

Circus in Isabelle/UTP

Simon Foster James Baxter Ana Cavalcanti Jim Woodcock
Samuel Canham

March 15, 2019

Contents

1	Introduction	1
2	Circus Trace Merge	1
2.1	Function Definition	2
2.2	Lifted Trace Merge	2
2.3	Trace Merge Lemmas	2
3	Syntax and Translations for Event Prefix	3
4	Circus Parallel Composition	6
4.1	Merge predicates	6
4.2	Parallel operator	20
4.3	Parallel Laws	22
5	Hiding	33
5.1	Hiding in peri- and postconditions	33
5.2	Hiding in preconditions	34
5.3	Hiding Operator	35
6	Meta theory for Circus	37
7	Easy to use Circus-M parser	37

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 ::
  '∅ set ⇒ '∅ list ⇒ '∅ list ⇒ '∅ list set where
tr-par cs [] = {} |
tr-par cs (e # t) [] = (if e ∈ cs then {} else {[e]} ∩ (tr-par cs t [])) |
tr-par cs [] (e # t) = (if e ∈ cs then {} else {[e]} ∩ (tr-par cs [] t)) |
tr-par cs (e1 # t1) (e2 # t2) =
  (if e1 = e2
   then
    if e1 ∈ cs
    then {[e1]} ∩ (tr-par cs t1 t2)
    else
      ({[e1]} ∩ (tr-par cs t1 (e2 # t2))) ∪
      ({[e2]} ∩ (tr-par cs (e1 # t1) t2))
  else
    if e1 ∈ cs then
      if e2 ∈ cs then {}
      else
        {[e2]} ∩ (tr-par cs (e1 # t1) t2)
    else
      if e2 ∈ cs then
        {[e1]} ∩ (tr-par cs t1 (e2 # t2))
      else
        {[e1]} ∩ (tr-par cs t1 (e2 # t2)) ∪
        {[e2]} ∩ (tr-par cs (e1 # t1) t2)

```

abbreviation *tr-inter* :: '∅ list ⇒ '∅ list ⇒ '∅ list set (**infixr** |||_t 100) **where**
x |||_t *y* ≡ *tr-par* {} *x y*

2.2 Lifted Trace Merge

```

syntax -utr-par ::
  logic ⇒ logic ⇒ logic ⇒ logic ((- ★-/ -) [100, 0, 101] 100)

```

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

translations

t1 ★_{cs} *t2* == (CONST *bop*) (CONST *tr-par* cs) *t1 t2*

2.3 Trace Merge Lemmas

lemma *tr-par-empty*:

```

tr-par cs t1 [] = {takeWhile (λx. x ∉ cs) t1}
tr-par cs [] t2 = {takeWhile (λx. x ∉ cs) t2}
— Subgoal 1
apply (induct t1; simp)
— Subgoal 2
apply (induct t2; simp)
done

```

lemma *tr-par-sym*:

```

tr-par cs t1 t2 = tr-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
   $a \rightarrow P == CONST PrefixCSP \ll a \gg P$ 

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* :: $id \Rightarrow (' \sigma, ' \varepsilon) \text{ action} \Rightarrow \text{prefix-elem}' (?'(- : / -'))$

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

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

syntax *-output-prefix* :: $uexp \Rightarrow \text{prefix-elem}' (!'(-'))$

syntax *-output-prefix* :: $uexp \Rightarrow \text{prefix-elem}' (.'(-'))$

syntax (**output**) *-output-prefix-pp* :: $uexp \Rightarrow \text{prefix-elem}' (!'(-'))$

syntax

-prefix-aux :: $pttrn \Rightarrow \text{logic} \Rightarrow \text{prefix-elem}'$

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

syntax *-mixed-prefix* :: $\text{prefix-elem}' \Rightarrow \text{mixed-prefix}' \Rightarrow \text{mixed-prefix}' (--)$

syntax

-prefix-action ::

$('a, ' \varepsilon) \text{ chan} \Rightarrow \text{mixed-prefix}' \Rightarrow (' \sigma, ' \varepsilon) \text{ action} \Rightarrow (' \sigma, ' \varepsilon) \text{ action}$

$((-- \rightarrow / -) [63, 63, 62] 62)$

Syntax translations

definition *lconj* :: $('a \Rightarrow ' \alpha \text{ upred}) \Rightarrow ('b \Rightarrow ' \alpha \text{ upred}) \Rightarrow ('a \times 'b \Rightarrow ' \alpha \text{ upred})$ (**infixr** \wedge_l 35)

where [*upred-defs*]: $(P \wedge_l Q) \equiv (\lambda (x,y). P \ x \wedge Q \ y)$

definition *outp-constraint* (**infix** $=_o$ 60) **where**

[*upred-defs*]: *outp-constraint* $v \equiv (\lambda x. \ll x \gg =_u v)$

translations

-simple-input-prefix $x \equiv \text{-input-prefix } x \text{ true}$

-mixed-prefix $(\text{-input-prefix } x \ P) (\text{-prefix-aux } y \ Q) \rightarrow$

-prefix-aux $(\text{-pattern } x \ y) ((\lambda x. P) \wedge_l Q)$

-mixed-prefix $(\text{-output-prefix } P) (\text{-prefix-aux } y \ Q) \rightarrow$

-prefix-aux $(\text{-pattern -idtdummy } y) ((\text{CONST outp-constraint } P) \wedge_l Q)$

-end-prefix $(\text{-input-prefix } x \ P) \rightarrow \text{-prefix-aux } x \ (\lambda x. P)$

-end-prefix $(\text{-output-prefix } P) \rightarrow \text{-prefix-aux -idtdummy } (\text{CONST outp-constraint } P)$

-prefix-action $c (\text{-prefix-aux } x \ P) \ A == (\text{CONST InputCSP}) \ c \ P \ (\lambda x. A)$

Basic print translations; more work needed

translations

-simple-input-prefix $x \leq \text{-input-prefix } x \text{ true}$

-output-prefix $v \leq \text{-prefix-aux } p \ (\text{CONST outp-constraint } v)$

-output-prefix $u (\text{-output-prefix } v)$

$\leq \text{-prefix-aux } p \ (\lambda(x1, y1). \text{CONST outp-constraint } u \ x2 \wedge \text{CONST outp-constraint } v \ y2)$

-input-prefix $x \ P \leq \text{-prefix-aux } v \ (\lambda x. P)$

$x!(v) \rightarrow P \leq \text{CONST OutputCSP } x \ v \ P$

term $x!(1)!(y) \rightarrow P$

term $x?(v) \rightarrow P$

term $x?(v:\text{false}) \rightarrow P$

term $x!(\langle 1 \rangle) \rightarrow P$

term $x?(v)!(1) \rightarrow P$

term $x!(\langle 1 \rangle)!(2)?(v:\text{true}) \rightarrow P$

Basic translations for state variable communications

syntax

$-csp-input-var :: logic \Rightarrow id \Rightarrow logic \Rightarrow logic \ (-?)'(-') [63, 0, 0] 62)$
 $-csp-inputu-var :: logic \Rightarrow id \Rightarrow logic \ (-?)'(-') [63, 0] 62)$
 $-csp-output-var :: logic \Rightarrow uexp \Rightarrow logic \ (-!)'(-') [63, 0] 62)$

term *OutputCSP*

translations

$c?(x:A) \rightarrow CONST\ InputVarCSP\ c\ x\ A$
 $c?(x) \rightarrow CONST\ InputVarCSP\ c\ x\ (\lambda\ x.\ true)$
 $c?(x:A) \leqslant CONST\ InputVarCSP\ c\ x\ (\lambda\ x'.\ A)$
 $c?(x) \leqslant c?(x:true)$
 $-csp-output-var\ c\ e == CONST\ DoCSP\ (c.e)_u$

lemma *outp-constraint-prod*:

$(outp-constraint\ \ll a \gg\ x \wedge outp-constraint\ \ll b \gg\ y) =$
 $outp-constraint\ \ll (a, b) \gg\ (x, y)$
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*]:

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

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

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

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

assumes $\bigwedge y. P(y)\ is\ RR$
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 $\bigwedge y. P(y)\ is\ RR$
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 is NCSP*

shows $x!(P) \rightarrow A = \text{OutputCSP } x \ P \ A$ (**is** ?lhs = ?rhs)

by (rdes-eq cls: assms)

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

$P \text{ is NCSP} \implies a.(\ll i \gg).(\ll j \gg) \rightarrow P = \text{OutputCSP } a \ll (i, j) \gg P$

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]:

$P \text{ is NCSP} \implies a.(\ll i \gg).(\ll j \gg).(\ll k \gg) \rightarrow P = \text{OutputCSP } a \ll (i, j, k) \gg P$

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

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

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

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

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

definition *CSPInterMerge* **where**

[upred-defs]: $\text{CSPInterMerge } P \ cs \ Q = (P \parallel_{(\exists \ \$st' \cdot N_C \ 0_L \ cs \ 0_L)} Q)$

definition *CSPFinalMerge* **where**

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

syntax

-cinter-merge :: $logic \Rightarrow logic \Rightarrow logic \Rightarrow logic \ (- \ll - \gg^I - [85, 0, 86] \ 86)$

