Circus in Isabelle/UTP

Simon Foster James Baxter Ana Cavalcanti Jim Woodcock
Samuel Canham

April 20, 2018

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* ⇒ $'\vartheta$ *list* ⇒ $'\vartheta$ *list* ⇒ $'\vartheta$ *list set* **where**
tr-par cs [] [] = {[]} |
tr-par cs (*e* # *t*) [] = (*if e* ∈ *cs then* {[]} *else* {[*e*]} ⌢ (*tr-par cs t* [])) |
tr-par cs [] (*e* # *t*) = (*if e* ∈ *cs then* {[]} *else* {[*e*]} ⌢ (*tr-par cs* [] *t*)) |
tr-par cs (*e*$_1$ # *t*$_1$) (*e*$_2$ # *t*$_2$) =
(*if e*$_1$ = *e*$_2$
then
if e$_1$ ∈ *cs* (∗ ∧ *e*$_2$ ∈ *cs* ∗)

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

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

2.2 Lifted Trace Merge

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

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

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

2.3 Trace Merge Lemmas

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

lemma *tr-par-sym*:
tr-par cs t1 t2 = *tr-par cs t2 t1*
apply (*induct t1 arbitrary*: *t2*)
— Subgoal 1
apply (*simp add*: *tr-par-empty*)
— Subgoal 2
apply (*induct-tac t2*)
— Subgoal 2.1
apply (*clarsimp*)
— Subgoal 2.2
apply (*clarsimp*)
apply (*blast*)
done

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

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

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

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

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

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

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

end

3 Syntax and Translations for Event Prefix

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

syntax
-simple-prefix :: *logic* ⇒ *logic* ⇒ *logic* (- → - [*81*, *80*] *80*)

translations
$a \rightarrow P == CONST\ PrefixCSP\ \ll a\gg\ P$

We next configure a syntax for mixed prefixes.

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

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

Input Prefix: $\ldots?(x)$

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

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

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

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

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

syntax *-output-prefix* :: ('a, 'σ) *uexpr* ⇒ *prefix-elem′* (!′(-′))
syntax *-output-prefix* :: ('a, 'σ) *uexpr* ⇒ *prefix-elem′* (.′(-′))

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

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

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

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

syntax
-prefix-action ::
$('a, 'ε)$ *chan* ⇒ *mixed-prefix'* ⇒ $('σ, 'ε)$ *action* ⇒ $('σ, 'ε)$ *action*
((-- →/ -) [*81*, *81*, *80*] *80*)

Syntax translations

definition *lconj* :: $('a \Rightarrow 'α$ *upred*$) \Rightarrow ('b \Rightarrow 'α$ *upred*$) \Rightarrow ('a \times 'b \Rightarrow 'α$ *upred*$)$ (**infixr** $\wedge_l$ *35*)
where [*upred-defs*]: $(P \wedge_l Q) \equiv (\lambda\ (x,y).\ P\ x \wedge Q\ y)$

definition *outp-constraint* (**infix** $=_o$ *60*) **where**
[*upred-defs*]: *outp-constraint* $v \equiv (\lambda\ x.\ \ll x \gg =_u v)$

translations
-simple-input-prefix x ⇌ *-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*!(⟨*1*⟩) → *P*
term *x?*(*v*)!(*1*) → *P*
term *x*!(⟨*1*⟩)!(*2*)*?*(*v*:*true*) → *P*

Basic translations for state variable communications

syntax
-csp-input-var :: *logic* ⇒ *id* ⇒ *logic* ⇒ *logic* ⇒ *logic* (-**?**$-:- → - [*81*, *0*, *0*, *80*] *80*)
-csp-inputu-var :: *logic* ⇒ *id* ⇒ *logic* ⇒ *logic* (-**?**$- → - [*81*, *0*, *80*] *80*)

translations
*c***?**$*x*:*A* → *P* ⇀ *CONST InputVarCSP c x A P*
*c***?**$*x* → *P* ⇀ *CONST InputVarCSP c x* (λ *x*. *true*) *P*

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

lemma *outp-constraint-prod*:
$(outp\text{-}constraint\ \ll a \gg\ x \wedge outp\text{-}constraint\ \ll b \gg\ y) =$
$outp\text{-}constraint\ \ll (a,\ b) \gg\ (x,\ y)$
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*]:
$[\![\bigwedge i.\ P\ i\ is\ R1]\!] \Longrightarrow (\bigsqcap x \cdot [\ll i \gg =_o x]_{S<} \wedge P(x)) = P(i)$
by (*rel-blast*)

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

lemma *USUP-eq-event-eq* [*rpred*]:
assumes $\bigwedge y.\ P(y)\ is\ RR$
shows $(\bigsqcup y \cdot [v =_o y]_{S<} \Rightarrow_r P(y)) = P(y)[\![y \rightarrow \lceil v \rceil_{S\leftarrow}]\!]$
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 (*rule SRD-eq-intro*, *simp-all add*: *assms closure rdes*, *rel-auto+*)

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

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

end

4 Circus Parallel Composition

theory *utp-circus-parallel*
imports
utp-circus-prefix
utp-circus-traces
begin

4.1 Merge predicates

definition *CSPInnerMerge* :: $('\alpha \Longrightarrow '\sigma) \Rightarrow '\psi\ set \Rightarrow ('\beta \Longrightarrow '\sigma) \Rightarrow (('\sigma,'\psi)\ st\text{-}csp)\ merge\ (N_C)$ **where**
[*upred-defs*]:
CSPInnerMerge ns1 cs ns2 = (
$\$ref' \subseteq_u (((\$0{-}ref \cup_u \$1{-}ref) \cap_u \ll cs \gg) \cup_u ((\$0{-}ref \cap_u \$1{-}ref) - \ll cs \gg) \wedge$
$\$tr_< \leq_u \$tr' \wedge$
$(\$tr' - \$tr_<) \in_u (\$0{-}tr - \$tr_<) \star_{\ll cs \gg} (\$1{-}tr - \$tr_<) \wedge$
$(\$0{-}tr - \$tr_<) \restriction_u \ll cs \gg =_u (\$1{-}tr - \$tr_<) \restriction_u \ll cs \gg \wedge$
$\$st' =_u (\$st_< \oplus \$0{-}st\ on\ \&ns1) \oplus \$1{-}st\ on\ \&ns2)$

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

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

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

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* $\Rightarrow$ *salpha* $\Rightarrow$ *logic* $\Rightarrow$ *salpha* $\Rightarrow$ *logic* $\Rightarrow$ *logic* (- $[\![\text{-}|\text{-}|\text{-}]\!]^I$ - [85,0,0,0,86] 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* $\Rightarrow$ *salpha* $\Rightarrow$ *logic* $\Rightarrow$ *salpha* $\Rightarrow$ *logic* $\Rightarrow$ *logic* (- $wr[\text{-}|\text{-}|\text{-}]_C$ - [85,0,0,0,86] 86)

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

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 (∃ $*ref* ´ · *P*) *is R2m*
proof −
have *R2m*(∃ $*ref* ´ · *R2m P*) = (∃ $*ref* ´ · *R2m P*)
by (*rel-auto*)
thus *?thesis*
by (*metis Healthy-def′ assms*)
qed

lemma *CSPInnerMerge-unrests* [*unrest*]:
$*ok*$_<$ ♯ N_C *ns1 cs ns2*
$*wait*$_<$ ♯ N_C *ns1 cs ns2*
by (*rel-auto*)+

lemma *CSPInterMerge-RR-closed* [*closure*]:
assumes *P is RR Q is RR*
shows *P* ⟦*ns1*|*cs*|*ns2*⟧^{I} *Q is RR*
by (*simp add*: *CSPInterMerge-def parallel-RR-closed assms closure unrest*)

lemma *CSPInterMerge-unrest-st′* [*unrest*]:
$*st* ´ ♯ *P* ⟦*ns1*|*cs*|*ns2*⟧^{I} *Q*
by (*rel-auto*)

