

Circus in Isabelle/UTP

Simon Foster James Baxter Ana Cavalcanti Jim Woodcock
Samuel Canham

April 17, 2018

Contents

1	Introduction	1
2	Circus Trace Merge	1
2.1	Function Definition	1
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	16
4.3	Parallel Laws	18
5	Meta theory for Circus	24

1 Introduction

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

2 Circus Trace Merge

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

2.1 Function Definition

```
fun tr-par ::
  'a set => 'a list => 'a list => 'a list set where
tr-par cs [] [] = {} |
tr-par cs (e # t) [] = (if e ∈ cs then {} else {[e]} ∩ (tr-par cs t [])) |
tr-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 (* ∧ e2 ∈ cs *)
```

```

    then {[e1]}  $\frown$  (tr-par cs t1 t2)
  else
    ({[e1]}  $\frown$  (tr-par cs t1 (e2 # t2)))  $\cup$ 
    ({[e2]}  $\frown$  (tr-par cs (e1 # t1) t2))
else
  if e1  $\in$  cs then
    if e2  $\in$  cs then {}
  else
    {[e2]}  $\frown$  (tr-par cs (e1 # t1) t2)
else
  if e2  $\in$  cs then
    {[e1]}  $\frown$  (tr-par cs t1 (e2 # t2))
  else
    {[e1]}  $\frown$  (tr-par cs t1 (e2 # t2))  $\cup$ 
    {[e2]}  $\frown$  (tr-par cs (e1 # t1) t2)

```

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

2.2 Lifted Trace Merge

syntax -utr-par ::
logic \Rightarrow *logic* \Rightarrow *logic* \Rightarrow *logic* ((- \star - / -) [100, 0, 101] 100)

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

translations

t1 \star_{cs} *t2* \equiv (CONST *trop*) (CONST tr-par) cs *t1 t2*

2.3 Trace Merge Lemmas

lemma *tr-par-empty*:

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

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

— Subgoal 1

apply (induct *t1*; simp)

— Subgoal 2

apply (induct *t2*; simp)

done

lemma *tr-par-sym*:

tr-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* |||_t *y* = *y* |||_t *x*

by (simp add: tr-par-sym)

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

lemma *trace-merge-empty-single* [simp]:
 $a \in cs \implies \langle \rangle \star_{\ll cs \gg} \langle \ll a \gg \rangle = \{\langle \rangle\}_u$
 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]:
 $v \hat{^}_u \langle \rangle = v \langle \rangle \hat{^}_u v = v$
 $v + \langle \rangle = v \langle \rangle + v = v$
 $bop (op \#) x xs \hat{^}_u ys = bop (op \#) x (xs \hat{^}_u ys)$
 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 :: $logic \Rightarrow logic \Rightarrow logic \ (- \rightarrow - \ [81, 80] \ 80)$

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

Input Prefix: $\dots?(x)$

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

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

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

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

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

syntax *-output-prefix* :: $('a, ' \sigma) \ uexpr \Rightarrow prefix\text{-}elem' \ (!'(-))$

syntax *-output-prefix* :: $('a, ' \sigma) \ uexpr \Rightarrow prefix\text{-}elem' \ (.'(-))$

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

syntax

-prefix-aux :: pttrn ⇒ logic ⇒ prefix-elim'

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

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

syntax

-prefix-action ::

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

((-- →/ -) [81, 81, 80] 80)

Syntax translations

definition lconj :: ('a ⇒ 'α upred) ⇒ ('b ⇒ 'α upred) ⇒ ('a × 'b ⇒ 'α upred) (infixr ∧_l 35)

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

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

[upred-defs]: outp-constraint v ≡ (λ x. «x» =_u v)

translations

-simple-input-prefix x ⇐ -input-prefix x true

-mixed-prefix (-input-prefix x P) (-prefix-aux y Q) ⇐

-prefix-aux (-pattern x y) ((λ x. P) ∧_l Q)

-mixed-prefix (-output-prefix P) (-prefix-aux y Q) ⇐

-prefix-aux (-pattern -iddummy y) ((CONST outp-constraint P) ∧_l Q)

-end-prefix (-input-prefix x P) ⇐ -prefix-aux x (λ x. P)

-end-prefix (-output-prefix P) ⇐ -prefix-aux -iddummy (CONST outp-constraint P)

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

Basic print translations; more work needed

translations

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

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

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

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

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

x!(v) → P <= CONST OutputCSP x v P

term x!(1)!(y) → P

term x?(v) → P

term x?(v:false) → P

term x!(⟨1⟩) → P

term x?(v)!(1) → P

term x!(⟨1⟩)!(2)?(v:true) → P

Basic translations for state variable communications

syntax

-csp-input-var :: logic ⇒ id ⇒ logic ⇒ logic ⇒ logic (-?%-> - [81, 0, 0, 80] 80)

-csp-inputu-var :: logic ⇒ id ⇒ logic ⇒ logic (-?%-> - [81, 0, 80] 80)

translations

c?%x:A → P ⇐ CONST InputVarCSP c x A P

c?%x → P ⇐ CONST InputVarCSP c x (λ x. true) P

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

lemma *outp-constraint-prod*:

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

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

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

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

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

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

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

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

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

proof –

have $(\bigcup y \cdot [v =_o y]_{S<} \Rightarrow_r RR(P(y))) = RR(P(y)) \ll y \rightarrow [v]_{S\leftarrow} \gg$
apply (*rel-simp, safe*)
apply *metis*
apply *blast*
apply *simp*
done
thus *?thesis*
by (*simp add: Healthy-if assms*)

qed

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

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

proof –

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

qed

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

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

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

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*)

by (rule RDM-intro, simp add: closure, simp-all add: CSPInnerMerge-def unrest)

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

assumes P is R2m

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

proof –

have $R2m(\exists \text{\$ref}' \cdot R2m P) = (\exists \text{\$ref}' \cdot R2m P)$

by (rel-auto)

thus ?thesis

by (metis Healthy-def' assms)

qed

lemma *CSPInnerMerge-unrests* [unrest]:

$\$ok_{<} \# N_C \text{ ns1 cs ns2}$

$\$wait_{<} \# N_C \text{ ns1 cs ns2}$

by (rel-auto)+

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

assumes P is RR Q is RR

shows $P \llbracket \text{ns1} | \text{cs} | \text{ns2} \rrbracket^I Q$ is RR

by (simp add: CSPInterMerge-def parallel-RR-closed assms closure unrest)

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

$\$st' \# P \llbracket \text{ns1} | \text{cs} | \text{ns2} \rrbracket^I Q$

by (rel-auto)

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

assumes P is RR Q is RR

shows $P \llbracket \text{ns1} | \text{cs} | \text{ns2} \rrbracket^F Q$ is RR

by (simp add: CSPFinalMerge-def parallel-RR-closed assms closure unrest)

lemma *CSPInnerMerge-empty-Interleave*:

