

Generalised Reactive Processes

Simon Foster

Samuel Canham

April 4, 2018

Contents

1	Reactive Processes Core Definitions	2
1.1	Alphabet and Signature	2
1.2	Reactive Lemmas	4
1.3	Trace contribution lens	4
2	Reactive Healthiness Conditions	5
2.1	R1: Events cannot be undone	5
2.2	R2: No dependence upon trace history	9
2.3	R3: No activity while predecessor is waiting	17
2.4	R4: The trace strictly increases	18
2.5	R5: The trace does not increase	19
2.6	RP laws	20
2.7	UTP theories	21
3	Reactive Parallel-by-Merge	22
4	Reactive Relations	25
4.1	Healthiness Conditions	26
4.2	Reactive relational operators	27
4.3	Unrestriction and substitution laws	28
4.4	Closure laws	29
4.5	Reactive relational calculus	33
4.6	UTP theory	36
4.7	Instantaneous Reactive Relations	37
5	Reactive Conditions	37
5.1	Healthiness Conditions	37
5.2	Closure laws	38
6	Reactive Programs	40
6.1	Stateful reactive alphabet	41
6.2	State Lifting	43
6.3	Reactive Program Operators	43
6.3.1	State Substitution	43
6.3.2	Assignment	45
6.3.3	Conditional	46
6.3.4	Assumptions	46
6.3.5	State Abstraction	47

6.3.6	Reactive Frames and Extensions	47
6.4	Stateful Reactive specifications	49
7	Reactive Weakest Preconditions	51
8	Reactive Hoare Logic	54
9	Meta-theory for Generalised Reactive Processes	55

1 Reactive Processes Core Definitions

```

theory utp-rea-core
imports
  UTP-Toolkit.Trace-Algebra
  UTP.utp-concurrency
  UTP-Designs.utp-designs
begin recall-syntax

```

1.1 Alphabet and Signature

The alphabet of reactive processes contains a boolean variable *wait*, which denotes whether a process is exhibiting an intermediate observation. It also has the variable *tr* which denotes the trace history of a process. The type parameter *'t* represents the trace model being used, which must form a trace algebra [3], and thus provides the theory of “generalised reactive processes” [3]. The reactive process alphabet also extends the design alphabet, and thus includes the *ok* variable. For more information on these, see the UTP book [4], or the associated tutorial [2].

```

alphabet 't::trace rp-vars = des-vars +
  wait :: bool
  tr   :: 't

```

```

type-synonym ('t, 'α) rp = ('t, 'α) rp-vars-scheme des

```

```

type-synonym ('t, 'α, 'β) rel-rp = (('t, 'α) rp, ('t, 'β) rp) urel
type-synonym ('t, 'α) hrel-rp = ('t, 'α) rp hrel

```

```

translations
  (type) ('t, 'α) rp <= (type) ('t, 'α) rp-vars-scheme des
  (type) ('t, 'α) rp <= (type) ('t, 'α) rp-vars-ext des
  (type) ('t, 'α, 'β) rel-rp <= (type) (('t, 'α) rp, ('t, 'β) rp) urel
  (type) ('t, 'α) hrel-rp <= (type) ('t, 'α) rp hrel

```

As for designs, we set up various types to represent reactive processes. The main types to be used are *('t, 'α, 'β) rel-rp* and *('t, 'α) hrel-rp*, which correspond to heterogeneous/homogeneous reactive processes whose trace model is *'t* and alphabet types are *'α* and *'β*. We also set up some useful syntax translations for these.

```

notation rp-vars-child-lensa (Σr)
notation rp-vars-child-lens (ΣR)

```

```

syntax
  -svid-rea-alpha :: svid (ΣR)

```

```

translations

```

-svid-rea-alpha \Rightarrow *CONST rp-vars-child-lens*

Lens Σ_R exists because reactive alphabets are extensible. Σ_R points to the portion of the alphabet / state space that is neither *ok*, *wait*, or *tr*.

```
declare rp-vars.splits [alpha-splits]
declare rp-vars.defs [lens-defs]
declare zero-list-def [upred-defs]
declare plus-list-def [upred-defs]
declare prefixE [elim]
```

The two locale interpretations below are a technicality to improve automatic proof support via the predicate and relational tactics. This is to enable the (re-)interpretation of state spaces to remove any occurrences of lens types after the proof tactics *pred-simp* and *rel-simp*, or any of their derivatives have been applied. Eventually, it would be desirable to automate both interpretations as part of a custom outer command for defining alphabets.

interpretation *rp-vars*:

```
lens-interp  $\lambda(ok, r). (ok, wait_v\ r, tr_v\ r, more\ r)$ 
apply (unfold-locales)
apply (rule injI)
apply (clarsimp)
done
```

interpretation *rp-vars-rel*: *lens-interp* $\lambda(ok, ok', r, r').$

```
(ok, ok', wait_v\ r, wait_v\ r', tr_v\ r, tr_v\ r', more\ r, more\ r')
apply (unfold-locales)
apply (rule injI)
apply (clarsimp)
done
```

The following syntactic orders exist to help to order lens names when, for example, performing substitution, to achieve normalisation of terms.

lemma *rea-var-ords* [*usubst*]:

```
 $\$tr \prec_v \$tr' \$wait \prec_v \$wait'$ 
 $\$ok \prec_v \$tr \$ok' \prec_v \$tr' \$ok \prec_v \$tr' \$ok' \prec_v \$tr$ 
 $\$ok \prec_v \$wait \$ok' \prec_v \$wait' \$ok \prec_v \$wait' \$ok' \prec_v \$wait$ 
 $\$tr \prec_v \$wait \$tr' \prec_v \$wait' \$tr \prec_v \$wait' \$tr' \prec_v \$wait$ 
by (simp-all add: var-name-ord-def)
```

abbreviation *wait-f*::(*t*::*trace*, α , β) *rel-rp* \Rightarrow (*t*, α , β) *rel-rp*

where *wait-f* *R* $\equiv R[\text{false}/\$wait]$

abbreviation *wait-t*::(*t*::*trace*, α , β) *rel-rp* \Rightarrow (*t*, α , β) *rel-rp*

where *wait-t* *R* $\equiv R[\text{true}/\$wait]$

syntax

```
-wait-f :: logic  $\Rightarrow$  logic (-f [1000] 1000)
-wait-t :: logic  $\Rightarrow$  logic (-t [1000] 1000)
```

translations

```
 $P_f \equiv \text{CONST usubst (CONST subst-upd CONST id (CONST ivar CONST wait) false) } P$ 
 $P_t \equiv \text{CONST usubst (CONST subst-upd CONST id (CONST ivar CONST wait) true) } P$ 
```

abbreviation *lift-rea* :: $- \Rightarrow -([\cdot]_R)$ **where**

$[P]_R \equiv P \oplus_p (\Sigma_R \times_L \Sigma_R)$

abbreviation $drop\text{-}rea :: ('t::trace, '\alpha, '\beta) \text{ rel-}rp \Rightarrow ('alpha, '\beta) \text{ urel } (\lfloor _ \rfloor_R)$ **where**
 $\lfloor P \rfloor_R \equiv P \vdash_e (\Sigma_R \times_L \Sigma_R)$

abbreviation $rea\text{-}pre\text{-}lift :: - \Rightarrow - (\lfloor _ \rfloor_{R<})$ **where** $\lceil n \rceil_{R<} \equiv \lceil \lceil n \rceil_{<} \rceil_R$

1.2 Reactive Lemmas

lemma $unrest\text{-}ok\text{-}lift\text{-}rea \ [unrest]:$
 $\$ok \# \lceil P \rceil_R \$ok' \# \lceil P \rceil_R$
by $(pred\text{-}auto)+$

lemma $unrest\text{-}wait\text{-}lift\text{-}rea \ [unrest]:$
 $\$wait \# \lceil P \rceil_R \$wait' \# \lceil P \rceil_R$
by $(pred\text{-}auto)+$

lemma $unrest\text{-}tr\text{-}lift\text{-}rea \ [unrest]:$
 $\$tr \# \lceil P \rceil_R \$tr' \# \lceil P \rceil_R$
by $(pred\text{-}auto)+$

lemma $wait\text{-}tr\text{-}bij\text{-}lemma: \text{bij-lens } (wait_a +_L tr_a +_L \Sigma_r)$
by $(unfold\text{-}locales, \text{ auto simp add: lens-defs})$

lemma $des\text{-}lens\text{-}equiv\text{-}wait\text{-}tr\text{-}rest: \Sigma_D \approx_L wait +_L tr +_L \Sigma_R$

proof $-$

have $wait +_L tr +_L \Sigma_R = (wait_a +_L tr_a +_L \Sigma_r) ;_L \Sigma_D$
by $(simp \text{ add: plus-lens-distr wait-def tr-def rp-vars-child-lens-def})$
also have $\dots \approx_L 1_L ;_L \Sigma_D$
using $lens\text{-}equiv\text{-}via\text{-}bij \ wait\text{-}tr\text{-}bij\text{-}lemma$ **by** $auto$
also have $\dots = \Sigma_D$
by $(simp)$
finally show $?thesis$
using $lens\text{-}equiv\text{-}sym$ **by** $blast$

qed

lemma $rea\text{-}lens\text{-}bij: \text{bij-lens } (ok +_L wait +_L tr +_L \Sigma_R)$

proof $-$

have $ok +_L wait +_L tr +_L \Sigma_R \approx_L ok +_L \Sigma_D$
using $des\text{-}lens\text{-}equiv\text{-}wait\text{-}tr\text{-}rest \ des\text{-}vars\text{-}indeps \ lens\text{-}equiv\text{-}sym \ lens\text{-}plus\text{-}eq\text{-}right$ **by** $blast$
also have $\dots \approx_L 1_L$
using $bij\text{-}lens\text{-}equiv\text{-}id[ok +_L \Sigma_D]$ **by** $(simp \text{ add: ok-des-bij-lens})$
finally show $?thesis$
by $(simp \text{ add: bij-lens-equiv-id})$

qed

lemma $seqr\text{-}wait\text{-}true \ [usubst]: (P ;; Q)_t = (P_t ;; Q)$
by $(rel\text{-}auto)$

lemma $seqr\text{-}wait\text{-}false \ [usubst]: (P ;; Q)_f = (P_f ;; Q)$
by $(rel\text{-}auto)$

1.3 Trace contribution lens

The following lens represents the portion of the state-space that is the difference between tr' and tr , that is the contribution that a process is making to the trace history.

definition $tcontr :: 't::trace \implies ('t, 'α) rp \times ('t, 'α) rp (tt)$ **where**
 $[lens-defs]:$
 $tcontr = (\mid lens-get = (\lambda s. get(\$tr')_v s - get(\$tr)_v s) ,$
 $lens-put = (\lambda s v. put(\$tr')_v s (get(\$tr)_v s + v)) \mid)$

definition $itrace :: 't::trace \implies ('t, 'α) rp \times ('t, 'α) rp (it)$ **where**
 $[lens-defs]:$
 $itrace = (\mid lens-get = get(\$tr)_v ,$
 $lens-put = (\lambda s v. put(\$tr')_v (put(\$tr)_v s v) v) \mid)$

lemma $tcontr-mwb-lens [simp]: mwb-lens tt$
by $(unfold-locales, simp-all add: lens-defs prod.case-eq-if)$

lemma $itrace-mwb-lens [simp]: mwb-lens it$
by $(unfold-locales, simp-all add: lens-defs prod.case-eq-if)$

syntax
 $-svid-tcontr :: svid (tt)$
 $-svid-itrace :: svid (it)$

translations
 $-svid-tcontr == CONST tcontr$
 $-svid-itrace == CONST itrace$

lemma $tcontr-alt-def: \&tt = (\$tr' - \$tr)$
by $(rel-auto)$

lemma $tcontr-alt-def': utp-expr.var tt = (\$tr' - \$tr)$
by $(rel-auto)$

lemma $tt-indeps [simp]:$
assumes $x \bowtie (\$tr)_v x \bowtie (\$tr')_v$
shows $x \bowtie tt \bowtie x$
using $assms$
by $(unfold lens-indep-def, safe, simp-all add: tcontr-def, (metis lens-indep-get var-update-out)+)$

end

2 Reactive Healthiness Conditions

theory $utp-rea-healths$
imports $utp-rea-core$
begin

2.1 R1: Events cannot be undone

definition $R1 :: ('t::trace, 'α, 'β) rel-rp \Rightarrow ('t, 'α, 'β) rel-rp$ **where**
 $R1-def [upred-defs]: R1 (P) = (P \wedge (\$tr \leq_u \$tr'))$

lemma $R1-idem: R1(R1(P)) = R1(P)$
by $pred-auto$

lemma $R1-Idempotent [closure]: Idempotent R1$
by $(simp add: Idempotent-def R1-idem)$

lemma *R1-mono*: $P \sqsubseteq Q \implies R1(P) \sqsubseteq R1(Q)$
by *pred-auto*

lemma *R1-Monotonic*: *Monotonic R1*
by (*simp add: mono-def R1-mono*)

lemma *R1-Continuous*: *Continuous R1*
by (*auto simp add: Continuous-def, rel-auto*)

lemma *R1-unrest* [*unrest*]: $\llbracket x \bowtie \text{in-var } tr; x \bowtie \text{out-var } tr; x \# P \rrbracket \implies x \# R1(P)$
by (*simp add: R1-def unrest lens-indep-sym*)

lemma *R1-false*: $R1(\text{false}) = \text{false}$
by *pred-auto*

lemma *R1-conj*: $R1(P \wedge Q) = (R1(P) \wedge R1(Q))$
by *pred-auto*

lemma *conj-R1-closed-1* [*closure*]: $P \text{ is } R1 \implies (P \wedge Q) \text{ is } R1$
by (*rel-blast*)

lemma *conj-R1-closed-2* [*closure*]: $Q \text{ is } R1 \implies (P \wedge Q) \text{ is } R1$
by (*rel-blast*)

lemma *R1-disj*: $R1(P \vee Q) = (R1(P) \vee R1(Q))$
by *pred-auto*

lemma *disj-R1-closed* [*closure*]: $\llbracket P \text{ is } R1; Q \text{ is } R1 \rrbracket \implies (P \vee Q) \text{ is } R1$
by (*simp add: Healthy-def R1-def utp-pred-laws.inf-sup-distrib2*)

lemma *R1-impl*: $R1(P \Rightarrow Q) = ((\neg R1(\neg P)) \Rightarrow R1(Q))$
by (*rel-auto*)

lemma *R1-inf*: $R1(P \sqcap Q) = (R1(P) \sqcap R1(Q))$
by *pred-auto*

lemma *R1-USUP*:
 $R1(\bigsqcap i \in A \cdot P(i)) = (\bigsqcap i \in A \cdot R1(P(i)))$
by (*rel-auto*)

lemma *R1-Sup* [*closure*]: $\llbracket \bigwedge P. P \in A \implies P \text{ is } R1; A \neq \{\} \rrbracket \implies \bigsqcap A \text{ is } R1$
using *R1-Continuous* **by** (*auto simp add: Continuous-def Healthy-def*)

lemma *R1-UNIF*:
assumes $A \neq \{\}$
shows $R1(\bigsqcup i \in A \cdot P(i)) = (\bigsqcup i \in A \cdot R1(P(i)))$
using *assms* **by** (*rel-auto*)

lemma *R1-UNIF-ind*:
 $R1(\bigsqcup i \cdot P(i)) = (\bigsqcup i \cdot R1(P(i)))$
by (*rel-auto*)

lemma *UNIF-ind-R1-closed* [*closure*]:
 $\llbracket \bigwedge i. P(i) \text{ is } R1 \rrbracket \implies (\bigsqcup i \cdot P(i)) \text{ is } R1$
by (*rel-blast*)

lemma *UINF-R1-closed* [closure]:

$\llbracket \bigwedge i. P \ i \text{ is } R1 \rrbracket \implies (\bigcap i \in A \cdot P \ i) \text{ is } R1$
by (*rel-blast*)

lemma *tr-ext-conj-R1* [closure]:

$\$tr' =_u \$tr \hat{\ }_u e \wedge P \text{ is } R1$
by (*rel-auto*, *simp add: Prefix-Order.prefixI*)

lemma *tr-id-conj-R1* [closure]:

$\$tr' =_u \$tr \wedge P \text{ is } R1$
by (*rel-auto*)

lemma *R1-extend-conj*: $R1(P \wedge Q) = (R1(P) \wedge Q)$

by *pred-auto*

lemma *R1-extend-conj'*: $R1(P \wedge Q) = (P \wedge R1(Q))$

by *pred-auto*

lemma *R1-cond*: $R1(P \triangleleft b \triangleright Q) = (R1(P) \triangleleft b \triangleright R1(Q))$

by (*rel-auto*)

lemma *R1-cond'*: $R1(P \triangleleft b \triangleright Q) = (R1(P) \triangleleft R1(b) \triangleright R1(Q))$

by (*rel-auto*)

lemma *R1-negate-R1*: $R1(\neg R1(P)) = R1(\neg P)$

by *pred-auto*

lemma *R1-wait-true* [usubst]: $(R1 \ P)_t = R1(P)_t$

by *pred-auto*

lemma *R1-wait-false* [usubst]: $(R1 \ P)_f = R1(P)_f$

by *pred-auto*

lemma *R1-wait'-true* [usubst]: $(R1 \ P)\llbracket true/\$wait' \rrbracket = R1(P\llbracket true/\$wait' \rrbracket)$

by (*rel-auto*)

lemma *R1-wait'-false* [usubst]: $(R1 \ P)\llbracket false/\$wait' \rrbracket = R1(P\llbracket false/\$wait' \rrbracket)$

by (*rel-auto*)

lemma *R1-wait-false-closed* [closure]: $P \text{ is } R1 \implies P\llbracket false/\$wait \rrbracket \text{ is } R1$

by (*rel-auto*)

lemma *R1-wait'-false-closed* [closure]: $P \text{ is } R1 \implies P\llbracket false/\$wait' \rrbracket \text{ is } R1$

by (*rel-auto*)

lemma *R1-skip*: $R1(II) = II$

by (*rel-auto*)

lemma *skip-is-R1* [closure]: $II \text{ is } R1$

by (*rel-auto*)

lemma *subst-R1*: $\llbracket \$tr \ \sharp \ \sigma; \$tr' \ \sharp \ \sigma \rrbracket \implies \sigma \ \dagger \ (R1 \ P) = R1(\sigma \ \dagger \ P)$

by (*simp add: R1-def usubst*)

lemma *subst-R1-closed* [closure]: $\llbracket \$tr \# \sigma; \$tr' \# \sigma; P \text{ is } R1 \rrbracket \implies \sigma \dagger P \text{ is } R1$
by (metis Healthy-def subst-R1)

lemma *R1-by-refinement*:
 $P \text{ is } R1 \longleftrightarrow ((\$tr \leq_u \$tr') \sqsubseteq P)$
by (rel-blast)

lemma *R1-trace-extension* [closure]:
 $\$tr' \geq_u \$tr \wedge e \text{ is } R1$
by (rel-auto)

lemma *tr-le-trans*:
 $((\$tr \leq_u \$tr') ;; (\$tr \leq_u \$tr')) = (\$tr \leq_u \$tr')$
by (rel-auto)

lemma *R1-seqr*:
 $R1(R1(P) ;; R1(Q)) = (R1(P) ;; R1(Q))$
by (rel-auto)

lemma *R1-seqr-closure* [closure]:
assumes $P \text{ is } R1 \ Q \text{ is } R1$
shows $(P ;; Q) \text{ is } R1$
using *assms unfolding R1-by-refinement*
by (metis seqr-mono tr-le-trans)

lemma *R1-power* [closure]: $P \text{ is } R1 \implies P^n \text{ is } R1$
by (induct n, simp-all add: upred-semiring.power-Suc closure)

lemma *R1-true-comp* [simp]: $(R1(true) ;; R1(true)) = R1(true)$
by (rel-auto)

lemma *R1-ok'-true*: $(R1(P))^t = R1(P^t)$
by pred-auto

lemma *R1-ok'-false*: $(R1(P))^f = R1(P^f)$
by pred-auto

lemma *R1-ok-true*: $(R1(P))\llbracket true/\$ok \rrbracket = R1(P\llbracket true/\$ok \rrbracket)$
by pred-auto

lemma *R1-ok-false*: $(R1(P))\llbracket false/\$ok \rrbracket = R1(P\llbracket false/\$ok \rrbracket)$
by pred-auto

lemma *seqr-R1-true-right*: $((P ;; R1(true)) \vee P) = (P ;; (\$tr \leq_u \$tr'))$
by (rel-auto)

lemma *conj-R1-true-right*: $(P;;R1(true) \wedge Q;;R1(true)) ;; R1(true) = (P;;R1(true) \wedge Q;;R1(true))$
apply (rel-auto) **using** dual-order.trans **by** blast+

lemma *R1-extend-conj-unrest*: $\llbracket \$tr \# Q; \$tr' \# Q \rrbracket \implies R1(P \wedge Q) = (R1(P) \wedge Q)$
by pred-auto

lemma *R1-extend-conj-unrest'*: $\llbracket \$tr \# P; \$tr' \# P \rrbracket \implies R1(P \wedge Q) = (P \wedge R1(Q))$
by pred-auto

lemma *R1-tr'-eq-tr*: $R1(\$tr' =_u \$tr) = (\$tr' =_u \$tr)$
by (*rel-auto*)

lemma *R1-tr-less-tr'*: $R1(\$tr <_u \$tr') = (\$tr <_u \$tr')$
by (*rel-auto*)

lemma *tr-strict-prefix-R1-closed* [*closure*]: $\$tr <_u \tr' is *R1*
by (*rel-auto*)

lemma *R1-H2-commute*: $R1(H2(P)) = H2(R1(P))$
by (*simp add: H2-split R1-def usubst, rel-auto*)

