Abstraction  

definition state-srea ::  
  ′s itself \Rightarrow (′s,′t::trace,′α,′β) rel-rsp \Rightarrow (unit,′t,′α,′β) rel-rsp where  
  [upred-defs]: state-srea t P = (\( \exists \{\$\hat{s},\$\hat{\hat{s}}\} \cdot P\))  

syntax  
  -state-srea :: type \Rightarrow logic \Rightarrow logic (state - - [0,200] 200)  

translations  
  state ′a :: P == CONST state-srea TYPE(′a) P  

lemma R1-state-srea: R1(state ′a :: P) = (state ′a :: R1(P))  
  by (rel-auto)  

lemma R2c-state-srea: R2c(state ′a :: P) = (state ′a :: R2c(P))  
  by (rel-auto)  

lemma R3h-state-srea: R3h(state ′a :: P) = (state ′a :: R3h(P))  
  by (rel-auto)  

lemma RD1-state-srea: RD1(state ′a :: P) = (state ′a :: RD1(P))  
  by (rel-auto)  

lemma RD2-state-srea: RD2(state ′a :: P) = (state ′a :: RD2(P))  
  by (rel-auto)
lemma RD3-state-srea: RD3(state 'a · P) = (state 'a · RD3(P))
  by (rel-auto, blast+)

lemma SRD-state-srea [closure]: P is SRD ⇒ state 'a · P is SRD
  by (simp add: Healthy-def R1-state-srea R2c-state-srea R3h-state-srea RD1-state-srea RD2-state-srea RHS-def SRD-def)

lemma NSRD-state-srea [closure]: P is NSRD ⇒ state 'a · P is NSRD
  by (metis Healthy-def NSRD-is-RD3 NSRD-is-SRD RD3-state-srea SRD-RD3-implies-NSRD SRD-state-srea)

lemma preR-state-srea [rdes]: preR(state 'a · P) = (∀ {st, $st*} · preR(P)) S
  by (simp add: state-srea-def, rel-auto)

lemma periR-state-srea [rdes]: periR(state 'a · P) = state 'a · periR(P)
  by (rel-auto)

lemma postR-state-srea [rdes]: postR(state 'a · P) = state 'a · postR(P)
  by (rel-auto)

12.3 While Loop

definition WhileR :: 's upred ⇒ ('s, 't::size-trace, 'a) hrel-rsp ⇒ ('s, 't, 'a) hrel-rsp (whileR - do - od)
where
WhileR b P = (µR X · (P ;; X) ⊲ b ⊲R II_R)

lemma Sup-power-false:
  fixes F :: 'α upred ⇒ 'α upred
  shows (⨈ i. (F `· i) false) = (⨈ i. (F `· (i+1)) false)
  proof –
    have (⨈ i. (F `· i) false) = (F `· 0) false ∩ (⨈ i. (F `· (i+1)) false)
      by (subst Sup-power-expand, simp)
    also have ... = (∨ i. (F `· (i+1)) false)
      by (simp)
    finally show ?thesis .
  qed

