

# Circus in Isabelle/UTP

Simon Foster      James Baxter      Ana Cavalcanti      Jim Woodcock  
                         Samuel Canham

November 16, 2018

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Circus Trace Merge</b>	<b>1</b>
2.1	Function Definition . . . . .	1
2.2	Lifted Trace Merge . . . . .	2
2.3	Trace Merge Lemmas . . . . .	2
<b>3</b>	<b>Syntax and Translations for Event Prefix</b>	<b>3</b>
<b>4</b>	<b>Circus Parallel Composition</b>	<b>6</b>
4.1	Merge predicates . . . . .	6
4.2	Parallel operator . . . . .	17
4.3	Parallel Laws . . . . .	19
<b>5</b>	<b>Hiding</b>	<b>30</b>
5.1	Hiding in peri- and postconditions . . . . .	30
5.2	Hiding in preconditions . . . . .	31
5.3	Hiding Operator . . . . .	32
<b>6</b>	<b>Meta theory for Circus</b>	<b>34</b>

## 1 Introduction

This document contains a mechanisation in Isabelle/UTP [1] of Circus [2].

## 2 Circus Trace Merge

```
theory utp-circus-traces
  imports UTP-Stateful-Failures.utp-sf-rdes
begin
```

### 2.1 Function Definition

```
fun tr-par ::
  'a set ⇒ 'a list ⇒ 'a list ⇒ 'a list set where
tr-par cs [] = {} |
tr-par cs (e # t) [] = (if e ∈ cs then {} else {[e]} ∩ (tr-par cs t [])) |
```

$tr\text{-}par\ cs\ []\ (e\ \# \ t) = (if\ e \in cs\ then\ \{\}\ else\ \{e\}) \frown (tr\text{-}par\ cs\ []\ t) \mid$   
 $tr\text{-}par\ cs\ (e_1\ \# \ t_1)\ (e_2\ \# \ t_2) =$   
 $(if\ e_1 = e_2$   
 $\quad then$   
 $\quad if\ e_1 \in cs$   
 $\quad \quad then\ \{[e_1]\} \frown (tr\text{-}par\ cs\ t_1\ t_2)$   
 $\quad \quad else$   
 $\quad \quad (\{[e_1]\} \frown (tr\text{-}par\ cs\ t_1\ (e_2\ \# \ t_2))) \cup$   
 $\quad \quad (\{[e_2]\} \frown (tr\text{-}par\ cs\ (e_1\ \# \ t_1)\ t_2))$   
 $\quad else$   
 $\quad if\ e_1 \in cs\ then$   
 $\quad \quad if\ e_2 \in cs\ then\ \{\}$   
 $\quad \quad else$   
 $\quad \quad \{[e_2]\} \frown (tr\text{-}par\ cs\ (e_1\ \# \ t_1)\ t_2)$   
 $\quad else$   
 $\quad \quad if\ e_2 \in cs\ then$   
 $\quad \quad \quad \{[e_1]\} \frown (tr\text{-}par\ cs\ t_1\ (e_2\ \# \ t_2))$   
 $\quad \quad \quad else$   
 $\quad \quad \quad (\{[e_1]\} \frown (tr\text{-}par\ cs\ t_1\ (e_2\ \# \ t_2))) \cup$   
 $\quad \quad \quad (\{[e_2]\} \frown (tr\text{-}par\ cs\ (e_1\ \# \ t_1)\ t_2))$

**abbreviation**  $tr\text{-}inter :: 'a\ list \Rightarrow 'a\ list \Rightarrow 'a\ list\ set$  (**infixr**  $|||_t\ 100$ ) **where**  
 $x\ |||_t\ y \equiv tr\text{-}par\ \{\}\ x\ y$

## 2.2 Lifted Trace Merge

**syntax**  $-utr\text{-}par ::$   
 $logic \Rightarrow logic \Rightarrow logic \Rightarrow logic\ ((- \ \star_- / \ -)\ [100, 0, 101]\ 100)$

The function *trop* is used to lift ternary operators.

**translations**

$t1\ \star_{cs}\ t2 == (CONST\ bop)\ (CONST\ tr\text{-}par\ cs)\ t1\ t2$

## 2.3 Trace Merge Lemmas

**lemma** *tr-par-empty*:

$tr\text{-}par\ cs\ t1\ [] = \{takeWhile\ (\lambda x. x \notin cs)\ t1\}$

$tr\text{-}par\ cs\ []\ t2 = \{takeWhile\ (\lambda x. x \notin cs)\ t2\}$

— Subgoal 1

**apply** (*induct t1; simp*)

— Subgoal 2

**apply** (*induct t2; simp*)

**done**

**lemma** *tr-par-sym*:

$tr\text{-}par\ cs\ t1\ t2 = tr\text{-}par\ cs\ t2\ t1$

**apply** (*induct t1 arbitrary; t2*)

— Subgoal 1

**apply** (*simp add: tr-par-empty*)

— Subgoal 2

**apply** (*induct-tac t2*)

— Subgoal 2.1

**apply** (*clarsimp*)

— Subgoal 2.2

**apply** (*clarsimp*)

**apply** (*blast*)

done

**lemma** *tr-inter-sym*:  $x \parallel_t y = y \parallel_t x$   
 by (*simp add: tr-par-sym*)

**lemma** *trace-merge-nil* [*simp*]:  $x \star_{\{\}} \langle \rangle = \{x\}_u$   
 by (*pred-auto, simp-all add: tr-par-empty, metis takeWhile-eq-all-conv*)

**lemma** *trace-merge-empty* [*simp*]:  
 $(\langle \rangle \star_{cs} \langle \rangle) = \{\langle \rangle\}_u$   
 by (*rel-auto*)

**lemma** *trace-merge-single-empty* [*simp*]:  
 $a \in cs \implies \langle \ll a \gg \rangle \star_{cs} \langle \rangle = \{\langle \rangle\}_u$   
 by (*rel-auto*)

**lemma** *trace-merge-empty-single* [*simp*]:  
 $a \in cs \implies \langle \rangle \star_{cs} \langle \ll a \gg \rangle = \{\langle \rangle\}_u$   
 by (*rel-auto*)

**lemma** *trace-merge-commute*:  $t_1 \star_{cs} t_2 = t_2 \star_{cs} t_1$   
 by (*rel-simp, simp add: tr-par-sym*)

**lemma** *csp-trace-simps* [*simp*]:  
 $v + \langle \rangle = v \langle \rangle + v = v$   
 $bop \ (\#) \ x \ xs \ \hat{\ }_u \ ys = bop \ (\#) \ x \ (xs \ \hat{\ }_u \ ys)$   
 by (*rel-auto*)+

end

### 3 Syntax and Translations for Event Prefix

**theory** *utp-circus-prefix*  
**imports** *UTP-Stateful-Failures.utp-sf-rdes*  
**begin**

**syntax**  
*-simple-prefix* :: *logic*  $\Rightarrow$  *logic*  $\Rightarrow$  *logic*  $(- \rightarrow - [63, 62] 62)$

**translations**  
 $a \rightarrow P == \text{CONST PrefixCSP } \ll a \gg P$

We next configure a syntax for mixed prefixes.

**nonterminal** *prefix-elem'* **and** *mixed-prefix'*

**syntax** *-end-prefix* :: *prefix-elem'*  $\Rightarrow$  *mixed-prefix'*  $(-)$

Input Prefix:  $\dots?(x)$

**syntax** *-simple-input-prefix* :: *id*  $\Rightarrow$  *prefix-elem'*  $(?'(-'))$

Input Prefix with Constraint:  $\dots?(x : P)$

**syntax** *-input-prefix* :: *id*  $\Rightarrow$   $(\sigma, \varepsilon)$  *action*  $\Rightarrow$  *prefix-elem'*  $(?'(- : / -'))$

Output Prefix:  $\dots![v]e$

A variable name must currently be provided for outputs, too. Fix?!

**syntax** *-output-prefix* :: ('a, 'σ) uexpr ⇒ prefix-elim' (!'(-'))

**syntax** *-output-prefix* :: ('a, 'σ) uexpr ⇒ prefix-elim' ('(-'))

**syntax** (**output**) *-output-prefix-pp* :: ('a, 'σ) uexpr ⇒ prefix-elim' (!'(-'))

**syntax**

*-prefix-aux* :: pttm ⇒ logic ⇒ prefix-elim'

Mixed-Prefix Action:  $c \dots (prefix) \rightarrow A$

**syntax** *-mixed-prefix* :: prefix-elim' ⇒ mixed-prefix' ⇒ mixed-prefix' (--)

**syntax**

*-prefix-action* ::

('a, 'ε) chan ⇒ mixed-prefix' ⇒ ('σ, 'ε) action ⇒ ('σ, 'ε) action

((-- →/ -) [63, 63, 62] 62)

Syntax translations

**definition** *lconj* :: ('a ⇒ 'α upred) ⇒ ('b ⇒ 'α upred) ⇒ ('a × 'b ⇒ 'α upred) (**infixr** ∧<sub>l</sub> 35)

**where** [*upred-defs*]: (P ∧<sub>l</sub> Q) ≡ (λ (x,y). P x ∧ Q y)

**definition** *outp-constraint* (**infix** =<sub>o</sub> 60) **where**

[*upred-defs*]: *outp-constraint* v ≡ (λ x. <x> =<sub>u</sub> v)

**translations**

*-simple-input-prefix* x ≡ *-input-prefix* x true

*-mixed-prefix* (*-input-prefix* x P) (*-prefix-aux* y Q) →

*-prefix-aux* (*-pattern* x y) ((λ x. P) ∧<sub>l</sub> Q)

*-mixed-prefix* (*-output-prefix* P) (*-prefix-aux* y Q) →

*-prefix-aux* (*-pattern* -idtdummy y) ((CONST *outp-constraint* P) ∧<sub>l</sub> Q)

*-end-prefix* (*-input-prefix* x P) → *-prefix-aux* x (λ x. P)

*-end-prefix* (*-output-prefix* P) → *-prefix-aux* -idtdummy (CONST *outp-constraint* P)

*-prefix-action* c (*-prefix-aux* x P) A == (CONST *InputCSP*) c P (λx. A)

Basic print translations; more work needed

**translations**

*-simple-input-prefix* x <= *-input-prefix* x true

*-output-prefix* v <= *-prefix-aux* p (CONST *outp-constraint* v)

*-output-prefix* u (*-output-prefix* v)

<= *-prefix-aux* p (λ(x1, y1). CONST *outp-constraint* u x2 ∧ CONST *outp-constraint* v y2)

*-input-prefix* x P <= *-prefix-aux* v (λx. P)

x!(v) → P <= CONST *OutputCSP* x v P

**term** x!(1)!(y) → P

**term** x?(v) → P

**term** x?(v:false) → P

**term** x!((1)) → P

**term** x?(v)!(1) → P

**term** x!((1))!(2)?(v:true) → P

Basic translations for state variable communications

**syntax**

*-csp-input-var* :: logic ⇒ id ⇒ logic ⇒ logic (-?%- [63, 0, 60] 62)

*-csp-inputu-var* :: logic ⇒ id ⇒ logic (-?%- [63, 60] 62)

## translations

$c?\$x:A \rightarrow \text{CONST InputVarCSP } c \ x \ A$   
 $c?\$x \rightarrow \text{CONST InputVarCSP } c \ x \ (\lambda \ x. \text{true})$   
 $c?\$x:A \leq \text{CONST InputVarCSP } c \ x \ (\lambda \ x'. \ A)$   
 $c?\$x \leq c?\$x:\text{true}$

## lemma outp-constraint-prod:

$(\text{outp-constraint } \ll a \gg x \wedge \text{outp-constraint } \ll b \gg y) =$   
 $\text{outp-constraint } \ll (a, b) \gg (x, y)$   
**by** (*simp add: outp-constraint-def, pred-auto*)

## lemma subst-outp-constraint [usubst]:

$\sigma \uparrow (v =_o x) = (\sigma \uparrow v =_o x)$   
**by** (*rel-auto*)

## lemma UINF-one-point-simp [rpred]:

$\ll \bigwedge i. P \ i \text{ is } R1 \gg \implies (\bigcap x \cdot \ll i \gg =_o x)_{S<} \wedge P(x) = P(i)$   
**by** (*rel-blast*)

## lemma USUP-one-point-simp [rpred]:

$\ll \bigwedge i. P \ i \text{ is } R1 \gg \implies (\bigcup x \cdot \ll i \gg =_o x)_{S<} \Rightarrow_r P(x) = P(i)$   
**by** (*rel-blast*)

## lemma USUP-eq-event-eq [rpred]:

**assumes**  $\bigwedge y. P(y) \text{ is } RR$   
**shows**  $(\bigcup y \cdot [v =_o y]_{S<} \Rightarrow_r P(y)) = P(y) \ll y \rightarrow [v]_{S\leftarrow} \gg$

**proof** –

**have**  $(\bigcup y \cdot [v =_o y]_{S<} \Rightarrow_r RR(P(y))) = RR(P(y)) \ll y \rightarrow [v]_{S\leftarrow} \gg$   
**apply** (*rel-simp, safe*)  
**apply** *metis*  
**apply** *blast*  
**apply** *simp*  
**done**

**thus** *?thesis*

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

**qed**

## lemma UINF-eq-event-eq [rpred]:

**assumes**  $\bigwedge y. P(y) \text{ is } RR$   
**shows**  $(\bigcap y \cdot [v =_o y]_{S<} \wedge P(y)) = P(y) \ll y \rightarrow [v]_{S\leftarrow} \gg$

**proof** –

**have**  $(\bigcap y \cdot [v =_o y]_{S<} \wedge RR(P(y))) = RR(P(y)) \ll y \rightarrow [v]_{S\leftarrow} \gg$   
**by** (*rel-simp, safe, metis*)

**thus** *?thesis*

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

**qed**

Proofs that the input constrained parser versions of output is the same as the regular definition.

## lemma output-prefix-is-OutputCSP [simp]:

**assumes**  $A \text{ is } NCSP$   
**shows**  $x!(P) \rightarrow A = \text{OutputCSP } x \ P \ A$  (**is** *?lhs = ?rhs*)  
**by** (*rule SRD-eq-intro, simp-all add: assms closure rdes, rel-auto+*)

## lemma OutputCSP-pair-simp [simp]:

$P$  is NCSP  $\implies a.(\ll i \gg).(\ll j \gg) \rightarrow P = \text{OutputCSP } a \ll (i, j) \gg P$   
**using** *output-prefix-is-OutputCSP*[of  $P$   $a$ ]  
**by** (*simp add: outp-constraint-prod lconj-def InputCSP-def closure del: output-prefix-is-OutputCSP*)

**lemma** *OutputCSP-triple-simp* [*simp*]:

$P$  is NCSP  $\implies a.(\ll i \gg).(\ll j \gg).(\ll k \gg) \rightarrow P = \text{OutputCSP } a \ll (i, j, k) \gg P$   
**using** *output-prefix-is-OutputCSP*[of  $P$   $a$ ]  
**by** (*simp add: outp-constraint-prod lconj-def InputCSP-def closure del: output-prefix-is-OutputCSP*)

**end**

## 4 Circus Parallel Composition

**theory** *utp-circus-parallel*

**imports**

*utp-circus-prefix*

*utp-circus-traces*

**begin**

### 4.1 Merge predicates

**definition** *CSPInnerMerge* ::  $(\alpha \implies \sigma) \Rightarrow \psi \text{ set} \Rightarrow (\beta \implies \sigma) \Rightarrow ((\sigma, \psi) \text{ st-csp}) \text{ merge } (N_C)$  **where**  
[*upred-defs*]:

$\text{CSPInnerMerge } ns1 \text{ cs } ns2 =$   
 $\$ref' \subseteq_u ((\$0-ref \cup_u \$1-ref) \cap_u \ll cs \gg) \cup_u ((\$0-ref \cap_u \$1-ref) - \ll cs \gg) \wedge$   
 $\$tr_{<} \leq_u \$tr' \wedge$   
 $(\$tr' - \$tr_{<}) \in_u (\$0-tr - \$tr_{<}) \star_{cs} (\$1-tr - \$tr_{<}) \wedge$   
 $(\$0-tr - \$tr_{<}) \upharpoonright_u \ll cs \gg =_u (\$1-tr - \$tr_{<}) \upharpoonright_u \ll cs \gg \wedge$   
 $\$st' =_u (\$st_{<} \oplus \$0-st \text{ on } \&ns1) \oplus \$1-st \text{ on } \&ns2)$

**definition** *CSPInnerInterleave* ::  $(\alpha \implies \sigma) \Rightarrow (\beta \implies \sigma) \Rightarrow ((\sigma, \psi) \text{ st-csp}) \text{ merge } (N_I)$  **where**  
[*upred-defs*]:

$N_I \text{ ns1 ns2} =$   
 $\$ref' \subseteq_u (\$0-ref \cap_u \$1-ref) \wedge$   
 $\$tr_{<} \leq_u \$tr' \wedge$   
 $(\$tr' - \$tr_{<}) \in_u (\$0-tr - \$tr_{<}) \star_{\{\}} (\$1-tr - \$tr_{<}) \wedge$   
 $\$st' =_u (\$st_{<} \oplus \$0-st \text{ on } \&ns1) \oplus \$1-st \text{ on } \&ns2)$

An intermediate merge hides the state, whilst a final merge hides the refusals.