2.2 R2: No dependence upon trace history

There are various ways of expressing *R2*, which are enumerated below.

definition *R2a* :: $(t::trace, 'α, 'β) \text{ rel-rp} \Rightarrow (t, 'α, 'β) \text{ rel-rp}$ **where**
[*upred-defs*]: $R2a(P) = (\prod s \cdot P[\ll s \gg, \ll s \gg + (\$tr' - \$tr) / \$tr, \$tr'])$

definition *R2a'* :: $(t::trace, 'α, 'β) \text{ rel-rp} \Rightarrow (t, 'α, 'β) \text{ rel-rp}$ **where**
[*upred-defs*]: $R2a' P = (R2a(P) \triangleleft R1(true) \triangleright P)$

definition *R2s* :: $(t::trace, 'α, 'β) \text{ rel-rp} \Rightarrow (t, 'α, 'β) \text{ rel-rp}$ **where**
[*upred-defs*]: $R2s(P) = (P[\$tr' - \$tr / \$tr'])$

definition *R2* :: $(t::trace, 'α, 'β) \text{ rel-rp} \Rightarrow (t, 'α, 'β) \text{ rel-rp}$ **where**
[*upred-defs*]: $R2(P) = R1(R2s(P))$

definition *R2c* :: $(t::trace, 'α, 'β) \text{ rel-rp} \Rightarrow (t, 'α, 'β) \text{ rel-rp}$ **where**
[*upred-defs*]: $R2c(P) = (R2s(P) \triangleleft R1(true) \triangleright P)$

R2a and *R2s* are the standard definitions from the UTP book [4]. An issue with these forms is that their definition depends upon *R1* also being satisfied [3], since otherwise the trace minus operator is not well defined. We overcome this with our own version, *R2c*, which applies *R2s* if *R1* holds, and otherwise has no effect. This latter healthiness condition can therefore be reasoned about independently of *R1*, which is useful in some circumstances.

lemma *unrest-ok-R2s* [*unrest*]: $\$ok \# P \Longrightarrow \$ok \# R2s(P)$
by (*simp add: R2s-def unrest*)

lemma *unrest-ok'-R2s* [*unrest*]: $\$ok' \# P \Longrightarrow \$ok' \# R2s(P)$
by (*simp add: R2s-def unrest*)

lemma *unrest-ok-R2c* [*unrest*]: $\$ok \# P \Longrightarrow \$ok \# R2c(P)$
by (*simp add: R2c-def unrest*)

lemma *unrest-ok'-R2c* [*unrest*]: $\$ok' \# P \Longrightarrow \$ok' \# R2c(P)$
by (*simp add: R2c-def unrest*)

lemma *R2s-unrest* [*unrest*]: $\ll vwb\text{-lens } x; x \bowtie in\text{-var } tr; x \bowtie out\text{-var } tr; x \# P \gg \Longrightarrow x \# R2s(P)$
by (*simp add: R2s-def unrest usubst lens-indep-sym*)

lemma *R2s-subst-wait-true* [*usubst*]:
 $(R2s(P))[\$true / \$wait] = R2s(P[\$true / \$wait])$
by (*simp add: R2s-def usubst unrest*)

lemma *R2s-subst-wait'-true* [usubst]:
 $(R2s(P))\llbracket true/\$wait' \rrbracket = R2s(P\llbracket true/\$wait' \rrbracket)$
by (*simp add: R2s-def usubst unrest*)

lemma *R2-subst-wait-true* [usubst]:
 $(R2(P))\llbracket true/\$wait \rrbracket = R2(P\llbracket true/\$wait \rrbracket)$
by (*simp add: R2-def R1-def R2s-def usubst unrest*)

lemma *R2-subst-wait'-true* [usubst]:
 $(R2(P))\llbracket true/\$wait' \rrbracket = R2(P\llbracket true/\$wait' \rrbracket)$
by (*simp add: R2-def R1-def R2s-def usubst unrest*)

lemma *R2-subst-wait-false* [usubst]:
 $(R2(P))\llbracket false/\$wait \rrbracket = R2(P\llbracket false/\$wait \rrbracket)$
by (*simp add: R2-def R1-def R2s-def usubst unrest*)

lemma *R2-subst-wait'-false* [usubst]:
 $(R2(P))\llbracket false/\$wait' \rrbracket = R2(P\llbracket false/\$wait' \rrbracket)$
by (*simp add: R2-def R1-def R2s-def usubst unrest*)

lemma *R2c-R2s-absorb*: $R2c(R2s(P)) = R2s(P)$
by (*rel-auto*)

lemma *R2a-R2s*: $R2a(R2s(P)) = R2s(P)$
by (*rel-auto*)

lemma *R2s-R2a*: $R2s(R2a(P)) = R2a(P)$
by (*rel-auto*)

lemma *R2a-equiv-R2s*: $P \text{ is } R2a \iff P \text{ is } R2s$
by (*metis Healthy-def' R2a-R2s R2s-R2a*)

lemma *R2a-idem*: $R2a(R2a(P)) = R2a(P)$
by (*rel-auto*)

lemma *R2a'-idem*: $R2a'(R2a'(P)) = R2a'(P)$
by (*rel-auto*)

lemma *R2a-mono*: $P \sqsubseteq Q \implies R2a(P) \sqsubseteq R2a(Q)$
by (*rel-blast*)

lemma *R2a'-mono*: $P \sqsubseteq Q \implies R2a'(P) \sqsubseteq R2a'(Q)$
by (*rel-blast*)

lemma *R2a'-weakening*: $R2a'(P) \sqsubseteq P$
apply (*rel-simp*)
apply (*rename-tac ok wait tr more ok' wait' tr' more'*)
apply (*rule-tac x=tr in exI*)
apply (*simp add: diff-add-cancel-left'*)
done

lemma *R2s-idem*: $R2s(R2s(P)) = R2s(P)$
by (*pred-auto*)

lemma *R2-idem*: $R2(R2(P)) = R2(P)$

by (pred-auto)

lemma *R2-mono*: $P \sqsubseteq Q \implies R2(P) \sqsubseteq R2(Q)$
by (pred-auto)

lemma *R2-implies-R1* [closure]: $P \text{ is } R2 \implies P \text{ is } R1$
by (rel-blast)

lemma *R2c-Continuous*: Continuous *R2c*
by (rel-simp)

lemma *R2c-lit*: $R2c(\ll x \gg) = \ll x \gg$
by (rel-auto)

lemma *tr-strict-prefix-R2c-closed* [closure]: $\$tr <_u \tr' is *R2c*
by (rel-auto)

lemma *R2s-conj*: $R2s(P \wedge Q) = (R2s(P) \wedge R2s(Q))$
by (pred-auto)

lemma *R2-conj*: $R2(P \wedge Q) = (R2(P) \wedge R2(Q))$
by (pred-auto)

lemma *R2s-disj*: $R2s(P \vee Q) = (R2s(P) \vee R2s(Q))$
by pred-auto

lemma *R2s-USUP*:
 $R2s(\bigsqcap i \in A \cdot P(i)) = (\bigsqcap i \in A \cdot R2s(P(i)))$
by (simp add: R2s-def usubst)

lemma *R2c-USUP*:
 $R2c(\bigsqcap i \in A \cdot P(i)) = (\bigsqcap i \in A \cdot R2c(P(i)))$
by (rel-auto)

lemma *R2s-UINF*:
 $R2s(\bigsqcup i \in A \cdot P(i)) = (\bigsqcup i \in A \cdot R2s(P(i)))$
by (simp add: R2s-def usubst)

lemma *R2c-UINF*:
 $R2c(\bigsqcup i \in A \cdot P(i)) = (\bigsqcup i \in A \cdot R2c(P(i)))$
by (rel-auto)

lemma *R2-disj*: $R2(P \vee Q) = (R2(P) \vee R2(Q))$
by (pred-auto)

lemma *R2s-not*: $R2s(\neg P) = (\neg R2s(P))$
by pred-auto

lemma *R2s-condr*: $R2s(P \triangleleft b \triangleright Q) = (R2s(P) \triangleleft R2s(b) \triangleright R2s(Q))$
by (rel-auto)

lemma *R2-condr*: $R2(P \triangleleft b \triangleright Q) = (R2(P) \triangleleft R2(b) \triangleright R2(Q))$
by (rel-auto)

lemma *R2-condr'*: $R2(P \triangleleft b \triangleright Q) = (R2(P) \triangleleft R2s(b) \triangleright R2(Q))$

by (*rel-auto*)

lemma *R2s-ok*: $R2s(\$ok) = \ok
by (*rel-auto*)

lemma *R2s-ok'*: $R2s(\$ok') = \ok'
by (*rel-auto*)

lemma *R2s-wait*: $R2s(\$wait) = \$wait$
by (*rel-auto*)

lemma *R2s-wait'*: $R2s(\$wait') = \$wait'$
by (*rel-auto*)

lemma *R2s-true*: $R2s(true) = true$
by *pred-auto*

lemma *R2s-false*: $R2s(false) = false$
by *pred-auto*

lemma *true-is-R2s*:
true is R2s
by (*simp add: Healthy-def R2s-true*)

lemma *R2s-lift-rea*: $R2s(\lceil P \rceil_R) = \lceil P \rceil_R$
by (*simp add: R2s-def usubst unrest*)

lemma *R2c-lift-rea*: $R2c(\lceil P \rceil_R) = \lceil P \rceil_R$
by (*simp add: R2c-def R2s-lift-rea cond-idem usubst unrest*)

lemma *R2c-true*: $R2c(true) = true$
by (*rel-auto*)

lemma *R2c-false*: $R2c(false) = false$
by (*rel-auto*)

lemma *R2c-and*: $R2c(P \wedge Q) = (R2c(P) \wedge R2c(Q))$
by (*rel-auto*)

lemma *conj-R2c-closed* [*closure*]: $\llbracket P \text{ is } R2c; Q \text{ is } R2c \rrbracket \implies (P \wedge Q) \text{ is } R2c$
by (*simp add: Healthy-def R2c-and*)

lemma *R2c-disj*: $R2c(P \vee Q) = (R2c(P) \vee R2c(Q))$
by (*rel-auto*)

lemma *R2c-inf*: $R2c(P \sqcap Q) = (R2c(P) \sqcap R2c(Q))$
by (*rel-auto*)

lemma *R2c-not*: $R2c(\neg P) = (\neg R2c(P))$
by (*rel-auto*)

lemma *R2c-ok*: $R2c(\$ok) = (\$ok)$
by (*rel-auto*)

lemma *R2c-ok'*: $R2c(\$ok') = (\$ok')$

by (rel-auto)

lemma *R2c-wait*: $R2c(\$wait) = \$wait$
by (rel-auto)

lemma *R2c-wait'*: $R2c(\$wait') = \$wait'$
by (rel-auto)

lemma *R2c-wait'-true* [usubst]: $(R2c\ P)[\$true/\$wait'] = R2c(P[\$true/\$wait'])$
by (rel-auto)

lemma *R2c-wait'-false* [usubst]: $(R2c\ P)[\$false/\$wait'] = R2c(P[\$false/\$wait'])$
by (rel-auto)

lemma *R2c-tr'-minus-tr*: $R2c(\$tr' =_u \$tr) = (\$tr' =_u \$tr)$
apply (rel-auto) using minus-zero-eq by blast

lemma *R2c-tr'-ge-tr*: $R2c(\$tr' \geq_u \$tr) = (\$tr' \geq_u \$tr)$
by (rel-auto)

lemma *R2c-tr-less-tr'*: $R2c(\$tr <_u \$tr') = (\$tr <_u \$tr')$
by (rel-auto)

lemma *R2c-condr*: $R2c(P \triangleleft b \triangleright Q) = (R2c(P) \triangleleft R2c(b) \triangleright R2c(Q))$
by (rel-auto)

lemma *R2c-shAll*: $R2c(\forall x \cdot P\ x) = (\forall x \cdot R2c(P\ x))$
by (rel-auto)

lemma *R2c-impl*: $R2c(P \Rightarrow Q) = (R2c(P) \Rightarrow R2c(Q))$
by (metis (no-types, lifting) R2c-and R2c-not double-negation impl-alt-def not-conj-deMorgans)

lemma *R2c-skip-r*: $R2c(II) = II$
proof –
have $R2c(II) = R2c(\$tr' =_u \$tr \wedge II \upharpoonright_{\alpha} tr)$
by (subst skip-r-unfold[of tr], simp-all)
also have $\dots = (R2c(\$tr' =_u \$tr) \wedge II \upharpoonright_{\alpha} tr)$
by (simp add: R2c-and, simp add: R2c-def R2s-def usubst unrest cond-idem)
also have $\dots = (\$tr' =_u \$tr \wedge II \upharpoonright_{\alpha} tr)$
by (simp add: R2c-tr'-minus-tr)
finally show ?thesis
by (subst skip-r-unfold[of tr], simp-all)
qed

lemma *R1-R2c-commute*: $R1(R2c(P)) = R2c(R1(P))$
by (rel-auto)

lemma *R1-R2c-is-R2*: $R1(R2c(P)) = R2(P)$
by (rel-auto)

lemma *R1-R2s-R2c*: $R1(R2s(P)) = R1(R2c(P))$
by (rel-auto)

lemma *R1-R2s-tr-wait*:
 $R1(R2s(\$tr' =_u \$tr \wedge \$wait')) = (\$tr' =_u \$tr \wedge \$wait')$

apply *rel-auto* **using** *minus-zero-eq* **by** *blast*

lemma *R1-R2s-tr'-eq-tr*:

$R1 (R2s (\$tr' =_u \$tr)) = (\$tr' =_u \$tr)$

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

lemma *R1-R2s-tr'-extend-tr*:

$\llbracket \$tr \# v; \$tr' \# v \rrbracket \implies R1 (R2s (\$tr' =_u \$tr \hat{~}_u v)) = (\$tr' =_u \$tr \hat{~}_u v)$

apply (*rel-auto*)

apply (*metis append-minus*)

apply (*simp add: Prefix-Order.prefixI*)

done

lemma *R2-tr-prefix*: $R2(\$tr \leq_u \$tr') = (\$tr \leq_u \$tr')$

by (*pred-auto*)

lemma *R2-form*:

$R2(P) = (\exists tt_0 \cdot P[\llbracket 0/\$tr \rrbracket \llbracket \langle tt_0 \rangle / \$tr' \rrbracket] \wedge \$tr' =_u \$tr + \langle tt_0 \rangle)$

by (*rel-auto*, *metis trace-class.add-diff-cancel-left trace-class.le-iff-add*)

lemma *R2-subst-tr*:

assumes *P is R2*

shows $[\$tr \mapsto_s tr_0, \$tr' \mapsto_s tr_0 + t] \dagger P = [\$tr \mapsto_s 0, \$tr' \mapsto_s t] \dagger P$

proof –

have $[\$tr \mapsto_s tr_0, \$tr' \mapsto_s tr_0 + t] \dagger R2 P = [\$tr \mapsto_s 0, \$tr' \mapsto_s t] \dagger R2 P$

by (*rel-auto*)

thus *?thesis*

by (*simp add: Healthy-if assms*)

qed

lemma *R2-seqr-form*:

shows $(R2(P) ;; R2(Q)) =$

$(\exists tt_1 \cdot \exists tt_2 \cdot ((P[\llbracket 0/\$tr \rrbracket \llbracket \langle tt_1 \rangle / \$tr' \rrbracket] ;; (Q[\llbracket 0/\$tr \rrbracket \llbracket \langle tt_2 \rangle / \$tr' \rrbracket]))$
 $\wedge (\$tr' =_u \$tr + \langle tt_1 \rangle + \langle tt_2 \rangle))$

proof –

have $(R2(P) ;; R2(Q)) = (\exists tr_0 \cdot (R2(P))[\llbracket \langle tr_0 \rangle / \$tr' \rrbracket] ;; (R2(Q))[\llbracket \langle tr_0 \rangle / \$tr' \rrbracket])$

by (*subst seqr-middle[of tr], simp-all*)

also have ... =

$(\exists tr_0 \cdot \exists tt_1 \cdot \exists tt_2 \cdot ((P[\llbracket 0/\$tr \rrbracket \llbracket \langle tt_1 \rangle / \$tr' \rrbracket] \wedge \langle tr_0 \rangle =_u \$tr + \langle tt_1 \rangle) ;;$
 $(Q[\llbracket 0/\$tr \rrbracket \llbracket \langle tt_2 \rangle / \$tr' \rrbracket] \wedge \$tr' =_u \langle tr_0 \rangle + \langle tt_2 \rangle)))$

by (*simp add: R2-form usubst unrest uquant-lift, rel-blast*)

also have ... =

$(\exists tr_0 \cdot \exists tt_1 \cdot \exists tt_2 \cdot ((\langle tr_0 \rangle =_u \$tr + \langle tt_1 \rangle \wedge P[\llbracket 0/\$tr \rrbracket \llbracket \langle tt_1 \rangle / \$tr' \rrbracket] ;;$
 $(Q[\llbracket 0/\$tr \rrbracket \llbracket \langle tt_2 \rangle / \$tr' \rrbracket] \wedge \$tr' =_u \langle tr_0 \rangle + \langle tt_2 \rangle)))$

by (*simp add: conj-comm*)

also have ... =

$(\exists tt_1 \cdot \exists tt_2 \cdot \exists tr_0 \cdot ((P[\llbracket 0/\$tr \rrbracket \llbracket \langle tt_1 \rangle / \$tr' \rrbracket] ;; (Q[\llbracket 0/\$tr \rrbracket \llbracket \langle tt_2 \rangle / \$tr' \rrbracket]))$
 $\wedge \langle tr_0 \rangle =_u \$tr + \langle tt_1 \rangle \wedge \$tr' =_u \langle tr_0 \rangle + \langle tt_2 \rangle)$

by (*rel-blast*)

also have ... =

$(\exists tt_1 \cdot \exists tt_2 \cdot ((P[\llbracket 0/\$tr \rrbracket \llbracket \langle tt_1 \rangle / \$tr' \rrbracket] ;; (Q[\llbracket 0/\$tr \rrbracket \llbracket \langle tt_2 \rangle / \$tr' \rrbracket]))$
 $\wedge (\exists tr_0 \cdot \langle tr_0 \rangle =_u \$tr + \langle tt_1 \rangle \wedge \$tr' =_u \langle tr_0 \rangle + \langle tt_2 \rangle))$

by (*rel-auto*)

also have ... =

$(\exists tt_1 \cdot \exists tt_2 \cdot ((P[\llbracket 0/\$tr \rrbracket \llbracket \langle tt_1 \rangle / \$tr' \rrbracket] ;; (Q[\llbracket 0/\$tr \rrbracket \llbracket \langle tt_2 \rangle / \$tr' \rrbracket]))$

$\wedge (\$tr' =_u \$tr + \langle tt_1 \rangle + \langle tt_2 \rangle)$
 by (rel-auto)
 finally show ?thesis .
 qed

lemma *R2-seqr-form'*:
 assumes *P is R2 Q is R2*
 shows $P ;; Q =$
 $(\exists tt_1 \cdot \exists tt_2 \cdot ((P[0/\$tr][\langle tt_1 \rangle/\$tr'] ;; (Q[0/\$tr][\langle tt_2 \rangle/\$tr'])))$
 $\wedge (\$tr' =_u \$tr + \langle tt_1 \rangle + \langle tt_2 \rangle)$
 using *R2-seqr-form*[of *P Q*] by (simp add: Healthy-if assms)

lemma *R2-seqr-form''*:
 assumes *P is R2 Q is R2*
 shows $P ;; Q =$
 $(\exists (tt_1, tt_2) \cdot ((P[0, \langle tt_1 \rangle/\$tr, \$tr'] ;; (Q[0, \langle tt_2 \rangle/\$tr, \$tr'])))$
 $\wedge (\$tr' =_u \$tr + \langle tt_1 \rangle + \langle tt_2 \rangle)$
 by (subst *R2-seqr-form'*, simp-all add: assms, rel-auto)

lemma *R2-tr-middle*:
 assumes *P is R2 Q is R2*
 shows $(\exists tr_0 \cdot (P[\langle tr_0 \rangle/\$tr'] ;; Q[\langle tr_0 \rangle/\$tr]) \wedge \langle tr_0 \rangle \leq_u \$tr') = (P ;; Q)$
proof –
 have $(P ;; Q) = (\exists tr_0 \cdot (P[\langle tr_0 \rangle/\$tr'] ;; Q[\langle tr_0 \rangle/\$tr]))$
 by (simp add: seqr-middle)
 also have $\dots = (\exists tr_0 \cdot ((R2\ P)[\langle tr_0 \rangle/\$tr'] ;; (R2\ Q)[\langle tr_0 \rangle/\$tr]))$
 by (simp add: assms Healthy-if)
 also have $\dots = (\exists tr_0 \cdot ((R2\ P)[\langle tr_0 \rangle/\$tr'] ;; (R2\ Q)[\langle tr_0 \rangle/\$tr]) \wedge \langle tr_0 \rangle \leq_u \$tr')$
 by (rel-auto)
 also have $\dots = (\exists tr_0 \cdot (P[\langle tr_0 \rangle/\$tr'] ;; Q[\langle tr_0 \rangle/\$tr]) \wedge \langle tr_0 \rangle \leq_u \$tr')$
 by (simp add: assms Healthy-if)
 finally show ?thesis ..
 qed

lemma *R2-seqr-distribute*:
 fixes $P :: ('t::trace, 'α, 'β) \text{rel-rp}$ and $Q :: ('t, 'β, 'γ) \text{rel-rp}$
 shows $R2(R2(P) ;; R2(Q)) = (R2(P) ;; R2(Q))$
proof –
 have $R2(R2(P) ;; R2(Q)) =$
 $((\exists tt_1 \cdot \exists tt_2 \cdot (P[0/\$tr][\langle tt_1 \rangle/\$tr'] ;; Q[0/\$tr][\langle tt_2 \rangle/\$tr'])(\$tr' - \$tr)/\$tr')$
 $\wedge \$tr' - \$tr =_u \langle tt_1 \rangle + \langle tt_2 \rangle) \wedge \$tr' \geq_u \$tr)$
 by (simp add: R2-seqr-form, simp add: R2s-def usubst unrest, rel-auto)
 also have $\dots =$
 $((\exists tt_1 \cdot \exists tt_2 \cdot (P[0/\$tr][\langle tt_1 \rangle/\$tr'] ;; Q[0/\$tr][\langle tt_2 \rangle/\$tr'])(\langle tt_1 \rangle + \langle tt_2 \rangle)/\$tr')$
 $\wedge \$tr' - \$tr =_u \langle tt_1 \rangle + \langle tt_2 \rangle) \wedge \$tr' \geq_u \$tr)$
 by (subst subst-eq-replace, simp)
 also have $\dots =$
 $((\exists tt_1 \cdot \exists tt_2 \cdot (P[0/\$tr][\langle tt_1 \rangle/\$tr'] ;; Q[0/\$tr][\langle tt_2 \rangle/\$tr']$
 $\wedge \$tr' - \$tr =_u \langle tt_1 \rangle + \langle tt_2 \rangle) \wedge \$tr' \geq_u \$tr)$
 by (rel-auto)
 also have $\dots =$
 $(\exists tt_1 \cdot \exists tt_2 \cdot (P[0/\$tr][\langle tt_1 \rangle/\$tr'] ;; Q[0/\$tr][\langle tt_2 \rangle/\$tr']$
 $\wedge (\$tr' - \$tr =_u \langle tt_1 \rangle + \langle tt_2 \rangle \wedge \$tr' \geq_u \$tr))$
 by pred-auto
 also have $\dots =$