theorem WhileR-iter-expand:
  assumes P is NSRD P is Productive
  shows whileR b d P od = (⨈ i · (P ⊲ b ⊲R II_R) ^ i ;; (P ;; Miracle ⊲ b ⊲R II_R)) (is ?lhs = ?rhs)
  proof –
    have 1: Continuous (λX. P ;; SRD X)
      using SRD-Continuous
      by (clarsimp simp add: Continuous-def seq-SUP-distl[THEN sym], drule-tac x=A in spec, simp)
    have 2: Continuous (λX. P ;; SRD X ⊲ b ⊲R II_R)
      by (simp add: 1 closure assms)
    have ?lhs = (µR X · P ;; X ⊲ b ⊲R II_R)
      by (simp add: WhileR-def)
    also have ... = (µX · P ;; SRD(X) ⊲ b ⊲R II_R)
      by (auto simp add: srd-mu-equivalence closure assms)
    also have ... = (µx · P ;; SRD(X) ⊲ b ⊲R II_R)
      by (auto simp add: guarded-fp-uniqueness Guarded-if-Productive[OF assms] funcsetI closure assms)
    also have ... = (∨ i. (λX. P ;; SRD X ⊲ b ⊲R II_R) `· i) false)
      by (simp add: sup-continuous-lfp 2 sup-continuous-Continuous false-upred-def)
    also have ... = (∨ i. (λX. P ;; SRD X ⊲ b ⊲R II_R) `· (i+1)) false)
      by (simp add: Sup-power-false)
    also have ... = (∨ i. (P ⊲ b ⊲R II_R) ^ i ;; (P ;; Miracle ⊲ b ⊲R II_R))
  qed
proof (rule SUP-cong, simp)
  fix i
  show \((\lambda X. P ;; SRD X \triangleleft b \triangleright R II_R) \sim (i + 1)\) false = \((P \triangleleft b \triangleright R II_R) \sim i ;; (P ;; Miracle \triangleleft b \triangleright R II_R)\)
proof (induct i)
  case 0
  thm if-eq-cancel
  then show ??case
    by (simp, metis srdes-cond-def srdes-theory-continuous.healthy-top)
next
case (Suc i)
  show ??case
proof (induct i)
  case 0
  then show ??thesis
  by (simp add: NSRD-srdes-skip SRD-cond-srea SRD-left-unit SRD-seqr-closure assms(1) power.power-eq-if-seq-left-unit srdes-theory-continuous.top-closed)
next
case (Suc i)
  have \(1: II_R ;; ((P \triangleleft b \triangleright R II_R) ;; (P \triangleleft b \triangleright R II_R) \sim i) = ((P \triangleleft b \triangleright R II_R) ;; (P \triangleleft b \triangleright R II_R) \sim i)\)
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

also have \(...) = (\prod\(i \cdot (P \triangleleft b \triangleright R II_R) \sim i ;; (P ;; Miracle \triangleleft b \triangleright R II_R)\))
  then show ??thesis
  by (simp add: UINF-as-Sup-collect')
finally show ??thesis .
qed

theorem WhileR-star-expand:
  assumes P is NSRD P is Productive
  shows whileR b do P od = (P \triangleleft b \triangleright R II_R) * R ;; (P ;; Miracle \triangleleft b \triangleright R II_R) (is \?lhs = \?rhs)
proof
  have \?lhs = (\prod\(i \cdot (P \triangleleft b \triangleright R II_R) \sim i) ;; (P ;; Miracle \triangleleft b \triangleright R II_R)\)
by (simp add: WhileR-iter-expand seq-UINF-distr' assms)
also have \( \vdash (P \land b \Rightarrow R) \); (P \vdash Miracle \land b \Rightarrow R) \)

by (simp add: ustar-def)
also have \( \vdash \vdash (P \land b \Rightarrow R) \); (P \vdash Miracle \land b \Rightarrow R) \)

by (simp add: seq-assoc SRD-left-unit closure assms)
also have \( \vdash (P \land b \Rightarrow R) \); (P \vdash Miracle \land b \Rightarrow R) \)

by (simp add: nsrd-thy.Star-def)
finally show ?thesis .

qed

lemma WhileR-NSRD-closed [closure]:
assumes P is NSRD P is Productive
shows whileR b do P od is NSRD
by (simp add: WhileR-star-expand assms closure)

theorem WhileR-iter-form-lemma:
assumes P is NSRD
shows (P \land b \Rightarrow R) \star \vdash (P \land b \Rightarrow R) \star (Miracle \land b \Rightarrow R) \)

by (simp add: AssumeR-NSRD right-unit NSRD-srd-skip assms (1) cond-srea-AssumeR-form)
also have \( \vdash (|b| R \vdash P) \vdash (\vdash [|b| R \vdash P) \vdash (- |b| R) \rightarrow (- |b| R) \rightarrow (P \vdash Miracle \land b \Rightarrow R) \)

