Circus in Isabelle/UTP

Simon Foster James Baxter Ana Cavalcanti Jim Woodcock
Samuel Canham

October 21, 2021

Contents

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* ::
$'\vartheta$ *set* $\Rightarrow$ $'\vartheta$ *list* $\Rightarrow$ $'\vartheta$ *list* $\Rightarrow$ $'\vartheta$ *list set* **where**
tr-par cs [] [] = {[]} |
tr-par cs $(e \# t)$ [] = (*if* $e \in cs$ *then* {[]} *else* {[e]} $\frown$ (*tr-par cs t* [])) |
tr-par cs [] $(e \# t)$ = (*if* $e \in cs$ *then* {[]} *else* {[e]} $\frown$ (*tr-par cs* [] *t*)) |
tr-par cs $(e_1 \# t_1)$ $(e_2 \# t_2)$ =
(*if* $e_1 = e_2$
then
if $e_1 \in cs$
then $\{[e_1]\} \frown$ (*tr-par cs* t_1 t_2)
else
$(\{[e_1]\} \frown$ (*tr-par cs* t_1 $(e_2 \# t_2)$)) $\cup$
$(\{[e_2]\} \frown$ (*tr-par cs* $(e_1 \# t_1)$ t_2))
else
if $e_1 \in cs$ *then*
if $e_2 \in cs$ *then* {[]}
else
$\{[e_2]\} \frown$ (*tr-par cs* $(e_1 \# t_1)$ t_2)
else
if $e_2 \in cs$ *then*
$\{[e_1]\} \frown$ (*tr-par cs* t_1 $(e_2 \# t_2)$)
else
$\{[e_1]\} \frown$ (*tr-par cs* t_1 $(e_2 \# t_2)$) $\cup$
$\{[e_2]\} \frown$ (*tr-par cs* $(e_1 \# t_1)$ t_2))

abbreviation *tr-inter* :: $'\vartheta$ *list* $\Rightarrow$ $'\vartheta$ *list* $\Rightarrow$ $'\vartheta$ *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* == (*CONST bop*) (*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*: $x \mid\mid\mid_t y = y \mid\mid\mid_t x$
  by (*simp add*: *tr-par-sym*)

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

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

lemma *trace-merge-single-empty* [*simp*]:
  $a \in cs \Longrightarrow U([«a»]) \star_{cs} U([]) = U(\{[]\})$
  by (*rel-auto*)

lemma *trace-merge-empty-single* [*simp*]:
  $a \in cs \Longrightarrow U([]) \star_{cs} U([«a»]) = U(\{[]\})$
  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*]:
  $U(v + []) = v$ $U([] + v) = v$
  $bop\ (\#)\ x\ xs\ ^\frown_u\ ys = bop\ (\#)\ x\ (xs\ ^\frown_u\ ys)$
  by (*rel-auto*)+

Alternative characterisation of traces, adapted from CSP-Prover

inductive-set
  $parx :: 'a\ set => ('a\ list * 'a\ list * 'a\ list)\ set$
  for $X :: 'a\ set$

where
parx-nil-nil [*intro*]:
  $([], [], []) \in parx\ X$ |

parx-Ev-nil [*intro*]:
  $[|\ (u, s, []) \in parx\ X\ ;\ a \notin X\ |]$
   $==> (a\ \#\ u,\ a\ \#\ s,\ []) \in parx\ X$ |

parx-nil-Ev [*intro*]:
  $[|\ (u, [], t) \in parx\ X\ ;\ a \notin X\ |]$
   $==> (a\ \#\ u,\ [],\ a\ \#\ t) \in parx\ X$ |

parx-Ev-sync [*intro*]:
  $[|\ (u, s, t) \in parx\ X\ ;\ a \in X\ |]$
   $==> (a\ \#\ u,\ a\ \#\ s,\ a\ \#\ t) \in parx\ X$ |

parx-Ev-left [*intro*]:
  [| (*u*, *s*, *t*) ∈ *parx X* ; *a* ∉ *X* |]
   ==> (*a* # *u*, *a* # *s*, *t*) ∈ *parx X* |

parx-Ev-right [*intro*]:
  [| (*u*, *s*, *t*) ∈ *parx X* ; *a* ∉ *X* |]
   ==> (*a* # *u*, *s*, *a* # *t*) ∈ *parx X*

lemma *parx-implies-tr-par*: $(t,\ t_1,\ t_2) \in parx\ cs \Longrightarrow t \in tr\text{-}par\ cs\ t_1\ t_2$
  apply (*induct rule*: *parx.induct*)
      apply (*auto*)
   apply (*case-tac t*)
    apply (*auto*)
  apply (*case-tac s*)
   apply (*auto*)
  done

end

3 Syntax and Translations for Event Prefix

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

syntax
  -simple-prefix :: *logic* ⇒ *logic* ⇒ *logic* (- ⟶ - [*63*, *62*] *62*)

translations
  a → *P* == *CONST PrefixCSP* «*a*» *P*

We next configure a syntax for mixed prefixes.

nonterminal *prefix-elem′* **and** *mixed-prefix′*

syntax *-end-prefix* :: *prefix-elem′* ⇒ *mixed-prefix′* (-)

Input Prefix: . . .?(*x*)

syntax *-simple-input-prefix* :: *id* ⇒ *prefix-elem′* (?′(-′))

Input Prefix with Constraint: . . .?(*x* : *P*)

syntax *-input-prefix* :: *id* ⇒ (′σ, ′ε) *action* ⇒ *prefix-elem′* (?′(- :/ -′))

Output Prefix: . . .![*v*]*e*

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

syntax *-output-prefix* :: *logic* ⇒ *prefix-elem′* (!′(-′))
syntax *-output-prefix* :: *logic* ⇒ *prefix-elem′* (.′(-′))

syntax (**output**) *-output-prefix-pp* :: *logic* ⇒ *prefix-elem′* (!′(-′))

syntax
  -prefix-aux :: *pttrn* ⇒ *logic* ⇒ *prefix-elem′*

Mixed-Prefix Action: *c*. . .(*prefix*) ⟶ *A*

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

syntax
  -prefix-action ::
  (*'a*, *'ε*) *chan* ⇒ *mixed-prefix'* ⇒ (*'σ*, *'ε*) *action* ⇒ (*'σ*, *'ε*) *action*
  ((-- →/ -) [*63*, *63*, *62*] *62*)

Syntax translations

definition *lconj* :: (*'a* ⇒ *'α upred*) ⇒ (*'b* ⇒ *'α upred*) ⇒ (*'a* × *'b* ⇒ *'α upred*) (**infixr** $\wedge_l$ *35*)
where [*upred-defs*]: ($P \wedge_l Q$) ≡ (λ (*x*,*y*). *P x* ∧ *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*) $\wedge_l$ *Q*)
  -mixed-prefix (*-output-prefix P*) (*-prefix-aux y Q*) ⇀
  -prefix-aux (*-pattern -idtdummy y*) ((*CONST outp-constraint P*) $\wedge_l$ *Q*)
  -end-prefix (*-input-prefix x P*) ⇀ *-prefix-aux x* (λ *x*. *P*)
  -end-prefix (*-output-prefix P*) ⇀ *-prefix-aux -idtdummy* (*CONST outp-constraint P*)
  -prefix-action c (*-prefix-aux x P*) *A* == (*CONST InputCSP*) *c P* (λ*x*. *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* (λ*x*. *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*!(*U*([*1*])) → *P*
term *x*?(*v*)!(*1*) → *P*
term *x*!(*U*([*1*]))!(*2*)?(*v*:*true*) → *P*

Basic translations for state variable communications

syntax
  -csp-input-var :: *logic* ⇒ *id* ⇒ *logic* ⇒ *logic* (-?'(-:-') [*63*, *0*, *0*] *62*)
  -csp-inputu-var :: *logic* ⇒ *id* ⇒ *logic* (-?'(-') [*63*, *0*] *62*)
  -csp-output-var :: *logic* ⇒ *logic* ⇒ *logic* (-!'(-') [*63*, *0*] *62*)

translations
  c?(*x*:*A*) ⇀ *CONST InputVarCSP c x A*
  c?(*x*) ⇀ *CONST InputVarCSP c x* (λ *x*. *true*)
  c?(*x*:*A*) <= *CONST InputVarCSP c x* (λ *x'*. *A*)
  c?(*x*) <= *c*?(*x*:*true*)
  -csp-output-var c e == *CONST DoCSP* $(c{\cdot}e)_u$

lemma *outp-constraint-prod*:
  (*outp-constraint* «*a*» *x* ∧ *outp-constraint* «*b*» *y*) =

outp-constraint «(a, b)» (x, y)
by (*simp add*: *outp-constraint-def*, *pred-auto*)

lemma *subst-outp-constraint* [*usubst*]:
$\sigma \dagger (v =_o x) = (\sigma \dagger v =_o x)$
by (*rel-auto*)

lemma *UINF-one-point-simp* [*rpred*]:
$[\![\bigwedge i.\ P\ i\ is\ R1]\!] \Longrightarrow (\bigsqcap\ x \cdot [«i» =_o x]_{S<} \wedge P(x)) = P(i)$
by (*rel-blast*)

lemma *USUP-one-point-simp* [*rpred*]:
$[\![\bigwedge i.\ P\ i\ is\ R1]\!] \Longrightarrow (\bigsqcup\ x \cdot [«i» =_o x]_{S<} \Rightarrow_r P(x)) = P(i)$
by (*rel-blast*)

lemma *USUP-eq-event-eq* [*rpred*]:
assumes $\bigwedge y.\ P(y)\ is\ RR$
shows $(\bigsqcup\ y \cdot [v =_o y]_{S<} \Rightarrow_r P(y)) = P(y)[\![y \rightarrow \lceil v \rceil_{S\leftarrow}]\!]$
proof –
have $(\bigsqcup\ y \cdot [v =_o y]_{S<} \Rightarrow_r RR(P(y))) = RR(P(y))[\![y \rightarrow \lceil v \rceil_{S\leftarrow}]\!]$
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 $(\bigsqcap\ y \cdot [v =_o y]_{S<} \wedge P(y)) = P(y)[\![y \rightarrow \lceil v \rceil_{S\leftarrow}]\!]$
proof –
have $(\bigsqcap\ y \cdot [v =_o y]_{S<} \wedge RR(P(y))) = RR(P(y))[\![y \rightarrow \lceil v \rceil_{S\leftarrow}]\!]$
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 $x!(P) \rightarrow A = OutputCSP\ x\ P\ A$ (**is** *?lhs* = *?rhs*)
by (*rdes-eq cls*: *assms*)

lemma *OutputCSP-pair-simp* [*simp*]:
P is NCSP $\Longrightarrow$ $a.(«i»).(«j») \rightarrow P = OutputCSP\ a\ «(i,j)»\ P$
using *output-prefix-is-OutputCSP*[*of P a*]
by (*simp add*: *outp-constraint-prod lconj-def InputCSP-def closure del*: *output-prefix-is-OutputCSP*)

lemma *OutputCSP-triple-simp* [*simp*]:
P is NCSP $\Longrightarrow$ $a.(«i»).(«j»).(«k») \rightarrow P = OutputCSP\ a\ «(i,j,k)»\ P$
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* :: $('\alpha \Longrightarrow '\sigma) \Rightarrow '\psi\ set \Rightarrow ('\beta \Longrightarrow '\sigma) \Rightarrow (('\sigma,'\psi)\ sfrd)\ merge\ (N_C)$ **where**
  [*upred-defs*]:
  $CSPInnerMerge\ ns1\ cs\ ns2 = ($
    $\$ref \subseteq_u ((\$0{:}ref \cup_u \$1{:}ref) \cap_u «cs») \cup_u ((\$0{:}ref \cap_u \$1{:}ref) - «cs») \wedge$
    $\$<{:}tr \leq_u \$tr' \wedge$
    $(\$tr' - \$<{:}tr) \in_u (\$0{:}tr - \$<{:}tr) \star_{cs} (\$1{:}tr - \$<{:}tr) \wedge$
    $(\$0{:}tr - \$<{:}tr) \upharpoonright_u «cs» =_u (\$1{:}tr - \$<{:}tr) \upharpoonright_u «cs» \wedge$
    $\$st' =_u (\$<{:}st \oplus \$0{:}st\ on\ \&ns1) \oplus \$1{:}st\ on\ \&ns2)$

definition *CSPInnerInterleave* :: $('\alpha \Longrightarrow '\sigma) \Rightarrow ('\beta \Longrightarrow '\sigma) \Rightarrow (('\sigma,'\psi)\ sfrd)\ merge\ (N_I)$ **where**
  [*upred-defs*]:
  $N_I\ ns1\ ns2 = ($
    $\$ref \subseteq_u (\$0{:}ref \cap_u \$1{:}ref) \wedge$
    $\$<{:}tr \leq_u \$tr' \wedge$
    $(\$tr' - \$<{:}tr) \in_u (\$0{:}tr - \$<{:}tr) \star_{\{\}} (\$1{:}tr - \$<{:}tr) \wedge$
    $\$st' =_u (\$<{:}st \oplus \$0{:}st\ on\ \&ns1) \oplus \$1{:}st\ on\ \&ns2)$