**definition** *CSPInterMerge* **where**

[*upred-defs*]:  $\text{CSPInterMerge } P \text{ ns1 cs ns2 } Q = (P \parallel_{(\exists \$st' \cdot N_C \text{ ns1 cs ns2})} Q)$

**definition** *CSPFinalMerge* **where**

[*upred-defs*]:  $\text{CSPFinalMerge } P \text{ ns1 cs ns2 } Q = (P \parallel_{(\exists \$ref' \cdot N_C \text{ ns1 cs ns2})} Q)$

**syntax**

*-cinter-merge* ::  $logic \Rightarrow salpha \Rightarrow logic \Rightarrow salpha \Rightarrow logic \Rightarrow logic \text{ (- } \ll \cdot \mid \cdot \mid \cdot \rrbracket^I \text{ - } [85, 0, 0, 0, 86] \text{ 86)}$   
*-cfinal-merge* ::  $logic \Rightarrow salpha \Rightarrow logic \Rightarrow salpha \Rightarrow logic \Rightarrow logic \text{ (- } \ll \cdot \mid \cdot \mid \cdot \rrbracket^F \text{ - } [85, 0, 0, 0, 86] \text{ 86)}$   
*-wrC* ::  $logic \Rightarrow salpha \Rightarrow logic \Rightarrow salpha \Rightarrow logic \Rightarrow logic \text{ (- wr } \ll \cdot \mid \cdot \mid \cdot \rrbracket_C \text{ - } [85, 0, 0, 0, 86] \text{ 86)}$

**translations**

*-cinter-merge*  $P \text{ ns1 cs ns2 } Q == \text{CONST } \text{CSPInterMerge } P \text{ ns1 cs ns2 } Q$   
*-cfinal-merge*  $P \text{ ns1 cs ns2 } Q == \text{CONST } \text{CSPFinalMerge } P \text{ ns1 cs ns2 } Q$   
*-wrC*  $P \text{ ns1 cs ns2 } Q == P \text{ wr}_R(N_C \text{ ns1 cs ns2}) Q$

**lemma** *CSPInnerMerge-R2m* [closure]:  $N_C \text{ ns1 cs ns2 is R2m}$   
 by (rel-auto)

**lemma** *CSPInnerMerge-RDM* [closure]:  $N_C \text{ ns1 cs ns2 is RDM}$   
 by (rule RDM-intro, simp add: closure, simp-all add: CSPInnerMerge-def unrest)

**lemma** *ex-ref'-R2m-closed* [closure]:

assumes  $P \text{ is R2m}$   
 shows  $(\exists \text{ \$ref' } \cdot P) \text{ is R2m}$

**proof** –

have  $R2m(\exists \text{ \$ref' } \cdot R2m P) = (\exists \text{ \$ref' } \cdot R2m P)$   
 by (rel-auto)

thus ?thesis  
 by (metis Healthy-def' assms)

**qed**

**lemma** *CSPInnerMerge-unrests* [unrest]:

$\$ok_{<} \# N_C \text{ ns1 cs ns2}$   
 $\$wait_{<} \# N_C \text{ ns1 cs ns2}$   
 by (rel-auto)+

**lemma** *CSPInterMerge-RR-closed* [closure]:

assumes  $P \text{ is RR } Q \text{ is RR}$   
 shows  $P \llbracket ns1 | cs | ns2 \rrbracket^I Q \text{ is RR}$   
 by (simp add: CSPInterMerge-def parallel-RR-closed assms closure unrest)

**lemma** *CSPInterMerge-unrest-ref* [unrest]:

assumes  $P \text{ is CRR } Q \text{ is CRR}$   
 shows  $\$ref \# P \llbracket ns1 | cs | ns2 \rrbracket^I Q$

**proof** –

have  $\$ref \# CRR(P) \llbracket ns1 | cs | ns2 \rrbracket^I CRR(Q)$   
 by (rel-blast)

thus ?thesis  
 by (simp add: Healthy-if assms)

**qed**

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

$\$st' \# P \llbracket ns1 | cs | ns2 \rrbracket^I Q$   
 by (rel-auto)

**lemma** *CSPInterMerge-CRR-closed* [closure]:

assumes  $P \text{ is CRR } Q \text{ is CRR}$   
 shows  $P \llbracket ns1 | cs | ns2 \rrbracket^I Q \text{ is CRR}$   
 by (simp add: CRR-implies-RR CRR-intro CSPInterMerge-RR-closed CSPInterMerge-unrest-ref assms)

**lemma** *CSPFinalMerge-RR-closed* [closure]:

assumes  $P \text{ is RR } Q \text{ is RR}$   
 shows  $P \llbracket ns1 | cs | ns2 \rrbracket^F Q \text{ is RR}$   
 by (simp add: CSPFinalMerge-def parallel-RR-closed assms closure unrest)

**lemma** *CSPFinalMerge-unrest-ref* [unrest]:

assumes  $P \text{ is CRR } Q \text{ is CRR}$   
 shows  $\$ref \# P \llbracket ns1 | cs | ns2 \rrbracket^F Q$

**proof** –

have  $\$ref \# CRR(P) \llbracket ns1 | cs | ns2 \rrbracket^F CRR(Q)$

by (rel-blast)  
 thus ?thesis  
 by (simp add: Healthy-if assms)  
 qed

**lemma** *CSPFinalMerge-CRR-closed* [closure]:  
 assumes  $P$  is CRR  $Q$  is CRR  
 shows  $P \llbracket ns1 | cs | ns2 \rrbracket^F Q$  is CRR  
 by (simp add: CRR-implies-RR CRR-intro CSPFinalMerge-RR-closed CSPFinalMerge-unrest-ref assms)

**lemma** *CSPInnerMerge-empty-Interleave*:  
 $N_C ns1 \{\} ns2 = N_I ns1 ns2$   
 by (rel-auto)

**definition** *CSPMerge* ::  $('α \implies 'σ) \Rightarrow 'ψ \text{ set} \Rightarrow ('β \implies 'σ) \Rightarrow (('σ, 'ψ) \text{ st-csp}) \text{ merge } (M_C)$  **where**  
 $[upred-defs]: M_C ns1 cs ns2 = M_R(N_C ns1 cs ns2) ;; Skip$

**definition** *CSPInterleave* ::  $('α \implies 'σ) \Rightarrow ('β \implies 'σ) \Rightarrow (('σ, 'ψ) \text{ st-csp}) \text{ merge } (M_I)$  **where**  
 $[upred-defs]: M_I ns1 ns2 = M_R(N_I ns1 ns2) ;; Skip$

**lemma** *swap-CSPInnerMerge*:  
 $ns1 \bowtie ns2 \implies swap_m ;; (N_C ns1 cs ns2) = (N_C ns2 cs ns1)$   
 apply (rel-auto)  
 using tr-par-sym apply blast  
 apply (simp add: lens-indep-comm)  
 using tr-par-sym apply blast  
 apply (simp add: lens-indep-comm)  
 done

**lemma** *SymMerge-CSPInnerMerge-NS* [closure]:  $N_C 0_L cs 0_L$  is SymMerge  
 by (simp add: Healthy-def swap-CSPInnerMerge)

**lemma** *SymMerge-CSPInnerInterleave* [closure]:  
 $N_I 0_L 0_L$  is SymMerge  
 by (metis CSPInnerMerge-empty-Interleave SymMerge-CSPInnerMerge-NS)

**lemma** *SymMerge-CSPInnerInterleave* [closure]:  
 $AssocMerge (N_I 0_L 0_L)$   
 apply (rel-auto)  
 apply (rename-tac tr tr<sub>2</sub>' ref<sub>0</sub> tr<sub>0</sub>' ref<sub>0</sub>' tr<sub>1</sub>' ref<sub>1</sub>' tr' ref<sub>2</sub>' tr<sub>i</sub>' ref<sub>3</sub>')  
 oops

**lemma** *CSPInterMerge-false* [rpred]:  $P \llbracket ns1 | cs | ns2 \rrbracket^I false = false$   
 by (simp add: CSPInterMerge-def)

**lemma** *CSPFinalMerge-false* [rpred]:  $P \llbracket ns1 | cs | ns2 \rrbracket^F false = false$   
 by (simp add: CSPFinalMerge-def)

**lemma** *CSPInterMerge-commute*:  
 assumes  $ns1 \bowtie ns2$   
 shows  $P \llbracket ns1 | cs | ns2 \rrbracket^I Q = Q \llbracket ns2 | cs | ns1 \rrbracket^I P$   
**proof** –  
 have  $P \llbracket ns1 | cs | ns2 \rrbracket^I Q = P \parallel_{\exists} \$st' \cdot N_C ns1 cs ns2 Q$   
 by (simp add: CSPInterMerge-def)  
 also have  $\dots = P \parallel_{\exists} \$st' \cdot (swap_m ;; N_C ns2 cs ns1) Q$



by (simp add: swap-CSPInnerMerge lens-indep-sym assms)  
 also have ... =  $P \parallel_{\text{swap}_m} ; (\exists \$st' \cdot N_C \text{ ns2 cs ns1}) Q$   
 by (simp add: seqr-exists-right)  
 also have ... =  $Q \parallel_{(\exists \$st' \cdot N_C \text{ ns2 cs ns1})} P$   
 by (simp add: par-by-merge-commute-swap[THEN sym])  
 also have ... =  $Q \llbracket \text{ns2} | \text{cs} | \text{ns1} \rrbracket^I P$   
 by (simp add: CSPInterMerge-def)  
 finally show ?thesis .  
 qed

**lemma** CSPFinalMerge-commute:

assumes  $\text{ns1} \bowtie \text{ns2}$   
 shows  $P \llbracket \text{ns1} | \text{cs} | \text{ns2} \rrbracket^F Q = Q \llbracket \text{ns2} | \text{cs} | \text{ns1} \rrbracket^F P$   
**proof** –  
 have  $P \llbracket \text{ns1} | \text{cs} | \text{ns2} \rrbracket^F Q = P \parallel_{\exists \$ref' \cdot N_C \text{ ns1 cs ns2}} Q$   
 by (simp add: CSPFinalMerge-def)  
 also have ... =  $P \parallel_{\exists \$ref' \cdot (\text{swap}_m ; N_C \text{ ns2 cs ns1})} Q$   
 by (simp add: swap-CSPInnerMerge lens-indep-sym assms)  
 also have ... =  $P \parallel_{\text{swap}_m} ; (\exists \$ref' \cdot N_C \text{ ns2 cs ns1}) Q$   
 by (simp add: seqr-exists-right)  
 also have ... =  $Q \parallel_{(\exists \$ref' \cdot N_C \text{ ns2 cs ns1})} P$   
 by (simp add: par-by-merge-commute-swap[THEN sym])  
 also have ... =  $Q \llbracket \text{ns2} | \text{cs} | \text{ns1} \rrbracket^F P$   
 by (simp add: CSPFinalMerge-def)  
 finally show ?thesis .  
 qed

Important theorem that shows the form of a parallel process

**lemma** CSPInnerMerge-form:

fixes  $P Q :: ('s, 'v) \text{ action}$   
 assumes  $\text{vwb-lens ns1 vwb-lens ns2 } P \text{ is } RR \text{ } Q \text{ is } RR$   
 shows  

$$P \parallel_{N_C \text{ ns1 cs ns2}} Q =$$

$$(\exists (\text{ref}_0, \text{ref}_1, \text{st}_0, \text{st}_1, \text{tt}_0, \text{tt}_1) \cdot$$

$$P \llbracket \langle \text{ref}_0 \rangle, \langle \text{st}_0 \rangle, \langle \rangle, \langle \text{tt}_0 \rangle / \$ref', \$st', \$tr, \$tr' \rrbracket \wedge Q \llbracket \langle \text{ref}_1 \rangle, \langle \text{st}_1 \rangle, \langle \rangle, \langle \text{tt}_1 \rangle / \$ref', \$st', \$tr, \$tr' \rrbracket$$

$$\wedge \$ref' \sqsubseteq_u ((\langle \text{ref}_0 \rangle \cup_u \langle \text{ref}_1 \rangle) \cap_u \langle \text{cs} \rangle) \cup_u ((\langle \text{ref}_0 \rangle \cap_u \langle \text{ref}_1 \rangle) - \langle \text{cs} \rangle)$$

$$\wedge \$tr \leq_u \$tr'$$

$$\wedge \& \text{tt} \in_u \langle \text{tt}_0 \rangle \star_{cs} \langle \text{tt}_1 \rangle$$

$$\wedge \langle \text{tt}_0 \rangle \upharpoonright_u \langle \text{cs} \rangle =_u \langle \text{tt}_1 \rangle \upharpoonright_u \langle \text{cs} \rangle$$

$$\wedge \$st' =_u (\$st \oplus \langle \text{st}_0 \rangle \text{ on } \& \text{ns1}) \oplus \langle \text{st}_1 \rangle \text{ on } \& \text{ns2})$$

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

**proof** –

have  $P : (\exists \{ \$ok', \$wait' \} \cdot R2(P)) = P$  (is ?P' = -)  
 by (simp add: ex-unrest ex-plus Healthy-if assms RR-implies-R2 unrest closure)  
 have  $Q : (\exists \{ \$ok', \$wait' \} \cdot R2(Q)) = Q$  (is ?Q' = -)  
 by (simp add: ex-unrest ex-plus Healthy-if assms RR-implies-R2 unrest closure)  
 from assms(1,2)  
 have  $?P' \parallel_{N_C \text{ ns1 cs ns2}} ?Q' =$   

$$(\exists (\text{ref}_0, \text{ref}_1, \text{st}_0, \text{st}_1, \text{tt}_0, \text{tt}_1) \cdot$$

$$?P' \llbracket \langle \text{ref}_0 \rangle, \langle \text{st}_0 \rangle, \langle \rangle, \langle \text{tt}_0 \rangle / \$ref', \$st', \$tr, \$tr' \rrbracket \wedge ?Q' \llbracket \langle \text{ref}_1 \rangle, \langle \text{st}_1 \rangle, \langle \rangle, \langle \text{tt}_1 \rangle / \$ref', \$st', \$tr, \$tr' \rrbracket$$

$$\wedge \$ref' \sqsubseteq_u ((\langle \text{ref}_0 \rangle \cup_u \langle \text{ref}_1 \rangle) \cap_u \langle \text{cs} \rangle) \cup_u ((\langle \text{ref}_0 \rangle \cap_u \langle \text{ref}_1 \rangle) - \langle \text{cs} \rangle)$$

$$\wedge \$tr \leq_u \$tr'$$

$$\wedge \& \text{tt} \in_u \langle \text{tt}_0 \rangle \star_{cs} \langle \text{tt}_1 \rangle$$

$$\wedge \langle \text{tt}_0 \rangle \upharpoonright_u \langle \text{cs} \rangle =_u \langle \text{tt}_1 \rangle \upharpoonright_u \langle \text{cs} \rangle$$

$\wedge \$st' =_u (\$st \oplus \langle st_0 \rangle \text{ on } \&ns1) \oplus \langle st_1 \rangle \text{ on } \&ns2)$   
**apply** (*simp add: par-by-merge-alt-def, rel-auto, blast*)  
**apply** (*rename-tac ok wait tr st ref tr' ref' ref<sub>0</sub> ref<sub>1</sub> st<sub>0</sub> st<sub>1</sub> tr<sub>0</sub> ok<sub>0</sub> tr<sub>1</sub> wait<sub>0</sub> ok<sub>1</sub> wait<sub>1</sub>*)  
**apply** (*rule-tac x=ok in exI*)  
**apply** (*rule-tac x=wait in exI*)  
**apply** (*rule-tac x=tr in exI*)  
**apply** (*rule-tac x=st in exI*)  
**apply** (*rule-tac x=ref in exI*)  
**apply** (*rule-tac x=tr @ tr<sub>0</sub> in exI*)  
**apply** (*rule-tac x=st<sub>0</sub> in exI*)  
**apply** (*rule-tac x=ref<sub>0</sub> in exI*)  
**apply** (*auto*)  
**apply** (*metis Prefix-Order.prefixI append-minus*)  
**done**  
**thus** *?thesis*  
**by** (*simp add: P Q*)  
**qed**

**lemma** *CSPInterMerge-form:*

**fixes**  $P Q :: ('\sigma, '\varphi)$  *action*

**assumes** *vwb-lens ns1 vwb-lens ns2 P is RR Q is RR*

**shows**

$P \llbracket ns1 | cs | ns2 \rrbracket^I Q =$

$(\exists (ref_0, ref_1, st_0, st_1, tt_0, tt_1) \cdot$   
 $P \llbracket \langle ref_0 \rangle, \langle st_0 \rangle, \langle \rangle, \langle tt_0 \rangle / \$ref', \$st', \$tr, \$tr' \rrbracket \wedge Q \llbracket \langle ref_1 \rangle, \langle st_1 \rangle, \langle \rangle, \langle tt_1 \rangle / \$ref', \$st', \$tr, \$tr' \rrbracket$   
 $\wedge \$ref' \subseteq_u ((\langle ref_0 \rangle \cup_u \langle ref_1 \rangle) \cap_u \langle cs \rangle) \cup_u ((\langle ref_0 \rangle \cap_u \langle ref_1 \rangle) - \langle cs \rangle)$   
 $\wedge \$tr \leq_u \$tr'$   
 $\wedge \&tt \in_u \langle tt_0 \rangle \star_{cs} \langle tt_1 \rangle$   
 $\wedge \langle tt_0 \rangle \upharpoonright_u \langle cs \rangle =_u \langle tt_1 \rangle \upharpoonright_u \langle cs \rangle)$

(**is** *?lhs = ?rhs*)