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

definition *CSPMerge* :: ('*α* ⟹ '*σ*) ⇒ '*ψ set* ⇒ ('*β* ⟹ '*σ*) ⇒ (('*σ*,'*ψ*) *st-csp*) *merge* (M_C) **where**
[*upred-defs*]: M_C *ns1 cs ns2* = M_R(N_C *ns1 cs ns2*) ;; *Skip*

definition *CSPInterleave* :: ('*α* ⟹ '*σ*) ⇒ ('*β* ⟹ '*σ*) ⇒ (('*σ*,'*ψ*) *st-csp*) *merge* (M_I) **where**
[*upred-defs*]: M_I *ns1 ns2* = M_R(N_I *ns1 ns2*) ;; *Skip*

lemma *swap-CSPInnerMerge*:
ns1 ⋈ *ns2* ⟹ $swap_m$;; (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-false* [*rpred*]: $P \ [\![ns1|cs|ns2]\!]^I \ false = false$
by (*simp add*: *CSPInterMerge-def*)

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

lemma *CSPInterMerge-commute*:
assumes *ns1* $\bowtie$ *ns2*
shows $P \ [\![ns1|cs|ns2]\!]^I \ Q = Q \ [\![ns2|cs|ns1]\!]^I \ P$
proof −
have $P \ [\![ns1|cs|ns2]\!]^I \ Q = P \ \|_{\exists \ \$st' \ \cdot \ N_C \ ns1 \ cs \ ns2} \ Q$
by (*simp add*: *CSPInterMerge-def*)
also have ... $= P \ \|_{\exists \ \$st' \ \cdot \ (swap_m \ ;; \ N_C \ ns2 \ cs \ ns1)} \ Q$
by (*simp add*: *swap-CSPInnerMerge lens-indep-sym assms*)
also have ... $= P \ \|_{swap_m \ ;; \ (\exists \ \$st' \ \cdot \ N_C \ ns2 \ cs \ ns1)} \ Q$
by (*simp add*: *seqr-exists-right*)
also have ... $= Q \ \|_{(\exists \ \$st' \ \cdot \ N_C \ ns2 \ cs \ ns1)} \ P$
by (*simp add*: *par-by-merge-commute-swap*[*THEN sym*])
also have ... $= Q \ [\![ns2|cs|ns1]\!]^I \ P$
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 $P \ [\![ns1|cs|ns2]\!]^F \ Q = P \ \|_{\exists \ \$ref' \ \cdot \ N_C \ ns1 \ cs \ ns2} \ Q$
by (*simp add*: *CSPFinalMerge-def*)
also have ... $= P \ \|_{\exists \ \$ref' \ \cdot \ (swap_m \ ;; \ N_C \ ns2 \ cs \ ns1)} \ Q$
by (*simp add*: *swap-CSPInnerMerge lens-indep-sym assms*)
also have ... $= P \ \|_{swap_m \ ;; \ (\exists \ \$ref' \ \cdot \ N_C \ ns2 \ cs \ ns1)} \ Q$
by (*simp add*: *seqr-exists-right*)
also have ... $= Q \ \|_{(\exists \ \$ref' \ \cdot \ N_C \ ns2 \ cs \ ns1)} \ P$
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

$P \parallel_{N_C\ ns1\ cs\ ns2} Q =$
$(\exists\ (ref_0,\ ref_1,\ st_0,\ st_1,\ tt_0,\ tt_1) \cdot$
$P[\![\ll ref_0 \gg,\ll st_0 \gg,\langle\rangle,\ll tt_0 \gg/\$ref',\$st',\$tr,\$tr']\!] \wedge Q[\![\ll ref_1 \gg,\ll st_1 \gg,\langle\rangle,\ll tt_1 \gg/\$ref',\$st',\$tr,\$tr']\!]$
$\wedge\ \$ref' \subseteq_u ((\ll ref_0 \gg \cup_u \ll ref_1 \gg) \cap_u \ll cs \gg) \cup_u ((\ll ref_0 \gg \cap_u \ll ref_1 \gg) - \ll cs \gg)$
$\wedge\ \$tr \leq_u \$tr'$
$\wedge\ \&tt \in_u \ll tt_0 \gg \star_{\ll cs \gg} \ll tt_1 \gg$
$\wedge\ \ll tt_0 \gg \restriction_u \ll cs \gg =_u \ll tt_1 \gg \restriction_u \ll cs \gg$
$\wedge\ \$st' =_u (\$st \oplus \ll st_0 \gg\ on\ \&ns1) \oplus \ll st_1 \gg\ on\ \&ns2)$
(**is** *?lhs = ?rhs*)

proof −
have *P*:($\exists$ {$ok',$wait'} · *R2*(*P*)) = *P* (**is** *?P' = -*)
by (*simp add*: *ex-unrest ex-plus Healthy-if assms RR-implies-R2 unrest closure*)
have *Q*:($\exists$ {$ok',$wait'} · *R2*(*Q*)) = *Q* (**is** *?Q' = -*)
by (*simp add*: *ex-unrest ex-plus Healthy-if assms RR-implies-R2 unrest closure*)
from *assms*(*1*,*2*)
have *?P'* $\parallel_{N_C\ ns1\ cs\ ns2}$ *?Q'* =
$(\exists\ (ref_0,\ ref_1,\ st_0,\ st_1,\ tt_0,\ tt_1) \cdot$
?P'$[\![\ll ref_0 \gg,\ll st_0 \gg,\langle\rangle,\ll tt_0 \gg/\$ref',\$st',\$tr,\$tr']\!] \wedge$ *?Q'*$[\![\ll ref_1 \gg,\ll st_1 \gg,\langle\rangle,\ll tt_1 \gg/\$ref',\$st',\$tr,\$tr']\!]$
$\wedge\ \$ref' \subseteq_u ((\ll ref_0 \gg \cup_u \ll ref_1 \gg) \cap_u \ll cs \gg) \cup_u ((\ll ref_0 \gg \cap_u \ll ref_1 \gg) - \ll cs \gg)$
$\wedge\ \$tr \leq_u \$tr'$
$\wedge\ \&tt \in_u \ll tt_0 \gg \star_{\ll cs \gg} \ll tt_1 \gg$
$\wedge\ \ll tt_0 \gg \restriction_u \ll cs \gg =_u \ll tt_1 \gg \restriction_u \ll cs \gg$
$\wedge\ \$st' =_u (\$st \oplus \ll st_0 \gg\ on\ \&ns1) \oplus \ll st_1 \gg\ on\ \&ns2)$
apply (*simp add*: *par-by-merge-alt-def*, *rel-auto*, *blast*)
apply (*rename-tac ok wait tr st ref tr' ref' ref*$_0$ *ref*$_1$ *st*$_0$ *st*$_1$ *tr*$_0$ *ok*$_0$ *tr*$_1$ *wait*$_0$ *ok*$_1$ *wait*$_1$)
apply (*rule-tac x=ok* **in** *exI*)
apply (*rule-tac x=wait* **in** *exI*)
apply (*rule-tac x=tr* **in** *exI*)
apply (*rule-tac x=st* **in** *exI*)
apply (*rule-tac x=ref* **in** *exI*)
apply (*rule-tac x=tr @ tr*$_0$ **in** *exI*)
apply (*rule-tac x=st*$_0$ **in** *exI*)
apply (*rule-tac x=ref*$_0$ **in** *exI*)
apply (*auto*)
apply (*metis Prefix-Order.prefixI append-minus*)
done
thus *?thesis*
by (*simp add*: *P Q*)
qed