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

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

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

syntax
  -cinter-merge :: $logic \Rightarrow logic \Rightarrow logic \Rightarrow logic$ (- $[\![\text{-}]\!]^I$ - [85,0,86] 86)
  -cfinal-merge :: $logic \Rightarrow salpha \Rightarrow logic \Rightarrow salpha \Rightarrow logic \Rightarrow logic$ (- $[\![\text{-}|\text{-}|\text{-}]\!]^F$ - [85,0,0,0,86] 86)
  -wrC :: $logic \Rightarrow logic \Rightarrow logic \Rightarrow logic$ (- $wr[\text{-}]_C$ - [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 == $P\ wr_R(N_C\ 0_L\ cs\ 0_L)\ Q$

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*]:
  $\$<:ok \sharp N_C\ ns1\ cs\ ns2$
  $\$<:wait \sharp N_C\ ns1\ cs\ ns2$
  by (*rel-auto*)+

lemma *CSPInterMerge-RR-closed* [*closure*]:
  assumes *P is RR Q is RR*
  shows $P\ [\![cs]\!]^I\ Q$ *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 $\$ref \sharp P\ [\![cs]\!]^I\ Q$
proof −
  have $\$ref \sharp CRR(P)\ [\![cs]\!]^I\ CRR(Q)$
    by (*rel-blast*)
  thus *?thesis*
    by (*simp add*: *Healthy-if assms*)
qed

lemma *CSPInterMerge-unrest-st′* [*unrest*]:
  $\$st' \sharp P\ [\![cs]\!]^I\ Q$
  by (*rel-auto*)

lemma *CSPInterMerge-CRR-closed* [*closure*]:
  assumes *P is CRR Q is CRR*
  shows $P\ [\![cs]\!]^I\ Q$ *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 $P\ [\![ns1|cs|ns2]\!]^F\ Q$ *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 $\$ref \sharp P\ [\![ns1|cs|ns2]\!]^F\ Q$
proof −
  have $\$ref \sharp CRR(P)\ [\![ns1|cs|ns2]\!]^F\ CRR(Q)$
    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 $P\ [\![ns1|cs|ns2]\!]^F\ Q$ *is CRR*
 by (*simp add*: *CRR-implies-RR CRR-intro CSPFinalMerge-RR-closed CSPFinalMerge-unrest-ref assms*)

lemma *CSPFinalMerge-unrest-ref'* [*unrest*]:
  assumes P *is* CRR Q *is* CRR
  shows $\$ref \sharp P [\![ns1|cs|ns2]\!]^F \; Q$
proof −
  have $\$ref \sharp CRR(P) [\![ns1|cs|ns2]\!]^F \; CRR(Q)$
    by (*rel-blast*)
  thus *?thesis*
    by (*simp add*: *Healthy-if assms*)
qed

lemma *CSPFinalMerge-CRF-closed* [*closure*]:
  assumes P *is* CRF Q *is* CRF
  shows $P [\![ns1|cs|ns2]\!]^F \; Q$ *is* CRF
  by (*rule CRF-intro*, *simp-all add*: *assms unrest closure*)

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

definition *CSPMerge* :: $('\alpha \Longrightarrow '\sigma) \Rightarrow '\psi \; set \Rightarrow ('\beta \Longrightarrow '\sigma) \Rightarrow (('\sigma,'\psi) \; sfrd) \; merge \; (M_C)$ **where**
[*upred-defs*]: $M_C \; ns1 \; cs \; ns2 = M_R(N_C \; ns1 \; cs \; ns2) \; ;; \; Skip$

definition *CSPInterleave* :: $('\alpha \Longrightarrow '\sigma) \Rightarrow ('\beta \Longrightarrow '\sigma) \Rightarrow (('\sigma,'\psi) \; sfrd) \; merge \; (M_I)$ **where**
[*upred-defs*]: $M_I \; ns1 \; ns2 = M_R(N_I \; ns1 \; ns2) \; ;; \; Skip$

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

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

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

lemma *SymMerge-CSPInnerInterleave* [*closure*]:
  AssocMerge $(N_I \; 0_L \; 0_L)$
  apply (*rel-auto*)
  apply (*rename-tac* $tr \; tr_2{}' \; ref_0 \; tr_0{}' \; ref_0{}' \; tr_1{}' \; ref_1{}' \; tr' \; ref_2{}' \; tr_i{}' \; ref_3{}'$)
oops

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

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

lemma *CSPFinalMerge-right-false* [*rpred*]: $P [\![ns1|cs|ns2]\!]^F \; false = false$

by (*simp add*: *CSPFinalMerge-def*)

lemma *CSPFinalMerge-left-false* [*rpred*]: $false \ [\![ns1|cs|ns2]\!]^F \ P = false$
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 $... = Q \parallel_{N_C \ ns2 \ cs \ ns1} P$
by (*metis par-by-merge-commute-swap*)
finally show *?thesis* **.**
qed

lemma *CSPInterMerge-commute*:
$P \ [\![cs]\!]^I \ Q = Q \ [\![cs]\!]^I \ P$
proof –
have $P \ [\![cs]\!]^I \ Q = P \parallel_{\exists \ \$st\acute{} \ \cdot \ N_C \ 0_L \ cs \ 0_L} Q$
by (*simp add*: *CSPInterMerge-def*)
also have $... = P \parallel_{\exists \ \$st\acute{} \ \cdot \ (swap_m \ ;; \ N_C \ 0_L \ cs \ 0_L)} Q$
by (*simp add*: *swap-CSPInnerMerge lens-indep-sym*)
also have $... = P \parallel_{swap_m \ ;; \ (\exists \ \$st\acute{} \ \cdot \ N_C \ 0_L \ cs \ 0_L)} Q$
by (*simp add*: *seqr-exists-right*)
also have $... = Q \parallel_{(\exists \ \$st\acute{} \ \cdot \ N_C \ 0_L \ cs \ 0_L)} P$
by (*simp add*: *par-by-merge-commute-swap*[*THEN sym*])
also have $... = Q \ [\![cs]\!]^I \ P$
by (*simp add*: *CSPInterMerge-def*)
finally show *?thesis* **.**
qed

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

Important theorem that shows the form of a parallel process

lemma *CSPInnerMerge-form*:
fixes $P \ Q :: ('\sigma,'\varphi)$ *action*
assumes *vwb-lens ns1 vwb-lens ns2 P is RR Q is RR*
shows

$P \parallel_{N_C}\ \mathit{ns1}\ \mathit{cs}\ \mathit{ns2}\ Q =$
$(\exists\ (\mathit{ref}_0,\ \mathit{ref}_1,\ \mathit{st}_0,\ \mathit{st}_1,\ \mathit{tt}_0,\ \mathit{tt}_1) \cdot$
$P[\![«\mathit{ref}_0»,«\mathit{st}_0»,«[]»,«\mathit{tt}_0»/\$\mathit{ref},\$\acute{\mathit{st}},\$\mathit{tr},\$\acute{\mathit{tr}}]\!] \land Q[\![«\mathit{ref}_1»,«\mathit{st}_1»,«[]»,«\mathit{tt}_1»/\$\mathit{ref},\$\acute{\mathit{st}},\$\mathit{tr},\$\acute{\mathit{tr}}]\!]$
$\land\ \$\mathit{ref} \subseteq_u ((«\mathit{ref}_0» \cup_u «\mathit{ref}_1») \cap_u «\mathit{cs}») \cup_u ((«\mathit{ref}_0» \cap_u «\mathit{ref}_1») - «\mathit{cs}»)$
$\land\ \$\mathit{tr} \le_u \$\acute{\mathit{tr}}$
$\land\ \&\mathit{tt} \in_u «\mathit{tt}_0» \star_{\mathit{cs}} «\mathit{tt}_1»$
$\land\ «\mathit{tt}_0» \upharpoonright_u «\mathit{cs}» =_u «\mathit{tt}_1» \upharpoonright_u «\mathit{cs}»$
$\land\ \$\acute{\mathit{st}} =_u (\$\mathit{st} \oplus «\mathit{st}_0»\ \mathit{on}\ \&\mathit{ns1}) \oplus «\mathit{st}_1»\ \mathit{on}\ \&\mathit{ns2})$
(**is** *?lhs* = *?rhs*)
proof –
have P:$(\exists\ \{\$\acute{\mathit{ok}},\$\acute{\mathit{wait}}\} \cdot \mathit{R2}(P)) = P$ (**is** $?P' = $ -)
by (*simp add*: *ex-unrest ex-plus Healthy-if assms unrest closure*)
have Q:$(\exists\ \{\$\acute{\mathit{ok}},\$\acute{\mathit{wait}}\} \cdot \mathit{R2}(Q)) = Q$ (**is** $?Q' = $ -)
by (*simp add*: *ex-unrest ex-plus Healthy-if assms unrest closure*)
from *assms*(*1*,*2*)
have $?P' \parallel_{N_C}\ \mathit{ns1}\ \mathit{cs}\ \mathit{ns2}\ ?Q' =$
$(\exists\ (\mathit{ref}_0,\ \mathit{ref}_1,\ \mathit{st}_0,\ \mathit{st}_1,\ \mathit{tt}_0,\ \mathit{tt}_1) \cdot$
$?P'[\![«\mathit{ref}_0»,«\mathit{st}_0»,«[]»,«\mathit{tt}_0»/\$\mathit{ref},\$\acute{\mathit{st}},\$\mathit{tr},\$\acute{\mathit{tr}}]\!] \land ?Q'[\![«\mathit{ref}_1»,«\mathit{st}_1»,«[]»,«\mathit{tt}_1»/\$\mathit{ref},\$\acute{\mathit{st}},\$\mathit{tr},\$\acute{\mathit{tr}}]\!]$
$\land\ \$\mathit{ref} \subseteq_u ((«\mathit{ref}_0» \cup_u «\mathit{ref}_1») \cap_u «\mathit{cs}») \cup_u ((«\mathit{ref}_0» \cap_u «\mathit{ref}_1») - «\mathit{cs}»)$
$\land\ \$\mathit{tr} \le_u \$\acute{\mathit{tr}}$
$\land\ \&\mathit{tt} \in_u «\mathit{tt}_0» \star_{\mathit{cs}} «\mathit{tt}_1»$
$\land\ «\mathit{tt}_0» \upharpoonright_u «\mathit{cs}» =_u «\mathit{tt}_1» \upharpoonright_u «\mathit{cs}»$
$\land\ \$\acute{\mathit{st}} =_u (\$\mathit{st} \oplus «\mathit{st}_0»\ \mathit{on}\ \&\mathit{ns1}) \oplus «\mathit{st}_1»\ \mathit{on}\ \&\mathit{ns2})$
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* :: $('\sigma,'\varphi)$ *action*
assumes *P is RR Q is RR*
shows
$P\ [\![\mathit{cs}]\!]^I\ Q =$
$(\exists\ (\mathit{ref}_0,\ \mathit{ref}_1,\ \mathit{st}_0,\ \mathit{st}_1,\ \mathit{tt}_0,\ \mathit{tt}_1) \cdot$
$P[\![«\mathit{ref}_0»,«\mathit{st}_0»,«[]»,«\mathit{tt}_0»/\$\mathit{ref},\$\acute{\mathit{st}},\$\mathit{tr},\$\acute{\mathit{tr}}]\!] \land Q[\![«\mathit{ref}_1»,«\mathit{st}_1»,«[]»,«\mathit{tt}_1»/\$\mathit{ref},\$\acute{\mathit{st}},\$\mathit{tr},\$\acute{\mathit{tr}}]\!]$
$\land\ \$\mathit{ref} \subseteq_u ((«\mathit{ref}_0» \cup_u «\mathit{ref}_1») \cap_u «\mathit{cs}») \cup_u ((«\mathit{ref}_0» \cap_u «\mathit{ref}_1») - «\mathit{cs}»)$
$\land\ \$\mathit{tr} \le_u \$\acute{\mathit{tr}}$
$\land\ \&\mathit{tt} \in_u «\mathit{tt}_0» \star_{\mathit{cs}} «\mathit{tt}_1»$
$\land\ «\mathit{tt}_0» \upharpoonright_u «\mathit{cs}» =_u «\mathit{tt}_1» \upharpoonright_u «\mathit{cs}»)$
(**is** *?lhs* = *?rhs*)
proof –
have $?lhs = (\exists\ \$\acute{\mathit{st}} \cdot P \parallel_{N_C}\ \mathit{0}_L\ \mathit{cs}\ \mathit{0}_L\ Q)$
by (*simp add*: *CSPInterMerge-def par-by-merge-def seqr-exists-right*)

