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 (Unpublished)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.

eprints@whiterose.ac.uk
https://eprints.whiterose.ac.uk/
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 | 42 |
| 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 64
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 91
11.2 Assignment 91
11.3 Conditional 92
11.4 Assumptions 93
11.5 Guarded commands 94
11.6 Generalised Alternation 94
11.7 Choose 96
11.8 State Abstraction 96
11.9 While Loop 97
11.10 Iteration Construction 101
11.11 Substitution Laws 102
11.12 Algebraic Laws 102
11.13 Lifting designs to reactive designs 103

12 Instantaneous Reactive Designs 105

13 Meta-theory for Reactive Designs 108
1 Introduction

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

2 Reactive Designs Healthiness Conditions

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

2.1 Preliminaries

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

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

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

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

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

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

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

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

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

2.2 Identities

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

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

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

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

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

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

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

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

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

2.3 RD1: Divergence yields arbitrary traces

definition RD1 :: \(((\',\text{trace}',\alpha',\beta') \ rel-rp) \Rightarrow (\',\alpha',\beta') \ rel-rp\) \ where 
upred-defs: \( RD1(P) = (P \lor (\neg \$ok \land \$tr \leq_u \$tr')) \)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

proof –
have \((R1(\text{true}) ;; R1(RD1(P))) = R1(\text{true})\)
  by (rel-auto)
thus ?thesis
  by (simp add: Healthy-if assms(1) assms(2))
qed

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

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

lemma RD1-alt-def:
assumes \(P\) is \(R1\)
shows \(RD1(P) = (\exists \$st \cdot II_c :: II_c ;; \text{wait} \cdot P)\)

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

2.4 R3c and R3h: Reactive design versions of R3

definition R3c :: ('t::trace, ('a) hrel-rp ⇒ ('t, 'α) hrel-rp)
  where \[upred-defs\]: \(R3c(P) = (II_c :: II_c ;; \text{wait} \cdot P)\)

definition R3h :: ('s, ('t::trace, ('a) hrel-rsp ⇒ ('s, 't, 'α) hrel-rsp)
  where \(R3h(P) = ((\exists \$st \cdot II_c :: \text{wait} \cdot P)\)

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

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

lemma R3c-mono: \(P ⊆ Q \Rightarrow R3c(P) ⊆ R3c(Q)\)
  by (rel-auto)

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

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

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

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

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

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

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

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

lemma R3h-cond: R3h(P ⊆ b ⊇ Q) = (R3h(P) ⊆ b ⊇ R3h(Q))
    by (rel-auto)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma \( R3c\)-\( R3h\)-left-seq-closure:
assumes \( P \) is \( R3 \) \( Q \) is \( R3c \)
shows \( (P ; Q) \) is \( R3c \)
proof -
have \( (P ; Q) = ((P ; Q)[true/\$wait] < \$wait \triangleright (P ; Q)) \)
  by (metis cond-var-split cond-var-subst-right in-var-avar wait-vwb-lens)
also have \( \ldots = (((I \& \$wait \triangleright P) ; Q)[true/\$wait] < \$wait \triangleright (P ; Q)) \)
  by (metis Healthy-def \( R3c\)-def assms(1))
also have \( \ldots = (((I[true/\$wait] ; Q) < \$wait \triangleright (P ; Q)) \)
  by (subst-tac)
also have \( \ldots = (((I \& \$wait`) ; Q) < \$wait \triangleright (P ; Q)) \)
  by (metis (no-types, lifting) cond-def conj-pos-var-subst seqr-pre-var-out skip-var utp-pred-laws.inf-left-idem wait-vwb-lens)
also have \( \ldots = (((I[true/\$wait`] ; Q[true/\$wait`]) < \$wait \triangleright (P ; Q)) \)
  by (metis seqr-pre-transfer seqr-right-one-point true-alt-def uovar-cover upred-eq-true utp-rel.unrest-ouvar vwb-lens-mub wait-vwb-lens)
also have \( \ldots = (((I[true/\$wait`] ; (Ic < \$wait \triangleright Q)[true/\$wait`]) < \$wait \triangleright (P ; Q)) \)
  by (metis Healthy-def \( R3c\)-def assms(2))
also have \( \ldots = (((I[true/\$wait`] ; Ic[true/\$wait`]) < \$wait \triangleright (P ; Q)) \)
  by (subst-tac)
also have \( \ldots = (((I \& \$wait`) ; Ic) < \$wait \triangleright (P ; Q)) \)
  by (metis seqr-pre-transfer seqr-right-one-point true-alt-def uovar-cover upred-eq-true utp-rel.unrest-ouvar vwb-lens-mub wait-vwb-lens)
also have \( \ldots = ((I \& \$wait`) ; Ic) < \$wait \triangleright (P ; Q)) \)
  by (simp add: cond-def seqr-pre-transfer utp-rel.unrest-ouvar)
also have \( \ldots = (Ic < \$wait \triangleright (P ; Q)) \)
  by simp
also have \( \ldots = (Ic < \$wait \triangleright (P ; Q)) \)
  by simp
finally show \( ?\thesis \)
by (simp add: Healthy-def)
qed

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

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

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

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

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

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

lemma $R3h$-wait-true:
assumes $P$ is $R3h$
shows $P \; t = II_R \; t$
proof –
  have $P \; t = (II_R \triangleleft \text{ $wait$} \triangleright P) \; t$
    by (metis Healthy-if $R3h$-form assms)
  also have 
    $\ldots = II_R \; t$
    by (simp add: usubst)
  finally show ?thesis .
qed

2.5 RD2: A reactive specification cannot require non-termination

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

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

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

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

lemma RD2-mono: $P \subseteq Q \Rightarrow RD2(P) \subseteq RD2(Q)$
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: \( RD_2(R_3h(P)) = R_3h(RD_2(P)) \)
by (rel-auto)

2.6 Major healthiness conditions

**definition RH** :: \( (t::trace,\alpha) \text{ hrel-\text{rp}} \Rightarrow (t',\alpha) \text{ hrel-\text{rp}} (R) \)
where [upred-defs]: \( RH(P) = R_1(R_2c(R_3c(P))) \)

**definition RHS** :: \( (s,t::trace,\alpha) \text{ hrel-\text{rsp}} \Rightarrow (s,t',\alpha) \text{ hrel-\text{rsp}} (R_s) \)
where [upred-defs]: \( RHS(P) = R_1(R_2c(R_3h(P))) \)

**definition RD** :: \( (t::trace,\alpha) \text{ hrel-\text{rp}} \Rightarrow (t',\alpha) \text{ hrel-\text{rp}} \)
where [upred-defs]: \( RD(P) = RD_1(RD_2(RP(P))) \)

**definition SRD** :: \( (s,t::trace,\alpha) \text{ hrel-\text{rsp}} \Rightarrow (s,t',\alpha) \text{ hrel-\text{rsp}} \)
where [upred-defs]: \( SRD(P) = RD_1(RD_2(RHS(P))) \)

**lemma RH-comp**: \( RH = R_1 \circ R_2c \circ R_3c \)
by (auto simp add: RH-def)

**lemma RHS-comp**: \( RHS = R_1 \circ R_2c \circ R_3h \)
by (auto simp add: RHS-def)

**lemma RD-comp**: \( RD = RD_1 \circ RD_2 \circ RP \)
by (auto simp add: RD-def)

**lemma SRD-comp**: \( SRD = RD_1 \circ RD_2 \circ RHS \)
by (auto simp add: SRD-def)

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

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

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

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

**lemma RHS-idem**: \( R_s(R_s(P)) = R_s(P) \)
by (simp add: R1-R2c-is-R2 R1-R3h-commute R2-idem R2c-R3c-commute R2c-idem R3c-idem RHS-def)

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

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

**lemma RHS-mono**: \( P \subseteq Q \Rightarrow R_s(P) \subseteq R_s(Q) \)
using mono-def RHS-Monotonic by blast

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

lemma RHS-inf: \( R_s(\mathcal{P} \cap \mathcal{Q}) = R_s(\mathcal{P}) \cap R_s(\mathcal{Q}) \)
using Continuous-Disjunctous Disjunctuous-def RHS-Continuous by auto

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

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

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

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

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

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

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

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

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

lemma RD2-RHS-commute: \( RD2(R_s(\mathcal{P})) = R_s(RD2(\mathcal{P})) \)
by (metis R1-H2-commute R2c-H2-commute RD2-R3h-commute RD2-def RHS-def)

lemma SRD-idem: \( SRD(SRD(\mathcal{P})) = SRD(\mathcal{P}) \)
by (simp add: RD1-RD2-commute RD1-RHS-commute RD1-idem RD2-idem RHS-idem SRD-def)

lemma SRD-Idempotent [closure]: Idempotent SRD
by (simp add: Idempotent-def SRD-idem)
lemma SRD-Monotonic: Monotonic SRD
  by (simp add: Monotonic-comp RD1-Monotonic RD2-Monotonic RHS-Monotonic SRD-comp)

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

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

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

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)
  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)

lemma SRD-left-zero-2:
  assumes P is SRD
  shows (∃ $st · II)[true,true/$ok,$wait] ;; P = (∃ $st · II)[true,true/$ok,$wait]
proof –
  have (∃ $st · II)[true,true/$ok,$wait] ;; R3h(P) = (∃ $st · II)[true,true/$ok,$wait]
    by (rel-auto)
  thus ?thesis
    by (simp add: Healthy-if SRD-healths)

12
2.7 UTP theories

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

typedecl RDES
typedecl SRDES

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

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

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

definition sdes-hcond :: (SRDES, ('s,'t::trace,'α) rsp) uthy ⇒ (('s,'t,'α) rsp × ('s,'t,'α) rsp) health where
[upred-defs]: sdes-hcond T = SRD
end

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

interpretation rdes-theory-continuous: utp-theory-continuous UTHY(RDES, ('t::trace,'α) rp)
  rewrites ∧ P. P ∈ carrier (uthy-order RDES) ⟷ P is RD
  and carrier (uthy-order RDES) → carrier (uthy-order RDES) ≡ [RD]₇ → [RD]₇
  and le (uthy-order RDES) = op ⊆
  and eq (uthy-order RDES) = op =
  by (unfold-locales, simp-all add: rdes-hcond-def RD-Continuous)

interpretation rdes-real-galois:
galois-connection (RDES ⇔ (RD₁ ⊕ RD₂.R₃) → REA)

proof (simp add: mk-conn-def, rule galois-connectionI', simp-all add: utp-partial-order rdes-hcond-def rea-hcond-def)
  show R₃ ∈ [RD]₇ → [RP]₇
    by (metis (no-types, lifting) Healthy-def' Pi-I R₃-RD-RP RP-idem mem-Collect-eq)
  show RD₁ ⊕ RD₂ ∈ [RP]₇ → [RD]₇₇
    by (simp add: Pi-iff Healthy-def, metis RD-def RD-idem)
  show isotone (utp-order RD) (utp-order RP) R₃
    by (simp add: R₃-Monotonic isotone-utp-orderI)
  show isotone (utp-order RP) (utp-order RD) (RD₁ ⊕ RD₂)
    by (simp add: Monotonic-comp RD₁-Monotonic RD₂-Monotonic isotone-utp-orderI)

fix P :: ('a, 'b) hrel-rp
assume P is RD
thus P ⊆ RD₁ (RD₂ (R₃ P))
  by (metis Healthy-if R₃-RD-RP RD-def RP-idem eq-iff)

next
fix P :: ('a, 'b) hrel-rp
assume a: P is RP
thus R₃ (RD₁ (RD₂ P)) ⊆ 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)) ⊆ P
by (rel-auto)
ultimately show thesis
by (metis Healthy-if RP-mono a)
qed

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

interpretation srdes-theory: utp-theory UTHY(SRDES, ('s', 't': trace, 'α') rsp)
by (unfold-locales, simp-all add: srdes-hcond-def SRD-idem)

interpretation srdes-theory-continuous: utp-theory-continuous UTHY(SRDES, ('s', 't': trace, 'α') rsp)
rewrites ∀ P. P ∈ carrier (uthy-order SRDES) ↔ P is SRD
and P is HSRDES ↔ P is SRD
and (µ X · F (HSRDES X)) = (µ X · F (SRD X))
and carrier (uthy-order SRDES) → carrier (uthy-order SRDES) ≡ [SRD]₇ → [SRD]₇
and [HSRDES城市发展] → [HSRDES城市发展] ≡ [SRD]城市发展 → [SRD]城市发展
and le (uthy-order SRDES) = op ⊆
and eq (uthy-order SRDES) = op =
by (unfold-locales, simp-all add: srdes-hcond-def SRD-Continuous)

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

abbreviation Chaos :: ('s', 't': trace, 'α) hrel-rsp where
Chaos ≡ ⊥SRDES

abbreviation Miracle :: ('s', 't': trace, 'α) hrel-rsp where
Miracle ≡ ⊤SRDES

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

abbreviation srd-lfp (µR) where µR F ≡ µSRDES F
abbreviation srd-gfp (νR) where νR F ≡ νSRDES F

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

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

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

lemma srd-mu-equiv:
assumes Monotonic F F ∈ [SRD]城市发展 → [SRD]城市发展
shows \((\mu_R X \cdot F(X)) = (\mu X \cdot F(SRD(X)))\)
by \((\text{metis assms srdes-hcond-def srdes-theory-continuous.utp-lfp-def})\)
end

3 Reactive Design Specifications

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

3.1 Reactive design forms

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

lemma Chaos-def: \(\text{Chaos} = R_s(\text{false} \vdash \text{true})\)
proof –
  have \(\text{Chaos} = SRD(\text{false})\)
  by (metis srdes-hcond-def srdes-theory-continuous.healthy-bottom)
also have \(\ldots = R_u(H(\text{true}))\)
  by (simp add: SRD-RHS-H1-H2)
also have \(\ldots = R_u(\text{false} \vdash \text{false})\)
  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 \(\text{Miracle} = SRD(\text{false})\)
  by (metis srdes-hcond-def srdes-theory-continuous.healthy-top)
also have \(\ldots = R_u(H(\text{false}))\)
  by (simp add: SRD-RHS-H1-H2)
also have \(\ldots = R_u(\text{true} \vdash \text{false})\)
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 \(\$ok' \not\subset P \; \$ok' \not\subset Q\)
  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 \(\$ok' \not\subset P \; \$ok' \not\subset Q\)
  shows \(RD2(R_s(P \vdash Q)) = R_s(P \vdash Q)\)
using assms
lemma `wait-false-design`:
\[(P \vdash Q) \text{ f} = ((P \text{ f}) \vdash (Q \text{ f}))\]
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 \[\ldots = RD1(H2(R1(R2s(R3c(P))))))\]
  by (simp add: R1-R2s-R2c RD2-def)
also have \[\ldots = R1(R1(H2(R3c(R1(P))))))\]
  by (simp add: R1-H2-commute)
also have \[\ldots = R2(R1(H1(R3c(H2(R1(P))))))\]
  by (metis RD2-R3c-commute RD2-def)
also have \[\ldots = RH(H(R1(P))))\]
  by (metis RD-RH-design-form)
also have \[\ldots = RH(H(P))\]
  by (simp add: R1-R3c-commute RD1-via-R1)
also have \[\ldots = R2(R3c(R1(H(R1(P))))))\]
  by (simp add: R1-R3c-commute RD1-via-R1)
also have \[\ldots = RH((\neg P^f) \vdash P^f)\]
  by (simp add: H1-H2-eq-design)
also have \[\ldots = R((\neg P^f) \vdash P^f)\]
  by (metis RD-RH-design-form)
finally show \[\text{thesis} \].

qed

lemma `RD-reactive-design`:
assumes \[P \text{ is RD}\]
shows \[R((\neg P^f) \vdash P^f) = P\]
by (metis RD-RH-design-form)

lemma `SRD-RH-design-form`:
\[SRD(P) = R_s((\neg P^f) \vdash P^f)\]
proof

have \[ \text{SRD}(P) = R_1(R_2c(R_3h(RD_1(RD_2(R_1(P)))))) \]
  by (metis (no-types, lifting) R1-H2-commute R1-R2c-commute R1-R3h-commute R1-idem R2c-H2-commute RD1-R1-commute RD1-R2c-commute RD1-R3h-commute RD2-R3h-commute RD2-def RHS-def SRD-def)

also have \[ ... = R_1(R_2c(H(P))) \]
  by (metis (no-types, lifting) R1-H2-commute R1-R2c-commute R1-R3h-commute R2c-H2-commute RD1-R1-commute RD1-R2c-commute RD1-R3h-commute RD2-R3h-commute RD2-def RHS-def SRD-def)

also have \[ ... = R_1(R_2s(R_3h(H(P)))) \]
  by (metis (no-types, lifting) R1-H2-commute R1-R2c-is-R2 R1-R3h-commute R2-R1-form RD1-via-R1 RD2-def)

also have \[ ... = R_s(H(P)) \]
  by (simp add: R1-R2s-R2c RHS-def)

also have \[ ... = R_s((\neg P)[f] \vdash P[t]) \]
  by (simp add: H1-H2-eq-design)

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

qed

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

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

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

lemma \( \text{SRD-RHS-H1-H2} \): \( \text{SRD}(P) = R_s(H(P)) \)
  by (metis (no-types, lifting) H1-H2-eq-design R3h-subst-wait RHS-def SRD-RH-design-form subst-not wait-false-design)

3.2 Auxiliary healthiness conditions

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

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

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

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

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

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

lemma unrest-ok’-R3c-[pre] [unrest]: \( \#ok’ \not\in P \implies \#ok’ \not\in R_3c-[\text{pre}] (P) \)
  by (simp add: R3c-pre-def cond-def unrest)
Lemma unrest-ok-R3c-post \([\text{unrest}]\): \(\text{ok} \not\in P \implies \text{ok} \not\in R3c-post(P)\)
\(\text{by (simp add: R3c-post-def cond-def unrest)}\)

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

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

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

3.3 Composition laws

Theorem R1-design-composition:

Fixes \(P, Q \in \{t, r\} \cdot \text{trace, } \alpha, \beta, \gamma\) rel-rp

And \(R, S \in \{t, \beta, \gamma\} \cdot \text{rel-rp}\)

Assumes \(\text{ok}' \not\in P \text{ ok}' \not\in Q \text{ ok} \not\in R \text{ ok} \not\in S\)

Shows
\(\Pi 1(P \equiv Q) \equiv \Pi 1(R \equiv S) =
\Pi 1((\neg (\Pi 1(\neg P)) \equiv \Pi 1(\true)) \lor \neg (\Pi 1(\neg Q) \equiv \Pi 1(\neg R)) \lor (\Pi 1(Q) \equiv \Pi 1(S)))\)

Proof —

Have \(\Pi 1(P \equiv Q) \equiv \Pi 1(R \equiv S) = (\exists \text{ ok} \equiv (\Pi 1(P \equiv Q)[\langle \text{ok} \equiv \true \rangle; (\Pi 1(R \equiv S)[\langle \text{ok} \equiv \true \rangle)])\)

Using seqr-middle ok-vwb-lens by blast

Also from assms have \(\Pi \equiv (\exists \text{ ok} \equiv (\Pi 1(\neg P) \equiv (\langle \text{ok} \equiv \true \rangle \land Q)) \equiv \Pi 1((\langle \text{ok} \equiv \true \rangle \land R) \equiv (\langle \text{ok} \equiv \true \rangle \land S)))\)

By (simp add: design-def R1-def usubst unrest)

Also from assms have \(\Pi \equiv (((\Pi 1(\langle \text{ok} \equiv \true \rangle \land P) \equiv (\true \land Q)) \equiv \Pi 1((\true \land R) \equiv (\langle \text{ok} \equiv \true \rangle \land S)))\)

By (simp add: false-alt-def true-alt-def)

Also from assms have \(\Pi \equiv (((\Pi 1((\langle \text{ok} \equiv \true \rangle \land P) \equiv (\langle \text{ok} \equiv \true \rangle \land S)))\)

By (simp)

Also from assms have \(\Pi \equiv (((\Pi 1(\neg \text{ok} \lor \neg P \lor Q) \equiv \Pi 1(\neg R \lor (\langle \text{ok} \equiv \true \rangle \land S)))\)

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

Also from assms have \(\Pi \equiv (((\Pi 1(\neg \text{ok} \lor \neg P \lor Q) \equiv \Pi 1(\neg R \lor (\langle \text{ok} \equiv \true \rangle \land S)))\)

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

Also from assms have \(\Pi \equiv (((\Pi 1(\neg \text{ok} \lor \neg P \lor Q) \equiv \Pi 1(\neg R \lor (\langle \text{ok} \equiv \true \rangle \land S)))\)

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

Also from assms have \(\Pi \equiv (((\Pi 1(Q) \equiv \Pi 1(\neg R \lor (\langle \text{ok} \equiv \true \rangle \land S)))\)

By (rel-blast)

Also from assms have \(\Pi \equiv (((\Pi 1(Q) \equiv \Pi 1(\neg R \lor (\langle \text{ok} \equiv \true \rangle \land S)))\)

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

Also have \(\Pi \equiv (((\Pi 1(Q) \equiv \Pi 1(\neg R \lor (\langle \text{ok} \equiv \true \rangle \land S)))\)

By (simp add: R1-disj seqr-or-distl)

Also have \(\Pi \equiv (((\Pi 1(Q) \equiv \Pi 1(\neg R \lor (\langle \text{ok} \equiv \true \rangle \land S)))\)

By (simp add: R1-disj)}
proof  
  have \((R1(\neg \$ok) :: (\ell,\alpha,\ell) \text{ rel-rp}) :: R1(\text{true})) = \)
  \((R1(\neg \$ok) :: (\ell,\alpha,\ell) \text{ rel-rp}) \)
  by (rel-auto) 
  thus \(\text{thesis} \)
  by simp 
qed 

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

theorem R1-design-composition-RR: 
  assumes P is RR Q is RR R is RR S is RR 
  shows \((R1(P \vdash Q) :: R1(R \vdash S)) = R1(((\neg, P) wp_r false \land Q wp_r R) \vdash (Q :: S))\) 
  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)) 
  by (simp add: R2s-def design-def usubst)
lemma R2c-design: $R2c(P \vdash Q) = (R2c(P) \vdash R2c(Q))$
by (simp add: design-def impl-alt-def R2c-disj R2c-not R2c-ok R2c-and R2c-ok')

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

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

lemma R3c-R1-design-composition:
assumes $\$ok' \notin P \$ok' \notin Q \$ok \notin R \$ok \notin S$
shows $(R3c(R1(P \vdash Q)) ;; R3c(R1(R \vdash S))) =$
$R3c(R1((\neg (R1(\neg P)) ;; R1(true)) \land \neg ((R1(Q) \land \neg $wait$') ;; R1(\neg R)))
\vdash (R1(Q) ;; ([/[D \land $wait$ \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 $wait$') ;; R1 (\neg R)))$
by (rel-auto, blast+)

have 3:$(R1 (R3c-post Q) ;; R1 (R3c-post S)) = R3c-post (R1 Q ;; ([/[D \land $wait$ \triangleright R1(S)]))$
by (rel-auto)

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

qed

lemma R3h-R1-design-composition:
assumes $\$ok' \notin P \$ok' \notin Q \$ok \notin R \$ok \notin S$
shows $(R3h(R1(P \vdash Q)) ;; R3h(R1(R \vdash S))) =$
$R3h(R1((\neg (R1(\neg P)) ;; R1(true)) \land \neg ((R1(Q) \land \neg $wait$') ;; R1(\neg R)))
\vdash (R1(Q) ;; (\exists st \cdot /[D \land $wait$ \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 $wait$') ;; R1 (\neg R)))$
by (rel-auto, blast+)

have 3:$(R1 (R3h-post Q) ;; R1 (R3h-post S)) = R3h-post (R1 Q ;; ([/[D \land $wait$ \triangleright R1(S)]))$
by (rel-auto, blast+)

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

qed

lemma R2-design-composition:
assumes $\$ok' \notin P \$ok' \notin Q \$ok \notin R \$ok \notin S$
shows $(R2(P \vdash Q) ;; R2(R \vdash S)) =$
$R2((\neg (R1 (\neg R2c P) ;; R1 true) \land \neg (R1 (R2c Q) ;; R1 (\neg R2c R))) \vdash (R1 (R2c Q) ;; R1 (R2c S)))$
apply (simp add: R2-R2c-def R2c-design R1-design-composition assms unrest R2c-not R2c-and R2c-disj
R1-R2c-commute \Rightarrow R2c-idem R2c-R1-seq
\begin{proof}
apply (metis (no-types, lifting) R2c-R1-seq R2c-not R2c-true)
done
\end{proof}

\begin{lemma} RH-design-composition:
\begin{assumes}$R_1 \Rightarrow P \Rightarrow \neg \neg Q \Rightarrow R \Rightarrow S$$\end{assumes}
\begin{shows}(RH(P \Rightarrow Q)) \Rightarrow (RH(R \Rightarrow S)) = RH((\neg (R_1 \Rightarrow \neg R_2s P)) \Rightarrow R_1 true) \Rightarrow \neg ((R_1 (R2s Q) \Rightarrow \neg \$wait')) \Rightarrow R_1 (\neg \neg R_2s R)) \Rightarrow (R_1 (R2s Q)) \Rightarrow (\neg ||D \& \$wait \Rightarrow R_1 (R2s S))))\end{shows}
\end{lemma}

\begin{proof}
\begin{have}1: R2c (R1 (\neg R2s P) \Rightarrow R1 true) = (R1 (\neg R2s P) \Rightarrow R1 true)\end{have}
\begin{proof}
\begin{have}1:(R1 (\neg R2s P) \Rightarrow R1 true) = (R1(R2 (\neg P) \Rightarrow R2 true))\end{have}
by (rel-auto)
\begin{have}R2c(R1(R2 (\neg P) \Rightarrow R2 true)) = R2c(R1(R2 (\neg P) \Rightarrow R2 true))\end{have}
using R2c-not by blast
\begin{also have}R2c(R2 (\neg P) \Rightarrow R2 true)\end{also have}
by (metis R1-R2c-commute R1-R2c-is-R2)
\begin{also have}R2 (\neg P) \Rightarrow R2 true\end{also have}
by (simp add: R2-seq-distribute)
\begin{finally show}thesis\end{finally show}
by (simp add: 1)
\end{proof}
\end{have}
\begin{have}2:R2c ((R1 (R2s Q) \Rightarrow \neg \$wait') \Rightarrow R1 (\neg R2s R)) = ((R1 (R2s Q) \Rightarrow \neg \$wait') \Rightarrow R1 (\neg R2s R))\end{have}
\begin{proof}
\begin{have}((R1 (R2s Q) \Rightarrow \neg \$wait') \Rightarrow R1 (\neg R2s R)) = R1 (R2 (Q \Rightarrow \neg \$wait') \Rightarrow R2 (\neg R))\end{have}
by (rel-auto)
\begin{hence}R2c ((R1 (R2s Q) \Rightarrow \neg \$wait') \Rightarrow R1 (\neg R2s R)) = (R2 (Q \Rightarrow \neg \$wait') \Rightarrow R2 (\neg R))\end{hence}
by (metis R1-R2c-commute R1-R2c-is-R2 R2-seq-distribute)
\begin{also have}R2 (\neg P) \Rightarrow R2 true\end{also have}
by (simp add: R2-def R2-not R2-s_true)
\begin{finally show}thesis\end{finally show}
by (rel-auto)
\end{proof}
\end{have}
\begin{have}3:R2c((R1 (R2s Q) \Rightarrow (||D \& \$wait \Rightarrow R1 (R2s S)))) = (R1 (R2s Q) \Rightarrow (||D \& \$wait \Rightarrow R1 (R2s S))))\end{have}
\begin{proof}
\begin{have}R2c(((R1 (R2s Q))true/\$wait') \Rightarrow (||D \& \$wait \Rightarrow R1 (R2s S))true/\$wait') = ((R1 (R2s Q))true/\$wait') \Rightarrow (||D \& \$wait \Rightarrow R1 (R2s S))true/\$wait')\end{have}
by (simp add: usual-2-cond-unit-T R1_def R2s_def)
\begin{also have}R2c(R2 (Qtrue/\$wait') \Rightarrow R2(||Dtrue/\$wait') = R2c(R2 (Qtrue/\$wait') \Rightarrow R2(||Dtrue/\$wait')\end{also have}
by (metis R2-s_def R2-des-lift-skip R2-subst-wait_true)
\begin{also have}R2c(Qtrue/\$wait') \Rightarrow R2(||Dtrue/\$wait')\end{also have}
using R2c-seq by blast
\begin{also have}R2c(Qtrue/\$wait') \Rightarrow R2(||Dtrue/\$wait')\end{also have}
apply (simp add: usual-2-cond-unit-T R1_def R2s_def)
apply (metis R2-s_def R2-des-lift-skip R2-subst-wait_true R2-subst-wait_true)
done
\end{proof}
\end{have}
\end{proof}

21
finally show \( \theta \text{thesis} \).

qed

moreover have \( R2c(((R1 (R2s Q)) [\text{false}/$\text{wait}^*] ;; ([\Pi] D \triangleleft $$\text{wait} \triangleright R1 (R2s S))] [\text{false}/$\text{wait}]) = ((R1 (R2s Q))[\text{false}/$\text{wait}^*] ;; ([\Pi] D \triangleleft $$\text{wait} \triangleright R1 (R2s S))] [\text{false}/$\text{wait}])

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

(meta (no-types, hide-lams) \( R1\text{-wait}'\text{-false} R1\text{-wait-false} R2\text{-def} R2\text{-subst-wait}'\text{-false} R2\text{-subst-wait-false} \)

ultimately show \( \theta \text{thesis} \)

proof –

have \( [\Pi] D \triangleleft $$\text{wait} \triangleright R1 (R2s S) = R2 (\Pi) \triangleleft $$\text{wait} \triangleright S)

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

then show \( \theta \text{thesis} \)

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

qed

have \( (R1 (R2s (R3c(P \vdash Q))) ; R1 (R2s (R3c(R \vdash S)))) = (R3c(R2s (P \vdash Q)) ;; R3c(R1 (R2s (P \vdash Q)) ;; R3c(R1 (R2s (P \vdash Q))))

by (metis (no-types, hide-lams) R1-R2-R2c-R2c-R2c-commute R2c-R3c-commute R2s-design)

also have \( ... = R3c(R1 (\neg (R1 (\neg R2s P) ;; R1 true) \land \neg ((R1 (R2s Q) \land \neg $$\text{wait}^*)) ;; R1 (\neg R2s R))) \vdash (R1 (R2s Q) ;; (\Pi D \triangleleft $$\text{wait} \triangleright R1 (R2s S))))