-cfinal-merge :: logic \Rightarrow salpha \Rightarrow logic \Rightarrow salpha \Rightarrow logic \Rightarrow logic (- $\llbracket - \rrbracket^F$ - [85,0,0,0,86] 86)
 -wrC :: logic \Rightarrow logic \Rightarrow logic \Rightarrow logic (- wr[-]_C - [85,0,86] 86)

translations

-cinter-merge P cs Q == CONST CSPInterMerge P cs Q
 -cfinal-merge P $ns1$ cs $ns2$ Q == CONST CSPFinalMerge P $ns1$ cs $ns2$ Q
 -wrC P cs Q == P wr_R(N_C θ_L cs θ_L) Q

lemma CSPInnerMerge-R2m [closure]: N_C $ns1$ cs $ns2$ is R2m
 by (rel-auto)

lemma CSPInnerMerge-RDM [closure]: N_C $ns1$ 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 is R2m
 shows $(\exists \$ref' \cdot P)$ is R2m

proof -

have $R2m(\exists \$ref' \cdot R2m P) = (\exists \$ref' \cdot R2m P)$
 by (rel-auto)
 thus ?thesis
 by (metis Healthy-def' assms)

qed

lemma CSPInnerMerge-unrests [unrest]:
 $\$ok_{<} \# N_C$ $ns1$ cs $ns2$
 $\$wait_{<} \# N_C$ $ns1$ cs $ns2$
 by (rel-auto)+

lemma CSPInterMerge-RR-closed [closure]:
 assumes P is RR Q is RR
 shows $P \llbracket cs \rrbracket^I Q$ is RR
 by (simp add: CSPInterMerge-def parallel-RR-closed assms closure unrest)

lemma CSPInterMerge-unrest-ref [unrest]:
 assumes P is CRR Q is CRR
 shows $\$ref \# P \llbracket cs \rrbracket^I Q$

proof -

have $\$ref \# CRR(P) \llbracket cs \rrbracket^I CRR(Q)$
 by (rel-blast)
 thus ?thesis
 by (simp add: Healthy-if assms)

qed

lemma CSPInterMerge-unrest-st' [unrest]:
 $\$st' \# P \llbracket cs \rrbracket^I Q$
 by (rel-auto)

lemma CSPInterMerge-CRR-closed [closure]:
 assumes P is CRR Q is CRR
 shows $P \llbracket cs \rrbracket^I Q$ is CRR
 by (simp add: CRR-implies-RR CRR-intro CSPInterMerge-RR-closed CSPInterMerge-unrest-ref assms)

lemma CSPFinalMerge-RR-closed [closure]:
 assumes P is RR Q is RR

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

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

assumes P is CRR Q 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* :: $('α \implies 'σ) \Rightarrow 'ψ \text{ set} \Rightarrow ('β \implies 'σ) \Rightarrow (('σ, 'ψ) \text{ sfrd}) \text{ merge } (M_C)$ **where**
[upred-defs]: $M_C ns1 cs ns2 = M_R(N_C ns1 cs ns2) ;; Skip$

definition *CSPInterleave* :: $('α \implies 'σ) \Rightarrow ('β \implies 'σ) \Rightarrow (('σ, 'ψ) \text{ sfrd}) \text{ merge } (M_I)$ **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 tr₂' ref₀ tr₀' ref₀' tr₁' ref₁' tr' ref₂' tr_i' ref₃'*)

oops

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

lemma *CSPInterMerge-left-false* [*rpred*]: $false \llbracket cs \rrbracket^I P = false$
by (*rel-auto*)

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

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

lemma *CSPInnerMerge-commute*:

assumes $ns1 \bowtie ns2$

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

proof –

have $P \parallel_{N_C} ns1 \ cs \ ns2 \ Q = P \parallel_{swap_m} ;; N_C \ ns2 \ cs \ ns1 \ Q$
 by (simp add: assms lens-indep-sym swap-CSPInnerMerge)

also have $\dots = Q \parallel_{N_C} ns2 \ cs \ ns1 \ P$
 by (metis par-by-merge-commute-swap)

finally show ?thesis .

qed

lemma *CSPInterMerge-commute*:

$P \llbracket cs \rrbracket^I Q = Q \llbracket cs \rrbracket^I P$

proof –

have $P \llbracket cs \rrbracket^I Q = P \parallel_{\exists \$st' \cdot N_C \ 0_L \ cs \ 0_L} Q$
 by (simp add: CSPInterMerge-def)

also have $\dots = P \parallel_{\exists \$st' \cdot (swap_m ;; N_C \ 0_L \ cs \ 0_L)} Q$
 by (simp add: swap-CSPInnerMerge lens-indep-sym)

also have $\dots = P \parallel_{swap_m} ;; (\exists \$st' \cdot N_C \ 0_L \ cs \ 0_L) \ Q$
 by (simp add: seqr-exists-right)

also have $\dots = Q \parallel_{(\exists \$st' \cdot N_C \ 0_L \ cs \ 0_L)} P$
 by (simp add: par-by-merge-commute-swap[THEN sym])

also have $\dots = Q \llbracket cs \rrbracket^I P$
 by (simp add: CSPInterMerge-def)

finally show ?thesis .

qed

lemma *CSPFinalMerge-commute*:

assumes $ns1 \bowtie ns2$

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

proof –

have $P \llbracket ns1|cs|ns2 \rrbracket^F Q = P \parallel_{\exists \$ref' \cdot N_C \ ns1 \ cs \ ns2} Q$
 by (simp add: CSPFinalMerge-def)

also have $\dots = P \parallel_{\exists \$ref' \cdot (swap_m ;; N_C \ ns2 \ cs \ ns1)} Q$
 by (simp add: swap-CSPInnerMerge lens-indep-sym assms)

also have $\dots = P \parallel_{swap_m} ;; (\exists \$ref' \cdot N_C \ ns2 \ cs \ ns1) \ Q$
 by (simp add: seqr-exists-right)

also have $\dots = Q \parallel_{(\exists \$ref' \cdot N_C \ ns2 \ cs \ ns1)} P$
 by (simp add: par-by-merge-commute-swap[THEN sym])

also have $\dots = Q \llbracket ns2|cs|ns1 \rrbracket^F P$
 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 *vwb-lens ns1 vwb-lens ns2 P is RR Q is RR*
shows
 $P \parallel_{N_C}^{ns1\ cs\ ns2} Q =$
 $(\exists (ref_0, ref_1, st_0, st_1, tt_0, tt_1) \cdot$
 $P[\langle\langle ref_0 \rangle\rangle, \langle\langle st_0 \rangle\rangle, \langle\langle tt_0 \rangle\rangle / \$ref', \$st', \$tr, \$tr'] \wedge Q[\langle\langle ref_1 \rangle\rangle, \langle\langle st_1 \rangle\rangle, \langle\langle tt_1 \rangle\rangle / \$ref', \$st', \$tr, \$tr']$
 $\wedge \$ref' \subseteq_u ((\langle\langle ref_0 \rangle\rangle \cup_u \langle\langle ref_1 \rangle\rangle) \cap_u \langle\langle cs \rangle\rangle) \cup_u ((\langle\langle ref_0 \rangle\rangle \cap_u \langle\langle ref_1 \rangle\rangle) - \langle\langle cs \rangle\rangle)$
 $\wedge \$tr \leq_u \tr'
 $\wedge \&tt \in_u \langle\langle tt_0 \rangle\rangle \star_{cs} \langle\langle tt_1 \rangle\rangle$
 $\wedge \langle\langle tt_0 \rangle\rangle \upharpoonright_u \langle\langle cs \rangle\rangle =_u \langle\langle tt_1 \rangle\rangle \upharpoonright_u \langle\langle cs \rangle\rangle$
 $\wedge \$st' =_u (\$st \oplus \langle\langle st_0 \rangle\rangle\ on\ \&ns1) \oplus \langle\langle st_1 \rangle\rangle\ on\ \&ns2)$
(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 $?P' \parallel_{N_C}^{ns1\ cs\ ns2} ?Q' =$
 $(\exists (ref_0, ref_1, st_0, st_1, tt_0, tt_1) \cdot$
 $?P'[\langle\langle ref_0 \rangle\rangle, \langle\langle st_0 \rangle\rangle, \langle\langle tt_0 \rangle\rangle / \$ref', \$st', \$tr, \$tr'] \wedge ?Q'[\langle\langle ref_1 \rangle\rangle, \langle\langle st_1 \rangle\rangle, \langle\langle tt_1 \rangle\rangle / \$ref', \$st', \$tr, \$tr']$
 $\wedge \$ref' \subseteq_u ((\langle\langle ref_0 \rangle\rangle \cup_u \langle\langle ref_1 \rangle\rangle) \cap_u \langle\langle cs \rangle\rangle) \cup_u ((\langle\langle ref_0 \rangle\rangle \cap_u \langle\langle ref_1 \rangle\rangle) - \langle\langle cs \rangle\rangle)$
 $\wedge \$tr \leq_u \tr'
 $\wedge \&tt \in_u \langle\langle tt_0 \rangle\rangle \star_{cs} \langle\langle tt_1 \rangle\rangle$
 $\wedge \langle\langle tt_0 \rangle\rangle \upharpoonright_u \langle\langle cs \rangle\rangle =_u \langle\langle tt_1 \rangle\rangle \upharpoonright_u \langle\langle cs \rangle\rangle$
 $\wedge \$st' =_u (\$st \oplus \langle\langle st_0 \rangle\rangle\ on\ \&ns1) \oplus \langle\langle st_1 \rangle\rangle\ on\ \&ns2)$
apply (*simp add: par-by-merge-alt-def, rel-auto, blast*)
apply (*rename-tac ok wait tr st ref tr' ref' ref_0 ref_1 st_0 st_1 tr_0 ok_0 tr_1 wait_0 ok_1 wait_1*)
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_0 in exI*)
apply (*rule-tac x=st_0 in exI*)
apply (*rule-tac x=ref_0 in exI*)
apply (*auto*)
apply (*metis Prefix-Order.prefixI append-minus*)
done
thus *?thesis*
by (*simp add: P Q*)
qed