$N_C \text{ ns1 } \{ \} \text{ ns2} = N_I \text{ ns1 ns2}$

by (rel-auto)

definition *CSPMerge* :: $(\alpha \Rightarrow \sigma) \Rightarrow \psi \text{ set} \Rightarrow (\beta \Rightarrow \sigma) \Rightarrow ((\sigma, \psi) \text{ st-csp}) \text{ merge } (M_C)$ **where**
[upred-defs]: $M_C \text{ ns1 cs ns2} = M_R(N_C \text{ ns1 cs ns2}) \;; \text{Skip}$

definition *CSPInterleave* :: $(\alpha \Rightarrow \sigma) \Rightarrow (\beta \Rightarrow \sigma) \Rightarrow ((\sigma, \psi) \text{ st-csp}) \text{ merge } (M_I)$ **where**
[upred-defs]: $M_I \text{ ns1 ns2} = M_R(N_I \text{ ns1 ns2}) \;; \text{Skip}$

lemma *swap-CSPInnerMerge*:

$\text{ns1} \bowtie \text{ns2} \Rightarrow \text{swap}_m \;; (N_C \text{ ns1 cs ns2}) = (N_C \text{ ns2 cs ns1})$

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]: $N_C \text{ } 0_L \text{ cs } 0_L$ is SymMerge

by (simp add: Healthy-def swap-CSPInnerMerge)

lemma *SymMerge-CSPInnerInterleave* [closure]:

$N_I \text{ } 0_L \text{ } 0_L$ 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-false [rpred]: $P \llbracket ns1|cs|ns2 \rrbracket^I false = false$

by (simp add: CSPInterMerge-def)

lemma CSPFinalMerge-false [rpred]: $P \llbracket ns1|cs|ns2 \rrbracket^F false = false$

by (simp add: CSPFinalMerge-def)

lemma CSPInterMerge-commute:

assumes $ns1 \bowtie ns2$

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

proof –

have $P \llbracket ns1|cs|ns2 \rrbracket^I Q = P \parallel_{\exists \$st' \cdot N_C ns1 cs ns2} Q$

by (simp add: CSPInterMerge-def)

also have $\dots = P \parallel_{\exists \$st' \cdot (swap_m ;; N_C ns2 cs ns1)} Q$

by (simp add: swap-CSPInnerMerge lens-indep-sym assms)

also have $\dots = P \parallel_{swap_m ;; (\exists \$st' \cdot N_C ns2 cs ns1)} Q$

by (simp add: seqr-exists-right)

also have $\dots = Q \parallel_{(\exists \$st' \cdot N_C ns2 cs ns1)} P$

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

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

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 $P Q :: ('σ, 'φ)$ action

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

shows

$P \parallel_{N_C}^{ns1\ cs\ ns2} Q =$
 $(\exists (ref_0, ref_1, st_0, st_1, tt_0, tt_1) \cdot$
 $P[\llbracket \langle ref_0 \rangle, \langle st_0 \rangle, \langle \rangle, \langle tt_0 \rangle / \$ref', \$st', \$tr, \$tr' \rrbracket \wedge Q[\llbracket \langle ref_1 \rangle, \langle st_1 \rangle, \langle \rangle, \langle tt_1 \rangle / \$ref', \$st', \$tr, \$tr' \rrbracket$
 $\wedge \$ref' \subseteq_u ((\langle ref_0 \rangle \cup_u \langle ref_1 \rangle) \cap_u \langle cs \rangle) \cup_u ((\langle ref_0 \rangle \cap_u \langle ref_1 \rangle) - \langle cs \rangle)$
 $\wedge \$tr \leq_u \tr'
 $\wedge \&tt \in_u \langle tt_0 \rangle \star_{\langle cs \rangle} \langle tt_1 \rangle$
 $\wedge \langle tt_0 \rangle \upharpoonright_u \langle cs \rangle =_u \langle tt_1 \rangle \upharpoonright_u \langle cs \rangle$
 $\wedge \$st' =_u (\$st \oplus \langle st_0 \rangle \text{ on } \&ns1) \oplus \langle st_1 \rangle \text{ on } \&ns2)$
 $(\text{is } ?lhs = ?rhs)$
proof –
have $P: (\exists \{ \$ok', \$wait' \} \cdot R2(P)) = P \text{ (is } ?P' = -)$
by (*simp add: ex-unrest ex-plus Healthy-if assms RR-implies-R2 unrest closure*)
have $Q: (\exists \{ \$ok', \$wait' \} \cdot R2(Q)) = Q \text{ (is } ?Q' = -)$
by (*simp add: ex-unrest ex-plus Healthy-if assms RR-implies-R2 unrest closure*)
from *assms(1,2)*
have $?P' \parallel_{N_C}^{ns1\ cs\ ns2} ?Q' =$
 $(\exists (ref_0, ref_1, st_0, st_1, tt_0, tt_1) \cdot$
 $?P'[\llbracket \langle ref_0 \rangle, \langle st_0 \rangle, \langle \rangle, \langle tt_0 \rangle / \$ref', \$st', \$tr, \$tr' \rrbracket \wedge ?Q'[\llbracket \langle ref_1 \rangle, \langle st_1 \rangle, \langle \rangle, \langle tt_1 \rangle / \$ref', \$st', \$tr, \$tr' \rrbracket$
 $\wedge \$ref' \subseteq_u ((\langle ref_0 \rangle \cup_u \langle ref_1 \rangle) \cap_u \langle cs \rangle) \cup_u ((\langle ref_0 \rangle \cap_u \langle ref_1 \rangle) - \langle cs \rangle)$
 $\wedge \$tr \leq_u \tr'
 $\wedge \&tt \in_u \langle tt_0 \rangle \star_{\langle cs \rangle} \langle tt_1 \rangle$
 $\wedge \langle tt_0 \rangle \upharpoonright_u \langle cs \rangle =_u \langle tt_1 \rangle \upharpoonright_u \langle cs \rangle$
 $\wedge \$st' =_u (\$st \oplus \langle st_0 \rangle \text{ on } \&ns1) \oplus \langle st_1 \rangle \text{ on } \&ns2)$
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 $P\ Q :: ('σ, 'φ) \text{ action}$