**proof** –

**have** *?lhs =*  $(\exists \$st' \cdot P \parallel_{N_C} ns1\ cs\ ns2\ Q)$

**by** (*simp add: CSPInterMerge-def par-by-merge-def seqr-exists-right*)

**also have** ... =

$(\exists \$st' \cdot$

$(\exists (ref_0, ref_1, st_0, st_1, tt_0, tt_1) \cdot$   
 $P \llbracket \langle ref_0 \rangle, \langle st_0 \rangle, \langle \rangle, \langle tt_0 \rangle / \$ref', \$st', \$tr, \$tr' \rrbracket \wedge Q \llbracket \langle ref_1 \rangle, \langle st_1 \rangle, \langle \rangle, \langle tt_1 \rangle / \$ref', \$st', \$tr, \$tr' \rrbracket$   
 $\wedge \$ref' \subseteq_u ((\langle ref_0 \rangle \cup_u \langle ref_1 \rangle) \cap_u \langle cs \rangle) \cup_u ((\langle ref_0 \rangle \cap_u \langle ref_1 \rangle) - \langle cs \rangle)$   
 $\wedge \$tr \leq_u \$tr'$   
 $\wedge \&tt \in_u \langle tt_0 \rangle \star_{cs} \langle tt_1 \rangle$   
 $\wedge \langle tt_0 \rangle \upharpoonright_u \langle cs \rangle =_u \langle tt_1 \rangle \upharpoonright_u \langle cs \rangle$   
 $\wedge \$st' =_u (\$st \oplus \langle st_0 \rangle \text{ on } \&ns1) \oplus \langle st_1 \rangle \text{ on } \&ns2))$

**by** (*simp add: CSPInnerMerge-form assms*)

**also have** ... = *?rhs*

**by** (*rel-blast*)

**finally show** *?thesis* .

**qed**

**lemma** *CSPFinalMerge-form:*

**fixes**  $P Q :: ('\sigma, '\varphi)$  *action*

**assumes** *vwb-lens ns1 vwb-lens ns2 P is RR Q is RR*  $\$ref' \# P \$ref' \# Q$

**shows**

$(P \llbracket ns1 | cs | ns2 \rrbracket^F Q) =$

$(\exists (st_0, st_1, tt_0, tt_1) \cdot$   
 $P \llbracket \langle st_0 \rangle, \langle \rangle, \langle tt_0 \rangle / \$st', \$tr, \$tr' \rrbracket \wedge Q \llbracket \langle st_1 \rangle, \langle \rangle, \langle tt_1 \rangle / \$st', \$tr, \$tr' \rrbracket$

$\wedge \$tr \leq_u \$tr'$   
 $\wedge \&tt \in_u \langle\langle tt_0 \rangle\rangle \star_{cs} \langle\langle tt_1 \rangle\rangle$   
 $\wedge \langle\langle tt_0 \rangle\rangle \downarrow_u \langle\langle cs \rangle\rangle =_u \langle\langle tt_1 \rangle\rangle \downarrow_u \langle\langle cs \rangle\rangle$   
 $\wedge \$st' =_u (\$st \oplus \langle\langle st_0 \rangle\rangle \text{ on } \&ns1) \oplus \langle\langle st_1 \rangle\rangle \text{ on } \&ns2)$   
 (is ?lhs = ?rhs)  
**proof** –  
 have ?lhs =  $(\exists \$ref' \cdot P \parallel_{N_C} ns1 \ cs \ ns2 \ Q)$   
 by (simp add: CSPFinalMerge-def par-by-merge-def seqr-exists-right)  
 also have ... =  
 $(\exists \$ref' \cdot$   
 $(\exists (ref_0, ref_1, st_0, st_1, tt_0, tt_1) \cdot$   
 $P[\langle\langle ref_0 \rangle\rangle, \langle\langle st_0 \rangle\rangle, \langle\langle tt_0 \rangle\rangle / \$ref', \$st', \$tr, \$tr'] \wedge Q[\langle\langle ref_1 \rangle\rangle, \langle\langle st_1 \rangle\rangle, \langle\langle tt_1 \rangle\rangle / \$ref', \$st', \$tr, \$tr']$   
 $\wedge \$ref' \subseteq_u ((\langle\langle ref_0 \rangle\rangle \cup_u \langle\langle ref_1 \rangle\rangle) \cap_u \langle\langle cs \rangle\rangle) \cup_u ((\langle\langle ref_0 \rangle\rangle \cap_u \langle\langle ref_1 \rangle\rangle) - \langle\langle cs \rangle\rangle)$   
 $\wedge \$tr \leq_u \$tr'$   
 $\wedge \&tt \in_u \langle\langle tt_0 \rangle\rangle \star_{cs} \langle\langle tt_1 \rangle\rangle$   
 $\wedge \langle\langle tt_0 \rangle\rangle \downarrow_u \langle\langle cs \rangle\rangle =_u \langle\langle tt_1 \rangle\rangle \downarrow_u \langle\langle cs \rangle\rangle$   
 $\wedge \$st' =_u (\$st \oplus \langle\langle st_0 \rangle\rangle \text{ on } \&ns1) \oplus \langle\langle st_1 \rangle\rangle \text{ on } \&ns2))$   
 by (simp add: CSPInnerMerge-form assms)  
 also have ... =  
 $(\exists \$ref' \cdot$   
 $(\exists (ref_0, ref_1, st_0, st_1, tt_0, tt_1) \cdot$   
 $(\exists \$ref' \cdot P)[\langle\langle ref_0 \rangle\rangle, \langle\langle st_0 \rangle\rangle, \langle\langle tt_0 \rangle\rangle / \$ref', \$st', \$tr, \$tr'] \wedge (\exists \$ref' \cdot Q)[\langle\langle ref_1 \rangle\rangle, \langle\langle st_1 \rangle\rangle, \langle\langle tt_1 \rangle\rangle / \$ref', \$st', \$tr, \$tr']$   
 $\wedge \$ref' \subseteq_u ((\langle\langle ref_0 \rangle\rangle \cup_u \langle\langle ref_1 \rangle\rangle) \cap_u \langle\langle cs \rangle\rangle) \cup_u ((\langle\langle ref_0 \rangle\rangle \cap_u \langle\langle ref_1 \rangle\rangle) - \langle\langle cs \rangle\rangle)$   
 $\wedge \$tr \leq_u \$tr'$   
 $\wedge \&tt \in_u \langle\langle tt_0 \rangle\rangle \star_{cs} \langle\langle tt_1 \rangle\rangle$   
 $\wedge \langle\langle tt_0 \rangle\rangle \downarrow_u \langle\langle cs \rangle\rangle =_u \langle\langle tt_1 \rangle\rangle \downarrow_u \langle\langle cs \rangle\rangle$   
 $\wedge \$st' =_u (\$st \oplus \langle\langle st_0 \rangle\rangle \text{ on } \&ns1) \oplus \langle\langle st_1 \rangle\rangle \text{ on } \&ns2))$   
 by (simp add: ex-unrest assms)  
 also have ... =  
 $(\exists (st_0, st_1, tt_0, tt_1) \cdot$   
 $(\exists \$ref' \cdot P)[\langle\langle st_0 \rangle\rangle, \langle\langle tt_0 \rangle\rangle / \$st', \$tr, \$tr'] \wedge (\exists \$ref' \cdot Q)[\langle\langle st_1 \rangle\rangle, \langle\langle tt_1 \rangle\rangle / \$st', \$tr, \$tr']$   
 $\wedge \$tr \leq_u \$tr'$   
 $\wedge \&tt \in_u \langle\langle tt_0 \rangle\rangle \star_{cs} \langle\langle tt_1 \rangle\rangle$   
 $\wedge \langle\langle tt_0 \rangle\rangle \downarrow_u \langle\langle cs \rangle\rangle =_u \langle\langle tt_1 \rangle\rangle \downarrow_u \langle\langle cs \rangle\rangle$   
 $\wedge \$st' =_u (\$st \oplus \langle\langle st_0 \rangle\rangle \text{ on } \&ns1) \oplus \langle\langle st_1 \rangle\rangle \text{ on } \&ns2)$   
 by (rel-blast)  
 also have ... = ?rhs  
 by (simp add: ex-unrest assms)  
 finally show ?thesis .  
**qed**

**lemma** CSPInterleave-merge:  $M_I \ ns1 \ ns2 = M_C \ ns1 \ \{\} \ ns2$

by (rel-auto)

**lemma** csp-wrR-def:

$P \ wr[ns1|cs|ns2]_C \ Q = (\neg_r ((\neg_r \ Q) ;; U0 \wedge P ;; U1 \wedge \$st_{<}' =_u \$st \wedge \$tr_{<}' =_u \$tr) ;; N_C \ ns1 \ cs \ ns2 ;; R1 \ true)$

by (rel-auto, metis+)

**lemma** csp-wrR-CRC-closed [closure]:

assumes  $P$  is CRR  $Q$  is CRR

shows  $P \ wr[ns1|cs|ns2]_C \ Q$  is CRC

**proof** –

have  $\$ref \ \sharp \ P \ wr[ns1|cs|ns2]_C \ Q$

by (simp add: csp-wrR-def rpred closure assms unrest)

thus *?thesis*  
 by (rule *CRC-intro*, *simp-all add: closure assms*)  
 qed

**lemma** *ref'-unrest-final-merge* [*unrest*]:  
 $\$ref' \# P \llbracket ns1 | cs | ns2 \rrbracket^F Q$   
 by (*rel-auto*)

**lemma** *inter-merge-CDC-closed* [*closure*]:  
 $P \llbracket ns1 | cs | ns2 \rrbracket^I Q$  is CDC  
 using *le-less-trans* by (*rel-blast*)

**lemma** *merge-csp-do-left*:  
 assumes *vwb-lens ns1 vwb-lens ns2 ns1  $\bowtie$  ns2 P is RR*  
 shows  $\Phi(s_0, \sigma_0, t_0) \parallel_{N_C} ns1 \ cs \ ns2 \ P =$

$(\exists (ref_1, st_1, tt_1) \cdot$   
 $[s_0]_{S<} \wedge$   
 $[\$ref' \mapsto_s \langle ref_1 \rangle, \$st' \mapsto_s \langle st_1 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_1 \rangle] \dagger P \wedge$   
 $\$ref' \subseteq_u \langle cs \rangle \cup_u (\langle ref_1 \rangle - \langle cs \rangle) \wedge$   
 $[\langle trace \rangle \in_u t_0 \star_{cs} \langle tt_1 \rangle \wedge t_0 \upharpoonright_u \langle cs \rangle =_u \langle tt_1 \rangle \upharpoonright_u \langle cs \rangle]_t \wedge$   
 $\$st' =_u \$st \oplus \langle \sigma_0 \rangle (\$st)_a \text{ on } \&ns1 \oplus \langle st_1 \rangle \text{ on } \&ns2)$

(is *?lhs = ?rhs*)

**proof** –

have *?lhs* =

$(\exists (ref_0, ref_1, st_0, st_1, tt_0, tt_1) \cdot$   
 $[\$ref' \mapsto_s \langle ref_0 \rangle, \$st' \mapsto_s \langle st_0 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_0 \rangle] \dagger \Phi(s_0, \sigma_0, t_0) \wedge$   
 $[\$ref' \mapsto_s \langle ref_1 \rangle, \$st' \mapsto_s \langle st_1 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_1 \rangle] \dagger P \wedge$   
 $\$ref' \subseteq_u (\langle ref_0 \rangle \cup_u \langle ref_1 \rangle) \cap_u \langle cs \rangle \cup_u (\langle ref_0 \rangle \cap_u \langle ref_1 \rangle - \langle cs \rangle) \wedge$   
 $\$tr \leq_u \$tr' \wedge$   
 $\&tt \in_u \langle tt_0 \rangle \star_{cs} \langle tt_1 \rangle \wedge \langle tt_0 \rangle \upharpoonright_u \langle cs \rangle =_u \langle tt_1 \rangle \upharpoonright_u \langle cs \rangle \wedge \$st' =_u \$st \oplus \langle st_0 \rangle \text{ on } \&ns1$   
 $\oplus \langle st_1 \rangle \text{ on } \&ns2)$

by (*simp add: CSPInnerMerge-form assms closure*)

also have ... =

$(\exists (ref_1, st_1, tt_1) \cdot$   
 $[s_0]_{S<} \wedge$   
 $[\$ref' \mapsto_s \langle ref_1 \rangle, \$st' \mapsto_s \langle st_1 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_1 \rangle] \dagger P \wedge$   
 $\$ref' \subseteq_u \langle cs \rangle \cup_u (\langle ref_1 \rangle - \langle cs \rangle) \wedge$   
 $[\langle trace \rangle \in_u t_0 \star_{cs} \langle tt_1 \rangle \wedge t_0 \upharpoonright_u \langle cs \rangle =_u \langle tt_1 \rangle \upharpoonright_u \langle cs \rangle]_t \wedge$   
 $\$st' =_u \$st \oplus \langle \sigma_0 \rangle (\$st)_a \text{ on } \&ns1 \oplus \langle st_1 \rangle \text{ on } \&ns2)$

by (*rel-blast*)

finally show *?thesis* .

qed

**lemma** *merge-csp-do-right*:

assumes *vwb-lens ns1 vwb-lens ns2 ns1  $\bowtie$  ns2 P is RR*

shows  $P \parallel_{N_C} ns1 \ cs \ ns2 \ \Phi(s_1, \sigma_1, t_1) =$

$(\exists (ref_0, st_0, tt_0) \cdot$   
 $[\$ref' \mapsto_s \langle ref_0 \rangle, \$st' \mapsto_s \langle st_0 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_0 \rangle] \dagger P \wedge$   
 $[s_1]_{S<} \wedge$   
 $\$ref' \subseteq_u \langle cs \rangle \cup_u (\langle ref_0 \rangle - \langle cs \rangle) \wedge$   
 $[\langle trace \rangle \in_u \langle tt_0 \rangle \star_{cs} t_1 \wedge \langle tt_0 \rangle \upharpoonright_u \langle cs \rangle =_u t_1 \upharpoonright_u \langle cs \rangle]_t \wedge$   
 $\$st' =_u \$st \oplus \langle st_0 \rangle \text{ on } \&ns1 \oplus \langle \sigma_1 \rangle (\$st)_a \text{ on } \&ns2)$

(is *?lhs = ?rhs*)

**proof** –

have *?lhs* =

$(\exists (ref_0, ref_1, st_0, st_1, tt_0, tt_1) \cdot$   
 $[\$ref' \mapsto_s \langle\langle ref_0 \rangle\rangle, \$st' \mapsto_s \langle\langle st_0 \rangle\rangle, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \langle\langle tt_0 \rangle\rangle] \dagger P \wedge$   
 $[\$ref' \mapsto_s \langle\langle ref_1 \rangle\rangle, \$st' \mapsto_s \langle\langle st_1 \rangle\rangle, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \langle\langle tt_1 \rangle\rangle] \dagger \Phi(s_1, \sigma_1, t_1) \wedge$   
 $\$ref' \subseteq_u (\langle\langle ref_0 \rangle\rangle \cup_u \langle\langle ref_1 \rangle\rangle) \cap_u \langle\langle cs \rangle\rangle \cup_u (\langle\langle ref_0 \rangle\rangle \cap_u \langle\langle ref_1 \rangle\rangle - \langle\langle cs \rangle\rangle) \wedge$   
 $\$tr \leq_u \$tr' \wedge$   
 $\&tt \in_u \langle\langle tt_0 \rangle\rangle \star_{cs} \langle\langle tt_1 \rangle\rangle \wedge \langle\langle tt_0 \rangle\rangle \upharpoonright_u \langle\langle cs \rangle\rangle =_u \langle\langle tt_1 \rangle\rangle \upharpoonright_u \langle\langle cs \rangle\rangle \wedge \$st' =_u \$st \oplus \langle\langle st_0 \rangle\rangle$  on  
 $\&ns1 \oplus \langle\langle st_1 \rangle\rangle$  on  $\&ns2$ )  
**by** (*simp add: CSPInnerMerge-form assms closure*)  
**also have** ... = ?rhs  
**by** (*rel-blast*)  
**finally show** ?thesis .  
**qed**

The result of merge two terminated stateful traces is to (1) require both state preconditions hold, (2) merge the traces using, and (3) merge the state using a parallel assignment.