lemma *CSPInterMerge-form:*

fixes $P\ Q :: ('σ, 'φ)\ action$

assumes *P is RR Q is RR*

shows

$P \parallel_{cs}^I Q =$

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

(is ?lhs = ?rhs)

proof –

have ?lhs = $(\exists \$st' \cdot P \parallel_{N_C} 0_L \text{ cs } 0_L Q)$
by (simp add: CSPInterMerge-def par-by-merge-def seqr-exists-right)
also have ... =
 $(\exists \$st' \cdot$
 $(\exists (ref_0, ref_1, st_0, st_1, tt_0, tt_1) \cdot$
 $P[\langle\langle ref_0 \rangle\rangle, \langle\langle st_0 \rangle\rangle, \langle\rangle, \langle\langle tt_0 \rangle\rangle / \$ref', \$st', \$tr, \$tr'] \wedge Q[\langle\langle ref_1 \rangle\rangle, \langle\langle st_1 \rangle\rangle, \langle\rangle, \langle\langle tt_1 \rangle\rangle / \$ref', \$st', \$tr, \$tr']$
 $\wedge \$ref' \subseteq_u ((\langle\langle ref_0 \rangle\rangle \cup_u \langle\langle ref_1 \rangle\rangle) \cap_u \langle\langle cs \rangle\rangle) \cup_u ((\langle\langle ref_0 \rangle\rangle \cap_u \langle\langle ref_1 \rangle\rangle) - \langle\langle cs \rangle\rangle)$
 $\wedge \$tr \leq_u \tr'
 $\wedge \&tt \in_u \langle\langle tt_0 \rangle\rangle \star_{cs} \langle\langle tt_1 \rangle\rangle$
 $\wedge \langle\langle tt_0 \rangle\rangle \upharpoonright_u \langle\langle cs \rangle\rangle =_u \langle\langle tt_1 \rangle\rangle \upharpoonright_u \langle\langle cs \rangle\rangle$
 $\wedge \$st' =_u (\$st \oplus \langle\langle st_0 \rangle\rangle \text{ on } \emptyset) \oplus \langle\langle st_1 \rangle\rangle \text{ on } \emptyset))$
by (simp add: CSPInnerMerge-form pr-var-def 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

$(P \llbracket ns1 | cs | ns2 \rrbracket^F Q) =$
 $(\exists (st_0, st_1, tt_0, tt_1) \cdot$
 $P[\langle\langle st_0 \rangle\rangle, \langle\rangle, \langle\langle tt_0 \rangle\rangle / \$st', \$tr, \$tr'] \wedge Q[\langle\langle st_1 \rangle\rangle, \langle\rangle, \langle\langle tt_1 \rangle\rangle / \$st', \$tr, \$tr']$
 $\wedge \$tr \leq_u \tr'
 $\wedge \&tt \in_u \langle\langle tt_0 \rangle\rangle \star_{cs} \langle\langle tt_1 \rangle\rangle$
 $\wedge \langle\langle tt_0 \rangle\rangle \upharpoonright_u \langle\langle cs \rangle\rangle =_u \langle\langle tt_1 \rangle\rangle \upharpoonright_u \langle\langle cs \rangle\rangle$
 $\wedge \$st' =_u (\$st \oplus \langle\langle st_0 \rangle\rangle \text{ on } \&ns1) \oplus \langle\langle st_1 \rangle\rangle \text{ on } \&ns2)$

(is ?lhs = ?rhs)

proof –

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

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

also have ... =

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

by (simp add: CSPInnerMerge-form assms)

also have ... =

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

by (simp add: ex-unrest assms)

also have ... =

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

$\wedge \$tr \leq_u \tr'
 $\wedge \&tt \in_u \langle\langle tt_0 \rangle\rangle \star_{cs} \langle\langle tt_1 \rangle\rangle$
 $\wedge \langle\langle tt_0 \rangle\rangle \vdash_u \langle\langle cs \rangle\rangle =_u \langle\langle tt_1 \rangle\rangle \vdash_u \langle\langle cs \rangle\rangle$
 $\wedge \$st' =_u (\$st \oplus \langle\langle st_0 \rangle\rangle \text{ on } \&ns1) \oplus \langle\langle st_1 \rangle\rangle \text{ on } \&ns2)$
 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*:
 $P \ wr[cs]_C \ Q = (\neg_r ((\neg_r \ Q) ;; U0 \wedge P ;; U1 \wedge \$st_{<}' =_u \$st \wedge \$tr_{<}' =_u \$tr) ;; N_C \ 0_L \ cs \ 0_L ;; R1 \ true)$
 by (rel-auto, metis+)

lemma *csp-wrR-ns-irr*:
 $(P \ wr_R(N_C \ ns1 \ cs \ ns2) \ Q) = (P \ wr[cs]_C \ Q)$
 by (rel-auto)

lemma *csp-wrR-CRC-closed* [closure]:
 assumes $P \text{ is } CRR \ Q \text{ is } CRR$
 shows $P \ wr[cs]_C \ Q \text{ is } CRC$
proof –
 have $\$ref \ \# \ P \ wr[cs]_C \ Q$
 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]:
 $P \ \llbracket cs \rrbracket^I \ Q \text{ is } CDC$
 using le-less-trans by (rel-blast)

lemma *CSPInterMerge-alt-def*:
 $P \ \llbracket cs \rrbracket^I \ Q = (\exists \ \$st' \cdot P \ \parallel_{N_C \ 0_L \ cs \ 0_L} Q)$
 by (simp add: par-by-merge-def CSPInterMerge-def seqr-exists-right)

lemma *CSPFinalMerge-alt-def*:
 $P \ \llbracket ns1 | cs | ns2 \rrbracket^F \ Q = (\exists \ \$ref' \cdot P \ \parallel_{N_C \ ns1 \ cs \ ns2} Q)$
 by (simp add: par-by-merge-def CSPFinalMerge-def seqr-exists-right)

lemma *merge-csp-do-left*:
 assumes $vwb\text{-}lens \ ns1 \ vwb\text{-}lens \ ns2 \ ns1 \bowtie ns2 \ P \text{ is } RR$
 shows $\Phi(s_0, \sigma_0, t_0) \parallel_{N_C \ ns1 \ cs \ ns2} P =$
 $(\exists \ (ref_1, st_1, tt_1) \cdot$
 $\quad [s_0]_{S<} \wedge$
 $\quad [\$ref' \mapsto_s \langle\langle ref_1 \rangle\rangle, \$st' \mapsto_s \langle\langle st_1 \rangle\rangle, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \langle\langle tt_1 \rangle\rangle] \dagger P \wedge$
 $\quad \$ref' \subseteq_u \langle\langle cs \rangle\rangle \cup_u (\langle\langle ref_1 \rangle\rangle - \langle\langle cs \rangle\rangle) \wedge$

$$[\llbracket \text{trace} \rrbracket \in_u t_0 \star_{cs} \llbracket tt_1 \rrbracket \wedge t_0 \upharpoonright_u \llbracket cs \rrbracket =_u \llbracket tt_1 \rrbracket \upharpoonright_u \llbracket cs \rrbracket]_t \wedge$$

$$\$st' =_u \$st \oplus \langle \sigma_0 \rangle (\$st)_a \text{ on } \&ns1 \oplus \langle st_1 \rangle \text{ on } \&ns2)$$
 (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 ... =

$$(\exists (ref_1, st_1, tt_1) \cdot$$

$$[s_0]_{S<} \wedge$$

$$[\$ref' \mapsto_s \langle ref_1 \rangle, \$st' \mapsto_s \langle st_1 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_1 \rangle] \dagger P \wedge$$