also have ... =
$(\exists\ \$\acute{st} \cdot$
$(\exists\ (ref_0,\ ref_1,\ st_0,\ st_1,\ tt_0,\ tt_1) \cdot$
$P[\![«ref_0»,«st_0»,«[]»,«tt_0»/\$ref,\$\acute{st},\$tr,\$\acute{tr}]\!] \wedge Q[\![«ref_1»,«st_1»,«[]»,«tt_1»/\$ref,\$\acute{st},\$tr,\$\acute{tr}]\!]$
$\wedge\ \$ref \subseteq_u ((«ref_0» \cup_u «ref_1») \cap_u «cs») \cup_u ((«ref_0» \cap_u «ref_1») - «cs»)$
$\wedge\ \$tr \leq_u \$\acute{tr}$
$\wedge\ \&tt \in_u «tt_0» \star_{cs} «tt_1»$
$\wedge\ «tt_0» \restriction_u «cs» =_u «tt_1» \restriction_u «cs»$
$\wedge\ \$\acute{st} =_u (\$st \oplus «st_0» \ on\ \emptyset) \oplus «st_1» \ on\ \emptyset))$
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* $\$ref \sharp P$ $\$ref \sharp Q$
shows
$(P\ [\![ns1|cs|ns2]\!]^F\ Q) =$
$(\exists\ (st_0,\ st_1,\ tt_0,\ tt_1) \cdot$
$P[\![«st_0»,«[]»,«tt_0»/\$\acute{st},\$tr,\$\acute{tr}]\!] \wedge Q[\![«st_1»,«[]»,«tt_1»/\$\acute{st},\$tr,\$\acute{tr}]\!]$
$\wedge\ \$tr \leq_u \$\acute{tr}$
$\wedge\ \&tt \in_u «tt_0» \star_{cs} «tt_1»$
$\wedge\ «tt_0» \restriction_u «cs» =_u «tt_1» \restriction_u «cs»$
$\wedge\ \$\acute{st} =_u (\$st \oplus «st_0» \ on\ \&ns1) \oplus «st_1» \ on\ \&ns2)$
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = $(\exists\ \$ref \cdot P \parallel_{N_C}\ ns1\ cs\ ns2\ Q)$
by (*simp add*: *CSPFinalMerge-def par-by-merge-def seqr-exists-right*)
also have ... =
$(\exists\ \$ref \cdot$
$(\exists\ (ref_0,\ ref_1,\ st_0,\ st_1,\ tt_0,\ tt_1) \cdot$
$P[\![«ref_0»,«st_0»,«[]»,«tt_0»/\$ref,\$\acute{st},\$tr,\$\acute{tr}]\!] \wedge Q[\![«ref_1»,«st_1»,«[]»,«tt_1»/\$ref,\$\acute{st},\$tr,\$\acute{tr}]\!]$
$\wedge\ \$ref \subseteq_u ((«ref_0» \cup_u «ref_1») \cap_u «cs») \cup_u ((«ref_0» \cap_u «ref_1») - «cs»)$
$\wedge\ \$tr \leq_u \$\acute{tr}$
$\wedge\ \&tt \in_u «tt_0» \star_{cs} «tt_1»$
$\wedge\ «tt_0» \restriction_u «cs» =_u «tt_1» \restriction_u «cs»$
$\wedge\ \$\acute{st} =_u (\$st \oplus «st_0» \ on\ \&ns1) \oplus «st_1» \ on\ \&ns2))$
by (*simp add*: *CSPInnerMerge-form assms*)
also have ... =
$(\exists\ \$ref \cdot$
$(\exists\ (ref_0,\ ref_1,\ st_0,\ st_1,\ tt_0,\ tt_1) \cdot$
$(\exists\ \$ref \cdot P)[\![«ref_0»,«st_0»,«[]»,«tt_0»/\$ref,\$\acute{st},\$tr,\$\acute{tr}]\!] \wedge (\exists\ \$ref \cdot Q)[\![«ref_1»,«st_1»,«[]»,«tt_1»/\$ref,\$\acute{st},\$tr,\$\acute{tr}]\!]$
$\wedge\ \$ref \subseteq_u ((«ref_0» \cup_u «ref_1») \cap_u «cs») \cup_u ((«ref_0» \cap_u «ref_1») - «cs»)$
$\wedge\ \$tr \leq_u \$\acute{tr}$
$\wedge\ \&tt \in_u «tt_0» \star_{cs} «tt_1»$
$\wedge\ «tt_0» \restriction_u «cs» =_u «tt_1» \restriction_u «cs»$
$\wedge\ \$\acute{st} =_u (\$st \oplus «st_0» \ on\ \&ns1) \oplus «st_1» \ on\ \&ns2))$
by (*simp add*: *ex-unrest assms*)
also have ... =
$(\exists\ (st_0,\ st_1,\ tt_0,\ tt_1) \cdot$
$(\exists\ \$ref \cdot P)[\![«st_0»,«[]»,«tt_0»/\$\acute{st},\$tr,\$\acute{tr}]\!] \wedge (\exists\ \$ref \cdot Q)[\![«st_1»,«[]»,«tt_1»/\$\acute{st},\$tr,\$\acute{tr}]\!]$
$\wedge\ \$tr \leq_u \$\acute{tr}$
$\wedge\ \&tt \in_u «tt_0» \star_{cs} «tt_1»$

$\wedge$ «tt_0» $\restriction_u$ «cs» $=_u$ «tt_1» $\restriction_u$ «cs»
$\wedge$ $\$st\acute{} =_u (\$st \oplus$ «st_0» *on* &$ns1$) $\oplus$ «st_1» *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* $\wedge$ \$<:$st\acute{}$ $=_u$ \$$st$ $\wedge$ \$<:$tr\acute{}$ $=_u$ \$$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 is CRR Q is CRR*
shows P *wr*$[cs]_C$ Q *is CRC*
proof –
have \$*ref* $\sharp$ 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* $\sharp$ P $[\![ns1|cs|ns2]\!]^F$ Q
by (*rel-auto*)

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

lemma *CSPInterMerge-alt-def*:
P $[\![cs]\!]^I$ Q = ($\exists$ \$$st\acute{}$ $\cdot$ P $\|_{N_C\ 0_L\ cs\ 0_L}$ Q)
by (*simp add*: *par-by-merge-def CSPInterMerge-def seqr-exists-right*)

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

lemma *merge-csp-do-left*:
assumes *vwb-lens ns1 vwb-lens ns2 ns1* $\bowtie$ *ns2 P is RR*
shows $\Phi(s_0,\sigma_0,t_0)$ $\|_{N_C\ ns1\ cs\ ns2}$ P =
($\exists$ (ref_1, st_1, tt_1) $\cdot$
$[s_0]_{S<}$ $\wedge$
[\$*ref* $\mapsto_s$ «ref_1», \$$st\acute{}$ $\mapsto_s$ «st_1», \$$tr$ $\mapsto_s$ «[]», \$$tr\acute{}$ $\mapsto_s$ «tt_1»] † P $\wedge$
\$*ref* $\subseteq_u$ «cs» $\cup_u$ («ref_1» − «cs») $\wedge$
[«*trace*» $\in_u$ t_0 $\star_{cs}$ «tt_1» $\wedge$ t_0 $\restriction_u$ «cs» $=_u$ «tt_1» $\restriction_u$ «cs»$]_t$ $\wedge$
\$$st\acute{}$ $=_u$ \$$st$ $\oplus$ (&**v** $\mapsto_s$ \$$st$) † σ_0 *on* &$ns1$ $\oplus$ «st_1» *on* &$ns2$)

(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* =
$(\exists\ (ref_0,\ ref_1,\ st_0,\ st_1,\ tt_0,\ tt_1) \cdot$
$[\$ref \mapsto_s «ref_0»,\ \$\acute{st} \mapsto_s «st_0»,\ \$tr \mapsto_s «[]»,\ \$\acute{tr} \mapsto_s «tt_0»] \dagger \Phi(s_0,\sigma_0,t_0) \wedge$
$[\$ref \mapsto_s «ref_1»,\ \$\acute{st} \mapsto_s «st_1»,\ \$tr \mapsto_s «[]»,\ \$\acute{tr} \mapsto_s «tt_1»] \dagger P \wedge$
$\$ref \subseteq_u («ref_0» \cup_u «ref_1») \cap_u «cs» \cup_u («ref_0» \cap_u «ref_1» - «cs») \wedge$
$\$tr \leq_u \$\acute{tr} \wedge$
$\&tt \in_u «tt_0» \star_{cs} «tt_1» \wedge «tt_0» \upharpoonright_u «cs» =_u «tt_1» \upharpoonright_u «cs» \wedge \$\acute{st} =_u \$st \oplus «st_0»\ on\ \&ns1 \oplus «st_1»$
$on\ \&ns2)$
by (*simp add*: *CSPInnerMerge-form assms closure*)
also have ... =
$(\exists\ (ref_1,\ st_1,\ tt_1) \cdot$
$[s_0]_{S<} \wedge$
$[\$ref \mapsto_s «ref_1»,\ \$\acute{st} \mapsto_s «st_1»,\ \$tr \mapsto_s «[]»,\ \$\acute{tr} \mapsto_s «tt_1»] \dagger P \wedge$
$\$ref \subseteq_u «cs» \cup_u («ref_1» - «cs») \wedge$
$[«trace» \in_u t_0 \star_{cs} «tt_1» \wedge t_0 \upharpoonright_u «cs» =_u «tt_1» \upharpoonright_u «cs»]_t \wedge$
$\$\acute{st} =_u \$st \oplus (\!|\&\mathbf{v} \mapsto_s \$st|\!) \dagger \sigma_0\ on\ \&ns1 \oplus «st_1»\ on\ \&ns2)$
by (*rel-blast*)
finally show *?thesis* **.**
qed

lemma *merge-csp-do-right*:
assumes *vwb-lens ns1 vwb-lens ns2 ns1* $\bowtie$ *ns2 P is RR*
shows $P \parallel_{N_C\ ns1\ cs\ ns2} \Phi(s_1,\sigma_1,t_1) =$
$(\exists\ (ref_0,\ st_0,\ tt_0) \cdot$
$[\$ref \mapsto_s «ref_0»,\ \$\acute{st} \mapsto_s «st_0»,\ \$tr \mapsto_s «[]»,\ \$\acute{tr} \mapsto_s «tt_0»] \dagger P \wedge$
$[s_1]_{S<} \wedge$
$\$ref \subseteq_u «cs» \cup_u («ref_0» - «cs») \wedge$
$[«trace» \in_u «tt_0» \star_{cs} t_1 \wedge «tt_0» \upharpoonright_u «cs» =_u t_1 \upharpoonright_u «cs»]_t \wedge$
$\$\acute{st} =_u \$st \oplus «st_0»\ on\ \&ns1 \oplus (\!|\&\mathbf{v} \mapsto_s \$st|\!) \dagger \sigma_1\ on\ \&ns2\ )$
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = $\Phi(s_1,\sigma_1,t_1) \parallel_{N_C\ ns2\ cs\ ns1} P$
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 $P \parallel_{N_C\ ns1\ cs\ ns2} \mathcal{E}(s_0,t_0,E_0) =$
$(\exists\ (ref_0,\ ref_1,\ st_0,\ st_1,\ tt_0) \cdot$
$[s_0]_{S<} \wedge$
$[\$ref \mapsto_s «ref_0»,\ \$\acute{st} \mapsto_s «st_0»,\ \$tr \mapsto_s «[]»,\ \$\acute{tr} \mapsto_s «tt_0»] \dagger P \wedge$
$(\forall\ e \cdot «e» \in_u \lceil E_0 \rceil_{S<} \Rightarrow «e» \notin_u «ref_1») \wedge$
$\$ref \subseteq_u («ref_0» \cup_u «ref_1») \cap_u «cs» \cup_u («ref_0» \cap_u «ref_1» - «cs») \wedge$
$[«trace» \in_u «tt_0» \star_{cs} t_0 \wedge «tt_0» \upharpoonright_u «cs» =_u t_0 \upharpoonright_u «cs»]_t \wedge$
$\$\acute{st} =_u \$st \oplus «st_0»\ on\ \&ns1 \oplus «st_1»\ on\ \&ns2)$
(**is** *?lhs* = *?rhs*)