lemma *CSPInterMerge-form*:
fixes *P Q* :: ('σ,'φ) *action*
assumes *vwb-lens ns1 vwb-lens ns2 P is RR Q is RR*
shows
$P\ [\![ns1|cs|ns2]\!]^I\ Q =$
$(\exists\ (ref_0,\ ref_1,\ st_0,\ st_1,\ tt_0,\ tt_1) \cdot$
$P[\![\ll ref_0 \gg,\ll st_0 \gg,\langle\rangle,\ll tt_0 \gg/\$ref',\$st',\$tr,\$tr']\!] \wedge Q[\![\ll ref_1 \gg,\ll st_1 \gg,\langle\rangle,\ll tt_1 \gg/\$ref',\$st',\$tr,\$tr']\!]$
$\wedge\ \$ref' \subseteq_u ((\ll ref_0 \gg \cup_u \ll ref_1 \gg) \cap_u \ll cs \gg) \cup_u ((\ll ref_0 \gg \cap_u \ll ref_1 \gg) - \ll cs \gg)$
$\wedge\ \$tr \leq_u \$tr'$
$\wedge\ \&tt \in_u \ll tt_0 \gg \star_{\ll cs \gg} \ll tt_1 \gg$
$\wedge\ \ll tt_0 \gg \restriction_u \ll cs \gg =_u \ll tt_1 \gg \restriction_u \ll cs \gg)$
(**is** *?lhs = ?rhs*)

proof −
have *?lhs* = ($\exists$ $st' · *P* $\parallel_{N_C\ ns1\ cs\ ns2}$ *Q*)
by (*simp add*: *CSPInterMerge-def par-by-merge-def seqr-exists-right*)

also have ... =
(∃ $st´ ·
(∃ (ref_0, ref_1, st_0, st_1, tt_0, tt_1) ·
$P⟦«ref_0»,«st_0»,⟨⟩,«tt_0»/$ref´,$st´,$tr,$tr´⟧ ∧ Q⟦«ref_1»,«st_1»,⟨⟩,«tt_1»/$ref´,$st´,$tr,$tr´⟧$
∧ $ref´ ⊆_u ((«ref_0» ∪_u «ref_1») ∩_u «cs») ∪_u («ref_0» ∩_u «ref_1») − «cs»)
∧ $tr ≤_u $tr´
∧ &tt ∈_u «tt_0» ⋆_«cs» «tt_1»
∧ «tt_0» ⌈_u «cs» =_u «tt_1» ⌈_u «cs»
∧ $st´ =_u ($st ⊕ «st_0» on &ns1) ⊕ «st_1» on &ns2))
by (*simp add*: *CSPInnerMerge-form assms*)
also have ... = *?rhs*
by (*rel-blast*)
finally show *?thesis* **.**
qed

lemma *CSPFinalMerge-form*:
fixes $P\ Q :: ('\sigma,'\varphi)\ action$
assumes *vwb-lens ns1 vwb-lens ns2 P is RR Q is RR* $ref´ ♯ P $ref´ ♯ Q
shows
$(P\ ⟦ns1|cs|ns2⟧^F\ Q) =$
(∃ (st_0, st_1, tt_0, tt_1) ·
$P⟦«st_0»,⟨⟩,«tt_0»/$st´,$tr,$tr´⟧ ∧ Q⟦«st_1»,⟨⟩,«tt_1»/$st´,$tr,$tr´⟧$
∧ $tr ≤_u $tr´
∧ &tt ∈_u «tt_0» ⋆_«cs» «tt_1»
∧ «tt_0» ⌈_u «cs» =_u «tt_1» ⌈_u «cs»
∧ $st´ =_u ($st ⊕ «st_0» on &ns1) ⊕ «st_1» on &ns2)
(**is** *?lhs = ?rhs*)
proof −
have $?lhs = (∃\ \$ref´ · P \|_{N_C\ ns1\ cs\ ns2}\ Q)$
by (*simp add*: *CSPFinalMerge-def par-by-merge-def seqr-exists-right*)
also have ... =
(∃ $ref´ ·
(∃ (ref_0, ref_1, st_0, st_1, tt_0, tt_1) ·
$P⟦«ref_0»,«st_0»,⟨⟩,«tt_0»/$ref´,$st´,$tr,$tr´⟧ ∧ Q⟦«ref_1»,«st_1»,⟨⟩,«tt_1»/$ref´,$st´,$tr,$tr´⟧$
∧ $ref´ ⊆_u ((«ref_0» ∪_u «ref_1») ∩_u «cs») ∪_u («ref_0» ∩_u «ref_1») − «cs»)
∧ $tr ≤_u $tr´
∧ &tt ∈_u «tt_0» ⋆_«cs» «tt_1»
∧ «tt_0» ⌈_u «cs» =_u «tt_1» ⌈_u «cs»
∧ $st´ =_u ($st ⊕ «st_0» on &ns1) ⊕ «st_1» on &ns2))
by (*simp add*: *CSPInnerMerge-form assms*)
also have ... =
(∃ $ref´ ·
(∃ (ref_0, ref_1, st_0, st_1, tt_0, tt_1) ·
(∃ $ref´ · P)⟦«ref_0»,«st_0»,⟨⟩,«tt_0»/$ref´,$st´,$tr,$tr´⟧ ∧ (∃ $ref´ · Q)⟦«ref_1»,«st_1»,⟨⟩,«tt_1»/$ref´,$st´,$tr
∧ $ref´ ⊆_u ((«ref_0» ∪_u «ref_1») ∩_u «cs») ∪_u («ref_0» ∩_u «ref_1») − «cs»)
∧ $tr ≤_u $tr´
∧ &tt ∈_u «tt_0» ⋆_«cs» «tt_1»
∧ «tt_0» ⌈_u «cs» =_u «tt_1» ⌈_u «cs»
∧ $st´ =_u ($st ⊕ «st_0» on &ns1) ⊕ «st_1» on &ns2))
by (*simp add*: *ex-unrest assms*)
also have ... =
(∃ (st_0, st_1, tt_0, tt_1) ·
(∃ $ref´ · P)⟦«st_0»,⟨⟩,«tt_0»/$st´,$tr,$tr´⟧ ∧ (∃ $ref´ · Q)⟦«st_1»,⟨⟩,«tt_1»/$st´,$tr,$tr´⟧$
∧ $tr ≤_u $tr´
∧ &tt ∈_u «tt_0» ⋆_«cs» «tt_1»