by (simp add: AssumeR-NSRD NSRD-seqr-closure nsrd-thy.Star-denest assms (1))
also have \( \vdash (|b| R \vdash P) \vdash (\vdash [|b| R \vdash P) \vdash (- |b| R) \rightarrow (- |b| R) \rightarrow (P \vdash Miracle \land b \Rightarrow R) \)

by (simp add: AssumeR-NSRD NSRD-seqr-closure nsrd-thy.Star-invol assms (1))
also have \( \vdash (|b| R \vdash P) \vdash (\vdash [|b| R \vdash P) \vdash (- |b| R) \rightarrow (- |b| R) \rightarrow (P \vdash Miracle \land b \Rightarrow R) \)

by (simp add: AssumeR-NSRD NSRD-Miracle NSRD-right-unit NSRD-seqr-closure NSRD-srd-skip assms (1) cond-srea-AssumeR-form)
also have \( \vdash (|b| R \vdash P) \vdash (\vdash [|b| R \vdash P) \vdash (- |b| R) \rightarrow (- |b| R) \rightarrow (P \vdash Miracle \land b \Rightarrow R) \)

by (simp add: upred-semiring.distrib-left)
also have \( \vdash (|b| R \vdash P) \vdash (\vdash [|b| R \vdash P) \vdash (- |b| R) \rightarrow (- |b| R) \rightarrow (P \vdash Miracle \land b \Rightarrow R) \)

by (simp add: RAI)
also have \( \vdash (|b| R \vdash P) \vdash (\vdash [|b| R \vdash P) \vdash (- |b| R) \rightarrow (- |b| R) \rightarrow (P \vdash Miracle) \)

by (simp add: AssumeR-comp AssumeR-false)
finally have \( \vdash (|b| R \vdash P) \vdash (\vdash [|b| R \vdash P) \vdash (- |b| R) \rightarrow (- |b| R) \rightarrow (P \vdash Miracle) \)

by (simp add: semilattice-sup-class.le-supI1)
thus ?thesis
by (simp add: semilattice-sup-class.le-iff-sup)

99
theorem WhileR-iter-form:
assumes P is NSRD P is Productive
shows while_R b do P od = ([b]₁ R ; P)⁺ R ; [¬ b]₃ 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 = II₃
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]₃ R ; P)⁺ R ; [¬ b]₃ R)
by (simp add: AssumeR-NSRD AssumeR-comp NSRD-seqr-closure Productive-seq-2 RA1 WhileR-iter-form assms)

theorem WhileR-rdes-def [rdes-def]:
assumes P is RC Q is RR R is RR $st' ⊢ Q R is R₄
shows while_R b do Rₛ(P ⊢ Q ∘ R) od =
Rₛ (([b]₃ R ; R)⁺ R ; Q ⊢ (b)⁺ R ; [b]₃ R ; R)⁺ R ; [¬ b]₃ R)
(is ?lhs = ?rhs)
proof —
have ?lhs = ([b]₃ R ; Rₛ(P ⊢ Q ∘ R))⁺ R ; [¬ b]₃ R
by (simp add: WhileR-iter-form Productive-rdes-RR-intro assms closure)
also have ... = ?rhs
by (simp add: rdes-def assms closure unrest rpred wp del: rea-star-wp)
finally show ?thesis .
qed

Refinement introduction law for reactive while loops

theorem WhileR-refine-intro:
assumes — Closure conditions
Q₁ is RC Q₂ is RR Q₃ is RR $st' ⊢ Q₂ Q₃ is R₄
— Refinement conditions
([b]₃ R ; Q₃)⁺ R wpₚ ([b]₃ R ; Q₁) ⊆ P₁
P₂ ⊆ [b]₃ R ; Q₂
P₂ ⊆ [b]₃ R ; Q₃ ; P₂
P₃ ⊆ [¬ b]₃ R
P₃ ⊆ [b]₃ R ; Q₃ ; P₃
shows Rₛ(P₁ ⊎ P₂ ⊎ P₃) ⊆ while_R b do Rₛ(Q₁ ⊎ Q₂ ⊎ Q₃) od
proof (simp add: rdes-def assms, rule srdes-tri-refine-intro)
show ([b]₃ R ; Q₃)⁺ R wpₚ ([b]₃ R ; Q₁) ⊆ P₁
by (simp add: assms)
show P₂ ⊆ (P₁ ∩ ([b]₃ R ; Q₃))⁺ R ; [b]₃ R ; Q₂
proof —
have \( P_2 \subseteq ([b]^r \land \neg [b]^r) \land Q_3 \land [b]^r \land Q_3 \quad \text{by} \quad (\text{simp add: asms} \text{ reuse} \text{-assume} \text{-RR} \text{-rel-thy} \text{-star-inductl} \text{-seq} \text{-RR} \text{-closed} \text{ seq} \text{-assoc}) \)