proof −
  have $?lhs = (\exists\ (ref_0,\ ref_1,\ st_0,\ st_1,\ tt_0,\ tt_1) \cdot$
      $[\$ref \mapsto_s «ref_0», \$st\acute{} \mapsto_s «st_0», \$tr \mapsto_s «[]», \$tr\acute{} \mapsto_s «tt_0»] \dagger P \wedge$
      $[\$ref \mapsto_s «ref_1», \$tr \mapsto_s «[]», \$tr\acute{} \mapsto_s «tt_1»] \dagger \mathcal{E}(s_0,t_0,\ E_0) \wedge$
      $\$ref \subseteq_u («ref_0» \cup_u «ref_1») \cap_u «cs» \cup_u («ref_0» \cap_u «ref_1» - «cs») \wedge$
      $\$tr \leq_u \$tr\acute{} \wedge \&tt \in_u «tt_0» \star_{cs} «tt_1» \wedge «tt_0» \upharpoonright_u «cs» =_u «tt_1» \upharpoonright_u «cs» \wedge \$st\acute{} =_u \$st \oplus «st_0»$ *on* $\&ns1 \oplus «st_1»$ *on* $\&ns2)$
    by (*simp add*: *CSPInnerMerge-form assms closure unrest usubst*)
  also have ... $= (\exists\ (ref_0,\ ref_1,\ st_0,\ st_1,\ tt_0,\ tt_1) \cdot [\$ref \mapsto_s «ref_0», \$st\acute{} \mapsto_s «st_0», \$tr \mapsto_s «[]», \$tr\acute{} \mapsto_s «tt_0»] \dagger P \wedge$
      $(\lceil s_0 \rceil_{S<} \wedge «tt_1» =_u \lceil t_0 \rceil_{S<} \wedge (\forall\ e \cdot «e» \in_u \lceil E_0 \rceil_{S<} \Rightarrow «e» \notin_u «ref_1»)) \wedge$
      $\$ref \subseteq_u («ref_0» \cup_u «ref_1») \cap_u «cs» \cup_u («ref_0» \cap_u «ref_1» - «cs») \wedge$
      $\$tr \leq_u \$tr\acute{} \wedge \&tt \in_u «tt_0» \star_{cs} «tt_1» \wedge «tt_0» \upharpoonright_u «cs» =_u «tt_1» \upharpoonright_u «cs» \wedge \$st\acute{} =_u \$st \oplus «st_0»$ *on* $\&ns1 \oplus «st_1»$ *on* $\&ns2)$
    by (*simp add*: *csp-enable-def usubst unrest*)
  also have ... $= (\exists\ (ref_0,\ ref_1,\ st_0,\ st_1,\ tt_0) \cdot$
      $[s_0]_{S<} \wedge$
      $[\$ref \mapsto_s «ref_0», \$st\acute{} \mapsto_s «st_0», \$tr \mapsto_s «[]», \$tr\acute{} \mapsto_s «tt_0»] \dagger P \wedge$
      $(\forall\ e \cdot «e» \in_u \lceil E_0 \rceil_{S<} \Rightarrow «e» \notin_u «ref_1») \wedge$
      $\$ref \subseteq_u («ref_0» \cup_u «ref_1») \cap_u «cs» \cup_u («ref_0» \cap_u «ref_1» - «cs») \wedge$
      $[«trace» \in_u «tt_0» \star_{cs} t_0 \wedge «tt_0» \upharpoonright_u «cs» =_u t_0 \upharpoonright_u «cs»]_t \wedge$
      $\$st\acute{} =_u \$st \oplus «st_0»$ *on* $\&ns1 \oplus «st_1»$ *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 $\mathcal{E}(s_0,t_0,E_0) \parallel_{N_C\ ns1\ cs\ ns2} P =$
      $(\exists\ (ref_0,\ ref_1,\ st_0,\ st_1,\ tt_0) \cdot$
      $[s_0]_{S<} \wedge$
      $[\$ref \mapsto_s «ref_0», \$st\acute{} \mapsto_s «st_1», \$tr \mapsto_s «[]», \$tr\acute{} \mapsto_s «tt_0»] \dagger P \wedge$
      $(\forall\ e \cdot «e» \in_u \lceil E_0 \rceil_{S<} \Rightarrow «e» \notin_u «ref_1») \wedge$
      $\$ref \subseteq_u («ref_0» \cup_u «ref_1») \cap_u «cs» \cup_u («ref_0» \cap_u «ref_1» - «cs») \wedge$
      $[«trace» \in_u t_0\ \star_{cs} «tt_0» \wedge «tt_0» \upharpoonright_u «cs» =_u t_0 \upharpoonright_u «cs»]_t \wedge$
      $\$st\acute{} =_u \$st \oplus «st_0»$ *on* $\&ns1 \oplus «st_1»$ *on* $\&ns2)$
  (**is** *?lhs* = *?rhs*)
proof −
  have $?lhs = P \parallel_{N_C\ ns2\ cs\ ns1} \mathcal{E}(s_0,t_0,E_0)$
    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* $\$ref \sharp P$
  shows $\Phi(s_0,\sigma_0,t_0)\ [\![ns1|cs|ns2]\!]^F\ P =$
    $(\exists\ (st_1,\ t_1) \cdot$
      $[s_0]_{S<} \wedge$
      $[\$st\acute{} \mapsto_s «st_1», \$tr \mapsto_s «[]», \$tr\acute{} \mapsto_s «t_1»] \dagger P \wedge$
      $[«trace» \in_u t_0 \star_{cs} «t_1» \wedge t_0 \upharpoonright_u «cs» =_u «t_1» \upharpoonright_u «cs»]_t \wedge$
      $\$st\acute{} =_u \$st \oplus (\!|\&\mathbf{v} \mapsto_s \$st|\!) \dagger \sigma_0$ *on* $\&ns1 \oplus «st_1»$ *on* $\&ns2)$

(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* =
$(\exists\ (st_0,\ st_1,\ tt_0,\ tt_1) \cdot$
$[\$st\acute{} \mapsto_s «st_0», \$tr \mapsto_s «[]», \$tr\acute{} \mapsto_s «tt_0»] \dagger \Phi(s_0,\sigma_0,t_0) \wedge$
$[\$st\acute{} \mapsto_s «st_1», \$tr \mapsto_s «[]», \$tr\acute{} \mapsto_s «tt_1»] \dagger RR(\exists\ \$ref \cdot P) \wedge$
$\$tr \leq_u \$tr\acute{} \wedge \&tt \in_u «tt_0» \star_{cs} «tt_1» \wedge «tt_0» \upharpoonright_u «cs» =_u «tt_1» \upharpoonright_u «cs» \wedge$
$\$st\acute{} =_u \$st \oplus «st_0»\ on\ \&ns1 \oplus «st_1»\ on\ \&ns2)$
by (*simp add*: *CSPFinalMerge-form ex-unrest Healthy-if unrest closure assms*)
also have ... =
$(\exists\ (st_1,\ tt_1) \cdot$
$[s_0]_{S<} \wedge$
$[\$st\acute{} \mapsto_s «st_1», \$tr \mapsto_s «[]», \$tr\acute{} \mapsto_s «tt_1»] \dagger RR(\exists\ \$ref \cdot P) \wedge$
$[«trace» \in_u t_0 \star_{cs} «tt_1» \wedge t_0 \upharpoonright_u «cs» =_u «tt_1» \upharpoonright_u «cs»]_t \wedge$
$\$st\acute{} =_u \$st \oplus (\!|\&\mathbf{v} \mapsto_s \$st|\!) \dagger \sigma_0\ on\ \&ns1 \oplus «st_1»\ on\ \&ns2)$
by (*rel-blast*)
also have ... =
$(\exists\ (st_1,\ t_1) \cdot$
$[s_0]_{S<} \wedge$
$[\$st\acute{} \mapsto_s «st_1», \$tr \mapsto_s «[]», \$tr\acute{} \mapsto_s «t_1»] \dagger P \wedge$
$[«trace» \in_u t_0 \star_{cs} «t_1» \wedge t_0 \upharpoonright_u «cs» =_u «t_1» \upharpoonright_u «cs»]_t \wedge$
$\$st\acute{} =_u \$st \oplus (\!|\&\mathbf{v} \mapsto_s \$st|\!) \dagger \sigma_0\ on\ \&ns1 \oplus «st_1»\ on\ \&ns2)$
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 $P\ [\![ns1|cs|ns2]\!]^F\ \Phi(s_1,\sigma_1,t_1) =$
$(\exists\ (st_0,\ t_0) \cdot$
$[\$st\acute{} \mapsto_s «st_0», \$tr \mapsto_s «[]», \$tr\acute{} \mapsto_s «t_0»] \dagger P \wedge$
$[s_1]_{S<} \wedge$
$[«trace» \in_u «t_0» \star_{cs} t_1 \wedge «t_0» \upharpoonright_u «cs» =_u t_1 \upharpoonright_u «cs»]_t \wedge$
$\$st\acute{} =_u \$st \oplus «st_0»\ on\ \&ns1 \oplus (\!|\&\mathbf{v} \mapsto_s \$st|\!) \dagger \sigma_1\ on\ \&ns2)$
(**is** *?lhs* = *?rhs*)
proof −
have $P\ [\![ns1|cs|ns2]\!]^F\ \Phi(s_1,\sigma_1,t_1) = \Phi(s_1,\sigma_1,t_1)\ [\![ns2|cs|ns1]\!]^F\ P$
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 $\Phi(s_1,\sigma_1,t_1)\ [\![ns1|cs|ns2]\!]^F\ \Phi(s_2,\sigma_2,t_2) =$
$([s_1 \wedge s_2]_{S<} \wedge [«trace» \in_u t_1 \star_{cs} t_2 \wedge t_1 \upharpoonright_u «cs» =_u t_2 \upharpoonright_u «cs»]_t \wedge [\langle \sigma_1\ [\&ns1|\&ns2]_s\ \sigma_2 \rangle_a]_{S'})$
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* =
$(\exists\ (st_0,\ st_1,\ tt_0,\ tt_1) \cdot$
$[\$st\acute{} \mapsto_s «st_0», \$tr \mapsto_s «[]», \$tr\acute{} \mapsto_s «tt_0»] \dagger \Phi(s_1,\sigma_1,t_1) \wedge$

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

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

lemma *CSPFinalMerge-UINF-mem-left* [*rpred*]:
$(\bigsqcap\ i{\in}A \cdot P(i))\ [\![ns1|cs|ns2]\!]^F\ Q = (\bigsqcap\ i{\in}A \cdot P(i)\ [\![ns1|cs|ns2]\!]^F\ Q)$
by (*simp add*: *CSPFinalMerge-def par-by-merge-USUP-mem-left*)

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

lemma *CSPFinalMerge-UINF-mem-right* [*rpred*]:
$P\ [\![ns1|cs|ns2]\!]^F\ (\bigsqcap\ i{\in}A \cdot Q(i)) = (\bigsqcap\ i{\in}A \cdot P\ [\![ns1|cs|ns2]\!]^F\ Q(i))$
by (*simp add*: *CSPFinalMerge-def par-by-merge-USUP-mem-right*)

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

lemma *InterMerge-csp-enable-left*:
assumes *P is RR* $\$st' \sharp P$
shows $\mathcal{E}(s_0,t_0,E_0)\ [\![cs]\!]^I\ P =$
$(\exists\ (ref_0,\ ref_1,\ t_1) \cdot$
$[s_0]_{S<} \wedge (\forall\ e \cdot «e» \in_u \lceil E_0 \rceil_{S<} \Rightarrow «e» \notin_u «ref_0») \wedge$
$[\$ref \mapsto_s «ref_1», \$tr \mapsto_s «[]», \$tr' \mapsto_s «t_1»] \dagger P \wedge$
$\$ref \subseteq_u («ref_0» \cup_u «ref_1») \cap_u «cs» \cup_u («ref_0» \cap_u «ref_1» - «cs») \wedge$
$[«trace» \in_u t_0 \star_{cs} «t_1» \wedge t_0 \upharpoonright_u «cs» =_u «t_1» \upharpoonright_u «cs»]_t)$
(**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*:
$\mathcal{E}(s_1,t_1,E_1)\ [\![cs]\!]^I\ \mathcal{E}(s_2,t_2,E_2) =$
$([s_1 \wedge s_2]_{S<} \wedge$
$(\forall\ e{\in}\lceil(E_1 \cap_u E_2 \cap_u «cs») \cup_u ((E_1 \cup_u E_2) - «cs»)\rceil_{S<} \cdot «e» \notin_u \$ref) \wedge$
$[«trace» \in_u t_1 \star_{cs} t_2 \wedge t_1 \upharpoonright_u «cs» =_u t_2 \upharpoonright_u «cs»]_t)$
(**is** *?lhs* = *?rhs*)

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

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

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

by (*rel-auto*)
also have ... = $([s_1 \wedge s_2]_{S<} \wedge (\forall\ e \in \lceil (E_1 - «cs») \rceil_{S<} \cdot «e» \notin_u \$ref) \wedge$
$[«trace» \in_u t_1 \star_{cs} t_2 \wedge t_1 \upharpoonright_u «cs» =_u t_2 \upharpoonright_u «cs»]_t)$
by (*rel-auto*)
also have ... = $(\exists\ trace \cdot \mathcal{E}(s_1 \wedge s_2 \wedge «trace» \in_u t_1 \star_{cs} t_2 \wedge t_1 \upharpoonright_u «cs» =_u t_2 \upharpoonright_u «cs», «trace», E_1 - «cs»))$
by (*rel-auto*)
finally show *?thesis* **.**
qed

lemma *InterMerge-csp-do-csp-enable* [*rpred*]:
$\Phi(s_1,\sigma_1,t_1)\ [\![cs]\!]^I\ \mathcal{E}(s_2,t_2,E_2) =$
$(\exists\ trace \cdot \mathcal{E}(s_1 \wedge s_2 \wedge «trace» \in_u t_1 \star_{cs} t_2 \wedge t_1 \upharpoonright_u «cs» =_u t_2 \upharpoonright_u «cs»,«trace»,E_2 - «cs»))$
(**is** *?lhs* = *?rhs*)
proof –
have $\Phi(s_1,\sigma_1,t_1)\ [\![cs]\!]^I\ \mathcal{E}(s_2,t_2,E_2) = \mathcal{E}(s_2,t_2,E_2)\ [\![cs]\!]^I\ \Phi(s_1,\sigma_1,t_1)$
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*]:
$(P \vee Q)\ [\![cs]\!]^I\ R = (P\ [\![cs]\!]^I\ R \vee Q\ [\![cs]\!]^I\ R)$
by (*simp add*: *CSPInterMerge-def par-by-merge-or-left*)