$((\exists \text{ } tt_1 \cdot \exists \text{ } tt_2 \cdot (P[0/\$tr][\ll tt_1 \gg / \$tr'] ;; Q[0/\$tr][\ll tt_2 \gg / \$tr'])$
 $\wedge \$tr' =_u \$tr + \ll tt_1 \gg + \ll tt_2 \gg))$
proof –
have $\bigwedge \text{ } tt_1 \text{ } tt_2. (((\$tr' - \$tr =_u \ll tt_1 \gg + \ll tt_2 \gg) \wedge \$tr' \geq_u \$tr) :: ('t, ' \alpha, ' \gamma) \text{ rel-rp})$
 $= (\$tr' =_u \$tr + \ll tt_1 \gg + \ll tt_2 \gg)$
apply (*rel-auto*)
apply (*metis add.assoc diff-add-cancel-left'*)
apply (*simp add: add.assoc*)
apply (*meson le-add order-trans*)
done
thus *?thesis* **by** *simp*
qed
also have $\dots = (R2(P) ;; R2(Q))$
by (*simp add: R2-seqr-form*)
finally show *?thesis* .
qed

lemma *R2-seqr-closure* [*closure*]:
assumes *P is R2 Q is R2*
shows $(P ;; Q) \text{ is } R2$
by (*metis Healthy-def' R2-seqr-distribute assms(1) assms(2)*)

lemma *false-R2* [*closure*]: *false is R2*
by (*rel-auto*)

lemma *R1-R2-commute*:
 $R1(R2(P)) = R2(R1(P))$
by *pred-auto*

lemma *R2-R1-form*: $R2(R1(P)) = R1(R2s(P))$
by (*rel-auto*)

lemma *R2s-H1-commute*:
 $R2s(H1(P)) = H1(R2s(P))$
by (*rel-auto*)

lemma *R2s-H2-commute*:
 $R2s(H2(P)) = H2(R2s(P))$
by (*simp add: H2-split R2s-def usubst*)

lemma *R2-R1-seq-drop-left*:
 $R2(R1(P) ;; R1(Q)) = R2(P ;; R1(Q))$
by (*rel-auto*)

lemma *R2c-idem*: $R2c(R2c(P)) = R2c(P)$
by (*rel-auto*)

lemma *R2c-Idempotent* [*closure*]: *Idempotent R2c*
by (*simp add: Idempotent-def R2c-idem*)

lemma *R2c-Monotonic* [*closure*]: *Monotonic R2c*
by (*rel-auto*)

lemma *R2c-H2-commute*: $R2c(H2(P)) = H2(R2c(P))$
by (*simp add: H2-split R2c-disj R2c-def R2s-def usubst, rel-auto*)

lemma *R2c-seq*: $R2c(R2(P) ;; R2(Q)) = (R2(P) ;; R2(Q))$
 by (metis (no-types, lifting) R1-R2c-commute R1-R2c-is-R2 R2-seqr-distribute R2c-idem)

lemma *R2-R2c-def*: $R2(P) = R1(R2c(P))$
 by (rel-auto)

lemma *R2-comp-def*: $R2 = R1 \circ R2c$
 by (auto simp add: R2-R2c-def)

lemma *R2c-R1-seq*: $R2c(R1(R2c(P)) ;; R1(R2c(Q))) = (R1(R2c(P)) ;; R1(R2c(Q)))$
 using R2c-seq[of P Q] by (simp add: R2-R2c-def)

lemma *R1-R2c-seqr-distribute*:
 fixes $P :: ('t::trace, 'α, 'β) \text{rel-rp}$ and $Q :: ('t, 'β, 'γ) \text{rel-rp}$
 assumes $P \text{ is } R1 \ P \text{ is } R2c \ Q \text{ is } R1 \ Q \text{ is } R2c$
 shows $R1(R2c(P ;; Q)) = P ;; Q$
 by (metis Healthy-if R1-seqr R2c-R1-seq assms)

lemma *R2-R1-true*:
 $R2(R1(true)) = R1(true)$
 by (simp add: R2-R1-form R2s-true)

lemma *R1-true-R2 [closure]*: $R1(true) \text{ is } R2$
 by (rel-auto)

lemma *R1-R2s-R1-true-lemma*:
 $R1(R2s(R1(\neg R2s P) ;; R1 true)) = R1(R2s((\neg P) ;; R1 true))$
 by (rel-auto)

lemma *R2c-healthy-R2s*: $P \text{ is } R2c \implies R1(R2s(P)) = R1(P)$
 by (simp add: Healthy-def R1-R2s-R2c)

2.3 R3: No activity while predecessor is waiting

definition $R3 :: ('t::trace, 'α) \text{hrel-rp} \Rightarrow ('t, 'α) \text{hrel-rp}$ where
 $[upred-defs]: R3(P) = (II \triangleleft \$wait \triangleright P)$

lemma *R3-idem*: $R3(R3(P)) = R3(P)$
 by (rel-auto)

lemma *R3-Idempotent [closure]*: *Idempotent* $R3$
 by (simp add: Idempotent-def R3-idem)

lemma *R3-mono*: $P \sqsubseteq Q \implies R3(P) \sqsubseteq R3(Q)$
 by (rel-auto)

lemma *R3-Monotonic*: *Monotonic* $R3$
 by (simp add: mono-def R3-mono)

lemma *R3-Continuous*: *Continuous* $R3$
 by (rel-auto)

lemma *R3-conj*: $R3(P \wedge Q) = (R3(P) \wedge R3(Q))$
 by (rel-auto)

lemma *R3-disj*: $R3(P \vee Q) = (R3(P) \vee R3(Q))$
by (*rel-auto*)

lemma *R3-USUP*:
assumes $A \neq \{\}$
shows $R3(\prod i \in A \cdot P(i)) = (\prod i \in A \cdot R3(P(i)))$
using *assms* **by** (*rel-auto*)

lemma *R3-UINF*:
assumes $A \neq \{\}$
shows $R3(\bigsqcup i \in A \cdot P(i)) = (\bigsqcup i \in A \cdot R3(P(i)))$
using *assms* **by** (*rel-auto*)

lemma *R3-condr*: $R3(P \triangleleft b \triangleright Q) = (R3(P) \triangleleft b \triangleright R3(Q))$
by (*rel-auto*)

lemma *R3-skipr*: $R3(II) = II$
by (*rel-auto*)

lemma *R3-form*: $R3(P) = ((\$wait \wedge II) \vee (\neg \$wait \wedge P))$
by (*rel-auto*)

lemma *wait-R3*:
 $(\$wait \wedge R3(P)) = (II \wedge \$wait')$
by (*rel-auto*)

lemma *nwait-R3*:
 $(\neg \$wait \wedge R3(P)) = (\neg \$wait \wedge P)$
by (*rel-auto*)

lemma *R3-semir-form*:
 $(R3(P) ;; R3(Q)) = R3(P ;; R3(Q))$
by (*rel-auto*)

lemma *R3-semir-closure*:
assumes P is *R3* Q is *R3*
shows $(P ;; Q)$ is *R3*
using *assms*
by (*metis Healthy-def' R3-semir-form*)

lemma *R1-R3-commute*: $R1(R3(P)) = R3(R1(P))$
by (*rel-auto*)

lemma *R2-R3-commute*: $R2(R3(P)) = R3(R2(P))$
apply (*rel-auto*)
using *minus-zero-eq* **apply** *blast+*
done

2.4 R4: The trace strictly increases

definition $R4 :: ('t::trace, 'a, 'b) \text{ rel-rp} \Rightarrow ('t, 'a, 'b) \text{ rel-rp}$ **where**
 $[upred-defs]: R4(P) = (P \wedge \$tr <_u \$tr')$

lemma *R4-implies-R1* [*closure*]: P is *R4* $\implies P$ is *R1*
using *less-iff* **by** *rel-blast*

lemma *R4-iff-refine*:

$P \text{ is } R4 \iff (\$tr <_u \$tr') \sqsubseteq P$
by (*rel-blast*)

lemma *R4-idem*: $R4 (R4 P) = R4 P$

by (*rel-auto*)

lemma *R4-false*: $R4(false) = false$

by (*rel-auto*)

lemma *R4-conj*: $R4(P \wedge Q) = (R4(P) \wedge R4(Q))$

by (*rel-auto*)

lemma *R4-disj*: $R4(P \vee Q) = (R4(P) \vee R4(Q))$

by (*rel-auto*)

lemma *R4-is-R4 [closure]*: $R4(P) \text{ is } R4$

by (*rel-auto*)

lemma *false-R4 [closure]*: $false \text{ is } R4$

by (*rel-auto*)

lemma *UINF-R4-closed [closure]*:

$\llbracket \bigwedge i. P \text{ is } R4 \rrbracket \implies (\bigcap i \cdot P \text{ is } R4$
by (*rel-blast*)

lemma *conj-R4-closed [closure]*:

$\llbracket P \text{ is } R4; Q \text{ is } R4 \rrbracket \implies (P \wedge Q) \text{ is } R4$
by (*simp add: Healthy-def R4-conj*)

lemma *disj-R4-closed [closure]*:

$\llbracket P \text{ is } R4; Q \text{ is } R4 \rrbracket \implies (P \vee Q) \text{ is } R4$
by (*simp add: Healthy-def R4-disj*)

lemma *seq-R4-closed-1 [closure]*:

$\llbracket P \text{ is } R4; Q \text{ is } R1 \rrbracket \implies (P ;; Q) \text{ is } R4$
using *less-le-trans* **by** *rel-blast*

lemma *seq-R4-closed-2 [closure]*:

$\llbracket P \text{ is } R1; Q \text{ is } R4 \rrbracket \implies (P ;; Q) \text{ is } R4$
using *le-less-trans* **by** *rel-blast*

2.5 R5: The trace does not increase

definition *R5* :: $(t::trace, '\alpha, '\beta) \text{ rel-rp} \Rightarrow (t, '\alpha, '\beta) \text{ rel-rp}$ **where**
[upred-defs]: $R5(P) = (P \wedge \$tr =_u \$tr')$

lemma *R5-implies-R1 [closure]*: $P \text{ is } R5 \implies P \text{ is } R1$

using *eq-iff* **by** *rel-blast*

lemma *R5-iff-refine*:

$P \text{ is } R5 \iff (\$tr =_u \$tr') \sqsubseteq P$
by (*rel-blast*)

lemma *R5-conj*: $R5(P \wedge Q) = (R5(P) \wedge R5(Q))$

by (*rel-auto*)

lemma *R5-disj*: $R5(P \vee Q) = (R5(P) \vee R5(Q))$
by (*rel-auto*)

lemma *R4-R5*: $R4(R5 P) = false$
by (*rel-auto*)

lemma *R5-R4*: $R5(R4 P) = false$
by (*rel-auto*)

lemma *UINF-R5-closed* [*closure*]:
 $\llbracket \bigwedge i. P i \text{ is } R5 \rrbracket \implies (\bigcap i. P i) \text{ is } R5$
by (*rel-blast*)

lemma *conj-R5-closed* [*closure*]:
 $\llbracket P \text{ is } R5; Q \text{ is } R5 \rrbracket \implies (P \wedge Q) \text{ is } R5$
by (*simp add: Healthy-def R5-conj*)

lemma *disj-R5-closed* [*closure*]:
 $\llbracket P \text{ is } R5; Q \text{ is } R5 \rrbracket \implies (P \vee Q) \text{ is } R5$
by (*simp add: Healthy-def R5-disj*)

lemma *seq-R5-closed* [*closure*]:
 $\llbracket P \text{ is } R5; Q \text{ is } R5 \rrbracket \implies (P ;; Q) \text{ is } R5$
by (*rel-auto, metis*)

2.6 RP laws

definition *RP-def* [*upred-defs*]: $RP(P) = R1(R2c(R3(P)))$

lemma *RP-comp-def*: $RP = R1 \circ R2c \circ R3$
by (*auto simp add: RP-def*)

lemma *RP-alt-def*: $RP(P) = R1(R2(R3(P)))$
by (*metis R1-R2c-is-R2 R1-idem RP-def*)

lemma *RP-intro*: $\llbracket P \text{ is } R1; P \text{ is } R2; P \text{ is } R3 \rrbracket \implies P \text{ is } RP$
by (*simp add: Healthy-def' RP-alt-def*)

lemma *RP-idem*: $RP(RP(P)) = RP(P)$
by (*simp add: R1-R2c-is-R2 R2-R3-commute R2-idem R3-idem RP-def*)

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

lemma *RP-mono*: $P \sqsubseteq Q \implies RP(P) \sqsubseteq RP(Q)$
by (*simp add: R1-R2c-is-R2 R2-mono R3-mono RP-def*)

lemma *RP-Monotonic*: *Monotonic* *RP*
by (*simp add: mono-def RP-mono*)

lemma *RP-Continuous*: *Continuous* *RP*
by (*simp add: Continuous-comp R1-Continuous R2c-Continuous R3-Continuous RP-comp-def*)

lemma *RP-skip*:
 $RP(II) = II$

by (simp add: R1-skip R2c-skip-r R3-skipr RP-def)

lemma *RP-skip-closure*:

II is RP

by (simp add: Healthy-def' RP-skip)

lemma *RP-seq-closure*:

assumes *P is RP Q is RP*

shows *(P ;; Q) is RP*

proof (rule *RP-intro*)

show *(P ;; Q) is R1*

by (metis Healthy-def R1-seqr RP-def assms)

thus *(P ;; Q) is R2*

by (metis Healthy-def' R2-R2c-def R2c-R1-seq RP-def assms)

show *(P ;; Q) is R3*

by (metis (no-types, lifting) Healthy-def' R1-R2c-is-R2 R2-R3-commute R3-idem R3-semir-form RP-def assms)

qed

2.7 UTP theories

typeddecl *REA*

abbreviation $REA \equiv UTHY(REA, ('t::trace, 'α) rp)$

overloading

rea-hcond == utp-hcond :: (REA, ('t::trace, 'α) rp) uthy \Rightarrow (('t, 'α) rp \times ('t, 'α) rp) health

rea-unit == utp-unit :: (REA, ('t::trace, 'α) rp) uthy \Rightarrow ('t, 'α) hrel-rp

begin

definition *rea-hcond* :: $(REA, ('t::trace, 'α) rp) uthy \Rightarrow (('t, 'α) rp \times ('t, 'α) rp) health$

where [upred-defs]: *rea-hcond* *T* = *RP*

definition *rea-unit* :: $(REA, ('t::trace, 'α) rp) uthy \Rightarrow ('t, 'α) hrel-rp$

where [upred-defs]: *rea-unit* *T* = *II*

end

interpretation *rea-utp-theory*: *utp-theory* $UTHY(REA, ('t::trace, 'α) rp)$

rewrites *carrier* (*uthy-order* *REA*) = $\llbracket RP \rrbracket_H$

by (simp-all add: *rea-hcond-def utp-theory-def RP-idem*)

interpretation *rea-utp-theory-mono*: *utp-theory-continuous* $UTHY(REA, ('t::trace, 'α) rp)$

rewrites *carrier* (*uthy-order* *REA*) = $\llbracket RP \rrbracket_H$

by (unfold-locales, simp-all add: *RP-Continuous rea-hcond-def*)

interpretation *rea-utp-theory-rel*: *utp-theory-unital* $UTHY(REA, ('t::trace, 'α) rp)$

rewrites *carrier* (*uthy-order* *REA*) = $\llbracket RP \rrbracket_H$

by (unfold-locales, simp-all add: *rea-hcond-def rea-unit-def RP-seq-closure RP-skip-closure*)

lemma *rea-top*: $\top_{REA} = (\$wait \wedge II)$

proof –

have $\top_{REA} = RP(false)$

by (simp add: *rea-utp-theory-mono.healthy-top*, simp add: *rea-hcond-def*)

also have $\dots = (\$wait \wedge II)$

by (rel-auto, metis minus-zero-eq)

finally show *?thesis* .

qed

lemma *rea-top-left-zero*:

```

assumes  $P$  is  $RP$ 
shows  $(\top_{REA} ;; P) = \top_{REA}$ 
proof –
  have  $(\top_{REA} ;; P) = ((\$wait \wedge II) ;; R3(P))$ 
    by (metis (no-types, lifting) Healthy-def R1-R2c-is-R2 R2-R3-commute R3-idem RP-def assms
rea-top)
  also have  $\dots = (\$wait \wedge R3(P))$ 
    by (rel-auto)
  also have  $\dots = (\$wait \wedge II)$ 
    by (metis R3-skipr wait-R3)
  also have  $\dots = \top_{REA}$ 
    by (simp add: rea-top)
  finally show ?thesis .
qed

```

lemma *rea-bottom*: $\perp_{REA} = R1(\$wait \Rightarrow II)$

```

proof –
  have  $\perp_{REA} = RP(true)$ 
    by (simp add: rea-utp-theory-mono.healthy-bottom, simp add: rea-hcond-def)
  also have  $\dots = R1(\$wait \Rightarrow II)$ 
    by (rel-auto, metis minus-zero-eq)
  finally show ?thesis .
qed

```

end

3 Reactive Parallel-by-Merge

```

theory utp-rea-parallel
imports utp-rea-healths
begin

```

We show closure of parallel by merge under the reactive healthiness conditions by means of suitable restrictions on the merge predicate. We first define healthiness conditions for R1 and R2 merge predicates.

definition $R1m :: ('t :: trace, 'α) \text{rp merge} \Rightarrow ('t, 'α) \text{rp merge}$
where [*upred-defs*]: $R1m(M) = (M \wedge \$tr_{<} \leq_u \$tr')$

definition $R1m' :: ('t :: trace, 'α) \text{rp merge} \Rightarrow ('t, 'α) \text{rp merge}$
where [*upred-defs*]: $R1m'(M) = (M \wedge \$tr_{<} \leq_u \$tr' \wedge \$tr_{<} \leq_u \$0-tr \wedge \$tr_{<} \leq_u \$1-tr)$

A merge predicate can access the history through *tr*, as usual, but also through *0.tr* and *1.tr*. Thus we have to remove the latter two histories as well to satisfy R2 for the overall construction.

term $M \llbracket 0, x, k/y, z, a \rrbracket$

term $M \llbracket 0, \$tr' - \$tr_{<}, \$0-tr - \$tr_{<}, \$1-tr - \$tr_{<}/\$tr_{<}, \$tr', \$0-tr, \$1-tr \rrbracket$

definition $R2m :: ('t :: trace, 'α) \text{rp merge} \Rightarrow ('t, 'α) \text{rp merge}$
where [*upred-defs*]: $R2m(M) = R1m(M \llbracket 0, \$tr' - \$tr_{<}, \$0-tr - \$tr_{<}, \$1-tr - \$tr_{<}/\$tr_{<}, \$tr', \$0-tr, \$1-tr \rrbracket)$

definition $R2m' :: ('t :: trace, 'α) \text{rp merge} \Rightarrow ('t, 'α) \text{rp merge}$
where [*upred-defs*]: $R2m'(M) = R1m'(M \llbracket 0, \$tr' - \$tr_{<}, \$0-tr - \$tr_{<}, \$1-tr - \$tr_{<}/\$tr_{<}, \$tr', \$0-tr, \$1-tr \rrbracket)$

definition $R2cm :: ('t :: trace, 'α) \text{rp merge} \Rightarrow ('t, 'α) \text{rp merge}$

where $[upred-defs]: R2cm(M) = M \llbracket 0, \$tr' - \$tr_<, \$0-tr - \$tr_<, \$1-tr - \$tr_< / \$tr_<, \$tr', \$0-tr, \$1-tr \rrbracket$
 $\triangleleft \$tr_< \leq_u \$tr' \triangleright M$

lemma $R2m'$ -form:

$R2m'(M) =$
 $(\exists (tt_p, tt_0, tt_1) \cdot M \llbracket 0, \langle\langle tt_p \rangle\rangle, \langle\langle tt_0 \rangle\rangle, \langle\langle tt_1 \rangle\rangle / \$tr_<, \$tr', \$0-tr, \$1-tr \rrbracket$
 $\wedge \$tr' =_u \$tr_< + \langle\langle tt_p \rangle\rangle$
 $\wedge \$0-tr =_u \$tr_< + \langle\langle tt_0 \rangle\rangle$
 $\wedge \$1-tr =_u \$tr_< + \langle\langle tt_1 \rangle\rangle)$
by (*rel-auto*, *metis diff-add-cancel-left'*)

lemma $R1m$ -idem: $R1m(R1m(P)) = R1m(P)$

by (*rel-auto*)

lemma $R1m$ -seq-lemma: $R1m(R1m(M) ;; R1(P)) = R1m(M) ;; R1(P)$

by (*rel-auto*)

lemma $R1m$ -seq [closure]:

assumes M is $R1m$ P is $R1$

shows $M ;; P$ is $R1m$

proof –

from *assms* **have** $R1m(M ;; P) = R1m(R1m(M) ;; R1(P))$

by (*simp add: Healthy-if*)

also have $\dots = R1m(M) ;; R1(P)$

by (*simp add: R1m-seq-lemma*)

also have $\dots = M ;; P$

by (*simp add: Healthy-if assms*)

finally show *?thesis*

by (*simp add: Healthy-def*)

qed

lemma $R2m$ -idem: $R2m(R2m(P)) = R2m(P)$

by (*rel-auto*)

lemma $R2m$ -seq-lemma: $R2m'(R2m'(M) ;; R2(P)) = R2m'(M) ;; R2(P)$

apply (*simp add: R2m'-form R2-form*)

apply (*rel-auto*)

apply (*metis (no-types, lifting) add.assoc*)

done

lemma $R2m'$ -seq [closure]:

assumes M is $R2m'$ P is $R2$

shows $M ;; P$ is $R2m'$

by (*metis Healthy-def' R2m-seq-lemma assms(1) assms(2)*)

lemma $R1$ -par-by-merge [closure]:

M is $R1m \implies (P \parallel_M Q)$ is $R1$

by (*rel-blast*)

lemma $R2$ - $R2m'$ -pbm: $R2(P \parallel_M Q) = (R2(P) \parallel_{R2m'(M)} R2(Q))$

proof –

have $(R2(P) \parallel_{R2m'(M)} R2(Q)) = ((R2(P) \parallel_s R2(Q)) ;;$

$(\exists (tt_p, tt_0, tt_1) \cdot M \llbracket 0, \langle\langle tt_p \rangle\rangle, \langle\langle tt_0 \rangle\rangle, \langle\langle tt_1 \rangle\rangle / \$tr_<, \$tr', \$0-tr, \$1-tr \rrbracket$
 $\wedge \$tr' =_u \$tr_< + \langle\langle tt_p \rangle\rangle$
 $\wedge \$0-tr =_u \$tr_< + \langle\langle tt_0 \rangle\rangle$

$\wedge \$1-tr =_u \$tr_{<} + \langle tt_1 \rangle)$

by (*simp add: par-by-merge-def R2m'-form*)

also have ... = $(\exists (tt_p, tt_0, tt_1) \cdot ((R2(P) \parallel_s R2(Q)) \;; (M[0, \langle tt_p \rangle, \langle tt_0 \rangle, \langle tt_1 \rangle] / \$tr_{<}, \$tr', \$0-tr, \$1-tr]$
 $\wedge \$tr' =_u \$tr_{<} + \langle tt_p \rangle$
 $\wedge \$0-tr =_u \$tr_{<} + \langle tt_0 \rangle$
 $\wedge \$1-tr =_u \$tr_{<} + \langle tt_1 \rangle)))$

by (*rel-blast*)

also have ... = $(\exists (tt_p, tt_0, tt_1) \cdot (((R2(P) \parallel_s R2(Q)) \wedge \$0-tr' =_u \$tr_{<} + \langle tt_0 \rangle \wedge \$1-tr' =_u$
 $\$tr_{<} + \langle tt_1 \rangle) \;;$
 $(M[0, \langle tt_p \rangle, \langle tt_0 \rangle, \langle tt_1 \rangle] / \$tr_{<}, \$tr', \$0-tr, \$1-tr] \wedge \$tr' =_u \$tr_{<} +$
 $\langle tt_p \rangle)))$

by (*rel-blast*)

also have ... = $(\exists (tt_p, tt_0, tt_1) \cdot (((R2(P) \parallel_s R2(Q)) \wedge \$0-tr' =_u \$tr_{<} + \langle tt_0 \rangle \wedge \$1-tr' =_u$
 $\$tr_{<} + \langle tt_1 \rangle) \;;$
 $(M[0, \langle tt_p \rangle, \langle tt_0 \rangle, \langle tt_1 \rangle] / \$tr_{<}, \$tr', \$0-tr, \$1-tr]) \wedge \$tr' =_u \$tr_{<} +$
 $\langle tt_p \rangle)$

by (*rel-blast*)

also have ... = $(\exists (tt_p, tt_0, tt_1) \cdot (((R2(P) \wedge \$tr' =_u \$tr + \langle tt_0 \rangle) \parallel_s (R2(Q) \wedge \$tr' =_u \$tr +$
 $\langle tt_1 \rangle)) \;;$
 $(M[0, \langle tt_p \rangle, \langle tt_0 \rangle, \langle tt_1 \rangle] / \$tr_{<}, \$tr', \$0-tr, \$1-tr]) \wedge \$tr' =_u \$tr_{<} +$
 $\langle tt_p \rangle)$

by (*rel-auto, blast, metis le-add trace-class.add-diff-cancel-left*)

also have ... = $(\exists (tt_p, tt_0, tt_1) \cdot (((\exists tt_0' \cdot P[0, \langle tt_0' \rangle] / \$tr, \$tr'] \wedge \$tr' =_u \$tr + \langle tt_0' \rangle) \wedge$
 $\$tr' =_u \$tr + \langle tt_0 \rangle)$
 $\parallel_s ((\exists tt_1' \cdot Q[0, \langle tt_1' \rangle] / \$tr, \$tr'] \wedge \$tr' =_u \$tr + \langle tt_1' \rangle) \wedge \$tr' =_u$
 $\$tr + \langle tt_1 \rangle)) \;;$
 $(M[0, \langle tt_p \rangle, \langle tt_0 \rangle, \langle tt_1 \rangle] / \$tr_{<}, \$tr', \$0-tr, \$1-tr]) \wedge \$tr' =_u \$tr_{<} +$
 $\langle tt_p \rangle)$

by (*simp add: R2-form usubst*)

also have ... = $(\exists (tt_p, tt_0, tt_1) \cdot (((P[0, \langle tt_0 \rangle] / \$tr, \$tr'] \wedge \$tr' =_u \$tr + \langle tt_0 \rangle)$
 $\parallel_s (Q[0, \langle tt_1 \rangle] / \$tr, \$tr'] \wedge \$tr' =_u \$tr + \langle tt_1 \rangle)) \;;$
 $(M[0, \langle tt_p \rangle, \langle tt_0 \rangle, \langle tt_1 \rangle] / \$tr_{<}, \$tr', \$0-tr, \$1-tr]) \wedge \$tr' =_u \$tr_{<} +$
 $\langle tt_p \rangle)$

by (*rel-auto, metis left-cancel-monoid-class.add-left-imp-eq, blast*)

also have ... = $R2(P \parallel_M Q)$

by (*rel-auto, blast, metis diff-add-cancel-left'*)

finally show ?thesis ..

qed

lemma *R2m-R2m'-pbm*: $(R2(P) \parallel_{R2m(M)} R2(Q)) = (R2(P) \parallel_{R2m'(M)} R2(Q))$

by (*rel-blast*)

lemma *R2-par-by-merge* [closure]:

assumes *P is R2 Q is R2 M is R2m*

shows $(P \parallel_M Q)$ is *R2*

by (*metis Healthy-def' R2-R2m'-pbm R2m-R2m'-pbm assms(1) assms(2) assms(3)*)

lemma *R2-par-by-merge'* [closure]:

assumes *P is R2 Q is R2 M is R2m'*

shows $(P \parallel_M Q)$ is *R2*

by (*metis Healthy-def' R2-R2m'-pbm assms(1) assms(2) assms(3)*)

lemma *R1m-skip-merge*: $R1m(skip_m) = skip_m$

by (*rel-auto*)

lemma *R1m-disj*: $R1m(P \vee Q) = (R1m(P) \vee R1m(Q))$
by (*rel-auto*)

lemma *R1m-conj*: $R1m(P \wedge Q) = (R1m(P) \wedge R1m(Q))$
by (*rel-auto*)

lemma *R2m-skip-merge*: $R2m(skip_m) = skip_m$
apply (*rel-auto*) **using** *minus-zero-eq* **by** *blast*

lemma *R2m-disj*: $R2m(P \vee Q) = (R2m(P) \vee R2m(Q))$
by (*rel-auto*)

lemma *R2m-conj*: $R2m(P \wedge Q) = (R2m(P) \wedge R2m(Q))$
by (*rel-auto*)

definition *R3m* :: $('t :: trace, 'a) \text{ rp merge} \Rightarrow ('t, 'a) \text{ rp merge}$ **where**
 $[upred-defs]: R3m(M) = skip_m \triangleleft \$wait_{<} \triangleright M$

lemma *R3-par-by-merge*:

assumes

P is R3 Q is R3 M is R3m

shows $(P \parallel_M Q) \text{ is } R3$

proof –

have $(P \parallel_M Q) = ((P \parallel_M Q)[\![true/\$wait]\!] \triangleleft \$wait \triangleright (P \parallel_M Q))$

by (*metis cond-L6 cond-var-split in-var-uvar wait-vwb-lens*)

also have $\dots = (((R3\ P)[\![true/\$wait]\!] \parallel (R3m\ M)[\![true/\$wait_{<}\!]\!] (R3\ Q)[\![true/\$wait]\!] \triangleleft \$wait \triangleright (P \parallel_M Q))$

by (*subst-tac, simp add: Healthy-if assms*)

also have $\dots = ((II[\![true/\$wait]\!] \parallel skip_m[\![true/\$wait_{<}\!]\!] II[\![true/\$wait]\!] \triangleleft \$wait \triangleright (P \parallel_M Q))$

by (*simp add: R3-def R3m-def usubst*)

also have $\dots = ((II \parallel skip_m\ II)[\![true/\$wait]\!] \triangleleft \$wait \triangleright (P \parallel_M Q))$

by (*subst-tac*)

also have $\dots = (II \triangleleft \$wait \triangleright (P \parallel_M Q))$

by (*simp add: cond-var-subst-left par-by-merge-skip*)

also have $\dots = R3(P \parallel_M Q)$

by (*simp add: R3-def*)

finally show *?thesis*

by (*simp add: Healthy-def*)

qed

lemma *SymMerge-R1-true* [*closure*]:

$M \text{ is } SymMerge \implies M ;; R1(true) \text{ is } SymMerge$

by (*rel-auto*)

end

4 Reactive Relations

theory *utp-rea-rel*

imports

utp-rea-healths

UTP-KAT.utp-kleene

begin

This theory defines a predicate calculus for R1-R2 predicates as an extension of the standard

alphabetised predicate calculus.

4.1 Healthiness Conditions

definition $RR :: ('t::trace, 'α, 'β) rel-rp \Rightarrow ('t, 'α, 'β) rel-rp$ **where**
 $[upred-defs]: RR(P) = (\exists \{ \$ok, \$ok', \$wait, \$wait' \} \cdot R2(P))$

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

lemma $RR-Idempotent$ [$closure$]: $Idempotent\ RR$
by ($simp\ add: Idempotent-def\ RR-idem$)

lemma $RR-Continuous$ [$closure$]: $Continuous\ RR$
by ($rel-blast$)

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

lemma $R2c-RR: R2c(RR(P)) = RR(P)$
by ($rel-auto$)

lemma $RR-implies-R1$ [$closure$]: $P\ is\ RR \Longrightarrow P\ is\ R1$
by ($metis\ Healthy-def\ R1-RR$)

lemma $RR-implies-R2c: P\ is\ RR \Longrightarrow P\ is\ R2c$
by ($metis\ Healthy-def\ R2c-RR$)

lemma $RR-implies-R2$ [$closure$]: $P\ is\ RR \Longrightarrow P\ is\ R2$
by ($metis\ Healthy-def\ R1-RR\ R2-R2c-def\ R2c-RR$)

lemma $RR-intro$:
assumes $\$ok \# P\ \$ok' \# P\ \$wait \# P\ \$wait' \# P\ P\ is\ R1\ P\ is\ R2c$
shows $P\ is\ RR$
by ($simp\ add: RR-def\ Healthy-def\ ex-plus\ R2-R2c-def, simp\ add: Healthy-if\ assms\ ex-unrest$)

lemma $RR-R2-intro$:
assumes $\$ok \# P\ \$ok' \# P\ \$wait \# P\ \$wait' \# P\ P\ is\ R2$
shows $P\ is\ RR$
by ($simp\ add: RR-def\ Healthy-def\ ex-plus, simp\ add: Healthy-if\ assms\ ex-unrest$)

lemma $RR-unrests$ [$unrest$]:
assumes $P\ is\ RR$
shows $\$ok \# P\ \$ok' \# P\ \$wait \# P\ \$wait' \# P$

proof –
have $\$ok \# RR(P)\ \$ok' \# RR(P)\ \$wait \# RR(P)\ \$wait' \# RR(P)$
by ($simp-all\ add: RR-def\ ex-plus\ unrest$)
thus $\$ok \# P\ \$ok' \# P\ \$wait \# P\ \$wait' \# P$
by ($simp-all\ add: assms\ Healthy-if$)

qed

lemma $RR-refine-intro$:
assumes $P\ is\ RR\ Q\ is\ RR \wedge t. P[[0, \langle t \rangle / \$tr, \$tr']] \sqsubseteq Q[[0, \langle t \rangle / \$tr, \$tr']]$
shows $P \sqsubseteq Q$
proof –

have $\bigwedge t. (RR\ P) \llbracket 0, \langle t \rangle / \$tr, \$tr' \rrbracket \sqsubseteq (RR\ Q) \llbracket 0, \langle t \rangle / \$tr, \$tr' \rrbracket$
 by (simp add: Healthy-if assms)
 hence $RR(P) \sqsubseteq RR(Q)$
 by (rel-auto)
 thus ?thesis
 by (simp add: Healthy-if assms)
 qed

lemma *R4-RR-closed* [closure]:
 assumes P is RR
 shows $R4(P)$ is RR
proof –
 have $R4(RR(P))$ is RR
 by (rel-blast)
 thus ?thesis
 by (simp add: Healthy-if assms)
 qed

lemma *R5-RR-closed* [closure]:
 assumes P is RR
 shows $R5(P)$ is RR
proof –
 have $R5(RR(P))$ is RR
 using minus-zero-eq by rel-auto
 thus ?thesis
 by (simp add: Healthy-if assms)
 qed

4.2 Reactive relational operators

named-theorems *rpred*

abbreviation *rea-true* :: $('t::trace, 'α, 'β)$ *rel-rp* ($true_r$) **where**
 $true_r \equiv R1(true)$

definition *rea-not* :: $('t::trace, 'α, 'β)$ *rel-rp* \Rightarrow $('t, 'α, 'β)$ *rel-rp* $(\neg_r - [40] 40)$
where [*upred-defs*]: $(\neg_r P) = R1(\neg P)$

definition *rea-diff* :: $('t::trace, 'α, 'β)$ *rel-rp* \Rightarrow $('t, 'α, 'β)$ *rel-rp* \Rightarrow $('t, 'α, 'β)$ *rel-rp* (**infixl** \neg_r 65)
where [*upred-defs*]: $rea_diff\ P\ Q = (P \wedge \neg_r Q)$

definition *rea-impl* ::
 $('t::trace, 'α, 'β)$ *rel-rp* \Rightarrow $('t, 'α, 'β)$ *rel-rp* \Rightarrow $('t, 'α, 'β)$ *rel-rp* (**infixr** \Rightarrow_r 25)
where [*upred-defs*]: $(P \Rightarrow_r Q) = (\neg_r P \vee Q)$

definition *rea-lift* :: $('t::trace, 'α, 'β)$ *rel-rp* \Rightarrow $('t, 'α, 'β)$ *rel-rp* $([-]_r)$
where [*upred-defs*]: $[P]_r = R1(P)$

definition *rea-skip* :: $('t::trace, 'α)$ *hrel-rp* (II_r)
where [*upred-defs*]: $II_r = (\$tr' =_u \$tr \wedge \$\Sigma_R' =_u \$\Sigma_R)$

definition *rea-assert* :: $('t::trace, 'α)$ *hrel-rp* \Rightarrow $('t, 'α)$ *hrel-rp* $(\{-\}_r)$
where [*upred-defs*]: $\{b\}_r = (II_r \vee \neg_r b)$

Trace contribution substitution: make a substitution for the trace contribution lens tt , and apply $R1$ to make the resulting predicate healthy again.

definition *rea-subst* :: (*t*::*trace*, *'α*) *hrel-rp* ⇒ (*t*, (*t*, *'α*) *rp*) *hexpr* ⇒ (*t*, *'α*) *hrel-rp* (*-*⌊*-*⌋_{*r*} [999,0]
999)
where [*upred-defs*]: *P*⌊*v*⌋_{*r*} = *R1*(*P*⌊*v* & *tt*⌋)

4.3 Unrestriction and substitution laws

lemma *rea-true-unrest* [*unrest*]:

⌊ *x* ⋈ (*\$tr*)_{*v*}; *x* ⋈ (*\$tr'*)_{*v*} ⌋ ⇒ *x* # *true*_{*r*}
by (*simp add: R1-def unrest lens-indep-sym*)

lemma *rea-not-unrest* [*unrest*]:

⌊ *x* ⋈ (*\$tr*)_{*v*}; *x* ⋈ (*\$tr'*)_{*v*}; *x* # *P* ⌋ ⇒ *x* # ¬_{*r*} *P*
by (*simp add: rea-not-def R1-def unrest lens-indep-sym*)

lemma *rea-impl-unrest* [*unrest*]:

⌊ *x* ⋈ (*\$tr*)_{*v*}; *x* ⋈ (*\$tr'*)_{*v*}; *x* # *P*; *x* # *Q* ⌋ ⇒ *x* # (*P* ⇒_{*r*} *Q*)
by (*simp add: rea-impl-def unrest*)

lemma *rea-true-usubst* [*usubst*]:

⌊ *\$tr* # *σ*; *\$tr'* # *σ* ⌋ ⇒ *σ* † *true*_{*r*} = *true*_{*r*}
by (*simp add: R1-def usubst*)

lemma *rea-not-usubst* [*usubst*]:

⌊ *\$tr* # *σ*; *\$tr'* # *σ* ⌋ ⇒ *σ* † (¬_{*r*} *P*) = (¬_{*r*} *σ* † *P*)
by (*simp add: rea-not-def R1-def usubst*)

lemma *rea-impl-usubst* [*usubst*]:

⌊ *\$tr* # *σ*; *\$tr'* # *σ* ⌋ ⇒ *σ* † (*P* ⇒_{*r*} *Q*) = (*σ* † *P* ⇒_{*r*} *σ* † *Q*)
by (*simp add: rea-impl-def usubst R1-def*)

lemma *rea-true-usubst-tt* [*usubst*]:

R1(*true*)⌊*e* & *tt*⌋ = *true*
by (*rel-simp*)

lemma *unrest-rea-subst* [*unrest*]:

⌊ *mwb-lens* *x*; *x* ⋈ (*\$tr*)_{*v*}; *x* ⋈ (*\$tr'*)_{*v*}; *x* # *v*; *x* # *P* ⌋ ⇒ *x* # *P*⌊*v*⌋_{*r*}
by (*simp add: rea-subst-def R1-def unrest lens-indep-sym*)

lemma *rea-substs* [*usubst*]:

*true*_{*r*}⌊*v*⌋_{*r*} = *true*_{*r*} *true*⌊*v*⌋_{*r*} = *true*_{*r*} *false*⌊*v*⌋_{*r*} = *false*
(¬_{*r*} *P*)⌊*v*⌋_{*r*} = (¬_{*r*} *P*⌊*v*⌋_{*r*}) (*P* ∧ *Q*)⌊*v*⌋_{*r*} = (*P*⌊*v*⌋_{*r*} ∧ *Q*⌊*v*⌋_{*r*}) (*P* ∨ *Q*)⌊*v*⌋_{*r*} = (*P*⌊*v*⌋_{*r*} ∨ *Q*⌊*v*⌋_{*r*})
(*P* ⇒_{*r*} *Q*)⌊*v*⌋_{*r*} = (*P*⌊*v*⌋_{*r*} ⇒_{*r*} *Q*⌊*v*⌋_{*r*})
by (*rel-auto*+

lemma *rea-substs-lattice* [*usubst*]:

(⋂ *i* · *P*(*i*))⌊*v*⌋_{*r*} = (⋂ *i* · (*P*(*i*))⌊*v*⌋_{*r*})
(⋂ *i* ∈ *A* · *P*(*i*))⌊*v*⌋_{*r*} = (⋂ *i* ∈ *A* · (*P*(*i*))⌊*v*⌋_{*r*})
(⋃ *i* · *P*(*i*))⌊*v*⌋_{*r*} = (⋃ *i* · (*P*(*i*))⌊*v*⌋_{*r*})
by (*rel-auto*+

lemma *rea-subst-USUP-set* [*usubst*]:

A ≠ { } ⇒ (⋃ *i* ∈ *A* · *P*(*i*))⌊*v*⌋_{*r*} = (⋃ *i* ∈ *A* · (*P*(*i*))⌊*v*⌋_{*r*})
by (*rel-auto*+

4.4 Closure laws

lemma *rea-lift-R1* [closure]: $[P]_r$ is *R1*
by (*rel-simp*)

lemma *R1-rea-not*: $R1(\neg_r P) = (\neg_r P)$
by (*rel-auto*)

lemma *R1-rea-not'*: $R1(\neg_r P) = (\neg_r R1(P))$
by (*rel-auto*)

lemma *R2c-rea-not*: $R2c(\neg_r P) = (\neg_r R2c(P))$
by (*rel-auto*)

lemma *RR-rea-not*: $RR(\neg_r RR(P)) = (\neg_r RR(P))$
by (*rel-auto*)

lemma *R1-rea-impl*: $R1(P \Rightarrow_r Q) = (P \Rightarrow_r R1(Q))$
by (*rel-auto*)

lemma *R1-rea-impl'*: $R1(P \Rightarrow_r Q) = (R1(P) \Rightarrow_r R1(Q))$
by (*rel-auto*)

lemma *R2c-rea-impl*: $R2c(P \Rightarrow_r Q) = (R2c(P) \Rightarrow_r R2c(Q))$
by (*rel-auto*)

lemma *RR-rea-impl*: $RR(RR(P) \Rightarrow_r RR(Q)) = (RR(P) \Rightarrow_r RR(Q))$
by (*rel-auto*)

lemma *rea-true-R1* [closure]: $true_r$ is *R1*
by (*rel-auto*)

lemma *rea-true-R2c* [closure]: $true_r$ is *R2c*
by (*rel-auto*)

lemma *rea-true-RR* [closure]: $true_r$ is *RR*
by (*rel-auto*)

lemma *rea-not-R1* [closure]: $\neg_r P$ is *R1*
by (*rel-auto*)

lemma *rea-not-R2c* [closure]: P is *R2c* $\implies \neg_r P$ is *R2c*
by (*simp add: Healthy-def rea-not-def R1-R2c-commute[THEN sym] R2c-not*)

lemma *rea-not-R2-closed* [closure]:
 P is *R2* $\implies (\neg_r P)$ is *R2*
by (*simp add: Healthy-def' R1-rea-not' R2-R2c-def R2c-rea-not*)

lemma *rea-no-RR* [closure]:
 $\llbracket P \text{ is } RR \rrbracket \implies (\neg_r P)$ is *RR*
by (*metis Healthy-def' RR-rea-not*)

lemma *rea-impl-R1* [closure]:
 Q is *R1* $\implies (P \Rightarrow_r Q)$ is *R1*
by (*rel-blast*)

lemma *rea-impl-R2c [closure]*:

$\llbracket P \text{ is } R2c; Q \text{ is } R2c \rrbracket \implies (P \Rightarrow_r Q) \text{ is } R2c$

by (*simp add: rea-impl-def Healthy-def rea-not-def R1-R2c-commute [THEN sym] R2c-not R2c-disj*)

lemma *rea-impl-R2 [closure]*:

$\llbracket P \text{ is } R2; Q \text{ is } R2 \rrbracket \implies (P \Rightarrow_r Q) \text{ is } R2$

by (*rel-blast*)

lemma *rea-impl-RR [closure]*:

$\llbracket P \text{ is } RR; Q \text{ is } RR \rrbracket \implies (P \Rightarrow_r Q) \text{ is } RR$

by (*metis Healthy-def' RR-rea-impl*)

lemma *conj-RR [closure]*:

$\llbracket P \text{ is } RR; Q \text{ is } RR \rrbracket \implies (P \wedge Q) \text{ is } RR$

by (*meson RR-implies-R1 RR-implies-R2c RR-intro RR-unrests(1-4) conj-R1-closed-1 conj-R2c-closed unrest-conj*)

lemma *disj-RR [closure]*:

$\llbracket P \text{ is } RR; Q \text{ is } RR \rrbracket \implies (P \vee Q) \text{ is } RR$

by (*metis Healthy-def' R1-RR R1-idem R1-rea-not' RR-rea-impl RR-rea-not disj-comm double-negation rea-impl-def rea-not-def*)

lemma *USUP-mem-RR-closed [closure]*:

assumes $\bigwedge i. i \in A \implies P \ i \text{ is } RR \ A \neq \{\}$

shows $(\bigsqcup_{i \in A} P(i)) \text{ is } RR$

proof –

have $1: (\bigsqcup_{i \in A} P(i)) \text{ is } R1$

by (*unfold Healthy-def, subst R1-UNF, simp-all add: Healthy-if assms closure cong: USUP-cong*)

have $2: (\bigsqcup_{i \in A} P(i)) \text{ is } R2c$

by (*unfold Healthy-def, subst R2c-UNF, simp-all add: Healthy-if assms RR-implies-R2c closure cong: USUP-cong*)

show *?thesis*

using $1\ 2$ **by** (*rule-tac RR-intro, simp-all add: unrest assms*)

qed

lemma *USUP-ind-RR-closed [closure]*:

assumes $\bigwedge i. P \ i \text{ is } RR$

shows $(\bigsqcup i \cdot P(i)) \text{ is } RR$

using *USUP-mem-RR-closed[of UNIV P]* **by** (*simp add: assms*)

lemma *UNF-mem-RR-closed [closure]*:

assumes $\bigwedge i. P \ i \text{ is } RR$

shows $(\bigsqcap_{i \in A} P(i)) \text{ is } RR$

proof –

have $1: (\bigsqcap_{i \in A} P(i)) \text{ is } R1$

by (*unfold Healthy-def, subst R1-USUP, simp-all add: Healthy-if assms closure*)

have $2: (\bigsqcap_{i \in A} P(i)) \text{ is } R2c$

by (*unfold Healthy-def, subst R2c-USUP, simp-all add: Healthy-if assms RR-implies-R2c closure*)

show *?thesis*

using $1\ 2$ **by** (*rule-tac RR-intro, simp-all add: unrest assms*)

qed

lemma *UNF-ind-RR-closed [closure]*:

assumes $\bigwedge i. P \ i \text{ is } RR$

shows $(\bigsqcap i \cdot P(i)) \text{ is } RR$

```

using UINF-mem-RR-closed[of P UNIV] by (simp add: assms)

lemma USUP-elem-RR [closure]:
  assumes  $\bigwedge i. P\ i\ \text{is}\ RR\ A \neq \{\}$ 
  shows  $(\bigsqcup i \in A \cdot P\ i)\ \text{is}\ RR$ 
proof -
  have 1:  $(\bigsqcup i \in A \cdot P(i))\ \text{is}\ R1$ 
    by (unfold Healthy-def, subst R1-UINF, simp-all add: Healthy-if assms closure)
  have 2:  $(\bigsqcup i \in A \cdot P(i))\ \text{is}\ R2c$ 
    by (unfold Healthy-def, subst R2c-UINF, simp-all add: Healthy-if assms RR-implies-R2c closure)
  show ?thesis
    using 1 2 by (rule-tac RR-intro, simp-all add: unrest assms)
qed

lemma seq-RR-closed [closure]:
  assumes  $P\ \text{is}\ RR\ Q\ \text{is}\ RR$ 
  shows  $P\ ;;\ Q\ \text{is}\ RR$ 
  unfolding Healthy-def
  by (simp add: RR-def Healthy-if assms closure RR-implies-R2 ex-unrest unrest)

lemma power-Suc-RR-closed [closure]:
   $P\ \text{is}\ RR \implies P\ ;;\ P\ ^\wedge\ i\ \text{is}\ RR$ 
  by (induct i, simp-all add: closure upred-semiring.power-Suc)

lemma seqr-iter-RR-closed [closure]:
   $\llbracket I \neq \square; \bigwedge i. i \in \text{set}(I) \implies P(i)\ \text{is}\ RR \rrbracket \implies (;\; i : I \cdot P(i))\ \text{is}\ RR$ 
  apply (induct I, simp-all)
  apply (rename-tac i I)
  apply (case-tac I)
  apply (simp-all add: seq-RR-closed)
done

lemma cond-tt-RR-closed [closure]:
  assumes  $P\ \text{is}\ RR\ Q\ \text{is}\ RR$ 
  shows  $P \triangleleft \$tr' =_u \$tr \triangleright Q\ \text{is}\ RR$ 
  apply (rule RR-intro)
  apply (simp-all add: unrest assms)
  apply (simp-all add: Healthy-def)
  apply (simp-all add: R1-cond R2c-condr Healthy-if assms RR-implies-R2c closure R2c-tr'-minus-tr)
done

lemma rea-skip-RR [closure]:
   $\Pi_r\ \text{is}\ RR$ 
  apply (rel-auto) using minus-zero-eq by blast

lemma tr'-eq-tr-RR-closed [closure]:  $\$tr' =_u \$tr\ \text{is}\ RR$ 
  apply (rel-auto) using minus-zero-eq by auto

lemma inf-RR-closed [closure]:
   $\llbracket P\ \text{is}\ RR; Q\ \text{is}\ RR \rrbracket \implies P \sqcap Q\ \text{is}\ RR$ 
  by (simp add: disj-RR uinf-or)

lemma conj-tr-strict-RR-closed [closure]:
  assumes  $P\ \text{is}\ RR$ 
  shows  $(P \wedge \$tr <_u \$tr')\ \text{is}\ RR$ 

```

```

proof –
  have  $RR(RR(P) \wedge \$tr <_u \$tr') = (RR(P) \wedge \$tr <_u \$tr')$ 
    by (rel-auto)
  thus ?thesis
    by (metis Healthy-def assms)
qed

lemma rea-assert-RR-closed [closure]:
  assumes b is RR
  shows  $\{b\}_r$  is RR
  by (simp add: closure assms rea-assert-def)

lemma upower-RR-closed [closure]:
   $\llbracket i > 0; P \text{ is } RR \rrbracket \implies P \wedge i \text{ is } RR$ 
  apply (induct i, simp-all)
  apply (rename-tac i)
  apply (case-tac i = 0)
  apply (simp-all add: closure upred-semiring.power-Suc)
done

lemma seq-power-RR-closed [closure]:
  assumes P is RR Q is RR
  shows  $(P \wedge i) ;; Q \text{ is } RR$ 
  by (metis assms neq0-conv seq-RR-closed seqr-left-unit upower-RR-closed upred-semiring.power-0)

lemma ustar-right-RR-closed [closure]:
  assumes P is RR Q is RR
  shows  $P ;; Q^* \text{ is } RR$ 
proof –
  have  $P ;; Q^* = P ;; (\bigcap i \in \{0..\} \cdot Q \wedge i)$ 
    by (simp add: ustar-def)
  also have  $\dots = P ;; (II \sqcap (\bigcap i \in \{1..\} \cdot Q \wedge i))$ 
    by (metis One-nat-def UINF-atLeast-first upred-semiring.power-0)
  also have  $\dots = (P \vee P ;; (\bigcap i \in \{1..\} \cdot Q \wedge i))$ 
    by (simp add: disj-upred-def[THEN sym] seqr-or-distr)
  also have  $\dots \text{ is } RR$ 
  proof –
    have  $(\bigcap i \in \{1..\} \cdot Q \wedge i) \text{ is } RR$ 
      by (rule UINF-mem-Continuous-closed, simp-all add: assms closure)
    thus ?thesis
      by (simp add: assms closure)
  qed
  finally show ?thesis .
qed

lemma ustar-left-RR-closed [closure]:
  assumes P is RR Q is RR
  shows  $P^* ;; Q \text{ is } RR$ 
proof –
  have  $P^* ;; Q = (\bigcap i \in \{0..\} \cdot P \wedge i) ;; Q$ 
    by (simp add: ustar-def)
  also have  $\dots = (II \sqcap (\bigcap i \in \{1..\} \cdot P \wedge i)) ;; Q$ 
    by (metis One-nat-def UINF-atLeast-first upred-semiring.power-0)
  also have  $\dots = (Q \vee (\bigcap i \in \{1..\} \cdot P \wedge i) ;; Q)$ 
    by (simp add: disj-upred-def[THEN sym] seqr-or-distl)

```


also have ... is RR
proof –
 have $(\prod i \in \{1..\} \cdot P \hat{=} i)$ is RR
 by (rule UINF-mem-Continuous-closed, simp-all add: assms closure)
 thus ?thesis
 by (simp add: assms closure)
qed
 finally show ?thesis .
qed

lemma *uplus-RR-closed* [closure]: P is RR $\implies P^+$ is RR
 by (simp add: uplus-def ustar-right-RR-closed)

lemma *trace-ext-prefix-RR* [closure]:
 $\llbracket \$tr \# e; \$ok \# e; \$wait \# e; out\alpha \# e \rrbracket \implies \$tr \hat{=}^u e \leq_u \$tr' \text{ is RR}$
apply (rel-auto)
apply (metis (no-types, lifting) Prefix-Order.same-prefix-prefix less-eq-list-def prefix-concat-minus zero-list-def)
apply (metis append-minus list-append-prefixD minus-cancel-le order-refl)
done

lemma *rea-subst-R1-closed* [closure]: $P\llbracket v \rrbracket_r$ is R1
 by (rel-auto)

lemma *R5-comp* [rpred]:
 assumes P is RR Q is RR
 shows $R5(P ;; Q) = R5(P) ;; R5(Q)$
proof –
 have $R5(RR(P) ;; RR(Q)) = R5(RR(P)) ;; R5(RR(Q))$
 by (rel-auto; force)
 thus ?thesis
 by (simp add: Healthy-if assms)
qed

lemma *R4-comp* [rpred]:
 assumes P is R4 Q is RR
 shows $R4(P ;; Q) = P ;; Q$
proof –
 have $R4(R4(P) ;; RR(Q)) = R4(P) ;; RR(Q)$
 by (rel-auto, blast)
 thus ?thesis
 by (simp add: Healthy-if assms)
qed

4.5 Reactive relational calculus

lemma *rea-skip-unit* [rpred]:
 assumes P is RR
 shows $P ;; II_r = P II_r ;; P = P$
proof –
 have 1: $RR(P) ;; II_r = RR(P)$
 by (rel-auto)
 have 2: $II_r ;; RR(P) = RR(P)$
 by (rel-auto)
 from 1 2 show $P ;; II_r = P II_r ;; P = P$
 by (simp-all add: Healthy-if assms)
qed

lemma *rea-true-conj* [*rpred*]:
assumes *P is R1*
shows $(\text{true}_r \wedge P) = P \ (P \wedge \text{true}_r) = P$
using *assms*
by (*simp-all add: Healthy-def R1-def utp-pred-laws.inf-commute*)

lemma *rea-true-disj* [*rpred*]:
assumes *P is R1*
shows $(\text{true}_r \vee P) = \text{true}_r \ (P \vee \text{true}_r) = \text{true}_r$
using *assms* **by** (*metis Healthy-def R1-disj disj-comm true-disj-zero*)

lemma *rea-not-not* [*rpred*]: $P \text{ is } R1 \implies (\neg_r \neg_r P) = P$
by (*simp add: rea-not-def R1-negate-R1 Healthy-if*)

lemma *rea-not-rea-true* [*simp*]: $(\neg_r \text{true}_r) = \text{false}$
by (*simp add: rea-not-def R1-negate-R1 R1-false*)

lemma *rea-not-false* [*simp*]: $(\neg_r \text{false}) = \text{true}_r$
by (*simp add: rea-not-def*)

lemma *rea-true-impl* [*rpred*]:
 $P \text{ is } R1 \implies (\text{true}_r \Rightarrow_r P) = P$
by (*simp add: rea-not-def rea-impl-def R1-negate-R1 R1-false Healthy-if*)

lemma *rea-true-impl'* [*rpred*]:
 $P \text{ is } R1 \implies (\text{true} \Rightarrow_r P) = P$
by (*simp add: rea-not-def rea-impl-def R1-negate-R1 R1-false Healthy-if*)

lemma *rea-false-impl* [*rpred*]:
 $P \text{ is } R1 \implies (\text{false} \Rightarrow_r P) = \text{true}_r$
by (*simp add: rea-impl-def rpred Healthy-if*)

lemma *rea-impl-true* [*simp*]: $(P \Rightarrow_r \text{true}_r) = \text{true}_r$
by (*rel-auto*)

lemma *rea-impl-false* [*simp*]: $(P \Rightarrow_r \text{false}) = (\neg_r P)$
by (*rel-simp*)

lemma *rea-imp-refl* [*rpred*]: $P \text{ is } R1 \implies (P \Rightarrow_r P) = \text{true}_r$
by (*rel-blast*)

lemma *rea-impl-conj* [*rpred*]:
 $(P \Rightarrow_r Q \Rightarrow_r R) = ((P \wedge Q) \Rightarrow_r R)$
by (*rel-auto*)

lemma *rea-impl-mp* [*rpred*]:
 $(P \wedge (P \Rightarrow_r Q)) = (P \wedge Q)$
by (*rel-auto*)

lemma *rea-impl-conj-combine* [*rpred*]:
 $((P \Rightarrow_r Q) \wedge (P \Rightarrow_r R)) = (P \Rightarrow_r Q \wedge R)$
by (*rel-auto*)

lemma *rea-impl-alt-def*:

assumes Q is $R1$
shows $(P \Rightarrow_r Q) = R1(P \Rightarrow Q)$
proof –
have $(P \Rightarrow_r R1(Q)) = R1(P \Rightarrow Q)$
by (*rel-auto*)
thus *?thesis*
by (*simp add: assms Healthy-if*)
qed

lemma *rea-not-true* [*simp*]: $(\neg_r \text{true}) = \text{false}$
by (*rel-auto*)

lemma *rea-not-demorgan1* [*simp*]:
 $(\neg_r (P \wedge Q)) = (\neg_r P \vee \neg_r Q)$
by (*rel-auto*)

lemma *rea-not-demorgan2* [*simp*]:
 $(\neg_r (P \vee Q)) = (\neg_r P \wedge \neg_r Q)$
by (*rel-auto*)

lemma *rea-not-or* [*rpred*]:
 $P \text{ is } R1 \implies (P \vee \neg_r P) = \text{true}_r$
by (*rel-blast*)

lemma *rea-not-and* [*simp*]:
 $(P \wedge \neg_r P) = \text{false}$
by (*rel-auto*)

lemma *rea-not-INFIMUM* [*simp*]:
 $(\neg_r (\bigsqcup_{i \in A} Q(i))) = (\prod_{i \in A} \neg_r Q(i))$
by (*rel-auto*)

lemma *rea-not-USUP* [*simp*]:
 $(\neg_r (\bigsqcup_{i \in A} Q(i))) = (\prod_{i \in A} \neg_r Q(i))$
by (*rel-auto*)

lemma *rea-not-SUPREMUM* [*simp*]:
 $A \neq \{\} \implies (\neg_r (\prod_{i \in A} Q(i))) = (\bigsqcup_{i \in A} \neg_r Q(i))$
by (*rel-auto*)

lemma *rea-not-UNIF* [*simp*]:
 $A \neq \{\} \implies (\neg_r (\prod_{i \in A} Q(i))) = (\bigsqcup_{i \in A} \neg_r Q(i))$
by (*rel-auto*)

lemma *USUP-mem-rea-true* [*simp*]: $A \neq \{\} \implies (\bigsqcup_{i \in A} \text{true}_r) = \text{true}_r$
by (*rel-auto*)

lemma *USUP-ind-rea-true* [*simp*]: $(\bigsqcup_{i \in A} \text{true}_r) = \text{true}_r$
by (*rel-auto*)

lemma *UNIF-ind-rea-true* [*rpred*]: $A \neq \{\} \implies (\prod_{i \in A} \text{true}_r) = \text{true}_r$
by (*rel-auto*)

lemma *UNIF-rea-impl*: $(\prod_{P \in A} P \Rightarrow_r G(P)) = ((\bigsqcup_{P \in A} P \Rightarrow_r G(P)) \Rightarrow_r (\prod_{P \in A} G(P)))$
by (*rel-auto*)

lemma *rea-not-shEx* [*rpred*]: $(\neg_r \text{ shEx } P) = (\text{ shAll } (\lambda x. \neg_r P x))$
by (*rel-auto*)

lemma *rea-assert-true*:
 $\{\text{true}_r\}_r = II_r$
by (*rel-auto*)

lemma *rea-false-true*:
 $\{\text{false}\}_r = \text{true}_r$
by (*rel-auto*)

declare *R4-idem* [*rpred*]
declare *R4-false* [*rpred*]
declare *R4-conj* [*rpred*]
declare *R4-disj* [*rpred*]

declare *R4-R5* [*rpred*]
declare *R5-R4* [*rpred*]

declare *R5-conj* [*rpred*]
declare *R5-disj* [*rpred*]

lemma *R4-USUP* [*rpred*]: $I \neq \{\} \implies R4(\bigsqcup_{i \in I} P(i)) = (\bigsqcup_{i \in I} R4(P(i)))$
by (*rel-auto*)

lemma *R5-USUP* [*rpred*]: $I \neq \{\} \implies R5(\bigsqcup_{i \in I} P(i)) = (\bigsqcup_{i \in I} R5(P(i)))$
by (*rel-auto*)

lemma *R4-UINF* [*rpred*]: $R4(\bigsqcap_{i \in I} P(i)) = (\bigsqcap_{i \in I} R4(P(i)))$
by (*rel-auto*)

lemma *R5-UINF* [*rpred*]: $R5(\bigsqcap_{i \in I} P(i)) = (\bigsqcap_{i \in I} R5(P(i)))$
by (*rel-auto*)