**lemma** *FinalMerge-csp-do-left:*

**assumes** *vwb-lens ns1 vwb-lens ns2 ns1  $\bowtie$  ns2 P is RR  $\$ref' \nmid P$*

**shows**  $\Phi(s_0, \sigma_0, t_0) \llbracket ns1 | cs | ns2 \rrbracket^F P =$

$(\exists (st_1, t_1) \cdot$   
 $[s_0]_{S<} \wedge$   
 $[\$st' \mapsto_s \langle\langle st_1 \rangle\rangle, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \langle\langle t_1 \rangle\rangle] \dagger P \wedge$   
 $[\langle\langle trace \rangle\rangle \in_u t_0 \star_{cs} \langle\langle t_1 \rangle\rangle \wedge t_0 \upharpoonright_u \langle\langle cs \rangle\rangle =_u \langle\langle t_1 \rangle\rangle \upharpoonright_u \langle\langle cs \rangle\rangle]_t \wedge$   
 $\$st' =_u \$st \oplus \langle\langle \sigma_0 \rangle\rangle (\$st)_a$  on  $\&ns1 \oplus \langle\langle st_1 \rangle\rangle$  on  $\&ns2$ )

(**is** ?lhs = ?rhs)

**proof** –

**have** ?lhs =

$(\exists (st_0, st_1, tt_0, tt_1) \cdot$   
 $[\$st' \mapsto_s \langle\langle st_0 \rangle\rangle, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \langle\langle tt_0 \rangle\rangle] \dagger \Phi(s_0, \sigma_0, t_0) \wedge$   
 $[\$st' \mapsto_s \langle\langle st_1 \rangle\rangle, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \langle\langle tt_1 \rangle\rangle] \dagger RR(\exists \$ref' \cdot P) \wedge$   
 $\$tr \leq_u \$tr' \wedge \&tt \in_u \langle\langle tt_0 \rangle\rangle \star_{cs} \langle\langle tt_1 \rangle\rangle \wedge \langle\langle tt_0 \rangle\rangle \upharpoonright_u \langle\langle cs \rangle\rangle =_u \langle\langle tt_1 \rangle\rangle \upharpoonright_u \langle\langle cs \rangle\rangle \wedge$   
 $\$st' =_u \$st \oplus \langle\langle st_0 \rangle\rangle$  on  $\&ns1 \oplus \langle\langle st_1 \rangle\rangle$  on  $\&ns2$ )

**by** (*simp add: CSPFinalMerge-form ex-unrest Healthy-if unrest closure assms*)

**also have** ... =

$(\exists (st_1, tt_1) \cdot$   
 $[s_0]_{S<} \wedge$   
 $[\$st' \mapsto_s \langle\langle st_1 \rangle\rangle, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \langle\langle tt_1 \rangle\rangle] \dagger RR(\exists \$ref' \cdot P) \wedge$   
 $[\langle\langle trace \rangle\rangle \in_u t_0 \star_{cs} \langle\langle tt_1 \rangle\rangle \wedge t_0 \upharpoonright_u \langle\langle cs \rangle\rangle =_u \langle\langle tt_1 \rangle\rangle \upharpoonright_u \langle\langle cs \rangle\rangle]_t \wedge$   
 $\$st' =_u \$st \oplus \langle\langle \sigma_0 \rangle\rangle (\$st)_a$  on  $\&ns1 \oplus \langle\langle st_1 \rangle\rangle$  on  $\&ns2$ )

**by** (*rel-blast*)

**also have** ... =

$(\exists (st_1, t_1) \cdot$   
 $[s_0]_{S<} \wedge$   
 $[\$st' \mapsto_s \langle\langle st_1 \rangle\rangle, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \langle\langle t_1 \rangle\rangle] \dagger P \wedge$   
 $[\langle\langle trace \rangle\rangle \in_u t_0 \star_{cs} \langle\langle t_1 \rangle\rangle \wedge t_0 \upharpoonright_u \langle\langle cs \rangle\rangle =_u \langle\langle t_1 \rangle\rangle \upharpoonright_u \langle\langle cs \rangle\rangle]_t \wedge$   
 $\$st' =_u \$st \oplus \langle\langle \sigma_0 \rangle\rangle (\$st)_a$  on  $\&ns1 \oplus \langle\langle st_1 \rangle\rangle$  on  $\&ns2$ )

**by** (*simp add: ex-unrest Healthy-if unrest closure assms*)

**finally show** ?thesis .

**qed**

**lemma** *FinalMerge-csp-do-right:*

**assumes** *vwb-lens ns1 vwb-lens ns2 ns1  $\bowtie$  ns2 P is RR  $\$ref' \nmid P$*

**shows**  $P \llbracket ns1 | cs | ns2 \rrbracket^F \Phi(s_1, \sigma_1, t_1) =$

$(\exists (st_0, t_0) \cdot$   
 $[\$st' \mapsto_s \langle\langle st_0 \rangle\rangle, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \langle\langle t_0 \rangle\rangle] \dagger P \wedge$   
 $[s_1]_{S<} \wedge$

$$[\llbracket \text{trace} \rrbracket \in_u \llbracket t_0 \rrbracket \star_{cs} t_1 \wedge \llbracket t_0 \rrbracket \downarrow_u \llbracket cs \rrbracket =_u t_1 \downarrow_u \llbracket cs \rrbracket]_t \wedge$$

$$\$st' =_u \$st \oplus \llbracket st_0 \rrbracket \text{ on } \&ns1 \oplus \llbracket \sigma_1 \rrbracket (\$st)_a \text{ on } \&ns2)$$
 (is ?lhs = ?rhs)

**proof** –  
 have  $P \llbracket ns1|cs|ns2 \rrbracket^F \Phi(s_1, \sigma_1, t_1) = \Phi(s_1, \sigma_1, t_1) \llbracket ns2|cs|ns1 \rrbracket^F P$   
 by (simp add: assms CSPFinalMerge-commute)  
 also have ... = ?rhs  
 apply (simp add: FinalMerge-csp-do-left assms lens-indep-sym conj-comm)  
 apply (rel-auto)  
 using assms(3) lens-indep.lens-put-comm tr-par-sym apply fastforce+  
 done  
 finally show ?thesis .  
**qed**

**lemma** *FinalMerge-csp-do*:  
 assumes  $vwb\text{-}lens\ ns1\ vwb\text{-}lens\ ns2\ ns1 \bowtie ns2$   
 shows  $\Phi(s_1, \sigma_1, t_1) \llbracket ns1|cs|ns2 \rrbracket^F \Phi(s_2, \sigma_2, t_2) =$   

$$([s_1 \wedge s_2]_{S<} \wedge [\llbracket \text{trace} \rrbracket \in_u t_1 \star_{cs} t_2 \wedge t_1 \downarrow_u \llbracket cs \rrbracket =_u t_2 \downarrow_u \llbracket cs \rrbracket]_t \wedge [\langle \sigma_1 [\&ns1|\&ns2]_s \sigma_2 \rangle_a]_{S'})$$
 (is ?lhs = ?rhs)

**proof** –  
 have ?lhs =  

$$(\exists (st_0, st_1, tt_0, tt_1) \cdot$$

$$[\$st' \mapsto_s \llbracket st_0 \rrbracket, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \llbracket tt_0 \rrbracket] \dagger \Phi(s_1, \sigma_1, t_1) \wedge$$

$$[\$st' \mapsto_s \llbracket st_1 \rrbracket, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \llbracket tt_1 \rrbracket] \dagger \Phi(s_2, \sigma_2, t_2) \wedge$$

$$\$tr \leq_u \$tr' \wedge \llbracket tt_0 \rrbracket \star_{cs} \llbracket tt_1 \rrbracket \wedge \llbracket tt_0 \rrbracket \downarrow_u \llbracket cs \rrbracket =_u \llbracket tt_1 \rrbracket \downarrow_u \llbracket cs \rrbracket \wedge$$

$$\$st' =_u \$st \oplus \llbracket st_0 \rrbracket \text{ on } \&ns1 \oplus \llbracket st_1 \rrbracket \text{ on } \&ns2)$$
 by (simp add: CSPFinalMerge-form unrest closure assms)  
 also have ... =  

$$([s_1 \wedge s_2]_{S<} \wedge [\llbracket \text{trace} \rrbracket \in_u t_1 \star_{cs} t_2 \wedge t_1 \downarrow_u \llbracket cs \rrbracket =_u t_2 \downarrow_u \llbracket cs \rrbracket]_t \wedge [\langle \sigma_1 [\&ns1|\&ns2]_s \sigma_2 \rangle_a]_{S'})$$
 by (rel-auto)  
 finally show ?thesis .  
**qed**

**lemma** *FinalMerge-csp-do' [rpred]*:  
 assumes  $vwb\text{-}lens\ ns1\ vwb\text{-}lens\ ns2\ ns1 \bowtie ns2$   
 shows  $\Phi(s_1, \sigma_1, t_1) \llbracket ns1|cs|ns2 \rrbracket^F \Phi(s_2, \sigma_2, t_2) =$   

$$(\exists \text{ trace} \in [t_1 \star_{cs} t_2]_{S<} \cdot$$

$$\Phi(s_1 \wedge s_2 \wedge t_1 \downarrow_u \llbracket cs \rrbracket =_u t_2 \downarrow_u \llbracket cs \rrbracket, \sigma_1 [\&ns1|\&ns2]_s \sigma_2, \llbracket \text{trace} \rrbracket))$$
 by (simp add: FinalMerge-csp-do assms, rel-auto)

**lemma** *CSPFinalMerge-UINF-ind-left [rpred]*:  
 $(\bigcap i \cdot P(i)) \llbracket ns1|cs|ns2 \rrbracket^F Q = (\bigcap i \cdot P(i) \llbracket ns1|cs|ns2 \rrbracket^F Q)$   
 by (simp add: CSPFinalMerge-def par-by-merge-USUP-ind-left)

**lemma** *CSPFinalMerge-UINF-ind-right [rpred]*:  
 $P \llbracket ns1|cs|ns2 \rrbracket^F (\bigcap i \cdot Q(i)) = (\bigcap i \cdot P \llbracket ns1|cs|ns2 \rrbracket^F Q(i))$   
 by (simp add: CSPFinalMerge-def par-by-merge-USUP-ind-right)

**lemma** *InterMerge-csp-enable*:  
 assumes  $vwb\text{-}lens\ ns1\ vwb\text{-}lens\ ns2\ ns1 \bowtie ns2$   
 shows  $\mathcal{E}(s_1, t_1, E_1) \llbracket ns1|cs|ns2 \rrbracket^I \mathcal{E}(s_2, t_2, E_2) =$   

$$([s_1 \wedge s_2]_{S<} \wedge$$

$(\forall e \in [(E_1 \cap_u E_2 \cap_u \langle cs \rangle) \cup_u ((E_1 \cup_u E_2) - \langle cs \rangle)]_{S<} \cdot \langle e \rangle \notin_u \$ref') \wedge$   
 $[\langle trace \rangle \in_u t_1 \star_{cs} t_2 \wedge t_1 \upharpoonright_u \langle cs \rangle =_u t_2 \upharpoonright_u \langle cs \rangle]_t$   
 (is ?lhs = ?rhs)  
**proof** –  
 have ?lhs =  
 $(\exists (ref_0, ref_1, st_0, st_1, tt_0, tt_1) \cdot$   
 $[\$ref' \mapsto_s \langle ref_0 \rangle, \$st' \mapsto_s \langle st_0 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_0 \rangle] \dagger \mathcal{E}(s_1, t_1, E_1) \wedge$   
 $[\$ref' \mapsto_s \langle ref_1 \rangle, \$st' \mapsto_s \langle st_1 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_1 \rangle] \dagger \mathcal{E}(s_2, t_2, E_2) \wedge$   
 $\$ref' \subseteq_u (\langle ref_0 \rangle \cup_u \langle ref_1 \rangle) \cap_u \langle cs \rangle \cup_u (\langle ref_0 \rangle \cap_u \langle ref_1 \rangle - \langle cs \rangle) \wedge$   
 $\$tr \leq_u \$tr' \wedge \&tt \in_u \langle tt_0 \rangle \star_{cs} \langle tt_1 \rangle \wedge \langle tt_0 \rangle \upharpoonright_u \langle cs \rangle =_u \langle tt_1 \rangle \upharpoonright_u \langle cs \rangle)$   
 by (simp add: CSPInterMerge-form unrest closure assms)  
 also have ... =  
 $(\exists (ref_0, ref_1, tt_0, tt_1) \cdot$   
 $[\$ref' \mapsto_s \langle ref_0 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_0 \rangle] \dagger \mathcal{E}(s_1, t_1, E_1) \wedge$   
 $[\$ref' \mapsto_s \langle ref_1 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_1 \rangle] \dagger \mathcal{E}(s_2, t_2, E_2) \wedge$   
 $\$ref' \subseteq_u (\langle ref_0 \rangle \cup_u \langle ref_1 \rangle) \cap_u \langle cs \rangle \cup_u (\langle ref_0 \rangle \cap_u \langle ref_1 \rangle - \langle cs \rangle) \wedge$   
 $\$tr \leq_u \$tr' \wedge \&tt \in_u \langle tt_0 \rangle \star_{cs} \langle tt_1 \rangle \wedge \langle tt_0 \rangle \upharpoonright_u \langle cs \rangle =_u \langle tt_1 \rangle \upharpoonright_u \langle cs \rangle)$   
 by (rel-auto)  
 also have ... =  
 $([s_1 \wedge s_2]_{S<} \wedge$   
 $(\forall e \in [(E_1 \cap_u E_2 \cap_u \langle cs \rangle) \cup_u ((E_1 \cup_u E_2) - \langle cs \rangle)]_{S<} \cdot \langle e \rangle \notin_u \$ref') \wedge$   
 $[\langle trace \rangle \in_u t_1 \star_{cs} t_2 \wedge t_1 \upharpoonright_u \langle cs \rangle =_u t_2 \upharpoonright_u \langle cs \rangle]_t$   
 $)$   
 apply (rel-auto)  
 apply (rename-tac tr st tr' ref')  
 apply (rule-tac x=–  $\llbracket E_1 \rrbracket_e$  st in exI)  
 apply (simp)  
 apply (rule-tac x=–  $\llbracket E_2 \rrbracket_e$  st in exI)  
 apply (auto)  
 done  
 finally show ?thesis .  
 qed

**lemma** *InterMerge-csp-enable' [rpred]:*  
 assumes vwb-lens ns1 vwb-lens ns2 ns1  $\bowtie$  ns2  
 shows  $\mathcal{E}(s_1, t_1, E_1) \llbracket ns1 | cs | ns2 \rrbracket^I \mathcal{E}(s_2, t_2, E_2) =$   
 $(\exists trace \in [t_1 \star_{cs} t_2]_{S<} \cdot$   
 $\mathcal{E}(s_1 \wedge s_2 \wedge t_1 \upharpoonright_u \langle cs \rangle =_u t_2 \upharpoonright_u \langle cs \rangle$   
 $, \langle trace \rangle$   
 $, (E_1 \cap_u E_2 \cap_u \langle cs \rangle) \cup_u ((E_1 \cup_u E_2) - \langle cs \rangle)))$   
 by (simp add: InterMerge-csp-enable assms, rel-auto)

**lemma** *InterMerge-csp-enable-csp-do:*  
 assumes vwb-lens ns1 vwb-lens ns2 ns1  $\bowtie$  ns2  
 shows  $\mathcal{E}(s_1, t_1, E_1) \llbracket ns1 | cs | ns2 \rrbracket^I \Phi(s_2, \sigma_2, t_2) =$   
 $([s_1 \wedge s_2]_{S<} \wedge (\forall e \in [(E_1 - \langle cs \rangle)]_{S<} \cdot \langle e \rangle \notin_u \$ref') \wedge$   
 $[\langle trace \rangle \in_u t_1 \star_{cs} t_2 \wedge t_1 \upharpoonright_u \langle cs \rangle =_u t_2 \upharpoonright_u \langle cs \rangle]_t)$   
 (is ?lhs = ?rhs)

**proof** –  
 have ?lhs =  
 $(\exists (ref_0, ref_1, st_0, st_1, tt_0, tt_1) \cdot$   
 $[\$ref' \mapsto_s \langle ref_0 \rangle, \$st' \mapsto_s \langle st_0 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_0 \rangle] \dagger \mathcal{E}(s_1, t_1, E_1) \wedge$   
 $[\$ref' \mapsto_s \langle ref_1 \rangle, \$st' \mapsto_s \langle st_1 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_1 \rangle] \dagger \Phi(s_2, \sigma_2, t_2) \wedge$   
 $\$ref' \subseteq_u (\langle ref_0 \rangle \cup_u \langle ref_1 \rangle) \cap_u \langle cs \rangle \cup_u (\langle ref_0 \rangle \cap_u \langle ref_1 \rangle - \langle cs \rangle) \wedge$   
 $\$tr \leq_u \$tr' \wedge \&tt \in_u \langle tt_0 \rangle \star_{cs} \langle tt_1 \rangle \wedge \langle tt_0 \rangle \upharpoonright_u \langle cs \rangle =_u \langle tt_1 \rangle \upharpoonright_u \langle cs \rangle)$

by (simp add: CSPInterMerge-form unrest closure assms)  
 also have ... =  

$$(\exists (ref_0, ref_1, tt_0) \cdot$$

$$[\$ref' \mapsto_s \langle\langle ref_0 \rangle\rangle, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \langle\langle tt_0 \rangle\rangle] \dagger \mathcal{E}(s_1, t_1, E_1) \wedge$$

$$[s_2]_{S<} \wedge$$

$$\$ref' \subseteq_u (\langle\langle ref_0 \rangle\rangle \cup_u \langle\langle ref_1 \rangle\rangle) \cap_u \langle\langle cs \rangle\rangle \cup_u (\langle\langle ref_0 \rangle\rangle \cap_u \langle\langle ref_1 \rangle\rangle - \langle\langle cs \rangle\rangle) \wedge$$

$$[\langle\langle trace \rangle\rangle \in_u t_1 \star_{cs} t_2 \wedge t_1 \downarrow_u \langle\langle cs \rangle\rangle =_u t_2 \downarrow_u \langle\langle cs \rangle\rangle]_t)$$
 by (rel-auto)  
 also have ... =  $([s_1 \wedge s_2]_{S<} \wedge (\forall e \in [(E_1 - \langle\langle cs \rangle\rangle)]_{S<} \cdot \langle\langle e \rangle\rangle \notin_u \$ref')) \wedge$   

$$[\langle\langle trace \rangle\rangle \in_u t_1 \star_{cs} t_2 \wedge t_1 \downarrow_u \langle\langle cs \rangle\rangle =_u t_2 \downarrow_u \langle\langle cs \rangle\rangle]_t)$$
 by (rel-auto)  
 finally show ?thesis .  
 qed