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

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

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

lemma *CSPInterMerge-UINF-mem-left* [*rpred*]:
$(\bigsqcap\ i \in A \cdot P(i))\ [\![cs]\!]^I\ Q = (\bigsqcap\ i \in A \cdot P(i)\ [\![cs]\!]^I\ Q)$
by (*simp add*: *CSPInterMerge-def par-by-merge-USUP-mem-left*)

lemma *CSPInterMerge-UINF-ind-left* [*rpred*]:
$(\bigsqcap\ i \cdot P(i))\ [\![cs]\!]^I\ Q = (\bigsqcap\ i \cdot P(i)\ [\![cs]\!]^I\ Q)$
by (*simp add*: *CSPInterMerge-def par-by-merge-USUP-ind-left*)

lemma *CSPInterMerge-UINF-mem-right* [*rpred*]:
$P\ [\![cs]\!]^I\ (\bigsqcap\ i \in A \cdot Q(i)) = (\bigsqcap\ i \in A \cdot P\ [\![cs]\!]^I\ Q(i))$
by (*simp add*: *CSPInterMerge-def par-by-merge-USUP-mem-right*)

lemma *CSPInterMerge-UINF-ind-right* [*rpred*]:
$P\ [\![cs]\!]^I\ (\bigsqcap\ i \cdot Q(i)) = (\bigsqcap\ i \cdot P\ [\![cs]\!]^I\ Q(i))$
by (*simp add*: *CSPInterMerge-def par-by-merge-USUP-ind-right*)

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

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

lemma *CSPInterMerge-shEx-right* [*rpred*]:
$P\ [\![cs]\!]^I\ (\exists\ i \cdot Q(i)) = (\exists\ i \cdot P\ [\![cs]\!]^I\ Q(i))$
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*: $(x \leq_u y) = (\exists\ z \cdot y =_u x\ \widehat{}_u\ «z»)$
by (*rel-auto*)

lemma *trace-pred-R1-true*: $[P(trace)]_t\ ;;\ R1\ true = [(\exists\ tt_0 \cdot «tt_0» \leq_u «trace» \wedge P(tt_0))]_t$
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*]:
$\Phi(s_1,\sigma_1,t_1)\ wr[cs]_C\ \mathcal{I}(s_2,\ t_2) =$
$(\forall\ (tt_0,\ tt_1) \cdot \mathcal{I}(s_1 \wedge s_2 \wedge «tt_1» \in_u (t_2\ \widehat{}_u\ «tt_0»)\ \star_{cs}\ t_1\ \wedge t_2\ \widehat{}_u\ «tt_0» \restriction_u «cs» =_u t_1 \restriction_u «cs»,\ «tt_1»))$
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* =
$(\neg_r\ (\exists\ (ref_0,\ st_0,\ tt_0) \cdot$
$[\$ref \mapsto_s «ref_0»,\ \$st\acute{} \mapsto_s «st_0»,\ \$tr \mapsto_s «[]»,\ \$tr\acute{} \mapsto_s «tt_0»]\ \dagger\ (\neg_r\ \mathcal{I}(s_2,\ t_2)) \wedge$
$[s_1]_{S<} \wedge$
$\$ref \subseteq_u «cs» \cup_u («ref_0» - «cs») \wedge$
$[«trace» \in_u «tt_0» \star_{cs} t_1 \wedge «tt_0» \restriction_u «cs» =_u t_1 \restriction_u «cs»]_t \wedge$
$\$st\acute{} =_u \$st)\ ;;\ R1\ true)$
by (*simp add*: *wrR-def par-by-merge-seq-remove merge-csp-do-right pr-var-def closure Healthy-if rpred*, *rel-auto*)
also have ... =
$(\neg_r\ (\exists\ tt_0 \cdot (\lceil s_2 \rceil_{S<} \wedge \lceil t_2 \rceil_{S<} \leq_u «tt_0») \wedge [s_1]_{S<} \wedge$
$[«trace» \in_u «tt_0» \star_{cs} t_1 \wedge «tt_0» \restriction_u «cs» =_u t_1 \restriction_u «cs»]_t)\ ;;\ R1\ true)$
by (*rel-auto*)
also have ... =
$(\neg_r\ (\exists\ tt_0 \cdot (\lceil s_2 \rceil_{S<} \wedge (\exists\ tt_1 \cdot «tt_0» =_u \lceil t_2 \rceil_{S<}\ \widehat{}_u\ «tt_1»)) \wedge [s_1]_{S<} \wedge$
$[«trace» \in_u «tt_0» \star_{cs} t_1 \wedge «tt_0» \restriction_u «cs» =_u t_1 \restriction_u «cs»]_t)\ ;;\ R1\ true)$
by (*simp add*: *utrace-leq*)
also have ... =
$(\neg_r\ (\exists\ tt_1 \cdot [s_1 \wedge s_2 \wedge «trace» \in_u (t_2\ \widehat{}_u\ «tt_1»)\ \star_{cs}\ t_1 \wedge t_2\ \widehat{}_u\ «tt_1» \restriction_u «cs» =_u t_1 \restriction_u «cs»]_t)\ ;;\ R1\ true)$
by (*rel-auto*)
also have ... =
$(\forall\ tt_1 \cdot \neg_r\ ([s_1 \wedge s_2 \wedge «trace» \in_u (t_2\ \widehat{}_u\ «tt_1»)\ \star_{cs}\ t_1 \wedge t_2\ \widehat{}_u\ «tt_1» \restriction_u «cs» =_u t_1 \restriction_u «cs»]_t\ ;;\ R1\ true))$
by (*rel-auto*)
also have ... =
$(\forall\ (tt_0,\ tt_1) \cdot \neg_r\ ([s_1 \wedge s_2 \wedge «tt_0» \leq_u «trace» \wedge «tt_0» \in_u (t_2\ \widehat{}_u\ «tt_1»)\ \star_{cs}\ t_1 \wedge t_2\ \widehat{}_u\ «tt_1» \restriction_u «cs» =_u t_1 \restriction_u «cs»]_t))$
by (*simp add*: *trace-pred-R1-true*, *rel-auto*)

also have ... = *?rhs*
by (*rel-auto*)
finally show *?thesis* **.**
qed

lemma *wrC-csp-do-false* [*wp*]:
$\Phi(s_1,\sigma_1,t_1)$ $wr[cs]_C$ *false* =
$(\forall\ (tt_0,\ tt_1) \cdot \mathcal{I}(s_1 \wedge «tt_1» \in_u «tt_0» \star_{cs} t_1 \wedge «tt_0» \restriction_u «cs» =_u t_1 \restriction_u «cs»,«tt_1»))$
(**is** *?lhs* = *?rhs*)
proof –
have *?lhs* = $\Phi(s_1,\sigma_1,t_1)$ $wr[cs]_C$ $\mathcal{I}(true,\ «[]»)$
by (*simp add*: *rpred*)
also have ... = *?rhs*
by (*simp add*: *wp*)
finally show *?thesis* **.**
qed

lemma *wrC-csp-enable-init* [*wp*]:
fixes t_1 t_2 :: ('*a list*, '*b*) *uexpr*
shows
$\mathcal{E}(s_1,t_1,E_1)$ $wr[cs]_C$ $\mathcal{I}(s_2,\ t_2)$ =
$(\forall\ (tt_0,\ tt_1) \cdot \mathcal{I}(s_1 \wedge s_2 \wedge «tt_1» \in_u (t_2\ \widehat{}_u\ «tt_0») \star_{cs} t_1\ \wedge t_2\ \widehat{}_u\ «tt_0» \restriction_u «cs» =_u t_1 \restriction_u «cs»,\ «tt_1»))$
(**is** *?lhs* = *?rhs*)
proof –
have *?lhs* =
$(\neg_r\ (\exists\ (ref_0,\ ref_1,\ st_0,\ st_1 :: 'b,$
$tt_0) \cdot [s_1]_{S<} \wedge$
$[\$ref \mapsto_s «ref_0»,\ \$st\acute{} \mapsto_s «st_0»,\ \$tr \mapsto_s «[]»,\ \$tr\acute{} \mapsto_s «tt_0»] \dagger (\neg_r\ \mathcal{I}(s_2,t_2)) \wedge$
$(\forall\ e \cdot «e» \in_u \lceil E_1 \rceil_{S<} \Rightarrow «e» \notin_u «ref_1») \wedge$
$\$ref \subseteq_u («ref_0» \cup_u «ref_1») \cap_u «cs» \cup_u («ref_0» \cap_u «ref_1» - «cs») \wedge$
$[«trace» \in_u «tt_0» \star_{cs} t_1 \wedge «tt_0» \restriction_u «cs» =_u t_1 \restriction_u «cs»]_t \wedge \$st\acute{} =_u \$st)\ ;;_h$
R1 true)
by (*simp add*: *wrR-def par-by-merge-seq-remove merge-csp-enable-right pr-var-def closure Healthy-if rpred*, *rel-auto*)
also have ... =
$(\neg_r\ (\exists\ tt_0 \cdot (\lceil s_2 \rceil_{S<} \wedge \lceil t_2 \rceil_{S<} \le_u «tt_0») \wedge [s_1]_{S<} \wedge$
$[«trace» \in_u «tt_0» \star_{cs} t_1 \wedge «tt_0» \restriction_u «cs» =_u t_1 \restriction_u «cs»]_t)\ ;;\ R1\ true)$
by (*rel-blast*)
also have ... =
$(\neg_r\ (\exists\ tt_0 \cdot (\lceil s_2 \rceil_{S<} \wedge (\exists\ tt_1 \cdot «tt_0» =_u \lceil t_2 \rceil_{S<}\ \widehat{}_u\ «tt_1»)) \wedge [s_1]_{S<} \wedge$
$[«trace» \in_u «tt_0» \star_{cs} t_1 \wedge «tt_0» \restriction_u «cs» =_u t_1 \restriction_u «cs»]_t)\ ;;\ R1\ true)$
by (*simp add*: *utrace-leq*)
also have ... =
$(\neg_r\ (\exists\ tt_1 \cdot [s_1 \wedge s_2 \wedge «trace» \in_u (t_2\ \widehat{}_u\ «tt_1») \star_{cs} t_1 \wedge t_2\ \widehat{}_u\ «tt_1» \restriction_u «cs» =_u t_1 \restriction_u «cs»]_t)\ ;;$
R1 true)
by (*rel-auto*)
also have ... =
$(\forall\ tt_1 \cdot \neg_r\ ([s_1 \wedge s_2 \wedge «trace» \in_u (t_2\ \widehat{}_u\ «tt_1») \star_{cs} t_1 \wedge t_2\ \widehat{}_u\ «tt_1» \restriction_u «cs» =_u t_1 \restriction_u «cs»]_t\ ;;$
R1 true))
by (*rel-auto*)
also have ... =
$(\forall\ (tt_0,\ tt_1) \cdot \neg_r\ ([s_1 \wedge s_2 \wedge «tt_0» \le_u «trace» \wedge «tt_0» \in_u (t_2\ \widehat{}_u\ «tt_1») \star_{cs} t_1 \wedge t_2\ \widehat{}_u\ «tt_1» \restriction_u$
$«cs» =_u t_1 \restriction_u «cs»]_t))$
by (*simp add*: *trace-pred-R1-true*, *rel-auto*)
also have ... = *?rhs*

by (*rel-auto*)
finally show *?thesis* **.**
qed

lemma *wrC-csp-enable-false* [*wp*]:
$\mathcal{E}(s_1,t_1,E)$ *wr*[*cs*]$_C$ *false* =
($\forall$ (tt_0, tt_1) $\cdot$ $\mathcal{I}(s_1 \wedge$ «tt_1» $\in_u$ «tt_0» $\star_{cs}$ $t_1 \wedge$ «tt_0» $\restriction_u$ «cs» $=_u$ t_1 $\restriction_u$ «cs»,«tt_1»))
(**is** *?lhs* = *?rhs*)
proof –
have *?lhs* = $\mathcal{E}(s_1,t_1,E)$ *wr*[*cs*]$_C$ $\mathcal{I}$(*true*, «[]»)
by (*simp add*: *rpred*)
also have ... = *?rhs*
by (*simp add*: *wp*)
finally show *?thesis* **.**
qed