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

shows

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

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

proof –

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

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

also have ... =

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

by (simp add: CSPInnerMerge-form assms)

also have ... = ?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

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

(is ?lhs = ?rhs)

proof –

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

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

also have ... =

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

by (simp add: CSPInnerMerge-form assms)

also have ... =

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

by (simp add: ex-unrest assms)

also have ... =

($\exists (st_0, st_1, tt_0, tt_1)$.
 ($\exists \$ref' \cdot P$) $\llbracket \langle\langle st_0 \rangle\rangle, \langle\langle st_1 \rangle\rangle, \langle\langle tt_0 \rangle\rangle / \$st', \$tr, \$tr' \rrbracket \wedge (\exists \$ref' \cdot Q) \llbracket \langle\langle st_1 \rangle\rangle, \langle\langle st_1 \rangle\rangle, \langle\langle tt_1 \rangle\rangle / \$st', \$tr, \$tr' \rrbracket$
 $\wedge \$tr \leq_u \tr'
 $\wedge \&tt \in_u \langle\langle tt_0 \rangle\rangle \star_{\langle\langle cs \rangle\rangle} \langle\langle tt_1 \rangle\rangle$

$\wedge \ll tt_0 \gg \upharpoonright_u \ll cs \gg =_u \ll tt_1 \gg \upharpoonright_u \ll cs \gg$
 $\wedge \$st' =_u (\$st \oplus \ll st_0 \gg \text{ on } \&ns1) \oplus \ll st_1 \gg \text{ on } \&ns2)$
 by (*rel-blast*)
 also have ... = ?rhs
 by (*simp add: ex-unrest assms*)
 finally show ?thesis .
 qed

lemma *merge-csp-do-left*:

assumes *vwb-lens ns1 vwb-lens ns2 ns1 \bowtie ns2 P is RR*
shows $\Phi(s_0, \sigma_0, t_0) \parallel_{N_C} ns1 \text{ cs } ns2 \text{ } P =$
 $(\exists (ref_1, st_1, tt_1) \cdot$
 $[s_0]_{S<} \wedge$
 $[\$ref' \mapsto_s \ll ref_1 \gg, \$st' \mapsto_s \ll st_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll tt_1 \gg] \dagger P \wedge$
 $\$ref' \subseteq_u \ll cs \gg \cup_u (\ll ref_1 \gg - \ll cs \gg) \wedge$
 $[\ll trace \gg \in_u t_0 \star \ll cs \gg \ll tt_1 \gg \wedge t_0 \upharpoonright_u \ll cs \gg =_u \ll tt_1 \gg \upharpoonright_u \ll cs \gg]_t \wedge$
 $\$st' =_u \$st \oplus \ll \sigma_0 \gg (\$st)_a \text{ on } \&ns1 \oplus \ll st_1 \gg \text{ on } \&ns2)$
 (is ?lhs = ?rhs)
proof –
 have ?lhs =
 $(\exists (ref_0, ref_1, st_0, st_1, tt_0, tt_1) \cdot$
 $[\$ref' \mapsto_s \ll ref_0 \gg, \$st' \mapsto_s \ll st_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll tt_0 \gg] \dagger \Phi(s_0, \sigma_0, t_0) \wedge$
 $[\$ref' \mapsto_s \ll ref_1 \gg, \$st' \mapsto_s \ll st_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll tt_1 \gg] \dagger P \wedge$
 $\$ref' \subseteq_u (\ll ref_0 \gg \cup_u \ll ref_1 \gg) \cap_u \ll cs \gg \cup_u (\ll ref_0 \gg \cap_u \ll ref_1 \gg - \ll cs \gg) \wedge$
 $\$tr \leq_u \$tr' \wedge$
 $\&tt \in_u \ll tt_0 \gg \star \ll cs \gg \ll tt_1 \gg \wedge \ll tt_0 \gg \upharpoonright_u \ll cs \gg =_u \ll tt_1 \gg \upharpoonright_u \ll cs \gg \wedge \$st' =_u \$st \oplus \ll st_0 \gg \text{ on } \&ns1$
 $\oplus \ll st_1 \gg \text{ on } \&ns2)$
 by (*simp add: CSPInnerMerge-form assms closure*)
 also have ... =
 $(\exists (ref_1, st_1, tt_1) \cdot$
 $[s_0]_{S<} \wedge$
 $[\$ref' \mapsto_s \ll ref_1 \gg, \$st' \mapsto_s \ll st_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll tt_1 \gg] \dagger P \wedge$
 $\$ref' \subseteq_u \ll cs \gg \cup_u (\ll ref_1 \gg - \ll cs \gg) \wedge$
 $[\ll trace \gg \in_u t_0 \star \ll cs \gg \ll tt_1 \gg \wedge t_0 \upharpoonright_u \ll cs \gg =_u \ll tt_1 \gg \upharpoonright_u \ll cs \gg]_t \wedge$
 $\$st' =_u \$st \oplus \ll \sigma_0 \gg (\$st)_a \text{ on } \&ns1 \oplus \ll st_1 \gg \text{ on } \&ns2)$
 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 $P \parallel_{N_C} ns1 \text{ cs } ns2 \text{ } \Phi(s_1, \sigma_1, t_1) =$
 $(\exists (ref_0, st_0, tt_0) \cdot$
 $[\$ref' \mapsto_s \ll ref_0 \gg, \$st' \mapsto_s \ll st_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll tt_0 \gg] \dagger P \wedge$
 $[s_1]_{S<} \wedge$
 $\$ref' \subseteq_u \ll cs \gg \cup_u (\ll ref_0 \gg - \ll cs \gg) \wedge$
 $[\ll trace \gg \in_u \ll tt_0 \gg \star \ll cs \gg t_1 \wedge \ll tt_0 \gg \upharpoonright_u \ll cs \gg =_u t_1 \upharpoonright_u \ll cs \gg]_t \wedge$
 $\$st' =_u \$st \oplus \ll st_0 \gg \text{ on } \&ns1 \oplus \ll \sigma_1 \gg (\$st)_a \text{ on } \&ns2)$
 (is ?lhs = ?rhs)
proof –
 have ?lhs =
 $(\exists (ref_0, ref_1, st_0, st_1, tt_0, tt_1) \cdot$
 $[\$ref' \mapsto_s \ll ref_0 \gg, \$st' \mapsto_s \ll st_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll tt_0 \gg] \dagger P \wedge$
 $[\$ref' \mapsto_s \ll ref_1 \gg, \$st' \mapsto_s \ll st_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll tt_1 \gg] \dagger \Phi(s_1, \sigma_1, t_1) \wedge$
 $\$ref' \subseteq_u (\ll ref_0 \gg \cup_u \ll ref_1 \gg) \cap_u \ll cs \gg \cup_u (\ll ref_0 \gg \cap_u \ll ref_1 \gg - \ll cs \gg) \wedge$

$\$tr \leq_u \$tr' \wedge$
 $\&tt \in_u \ll tt_0 \gg \star \ll cs \gg \ll tt_1 \gg \wedge \ll tt_0 \gg \upharpoonright_u \ll cs \gg =_u \ll tt_1 \gg \upharpoonright_u \ll cs \gg \wedge \$st' =_u \$st \oplus \ll st_0 \gg \text{ on } \&ns1 \oplus \ll st_1 \gg \text{ on } \&ns2)$
 by (simp add: CSPInnerMerge-form assms closure)
 also have ... = ?rhs
 by (rel-blast)
 finally show ?thesis .
 qed