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

$$[\llbracket \text{trace} \rrbracket \in_u t_0 \star_{cs} \llbracket tt_1 \rrbracket \wedge t_0 \upharpoonright_u \langle cs \rangle =_u \langle tt_1 \rangle \upharpoonright_u \langle cs \rangle]_t \wedge$$

$$\$st' =_u \$st \oplus \langle \sigma_0 \rangle (\$st)_a \text{ on } \&ns1 \oplus \langle st_1 \rangle \text{ on } \&ns2)$$
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 $P \parallel_{N_C} ns1 \text{ cs } ns2 \Phi(s_1, \sigma_1, t_1) =$

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

$$[\$ref' \mapsto_s \langle ref_0 \rangle, \$st' \mapsto_s \langle st_0 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_0 \rangle] \dagger P \wedge$$

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

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

$$[\llbracket \text{trace} \rrbracket \in_u \langle tt_0 \rangle \star_{cs} t_1 \wedge \langle tt_0 \rangle \upharpoonright_u \langle cs \rangle =_u t_1 \upharpoonright_u \langle cs \rangle]_t \wedge$$

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

proof –

have ?lhs = $\Phi(s_1, \sigma_1, t_1) \parallel_{N_C} ns2 \text{ cs } ns1 P$

by (simp add: CSPInnerMerge-commute assms)

also from assms **have** ... = ?rhs

apply (simp add: assms merge-csp-do-left lens-indep-sym)

apply (rel-auto)

using assms(3) lens-indep-comm tr-par-sym **apply** fastforce

using assms(3) lens-indep.lens-put-comm tr-par-sym **apply** fastforce

done

finally show ?thesis .

qed

lemma merge-csp-enable-right:

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

shows $P \parallel_{N_C} ns1 \text{ cs } ns2 \mathcal{E}(s_0, t_0, E_0) =$

$$(\exists (ref_0, ref_1, st_0, st_1, tt_0) \cdot$$

$$[s_0]_{S<} \wedge$$

$$[\$ref' \mapsto_s \langle ref_0 \rangle, \$st' \mapsto_s \langle st_0 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_0 \rangle] \dagger P \wedge$$

$$(\forall e \cdot \langle e \rangle \in_u [E_0]_{S<} \Rightarrow \langle e \rangle \notin_u \langle ref_1 \rangle) \wedge$$

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

$$[\llbracket \text{trace} \rrbracket \in_u \langle tt_0 \rangle \star_{cs} t_0 \wedge \langle tt_0 \rangle \upharpoonright_u \langle cs \rangle =_u t_0 \upharpoonright_u \langle cs \rangle]_t \wedge$$

$\$st' =_u \$st \oplus \langle st_0 \rangle \text{ on } \&ns1 \oplus \langle st_1 \rangle \text{ on } \&ns2)$
 (is ?lhs = ?rhs)
proof –
 have ?lhs = $(\exists (ref_0, ref_1, st_0, st_1, tt_0, tt_1) \cdot$
 $[\$ref' \mapsto_s \langle ref_0 \rangle, \$st' \mapsto_s \langle st_0 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_0 \rangle] \dagger P \wedge$
 $[\$ref' \mapsto_s \langle ref_1 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_1 \rangle] \dagger \mathcal{E}(s_0, t_0, E_0) \wedge$
 $\$ref' \subseteq_u (\langle ref_0 \rangle \cup_u \langle ref_1 \rangle) \cap_u \langle cs \rangle \cup_u (\langle ref_0 \rangle \cap_u \langle ref_1 \rangle - \langle cs \rangle) \wedge$
 $\$tr \leq_u \$tr' \wedge \&tt \in_u \langle tt_0 \rangle \star_{cs} \langle tt_1 \rangle \wedge \langle tt_0 \rangle \upharpoonright_u \langle cs \rangle =_u \langle tt_1 \rangle \upharpoonright_u \langle cs \rangle \wedge \$st' =_u \$st$
 $\oplus \langle st_0 \rangle \text{ on } \&ns1 \oplus \langle st_1 \rangle \text{ on } \&ns2)$
 by (simp add: CSPInnerMerge-form assms closure unrest usubst)
 also have ... = $(\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'
 $\mapsto_s \langle tt_0 \rangle] \dagger P \wedge$
 $([s_0]_{S<} \wedge \langle tt_1 \rangle =_u [t_0]_{S<} \wedge (\forall e \cdot \langle e \rangle \in_u [E_0]_{S<} \Rightarrow \langle e \rangle \notin_u \langle ref_1 \rangle)) \wedge$
 $\$ref' \subseteq_u (\langle ref_0 \rangle \cup_u \langle ref_1 \rangle) \cap_u \langle cs \rangle \cup_u (\langle ref_0 \rangle \cap_u \langle ref_1 \rangle - \langle cs \rangle) \wedge$
 $\$tr \leq_u \$tr' \wedge \&tt \in_u \langle tt_0 \rangle \star_{cs} \langle tt_1 \rangle \wedge \langle tt_0 \rangle \upharpoonright_u \langle cs \rangle =_u \langle tt_1 \rangle \upharpoonright_u \langle cs \rangle \wedge \$st' =_u \$st$
 $\oplus \langle st_0 \rangle \text{ on } \&ns1 \oplus \langle st_1 \rangle \text{ on } \&ns2)$
 by (simp add: csp-enable-def usubst unrest)
 also have ... = $(\exists (ref_0, ref_1, st_0, st_1, tt_0) \cdot$
 $[s_0]_{S<} \wedge$
 $[\$ref' \mapsto_s \langle ref_0 \rangle, \$st' \mapsto_s \langle st_0 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_0 \rangle] \dagger P \wedge$
 $(\forall e \cdot \langle e \rangle \in_u [E_0]_{S<} \Rightarrow \langle e \rangle \notin_u \langle ref_1 \rangle) \wedge$
 $\$ref' \subseteq_u (\langle ref_0 \rangle \cup_u \langle ref_1 \rangle) \cap_u \langle cs \rangle \cup_u (\langle ref_0 \rangle \cap_u \langle ref_1 \rangle - \langle cs \rangle) \wedge$
 $[\langle trace \rangle \in_u \langle tt_0 \rangle \star_{cs} t_0 \wedge \langle tt_0 \rangle \upharpoonright_u \langle cs \rangle =_u t_0 \upharpoonright_u \langle cs \rangle]_t \wedge$
 $\$st' =_u \$st \oplus \langle st_0 \rangle \text{ on } \&ns1 \oplus \langle st_1 \rangle \text{ on } \&ns2)$
 by (rel-blast)
 finally show ?thesis .
qed

lemma merge-csp-enable-left:

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

shows $\mathcal{E}(s_0, t_0, E_0) \parallel_{N_C} ns1 \text{ cs } ns2 \ P =$

$(\exists (ref_0, ref_1, st_0, st_1, tt_0) \cdot$
 $[s_0]_{S<} \wedge$
 $[\$ref' \mapsto_s \langle ref_0 \rangle, \$st' \mapsto_s \langle st_1 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_0 \rangle] \dagger P \wedge$
 $(\forall e \cdot \langle e \rangle \in_u [E_0]_{S<} \Rightarrow \langle e \rangle \notin_u \langle ref_1 \rangle) \wedge$
 $\$ref' \subseteq_u (\langle ref_0 \rangle \cup_u \langle ref_1 \rangle) \cap_u \langle cs \rangle \cup_u (\langle ref_0 \rangle \cap_u \langle ref_1 \rangle - \langle cs \rangle) \wedge$
 $[\langle trace \rangle \in_u t_0 \star_{cs} \langle tt_0 \rangle \wedge \langle tt_0 \rangle \upharpoonright_u \langle cs \rangle =_u t_0 \upharpoonright_u \langle cs \rangle]_t \wedge$
 $\$st' =_u \$st \oplus \langle st_0 \rangle \text{ on } \&ns1 \oplus \langle st_1 \rangle \text{ on } \&ns2)$

(is ?lhs = ?rhs)

proof –

have ?lhs = $P \parallel_{N_C} ns2 \text{ cs } ns1 \ \mathcal{E}(s_0, t_0, E_0)$

by (simp add: CSPInnerMerge-commute assms)

also from assms **have** ... = ?rhs

apply (simp add: merge-csp-enable-right assms(4) lens-indep-sym)

apply (rel-auto)

oops

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 $\$ref' \not\# P$

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

$(\exists (st_1, t_1) \cdot$
 $[s_0]_{S<} \wedge$
 $[\$st' \mapsto_s \langle st_1 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle t_1 \rangle] \dagger P \wedge$

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

$$\$st' =_u \$st \oplus \llbracket \sigma_0 \rrbracket (\$st)_a \text{ on } \&ns1 \oplus \llbracket st_1 \rrbracket \text{ on } \&ns2)$$
 (is ?lhs = ?rhs)

proof –

have ?lhs =

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

$$[\$st' \mapsto_s \llbracket st_0 \rrbracket, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \llbracket tt_0 \rrbracket] \dagger \Phi(s_0, \sigma_0, t_0) \wedge$$

$$[\$st' \mapsto_s \llbracket st_1 \rrbracket, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \llbracket tt_1 \rrbracket] \dagger RR(\exists \$ref' \cdot P) \wedge$$

$$\$tr \leq_u \$tr' \wedge \&tt \in_u \llbracket tt_0 \rrbracket \star_{cs} \llbracket tt_1 \rrbracket \wedge \llbracket tt_0 \rrbracket \upharpoonright_u \llbracket cs \rrbracket =_u \llbracket tt_1 \rrbracket \upharpoonright_u \llbracket cs \rrbracket \wedge$$