4.2 Parallel operator

syntax
-par-circus :: *logic* ⇒ *salpha* ⇒ *logic* ⇒ *salpha* ⇒ *logic* ⇒ *logic* (- ⟦-|-|-⟧ - [*75,0,0,0,76*] *76*)
-par-csp :: *logic* ⇒ *logic* ⇒ *logic* ⇒ *logic* (- ⟦-⟧$_C$ - [*75,0,76*] *76*)
-inter-circus :: *logic* ⇒ *salpha* ⇒ *salpha* ⇒ *logic* ⇒ *logic* (- ⟦-‖-⟧ - [*75,0,0,76*] *76*)

translations
-par-circus P ns1 cs ns2 Q == *P* $\|_{M_C}$ *ns1 cs ns2 Q*
-par-csp P cs Q == *-par-circus P* 0_L *cs* 0_L *Q*
-inter-circus P ns1 ns2 Q == *-par-circus P ns1* {} *ns2 Q*

abbreviation *InterleaveCSP* :: ('*s*, '*e*) *action* ⇒ ('*s*, '*e*) *action* ⇒ ('*s*, '*e*) *action* (**infixr** ||| *75*)
where *P* ||| *Q* ≡ *P* ⟦∅‖∅⟧ *Q*

abbreviation *SynchroniseCSP* :: ('*s*, '*e*) *action* ⇒ ('*s*, '*e*) *action* ⇒ ('*s*, '*e*) *action* (**infixr** || *75*)
where *P* || *Q* ≡ *P* ⟦*UNIV*⟧$_C$ *Q*

definition *CSP5* :: 'φ *process* ⇒ 'φ *process* **where**
[*upred-defs*]: *CSP5*(*P*) = (*P* ||| *Skip*)

definition *C2* :: ('σ, 'φ) *action* ⇒ ('σ, 'φ) *action* **where**
[*upred-defs*]: *C2*(*P*) = (*P* ⟦Σ‖{}‖∅⟧ *Skip*)

definition *CACT* :: ('*s*, '*e*) *action* ⇒ ('*s*, '*e*) *action* **where**
[*upred-defs*]: *CACT*(*P*) = *C2*(*NCSP*(*P*))

abbreviation *CPROC* :: '*e process* ⇒ '*e process* **where**
CPROC(*P*) ≡ *CACT*(*P*)

lemma *Skip-right-form*:
assumes P_1 *is RC* P_2 *is RR* P_3 *is RR* \$*st*´ ♯ P_2
shows $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3)$;; *Skip* = $\mathbf{R}_s(P_1 \vdash P_2 \diamond (\exists\ \$ref \cdot P_3))$
proof –
have *1*:$RR(P_3)$;; $\Phi(true,id_s,$«[]»$)$ = $(\exists\ \$ref \cdot RR(P_3))$
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*
$\$st\acute{} \sharp P_2$ $\$st\acute{} \sharp Q_2$
$ns1 \bowtie ns2$
shows $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3)$ $[\![ns1 \| cs \| ns2]\!]$ $\mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3) =$
$\mathbf{R}_s$ $(((Q_1 \Rightarrow_r Q_2)\ wr[cs]_C\ P_1 \wedge (Q_1 \Rightarrow_r Q_3)\ wr[cs]_C\ P_1 \wedge$
$(P_1 \Rightarrow_r P_2)\ wr[cs]_C\ Q_1 \wedge (P_1 \Rightarrow_r P_3)\ wr[cs]_C\ Q_1) \vdash$
$(P_2\ [\![cs]\!]^I\ Q_2 \vee P_3\ [\![cs]\!]^I\ Q_2 \vee P_2\ [\![cs]\!]^I\ Q_3) \diamond$
$(P_3\ [\![ns1|cs|ns2]\!]^F\ Q_3))$
(**is** $?P$ $[\![ns1\|cs\|ns2]\!]$ $?Q = ?rhs$)
proof −
have *1*: $\bigwedge P\ Q.\ P\ wr_R(N_C\ ns1\ cs\ ns2)\ Q = P\ wr[cs]_C\ Q$ $\bigwedge P\ Q.\ P\ wr_R(N_C\ ns2\ cs\ ns1)\ Q = P\ wr[cs]_C\ Q$
by (*rel-auto*)+
have *2*: $(\exists\ \$st\acute{} \cdot N_C\ ns1\ cs\ ns2) = (\exists\ \$st\acute{} \cdot N_C\ 0_L\ cs\ 0_L)$
by (*rel-auto*)
have $?P$ $[\![ns1\|cs\|ns2]\!]$ $?Q = (?P \|_{M_R(N_C\ ns1\ cs\ ns2)}\ ?Q)\ ;;_h\ Skip$
by (*simp add*: *CSPMerge-def par-by-merge-seq-add*)
also
have ... = $\mathbf{R}_s$ $(((Q_1 \Rightarrow_r Q_2)\ wr[cs]_C\ P_1 \wedge$
$(Q_1 \Rightarrow_r Q_3)\ wr[cs]_C\ P_1 \wedge$
$(P_1 \Rightarrow_r P_2)\ wr[cs]_C\ Q_1 \wedge$
$(P_1 \Rightarrow_r P_3)\ wr[cs]_C\ Q_1) \vdash$
$(P_2\ [\![cs]\!]^I\ Q_2 \vee$
$P_3\ [\![cs]\!]^I\ Q_2 \vee$
$P_2\ [\![cs]\!]^I\ Q_3) \diamond$
$P_3 \|_{N_C\ ns1\ cs\ ns2}\ Q_3)\ ;;_h\ Skip$
by (*simp add*: *parallel-rdes-def swap-CSPInnerMerge CSPInterMerge-def closure assms 1 2*)
also
have ... = $\mathbf{R}_s$ $(((Q_1 \Rightarrow_r Q_2)\ wr[cs]_C\ P_1 \wedge$
$(Q_1 \Rightarrow_r Q_3)\ wr[cs]_C\ P_1 \wedge$
$(P_1 \Rightarrow_r P_2)\ wr[cs]_C\ Q_1 \wedge$
$(P_1 \Rightarrow_r P_3)\ wr[cs]_C\ Q_1) \vdash$
$(P_2\ [\![cs]\!]^I\ Q_2 \vee$
$P_3\ [\![cs]\!]^I\ Q_2 \vee$
$P_2\ [\![cs]\!]^I\ Q_3) \diamond$
$(\exists\ \$ref \cdot (P_3 \|_{N_C\ ns1\ cs\ ns2}\ Q_3)))$
by (*simp add*: *Skip-right-form closure parallel-RR-closed assms unrest*)
also
have ... = $\mathbf{R}_s$ $(((Q_1 \Rightarrow_r Q_2)\ wr[cs]_C\ P_1 \wedge$
$(Q_1 \Rightarrow_r Q_3)\ wr[cs]_C\ P_1 \wedge$
$(P_1 \Rightarrow_r P_2)\ wr[cs]_C\ Q_1 \wedge$
$(P_1 \Rightarrow_r P_3)\ wr[cs]_C\ Q_1) \vdash$
$(P_2\ [\![cs]\!]^I\ Q_2 \vee$
$P_3\ [\![cs]\!]^I\ Q_2 \vee$
$P_2\ [\![cs]\!]^I\ Q_3) \diamond$
$(P_3\ [\![ns1|cs|ns2]\!]^F\ Q_3))$
proof −
have $(\exists\ \$ref \cdot (P_3 \|_{N_C\ ns1\ cs\ ns2}\ Q_3)) = (P_3\ [\![ns1|cs|ns2]\!]^F\ Q_3)$
by (*rel-blast*)
thus *?thesis* **by** *simp*
qed
finally show *?thesis* **.**
qed

4.3 Parallel Laws

lemma *ParCSP-expand*:
  $P \; [\![ns1\|cs\|ns2]\!] \; Q = (P \parallel_{RN_C \; ns1 \; cs \; ns2} Q) \;;; Skip$
  by (*simp add*: *CSPMerge-def par-by-merge-seq-add*)

lemma *parallel-is-CSP* [*closure*]:
  assumes P *is CSP* Q *is CSP*
  shows $(P \; [\![ns1\|cs\|ns2]\!] \; Q)$ *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-NCSP* [*closure*]:
  assumes $ns1 \bowtie ns2$ P *is NCSP* Q *is NCSP*
  shows $(P \; [\![ns1\|cs\|ns2]\!] \; Q)$ *is NCSP*
proof –
  have $(P \; [\![ns1\|cs\|ns2]\!] \; Q) = (\mathbf{R}_s(pre_R \; P \vdash peri_R \; P \diamond post_R \; P) \; [\![ns1\|cs\|ns2]\!] \; \mathbf{R}_s(pre_R \; Q \vdash peri_R \; Q$
$\diamond \; post_R \; Q))$
  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 $(P \; [\![ns1\|cs\|ns2]\!] \; Q) = (Q \; [\![ns2\|cs\|ns1]\!] \; P)$
proof –
  have $(P \; [\![ns1\|cs\|ns2]\!] \; Q) = P \parallel_{swap_m \;;; \; (M_C \; ns2 \; cs \; ns1)} Q$
  by (*simp add*: *CSPMerge-def seqr-assoc*[*THEN sym*] *swap-merge-rd swap-CSPInnerMerge lens-indep-sym assms*)
  also have ... $= Q \; [\![ns2\|cs\|ns1]\!] \; P$
    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 process*
  assumes P *is NCSP*
  shows $CSP5(P) = C2(P)$
  unfolding *CSP5-def C2-def* **by** (*rdes-eq cls*: *assms*)

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

lemma *csp-do-triv-merge*:
  assumes P *is CRF*
  shows $P \; [\![\Sigma|\{\}|\emptyset]\!]^F \; \Phi(true, id_s, «[]») = P$ (**is** *?lhs* = *?rhs*)
proof –
  have $?lhs = (\exists \; (st_0, \; t_0) \cdot [\$st \mapsto_s «st_0», \$tr \mapsto_s «[]», \$tr' \mapsto_s «t_0»] \dagger CRF(P) \wedge [true]_{S<} \wedge [«trace»$

$=_u$ «t_0»$]_t \wedge \$\acute{st} =_u \$st \oplus$ «st_0» *on* &**v** $\oplus$ «*id*»$(\$st)_a$ *on* $\emptyset$)
  by (*simp add*: *FinalMerge-csp-do-right assms closure unrest Healthy-if*, *rel-auto*)
 also have ... = *CRF*(P)
  by (*rel-auto*)
 finally show *?thesis*
  by (*simp add*: *assms Healthy-if*)
qed

lemma *csp-do-triv-wr*:
 assumes P *is CRC*
 shows Φ(*true*,id_s,«[]») $wr[\{\}]_C$ $P = P$ (**is** *?lhs* = *?rhs*)
proof –
 have *?lhs* = ($\neg_r$ ($\exists$ (ref_0, st_0, tt_0) $\cdot$
   [\$*ref* $\mapsto_s$ «ref_0», $\$\acute{st} \mapsto_s$ «st_0», \$*tr* $\mapsto_s$ «[]», $\$\acute{tr} \mapsto_s$ «tt_0»] $\dagger$ ($\exists$ \$*ref*;$\$\acute{st}$ $\cdot$ *RR*($\neg_r$ P)) $\wedge$
   \$*ref* $\subseteq_u$ «ref_0» $\wedge$ [«*trace*» $=_u$ «tt_0»$]_t$ $\wedge$
   $\$\acute{st} =_u \$st$) ;; *R1 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*, *rel-auto*)
 also have ... = ($\neg_r$ ($\exists$ \$*ref*;$\$\acute{st}$ $\cdot$ *RR*($\neg_r$ P)) ;; *R1 true*)
  by (*rel-auto*, *meson order-refl*)
 also have ... = ($\neg_r$ ($\neg_r$ P) ;; *R1 true*)
  by (*simp add*: *Healthy-if closure ex-unrest unrest assms*)
 also have ... = P
  by (*metis CRC-implies-RC Healthy-def RC1-def RC-implies-RC1 assms*)
 finally show *?thesis* **.**
qed

lemma *C2-form*:
 assumes P *is NCSP*
 shows *C2*(P) = $\mathbf{R}_s$ (pre_R $P \vdash$ ($\exists$ ref_0 $\cdot$ $peri_R$ P⟦«ref_0»/\$*ref*⟧ $\wedge$ \$*ref* $\subseteq_u$ «ref_0») $\diamond$ $post_R$ P)
proof –
 have *1*:Φ(*true*,id_s,«[]») $wr[\{\}]_C$ pre_R P = pre_R P (**is** *?lhs* = *?rhs*)
  by (*simp add*: *csp-do-triv-wr closure assms*)
 have *2*: (pre_R $P \Rightarrow_r$ $peri_R$ P) $[\![\{\}]\!]^I$ Φ(*true*,id_s,«[]») =
   ($\exists$ ref_0 $\cdot$ ($peri_R$ P)⟦«ref_0»/\$*ref*⟧ $\wedge$ \$*ref* $\subseteq_u$ «ref_0») (**is** *?lhs* = *?rhs*)
 proof –
  have *?lhs* = $peri_R$ P $[\![\{\}]\!]^I$ Φ(*true*,id_s,«[]»)
   by (*simp add*: *SRD-peri-under-pre closure assms unrest*)
  also have ... = ($\exists$ $\$\acute{st}$ $\cdot$ ($peri_R$ P $\|_{N_C\ 0_L\ \{\}\ 0_L}$ Φ(*true*,id_s,«[]»)))
   by (*simp add*: *CSPInterMerge-def par-by-merge-def seqr-exists-right*)
  also have ... =
   ($\exists$ $\$\acute{st}$ $\cdot$ $\exists$ (ref_0, st_0, tt_0) $\cdot$
    [\$*ref* $\mapsto_s$ «ref_0», $\$\acute{st} \mapsto_s$ «st_0», \$*tr* $\mapsto_s$ «[]», $\$\acute{tr} \mapsto_s$ «tt_0»] $\dagger$ ($\exists$ $\$\acute{st}$ $\cdot$ *RR*($peri_R$ P)) $\wedge$
    \$*ref* $\subseteq_u$ «ref_0» $\wedge$ [«*trace*» $=_u$ «tt_0»$]_t$ $\wedge$ $\$\acute{st} =_u \$st$)
  by (*simp add*: *merge-csp-do-right pr-var-def assms Healthy-if closure rpred unrest ex-unrest*, *rel-auto*)
  also have ... =
   ($\exists$ ref_0 $\cdot$ ($\exists$ $\$\acute{st}$ $\cdot$ *RR*($peri_R$ P))⟦«ref_0»/\$*ref*⟧ $\wedge$ \$*ref* $\subseteq_u$ «ref_0»)
   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) $[\![\Sigma|\{\}|\emptyset]\!]^F$ Φ(*true*,id_s,«[]») = $post_R$(P) (**is** *?lhs* = *?rhs*)
  by (*simp add*: *csp-do-triv-merge SRD-post-under-pre unrest assms closure*)
 show *?thesis*