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

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

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

qed

lemma \( \text{RHS-design-composition} \):

assumes \( \text{ok}' P \equiv P \equiv Q \equiv Q \equiv \text{ok} \equiv R \equiv S \equiv S \)

shows \( \text{ok}(P \vdash Q) ;; \text{ok}(R \vdash S) = (\text{ok}(\neg (\neg R2s P) ;; R1 true) \land \neg ((\neg R2s Q) \land (\neg $$\text{wait}^*)) ;; R1 (\neg R2s R))) \vdash (R1 (R2s Q) ;; (\exists \text{st} (\Pi D) \triangleleft $$\text{wait} \triangleright R1 (R2s S)))

proof –

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

proof –

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

by (rel-auto, blast)

have \( R2c(R1(R2 (\neg P) ;; R2 true)) = R2c(R1(R2 (\neg P) ;; R2 true)) \) using R2c-not by blast

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

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

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

by (simp add: R2c-seq-distribute)

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

by (simp add: R2c-def R2s-not R2s-true)

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

by (simp add: 1)

qed

have \( 2:R2c(((R1 (R2s Q) \land (\neg $$\text{wait}^*)) ;; R1 (\neg R2s R)) = ((R1 (R2s Q) \land (\neg $$\text{wait}^*)) ;; R1 (\neg R2s R)) \)

22
proof

have 
((R1 (R2s Q) ∧ ¬ $\text{wait'}$) ;; R1 (¬ R2s R)) = R1 (R2 (Q ∧ ¬ $\text{wait'}$) ;; R2 (¬ R))
by (rel-auto, blast+)

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

qed

have 3.R2c(((R1 (R2s Q) ;; (∃ $\text{st} \cdot [I]_D) < $\text{wait} \triangleright R1 (R2s S)))) = (R1 (R2s Q) ;; (∃ $\text{st} \cdot [I]_D) < $\text{wait} \triangleright R1 (R2s S)))
proof

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

qed

moreover have R2c(((R1 (R2s Q))[false/$\text{wait'}$] ;; (∃ $\text{st} \cdot [I]_D) < $\text{wait} \triangleright R1 (R2s S))[false/$\text{wait}']))) = ((R1 (R2s Q))[false/$\text{wait'}$] ;; (∃ $\text{st} \cdot [I]_D) < $\text{wait} \triangleright R1 (R2s S))[false/$\text{wait}'])))
by (simp add: usubst)
(metis (no-types, lifting) R1-wait'-false R1-wait 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) ∧ ¬ $\text{wait'}$) ;; R1 (¬ R2s R))) ⊢
(R1 (R2s Q) ;; (∃ $\text{st} \cdot [I]_D) < $\text{wait} \triangleright R1 (R2s S))))
by (simp add: R3h-R1-design-composition assms unrest)
also have ...
R3h(R1(R2c((¬ R1 (¬ R2s P) ;; R1 true) ∧ ¬ ((R1 (R2s Q) ∧ ¬ $\text{wait'}$) ;; R1 (¬ R2s R)))) ⊢
(R1 (R2s Q) ;; (∃ $\text{st} \cdot [I]_D) < $\text{wait} \triangleright R1 (R2s S))))
by (simp add: R2c-design R2c-and R2c-not 1 2 3)
finally show ?thesis
by (simp add: R1-R2s-R2c R1-R3h-commute R2c-R3h-commute RHS-def)

qed

lemma RHS-R2s-design-composition:
assumes
shows \((R \models (\neg (R1 (\neg P) \supset R1 \text{ true}) \land \neg ((R1 Q \land \neg \text{ wait } \supset R1 (~ R)))) \supset (R1 Q ;; (\exists \text{ st } \cdot [\text{ II }] D) \triangleleft \text{ wait } \triangleright R1 S))\)

proof
- have \(f1: R2s P = P\)
  by (meson Healthy-def assms\((5)\))
- have \(f2: R2s Q = Q\)
  by (meson Healthy-def assms\((6)\))
- have \(f3: R2s R = R\)
  by (meson Healthy-def assms\((7)\))
- have \(R2s S = S\)
  by (meson Healthy-def assms\((8)\))
then show \(?thesis\)
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 RH-design-export-R2: \(R_s(P \vdash Q) = R_s(P \vdash R2(Q))\)
by (rel-auto)

lemma R1-design-R1-pre: \(R_s((\neg R1 P) \vdash R) = R_s((\neg P) \vdash R)\)
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: \(R_s ((\neg R1 P) \land Q) \vdash R) = R_s ((\neg P) \land Q) \vdash R)\)
by (rel-auto)

lemma RHS-pre-lemma: \((R_s P)^f = R1(R2c(P^f))\)
3.4 Refinement introduction laws

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

lemma R1-design-refine:
assumes
\[ P_1 \text{ is } R1 \quad P_2 \text{ is } R1 \quad Q_1 \text{ is } R1 \quad Q_2 \text{ is } R1 \]
\[ \$ok \not\in P_1 \quad \$ok \not\in P_2 \quad \$ok \not\in Q_1 \quad \$ok \not\in Q_2 \]
\[ \exists R1 \vdash Q_1 \quad \exists R1 \vdash Q_2 \]
shows \( R1(P_1 \vdash P_2) \subseteq R1(Q_1 \vdash Q_2) \)
\[ \iff \quad \exists P_1 \Rightarrow Q_1 \quad \land \quad \exists P_1 \Rightarrow Q_2 \]
proof –
\[ \text{have } R1((\exists \$ok:\$ok \cdot P_1) \vdash (\exists \$ok:\$ok \cdot P_2)) \subseteq R1((\exists \$ok:\$ok \cdot Q_1) \vdash (\exists \$ok:\$ok \cdot Q_2)) \]
\[ \iff \quad (\exists \$ok:\$ok \cdot P_1) \Rightarrow (\exists \$ok:\$ok \cdot Q_1) \quad \land \quad (\exists \$ok:\$ok \cdot P_1) \Rightarrow (\exists \$ok:\$ok \cdot Q_2) \]
\[ \land \quad Q_2 \Rightarrow R1((\exists \$ok:\$ok \cdot P_2) \]
by (rel-auto, meson+)
thus \[ ?thesis \]
by (simp-all add: ex-unrest ex-plus Healthy-if assms)
qed

lemma R1-design-refine-RR:
assumes \[ P_1 \text{ is } RR \quad P_2 \text{ is } RR \quad Q_1 \text{ is } RR \quad Q_2 \text{ is } RR \]
shows \( R1(P_1 \vdash P_2) \subseteq R1(Q_1 \vdash Q_2) \)
\[ \iff \quad \exists P_1 \Rightarrow Q_1 \quad \land \quad \exists P_1 \Rightarrow Q_2 \]
by (simp add: R1-design-refine assms unrest closure)

lemma RHS-design-refine:
assumes \[ P_1 \text{ is } R1 \quad P_2 \text{ is } R1 \quad Q_1 \text{ is } R1 \quad Q_2 \text{ is } R1 \]
\[ P_1 \text{ is } R2c \quad P_2 \text{ is } R2c \quad Q_1 \text{ is } R2c \quad Q_2 \text{ is } R2c \]
\[ \$ok \not\in P_1 \quad \$ok \not\in P_2 \quad \$ok \not\in Q_1 \quad \$ok \not\in Q_2 \]
\[ \exists R1 \vdash Q_1 \quad \exists R1 \vdash Q_2 \]
\[ \exists wait \not\in P_1 \quad \exists wait \not\in P_2 \quad \exists wait \not\in Q_1 \quad \exists wait \not\in Q_2 \]
shows \( R_{s}(P_1 \vdash P_2) \subseteq R_{s}(Q_1 \vdash Q_2) \)
\[ \iff \quad \exists P_1 \Rightarrow Q_1 \quad \land \quad \exists P_1 \Rightarrow Q_2 \]
proof –
\[ \text{have } R_{s}(P_1 \vdash P_2) \subseteq R_{s}(Q_1 \vdash Q_2) \implies R_{s}(R3h(R2c(P_1 \vdash P_2)) \subseteq R_{s}(R3h(R2c(Q_1 \vdash Q_2))) \]
by (simp add: R2c-R3h-commute RHS-def)
also have \[ ... \implies R_{s}(R3h(P_1 \vdash P_2)) \subseteq R_{s}(R3h(Q_1 \vdash Q_2)) \]
by (simp add: Healthy-if R2c-design assms)
also have \[ ... \implies R_{s}(R3h(P_1 \vdash P_2))[false/\$wait] \subseteq R_{s}(R3h(Q_1 \vdash Q_2))[false/\$wait] \]
by (rel-auto, meson+)
also have \[ ... \implies R_{s}(R3h(P_1 \vdash P_2))[false/\$wait] \subseteq R_{s}(Q_1 \vdash Q_2)[false/\$wait] \]
by (rel-auto)
also have \[ ... \implies R_{s}(P_1 \vdash P_2) \subseteq R_{s}(Q_1 \vdash Q_2) \]
by (simp add: usubst assms closure unrest)
also have \[ ... \implies P_1 \Rightarrow Q_1 \quad \land \quad P_1 \Rightarrow Q_2 \]
by (simp add: R1-design-refine assms)
finally show \[ ?thesis \]
qed

lemma srdes-refine-intro:
assumes \[ P_1 \Rightarrow P_2 \quad \exists P_1 \Rightarrow Q_1 \quad \land \quad \exists P_2 \Rightarrow Q_2 \]
shows \( R_{s}(P_1 \vdash Q_1) \subseteq R_{s}(P_2 \vdash Q_2) \)
by (simp add: RHS-mono assms design-refine-intro)
3.5 Distribution laws

lemma \textit{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 \textit{RHS-inf} design-choice)

lemma \textit{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 \textit{RHS-sup} design-inf)

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

end

4 Reactive Design Triples

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

4.1 Diamond notation

definition \textit{wait’-cond} :: 
(\(’t::\text{trace},’\alpha,’\beta\) rel-rp \(\Rightarrow (’t,’\alpha,’\beta)\) rel-rp \(\Rightarrow (’t,’\alpha,’\beta)\) rel-rp (infixr \circ 65) where

[upred-defs]: \( P \circ Q = (P \triangleq \$\text{wait’} \circ Q) \)

lemma \textit{wait’-cond-unrest} [unrest] :
\[ \text{out-var \ wait} \triangleright x \cdot x \not \in P \cdot x \not \in Q \] \(\implies\) \( x \not \in (P \circ Q) \)
by (simp add: \textit{wait’-cond-def unrest})

lemma \textit{wait’-cond-subst} [usubst] :
\( $\text{wait’} \uplus \sigma \implies \sigma \uplus (P \circ Q) = (\sigma \uplus P) \circ (\sigma \uplus Q) \)
by (simp add: \textit{wait’-cond-def usubst unrest})

lemma \textit{wait’-cond-left-false} [usubst] :
false \(\circ P = (\neg \$\text{wait’} \land P) \)
by (rel-auto)

lemma \textit{wait’-cond-seq} [usubst] :
\( (P \circ Q) \triangleq R = ((P \triangleq (\$\text{wait} \land R)) \lor (Q \triangleq (\neg \$\text{wait} \land R))) \)
by (simp add: \textit{wait’-cond-def cond-def sepr-or-distl, rel-blast})

lemma \textit{wait’-cond-true} [usubst] :
\( P \circ Q \land \$\text{wait’} = (P \land \$\text{wait’}) \)
by (rel-auto)

lemma \textit{wait’-cond-false} [usubst] :
\( P \circ Q \land (\neg \$\text{wait’}) = (Q \land (\neg \$\text{wait’})) \)
by (rel-auto)

lemma \textit{wait’-cond-idem} [usubst] :
\( P \circ P = P \)
by (rel-auto)

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

lemma \textit{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]: \((P \odot Q)[false/\$wait'] = Q[false/\$wait']\)
  by (rel-auto)

lemma subst-wait’-left-subst: \((P[true/\$wait'] \odot Q) = (P \odot Q)\)
  by (rel-auto)

lemma subst-wait’-right-subst: \((P \odot Q[false/\$wait']) = (P \odot Q)\)
  by (rel-auto)

lemma wait’-cond-split: \(P[true/\$wait'] \odot P[false/\$wait'] = P\)
  by (simp add: wait’-cond-def cond-var-split)

lemma wait’-cond-associ [simp]: \(P \odot Q \odot R = P \odot R\)
  by (rel-auto)

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

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

lemma R1-wait’-cond: \(R1(P \odot Q) = R1(P) \odot R1(Q)\)
  by (rel-auto)

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

lemma R2-wait’-cond: \(R2(P \odot Q) = R2(P) \odot R2(Q)\)
  by (simp add: R2-def R2s-wait’-cond R1-wait’-cond)

lemma wait’-cond-R1-closed [closure]:
\[\[ P \iff R1: Q is R1 \] \Longrightarrow P \odot Q is R1\]
  by (simp add: Healthy-def R1-wait’-cond)

lemma wait’-cond-R2c-closed [closure]: \[ P \iff R2c: Q is R2c \] \Longrightarrow P \odot Q is R2c
  by (simp add: R2c-condr wait’-cond-def Healthy-def, rel-auto)

4.2 Export laws

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

lemma RH-design-post-R1: \(R(P \iff Q \odot R1(R)) = R(P \iff Q \odot R)\)
  by (metis R1-wait’-cond RH-design-export-R1 RH-design-peri-R1)

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

lemma RH-design-post-R2s: \(R(P \iff Q \odot R2s(R)) = R(P \iff Q \odot R)\)
  by (metis (no-types, lifting) R2s-idem R2s-wait’-cond RH-design-export-R2s)

lemma RH-design-peri-R2c: \(R(P \iff R2c(Q) \odot R) = R(P \iff Q \odot R)\)
  by (metis R1-R2s-R2c RH-design-peri-R1 RH-design-peri-R2s)

lemma RHS-design-peri-R1: \(R_s(P \iff R1(Q) \odot R) = R_s(P \iff Q \odot R)\)
by (metis (no-types, lifting) R1-idem R1-wait'-cond RHS-design-export-R1)

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

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

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

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

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

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

4.3 Pre-, peri-, and postconditions

4.3.1 Definitions

abbreviation \( \text{pre}_s \equiv \{ \text{ok} \mapsto \text{true}, \text{ok}' \mapsto \text{false}, \text{wait} \mapsto \text{false} \} \)
abbreviation \( \text{cmt}_s \equiv \{ \text{ok} \mapsto \text{true}, \text{ok}' \mapsto \text{true}, \text{wait} \mapsto \text{false} \} \)
abbreviation \( \text{peri}_s \equiv \{ \text{ok} \mapsto \text{true}, \text{ok}' \mapsto \text{true}, \text{wait} \mapsto \text{false}, \text{wait}' \mapsto \text{true} \} \)
abbreviation \( \text{post}_s \equiv \{ \text{ok} \mapsto \text{true}, \text{ok}' \mapsto \text{true}, \text{wait} \mapsto \text{false}, \text{wait}' \mapsto \text{false} \} \)

abbreviation \( \text{np}_{\text{pre}}(P) \equiv \text{pre}_s \uparrow P \)
definition [upred-defs]: \( \text{pre}_R(P) = (\neg \text{np}_{\text{pre}}(P)) \)
definition [upred-defs]: \( \text{cmt}_R(P) = R1(\text{cmt}_s \uparrow P) \)
definition [upred-defs]: \( \text{peri}_R(P) = R1(\text{peri}_s \uparrow P) \)
definition [upred-defs]: \( \text{post}_R(P) = R1(\text{post}_s \uparrow P) \)

4.3.2 Unrestriction laws

lemma ok-pre-unrest [unrest]: \( \text{ok} \notin P \)
by (simp add: \( \text{pre}_{R \Rightarrow \text{unrest}} \text{unrest} \text{subst} \))

lemma ok-peri-unrest [unrest]: \( \text{ok} \notin P \)
by (simp add: \( \text{peri}_{R \Rightarrow \text{unrest}} \text{unrest} \text{subst} \))

lemma ok-post-unrest [unrest]: \( \text{ok} \notin P \)
by (simp add: \( \text{post}_{R \Rightarrow \text{unrest}} \text{unrest} \text{subst} \))

lemma ok-cmt-unrest [unrest]: \( \text{ok} \notin P \)
by (simp add: \( \text{cmt}_{R \Rightarrow \text{unrest}} \text{unrest} \text{subst} \))

lemma ok'-pre-unrest [unrest]: \( \text{ok}' \notin P \)
by (simp add: \( \text{pre}_{R \Rightarrow \text{unrest}} \text{unrest} \text{subst} \))
lemma ok′-peri-unrest [unrest]: $\mathsf{ok} \not\in \mathsf{peri}_R P$
by (simp add: peri_R-def unrest usubst)

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

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

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

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

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

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

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

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

4.3.3 Substitution laws

lemma pre_s-design: $\mathsf{pre}_s \vdash (P \vdash Q) = (\neg \mathsf{pre}_s \vdash P)$
by (simp add: design-def pre_R-def usubst)

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

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

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

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

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

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

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

lemma post_s-R1 [usubst]: $\mathsf{post}_s \upharpoonright R1(\mathsf{post}_s \vdash P) = R1(\mathsf{post}_s \upharpoonright P)$
by (simp add: R1-def usubst)
lemma post\_s-R2c \([\text{subst}]: post\_s \downarrow R2c(P) = R2c(post\_s \downarrow P)
\)  
by \((\text{simp add: R2c-def R2s-def usubst})\)

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

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

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

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

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

lemma rea-cmt-RHS-design: \(\text{cmt}_R(\text{R}_s(P \vdash Q) = R1(R2c(cmt\_s \downarrow (P \Rightarrow Q)))\)
by \((\text{simp add: RHS-def usubst R3h-def cmtR-def cmt\_s-design R1-idem})\)

lemma rea-peri-RHS-design: \(\text{peri}_R(\text{R}_s(P \vdash Q \circ R) = R1(R2c(peri\_s \downarrow (P \Rightarrow r Q)))\)
by \((\text{simp add: RHS-def usubst periR-def R3h-def peri\_s-design, rel-auto})\)

lemma rea-post-RHS-design: \(\text{post}_R(\text{R}_s(P \vdash Q \circ R) = R1(R2c(post\_s \downarrow (P \Rightarrow r R)))\)
by \((\text{simp add: RHS-def usubst postR-def R3h-def post\_s-design, rel-auto})\)

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

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

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

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

4.3.4 Healthiness laws

lemma wait\_r\_unrest-pre-SRD \([\text{unrest}]:\n\\$\text{wait}^* \notin \text{pre}_R(P) \implies \$\text{wait}^* \notin \text{pre}_R(\text{SRD } P)\)
apply \((\text{rel-auto})\)
using \(\text{least-zero}\) apply \(\text{blast+}\)
done

lemma R1-R2s-cmt-SRD:
assumes \(P \text{ is SRD}\)
shows \(R1(R2s(cmt\_R(P))) = cmt\_R(P)\)
by \((\text{metis (no-types, lifting) R1-R2c-commute R1-R2s-R2c R1-idem R2c-idem SRD-reactive-design asms rea-cmt-RHS-design})\)
lemma R1-R2s-peri-SRD:
  assumes P is SRD
  shows R1(R2s(peri_R(P))) = peri_R(P)
  by (metis (no-types, hide-lams) Healthy-def R1-R2s-R2c R2-def R2-idem RHS-def SRD-RH-design-form assms R1-idem peri_R-def peri_R-R1 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 peri_R-SRD-R1 [closure]: P is SRD \implies peri_R(P) is R1
by (simp add: Healthy-def' R1-peri-SRD)

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

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

lemma post_R-SRD-R1 [closure]: P is SRD \implies post_R(P) is R1
by (simp add: Healthy-def' R1-post-SRD)
**Lemma R1-R2c-post-RHS:**

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

shows \( R1(R2c(post_R(P))) = post_R(P) \)

by (metis R1-R2s-R2c R1-R2s-post-SRD assms)

**Lemma R2c-cmt-conj-wait':**

\( P \text{ is SRD } \Rightarrow \ R2c(cmt_R P \land \neg \text{wait'}) = (cmt_R P \land \neg \text{wait'}) \)

by (simp add: R2-def R2s-conj R2s-not R2s-wait' R1-extend-conj R1-R2s-cmt-SRD)

**Lemma preR-R2c-closed [closure]:** \( P \text{ is SRD } \Rightarrow \ \text{pre}_R(P) \text{ is R2c} \)

by (simp add: Healthy-def)

**Lemma periR-R2c-closed [closure]:** \( P \text{ is SRD } \Rightarrow \ \text{peri}_R(P) \text{ is R2c} \)

by (simp add: Healthy-def)

**Lemma wpR-trace-ident-pre [wp]:**

\( (\exists tr' =u tr \land \lceil I \rceil_R) \Rightarrow wp_R \text{ pre}_R P = \text{ pre}_R P \)

by (rel-auto)

**Lemma R1-preR [closure]:**

\( \text{pre}_R(P) \text{ is R1} \)

by (rel-auto)

**Lemma trace-ident-left-periR:**

\( (\exists tr' =u tr \land \lceil I \rceil_R) \Rightarrow \text{peri}_R(P) = \text{peri}_R(P) \)

by (rel-auto)

**Lemma trace-ident-left-postR:**

\( (\exists tr' =u tr \land \lceil I \rceil_R) \Rightarrow \text{post}_R(P) = \text{post}_R(P) \)

by (rel-auto)

**Lemma trace-ident-right-postR:**

\( \text{post}_R(P) ; (\exists tr' =u tr \land \lceil I \rceil_R) = \text{post}_R(P) \)

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

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

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

4.3.5 Calculation laws

lemma \textit{wait'}-\textit{cond-peri-post-cmt} [\textit{rdes}]:
cmt_R P = peri_R (P \diamond post_R P)
by (rel-auto)

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

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

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

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

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

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

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

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

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

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

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

lemma \textit{postR-srdes-skip} [\textit{rdes}]: \( \text{post}_R(II_R) = (\text{str'} = u \text{str} \land [II]_R) \)

33
by (rel-auto)

lemma preR-INF [rdes]: $A \neq \{\} \implies \text{pre}_R(\bigwedge A \cdot P \in A \cdot \text{pre}_R(P))$
by (rel-auto)

lemma periR-INF [rdes]: $\text{peri}_R(\bigvee A \cdot \text{peri}_R(P))$
by (rel-auto)

lemma postR-INF [rdes]: $\text{post}_R(\bigvee A \cdot \text{post}_R(P))$
by (rel-auto)

lemma preR-UINF [rdes]: $\text{pre}_R(\bigcup i \cdot P(i)) = (\bigcup i \cdot \text{pre}_R(P(i))$
by (rel-auto)

lemma periR-UINF [rdes]: $\text{peri}_R(\bigcap i \cdot P(i)) = (\bigcap i \cdot \text{peri}_R(P(i))$
by (rel-auto)

lemma postR-UINF [rdes]: $\text{post}_R(\bigcap i \cdot P(i)) = (\bigcap i \cdot \text{post}_R(P(i))$
by (rel-auto)

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

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

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

lemma periR-UINF-member [rdes]: $\text{peri}_R(\bigcap i \in A \cdot P(i)) = (\bigcap i \in A \cdot \text{peri}_R(P(i))$
by (rel-auto)

lemma periR-UINF-member-2 [rdes]: $\text{peri}_R(\bigcap (i,j) \in A \cdot P \cdot i \cdot j) = (\bigcap (i,j) \in A \cdot \text{peri}_R(P \cdot i \cdot j)$
by (rel-auto)

lemma periR-UINF-member-3 [rdes]: $\text{peri}_R(\bigcap (i,j,k) \in A \cdot P \cdot i \cdot j \cdot k) = (\bigcap (i,j,k) \in A \cdot \text{peri}_R(P \cdot i \cdot j \cdot k)$
by (rel-auto)

lemma postR-UINF-member [rdes]: $\text{post}_R(\bigcap i \in A \cdot P(i)) = (\bigcap i \in A \cdot \text{post}_R(P(i))$
by (rel-auto)

lemma postR-UINF-member-2 [rdes]: $\text{post}_R(\bigcap (i,j) \in A \cdot P \cdot i \cdot j) = (\bigcap (i,j) \in A \cdot \text{post}_R(P \cdot i \cdot j)$
by (rel-auto)

lemma postR-UINF-member-3 [rdes]: $\text{post}_R(\bigcap (i,j,k) \in A \cdot P \cdot i \cdot j \cdot k) = (\bigcap (i,j,k) \in A \cdot \text{post}_R(P \cdot i \cdot j \cdot k)$
by (rel-auto)

lemma preR-inf [rdes]: $\text{pre}_R(P \cap Q) = (\text{pre}_R(P) \land \text{pre}_R(Q))$
by (rel-auto)

lemma periR-inf [rdes]: $\text{peri}_R(P \cap Q) = (\text{peri}_R(P) \lor \text{peri}_R(Q))$
by (rel-auto)

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

**lemma** \textit{srdes-skip-tri-design [rdes-def]}: \( \Pi_R = \text{R}_s(\text{true}_r \vdash \text{false} \circ \Pi_r) \)
\text{by (simp add: srdes-skip-def, rel-auto)}

**lemma** \textit{Chaos-tri-def [rdes-def]}: \( \text{Chaos} = \text{R}_s(\text{false} \vdash \text{false} \circ \text{false}) \)
\text{by (simp add: Chaos-def design-false-pre)}

**lemma** \textit{Miracle-tri-def [rdes-def]}: \( \text{Miracle} = \text{R}_s(\text{true}_r \vdash \text{false} \circ \text{false}) \)
\text{by (simp add: Miracle-def R1-design-R1-pre wait'-cond-idem)}

**lemma** \textit{RHS-tri-design-form}:
\hspace{1cm} assumes \( P_1 \) is RR \( P_2 \) is RR \( P_3 \) is RR
\hspace{1cm} shows \( \text{R}_s(P_1 \vdash P_2 \circ P_3) = (\Pi_R \circ \text{RR} \circ \Pi_R) (P_1 \circ \text{RR} (P_2) \circ \Pi_R (P_3))) \)
\text{proof} –
\hspace{1cm} have \( \text{R}_s(\text{RR}(P_1) \vdash \text{RR}(P_2) \circ \text{RR}(P_3)) = (\Pi_R \circ \text{RR} \circ \Pi_R (P_1) \circ \text{RR} (P_2)) \circ \Pi_R (P_3))) \)
\hspace{1cm} \text{apply (rel-auto) using minus-zero-eq by blast}
\hspace{1cm} \text{thus } \exists \text{thesis}
\hspace{1cm} \text{by (simp add: Healthy-if assms)}
\text{qed}

**lemma** \textit{RHS-design-pre-post-form}:
\( \text{R}_s((\neg P^f_1) \vdash P^f_2) = \text{R}_s(\text{pre}_R(P) \vdash \text{cmt}_R(P)) \)
\text{by (simp add: design-subst-ok)}
\hspace{1cm} \text{also have } \ldots = \text{R}_s(\text{pre}_R(P) \vdash \text{cmt}_R(P))
\hspace{1cm} \text{by (simp add: pre-R-def cmt-R-def usubst, rel-auto)}
\hspace{1cm} \text{finally show } \exists \text{thesis} .
\text{qed}

**lemma** \textit{SRD-as-reactive-design}:
\( \text{SRD}(P) = \text{R}_s(\text{pre}_R(P) \vdash \text{cmt}_R(P)) \)
\text{by (simp add: RHS-design-pre-post-form SRD-RH-design-form)}

**lemma** \textit{SRD-reactive-design-alt}:
\hspace{1cm} assumes \( P \) is SRD
\hspace{1cm} shows \( \text{R}_s(\text{pre}_R(P) \vdash \text{cmt}_R(P)) = P \)
\text{proof} –
\hspace{1cm} have \( \text{R}_s(\text{pre}_R(P) \vdash \text{cmt}_R(P)) = \text{R}_s((\neg P^f_1) \vdash P^f_2) \)
\hspace{1cm} \text{by (simp add: RHS-design-pre-post-form)}
\hspace{1cm} \text{thus } \exists \text{thesis}
\hspace{1cm} \text{by (simp add: SRD-reactive-design assms)}
\text{qed}
lemma SRD-reactive-tri-design-lemma:
SRD(P) = Rₘ[(¬ P'[::-1][true/$\$wait\'] ∨ P'[::-1][false/$\$wait\'])
by (simp add: SRD-RH-design-form wait'-cond-split)

lemma SRD-as-reactive-tri-design:
SRD(P) = Rₘ(preₐ(P) ⊓ periₐ(P) ⊓ postₐ(P))
proof –
  have SRD(P) = Rₘ[(¬ P'[::-1][true/$\$wait\'] ∨ P'[::-1][false/$\$wait\'])
  by (simp add: SRD-RH-design-form wait'-cond-split)
  also have ... = Rₘ(preₐ(P) ⊓ periₐ(P) ⊓ postₐ(P))
  apply (simp add: usubst)
  apply (subst design-subst-ok-ok[THEN sym])
  apply (simp add: preₐ-def periₐ-def postₐ-def usubst unrest)
  apply (rel-auto)
done
finally show ?thesis .
qed

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

lemma SRD-elim [RD-elim]: [ P is SRD; Q(Rₘ(preₐ(P) ⊓ periₐ(P) ⊓ postₐ(P))) ] ⇒ Q(P)
by (simp add: SRD-reactive-tri-design)

lemma RHS-tri-design-is-SRD [closure]:
assumes $ok' $ R $ Q $ ok' $ R
shows Rₘ(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 Rₘ(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]ₙ
shows (⊔ P ∈ A · R₁ (R2s (cmtₐ P))) = (⊔ P ∈ A · cmtₐ P)
by (rule USUP-cong[of A], metis (mono-tags, lifting) Ball-Collect R1-R2s-cmt-SRD assms)

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

4.4.1 Order laws

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

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

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

theorem RH-tri-design-composition:
assumes $\langle \mathsf{ok} \rangle \nmid P \mathsf{ok} \nmid Q_1 \mathsf{ok} \nmid Q_2 \mathsf{ok} \nmid R \mathsf{ok} \nmid S_1 \mathsf{ok} \nmid S_2$
shows $(\mathsf{RH}(P \implies Q_1 \circ Q_2) ; \mathsf{RH}(R \implies S_1 \circ S_2)) = \mathsf{RH}((\neg (R_1 (\neg R_2s P) ; R_1 \text{true}) \land \neg (\langle R_1 (R_2s Q_2) \land \neg \mathsf{wait}' \rangle ; R_1 (\neg R_2s R))) \implies ((Q_1 \lor (R_1 (R_2s Q_2) ; R_1 (R_2s S_1))) \circ ((R_1 (R_2s Q_2) ; R_1 (R_2s S_2))))
proof –
have 1: $(\neg (\langle R_1 (R_2s (Q_1 \circ Q_2) \land \neg \mathsf{wait}' \rangle ; R_1 (\neg R_2s R))) = (\neg (\langle R_1 (R_2s Q_2) \land \neg \mathsf{wait}' \rangle ; R_1 (\neg R_2s R)))$
by (metis (no-types, hide-lams) R1-extend-conj R2s-conj R2s-not R2s-wait' wait'-cond-false)

have 2: $(\langle R_1 (R_2s (Q_1 \circ Q_2)) ; ([I]_D \land \mathsf{wait} \lor R_1 (R_2s (S_1 \circ S_2))) = ((\langle R_1 (R_2s Q_1) \lor (R_1 (R_2s Q_2)) ; R_1 (R_2s S_1)) \circ (R_1 (R_2s Q_2) ; R_1 (R_2s S_2)))$
by (rel-auto)

also have ... = $\langle (R_1 (R_2s Q_1)) \land \mathsf{wait}' \rangle$
by (rel-auto)

also from assms(2) have ... = $\langle (R_1 (R_2s Q_1)) \land \mathsf{wait}' \rangle$
by (simp add: lift-des-skip-dr-unit-unrest unrest)

finally show ?thesis .