**lemma** *InterMerge-csp-enable-csp-do' [rpred]*:  
 assumes *vwb-lens ns1 vwb-lens ns2 ns1  $\bowtie$  ns2*  
 shows  $\mathcal{E}(s_1, t_1, E_1) \llbracket ns1 | cs | ns2 \rrbracket^I \Phi(s_2, \sigma_2, t_2) =$   

$$(\bigcap trace \mid \langle\langle trace \rangle\rangle \in_u [t_1 \star_{cs} t_2]_{S<} \cdot$$

$$\mathcal{E}(s_1 \wedge s_2 \wedge t_1 \downarrow_u \langle\langle cs \rangle\rangle =_u t_2 \downarrow_u \langle\langle cs \rangle\rangle, \langle\langle trace \rangle\rangle, E_1 - \langle\langle cs \rangle\rangle))$$
 by (simp add: InterMerge-csp-enable-csp-do assms, rel-auto)

**lemma** *InterMerge-csp-do-csp-enable*:  
 assumes *vwb-lens ns1 vwb-lens ns2 ns1  $\bowtie$  ns2*  
 shows  $\Phi(s_1, \sigma_1, t_1) \llbracket ns1 | cs | ns2 \rrbracket^I \mathcal{E}(s_2, t_2, E_2) =$   

$$([s_1 \wedge s_2]_{S<} \wedge (\forall e \in [(E_2 - \langle\langle cs \rangle\rangle)]_{S<} \cdot \langle\langle e \rangle\rangle \notin_u \$ref')) \wedge$$

$$[\langle\langle trace \rangle\rangle \in_u t_1 \star_{cs} t_2 \wedge t_1 \downarrow_u \langle\langle cs \rangle\rangle =_u t_2 \downarrow_u \langle\langle cs \rangle\rangle]_t)$$
 (is ?lhs = ?rhs)

**proof** –  
 have  $\Phi(s_1, \sigma_1, t_1) \llbracket ns1 | cs | ns2 \rrbracket^I \mathcal{E}(s_2, t_2, E_2) = \mathcal{E}(s_2, t_2, E_2) \llbracket ns2 | cs | ns1 \rrbracket^I \Phi(s_1, \sigma_1, t_1)$   
 by (simp add: CSPInterMerge-commute assms)  
 also have ... = ?rhs  
 by (simp add: InterMerge-csp-enable-csp-do assms lens-indep-sym trace-merge-commute conj-comm eq-upred-sym)  
 finally show ?thesis .  
 qed

**lemma** *InterMerge-csp-do-csp-enable' [rpred]*:  
 assumes *vwb-lens ns1 vwb-lens ns2 ns1  $\bowtie$  ns2*  
 shows  $\Phi(s_1, \sigma_1, t_1) \llbracket ns1 | cs | ns2 \rrbracket^I \mathcal{E}(s_2, t_2, E_2) =$   

$$(\bigcap trace \mid \langle\langle trace \rangle\rangle \in_u [t_1 \star_{cs} t_2]_{S<} \cdot$$

$$\mathcal{E}(s_1 \wedge s_2 \wedge t_1 \downarrow_u \langle\langle cs \rangle\rangle =_u t_2 \downarrow_u \langle\langle cs \rangle\rangle, \langle\langle trace \rangle\rangle, E_2 - \langle\langle cs \rangle\rangle))$$
 by (simp add: InterMerge-csp-do-csp-enable assms, rel-auto)

**lemma** *CSPInterMerge-or-left [rpred]*:  
 $(P \vee Q) \llbracket ns1 | cs | ns2 \rrbracket^I R = (P \llbracket ns1 | cs | ns2 \rrbracket^I R \vee Q \llbracket ns1 | cs | ns2 \rrbracket^I R)$   
 by (simp add: CSPInterMerge-def par-by-merge-or-left)

**lemma** *CSPInterMerge-or-right [rpred]*:  
 $P \llbracket ns1 | cs | ns2 \rrbracket^I (Q \vee R) = (P \llbracket ns1 | cs | ns2 \rrbracket^I Q \vee P \llbracket ns1 | cs | ns2 \rrbracket^I R)$   
 by (simp add: CSPInterMerge-def par-by-merge-or-right)

**lemma** *CSPFinalMerge-or-left [rpred]*:  
 $(P \vee Q) \llbracket ns1 | cs | ns2 \rrbracket^F R = (P \llbracket ns1 | cs | ns2 \rrbracket^F R \vee Q \llbracket ns1 | cs | ns2 \rrbracket^F R)$   
 by (simp add: CSPFinalMerge-def par-by-merge-or-left)



**lemma** *CSPFinalMerge-or-right* [rpred]:

$P \llbracket ns1 | cs | ns2 \rrbracket^F (Q \vee R) = (P \llbracket ns1 | cs | ns2 \rrbracket^F Q \vee P \llbracket ns1 | cs | ns2 \rrbracket^F R)$   
 by (simp add: CSPFinalMerge-def par-by-merge-or-right)

**lemma** *CSPInterMerge-UINF-ind-left* [rpred]:

$(\bigcap i \cdot P(i)) \llbracket ns1 | cs | ns2 \rrbracket^I Q = (\bigcap i \cdot P(i) \llbracket ns1 | cs | ns2 \rrbracket^I Q)$   
 by (simp add: CSPInterMerge-def par-by-merge-USUP-ind-left)

**lemma** *CSPInterMerge-UINF-ind-right* [rpred]:

$P \llbracket ns1 | cs | ns2 \rrbracket^I (\bigcap i \cdot Q(i)) = (\bigcap i \cdot P \llbracket ns1 | cs | ns2 \rrbracket^I Q(i))$   
 by (simp add: CSPInterMerge-def par-by-merge-USUP-ind-right)

**lemma** *par-by-merge-seq-remove*:  $(P \parallel_M ; R \ Q) = (P \parallel_M Q) ; R$

by (simp add: par-by-merge-seq-add[THEN sym])

**lemma** *merge-csp-do-right*:

**assumes** *vwb-lens ns1 vwb-lens ns2 ns1  $\bowtie$  ns2 P is RC*

**shows**  $\Phi(s_1, \sigma_1, t_1) \text{ wr}[ns1 | cs | ns2]_C P = \text{undefined}$

(is ?lhs = ?rhs)

**proof** –

**have** ?lhs =

$(\neg_r (\exists (ref_0, st_0, tt_0) \cdot$   
 $[\$ref' \mapsto_s \ll ref_0 \gg, \$st' \mapsto_s \ll st_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll tt_0 \gg] \dagger (\neg_r RC(P)) \wedge$   
 $[s_1]_{S<} \wedge$   
 $\$ref' \subseteq_u \ll cs \gg \cup_u (\ll ref_0 \gg - \ll cs \gg) \wedge$   
 $\ll trace \gg \in_u \ll tt_0 \gg \star_{cs} t_1 \wedge \ll tt_0 \gg \upharpoonright_u \ll cs \gg =_u t_1 \upharpoonright_u \ll cs \gg]_t \wedge$   
 $\$st' =_u \$st \oplus \ll st_0 \gg \text{ on } \&ns1 \oplus \ll \sigma_1 \gg (\$st)_a \text{ on } \&ns2) ; R1 \text{ true})$

by (simp add: wrR-def par-by-merge-seq-remove merge-csp-do-right closure assms Healthy-if rpred)

**also have** ... =

$(\neg_r (\exists (ref_0, st_0, tt_0) \cdot$   
 $[\$ref' \mapsto_s \ll ref_0 \gg, \$st' \mapsto_s \ll st_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll tt_0 \gg] \dagger (\neg_r RC(P)) \wedge$   
 $[s_1]_{S<} \wedge$   
 $\$ref' \subseteq_u \ll cs \gg \cup_u (\ll ref_0 \gg - \ll cs \gg) \wedge$   
 $\ll trace \gg \in_u \ll tt_0 \gg \star_{cs} t_1 \wedge \ll tt_0 \gg \upharpoonright_u \ll cs \gg =_u t_1 \upharpoonright_u \ll cs \gg]_t ; \text{true}_r \wedge$   
 $\$st' =_u \$st \oplus \ll st_0 \gg \text{ on } \&ns1 \oplus \ll \sigma_1 \gg (\$st)_a \text{ on } \&ns2))$

**apply** (rel-auto)

**oops**

## 4.2 Parallel operator

**syntax**

*-par-circus* :: *logic*  $\Rightarrow$  *salpha*  $\Rightarrow$  *logic*  $\Rightarrow$  *salpha*  $\Rightarrow$  *logic*  $\Rightarrow$  *logic* (-  $\llbracket - \parallel - \rrbracket$  - [75,0,0,0,76] 76)

*-par-csp* :: *logic*  $\Rightarrow$  *logic*  $\Rightarrow$  *logic*  $\Rightarrow$  *logic* (-  $\llbracket - \rrbracket_C$  - [75,0,76] 76)

*-inter-circus* :: *logic*  $\Rightarrow$  *salpha*  $\Rightarrow$  *salpha*  $\Rightarrow$  *logic*  $\Rightarrow$  *logic* (-  $\llbracket - \parallel - \rrbracket$  - [75,0,0,76] 76)

**translations**

*-par-circus*  $P \ ns1 \ cs \ ns2 \ Q == P \parallel_{M_C} ns1 \ cs \ ns2 \ Q$

*-par-csp*  $P \ cs \ Q == \text{-par-circus } P \ 0_L \ cs \ 0_L \ Q$

*-inter-circus*  $P \ ns1 \ ns2 \ Q == \text{-par-circus } P \ ns1 \ \{\} \ ns2 \ Q$

**abbreviation** *InterleaveCSP* :: (*'s*, *'e*) *action*  $\Rightarrow$  (*'s*, *'e*) *action*  $\Rightarrow$  (*'s*, *'e*) *action* (**infixr**  $\parallel$  75)

**where**  $P \parallel Q \equiv P \llbracket \emptyset \parallel \emptyset \rrbracket Q$

**abbreviation** *SynchroniseCSP* :: (*'s*, *'e*) *action*  $\Rightarrow$  (*'s*, *'e*) *action*  $\Rightarrow$  (*'s*, *'e*) *action* (**infixr**  $\parallel$  75)

where  $P \parallel Q \equiv P \llbracket UNIV \rrbracket_C Q$

**definition**  $CSP5 :: ' \varphi \text{ process} \Rightarrow ' \varphi \text{ process}$  **where**

$[upred-defs]: CSP5(P) = (P \parallel Skip)$

**definition**  $C2 :: (' \sigma, ' \varphi) \text{ action} \Rightarrow (' \sigma, ' \varphi) \text{ action}$  **where**

$[upred-defs]: C2(P) = (P \llbracket \Sigma \parallel \{ \} \parallel \emptyset \rrbracket Skip)$

**definition**  $CACT :: (' s, ' e) \text{ action} \Rightarrow (' s, ' e) \text{ action}$  **where**

$[upred-defs]: CACT(P) = C2(NCSP(P))$

**abbreviation**  $CPROC :: ' e \text{ process} \Rightarrow ' e \text{ process}$  **where**

$CPROC(P) \equiv CACT(P)$

**lemma** *Skip-right-form*:

**assumes**  $P_1 \text{ is } RC \ P_2 \text{ is } RR \ P_3 \text{ is } RR \ \$st' \# P_2$

**shows**  $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) ;; Skip = \mathbf{R}_s(P_1 \vdash P_2 \diamond (\exists \$ref' \cdot P_3))$

**proof** –

**have**  $1:RR(P_3) ;; \Phi(true, id, \langle \rangle) = (\exists \$ref' \cdot RR(P_3))$

**by** (*rel-auto*)

**show** *?thesis*

**by** (*rdes-simp cls: assms, metis 1 Healthy-if assms(3)*)

**qed**

**lemma** *ParCSP-rdes-def* [*rdes-def*]:

**fixes**  $P_1 :: (' s, ' e) \text{ action}$

**assumes**  $P_1 \text{ is } CRC \ Q_1 \text{ is } CRC \ P_2 \text{ is } CRR \ Q_2 \text{ is } CRR \ P_3 \text{ is } CRR \ Q_3 \text{ is } CRR$

$\$st' \# P_2 \ \$st' \# Q_2$

$ns1 \bowtie ns2$

**shows**  $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) \llbracket ns1 \parallel cs \parallel ns2 \rrbracket \mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3) =$

$\mathbf{R}_s(((Q_1 \Rightarrow_r Q_2) \text{ wr}[ns1|cs|ns2]_C P_1 \wedge$   
 $(Q_1 \Rightarrow_r Q_3) \text{ wr}[ns1|cs|ns2]_C P_1 \wedge$   
 $(P_1 \Rightarrow_r P_2) \text{ wr}[ns2|cs|ns1]_C Q_1 \wedge$   
 $(P_1 \Rightarrow_r P_3) \text{ wr}[ns2|cs|ns1]_C Q_1) \vdash$   
 $((P_1 \Rightarrow_r P_2) \llbracket ns1|cs|ns2 \rrbracket^I (Q_1 \Rightarrow_r Q_2) \vee$   
 $(P_1 \Rightarrow_r P_3) \llbracket ns1|cs|ns2 \rrbracket^I (Q_1 \Rightarrow_r Q_2) \vee$   
 $(P_1 \Rightarrow_r P_2) \llbracket ns1|cs|ns2 \rrbracket^I (Q_1 \Rightarrow_r Q_3)) \diamond$   
 $((P_1 \Rightarrow_r P_3) \llbracket ns1|cs|ns2 \rrbracket^F (Q_1 \Rightarrow_r Q_3)))$

(**is**  $?P \llbracket ns1 \parallel cs \parallel ns2 \rrbracket ?Q = ?rhs$ )

**proof** –

**have**  $?P \llbracket ns1 \parallel cs \parallel ns2 \rrbracket ?Q = (?P \parallel_{M_R(N_C \ ns1 \ cs \ ns2)} ?Q) ;;_h Skip$

**by** (*simp add: CSPMerge-def par-by-merge-seq-add*)

**also**

**have**  $\dots = \mathbf{R}_s(((Q_1 \Rightarrow_r Q_2) \text{ wr}[ns1|cs|ns2]_C P_1 \wedge$

$(Q_1 \Rightarrow_r Q_3) \text{ wr}[ns1|cs|ns2]_C P_1 \wedge$   
 $(P_1 \Rightarrow_r P_2) \text{ wr}[ns2|cs|ns1]_C Q_1 \wedge$   
 $(P_1 \Rightarrow_r P_3) \text{ wr}[ns2|cs|ns1]_C Q_1) \vdash$   
 $((P_1 \Rightarrow_r P_2) \llbracket ns1|cs|ns2 \rrbracket^I (Q_1 \Rightarrow_r Q_2) \vee$   
 $(P_1 \Rightarrow_r P_3) \llbracket ns1|cs|ns2 \rrbracket^I (Q_1 \Rightarrow_r Q_2) \vee$   
 $(P_1 \Rightarrow_r P_2) \llbracket ns1|cs|ns2 \rrbracket^I (Q_1 \Rightarrow_r Q_3)) \diamond$   
 $(P_1 \Rightarrow_r P_3) \parallel_{N_C \ ns1 \ cs \ ns2} (Q_1 \Rightarrow_r Q_3)) ;;_h Skip$

**by** (*simp add: parallel-rdes-def swap-CSPInnerMerge CSPInterMerge-def closure assms*)

**also**

**have**  $\dots = \mathbf{R}_s(((Q_1 \Rightarrow_r Q_2) \text{ wr}[ns1|cs|ns2]_C P_1 \wedge$

$(Q_1 \Rightarrow_r Q_3) \text{ wr}[ns1|cs|ns2]_C P_1 \wedge$

$$\begin{aligned}
& (P_1 \Rightarrow_r P_2) \text{ wr}[ns2|cs|ns1]_C Q_1 \wedge \\
& (P_1 \Rightarrow_r P_3) \text{ wr}[ns2|cs|ns1]_C Q_1 \vdash \\
& ((P_1 \Rightarrow_r P_2) \llbracket ns1|cs|ns2 \rrbracket^I (Q_1 \Rightarrow_r Q_2) \vee \\
& (P_1 \Rightarrow_r P_3) \llbracket ns1|cs|ns2 \rrbracket^I (Q_1 \Rightarrow_r Q_2) \vee \\
& (P_1 \Rightarrow_r P_2) \llbracket ns1|cs|ns2 \rrbracket^I (Q_1 \Rightarrow_r Q_3)) \diamond \\
& (\exists \$ref' \cdot ((P_1 \Rightarrow_r P_3) \parallel_{N_C} ns1 \text{ cs } ns2 (Q_1 \Rightarrow_r Q_3))) \\
& \text{by (simp add: Skip-right-form closure parallel-RR-closed assms unrest)} \\
& \text{also} \\
& \text{have ... = } \mathbf{R}_s (((Q_1 \Rightarrow_r Q_2) \text{ wr}[ns1|cs|ns2]_C P_1 \wedge \\
& (Q_1 \Rightarrow_r Q_3) \text{ wr}[ns1|cs|ns2]_C P_1 \wedge \\
& (P_1 \Rightarrow_r P_2) \text{ wr}[ns2|cs|ns1]_C Q_1 \wedge \\
& (P_1 \Rightarrow_r P_3) \text{ wr}[ns2|cs|ns1]_C Q_1) \vdash \\
& ((P_1 \Rightarrow_r P_2) \llbracket ns1|cs|ns2 \rrbracket^I (Q_1 \Rightarrow_r Q_2) \vee \\
& (P_1 \Rightarrow_r P_3) \llbracket ns1|cs|ns2 \rrbracket^I (Q_1 \Rightarrow_r Q_2) \vee \\
& (P_1 \Rightarrow_r P_2) \llbracket ns1|cs|ns2 \rrbracket^I (Q_1 \Rightarrow_r Q_3)) \diamond \\
& ((P_1 \Rightarrow_r P_3) \llbracket ns1|cs|ns2 \rrbracket^F (Q_1 \Rightarrow_r Q_3))) \\
& \text{proof -} \\
& \text{have } (\exists \$ref' \cdot ((P_1 \Rightarrow_r P_3) \parallel_{N_C} ns1 \text{ cs } ns2 (Q_1 \Rightarrow_r Q_3))) = ((P_1 \Rightarrow_r P_3) \llbracket ns1|cs|ns2 \rrbracket^F (Q_1 \Rightarrow_r \\
& Q_3)) \\
& \text{by (rel-blast)} \\
& \text{thus ?thesis by simp} \\
& \text{qed} \\
& \text{finally show ?thesis .} \\
& \text{qed}
\end{aligned}$$