proof −
  have $C2(P) = \mathbf{R}_s\ (\Phi(true,id_s,«[]»)\ wr[\{\}]_C\ pre_R\ P \vdash$
  $(pre_R\ P \Rightarrow_r peri_R\ P)\ [\![\{\}]\!]^I\ \Phi(true,id_s,«[]») \diamond (pre_R\ P \Rightarrow_r post_R\ P)\ [\![\Sigma|\{\}|\emptyset]\!]^F\ \Phi(true,id_s,«[]»))$
  by (*simp add*: *C2-def*, *rdes-simp cls*: *assms*)
  also have ... $= \mathbf{R}_s\ (pre_R\ P \vdash (\exists\ ref_0 \cdot peri_R\ P[\![«ref_0»/\$ref]\!] \wedge \$ref \subseteq_u «ref_0») \diamond post_R\ P)$
  by (*simp add*: *1 2 3*)
  finally show *?thesis* **.**
qed
qed

lemma *C2-CDC-form*:
assumes P *is NCSP*
shows $C2(P) = \mathbf{R}_s\ (pre_R\ P \vdash CDC(peri_R\ P) \diamond post_R\ P)$
by (*simp add*: *C2-form assms CDC-def*)

lemma *C2-rdes-def*:
assumes P_1 *is CRC* P_2 *is CRR* P_3 *is CRR* $\$st\acute{}\ \sharp\ P_2$ $\$ref\ \sharp\ P_3$
shows $C2(\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3)) = \mathbf{R}_s(P_1 \vdash CDC(P_2) \diamond P_3)$
by (*simp add*: *C2-form assms closure rdes unrest usubst*, *rel-auto*)

lemma *C2-NCSP-intro*:
assumes P *is NCSP* $peri_R(P)$ *is CDC*
shows P *is C2*
proof −
have $C2(P) = \mathbf{R}_s\ (pre_R\ P \vdash CDC(peri_R\ P) \diamond post_R\ P)$
  by (*simp add*: *C2-CDC-form assms*(*1*))
also have ... $= \mathbf{R}_s\ (pre_R\ P \vdash peri_R\ P \diamond post_R\ P)$
  by (*simp add*: *Healthy-if assms*)
also have ... $= P$
  by (*simp add*: *NCSP-implies-CSP SRD-reactive-tri-design assms*(*1*))
finally show *?thesis*
  by (*simp add*: *Healthy-def*)
qed

lemma *C2-rdes-intro*:
assumes P_1 *is CRC* P_2 *is CRR* P_2 *is CDC* P_3 *is CRR* $\$st\acute{}\ \sharp\ P_2$ $\$ref\ \sharp\ P_3$
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 $peri_R(P) = peri_R\ (\mathbf{R}_s\ (pre_R\ P \vdash CDC(peri_R\ P) \diamond post_R\ P))$
  by (*metis C2-CDC-form Healthy-if assms*(*1*) *assms*(*2*))
also have ... $= CDC\ (pre_R\ P \Rightarrow_r peri_R\ P)$
  by (*simp add*: *rdes rpred assms closure unrest del*: *NSRD-peri-under-pre*)
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 *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* $\$st \sharp P_2$ $\$ref \sharp P_3$
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*(*NCSP*(*P*)) = *NCSP*(*C2*(*P*))
proof –
have *1*: *C2*(*NCSP*(*P*)) = *C2*(*P*)
by (*simp add*: *assms Healthy-if*)
also have ... = $\mathbf{R}_s$ (pre_R $P \vdash$ *CDC*($peri_R$ P) $\diamond$ $post_R$ P)
by (*simp add*: *C2-CDC-form assms*)
also have ... *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*($\mathbf{R}_s$($pre_R(P) \vdash peri_R(P) \diamond post_R(P)$))))
by (*simp add*: *NCSP-implies-NSRD NSRD-is-SRD SRD-reactive-tri-design assms*)
also have ... = $\mathbf{R}_s$ (pre_R $P \vdash$ *CDC* ($peri_R$ P) $\diamond$ $post_R$ P)
by (*simp add*: *C2-rdes-def closure assms unrest CDC-idem*)
also have ... = *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 ... = *NCSP*(*C2*(*NCSP*(*P*)))
by (*simp add*: *C2-NCSP-quasi-commute Healthy-Idempotent NCSP-Idempotent*)
also have ... *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*]:
$[\![X \textit{ is } CACT; P(\mathbf{R}_s(pre_R(X) \vdash peri_R(X) \diamond post_R(X)))]\!] \Longrightarrow P(X)$
using *CACT-implies-NCSP NCSP-elim* **by** *blast*

lemma *Miracle-C2-closed* [*closure*]: *Miracle is C2*
by (*rdes-simp*, *rule C2-rdes-intro*, *simp-all add*: *closure unrest*)

lemma *Chaos-C2-closed* [*closure*]: *Chaos is C2*
by (*rdes-simp*, *rule C2-rdes-intro*, *simp-all add*: *closure unrest*)

lemma *Skip-C2-closed* [*closure*]: *Skip is C2*
by (*rdes-simp*, *rule C2-rdes-intro*, *simp-all add*: *closure unrest*)

lemma *Stop-C2-closed* [*closure*]: *Stop is C2*
by (*rdes-simp*, *rule C2-rdes-intro*, *simp-all add*: *closure unrest*)

lemma *Miracle-CACT-closed* [*closure*]: *Miracle is CACT*
by (*simp add*: *CACT-intro Miracle-C2-closed csp-theory.top-closed*)

lemma *Chaos-CACT-closed* [*closure*]: *Chaos is CACT*
by (*simp add*: *CACT-intro closure*)

lemma *Skip-CACT-closed* [*closure*]: *Skip is CACT*
by (*simp add*: *CACT-intro closure*)

lemma *Stop-CACT-closed* [*closure*]: *Stop is CACT*
by (*simp add*: *CACT-intro closure*)

lemma *seq-C2-closed* [*closure*]:
assumes *P is NCSP P is C2 Q is NCSP Q is C2*
shows *P* ;; *Q is C2*
by (*rdes-simp cls*: *assms*(*1*,*3*), *rule C2-rdes-intro*, *simp-all add*: *closure assms unrest*)

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

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

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

lemma *map-st-ext-CDC-closed* [*closure*]:
assumes *P is CDC*
shows $P \oplus_r$ *map-st*$_L$[*a*] *is CDC*
proof −

have $CDC\ P \oplus_r\ map\text{-}st_L[a]$ *is CDC*
by (*rel-auto*)
thus *?thesis*
by (*simp add*: *assms Healthy-if*)
qed

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

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

lemma *UINF-C2-closed* [*closure*]:
assumes $A \neq \{\}$ $\bigwedge i.\ i \in A \Longrightarrow P(i)$ *is NCSP* $\bigwedge i.\ i \in A \Longrightarrow P(i)$ *is C2*
shows $(\bigsqcap\ i{\in}A \cdot P(i))$ *is C2*
proof –
have $(\bigsqcap\ i{\in}A \cdot P(i)) = (\bigsqcap\ i{\in}A \cdot \mathbf{R}_s(pre_R(P(i)) \vdash peri_R(P(i)) \diamond post_R(P(i))))$
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 $A \neq \{\}$ $\bigwedge i.\ i \in A \Longrightarrow P(i)$ *is CACT*
shows $(\bigsqcap\ i{\in}A \cdot P(i))$ *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 $b \to_R P$ *is C2*
by (*rdes-simp cls*: *assms*, *rule C2-rdes-intro*, *simp-all add*: *closure unrest assms*)

lemma *SpecC-CACT* [*closure*]: $a{:}[p,q]_C$ *is CACT*
by (*simp add*: *SpecC-def*, *rule CACT-rdes-intro*, *simp-all add*: *closure*; *rel-auto*)

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*]: $sinv_R(b)$ *is CACT*
by (*simp add*: *CACT-rdes-intro rdes-def closure unrest*)

lemma *AlternateR-C2-closed* [*closure*]:
assumes
$\bigwedge i.\ i \in A \Longrightarrow P(i)$ *is NCSP Q is NCSP*
$\bigwedge i.\ i \in A \Longrightarrow P(i)$ *is C2 Q is C2*
shows ($if_R\ i{\in}A \cdot g(i) \to P(i)$ *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 $\bigwedge i.\ i \in A \Longrightarrow P(i)$ *is CACT Q is CACT*
shows ($if_R\ i{\in}A \cdot g(i) \to P(i)$ *else Q fi*) *is CACT*
by (*rule CACT-intro*, *simp-all add*: *closure assms*)

lemma *AlternateR-list-C2-closed* [*closure*]:
assumes
$\bigwedge b\ P.\ (b, P) \in set\ A \Longrightarrow P$ *is NCSP Q is NCSP*
$\bigwedge b\ P.\ (b, P) \in set\ A \Longrightarrow P$ *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 $\bigwedge b\ P.\ (b, P) \in set\ A \Longrightarrow P$ *is CACT Q is CACT*
shows (*AlternateR-list A Q*) *is CACT*
by (*rule CACT-intro*, *simp-all add*: *closure assms*)

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

lemma *WhileC-C2-closed* [*closure*]:
 assumes *P is NCSP P is Productive P is C2*
 shows $while_C$ *b do P od is C2*
proof –
 have $while_C$ *b do P od* = $while_C$ *b do Productive*($\mathbf{R}_s$ (pre_R *P* ⊢ $peri_R$ *P* ⋄ $post_R$ *P*)) *od*
  by (*simp add*: *assms Healthy-if SRD-reactive-tri-design closure*)
 also have ... = $while_C$ *b do* $\mathbf{R}_s$ (pre_R *P* ⊢ $peri_R$ *P* ⋄ *R4*($post_R$ *P*)) *od*
  by (*simp add*: *Productive-RHS-design-form unrest assms rdes closure R4-def*)
 also have ... *is C2*
  by (*simp add*: *WhileC-def*, *simp add*: *closure assms unrest rdes-def C2-rdes-intro*)
 finally show *?thesis* **.**
qed

lemma *WhileC-CACT-closed* [*closure*]:
 assumes *P is CACT P 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
  ⋀ *i*. *i* ∈ *A* ⟹ *P*(*i*) *is NCSP* ⋀ *i*. *i* ∈ *A* ⟹ *P*(*i*) *is Productive* ⋀ *i*. *i* ∈ *A* ⟹ *P*(*i*) *is C2*
 shows (do_C *i*∈*A* · *g*(*i*) → *P*(*i*) *od*) *is C2*
 unfolding *IterateC-def* **by** (*simp add*: *closure assms*)

lemma *IterateC-CACT-closed* [*closure*]:
 assumes
  ⋀ *i*. *i* ∈ *A* ⟹ *P*(*i*) *is CACT* ⋀ *i*. *i* ∈ *A* ⟹ *P*(*i*) *is Productive*
 shows (do_C *i*∈*A* · *g*(*i*) → *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
  ⋀ *b P*. (*b*, *P*) ∈ *set A* ⟹ *P is NCSP*
  ⋀ *b P*. (*b*, *P*) ∈ *set A* ⟹ *P is Productive*
  ⋀ *b P*. (*b*, *P*) ∈ *set A* ⟹ *P 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
  ⋀ *b P*. (*b*, *P*) ∈ *set A* ⟹ *P is CACT*
  ⋀ *b P*. (*b*, *P*) ∈ *set A* ⟹ *P 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 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 $g\ \&_C\ P$ *is CACT*
by (*rule CACT-intro*, *simp-all add*: *closure assms*)

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

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

lemma *PrefixCSP-C2-closed* [*closure*]:
assumes *P is NCSP P is C2*
shows $a \rightarrow_C P$ *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 $a \rightarrow_C P$ *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 $\bigwedge i.\ i \in I \Longrightarrow P(i)$ *is NCSP* $\bigwedge i.\ i \in I \Longrightarrow P(i)$ *is C2*
shows $(\Box\ i{\in}I \cdot P(i))$ *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′ False*)
qed

lemma *ExtChoice-CACT-closed* [*closure*]:
assumes $\bigwedge i.\ i \in I \Longrightarrow P(i)$ *is CACT*
shows $(\Box\ i{\in}I \cdot P(i))$ *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 $P \Box Q$ *is C2*
proof −
have $P \Box Q = (\Box\ I{\in}\{P,Q\} \cdot I)$
by (*metis eq-id-iff 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 $P \Box Q$ *is CACT*

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

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

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

lemma *parallel-C2-closed* [*closure*]:
assumes *ns1* $\bowtie$ *ns2 P is NCSP Q is NCSP P is C2 Q is C2*
shows (P $[\![ns1 \| cs \| ns2]\!]$ Q) *is C2*
proof –
have (P $[\![ns1 \| cs \| ns2]\!]$ Q) = ($\mathbf{R}_s$(pre_R P $\vdash$ $peri_R$ P $\diamond$ $post_R$ P) $[\![ns1 \| cs \| ns2]\!]$ $\mathbf{R}_s$(pre_R Q $\vdash$ $peri_R$ Q $\diamond$ $post_R$ 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 $[\![ns1 \| cs \| ns2]\!]$ 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 $(P \,\square\, Q)(\!|f|\!)_C = (P(\!|f|\!)_C \,\square\, Q(\!|f|\!)_C)$
by (*rdes-eq-split cls*: *assms*)

lemma *rename-extChoice*:
assumes *inj f P is CACT Q is CACT*
shows $(P \,\square\, Q)(\!|f|\!)_C = (P(\!|f|\!)_C \,\square\, Q(\!|f|\!)_C)$
by (*simp add*: *CACT-implies-C2 CACT-implies-NCSP assms rename-extChoice-pre*)