qed

moreover have $(\langle R_1 (R_2s Q_2) \land \neg \mathsf{wait} \land ([I]_D \land \mathsf{wait} \lor R_1 (R_2s S_1) \circ R_1 (R_2s S_2))) = ((\langle R_1 (R_2s Q_2) \land \neg \mathsf{wait} \land ([I]_D \land \mathsf{wait} \lor R_1 (R_2s S_1) \circ R_1 (R_2s S_2)))$
by (metis (no-types, lifting) cond-def conj-disj-not-abs utp-pred-laws.double-compl utp-pred-laws.inf.left-idem utp-pred-laws.sup-assoc utp-pred-laws.sup-inf-absorb)

also have ... = $\langle (R_1 (R_2s Q_2)) \land (R_1 (R_2s S_1)) \circ (R_1 (R_2s S_2)) \rangle$
by (metis false-alt-def seq-right-one-point upred-eq-false wait-vw-lens)

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

finally show ?thesis .

qed

moreover have $(\langle R_1 (R_2s Q_1) \land \mathsf{wait}' \lor (\langle R_1 (R_2s Q_2) \land (R_1 (R_2s S_1)) \circ (R_1 (R_2s S_2))))$
by (simp add: wait'-cond-def cond-seq-right-distr cond-and-T-integrate unrest)

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

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

qed

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

proof –
  have 1:\((\neg ((R1(R2s(Q1 \circ Q2)) \land \neg wait`) ;; R1(\neg R2s R)))) = \((\neg (R1(R2s Q2) \land \neg wait`) ;; R1(\neg R2s R))\)
    by (metis (no-types, hide-lams) R1-extend-conj R2s-conj R2s-not R2s-wait' wait'-cond-false)
  have 2:\((R1(R2s(Q1 \circ Q2)) ;; (\exists st \cdot [II]_D) \land \neg wait \lor (R1(R2s S1) ;; R1(R2s S2)))) = 
    ((\exists st' \cdot R1(R2s Q1)) \land (\exists st \cdot [II]_D))\)
    by (rel-auto, blast+)
  also have \(\exists st' \cdot (R1(R2s Q1)) \land (\exists st \cdot [II]_D)\)
    by (rel-auto)
  also from assms(2) have \((\exists st' \cdot (R1(R2s Q1)) \land (\exists st \cdot [II]_D))\)
    by (rel-auto, blast)
  finally show ?thesis .
qed

moreover have \((R1(R2s Q2)) ;; (\neg wait \land (\exists st \cdot [II]_D) \land \neg wait \lor (R1(R2s S1) \circ R1(R2s S2))))\) = 
  \((R1(R2s Q2)) ;; (R1(R2s S1) \circ R1(R2s S2)))\)
proof
have \((R1 \ R2s Q2) \land \sim \$\text{wait} \land \exists \, (!?st :: ['H'?]). \sim \$\text{wait} \lor R1 (R2s S1) \land R1 (R2s S2))\)
\(= (R1 (R2s Q2)) \land \sim \$\text{wait} \land (R1 (R2s S1)) \land R1 (R2s S2))\)
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 \(\ldots = ((R1 (R2s Q2)) \land \sim \$\text{wait} \lor R1 (R2s S1) \land R1 (R2s S2))\)
by (metis false-alt-def sqrr-right-one-point upred-eq-false wait-vb-lens)
also have \(\ldots = ((R1 (R2s Q2)) \land (R1 (R2s S1) \land R1 (R2s S2))\)
by (simp add: wait'-cond-def unsubst unrest assms)
finally show \(?thesis\).
qed

moreover
have \((R1 (R2s Q2) \land \sim \$\text{wait'}) \lor (R1 (R2s Q2)) \land (R1 (R2s S1) \land R1 (R2s S2))\)
\(= (R1 (R2s Q2) \land (R1 (R2s Q2)) \land (R1 (R2s S1)) \land (R1 (R2s Q2)) \land (R1 (R2s S2))\)
by (simp add: wait'-cond-def cond-seq-right-distr cond-and-T-integrate unrest)

ultimately show \(?thesis\).
by (simp add: R2s-wait'-cond R1-wait'-cond wait'-cond-seq ex-conj-contr-right unrest)
(simp add: cond-and-T-integrate cond-seq-right-distr unrest-var wait'-cond-def)

qed

from assms(7,8) have \(3: (R1 (R2s Q2) \land \sim \$\text{wait'}) \land (\sim \sim R2s R) = R1 (R2s Q2) \land R1 (\sim \sim R2s R)\)
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 unrest)
apply (simp add: 1 2 3)
apply (simp add: R1-R2s-R2c RHS-design-lemma1)
apply (metis R1-R2c-ex-st RHS-design-lemma1)
done

qed

theorem RHS-tri-design-composition-wp:
assumes \$\text{st'} \notin P \land \sim \$\text{wait'} \land Q1 \land \sim \$\text{ok} \land Q2 \land \sim \$\text{ok} \land S1 \land \sim \$\text{ok} \land S2
\$\text{wait} \land R \land \sim \$\text{wait} \land Q2 \land \sim \$\text{wait} \land S1 \land \sim \$\text{wait} \land S2
P is R2c Q1 is R1 Q1 is R2c Q2 is R1 Q2 is R2c
R is R2c S1 is R1 S1 is R2c S2 is R1 S2 is R2c
shows \(R_4(P \vdash Q_1 \land Q_2) :: R_4(R \vdash S_1 \land S_2) =\)
\(R_4(((\sim \sim P) wp_r \land Q2 wp_r R) \vdash (((\sim \sim \$\text{st'} \land Q_1) \land (Q2 \land S_1)) \land (Q2 \land S_2)))\) (is \(?lhs = ?rhs\))

proof
have \(?lhs = R_4(((\sim \sim R1 \sim \sim P) \land Q2 \land \sim \sim Q2 \land R1 \sim \sim R)) \vdash (((\sim \sim \$\text{st'} \land Q_1) \land (Q2 \land S_1)) \land (Q2 \land S_2)))\)
by (simp add: RHS-tri-design-composition assms Healthy-if R2c-healthy-R2c disj-upred-def)
(metis (no-types, hide-lams) R1-negate-R1 R2c-healthy-R2s assms(11,16))
also have \(?rhs\)
by (rel-auto)
finally show \( \text{thesis} \).

\[ \text{qed} \]

**Theorem:** \( \text{RHS-tri-design-composition-RR-wp} \)

**Assumptions:** \( P \) is RR \( Q_1 \) is RR \( Q_2 \) is RR
\( R \) is RR \( S_1 \) is RR \( S_2 \) is RR

**Shows:** \( R_4 (P \triangleright Q_1 \circ Q_2) \circ (Q_2 \circ S_2) \)

\[ \text{by (simp add: RHS-tri-design-composition-wp add: closure assms unrest RR-implies-R2c)} \]

**Lemma:** \( \text{RHS-tri-normal-design-composition} \)

**Assumptions:**
\( $\text{ok}^+ P \not\in Q_1 \not\in Q_2 \not\in R \not\in S_1 \not\in S_2 \)
\( $\text{wait} P \not\in Q_2 \not\in S_1 \not\in S_2 \)
\( P \) is RR \( Q_1 \) is RR \( Q_2 \) is RR \( R \) is RR \( S_1 \) is RR \( S_2 \) is RR
\( R_1 (\not\in P) \circ R_1 (\text{true}) = R_1 (\not\in P) \not\in Q_1 \)

**Shows:**
\[ R_4 (P \triangleright Q_1 \circ Q_2) \circ (Q_1 \circ S_2) \]

\[ \text{by (simp-all add: RHS-tri-design-composition-wp rea-not-def assms unrest)} \]

**Finally show:** \( \text{thesis} \).

\[ \text{qed} \]

**Lemma:** \( \text{RHS-tri-normal-design-composition'} \) [rdes-def]

**Assumptions:** \( P \) is RR \( Q_1 \) is RR \( Q_2 \) is RR \( R \) is RR \( S_1 \) is RR \( S_2 \) is RR

**Shows:**
\[ R_4 (P \triangleright Q_1 \circ Q_2) \circ (Q_1 \circ S_2) \]

\[ \text{by (simp add: Healthy-def RC1-def rea-not-def)} \]

\[ \text{(metis R1-negate-R1 R1-seqr utp-pred-laws.double-compl)} \]

**Thus:** \( \text{thesis} \)

**By (simp add: RHS-tri-normal-design-composition assms closure unrest RR-implies-R2c)\]**

\[ \text{qed} \]

**Lemma:** \( \text{RHS-tri-design-right-unit-lemma} \)

**Assumptions:**
\( $\text{ok}^+ P \not\in Q \not\in R \not\in S \)

**Shows:**
\[ R_4 (P \triangleright Q \circ R) \circ (Q \circ S) \]

\[ \text{by (simp add: srdes-skip-tri-design, rel-auto)} \]

**Proof:**

**From:** \( \text{assms(3,4)} \) \( R_1 (R_2S) \circ R_1 (R_2S (\text{str} = u \text{ str} \land [\text{rr}] R) = R_1 (R_2S R) \)

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

40
thus \(?\text{thesis}\)
by simp

qed

also have ... = R_s((\neg (\neg P) \land R1 \land true) \implies ((\exists st' \cdot Q) \land R))
by (metis (no-types, lifting) R1-R2s-R1-true-lemma R1-R2s-R2c R2c-not RHS-design-R2c-pre RHS-design-neg-R1-pre RHS-design-post-R1 RHS-design-post-R2s)

also have ... = R_s((\neg (\neg P) \land true_r) \implies ((\exists st' \cdot Q) \land R))
by (rel-auto)
finally show \(?\text{thesis}\)

qed

lemma \text{SRD-composition-up:}
assumes P is SRD Q is SRD
shows (P \land Q) = (R_s (\neg (\neg R) \land \neg \neg P) \land true \land \neg \neg Q \land \neg \neg \neg P \land \neg \neg \neg \neg Q) \implies 
((\exists st' \cdot \neg \neg R) \land \neg \neg \neg \neg P \land \neg \neg \neg \neg \neg Q) \implies (\exists st' \cdot \neg \neg R) \land \neg \neg \neg \neg P \land \neg \neg \neg \neg \neg Q)
(is \(?\text{lhs} = \ ?\text{rhs}\))

proof
have (P \land Q) = (R_s (\neg (\neg R) \land \neg \neg P) \land true \land \neg \neg Q \land \neg \neg \neg P \land \neg \neg \neg \neg Q) \implies (\exists st' \cdot \neg \neg R) \land \neg \neg \neg \neg P \land \neg \neg \neg \neg \neg Q)
by (simp add: SRD-reactive-tri-design assms(1) assms(2))
also from assms
have ... = \ ?\text{rhs}
by (simp add: RHS-tri-design-composition-up disj-upred-def unrest assms closure)
finally show \(?\text{thesis}\)

qed

4.6 Refinement introduction laws

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

proof

have \(?\text{lhs} \iff 'P_1 \implies Q_1' \land 'P_1 \land Q_2 \implies P_2' \land 'P_1 \land Q_3 \implies P_3'
by (simp add: RHS-design-refine assms closure RR-implies-R2c unrest ex-unrest)
also have ... \iff 'P_1 \implies Q_1' \land 'P_1 \land Q_2 \implies P_2', 'P_1 \land Q_3 \implies P_3'
by (rel-auto)
also have ... \iff 'P_1 \implies Q_1' \land 'P_1 \land Q_2 \implies P_2', 'P_1 \land Q_3 \implies P_3'
by (rel-auto, metis)
also have ... \iff \ ?\text{rhs}
by (simp add: usubst unrest assms)
finally show \(?\text{thesis}\)

qed

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

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

lemma SRD-peri-under-pre:
  assumes \( P \) is SRD \$\text{wait'} \$ \( \not\pre_R(P) \)
  shows \( \pre_R(P) \Rightarrow \peri_R(P) \) = \( \peri_R(P) \)
  proof
    have \( \peri_R(P) = \peri_R(R_* (\pre_R(P) \vdash \peri_R(P) \circ \post_R(P))) \)
      by (simp add: SRD-reactive-tri-design assms)
    also have \( \ldots = (\pre_R P \Rightarrow \peri_R P) \)
      by (simp add: rea-pre-RHS-design rea-peri-RHS-design assms)
    finally show \( \text{thesis} .. \)
  qed

lemma SRD-post-under-pre:
  assumes \( P \) is SRD \$\text{wait'} \$ \( \not\pre_R(P) \)
  shows \( \pre_R(P) \Rightarrow \post_R(P) \) = \( \post_R(P) \)
  proof
    have \( \post_R(P) = \post_R(R_* (\pre_R(P) \vdash \peri_R(P) \circ \post_R(P))) \)
      by (simp add: SRD-reactive-tri-design assms)
    also have \( \ldots = (\pre_R P \Rightarrow \post_R P) \)
      by (simp add: rea-pre-RHS-design rea-post-RHS-design assms)
    finally show \( \text{thesis} .. \)
  qed

lemma SRD-refine-intro:
  assumes \( P \) is SRD \( Q \) is SRD
  \( \pre_R(P) \Rightarrow \pre_R(Q) \) \( \peri_R(P) \land \peri_R(Q) \Rightarrow \peri_R(P) \land \peri_R(Q) \Rightarrow \peri_R(P) \land \post_R(Q) \Rightarrow \post_R(P) \)
  shows \( P \subseteq Q \)
  by (metis SRD-reactive-tri-design assms(1) assms(2) assms(3) assms(4) assms(5) srdes-tri-refine-intro)

lemma SRD-refine-intro':
  assumes \( P \) is SRD \( Q \) is SRD
  \( \pre_R(P) \Rightarrow \pre_R(Q) \) \( \peri_R(P) \subseteq (\pre_R(P) \land \peri_R(Q)) \) \( \post_R(P) \subseteq (\pre_R(P) \land \post_R(Q)) \)
  shows \( P \subseteq Q \)
  using assms by (rule-tac SRD-refine-intro, simp-all add: refBy-order)

lemma SRD-eq-intro:
  assumes \( P \) is SRD \( Q \) is SRD \( \pre_R(P) = \pre_R(Q) \) \( \peri_R(P) = \peri_R(Q) \) \( \post_R(P) = \post_R(Q) \)
  shows \( P = Q \)
  by (metis SRD-reactive-tri-design assms)

4.7 Closure laws

lemma SRD-srdes-skip [closure]: \( II_R \) 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_a (((¬ pre_a P) wp false ∧ post_a P wp pre_a Q) ⊢
    ((∃ $s' · peri_a P) ∨ post_a P ;; peri_a Q) ⊢ post_a P ;; post_a 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 =⇒ P ^ 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 =⇒ P ;; P ^ n is SRD
by (metis SRD-power-Suc upred-semiring power-Suc)

lemma uplus-SRD-closed [closure]: P is SRD =⇒ P + is SRD
by (simp add: uplus-power-def closure)

lemma SRD-Sup-closure [closure]:
assumes A ⊆ [SRD] A ≠ {}
shows (⨆ A) is SRD
proof –
  have SRD (∨ A) = (∨ (SRD 'A))
  by (simp add: ContinuousD SRD-Continuous assms(2))
also have ... = (⨆ 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_a(P_1 ⊢ P_2 ∨ P_3) ∩ R_a(Q_1 ⊢ Q_2 ∨ Q_3) = R_a((P_1 ∧ Q_1) ⊢ (P_2 ∨ Q_2) ∨ (P_3 ∨ Q_3))
apply (simp add: RHS-design-choice)
apply (rule cong[of R_a R_a])
apply (simp)
apply (rel-auto)
done

lemma RHS-tri-design-disj [rdes-def]:
  (R_a(P_1 ⊢ P_2 ∨ P_3) ∨ R_a(Q_1 ⊢ Q_2 ∨ Q_3)) = R_a((P_1 ∨ Q_1) ⊢ (P_2 ∨ Q_2) ∨ (P_3 ∨ Q_3))
by (simp add: RHS-tri-design-choice disj-upred-def)

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

43
\[
R_s(P_1 \vdash P_2 \land P_3) \cup R_s(Q_1 \vdash Q_2 \land Q_3) = R_s((P_1 \lor Q_1) \vdash ((P_1 \Rightarrow P_2) \land (Q_1 \Rightarrow Q_3)) \circ ((P_1 \Rightarrow R_3) \land (Q_1 \Rightarrow R_3)))
\]
by (simp add: RHS-design-sup, rel-auto)

**Lemma** \( \text{RHS-tri-design-conj} [rdes-def]: \)
\[
(R_s(P_1 \vdash P_2 \land P_3) \land R_s(Q_1 \vdash Q_2 \land Q_3)) = R_s((P_1 \lor Q_1) \vdash ((P_1 \Rightarrow P_2) \land (Q_1 \Rightarrow Q_3)) \circ ((P_1 \Rightarrow R_3) \land (Q_1 \Rightarrow R_3)))
\]
by (simp add: RHS-tri-design-sup conj-uppred-def)

**Lemma** \( \text{SRD-UINF} [rdes-def]: \)
\[
\text{assumes } A \neq \{ \} \land A \subseteq [\text{SRD}]_l
\]
\[
\text{shows } \prod A = R_s(\prod P \in A \cdot pre_R(P)) \vdash (( \prod P \in A \cdot peri_R(P)) \circ ( \prod P \in A \cdot post_R(P)))
\]
proof
- have \( \prod A = R_s(\prod P \in A \cdot peri_R(P) \circ \prod P \in A \) by (metis SRD-as-reactive-tri-design assms srdes-hcond-def srdes-theory-continuous,healthy-inf srdes-theory-continuous,healthy-inf-def)
also have \( \ldots = R_s(\prod P \in A \cdot peri_R(P) \circ ( \prod P \in A \cdot post_R(P))) \)
finally show \( \text{thesis} \)
qed

**Lemma** \( \text{RHS-tri-design-USUP} [rdes-def]: \)
\[
\text{assumes } A \neq \{ \} \land A \subseteq \text{SRD}
\]
\[
\text{shows } (\prod i \in A \cdot R_s(P(i) \vdash Q(i)) \circ R(i)) = R_s((\prod i \in A \cdot P(i)) \vdash (\prod i \in A \cdot Q(i)) \circ (\prod i \in A \cdot R(i)))
\]
by (subst RHS-linf[of assms, THEN sym], simp add: design-UINF-mem assms, rel-auto)

**Lemma** \( \text{SRD-UINF-mem}: \)
\[
\text{assumes } A \neq \{ \} \land A \subseteq \text{SRD}
\]
\[
\text{shows } (\prod i \in A \cdot P i) = R_s((\prod i \in A \cdot pre_R(P i)) \vdash (\prod i \in A \cdot peri_R(P i)) \circ (\prod i \in A \cdot post_R(P i)))
\]
(is \( \text{lhs} = \text{rhs} \)
proof
- have \( \text{lhs} = (\prod (P \cdot A)) \)
by (rel-auto)
also have \( \ldots = R_s(\prod P \in A \cdot pre_R(P) \circ Pa) \)
also have \( \ldots \)
by (subst rdes-def, simp-all add: assms image-subsetI)
also have \( \ldots = \text{thesis} \)
qed

**Lemma** \( \text{RHS-tri-design-UINF-ind} [rdes-def]: \)
\[
(\prod i \cdot R_s(P(i) \vdash P_2(i) \circ P_3(i))) = R_s((\prod i \cdot P_1(i) \vdash (\prod i \cdot P_2(i)) \circ (\prod i \cdot P_3(i)))
\]
by (rel-auto)

**Lemma** \( \text{cond-srea-form} [rdes-def]: \)
\[
R_s(P \vdash Q_1 \circ Q_2) \circ b \triangleright_R R_s(R \vdash S_1 \circ S_2) = R_s(P \circ Q_1 \circ Q_2) \circ R2c([b]_{S <}) \circ R_s(R \vdash S_1 \circ S_2)
\]
proof
- have \( R_s(P \vdash Q_1 \circ Q_2) \circ b \triangleright_R R_s(R \vdash S_1 \circ S_2) = R_s(P \vdash Q_1 \circ Q_2) \circ R2c([b]_{S <}) \circ R_s(R \vdash S_1 \circ S_2) \)
by (pred-auto)
also have \( \ldots \)
by (simp add: RHS-trie lift-cond-srea-def)
also have \( \ldots = R_s((P \triangleright R) \vdash (Q_1 \triangleright S_1) \triangleright (Q_2 \triangleright S_2)) \)
by (simp add: condr design-condr lift-cond-srea-def)
also have \( \ldots = R_s((P \triangleright R) \vdash (Q_1 \triangleright S_1) \triangleright (Q_2 \triangleright S_2)) \)
by (rule cong[of \( R_s \), simp, rel-auto)
finally show \?thesis.
qed

lemma SRD-cond-srea [closure]:
assumes \( P \) is SRD \( Q \) is SRD
shows \( P \triangleright b \triangleright R \triangleright Q \) is SRD
proof -
have \( P \triangleright b \triangleright R \triangleright Q = R_s((\pre R(P) \triangleright \peri R(P) \circ \post R(P)) \triangleright b \triangleright R \triangleright R_s((\pre R(Q) \triangleright \peri R(Q) \circ \post R(Q))) \)
by (simp add: SRD-reactive-tri-design assms)
also have \( \ldots = R_s((\pre R P \triangleright b \triangleright \peri R(Q)) \circ \post R(P) (P \triangleright b \triangleright R \triangleright Q)) \)
by (simp add: cond-srea-form)
also have \( \ldots \) 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 \( II \triangleright R P = P \)
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]:
\( II \triangleright R II = II \)
by (simp add: SRD-left-unit closure)

lemma SRD-right-unit-tri-lemma:
assumes \( P \) is SRD
shows \( P \triangleright b \triangleright R \triangleright P = R_s((\false \triangleright \true) \circ \pre R(P) (P \triangleright b \triangleright R \triangleright P) \circ \peri R(P) \circ \post R(P)) \)
by (simp add: SRD-composition-wp closure rdes wp rpred trace-ident-right-pos tR assms)

lemma Miracle-left-zero:
assumes \( P \) is SRD
shows \( Miracle \triangleright P = Miracle \)
proof -
have \( Miracle \triangleright P = R_s((\false \triangleright \true) \circ \peri R(P) \circ \post R(P)) \)
by (simp add: Miracle-def SRD-reactive-design-alt assms)
also have \( \ldots = R_s((\true \triangleright \false) \circ \peri R(P) \circ \post R(P)) \)
by (simp add: RHS-design-composition unrest R1-false R2s-false R2s-true)
also have \( \ldots = Miracle \)
by (simp add: Miracle-def)
finally show \?thesis.
qed

lemma Chaos-left-zero:
assumes \( P \) is SRD
shows \( (Chaos \triangleright P) = Chaos \)
proof -
have \( Chaos \triangleright P = R_s((\false \triangleright \true) \circ \peri R(P) \circ \post R(P)) \)
Stateful reactive designs are left unital

4.10 Recursion laws

lemma mono-srd-iter:
  assumes mono $F \in [\text{SRD}]_H \to [\text{SRD}]_H$
  shows $\lambda X. R_s(\text{pre}_R(F X) \vdash \text{peri}_R(F X) \circ \text{post}_R(F X))$
  apply (rule mono)
  apply (rule srdes-unit-refine-intro)
  apply (meson assms(1) monoE preR-antitone utp-pred-laws.le-infI2)
  apply (meson assms(1) monoE periR-monotone utp-pred-laws.le-infI2)
  apply (meson assms(1) monoE postR-monotone utp-pred-laws.le-infI2)
  done

lemma mu-srd-SRD:
  assumes mono $F \in [\text{SRD}]_H \to [\text{SRD}]_H$
  shows $(\mu X \cdot R_s(\text{pre}_R(F X) \vdash \text{peri}_R(F X) \circ \text{post}_R(F X)))$ 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)

done

**Lemma mu-srd-iter:**

**Assumes** \( \mu X \cdot \mathbf{R}_s(\text{pre}_R(F(X)) \vdash \text{peri}_R(F(X)) \circ \text{post}_R(F(X))) = F(\mu X \cdot \mathbf{R}_s(\text{pre}_R(F(X)) \vdash \text{peri}_R(F(X)) \circ \text{post}_R(F(X))) \)

**Shows** \( \mu X \cdot \mathbf{R}_s(\text{pre}_R(F(X)) \vdash \text{peri}_R(F(X)) \circ \text{post}_R(F(X))) \)

**Proof** –

**Have 1:** \( F(\mu X \cdot \mathbf{R}_s(\text{pre}_R(F(X)) \vdash \text{peri}_R(F(X)) \circ \text{post}_R(F(X))) \) is SRD

**By** (simp add: Healthy-apply-closed assms 1 assms 2 mu-srd-SRD)

**Have 2:** Monotonic \( \text{SRDES} \) \( F \)

**By** (simp add: assms 1 mono-Monotone-utp-order)

**Hence 3:** \( \mu X \cdot \mathbf{R}_s(\text{pre}_R(F(X)) \vdash \text{peri}_R(F(X)) \circ \text{post}_R(F(X))) \subseteq F(\mu R F) \)

**Using** SRD-reactive-tri-design by force

**Thus** \( \mu X \cdot \mathbf{R}_s(\text{pre}_R(F(X)) \vdash \text{peri}_R(F(X)) \circ \text{post}_R(F(X))) \)

**By** (simp add: 2 srdes-theory-continuous.weak LFP-lemma3 gfp-upperbound assms)

**QED**

**Lemma Monotonic-SRD-comp [closure]:** Monotonic \( \text{op} ;; \text{P} \circ \text{SRD} \)

**By** (simp add: mono-def R1-R2c-is-R2 R2-mono R3h-mono RD1-mono RD2-mono RHS-def SRD-def seqr-mono)

end

5 Normal Reactive Designs

**Theory utp-rdes-normal**

**Imports**

\( \text{utp-rdes-triples} \)

\( \text{UTP-KAT.utp-kleene} \)

**Begin**

This additional healthiness condition is analogous to H3

**Definition RD3 where**

\( \text{upred-defs}: \text{RD3}(P) = P ;; \Pi_R \)

**Lemma RD3-idem:** \( \text{RD3}(\text{RD3}(P)) = \text{RD3}(P) \)

**Proof** –

**Have a:** \( \Pi_R ;; \Pi_R = \Pi_R \)

**By** (simp add: SRD-left-unit SRD-srdes-skip)

**Show** \( \text{thesis} \)

**By** (simp add: RD3-def seqr-assoc a)

47
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' \not\in peri_R(P)
  shows P is RD3
proof –
  have RD3(P) = R_a ((\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_a ((\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_a ((\neg_r pre_R(P)) wp_r false \vdash cmt_R P)
    by (simp add: wait'-cond-peri-post-cmt)
  also have ... = R_a (pre_R P \vdash cmt_R P)
    by (simp add: assms(2) rpred wp-rea-def R1-preR)
  finally show ?thesis
    by (metis Healthy-def SRD-as-reactive-design assms(1))
qed

lemma RHS-tri-design-right-unit-lema:
  assumes $ok' \not\in P \not\in Q$ $ok' \not\in R$ $\exists $wait' \not\in R
  shows R_a(P \vdash Q \circ R) :: II_R = R_a((\neg_r (\neg_r P)) :: true_r) \vdash ((\exists $st' \cdot Q) \circ R)
proof –
  have R_a(P \vdash Q \circ R) :: II_R = R_a(P \vdash Q \circ R) :: R_a(true \vdash false \circ ($tr' = u \ tr \land [II]_R))
    by (simp add: srdex-skip-tri-design, rel-auto)
  also have ... = R_a ((\neg R1 (\neg 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 ... = Rₙ ((¬ R₁ (¬ R₂ s P) ;; R₁ true) ⊨ (∃ $st' ⋄ Q) ⋄ R₁ (R₂ s R))

proof
  from assms(3,4) have (R₁ (R₂ s R) ;; R₁ (R₂ s ($tr' = u $tr ∧ [I]ₙ))) = R₁ (R₂ s R)
  by (rel-auto, metis (no-types, lifting) minus-zero-eq, meson order-refl trace-class.diff-cancel)
thus ?thesis
  by simp
qed

also have ... = Rₙ((¬ 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-R₁-pre
RHS-design-post-R₁ RHS-design-post-R₂s)

finally show ?thesis .

qed

lemma RHS-tri-design-RD3-intro:
  assumes $ok' ∤ P $ok' ⋄ Q $ok' ⋄ R $st' ∤ Q $wait' ⋄ R
  shows Rₙ(P ⊨ Q ⋄ R) is RD₃
  apply (simp add: Healthy-def RD₃-def)
  apply (subst RHS-tri-design-right-unit-lemma)
  apply (simp-all add: assms unrest closure rpred)
done

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

lemma R₁-right-unit-lemma:
  [ outa $b; outa $e ] ⊨ (¬$r $b ∨ $tr $e ≤₀ $tr') ;; R₁(true) = (¬$r $b ∨ $tr $e ≤₀ $tr')
  by (rel-auto, blast, metis (no-types, lifting) dual-order.trans)

lemma RHS-tri-design-RD3-intro-form:
  assumes outa $b outa $e $ok' ∤ Q $st' ∤ Q $ok' ⋄ R $wait' ⋄ R
  shows Rₙ((b ∧ ¬$r $st $e ≤₀ $tr') ⊨ Q ⋄ R) is RD₃
  apply (rule RHS-tri-design-RD₃-intro)
  apply (simp-all add: assms unrest closure rpred)
  apply (subt R₁-right-unit-lemma)
  apply (simp-all add: assms unrest)
done

definition NSRD :: ('s, 't::trace, 'α) hrel-rsp ⇒ ('s, 't, 'α) hrel-rsp
  where [upred-defs]: NSRD = R₁ ⋄ R₃ ⋄ RHS

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

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

lemma NSRD-elim [RD-elim]:
  [ P is NSRD; Q(Rₙ(periₙ(P) ⊨ periₙ(P) ⋄ postₙ(P))) ] ⇒ Q(P)
by (simp add: RD-alt closure)

lemma NSRD-Idempotent [closure]: Idempotent NSRD

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

lemma NSRD-form: 
NSRD(P) = R_s((¬_r (¬_r pre_R(P)) ;; R1 true) ⊨ ((∃ $st' · peri_R(P)) ∘ post_R(P)))
proof –
  have NSRD(P) = RD3(SRD(P))
    by (metis (no-types, lifting) NSRD-def RD1-RD3-commute RD3-left-subsumes-RD2 SRD-def comp-def)
  also have ... = RD3(R_s(pre_R(P) ⊨ peri_R(P) ∘ post_R(P)))
    by (simp add: SRD-as-reactive-tri-design)
  also have ... = R_s(R_s(pre_R(P) ⊨ peri_R(P) ∘ post_R(P)) ;; H_R)
    by (simp add: RD3-def)
  also have ... = R_s(((¬_r (¬_r pre_R(P)) ;; R1 true) ⊨ ((∃ $st' · peri_R(P)) ∘ post_R(P)))
    by (simp add: RHS-tri-design-right-unit-lemma unrest)
finally show ?thesis.