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

lemma *FinalMerge-csp-do-left:*

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

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

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

(is ?lhs = ?rhs)

proof –

have ?lhs =

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

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

also have ... =

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

by (rel-blast)

also have ... =

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

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 $P \ll ns1 | cs | ns2 \gg^F \Phi(s_1, \sigma_1, t_1) =$

$(\exists (st_0, t_0) \cdot$
 $[\$st' \mapsto_s \ll st_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_0 \gg] \dagger P \wedge$
 $[s_1]_{S<} \wedge$
 $[\ll trace \gg \in_u \ll t_0 \gg \star \ll cs \gg t_1 \wedge \ll t_0 \gg \upharpoonright_u \ll cs \gg =_u t_1 \upharpoonright_u \ll cs \gg]_t \wedge$
 $\$st' =_u \$st \oplus \ll st_0 \gg \text{ on } \&ns1 \oplus \ll \sigma_1 \gg (\$st)_a \text{ on } \&ns2)$

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

lemma *FinalMerge-csp-do' [rpred]*:

assumes *vwb-lens ns1 vwb-lens ns2 ns1 \bowtie ns2*
shows $\Phi(s_1, \sigma_1, t_1) \llbracket ns1|cs|ns2 \rrbracket^F \Phi(s_2, \sigma_2, t_2) =$
 $(\bigcap \text{trace} \mid \llbracket trace \rrbracket \in_u \lceil t_1 \star_{\llbracket cs \rrbracket} t_2 \rceil_{S<} \cdot$
 $\Phi(s_1 \wedge s_2 \wedge t_1 \upharpoonright_u \llbracket cs \rrbracket =_u t_2 \upharpoonright_u \llbracket cs \rrbracket, \sigma_1 [\&ns1|\&ns2]_s \sigma_2, \llbracket trace \rrbracket))$
by (*simp add: FinalMerge-csp-do assms, rel-auto*)

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

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

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

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

lemma *InterMerge-csp-enable*:

assumes *vwb-lens ns1 vwb-lens ns2 ns1 \bowtie ns2*
shows $\mathcal{E}(s_1, t_1, E_1) \llbracket ns1|cs|ns2 \rrbracket^I \mathcal{E}(s_2, t_2, E_2) =$
 $([s_1 \wedge s_2]_{S<} \wedge$
 $(\forall e \in \lceil (E_1 \cap_u E_2 \cap_u \llbracket cs \rrbracket) \cup_u ((E_1 \cup_u E_2) - \llbracket cs \rrbracket) \rceil_{S<} \cdot \llbracket e \rrbracket \notin_u \$ref') \wedge$
 $\llbracket trace \rrbracket \in_u t_1 \star_{\llbracket cs \rrbracket} t_2 \wedge t_1 \upharpoonright_u \llbracket cs \rrbracket =_u t_2 \upharpoonright_u \llbracket cs \rrbracket]_t)$
(is ?lhs = ?rhs)
proof –
have ?lhs =

$(\exists (ref_0, ref_1, st_0, st_1, tt_0, tt_1) \cdot$
 $[\$ref' \mapsto_s \langle\langle ref_0 \rangle\rangle, \$st' \mapsto_s \langle\langle st_0 \rangle\rangle, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \langle\langle tt_0 \rangle\rangle] \dagger \mathcal{E}(s_1, t_1, E_1) \wedge$
 $[\$ref' \mapsto_s \langle\langle ref_1 \rangle\rangle, \$st' \mapsto_s \langle\langle st_1 \rangle\rangle, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \langle\langle tt_1 \rangle\rangle] \dagger \mathcal{E}(s_2, t_2, E_2) \wedge$
 $\$ref' \subseteq_u (\langle\langle ref_0 \rangle\rangle \cup_u \langle\langle ref_1 \rangle\rangle) \cap_u \langle\langle cs \rangle\rangle \cup_u (\langle\langle ref_0 \rangle\rangle \cap_u \langle\langle ref_1 \rangle\rangle - \langle\langle cs \rangle\rangle) \wedge$
 $\$tr \leq_u \$tr' \wedge \&tt \in_u \langle\langle tt_0 \rangle\rangle \star \langle\langle cs \rangle\rangle \langle\langle tt_1 \rangle\rangle \wedge \langle\langle tt_0 \rangle\rangle \upharpoonright_u \langle\langle cs \rangle\rangle =_u \langle\langle tt_1 \rangle\rangle \upharpoonright_u \langle\langle cs \rangle\rangle)$
 by (simp add: CSPInterMerge-form unrest closure assms)
 also have ... =
 $(\exists (ref_0, ref_1, tt_0, tt_1) \cdot$
 $[\$ref' \mapsto_s \langle\langle ref_0 \rangle\rangle, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \langle\langle tt_0 \rangle\rangle] \dagger \mathcal{E}(s_1, t_1, E_1) \wedge$
 $[\$ref' \mapsto_s \langle\langle ref_1 \rangle\rangle, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \langle\langle tt_1 \rangle\rangle] \dagger \mathcal{E}(s_2, t_2, E_2) \wedge$
 $\$ref' \subseteq_u (\langle\langle ref_0 \rangle\rangle \cup_u \langle\langle ref_1 \rangle\rangle) \cap_u \langle\langle cs \rangle\rangle \cup_u (\langle\langle ref_0 \rangle\rangle \cap_u \langle\langle ref_1 \rangle\rangle - \langle\langle cs \rangle\rangle) \wedge$
 $\$tr \leq_u \$tr' \wedge \&tt \in_u \langle\langle tt_0 \rangle\rangle \star \langle\langle cs \rangle\rangle \langle\langle tt_1 \rangle\rangle \wedge \langle\langle tt_0 \rangle\rangle \upharpoonright_u \langle\langle cs \rangle\rangle =_u \langle\langle tt_1 \rangle\rangle \upharpoonright_u \langle\langle cs \rangle\rangle)$
 by (rel-auto)
 also have ... =
 $([s_1 \wedge s_2]_{S<} \wedge$
 $(\forall e \in [(E_1 \cap_u E_2 \cap_u \langle\langle cs \rangle\rangle) \cup_u ((E_1 \cup_u E_2) - \langle\langle cs \rangle\rangle)]_{S<} \cdot \langle\langle e \rangle\rangle \notin_u \$ref') \wedge$
 $[\langle\langle trace \rangle\rangle \in_u t_1 \star \langle\langle cs \rangle\rangle t_2 \wedge t_1 \upharpoonright_u \langle\langle cs \rangle\rangle =_u t_2 \upharpoonright_u \langle\langle cs \rangle\rangle]_t$
 $)$
 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]:
 assumes *vwb-lens ns1 vwb-lens ns2 ns1* \bowtie *ns2*
 shows $\mathcal{E}(s_1, t_1, E_1) \llbracket ns1 | cs | ns2 \rrbracket^I \mathcal{E}(s_2, t_2, E_2) =$
 $(\bigcap \text{trace} \mid \langle\langle trace \rangle\rangle \in_u [t_1 \star \langle\langle cs \rangle\rangle t_2]_{S<} \cdot$
 $\mathcal{E}(s_1 \wedge s_2 \wedge t_1 \upharpoonright_u \langle\langle cs \rangle\rangle =_u t_2 \upharpoonright_u \langle\langle cs \rangle\rangle$
 $, \langle\langle trace \rangle\rangle$
 $, (E_1 \cap_u E_2 \cap_u \langle\langle cs \rangle\rangle) \cup_u ((E_1 \cup_u E_2) - \langle\langle cs \rangle\rangle)))$
 by (simp add: InterMerge-csp-enable assms, rel-auto)