thus ?thesis
by (simp add: utp-pred-laws.le-infl)
qed

show \( P_3 \subseteq ([b]^r \land \neg [b]^r) \land Q_3 \land [b]^r \land Q_3 \)
proof
have \( P_3 \subseteq ([b]^r \land \neg [b]^r) \land Q_3 \land [b]^r \land Q_3 \quad \text{by} \quad (\text{simp add: asms} \text{ reuse} \text{-assume} \text{-RR} \text{-rel-thy} \text{-star-inductl} \text{-seq} \text{-assoc}) \)
thus ?thesis
by (simp add: utp-pred-laws.le-infl)
qed

12.4 Iteration Construction

definition \text{IterateR}:: a set \Rightarrow (a \Rightarrow 's upred) \Rightarrow (a \Rightarrow ('s, 't::size-trace, 'a) hrel-rsp) \Rightarrow (a, 't, 'a) hrel-rsp
where \text{IterateR} A g P = while_R (\bigvee i \in A \cdot g(i)) \cdot do (if_R i:A \cdot g(i) \Rightarrow P(i, fi)) \cdot od

syntax
- \text{-iter-srd} :: ptrtn \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow (do_R -\cdot \Rightarrow -\cdot \Rightarrow -\cdot \Rightarrow -\cdot \Rightarrow -\cdot fi)

translations
- \text{-iter-srd} x A g P ==> \text{CONST IterateR} A (\lambda x. g) (\lambda x. P)
- \text{-iter-srd} x A g P <= Const \text{IterateR} A (\lambda x. g) (\lambda x'. P)

lemma \text{IterateR-NSRD-closed} [closure]:
assumes \quad \Lambda \ i. i \in I \Rightarrow P(i) \text{ is NSRD}
\quad \Lambda \ i. i \in I \Rightarrow P(i) \text{ is Productive}
shows \quad do_R i\in I \cdot g(i) \Rightarrow P(i, fi) \text{ is NSRD}
by (simp add: \text{IterateR-def} closure asms)

lemma \text{IterateR-empty}:
\quad do_R i\in \{\} \cdot g(i) \Rightarrow P(i, fi) = II_R
by (simp add: \text{IterateR-def} srd-mu-equiv closure rpred gfp-const WhileR-false)

lemma \text{IterateR-singleton}:
assumes \quad P k \text{ is NSRD} \text{ P k is Productive}
shows \quad do_R i\in \{k\} \cdot g(i) \Rightarrow P(i, fi) = \text{while}_R g(k) \cdot do P(k) \cdot od \quad (\text{is} \ ?lhs = \ ?rhs)
proof –
have \ ?lhs = \text{while}_R g k \Rightarrow P(k, fi) \Rightarrow P(k) \cdot od
by (simp add: \text{IterateR-def} AlternateR-singleton asms closure)
also have \ ... = \text{while}_R g k \Rightarrow P(k) \cdot od
by (simp add: \text{WhileR-insert-assume} closure asms)
also have \ ... = \text{while}_R g k \Rightarrow P(k) \cdot od
by (simp add: AssumeR-cond-left NSRD-Chaos WhileR-insert-assume asms)
finally show \ ?thesis.
qed