qed

lemma NSRD-healthy-form:
assumes P is NSRD
shows R_s((¬_r (¬_r pre_R(P)) ;; R1 true) ⊨ ((∃ $st' · peri_R(P)) ∘ post_R(P))) = P
by (metis Healthy-def NSRD-form assms)

lemma NSRD-Sup-closure [closure]:
assumes A ⊆ [NSRD]_H A ≠ {} 
shows ∩ A is NSRD
proof –
  have NSRD (∩ A) = (∩ (NSRD 'A))
    by (simp add: ContinuousD NSRD-Continuous assms(2))
  also have ... = (∩ A)
    by (simp only: Healthy-carrier-image assms)
finally show ?thesis by (simp add: Healthy-def)

qed

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

lemma NRSD-SUP-closure [closure]: 
[ ∩ i. i ∈ A ⇒ P(i) is NSRD; A ≠ {} ] ⇒ (∩i∈A. P(i)) is NSRD
by (rule NSRD-Sup-closure, auto)

lemma NSRD-neg-pre-unit:
assumes P is NSRD
shows (¬_r pre_R(P)) ;; true = (¬_r pre_R(P))
proof –
  have (¬_r pre_R(P)) = (¬_r pre_R(R_s((¬_r (¬_r pre_R(P)) ;; R1 true) ⊨ ((∃ $st' · peri_R(P)) ∘ post_R(P))))))
    by (simp add: NSRD-healthy-form assms)
  also have ... = R1 (R2c ((¬_r pre_R P) ;; R1 true))
by (simp add: rea-pre-RHS-design R1-negate-R1 R1-idem R1-rea-not R2c-rea-not usubst rpred unrest closure)
also have ... = (¬ 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 (¬ (¬ r pre R P))) :: Q = (¬ r pre R (¬ (¬ r pre R P)))
  by (metis (no-types, hide-lams) NSRD-neg-pre-unit RD1-left-zero assms)

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

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

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

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

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

lemma NSRD-intro:
assumes \( P \) is SRD \( \neg \) pre\(_R\)(\( P \)) \( \vdash \) true
shows \( P \) is NSRD
proof
have \( \text{NSRD}(P) = R_s((\neg_r \neg_r \text{pre}_R(P)) \vdash ((\exists st' \cdot \text{peri}_R(P)) \circ \text{post}_R(P))) \)
  by (simp add: NSRD-form)
also have ... = \( R_s(\text{pre}_R P \vdash \text{peri}_R P \circ \text{post}_R P) \)
  by (simp add: assms ex-unrest rpred closure)
also have ...
  by (simp add: SRD-reactive-tri-design assms (1))
finally show ?thesis
  using Healthy-def by blast
qed

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

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

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

lemma SRD-RD3-implies-NSRD:
\[ \[ P \text{ is SRD}; P \text{ is RD3 } \] \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:
\( P \) is NSRD \iff \((P \text{ is SRD}) \land (\neg_r \text{pre}_R(P)) \vdash R1(\text{true}) = (\neg_r \text{pre}_R(P)) \land (st' \cdot \text{peri}_R(P))\)
by (meson NSRD-intro NSRD-is-SRD NSRD-neg-pre-unit NSRD-st'-unrest-peri)

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

lemma NSRD-refine-elim:
assumes

52
\[ P \subseteq Q \text{ and } P \text{ is NSRD, } Q \text{ is NSRD} \]
\[
\begin{align*}
\text{[/ pre}_R(P) \Rightarrow \text{pre}_R(Q) \text{; } \text{pre}_R(P) \wedge \text{peri}_R(Q) \Rightarrow \text{peri}_R(P) \text{; } \text{pre}_R(P) \wedge \text{post}_R(Q) \Rightarrow \text{post}_R(P) \}
\end{align*}
\]
\[ \Rightarrow R \]
\[ \text{shows } R \]
\[ \text{proof} - \]
\[ \text{have } R_s(\text{pre}_R(P) \Rightarrow \text{peri}_R(P) \circ \text{post}_R(P)) \subseteq R_s(\text{pre}_R(Q) \Rightarrow \text{peri}_R(Q) \circ \text{post}_R(Q)) \]
\[ \text{by (simp add: NSRD-is-SRD SRD-reactive-tri-design assms(1) assms(2) assms(3))} \]
\[ \text{hence } 1: \text{pre}_R P \Rightarrow \text{pre}_R Q \text{ and } 2: \text{pre}_R P \wedge \text{peri}_R Q \Rightarrow \text{peri}_R P \text{ and } 3: \text{pre}_R P \wedge \text{post}_R Q \Rightarrow \text{post}_R P \]
\[ \text{by (simp-all add: RHS-tri-design-refine assms closure)} \]
\[ \text{with assms(4) show } \Box \text{thesis} \]
\[ \text{by simp} \]
\[ \text{qed} \]

\text{lemma NSRD-right-unit: } P \text{ is NSRD } \Rightarrow P :: H_R = P \]
\[ \text{by (metis Healthy-if NSRD-is-RD3 RD3-def)} \]

\text{lemma NSRD-composition-wp:}
\[ \text{assumes } P \text{ is NSRD } Q \text{ is SRD} \]
\[ \text{shows } P :: Q = \]
\[ R_s((\text{pre}_R P \wedge \text{post}_R P \text{ wp}) \circ \text{pre}_R Q) \Rightarrow (\text{pre}_R P \vee (\text{post}_R P :: \text{peri}_R Q)) \circ (\text{post}_R P :: \text{post}_R Q) \]
\[ \text{by (simp add: SRD-composition-wp assms NSRD-is-SRD wp-rea-def NSRD-neg-pre-unit NSRD-st'-unrest-peri R1-negate-R1 R1-rea-not R1-rea-R2c-closed R1-rea-R2c-post R1-rea-R2c-rea-not assms(1) assms(2))} \]
\[ \text{also have } \ldots = \text{pre}_R R2c(R2c \circ (\text{pre}_R P :: (\neg_\gamma \text{ peri}_R Q))) \]
\[ \text{by (simp add: R1-seqr R2c-R1-seq calculation)} \]
\[ \text{finally show } \Box \text{thesis} \]
\[ \text{..} \]
\[ \text{qed} \]

\text{lemma preR-NSRD-seq-lemma:}
\[ \text{assumes } P \text{ is NSRD } Q \text{ is SRD} \]
\[ \text{shows } R1(R2c(\text{post}_R P :: (\neg_\gamma \text{ peri}_R Q))) = \text{post}_R P :: (\neg_\gamma \text{ peri}_R Q) \]
\[ \text{proof} - \]
\[ \text{have } \text{post}_R P :: (\neg_\gamma \text{ peri}_R Q) = R1(R2c(\text{post}_R P)) :: R1(R2c(\neg_\gamma \text{ peri}_R Q)) \]
\[ \text{by (simp add: NSRD-is-SRD R1-R2c-post-RHS R1-rea-not R2c-preR R2c-rea-not assms(1) assms(2))} \]
\[ \text{also have } \ldots = R1(R2c(\text{post}_R P :: (\neg_\gamma \text{ peri}_R Q))) \]
\[ \text{by (simp add: R1-seqr R2c-R1-seq calculation)} \]
\[ \text{finally show } \Box \text{thesis} \]
\[ \text{..} \]
\[ \text{qed} \]

\text{lemma preR-NSRD-seq [rdes]:}
\[ \text{assumes } P \text{ is NSRD } Q \text{ is SRD} \]
\[ \text{shows } \text{pre}_R(P :: Q) = (\text{pre}_R P \wedge \text{post}_R P \text{ wp}) \circ \text{pre}_R Q) \]
\[ \text{by (simp add: NSRD-composition-wp assms rea-pre-RHS-design usubst unrest wp-rea-def R2c-disj R1-disj R2c-and R2c-preR R1-R2c-commute[THEN sym] R1-extend-conj' R1-idem R2c-not closure)} \]
\[ \text{(metis (no-types, lifting) Healthy-def Healthy-if NSRD-is-SRD R1-R2c-commute R1-R2c-seqr-distribute R1-seqr-closure assms(1) assms(2) post-R-R2c-closed post-RSRD-R1 preR-R2c-closed rea-not-R1 rea-not-R2c)} \]

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

\text{lemma postR-NSRD-seq [rdes]:}
\[ \text{assumes } P \text{ is NSRD } Q \text{ is NSRD} \]
\[ \text{shows } \text{post}_R(P :: Q) = ((\text{pre}_R P \wedge \text{post}_R P \text{ wp}) \circ \text{pre}_R Q) \Rightarrow (\text{post}_R P :: \text{post}_R Q)) \]
by (simp add: NSRD-composition-wp assms closure rea-post-RHS-design usbst unrest wp-rea-def
R1-extend-conj' R1-disj R1-R2c-seqp-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 closure sqrr-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: sqrr-or-distl NSRD-neg-pre-unit assms closure rdss unrest)
qed

lemma RHS-tri-normal-design-composition':
assumes $ok' $ P $ok' $ Q1 $ok' $ Q2 $ok' $ R $ok' $ S1 $ok' $ S2
$wait' $ R $wait' $ Q2 $wait' $ S1 $wait' $ S2
P is R2c Q1 is R1 Q1 is R2c Q2 is R1 Q2 is R2c
R is R2c S1 is R1 S1 is R2c S2 is R1 S2 is R2c
R1 (~ P) ;; R1(true) = R1 (~ P) $st' $ Q1
shows R_s((P ⊢ Q1 ⊓ Q2) ;; R_s(R ⊢ S1 ⊓ S2)
  = R_s((P ⊓ Q2 wp_r, R) ⊢ (Q1 ∨ (Q2 ;; S1)) ⊓ (Q2 ;; S2))
proof –
  have R_s((P ⊢ Q1 ⊓ Q2) ;; R_s(R ⊢ S1 ⊓ S2) =
    R_s((R1 (~ P) wp_r false ∧ Q2 wp_r R) ⊢ (∃ $st' · Q1) ⊓ (Q2 ;; S1)) ⊓ (Q2 ;; S2)
    by (simp-all add: RHS-tri-design-composition-wp rea-not-def assms unrest)
also have ... = R_s((P ⊓ Q2 wp_r, R) ⊢ (Q1 ∨ (Q2 ;; S1)) ⊓ (Q2 ;; S2))
    by (simp add: assms wp-rea-def ex-unrest, rel-auto)
finally show ?thesis ·
qed

lemma RHS-tri-normal-design-composition' [rdss-def]:
assumes P is RC Q1 is RR $st' Q1 Q2 is RR R is RR S1 is RR S2 is RR
shows R_r((P ⊢ Q1 ⊓ Q2) ;; R_r(R ⊢ S1 ⊓ S2)
  = R_r((P ⊓ Q2 wp_r, R) ⊢ (Q1 ∨ (Q2 ;; S1)) ⊓ (Q2 ;; S2))
proof –
  have R1 (~ P) ;; R1 true = R1 (~ P)
    using RC-implies-RC1[OF assms(1)]
    by (simp add: Healthy-def RC1-def rea-not-def)
thus ?thesis
    by (simp add: RHS-tri-normal-design-composition assms closure unrest RR-implies-R2c)
qed

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

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

54
lemma \textit{NSRD-srd-skip} [closure]: \(RI_R \text{ is NSRD}\)
by (rule NSRD-intro, simp-all add: rdes closure unrest)

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

lemma \textit{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 \textit{NSRD-right-Miracle-tri-lemma}:
assumes \(P \text{ is NSRD}\)
shows \(P \land \text{Miracle} = R_s (pre_R P \vdash peri_R P \diamond false)\)
by (simp add: NSRD-composition-wp closure assms rdes wp rpred)

The set of non-terminating behaviours is a subset

lemma \textit{NSRD-right-Miracle-refines}:
assumes \(P \text{ is NSRD}\)
shows \(P \sqsubseteq P \land \text{Miracle}\)
proof –
  have \(R_s (pre_R P \vdash peri_R P \diamond post_R P) \sqsubseteq R_s (pre_R P \vdash peri_R P \diamond false)\)
  by (rule srdes-tri-refine-intro, rel-auto+)
  thus \(?\text{thesis}\)
  by (simp add: NSRD-elim NSRD-right-Miracle-tri-lemma assms)
qed

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

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

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

lemma \textit{preR-power}:
assumes \(P \text{ is NSRD}\)
shows \(pre_R (P \uplus P^\uplus n) = (\bigsqcup_{i \in \{0..n\}}. (post_R(P) \uplus i) \ wp_r (pre_R(P)))\)
proof (induct n)
case 0
  then show \(?\text{case}\)
  by (simp add: wp closure)
next
case (Suc n) note \(\text{hyp = this}\)
have $\text{pre}_R (P \cdot (\text{Suc } n + 1)) = \text{pre}_R (P ;; P \cdot (n+1))$
    by (simp add: upred-semiring-power-Suc)
also have ... = ($\text{pre}_R P \land \text{post}_R P \cdot \text{wp}_r \cdot \text{pre}_R (P \cdot (\text{Suc } n)))$
    using NSRD-iff assms preR-NSRD-seq power-Suc-NSRD-closed by fastforce
also have ... = ($\text{pre}_R P \land \text{post}_R P \cdot \text{wp}_r (\bigcup i \in \{0..n\}. \text{post}_R P \cdot i \cdot \text{wp}_r \cdot \text{pre}_R P))$
    by (simp add: hyp upred-semiring-power-Suc)
also have ... = ($\text{pre}_R P \land (\bigcup i \in \{0..n\}. \text{post}_R P \cdot \text{wp}_r (\text{post}_R P \cdot i \cdot \text{wp}_r \cdot \text{pre}_R P)))$
    by (simp add: wp)
also have ... = ($\text{pre}_R P \land (\bigcup i \in \{0..n\}. (\text{post}_R P \cdot (i+1) \cdot \text{wp}_r \cdot \text{pre}_R P))$
    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 $\text{thesis}$
      by (simp add: wp-rea-def upred-semiring-power-Suc seqr-associative rpred closure assms)
qed
also have ... = ($\text{post}_R P \cdot 0 \cdot \text{wp}_r \cdot \text{pre}_R P \land (\bigcup i \in \{0..n\}. (\text{post}_R P \cdot (i+1) \cdot \text{wp}_r \cdot \text{pre}_R P))$
    by (simp add: wp assms closure)
also have ... = ($\text{post}_R P \cdot 0 \cdot \text{wp}_r \cdot \text{pre}_R P \land (\bigcup i \in \{1..\text{Suc } n\}. (\text{post}_R P \cdot i \cdot \text{wp}_r \cdot \text{pre}_R P))$
    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 $\text{thesis}$ by simp
    qed
also have ... = ($\bigcup i \in \text{insert } 0 \{1..\text{Suc } n\}. (\text{post}_R P \cdot i \cdot \text{wp}_r \cdot \text{pre}_R P)$
    by (simp add: conj-upred-def)
also have ... = ($\bigcup i \in \{0..\text{Suc } n\}. \text{post}_R P \cdot i \cdot \text{wp}_r \cdot \text{pre}_R P$
    by (simp add: atLeast0-atMost-Suc-eq-insert-0)
finally show $?\text{thesis}$ by (simp add: upred-semiring-power-Suc)
qed

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

lemma perir-power' [rdes]:

57
assumes $P$ is NSRD
shows $peri_R(P; P^* n) = (pre_R(P^*(Suc n)) \Rightarrow_r (\prod i \in \{0..n\} \cdot post_R(P)^* i) ;; peri_R(P))$
by (simp add: periR-power assms UNF-as-Sup[THEN sym])

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

lemma $postR$-power $[rdes]$:
assumes $P$ is NSRD
shows $post_R(P; P^* n) = (pre_R(P^*(Suc n)) \Rightarrow_r post_R(P)^* Suc n)$
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 $post_R(P^*(Suc n + 1)) = post_R(P; P^*(n+1))$
  by (simp)
also have $... = (pre_R(P^*(Suc n + 1)) \Rightarrow_r (post_R P; post_R (P; P^* n)))$
  by (simp add: closure assms $rdes$
also have $... = (pre_R(P^*(Suc n + 1)) \Rightarrow_r (post_R P; (pre_R (P^* Suc n) \Rightarrow_r post_R P^* Suc n)))$
  by (simp only: $hyp$
also have $... = (pre_R P \Rightarrow_r (post_R P wp_r pre_R (P^* Suc n) \Rightarrow_r post_R P ;; (pre_R (P^* Suc n) \Rightarrow_r post_R P^* Suc n)))$
  by (simp add: rdes closure assms, pred-auto)
also have $... = (pre_R P wp_r pre_R (P^* Suc n) \Rightarrow_r post_R P^* Suc (Suc n))$
  by (metis (no-types, lifting) Healthy-if NSRD-is-SRD NSRD-power-Suc $R1$-power assms $hyp$ $postR$-SRD-$R1$
upred-semiring.power-Suc $wp$-rea-impl-lemma)
also have $... = (pre_R (P^*(Suc (Suc n))) \Rightarrow_r post_R P^* Suc (Suc n))$
  by (simp add: $rdes$ closure assms)
finally show $?case$ by (simp)

qed

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

lemma $power$-$rdes$-def $[rdes-def]$:
assumes $P$ is $RC$ $Q$ is $RR$ $R$ is $RR$ $\$st' $\not=$ $Q$
shows $(R,(P \mapsto Q \circ R))^* (Suc n) = R,(\underline{\bigcap i \in \{0..n\} \cdot (R^* i)} \mapsto (\prod i \in \{0..n\} \cdot R^* i) ;; Q) \circ (R^* Suc n))$
proof (induct $n$
  case 0
  then show $?case$
    by (simp add: $wp$ assms closure)

next
case $(Suc n)$
have 1: \((P \land (\bigsqcup i \in \{0..n\} \cdot R \wp_r (R \setminus i \wp_r \cdot P))) = (\bigsqcup i \in \{0..Suc n\} \cdot R \setminus i \wp_r \cdot P)\)
(is \(lhs = rhs\))

proof -
  have \(lhs = (P \land (\bigsqcup i \in \{0..n\} \cdot (R \setminus Suc i \wp_r \cdot P)))\)
    by (simp add: wp closure assms)
  also have \(\cdots\) (some complex proof steps not shown)
  finally show \(\cdots\) (some complex proof steps not shown)
qed

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

proof -
  have \(lhs = (Q \lor (\prod i \in \{0..n\} \cdot R \setminus Suc i) ;; Q)\)
    by (simp add: seqr-assoc THEN sym seq-UNF-distr)
  also have \(\cdots\) (some complex proof steps not shown)
  finally show \(\cdots\) (some complex proof steps not shown)
qed

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

proof -
  have \((\prod i \in \{0..n\} \cdot R \setminus i) ;; Q = (\prod i \in \{0..n\} \cdot R \setminus i) ;; Q\)
    by (simp add: UINF-as-Sup-collect)
  also have \(\cdots\) (some complex proof steps not shown)
  finally show \(\cdots\) (some complex proof steps not shown)
qed

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

**theorem** uplus-rdes-def [rdes-def]:
**assumes** P is RC Q is RR R is RR $st' \not\in Q$
**shows** ($R_s(P \vdash Q \circ R) \uplus = R_s(R^{*\uplus} \uplus, P \vdash R^{*\uplus}; Q \circ R)$)
**proof**
  1. $1: (i \cdot \emptyset i) ;; Q = R^{*\uplus} ;; Q$
  2. by (metis (no-types) RA1 assms rea-unit-uplus seq-unit-uplus)
**show** ?thesis
  1. by (simp add: uplus-power-def seq-UINF-distr wp closure assms rdes-def)
**qed**

### 5. UTP theory

**typedef** NSRDES

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

**overloading**
- **nsrdes-hcond** = uhp-hcond :: (NSRDES, (s,t::trace,α) rsp) uthy ⇒ ((s,t,α) rsp × (s,t,α) rsp)

**health**
- **nsrdes-unit** = uhp-unit :: (NSRDES, (s,t::trace,α) rsp) uthy ⇒ (s, t, α) hrel-rsp

**definition**
- **nsrdes-hcond** :: (NSRDES, (s,t::trace,α) rsp) uthy ⇒ ((s,t,α) rsp × (s,t,α) rsp)

**begin**

**interpretation**
- **nsrd-thy**: upth-theory-continuous UTHY(NSRDES, (s,t::trace,α) rsp)

**rewrites**
- $P \cdot P \in \text{carrier}(\text{uthy-order NSRDES}) \iff P \text{ is NSRD}$
- $P$ is $H_{\text{NSRDES}} \iff P \text{ is NSRD}$
- $(\mu X \cdot F (H_{\text{NSRDES}} X)) = (\mu X \cdot F (\text{NSRD} X))$
- $\text{carrier} (\text{uthy-order NSRDES}) \rightarrow \text{carrier} (\text{uthy-order NSRDES}) \equiv [\text{NSRD}]_H \rightarrow [\text{NSRD}]_H$
- $H_{\text{NSRDES}} \equiv [\text{NSRD}]_H \rightarrow [\text{NSRD}]_H$
- $T_{\text{NSRDES}} = \text{Miracle}$
- $\exists \text{NSRDES} = H_R$
- $le (\text{uthy-order NSRDES}) = \text{op} \subset$

**proof**

**interpret**
- **lat**: upth-theory-continuous UTHY(NSRDES, (s,t,α) rsp)

**by** (unfold-locale, simp-all add: nrsdes-hcond-def nrsdes-unit-def closure Healthy-if)

**show**
- $1: T_{\text{NSRDES}} = (\text{Miracle} :: (s,t,\alpha) hrel-rsp)$

**by** (metis NSRD-Miracle NSRD-is-SRD lat.top-healthy lat.uthy-continuous-axioms nrsdes-hcond-def nsdes-theory-continuous Meet-top upth-continuous.add-commute upth-theory-continuous.meet-top)

**thus**
- upth-theory-continuous UTHY(NSRDES, (s,t,α) rsp)

**by** (unfold-locale, simp-all add: nrsdes-hcond-def nrsdes-unit-def closure Healthy-if Miracle-left-zero SRD-left-unit NSRD-right-unit)

**qed** (simp-all add: nrsdes-hcond-def nrsdes-unit-def closure Healthy-if)

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

abbreviation TestR (test_R) where
test_R P ≡ utest NSRDES P

abbreviation StarR :: ('s, 't:trace, 'α) hrel-rsp ⇒ ('s, 't, 'α) hrel-rsp (•^R [999] 999) where
StarR P ≡ P*NSRDES

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

lemma Star-R-rdes-def [rdes-def]:
assumes P is RC Q is RR R is RR $s t` z Q
shows (R,(P ⊢ Q o R))^R = R_s(([R]^r wp, P) ⊢ R^r ;; Q o R^r)
bysimp add: rrel-thy.Star-alt-def nsrd-thy.Star-alt-def assms closure rdes-def unrest rpred disj-upred-def

end

6 Syntax for reactive design contracts

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

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

definition mk-RD :: 's apred ⇒ ('t:trace ⇒ 's apred) ⇒ ('t ⇒ 's hrel) ⇒ ('s, 't, 'α) hrel-rsp where
mk-RD P Q R = R_s(([P]_S< [Q(x)]_S< [x→&tt] o [R(x)]_S[x→&tt]))

definition trace-pred :: ('t:trace ⇒ 's apred) ⇒ ('s, 't, 'α) hrel-rsp where
[upred-defs]: trace-pred P = ([P x])_S<[x→&tt]

syntax
  -trace-var :: logic
  -mk-RD :: logic ⇒ logic ⇒ logic ⇒ logic ([/] ⊢ /) [ / ]R
  -trace-pred :: logic ⇒ logic ([/]_)

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

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

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

lemma preR-mk-RD [rdes]: pre\( R([P \vdash Q(\text{trace}) \mid R(\text{trace})]_R) = R1([P]_{S<}) \)
by (simp add: mk-RD-def rea-pre-RHS-design usubst unrest R2c-not R2c-lift-state-pre)

lemma trace-pred-RR-closed [closure]:
\[ P \vdash Q(\text{trace}) \]
by (rel-auto)

lemma unrest-trace-pred-st′ [unrest]:
\[ st`\#[P \vdash Q(\text{trace})] \]
by (rel-auto)

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

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

lemma postR-mk-RD [rdes]: post\( R([P \vdash Q(\text{trace}) \mid R(\text{trace})]_R) = ([P]_{S<} \rightarrow_r R1(((R(\text{trace})]_S)[\text{trace} \rightarrow \& \text{tt}])) \)
by (simp add: mk-RD-def rea-post-RHS-design usubst 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
\[ Q \text{ is SRD} \quad [P_1]_{S<} \Rightarrow pre\text{ }R \quad Q' \]
\[ [P_1]_{S<} \land peri\text{ }R \Rightarrow [P_2 \ x]_{S<}[x \rightarrow \& \text{tt}] \]
\[ [P_1]_{S<} \land post\text{ }R \Rightarrow [P_3 \ x]_{S}[x \rightarrow \& \text{tt}] \]
shows
\[ [P_1 \vdash P_2(\text{trace}) \mid P_3(\text{trace})]_R \subseteq Q \]
proof
  have \[ [P_1 \vdash P_2(\text{trace}) \mid P_3(\text{trace})]_R \subseteq R_\text{s}(pre\text{ }R(Q) \vdash peri\text{ }R(Q) \circ post\text{ }R(Q)) \]
  using assms
  by (simp add: mk-RD-def, rule-tac srdes-tri-refine-intro, simp-all)
thus \[ \text{?thesis} \]
by (simp add: SRD-reactive-tri-design assms(1))
qed

end

7 Reactive design tactics

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

Theorems for normalisation

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

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

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

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

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

Tactic to simplify the definition of a reactive design

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

Tactic to split a refinement conjecture into three POs

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

Tactic to split an equality conjecture into three POs

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

Tactic to prove a refinement

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

Tactics to prove an equality

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

Via antisymmetry

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

Tactic to calculate pre/peri/postconditions from reactive designs

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

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

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

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

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

end
8 Reactive design parallel-by-merge

theory utp-rdes-parallel

imports
  utp-rdes-normal
  utp-rdes-tactics

begin

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

lemma st-U0-alpha: \( \exists \, s \cdot [II]_0 = (\exists \, s \cdot [II]_1) \)
by (rel-auto)

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

definition skip-rm :: \((\text{'s}::\text{'s}) \cdot \text{trace} \cdot \alpha) \Rightarrow \text{trace} \Rightarrow (\text{II} \cdot \alpha)\)
where
[upred-defs]: \( \text{II} _ \alpha = (\exists \, s \cdot \text{skip} _ m \cdot (\neg \, \$ok _ < \land \text{str} _ \leq _ u \text{str} _ \neg ')) \)

definition [upred-defs]: \( \text{R3h} _ \alpha (M) = (\text{II} _ \alpha \# _ {\text{top}} \# _ {\text{bottom}} M) \)

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

lemma R3h-par-by-merge [closure]:
assumes \( P \) is \( R3h \) \( Q \) is \( R3h \) \( M \) is \( R3h \)
shows \( (P \# _ {\text{top}} \# _ {\text{bottom}} Q) \) is \( R3h \)
proof –
  have \( (P \# _ {\text{top}} \# _ {\text{bottom}} Q) = (((P \# _ {\text{top}} \# _ {\text{bottom}} Q)[\text{true} / \$ok] \circ \$ok \triangleright (P \# _ {\text{top}} \# _ {\text{bottom}} Q)[\text{false} / \$ok])[\text{true} / \text{wait}] \circ \text{wait} \triangleright (P \# _ {\text{top}} \# _ {\text{bottom}} Q)) \)
by (simp add: cond-var-subst-left cond-var-subst-right)
  also have \( \ldots = (((P \# _ {\text{top}} \# _ {\text{bottom}} Q)[\text{true} / \$ok, \text{wait}] \circ \$ok \triangleright (P \# _ {\text{top}} \# _ {\text{bottom}} Q)[\text{false} / \$ok, \text{wait}] \circ \text{wait} \triangleright (P \# _ {\text{top}} \# _ {\text{bottom}} Q)) \)
by (rel-auto)
  also have \( \ldots = (((P \# _ {\text{top}} \# _ {\text{bottom}} Q)[\text{true} / \$ok, \text{wait}] \circ \$ok \triangleright (P \# _ {\text{top}} \# _ {\text{bottom}} Q)[\text{false} / \$ok, \text{wait}] \circ \text{wait} \triangleright (P \# _ {\text{top}} \# _ {\text{bottom}} Q)) \)
by (simp add: closure assms unrest)
  finally show \text{thesis} by (simp add: closure assms unrest)
qed

also have \( \ldots = (((P \# _ {\text{top}} \# _ {\text{bottom}} Q)[\text{true} / \$ok, \text{wait}] \circ \$ok \triangleright (R1 (\text{true})))[\text{false} / \$ok, \text{wait}) \circ \text{wait} \triangleright (P \# _ {\text{top}} \# _ {\text{bottom}} Q)) \)
proof –
  have \( (P \# _ {\text{top}} \# _ {\text{bottom}} Q)[\text{false} / \$ok, \text{wait}] = (((P \# _ {\text{top}} \# _ {\text{bottom}} Q)[\text{true} / \$ok] \circ \$ok \triangleright (P \# _ {\text{top}} \# _ {\text{bottom}} Q)[\text{false} / \$ok])[\text{true} / \text{wait}] \circ \text{wait} \triangleright (P \# _ {\text{top}} \# _ {\text{bottom}} Q)) \)
by (simp add: par-by-merge-def U0-as-alpha U1-as-alpha assms Healthy-if)
  also have \( \ldots = (((P \# _ {\text{top}} \# _ {\text{bottom}} Q)[\text{true} / \$ok, \text{wait}] \circ \$ok \triangleright (P \# _ {\text{top}} \# _ {\text{bottom}} Q)[\text{false} / \$ok, \text{wait}] \circ \text{wait} \triangleright (P \# _ {\text{top}} \# _ {\text{bottom}} Q)) \)
by (rel-auto)
  also have \( \ldots = (((P \# _ {\text{top}} \# _ {\text{bottom}} Q)[\text{true} / \$ok, \text{wait}] \circ \$ok \triangleright (P \# _ {\text{top}} \# _ {\text{bottom}} Q)[\text{false} / \$ok, \text{wait}] \circ \text{wait} \triangleright (P \# _ {\text{top}} \# _ {\text{bottom}} Q)) \)
by (simp add: closure assms Healthy-if)
  also have \( \ldots = (((P \# _ {\text{top}} \# _ {\text{bottom}} Q)[\text{true} / \$ok, \text{wait}] \circ \$ok \triangleright (P \# _ {\text{top}} \# _ {\text{bottom}} Q)[\text{false} / \$ok, \text{wait}] \circ \text{wait} \triangleright (P \# _ {\text{top}} \# _ {\text{bottom}} Q)) \)
by (rel-auto)
  finally show \text{thesis} by (simp add: closure assms unrest)
qed
also have ... = (R1(true))[false,true/$\$ok,\$wait]  
  by (rel-blast)  
finally show ?thesis by simp 
qed

also have ... = (((|$\$st \cdot II|$ < $\$ok \triangleright R1(true)$) < $\$wait \triangleright (P ||_M Q)$)  
  by (rel-auto)  
also have ... = R3h(P ||_M Q)  
  by (simp add: R3h-cases)  
finally show ?thesis  
  by (simp add: Healthy-def) 
qed

definition [upred-defs]: RD1m(M) = (M \lor \neg $\$ok \subset \land \$str_{\leq u} \land \$tr \')

lemma RD1-par-by-merge [closure]:  
  assumes P is R1 Q is R1 M is R1m P is RD1 Q is RD1 M is RD1m  
  shows (P ||_M Q) is RD1  
proof  
  have 1: (RD1(R1(P)) ||_RD1m(R1m(M))) RD1(R1(Q)))[false/$\$ok] = R1(true)  
    by (rel-blast)  
  have (P ||_M Q) = (P ||_M Q)[true/$\$ok \triangleright $\$ok \triangleright (P ||_M Q)[false/$\$ok]  
    by (simp add: cond-var-split)  
  also have ... = R1(P ||_M Q) < $\$ok \triangleright R1(true)  
    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 ||_M Q)  
    by (simp add: Healthy-if RD1-alt-def assms(3))  
finally show ?thesis  
    by (simp add: Healthy-def) 
qed