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

proof –
 have ?lhs =
 $(\exists (ref_0, ref_1, st_0, st_1, tt_0, tt_1) \cdot$
 $[\$ref' \mapsto_s \langle\langle ref_0 \rangle\rangle, \$st' \mapsto_s \langle\langle st_0 \rangle\rangle, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \langle\langle tt_0 \rangle\rangle] \dagger \mathcal{E}(s_1, t_1, E_1) \wedge$
 $[\$ref' \mapsto_s \langle\langle ref_1 \rangle\rangle, \$st' \mapsto_s \langle\langle st_1 \rangle\rangle, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \langle\langle tt_1 \rangle\rangle] \dagger \Phi(s_2, \sigma_2, t_2) \wedge$
 $\$ref' \subseteq_u (\langle\langle ref_0 \rangle\rangle \cup_u \langle\langle ref_1 \rangle\rangle) \cap_u \langle\langle cs \rangle\rangle \cup_u (\langle\langle ref_0 \rangle\rangle \cap_u \langle\langle ref_1 \rangle\rangle - \langle\langle cs \rangle\rangle) \wedge$
 $\$tr \leq_u \$tr' \wedge \&tt \in_u \langle\langle tt_0 \rangle\rangle \star \langle\langle cs \rangle\rangle \langle\langle tt_1 \rangle\rangle \wedge \langle\langle tt_0 \rangle\rangle \upharpoonright_u \langle\langle cs \rangle\rangle =_u \langle\langle tt_1 \rangle\rangle \upharpoonright_u \langle\langle cs \rangle\rangle)$
 by (simp add: CSPInterMerge-form unrest closure assms)
 also have ... =
 $(\exists (ref_0, ref_1, tt_0) \cdot$
 $[\$ref' \mapsto_s \langle\langle ref_0 \rangle\rangle, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \langle\langle tt_0 \rangle\rangle] \dagger \mathcal{E}(s_1, t_1, E_1) \wedge$
 $[s_2]_{S<} \wedge$

$\$ref' \subseteq_u (\llbracket ref_0 \rrbracket \cup_u \llbracket ref_1 \rrbracket) \cap_u \llbracket cs \rrbracket \cup_u (\llbracket ref_0 \rrbracket \cap_u \llbracket ref_1 \rrbracket - \llbracket cs \rrbracket) \wedge$
 $[\llbracket trace \rrbracket \in_u t_1 \star \llbracket cs \rrbracket t_2 \wedge t_1 \downarrow_u \llbracket cs \rrbracket =_u t_2 \downarrow_u \llbracket cs \rrbracket]_t)$
 by (*rel-auto*)
 also have ... = $([s_1 \wedge s_2]_{S<} \wedge (\forall e \in [(E_1 - \llbracket cs \rrbracket)]_{S<} \cdot \llbracket e \rrbracket \notin_u \$ref') \wedge$
 $[\llbracket trace \rrbracket \in_u t_1 \star \llbracket cs \rrbracket t_2 \wedge t_1 \downarrow_u \llbracket cs \rrbracket =_u t_2 \downarrow_u \llbracket cs \rrbracket]_t)$
 by (*rel-auto*)
 finally show ?thesis .
 qed

lemma *InterMerge-csp-enable-csp-do'* [rpred]:
 assumes *vwb-lens ns1 vwb-lens ns2 ns1 \bowtie ns2*
 shows $\mathcal{E}(s_1, t_1, E_1) \llbracket ns1 | cs | ns2 \rrbracket^I \Phi(s_2, \sigma_2, t_2) =$
 $(\bigcap trace \mid \llbracket trace \rrbracket \in_u [t_1 \star \llbracket cs \rrbracket t_2]_{S<} \cdot$
 $\mathcal{E}(s_1 \wedge s_2 \wedge t_1 \downarrow_u \llbracket cs \rrbracket =_u t_2 \downarrow_u \llbracket cs \rrbracket, \llbracket trace \rrbracket, E_1 - \llbracket cs \rrbracket))$
 by (*simp add: InterMerge-csp-enable-csp-do assms, rel-auto*)

lemma *InterMerge-csp-do-csp-enable*:
 assumes *vwb-lens ns1 vwb-lens ns2 ns1 \bowtie ns2*
 shows $\Phi(s_1, \sigma_1, t_1) \llbracket ns1 | cs | ns2 \rrbracket^I \mathcal{E}(s_2, t_2, E_2) =$
 $([s_1 \wedge s_2]_{S<} \wedge (\forall e \in [(E_2 - \llbracket cs \rrbracket)]_{S<} \cdot \llbracket e \rrbracket \notin_u \$ref') \wedge$
 $[\llbracket trace \rrbracket \in_u t_1 \star \llbracket cs \rrbracket t_2 \wedge t_1 \downarrow_u \llbracket cs \rrbracket =_u t_2 \downarrow_u \llbracket cs \rrbracket]_t)$
 (is ?lhs = ?rhs)

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

lemma *InterMerge-csp-do-csp-enable'* [rpred]:
 assumes *vwb-lens ns1 vwb-lens ns2 ns1 \bowtie ns2*
 shows $\Phi(s_1, \sigma_1, t_1) \llbracket ns1 | cs | ns2 \rrbracket^I \mathcal{E}(s_2, t_2, E_2) =$
 $(\bigcap trace \mid \llbracket trace \rrbracket \in_u [t_1 \star \llbracket cs \rrbracket t_2]_{S<} \cdot$
 $\mathcal{E}(s_1 \wedge s_2 \wedge t_1 \downarrow_u \llbracket cs \rrbracket =_u t_2 \downarrow_u \llbracket cs \rrbracket, \llbracket trace \rrbracket, E_2 - \llbracket cs \rrbracket))$
 by (*simp add: InterMerge-csp-do-csp-enable assms, rel-auto*)