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

lemma *interleave-unit*:

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

lemma *parallel-miracle*:
P *is NCSP* $\Longrightarrow$ *Miracle* $[\![ns1\|cs\|ns2]\!]$ P = *Miracle*
by (*simp add*: *CSPMerge-def par-by-merge-seq-add*[*THEN sym*] *Miracle-parallel-left-zero Skip-right-unit closure*)

lemma *parallel-assigns*:
assumes *vwb-lens ns1 vwb-lens ns2 ns1* $\bowtie$ *ns2* $x \subseteq_L$ *ns1* $y \subseteq_L$ *ns2*
shows $(x :=_C u)$ $[\![ns1\|cs\|ns2]\!]$ $(y :=_C v) = x, y :=_C u, v$
using *assms* **by** (*rdes-eq*)

definition *Accept* :: ($'s$, $'e$) *action* **where**
[*upred-defs*, *rdes-def*]: *Accept* = $\mathbf{R}_s(true_r \vdash \mathcal{E}(true,«[]»,«UNIV») \diamond false)$

definition [*upred-defs*, *rdes-def*]: $CACC(P) = (P \vee Accept)$

lemma *CACC-form*:
assumes P_1 *is RC* P_2 *is RR* $\$st\acute{} \sharp P_2$ P_3 *is RR*
shows $CACC(\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3)) = \mathbf{R}_s(P_1 \vdash (\mathcal{E}(true,«[]»,«UNIV») \vee P_2) \diamond P_3)$
by (*rdes-eq cls*: *assms*)

lemma *CACC-refines-Accept*:
assumes *P is CACC*
shows $P \sqsubseteq Accept$
proof −
have $CACC(P) \sqsubseteq Accept$ **by** *rel-auto*
thus *?thesis* **by** (*simp add*: *Healthy-if assms*)
qed

lemma *DoCSP-CACC* [*closure*]: $do_C(e)$ *is CACC*
unfolding *Healthy-def* **by** (*rdes-eq*)

lemma *CACC-seq-closure-left* [*closure*]:
assumes *P is NCSP P is CACC Q is NCSP*
shows $(P \;;\; Q)$ *is CACC*
proof −
have *1*: $(P \;;\; Q) = CACC(P) \;;\; Q$
by (*simp add*: *Healthy-if assms*(*2*))
also have *2*:... = $\mathbf{R}_s$ $((pre_R\ P \wedge post_R\ P\ wp_r\ pre_R\ Q) \vdash (peri_R\ P \vee \mathcal{E}(true,«[]»,\ «UNIV») \vee post_R\ P \;;\; peri_R\ Q) \diamond post_R\ P \;;\; post_R\ Q)$
by (*rdes-simp cls*: *assms*)
also have ... = *CACC*(...)
by (*rdes-eq cls*: *assms*)
also have ... = $CACC(P \;;\; Q)$
by (*simp add*: *1 2*)
finally show *?thesis*
by (*simp add*: *Healthy-def*)
qed

lemma *CACC-seq-closure-right*:

assumes *P is NCSP P* ;; *Chaos* = *Chaos Q is NCSP Q is CACC*
shows (*P* ;; *Q*) *is CACC*
oops

lemma *Chaos-is-CACC* [*closure*]: *Chaos is CACC*
unfolding *Healthy-def* **by** (*rdes-eq*)

lemma *intChoice-is-CACC* [*closure*]:
assumes *P is NCSP P is CACC Q is NCSP Q is CACC*
shows $P \sqcap Q$ *is CACC*
proof –
have $CACC(P) \sqcap CACC(Q)$ *is CACC*
unfolding *Healthy-def* **by** (*rdes-eq cls*: *assms*)
thus *?thesis*
by (*simp add*: *Healthy-if assms*(*2*) *assms*(*4*))
qed

lemma *extChoice-is-CACC* [*closure*]:
assumes *P is NCSP P is CACC Q is NCSP Q is CACC*
shows $P \Box Q$ *is CACC*
proof –
have $CACC(P) \Box CACC(Q)$ *is CACC*
unfolding *Healthy-def* **by** (*rdes-eq cls*: *assms*)
thus *?thesis*
by (*simp add*: *Healthy-if assms*(*2*) *assms*(*4*))
qed

lemma *Chaos-par-zero*:
assumes *P is NCSP P is CACC* $ns1 \bowtie ns2$
shows *Chaos* $[\![ns1 \| cs \| ns2]\!]$ $P = Chaos$
proof –
have *pprop*: $(\forall\ (tt_0,\ tt_1) \cdot \mathcal{I}(«tt_1» \in_u «tt_0» \star_{cs} «[]» \wedge «tt_0» \upharpoonright_u «cs» =_u «[]» \upharpoonright_u «cs»,«tt_1»)) = false$
by (*rel-simp*, *auto simp add*: *tr-par-empty*)
(*metis append-Nil2 seq-filter-Nil takeWhile.simps*(*1*))

have *1*: $P = \mathbf{R}_s(pre_R(P) \vdash peri_R(P) \diamond post_R(P))$
by (*simp add*: *NCSP-implies-NSRD NSRD-is-SRD SRD-reactive-tri-design assms*(*1*))

have ... $\sqsubseteq \mathbf{R}_s(true_r \vdash \mathcal{E}(true,«[]»,«UNIV») \diamond false)$
by (*metis 1 Accept-def CACC-refines-Accept assms*(*2*))

hence $peri_R\ P \sqsubseteq (pre_R\ P \wedge \mathcal{E}(true,«[]»,\ «UNIV»))$
by (*auto simp add*: *RHS-tri-design-refine′ closure assms*)

hence $peri_R(P) = ((pre_R\ P \wedge \mathcal{E}(true,«[]»,\ «UNIV»)) \vee peri_R(P))$
by (*simp add*: *assms*(*2*) *utp-pred-laws.sup.absorb2*)

with *1* **have** $P = \mathbf{R}_s(pre_R(P) \vdash (pre_R(P) \wedge \mathcal{E}(true,\ «[]»,\ «UNIV») \vee peri_R(P)) \diamond post_R(P))$
by (*simp add*: *NCSP-implies-CSP SRD-reactive-tri-design assms*(*1*))

also have ... $= \mathbf{R}_s(pre_R(P) \vdash (\mathcal{E}(true,\ «[]»,\ «UNIV») \vee peri_R(P)) \diamond post_R(P))$
by (*rel-auto*)

also have *Chaos* $[\![ns1 \| cs \| ns2]\!]$ ... = *Chaos*
by (*rdes-simp cls*: *assms*, *simp add*: *pprop*)

finally show *?thesis* **.**
qed

lemma *Chaos-inter-zero*:
assumes *P is NCSP P is CACC*
shows *Chaos* ||| *P* = *Chaos*
by (*simp add*: *Chaos-par-zero assms*(*1*) *assms*(*2*))

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*]: $hide_r\ P\ E = (\exists\ s \cdot (P⟦\$tr\ \hat{}_u «s»,(«E»\cup_u\$ref)/\$tr',\$ref⟧ \wedge \$tr' =_u \$tr\ \hat{}_u(«s»\restriction_u«-E»)))$

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*]:
$hide_r\ \mathcal{E}(s,\ t,\ E)\ F = \mathcal{E}(s \wedge E - «F» =_u E,\ t \restriction_u «-F»,\ E)$
by (*rel-auto*)

lemma *hide-rea-csp-do* [*rpred*]: $hide_r\ \Phi(s,\sigma,t)\ E = \Phi(s,\sigma,t \restriction_u «-E»)$
by (*rel-auto*)

lemma *filter-eval* [*simp*]:
  $(bop\ Cons\ x\ xs) \restriction_u E = (bop\ Cons\ x\ (xs \restriction_u E) \triangleleft x \in_u E \triangleright xs \restriction_u E)$
  by (*rel-simp*)

lemma *hide-rea-seq* [*rpred*]:
  assumes P *is CRR* $\$ref \sharp P$ 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*]:
  $hide_r\ (R1\ true)\ A\ ;;\ R1\ true = R1\ true$
  by (*rel-auto*, *metis append-Nil2 seq-filter-Nil*)

lemma *hide-rea-shEx* [*rpred*]: $hide_r\ (\exists\ i \cdot P(i))\ cs = (\exists\ i \cdot hide_r\ (P\ i)\ cs)$
  by (*rel-auto*)

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

lemma *hide-rea-twice* [*rpred*]: $hide_r\ (hide_r\ P\ A)\ B = hide_r\ P\ (A \cup B)$
  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*]: $\$st\acute{} \sharp P \Longrightarrow \$st\acute{} \sharp hide_r\ P\ E$
  by (*simp add*: *hide-rea-def unrest*)

lemma *ref′-unrest-hide-rea* [*unrest*]: $\$ref\acute{} \sharp P \Longrightarrow \$ref\acute{} \sharp hide_r\ P\ E$
  by (*simp add*: *hide-rea-def unrest usubst*)

5.2 Hiding in preconditions

definition *abs-rea* :: $('s, 'e)$ *action* $\Rightarrow$ $'e$ *set* $\Rightarrow$ $('s, 'e)$ *action* (abs_r) **where**
[*upred-defs*]: $abs_r\ P\ E = (\neg_r\ (hide_r\ (\neg_r\ P)\ E\ ;;\ true_r))$

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

lemma *abs-rea-conj* [*rpred*]: $abs_r\ (P \wedge Q)\ E = (abs_r\ P\ E \wedge abs_r\ Q\ E)$
  by (*rel-blast*)

lemma *abs-rea-true* [*rpred*]: $abs_r\ true_r\ E = true_r$
  by (*rel-auto*)

lemma *abs-rea-RC-closed* [*closure*]:
  assumes *P is CRR*
  shows abs_r *P E is CRC*
proof –
  have *RC1* $(abs_r\ (CRR\ P)\ E) = abs_r\ (CRR\ P)\ E$
    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* $\Longrightarrow abs_r\ (\neg_r\ P)\ E = (\neg_r\ hide_r\ P\ E\ ;;\ R1\ true)$
  by (*simp add*: *abs-rea-def rpred closure*)

lemma *abs-rea-wpR* [*rpred*]:
  assumes *P is CRR* $\$ref \sharp P$ *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** $\backslash_C$ *80*) **where**
[*upred-defs*]:
$HideCSP\ P\ E = \mathbf{R}_s(abs_r(pre_R(P))\ E \vdash hide_r\ (peri_R(P))\ E \diamond hide_r\ (post_R(P))\ E)$

lemma *HideCSP-rdes-def* [*rdes-def*]:
assumes *P is CRC Q is CRR R is CRR*
shows $\mathbf{R}_s(P \vdash Q \diamond R) \backslash_C A = \mathbf{R}_s(abs_r(P)\ A \vdash hide_r\ Q\ A \diamond hide_r\ R\ A)$ (**is** *?lhs* = *?rhs*)
proof –
have $?lhs = \mathbf{R}_s\ (abs_r\ P\ A \vdash hide_r\ (P \Rightarrow_r Q)\ A \diamond hide_r\ (P \Rightarrow_r R)\ A)$
by (*simp add*: *HideCSP-def rdes assms closure*)
also have ... $= \mathbf{R}_s\ (abs_r\ P\ A \vdash (abs_r\ P\ A \Rightarrow_r hide_r\ (P \Rightarrow_r Q)\ A) \diamond (abs_r\ P\ A \Rightarrow_r hide_r\ (P \Rightarrow_r R)\ A))$
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* $\Longrightarrow$ $P \backslash_C E$ *is NCSP*
by (*simp add*: *HideCSP-def closure unrest*)

lemma *HideCSP-C2-closed* [*closure*]:
assumes *P is NCSP P is C2*
shows $P \backslash_C E$ *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 $P \backslash_C E$ *is CACT*
by (*rule CACT-intro*, *simp-all add*: *closure assms*)

lemma *HideCSP-Chaos*: $Chaos \backslash_C E = Chaos$
by (*rdes-simp*)

lemma *HideCSP-Miracle*: $Miracle \backslash_C A = Miracle$
by (*rdes-eq*)

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

lemma *HideCSP-cond*:
assumes *P is NCSP Q is NCSP*
shows $(P \triangleleft b \triangleright_R Q) \backslash_C A = (P \backslash_C A \triangleleft b \triangleright_R Q \backslash_C A)$
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*]:
$do_C(a) \setminus_C A = (Skip \triangleleft (a \in_u «A») \triangleright_R do_C(a))$
by (*rdes-eq*)

lemma *HideCSP-PrefixCSP*:
assumes *P is NCSP*
shows $(a \rightarrow_C P) \setminus_C A = ((P \setminus_C A) \triangleleft (a \in_u «A») \triangleright_R (a \rightarrow_C (P \setminus_C A)))$
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*
begin recall-syntax

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

no-adhoc-overloading
uassigns assigns-r

adhoc-overloading
uassigns AssignsCSP

notation *GuardCSP* (**infixr** && *60*)

utp-lift-notation *GuardCSP* (*1*)

purge-notation *while-top* (*while - do - od*)

notation *WhileC* (*while - do - od*)

utp-lift-notation *WhileC* (*1*)

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.