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

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

definition nmerge-rd0 (N0) where  
  [upred-defs]: N0(M) = ($\$wait \cdot u (|$\$0 - \$wait \lor |$I - \$wait$) \land |$\$str_{\leq u} \land \$tr \')  
    \land (\exists |$\$0 - \$ok;|$I - \$ok;|$\$ok \cdot;|$\$0 - \$wait;|$I - \$wait;|$\$wait;|$\$wait \cdot M))
definition nmerge-rd1 \((N_1)\) where
\[
\begin{align*}
\text{[upred-defs]: } N_1(M) &= (\$ok' =_u (\$0-\text{ok} \land \$1-\text{ok}) \land N_0(M))
\end{align*}
\]

definition nmerge-rd \((N_R)\) where
\[
\begin{align*}
\text{[upred-defs]: } N_R(M) &= (\exists \$st < \cdot \$v' =_u \$v_\prec) < \$wait_\prec \triangleright N_1(M) \prec \$ok_\prec \triangleright (\$tr_\prec \leq_u \$tr')
\end{align*}
\]

definition merge-rd1 \((M_1)\) where
\[
\begin{align*}
\text{[upred-defs]: } M_1(M) &= (N_1(M) ; ; II_R)
\end{align*}
\]

definition merge-rd \((M_R)\) where
\[
\begin{align*}
\text{[upred-defs]: } M_R(M) &= N_R(M) ; ; II_R
\end{align*}
\]

abbreviation rdes-par \((\cdot \parallel_R \cdot)\) where
\[
\begin{align*}
P \parallel_R M Q \equiv P \parallel M_R(M) Q
\end{align*}
\]

Healthiness condition for reactive design merge predicates
\[
\begin{align*}
\text{definition [upred-defs]: } \text{RDM}(M) = R2m(\exists \$0-\text{ok}; \$1-\text{ok}; \$ok_\prec; \$ok'; \$0-\text{wait}; \$1-\text{wait}; \$\text{wait}_\prec; \$\text{wait}' \cdot M)
\end{align*}
\]

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

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

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

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

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

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

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

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

lemma RDM-anrests [unrest]:
assumes M is RDM
shows $0 \rightarrow ok \parallel M \parallel 1 \rightarrow ok \parallel M \parallel \text{ok} \parallel M \parallel \text{ok} \parallel M
shows M is RDM
by (metis no-types, hide-lams Healthy-def R1m-idem R2m-def RDM-def)

lemma ex-st'\parallel R2m-closed [closure]:
assumes P is R2m
shows P \parallel \exists st' \rightarrow P is R2m
proof
also have ... by (rule RR-R2-intro, simp-all: unrest assms RR-implies-R2 closure)
also have ...
proof
also have ...
by (simp add: true-alt-def [THEN sym] false-alt-def [THEN sym] disj-assoc)
lemma skip-sra-thesis [usubst]:
\[ H_R^f = R1(\neg \$ok) \]
by (rel-auto)

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

lemma parallel-assm-lemma:
assumes \( P \) is RD2
shows \( \text{pre}_s \uplus (P \parallel M_R(M)) \ Q = (((\text{pre}_s \uplus P) \parallel N_0(M) \uplus R1(\text{true})) (\text{cmt}_s \uplus Q)) \)
\[ \vee ((\text{cmt}_s \uplus P) \parallel N_0(M) \uplus R1(\text{true}) (\text{pre}_s \uplus Q)) \]
proof –
have \( \text{pre}_s \uplus (P \parallel M_R(M)) \ Q = \text{pre}_s \uplus ((P \parallel Q) :: M_R(M)) \)
by (simp add: par-by-merge-def)
also have \( ... = (((P \parallel Q)[\text{true},\text{false}/\$\text{ok},\$\text{wait}] :: N_R M :: R1(\neg \$\text{ok})) \)
by (simp add: merge-rd-def usubst, rel-auto)
also have \( ... = (((P[\text{true},\text{false}/\$\text{ok},\$\text{wait}] :: Q[\text{true},\text{false}/\$\text{ok},\$\text{wait}]) :: N_1(M) :: R1(\neg \$\text{ok})) \)
by (rel-auto robust, (metis)+)
also have \( ... = (((P[\text{true},\text{false}/\$\text{ok},\$\text{wait}] \parallel Q[\text{true},\text{false}/\$\text{ok},\$\text{wait}]) :: N_1(M) :: R1(\neg \$\text{ok})) \)
by (simp add: parallel-ok-cases, subst-tac)
also have \( ... = ((?C1 \lor (?C2 \lor (?C3 \lor (?C4)))) \)
by (subst parallel-ok-cases, subst-tac)
also have \( ... = (?C2 \lor (?C3)) \)
proof –
have \( \neg \ ?C1 = \text{false} \)
by (rel-auto)
moreover have \( \neg \ ?C4 \Rightarrow \ ?C3 \) (is \( \neg (A \parallel ?B) \Rightarrow (?C :: ?D) \))
proof –
from assms have \( \neg \ ?P^f \Rightarrow \ ?P^t \)
by (metis RD2-def H2-equivalence Healthy-def)
hence \( ?P^f \Rightarrow \ ?P^t \)
by (rel-auto)
have \( \neg \ ?A \Rightarrow \ ?C \)
using \( P \) by (rel-auto)
moreover have \( \neg \ ?B \Rightarrow \ ?D \)
by (rel-auto)
ultimately show \( \text{thesis} \)
by (simp add: impl-seq-mono)
proof
ultimately show \( \text{thesis} \)
by (simp add: substumption2)
proof
also have \( ... = ( \)

68
proof

lemma JL1: \((M_1 M)^! [false, true \parallel 0 - ok, \$1 - ok] = N_0(M) ;; R1(true))\)
by (rel-blast)

lemma JL2: \((M_1 M)^! [true, false \parallel 0 - ok, \$1 - ok] = N_0(M) ;; R1(true))\)
by (rel-blast)

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

lemma parallel-commitment-lemma-1:
assumes $P$ is RD2
shows $\text{cntr} \vdash (P \parallel M_R(M) Q) =$
$
((\text{cntr} \vdash P) \parallel (\$ok \land N_0 M) ; ; H_{R^t} (\text{cntr} \vdash Q)) \lor
((\text{pre} \vdash P) \parallel N_0(M) ; ; R_1(\text{true}) (\text{cntr} \vdash Q)) \lor
((\text{cntr} \vdash P) \parallel N_0(M) ; ; R_1(\text{true}) (\text{pre} \vdash Q))$

proof –
  have $\text{cntr} \vdash (P \parallel M_R(M) Q) = (P[\text{true, false}/\$ok, \$wait] \parallel (M_1 M)^t Q[\text{true, false}/\$ok, \$wait])$
  by (simp add: usubst rel-auto)
  also have $\ldots = ((P[\text{true, false}/\$ok, \$wait] \parallel Q[\text{true, false}/\$ok, \$wait]) ; ; (M_1 M)^t)$
  by (simp add: par-by-merge-def)
  also have $\ldots =$
$
((\text{cntr} \vdash P) \parallel s (\text{cntr} \vdash Q)) ; ; (M_1 M)^t[\text{true, true}/\$0-ok, \$1-ok]) \lor
((\text{pre} \vdash P) \parallel s (\text{cntr} \vdash Q)) ; ; (M_1 M)^t[\text{false, true}/\$0-ok, \$1-ok]) \lor
((\text{cntr} \vdash P) \parallel s (\text{pre} \vdash Q)) ; ; (M_1 M)^t[\text{true, true}/\$0-ok, \$1-ok]) \lor
((\text{pre} \vdash P) \parallel s (\text{pre} \vdash Q)) ; ; (M_1 M)^t[\text{false, false}/\$0-ok, \$1-ok])$
  by (subst parallel-ok-cases, subst-tac)
  also have $\ldots =$
$
((\text{cntr} \vdash P) \parallel s (\text{cntr} \vdash Q)) ; ; (M_1 M)^t[\text{true, true}/\$0-ok, \$1-ok]) \lor
((\text{pre} \vdash P) \parallel s (\text{cntr} \vdash Q)) ; ; (N_0(M) ; ; R_1(\text{true})) \lor
((\text{cntr} \vdash P) \parallel s (\text{pre} \vdash Q)) ; ; (N_0(M) ; ; R_1(\text{true})) \lor
((\text{pre} \vdash P) \parallel s (\text{pre} \vdash Q)) ; ; (N_0(M) ; ; R_1(\text{true})))$
(is - = (?C1 \lor p ?C2 \lor p ?C3 \lor p ?C4))
  by (simp add: JL1 JL2 JL3)
  also have $\ldots =$
$
((\text{cntr} \vdash P) \parallel s (\text{cntr} \vdash Q)) ; ; (M_1(M)^t[\text{true, true}/\$0-ok, \$1-ok]) \lor
((\text{pre} \vdash P) \parallel s (\text{cntr} \vdash Q)) ; ; (N_0(M) ; ; R_1(\text{true})) \lor
((\text{cntr} \vdash P) \parallel s (\text{pre} \vdash Q)) ; ; (N_0(M) ; ; R_1(\text{true})))$
  proof –
  from asms have $\vdash p' \Rightarrow p^t$.
  by (metis RD2-def H2-equivalence Healthy-def)
  hence $P; p^f_j \Rightarrow p^{t,f_j}$
  by (rel-auto)
  have $\vdash ?C4 \Rightarrow ?C3$ (is $\vdash (?A ; ; ?B) \Rightarrow (?C ; ; ?D)$)
  proof –
  have $\vdash ?A \Rightarrow ?C$.
  using $P$ by (rel-auto)
  thus $\vdash ?thesis$
  by (simp add: impl-seq-mono)
  qed
  thus $\vdash ?thesis$
  by (simp add: subsumption2)
  qed
  finally show $\vdash ?thesis$
  by (simp add: par-by-merge-def JL4)
  qed

lemma parallel-commitment-lemma-2:
assumes $P$ is RD2
shows $\text{cntr} \vdash (P \parallel M_R(M) Q) =$
$
(((\text{cntr} \vdash P) \parallel (\$ok \land N_0 M) ; ; H_{R^t} (\text{cntr} \vdash Q)) \lor\text{pre} \vdash (P \parallel M_R(M) Q))$
by (simp add: parallel-commitment-lemma-1 assms parallel-assm-lemma)

lemma parallel-commitment-lemma-3:
  \( M \text{ is } R1m \implies (\langle \text{ok} \rangle \land N_0 \text{ M}) ; ; H_{R'} \text{ is } R1m \)
by (rel-simp, safe,metis+)

lemma parallel-commitment:
  assumes \( P \text{ is SRD } Q \text{ is SRD } M \text{ is RDM} \)
  shows \( \text{cmt}_R(P \parallel M_{R}(M) \parallel Q) \implies R_{\text{true}} \text{ cmt}_R(Q) \)\)
by (simp add: parallel-commitment-lemma-2 parallel-commitment-lemma-3 Healthy-if assms cmt R-def pre$_s$-SRD closure rea-impl-def disj-comm unrest)

theorem parallel-reactive-design:
  assumes \( P \text{ is SRD } Q \text{ is SRD } M \text{ is RDM} \)
  shows \( (P \parallel M_{R}(M) \parallel Q) = R_s(\neg_r (\neg_r \text{ pre}_{R}(P)) \parallel N_0(M) :: R_{\text{true}} \text{ cmt}_R(Q)) \land \neg_r (\text{cmt}_R(P) \parallel N_0(M) :: R_{\text{true}} (\neg_r \text{ pre}_{R}(Q))) \land (\text{cmt}_R(P) \parallel (\langle \text{ok} \rangle \land N_0 \text{ M}) ; ; H_{R'} \text{ cmt}_R(Q))) \)
(is ?lhs = ?rhs)
proof –
  have \( (P \parallel M_{R}(M) \parallel Q) = R_s(\text{pre}_{R}(P \parallel M_{R}(M) \parallel Q)) \)
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:
  \( (\langle \text{ok} \rangle \land P) ; ; H_{R}[\text{true,true}/\text{ok'}, \text{wait}'] = (\exists \text{ st'} \cdot P)[\text{true,true}/\text{ok'},\text{wait'}] \)
(is \( ?lhs = \ ?rhs \))
proof –
  have \( ?lhs = (\langle \text{ok} \rangle \land P) ; ; (\exists \text{ st } \cdot II)[\text{true,true}/\text{ok'}, \text{wait}'] \)
by (rel-blast)
also have \( ... = \ ?rhs \)
by (rel-auto)
finally show \( ?thesis \).
qed

lemma parallel-pericondition-lemma2:
  assumes \( M \text{ is RDM} \)
  shows \( (\exists \text{ st'} \cdot N_0(M))[\text{true,true}/\text{ok'}, \text{wait}'] = (\langle \text{0\text{-}wait} \lor \text{1\text{-}wait} \rangle \land (\exists \text{ st'} \cdot M)) \)
proof –
  have \( (\exists \text{ st'} \cdot N_0(M))[\text{true,true}/\text{ok'}, \text{wait}'] = (\exists \text{ st'} \cdot (\langle \text{0\text{-}wait} \lor \text{1\text{-}wait} \rangle \land \text{str'} \geq \text{str} < \land M) \)
by (simp add: usubst unrest nmerge-rd0-def ex-unrest Healthy-if R1m-def assms)
also have \( ... = (\exists \text{ st'} \cdot (\langle \text{0\text{-}wait} \lor \text{1\text{-}wait} \rangle \land M) \)
by (metis (no_types, hide_lams) Healthy-if R1m-def R1m-idem R2m-def RDM-def assms utp-pred-laws.inf-commute)
also have \( ... = (\langle \text{0\text{-}wait} \lor \text{1\text{-}wait} \rangle \land (\exists \text{ st'} \cdot M)) \)
by (rel-auto)
finally show \( ?thesis \).
qed

lemma parallel-pericondition-lemma3:
  \( (\langle \text{0\text{-}wait} \lor \text{1\text{-}wait} \rangle \land (\exists \text{ st'} \cdot M)) = (\langle \text{0\text{-}wait} \land \text{1\text{-}wait} \rangle \land (\exists \text{ st'} \cdot M)) \lor (\neg \langle \text{0\text{-}wait} \land \text{1\text{-}wait} \land (\exists \text{ st'} \cdot M)) \lor (\langle \text{0\text{-}wait} \land \text{1\text{-}wait} \land (\exists \text{ st'} \cdot M)) \lor (\langle \text{0\text{-}wait} \land \text{1\text{-}wait} \land (\exists \text{ st'} \cdot M))) \)

by (rel-auto)

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

*fixes* $P :: 's,t::trace,'a* 
*assumes* $P :: SRD Q is SRD RDM

*shows* \( \text{peri}_R(P \parallel M_R(M) Q) = (\text{peri}_R(P \parallel M_R Q) \Rightarrow \text{peri}_R(P) \parallel \exists s_\text{st} \cdot M \text{peri}_R(Q)) \)

*proof –

  *have* \( \text{peri}_R(P \parallel M_R(M) Q) = 
    (\text{peri}_R(P \parallel M_R Q) \Rightarrow \text{peri}_R(P) \parallel \exists s_\text{st} \cdot M \text{peri}_R(Q)) \)
    by (simp add: \text{peri-cmt-def parallel-commitment SRD-healths asms ubsub unrest assms})

  *also have* ... = \( (\text{peri}_R(P \parallel M_R Q) \Rightarrow \text{peri}_R(P) \parallel \exists s_\text{st} \cdot M \text{peri}_R(Q)) \)
    by (simp add: parallel-pericondition-lemma1)

  *also have* ... = \( (\text{peri}_R(P \parallel M_R Q) \Rightarrow \text{peri}_R(P) \parallel \exists s_\text{st} \cdot M \text{peri}_R(Q)) \)
    by (simp add: parallel-pericondition-lemma2 assms)

  *also have* ... = \( (\text{peri}_R(P \parallel M_R Q) \Rightarrow \text{peri}_R(P) \parallel \exists s_\text{st} \cdot M \text{peri}_R(Q)) \)
    by (simp add: parallel-pericondition-lemma3 seqr-or-distr)

  *also have* ... = \( (\text{peri}_R(P \parallel M_R Q) \Rightarrow \text{peri}_R(P) \parallel \exists s_\text{st} \cdot M \text{peri}_R(Q)) \)
    by (simp add: seqr-right-one-point-true seqr-right-one-point-false cmt-R-def post-R-def peri-R-def ubsub unrest assms)

  *also have* ... = \( (\text{peri}_R(P \parallel M_R Q) \Rightarrow \text{peri}_R(P) \parallel \exists s_\text{st} \cdot M \text{peri}_R(Q)) \)
    by (simp add: par-by-merge-alt-def)

  *finally show* \(?thesis .

**qed**

**lemma** parallel-postcondition-lemma1:

\( \text{peri}_R \text{peri}_R \text{peri}_R \text{peri}_R Q) \parallel \exists s_\text{st} \cdot M \text{peri}_R(Q) \)

*proof –

  *have* \(?lhs = \exists \text{peri}_R(Q) \parallel \exists s_\text{st} \cdot M \text{peri}_R(Q) \)
    by (rel-blast)

  *also have* ... = \(?rhs \parallel \exists s_\text{st} \cdot M \text{peri}_R(Q) \)
    by (rel-auto)

  *finally show* \(?thesis .

**qed**

**lemma** parallel-postcondition-lemma2:

*assumes* $M :: RDM

*shows* \( (\exists \text{peri}_R(Q) \parallel \exists s_\text{st} \cdot M \text{peri}_R(Q) \)

*proof –

  *have* \( (\exists \text{peri}_R(Q) \parallel \exists s_\text{st} \cdot M \text{peri}_R(Q) \)
    by (simp add: ubsub unrest ubsub unrest assms)

  *also have* ... = \( (\exists \text{peri}_R(Q) \parallel \exists s_\text{st} \cdot M \text{peri}_R(Q) \)
    by (metis Healthy-if R1m-def RDM-R1m assms ubsub unrest assms)

  *finally show* \(?thesis .

**qed**
proof

have \( \text{post}(P \parallel M_R(M), Q) = \text{pre}(P \parallel M_R(M), Q) \Rightarrow \text{post}(P) \mid M \text{ post}(Q) \)

by simp

also have \( \text{post}(P \parallel M_R(M), Q) \Rightarrow \text{post}(P) \mid M \text{ post}(Q) \)

by simp

finally show \(?thesis\).

qed

lemma parallel-postcondition-lemma:

fixes \( M :: (\text{'s, 't::trace, 'a}) \text{ rsp merge} \)

assumes \( P \text{ is SRD} \quad Q \text{ is SRD} \quad M \text{ is RDM} \)

shows \( \neg \text{ pre}(P) \parallel M_R(M) \quad \Rightarrow \text{ post}(P) \parallel M \text{ post}(Q) \)

proof

have \( \neg \text{ pre}(P) \parallel M_R(M) \quad \Rightarrow \text{ post}(P) \parallel M \text{ post}(Q) \)

by simp

also have \( \neg \text{ pre}(P) \parallel M_R(M) \quad \Rightarrow \text{ post}(P) \parallel M \text{ post}(Q) \)

by simp

finally show \(?thesis\).

qed

lemma parallel-precondition-lemma:

fixes \( M :: (\text{'s, 't::trace, 'a}) \text{ rsp merge} \)

assumes \( P \text{ is NSRD} \quad Q \text{ is NSRD} \quad M \text{ is RDM} \)

shows \( \neg \text{ pre}(P) \parallel M_R(M) \quad \Rightarrow \text{ post}(P) \parallel M \text{ post}(Q) \)

proof

have \( \neg \text{ pre}(P) \parallel M_R(M) \quad \Rightarrow \text{ post}(P) \parallel M \text{ post}(Q) \)

by simp

also have \( \neg \text{ pre}(P) \parallel M_R(M) \quad \Rightarrow \text{ post}(P) \parallel M \text{ post}(Q) \)

by simp

finally show \(?thesis\).

qed
thus $\?thesis$
by (simp add: nmerge-rl0-def unrest assms closure ex-unrest usbst)

qed
also have ... = ((\neg \pre R P \parallel M ;; R1(true) peri R Q \lor (\neg \pre R P) \parallel M ;; R1(true) post R Q)
by (simp add: par-by-merge-alt-def)

finally show $\?thesis$ .

qed

lemma swap-nmerge-rl0:
  swap \_\_ \_; N_0(M) = N_0(swap \_\_ \_; M)
by (rel-auto, meson+)

lemma SymMerge-nmerge-rl0 [closure]:
  M is SymMerge \implies N_0(M) is SymMerge
by (rel-auto, meson+)

lemma swap-merge-rl':
  swap \_\_ \_; N_R(M) = N_R(swap \_\_ \_; M)
by (rel-blast)

lemma swap-merge-rl:
  swap \_\_ \_; M_R(M) = M_R(swap \_\_ \_; M)
by (simp add: merge-rl-def seqr-assoc[THEN sym] swap-merge-rl')

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

lemma nmerge-rl1-merge3:
  assumes M is RDM
  shows M3(N_1(M)) = (\$ok' = u (\$0-ok \land \$1-ok \land \$1-1-ok) \land
  \$wait' = u (\$0-wait \lor \$1-0-wait \land \$1-1-wait) \land
  M3(M))
proof -
  have M3(N_1(M)) = M3(\$ok' = u (\$0-ok \land \$1-ok) \land
  \$wait' = u (\$0-wait \lor \$1-wait) \land
  \$tr < u \$tr' \land
  (3 \{\$0-ok, \$1-ok, \$ok <, \$ok', \$0-wait, \$1-wait, \$wait <, \$wait' \cdot RDM(M))
by (simp add: nmerge-rl1-def nmerge-rl0-def assms Healthy-if)
also have ... = M3(\$ok' = u (\$0-ok \land \$1-ok) \land \$wait' = u (\$0-wait \lor \$1-wait) \land RDM(M))

74
by (rel-blast)
also have "... = (ążok’ = u (явление 0 - ok ∧ $1$ - ok ∧ $1$ - ok) ∧ $w_\text{ait}$ = u (явление 0 - wait ∨ $1$ - wait) ∨ $1$ - wait) ∧ $M_3(RDM(M))" by (rel-blast)
also have "... = (ążok’ = u (явление 0 - ok ∧ $1$ - ok ∧ $1$ - ok) ∧ $w_\text{ait}$ = u (явление 0 - wait ∨ $1$ - wait) ∨ $1$ - wait) ∧ $M_3(M))"
by (simp add: assms Healthy-if)
finally show "?thesis".
qed

lemma nmerge-rd-merge3:
$M_3(N_\text{rd}R(M)) = (\exists \text{st}_\text{\textless} \cdot \text{\textless}v’ = u \text{\textless}w_\text{ait} < M_3(N_1 M) < \text{\textless}sok_\text{\textless} < (\text{\textless}tr_\text{\textless} \leq u \text{\textless}tr’)
by (rel-blast)

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

lemma parallel-precondition:
fixes $P$ :: (’$t$:trace,’$α$:trace,’$\alpha$:trace) rmp merge
assumes $P$ is NSRD $Q$ is NSRD $M$ is RDM
shows preR($P$ || $M_R(M)$) $Q$ =
(¬r ((¬r preR $P$) || $M :: R1(true) peri R Q) ∧ ¬r ((¬r preR $P$) || $M :: R1(true) post R Q) ∧
¬r ((¬r preR $Q$) ||(swap_m :: $M$) :: R1(true) peri R P) ∧ ¬r ((¬r preR $Q$) ||(swap_m :: $M$) :: R1(true) post R P))
proof -
have a: (¬r preR($P$) || $N_0(M)$ :: R1(true) cnt r $Q$ =
((¬r preR $P$) || $M :: R1(true) peri R Q ∨ (¬r preR $P$) || $M :: R1(true) post R Q)$
by (simp add: parallel-precondition-lemma assms)

have b: (¬r cnt r $P$ || $N_0(M)$ :: R1 true (¬r preR $Q$) =
(¬r (¬r preR($Q$)) || $N_0$(swap_m :: $M$) :: R1(true) cnt r($P$))

have c: (¬r preR($Q$)) || $N_0$(swap_m :: $M$) :: R1(true) cnt r($P$) =
((¬r preR $Q$) ||(swap_m :: $M$) :: R1(true) peri R P ∨ (¬r preR $Q$) ||(swap_m :: $M$) :: R1(true) post R $P$)
by (simp add: parallel-precondition-lemma closure assms)

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

Weakest Parallel Precondition
definition wrR ::
(’$t$:trace,’$α$:trace) hrel-rp ⇒
(’$t$ :: trace,’$α$:trace) rmp merge ⇒
(’$t$, ’$α$:trace) hrel-rp ⇒
\((t, \alpha)\) krel-rp \((- \ \text{wr}_R'(\cdot) - [60,0,61] 61)\)

where \([\text{upred-defs}]: Q \ \text{wr}_R(M) P = (\neg_r ((\neg\_r P) \parallel_M ;; R1(\text{true}) Q))\)

**Lemma wr-R1 [closure]:**
- \(M\) is \(R1m\) \(\Rightarrow\) \(Q\ \text{wr}_R(M) P\) is \(R1\)
- by (simp add: wr-R-def closure)

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

**Lemma wr-R2-lemma:**
- assumes \(P\) is \(R2\) \(Q\) is \(R2\) \(M\) is \(R2m\)
- shows \((\neg_r P) \parallel_M Q\) \(\parallel\) \(R1(\text{true})\) is \(R2\)
- proof
  - have \((\neg_r P) \parallel_M Q\) is \(R2\)
  - by (simp add: closure assms)
  - thus \(?thesis
  - by (simp add: closure)
- qed

**Lemma wr-R-R2 [closure]:**
- assumes \(P\) is \(R2\) \(Q\) is \(R2\) \(M\) is \(R2m\)
- shows \(Q\ \text{wr}_R(M) P\) is \(R2\)
- proof
  - have \((\neg_r P) \parallel_M Q\) is \(R2\)
  - by (simp add: wr-R-R2-lemma assms)
  - thus \(?thesis
  - by (simp add: wr-R-def par-by-merge-seq-add closure)
- qed

**Lemma wr-R-RR [closure]:**
- assumes \(P\) is \(RR\) \(Q\) is \(RR\) \(M\) is \(RDM\)
- shows \(Q\ \text{wr}_R(M) P\) is \(RR\)
- apply (rule RR-intro)
- apply (simp-all add: unrest assms closure wr-R-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 wr-R-R2-lemma)
- done

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

**Lemma uppr-choice [wp]: \((P \lor Q) \ \text{wr}_R(M) R = (P \ \text{wr}_R(M) R \land Q \ \text{wr}_R(M) R)\)**
- proof
  - have \((P \lor Q) \ \text{wr}_R(M) R\) = \(\neg_r \neg_c ((\neg_r R) \parallel U0 \land (P \parallel U1 \lor Q \parallel U1) \land \$v <'_u =_u \$v \parallel_M ;; \text{true})\)
    - by (simp add: wr-R-def par-by-merge-def seqr-or-distr)
  - also have \(... = (\neg_r ((\neg_r R) \parallel U0 \land P \parallel U1 \land \$v <'_u =_u \$v \lor (\neg_r R) \parallel U0 \land Q \parallel U1 \land \$v <'_u =_u \$v \land \text{true})\)
    - by (simp add: wr-R-def closure)
  - qed
proof

by (simp add: conj-disj-distr utp-pred-laws,inf-sup-distrib2)
also have ... = \((-\tau \cdot ((-\tau \cdot R) \land U_0 \land P \land U_1 \land \$v') =_u \$v) \land M \land \tau\lor ((-\tau \cdot R) \land U_0 \land Q \land U_1 \land \$v' =_u \$v) \land M \land \tau\lor (R))

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

lemma uppR-miracle [wp]: false \(\wr_R(M) \land P = \tau\lor\)
by (simp add: wr-def)

lemma uppR-true [wp]: \(P \wr_R(M) \land \tau\lor = \tau\lor\)
by (simp add: wr-def)

lemma parallel-precondition-wr [rdes]:
assumes P is NSRD Q is NSRD M is RDM
shows \(\pre_R(P \parallel_{M(R)} Q) = (\per_R(Q) \land \wr_R(M) \land \pre_R(P) \land \post_R(Q) \land \wr_R(M) \land \pre_R(P) \land \per_R(Q) \land \wr_R(swap_m :: M) \land \pre_R(Q) \land \post_R(P) \land \wr_R(swap_m :: M) \land \pre_R(Q))\)
by (simp add: assms parallel-precondition-wr-def)

lemma parallel-rdes-def [rdes-def]:
assumes P₁ is RC P₂ is RR P₃ is RR Q₁ is RC Q₂ is RR Q₃ is RR
\(\$st' :: P₂ \parallel_{M(R)} Q₂\)
M is RDM
shows \(R_r((\tau\lor \parallel_{M(R)} \tau\lor) \parallel_{M(R)} (\tau\lor \parallel_{M(R)} \tau\lor)) = R_r((\tau\lor \parallel_{M(R)} (\tau\lor \parallel_{M(R)} \tau\lor)) \parallel_{M(R)} (\tau\lor \parallel_{M(R)} \tau\lor))\)
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 \(\parallel_{R_M} P = \text{Miracle}\)
proof
have \(\pre_R(\text{Miracle} \parallel_{R_M} P) = \tau\lor\)
by (simp add: parallel-assm wait'-cond-idem rdes closure assms)
mOREover hence \(\text{cmt}_R(\text{Miracle} \parallel_{R_M} P) = \text{false}\)
by (simp add: rdes closure wait'-cond-idem SRD-healths assms)
ultimately have \(\text{Miracle} \parallel_{R_M} P = R_r(\text{true}, \parallel\text{false})\)
by (metis NSRD-iff SRD-reactive-design-alt assms par-rdes-NSRD srdes-theory-continuous.weak.top-closed)
thus ?thesis
by (simp add: Miracle-def R1-design-R1-pre)
qed

lemma Miracle-parallel-right-zero:
assumes $P$ is SRD $M$ is RDM
shows $P \parallel_{RM} \text{Miracle} = \text{Miracle}$
proof –
  have $\text{pre}_R(P \parallel_{RM} \text{Miracle}) = \text{true}$,
    by (simp add: wait'-cond-idem parallel-asm rdes closure assms)
moreover hence $\text{cmt}_R(P \parallel_{RM} \text{Miracle}) = \text{false}$
    by (simp add: wait'-cond-idem rdes closure SRD-healths assms)
ultimately have $P \parallel_{RM} \text{Miracle} = R_s(\text{true}, \vdash \text{false})$
    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 :: (('s, 't::trace, unit) rsp) merge ($N_B$) where
  [upred-defs]: BasicMerge = ($\text{tr}_< \leq_u \text{tr}' \land \text{tr}' - \text{tr}_< =_u \text{tr}_0 - \text{tr} - \text{tr}_< \land \text{tr}' - \text{tr}_< =_u \text{tr}_1 - \text{tr} - \text{tr}_< \land \text{tr}' - \text{tr}_< =_u \text{tr}_2 - \text{tr} - \text{tr}_<)$

abbreviation rbasic-par ($\parallel$) - [85,86] 85 where
  $P \parallel_B Q \equiv P \parallel_{M_R(N_B)} Q$