$\wedge$ «tt_0» $\restriction_u$ «cs» $=_u$ «tt_1» $\restriction_u$ «cs»
$\wedge$ $st´ =_u ($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 *merge-csp-do-left*:
assumes *vwb-lens ns1 vwb-lens ns2 ns1* $\bowtie$ *ns2 P is RR*
shows $\Phi(s_0,\sigma_0,t_0) \parallel_{N_C}$ *ns1 cs ns2* $P =$
$(\exists\ (ref_1,\ st_1,\ tt_1) \cdot$
$[s_0]_{S<} \wedge$
$[\$ref´ \mapsto_s$ «ref_1», $\$st´ \mapsto_s$ «st_1», $\$tr \mapsto_s \langle\rangle$, $\$tr´ \mapsto_s$ «tt_1»$] \dagger P \wedge$
$\$ref´ \subseteq_u$ «cs» $\cup_u$ («ref_1» $-$ «cs») $\wedge$
$[$«$trace$» $\in_u t_0 \star$«cs» «tt_1» $\wedge\ t_0 \restriction_u$ «cs» $=_u$ «tt_1» $\restriction_u$ «cs»$]_t \wedge$
$\$st´ =_u \$st \oplus$ «σ_0»($\$st)_a$ *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», $\$st´ \mapsto_s$ «st_0», $\$tr \mapsto_s \langle\rangle$, $\$tr´ \mapsto_s$ «tt_0»$] \dagger \Phi(s_0,\sigma_0,t_0) \wedge$
$[\$ref´ \mapsto_s$ «ref_1», $\$st´ \mapsto_s$ «st_1», $\$tr \mapsto_s \langle\rangle$, $\$tr´ \mapsto_s$ «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») $\wedge$
$\$tr \leq_u \$tr´ \wedge$
&$tt \in_u$ «tt_0» $\star$«cs» «tt_1» $\wedge$ «tt_0» $\restriction_u$ «cs» $=_u$ «tt_1» $\restriction_u$ «cs» $\wedge\ \$st´ =_u \$st \oplus$ «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», $\$st´ \mapsto_s$ «st_1», $\$tr \mapsto_s \langle\rangle$, $\$tr´ \mapsto_s$ «tt_1»$] \dagger P \wedge$
$\$ref´ \subseteq_u$ «cs» $\cup_u$ («ref_1» $-$ «cs») $\wedge$
$[$«$trace$» $\in_u t_0 \star$«cs» «tt_1» $\wedge\ t_0 \restriction_u$ «cs» $=_u$ «tt_1» $\restriction_u$ «cs»$]_t \wedge$
$\$st´ =_u \$st \oplus$ «σ_0»($\$st)_a$ *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», $\$st´ \mapsto_s$ «st_0», $\$tr \mapsto_s \langle\rangle$, $\$tr´ \mapsto_s$ «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» $\restriction_u$ «cs» $=_u t_1 \restriction_u$ «cs»$]_t \wedge$
$\$st´ =_u \$st \oplus$ «st_0» *on* &*ns1* $\oplus$ «σ_1»($\$st)_a$ *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», $\$st´ \mapsto_s$ «st_0», $\$tr \mapsto_s \langle\rangle$, $\$tr´ \mapsto_s$ «tt_0»$] \dagger P \wedge$
$[\$ref´ \mapsto_s$ «ref_1», $\$st´ \mapsto_s$ «st_1», $\$tr \mapsto_s \langle\rangle$, $\$tr´ \mapsto_s$ «tt_1»$] \dagger \Phi(s_1,\sigma_1,t_1) \wedge$
$\$ref´ \subseteq_u$ («ref_0» $\cup_u$ «ref_1») $\cap_u$ «cs» $\cup_u$ («ref_0» $\cap_u$ «ref_1» $-$ «cs») $\wedge$

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

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

lemma *FinalMerge-csp-do-left*:
assumes *vwb-lens ns1 vwb-lens ns2 ns1* $\bowtie$ *ns2 P is RR* $\$ref' \sharp P$
shows $\Phi(s_0,\sigma_0,t_0)$ $[\![ns1|cs|ns2]\!]^F$ P =
$(\exists\ (st_1,\ t_1) \cdot$
$[s_0]_{S<} \wedge$
$[\$st' \mapsto_s \ll st_1 \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_1 \gg] \dagger P \wedge$
$[\ll trace \gg \in_u t_0 \star_{\ll cs \gg} \ll t_1 \gg \wedge t_0 \restriction_u \ll cs \gg =_u \ll t_1 \gg \restriction_u \ll cs \gg]_t \wedge$
$\$st' =_u \$st \oplus \ll \sigma_0 \gg (\$st)_a$ *on* $\&ns1 \oplus \ll st_1 \gg$ *on* $\&ns2$)
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* =
$(\exists\ (st_0,\ st_1,\ tt_0,\ tt_1) \cdot$
$[\$st' \mapsto_s \ll st_0 \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll tt_0 \gg] \dagger \Phi(s_0,\sigma_0,t_0) \wedge$
$[\$st' \mapsto_s \ll st_1 \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll tt_1 \gg] \dagger RR(\exists\ \$ref' \cdot P) \wedge$
$\$tr \leq_u \$tr' \wedge \&tt \in_u \ll tt_0 \gg \star_{\ll cs \gg} \ll tt_1 \gg \wedge \ll tt_0 \gg \restriction_u \ll cs \gg =_u \ll tt_1 \gg \restriction_u \ll cs \gg \wedge$
$\$st' =_u \$st \oplus \ll st_0 \gg$ *on* $\&ns1 \oplus \ll st_1 \gg$ *on* $\&ns2$)
by (*simp add*: *CSPFinalMerge-form ex-unrest Healthy-if unrest closure assms*)
also have ... =
$(\exists\ (st_1,\ tt_1) \cdot$
$[s_0]_{S<} \wedge$
$[\$st' \mapsto_s \ll st_1 \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll tt_1 \gg] \dagger RR(\exists\ \$ref' \cdot P) \wedge$
$[\ll trace \gg \in_u t_0 \star_{\ll cs \gg} \ll t_1 \gg \wedge t_0 \restriction_u \ll cs \gg =_u \ll t_1 \gg \restriction_u \ll cs \gg]_t \wedge$
$\$st' =_u \$st \oplus \ll \sigma_0 \gg (\$st)_a$ *on* $\&ns1 \oplus \ll st_1 \gg$ *on* $\&ns2$)
by (*rel-blast*)
also have ... =
$(\exists\ (st_1,\ t_1) \cdot$
$[s_0]_{S<} \wedge$
$[\$st' \mapsto_s \ll st_1 \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_1 \gg] \dagger P \wedge$
$[\ll trace \gg \in_u t_0 \star_{\ll cs \gg} \ll t_1 \gg \wedge t_0 \restriction_u \ll cs \gg =_u \ll t_1 \gg \restriction_u \ll cs \gg]_t \wedge$
$\$st' =_u \$st \oplus \ll \sigma_0 \gg (\$st)_a$ *on* $\&ns1 \oplus \ll st_1 \gg$ *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 $[\![ns1|cs|ns2]\!]^F$ $\Phi(s_1,\sigma_1,t_1)$ =
$(\exists\ (st_0,\ t_0) \cdot$
$[\$st' \mapsto_s \ll st_0 \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_0 \gg] \dagger P \wedge$
$[s_1]_{S<} \wedge$
$[\ll trace \gg \in_u \ll t_0 \gg \star_{\ll cs \gg} t_1 \wedge \ll t_0 \gg \restriction_u \ll cs \gg =_u t_1 \restriction_u \ll cs \gg]_t \wedge$
$\$st' =_u \$st \oplus \ll st_0 \gg$ *on* $\&ns1 \oplus \ll \sigma_1 \gg (\$st)_a$ *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 $\Phi(s_1,\sigma_1,t_1)\ [\![ns1|cs|ns2]\!]^F\ \Phi(s_2,\sigma_2,t_2) =$
$([s_1 \wedge s_2]_{S<} \wedge [\ll trace\gg \in_u t_1 \star_{\ll cs\gg} t_2 \wedge t_1 \restriction_u \ll cs\gg =_u t_2 \restriction_u \ll cs\gg]_t \wedge [\langle \sigma_1\ [\&ns1|\&ns2]_s\ \sigma_2\rangle_a]_{S'})$
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* =
$(\exists\ (st_0,\ st_1,\ tt_0,\ tt_1) \cdot$
$[\$st' \mapsto_s \ll st_0\gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll tt_0\gg] \dagger \Phi(s_1,\sigma_1,t_1) \wedge$
$[\$st' \mapsto_s \ll st_1\gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll tt_1\gg] \dagger \Phi(s_2,\sigma_2,t_2) \wedge$
$\$tr \leq_u \$tr' \wedge \&tt \in_u \ll tt_0\gg \star_{\ll cs\gg} \ll tt_1\gg \wedge \ll tt_0\gg \restriction_u \ll cs\gg =_u \ll tt_1\gg \restriction_u \ll cs\gg \wedge$
$\$st' =_u \$st \oplus \ll st_0\gg$ *on* $\&ns1 \oplus \ll st_1\gg$ *on* $\&ns2$)
by (*simp add*: *CSPFinalMerge-form unrest closure assms*)
also have ... =
$([s_1 \wedge s_2]_{S<} \wedge [\ll trace\gg \in_u t_1 \star_{\ll cs\gg} t_2 \wedge t_1 \restriction_u \ll cs\gg =_u t_2 \restriction_u \ll cs\gg]_t \wedge [\langle \sigma_1\ [\&ns1|\&ns2]_s\ \sigma_2\rangle_a]_{S'})$
by (*rel-auto*)
finally show *?thesis* **.**
qed