12.5 Substitution Laws

lemma \text{srd-subst-Chaos} [asubst]:
\quad \sigma \upharpoonright_S \text{ Chaos} = \text{ Chaos}
by (rdes-simp)
lemma srd-subst-Miracle [ usurst ]:
\( \sigma \uparrow_S \text{Miracle} = \text{Miracle} \)
by (rdes-simp)

lemma srd-subst-skip [ usurst ]:
\( \sigma \uparrow_S II_R = \langle \sigma \rangle_R \)
by (rdes-eq)

lemma srd-subst-assigns [ usurst ]:
\( \sigma \uparrow_S \langle \sigma \rangle_R = \langle \sigma \circ \sigma \rangle_R \)
by (rdes-eq)

12.6 Algebraic Laws

theorem assigns-srd-id: \( \langle \text{id} \rangle_R = II_R \)
by (rdes-eq)

theorem assigns-srd-comp: \( \langle \sigma \rangle_R \triangleright \langle \varrho \rangle_R = \langle \varrho \circ \sigma \rangle_R \)
by (rdes-eq)

theorem assigns-srd-Miracle: \( \langle \sigma \rangle_R \triangleright \text{Miracle} = \text{Miracle} \)
by (rdes-eq)

theorem assigns-srd-Chaos: \( \langle \sigma \rangle_R \triangleright \text{Chaos} = \text{Chaos} \)
by (rdes-eq)

theorem assigns-srd-cond : \( \langle \sigma \rangle_R \triangleright b \triangleright \langle \varrho \rangle_R = \langle \sigma \circ b \triangleright \varrho \rangle_R \)
by (rdes-eq)

theorem assigns-srd-left-seq:
assumes P is NSRD
shows \( \langle \sigma \rangle_R \triangleright P = \sigma \uparrow_S P \)
by (rdes-simp cls: assms)

lemma AlternateR-seq-distr:
assumes \( \bigwedge_i. A_i \text{ is NSRD } B \text{ is NSRD } C \text{ is NSRD} \)
shows \( \langle \text{if } R_i \in I \cdot g \rightarrow A_i \text{ else } B \text{ fi} \rangle_R = \langle \text{if } R_i \in I \cdot g \rightarrow A_i \triangleright C \text{ else } B \triangleright C \text{ fi} \rangle_R \)
proof (cases I = \{\})
  case True
  then show \(?thesis\) by (simp)
next
  case False
  then show \(?thesis\)
  by (simp add: AlternateR-def upred-semiring.distrib-right seq-UINF-distr gcmd-seq-distr assms(3))
qed

lemma AlternateR-is-cond-srea:
assumes A is NSRD B is NSRD
shows \( \langle \text{if } R_i \in \{a\} \cdot g \rightarrow A \text{ else } B \text{ fi} \rangle_R = \langle A \circ g \triangleright_R B \rangle \)
by (rdes-eq cls: assms)

lemma AlternateR-Chaos:
if \( R_i \in A \cdot g(i) \rightarrow \text{Chaos} \text{ fi} = \text{Chaos} \)
by (cases A = \{\}, simp, rdes-eq)
12.7 Lifting designs to reactive designs

**Definition** des-rea-lift :: 's hrel-des ⇒ ('s,t::trace,'a) hrel-rsp (R_D) where
[upred-defs]: R_D(P) = R_a([pre_D(P)]_S) ⊢ (false ∘ ($tr' = u $tr ∧ [post_D(P)]_S))

**Definition** des-rea-drop :: ('s,t::trace,'a) hrel-rsp ⇒ 's hrel-des (R_D) where

**Lemma** ndesign-rea-lift-inverse: D_R(R_D(p ⊨ n Q)) = p ⊨ n Q
apply (simp add: des-rea-lift-def des-rea-drop-def rea-pre-RHS-design rea-post-RHS-design)
apply (simp add: R1-def R2c-def R2s-def unsubst unrest)
done