4.6 UTP theory

We create a UTP theory of reactive relations which in particular provides Kleene star theorems
typedecl *RREL*

abbreviation *RREL* \equiv *UTHY*(*RREL*, (*t*::*trace*,*'* α) *rp*)

overloading

rrel-hcond \equiv *utp-hcond* :: (*RREL*, (*t*::*trace*,*'* α) *rp*) *uthy* \Rightarrow ((*t*,*'* α) *rp* \times (*t*,*'* α) *rp*) *health*
rrel-unit \equiv *utp-unit* :: (*RREL*, (*t*::*trace*,*'* α) *rp*) *uthy* \Rightarrow (*t*, *'* α) *hrel-rp*

begin

definition *rrel-hcond* :: (*RREL*, (*t*::*trace*,*'* α) *rp*) *uthy* \Rightarrow ((*t*,*'* α) *rp* \times (*t*,*'* α) *rp*) *health* **where**

[*upred-defs*]: *rrel-hcond* *T* = *RR*

definition *rrel-unit* :: (*RREL*, (*t*::*trace*,*'* α) *rp*) *uthy* \Rightarrow (*t*,*'* α) *hrel-rp* **where**

[*upred-defs*]: *rrel-unit* *T* = *II_r*

end

interpretation *rrel-thy*: *utp-theory-kleene* *UTHY*(*RREL*, (*t*::*trace*,*'* α) *rp*)

rewrites $\bigwedge P. P \in \text{carrier } (\text{uthy-order } RREL) \longleftrightarrow P \text{ is } RR$

and $P \text{ is } \mathcal{H}_{RREL} \longleftrightarrow P \text{ is } RR$

and *carrier* (*uthy-order* *RREL*) \rightarrow *carrier* (*uthy-order* *RREL*) $\equiv \llbracket RR \rrbracket_H \rightarrow \llbracket RR \rrbracket_H$
and $\llbracket \mathcal{H}_{RREL} \rrbracket_H \rightarrow \llbracket \mathcal{H}_{RREL} \rrbracket_H \equiv \llbracket RR \rrbracket_H \rightarrow \llbracket RR \rrbracket_H$
and $\top_{RREL} = \text{false}$
and $\mathcal{II}_{RREL} = II_r$
and *le* (*uthy-order* *RREL*) = *op* \sqsubseteq
proof –
interpret *lat*: *utp-theory-continuous* *UTHY*(*RREL*, ('*t*::*trace*, '*α*) *rp*)
by (*unfold-locals*, *simp-all* *add*: *rrel-hcond-def* *rrel-unit-def* *closure* *Healthy-if* *rpred*)
show 1: $\top_{RREL} = (\text{false} :: ('t, 'α) \text{hrel-rp})$
by (*metis* *Healthy-if* *lat.healthy-top* *rea-no-RR* *rea-not-rea-true* *rea-true-RR* *rrel-hcond-def*)
thus *utp-theory-kleene* *UTHY*(*RREL*, ('*t*, '*α*) *rp*)
by (*unfold-locals*, *simp-all* *add*: *rrel-hcond-def* *rrel-unit-def* *closure* *Healthy-if* *rpred*)
qed (*simp-all* *add*: *rrel-hcond-def* *rrel-unit-def* *closure* *Healthy-if* *rpred*)

declare *rrel-thy.top-healthy* [*simp del*]
declare *rrel-thy.bottom-healthy* [*simp del*]

abbreviation *rea-star* :: $- \Rightarrow -$ (*r [999] 999) **where**
 $P^{*r} \equiv P \star_{RREL}$

4.7 Instantaneous Reactive Relations

Instantaneous Reactive Relations, where the trace stays the same.

abbreviation *Instant* :: ('*t*::*trace*, '*α*) *hrel-rp* \Rightarrow ('*t*, '*α*) *hrel-rp* **where**
Instant(*P*) $\equiv RID(tr)(P)$

lemma *skip-rea-Instant* [*closure*]: *II_r* is *Instant*
by (*rel-auto*)

end

5 Reactive Conditions

theory *utp-rea-cond*
imports *utp-rea-rel*
begin

5.1 Healthiness Conditions

definition *RC1* :: ('*t*::*trace*, '*α*, '*β*) *rel-rp* \Rightarrow ('*t*, '*α*, '*β*) *rel-rp* **where**
[*upred-defs*]: $RC1(P) = (\neg_r (\neg_r P) ;; \text{true}_r)$

definition *RC* :: ('*t*::*trace*, '*α*, '*β*) *rel-rp* \Rightarrow ('*t*, '*α*, '*β*) *rel-rp* **where**
[*upred-defs*]: $RC = RC1 \circ RR$

lemma *RC-intro*: $\llbracket P \text{ is } RR; ((\neg_r (\neg_r P) ;; \text{true}_r) = P) \rrbracket \Longrightarrow P \text{ is } RC$
by (*simp* *add*: *Healthy-def* *RC1-def* *RC-def*)

lemma *RC-intro'*: $\llbracket P \text{ is } RR; P \text{ is } RC1 \rrbracket \Longrightarrow P \text{ is } RC$
by (*simp* *add*: *Healthy-def* *RC1-def* *RC-def*)

lemma *RC1-idem*: $RC1(RC1(P)) = RC1(P)$
by (*rel-auto*, (*blast intro*: *dual-order.trans*)+)

lemma *RC1-mono*: $P \sqsubseteq Q \Longrightarrow RC1(P) \sqsubseteq RC1(Q)$

```

by (rel-blast)

lemma RC1-prop:
  assumes  $P$  is RC1
  shows  $(\neg_r P) ;; R1 \text{ true} = (\neg_r P)$ 
proof -
  have  $(\neg_r P) = (\neg_r (RC1 P))$ 
    by (simp add: Healthy-if assms)
  also have  $\dots = (\neg_r P) ;; R1 \text{ true}$ 
    by (simp add: RC1-def rpred closure)
  finally show ?thesis ..
qed

lemma R2-RC:  $R2 (RC P) = RC P$ 
proof -
  have  $\neg_r RR P$  is RR
    by (metis (no-types) Healthy-Idempotent RR-Idempotent RR-rea-not)
  then show ?thesis
    by (metis (no-types) Healthy-def' R1-R2c-segr-distribute R2-R2c-def RC1-def RC-def RR-implies-R1
        RR-implies-R2c comp-apply rea-not-R2-closed rea-true-R1 rea-true-R2c)
qed

lemma RC-R2-def:  $RC = RC1 \circ RR$ 
by (auto simp add: RC-def fun-eq-iff R1-R2c-commute[THEN sym] R1-R2c-is-R2)

lemma RC-implies-R2:  $P$  is RC  $\implies P$  is R2
by (metis Healthy-def' R2-RC)

lemma RC-ex-ok-wait:  $(\exists \{ \$ok, \$ok', \$wait, \$wait' \} \cdot RC P) = RC P$ 
by (rel-auto)

```

An important property of reactive conditions is they are monotonic with respect to the trace. That is, P with a shorter trace is refined by P with a longer trace.

```

lemma RC-prefix-refine:
  assumes  $P$  is RC  $s \leq t$ 
  shows  $P \llbracket 0, \langle s \rangle / \$tr, \$tr' \rrbracket \sqsubseteq P \llbracket 0, \langle t \rangle / \$tr, \$tr' \rrbracket$ 
proof -
  from assms(2) have  $(RC P) \llbracket 0, \langle s \rangle / \$tr, \$tr' \rrbracket \sqsubseteq (RC P) \llbracket 0, \langle t \rangle / \$tr, \$tr' \rrbracket$ 
    apply (rel-auto)
    using dual-order.trans apply blast
  done
  thus ?thesis
    by (simp only: assms(1) Healthy-if)
qed

```

5.2 Closure laws

```

lemma RC-implies-RR [closure]:
  assumes  $P$  is RC
  shows  $P$  is RR
  by (metis Healthy-def RC-ex-ok-wait RC-implies-R2 RR-def assms)

lemma RC-implies-RC1:  $P$  is RC  $\implies P$  is RC1
  by (metis Healthy-def RC-R2-def RC-implies-RR comp-eq-dest-lhs)

```

lemma *RC1-trace-ext-prefix*:

$\text{out}\alpha \# e \implies \text{RC1}(\neg_r \$tr \hat{_u} e \leq_u \$tr') = (\neg_r \$tr \hat{_u} e \leq_u \$tr')$
 by (rel-auto, blast, metis (no-types, lifting) dual-order.trans)

lemma *RC1-conj*: $\text{RC1}(P \wedge Q) = (\text{RC1}(P) \wedge \text{RC1}(Q))$

by (rel-blast)

lemma *conj-RC1-closed* [closure]:

$\llbracket P \text{ is RC1}; Q \text{ is RC1} \rrbracket \implies P \wedge Q \text{ is RC1}$
 by (simp add: Healthy-def RC1-conj)

lemma *disj-RC1-closed* [closure]:

assumes $P \text{ is RC1 } Q \text{ is RC1}$

shows $(P \vee Q) \text{ is RC1}$

proof –

have $1:\text{RC1}(\text{RC1}(P) \vee \text{RC1}(Q)) = (\text{RC1}(P) \vee \text{RC1}(Q))$

apply (rel-auto) using dual-order.trans by blast+

show ?thesis

by (metis (no-types) Healthy-def 1 assms)

qed

lemma *conj-RC-closed* [closure]:

assumes $P \text{ is RC } Q \text{ is RC}$

shows $(P \wedge Q) \text{ is RC}$

by (metis Healthy-def RC-R2-def RC-implies-RR assms comp-apply conj-RC1-closed conj-RR)

lemma *rea-true-RC* [closure]: $\text{true}_r \text{ is RC}$

by (rel-auto)

lemma *false-RC* [closure]: $\text{false} \text{ is RC}$

by (rel-auto)

lemma *disj-RC-closed* [closure]: $\llbracket P \text{ is RC}; Q \text{ is RC} \rrbracket \implies (P \vee Q) \text{ is RC}$

by (metis Healthy-def RC-R2-def RC-implies-RR comp-apply disj-RC1-closed disj-RR)

lemma *UINF-mem-RC1-closed* [closure]:

assumes $\bigwedge i. P \ i \text{ is RC1}$

shows $(\bigcap i \in A \cdot P \ i) \text{ is RC1}$

proof –

have $1:\text{RC1}(\bigcap i \in A \cdot \text{RC1}(P \ i)) = (\bigcap i \in A \cdot \text{RC1}(P \ i))$

by (rel-auto, meson order.trans)

show ?thesis

by (metis (mono-tags, lifting) 1 Healthy-def' UINF-all-cong UINF-alt-def assms)

qed

lemma *UINF-mem-RC-closed* [closure]:

assumes $\bigwedge i. P \ i \text{ is RC}$

shows $(\bigcap i \in A \cdot P \ i) \text{ is RC}$

proof –

have $\text{RC}(\bigcap i \in A \cdot P \ i) = (\text{RC1} \circ \text{RR})(\bigcap i \in A \cdot P \ i)$

by (simp add: RC-def)

also have $\dots = \text{RC1}(\bigcap i \in A \cdot \text{RR}(P \ i))$

by (rel-blast)

also have $\dots = \text{RC1}(\bigcap i \in A \cdot \text{RC1}(P \ i))$

by (simp add: Healthy-if RC-implies-RR RC-implies-RC1 assms)

```

also have ... = ( $\bigcap_{i \in A} \cdot RC1(P\ i)$ )
  by (rel-auto, meson order.trans)
also have ... = ( $\bigcap_{i \in A} \cdot P\ i$ )
  by (simp add: Healthy-if RC-implies-RC1 assms)
finally show ?thesis
  by (simp add: Healthy-def)
qed

lemma UINF-ind-RC-closed [closure]:
  assumes  $\bigwedge i. P\ i$  is RC
  shows ( $\bigcap_{i \in A} \cdot P\ i$ ) is RC
  by (metis (no-types) UINF-as-Sup-collect' UINF-as-Sup-image UINF-mem-RC-closed assms)

lemma USUP-mem-RC1-closed [closure]:
  assumes  $\bigwedge i. i \in A \implies P\ i$  is RC1  $A \neq \{\}$ 
  shows ( $\bigcap_{i \in A} \cdot P\ i$ ) is RC1
proof -
  have  $RC1(\bigcap_{i \in A} \cdot P\ i) = RC1(\bigcap_{i \in A} \cdot RC1(P\ i))$ 
    by (simp add: Healthy-if assms(1) cong: USUP-cong)
  also from assms(2) have ... = ( $\bigcap_{i \in A} \cdot RC1(P\ i)$ )
    using dual-order.trans by (rel-blast)
  also have ... = ( $\bigcap_{i \in A} \cdot P\ i$ )
    by (simp add: Healthy-if assms(1) cong: USUP-cong)
  finally show ?thesis
    using Healthy-def by blast
qed

lemma USUP-mem-RC-closed [closure]:
  assumes  $\bigwedge i. i \in A \implies P\ i$  is RC  $A \neq \{\}$ 
  shows ( $\bigcap_{i \in A} \cdot P\ i$ ) is RC
  by (rule RC-intro', simp-all add: closure assms RC-implies-RC1)

lemma neg-trace-ext-prefix-RC [closure]:
   $\llbracket \$tr \# e; \$ok \# e; \$wait \# e; out\alpha \# e \rrbracket \implies \neg_r \$tr \hat{^}_u e \leq_u \$tr' \text{ is RC}$ 
  by (rule RC-intro, simp add: closure, metis RC1-def RC1-trace-ext-prefix)

lemma RC1-unrest:
   $\llbracket mwb\text{-}lens\ x; x \bowtie tr \rrbracket \implies \$x' \# RC1(P)$ 
  by (simp add: RC1-def unrest)

lemma RC-unrest-dashed [unrest]:
   $\llbracket P \text{ is RC}; mwb\text{-}lens\ x; x \bowtie tr \rrbracket \implies \$x' \# P$ 
  by (metis Healthy-if RC1-unrest RC-implies-RC1)

lemma RC1-RR-closed:  $P \text{ is RR} \implies RC1(P) \text{ is RR}$ 
  by (simp add: RC1-def closure)

end

```

6 Reactive Programs

```

theory utp-rea-prog
  imports utp-rea-cond
begin

```


6.1 Stateful reactive alphabet

$R3$ as presented in the UTP book and related publications is not sensitive to state, although reactive programs often need this property. Thus it is necessary to use a modification of $R3$ from Butterfield et al. [1] that explicitly states that intermediate waiting states do not propagate final state variables. In order to do this we need an additional observational variable that captures the program state that we call st . Upon this foundation, we can define operators for reactive programs.

alphabet $'s \text{ } rsp\text{-}vars = 't \text{ } rp\text{-}vars +$
 $st :: 's$

declare $rsp\text{-}vars.\text{defs} \text{ } [lens\text{-}defs]$

type-synonym $('s, 't, 'α) \text{ } rsp = ('t, ('s, 'α) \text{ } rsp\text{-}vars\text{-}scheme) \text{ } rp$
type-synonym $('s, 't, 'α, 'β) \text{ } rel\text{-}rsp = (('s, 't, 'α) \text{ } rsp, ('s, 't, 'β) \text{ } rsp) \text{ } urel$
type-synonym $('s, 't, 'α) \text{ } hrel\text{-}rsp = ('s, 't, 'α) \text{ } rsp \text{ } hrel$
type-synonym $('s, 't) \text{ } rdes = ('s, 't, unit) \text{ } hrel\text{-}rsp$

translations

$(type) ('s, 't, 'α) \text{ } rsp \leq (type) ('t, ('s, 'α) \text{ } rsp\text{-}vars\text{-}ext) \text{ } rp$
 $(type) ('s, 't, 'α) \text{ } rsp \leq (type) ('t, ('s, 'α) \text{ } rsp\text{-}vars\text{-}scheme) \text{ } rp$
 $(type) ('s, 't, unit) \text{ } rsp \leq (type) ('t, 's \text{ } rsp\text{-}vars) \text{ } rp$
 $(type) ('s, 't, 'α, 'β) \text{ } rel\text{-}rsp \leq (type) (('s, 't, 'α) \text{ } rsp, ('s1, 't1, 'β) \text{ } rsp) \text{ } urel$
 $(type) ('s, 't, 'α) \text{ } hrel\text{-}rsp \leq (type) ('s, 't, 'α) \text{ } rsp \text{ } hrel$
 $(type) ('s, 't) \text{ } rdes \leq (type) ('s, 't, unit) \text{ } hrel\text{-}rsp$

notation $rsp\text{-}vars\text{-}child\text{-}lens_a (\Sigma_s)$

notation $rsp\text{-}vars\text{-}child\text{-}lens (\Sigma_S)$

syntax

$-svid\text{-}st\text{-}alpha :: svid (\Sigma_S)$

translations

$-svid\text{-}st\text{-}alpha \Rightarrow CONST \text{ } rsp\text{-}vars\text{-}child\text{-}lens$

lemma $srea\text{-}var\text{-}ords \text{ } [usubst]:$

$\$st \prec_v \st'
 $\$ok \prec_v \$st \$ok' \prec_v \$st' \$ok \prec_v \$st' \$ok' \prec_v \st
 $\$st \prec_v \$wait \$st' \prec_v \$wait' \$st \prec_v \$wait' \$st' \prec_v \$wait$
 $\$st \prec_v \$tr \$st' \prec_v \$tr' \$st \prec_v \$tr' \$st' \prec_v \tr
by $(simp\text{-}all \text{ } add: \text{ } var\text{-}name\text{-}ord\text{-}def)$

lemma $st\text{-}bij\text{-}lemma: \text{ } bij\text{-}lens (st_a +_L \Sigma_s)$

by $(unfold\text{-}locales, \text{ } auto \text{ } simp \text{ } add: \text{ } lens\text{-}defs)$

lemma $rea\text{-}lens\text{-}equiv\text{-}st\text{-}rest: \Sigma_R \approx_L st +_L \Sigma_S$

proof $-$

have $st +_L \Sigma_S = (st_a +_L \Sigma_s) ;_L \Sigma_R$
by $(simp \text{ } add: \text{ } plus\text{-}lens\text{-}distr \text{ } st\text{-}def \text{ } rsp\text{-}vars\text{-}child\text{-}lens\text{-}def)$
also have $\dots \approx_L 1_L ;_L \Sigma_R$
using $lens\text{-}equiv\text{-}via\text{-}bij \text{ } st\text{-}bij\text{-}lemma$ **by** $auto$
also have $\dots = \Sigma_R$
by $(simp)$
finally show $?thesis$
using $lens\text{-}equiv\text{-}sym$ **by** $blast$

qed

lemma *srea-lens-bij*: *bij-lens* (*ok* +_L *wait* +_L *tr* +_L *st* +_L Σ_S)

proof –

have *ok* +_L *wait* +_L *tr* +_L *st* +_L $\Sigma_S \approx_L ok +_L wait +_L tr +_L \Sigma_R$
 by (*auto intro!lens-plus-cong*, *rule lens-equiv-sym*, *simp add: rea-lens-equiv-st-rest*)
 also have ... $\approx_L 1_L$
 using *bij-lens-equiv-id*[*of ok +_L wait +_L tr +_L Σ_R*] **by** (*simp add: rea-lens-bij*)
 finally show *?thesis*
 by (*simp add: bij-lens-equiv-id*)

qed

lemma *st-qual-alpha* [*alpha*]: *x* ;_L *fst*_L ;_L *st* ×_L *st* = ($\$st:x$)_v

by (*metis (no-types, hide-lams) in-var-def in-var-prod-lens lens-comp-assoc st-vwb-lens vwb-lens-wb*)

interpretation *alphabet-state*:

lens-interp $\lambda(ok, wait, tr, r). (ok, wait, tr, st_v\ r, more\ r)$
 apply (*unfold-locales*)
 apply (*rule injI*)
 apply (*clarsimp*)
 done

interpretation *alphabet-state-rel*: *lens-interp* $\lambda(ok, ok', wait, wait', tr, tr', r, r').$

 (*ok, ok', wait, wait', tr, tr', st_v\ r, st_v\ r', more\ r, more\ r'*)
 apply (*unfold-locales*)
 apply (*rule injI*)
 apply (*clarsimp*)
 done

lemma *unrest-st'-neg-RC* [*unrest*]:

assumes *P is RR P is RC*
 shows $\$st' \# P$

proof –

have *P* = ($\neg_r \neg_r P$)
 by (*simp add: closure rpred assms*)
 also have ... = ($\neg_r (\neg_r P) ;; true_r$)
 by (*metis Healthy-if RC1-def RC-implies-RC1 assms(2) calculation*)
 also have $\$st' \# \dots$
 by (*rel-auto*)
 finally show *?thesis* .