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

also have ... =

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

$$[s_0]_{S<} \wedge$$

$$[\$st' \mapsto_s \llbracket st_1 \rrbracket, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \llbracket tt_1 \rrbracket] \dagger RR(\exists \$ref' \cdot P) \wedge$$

$$[\llbracket \text{trace} \rrbracket \in_u t_0 \star_{cs} \llbracket tt_1 \rrbracket \wedge t_0 \upharpoonright_u \llbracket cs \rrbracket =_u \llbracket tt_1 \rrbracket \upharpoonright_u \llbracket cs \rrbracket]_t \wedge$$

$$\$st' =_u \$st \oplus \llbracket \sigma_0 \rrbracket (\$st)_a \text{ on } \&ns1 \oplus \llbracket st_1 \rrbracket \text{ on } \&ns2)$$
 by (rel-blast)

also have ... =

$$(\exists (st_1, t_1) \cdot$$

$$[s_0]_{S<} \wedge$$

$$[\$st' \mapsto_s \llbracket st_1 \rrbracket, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \llbracket t_1 \rrbracket] \dagger P \wedge$$

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

$$\$st' =_u \$st \oplus \llbracket \sigma_0 \rrbracket (\$st)_a \text{ on } \&ns1 \oplus \llbracket st_1 \rrbracket \text{ on } \&ns2)$$
 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' \sharp P*

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

$$(\exists (st_0, t_0) \cdot$$

$$[\$st' \mapsto_s \llbracket st_0 \rrbracket, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \llbracket t_0 \rrbracket] \dagger P \wedge$$

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

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

$$\$st' =_u \$st \oplus \llbracket st_0 \rrbracket \text{ on } \&ns1 \oplus \llbracket \sigma_1 \rrbracket (\$st)_a \text{ on } \&ns2)$$
 (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-lens ns1 vwb-lens ns2 ns1 \bowtie ns2*

shows $\Phi(s_1, \sigma_1, t_1) \llbracket ns1 | cs | ns2 \rrbracket^F \Phi(s_2, \sigma_2, t_2) =$

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

proof –

have ?lhs =
 $(\exists (st_0, st_1, tt_0, tt_1) \cdot$
 $[\$st' \mapsto_s \langle st_0 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_0 \rangle] \dagger \Phi(s_1, \sigma_1, t_1) \wedge$
 $[\$st' \mapsto_s \langle st_1 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_1 \rangle] \dagger \Phi(s_2, \sigma_2, t_2) \wedge$
 $\$tr \leq_u \$tr' \wedge \<tt \in_u \langle tt_0 \rangle \star_{cs} \langle tt_1 \rangle \wedge \langle tt_0 \rangle \upharpoonright_u \langle cs \rangle =_u \langle tt_1 \rangle \upharpoonright_u \langle cs \rangle \wedge$
 $\$st' =_u \$st \oplus \langle st_0 \rangle \text{ on } \&ns1 \oplus \langle st_1 \rangle \text{ on } \&ns2)$
by (simp add: CSPFinalMerge-form unrest closure assms)
also have ... =
 $([s_1 \wedge s_2]_{S<} \wedge [\langle trace \rangle \in_u t_1 \star_{cs} t_2 \wedge t_1 \upharpoonright_u \langle cs \rangle =_u t_2 \upharpoonright_u \langle cs \rangle]_t \wedge [\langle \sigma_1 [\&ns1 | \&ns2]_s \sigma_2 \rangle_a]_{S'})$
by (rel-auto)
finally show ?thesis .
qed

lemma FinalMerge-csp-do' [rpred]:
assumes vwb-lens ns1 vwb-lens ns2 ns1 \bowtie ns2
shows $\Phi(s_1, \sigma_1, t_1) \llbracket ns1 | cs | ns2 \rrbracket^F \Phi(s_2, \sigma_2, t_2) =$
 $(\exists trace \cdot \Phi(s_1 \wedge s_2 \wedge \langle trace \rangle \in_u t_1 \star_{cs} t_2 \wedge t_1 \upharpoonright_u \langle cs \rangle =_u t_2 \upharpoonright_u \langle cs \rangle, \sigma_1 [\&ns1 | \&ns2]_s \sigma_2,$
 $\langle trace \rangle))$
by (simp add: FinalMerge-csp-do assms, rel-auto)

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

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

lemma InterMerge-csp-enable-left:
assumes P is RR $\$st' \nVdash P$
shows $\mathcal{E}(s_0, t_0, E_0) \llbracket cs \rrbracket^I P =$
 $(\exists (ref_0, ref_1, t_1) \cdot$
 $[s_0]_{S<} \wedge (\forall e \cdot \langle e \rangle \in_u [E_0]_{S<} \Rightarrow \langle e \rangle \notin_u \langle ref_0 \rangle) \wedge$
 $[\$ref' \mapsto_s \langle ref_1 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle t_1 \rangle] \dagger P \wedge$
 $\$ref' \subseteq_u (\langle ref_0 \rangle \cup_u \langle ref_1 \rangle) \cap_u \langle cs \rangle \cup_u (\langle ref_0 \rangle \cap_u \langle ref_1 \rangle - \langle cs \rangle) \wedge$
 $[\langle trace \rangle \in_u t_0 \star_{cs} \langle t_1 \rangle \wedge t_0 \upharpoonright_u \langle cs \rangle =_u \langle t_1 \rangle \upharpoonright_u \langle cs \rangle]_t)$
(is ?lhs = ?rhs)
apply (simp add: CSPInterMerge-form ex-unrest Healthy-if unrest closure assms usubst)
apply (simp add: csp-enable-def usubst unrest assms closure)
apply (rel-auto)
done

lemma InterMerge-csp-enable:
 $\mathcal{E}(s_1, t_1, E_1) \llbracket cs \rrbracket^I \mathcal{E}(s_2, t_2, E_2) =$
 $([s_1 \wedge s_2]_{S<} \wedge$
 $(\forall e \in [(E_1 \cap_u E_2 \cap_u \langle cs \rangle) \cup_u ((E_1 \cup_u E_2) - \langle cs \rangle)]_{S<} \cdot \langle e \rangle \notin_u \$ref') \wedge$
 $[\langle trace \rangle \in_u t_1 \star_{cs} t_2 \wedge t_1 \upharpoonright_u \langle cs \rangle =_u t_2 \upharpoonright_u \langle cs \rangle]_t)$
(is ?lhs = ?rhs)

proof –
have ?lhs =
 $(\exists (ref_0, ref_1, st_0, st_1, tt_0, tt_1) \cdot$
 $[\$ref' \mapsto_s \langle ref_0 \rangle, \$st' \mapsto_s \langle st_0 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_0 \rangle] \dagger \mathcal{E}(s_1, t_1, E_1) \wedge$
 $[\$ref' \mapsto_s \langle ref_1 \rangle, \$st' \mapsto_s \langle st_1 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_1 \rangle] \dagger \mathcal{E}(s_2, t_2, E_2) \wedge$

$\$ref' \subseteq_u (\ll ref_0 \gg \cup_u \ll ref_1 \gg) \cap_u \ll cs \gg \cup_u (\ll ref_0 \gg \cap_u \ll ref_1 \gg - \ll cs \gg) \wedge$
 $\$tr \leq_u \$tr' \wedge \&tt \in_u \ll tt_0 \gg \star_{cs} \ll tt_1 \gg \wedge \ll tt_0 \gg \downarrow_u \ll cs \gg =_u \ll tt_1 \gg \downarrow_u \ll cs \gg)$
by (*simp add: CSPInterMerge-form unrest closure*)
also have ... =
 $(\exists (ref_0, ref_1, tt_0, tt_1) \cdot$
 $[\$ref' \mapsto_s \ll ref_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll tt_0 \gg] \dagger \mathcal{E}(s_1, t_1, E_1) \wedge$
 $[\$ref' \mapsto_s \ll ref_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll tt_1 \gg] \dagger \mathcal{E}(s_2, t_2, E_2) \wedge$
 $\$ref' \subseteq_u (\ll ref_0 \gg \cup_u \ll ref_1 \gg) \cap_u \ll cs \gg \cup_u (\ll ref_0 \gg \cap_u \ll ref_1 \gg - \ll cs \gg) \wedge$
 $\$tr \leq_u \$tr' \wedge \&tt \in_u \ll tt_0 \gg \star_{cs} \ll tt_1 \gg \wedge \ll tt_0 \gg \downarrow_u \ll cs \gg =_u \ll tt_1 \gg \downarrow_u \ll cs \gg)$
by (*rel-auto*)
also have ... =
 $([s_1 \wedge s_2]_{S<} \wedge$
 $(\forall e \in [(E_1 \cap_u E_2 \cap_u \ll cs \gg) \cup_u ((E_1 \cup_u E_2) - \ll cs \gg)]_{S<} \cdot \ll e \gg \notin_u \$ref') \wedge$
 $[\ll trace \gg \in_u t_1 \star_{cs} t_2 \wedge t_1 \downarrow_u \ll cs \gg =_u t_2 \downarrow_u \ll cs \gg]_t$
 $)$
apply (*rel-auto*)
apply (*rename-tac tr st tr' ref'*)
apply (*rule-tac x =- [E1]_e st in exI*)
apply (*simp*)
apply (*rule-tac x =- [E2]_e st in exI*)
apply (*auto*)
done
finally show ?thesis .
qed