**Lemma** ndesign-rea-lift-injective: assumes P is N Q R D P = R_D Q (is ?RP(P) = ?RQ(Q))
shows P = Q
proof –
  have ?RP([pre_D(P)]_S) ⊢ n post_D(P)) = ?RQ([pre_D(Q)]_S) ⊢ n post_D(Q)
  by (simp add: ndesign-form assms)
  hence [pre_D(P)]_S ⊢ n post_D(P) = [pre_D(Q)]_S ⊢ 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)) = (false ∘ [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)
done

qed

lemma R-D-mono:
assumes P is H Q is H P ⊑ Q
shows R_D(P) ⊑ R_D(Q)
apply (simp add: des-rea-lift-def)
apply (rule srdes-tri-refine-intro')
  apply (auto intro: H1-H2-refines assms aext-mono)
apply (rel-auto)
apply (metis (no-types, hide-lams) aext-mono assms)

Homomorphism laws

lemma R-D-Miracle:
R_D(⊤D) = Miracle
by (simp add: Miracle-def, rel-auto)

lemma R-D-Chaos:
R_D(⊥D) = Chaos
proof –
  have R_D(⊥D) = R_D(false ⊢ true)
    by (rel-auto)
  also have ... = R_a (false ⊢ false o ($tr' = u $tr))
    by (simp add: Chaos-def des-rea-lift-def alpha)
  also have ... = Chaos
    by (rel-auto)
  finally show ?thesis.
qed

lemma R-D-inf:
R_D(P \sqcap Q) = R_D(P) \sqcap R_D(Q)
by (rule antisym, rel-auto+)

lemma R-D-cond:
R_D(P ⊆ [b]_{D<} Q) = R_D(P) ⊆ R_D(Q)
by (rule antisym, rel-auto+)

lemma R-D-seq-ndesign:
R_D(p_1 \vdash_n Q_1) ; R_D(p_2 \vdash_n Q_2) = R_D((p_1 \vdash_n Q_1) ; (p_2 \vdash_n Q_2))
apply (rule antisym)
  apply (rule SRD-refine-intro)
    apply (simp-all add: closure rdes ndesign-composition-wp)
using dual-order_trans apply (rel-blast)
using dual-order_trans apply (rel-blast)
apply (rel-auto)
apply (rule SRD-refine-intro)
  apply (simp-all add: closure rdes ndesign-composition-wp)
  apply (rel-auto)
  apply (rel-auto)
apply (rel-auto)
done
lemma \(R-D\)-seq:
assumes \(P \text{ is } N, Q \text{ is } N\)
shows \(R_D(P) :: R_D(Q) = R_D(P :: Q)\)
by (metis \(R-D\)-seq-ndesign assms ndesign-form)

Thes laws are applicable only when there is no further alphabet extension

lemma \(R-D\)-skip:
\(R_D(H_D) = (H_R :: ('s, t::trace, unit) hrel-rsp)\)
apply (rel-auto) using minus-zero-eq by blast+

lemma \(R-D\)-assigns:
\(R_D(\langle \sigma \rangle_D) = (\langle \sigma \rangle_R :: ('s, t::trace, unit) hrel-rsp)\)
by (simp add: assigns-d-def des-rea-lift-def alpha assigns-srd-RHS-tri-des, rel-auto)

end

13 Instantaneous Reactive Designs

declaration utp-rdes-instant
import utp-rdes-prog
begin

definition ISRD1 :: ('s, 't::trace, 'a) hrel-rsp \sto ('s, 't, 'a) hrel-rsp where
[upred-defs]: ISRD1(P) = P \|\ R_{\sigma}(true \vdash false \circ (\$tr' = u \cdot \$tr))

definition ISRD :: ('s, 't::trace, 'a) hrel-rsp \sto ('s, 't, 'a) hrel-rsp where
[upred-defs]: ISRD = ISRD1 \circ NSRD

lemma ISRD1-idem: ISRD1(ISRD1(P)) = ISRD1(P)
by (rel-auto)

lemma ISRD1-monotonic: \(P \sqsubseteq Q \implies ISRD1(P) \sqsubseteq ISRD1(Q)\)
by (rel-auto)