qed

lemma *ex-st'-RR-closed* [*closure*]:

assumes *P is RR*
 shows $(\exists \$st' \cdot P) \text{ is } RR$

proof –

have $RR (\exists \$st' \cdot RR(P)) = (\exists \$st' \cdot RR(P))$
 by (*rel-auto*)
 thus *?thesis*
 by (*metis Healthy-def assms*)

qed

lemma *unrest-st'-R4* [*unrest*]:

$\$st' \# P \implies \$st' \# R4(P)$
 by (*rel-auto*)

lemma *unrest-st'-R5* [*unrest*]:
 $\$st' \# P \implies \$st' \# R5(P)$
by (*rel-auto*)

6.2 State Lifting

abbreviation *lift-state-rel* ($\lceil \cdot \rceil_S$)
where $\lceil P \rceil_S \equiv P \oplus_p (st \times_L st)$

abbreviation *drop-state-rel* ($\lfloor \cdot \rfloor_S$)
where $\lfloor P \rfloor_S \equiv P \upharpoonright_e (st \times_L st)$

abbreviation *lift-state-pre* ($\lceil \cdot \rceil_{S<}$)
where $\lceil p \rceil_{S<} \equiv \lceil \lceil p \rceil \rceil_S$

abbreviation *drop-state-pre* ($\lfloor \cdot \rfloor_{S<}$)
where $\lfloor p \rfloor_{S<} \equiv \lfloor \lfloor p \rfloor \rfloor_S$

abbreviation *lift-state-post* ($\lceil \cdot \rceil_{S>}$)
where $\lceil p \rceil_{S>} \equiv \lceil \lceil p \rceil \rceil_S$

abbreviation *drop-state-post* ($\lfloor \cdot \rfloor_{S>}$)
where $\lfloor p \rfloor_{S>} \equiv \lfloor \lfloor p \rfloor \rfloor_S$

lemma *st'-unrest-st-lift-pred* [*unrest*]:
 $\$st' \# \lceil a \rceil_{S<}$
by (*pred-auto*)

lemma *out-alpha-unrest-st-lift-pre* [*unrest*]:
 $out\alpha \# \lceil a \rceil_{S<}$
by (*rel-auto*)

lemma *R1-st'-unrest* [*unrest*]: $\$st' \# P \implies \$st' \# R1(P)$
by (*simp add: R1-def unrest*)

lemma *R2c-st'-unrest* [*unrest*]: $\$st' \# P \implies \$st' \# R2c(P)$
by (*simp add: R2c-def unrest*)

lemma *st-lift-R1-true-right*: $\lceil b \rceil_{S<} ;; R1(true) = \lceil b \rceil_{S<}$
by (*rel-auto*)

lemma *R2c-lift-state-pre*: $R2c(\lceil b \rceil_{S<}) = \lceil b \rceil_{S<}$
by (*rel-auto*)

6.3 Reactive Program Operators

6.3.1 State Substitution

Lifting substitutions on the reactive state

definition *usubst-st-lift* ::
 $'s \text{ usubst} \Rightarrow ((s', t :: \text{trace}, \alpha) \text{ rsp} \times (s', t, \beta) \text{ rsp}) \text{ usubst} \ (\lceil \cdot \rceil_{S\sigma})$ **where**
 $[upred-defs]: \lceil \sigma \rceil_{S\sigma} = \lceil \sigma \oplus_s st \rceil_s$

abbreviation *st-subst* :: $'s \text{ usubst} \Rightarrow (s', t :: \text{trace}, \alpha, \beta) \text{ rel-rsp} \Rightarrow (s', t, \alpha, \beta) \text{ rel-rsp}$ (**infixr** \dagger_S 80)
where

$$\sigma \dagger_S P \equiv [\sigma]_{S\sigma} \dagger P$$

translations

$$\begin{aligned} \sigma \dagger_S P &\leq [\sigma \oplus_s st]_s \dagger P \\ \sigma \dagger_S P &\leq [\sigma]_{S\sigma} \dagger P \end{aligned}$$

lemma *st-lift-lemma*:

$$\begin{aligned} &[\sigma]_{S\sigma} = \sigma \oplus_s (fst_L ;_L (st \times_L st)) \\ &\text{by } (auto \text{ simp add: upred-defs lens-defs prod.case-eq-if}) \end{aligned}$$

lemma *unrest-st-lift* [*unrest*]:

$$\begin{aligned} &\text{fixes } x :: 'a \implies ('s, 't::trace, 'a) \text{ rsp} \times ('s, 't, 'a) \text{ rsp} \\ &\text{assumes } x \bowtie (\$st)_v \\ &\text{shows } x \# [\sigma]_{S\sigma} \text{ (is ?P)} \\ &\text{by (simp add: st-lift-lemma)} \\ &\quad (metis assms in-var-def in-var-prod-lens lens-comp-left-id st-vwb-lens unrest-subst-alpha-ext vwb-lens-wb) \end{aligned}$$

lemma *id-st-subst* [*usubst*]:

$$\begin{aligned} &[id]_{S\sigma} = id \\ &\text{by (pred-auto)} \end{aligned}$$

lemma *st-subst-comp* [*usubst*]:

$$\begin{aligned} &[\sigma]_{S\sigma} \circ [\varrho]_{S\sigma} = [\sigma \circ \varrho]_{S\sigma} \\ &\text{by (rel-auto)} \end{aligned}$$

definition *lift-cond-srea* ($[\cdot]_{S\leftarrow}$) **where**

$$[upred-defs]: [b]_{S\leftarrow} = [b]_{S<}$$

lemma *unrest-lift-cond-srea* [*unrest*]:

$$\begin{aligned} &x \# [b]_{S<} \implies x \# [b]_{S\leftarrow} \\ &\text{by (simp add: lift-cond-srea-def)} \end{aligned}$$

lemma *st-subst-RR-closed* [*closure*]:

$$\begin{aligned} &\text{assumes } P \text{ is } RR \\ &\text{shows } [\sigma]_{S\sigma} \dagger P \text{ is } RR \end{aligned}$$

proof –

$$\begin{aligned} &\text{have } RR([\sigma]_{S\sigma} \dagger RR(P)) = [\sigma]_{S\sigma} \dagger RR(P) \\ &\quad \text{by (rel-auto)} \\ &\text{thus ?thesis} \\ &\quad \text{by (metis Healthy-def assms)} \end{aligned}$$

qed

lemma *subst-lift-cond-srea* [*usubst*]: $\sigma \dagger_S [P]_{S\leftarrow} = [\sigma \dagger P]_{S\leftarrow}$

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

lemma *st-subst-rea-not* [*usubst*]: $\sigma \dagger_S (\neg_r P) = (\neg_r \sigma \dagger_S P)$

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

lemma *st-subst-seq* [*usubst*]: $\sigma \dagger_S (P ;; Q) = \sigma \dagger_S P ;; Q$

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

lemma *st-subst-RC-closed* [*closure*]:

$$\begin{aligned} &\text{assumes } P \text{ is } RC \\ &\text{shows } \sigma \dagger_S P \text{ is } RC \\ &\text{apply (rule RC-intro, simp add: closure assms)} \end{aligned}$$

apply (*simp add: st-subst-rea-not*[*THEN sym*] *st-subst-seq*[*THEN sym*])
apply (*metis Healthy-if RC1-def RC-implies-RC1 assms*)
done

6.3.2 Assignment

definition *rea-assigns* :: ($'s \Rightarrow 's$) \Rightarrow ($'s, 't::\text{trace}, 'a$) *hrel-rsp* ($\langle \cdot \rangle_r$) **where**
[upred-defs]: $\langle \sigma \rangle_r = (\$tr' =_u \$tr \wedge \lceil \langle \sigma \rangle_a \rceil_S \wedge \$\Sigma_S' =_u \$\Sigma_S)$

syntax

-assign-rea :: *svids* \Rightarrow *uexprs* \Rightarrow *logic* ($'(-)' :=_r '(-)'$)
-assign-rea :: *svids* \Rightarrow *uexprs* \Rightarrow *logic* (**infixr** $:=_r$ 90)

translations

-assign-rea *xs vs* \Rightarrow *CONST* *rea-assigns* (*-mk-usubst* (*CONST id*) *xs vs*)
-assign-rea *x v* \Leftarrow *CONST* *rea-assigns* (*CONST subst-upd* (*CONST id*) *x v*)
-assign-rea *x v* \Leftarrow *-assign-rea* (*-spvar x*) *v*
 $x, y :=_r u, v \Leftarrow$ *CONST* *rea-assigns* (*CONST subst-upd* (*CONST subst-upd* (*CONST id*) (*CONST svar x*) *u*) (*CONST svar y*) *v*)

lemma *rea-assigns-RR-closed* [*closure*]:

$\langle \sigma \rangle_r$ *is RR*
apply (*rel-auto*) **using** *minus-zero-eq* **by** *auto*

lemma *st-subst-assigns-rea* [*usubst*]:

$\sigma \dagger_S \langle \varrho \rangle_r = \langle \varrho \circ \sigma \rangle_r$
by (*rel-auto*)

lemma *st-subst-rea-skip* [*usubst*]:

$\sigma \dagger_S \Pi_r = \langle \sigma \rangle_r$
by (*rel-auto*)

lemma *rea-assigns-comp* [*rpred*]:

assumes *P is RR*
shows $\langle \sigma \rangle_r ;; P = \sigma \dagger_S P$

proof –

have $\langle \sigma \rangle_r ;; (RR P) = \sigma \dagger_S (RR P)$
by (*rel-auto*)
thus *?thesis*
by (*metis Healthy-def assms*)

qed

lemma *st-subst-RR* [*closure*]:

assumes *P is RR*
shows $(\sigma \dagger_S P)$ *is RR*

proof –

have $(\sigma \dagger_S RR(P))$ *is RR*
by (*rel-auto*)
thus *?thesis*
by (*simp add: Healthy-if assms*)

qed

lemma *rea-assigns-st-subst* [*usubst*]:

$\lceil \sigma \oplus_s st \rceil_s \dagger \langle \varrho \rangle_r = \langle \varrho \circ \sigma \rangle_r$
by (*rel-auto*)

6.3.3 Conditional

We guard the reactive conditional condition so that it can't be simplified by alphabet laws unless explicitly simplified.

abbreviation *cond-srea* ::
 $(\text{'s}, \text{'t}::\text{trace}, \text{'}\alpha, \text{'}\beta) \text{ rel-rsp} \Rightarrow$
 $\text{'s upred} \Rightarrow$
 $(\text{'s}, \text{'t}, \text{'}\alpha, \text{'}\beta) \text{ rel-rsp} \Rightarrow$
 $(\text{'s}, \text{'t}, \text{'}\alpha, \text{'}\beta) \text{ rel-rsp } ((\beta \triangleleft - \triangleright_R / -) [52, 0, 53] 52) \text{ where}$
 $\text{cond-srea } P \triangleleft b \triangleright_R Q \equiv P \triangleleft [b]_{S \leftarrow} \triangleright Q$

lemma *st-cond-assigns* [*rpred*]:
 $\langle \sigma \rangle_r \triangleleft b \triangleright_R \langle \varrho \rangle_r = \langle \sigma \triangleleft b \triangleright_s \varrho \rangle_r$
by (*rel-auto*)

lemma *cond-srea-RR-closed* [*closure*]:
assumes *P is RR Q is RR*
shows $P \triangleleft b \triangleright_R Q \text{ is } RR$
proof –
have $RR(RR(P) \triangleleft b \triangleright_R RR(Q)) = RR(P) \triangleleft b \triangleright_R RR(Q)$
by (*rel-auto*)
thus ?thesis
by (*metis Healthy-def' assms(1) assms(2)*)
qed

lemma *cond-srea-RC1-closed*:
assumes *P is RC1 Q is RC1*
shows $P \triangleleft b \triangleright_R Q \text{ is } RC1$
proof –
have $RC1(RC1(P) \triangleleft b \triangleright_R RC1(Q)) = RC1(P) \triangleleft b \triangleright_R RC1(Q)$
using *dual-order.trans* **by** (*rel-blast*)
thus ?thesis
by (*metis Healthy-def' assms*)
qed

lemma *cond-srea-RC-closed* [*closure*]:
assumes *P is RC Q is RC*
shows $P \triangleleft b \triangleright_R Q \text{ is } RC$
by (*rule RC-intro', simp-all add: closure cond-srea-RC1-closed RC-implies-RC1 assms*)

lemma *R4-cond* [*rpred*]: $R4(P \triangleleft b \triangleright_R Q) = (R4(P) \triangleleft b \triangleright_R R4(Q))$
by (*rel-auto*)

lemma *R5-cond* [*rpred*]: $R5(P \triangleleft b \triangleright_R Q) = (R5(P) \triangleleft b \triangleright_R R5(Q))$
by (*rel-auto*)

6.3.4 Assumptions

definition *rea-assume* :: $\text{'s upred} \Rightarrow (\text{'s}, \text{'t}::\text{trace}, \text{'}\alpha) \text{ hrel-rsp } ([\text{'}]^\top_r) \text{ where}$
 $[\text{'}]^\top_r = (H_r \triangleleft b \triangleright_R \text{false})$

lemma *rea-assume-RR* [*closure*]: $[\text{'}]^\top_r \text{ is } RR$
by (*simp add: rea-assume-def closure*)

lemma *rea-assume-false* [*rpred*]: $[\text{false}]^\top_r = \text{false}$

by (rel-auto)

lemma *rea-assume-true* [rpred]: [true][⊤]_r = *II*_r
by (rel-auto)

lemma *rea-assume-comp* [rpred]: [b][⊤]_r ;; [c][⊤]_r = [b ∧ c][⊤]_r
by (rel-auto)

6.3.5 State Abstraction

We introduce state abstraction by creating some lens functors that allow us to lift a lens on the state-space to one on the whole stateful reactive alphabet.

definition *lmap_R* :: ('a ⇒ 'b) ⇒ ('t::trace, 'a) rp ⇒ ('t, 'b) rp **where**
[lens-defs]: *lmap_R* = *lmap_D* ∘ *lmap*[rp-vars]

definition *map-rsp-st* ::
(σ ⇒ τ) ⇒
(σ, 'α) rsp-vars-scheme ⇒ (τ, 'α) rsp-vars-scheme **where**
[lens-defs]: *map-rsp-st* f = (λr. (|st_v = f (st_v r), ... = *rsp-vars.more* r))

definition *map-st-lens* ::
(σ ⇒ ψ) ⇒
((σ, τ::trace, 'α) rsp ⇒ (ψ, τ::trace, 'α) rsp) (map'-st_L) **where**
[lens-defs]:
map-st-lens l = *lmap_R* (|
 lens-get = *map-rsp-st* (*get_l*),
 lens-put = *map-rsp-st* o (*put_l*) o *rsp-vars.st_v*)

lemma *map-set-vwb* [simp]: *vwb-lens* X ⇒ *vwb-lens* (map-st_L X)
apply (unfold-locales, simp-all add: lens-defs)
apply (metis des-vars.surjective rp-vars.surjective rsp-vars.surjective)+
done

abbreviation *abs-st_L* ≡ (map-st_L 0_L) ×_L (map-st_L 0_L)

abbreviation *abs-st* (⟨-⟩_S) **where**
abs-st P ≡ P |_e *abs-st_L*

6.3.6 Reactive Frames and Extensions

definition *rea-frame* :: ('a ⇒ 'α) ⇒ ('α, 't::trace) rdes ⇒ ('α, 't) rdes **where**
[upred-defs]: *rea-frame* x P = *frame* (ok +_L wait +_L tr +_L (x ;_L st)) P

definition *rea-frame-ext* :: ('α ⇒ 'β) ⇒ ('α, 't::trace) rdes ⇒ ('β, 't) rdes **where**
[upred-defs]: *rea-frame-ext* a P = *rea-frame* a (rel-aext P (map-st_L a))

syntax

-*rea-frame* :: *salpha* ⇒ *logic* ⇒ *logic* (·:[·]_r [99,0] 100)
-*rea-frame-ext* :: *salpha* ⇒ *logic* ⇒ *logic* (·:[·]_r⁺ [99,0] 100)

translations

-*rea-frame* x P => *CONST* *rea-frame* x P
-*rea-frame* (-*salphaset* (-*salphamk* x)) P <= *CONST* *rea-frame* x P
-*rea-frame-ext* x P => *CONST* *rea-frame-ext* x P
-*rea-frame-ext* (-*salphaset* (-*salphamk* x)) P <= *CONST* *rea-frame-ext* x P

lemma *rea-frame-RR-closed* [closure]:
 assumes P is RR
 shows $x:[P]_r$ is RR
proof –
 have $RR(x:[RR\ P]_r) = x:[RR\ P]_r$
 by (*rel-auto*)
 thus ?thesis
 by (*metis Healthy-if Healthy-intro assms*)
qed

lemma *rea-aext-RR* [closure]:
 assumes P is RR
 shows *rel-aext* P (*map-st_L* x) is RR
proof –
 have *rel-aext* ($RR\ P$) (*map-st_L* x) is RR
 by (*rel-auto*)
 thus ?thesis
 by (*simp add: Healthy-if assms*)
qed

lemma *rea-frame-ext-RR-closed* [closure]:
 P is $RR \implies x:[P]_r^+$ is RR
 by (*simp add: rea-frame-ext-def closure*)

lemma *rel-aext-st-Instant-closed* [closure]:
 P is *Instant* \implies *rel-aext* P (*map-st_L* x) is *Instant*
 by (*rel-auto*)

lemma *rea-frame-ext-false* [frame]:
 $x:[false]_r^+ = false$
 by (*rel-auto*)

lemma *rea-frame-ext-skip* [frame]:
vwb-lens $x \implies x:[II_r]_r^+ = II_r$
 by (*rel-auto*)

lemma *rea-frame-ext-assigns* [frame]:
vwb-lens $x \implies x:[\langle \sigma \rangle_r]_r^+ = \langle \sigma \oplus_s x \rangle_r$
 by (*rel-auto*)

lemma *rea-frame-ext-cond* [frame]:
 $x:[P \triangleleft b \triangleright_R Q]_r^+ = x:[P]_r^+ \triangleleft (b \oplus_p x) \triangleright_R x:[Q]_r^+$
 by (*rel-auto*)

lemma *rea-frame-ext-seq* [frame]:
vwb-lens $x \implies x:[P ;; Q]_r^+ = x:[P]_r^+ ;; x:[Q]_r^+$
apply (*simp add: rea-frame-ext-def rea-frame-def alpha frame*)
apply (*subst frame-seq*)
apply (*simp-all add: plus-vwb-lens closure*)
apply (*rel-auto*)
done

lemma *rea-frame-ext-subst-indep* [*usubst*]:
 assumes $x \bowtie y \ \Sigma \ \nmid v \ P$ is RR

shows $\sigma(y \mapsto_s v) \uparrow_S x:[P]_r^+ = (\sigma \uparrow_S x:[P]_r^+) ;; y :=_r v$
proof –
from *assms*(1–2) **have** $\sigma(y \mapsto_s v) \uparrow_S x:[RR\ P]_r^+ = (\sigma \uparrow_S x:[RR\ P]_r^+) ;; y :=_r v$
by (*rel-auto*, (*metis* (*no-types*, *lifting*) *lens-indep.lens-put-comm lens-indep-get*)+)
thus ?thesis
by (*simp add: Healthy-if assms*)
qed

lemma *rea-frame-ext-subst-within* [*usubst*]:
assumes *vwb-lens* *x vwb-lens* *y* $\Sigma \# v$ *P* *is* *RR*
shows $\sigma(x:y \mapsto_s v) \uparrow_S x:[P]_r^+ = (\sigma \uparrow_S x:[y :=_r (v \downarrow_e x) ;; P]_r^+)$
proof –
from *assms*(1,3) **have** $\sigma(x:y \mapsto_s v) \uparrow_S x:[RR\ P]_r^+ = (\sigma \uparrow_S x:[y :=_r (v \downarrow_e x) ;; RR(P)]_r^+)$
by (*rel-auto*, *metis*+)
thus ?thesis
by (*simp add: assms Healthy-if*)
qed