lemma *InterMerge-csp-enable'* [*rpred*]:
 $\mathcal{E}(s_1, t_1, E_1) \ll cs \gg^I \mathcal{E}(s_2, t_2, E_2) =$
 $(\exists trace \cdot \mathcal{E}(s_1 \wedge s_2 \wedge \ll trace \gg \in_u t_1 \star_{cs} t_2 \wedge t_1 \downarrow_u \ll cs \gg =_u t_2 \downarrow_u \ll cs \gg$
 $, \ll trace \gg$
 $, (E_1 \cap_u E_2 \cap_u \ll cs \gg) \cup_u ((E_1 \cup_u E_2) - \ll cs \gg)))$
by (*simp add: InterMerge-csp-enable, rel-auto*)

lemma *InterMerge-csp-enable-csp-do* [*rpred*]:
 $\mathcal{E}(s_1, t_1, E_1) \ll cs \gg^I \Phi(s_2, \sigma_2, t_2) =$
 $(\exists trace \cdot \mathcal{E}(s_1 \wedge s_2 \wedge \ll trace \gg \in_u t_1 \star_{cs} t_2 \wedge t_1 \downarrow_u \ll cs \gg =_u t_2 \downarrow_u \ll cs \gg, \ll trace \gg, E_1 - \ll cs \gg))$
(is ?lhs = ?rhs)
proof –
have ?lhs =
 $(\exists (ref_0, ref_1, st_0, st_1, tt_0, tt_1) \cdot$
 $[\$ref' \mapsto_s \ll ref_0 \gg, \$st' \mapsto_s \ll st_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll tt_0 \gg] \dagger \mathcal{E}(s_1, t_1, E_1) \wedge$
 $[\$ref' \mapsto_s \ll ref_1 \gg, \$st' \mapsto_s \ll st_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll tt_1 \gg] \dagger \Phi(s_2, \sigma_2, t_2) \wedge$
 $\$ref' \subseteq_u (\ll ref_0 \gg \cup_u \ll ref_1 \gg) \cap_u \ll cs \gg \cup_u (\ll ref_0 \gg \cap_u \ll ref_1 \gg - \ll cs \gg) \wedge$
 $\$tr \leq_u \$tr' \wedge \&tt \in_u \ll tt_0 \gg \star_{cs} \ll tt_1 \gg \wedge \ll tt_0 \gg \downarrow_u \ll cs \gg =_u \ll tt_1 \gg \downarrow_u \ll cs \gg)$
by (*simp add: CSPInterMerge-form unrest closure*)
also have ... =
 $(\exists (ref_0, ref_1, tt_0) \cdot$
 $[\$ref' \mapsto_s \ll ref_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll tt_0 \gg] \dagger \mathcal{E}(s_1, t_1, E_1) \wedge$
 $[s_2]_{S<} \wedge$
 $\$ref' \subseteq_u (\ll ref_0 \gg \cup_u \ll ref_1 \gg) \cap_u \ll cs \gg \cup_u (\ll ref_0 \gg \cap_u \ll ref_1 \gg - \ll cs \gg) \wedge$
 $[\ll trace \gg \in_u t_1 \star_{cs} t_2 \wedge t_1 \downarrow_u \ll cs \gg =_u t_2 \downarrow_u \ll cs \gg]_t)$
by (*rel-auto*)
also have ... = $([s_1 \wedge s_2]_{S<} \wedge (\forall e \in [(E_1 - \ll cs \gg)]_{S<} \cdot \ll e \gg \notin_u \$ref') \wedge$
 $[\ll trace \gg \in_u t_1 \star_{cs} t_2 \wedge t_1 \downarrow_u \ll cs \gg =_u t_2 \downarrow_u \ll cs \gg]_t)$
by (*rel-auto*)
also have ... = $(\exists trace \cdot \mathcal{E}(s_1 \wedge s_2 \wedge \ll trace \gg \in_u t_1 \star_{cs} t_2 \wedge t_1 \downarrow_u \ll cs \gg =_u t_2 \downarrow_u \ll cs \gg, \ll trace \gg,$

$E_1 - \ll cs \gg$)
 by (rel-auto)
 finally show ?thesis .
 qed

lemma *InterMerge-csp-do-csp-enable* [rpred]:

$\Phi(s_1, \sigma_1, t_1) \ll cs \gg^I \mathcal{E}(s_2, t_2, E_2) =$
 $(\exists \text{ trace} \cdot \mathcal{E}(s_1 \wedge s_2 \wedge \ll \text{trace} \gg \in_u t_1 \star_{cs} t_2 \wedge t_1 \upharpoonright_u \ll cs \gg =_u t_2 \upharpoonright_u \ll cs \gg, \ll \text{trace} \gg, E_2 - \ll cs \gg))$
 (is ?lhs = ?rhs)

proof –

have $\Phi(s_1, \sigma_1, t_1) \ll cs \gg^I \mathcal{E}(s_2, t_2, E_2) = \mathcal{E}(s_2, t_2, E_2) \ll cs \gg^I \Phi(s_1, \sigma_1, t_1)$
 by (simp add: CSPInterMerge-commute)
 also have ... = ?rhs
 by (simp add: rpred trace-merge-commute eq-upred-sym, rel-auto)
 finally show ?thesis .

qed

lemma *CSPInterMerge-or-left* [rpred]:

$(P \vee Q) \ll cs \gg^I R = (P \ll cs \gg^I R \vee Q \ll cs \gg^I R)$
 by (simp add: CSPInterMerge-def par-by-merge-or-left)

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

$P \ll cs \gg^I (Q \vee R) = (P \ll cs \gg^I Q \vee P \ll cs \gg^I R)$
 by (simp add: CSPInterMerge-def par-by-merge-or-right)

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

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

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

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

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

$(\bigcap i \cdot P(i)) \ll cs \gg^I Q = (\bigcap i \cdot P(i) \ll cs \gg^I Q)$
 by (simp add: CSPInterMerge-def par-by-merge-USUP-ind-left)

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

$P \ll cs \gg^I (\bigcap i \cdot Q(i)) = (\bigcap i \cdot P \ll cs \gg^I Q(i))$
 by (simp add: CSPInterMerge-def par-by-merge-USUP-ind-right)

thm *UINF-is-exists*

lemma *CSPInterMerge-shEx-left* [rpred]:

$(\exists i \cdot P(i)) \ll cs \gg^I Q = (\exists i \cdot P(i) \ll cs \gg^I Q)$
 using CSPInterMerge-UINF-ind-left[of P cs Q]
 by (simp add: UINF-is-exists)

lemma *CSPInterMerge-shEx-right* [rpred]:

$P \ll cs \gg^I (\exists i \cdot Q(i)) = (\exists i \cdot P \ll cs \gg^I Q(i))$
 using CSPInterMerge-UINF-ind-right[of P cs Q]
 by (simp add: UINF-is-exists)

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 *utrace-leq*: $(x \leq_u y) = (\exists z \cdot y =_u x \hat{\cdot}_u \ll z \gg)$
by (*rel-auto*)

lemma *trace-pred-R1-true*: $[P(\text{trace})]_t \;; \; R1 \text{ true} = [(\exists tt_0 \cdot \ll tt_0 \gg \leq_u \ll \text{trace} \gg \wedge P(tt_0))]_t$
apply (*rel-auto*)
using *minus-cancel-le* **apply** *blast*
apply (*metis diff-add-cancel-left' le-add trace-class.add-diff-cancel-left trace-class.add-left-mono*)
done

lemma *wrC-csp-do-init* [*wp*]:
 $\Phi(s_1, \sigma_1, t_1) \text{ wr}[cs]_C \mathcal{I}(s_2, t_2) =$
 $(\forall (tt_0, tt_1) \cdot \mathcal{I}(s_1 \wedge s_2 \wedge \ll tt_1 \gg \in_u (t_2 \hat{\cdot}_u \ll tt_0 \gg) \star_{cs} t_1 \wedge t_2 \hat{\cdot}_u \ll tt_0 \gg \downarrow_u \ll cs \gg =_u t_1 \downarrow_u \ll cs \gg,$
 $\ll tt_1 \gg))$
(is ?lhs = ?rhs)
proof –
have *?lhs* =
 $(\neg_r (\exists (ref_0, st_0, tt_0) \cdot$
 $[\$ref' \mapsto_s \ll ref_0 \gg, \$st' \mapsto_s \ll st_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll tt_0 \gg] \dagger (\neg_r \mathcal{I}(s_2, t_2)) \wedge$
 $[s_1]_{S<} \wedge$
 $\$ref' \subseteq_u \ll cs \gg \cup_u (\ll ref_0 \gg - \ll cs \gg) \wedge$
 $[\ll \text{trace} \gg \in_u \ll tt_0 \gg \star_{cs} t_1 \wedge \ll tt_0 \gg \downarrow_u \ll cs \gg =_u t_1 \downarrow_u \ll cs \gg]_t \wedge$
 $\$st' =_u \$st) \;; \; R1 \text{ true})$
by (*simp add: wrR-def par-by-merge-seq-remove merge-csp-do-right pr-var-def closure Healthy-if rpred*)
also have ... =
 $(\neg_r (\exists tt_0 \cdot ([s_2]_{S<} \wedge [t_2]_{S<} \leq_u \ll tt_0 \gg) \wedge [s_1]_{S<} \wedge$
 $[\ll \text{trace} \gg \in_u \ll tt_0 \gg \star_{cs} t_1 \wedge \ll tt_0 \gg \downarrow_u \ll cs \gg =_u t_1 \downarrow_u \ll cs \gg]_t) \;; \; R1 \text{ true})$
by (*rel-auto*)
also have ... =
 $(\neg_r (\exists tt_0 \cdot ([s_2]_{S<} \wedge (\exists tt_1 \cdot \ll tt_0 \gg =_u [t_2]_{S<} \hat{\cdot}_u \ll tt_1 \gg)) \wedge [s_1]_{S<} \wedge$
 $[\ll \text{trace} \gg \in_u \ll tt_0 \gg \star_{cs} t_1 \wedge \ll tt_0 \gg \downarrow_u \ll cs \gg =_u t_1 \downarrow_u \ll cs \gg]_t) \;; \; R1 \text{ true})$
by (*simp add: utrace-leq*)
also have ... =
 $(\neg_r (\exists tt_1 \cdot [s_1 \wedge s_2 \wedge \ll \text{trace} \gg \in_u (t_2 \hat{\cdot}_u \ll tt_1 \gg) \star_{cs} t_1 \wedge t_2 \hat{\cdot}_u \ll tt_1 \gg \downarrow_u \ll cs \gg =_u t_1 \downarrow_u \ll cs \gg]_t)$
 $\;; \; R1 \text{ true})$
by (*rel-auto*)
also have ... =
 $(\forall tt_1 \cdot \neg_r ([s_1 \wedge s_2 \wedge \ll \text{trace} \gg \in_u (t_2 \hat{\cdot}_u \ll tt_1 \gg) \star_{cs} t_1 \wedge t_2 \hat{\cdot}_u \ll tt_1 \gg \downarrow_u \ll cs \gg =_u t_1 \downarrow_u \ll cs \gg]_t)$
 $\;; \; R1 \text{ true})$
by (*rel-auto*)
also have ... =
 $(\forall (tt_0, tt_1) \cdot \neg_r ([s_1 \wedge s_2 \wedge \ll tt_0 \gg \leq_u \ll \text{trace} \gg \wedge \ll tt_0 \gg \in_u (t_2 \hat{\cdot}_u \ll tt_1 \gg) \star_{cs} t_1 \wedge t_2 \hat{\cdot}_u \ll tt_1 \gg \downarrow_u$
 $\ll cs \gg =_u t_1 \downarrow_u \ll cs \gg]_t))$
by (*simp add: trace-pred-R1-true, rel-auto*)
also have ... = *?rhs*
by (*rel-auto*)
finally show *?thesis* .
qed

lemma *wrC-csp-enable-init* [*wp*]:
fixes $t_1 \ t_2 \:: \ ('a \text{ list}, 'b) \text{ uexpr}$
shows
 $\mathcal{E}(s_1, t_1, E_1) \text{ wr}[cs]_C \mathcal{I}(s_2, t_2) =$
 $(\forall (tt_0, tt_1) \cdot \mathcal{I}(s_1 \wedge s_2 \wedge \ll tt_1 \gg \in_u (t_2 \hat{\cdot}_u \ll tt_0 \gg) \star_{cs} t_1 \wedge t_2 \hat{\cdot}_u \ll tt_0 \gg \downarrow_u \ll cs \gg =_u t_1 \downarrow_u \ll cs \gg,$

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

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

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

proof –

have $1: \bigwedge P \ Q. P \text{ wr}_R(N_C \ ns1 \ cs \ ns2) \ Q = P \text{ wr}[cs]_C Q \bigwedge P \ Q. P \text{ wr}_R(N_C \ ns2 \ cs \ ns1) \ Q = P$

$\text{wr}[cs]_C Q$

by (*rel-auto*)

have $2: (\exists \$st' \cdot N_C \ ns1 \ cs \ ns2) = (\exists \$st' \cdot N_C \ 0_L \ cs \ 0_L)$

by (*rel-auto*)

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

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

$(P_1 \Rightarrow_r P_2) \text{ wr}[cs]_C Q_1 \wedge$

$(P_1 \Rightarrow_r P_3) \text{ wr}[cs]_C Q_1) \vdash$

$(P_2 \llbracket cs \rrbracket^I Q_2 \vee$

$P_3 \llbracket cs \rrbracket^I Q_2 \vee$

$P_2 \llbracket cs \rrbracket^I Q_3) \diamond$

$P_3 \parallel_{N_C \ ns1 \ cs \ ns2} Q_3) ;;_h Skip$

by (*simp add: parallel-rdes-def swap-CSPInnerMerge CSPInterMerge-def closure assms 1 2*)

also

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

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

$(P_1 \Rightarrow_r P_2) \text{ wr}[cs]_C Q_1 \wedge$

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

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 $(P \llbracket ns1 \parallel cs \parallel ns2 \rrbracket Q) = P \parallel_{\text{swap}_m} ; (M_C \text{ ns2 cs ns1}) Q$
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 $\text{CSP5}(P) = \text{C2}(P)$
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 $\text{C2}(P) = \mathbf{R}_s (pre_R P \vdash (\exists \text{ ref}_0 \cdot \text{peri}_R P \llbracket \llbracket \text{ref}_0 \rrbracket / \$\text{ref}' \rrbracket \wedge \$\text{ref}' \subseteq_u \llbracket \text{ref}_0 \rrbracket \rhd \text{post}_R P)$
proof –
have $1: \Phi(\text{true}, id, \langle \rangle) \text{ wr}[\{\}]_C pre_R P = pre_R P \text{ (is ?lhs = ?rhs)}$
proof –
have $?lhs = (\neg_r (\exists (\text{ref}_0, st_0, tt_0) \cdot$

$$[\$ref' \mapsto_s \llbracket \text{ref}_0 \rrbracket, \$st' \mapsto_s \llbracket st_0 \rrbracket, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \llbracket tt_0 \rrbracket] \dagger (\exists \$ref'; \$st' \cdot RR(\neg_r$$

$$pre_R P)) \wedge$$

$$\$ref' \subseteq_u \llbracket \text{ref}_0 \rrbracket \wedge [\llbracket \text{trace} \rrbracket =_u \llbracket tt_0 \rrbracket]_t \wedge$$

$$\$st' =_u \$st) ;; R1 \text{ true})$$

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 $2: (pre_R P \Rightarrow_r \text{peri}_R P) \llbracket \{\} \rrbracket^I \Phi(\text{true}, id, \langle \rangle) =$

$$(\exists \text{ ref}_0 \cdot (\text{peri}_R P) \llbracket \llbracket \text{ref}_0 \rrbracket / \$\text{ref}' \rrbracket \wedge \$\text{ref}' \subseteq_u \llbracket \text{ref}_0 \rrbracket) \text{ (is ?lhs = ?rhs)}$$

proof –
have $?lhs = \text{peri}_R P \llbracket \{\} \rrbracket^I \Phi(\text{true}, id, \langle \rangle)$
by (*simp add: SRD-peri-under-pre closure assms unrest*)
also have $\dots = (\exists \$st' \cdot (\text{peri}_R P \parallel_{N_C \ 0_L \ \{\}} \ 0_L \Phi(\text{true}, id, \langle \rangle)))$
by (*simp add: CSPInterMerge-def par-by-merge-def seqr-exists-right*)
also have $\dots =$

$$(\exists \$st' \cdot \exists (\text{ref}_0, st_0, tt_0) \cdot$$

$$[\$ref' \mapsto_s \llbracket \text{ref}_0 \rrbracket, \$st' \mapsto_s \llbracket st_0 \rrbracket, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \llbracket tt_0 \rrbracket] \dagger (\exists \$st' \cdot RR(\text{peri}_R P)) \wedge$$

$$\$ref' \subseteq_u \llbracket \text{ref}_0 \rrbracket \wedge [\llbracket \text{trace} \rrbracket =_u \llbracket tt_0 \rrbracket]_t \wedge \$st' =_u \$st)$$