lemma ISRD1-RHS-design-form:
assumes \(\$ok' \notin P \cdot \$ok' \notin Q \cdot \$ok' \notin R\)
shows ISRD1(R_a(P \vdash Q \circ R)) = R_a(P \vdash false \circ (R \land \$tr' = u \cdot \$tr))
using assms by (simp add: ISRD1-def choose-srd-def RHS-tri-design-par unrest, rel-auto)

lemma ISRD1-form:
ISRD1(SRD(P)) = R_a(pre_R(P) \vdash false \circ (post_R(P) \land \$tr' = u \cdot \$tr))
by (simp add: ISRD1-RHS-design-form SRD-as-reactive-tri-design unrest)

lemma ISRD1-rdes-def [rdes-def]:
\[ P \text{ is RR; } R \text{ is RR } \implies ISRD1(R_a(P \vdash Q \circ R)) = R_a(P \vdash false \circ (R \land \$tr' = u \cdot \$tr))\]
by (simp add: ISRD1-def rdes-def closure rpred)

lemma ISRD-intro:
assumes P is NSRD peri_R(P) = (\neg \pre_R(P)) (\$tr' = u \cdot \$tr) \sqsubseteq post_R(P)
shows \(P \text{ is ISRD}\)
proof
have \(R_a(P \vdash \neg \pre_R(P) \circ post_R(P)) \text{ is ISRD1}\)
apply (simp add: Healthy-def rdes-def closure assms(1-2))
using assms(3) least-zero apply (rel-blast)

105
done

hence $P$ is ISRD1
  by (simp add: SRD-reactive-tri-design closure assms(1))
thus $?thesis$
  by (simp add: ISRD-def Healthy-comp assms(1))

qed

lemma ISRD1-rdes-intro:
  assumes $P$ is RR $Q$ is RR ($\triangleright_\ell = \ell_\triangleright \subseteq Q$
  shows $R_s(\neg \mathbf{false} \triangleright_\ell Q)$ is ISRD1
unfolding Healthy-def
by (simp add: ISRD1-rdes-def assms closure unrest utp-pred-laws.info.absorb1)

lemma ISRD-rdes-intro [closures]:
  assumes $P$ is RC $Q$ is RR ($\triangleright_\ell = \ell_\triangleright \subseteq Q$
  shows $R_s(\neg \mathbf{false} \triangleright_\ell Q)$ is ISRD
unfolding Healthy-def
by (simp add: ISRD-def closure Healthy-if ISRD1-rdes-def assms unrest utp-pred-laws.info.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($P$) = $R_s((\neg_r (\neg_r \mathbf{pre}_R P) \cdot R1 \mathbf{true} \land R1 \mathbf{true}) \triangleright_\ell \mathbf{false} \circ (\mathbf{post}_R P \land \ell_\triangleright_\ell = \ell_\triangleright \subseteq P))$
    by (simp add: NSRD-form ISRD1-def ISRD-def RHS-tri-design-par rdes-def unrest closure)
moreover have $\ldots$ 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 [closures]:
  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 RD3-commute RD3-left-subsumes RD2)
also have $\ldots = ISRD1(RD3(P))$
    by (simp add: assms ISRD-implies-SRD Healthy-if)
also have $\ldots = ISRD1 (R_s ((\neg_r \mathbf{pre}_R P) \mathbf{wp}_R \mathbf{false}_h \triangleright_\ell (\exists \ell_\triangleright_\ell \cdot \mathbf{peri}_R P) \circ \mathbf{post}_R P)))$
    by (simp add: RD3-def subst SRD-right-unit-tri-lemma, simp-all add: assms ISRD-implies-SRD)
also have $\ldots = R_s ((\neg_r \mathbf{pre}_R P) \mathbf{wp}_R \mathbf{false}_h \triangleright_\ell \mathbf{false} \circ (\mathbf{post}_R P \land \ell_\triangleright_\ell = \ell_\triangleright \subseteq P))$
    by (simp add: RHS-tri-design-par ISRD1-def unrest choose-srd-def rpred closure ISRD-implies-SRD)
lemma ISRD-form:
assumes P is ISRD
shows R \( \cdot (\text{pre}_R(P) \vdash \text{false} \circ (\text{post}_R(P) \land \$tr^{-} = u \$tr)) = P \)
proof
have P = ISRD1(P)
  by (simp add: ISRD-implies-ISRD1 assms Healthy-if)
also have ... = ISRD1(R \( \cdot (\text{pre}_R(P) \vdash \text{peri}_R(P) \circ \text{post}_R(P)) \))
  by (simp add: SRD-reactive-tri-design ISRD-implies-SRD assms)
also have ... = R \( \cdot (\text{pre}_R(P) \vdash \text{false} \circ (\text{post}_R(P) \land \$tr^{-} = u \$tr)) \)
  by (simp add: ISRD1-rdes-def closure assms)
finally show \?thesis ..
qed