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

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

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

8.2 Simple parallel composition

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

lemma RHS-design-par:
  assumes $\text{ok'} \not\subseteq P_1 \text{ \text{ok'} \not\subseteq P_2}$
  shows $R_s(P_1 \vdash Q_1) \parallel_R R_s(P_2 \vdash Q_2) = R_s((P_1 \land P_2) \vdash (Q_1 \land Q_2))$
proof –
  have $R_s(P_1 \vdash Q_1) \parallel_R R_s(P_2 \vdash Q_2) =$
\[ R_s(P_1 \parallel true, false/sok, $\text{\$wait}$) \parallel Q_1[true, false/sok, $\text{\$wait}$] ]
\[ R_s(P_2 \parallel true, false/sok, $\text{\$wait}$) \parallel Q_2[true, false/sok, $\text{\$wait}$] ]
by (simp add: RHS-design-ok-wait)

also from `assms`

\[ R_s((R1 \parallel R2c(P_1)) \parallel R1 (R2c(P_2))) \parallel (R1 (R2c(P_1) \Rightarrow Q_1)) \parallel R1 (R2c(P_2) \Rightarrow Q_2)) ]
by (simp add: R2c-R3h-commute R2c-and R2c-design R2c-idem R2c-not RHS-def)

also have ...
[ `assms` applies `simp add: `RHS-design-par unrest assms`
by (rule cong[ `R1-R2s-R2c` ])
also have ...
[ `assms` applies `simp add: rel-auto`
done

finally show `?thesis` .

```
qed
```

**lemma** `RHS-tri-design-par`:
\[ \text{assumes } R_s(P_1 \parallel Q_1 \parallel R_1) ]
\[ R_s(P_2 \parallel Q_2 \parallel R_2) ]
\[ R_s((P_1 \wedge P_2) \parallel (Q_1 \wedge Q_2) \parallel (R_1 \wedge R_2)) ]
by (simp add: RHS-design-par assms unrest wait `r'-cond-conj-exchange`

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

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

```
end
```

9 \textbf{Productive Reactive Designs}

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

9.1 \textbf{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 \Rightarrow ('s, 't, 'a) hrel-rsp` where
\[\text{upred-defs: Productive}(P) = P \parallel_R \text{R}_u(\text{true} \vdash \text{true} \odot (\text{str} <_u \text{str}^-))\]

**Lemma** Productive-RHS-design-form:
- ** Assumes** $\text{ok}' \not\in P$ $\text{ok}' \not\in Q$ $\text{ok}' \not\in R$
- ** Shows** Productive($\text{R}_u(P \parallel Q \circ R)$) = $\text{R}_u(P \parallel Q \circ (R \land \text{str} <_u \text{str}^-))$

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

**Lemma** Productive-form:
- ** Assumes** $P$ is SRD ($\text{str} <_u \text{str}^-) \subseteq (\text{pre}_R(P) \land \text{post}_R(P))$ $\text{wait}^- \not\in \text{pre}_R(P)$
- ** Shows** $P$ is Productive

**Proof**
- ** Have** Productive($\text{SRD}(P)$) = $\text{R}_u(\text{pre}_R(P) \vdash \text{peri}_R(P) \circ (\text{post}_R(P) \land \text{str} <_u \text{str}^-))$
  - ** By** (simp add: Productive-def SRD-as-reactive-tri-design)
  - ** Also have** $\cdots = \text{R}_u(\text{pre}_R(P) \vdash \text{peri}_R(P) \circ (\text{post}_R(P) \land \text{str} <_u \text{str}^-))$
  - ** 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 ($\text{str} <_u \text{str}^-) \subseteq (\text{pre}_R(P) \land \text{post}_R(P))$ $\text{wait}^- \not\in \text{pre}_R(P)$
- ** Shows** $P$ is Productive

**Proof**
- ** Have** $P.R_u(\text{pre}_R(P) \vdash \text{peri}_R(P) \circ (\text{post}_R(P) \land \text{str} <_u \text{str}^-)) = P$
- ** By** (metis (no-types, hide-lams) design-export-pre wait'-cond-conj-exchange wait'-cond-idem)
- ** Also have** $\cdots = \text{R}_u(\text{pre}_R(P) \vdash (\text{pre}_R(P) \land \text{peri}_R(P)) \circ (\text{pre}_R(P) \land \text{post}_R(P)))$
- ** By** (metis assms(2) utp-pred-laws.inf.absorb1 utp-pred-laws.inf.assoc)
- ** Also have** $\cdots = R_u(\text{pre}_R(P) \vdash \text{peri}_R(P) \circ (\text{post}_R(P) \land \text{str} <_u \text{str}^-))$
- ** 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-post-refines-tr-increase:
- ** Assumes** $P$ is SRD $\text{SRD}(P)$ is Productive $\text{wait}^- \not\in \text{pre}_R(P)$
- ** Shows** ($\text{str} <_u \text{str}^-) \subseteq (\text{pre}_R(P) \land \text{post}_R(P))$

**Proof**
- ** Have** $\text{post}_R(P) = \text{post}_R(R_u(\text{pre}_R(P) \vdash \text{peri}_R(P) \circ (\text{post}_R(P) \land \text{str} <_u \text{str}^-)))$
  - ** By** (metis Healthy-def Productive-form assms(1) assms(2))
  - ** Also have** $\cdots = R_1(\text{R}_2(\text{pre}_R(P) \Rightarrow (\text{post}_R(P) \land \text{str} <_u \text{str}^-)))$
  - ** By** (simp add: real-post-RHS-design unrest subst assms, rel-auto)
- ** Also have** $\cdots = R_1((\text{pre}_R(P) \Rightarrow (\text{post}_R(P) \land \text{str} <_u \text{str}^-)))$
  - ** By** (simp add: R2c-impl R2c-preR R2c-postR R2c-and R2c-tr-less-tr' assms)
- ** Also have** ($\text{str} <_u \text{str}^-) \subseteq (\text{pre}_R(P) \land \cdots)$
  - ** By** (rel-auto)
  - ** Finally show** ?thesis

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

9.2 Reactive design calculations

lemma preR-Productive [rdes]:
assumes P is SRD
shows preR(Productive(P)) = preR(P)
proof –
  have preR(Productive(P)) = preR(R,(preR(P) ⊢ periR(P) o (postR(P) ∧ $tr < u $tr′))
  by (metis Healthy-def Productive-form assms)
thus ?thesis
  by (simp add: rea-pre-RHS-design usbst unrest R2c-not R2c-preR R1-preR Healthy-if assms)
qed

lemma periR-Productive [rdes]:
assumes P is NSRD
shows periR(Productive(P)) = periR(P)
proof –
  have periR(Productive(P)) = periR(R,(preR(P) ⊢ periR(P) o (postR(P) ∧ $tr < u $tr′))
  by (metis Healthy-def NSRD-is-SRD Productive-form assms)
also have ... = R1 (R2c (preR P ⇒ periR P))
  by (simp add: rea-peri-RHS-design usbst unrest R2c-not assms closure)
also have ... = (preR P ⇒ periR P)
  by (simp add: R1-rea-impl R2c-rea-impl R2c-preR R2c-peri-SRD R1-peri-SRD assms closure R1-tr-less-tr′ R2c-tr-less-tr′)
finally show ?thesis
  by (simp add: SRD-peri-under-pre assms closure unrest)
qed

lemma postR-Productive [rdes]:
assumes P is NSRD
shows postR(Productive(P)) = (preR(P) ⇒ postR(P) ∧ $tr < u $tr′)
proof –
  have postR(Productive(P)) = postR(R,(preR(P) ⊢ periR(P) o (postR(P) ∧ $tr < u $tr′))
  by (metis Healthy-def NSRD-is-SRD Productive-form assms)
also have ... = R1 (R2c (preR P ⇒ postR P ∧ $tr′ ≥ u $tr))
  by (simp add: rea-post-RHS-design usbst unrest assms closure)
also have ... = (preR P ⇒ postR P ∧ $tr′ ≥ u $tr)
  by (simp add: R1-rea-impl R2c-rea-impl R2c-preR R2c-and R1-extend-conj′ R2c-post-SRD R1-post-SRD assms closure R1-tr-less-tr′ R2c-tr-less-tr′)
finally show ?thesis
  by (simp add: R1-preR Healthy-def assms)
qed

lemma preR-frame-seq-export:
assumes P is NSRD P is Productive Q is NSRD
shows (preR P ∧ (preR P ∧ postR P) :: Q) = (preR P ∧ (postR P :: Q))
proof –
  have (preR P ∧ (postR P :: Q)) = (preR P ∧ ((preR P ⇒ postR P) :: Q))
  by (simp add: SRD-post-under-pre assms closure unrest)
also have ... = (preR P ∧ ((¬ preR P) :: Q ∨ (preR P ⇒ R1(postR P) :: Q)))
  by (simp add: NSRD-is-SRD R1-post-SRD assms(1) rea-impl-def segr-or-distl R1-preR Healthy-if)
also have ... = (preR P ∧ ((¬ preR P) :: Q ∨ (preR P ∧ postR P) :: Q))
proof –
  have (preR P ∧ ¬ preR P) = R1 true
  by (simp add: R1-preR rea-not-or)

81
proof

lemma 9.3 Closure laws

qed

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

9.3 Closure laws

lemma Productive-rdes-intro:
  assumes (\$tr <u \$tr`) ⊆ R \$ok' \$ P \$ok' \$ Q \$ok' \$ R \$wait \$ P \$wait' \$ P
  shows (R_0(P ⊨ Q ⊨ R)) is Productive
proof (rule Productive-intro)
  show R_0 (P ⊨ Q ⊨ R) is SRD
  by (simp add: RHS-tri-design-is-SRD assms)

from assm1 show (\$tr' >u \$tr) ⊆ (preR (R_0 (P ⊨ Q ⊨ R)) ∧ postR (R_0 (P ⊨ Q ⊨ R)))
  apply (simp add: rea-pre-RHS-design rea-post-RHS-design usubst assm unrest)
  apply (rel-auto)
  apply fastforce
  done

show \$wait' \$ preR (R_0 (P ⊨ Q ⊨ R))
  by (simp add: rea-pre-RHS-design rea-post-RHS-design usubst R1-def R2c-def R2s-def assm unrest)
qed

We use the R'_4 healthiness condition to characterise that the postcondition must extend the trace for a reactive design to be productive.

lemma Productive-rdes-RR-intro:
  assumes P is RR Q is RR R is RR R is R'_4
  shows (R_0(P ⊨ Q ⊨ R)) is Productive
  using assm 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 ⊨ Q is Productive
proof
  have P ⊨ Q =
    R_0(preR(P) ⊨ periR(P) ∨ (postR(P) ∧ \$tr <u \$tr')) ∨ R_0(preR(Q) ⊨ periR(Q) ∨ (postR(Q) ∧ \$tr <u \$tr'))
  by (metis Healthy-if Productive-form assms)

  also have ... = R_0 ((preR P ∧ preR Q) ⊨ (periR P ∨ periR Q) ∨ ((postR P ∧ \$tr' >u \$tr) ∨ (postR Q ∧ \$tr' >u \$tr)))
lemma Productive-cond-rea [closure]:
assumes P is SRD P is Productive Q is SRD Q is Productive
shows P _ b > R Q is Productive
proof –
  have P _ b > R Q =
    R_s (pre_R(P) ⊆ peri_R(P) ∨ (post_R(P) ∧ $tr <_u $tr')) _ b > R R_s (pre_R(Q) ⊆ peri_R(Q) ∨ (post_R(Q) ∧ $tr <_u $tr'))
      by (metis Healthy-if Productive-form assms)
  also have ... = R_s ((pre_R P _ b > R pre_R Q) ⊆ (peri_R P _ b > R peri_R Q) ∨ ((post_R P _ b > R (post_R Q)) ∧ $tr >_u $tr'))
    by (simp add: cond-srea-form)
  also have ... is Productive
    by (simp add: Healthy-def Productive-RHS-design-form unrest)
finally show ?thesis .

qed

lemma Productive-seq-1 [closure]:
assumes P is NSRD P is Productive Q is NSRD
shows P ;: Q is Productive
proof –
  have P ;: Q = R_s (pre_R(P) ⊆ peri_R(P) ∨ (post_R(P) ∧ $tr <_u $tr')) ;: R_s (pre_R(Q) ⊆ peri_R(Q) ∨ (post_R(Q)))
    by (metis Healthy-def NSRD-is-SRD SRD-reactive-tri-design Productive-form assms(1) assms(2) assms(3))
  also have ... = R_s ((pre_R P ∧ (post_R P ∧ $tr' >_u $tr) wp R pre_R Q) ⊆
    (peri_R P ∨ ((post_R P ∧ $tr' >_u $tr) ;: peri_R Q)) ∨ ((post_R P ∧ $tr >_u $tr) ;: post_R Q))
    by (simp add: RHS-tri-design-composition-wp rpred unrest closure assms wp NSRD-neg-pre-left-zero SRD-healths ex-unrest wp-rea-def disj-apped-def)
  also have ... = R_s ((pre_R P ∧ (post_R P ∧ $tr >_u $tr) wp R pre_R Q) ⊆
    (peri_R P ∨ ((post_R P ∧ $tr' >_u $tr) ;: peri_R Q)) ∨ ((post_R P ∧ $tr >_u $tr) ;: post_R Q ∧ $tr >_u $tr))
proof –
  have ((post_R P ∧ $tr' >_u $tr) ;: R1(post_R Q) = ((post_R P ∧ $tr' >_u $tr) ;: R1(post_R Q) ∧ $tr' >_u $tr)
    by (rel-auto)
  thus ?thesis
    by (simp add: NSRD-is-SRD R1-post-SRD assms)
qed

also have ... is Productive
  by (rule Productive-rdes-intro, simp-all add: unrest assms closure wp-rea-def)
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 (pre_R(P) ⊢ peri_R(P) ∨ (post_R(P));; R_s (pre_R(Q) ⊢ peri_R(Q) ∨ (post_R(Q) ∧ $tr <_u \text{Str}$))

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

also have ... = R_s ((pre_R P ∧ post_R Q wp_p pre_R Q) ⊢ (peri_R P ∨ (post_R P ;; peri_R Q)) ∨ (post_R P ;; (post_R Q ∧ $\text{Str} \geq_u \text{Str}$)))

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

also have ... = R_s ((pre_R P ∧ post_R Q wp_p pre_R Q) ⊢ (peri_R P ∨ (post_R P ;; peri_R Q)) ∨ (post_R P ;; (post_R Q ∧ $\text{Str} \geq_u \text{Str}$) ∧ $\text{Str} >_u \text{Str}$))

proof –

have (R1(post_R P) ;; (post_R Q ∧ $\text{Str} \geq_u \text{Str}$) ∧ $\text{Str} >_u \text{Str}$) = (R1(post_R P) ;; (post_R Q ∧ $\text{Str} >_u \text{Str}$))

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-rea-def)

finally show ?thesis .

qed

dem

t

e

10 Guarded Recursion

t
	e

10.1 Traces with a size measure

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

class size-trace = trace + size +

assumes

size-zero: size 0 = 0 and

size-nszero: s > 0 ⇒ size(s) > 0 and

size-plus: size (s + t) = size(s) + size(t)

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

begin

lemma size-mono: s ≤ t ⇒ size(s) ≤ size(t)

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

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

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

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

end

Both natural numbers and lists are measurable trace algebras.

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

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

syntax
-upsize :: logic \Rightarrow logic (size_u(\cdot))

translations
size_u(t) == CONST uop CONST size t

10.2 Guardedness

definition gvrt :: (('t::size-trace,α) rp \times ('t,α) rp) chain where
[upred-defs]: \( gvrt(n) \equiv (\forall tr \leq u. \forall tr' \land size_u(\& tt) < u < n) \)

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

lemma gvrt-limit: \( \bigwedge (range gvrt) = (\forall tr \leq u. tr') \)
by (rel-auto)

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

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

Guarded reactive designs yield unique fixed-points.

theorem guarded-fp-uniq:
assumes mono F F \in \[id\]_H \rightarrow \[SRD\]_H Guarded F
shows \( \mu F = \nu F \)
proof -
  have constr F 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. $tr' \land \nu F)
  apply (rule constr-fp-uniq)
  apply (simp add: assms)
  done
using gert-limit apply blast
done
moreover have ($\text{tr} \leq_u \text{tr}' \land F) = \mu F
proof
\begin{itemize}
  \item have $\mu F$ is $R1$
    \begin{itemize}
      \item by (rule SRD-healths(1), rule Healthy-mu, simp-all add: assms)
      \item thus $\nu$thesis
        \begin{itemize}
          \item by (metis Healthy-def R1-def conj-comm)
        \end{itemize}
    \end{itemize}
\end{itemize}
qed
moreover have ($\text{tr} \leq_u \text{tr}' \land F) = \nu F
proof
\begin{itemize}
  \item have $\nu F$ is $R1$
    \begin{itemize}
      \item by (rule SRD-healths(1), rule Healthy-nu, simp-all add: assms)
      \item thus $\nu$thesis
        \begin{itemize}
          \item by (metis Healthy-def R1-def conj-comm)
        \end{itemize}
    \end{itemize}
\end{itemize}
qed
ultimately show $\nu$thesis
\begin{itemize}
  \item by (simp)
\end{itemize}
qed

lemma Guarded-const [closure]: Guarded ($\lambda X. P$)
\begin{itemize}
  \item by (simp add: Guarded-def)
\end{itemize}

lemma UINF-Guarded [closure]:
\begin{itemize}
  \item assumes $\forall P. P \in A \Rightarrow \text{Guarded } P$
  \item shows $\text{Guarded} (\lambda X. \bigwedge P \in A. P(X))$
  \item proof (rule GuardedI)
    \begin{itemize}
      \item fix $n$
      \item have $\forall Y. (\bigwedge P \in A. P Y) \land\text{gert}(n+1)) = (\bigwedge P \in A. (P Y \land\text{gert}(n+1))) \land\text{gert}(n+1))$
      \item proof
        \begin{itemize}
          \item fix $Y$
          \item let $?lhs = ((\bigwedge P \in A. P Y) \land\text{gert}(n+1))$ and $?rhs = ((\bigwedge P \in A. (P Y \land\text{gert}(n+1))) \land\text{gert}(n+1))$
          \item have $a$:$?lhs[\text{false}/\text{ok}] = ?rhs[\text{false}/\text{ok}]$
            \begin{itemize}
              \item by (rel-auto)
            \end{itemize}
          \item have $b$:$?lhs[\text{true}/\text{ok}][\text{true}/\text{wait}] = ?rhs[\text{true}/\text{ok}][\text{true}/\text{wait}]$
            \begin{itemize}
              \item by (rel-auto)
            \end{itemize}
          \item have $c$:$?lhs[\text{true}/\text{ok}][\text{false}/\text{wait}] = ?rhs[\text{true}/\text{ok}][\text{false}/\text{wait}]$
            \begin{itemize}
              \item by (rel-auto)
            \end{itemize}
          \item show $?lhs = ?rhs$
            \begin{itemize}
              \item using $a$, $b$, $c$
                \begin{itemize}
                  \item by (rule-tac bool-eq-splitI[of in-var ok], simp, rule-tac bool-eq-splitI[of in-var wait], simp-all)
                \end{itemize}
            \end{itemize}
        \end{itemize}
    \end{itemize}
  \end{itemize}
\end{itemize}
qed
moreover have $((\bigwedge P \in A. (P X \land\text{gert}(n+1))) \land\text{gert}(n+1)) = ((\bigwedge P \in A. (P (X \land\text{gert}(n)) \land\text{gert}(n+1)))) \land\text{gert}(n+1))$
proof
\begin{itemize}
  \item have $((\bigwedge P \in A. (P X \land\text{gert}(n+1))) = ((\bigwedge P \in A. (P (X \land\text{gert}(n)) \land\text{gert}(n+1)))$
  \item proof (rule UINF-cong)
    \begin{itemize}
      \item fix $P$ assume $P \in A$
      \item thus $(P X \land\text{gert}(n+1)) = (P (X \land\text{gert}(n)) \land\text{gert}(n+1))$
        \begin{itemize}
          \item using Guarded-def assms by blast
        \end{itemize}
    \end{itemize}
  \end{itemize}
\end{itemize}
qed
ultimately show $((\bigwedge P \in A. P X) \land\text{gert}(n+1)) = ((\bigwedge P \in A. (P (X \land\text{gert}(n))) \land\text{gert}(n+1))$
\begin{itemize}
  \item by simp
\end{itemize}

86
lemma intChoice-Guarded [closure]:
  assumes Guarded P Guarded Q
  shows Guarded (\lambda X. P(X) \cap Q(X))
proof –
  have Guarded (\lambda X. \forall F \in \{P,Q\} \cdot F(X))
    by (rule UNINF-Guarded, auto simp add: assms)
  thus \?thesis
  by (simp)
qed

lemma cond-srea-Guarded [closure]:
  assumes Guarded P Guarded Q
  shows Guarded (\lambda X. P(X) \land b \in R Q(X))
using assms by (rel-auto)

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

lemma Guarded-if-Productive [closure]:
  fixes P :: (\'s, \'t::size-trace,\'a) hrel-rsp
  assumes P is NSRD P is Productive
  shows Guarded (\lambda X. P :: SRD(X))
proof (clarsimp simp add: Guarded-def)
  — We split the proof into three cases corresponding to valuations for ok, wait, and wait’ respectively.
  fix X n
  have a: (P :: SRD(X) \land gVRT (Suc n)) [false/ok] =
    (P :: SRD(X) \land gVRT (Suc n)) [false/ok]
    by (simp add: subst closure SRD-left-zero-1 assms)
  have b: ((P :: SRD(X) \land gVRT (Suc n)) [true/ok] [true/wait] =
    (P :: SRD(X) \land gVRT (Suc n)) [true/ok] [true/wait]
    by (simp add: subst closure SRD-left-zero-2 assms)
  have c: ((P :: SRD(X) \land gVRT (Suc n)) [true/ok] [false/wait] =
    (P :: SRD(X) \land gVRT (Suc n)) [true/ok] [false/wait]
    by (clarsimp simp add: Guarded-def)
proof –
  have 1: (P [true/wait] :: (SRD X) [true/wait] \land gVRT (Suc n)) [true/false/ok,\$\text{wait}]
    =
    (P [true/wait] :: (SRD (X \land gVRT n)) [true/false/ok,\$\text{wait}] \land gVRT (Suc n)) [true/false/ok,\$\text{wait}]
    by (metis (no-types, lifting) Healthy-def R\text{h}-wait-true SRD-healths(\beta) SRD-idem)
  have 2: (P [false/wait] :: (SRD X) [false/wait] \land gVRT (Suc n)) [true/false/ok,\$\text{wait}]
    =
    (P [false/wait] :: (SRD (X \land gVRT n)) [false/wait] \land gVRT (Suc n)) [true/false/ok,\$\text{wait}]
    by (clarsimp simp add: Healthy-def Productive-form assms(1) assms(2) NSRD-is-SRD)
  also have ... =
    (((\neg E \text{pre}_R P) :: (SRD (Y)) [false/wait] \land gVRT (Suc n)) [true/false/ok,\$\text{wait}]
    \land gVRT (Suc n)) [true/false/ok,\$\text{wait}]
  proof –
    fix Y :: (\'s, \'t,\'a) hrel-rsp
    have (P [false/\text{wait}] :: (SRD Y) [false/\text{wait}] \land gVRT (Suc n)) [true/false/ok,\$\text{wait}]
      =
      (((\neg E \text{pre}_R P) :: (SRD (Y)) [false/\text{wait}]) \land gVRT (Suc n)) [true/false/ok,\$\text{wait}]
      by (metis (no-types) Healthy-def Productive-form assms(1) assms(2) NSRD-is-SRD)
  qed
by (simp add: RHS-def R1-def R2c-def R2s-def R3h-def RD1-def RD2-def usbst unrest assms closure design-def)
also have ... =
  (((\gamma \ pre_R(P) \lor (\$ok' \land pre_R(P) \land \$tr < _u \$tr')))(false/$wait') \land (SRD Y)[false/$wait] \\
  \land gvt (Suc n)[true,false/$ok,$wait]
by (simp add: impl-af-def R2c-disj R1-disj R2c-not assms closure R2c-and R2c-preR rea-not-def R1-extend-conv' R2c-ok' R2c-post-SRD R1-tr-less-tr' R2c-tr-less-tr')
also have ... =
  (((\gamma \ pre_R(P) \lor (SRD(Y))[false/$wait] \lor (\$ok' \land post_R(P \land \$tr' > _u \$tr)) \lor \$ok' \land true) \lor (SRD Y)[false/$wait]) \land gvt (Suc n)[true,false/$ok,$wait]
by (simp add: usbst unrest assms closure seq-or-distl NSRD-neg-pre-left-zero SRD-healths)
also have ... =
  (((((\gamma \ pre_R(P) \lor (SRD(Y))[false/$wait] \lor (post_R(P \land \$tr' > _u \$tr)) \lor (SRD Y)[true,false/$ok,$wait]))) \\
  \land gvt (Suc n)[true,false/$ok,$wait])
proof
  have (\$ok' \land post_R(P \land \$tr' > _u \$tr)) \lor (SRD Y)[false/$wait] =
   \((\gamma \ pre_R(P)) \lor (SRD Y)[false/$wait]\lor \$ok' \land true) \lor (SRD Y)[false/$wait]
  by (rel-blast)
also have ... = (\gamma \ pre_R(P) \land \$tr' > _u \$tr)[true/$ok'] \lor (SRD(Y))[false/$wait][true/$ok] \\
  \lor using seqr-left-one-point[of ok (\gamma \ pre_R(P) \land \$tr' > _u \$tr) Tru (SRD Y)[false/$wait]]
  by (simp add: true-af-def[THEN sym])
finally show \\thesis by (simp add: usbst unrest)
qed
finally
show (P[false/$wait'] \land (SRD Y)[false/$wait] \land gvt (Suc n)[true,false/$ok,$wait]
  =
  (((\gamma \ pre_R(P)) \land (SRD(Y))[false/$wait] \lor (\gamma \ pre_R(P)) \lor (SRD Y)[true,false/$ok,$wait] \\
  \land gvt (Suc n))[true,false/$ok,$wait])
  \land gvt (Suc n)[true,false/$ok,$wait]
. qed

have 1:\((\gamma \ pre_R(P) \land \$tr' > _u \$tr)) \lor (SRD X)[true,false/$ok,$wait] \land gvt (Suc n) =
  \((\gamma \ pre_R(P)) \land (SRD X)[true,false/$ok,$wait] \lor (\gamma \ pre_R(P)) \lor (SRD X)[true,false/$ok,$wait]) \\
  \land gvt (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:\((\gamma \ pre_R(P)) \lor (SRD X)[false/$wait] = (\gamma \ pre_R(P)) \lor (SRD(X \land gvt n))[false/$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

show ?thesis
proof –
  have \((P ;; (SRD X) \land gvr\ (n+1)) [true/false/\$ok,\$wait]\) = 
    \((P[true/\$wait] ;; (SRD X)) [true/\$wait] \land gvr\ (n+1)) [true/false/\$ok,\$wait] \lor 
    \((P[false/\$wait] ;; (SRD X)) [false/\$wait] \land gvr\ (n+1)) [true/false/\$ok,\$wait])
  by (subst seqr-bool-split[of wait], simp-all add: usubst utp-pred-laws.distrib(4))

also
  have \(\ldots = ((P[true/\$wait] ;; (SRD X \land gvr\ n)) [true/\$wait] \land gvr\ (n+1)) [true/false/\$ok,\$wait]\)
    \lor 
    \((P[false/\$wait] ;; (SRD X \land gvr\ n)) [false/\$wait] \land gvr\ (n+1)) [true/false/\$ok,\$wait])
  by (simp add: 1 2)

also
  have \(\ldots = ((P[true/\$wait] ;; (SRD X \land gvr\ n)) [true/\$wait] \land gvr\ (n+1)) [true/false/\$ok,\$wait]\)
    \lor 
    \((P[false/\$wait] ;; (SRD X \land gvr\ n)) [false/\$wait] \land gvr\ (n+1)) [true/false/\$ok,\$wait])
  by (simp add: usubst utp-pred-laws.distrib(4))

also have \(\ldots = (P ;; (SRD X \land gvr\ n)) \land gvr\ (n+1)) [true/false/\$ok,\$wait]\)
  by (subst seqr-bool-split[of wait], simp-all add: usubst)

finally show ?thesis by (simp add: usubst)
qed

qed

show \((P ;; SRD(X) \land gvr\ (Suc\ n)) = (P ;; SRD(X) \land gvr\ (Suc\ n))\)
apply \((rule-tac bool-eq-splitI[of in-var ok])\)
apply \((simp-all add: a)\)
apply \((rule-tac bool-eq-splitI[of in-var wait])\)
apply \((simp-all add: b c)\)
done

qed

10.3 Tail recursive fixed-point calculations

declare upred-semiring.power-Suc [simp]

lemma mu-csp-form-1 [rdes]:
  fixes \(P :: (\text{'s, 't::size-trace, 'a) brel-rsp\)
  assumes \(P\) is NSRD \(P\) is Productive
  shows \(\mu X \cdot P ;; SRD(X) = (\bigcap i \cdot P ^\ast (i+1)) ;; \text{Miracle}\)
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 \subseteq \text{spec} \text{ simp})
    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 \(\ldots = ((\bigcap i \cdot ((\lambda X. \ P :: SRD\ X) ^\ast i) false)\)
    by (simp add: sup-continuous-lfp 1 sup-continuous-Continuous false-upred-def)
also have \(\ldots = ((\bigcap i \cdot ((\lambda X. \ P :: SRD\ X) ^\ast (i+1)) false)\)
    by (subst Sup-power-expand, simp)