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

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

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

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

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

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

lemma merge-csp-do-right:

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

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

(is ?lhs = ?rhs)

proof –

have ?lhs =

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

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

also have ... =

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

apply (rel-auto)

oops

4.2 Parallel operator

syntax

-par-circus :: logic \Rightarrow salpha \Rightarrow logic \Rightarrow salpha \Rightarrow logic \Rightarrow logic (- $\ll \cdot \parallel \cdot \parallel \cdot \parallel$ - [75,0,0,0,76] 76)
 -par-csp :: logic \Rightarrow logic \Rightarrow logic \Rightarrow logic (- $\ll \cdot \parallel_C \cdot \parallel_C$ - [75,0,76] 76)
 -inter-circus :: logic \Rightarrow salpha \Rightarrow salpha \Rightarrow logic \Rightarrow logic (- $\ll \cdot \parallel \cdot \parallel$ - [75,0,0,76] 76)
 -inter-csp :: logic \Rightarrow logic \Rightarrow logic (**infixr** \parallel 75)

translations

-par-circus $P \ ns1 \ cs \ ns2 \ Q == P \parallel_{M_C} ns1 \ cs \ ns2 \ Q$
 -par-csp $P \ cs \ Q == \text{-par-circus } P \ 0_L \ cs \ 0_L \ Q$
 -inter-circus $P \ ns1 \ ns2 \ Q == \text{-par-circus } P \ ns1 \ \{\} \ ns2 \ Q$
 -inter-csp $P \ Q == \text{-par-csp } P \ \{\} \ Q$

definition CSP5 :: (' σ , ' φ) action \Rightarrow (' σ , ' φ) action **where**

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

definition C2 :: (' σ , ' φ) action \Rightarrow (' σ , ' φ) action **where**

[upred-defs]: C2(P) = ($P \ll \Sigma \parallel \{\} \parallel \emptyset \gg \text{Skip}$)

lemma Skip-right-form:

assumes P_1 is RC P_2 is RR P_3 is RR $\$st' \# P_2$

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

proof –

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

by (rel-auto)

show ?thesis

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

qed

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

fixes $P_1 :: ('s, 'e)$ *action*

assumes P_1 is CRC Q_1 is CRC P_2 is CRR Q_2 is CRR P_3 is CRR Q_3 is CRR

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

$ns1 \bowtie ns2$

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

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

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

proof –

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

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

also

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

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

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

also

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

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

by (*simp add: Skip-right-form closure parallel-RR-closed assms unrest*)

also

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

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

proof –

have $(\exists \$ref' \cdot ((P_1 \Rightarrow_r P_3) \parallel_{N_C \ ns1 \ cs \ ns2} (Q_1 \Rightarrow_r Q_3))) = ((P_1 \Rightarrow_r P_3) \llbracket ns1 | cs | ns2 \rrbracket^F (Q_1 \Rightarrow_r Q_3))$

by (*rel-blast*)

thus $?thesis$ **by** *simp*

qed

finally show $?thesis$.

qed

4.3 Parallel Laws

lemma *ParCSP-expand*:

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

qed

lemma *parallel-is-CSP3 [closure]*:

assumes P is CSP P is CSP3 Q is CSP Q is CSP3
shows $(P \llbracket ns1 \parallel cs \parallel ns2 \rrbracket Q)$ is CSP3

proof –

have $(P \parallel_{M_R(N_C \ ns1 \ cs \ ns2)} Q)$ is CSP
by (*simp add: closure assms*)
hence $(P \parallel_{M_R(N_C \ ns1 \ cs \ ns2)} Q) ;; Skip$ is CSP
by (*simp add: closure*)
thus ?thesis
oops

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 $(P \llbracket ns1 \parallel cs \parallel ns2 \rrbracket Q) = P \parallel_{swap_m} ;; (M_C \ ns2 \ cs \ ns1) \ Q$
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

lemma *interleave-commute*:

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

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

lemma *C2-form*:

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

proof –

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

proof –

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

$$pre_R P)) \wedge$$

$$\begin{aligned} & \$ref' \subseteq_u \ll ref_0 \gg \wedge [\ll trace \gg =_u \ll tt_0 \gg]_t \wedge \\ & \$st' =_u \$st \oplus \ll st_0 \gg \text{ on } \Sigma \oplus \ll id \gg (\$st)_a \text{ on } \emptyset ;; R1 \text{ true} \end{aligned}$$
 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 ... = $(\neg_r (\exists \$ref'; \$st' \cdot RR(\neg_r pre_R P))) ;; R1 \text{ true}$
 by (rel-auto)

also have ... = $(\neg_r (\neg_r pre_R P)) ;; R1 \text{ true}$
 by (simp add: Healthy-if closure ex-unrest unrest assms)

also have ... = $pre_R P$
 by (simp add: NCSP-implies-NSRD NSRD-neg-pre-unit R1-preR assms rea-not-not)

finally show ?thesis .

qed

have 2: $(pre_R P \Rightarrow_r peri_R P) [\Sigma|\{\}|\emptyset]^I \Phi(true, id, \langle \rangle) =$
 $(\exists ref_0 \cdot (peri_R P) [\ll ref_0 \gg / \$ref'] \wedge \$ref' \subseteq_u \ll ref_0 \gg) \text{ (is ?lhs = ?rhs)}$

proof –

have ?lhs = $peri_R P [\Sigma|\{\}|\emptyset]^I \Phi(true, id, \langle \rangle)$
 by (simp add: SRD-peri-under-pre closure assms unrest)

also have ... = $(\exists \$st' \cdot (peri_R P \parallel_{N_C} 1_L \{\} 0_L \Phi(true, id, \langle \rangle)))$
 by (simp add: CSPInterMerge-def par-by-merge-def seqr-exists-right)

also have ... =
 $(\exists \$st' \cdot \exists (ref_0, st_0, tt_0) \cdot$
 $[\$ref' \mapsto_s \ll ref_0 \gg, \$st' \mapsto_s \ll st_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll tt_0 \gg] \dagger (\exists \$st' \cdot RR(peri_R P)) \wedge$
 $\$ref' \subseteq_u \ll ref_0 \gg \wedge [\ll trace \gg =_u \ll tt_0 \gg]_t \wedge \$st' =_u \$st \oplus \ll st_0 \gg \text{ on } \Sigma \oplus \ll id \gg (\$st)_a \text{ on } \emptyset)$

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

also have ... =
 $(\exists ref_0 \cdot (\exists \$st' \cdot RR(peri_R P)) [\ll ref_0 \gg / \$ref'] \wedge \$ref' \subseteq_u \ll ref_0 \gg)$
 by (rel-auto)

also have ... = ?rhs
 by (simp add: closure ex-unrest Healthy-if unrest assms)

finally show ?thesis .