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

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

(∃ (ref_0, ref_1, st_0, st_1, tt_0, tt_1) ·
[$ref´ ↦$_s «ref_0», $st´ ↦$_s «st_0», $tr ↦$_s ⟨⟩, $tr´ ↦$_s «tt_0»] † $\mathcal{E}(s_1,t_1,E_1)$ ∧
[$ref´ ↦$_s «ref_1», $st´ ↦$_s «st_1», $tr ↦$_s ⟨⟩, $tr´ ↦$_s «tt_1»] † $\mathcal{E}(s_2,t_2,E_2)$ ∧
$ref´ ⊆$_u («ref_0» ∪_u «ref_1») ∩_u «cs» ∪_u («ref_0» ∩_u «ref_1» − «cs») ∧
$tr ≤$_u $tr´ ∧ &$tt ∈$_u «$tt_0$» ⋆_«$cs$» «$tt_1$» ∧ «$tt_0$» ⌈_u «$cs$» =_u «$tt_1$» ⌈_u «$cs$»)
by (*simp add*: *CSPInterMerge-form unrest closure assms*)
also have ... =
(∃ (ref_0, ref_1, tt_0, tt_1) ·
[$ref´ ↦$_s «ref_0», $tr ↦$_s ⟨⟩, $tr´ ↦$_s «tt_0»] † $\mathcal{E}(s_1,t_1,E_1)$ ∧
[$ref´ ↦$_s «ref_1», $tr ↦$_s ⟨⟩, $tr´ ↦$_s «tt_1»] † $\mathcal{E}(s_2,t_2,E_2)$ ∧
$ref´ ⊆$_u («ref_0» ∪_u «ref_1») ∩_u «cs» ∪_u («ref_0» ∩_u «ref_1» − «cs») ∧
$tr ≤$_u $tr´ ∧ &$tt ∈$_u «$tt_0$» ⋆_«$cs$» «$tt_1$» ∧ «$tt_0$» ⌈_u «$cs$» =_u «$tt_1$» ⌈_u «$cs$»)
by (*rel-auto*)
also have ... =
($[s_1 ∧ s_2]_{S<}$ ∧
(∀ $e∈⌈(E_1 ∩_u E_2 ∩_u$ «cs») ∪_u ($(E_1 ∪_u E_2) −$ «cs»)$⌉_{S<}$ · «e» ∉_u $ref´) ∧
[«$trace$» ∈_u t_1 ⋆_«cs» $t_2 ∧ t_1 ⌈_u$ «cs» =_u $t_2 ⌈_u$ «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*]:
assumes *vwb-lens ns1 vwb-lens ns2 ns1* ⋈ *ns2*
shows $\mathcal{E}(s_1,t_1,E_1)$ $[\![ns1|cs|ns2]\!]^I$ $\mathcal{E}(s_2,t_2,E_2)$ =
(⋂ *trace* | «*trace*» ∈_u $⌈t_1$ ⋆_«cs» $t_2⌉_{S<}$ ·
$\mathcal{E}($ $s_1 ∧ s_2 ∧ t_1 ⌈_u$ «cs» =_u $t_2 ⌈_u$ «cs»
, «*trace*»
, $(E_1 ∩_u E_2 ∩_u$ «cs») ∪_u ($(E_1 ∪_u E_2) −$ «cs»)))
by (*simp add*: *InterMerge-csp-enable assms*, *rel-auto*)

lemma *InterMerge-csp-enable-csp-do*:
assumes *vwb-lens ns1 vwb-lens ns2 ns1* ⋈ *ns2*
shows $\mathcal{E}(s_1,t_1,E_1)$ $[\![ns1|cs|ns2]\!]^I$ $\Phi(s_2,\sigma_2,t_2)$ =
($[s_1 ∧ s_2]_{S<}$ ∧ (∀ $e∈⌈(E_1 −$ «cs»)$⌉_{S<}$ · «e» ∉_u $ref´) ∧
[«$trace$» ∈_u t_1 ⋆_«cs» $t_2 ∧ t_1 ⌈_u$ «cs» =_u $t_2 ⌈_u$ «cs»]_t)
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* =
(∃ (ref_0, ref_1, st_0, st_1, tt_0, tt_1) ·
[$ref´ ↦$_s «ref_0», $st´ ↦$_s «st_0», $tr ↦$_s ⟨⟩, $tr´ ↦$_s «tt_0»] † $\mathcal{E}(s_1,t_1,E_1)$ ∧
[$ref´ ↦$_s «ref_1», $st´ ↦$_s «st_1», $tr ↦$_s ⟨⟩, $tr´ ↦$_s «tt_1»] † $\Phi(s_2,\sigma_2,t_2)$ ∧
$ref´ ⊆$_u («ref_0» ∪_u «ref_1») ∩_u «cs» ∪_u («ref_0» ∩_u «ref_1» − «cs») ∧
$tr ≤$_u $tr´ ∧ &$tt ∈$_u «$tt_0$» ⋆_«$cs$» «$tt_1$» ∧ «$tt_0$» ⌈_u «$cs$» =_u «$tt_1$» ⌈_u «$cs$»)
by (*simp add*: *CSPInterMerge-form unrest closure assms*)
also have ... =
(∃ (ref_0, ref_1, tt_0) ·
[$ref´ ↦$_s «ref_0», $tr ↦$_s ⟨⟩, $tr´ ↦$_s «tt_0»] † $\mathcal{E}(s_1,t_1,E_1)$ ∧
$[s_2]_{S<}$ ∧

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

lemma *InterMerge-csp-enable-csp-do'* [*rpred*]:
assumes *vwb-lens ns1 vwb-lens ns2 ns1* $\bowtie$ *ns2*
shows $\mathcal{E}(s_1,t_1,E_1)\ [\![ns1|cs|ns2]\!]^I\ \Phi(s_2,\sigma_2,t_2) =$
$(\bigsqcap\ trace \mid \ll trace \gg \in_u \lceil t_1 \star_{\ll cs \gg} t_2 \rceil_{S<} \cdot$
$\mathcal{E}(s_1 \wedge s_2 \wedge t_1 \restriction_u \ll cs \gg =_u t_2 \restriction_u \ll cs \gg, \ll trace \gg, E_1 - \ll cs \gg))$
by (*simp add*: *InterMerge-csp-enable-csp-do assms*, *rel-auto*)

lemma *InterMerge-csp-do-csp-enable*:
assumes *vwb-lens ns1 vwb-lens ns2 ns1* $\bowtie$ *ns2*
shows $\Phi(s_1,\sigma_1,t_1)\ [\![ns1|cs|ns2]\!]^I\ \mathcal{E}(s_2,t_2,E_2) =$
$([s_1 \wedge s_2]_{S<} \wedge (\forall\ e \in \lceil (E_2 - \ll cs \gg) \rceil_{S<} \cdot \ll e \gg \notin_u \$ref\,') \wedge$
$[\ll trace \gg \in_u t_1 \star_{\ll cs \gg} t_2 \wedge t_1 \restriction_u \ll cs \gg =_u t_2 \restriction_u \ll cs \gg]_t)$
(**is** *?lhs* = *?rhs*)
proof −
have $\Phi(s_1,\sigma_1,t_1)\ [\![ns1|cs|ns2]\!]^I\ \mathcal{E}(s_2,t_2,E_2) = \mathcal{E}(s_2,t_2,E_2)\ [\![ns2|cs|ns1]\!]^I\ \Phi(s_1,\sigma_1,t_1)$
by (*simp add*: *CSPInterMerge-commute assms*)
also have ... = *?rhs*
by (*simp add*: *InterMerge-csp-enable-csp-do assms lens-indep-sym trace-merge-commute conj-comm eq-upred-sym*)
finally show *?thesis* **.**
qed

lemma *InterMerge-csp-do-csp-enable'* [*rpred*]:
assumes *vwb-lens ns1 vwb-lens ns2 ns1* $\bowtie$ *ns2*
shows $\Phi(s_1,\sigma_1,t_1)\ [\![ns1|cs|ns2]\!]^I\ \mathcal{E}(s_2,t_2,E_2) =$
$(\bigsqcap\ trace \mid \ll trace \gg \in_u \lceil t_1 \star_{\ll cs \gg} t_2 \rceil_{S<} \cdot$
$\mathcal{E}(s_1 \wedge s_2 \wedge t_1 \restriction_u \ll cs \gg =_u t_2 \restriction_u \ll cs \gg, \ll trace \gg, E_2 - \ll cs \gg))$
by (*simp add*: *InterMerge-csp-do-csp-enable assms*, *rel-auto*)