89
also have \( \ldots = (\prod i. (\lambda X. P ;; \text{SRD } X) ^\cdot (i+1)) \text{ false} \)
by (simp)
also have \( \ldots = (\prod i. P ^\cdot (i+1)) ;; \text{ Miracle} \)
proof (rule SUP-cong, simp-all)
  fix \( i \)
  show \( P ;; \text{SRD } (((\lambda X. P ;; \text{SRD } X) ^\cdot i) \text{ false}) = (P ;; P ^\cdot i) ;; \text{ Miracle} \)
proof (induct \( i \))
    case 0
    then show ?case
    by (simp, metis srdes-hcond-def srdes-theory-continuous.healthy-top)
  next
    case (Suc \( i \))
    then show ?case
    by (simp add: Healthy-if NSRD-is-SRD SRD-power-comp SRD-seqr-closure assms seqr-assoc THEN sym srdes-theory-continuous.weak.top-closed)
  qed
qed
also have \( \ldots = (\prod i. P ^\cdot (i+1)) ;; \text{ Miracle} \)
by (simp add: seq-Sup-distr)
finally show ?thesis
by (simp add: UINF-as-Sup[THEN sym])
qed

lemma mu-csp-form-NSRD [closure]:
  fixes \( P :: (\alpha \times \text{size-trace},\alpha) \text{ hrel-rsp} \)
  assumes \( P \text{ is NSRD } P \text{ is Productive} \)
  shows \((\mu X. P ;; \text{SRD}(X)) \text{ is NSRD} \)
by (simp add: mu-csp-form-1 assms closure)

lemma mu-csp-form-1':
  fixes \( P :: (\alpha \times \text{size-trace},\alpha) \text{ hrel-rsp} \)
  assumes \( P \text{ is NSRD } P \text{ is Productive} \)
  shows \((\mu X. P ;; \text{SRD}(X)) = (P ;; P^\cdot) ;; \text{ Miracle} \)
proof
  have \((\mu X. P ;; \text{SRD}(X)) = (\prod i \in \text{UNIV} \cdot P ;; P ^\cdot i) ;; \text{ Miracle} \)
  by (simp add: mu-csp-form-1 assms closure ustar-def)
  also have \( \ldots = (P ;; P^\cdot) ;; \text{ Miracle} \)
  by (simp only: seq-UINF-distl[THEN sym], simp add: ustar-def)
  finally show ?thesis .
qed

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

end

11 Reactive Design Programs

theory utp-rdes-prog
imports
  utp-rdes-normal
  utp-rdes-tactics
  utp-rdes-parallel
  utp-rdes-guarded
  UTP−KAT.utp-kleene
begin
11.1 State substitution

**lemma** srd-subst-RHS-tri-design [usubst]:
\[ [\sigma]_{S}\uparrow R_{s}(P \vdash Q \circ R) = R_{s}([\sigma]_{S}\uparrow P) \vdash ([\sigma]_{S}\uparrow Q) \circ ([\sigma]_{S}\uparrow R) \]
by (rel-auto)

**lemma** srd-subst-SRD-closed [closure]:
assumes \( P \) is SRD
shows \([\sigma]_{S}\uparrow P \) is SRD
proof -
have SRD([\sigma]_{S}\uparrow (SRD P)) = [\sigma]_{S}\uparrow (SRD P)
by (rel-auto)
thus ?thesis
by (metis Healthy-def assms)
qed

**lemma** preR-srd-subst [rdes]:
\pre_{R}(\textcircled{\[\sigma\]}S_{\uparrow}P) = \textcircled{\[\sigma\]}S_{\uparrow}\pre_{R}(P)
by (rel-auto)

**lemma** periR-srd-subst [rdes]:
\peri_{R}(\textcircled{\[\sigma\]}S_{\uparrow}P) = \textcircled{\[\sigma\]}S_{\uparrow}\peri_{R}(P)
by (rel-auto)

**lemma** postR-srd-subst [rdes]:
\post_{R}(\textcircled{\[\sigma\]}S_{\uparrow}P) = \textcircled{\[\sigma\]}S_{\uparrow}\post_{R}(P)
by (rel-auto)

**lemma** srd-subst-NSRD-closed [closure]:
assumes \( P \) is NSRD
shows \([\sigma]_{S}\uparrow P \) is NSRD
by (rule NSRD-RC-intro, simp-all add: closure rdes assms unrest)

11.2 Assignment

**definition** assigns-srd :: \( \text{'s usubst} \Rightarrow (\text{'t::trace, 'a) hrel-rsp \(\langle\cdot,\rangle_{R}\) where}
[upred-defs]: assigns-srd \( \sigma = R_{s}(\text{true} \vdash \text{false} \circ \text{\langle\sigma\rangle}_{\text{S}} \wedge \text{\Sigma}_{S} = \text{\Sigma}_{\text{S}})\)

**syntax**
\( \langle-\rangle := R \ (\cdot)\)

**translations**
\( \langle-\rangle \) svids \( \Rightarrow \) uexprs \( \Rightarrow \) logic \(\langle-\rangle := R \ (\cdot)\)

**lemma** assigns-srd-RHS-tri-des [rdes-def]:
\( \langle\sigma\rangle_{R} = R_{s}(\text{true} \vdash \text{false} \circ \langle\sigma\rangle_{\text{R}})\)
by (rel-auto)

**lemma** assigns-srd-NSRD-closed [closure]: \(\langle\sigma\rangle_{R} \) is NSRD
by (simp add: rdes-def closure unrest)
lemma \( \text{preR-assigns-srd [rdes]} \): \( \text{pre}_R(\langle \sigma \rangle_R) = \text{true} \)
by (simp add: rdes-def rdes closure)

lemma \( \text{periR-assigns-srd [rdes]} \): \( \text{peri}_R(\langle \sigma \rangle_R) = \text{false} \)
by (simp add: rdes-def rdes closure)

lemma \( \text{postR-assigns-srd [rdes]} \): \( \text{post}_R(\langle \sigma \rangle_R) = \langle \sigma \rangle_r \)
by (simp add: rdes-def rdes closure rpred)

11.3 Conditional

lemma \( \text{preR-cond-srea [rdes]} \):
\( \text{pre}_R(P \triangleleft b \triangleright R Q) = ([b]_{S<} \land \text{pre}_R(P) \lor \neg [b]_{S<} \land \text{peri}_R(Q)) \)
by (rel-auto)

lemma \( \text{periR-cond-srea [rdes]} \):
assumes \( P \text{ is SRD} \) \( Q \text{ is SRD} \)
shows \( \text{peri}_R(P \triangleleft b \triangleright R Q) = ([b]_{S<} \land \text{peri}_R(P) \lor \neg [b]_{S<} \land \text{peri}_R(Q)) \)
proof -
have \( \text{peri}_R(P \triangleleft b \triangleright R Q) = \text{peri}_R(R1(P) \triangleleft b \triangleright R1(Q)) \)
by (simp add: Healthy-if SRD-healths assms)
thus \( ?\text{thesis} \)
by (rel-auto)
qed

lemma \( \text{postR-cond-srea [rdes]} \):
assumes \( P \text{ is SRD} \) \( Q \text{ is SRD} \)
shows \( \text{post}_R(P \triangleleft b \triangleright R Q) = ([b]_{S<} \land \text{post}_R(P) \lor \neg [b]_{S<} \land \text{post}_R(Q)) \)
proof -
have \( \text{post}_R(P \triangleleft b \triangleright R Q) = \text{post}_R(R1(P) \triangleleft b \triangleright R1(Q)) \)
by (simp add: Healthy-if SRD-healths assms)
thus \( ?\text{thesis} \)
by (rel-auto)
qed

lemma \( \text{NSRD-cond-srea [closure]} \):
assumes \( P \text{ is NSRD} \) \( Q \text{ is NSRD} \)
shows \( P \triangleleft b \triangleright R Q \text{ is NSRD} \)
proof (rule NSRD-RC-intro)
show \( P \triangleleft b \triangleright R Q \text{ is SRD} \)
by (simp add: closure assms)
show \( \text{pre}_R(P \triangleleft b \triangleright R Q) \text{ is RC} \)
proof -
have 1: ([\neg b]_{S<} \lor \neg \text{pre}_R(P)) :: R1(true) = ([\neg b]_{S<} \lor \neg \text{pre}_R(P))
by (metis (no-types, lifting) NSRD-neg-pre-unit acex-not assms1 seq-or-distl st-lift-R1-true-right)
have 2: ([b]_{S<} \lor \neg \text{pre}_R(Q)) :: R1(true) = ([b]_{S<} \lor \neg \text{pre}_R(Q))
by (simp add: NSRD-neg-pre-unit assms seq-or-distl st-lift-R1-true-right)
show \( ?\text{thesis} \)
by (simp add: rdes closure assms)
qed

show \( \text{st ' \# peri}_R(P \triangleleft b \triangleright R Q) \)
by (simp add: rdes assms closure unrest)
qed
11.4 Assumptions

definition AssumeR :: 's cond ⇒ ('s::trace, 't::trace) hrel-rsp ([|] _ _ R) where
[upred-defs]: AssumeR b = II _ < b ∨ R Miracle

lemma AssumeR-rdes-def [rdes-def]:
[|b|] _ _ R = R_a(true_r ⊢ false ◦ [|b|] _ _ R)
unfolding AssumeR-def by (rdes-eq)

lemma AssumeR-NSRD [closure]: [|b|] _ _ R is NSRD
by (simp add: AssumeR-def closure)

lemma AssumeR-false: [|false|] _ _ R = Miracle
by (rel-auto)

lemma AssumeR-true: [|true|] _ _ R = II _
by (rel-auto)

lemma AssumeR-comp: [|b|] _ _ R ;; [|c|] _ _ R = [|b ∧ c|] _ _ R
by (rdes-simp)

lemma AssumeR-choice: [|b|] _ _ R ∧ [|c|] _ _ R = [|b ∨ c|] _ _ R
by (rdes-eq)

lemma AssumeR-refine-skip: II _ ⊑ [|b|] _ _ R
by (rdes-refine)

lemma AssumeR-test [closure]: test _ [|b|] _ _ R
by (simp add: AssumeR-refine-skip nsrd-thy.utest-intro)

lemma Star-AssumeR: [|b|] _ _ R⋆R = II _
by (simp add: AssumeR-NSRD AssumeR-test nsrd-thy.Star-test)

lemma cond-srea-AssumeR-form:
assumes P is NSRD Q is NSRD
shows P ◦ b ∨ R Q = ([|b|] _ _ R ;; P ∩ [¬b|] _ _ R ;; Q)
by (rdes-eq cls: assms)

lemma cond-srea-insert-assume:
assumes P is NSRD Q is NSRD
shows P ◦ b ∨ R Q = ([|b|] _ _ R ;; P ◦ b ∨ R [¬b|] _ _ R ;; Q)
by (simp add: AssumeR-NSRD AssumeR-comp NSRD-seqr-closure RA1 assms cond-srea-AssumeR-form)

lemma AssumeR-cond-left:
assumes P is NSRD Q is NSRD
shows [|b|] _ _ R ;; (P ◦ b ∨ R Q) = ([|b|] _ _ R ;; P)
by (rdes-eq cls: assms)

lemma AssumeR-cond-right:
assumes P is NSRD Q is NSRD
shows [¬b] _ _ R ;; (P ◦ b ∨ R Q) = ([¬b|] _ _ R ;; Q)
by (rdes-eq cls: assms)
11.5 Guarded commands

definition GuardedCommR :: 's cond ⇒ ('s, 't::trace, 'α) hrel-rsp ⇒ ('s, 't, 'α) hrel-rsp (→R - [85, 86]) where
gcmd-def[rdes-def]: GuardedCommR g A = A ⪰ g ⪰R Miracle

lemma gcmd-false[simp]: (false →R A) = Miracle
  unfolding gcmd-def by (pred-auto)

lemma gcmd-true[simp]: (true →R A) = A
  unfolding gcmd-def by (pred-auto)

lemma gcmd-SRD: assumes A is SRD shows (g →R A) is SRD
  by (simp add: gcmd-def SRD-cond-srea assms srdes-theory-continuous,weak,top-closed)

lemma gcmd-NSRD [closure]: assumes A is NSRD shows (g →R A) is NSRD
  by (simp add: gcmd-def NSRD-cond-srea assms NSRD-Miracle)

lemma gcmd-Productive [closure]: assumes A is NSRD A is Productive shows (g →R A) is Productive
  by (simp add: gcmd-def closure assms)

lemma gcmd-seq-distr: assumes B is NSRD shows (g →R A) ;; B = (g →R A) ;; (g →R B)
  by (simp add: Miracle-left-zero NSRD-is-SRD assms cond-st-distr gcmd-def)

lemma gcmd-nondet-distr: assumes A is NSRD B is NSRD shows (g →R (A ⊓ B)) = (g →R A) ⊓ (g →R B)
  by (rdes-eq cls: assms)

lemma AssumeR-as-gcmd: [b] R = b →R II R
  by (rdes-eq)

11.6 Generalised Alternation

definition AlternateR :: 'a set ⇒ ('a ⇒ 's upred) ⇒ ('a ⇒ ('s, 't::trace, 'α) hrel-rsp) ⇒ ('s, 't, 'α) hrel-rsp ⇒ ('s, 't, 'α) hrel-rsp
  where [upred-defs, rdes-def]: AlternateR I g A B = (Π i ∈ I · (g i) →R (A i)) ⊔ (¬ (∀ i ∈ I · g i) →R B)

definition AlternateR-list :: ('s upred × ('s, 't::trace, 'α) hrel-rsp) list ⇒ ('s, 't, 'α) hrel-rsp ⇒ ('s, 't, 'α) hrel-rsp
  where [upred-defs, ndes-simp]: AlternateR-list xs P = AlternateR [0..<length xs] (λ i. map fst xs ! i) (λ i. map snd xs ! i) P

syntax
  -altindR-els :: pttrn ⇒ logic ⇒ logic ⇒ logic ⇒ logic ⇒ logic (if R ∈· - · - → - else - f)
-altindR :: pttmR ⇒ logic ⇒ logic ⇒ logic ⇒ logic (if \( R \in \cdot \cdot \cdot \to \cdot \))

-altgcommR-els :: gcomms ⇒ logic ⇒ logic ⇒ logic ⇒ logic (if \( R/ \cdot \cdot \cdot -/ \))

-altgcommR :: gcomms ⇒ logic (if \( R/ \cdot \))

translations
if \( R i \in I \cdot g \to A \) else \( B \) \fi  \rightarrow CONSTR AlternateR I (λi. g) (λi. A) B
if \( R i \in I \cdot g \to A \) \fi  \rightarrow CONSTR AlternateR I (λi. g) (λi. A) (CONST Chaos)
if \( R (g i) \to A \) else \( B \) \fi  \rightarrow CONSTR AlternateR I g (λi. A) B
-altgcommR cs \rightarrow CONSTR AlternateR-list cs (CONST Chaos)
-altgcommR-els \( cs \) \leftarrow CONSTR AlternateR-list cs (CONST Chaos)
-altgcommR-els \( cs \) \rightarrow CONSTR AlternateR-list cs P
-altgcommR-els \( -(gcomm-show \ cs) \) \leftarrow CONSTR AlternateR-list cs P

lemma AlternateR-NSRD-closed [closure]:
assumes \( \forall i. i \in I \implies A \) is NSRD B is NSRD
shows \( \text{(cases } I = \{\}) \)
proof (cases \( I = \{\} \))
  case True
  then show \(?thesis\) by (simp add: AlternateR-def assms)
next
  case False
  then show \(?thesis\) by (simp add: AlternateR-def closure assms)
qed

lemma AlternateR-empty [simp]:
\( (if \( R i \in \{\} \cdot g \to A \) i else \( B \) \fi) = B \)
by (rdes-simp)

lemma AlternateR-Productive [closure]:
assumes \( \forall i. i \in I \implies A \) is NSRD B is NSRD
\( \forall i. i \in I \implies A \) is Productive B is Productive
shows \( \text{(cases } I = \{\}) \)
proof (cases \( I = \{\} \))
  case True
  then show \(?thesis\)
  by (simp add: assms(4))
next
  case False
  then show \(?thesis\)
  by (simp add: AlternateR-def closure assms)
qed

lemma AlternateR-singleton:
assumes \( A \) \( k \) is NSRD B is NSRD
shows \( (if \( R i \in \{k\} \cdot g \to A \) i else \( B \) \fi) = (A(k) \triangle g(k) \triangleright_R B) \)
by (simp add: AlternateR-def, rdes-eq cls: assms)

Convert an alternation over disjoint guards into a cascading if-then-else

lemma AlternateR-insert-cascade:
assumes \( \forall i. i \in I \implies A \) is NSRD
\( A \) \( k \) is NSRD B is NSRD
\( (g(k) \land (\forall i \in I \cdot g(i))) = \text{false} \)
shows \((\text{if } R \ i \in \text{insert } k \ I \cdot \ g \ i \to \ A \ i \ \text{else } \ B \ \text{fi}) = (A(k) \triangle g(k) \triangleright_{R} \ (\text{if } R \ i \in \ I \cdot \ g \ (i) \to \ A(i) \ \text{else } \ B \ \text{fi}))\)

proof \((\text{cases } I = \{\})\)

case True
then show ?thesis by \((\text{simp add: AlternateR-singleton assms})\)

next

case False
have 1: \((\prod i \cdot g \ i \to_{R} A i) = (\prod i \cdot g \ i \to_{R} \ R_{s}(\text{pre}_{R}(A i) \vdash \text{peri}_{R}(A i) \circ \text{post}_{R}(A i)))\)
by \((\text{simp add: NSRD-is-SRD SRD-reactive-tri-design assms(1)} \ \text{cong: UINF-cong})\)

from assms \((\text{4})\) show ?thesis
by \((\text{simp add: AlternateR-def 1 False cong: UINF-cong})\)

qed

11.7 Choose

definition choose-srd :: \((\ 's \cdot 't::trace \cdot \ 'a) \ hrel-rsp \ (\text{choose}_{R})\ \text{where}\)
\([\text{upred-defs, rdes-def}]\): \(\text{choose}_{R} = R_{s}(\text{true}_{r} \vdash \text{false} \circ \text{true}_{r})\)

lemma \(\text{preR-choose \ [rdes]}\): \(\text{pre}_{R}(\text{choose}_{R}) = \text{true}_{r}\)
by \((\text{rel-auto})\)

lemma \(\text{periR-choose \ [rdes]}\): \(\text{peri}_{R}(\text{choose}_{R}) = \text{false}\)
by \((\text{rel-auto})\)

lemma \(\text{postR-choose \ [rdes]}\): \(\text{post}_{R}(\text{choose}_{R}) = \text{true}_{r}\)
by \((\text{rel-auto})\)

lemma \(\text{choose-srd-SRD \ [closure]}\): \(\text{choose}_{R} \text{ is SRD}\)
by \((\text{simp add: choose-srd-def closure unrest})\)

lemma \(\text{NSRD-choose-srd \ [closure]}\): \(\text{choose}_{R} \text{ is NSRD}\)
by \((\text{rule NSRD-intro, simp-all add: closure unrest rdes})\)

11.8 State Abstraction

definition state-srea :: \(\ 's \cdot \ 'a \cdot P \Rightarrow \ 'a \cdot P \Rightarrow \ 's \cdot \ 't\cdot trace \cdot \ 'a \cdot \ 'b \ \text{rel-rsp} \Rightarrow \ (\text{unit}, \ 't, \ 'a, \ 'b) \ \text{rel-rsp} \ \text{where}\)
\([\text{upred-defs}]\): \(\text{state-srea} t \ P = (\exists \ (\$st,\$st') \cdot P)_{S}\)

syntax
\(-\text{state-srea} :: \text{type} \Rightarrow \text{logic} \Rightarrow \text{logic} \ (\text{state} \cdot \cdot \ [0,200])\ 200\)

translations
\(-\text{state}'a \cdot P == \text{CONST state-srea TYPE}'a \ P\)

lemma \(\text{R1-state-srea} \ •\ R1(\text{state}'a \cdot P) = (\text{state}'a \cdot R1(P))\)
by \((\text{rel-auto})\)

lemma \(\text{R2c-state-srea} \ •\ R2c(\text{state}'a \cdot P) = (\text{state}'a \cdot R2c(P))\)
by \((\text{rel-auto})\)

lemma \(\text{R3h-state-srea} \ •\ R3h(\text{state}'a \cdot P) = (\text{state}'a \cdot R3h(P))\)
by \((\text{rel-auto})\)

lemma \(\text{RD1-state-srea} \ •\ RD1(\text{state}'a \cdot P) = (\text{state}'a \cdot RD1(P))\)
by \((\text{rel-auto})\)

96
lemma **RD2-state-srea**: \(RD2(\text{state} 'a \cdot P) = (\text{state} 'a \cdot RD2(P))\)
by (rel-auto)

lemma **RD3-state-srea**: \(RD3(\text{state} 'a \cdot P) = (\text{state} 'a \cdot RD3(P))\)
by (rel-auto, blast+)

lemma **SRD-state-srea [closure]**: \(P \text{ is SRD} \implies \text{state} 'a \cdot P \text{ is SRD}\)
by (simp add: Healthy-def R1-state-srea R2c-state-srea R3h-state-srea RD1-state-srea RD2-state-srea RHS-def SRD-def)

lemma **NSRD-state-srea [closure]**: \(P \text{ is NSRD} \implies \text{state} 'a \cdot P \text{ is NSRD}\)
by (metis Healthy-def NSRD-is-RD3 NSRD-is-SRD RD3-state-srea SRD-RD3-implies-NSRD SRD-state-srea)

lemma **preR-state-srea [rdes]**: \(\text{pre}_R(\text{state} 'a \cdot P) = \langle \forall \, \{$st, \$st\} \cdot \text{pre}_R(P)\rangle_S\)
by (simp add: state-def, rel-auto)

lemma **periR-state-srea [rdes]**: \(\text{peri}_R(\text{state} 'a \cdot P) = \text{state} 'a \cdot \text{peri}_R(P)\)
by (rel-auto)

lemma **postR-state-srea [rdes]**: \(\text{post}_R(\text{state} 'a \cdot P) = \text{state} 'a \cdot \text{post}_R(P)\)
by (rel-auto)

### 11.9 While Loop

definition **WhileR** :: 's upred \Rightarrow ('s, 't::size-trace, 'a) hrel-rsp \Rightarrow ('s, 't, 'a) hrel-rsp (while\_R - do - od)

where
\(\text{WhileR} \ b \ P = (\mu_R X \cdot (P ;; X) \triangleleft b \triangleright_R H_R)\)

lemma **Sup-power-false**:
fixes \(F :: 'a \text{ upred} \Rightarrow 'a \text{ upred}\)
shows \((\Pi i. (F ^^ i \text{ false})) = (\Pi i. (F ^^ (i+1) \text{ false}))\)
proof
also have \((\Pi i. (F ^^ i \text{ false})) = (F ^^ 0 \text{ false} \cap (\Pi i. (F ^^ (i+1) \text{ false})))\)
by (subst Sup-power-expand, simp)
also have \((\Pi i. (F ^^ (i+1) \text{ false}))\)
by (simp)
finally show \(?lhs \implies \text{thesis}\)
qed

theorem **WhileR-iter-expand**:
assumes \(P \text{ is NSRD} P \text{ is Productive}\)
shows \(\text{while}_R b \text{ do } P \text{ od} = (\Pi i. (P \triangleleft b \triangleright_R \text{H}_R) \triangleleft i ;; (P ;; (\text{Miracle} \triangleleft b \triangleright_R \text{H}_R)))\) (is \(?lhs = \text{thesis}\))
proof
have \(1\text{-Continuous (}\lambda X. P ;; \text{SRD} X\)\)
using SRD-Continuous
by (clarsimp simp add: Continuous-def seq-SUP-distl[THEN sym], drule-tac \(x = A\ in\ spec, simp\)
have \(2\text{-Continuous (}\lambda X. P ;; \text{SRD} X \triangleleft b \triangleright_R \text{H}_R\)\)
by (simp add: 1 closure assms)
have \(?lhs = (\mu_R X \cdot P ;; X \triangleleft b \triangleright_R \text{H}_R)\)
by (simp add: WhileR-def)
also have \(\text{thesis} = (\lambda X. P ;; \text{SRD}(X) \triangleleft b \triangleright_R \text{H}_R)\)
by (auto simp add: srd-mu-equiv closure assms)
also have \(\text{thesis} = (\nu X. P ;; \text{SRD}(X) \triangleleft b \triangleright_R \text{H}_R)\)
by (auto simp add: guarded-fp-uniq Guarded-if-Productive[OF assms] funcsetI closure assms)
also have \(\text{thesis} = (\Pi i. (\lambda X. P ;; \text{SRD} X \triangleleft b \triangleright_R \text{H}_R) \triangleleft i \text{ false})\)
by (simp add: sup-continuous-lfp 2 sup-continuous-Continuous false-upred-def)
also have \( \ldots = (\prod i \cdot (\lambda X. P \land SRD X \triangleleft b \triangleright^i R I_R) \circledast (i+1)) \) false
  by (simp add: Sup-power-false)
also have \( \ldots = (\prod i \cdot (P \land b \triangleright R I_R)^i) \circledast (P \land SRD \circ b \triangleright R I_R)) \)
proof (rule SUP-cong, simp)
  fix \( i \)
  show \((\lambda X. P \land SRD X \triangleleft b \triangleright^i R I_R) \circledast (i+1)) \) false = \( (P \land b \triangleright R I_R)^i \circledast (P \land Miracle \circ b \triangleright R I_R)) \)
    proof (induct \( i \)
      case 0
      thm if-eq-cancel
      then show ?case
        by (simp,metis srdes-hcond-def srdes-theory-continuous.healthy-top)
    next
      case (Suc \( i \)
    show ?thesis
      proof (case)
        have \((\lambda X. P \land SRD X \triangleleft b \triangleright^i R I_R) \circledast (Suc i + 1)) \) false = \( P \land SRD ((\lambda X. P \land SRD X \triangleleft b \triangleright^i R I_R) \circledast (i+1)) \) false = \( b \triangleright R I_R \)
          by simp
        also have \( \ldots = P \land SRD ((P \land b \triangleright R I_R)^i \circledast (P \land Miracle \circ b \triangleright R I_R)) \circledast b \triangleright R I_R \)
          using Suc.hyps by auto
        also have \( \ldots = P \land SRD ((P \land b \triangleright R I_R)^i \circledast (P \land Miracle \circ b \triangleright R I_R)) \circledast b \triangleright R I_R \)
          by (metis (no-types, lifting) Healthy-if NSRD-cond-srea NSRD-is-SRD NSRD-power-Suc NSRD-srd-skip SRD-cond-srea SRD-seqr-closure assms(1) power.power-eq-if seqr-left-unit srdes-theory-continuous.top-closed)
        also have \( \ldots = (P \land b \triangleright R I_R)^i \circledast (P \land Miracle \circ b \triangleright R I_R) \)
          proof (induct \( i \)
            case 0
            then show ?case
          next
            case (Suc \( i \)
          have \( 1 \cdot (P \land b \triangleright R I_R) \cdot (P \land b \triangleright R I_R)^i \) \( = ((P \land b \triangleright R I_R) \cdot (P \land b \triangleright R I_R)^i) \)
              by (simp add: NSRD-is-SRD RA1 SRD-cond-srea SRD-left-unit SRD-srd-skip assms(1))
            then show ?case
              proof
                \( \ldots = (\lambda u \cdot (P \land b \triangleright R I_R) \cdot (Suc i) \cdot (P \land (Miracle) \circ b \triangleright R (I_R)) \circ b \triangleright R (I_R)) \)
                  = \( ((u \land b \triangleright R I_R) \cdot (P \land b \triangleright R I_R)^i \Land (Suc i) \cdot (P \land (Miracle) \circ b \triangleright R (I_R)) \)
                    by (metis (no-types) Suc.hyps 1 cond-L6 cond-st-distr power.power-power-Suc)
              then show ?thesis
                by (simp add: RA1 upred-semiring.power-Suc)
              qed
            qed
          qed
        finally show ?thesis .
      qed
    qed
  qed
also have \( \ldots = (\prod i \cdot (P \land b \triangleright R I_R)^i) \circledast (P \land Miracle \circ b \triangleright R I_R)) \)
  by (simp add: UNF-as-Sup-collect')
finally show ?thesis .
qed

theorem WhileR-star-expand:
  assumes \( P \land NSRD \ P \land Productive \)

98
shows $\text{while}_R \ b \ \text{do} \ P \ \text{od} = (P \ \langle b \ \triangleright_R \ \text{I}_R \rangle)^* \ ; \ (P ;; \ \text{Miracle} \ \langle b \ \triangleright_R \ \text{I}_R \rangle)$ (is $\exists \text{lhs} = \exists \text{rhs}$)

proof –

have $\exists \text{lhs} = ((\bigcap i \ast (P \ \langle a \ \triangleright_R \ \text{I}_R \rangle)^* \ ; \ (P ;; \ \text{Miracle} \ \langle b \ \triangleright_R \ \text{I}_R \rangle)$

by (simp add: WhileR-iter-expand seq-UINF-distr' assms)

also have $... = (P \ \langle b \ \triangleright_R \ \text{I}_R \rangle)^* \ ; \ (P ;; \ \text{Miracle} \ \langle b \ \triangleright_R \ \text{I}_R \rangle)$

by (simp add: ustar-def)

also have $... = ((P \ \langle a \ \triangleright_R \ \text{I}_R \rangle)^* \ ; \ P ;; \ \text{Miracle} \ \langle b \ \triangleright_R \ \text{I}_R \rangle)$

by (simp add: seq-assoc SRD-left-unit closure assms)

also have $... = (P \ \langle a \ \triangleright_R \ \text{I}_R \rangle)^* \ ; \ (P ;; \ \text{Miracle} \ \langle b \ \triangleright_R \ \text{I}_R \rangle)$

by (simp add: nsrd-thy.Star-def)