### 4.3 Parallel Laws

**lemma** *ParCSP-expand*:

$P \llbracket ns1 \parallel cs \parallel ns2 \rrbracket Q = (P \parallel_{R N_C} ns1 \text{ cs } ns2 Q) ;; \text{Skip}$   
 by (simp add: CSPMerge-def par-by-merge-seq-add)

**lemma** *parallel-is-CSP [closure]*:

assumes  $P$  is CSP  $Q$  is CSP  
 shows  $(P \llbracket ns1 \parallel cs \parallel ns2 \rrbracket Q)$  is CSP

**proof** –

have  $(P \parallel_{M_R(N_C \text{ ns1 cs ns2})} Q)$  is CSP  
 by (simp add: closure assms)  
 hence  $(P \parallel_{M_R(N_C \text{ ns1 cs ns2})} Q) ;; \text{Skip}$  is CSP  
 by (simp add: closure)  
 thus ?thesis  
 by (simp add: CSPMerge-def par-by-merge-seq-add)

**qed**

**lemma** *parallel-is-NCSP [closure]*:

assumes  $ns1 \bowtie ns2$   $P$  is NCSP  $Q$  is NCSP  
 shows  $(P \llbracket ns1 \parallel cs \parallel ns2 \rrbracket Q)$  is NCSP

**proof** –

have  $(P \llbracket ns1 \parallel cs \parallel ns2 \rrbracket Q) = (\mathbf{R}_s(\text{pre}_R P \vdash \text{peri}_R P \diamond \text{post}_R P) \llbracket ns1 \parallel cs \parallel ns2 \rrbracket \mathbf{R}_s(\text{pre}_R Q \vdash \text{peri}_R Q \\
 \diamond \text{post}_R Q))$   
 by (metis NCSP-implies-NSRD NSRD-is-SRD SRD-reactive-design-alt assms wait'-cond-peri-post-cmt)  
 also have ... is NCSP  
 by (simp add: ParCSP-rdes-def assms closure unrest)  
 finally show ?thesis .  
**qed**

**theorem** *parallel-commutative*:

**assumes**  $ns1 \bowtie ns2$

**shows**  $(P \llbracket ns1 \parallel cs \parallel ns2 \rrbracket Q) = (Q \llbracket ns2 \parallel cs \parallel ns1 \rrbracket P)$

**proof** –

**have**  $(P \llbracket ns1 \parallel cs \parallel ns2 \rrbracket Q) = P \parallel_{\text{swap}_m} ;; (M_C \text{ } ns2 \text{ } cs \text{ } ns1) \text{ } Q$

**by** (*simp add: CSPMerge-def seqr-assoc [THEN sym] swap-merge-rd swap-CSPInnerMerge lens-indep-sym assms*)

**also have**  $\dots = Q \llbracket ns2 \parallel cs \parallel ns1 \rrbracket P$

**by** (*metis par-by-merge-commute-swap*)

**finally show** *?thesis* .

**qed**

*CSP5* is precisely *C2* when applied to a process

**lemma** *CSP5-is-C2*:

**fixes**  $P :: 'e \text{ process}$

**assumes**  $P \text{ is NCSP}$

**shows**  $CSP5(P) = C2(P)$

**unfolding** *CSP5-def C2-def* **by** (*rdes-eq cls: assms*)

The form of *C2* tells us that a normal CSP process has a downward closed set of refusals

**lemma** *C2-form*:

**assumes**  $P \text{ is NCSP}$

**shows**  $C2(P) = \mathbf{R}_s (pre_R P \vdash (\exists \text{ } ref_0 \cdot peri_R P \llbracket \langle ref_0 \rangle / \$ref' \rrbracket \wedge \$ref' \subseteq_u \langle ref_0 \rangle) \diamond post_R P)$

**proof** –

**have**  $1: \Phi(true, id, \langle \rangle) \text{ wr}[\Sigma | \{\} | \emptyset]_C pre_R P = pre_R P \text{ (is ?lhs = ?rhs)}$

**proof** –

**have**  $?lhs = (\neg_r (\exists (ref_0, st_0, tt_0) \cdot$

$[\$ref' \mapsto_s \langle ref_0 \rangle, \$st' \mapsto_s \langle st_0 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_0 \rangle] \dagger (\exists \$ref'; \$st' \cdot RR(\neg_r$

$pre_R P)) \wedge$

$\$ref' \subseteq_u \langle ref_0 \rangle \wedge [\langle trace \rangle =_u \langle tt_0 \rangle]_t \wedge$

$\$st' =_u \$st \oplus \langle st_0 \rangle \text{ on } \Sigma \oplus \langle id \rangle (\$st)_a \text{ on } \emptyset ;; R1 \text{ true})$

**by** (*simp add: wrR-def par-by-merge-seq-remove rpred merge-csp-do-right ex-unrest Healthy-if pr-var-def closure assms unrest usubst*)

**also have**  $\dots = (\neg_r (\exists \$ref'; \$st' \cdot RR(\neg_r pre_R P)) ;; R1 \text{ true})$

**by** (*rel-auto*)

**also have**  $\dots = (\neg_r (\neg_r pre_R P) ;; R1 \text{ true})$

**by** (*simp add: Healthy-if closure ex-unrest unrest assms*)

**also have**  $\dots = pre_R P$

**by** (*simp add: NCSP-implies-NSRD NSRD-neg-pre-unit R1-preR assms rea-not-not*)

**finally show** *?thesis* .

**qed**

**have**  $2: (pre_R P \Rightarrow_r peri_R P) \llbracket \Sigma | \{\} | \emptyset \rrbracket^I \Phi(true, id, \langle \rangle) =$

$(\exists ref_0 \cdot (peri_R P) \llbracket \langle ref_0 \rangle / \$ref' \rrbracket \wedge \$ref' \subseteq_u \langle ref_0 \rangle) \text{ (is ?lhs = ?rhs)}$

**proof** –

**have**  $?lhs = peri_R P \llbracket \Sigma | \{\} | \emptyset \rrbracket^I \Phi(true, id, \langle \rangle)$

**by** (*simp add: SRD-peri-under-pre closure assms unrest*)

**also have**  $\dots = (\exists \$st' \cdot (peri_R P \parallel_{N_C} 1_L \{\} 0_L \Phi(true, id, \langle \rangle)))$

**by** (*simp add: CSPInterMerge-def par-by-merge-def seqr-exists-right*)

**also have**  $\dots =$

$(\exists \$st' \cdot \exists (ref_0, st_0, tt_0) \cdot$

$[\$ref' \mapsto_s \langle ref_0 \rangle, \$st' \mapsto_s \langle st_0 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_0 \rangle] \dagger (\exists \$st' \cdot RR(peri_R P)) \wedge$

$\$ref' \subseteq_u \langle ref_0 \rangle \wedge [\langle trace \rangle =_u \langle tt_0 \rangle]_t \wedge \$st' =_u \$st \oplus \langle st_0 \rangle \text{ on } \Sigma \oplus \langle id \rangle (\$st)_a \text{ on } \emptyset)$

**by** (*simp add: merge-csp-do-right pr-var-def assms Healthy-if assms closure rpred unrest ex-unrest*)

**also have**  $\dots =$

$(\exists ref_0 \cdot (\exists \$st' \cdot RR(peri_R P)) \llbracket \langle ref_0 \rangle / \$ref' \rrbracket \wedge \$ref' \subseteq_u \langle ref_0 \rangle)$

by (*rel-auto*)  
 also have ... = ?rhs  
 by (*simp add: closure ex-unrest Healthy-if unrest assms*)  
 finally show ?thesis .  
 qed  
 have 3: ( $pre_R P \Rightarrow_r post_R P$ )  $\llbracket \Sigma | \{ \} | \emptyset \rrbracket^F \Phi(true, id, \langle \rangle) = post_R(P)$  (is ?lhs = ?rhs)  
 proof –  
 have ?lhs =  $post_R P \llbracket \Sigma | \{ \} | \emptyset \rrbracket^F \Phi(true, id, \langle \rangle)$   
 by (*simp add: SRD-post-under-pre closure assms unrest*)  
 also have ... =  $(\exists (st_0, t_0) \cdot$   
 $[\$st' \mapsto_s \ll st_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_0 \gg] \dagger RR(post_R P) \wedge$   
 $[\ll trace \gg =_u \ll t_0 \gg]_t \wedge \$st' =_u \$st \oplus \ll st_0 \gg \text{ on } \Sigma \oplus \ll id \gg (\$st)_a \text{ on } \emptyset)$   
 by (*simp add: FinalMerge-csp-do-right pr-var-def assms closure unrest rpred Healthy-if*)  
 also have ... =  $RR(post_R(P))$   
 by (*rel-auto*)  
 finally show ?thesis  
 by (*simp add: Healthy-if assms closure*)  
 qed  
 show ?thesis  
 proof –  
 have  $C2(P) = \mathbf{R}_s (\Phi(true, id, \langle \rangle) \text{ wr } [\Sigma | \{ \} | \emptyset]_C pre_R P \vdash$   
 $(pre_R P \Rightarrow_r peri_R P) \llbracket \Sigma | \{ \} | \emptyset \rrbracket^I \Phi(true, id, \langle \rangle) \diamond (pre_R P \Rightarrow_r post_R P) \llbracket \Sigma | \{ \} | \emptyset \rrbracket^F \Phi(true, id, \langle \rangle))$   
 by (*simp add: C2-def, rdes-simp cls: assms, simp add: id-def pr-var-def*)  
 also have ... =  $\mathbf{R}_s (pre_R P \vdash (\exists ref_0 \cdot peri_R P \llbracket \ll ref_0 \gg / \$ref' \rrbracket \wedge \$ref' \subseteq_u \ll ref_0 \gg) \diamond post_R P)$   
 by (*simp add: 1 2 3*)  
 finally show ?thesis .  
 qed  
 qed  
  
 lemma *C2-CDC-form*:  
 assumes  $P$  is NCSP  
 shows  $C2(P) = \mathbf{R}_s (pre_R P \vdash CDC(peri_R P) \diamond post_R P)$   
 by (*simp add: C2-form assms CDC-def*)  
  
 lemma *C2-rdes-def*:  
 assumes  $P_1$  is CRC  $P_2$  is CRR  $P_3$  is CRR  $\$st' \# P_2 \$ref' \# P_3$   
 shows  $C2(\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3)) = \mathbf{R}_s(P_1 \vdash CDC(P_2) \diamond P_3)$   
 by (*simp add: C2-form assms closure rdes unrest usubst, rel-auto*)  
  
 lemma *C2-NCSP-intro*:  
 assumes  $P$  is NCSP  $peri_R(P)$  is CDC  
 shows  $P$  is C2  
 proof –  
 have  $C2(P) = \mathbf{R}_s (pre_R P \vdash CDC(peri_R P) \diamond post_R P)$   
 by (*simp add: C2-CDC-form assms(1)*)  
 also have ... =  $\mathbf{R}_s (pre_R P \vdash peri_R P \diamond post_R P)$   
 by (*simp add: Healthy-if assms*)  
 also have ... =  $P$   
 by (*simp add: NCSP-implies-CSP SRD-reactive-tri-design assms(1)*)  
 finally show ?thesis  
 by (*simp add: Healthy-def*)  
 qed  
 qed  
  
 lemma *C2-rdes-intro*:  
 assumes  $P_1$  is CRC  $P_2$  is CRR  $P_2$  is CDC  $P_3$  is CRR  $\$st' \# P_2 \$ref' \# P_3$

shows  $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3)$  is  $C2$   
 unfolding *Healthy-def*  
 by (simp add: *C2-rdes-def* *assms unrest closure Healthy-if*)

lemma *C2-implies-CDC-peri* [closure]:

assumes  $P$  is NCSP  $P$  is  $C2$   
 shows  $\text{peri}_R(P)$  is CDC

proof –

have  $\text{peri}_R(P) = \text{peri}_R(\mathbf{R}_s(\text{pre}_R P \vdash \text{CDC}(\text{peri}_R P) \diamond \text{post}_R P))$   
 by (metis *C2-CDC-form Healthy-if* *assms(1)* *assms(2)*)  
 also have  $\dots = \text{CDC}(\text{pre}_R P \Rightarrow_r \text{peri}_R P)$   
 by (simp add: *rdes rpred* *assms closure unrest del: NSRD-peri-under-pre*)  
 also have  $\dots = \text{CDC}(\text{peri}_R P)$   
 by (simp add: *SRD-peri-under-pre closure unrest* *assms*)  
 finally show ?thesis  
 by (simp add: *Healthy-def*)

qed

lemma *CACT-intro*:

assumes  $P$  is NCSP  $P$  is  $C2$   
 shows  $P$  is CACT  
 by (metis *CACT-def Healthy-def* *assms(1)* *assms(2)*)

lemma *CACT-rdes-intro*:

assumes  $P_1$  is CRC  $P_2$  is CRR  $P_2$  is CDC  $P_3$  is CRR  $\$st' \# P_2 \$ref' \# P_3$   
 shows  $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3)$  is CACT  
 by (rule *CACT-intro*, simp add: *closure* *assms*, rule *C2-rdes-intro*, simp-all add: *assms*)

lemma *C2-NCSP-quasi-commute*:

assumes  $P$  is NCSP  
 shows  $C2(\text{NCSP}(P)) = \text{NCSP}(C2(P))$

proof –

have  $1: C2(\text{NCSP}(P)) = C2(P)$   
 by (simp add: *assms Healthy-if*)  
 also have  $\dots = \mathbf{R}_s(\text{pre}_R P \vdash \text{CDC}(\text{peri}_R P) \diamond \text{post}_R P)$   
 by (simp add: *C2-CDC-form* *assms*)  
 also have  $\dots$  is NCSP  
 by (rule *NCSP-rdes-intro*, simp-all add: *closure* *assms unrest*)  
 finally show ?thesis  
 by (simp add: *Healthy-if 1*)

qed

lemma *C2-quasi-idem*:

assumes  $P$  is NCSP  
 shows  $C2(C2(P)) = C2(P)$

proof –

have  $C2(C2(P)) = C2(C2(\mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \diamond \text{post}_R(P))))$   
 by (simp add: *NCSP-implies-NSRD NSRD-is-SRD SRD-reactive-tri-design* *assms*)  
 also have  $\dots = \mathbf{R}_s(\text{pre}_R P \vdash \text{CDC}(\text{peri}_R P) \diamond \text{post}_R P)$   
 by (simp add: *C2-rdes-def closure* *assms unrest CDC-idem*)  
 also have  $\dots = C2(P)$   
 by (simp add: *C2-CDC-form* *assms*)  
 finally show ?thesis .

qed

**lemma** *CACT-implies-NCSP* [closure]:

assumes *P* is *CACT*

shows *P* is *NCSP*

**proof** –

have  $P = C2(NCSP(NCSP(P)))$

by (*metis CACT-def Healthy-Idempotent Healthy-if NCSP-Idempotent assms*)

also have  $\dots = NCSP(C2(NCSP(P)))$

by (*simp add: C2-NCSP-quasi-commute Healthy-Idempotent NCSP-Idempotent*)

also have  $\dots$  is *NCSP*

by (*metis CACT-def Healthy-def assms calculation*)

finally show *?thesis* .