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

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

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

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

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

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

lemma *merge-csp-do-right*:
assumes *vwb-lens ns1 vwb-lens ns2 ns1* ⋈ *ns2 P is RC*
shows Φ(s_1,σ_1,t_1) *wr*[*ns1*|*cs*|*ns2*]$_C$ *P* = *undefined*
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* =
(¬$_r$ (∃ (*ref*$_0$, *st*$_0$, *tt*$_0$) ·
[$*ref*´ ↦$_s$ «*ref*$_0$», $*st*´ ↦$_s$ «*st*$_0$», $*tr* ↦$_s$ ⟨⟩, $*tr*´ ↦$_s$ «*tt*$_0$»] † (¬$_r$ *RC*(*P*)) ∧
[s_1]$_{S<}$ ∧
$*ref*´ ⊆$_u$ «*cs*» ∪$_u$ («*ref*$_0$» − «*cs*») ∧
[«*trace*» ∈$_u$ «*tt*$_0$» ⋆$_{«cs»}$ *t*$_1$ ∧ «*tt*$_0$» ⌈$_u$ «*cs*» =$_u$ *t*$_1$ ⌈$_u$ «*cs*»]$_t$ ∧
$*st*´ =$_u$ $*st* ⊕ «*st*$_0$» *on* &*ns1* ⊕ «$\sigma_1$»($*st*)$_a$ *on* &*ns2*) ;; *R1 true*)
by (*simp add*: *wrR-def par-by-merge-seq-remove merge-csp-do-right closure assms Healthy-if rpred*)
also have ... =
(¬$_r$ (∃ (*ref*$_0$, *st*$_0$, *tt*$_0$) ·
[$*ref*´ ↦$_s$ «*ref*$_0$», $*st*´ ↦$_s$ «*st*$_0$», $*tr* ↦$_s$ ⟨⟩, $*tr*´ ↦$_s$ «*tt*$_0$»] † (¬$_r$ *RC*(*P*)) ∧
[s_1]$_{S<}$ ∧
$*ref*´ ⊆$_u$ «*cs*» ∪$_u$ («*ref*$_0$» − «*cs*») ∧
[«*trace*» ∈$_u$ «*tt*$_0$» ⋆$_{«cs»}$ *t*$_1$ ∧ «*tt*$_0$» ⌈$_u$ «*cs*» =$_u$ *t*$_1$ ⌈$_u$ «*cs*»]$_t$;; *true*$_r$ ∧
$*st*´ =$_u$ $*st* ⊕ «*st*$_0$» *on* &*ns1* ⊕ «$\sigma_1$»($*st*)$_a$ *on* &*ns2*))
apply (*rel-auto*)

oops

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*)
-inter-csp :: *logic* ⇒ *logic* ⇒ *logic* (**infixr** ||| *75*)

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*
-inter-csp P Q == *-par-csp P* {} *Q*

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

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

lemma *Skip-right-form*:
assumes P_1 *is RC* P_2 *is RR* P_3 *is RR* $*st*´ ♯ P_2
shows **R**$_s$(P_1 ⊢ P_2 ⋄ P_3) ;; *Skip* = **R**$_s$(P_1 ⊢ P_2 ⋄ (∃ $*ref*´ · P_3))
proof −
have *1*:*RR*(P_3) ;; Φ(*true*,*id*,⟨⟩) = (∃ $*ref*´ · *RR*($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*
$\$st' \sharp P_2$ $\$st' \sharp Q_2$
$ns1 \bowtie ns2$
shows $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3)\ [\![ns1 \| cs \| ns2]\!]\ \mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3) =$
$\mathbf{R}_s\ (((Q_1 \Rightarrow_r Q_2)\ wr[ns1|cs|ns2]_C\ P_1 \wedge$
$(Q_1 \Rightarrow_r Q_3)\ wr[ns1|cs|ns2]_C\ P_1 \wedge$
$(P_1 \Rightarrow_r P_2)\ wr[ns2|cs|ns1]_C\ Q_1 \wedge$
$(P_1 \Rightarrow_r P_3)\ wr[ns2|cs|ns1]_C\ Q_1) \vdash$
$((P_1 \Rightarrow_r P_2)\ [\![ns1|cs|ns2]\!]^I\ (Q_1 \Rightarrow_r Q_2) \vee$
$(P_1 \Rightarrow_r P_3)\ [\![ns1|cs|ns2]\!]^I\ (Q_1 \Rightarrow_r Q_2) \vee$
$(P_1 \Rightarrow_r P_2)\ [\![ns1|cs|ns2]\!]^I\ (Q_1 \Rightarrow_r Q_3)) \diamond$
$((P_1 \Rightarrow_r P_3)\ [\![ns1|cs|ns2]\!]^F\ (Q_1 \Rightarrow_r Q_3)))$
(**is** *?P* $[\![ns1 \| cs \| ns2]\!]$ *?Q* = *?rhs*)
proof −
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 ... = $\mathbf{R}_s\ (((Q_1 \Rightarrow_r Q_2)\ wr[ns1|cs|ns2]_C\ P_1 \wedge$
$(Q_1 \Rightarrow_r Q_3)\ wr[ns1|cs|ns2]_C\ P_1 \wedge$
$(P_1 \Rightarrow_r P_2)\ wr[ns2|cs|ns1]_C\ Q_1 \wedge$
$(P_1 \Rightarrow_r P_3)\ wr[ns2|cs|ns1]_C\ Q_1) \vdash$
$((P_1 \Rightarrow_r P_2)\ [\![ns1|cs|ns2]\!]^I\ (Q_1 \Rightarrow_r Q_2) \vee$
$(P_1 \Rightarrow_r P_3)\ [\![ns1|cs|ns2]\!]^I\ (Q_1 \Rightarrow_r Q_2) \vee$
$(P_1 \Rightarrow_r P_2)\ [\![ns1|cs|ns2]\!]^I\ (Q_1 \Rightarrow_r Q_3)) \diamond$
$(P_1 \Rightarrow_r P_3)\ \|_{N_C\ ns1\ cs\ ns2}\ (Q_1 \Rightarrow_r Q_3))\ ;;_h$ *Skip*
by (*simp add*: *parallel-rdes-def swap-CSPInnerMerge CSPInterMerge-def closure assms*)
also
have ... = $\mathbf{R}_s\ (((Q_1 \Rightarrow_r Q_2)\ wr[ns1|cs|ns2]_C\ P_1 \wedge$
$(Q_1 \Rightarrow_r Q_3)\ wr[ns1|cs|ns2]_C\ P_1 \wedge$
$(P_1 \Rightarrow_r P_2)\ wr[ns2|cs|ns1]_C\ Q_1 \wedge$
$(P_1 \Rightarrow_r P_3)\ wr[ns2|cs|ns1]_C\ Q_1) \vdash$
$((P_1 \Rightarrow_r P_2)\ [\![ns1|cs|ns2]\!]^I\ (Q_1 \Rightarrow_r Q_2) \vee$
$(P_1 \Rightarrow_r P_3)\ [\![ns1|cs|ns2]\!]^I\ (Q_1 \Rightarrow_r Q_2) \vee$
$(P_1 \Rightarrow_r P_2)\ [\![ns1|cs|ns2]\!]^I\ (Q_1 \Rightarrow_r Q_3)) \diamond$
$(\exists\ \$ref' \cdot ((P_1 \Rightarrow_r P_3)\ \|_{N_C\ ns1\ cs\ ns2}\ (Q_1 \Rightarrow_r Q_3))))$
by (*simp add*: *Skip-right-form closure parallel-RR-closed assms unrest*)
also
have ... = $\mathbf{R}_s\ (((Q_1 \Rightarrow_r Q_2)\ wr[ns1|cs|ns2]_C\ P_1 \wedge$
$(Q_1 \Rightarrow_r Q_3)\ wr[ns1|cs|ns2]_C\ P_1 \wedge$
$(P_1 \Rightarrow_r P_2)\ wr[ns2|cs|ns1]_C\ Q_1 \wedge$
$(P_1 \Rightarrow_r P_3)\ wr[ns2|cs|ns1]_C\ Q_1) \vdash$
$((P_1 \Rightarrow_r P_2)\ [\![ns1|cs|ns2]\!]^I\ (Q_1 \Rightarrow_r Q_2) \vee$
$(P_1 \Rightarrow_r P_3)\ [\![ns1|cs|ns2]\!]^I\ (Q_1 \Rightarrow_r Q_2) \vee$
$(P_1 \Rightarrow_r P_2)\ [\![ns1|cs|ns2]\!]^I\ (Q_1 \Rightarrow_r Q_3)) \diamond$
$((P_1 \Rightarrow_r P_3)\ [\![ns1|cs|ns2]\!]^F\ (Q_1 \Rightarrow_r Q_3)))$
proof −
have $(\exists\ \$ref' \cdot ((P_1 \Rightarrow_r P_3)\ \|_{N_C\ ns1\ cs\ ns2}\ (Q_1 \Rightarrow_r Q_3))) = ((P_1 \Rightarrow_r P_3)\ [\![ns1|cs|ns2]\!]^F\ (Q_1 \Rightarrow_r Q_3))$
by (*rel-blast*)
thus *?thesis* **by** *simp*
qed
finally show *?thesis* **.**