qed

have 3: $(pre_R P \Rightarrow_r post_R P) [\Sigma|\{\}|\emptyset]^F \Phi(true, id, \langle \rangle) = post_R(P) \text{ (is ?lhs = ?rhs)}$

proof –

have ?lhs = $post_R P [\Sigma|\{\}|\emptyset]^F \Phi(true, id, \langle \rangle)$
 by (simp add: SRD-post-under-pre closure assms unrest)

also have ... = $(\exists (st_0, t_0) \cdot$
 $[\$st' \mapsto_s \ll st_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_0 \gg] \dagger RR(post_R P) \wedge$
 $[\ll trace \gg =_u \ll t_0 \gg]_t \wedge \$st' =_u \$st \oplus \ll st_0 \gg \text{ on } \Sigma \oplus \ll id \gg (\$st)_a \text{ on } \emptyset)$

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

also have ... = $RR(post_R(P))$
 by (rel-auto)

finally show ?thesis
 by (simp add: Healthy-if assms closure)

qed

show ?thesis

proof –

have $C2(P) = \mathbf{R}_s (\Phi(true, id, \langle \rangle) wr[\Sigma|\{\}|\emptyset]_C pre_R P \vdash$
 $(pre_R P \Rightarrow_r peri_R P) [\Sigma|\{\}|\emptyset]^I \Phi(true, id, \langle \rangle) \diamond (pre_R P \Rightarrow_r post_R P) [\Sigma|\{\}|\emptyset]^F \Phi(true, id, \langle \rangle))$
 by (simp add: C2-def, rdes-simp cls: assms, simp add: id-def pr-var-def)

also have ... = $\mathbf{R}_s (pre_R P \vdash (\exists ref_0 \cdot peri_R P [\ll ref_0 \gg / \$ref'] \wedge \$ref' \subseteq_u \ll ref_0 \gg) \diamond post_R P)$
 by (simp add: 1 2 3)

finally show ?thesis .

qed

qed

We define downward closure of the pericondition by the following healthiness condition

definition $CDC :: ('s, 'e) \text{ action} \Rightarrow ('s, 'e) \text{ action}$ **where**
 $[upred-defs]: CDC(P) = (\exists \text{ ref}_0 \cdot P \llbracket \llbracket \text{ref}_0 \rrbracket / \$\text{ref}' \rrbracket \wedge \$\text{ref}' \subseteq_u \llbracket \text{ref}_0 \rrbracket)$

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

lemma $CDC\text{-RR-commute}$: $CDC(RR(P)) = RR(CDC(P))$
by $(rel\text{-blast})$

lemma $CDC\text{-RR-closed}$ $[closure]$: $P \text{ is } RR \Rightarrow CDC(P) \text{ is } RR$
by $(metis CDC\text{-RR-commute Healthy-def})$

lemma $CDC\text{-unrest}$ $[unrest]$: $\llbracket vwb\text{-lens } x; (\$ref')_v \bowtie x; x \# P \rrbracket \Rightarrow x \# CDC(P)$
by $(simp \text{ add: } CDC\text{-def unrest usubst lens-indep-sym})$

lemma $CDC\text{-R4-commute}$: $CDC(R4(P)) = R4(CDC(P))$
by $(rel\text{-auto})$

lemma $R4\text{-CDC-closed}$ $[closure]$: $P \text{ is } CDC \Rightarrow R4(P) \text{ is } CDC$
by $(simp \text{ add: } CDC\text{-R4-commute Healthy-def})$

lemma $CDC\text{-R5-commute}$: $CDC(R5(P)) = R5(CDC(P))$
by $(rel\text{-auto})$

lemma $R5\text{-CDC-closed}$ $[closure]$: $P \text{ is } CDC \Rightarrow R5(P) \text{ is } CDC$
by $(simp \text{ add: } CDC\text{-R5-commute Healthy-def})$

lemma $rea\text{-true-CDC}$ $[closure]$: $true_r \text{ is } CDC$
by $(rel\text{-auto})$

lemma $false\text{-CDC}$ $[closure]$: $false \text{ is } CDC$
by $(rel\text{-auto})$

lemma $CDC\text{-UINF-closed}$ $[closure]$:
assumes $\bigwedge i. i \in I \Rightarrow P \ i \text{ is } CDC$
shows $(\bigcap i \in I \cdot P \ i) \text{ is } CDC$
using $assms$ **by** $(rel\text{-blast})$

lemma $CDC\text{-disj-closed}$ $[closure]$:
assumes $P \text{ is } CDC \ Q \text{ is } CDC$
shows $(P \vee Q) \text{ is } CDC$

proof –

have $CDC(P \vee Q) = (CDC(P) \vee CDC(Q))$
by $(rel\text{-auto})$

thus $?thesis$

by $(metis Healthy-def assms(1) assms(2))$

qed

lemma $CDC\text{-USUP-closed}$ $[closure]$:
assumes $\bigwedge i. i \in I \Rightarrow P \ i \text{ is } CDC$
shows $(\bigcup i \in I \cdot P \ i) \text{ is } CDC$
using $assms$ **by** $(rel\text{-blast})$

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

assumes P is CDC Q is CDC

shows $(P \wedge Q)$ is CDC

using *assms* **by** (*rel-auto*, *blast*, *meson*)

lemma *CDC-rea-impl* [*rpred*]:

$\$ref' \# P \implies CDC(P \Rightarrow_r Q) = (P \Rightarrow_r CDC(Q))$

by (*rel-auto*)

lemma *rea-impl-CDC-closed* [closure]:

assumes $\$ref' \# P$ Q is CDC

shows $(P \Rightarrow_r Q)$ is CDC

using *assms* **by** (*simp add: CDC-rea-impl Healthy-def*)

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

assumes Q is CDC

shows $(P ;; Q)$ is CDC

proof –

have $CDC(P ;; Q) = P ;; CDC(Q)$

by (*rel-blast*)

thus *?thesis*

by (*metis Healthy-def assms*)

qed

lemma *csp-enable-CDC* [closure]: $\mathcal{E}(s, t, E)$ is CDC

by (*rel-auto*)

lemma *C2-CDC-form*:

assumes P is NCSP

shows $C2(P) = \mathbf{R}_s(\text{pre}_R P \vdash CDC(\text{peri}_R P) \diamond \text{post}_R P)$

by (*simp add: C2-form assms CDC-def*)

lemma *C2-rdes-def*:

assumes P_1 is CRC P_2 is CRR P_3 is CRR $\$st' \# P_2$ $\$ref' \# P_3$

shows $C2(\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3)) = \mathbf{R}_s(P_1 \vdash CDC(P_2) \diamond P_3)$

by (*simp add: C2-form assms closure rdes unrest usubst, rel-auto*)

lemma *C2-NCSP-intro*:

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

shows P is C2

proof –

have $C2(P) = \mathbf{R}_s(\text{pre}_R P \vdash CDC(\text{peri}_R P) \diamond \text{post}_R P)$

by (*simp add: C2-CDC-form assms(1)*)

also have $\dots = \mathbf{R}_s(\text{pre}_R P \vdash \text{peri}_R P \diamond \text{post}_R P)$

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

also have $\dots = P$

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 P_1 is CRC P_2 is CRR P_2 is CDC P_3 is CRR $\$st' \# P_2$ $\$ref' \# P_3$

shows $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3)$ is C2

unfolding *Healthy-def*
by (*simp add: C2-rdes-def assms unrest closure Healthy-if*)

lemma *C2-implies-CDC-peri* [closure]:
assumes *P is NCSP P is C2*
shows *peri_R(P) is CDC*
proof –
have *peri_R(P) = peri_R (R_s (pre_R P ⊢ CDC(per_R P) ◇ post_R P))*
by (*metis C2-CDC-form Healthy-if assms(1) assms(2)*)
also have *... = CDC (pre_R P ⇒_r peri_R P)*
by (*simp add: rdes rpred assms closure unrest*)
also have *... = CDC (peri_R P)*
by (*simp add: SRD-peri-under-pre closure unrest assms*)
finally show *?thesis*
by (*simp add: Healthy-def*)
qed

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 *wp-rea-CRC* [closure]: $\llbracket P \text{ is CRR}; Q \text{ is CRC} \rrbracket \implies P \text{ wp}_r Q \text{ is CRC}$
by (*rule CRC-intro, simp-all add: unrest 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 *DoCSP-C2* [closure]:
do_C(a) is C2
by (*rdes-simp, rule C2-rdes-intro, simp-all add: closure unrest*)

lemma *PrefixCSP-C2-closed* [closure]:
assumes *P is NCSP P is C2*
shows *a →_C P is C2*
unfolding *PrefixCSP-def* **by** (*metis DoCSP-C2 Healthy-def NCSP-DoCSP NCSP-implies-CSP assms seq-C2-closed*)

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-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 CDC-CRR-closed [closure]:
 assumes P is CRR
 shows $CDC(P)$ is CRR
 by (rule CRR-intro, simp add: CDC-def unrest assms closure, simp add: unrest assms closure)

lemma C2-idem:
 assumes P is NCSP
 shows $C2(C2(P)) = C2(P)$ (is ?lhs = ?rhs)
 proof –
 have ?lhs = $\mathbf{R}_s(\text{pre}_R P \vdash (\text{pre}_R P \Rightarrow_r CDC(\text{peri}_R P)) \diamond (\text{pre}_R P \Rightarrow_r \text{post}_R P))$
 by (simp add: C2-CDC-form assms closure unrest rdes rpred CDC-idem)
 also have ... = $\mathbf{R}_s(\text{pre}_R P \vdash CDC(\text{pre}_R P \Rightarrow_r \text{peri}_R P) \diamond \text{post}_R P)$
 by (simp add: rpred unrest SRD-post-under-pre assms closure)
 also have ... = $\mathbf{R}_s(\text{pre}_R P \vdash CDC(\text{peri}_R P) \diamond \text{post}_R P)$
 by (simp add: unrest SRD-peri-under-pre assms closure)
 also have ... = $C2(P)$
 by (simp add: C2-CDC-form assms)
 finally show ?thesis .
 qed

lemma
 assumes $\text{vwb-lens } ns1 \text{ vwb-lens } ns2 \text{ } ns1 \bowtie ns2 \text{ } P \text{ is } RR$
 shows $P \text{ wr}[ns1|cs[ns2]]_C \text{ false} = \text{undefined}$ (is ?lhs = ?rhs)
 proof –
 have ?lhs = $(\neg_r (\exists (\text{ref}_0, \text{ref}_1, \text{st}_0, \text{st}_1, \text{tt}_0, \text{tt}_1) \cdot$
 $[\$ref' \mapsto_s \ll \text{ref}_0 \gg, \$st' \mapsto_s \ll \text{st}_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll \text{tt}_0 \gg] \dagger R1 \text{ true} \wedge$
 $[\$ref' \mapsto_s \ll \text{ref}_1 \gg, \$st' \mapsto_s \ll \text{st}_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll \text{tt}_1 \gg] \dagger P \wedge$
 $\$ref' \subseteq_u (\ll \text{ref}_0 \gg \cup_u \ll \text{ref}_1 \gg) \cap_u \ll cs \gg \cup_u (\ll \text{ref}_0 \gg \cap_u \ll \text{ref}_1 \gg - \ll cs \gg) \wedge$
 $\$tr \leq_u \$tr' \wedge$
 $\&tt \in_u \ll \text{tt}_0 \gg \star \ll cs \gg \ll \text{tt}_1 \gg \wedge \ll \text{tt}_0 \gg \upharpoonright_u \ll cs \gg =_u \ll \text{tt}_1 \gg \upharpoonright_u \ll cs \gg \wedge$
 $\$st' =_u \$st \oplus \ll \text{st}_0 \gg \text{ on } \&ns1 \oplus \ll \text{st}_1 \gg \text{ on } \&ns2) ;;$
 $R1 \text{ true})$
 by (simp add: wrR-def par-by-merge-seq-remove CSPInnerMerge-form assms closure usubst unrest)
 also have ... = $(\neg_r (\exists (\text{tt}_0, \text{tt}_1) \cdot$
 $[\$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll \text{tt}_1 \gg] \dagger P \wedge$
 $\$tr \leq_u \$tr' \wedge$
 $\&tt \in_u \ll \text{tt}_0 \gg \star \ll cs \gg \ll \text{tt}_1 \gg \wedge \ll \text{tt}_0 \gg \upharpoonright_u \ll cs \gg =_u \ll \text{tt}_1 \gg \upharpoonright_u \ll cs \gg) ;;$

```

      R1 true)
  by (rel-blast)
also have ... = ( $\neg_r$  ( $\exists$  ( $tt_0$ ,  $tt_1$ )  $\cdot$ 
  [ $\$tr \mapsto_s \langle \rangle$ ,  $\$tr' \mapsto_s \langle\langle tt_1 \rangle\rangle$ ]  $\dagger$   $RR(P)$ )  $\wedge$ 
   $\$tr \leq_u \$tr' \wedge$ 
   $\&tt \in_u \langle\langle tt_0 \rangle\rangle \star_{\langle\langle cs \rangle\rangle} \langle\langle tt_1 \rangle\rangle \wedge \langle\langle tt_0 \rangle\rangle \upharpoonright_u \langle\langle cs \rangle\rangle =_u \langle\langle tt_1 \rangle\rangle \upharpoonright_u \langle\langle cs \rangle\rangle$ ) ;;
      R1 true)
  by (simp add: Healthy-if-assms)
oops
end

```

5 Meta theory for Circus

```

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

```

References

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