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

qed

lemma WhileR-NSRD-closed [closure]:

assumes $P$ is NSRD $P$ is Productive

shows $\text{while}_R \ b \ \text{do} \ P \ \text{od} \ \text{is} \ \text{NSRD}$

by (simp add: WhileR-star-expand assms closure)

theorem WhileR-iter-form-lemma:

assumes $P$ is NSRD

shows $(P \ \langle a \ \triangleright_R \ \text{I}_R \rangle)^* \ ; \ (P ;; \ \text{Miracle} \ \langle b \ \triangleright_R \ \text{I}_R \rangle) = ((b)\uparrow \text{I}_R \ ; \ P)^* \ ; \ (\neg b)\uparrow \text{I}_R$

proof –

have $(P \ \langle b \ \triangleright_R \ \text{I}_R \rangle)^* \ ; \ (P ;; \ \text{Miracle} \ \langle b \ \triangleright_R \ \text{I}_R \rangle) = ((b)\uparrow \text{I}_R \ ; \ P) \cap (\neg b)\uparrow \text{I}_R \ ; \ (P ;; \ \text{Miracle} \ \langle b \ \triangleright_R \ \text{I}_R \rangle)$

by (simp add: AssumeR-NSRD NSRD-right-unit NSRD-srd-skip assms(1) cond-srwa-AssumeR-form)

also have $... = (((b)\uparrow \text{I}_R \ ; \ P)^* \ ; \ (\neg b)\uparrow \text{I}_R = (H \text{I}_R \cap (b)\uparrow \text{I}_R \ ; \ P)^* \ ; \ (\neg b)\uparrow \text{I}_R \ ; \ P)$

by (simp add: AssumeR-NSRD NSRD-srd-closure nsrd-thy.Star-denest assms(1))

also have $... = (((b)\uparrow \text{I}_R \ ; \ P)^* \ ; \ (\neg b)\uparrow \text{I}_R \ ; \ P) \ ; \ (\neg b)\uparrow \text{I}_R \ ; \ P$ (P ;; Miracle \ \langle b \ \triangleright_R \ \text{I}_R \rangle)

by (simp add: AssumeR-NSRD NSRD-seqr-closure nsrd-thy.Star-invassmns(1))

also have $... = (((b)\uparrow \text{I}_R \ ; \ P)^* \ ; \ (\neg b)\uparrow \text{I}_R \ ; \ P) \ ; \ (\neg b)\uparrow \text{I}_R \ ; \ P$ (P ;; Miracle \ \langle b \ \triangleright_R \ \text{I}_R \rangle)

by (simp add: AssumeR-NSRD NSRD-Miracle NSRD-right-unit NSRD-srd-closure NSRD-srd-skip assms(1) cond-srwa-AssumeR-form)

also have $... = (((b)\uparrow \text{I}_R \ ; \ P)^* \ ; \ (\neg b)\uparrow \text{I}_R \ ; \ P) \ ; \ (\neg b)\uparrow \text{I}_R \ ; \ P$

by (simp add: upred-semiring.distrib-left)

also have $... = (((b)\uparrow \text{I}_R \ ; \ P)^* \ ; \ (\neg b)\uparrow \text{I}_R \ ; \ P)$

by (simp add: ustar-def)

proof –

have $(((b)\uparrow \text{I}_R \ ; \ P)^* \ ; \ (\neg b)\uparrow \text{I}_R = (H \text{I}_R \cap (b)\uparrow \text{I}_R \ ; \ P)^* \ ; \ (\neg b)\uparrow \text{I}_R \ ; \ P)$

by (simp add: AssumeR-true NSRD-right-unit assms(1))

also have $... = ((\neg b)\uparrow \text{I}_R \cap (((b)\uparrow \text{I}_R \ ; \ P)^* \ ; \ (\neg b)\uparrow \text{I}_R \ ; \ P) \ ; \ (\neg b)\uparrow \text{I}_R \ ; \ P)$

by (simp add: AssumeR-choice upred-semiring.add-assoc upred-semiring.distrib-left upred-semiring.distrib-right)

also have $... = ((\neg b)\uparrow \text{I}_R \cap (((b)\uparrow \text{I}_R \ ; \ P)^* \ ; \ (\neg b)\uparrow \text{I}_R \ ; \ P) \ ; \ (\neg b)\uparrow \text{I}_R \ ; \ P) \ ; \ (\neg b)\uparrow \text{I}_R \ ; \ P)$

by (simp add: RA1)

also have $... = ((\neg b)\uparrow \text{I}_R \cap (((b)\uparrow \text{I}_R \ ; \ P)^* \ ; \ (\neg b)\uparrow \text{I}_R \ ; \ P) \ ; \ (\neg b)\uparrow \text{I}_R \ ; \ P) \ ; \ (\neg b)\uparrow \text{I}_R \ ; \ P)$

by (simp add: AssumeR-comp AssumeR-false)
finally have \((\{b\}^\top_R \circ P)^* \circ \lnot b \circ^\top_R \subseteq ((\{b\}^\top_R \circ P)^* \circ [b]^\top_R \circ P) \circ \text{Miracle}\)
by (simp add: semilattice-sup-class.le-sup1)
thus \(?thesis\)
by (simp add: semilattice-sup-class.le-iff)
qed
finally show \(?thesis\) .
qed

theorem WhileR-iter-form:
  assumes \(P\) is NSRD \(P\) is Productive
  shows while \(R\) do \(P\) od = \((\{b\}^\top_R \circ P)^* \circ [b]^\top_R \circ P\) \circ \text{Miracle}
by (simp add: WhileR-iter-form-lemma WhileR-star-expand assms)

theorem WhileR-true:
  assumes \(P\) is NSRD \(P\) is Productive
  shows while \(R\) true do \(P\) od = \(P^* \circ \text{Miracle}\)
by (simp add: WhileR-iter-form AssumeR-true AssumeR-false SRD-left-unit assms closure)

lemma WhileR-insert-assume:
  assumes \(P\) is NSRD \(P\) is Productive
  shows while \(R\) false do \(P\) od = \(II_R\)
by (simp add: WhileR-def rpred closure srdes-theory-continuous.LFP-const)

theorem WhileR-rdes-def [rdes-def]:
  assumes \(P\) is RC \(Q\) is RR \(R\) is RR $st' \n\notin Q R\)
  shows while \(R\) do \(P\) od = \(R\) \(\circ (P \circ Q \circ R)\).
  \((\{b\}^\top_r \circ R)^* \circ [b]^\top_r \circ R) \circ [(b)^\top_r \circ Q \circ (P)^* \circ [b]^\top_r \circ \lnot b]^\top_r\)
(is \(?lhs\) = \(?rhs\))
proof –
  have \(?lhs\) = \((\{b\}^\top_r \circ R, (P \circ Q \circ R))^{*R} \circ \lnot b]^\top_r\)
  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
  \(\text{Closure conditions}\)
  \(Q_1\) is RC \(Q_2\) is RR \(Q_3\) is RR $st' \n\notin Q_2 Q_3\)
  \(\text{Refinement conditions}\)
  \(\{b\}^\top_r \circ Q_3)^* \circ wp_r ([b]_{S <} \Rightarrow_r P_1 \subseteq P_1\)
  \(P_2 \subseteq [b]_{S <} \circ Q_2\)
  \(P_2 \subseteq [b]_{S <} \circ Q_3 \circ P_2\)
  \(P_3 \subseteq [b]_{S <} \circ Q_3 \circ P_3\)
  \(\text{shows} R, (P_1 \circ P_2 \circ P_3) \subseteq while_R R do R_\ast(Q_1 \circ Q_2 \circ Q_3) od\)
proof (simp add: rdes-def assms, rule srdes-tri-refine-intro)
show \(((b)^\top_r ; Q_3)^r \cup P_r, (b|_{S<} \Rightarrow_r Q_1) \subseteq P_1\)
by (simp add: assms)
show \(P_2 \subseteq (P_1 \land (b)^\top_r ; Q_3)^r \cup (b)^\top_r ; Q_2\)
proof –
have \(P_2 \subseteq ((b)^\top_r ; Q_3)^r \cup (b)^\top_r ; Q_2\)
by (simp add: assms rea-assume-RR rrel-thy Star-inductl seqR-RR-closed seqr-assoc)
thus \(\?thesis\)
by (simp add: utp-pred-laws le-imp2)
qed

\[11.10\] Iteration Construction

**definition IterateR**

\(\vdash (a \seteq \Rightarrow \; \text{upred}) \Rightarrow (a \Rightarrow (s, \text{size-trace}, \alpha \text{ hrel-rsp}) \Rightarrow (s, \text{t}, \alpha \text{ hrel-rsp})\)

**where IterateR A g P = whileR \((\forall i \in A \cdot g(i))\) do \((if R i \in A \cdot g(i) \Rightarrow P (i) \text{ R})\) od**

**definition IterateR-list**

\(\vdash (\text{upred} \times (s, \text{size-trace}, \alpha \text{ hrel-rsp}) \text{ list}) \Rightarrow (s, \text{t}, \alpha \text{ hrel-rsp})\)

**where [upred-defs, ndes-simp]**

\(\text{IterateR-list xs} = \text{IterateR} \{0.\langle \text{length} \; xs \rangle \} (\lambda i. \text{map} \; \text{fst} \; \text{xs} \; i) (\lambda i. \text{map} \; \text{snd} \; \text{xs} \; i)\)

**syntax**

\(-\text{iter-srd} \qquad \text{ptrn} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow (\text{do}_{R} \; \text{-} \; \cdots \; \rightarrow \; \text{R})\)

\(-\text{iter-gcommR} \qquad \text{gcomms} \Rightarrow \text{logic} \Rightarrow (\text{do}_{R} / \text{-} / \text{od})\)

**translations**

\(-\text{iter-srd} \; x \; A \; g \; P \Rightarrow \text{CONST} \; \text{IterateR} \; (\lambda \; x. \; A) (\lambda \; x. \; P)\)

\(-\text{iter-srd} \; x \; A \; g \; P \Leftarrow \text{CONST} \; \text{IterateR} \; (\lambda \; x. \; A) (\lambda \; x'. \; P)\)

\(-\text{iter-gcommR} \; cs \Rightarrow \text{CONST} \; \text{IterateR-list} \; cs\)

\(-\text{iter-gcommR} (-\text{gcomm-show} \; cs) \Rightarrow \text{CONST} \; \text{IterateR-list} \; cs\)

**lemma IterateR-NSRD-closed [closure]:**

**assumes**

\(\land \; i. \; i \in I \Rightarrow P(i) \text{ is NSRD}\)
\(\land \; i. \; i \in I \Rightarrow P(i) \text{ is Productive}\)

**shows** \(\text{do}_{R} \; i \in I \cdot g(i) \Rightarrow P(i) \; \text{fi} \text{ is NSRD}\)

**by (simp add: IterateR-def closure assms)**

**lemma IterateR-empty:**

\(\text{do}_{R} \; i \in \{\} \cdot g(i) \Rightarrow P(i) \; \text{fi} = \text{II}_{R}\)

**by (simp add: IterateR-def srd-mu-equiv closure rpred gfp-const WhileR-false)**

**lemma IterateR-singleton:**

**assumes**

\(P \; k \text{ is NSRD} \; P \; k \text{ is Productive}\)

**shows** \(\text{do}_{R} \; i \in \{k\} \cdot g(i) \Rightarrow P(i) \; \text{fi} = \text{whileR} \; g(k) \; \text{do} \; P(k) \; \text{od} \text{ is } \?lhs = \?rhs\)

**proof –**

**have** \(\?lhs = \text{whileR} \; g \; k \; \text{do} \; P \; k < g \; k \triangleright_{R} \text{ Chaos} \; \text{od}\)

**by (simp add: IterateR-def AlternateR-singleton assms closure)**
also have \( \ldots = \text{while}_R g \, k \, \text{do} \, [g \, k]^{\top}_R ;; (P \, k \, g \, k \, \triangleright_R \, \text{Chaos}) \, \text{od} \)
by \((\text{simp add: WhileR-insert-assume closure assms})\)
also have \( \ldots = \text{while}_R g \, k \, \text{do} \, P \, k \, \text{od} \)
by \((\text{simp add: AssumeR-cond-left NSRD-Chaos WhileR-insert-assume assms})\)
finally show \( \text{thesis} \).
qed

}\documentposition{declare \texttt{IterateR-list-def [rdes-def]}}
\documentposition{declare \texttt{IterateR-def [rdes-def]}}
\documentposition{method \texttt{unfold-iteration = simp add: IterateR-list-def IterateR-def AlternateR-list-def AlternateR-def UINF-upto-expand-first}}

11.11 Substitution Laws

\textbf{lemma srd-subst-Chaos [usubst]}:
\( \sigma \uparrow_S \text{Chaos} = \text{Chaos} \)
by \((\text{rdes-simp})\)

\textbf{lemma srd-subst-Miracle [usubst]}:
\( \sigma \uparrow_S \text{Miracle} = \text{Miracle} \)
by \((\text{rdes-simp})\)

\textbf{lemma srd-subst-skip [usubst]}:
\( \sigma \uparrow_S \text{II}_R = \langle \sigma \rangle_R \)
by \((\text{rdes-eq})\)

\textbf{lemma srd-subst-assigns [usubst]}:
\( \sigma \uparrow_S \langle \nu \rangle_R = \langle \nu \circ \sigma \rangle_R \)
by \((\text{rdes-eq})\)

11.12 Algebraic Laws

\textbf{theorem assigns-srd-id}:
\( \langle \text{id} \rangle_R = \text{II}_R \)
by \((\text{rdes-eq})\)

\textbf{theorem assigns-srd-comp}:
\( \langle \sigma \rangle_R ;; \langle \nu \rangle_R = \langle \nu \circ \sigma \rangle_R \)
by \((\text{rdes-eq})\)

\textbf{theorem assigns-srd-Miracle}:
\( \langle \sigma \rangle_R ;; \text{Miracle} = \text{Miracle} \)
by \((\text{rdes-eq})\)

\textbf{theorem assigns-srd-Chaos}:
\( \langle \sigma \rangle_R ;; \text{Chaos} = \text{Chaos} \)
by \((\text{rdes-eq})\)

\textbf{theorem assigns-srd-cond}:
\( \langle \sigma \rangle_R \triangleleft b \triangleright_R \langle \nu \rangle_R = \langle \sigma \triangleleft b \triangleright_s \nu \rangle_R \)
by \((\text{rdes-eq})\)

\textbf{theorem assigns-srd-left-seq}:
assumes \( P \) is NSRD
shows \( \langle \sigma \rangle_R ;; P = \sigma \uparrow_S P \)
by \((\text{rdes-simp cls: assms})\)

\textbf{lemma AlternateR-seq-distr}:
assumes \( \bigwedge_i. \, A \, i \, \text{is NSRD} \, B \, i \, \text{is NSRD} \, C \, i \, \text{is NSRD} \)
shows \( (\text{if}_R \, i \, \in \, I \cdot g \, i \rightarrow \, A \, i \, \text{else} \, B \, i) \, ;; \, C = (\text{if}_R \, i \, \in \, I \cdot g \, i \rightarrow \, A \, i \, ;; \, C \, \text{else} \, B \, ;; \, C \, \text{fi}) \)
proof (cases \( I = \{\} \))
  case True
  then show ?thesis by (simp)
next
  case False
  then show ?thesis
    by (simp add: AlternateR-def upred-semiring.distrib-right seq-UINF-distr gcmd-seq-distr assms (3))
qed

lemma AlternateR-is-cond-srea:
  assumes A is NSRD B is NSRD
  shows (\( \text{if} \ R i \in \{a\} \cdot g \rightarrow A \text{ else } B \text{ fi} \)) = (A \overset{g}{\triangleright} R B)
  by (rdes-eq cls: assms)

lemma AlternateR-Chaos:
  if \( R i \in A \cdot g(i) \rightarrow \text{Chaos} \) fi
  = \text{Chaos}
  by (cases A = {}, simp, rdes-eq)

lemma choose-srd-par:
  choose \( R \parallel R \) choose \( R \) = choose \( R \)
  by (rdes-eq)

11.13 Lifting designs to reactive designs

definition des-rea-lift :: \('s hrel-des\) \Rightarrow (\(\text{'s} \cdot \text{trace} \), \text{'t} :: trace, \text{'a}) hrel-rsp (R_D) where
  \[ upred-defs \]: \( R_D(P) = R_s(\lceil \text{pre}_D(P) \rceil_S \triangleright \text{false} \odot (\text{str'} = \text{str} \land [\text{post}_D(P)]_S) ) \)

definition des-rea-drop :: (\(\text{'s} \cdot \text{trace} \), \text{'t} :: trace, \text{'a}) hrel-rsp \Rightarrow \(\text{'s} \cdot \text{hrel-des} (D_R) \) where
  \[ upred-defs \]: \( D_R(P) = \lfloor (\text{pre}_R(P))[\text{str'/str}] \rceil_v \lceil_v \{ \text{str, str'} \} S \triangleright \text{false} \odot (\text{str'} = \text{str} \land [\text{post}_R(P)]_S) \)

lemma ndesign-rea-lift-inverse: \( D_R(R_D(p \triangleright_n Q)) = p \triangleright_n Q \)
  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 usbst unrest)
  apply (rel-auto)
  done

lemma ndesign-rea-lift-injective:
  assumes P is N Q is N R_D P = R_D Q (is ?RP(P) = ?RQ(Q))
  shows P = Q
proof -
  have ?RP([\text{pre}_D(P)]_S < \triangleright_n \text{post}_D(P)) = ?RQ([\text{pre}_D(Q)]_S < \triangleright_n \text{post}_D(Q))
    by (simp add: ndesign-form assms)
  hence \( \text{pre}_D(P) < \triangleright_n \text{post}_D(P) = \text{pre}_D(Q) < \triangleright_n \text{post}_D(Q) \)
    by (metis ndesign-rea-lift-inverse)
  thus ?thesis
    by (simp add: ndesign-form assms)
qed

lemma des-rea-lift-closure [closure]: \( R_D(P) \) is SRD
  by (simp add: des-rea-lift-def RHS-design-is-SRD unrest)

lemma preR-des-rea-lift [rdes]:
  \( \text{pre}_R(R_D(P)) = R1([\text{pre}_D(P)]_S) \)
  by (rel-auto)
lemma periR-des-rea-lift [rdes]:
peri\(_R(\mathcal{R}_D(P))\) = (false \& (\mathcal{P} \geq_{\mathcal{S}} (\$tr \leq_{\mathcal{S}} \$tr'))
by (rel-auto)

lemma postR-des-rea-lift [rdes]:
post\(_R(\mathcal{R}_D(P))\) = ((true \& (\mathcal{P} \geq_{\mathcal{S}} \neg (\$tr \leq_{\mathcal{S}} \$tr'))) \Rightarrow (\$tr' = \$tr \& (\mathcal{P} \geq_{\mathcal{S}} (\$tr')))
apply (rel-auto) using minus-zero-eq by blast

lemma ndes-rea-lift-closure [closure]:
assumes \(P\) is \(N\)
shows \(\mathcal{R}_D(P)\) is NSRD
proof –
obtain \(p Q\) where \(P\):
by (metis H1-H3-commute H1-H3-is-normal-design H1-idem Healthy-def assms)
show \(\vdash\)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 \(H\) \(P\) \(\preceq\) \(Q\)
shows \(\mathcal{R}_D(P)\) \(\preceq\) \(\mathcal{R}_D(Q)\)
apply (simp add: des-rea-lift-def)
apply (rule srdes-tri-refine-intro')
apply (auto intro: H1-H2-refines assms aext-mono)
apply (rel-auto)
apply (metis (no-types, hide-lams) aext-mono assms (3) design-post-choice
    semilattice-sup-class.sup.orderE utp-pred-laws.inf.coboundedII utp-pred-laws.inf.commute utp-pred-laws.sup.order-iff
    done)

Homomorphism laws

lemma R-D-Miracle:
\(\mathcal{R}_D(\top_D) = \text{Miracle}\)
by (simp add: Miracle-def, rel-auto)

lemma R-D-Chaos:
\(\mathcal{R}_D(\bot_D) = \text{Chaos}\)
proof –
have \(\mathcal{R}_D(\bot_D) = \mathcal{R}_D(\text{false} \uparrow_{\mathcal{R}} \text{true})\)
by (rel-auto)
also have \(= \mathcal{R}_a(\text{false} \uparrow \text{false} \circ (\$tr' =_{\mathcal{S}} \$tr'))\)
by (simp add: Chaos-def des-rea-lift-def alpha)
also have \(= \mathcal{R}_a(\text{true})\)
by (rel-auto)
also have \(= \text{Chaos}\)
by (simp add: Chaos-def design-false-pre)
finally show \(\vdash\)thesis .
qed

lemma R-D-inf:
\(\mathcal{R}_D(P \cap Q) = \mathcal{R}_D(P) \cap \mathcal{R}_D(Q)\)
by (rule antisym, rel-auto+)
lemma R-D-cond:
\[ R_D(P \triangleleft [b]_{D < \triangleright} Q) = R_D(P) \triangleleft b \triangleright R_D(Q) \]
by (rule antisym, rel-auto+)

lemma R-D-seq-ndesign:
\[ R_D(p_1 \vdash_n Q_1) \circ R_D(p_2 \vdash_n Q_2) = R_D((p_1 \vdash_n Q_1) \circ (p_2 \vdash_n Q_2)) \]
apply (rule antisym)
apply (rule SRD-refine-intro)
apply (simp-all add: closure rdes ndesign-composition-up)
apply (rel-auto)
apply (rel-auto)
apply (rel-auto)
don done

lemma R-D-seq:
assumes P is N Q is N
shows R_D(P) \circ R_D(Q) = R_D((p_1 \vdash_n Q_1) \circ (p_2 \vdash_n Q_2))
by (metis R-D-seq-ndesign assms ndesign-form)

These laws are applicable only when there is no further alphabet extension

lemma R-D-skip:
\[ R_D(\text{II}_D) = (\text{II}_R :: ('s,t::trace,unit) \text{hrel-rsp}) \]
apply (rel-auto)
using minus-zero-eq by blast+

lemma R-D-assigns:
\[ R_D(\langle \sigma \rangle_D) = (\langle \sigma \rangle_R :: ('s,t::trace,unit) \text{hrel-rsp}) \]
by (simp add: assigns-d-def des-rea-lift-def alpha assigns-srd-RHS-tri-des, rel-auto)

end

12 Instantaneous Reactive Designs

theory utp-rdes-instant
imports utp-rdes-prog
begin

definition ISRD1 :: ('s,t::trace,'a) hrel-rsp \Rightarrow ('s,t,'a) hrel-rsp where
[upred-defs]: ISRD1(P) = P \parallel R (true_t \vdash false' \circ (\$tr' = u \$tr'))

definition ISRD :: ('s,t::trace,'a) hrel-rsp \Rightarrow ('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 \subseteq Q \Rightarrow ISRD1(P) \subseteq ISRD1(Q)
by (rel-auto)

lemma ISRD1-RHS-design-form:
assumes \$ok' \# P \$ok' \# Q \$ok' \# R
shows $\text{ISRD1}(R_\alpha(P \vdash Q \odot R)) = R_\alpha(P \vdash \text{false} \odot (R \land \$tr^- = \_ \$tr))$
using assms by (simp add: ISRD1-def choose-srd-def RHS-tri-design-par unrest, rel-auto)

lemma ISRD1-form:
$\text{ISRD1}(\text{SRD}(P)) = R_\alpha(\pre_R(P) \vdash \text{false} \odot (\post_R(P) \land \$tr^- = \_ \$tr))$
by (simp add: ISRD1-RHS-design-form SRD-as-reactive-tri-design unrest)

lemma ISRD1-rdes-def [rdes-def]:
\[ \begin{array}{c}
\text{P is RR; R is RR} \\
\Rightarrow \text{ISRD1} (R_\alpha(P \vdash Q \odot R)) = R_\alpha(P \vdash \text{false} \odot (R \land \$tr^- = \_ \$tr))
\end{array} \]
by (simp add: ISRD1-def rdes-def closure rpred)

lemma ISRD-intro:
assumes P is NSRD peri R (P) = (¬ \pre_R(P)) ($\text{str}^- = \_ \text{str}$) ⊑ post_R(P)
shows P is ISRD
proof –
have R_\alpha(\pre_R(P) \vdash \text{peri}_R(P) \odot \post_R(P)) is ISRD1
  apply (simp add: Healthy-def rdes-def closure assms (1–2))
  using assms(3) least-zero apply (rel-blast)
done
hence P is ISRD1
  by (simp add: ISRD-def Healthy-comp assms (1))
thus ?thesis
  by (simp add: ISRD-def Healthy-comp assms (1))
qed

lemma ISRD1-rdes-intro:
assumes P is RR Q is RR ($\text{str}^- = \_ \text{str}$) ⊑ Q
shows R_\alpha(P \vdash \text{false} \odot Q) is ISRD1
unfolding Healthy-def
by (simp add: ISRD1-rdes-def assms closure unrest utp-pred-laws.inf.absorb1)

lemma ISRD-rdes-intro [closure]:
assumes P is RC Q is RR ($\text{str}^- = \_ \text{str}$) ⊑ Q
shows R_\alpha(P \vdash \text{false} \odot Q) is ISRD
unfolding Healthy-def
by (simp add: ISRD-def closure Healthy-if ISRD1-rdes-def assms unrest utp-pred-laws.inf.absorb1)

lemma ISRD-implies-ISRD1:
assumes P is ISRD
shows P is ISRD1
proof –
        have ISRD(P) is ISRD1
          by (simp add: ISRD-def Healthy-def ISRD1-idem)
        thus ?thesis
          by (simp add: assms Healthy-if)
qed

lemma ISRD-implies-SRD:
assumes P is ISRD
shows P is SRD
proof –
        have 1:ISRD(P) = R_\alpha((\neg \neg \pre_R(P) :: R \text{ true } \land R \text{ true}) \vdash \text{false} \odot (post_R P \land \$tr^- = \_ \$tr))
          by (simp add: NSRD-form ISRD1-def ISRD-def RHS-tri-design-par rdes-def unrest closure)
        moreover have ... is SRD
          by (simp add: closure unrest)

106
ultimately have \( \text{ISRD}(P) \) is \( \text{SRD} \)
  by (simp)
with \( \text{assms} \) show \( \text{thesis} \)
  by (simp add: Healthy-def)
qed

lemma ISRD-implies-NSRD [closure]:
  assumes \( P \) is ISRD
  shows \( P \) is NSRD
proof
  have \( 1 : \text{ISRD}(P) = \text{ISRD1}(\text{RD3}(\text{SRD}(P))) \)
    by (simp add: ISRD-def NSRD-def SRD-def, metis RD1-RD3-commute RD3-left-subsumes-RD2)
  also have \( ... = \text{ISRD1}(\text{RD3}(P)) \)
    by (simp add: \( \text{assms} \) ISRD-implies-SRD Healthy-if)
  also have \( \ldots = \text{RD3(} \text{R}_s(\text{pre} \text{R}(P) \vdash \text{peri} \text{R}(P) \bowtie (\exists \text{st'} \cdot \text{peri} \text{R}(P) \bowtie \text{post} \text{R}(P)) \text{)} \)
    by (simp add: RD3-def, subst SRD-right-unit-tri-lemma, simp-all add: \( \text{assms} \) ISRD-implies-SRD)
  also have \( \ldots = \text{R}_s(\text{pre} \text{R}(P) \vdash \text{peri} \text{R}(P) \bowtie \text{R} \bowtie (\exists \text{st'} \cdot \text{peri} \text{R}(P) \bowtie \text{post} \text{R}(P) \bowtie \text{R}) \text{)} \)
    by (simp add: RHS-tri-normal-design-composition′ closure assms unrest choose-srd-def rpred closure I SRD-implies-SRD)
  also have \( \ldots = (\ldots ; ; I_\text{R}) \)
    by (rdes-simp, simp add: RHS-tri-normal-design-composition′ closure assms unrest ISRD-implies-SRD wp rpred wp-rea-false-RC)
  also have \( \ldots \) is RD3
    by (simp add: Healthy-def RD3-def seqr-assoc)
finally show \( \text{thesis} \)
  by (simp add: RD3-implies-NSRD Healthy-if \( \text{assms} \) ISRD-implies-SRD)
qed

lemma ISRD-form:
  assumes \( P \) is ISRD
  shows \( \text{R}_s(\text{pre} \text{R}(P) \vdash \text{false} \bowtie (\text{post} \text{R}(P) \bowtie \text{false}) \bowtie (\exists \text{st'} \cdot \text{peri} \text{R}(P) \bowtie \text{post} \text{R}(P) \bowtie \text{R}) \text{)} \) = \( P \)
proof
  have \( P = \text{ISRD1}(P) \)
    by (simp add: ISRD-implies-ISRD1 \( \text{assms} \) Healthy-if)
  also have \( \ldots = \text{ISRD1}(\text{R}_s(\text{pre} \text{R}(P) \vdash \text{peri} \text{R}(P) \bowtie \text{post} \text{R}(P))) \)
    by (simp add: SRD-reactive-tri-design ISRD-implies-SRD \( \text{assms} \))
  also have \( \ldots = \text{R}_s(\text{pre} \text{R}(P) \vdash \text{false} \bowtie (\text{post} \text{R}(P) \bowtie \text{false} \bowtie (\exists \text{st'} \cdot \text{peri} \text{R}(P) \bowtie \text{post} \text{R}(P) \bowtie \text{R}) \text{)} \)
    by (simp add: ISRD1-rdes-def closure \( \text{assms} \))
  finally show \( \text{thesis} \)
  ..
qed

lemma ISRD-elim [RD-elim]:
\[ [P \text{ is ISRD}; Q(\text{R}_s(\text{pre} \text{R}(P) \vdash \text{false} \bowtie (\text{post} \text{R}(P) \bowtie \text{false})))] \implies Q(P) \]
by (simp add: ISRD-form)

lemma skip-srd-ISRD [closure]: \( \text{I}_\text{R} \) is ISRD
by (rule ISRD-intro, simp-all add: rdes closure)

lemma assigns-srd-ISRD [closure]: \( \langle \sigma \rangle_\text{R} \) 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 \( \text{assms} \))
apply (erule ISRD-elim)+
apply (simp add: rdes-def closure assms unrest)
apply (rule ISRD-rdes-intro)
  apply (simp-all add: rdes-def closure assms unrest)
apply (rel-auto)
done

lemma ISRD-Miracle-right-zero:
  assumes P is ISRD pre R
  shows P ;; Miracle = Miracle
  by (rdes-simp cls: assms)

A recursion whose body does not extend the trace results in divergence

lemma ISRD-recurse-Chaos:
  assumes P is ISRD post R P ;; true r
  shows (\u R X : P ;; X) = Chaos
proof –
  have 1: (\u R X : P ;; X) = (\u X X ;; SRD(X))
    by (auto simp add: srdes-theory-continuous.utp-lfp-def closure assms)
  have (\u X : P ;; SRD(X)) \u Chaos
    proof (rule gfp-upperbound)
      have P ;; Chaos \u 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)+
      thus P ;; SRD Chaos \u Chaos
        by (simp add: Healthy-if srdes-theory-continuous.bottom-closed)
    qed
  thus P ;; SRD Chaos \u 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:
  (\u R X : (σ) R ;; X) = Chaos
  by (rule ISRD-recurse-Chaos, simp-all add: closure rdes, rel-auto)
end

13 Meta-theory for Reactive Designs

theory utp-rea-designs
  imports
    utp-rdes-healths
    utp-rdes-designs
    utp-rdes-triples
    utp-rdes-normal
    utp-rdes-contracts
    utp-rdes-tactics
    utp-rdes-parallel
    utp-rdes-prog
    utp-rdes-instant
    utp-rdes-guarded
begin end
References