qed

4.3 Parallel Laws

lemma *ParCSP-expand*:
P ⟦*ns1*‖*cs*‖*ns2*⟧ *Q* = (*P* ‖$_{RN_C}$ *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* ‖$_{M_R(N_C\ ns1\ cs\ ns2)}$ *Q*) *is CSP*
by (*simp add*: *closure assms*)
hence (*P* ‖$_{M_R(N_C\ ns1\ cs\ ns2)}$ *Q*) ;; *Skip is CSP*
by (*simp add*: *closure*)
thus *?thesis*
by (*simp add*: *CSPMerge-def par-by-merge-seq-add*)
qed

lemma *parallel-is-CSP3* [*closure*]:
assumes *P is CSP P is CSP3 Q is CSP Q is CSP3*
shows (*P* ⟦*ns1*‖*cs*‖*ns2*⟧ *Q*) *is CSP3*
proof −
have (*P* ‖$_{M_R(N_C\ ns1\ cs\ ns2)}$ *Q*) *is CSP*
by (*simp add*: *closure assms*)
hence (*P* ‖$_{M_R(N_C\ ns1\ cs\ ns2)}$ *Q*) ;; *Skip is CSP*
by (*simp add*: *closure*)
thus *?thesis*
oops

theorem *parallel-commutative*:
assumes *ns1* ⋈ *ns2*
shows (*P* ⟦*ns1*‖*cs*‖*ns2*⟧ *Q*) = (*Q* ⟦*ns2*‖*cs*‖*ns1*⟧ *P*)
proof −
have (*P* ⟦*ns1*‖*cs*‖*ns2*⟧ *Q*) = *P* ‖$_{swap_m}$;; (*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

lemma *interleave-commute*:
P ||| *Q* = *Q* ||| *P*
using *parallel-commutative zero-lens-indep* **by** *blast*

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

lemma *C2-form*:
assumes *P is NCSP*
shows *C2*(*P*) = **R**$_s$ (*pre*$_R$ *P* ⊢ (∃ *ref*$_0$ · *peri*$_R$ *P*⟦«*ref*$_0$»/$*ref* ´⟧ ∧ $*ref* ´ ⊆$_u$ «*ref*$_0$») ⋄ *post*$_R$ *P*)
proof −
have *1*:Φ(*true*,*id*,⟨⟩) *wr*[Σ|{}|∅]$_C$ *pre*$_R$ *P* = *pre*$_R$ *P* (**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = (¬$_r$ (∃ (*ref*$_0$, *st*$_0$, *tt*$_0$) ·

$[\$ref\,' \mapsto_s \ll ref_0 \gg, \$st\,' \mapsto_s \ll st_0 \gg, \$tr \mapsto_s \langle\rangle, \$tr\,' \mapsto_s \ll tt_0 \gg] \dagger (\exists\ \$ref\,'; \$st\,' \cdot RR(\neg_r\ pre_R\ P)) \wedge$
$\$ref\,' \subseteq_u \ll ref_0 \gg \wedge [\ll trace \gg =_u \ll tt_0 \gg]_t \wedge$
$\$st\,' =_u \$st \oplus \ll st_0 \gg\ on\ \Sigma \oplus \ll id \gg (\$st)_a\ on\ \emptyset)\ ;;\ R1\ true)$

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*)

also have ... = $(\neg_r\ (\exists\ \$ref\,'; \$st\,' \cdot RR(\neg_r\ pre_R\ P))\ ;;\ R1\ true)$

by (*rel-auto*)

also have ... = $(\neg_r\ (\neg_r\ pre_R\ P)\ ;;\ R1\ true)$

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

also have ... = $pre_R\ P$

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

finally show *?thesis* **.**

qed

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

proof −

have $?lhs = peri_R\ P\ [\![\Sigma|\{\}|\emptyset]\!]^I\ \Phi(true,id,\langle\rangle)$

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

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

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

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

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

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

by (*rel-auto*)

also have ... = *?rhs*

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

finally show *?thesis* **.**

qed

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

proof −

have $?lhs = post_R\ P\ [\![\Sigma|\{\}|\emptyset]\!]^F\ \Phi(true,id,\langle\rangle)$

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

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

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

also have ... = $RR(post_R(P))$

by (*rel-auto*)

finally show *?thesis*

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

qed

show *?thesis*

proof −

have $C2(P) = \mathbf{R}_s\ (\Phi(true,id,\langle\rangle)\ wr[\Sigma|\{\}|\emptyset]_C\ pre_R\ P \vdash$
$(pre_R\ P \Rightarrow_r peri_R\ P)\ [\![\Sigma|\{\}|\emptyset]\!]^I\ \Phi(true,id,\langle\rangle) \diamond (pre_R\ P \Rightarrow_r post_R\ P)\ [\![\Sigma|\{\}|\emptyset]\!]^F\ \Phi(true,id,\langle\rangle))$

by (*simp add*: *C2-def*, *rdes-simp cls*: *assms*, *simp add*: *id-def pr-var-def*)

also have ... = $\mathbf{R}_s\ (pre_R\ P \vdash (\exists\ ref_0 \cdot peri_R\ P[\![\ll ref_0 \gg / \$ref\,']\!] \wedge \$ref\,' \subseteq_u \ll ref_0 \gg) \diamond post_R\ P)$

by (*simp add*: *1 2 3*)

finally show *?thesis* **.**

qed

qed

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

definition *CDC* :: ('*s*, '*e*) *action* ⇒ ('*s*, '*e*) *action* **where**
[*upred-defs*]: $CDC(P) = (\exists\ ref_0 \cdot P[\![\ll ref_0 \gg / \$ref\,']\!] \wedge \$ref\,' \subseteq_u \ll ref_0 \gg)$