by (*simp add: merge-csp-do-right pr-var-def assms Healthy-if closure rpred unrest ex-unrest*)
also have $\dots =$

$$(\exists \text{ ref}_0 \cdot (\exists \$st' \cdot RR(\text{peri}_R P)) \llbracket \llbracket \text{ref}_0 \rrbracket / \$\text{ref}' \rrbracket \wedge \$\text{ref}' \subseteq_u \llbracket \text{ref}_0 \rrbracket)$$

by (*rel-auto*)
also have $\dots = ?rhs$

by (simp add: closure ex-unrest Healthy-if unrest assms)
 finally show ?thesis .
 qed
 have 3: (pre_R P ⇒_r post_R P) [Σ|{}|∅]^F Φ(true, id, ⟨⟩) = post_R(P) (is ?lhs = ?rhs)
 proof –
 have ?lhs = post_R P [Σ|{}|∅]^F Φ(true, id, ⟨⟩)
 by (simp add: SRD-post-under-pre closure assms unrest)
 also have ... = (∃ (st₀, t₀) .
 [st' ↦_s <<st₀>>, tr ↦_s ⟨⟩, tr' ↦_s <<t₀>>] † RR(post_R P) ∧
 [<<trace>> =_u <<t₀>>]_t ∧ st' =_u st ⊕ <<st₀>> on Σ ⊕ <<id>>(\$st)_a on ∅)
 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) = R_s (Φ(true, id, ⟨⟩) wr[{}]_C pre_R P ⊢
 (pre_R P ⇒_r peri_R P) [{}]^I Φ(true, id, ⟨⟩) ◇ (pre_R P ⇒_r post_R P) [Σ|{}|∅]^F Φ(true, id, ⟨⟩))
 by (simp add: C2-def, rdes-simp cls: assms, simp add: id-def pr-var-def)
 also have ... = R_s (pre_R P ⊢ (∃ ref₀ . peri_R P [ref₀/\$ref'] ∧ \$ref' ⊆_u <<ref₀>> ◇ 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) = R_s (pre_R P ⊢ CDC(peri_R P) ◇ post_R P)
 by (simp add: C2-form assms CDC-def)

lemma C2-rdes-def:

assumes P₁ is CRC P₂ is CRR P₃ is CRR \$st' # P₂ \$ref' # P₃
 shows C2(R_s(P₁ ⊢ P₂ ◇ P₃)) = R_s(P₁ ⊢ CDC(P₂) ◇ P₃)
 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) = R_s (pre_R P ⊢ CDC(peri_R P) ◇ post_R P)
 by (simp add: C2-CDC-form assms(1))
 also have ... = R_s (pre_R P ⊢ peri_R P ◇ 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

lemma C2-rdes-intro:

assumes P₁ is CRC P₂ is CRR P₂ is CDC P₃ is CRR \$st' # P₂ \$ref' # P₃
 shows R_s(P₁ ⊢ P₂ ◇ P₃) 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 $P \parallel Q$ 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 $\text{CDC } P \oplus_r \text{map-st}_L[a]$ 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 $\text{vwb-lens } a$ 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 (*if_R i ∈ A · g(i) → P(i) else Q 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 (*if_R i ∈ A · g(i) → P(i) else Q 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
 $\bigwedge b P. (b, P) \in \text{set } A \implies P$ is C2 Q 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 $\bigwedge b P. (b, P) \in \text{set } A \implies P \text{ is CACT } Q \text{ is CACT}$
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 $P \text{ is NCSP } P \text{ is Productive } P \text{ is C2}$
shows *while_C b do P od is C2*

proof –

have *while_C b do P od = while_C b do Productive(**R_s** (*pre_R P* \vdash *peri_R P* \diamond *post_R P*)) od*
by (*simp add: assms Healthy-if SRD-reactive-tri-design closure*)
also have $\dots = \text{while}_C b \text{ do } \mathbf{R}_s (\text{pre}_R P \vdash \text{peri}_R P \diamond R4(\text{post}_R P)) \text{ od}$
by (*simp add: Productive-RHS-design-form unrest assms rdes closure R4-def*)
also have $\dots \text{ is C2}$
by (*simp add: closure assms unrest rdes-def C2-rdes-intro*)
finally show *?thesis* .

qed

lemma *WhileC-CACT-closed* [closure]:
assumes $P \text{ is CACT } P \text{ is Productive}$
shows *while_C 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_C 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_C 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 $g \&_C P \text{ is C2}$
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 $g \&_C P \text{ is CACT}$
by (*rule CACT-intro, simp-all add: closure assms*)

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

lemma *DoCSP-CACT* [closure]:
 $\text{do}_C(a) \text{ is CACT}$
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 $\text{state } 'a \cdot P$ is C2
 by (*rule C2-NCSP-intro, simp-all add: closure rdes assms*)

lemma *state-srea-CACt-closed* [closure]:
 assumes P is CACT
 shows $\text{state } 'a \cdot P$ 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 $(P \llbracket ns1 \rrbracket cs \llbracket ns2 \rrbracket Q)$ is C2
proof –
 have $(P \llbracket ns1 \rrbracket cs \llbracket ns2 \rrbracket Q) = (\mathbf{R}_s(\text{pre}_R P \vdash \text{peri}_R P \diamond \text{post}_R P) \llbracket ns1 \rrbracket cs \llbracket ns2 \rrbracket \mathbf{R}_s(\text{pre}_R Q \vdash \text{peri}_R Q \diamond \text{post}_R Q))$
 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 $(P \llbracket ns1 \rrbracket cs \llbracket ns2 \rrbracket Q)$ 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 $P(f)_C$ 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 $P(f)_C$ 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:*

$P \parallel Q = Q \parallel P$
by (*auto intro: parallel-commutative zero-lens-indep*)

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 $P \text{ wr}[cs]_C \text{ false} = \text{undefined}$ (**is** *?lhs = ?rhs*)

proof –

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

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

also have $\dots = (\neg_r (\exists (tt_0, tt_1) \cdot$
 $[\$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_1 \rangle] \dagger P \wedge$
 $\$tr \leq_u \$tr' \wedge$
 $\&tt \in_u \langle tt_0 \rangle \star_{cs} \langle tt_1 \rangle \wedge \langle tt_0 \rangle \upharpoonright_u \langle cs \rangle =_u \langle tt_1 \rangle \upharpoonright_u \langle cs \rangle) \;;$
R1 true)

by (*rel-blast*)

also have $\dots = (\neg_r (\exists (tt_0, tt_1) \cdot$
 $[\$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_1 \rangle] \dagger RR(P) \wedge$
 $\$tr \leq_u \$tr' \wedge$
 $\&tt \in_u \langle tt_0 \rangle \star_{cs} \langle tt_1 \rangle \wedge \langle tt_0 \rangle \upharpoonright_u \langle cs \rangle =_u \langle tt_1 \rangle \upharpoonright_u \langle cs \rangle) \;;$
R1 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_r*) **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_r 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_r 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-shEx [rpred]: hider (∃ i • P(i)) cs = (∃ i • hider (P i) cs)
  by (rel-auto)

```

```

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_r*) **where**
[upred-defs]: *abs_r* P E = (¬_r (hide_r (¬_r P) E ;; true_r))

```

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  $\implies abs_r (\neg_r P) E = (\neg_r hide_r P E ;; R1 true)$ 
  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 =  $abs_r (\neg_r hide_r (\neg_r P) A ;; R1 true) B$ 
  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 $HideCSP :: ('s, 'e) action \Rightarrow 'e set \Rightarrow ('s, 'e) action$ (**infixl** \setminus_C 80) **where**
 $[upred-defs]:$

$$\text{HideCSP } P \ E = \mathbf{R}_s(\text{abs}_r(\text{pre}_R(P))) \ E \vdash \text{hide}_r(\text{peri}_R(P)) \ E \diamond \text{hide}_r(\text{post}_R(P)) \ E$$

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 ... = $\mathbf{R}_s(\text{abs}_r \ P \ A \vdash (\text{abs}_r \ P \ A \Rightarrow_r \text{hide}_r \ (P \Rightarrow_r \ Q) \ A) \diamond (\text{abs}_r \ P \ A \Rightarrow_r \text{hide}_r \ (P \Rightarrow_r \ R) \ A))$

by (metis RHS-tri-design-conj conj-idem utp-pred-laws.sup.idem)

also have ... = ?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 $(b \ \&_C \ P) \setminus_C A = b \ \&_C \ (P \setminus_C A)$

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

7 Easy to use Circus-M parser

theory *utp-circus-easy-parser*
imports $utp-circus$ $UTP.utp-easy-parser$
begin recall-syntax

We change $:=$ so that it refers to the Circus operator

no-adhoc-overloading
 $uassigns$ $assigns-r$

adhoc-overloading
 $uassigns$ $AssignsCSP$

syntax

-*GuardCSP* :: *uexp* \Rightarrow *logic* \Rightarrow *logic* (**infixr** && 60)

no-translations

-*uwhile-top* *b P* == *CONST while-top* *b P*

translations

-*uwhile-top* *b P* == *CONST WhileR* *b P*

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.