6.4 Stateful Reactive specifications

definition *rea-st-rel* :: '*s* *hrel* \Rightarrow ('*s*, '*t*::*trace*, ' α , ' β) *rel-rsp* ($[-]_S$) **where**
[upred-defs]: *rea-st-rel* *b* = ($[b]_S \wedge \$tr' =_u \tr)

definition *rea-st-rel'* :: '*s* *hrel* \Rightarrow ('*s*, '*t*::*trace*, ' α , ' β) *rel-rsp* ($[-]_{S'}$) **where**
[upred-defs]: *rea-st-rel'* *b* = *R1*($[b]_{S'}$)

definition *rea-st-cond* :: '*s* *upred* \Rightarrow ('*s*, '*t*::*trace*, ' α , ' β) *rel-rsp* ($[-]_{S<}$) **where**
[upred-defs]: *rea-st-cond* *b* = *R1*($[b]_{S<}$)

definition *rea-st-post* :: '*s* *upred* \Rightarrow ('*s*, '*t*::*trace*, ' α , ' β) *rel-rsp* ($[-]_{S>}$) **where**
[upred-defs]: *rea-st-post* *b* = *R1*($[b]_{S>}$)

lemma *lift-state-pre-unrest* [*unrest*]: $x \bowtie (\$st)_v \Longrightarrow x \# [P]_{S<}$
by (*rel-simp*, *simp add: lens-indep-def*)

lemma *rea-st-rel-unrest* [*unrest*]:
 $\llbracket x \bowtie (\$tr)_v; x \bowtie (\$tr')_v; x \bowtie (\$st)_v; x \bowtie (\$st')_v \rrbracket \Longrightarrow x \# [P]_{S<}$
by (*simp add: add: rea-st-cond-def R1-def unrest lens-indep-sym*)

lemma *rea-st-cond-unrest* [*unrest*]:
 $\llbracket x \bowtie (\$tr)_v; x \bowtie (\$tr')_v; x \bowtie (\$st)_v \rrbracket \Longrightarrow x \# [P]_{S<}$
by (*simp add: add: rea-st-cond-def R1-def unrest lens-indep-sym*)

lemma *subst-st-cond* [*usubst*]: $[\sigma]_{S\sigma} \uparrow [P]_{S<} = [\sigma \uparrow P]_{S<}$
by (*rel-auto*)

lemma *rea-st-cond-R1* [*closure*]: $[b]_{S<}$ *is* *R1*
by (*rel-auto*)

lemma *rea-st-cond-R2c* [*closure*]: $[b]_{S<}$ *is* *R2c*
by (*rel-auto*)

lemma *rea-st-rel-RR* [*closure*]: $[P]_S$ *is* *RR*
using *minus-zero-eq* **by** (*rel-auto*)

lemma *rea-st-rel'-RR* [*closure*]: $[P]_{S'}$ *is* *RR*

by (*rel-auto*)

lemma *st-subst-rel* [*usubst*]:
 $\sigma \uparrow_S [P]_S = [[\sigma]_s \uparrow P]_S$
by (*rel-auto*)

lemma *st-rel-cond* [*rpred*]:
 $[P \triangleleft b \triangleright_r Q]_S = [P]_S \triangleleft b \triangleright_R [Q]_S$
by (*rel-auto*)

lemma *st-rel-false* [*rpred*]: $[false]_S = false$
by (*rel-auto*)

lemma *st-rel-skip* [*rpred*]:
 $[II]_S = (II_r :: ('s, 't::trace) rdes)$
by (*rel-auto*)

lemma *st-rel-seq* [*rpred*]:
 $[P ;; Q]_S = [P]_S ;; [Q]_S$
by (*rel-auto*)

lemma *st-rel-conj* [*rpred*]:
 $[P \wedge Q]_S = ([P]_S \wedge [Q]_S)$
by (*rel-auto*)

lemma *rea-st-cond-RR* [*closure*]: $[b]_{S<} \text{ is } RR$
by (*rule RR-intro, simp-all add: unrest closure*)

lemma *rea-st-cond-RC* [*closure*]: $[b]_{S<} \text{ is } RC$
by (*rule RC-intro, simp add: closure, rel-auto*)

lemma *rea-st-cond-true* [*rpred*]: $[true]_{S<} = true_r$
by (*rel-auto*)

lemma *rea-st-cond-false* [*rpred*]: $[false]_{S<} = false$
by (*rel-auto*)

lemma *st-cond-not* [*rpred*]: $(\neg_r [P]_{S<}) = [\neg P]_{S<}$
by (*rel-auto*)

lemma *st-cond-conj* [*rpred*]: $([P]_{S<} \wedge [Q]_{S<}) = [P \wedge Q]_{S<}$
by (*rel-auto*)

lemma *st-rel-assigns* [*rpred*]:
 $[\langle \sigma \rangle_a]_S = (\langle \sigma \rangle_r :: ('a, 't::trace) rdes)$
by (*rel-auto*)

lemma *cond-st-distr*: $(P \triangleleft b \triangleright_R Q) ;; R = (P ;; R \triangleleft b \triangleright_R Q ;; R)$
by (*rel-auto*)

lemma *cond-st-miracle* [*rpred*]: $P \text{ is } R1 \implies P \triangleleft b \triangleright_R false = ([b]_{S<} \wedge P)$
by (*rel-blast*)

lemma *cond-st-true* [*rpred*]: $P \triangleleft true \triangleright_R Q = P$
by (*rel-blast*)

lemma *cond-st-false* [*rpred*]: $P \triangleleft \text{false} \triangleright_R Q = Q$
by (*rel-blast*)

lemma *st-cond-true-or* [*rpred*]: $P \text{ is } R1 \implies (R1 \text{ true} \triangleleft b \triangleright_R P) = ([b]_{S<} \vee P)$
by (*rel-blast*)

lemma *st-cond-left-impl-RC-closed* [*closure*]:
 $P \text{ is } RC \implies ([b]_{S<} \Rightarrow_r P) \text{ is } RC$
by (*simp add: rea-impl-def rpred closure*)

end

7 Reactive Weakest Preconditions

theory *utp-rea-wp*
imports *utp-rea-prog*
begin

definition *wp-rea* ::
 $('t::\text{trace}, 'a) \text{ hrel-rp} \Rightarrow$
 $('t, 'a) \text{ hrel-rp} \Rightarrow$
 $('t, 'a) \text{ hrel-rp} \text{ (infix wp}_r \text{ 60)}$
where [*upred-defs*]: $P \text{ wp}_r Q = (\neg_r P ;; (\neg_r Q))$

lemma *in-var-unrest-wp-rea* [*unrest*]: $\llbracket \$x \# P; tr \bowtie x \rrbracket \implies \$x \# (P \text{ wp}_r Q)$
by (*simp add: wp-rea-def unrest R1-def rea-not-def*)

lemma *out-var-unrest-wp-rea* [*unrest*]: $\llbracket \$x' \# Q; tr \bowtie x \rrbracket \implies \$x' \# (P \text{ wp}_r Q)$
by (*simp add: wp-rea-def unrest R1-def rea-not-def*)

lemma *wp-rea-R1* [*closure*]: $P \text{ wp}_r Q \text{ is } R1$
by (*rel-auto*)

lemma *wp-rea-RR-closed* [*closure*]: $\llbracket P \text{ is } RR; Q \text{ is } RR \rrbracket \implies P \text{ wp}_r Q \text{ is } RR$
by (*simp add: wp-rea-def closure*)

lemma *wp-rea-impl-lemma*:
 $((P \text{ wp}_r Q) \Rightarrow_r (R1(P) ;; R1(Q \Rightarrow_r R))) = ((P \text{ wp}_r Q) \Rightarrow_r (R1(P) ;; R1(R)))$
by (*rel-auto, blast*)

lemma *wpR-R1-right* [*wp*]:
 $P \text{ wp}_r R1(Q) = P \text{ wp}_r Q$
by (*rel-auto*)

lemma *wp-rea-true* [*wp*]: $P \text{ wp}_r \text{ true} = \text{true}_r$
by (*rel-auto*)

lemma *wp-rea-conj* [*wp*]: $P \text{ wp}_r (Q \wedge R) = (P \text{ wp}_r Q \wedge P \text{ wp}_r R)$
by (*simp add: wp-rea-def seqr-or-distr*)

lemma *wp-rea-USUP-mem* [*wp*]:
 $A \neq \{\} \implies P \text{ wp}_r (\bigsqcup_{i \in A} Q(i)) = (\bigsqcup_{i \in A} P \text{ wp}_r Q(i))$
by (*simp add: wp-rea-def seq-UNF-distl*)

lemma *wp-rea-Inf-pre* [wp]:
 $P \text{ wp}_r (\bigsqcup_{i \in \{0..n::\text{nat}\}} Q(i)) = (\bigsqcup_{i \in \{0..n\}} P \text{ wp}_r Q(i))$
by (*simp add: wp-rea-def seq-SUP-distl*)

lemma *wp-rea-div* [wp]:
 $(\neg_r P ;; \text{true}_r) = \text{true}_r \implies \text{true}_r \text{ wp}_r P = \text{false}$
by (*simp add: wp-rea-def rpred, rel-blast*)

lemma *wp-rea-st-cond-div* [wp]:
 $P \neq \text{true} \implies \text{true}_r \text{ wp}_r [P]_{S<} = \text{false}$
by (*rel-auto*)

lemma *wp-rea-cond* [wp]:
 $\text{out}_\alpha \# b \implies (P \triangleleft b \triangleright Q) \text{ wp}_r R = P \text{ wp}_r R \triangleleft b \triangleright Q \text{ wp}_r R$
by (*simp add: wp-rea-def cond-seq-left-distr, rel-auto*)

lemma *wp-rea-RC-false* [wp]:
 $P \text{ is } RC \implies (\neg_r P) \text{ wp}_r \text{false} = P$
by (*metis Healthy-if RC1-def RC-implies-RC1 rea-not-false wp-rea-def*)

lemma *wp-rea-seq* [wp]:
assumes $Q \text{ is } R1$
shows $(P ;; Q) \text{ wp}_r R = P \text{ wp}_r (Q \text{ wp}_r R)$ (*is ?lhs = ?rhs*)
proof –
have $?rhs = R1 (\neg P ;; R1 (Q ;; R1 (\neg R)))$
by (*simp add: wp-rea-def rea-not-def R1-negate-R1 assms*)
also have $\dots = R1 (\neg P ;; (Q ;; R1 (\neg R)))$
by (*metis Healthy-if R1-seqr assms*)
also have $\dots = R1 (\neg (P ;; Q) ;; R1 (\neg R))$
by (*simp add: seqr-assoc*)
finally show *?thesis*
by (*simp add: wp-rea-def rea-not-def*)
qed

lemma *wp-rea-skip* [wp]:
assumes $Q \text{ is } R1$
shows $II \text{ wp}_r Q = Q$
by (*simp add: wp-rea-def rpred assms Healthy-if*)

lemma *wp-rea-rea-skip* [wp]:
assumes $Q \text{ is } RR$
shows $II_r \text{ wp}_r Q = Q$
by (*simp add: wp-rea-def rpred closure assms Healthy-if*)

lemma *power-wp-rea-RR-closed* [closure]:
 $\llbracket R \text{ is } RR; P \text{ is } RR \rrbracket \implies R \hat{~} i \text{ wp}_r P \text{ is } RR$
by (*induct i, simp-all add: wp closure*)

lemma *wp-rea-rea-assigns* [wp]:
assumes $P \text{ is } RR$
shows $\langle \sigma \rangle_r \text{ wp}_r P = \lceil \sigma \rceil_{S\sigma} \dagger P$
proof –
have $\langle \sigma \rangle_r \text{ wp}_r (RR P) = \lceil \sigma \rceil_{S\sigma} \dagger (RR P)$
by (*rel-auto*)
thus *?thesis*

by (metis Healthy-def assms)
qed

lemma wp-rea-miracle [wp]: false wp_r Q = true_r
by (simp add: wp-rea-def)

lemma wp-rea-disj [wp]: (P ∨ Q) wp_r R = (P wp_r R ∧ Q wp_r R)
by (rel-blast)

lemma wp-rea-UINF [wp]:
assumes A ≠ {}
shows (⋂ x∈A • P(x)) wp_r Q = (⋂ x∈A • P(x) wp_r Q)
by (simp add: wp-rea-def rea-not-def seq-UINF-distr not-UINF R1-UINF assms)

lemma wp-rea-choice [wp]:
(P □ Q) wp_r R = (P wp_r R ∧ Q wp_r R)
by (rel-blast)

lemma wp-rea-UINF-ind [wp]:
(⋂ i • P(i)) wp_r Q = (⋂ i • P(i) wp_r Q)
by (simp add: wp-rea-def rea-not-def seq-UINF-distr' not-UINF-ind R1-UINF-ind)

lemma rea-assume-wp [wp]:
assumes P is RC
shows ([b][⊤]_r wp_r P) = ([b]_{S<} ⇒_r P)
proof –
have ([b][⊤]_r wp_r RC P) = ([b]_{S<} ⇒_r RC P)
by (rel-auto)
thus ?thesis
by (simp add: Healthy-if assms)
qed

lemma rea-star-wp [wp]:
assumes P is RR Q is RR
shows P^{★r} wp_r Q = (⋂ i • P ^ i wp_r Q)
proof –
have P^{★r} wp_r Q = (Q ∧ P⁺ wp_r Q)
by (simp add: assms rrel-thy.Star-alt-def wp-rea-choice wp-rea-rea-skip)
also have ... = (II wp_r Q ∧ (⋂ i • P ^ Suc i wp_r Q))
by (simp add: uplus-power-def wp closure assms)
also have ... = (⋂ i • P ^ i wp_r Q)
proof –
have P[★] wp_r Q = P^{★r} wp_r Q
by (metis (no-types) RA1 assms(2) rea-no-RR rea-skip-unit(2) rrel-thy.Star-def wp-rea-def)
then show ?thesis
by (simp add: calculation ustar-def wp-rea-UINF-ind)
qed
finally show ?thesis .
qed

lemma wp-rea-R2-closed [closure]:
[[P is R2; Q is R2]] ⇒ P wp_r Q is R2
by (simp add: wp-rea-def closure)

lemma wp-rea-false-RC1': P is R2 ⇒ RC1(P wp_r false) = P wp_r false

by (simp add: wp-rea-def RC1-def closure rpred segr-assoc)

lemma wp-rea-false-RC1: $P \text{ is } R2 \implies P \text{ wp}_r \text{ false is } RC1$
 by (simp add :Healthy-def wp-rea-false-RC1')

lemma wp-rea-false-RR:
 $\llbracket \$ok \# P; \$wait \# P; P \text{ is } R2 \rrbracket \implies P \text{ wp}_r \text{ false is } RR$
 by (rule RR-R2-intro, simp-all add: unrest closure)

lemma wp-rea-false-RC:
 $\llbracket \$ok \# P; \$wait \# P; P \text{ is } R2 \rrbracket \implies P \text{ wp}_r \text{ false is } RC$
 by (rule RC-intro', simp-all add: wp-rea-false-RC1 wp-rea-false-RR)

lemma wp-rea-RC1: $\llbracket P \text{ is } RR; Q \text{ is } RC \rrbracket \implies P \text{ wp}_r Q \text{ is } RC1$
 by (rule Healthy-intro, simp add: wp-rea-def RC1-def rpred closure segr-assoc RC1-prop RC-implies-RC1)

lemma wp-rea-RC [closure]: $\llbracket P \text{ is } RR; Q \text{ is } RC \rrbracket \implies P \text{ wp}_r Q \text{ is } RC$
 by (rule RC-intro', simp-all add: wp-rea-RC1 closure)

lemma wpR-power-RC-closed [closure]:
 assumes $P \text{ is } RR \ Q \text{ is } RC$
 shows $P \wedge i \text{ wp}_r Q \text{ is } RC$
 by (metis RC-implies-RR RR-implies-R1 assms power.power-eq-if power-Suc-RR-closed wp-rea-RC wp-rea-skip)

end

8 Reactive Hoare Logic

theory utp-rea-hoare
imports utp-rea-prog
begin

definition hoare-rp :: $'\alpha \text{ upred} \Rightarrow (' \alpha, \text{real pos}) \text{ rdes} \Rightarrow ' \alpha \text{ upred} \Rightarrow \text{bool} (\llbracket - \rrbracket / - / \llbracket - \rrbracket_r)$ **where**
 $[\text{upred-defs}]: \text{hoare-rp } p \ Q \ r = ((\llbracket p \rrbracket_{S<} \Rightarrow \llbracket r \rrbracket_{S>}) \sqsubseteq Q)$

lemma hoare-rp-conseq:
 $\llbracket 'p \Rightarrow p'; 'q' \Rightarrow q'; \llbracket p' \rrbracket S \llbracket q' \rrbracket_r \rrbracket \implies \llbracket p \rrbracket S \llbracket q \rrbracket_r$
 by (rel-auto)

lemma hoare-rp-weaken:
 $\llbracket 'p \Rightarrow p'; \llbracket p' \rrbracket S \llbracket q \rrbracket_r \rrbracket \implies \llbracket p \rrbracket S \llbracket q \rrbracket_r$
 by (rel-auto)

lemma hoare-rp-strengthen:
 $\llbracket 'q' \Rightarrow q'; \llbracket p \rrbracket S \llbracket q' \rrbracket_r \rrbracket \implies \llbracket p \rrbracket S \llbracket q \rrbracket_r$
 by (rel-auto)

lemma false-pre-hoare-rp [hoare-safe]: $\llbracket \text{false} \rrbracket P \llbracket q \rrbracket_r$
 by (rel-auto)

lemma true-post-hoare-rp [hoare-safe]: $\llbracket p \rrbracket Q \llbracket \text{true} \rrbracket_r$
 by (rel-auto)

lemma miracle-hoare-rp [hoare-safe]: $\llbracket p \rrbracket \text{false} \llbracket q \rrbracket_r$

by (*rel-auto*)

lemma *assigns-hoare-rp* [*hoare-safe*]: ‘ $p \Rightarrow \sigma \dagger q$ ’ $\Longrightarrow \llbracket p \rrbracket \langle \sigma \rangle_r \llbracket q \rrbracket_r$
by *rel-auto*

lemma *skip-hoare-rp* [*hoare-safe*]: $\llbracket p \rrbracket II_r \llbracket p \rrbracket_r$
by *rel-auto*

lemma *seq-hoare-rp*: $\llbracket \llbracket p \rrbracket Q_1 \llbracket s \rrbracket_r ; \llbracket s \rrbracket Q_2 \llbracket r \rrbracket_r \rrbracket \Longrightarrow \llbracket p \rrbracket Q_1 ;; Q_2 \llbracket r \rrbracket_r$
by (*rel-auto*)

lemma *seq-est-hoare-rp* [*hoare-safe*]:
 $\llbracket \llbracket true \rrbracket Q_1 \llbracket p \rrbracket_r ; \llbracket p \rrbracket Q_2 \llbracket p \rrbracket_r \rrbracket \Longrightarrow \llbracket true \rrbracket Q_1 ;; Q_2 \llbracket p \rrbracket_r$
by (*rel-auto*)

lemma *seq-inv-hoare-rp* [*hoare-safe*]:
 $\llbracket \llbracket p \rrbracket Q_1 \llbracket p \rrbracket_r ; \llbracket p \rrbracket Q_2 \llbracket p \rrbracket_r \rrbracket \Longrightarrow \llbracket p \rrbracket Q_1 ;; Q_2 \llbracket p \rrbracket_r$
by (*rel-auto*)

lemma *cond-hoare-rp* [*hoare-safe*]:
 $\llbracket \llbracket b \wedge p \rrbracket P \llbracket r \rrbracket_r ; \llbracket \neg b \wedge p \rrbracket Q \llbracket r \rrbracket_r \rrbracket \Longrightarrow \llbracket p \rrbracket P \triangleleft b \triangleright_R Q \llbracket r \rrbracket_r$
by (*rel-auto*)

lemma *repeat-hoare-rp* [*hoare-safe*]:
 $\llbracket p \rrbracket Q \llbracket p \rrbracket_r \Longrightarrow \llbracket p \rrbracket Q \hat{\ }^n \llbracket p \rrbracket_r$
by (*induct n, rel-auto+*)

lemma *UINF-ind-hoare-rp* [*hoare-safe*]:
 $\llbracket \bigwedge i. \llbracket p \rrbracket Q(i) \llbracket r \rrbracket_r \rrbracket \Longrightarrow \llbracket p \rrbracket \bigcap i \cdot Q(i) \llbracket r \rrbracket_r$
by (*rel-auto*)

lemma *star-hoare-rp* [*hoare-safe*]:
 $\llbracket p \rrbracket Q \llbracket p \rrbracket_r \Longrightarrow \llbracket p \rrbracket Q^* \llbracket p \rrbracket_r$
by (*simp add: ustar-def hoare-safe*)

lemma *conj-hoare-rp* [*hoare-safe*]:
 $\llbracket \llbracket p_1 \rrbracket Q_1 \llbracket r_1 \rrbracket_r ; \llbracket p_2 \rrbracket Q_2 \llbracket r_2 \rrbracket_r \rrbracket \Longrightarrow \llbracket p_1 \wedge p_2 \rrbracket Q_1 \wedge Q_2 \llbracket r_1 \wedge r_2 \rrbracket_r$
by (*rel-auto*)

lemma *iter-hoare-rp* [*hoare-safe*]:
 $\llbracket I \rrbracket P \llbracket I \rrbracket_r \Longrightarrow \llbracket I \rrbracket P^{\star r} \llbracket I \rrbracket_r$
by (*simp add: star-hoare-rp utp-star-def rrel-unit-def seq-inv-hoare-rp skip-hoare-rp*)

end

9 Meta-theory for Generalised Reactive Processes

theory *utp-reactive*

imports

utp-rea-core

utp-rea-healths

utp-rea-parallel

utp-rea-rel

utp-rea-cond

utp-rea-prog

utp-rea-up
utp-rea-hoare
begin end

References

- [1] A. Butterfield, P. Gancarski, and J. Woodcock. State visibility and communication in unifying theories of programming. *Theoretical Aspects of Software Engineering*, 0:47–54, 2009.
- [2] A. Cavalcanti and J. Woodcock. A tutorial introduction to CSP in unifying theories of programming. In *Refinement Techniques in Software Engineering*, volume 3167 of *LNCS*, pages 220–268. Springer, 2006.
- [3] S. Foster, A. Cavalcanti, J. Woodcock, and F. Zeyda. Unifying theories of time with generalised reactive processes. *Accepted for Information Processing Letters*, Dec 2017. Preprint: <https://arxiv.org/abs/1712.10213>.
- [4] T. Hoare and J. He. *Unifying Theories of Programming*. Prentice-Hall, 1998.