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

lemma *CDC-RR-commute*: $CDC(RR(P)) = RR(CDC(P))$
by (*rel-blast*)

lemma *CDC-RR-closed* [*closure*]: *P is RR* ⟹ $CDC(P)$ *is RR*
by (*metis CDC-RR-commute Healthy-def*)

lemma *CDC-unrest* [*unrest*]: $[\![\textit{vwb-lens } x; (\$ref\,')_v \bowtie x; x \sharp P]\!] \Longrightarrow x \sharp CDC(P)$
by (*simp add*: *CDC-def unrest usubst lens-indep-sym*)

lemma *CDC-R4-commute*: $CDC(R4(P)) = R4(CDC(P))$
by (*rel-auto*)

lemma *R4-CDC-closed* [*closure*]: *P is CDC* ⟹ $R4(P)$ *is CDC*
by (*simp add*: *CDC-R4-commute Healthy-def*)

lemma *CDC-R5-commute*: $CDC(R5(P)) = R5(CDC(P))$
by (*rel-auto*)

lemma *R5-CDC-closed* [*closure*]: *P is CDC* ⟹ $R5(P)$ *is CDC*
by (*simp add*: *CDC-R5-commute Healthy-def*)

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

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

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

lemma *CDC-disj-closed* [*closure*]:
assumes *P is CDC Q is CDC*
shows $(P \vee Q)$ *is CDC*
proof −
have $CDC(P \vee Q) = (CDC(P) \vee CDC(Q))$
by (*rel-auto*)
thus *?thesis*
by (*metis Healthy-def assms*(*1*) *assms*(*2*))
qed

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

lemma *CDC-conj-closed* [*closure*]:
assumes *P is CDC Q is CDC*
shows $(P \wedge Q)$ *is CDC*
using *assms* **by** (*rel-auto*, *blast*, *meson*)

lemma *CDC-rea-impl* [*rpred*]:
$\$ref' \sharp P \Longrightarrow CDC(P \Rightarrow_r Q) = (P \Rightarrow_r CDC(Q))$
by (*rel-auto*)

lemma *rea-impl-CDC-closed* [*closure*]:
assumes $\$ref' \sharp P$ *Q is CDC*
shows $(P \Rightarrow_r Q)$ *is CDC*
using *assms* **by** (*simp add*: *CDC-rea-impl Healthy-def*)

lemma *seq-CDC-closed* [*closure*]:
assumes *Q is CDC*
shows $(P ;; Q)$ *is CDC*
proof −
have $CDC(P ;; Q) = P ;; CDC(Q)$
by (*rel-blast*)
thus *?thesis*
by (*metis Healthy-def assms*)
qed

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

lemma *C2-CDC-form*:
assumes *P is NCSP*
shows $C2(P) = \mathbf{R}_s\ (pre_R\ P \vdash CDC(peri_R\ P) \diamond post_R\ P)$
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* $\$st' \sharp P_2$ $\$ref' \sharp P_3$
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* $\$st' \sharp P_2$ $\$ref' \sharp P_3$
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*)
also have ... = $CDC\ (peri_R\ P)$
by (*simp add*: *SRD-peri-under-pre closure unrest assms*)
finally show *?thesis*
by (*simp add*: *Healthy-def*)
qed

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

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

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

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

lemma *wp-rea-CRC* [*closure*]: $[\![\ P\ is\ CRR;\ Q\ is\ CRC\ ]\!] \Longrightarrow P\ wp_r\ Q\ is\ CRC$
by (*rule CRC-intro*, *simp-all add*: *unrest closure*)

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

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

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

lemma *ExtChoice-C2-closed* [*closure*]:
assumes $\bigwedge i.\ i \in I \Longrightarrow P(i)$ *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-ind' False*)
qed

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*∈{*P*,*Q*} · *I*)
by (*simp add*: *extChoice-def*)
also have ... *is C2*
by (*rule ExtChoice-C2-closed*, *auto simp add*: *assms*)
finally show *?thesis* **.**
qed

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

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

lemma
assumes *vwb-lens ns1 vwb-lens ns2 ns1* ⋈ *ns2 P is RR*
shows *P wr*[*ns1*|*cs*|*ns2*]$_C$ *false* = *undefined* (**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = (¬$_r$ (∃ (*ref*$_0$, *ref*$_1$, *st*$_0$, *st*$_1$, *tt*$_0$, *tt*$_1$) ·
[$*ref*$´ ↦$_s$ «*ref*$_0$», $*st*$´ ↦$_s$ «*st*$_0$», $*tr* ↦$_s$ ⟨⟩, $*tr*$´ ↦$_s$ «*tt*$_0$»] † *R1 true* ∧
[$*ref*$´ ↦$_s$ «*ref*$_1$», $*st*$´ ↦$_s$ «*st*$_1$», $*tr* ↦$_s$ ⟨⟩, $*tr*$´ ↦$_s$ «*tt*$_1$»] † *P* ∧
$*ref*$´ ⊆$_u$ («*ref*$_0$» ∪$_u$ «*ref*$_1$») ∩$_u$ «*cs*» ∪$_u$ («*ref*$_0$» ∩$_u$ «*ref*$_1$» − «*cs*») ∧
$*tr* ≤$_u$ $*tr*$´ ∧
&*tt* ∈$_u$ «*tt*$_0$» ⋆$_{«cs»}$ «*tt*$_1$» ∧ «*tt*$_0$» ⌈$_u$ «*cs*» =$_u$ «*tt*$_1$» ⌈$_u$ «*cs*» ∧
$*st*$´ =$_u$ $*st* ⊕ «*st*$_0$» *on* &*ns1* ⊕ «*st*$_1$» *on* &*ns2*) ;;
R1 true)
by (*simp add*: *wrR-def par-by-merge-seq-remove CSPInnerMerge-form assms closure usubst unrest*)
also have ... = (¬$_r$ (∃ (*tt*$_0$, *tt*$_1$) ·
[$*tr* ↦$_s$ ⟨⟩, $*tr*$´ ↦$_s$ «*tt*$_1$»] † *P* ∧
$*tr* ≤$_u$ $*tr*$´ ∧
&*tt* ∈$_u$ «*tt*$_0$» ⋆$_{«cs»}$ «*tt*$_1$» ∧ «*tt*$_0$» ⌈$_u$ «*cs*» =$_u$ «*tt*$_1$» ⌈$_u$ «*cs*») ;;

$R1\ true$)
by (*rel-blast*)
also have ... = ($\neg_r$ ($\exists$ (tt_0, tt_1) ·
[$tr ↦_s ⟨⟩, $tr´ ↦_s «tt_1»] † RR(P) ∧
$tr ≤_u $tr´ ∧
&tt ∈_u «tt_0» ⋆_«cs» «tt_1» ∧ «tt_0» ↾_u «cs» =_u «tt_1» ↾_u «cs») ;;
$R1\ true$)
by (*simp add*: *Healthy-if assms*)
oops

end

5 Meta theory for Circus

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

References

[1] S. Foster, F. Zeyda, and J. Woodcock. Unifying heterogeneous state-spaces with lenses. In *ICTAC*, LNCS 9965. Springer, 2016.

[2] M. V. M. Oliveira. *Formal Derivation of State-Rich Reactive Programs using Circus*. PhD thesis, Department of Computer Science - University of York, UK, 2006. YCST-2006-02.