**qed**

**lemma** *CACT-implies-C2* [closure]:

assumes *P* is *CACT*

shows *P* is *C2*

by (*metis CACT-def CACT-implies-NCSP Healthy-def assms*)

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

by (*simp add: CACT-def C2-NCSP-quasi-commute[THEN sym] C2-quasi-idem Healthy-Idempotent Healthy-if NCSP-Idempotent*)

**lemma** *CACT-Idempotent*: *Idempotent CACT*

by (*simp add: CACT-idem Idempotent-def*)

**lemma** *PACT-elim* [*RD-elim*]:

$\llbracket X \text{ is } CACT; P(\mathbf{R}_s(\text{pre}_R(X) \vdash \text{peri}_R(X) \diamond \text{post}_R(X))) \rrbracket \implies P(X)$

using *CACT-implies-NCSP NCSP-elim* **by** *blast*

**lemma** *Miracle-C2-closed* [closure]: *Miracle is C2*

by (*rdes-simp, rule C2-rdes-intro, simp-all add: closure unrest*)

**lemma** *Chaos-C2-closed* [closure]: *Chaos is C2*

by (*rdes-simp, rule C2-rdes-intro, simp-all add: closure unrest*)

**lemma** *Skip-C2-closed* [closure]: *Skip is C2*

by (*rdes-simp, rule C2-rdes-intro, simp-all add: closure unrest*)

**lemma** *Stop-C2-closed* [closure]: *Stop is C2*

by (*rdes-simp, rule C2-rdes-intro, simp-all add: closure unrest*)

**lemma** *Miracle-CACT-closed* [closure]: *Miracle is CACT*

by (*simp add: CACT-intro Miracle-C2-closed csp-theory.top-closed*)

**lemma** *Chaos-CACT-closed* [closure]: *Chaos is CACT*

by (*simp add: CACT-intro closure*)

**lemma** *Skip-CACT-closed* [closure]: *Skip is CACT*

by (*simp add: CACT-intro closure*)

**lemma** *Stop-CACT-closed* [closure]: *Stop is CACT*

by (*simp add: CACT-intro closure*)

**lemma** *seq-C2-closed* [closure]:

assumes *P* is *NCSP* *P* is *C2* *Q* is *NCSP* *Q* is *C2*

**shows**  $P \;; \; Q$  is  $C2$   
**by** (*rdes-simp* *cls*: *assms*(1,3), *rule* *C2-rdes-intro*, *simp-all* *add*: *closure* *assms* *unrest*)

**lemma** *seq-CACT-closed* [*closure*]:  
**assumes**  $P$  is *CACT*  $Q$  is *CACT*  
**shows**  $P \;; \; Q$  is *CACT*  
**by** (*meson* *CACT-implies-C2* *CACT-implies-NCSP* *CACT-intro* *assms* *csp-theory.Healthy-Sequence* *seq-C2-closed*)

**lemma** *AssignsCSP-C2* [*closure*]:  $\langle \sigma \rangle_C$  is  $C2$   
**by** (*rdes-simp*, *rule* *C2-rdes-intro*, *simp-all* *add*: *closure* *unrest*)

**lemma** *AssignsCSP-CACT* [*closure*]:  $\langle \sigma \rangle_C$  is *CACT*  
**by** (*simp* *add*: *CACT-intro* *closure*)

**lemma** *map-st-ext-CDC-closed* [*closure*]:  
**assumes**  $P$  is *CDC*  
**shows**  $P \oplus_r \text{map-st}_L[a]$  is *CDC*  
**proof** –  
**have**  $CDC \; P \oplus_r \text{map-st}_L[a]$  is *CDC*  
**by** (*rel-auto*)  
**thus** *?thesis*  
**by** (*simp* *add*: *assms* *Healthy-if*)  
**qed**

**lemma** *rdes-frame-ext-C2-closed* [*closure*]:  
**assumes**  $P$  is *NCSP*  $P$  is  $C2$   
**shows**  $a:[P]_R^+$  is  $C2$   
**by** (*rdes-simp* *cls*:*assms*(2), *rule* *C2-rdes-intro*, *simp-all* *add*: *closure* *assms* *unrest*)

**lemma** *rdes-frame-ext-CACT-closed* [*closure*]:  
**assumes** *vwb-lens*  $a$   $P$  is *CACT*  
**shows**  $a:[P]_R^+$  is *CACT*  
**by** (*rule* *CACT-intro*, *simp-all* *add*: *closure* *assms*)

**lemma** *UINF-C2-closed* [*closure*]:  
**assumes**  $A \neq \{\}$   $\bigwedge i. i \in A \implies P(i)$  is *NCSP*  $\bigwedge i. i \in A \implies P(i)$  is  $C2$   
**shows**  $(\bigcap i \in A \cdot P(i))$  is  $C2$   
**proof** –  
**have**  $(\bigcap i \in A \cdot P(i)) = (\bigcap i \in A \cdot \mathbf{R}_s(\text{pre}_R(P(i)) \vdash \text{peri}_R(P(i)) \diamond \text{post}_R(P(i))))$   
**by** (*simp* *add*: *closure* *SRD-reactive-tri-design* *assms* *cong*: *UINF-cong*)  
**also have** ... is  $C2$   
**by** (*rdes-simp* *cls*: *assms*, *rule* *C2-rdes-intro*, *simp-all* *add*: *closure* *unrest* *assms*)  
**finally show** *?thesis* .  
**qed**

**lemma** *UINF-CACT-closed* [*closure*]:  
**assumes**  $A \neq \{\}$   $\bigwedge i. i \in A \implies P(i)$  is *CACT*  
**shows**  $(\bigcap i \in A \cdot P(i))$  is *CACT*  
**by** (*rule* *CACT-intro*, *simp-all* *add*: *assms* *closure*)

**lemma** *inf-C2-closed* [*closure*]:  
**assumes**  $P$  is *NCSP*  $Q$  is *NCSP*  $P$  is  $C2$   $Q$  is  $C2$   
**shows**  $P \sqcap Q$  is  $C2$   
**by** (*rdes-simp* *cls*: *assms*, *rule* *C2-rdes-intro*, *simp-all* *add*: *closure* *unrest* *assms*)



**lemma** *cond-CDC-closed* [closure]:

assumes  $P$  is CDC  $Q$  is CDC

shows  $P \triangleleft b \triangleright_R Q$  is CDC

**proof** –

have CDC  $P \triangleleft b \triangleright_R CDC Q$  is CDC

by (*rel-auto*)

thus ?thesis

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

**qed**

**lemma** *cond-C2-closed* [closure]:

assumes  $P$  is NCSP  $Q$  is NCSP  $P$  is C2  $Q$  is C2

shows  $P \triangleleft b \triangleright_R Q$  is C2

by (*rdes-simp cls: assms, rule C2-rdes-intro, simp-all add: closure unrest assms*)

**lemma** *cond-CACT-closed* [closure]:

assumes  $P$  is CACT  $Q$  is CACT

shows  $P \triangleleft b \triangleright_R Q$  is CACT

by (*rule CACT-intro, simp-all add: assms closure*)

**lemma** *gcomm-C2-closed* [closure]:

assumes  $P$  is NCSP  $P$  is C2

shows  $b \rightarrow_R P$  is C2

by (*rdes-simp cls: assms, rule C2-rdes-intro, simp-all add: closure unrest assms*)

**lemma** *AssumeCircus-CACT* [closure]:  $[b]_C$  is CACT

by (*metis AssumeCircus-NCSP AssumeCircus-def CACT-intro NCSP-Skip Skip-C2-closed gcomm-C2-closed*)

**lemma** *StateInvR-CACT* [closure]:  $\text{inv}_R(b)$  is CACT

by (*simp add: CACT-rdes-intro rdes-def closure unrest*)

**lemma** *AlternateR-C2-closed* [closure]:

assumes

$\bigwedge i. i \in A \implies P(i)$  is NCSP  $Q$  is NCSP

$\bigwedge i. i \in A \implies P(i)$  is C2  $Q$  is C2

shows  $(\text{if}_R i \in A \cdot g(i) \rightarrow P(i) \text{ else } Q \text{ fi})$  is C2

**proof** (*cases*  $A = \{\}$ )

case *True*

then show ?thesis

by (*simp add: assms(4)*)

next

case *False*

then show ?thesis

by (*simp add: AlternateR-def closure assms*)

**qed**

**lemma** *AlternateR-CACT-closed* [closure]:

assumes  $\bigwedge i. i \in A \implies P(i)$  is CACT  $Q$  is CACT

shows  $(\text{if}_R i \in A \cdot g(i) \rightarrow P(i) \text{ else } Q \text{ fi})$  is CACT

by (*rule CACT-intro, simp-all add: closure assms*)

**lemma** *AlternateR-list-C2-closed* [closure]:

assumes

$\bigwedge b P. (b, P) \in \text{set } A \implies P$  is NCSP  $Q$  is NCSP

$\bigwedge b P. (b, P) \in \text{set } A \implies P \text{ is } C2 \text{ } Q \text{ is } C2$   
**shows** (*AlternateR-list A Q*) *is C2*  
**apply** (*simp add: AlternateR-list-def*)  
**apply** (*rule AlternateR-C2-closed*)  
**apply** (*auto simp add: assms closure*)  
**apply** (*metis assms nth-mem prod.collapse*)+  
**done**

**lemma** *AlternateR-list-CACT-closed* [closure]:  
**assumes**  $\bigwedge b P. (b, P) \in \text{set } A \implies P \text{ is } CACT \text{ } Q \text{ is } CACT$   
**shows** (*AlternateR-list A Q*) *is CACT*  
**by** (*rule CACT-intro, simp-all add: closure assms*)

**lemma** *R4-CRR-closed* [closure]:  $P \text{ is } CRR \implies R4(P) \text{ is } CRR$   
**by** (*rule CRR-intro, simp-all add: closure unrest R4-def*)

**lemma** *WhileC-C2-closed* [closure]:  
**assumes**  $P \text{ is } NCSP \text{ } P \text{ is } Productive \text{ } P \text{ is } C2$   
**shows** *while<sub>C</sub> b do P od is C2*

**proof** –

**have** *while<sub>C</sub> b do P od = while<sub>C</sub> b do Productive(**R<sub>s</sub>** (*pre<sub>R</sub> P*  $\vdash$  *peri<sub>R</sub> P*  $\diamond$  *post<sub>R</sub> P*)) od*  
**by** (*simp add: assms Healthy-if SRD-reactive-tri-design closure*)  
**also have** *... = while<sub>C</sub> b do **R<sub>s</sub>** (*pre<sub>R</sub> P*  $\vdash$  *peri<sub>R</sub> P*  $\diamond$  *R4(post<sub>R</sub> P)*) od*  
**by** (*simp add: Productive-RHS-design-form unrest assms rdes closure R4-def*)  
**also have** *... is C2*  
**by** (*simp add: closure assms unrest rdes-def C2-rdes-intro*)  
**finally show** *?thesis* .

**qed**

**lemma** *WhileC-CACT-closed* [closure]:  
**assumes**  $P \text{ is } CACT \text{ } P \text{ is } Productive$   
**shows** *while<sub>C</sub> b do P od is CACT*  
**using** *CACT-implies-C2 CACT-implies-NCSP CACT-intro WhileC-C2-closed WhileC-NCSP-closed*  
**assms by blast**

**lemma** *IterateC-C2-closed* [closure]:  
**assumes**  
 $\bigwedge i. i \in A \implies P(i) \text{ is } NCSP \bigwedge i. i \in A \implies P(i) \text{ is } Productive \bigwedge i. i \in A \implies P(i) \text{ is } C2$   
**shows** (*do<sub>C</sub> i $\in$ A  $\cdot$  g(i)  $\rightarrow$  P(i) od*) *is C2*  
**unfolding** *IterateC-def* **by** (*simp add: closure assms*)

**lemma** *IterateC-CACT-closed* [closure]:  
**assumes**  
 $\bigwedge i. i \in A \implies P(i) \text{ is } CACT \bigwedge i. i \in A \implies P(i) \text{ is } Productive$   
**shows** (*do<sub>C</sub> i $\in$ A  $\cdot$  g(i)  $\rightarrow$  P(i) od*) *is CACT*  
**by** (*metis CACT-implies-C2 CACT-implies-NCSP CACT-intro IterateC-C2-closed IterateC-NCSP-closed*  
*assms*)

**lemma** *IterateC-list-C2-closed* [closure]:  
**assumes**  
 $\bigwedge b P. (b, P) \in \text{set } A \implies P \text{ is } NCSP$   
 $\bigwedge b P. (b, P) \in \text{set } A \implies P \text{ is } Productive$   
 $\bigwedge b P. (b, P) \in \text{set } A \implies P \text{ is } C2$   
**shows** (*IterateC-list A*) *is C2*  
**unfolding** *IterateC-list-def*

by (rule IterateC-C2-closed, (metis assms atLeastLessThan-iff nth-map nth-mem prod.collapse)+)

**lemma** *IterateC-list-CACT-closed* [closure]:  
**assumes**  
 $\bigwedge b P. (b, P) \in \text{set } A \implies P \text{ is CACT}$   
 $\bigwedge b P. (b, P) \in \text{set } A \implies P \text{ is Productive}$   
**shows** *(IterateC-list A) is CACT*  
**by** (metis CACT-implies-C2 CACT-implies-NCSP CACT-intro IterateC-list-C2-closed IterateC-list-NCSP-closed assms)

**lemma** *GuardCSP-C2-closed* [closure]:  
**assumes** *P is NCSP P is C2*  
**shows**  $g \&_u P \text{ is C2}$   
**by** (rdes-simp cls: assms(1), rule C2-rdes-intro, simp-all add: closure assms unrest)

**lemma** *GuardCSP-CACT-closed* [closure]:  
**assumes** *P is CACT*  
**shows**  $g \&_u P \text{ is CACT}$   
**by** (rule CACT-intro, simp-all add: closure assms)

**lemma** *DoCSP-C2* [closure]:  
 $do_C(a) \text{ is C2}$   
**by** (rdes-simp, rule C2-rdes-intro, simp-all add: closure unrest)

**lemma** *DoCSP-CACT* [closure]:  
 $do_C(a) \text{ is CACT}$   
**by** (rule CACT-intro, simp-all add: closure)

**lemma** *PrefixCSP-C2-closed* [closure]:  
**assumes** *P is NCSP P is C2*  
**shows**  $a \rightarrow_C P \text{ is C2}$   
**unfolding** *PrefixCSP-def* **by** (metis DoCSP-C2 Healthy-def NCSP-DoCSP NCSP-implies-CSP assms seq-C2-closed)

**lemma** *PrefixCSP-CACT-closed* [closure]:  
**assumes** *P is CACT*  
**shows**  $a \rightarrow_C P \text{ is CACT}$   
**using** *CACT-implies-C2 CACT-implies-NCSP CACT-intro NCSP-PrefixCSP PrefixCSP-C2-closed*  
**assms** **by** blast

**lemma** *ExtChoice-C2-closed* [closure]:  
**assumes**  $\bigwedge i. i \in I \implies P(i) \text{ is NCSP} \bigwedge i. i \in I \implies P(i) \text{ is C2}$   
**shows**  $(\square i \in I \cdot P(i)) \text{ is C2}$   
**proof** (cases  $I = \{\}$ )  
**case** *True*  
**then show** *?thesis* **by** (simp add: closure ExtChoice-empty)  
**next**  
**case** *False*  
**show** *?thesis*  
**by** (rule C2-NCSP-intro, simp-all add: assms closure unrest periR-ExtChoice-ind' False)  
**qed**

**lemma** *ExtChoice-CACT-closed* [closure]:  
**assumes**  $\bigwedge i. i \in I \implies P(i) \text{ is CACT}$   
**shows**  $(\square i \in I \cdot P(i)) \text{ is CACT}$

by (rule *CACT-intro*, simp-all add: closure assms)

**lemma** *extChoice-C2-closed* [closure]:  
 assumes  $P$  is NCSP  $P$  is C2  $Q$  is NCSP  $Q$  is C2  
 shows  $P \sqcap Q$  is C2  
**proof** –  
 have  $P \sqcap Q = (\sqcap I \in \{P, Q\} \cdot I)$   
 by (simp add: *extChoice-def*)  
 also have ... is C2  
 by (rule *ExtChoice-C2-closed*, auto simp add: assms)  
 finally show ?thesis .  
**qed**

**lemma** *extChoice-CACT-closed* [closure]:  
 assumes  $P$  is CACT  $Q$  is CACT  
 shows  $P \sqcap Q$  is CACT  
 by (rule *CACT-intro*, simp-all add: closure assms)

**lemma** *state-srea-C2-closed* [closure]:  
 assumes  $P$  is NCSP  $P$  is C2  
 shows state ' $a$  ·  $P$  is C2  
 by (rule *C2-NCSP-intro*, simp-all add: closure rdes assms)

**lemma** *state-srea-CACT-closed* [closure]:  
 assumes  $P$  is CACT  
 shows state ' $a$  ·  $P$  is CACT  
 by (rule *CACT-intro*, simp-all add: closure assms)

**lemma** *parallel-C2-closed* [closure]:  
 assumes  $ns1 \bowtie ns2$   $P$  is NCSP  $Q$  is NCSP  $P$  is C2  $Q$  is C2  
 shows  $(P \llbracket ns1 \parallel cs \parallel ns2 \rrbracket Q)$  is C2  
**proof** –  
 have  $(P \llbracket ns1 \parallel cs \parallel ns2 \rrbracket Q) = (\mathbf{R}_s(\text{pre}_R P \vdash \text{peri}_R P \diamond \text{post}_R P) \llbracket ns1 \parallel cs \parallel ns2 \rrbracket \mathbf{R}_s(\text{pre}_R Q \vdash \text{peri}_R Q \diamond \text{post}_R Q))$   
 by (metis *NCSP-implies-NSRD NSRD-is-SRD SRD-reactive-design-alt* assms wait'-cond-peri-post-cmt)  
 also have ... is C2  
 by (simp add: *ParCSP-rdes-def C2-rdes-intro* assms closure unrest)  
 finally show ?thesis .  
**qed**

**lemma** *parallel-CACT-closed* [closure]:  
 assumes  $ns1 \bowtie ns2$   $P$  is CACT  $Q$  is CACT  
 shows  $(P \llbracket ns1 \parallel cs \parallel ns2 \rrbracket Q)$  is CACT  
 by (meson *CACT-implies-C2 CACT-implies-NCSP CACT-intro* assms *parallel-C2-closed parallel-is-NCSP*)

**lemma** *RenameCSP-C2-closed* [closure]:  
 assumes  $P$  is NCSP  $P$  is C2  
 shows  $P \llbracket f \rrbracket_C$  is C2  
 by (simp add: *RenameCSP-def C2-rdes-intro RenameCSP-pre-CRC-closed* closure assms unrest)

**lemma** *RenameCSP-CACT-closed* [closure]:  
 assumes  $P$  is CACT  
 shows  $P \llbracket f \rrbracket_C$  is CACT  
 by (rule *CACT-intro*, simp-all add: closure assms)

This property depends on downward closure of the refusals

**lemma** *rename-extChoice-pre*:

**assumes** *inj f P is NCSP Q is NCSP P is C2 Q is C2*  
**shows**  $(P \sqcap Q) \llbracket f \rrbracket_C = (P \llbracket f \rrbracket_C \sqcap Q \llbracket f \rrbracket_C)$   
**by** (*rdes-eq-split cls: assms*)

**lemma** *rename-extChoice*:

**assumes** *inj f P is CACT Q is CACT*  
**shows**  $(P \sqcap Q) \llbracket f \rrbracket_C = (P \llbracket f \rrbracket_C \sqcap Q \llbracket f \rrbracket_C)$   
**by** (*simp add: CACT-implies-C2 CACT-implies-NCSP assms rename-extChoice-pre*)

**lemma** *interleave-commute*:

$P \parallel Q = Q \parallel P$   
**using** *parallel-commutative zero-lens-indep* **by** *blast*

**lemma** *interleave-unit*:

**assumes** *P is CPROC*  
**shows**  $P \parallel \text{Skip} = P$   
**by** (*metis CACT-implies-C2 CACT-implies-NCSP CSP5-def CSP5-is-C2 Healthy-if assms*)

**lemma** *parallel-miracle*:

$P \text{ is NCSP} \implies \text{Miracle } \llbracket ns1 \parallel cs \parallel ns2 \rrbracket P = \text{Miracle}$   
**by** (*simp add: CSPMerge-def par-by-merge-seq-add[THEN sym] Miracle-parallel-left-zero Skip-right-unit closure*)

**lemma**

**assumes** *vwb-lens ns1 vwb-lens ns2 ns1  $\bowtie$  ns2 P is RR*  
**shows**  $P \text{ wr}[ns1|cs|ns2]_C \text{ false} = \text{undefined}$  (**is** *?lhs = ?rhs*)

**proof** –

**have** *?lhs* =  $(\neg_r (\exists (ref_0, ref_1, st_0, st_1, tt_0, tt_1) \cdot$   
 $[\$ref' \mapsto_s \langle ref_0 \rangle, \$st' \mapsto_s \langle st_0 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_0 \rangle] \dagger R1 \text{ true} \wedge$   
 $[\$ref' \mapsto_s \langle ref_1 \rangle, \$st' \mapsto_s \langle st_1 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_1 \rangle] \dagger P \wedge$   
 $\$ref' \subseteq_u (\langle ref_0 \rangle \cup_u \langle ref_1 \rangle) \cap_u \langle cs \rangle \cup_u (\langle ref_0 \rangle \cap_u \langle ref_1 \rangle - \langle cs \rangle) \wedge$   
 $\$tr \leq_u \$tr' \wedge$   
 $\&tt \in_u \langle tt_0 \rangle \star_{cs} \langle tt_1 \rangle \wedge \langle tt_0 \rangle \upharpoonright_u \langle cs \rangle =_u \langle tt_1 \rangle \upharpoonright_u \langle cs \rangle \wedge$   
 $\$st' =_u \$st \oplus \langle st_0 \rangle \text{ on } \&ns1 \oplus \langle st_1 \rangle \text{ on } \&ns2) ;;$   
 $R1 \text{ true})$

**by** (*simp add: wrR-def par-by-merge-seq-remove CSPInnerMerge-form assms closure usubst unrest*)

**also have** ... =  $(\neg_r (\exists (tt_0, tt_1) \cdot$

$[\$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_1 \rangle] \dagger P \wedge$   
 $\$tr \leq_u \$tr' \wedge$   
 $\&tt \in_u \langle tt_0 \rangle \star_{cs} \langle tt_1 \rangle \wedge \langle tt_0 \rangle \upharpoonright_u \langle cs \rangle =_u \langle tt_1 \rangle \upharpoonright_u \langle cs \rangle) ;;$   
 $R1 \text{ true})$

**by** (*rel-blast*)

**also have** ... =  $(\neg_r (\exists (tt_0, tt_1) \cdot$

$[\$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_1 \rangle] \dagger RR(P) \wedge$   
 $\$tr \leq_u \$tr' \wedge$   
 $\&tt \in_u \langle tt_0 \rangle \star_{cs} \langle tt_1 \rangle \wedge \langle tt_0 \rangle \upharpoonright_u \langle cs \rangle =_u \langle tt_1 \rangle \upharpoonright_u \langle cs \rangle) ;;$   
 $R1 \text{ true})$

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

**oops**

**end**

## 5 Hiding

theory *utp-circus-hiding*  
 imports *utp-circus-parallel*  
 begin

### 5.1 Hiding in peri- and postconditions

**definition** *hide-rea* (*hide<sub>r</sub>*) **where**

[*upred-defs*]:  $hide_r P E = (\exists s \cdot (P[\$tr^u \ll s \gg, (\ll E \gg \cup_u \$ref') / \$tr', \$ref'] \wedge \$tr' =_u \$tr^u (\ll s \gg \downarrow_u \ll -E \gg)))$

**lemma** *hide-rea-CRR-closed* [*closure*]:

assumes *P is CRR*

shows *hide<sub>r</sub> P E is CRR*

**proof** –

have  $CRR(hide_r (CRR P) E) = hide_r (CRR P) E$

by (*rel-auto*, *fastforce* +)

thus ?thesis

by (*metis Healthy-def' assms*)

qed

**lemma** *hide-rea-CDC* [*closure*]:

assumes *P is CDC*

shows *hide<sub>r</sub> P E is CDC*

**proof** –

have  $CDC(hide_r (CDC P) E) = hide_r (CDC P) E$

by (*rel-blast*)

thus ?thesis

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

qed

**lemma** *hide-rea-false* [*rpred*]:  $hide_r false E = false$

by (*rel-auto*)

**lemma** *hide-rea-disj* [*rpred*]:  $hide_r (P \vee Q) E = (hide_r P E \vee hide_r Q E)$

by (*rel-auto*)

**lemma** *hide-rea-csp-enable* [*rpred*]:

$hide_r \mathcal{E}(s, t, E) F = \mathcal{E}(s \wedge E - \ll F \gg =_u E, t \downarrow_u \ll -F \gg, E)$

by (*rel-auto*)

**lemma** *hide-rea-csp-do* [*rpred*]:  $hide_r \Phi(s, \sigma, t) E = \Phi(s, \sigma, t \downarrow_u \ll -E \gg)$

by (*rel-auto*)

**lemma** *filter-eval* [*simp*]:

$(bop Cons x xs) \downarrow_u E = (bop Cons x (xs \downarrow_u E) \triangleleft x \in_u E \triangleright xs \downarrow_u E)$

by (*rel-simp*)

**lemma** *hide-rea-seq* [*rpred*]:

assumes *P is CRR*  $\$ref' \# P Q$  *is CRR*

shows  $hide_r (P ;; Q) E = hide_r P E ;; hide_r Q E$

**proof** –

have  $hide_r (CRR(\exists \$ref' \cdot P) ;; CRR(Q)) E = hide_r (CRR(\exists \$ref' \cdot P)) E ;; hide_r (CRR Q) E$

apply (*simp add: hide-rea-def usubst unrest CRR-seqr-form*)

apply (*simp add: CRR-form*)

apply (*rel-auto*)

```

    using seq-filter-append apply fastforce
    apply (metis seq-filter-append)
  done
thus ?thesis
  by (simp add: Healthy-if assms ex-unrest)
qed

```

```

lemma hide-rea-true-R1-true [rpred]:
  hider (R1 true) A ;; R1 true = R1 true
  by (rel-auto, metis append-Nil2 seq-filter-Nil)

```

```

lemma hide-rea-empty [rpred]:
  assumes P is RR
  shows hider P {} = P
proof -
  have hider (RR P) {} = (RR P)
    by (rel-auto; force)
  thus ?thesis
    by (simp add: Healthy-if assms)
qed

```

```

lemma hide-rea-twice [rpred]: hider (hider P A) B = hider P (A ∪ B)
  apply (rel-auto)
  apply (metis (no-types, hide-lams) semilattice-sup-class.sup-assoc)
  apply (metis (no-types, lifting) semilattice-sup-class.sup-assoc seq-filter-twice)
  done

```

```

lemma st'-unrest-hide-rea [unrest]: $st' # P ⇒ $st' # hider P E
  by (simp add: hide-rea-def unrest)

```

```

lemma ref'-unrest-hide-rea [unrest]: $ref' # P ⇒ $ref' # hider P E
  by (simp add: hide-rea-def unrest usubst)

```

## 5.2 Hiding in preconditions

**definition** *abs-rea* :: ('s, 'e) action ⇒ 'e set ⇒ ('s, 'e) action (*abs<sub>r</sub>*) **where**  
*[upred-defs]*: *abs<sub>r</sub>* P E = (¬<sub>r</sub> (hide<sub>r</sub> (¬<sub>r</sub> P) E ;; true<sub>r</sub>))

```

lemma abs-rea-false [rpred]: absr false E = false
  by (rel-simp, metis append.right-neutral seq-filter-Nil)

```

```

lemma abs-rea-conj [rpred]: absr (P ∧ Q) E = (absr P E ∧ absr Q E)
  by (rel-blast)

```

```

lemma abs-rea-true [rpred]: absr truer E = truer
  by (rel-auto)

```

```

lemma abs-rea-RC-closed [closure]:
  assumes P is CRR
  shows absr P E is CRC
proof -
  have RC1 (absr (CRR P) E) = absr (CRR P) E
    apply (rel-auto)
    apply (metis order-refl)
    apply blast
  done

```

hence  $abs_r P E$  is  $RC1$   
 by (simp add: assms Healthy-if Healthy-intro closure)  
 thus ?thesis  
 by (rule-tac CRC-intro'', simp-all add: abs-rea-def closure assms unrest)  
 qed

**lemma** *hide-rea-impl-under-abs*:  
 assumes  $P$  is  $CRC$   $Q$  is  $CRR$   
 shows  $(abs_r P A \Rightarrow_r hide_r (P \Rightarrow_r Q) A) = (abs_r P A \Rightarrow_r hide_r Q A)$   
 by (simp add: RC1-def abs-rea-def rea-impl-def rpred closure assms unrest)  
 (rel-auto, metis order-refl)

**lemma** *abs-rea-not-CRR*:  $P$  is  $CRR \implies abs_r (\neg_r P) E = (\neg_r hide_r P E ;; R1 \text{ true})$   
 by (simp add: abs-rea-def rpred closure)

**lemma** *abs-rea-wpR* [rpred]:  
 assumes  $P$  is  $CRR$   $\$ref' \# P Q$  is  $CRC$   
 shows  $abs_r (P wp_r Q) A = (hide_r P A) wp_r (abs_r Q A)$   
 by (simp add: wp-rea-def abs-rea-not-CRR hide-rea-seq assms closure)  
 (simp add: abs-rea-def rpred closure assms seqr-assoc)

**lemma** *abs-rea-empty* [rpred]:  
 assumes  $P$  is  $RC$   
 shows  $abs_r P \{\} = P$   
**proof** –  
 have  $abs_r (RC P) \{\} = (RC P)$   
 apply (rel-auto)  
 apply (metis diff-add-cancel-left' order-refl plus-list-def)  
 using dual-order.trans apply blast  
 done  
 thus ?thesis  
 by (simp add: Healthy-if assms)  
 qed

**lemma** *abs-rea-twice* [rpred]:  
 assumes  $P$  is  $CRC$   
 shows  $abs_r (abs_r P A) B = abs_r P (A \cup B)$  (is ?lhs = ?rhs)  
**proof** –  
 have ?lhs =  $abs_r (\neg_r hide_r (\neg_r P) A ;; R1 \text{ true}) B$   
 by (simp add: abs-rea-def)  
 thus ?thesis  
 by (simp add: abs-rea-def rpred closure unrest seqr-assoc assms)  
 qed

### 5.3 Hiding Operator

In common with the UTP book definition of hiding, this definition does not introduce divergence if there is an infinite sequence of events that are hidden. For this, we would need a more complex precondition which is left for future work.

**definition** *HideCSP* ::  $('s, 'e) \text{ action} \Rightarrow 'e \text{ set} \Rightarrow ('s, 'e) \text{ action} (\text{infixl } \setminus_C 80)$  **where**  
 [upred-defs]:  
 $HideCSP P E = \mathbf{R}_s(abs_r(pre_R(P)) E \vdash hide_r(peri_R(P)) E \diamond hide_r(post_R(P)) E)$

**lemma** *HideCSP-rdes-def* [rdes-def]:



**assumes**  $P$  is CRC  $Q$  is CRR  $R$  is CRR  
**shows**  $\mathbf{R}_s(P \vdash Q \diamond R) \setminus_C A = \mathbf{R}_s(\text{abs}_r(P) A \vdash \text{hide}_r Q A \diamond \text{hide}_r R A)$  (**is**  $?lhs = ?rhs$ )  
**proof** –  
**have**  $?lhs = \mathbf{R}_s(\text{abs}_r P A \vdash \text{hide}_r (P \Rightarrow_r Q) A \diamond \text{hide}_r (P \Rightarrow_r R) A)$   
**by** (*simp add: HideCSP-def rdes assms closure*)  
**also have**  $\dots = \mathbf{R}_s(\text{abs}_r P A \vdash (\text{abs}_r P A \Rightarrow_r \text{hide}_r (P \Rightarrow_r Q) A) \diamond (\text{abs}_r P A \Rightarrow_r \text{hide}_r (P \Rightarrow_r R) A))$   
**by** (*metis RHS-tri-design-conj conj-idem utp-pred-laws.sup.idem*)  
**also have**  $\dots = ?rhs$   
**by** (*metis RHS-tri-design-conj assms conj-idem hide-rea-impl-under-abs utp-pred-laws.sup.idem*)  
**finally show**  $?thesis$  .  
**qed**

**lemma** *HideCSP-NCSP-closed* [*closure*]:  $P$  is NCSP  $\implies P \setminus_C E$  is NCSP  
**by** (*simp add: HideCSP-def closure unrest*)

**lemma** *HideCSP-C2-closed* [*closure*]:  
**assumes**  $P$  is NCSP  $P$  is C2  
**shows**  $P \setminus_C E$  is C2  
**by** (*rdes-simp cls: assms, simp add: C2-rdes-intro closure unrest assms*)

**lemma** *HideCSP-CACT-closed* [*closure*]:  
**assumes**  $P$  is CACT  
**shows**  $P \setminus_C E$  is CACT  
**by** (*rule CACT-intro, simp-all add: closure assms*)

**lemma** *HideCSP-Chaos*:  $\text{Chaos} \setminus_C E = \text{Chaos}$   
**by** (*rdes-simp*)

**lemma** *HideCSP-Miracle*:  $\text{Miracle} \setminus_C A = \text{Miracle}$   
**by** (*rdes-eq*)

**lemma** *HideCSP-AssignsCSP*:  
 $\langle \sigma \rangle_C \setminus_C A = \langle \sigma \rangle_C$   
**by** (*rdes-eq*)

**lemma** *HideCSP-cond*:  
**assumes**  $P$  is NCSP  $Q$  is NCSP  
**shows**  $(P \triangleleft b \triangleright_R Q) \setminus_C A = (P \setminus_C A \triangleleft b \triangleright_R Q \setminus_C A)$   
**by** (*rdes-eq cls: assms*)

**lemma** *HideCSP-int-choice*:  
**assumes**  $P$  is NCSP  $Q$  is NCSP  
**shows**  $(P \sqcap Q) \setminus_C A = (P \setminus_C A \sqcap Q \setminus_C A)$   
**by** (*rdes-eq cls: assms*)

**lemma** *HideCSP-guard*:  
**assumes**  $P$  is NCSP  
**shows**  $(b \&_u P) \setminus_C A = b \&_u (P \setminus_C A)$   
**by** (*rdes-eq cls: assms*)

**lemma** *HideCSP-seq*:  
**assumes**  $P$  is NCSP  $Q$  is NCSP  
**shows**  $(P ;; Q) \setminus_C A = (P \setminus_C A ;; Q \setminus_C A)$   
**by** (*rdes-eq-split cls: assms*)

```

lemma HideCSP-DoCSP [rdes-def]:
   $do_C(a) \setminus_C A = (Skip \triangleleft (a \in_u \ll A \gg) \triangleright_R do_C(a))$ 
  by (rdes-eq)

lemma HideCSP-PrefixCSP:
  assumes P is NCSP
  shows  $(a \rightarrow_C P) \setminus_C A = ((P \setminus_C A) \triangleleft (a \in_u \ll A \gg) \triangleright_R (a \rightarrow_C (P \setminus_C A)))$ 
  apply (simp add: PrefixCSP-def Healthy-if HideCSP-seq HideCSP-DoCSP closure assms rdes rpred)
  apply (simp add: HideCSP-NCSP-closed Skip-left-unit assms cond-st-distr)
  done

lemma HideCSP-empty:
  assumes P is NCSP
  shows  $P \setminus_C \{\} = P$ 
  by (rdes-eq cls: assms)

lemma HideCSP-twice:
  assumes P is NCSP
  shows  $P \setminus_C A \setminus_C B = P \setminus_C (A \cup B)$ 
  by (rdes-simp cls: assms)

lemma HideCSP-Skip:  $Skip \setminus_C A = Skip$ 
  by (rdes-eq)

lemma HideCSP-Stop:  $Stop \setminus_C A = Stop$ 
  by (rdes-eq)

end

```

## 6 Meta theory for Circus

```

theory utp-circus
imports
  utp-circus-traces
  utp-circus-parallel
  utp-circus-hiding
begin end

```

## References

- [1] S. Foster, F. Zeyda, and J. Woodcock. Unifying heterogeneous state-spaces with lenses. In *ICTAC*, LNCS 9965. Springer, 2016.
- [2] M. V. M. Oliveira. *Formal Derivation of State-Rich Reactive Programs using Circus*. PhD thesis, Department of Computer Science - University of York, UK, 2006. YCST-2006-02.