lemma ISRD-elim [RD-elim]:
\[ P \text{ is ISRD; } Q(\text{R} \cdot (\text{pre}_R(P) \vdash \text{false} \circ (\text{post}_R(P) \land \$tr^{-} = u \$tr))) \implies Q(P) \]
by (simp add: ISRD-form)

lemma skip-srd-ISRD [closure]: \( II_R \) is ISRD
by (rule ISRD-intro, simp-all add: rdes closure)

lemma assigns-srd-ISRD [closure]: \( \langle \sigma \rangle_R \) is ISRD
by (rule ISRD-intro, simp-all add: rdes closure, rel-auto)

lemma seq-ISRD-closed:
assumes P is ISRD Q is ISRD
shows P ;; Q is ISRD
apply (insert assms)
apply (erule ISRD-elim)+
apply (simp add: rdes-def closure assms unrest)
apply (rule ISRD-rdes-intro)
  apply (simp-all add: rdes-def closure assms unrest)
apply (rel-auto)
done

lemma ISRD-Miracle-right-zero:
assumes P is ISRD \( \text{pre}_R(P) = \text{true}_r \)
shows P ;; Miracle = Miracle
by (rdes-simp cls: assms)

A recursion whose body does not extend the trace results in divergence

lemma ISRD-recurse-Chaos:
assumes P is ISRD \( \text{post}_R(P ;; \text{true}_r = \text{true}_r) \)
shows (\( \mu_R X \cdot P ;; X \)) = Chaos
proof
have 1: (\( \mu_R X \cdot P ;; X \)) = (\( \mu X \cdot P ;; \text{SRD}(X) \))
by (auto simp add: srdes-theory-continuous.utp-lfp-def closure assms)

have \((\mu X \cdot P :: SRD(X)) \subseteq Chaos\)
proof (rule gfp-upperbound)
  have \(P :: Chaos \subseteq Chaos\)
  apply (rdes-refine-split cls: assms)
  using assms(2) apply (rel-auto, metis (no-types, lifting) dual-order.antisym order-refl)
  apply (rel-auto)+
done
thus \(P :: SRD Chaos \subseteq Chaos\)
by (simp add: Healthy-if srdes-theory-continuous.bottom-closed)
qed

thus \(?thesis\)
by (metis 1 dual-order.antisym srdes-theory-continuous.LFP-closed srdes-theory-continuous.bottom-lower)

qed

lemma recursive-assign-Chaos:
\((\mu_R X \cdot \langle\sigma\rangle_R :: X) = Chaos\)
by (rule ISRD-recurse-Chaos, simp-all add: closure rdes, rel-auto)

end

14 Meta-theory for Reactive Designs

theory utp-rea-designs

imports
  utp-rdes-healths
  utp-rdes-designs
  utp-rdes-triples
  utp-rdes-normal
  utp-rdes-contracts
  utp-rdes-tactics
  utp-rdes-parallel
  utp-rdes-prag
  utp-rdes-instant
  utp-rdes-guarded

begin end

References

[1] S. Foster, A. Cavalcanti, S. Canham, J. Woodcock, and F. Zeyda. Unifying theories of
reactive design contracts. Submitted to Theoretical Computer Science, Dec 2017. Preprint:
