

Circus Modelling Language in Isabelle/UTP

Simon Foster James Baxter Ana Cavalcanti Jim Woodcock
Samuel Canham

March 2, 2018

Contents

1	Introduction	2
2	Circus Core Types	2
2.1	Circus Alphabet	3
2.2	Basic laws	3
2.3	Unrestriction laws	4
3	Circus Reactive Relations	5
3.1	Healthiness Conditions	5
3.2	Closure Properties	6
3.3	Introduction laws	10
3.4	Trace Substitution	10
3.5	Initial Interaction	12
3.6	Enabled Events	13
3.7	Completed Trace Interaction	14
4	Circus and CSP Healthiness Conditions	16
5	Definitions	16
5.1	Healthiness condition properties	17
5.2	CSP theories	28
5.3	Algebraic laws	29
6	Reactive Contracts for CSP/Circus with refusals	29
7	External Choice	30
7.1	Definitions and syntax	31
7.2	Basic laws	31
7.3	Algebraic laws	31
7.4	Reactive design calculations	31
7.5	Productivity and Guardedness	39
7.6	Algebraic laws	40

8	Circus and CSP Actions	43
8.1	Conditionals	43
8.2	Guarded commands	43
8.3	Alternation	43
8.4	Assignment	44
8.5	Assignment with update	44
8.6	State abstraction	45
8.7	Assumptions	46
8.8	Guards	46
8.9	Basic events	49
8.10	Event prefix	51
8.11	Guarded external choice	53
8.12	Input prefix	53
8.13	Algebraic laws	55
9	Syntax and Translations for Event Prefix	56
10	Recursion in Circus	59
10.1	Fixed-points	59
10.2	Example action expansion	60
11	Circus Trace Merge	61
11.1	Function Definition	61
11.2	Lifted Trace Merge	61
11.3	Trace Merge Lemmas	61
12	Circus Parallel Composition	62
12.1	Merge predicates	63
12.2	Parallel operator	73
12.3	Parallel Laws	74
13	Linking to the Failures-Divergences Model	78
13.1	Failures-Divergences Semantics	78
13.2	Circus Operators	80
13.3	Deadlock Freedom	88
14	Meta theory for Circus	88

1 Introduction

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

2 Circus Core Types

```
theory utp-circus-core
  imports UTP-Reactive-Designs.utp-rea-designs
begin
```

2.1 Circus Alphabet

alphabet $\text{'}\varphi \text{ csp-vars} = \text{'}\sigma \text{ rsp-vars} +$
 $\text{ref} :: \text{'}\varphi \text{ set}$

declare $\text{csp-vars.defs} \text{ [lens-defs]}$
declare $\text{csp-vars.splits} \text{ [alpha-splits]}$

The following two locale interpretations are a technicality to improve the behaviour of the automatic tactics. They enable (re)interpretation of state spaces in order to remove any occurrences of lens types, replacing them by tuple types after the tactics *pred-simp* and *rel-simp* are applied. Eventually, it would be desirable to automate preform these interpretations automatically as part of the **alphabet** command.

interpretation *alphabet-csp-prd*:
 $\text{lens-interp } \lambda(ok, wait, tr, m). (ok, wait, tr, \text{ref}_v m, \text{more } m)$
apply (*unfold-locales*)
apply (*rule injI*)
apply (*clarsimp*)
done

interpretation *alphabet-csp-rel*:
 $\text{lens-interp } \lambda(ok, ok', wait, wait', tr, tr', m, m').$
 $(ok, ok', wait, wait', tr, tr', \text{ref}_v m, \text{ref}_v m', \text{more } m, \text{more } m')$
apply (*unfold-locales*)
apply (*rule injI*)
apply (*clarsimp*)
done

lemma *circus-var-ords* [*usubst*]:
 $\text{\$ref} \prec_v \text{\$ref'}$
 $\text{\$ok} \prec_v \text{\$ref} \text{\$ok'} \prec_v \text{\$ref'} \text{\$ok} \prec_v \text{\$ref'} \text{\$ok'} \prec_v \text{\$ref}$
 $\text{\$ref} \prec_v \text{\$wait} \text{\$ref'} \prec_v \text{\$wait'} \text{\$ref} \prec_v \text{\$wait'} \text{\$ref'} \prec_v \text{\$wait}$
 $\text{\$ref} \prec_v \text{\$st} \text{\$ref'} \prec_v \text{\$st'} \text{\$ref} \prec_v \text{\$st'} \text{\$ref'} \prec_v \text{\$st}$
 $\text{\$ref} \prec_v \text{\$tr} \text{\$ref'} \prec_v \text{\$tr'} \text{\$ref} \prec_v \text{\$tr'} \text{\$ref'} \prec_v \text{\$tr}$
by (*simp-all add: var-name-ord-def*)

type-synonym $(\text{'}\sigma, \text{'}\varphi) \text{ st-csp} = (\text{'}\sigma, \text{'}\varphi \text{ list}, (\text{'}\varphi, \text{unit}) \text{ csp-vars-scheme}) \text{ rsp}$
type-synonym $(\text{'}\sigma, \text{'}\varphi) \text{ action} = (\text{'}\sigma, \text{'}\varphi) \text{ st-csp hrel}$
type-synonym $\text{'}\varphi \text{ csp} = (\text{unit}, \text{'}\varphi) \text{ st-csp}$
type-synonym $\text{'}\varphi \text{ rel-csp} = \text{'}\varphi \text{ csp hrel}$

There is some slight imprecision with the translations, in that we don't bother to check if the trace event type and refusal set event types are the same. Essentially this is because its very difficult to construct processes where this would be the case. However, it may be better to add a proper ML print translation in the future.

translations
 $(\text{type}) (\text{'}\sigma, \text{'}\varphi) \text{ st-csp} \leq (\text{type}) (\text{'}\sigma, \text{'}\varphi \text{ list}, \text{'}\varphi 1 \text{ csp-vars}) \text{ rsp}$
 $(\text{type}) (\text{'}\sigma, \text{'}\varphi) \text{ action} \leq (\text{type}) (\text{'}\sigma, \text{'}\varphi) \text{ st-csp hrel}$

notation $\text{csp-vars-child-lens}_a (\Sigma_c)$
notation $\text{csp-vars-child-lens} (\Sigma_C)$

2.2 Basic laws

lemma *R2c-tr-ext*: $R2c (\text{\$tr'} =_u \text{\$tr} \hat{^}_u \langle \lceil a \rceil_{S<} \rangle) = (\text{\$tr'} =_u \text{\$tr} \hat{^}_u \langle \lceil a \rceil_{S<} \rangle)$

by (rel-auto)

lemma *circus-alpha-bij-lens*:

bij-lens ($\{\$ok, \$ok', \$wait, \$wait', \$tr, \$tr', \$st, \$st', \$ref, \$ref'\}_\alpha :: - \implies ('s, 'e) \text{ st-csp} \times ('s, 'e) \text{ st-csp}$)
 by (unfold-locales, lens-simp+)

2.3 Unrestriction laws

lemma *pre-unrest-ref* [unrest]: $\$ref \# P \implies \$ref \# pre_R(P)$
 by (simp add: pre_R-def unrest)

lemma *peri-unrest-ref* [unrest]: $\$ref \# P \implies \$ref \# peri_R(P)$
 by (simp add: peri_R-def unrest)

lemma *post-unrest-ref* [unrest]: $\$ref \# P \implies \$ref \# post_R(P)$
 by (simp add: post_R-def unrest)

lemma *cmt-unrest-ref* [unrest]: $\$ref \# P \implies \$ref \# cmt_R(P)$
 by (simp add: cmt_R-def unrest)

lemma *st-lift-unrest-ref'* [unrest]: $\$ref' \# \lceil b \rceil_{S<} \implies$
 by (rel-auto)

lemma *RHS-design-ref-unrest* [unrest]:

$\llbracket \$ref \# P; \$ref \# Q \rrbracket \implies \$ref \# (\mathbf{R}_s(P \vdash Q)) \llbracket false / \$wait \rrbracket$
 by (simp add: RHS-def R1-def R2c-def R2s-def R3h-def design-def usubst unrest)

lemma *R1-ref-unrest* [unrest]: $\$ref \# P \implies \$ref \# R1(P)$
 by (simp add: R1-def unrest)

lemma *R2c-ref-unrest* [unrest]: $\$ref \# P \implies \$ref \# R2c(P)$
 by (simp add: R2c-def unrest)

lemma *R1-ref'-unrest* [unrest]: $\$ref' \# P \implies \$ref' \# R1(P)$
 by (simp add: R1-def unrest)

lemma *R2c-ref'-unrest* [unrest]: $\$ref' \# P \implies \$ref' \# R2c(P)$
 by (simp add: R2c-def unrest)

lemma *R2s-notin-ref'*: $R2s(\lceil \ll x \gg \rceil_{S<} \notin_u \$ref') = (\lceil \ll x \gg \rceil_{S<} \notin_u \$ref')$
 by (pred-auto)

lemma *unrest-circus-alpha*:

fixes $P :: ('e, 't) \text{ action}$

assumes

$\$ok \# P \ \$ok' \# P \ \$wait \# P \ \$wait' \# P \ \$tr \# P$
 $\$tr' \# P \ \$st \# P \ \$st' \# P \ \$ref \# P \ \$ref' \# P$

shows $\Sigma \# P$

by (rule bij-lens-unrest-all[OF circus-alpha-bij-lens], simp add: unrest assms)

lemma *unrest-all-circus-vars*:

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

assumes $\$ok \# P \ \$ok' \# P \ \$wait \# P \ \$wait' \# P \ \$ref \# P \ \Sigma \# r' \ \Sigma \# s \ \Sigma \# s' \ \Sigma \# t \ \Sigma \# t'$
 shows $\Sigma \# [\$ref' \mapsto_s r', \$st \mapsto_s s, \$st' \mapsto_s s', \$tr \mapsto_s t, \$tr' \mapsto_s t'] \dagger P$

using assms

by (simp add: bij-lens-unrest-all-eq[OF circus-alpha-bij-lens] unrest-plus-split plus-vwb-lens)

(simp add: unrest usubst closure)

lemma *unrest-all-circus-vars-st-st'*:

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

assumes $\$ok \# P \$ok' \# P \$wait \# P \$wait' \# P \$ref \# P \$ref' \# P \Sigma \# s \Sigma \# s' \Sigma \# t \Sigma \# t'$

shows $\Sigma \# [\$st \mapsto_s s, \$st' \mapsto_s s', \$tr \mapsto_s t, \$tr' \mapsto_s t'] \uparrow P$

using *assms*

by (simp add: bij-lens-unrest-all-eq[*OF* circus-alpha-bij-lens] unrest-plus-split plus-vwb-lens)

(simp add: unrest usubst closure)

lemma *unrest-all-circus-vars-st*:

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

assumes $\$ok \# P \$ok' \# P \$wait \# P \$wait' \# P \$ref \# P \$ref' \# P \$st' \# P \Sigma \# s \Sigma \# t \Sigma \# t'$

shows $\Sigma \# [\$st \mapsto_s s, \$tr \mapsto_s t, \$tr' \mapsto_s t'] \uparrow P$

using *assms*

by (simp add: bij-lens-unrest-all-eq[*OF* circus-alpha-bij-lens] unrest-plus-split plus-vwb-lens)

(simp add: unrest usubst closure)

lemma *unrest-any-circus-var*:

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

assumes $\$ok \# P \$ok' \# P \$wait \# P \$wait' \# P \$ref \# P \$ref' \# P \Sigma \# s \Sigma \# s' \Sigma \# t \Sigma \# t'$

shows $x \# [\$st \mapsto_s s, \$st' \mapsto_s s', \$tr \mapsto_s t, \$tr' \mapsto_s t'] \uparrow P$

by (simp add: unrest-all-var unrest-all-circus-vars-st-st' assms)

lemma *unrest-any-circus-var-st*:

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

assumes $\$ok \# P \$ok' \# P \$wait \# P \$wait' \# P \$ref \# P \$ref' \# P \$st' \# P \Sigma \# s \Sigma \# t \Sigma \# t'$

shows $x \# [\$st \mapsto_s s, \$tr \mapsto_s t, \$tr' \mapsto_s t'] \uparrow P$

by (simp add: unrest-all-var unrest-all-circus-vars-st assms)

end

3 Circus Reactive Relations

theory *utp-circus-rel*

imports *utp-circus-core*

begin

3.1 Healthiness Conditions

CSP Reactive Relations

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

[*upred-defs*]: $CRR(P) = (\exists \$ref \cdot RR(P))$

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

by (*rel-auto*)

lemma *Idempotent-CRR* [*closure*]: *Idempotent CRR*

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

lemma *CRR-intro*:

assumes $\$ref \# P$ *P is RR*

shows *P is CRR*

by (simp add: *CRR-def Healthy-def, simp add: Healthy-if assms ex-unrest*)

CSP Reactive Conditions

definition $CRC :: ('s, 'e) \text{ action} \Rightarrow ('s, 'e) \text{ action}$ **where**
 $[upred-defs]: CRC(P) = (\exists \$ref \cdot RC(P))$

lemma $CRC\text{-}intro$:
assumes $\$ref \# P$ P *is* RC
shows P *is* CRC
by (*simp add: CRC-def Healthy-def, simp add: Healthy-if assms ex-unrest*)

lemma $ref\text{-}unrest\text{-}RR$ [*unrest*]: $\$ref \# P \Longrightarrow \$ref \# RR\ P$
by (*rel-auto, blast+*)

lemma $ref\text{-}unrest\text{-}RC1$ [*unrest*]: $\$ref \# P \Longrightarrow \$ref \# RC1\ P$
by (*rel-auto, blast+*)

lemma $ref\text{-}unrest\text{-}RC$ [*unrest*]: $\$ref \# P \Longrightarrow \$ref \# RC\ P$
by (*simp add: RC-R2-def ref-unrest-RC1 ref-unrest-RR*)

lemma $RR\text{-}ex\text{-}ref$: $RR (\exists \$ref \cdot RR\ P) = (\exists \$ref \cdot RR\ P)$
by (*rel-auto*)

lemma $RC1\text{-}ex\text{-}ref$: $RC1 (\exists \$ref \cdot RC1\ P) = (\exists \$ref \cdot RC1\ P)$
by (*rel-auto, meson dual-order.trans*)

lemma $ex\text{-}ref'\text{-}RR\text{-}closed$ [*closure*]:
assumes P *is* RR
shows $(\exists \$ref' \cdot P)$ *is* RR
proof –
have $RR (\exists \$ref' \cdot RR(P)) = (\exists \$ref' \cdot RR(P))$
by (*rel-auto*)
thus *?thesis*
by (*metis Healthy-def assms*)
qed

lemma $CRC\text{-}idem$: $CRC(CRC(P)) = CRC(P)$
apply (*simp add: CRC-def ex-unrest unrest*)
apply (*simp add: RC-def RR-ex-ref*)
apply (*metis (no-types, hide-lams) Healthy-def RC1-RR-closed RC1-ex-ref RR-ex-ref RR-idem*)
done

lemma $Idempotent\text{-}CRC$ [*closure*]: *Idempotent* CRC
by (*simp add: CRC-idem Idempotent-def*)

3.2 Closure Properties

lemma $CRR\text{-}implies\text{-}RR$ [*closure*]:
assumes P *is* CRR
shows P *is* RR
proof –
have $RR(CRR(P)) = CRR(P)$
by (*rel-auto*)
thus *?thesis*
by (*metis Healthy-def' assms*)
qed

lemma *CRC-implies-RR* [closure]:
 assumes *P* is CRC
 shows *P* is RR
proof –
 have $RR(CRC(P)) = CRC(P)$
 by (*rel-auto*)
 (*metis* (*no-types*, *lifting*) *Prefix-Order.prefixE Prefix-Order.prefixI append.assoc append-minus*) +
 thus ?thesis
 by (*metis* *Healthy-def assms*)
qed

lemma *CRC-implies-RC* [closure]:
 assumes *P* is CRC
 shows *P* is RC
proof –
 have $RC1(CRC(P)) = CRC(P)$
 by (*rel-auto*, *meson dual-order.trans*)
 thus ?thesis
 by (*simp add: CRC-implies-RR Healthy-if RC1-def RC-intro assms*)
qed

lemma *CRR-unrest-ref* [*unrest*]: P is CRR $\implies \$ref \# P$
 by (*metis* *CRR-def CRR-implies-RR Healthy-def in-var-uvar ref-vwb-lens unrest-as-exists*)

lemma *CRC-implies-CRR* [closure]:
 assumes *P* is CRC
 shows *P* is CRR
 apply (*rule CRR-intro*)
 apply (*simp-all add: unrest assms closure*)
 apply (*metis* *CRC-def CRC-implies-RC Healthy-def assms in-var-uvar ref-vwb-lens unrest-as-exists*)
 done

lemma *unrest-ref'-neg-RC* [*unrest*]:
 assumes *P* is RR *P* is RC
 shows $\$ref' \# P$
proof –
 have $P = (\neg_r \neg_r P)$
 by (*simp add: closure rpred assms*)
 also have $\dots = (\neg_r (\neg_r P) ;; true_r)$
 by (*metis* *Healthy-if RC1-def RC-implies-RC1 assms(2) calculation*)
 also have $\$ref' \# \dots$
 by (*rel-auto*)
 finally show ?thesis .
qed

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

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

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

lemma *false-CRC* [closure]: *false* is CRC

by (rel-auto)

lemma *st-pred-CRR* [closure]: $[P]_{S<} \text{ is CRR}$
 by (rel-auto)

lemma *st-cond-CRC* [closure]: $[P]_{S<} \text{ is CRC}$
 by (rel-auto)

lemma *conj-CRC-closed* [closure]:
 $\llbracket P \text{ is CRC}; Q \text{ is CRC} \rrbracket \implies (P \wedge Q) \text{ is CRC}$
 by (rule CRC-intro, simp-all add: unrest closure)

lemma *disj-CRC-closed* [closure]:
 $\llbracket P \text{ is CRC}; Q \text{ is CRC} \rrbracket \implies (P \vee Q) \text{ is CRC}$
 by (rule CRC-intro, simp-all add: unrest closure)

lemma *shEx-CRR-closed* [closure]:
 assumes $\bigwedge x. P\ x \text{ is CRR}$
 shows $(\exists x. P(x)) \text{ is CRR}$
proof –
 have $CRR(\exists x. CRR(P(x))) = (\exists x. CRR(P(x)))$
 by (rel-auto)
 thus ?thesis
 by (metis Healthy-def assms shEx-cong)
qed

lemma *USUP-ind-CRR-closed* [closure]:
 assumes $\bigwedge i. P\ i \text{ is CRR}$
 shows $(\bigsqcup i. P(i)) \text{ is CRR}$
 by (rule CRR-intro, simp-all add: assms unrest closure)

lemma *UINF-ind-CRR-closed* [closure]:
 assumes $\bigwedge i. P\ i \text{ is CRR}$
 shows $(\bigcap i. P(i)) \text{ is CRR}$
 by (rule CRR-intro, simp-all add: assms unrest closure)

lemma *cond-tt-CRR-closed* [closure]:
 assumes $P \text{ is CRR } Q \text{ is CRR}$
 shows $P \triangleleft \$tr' =_u \$tr \triangleright Q \text{ is CRR}$
 by (rule CRR-intro, simp-all add: unrest assms closure)

lemma *rea-implies-CRR-closed* [closure]:
 $\llbracket P \text{ is CRR}; Q \text{ is CRR} \rrbracket \implies (P \Rightarrow_r Q) \text{ is CRR}$
 by (simp-all add: CRR-intro closure unrest)

lemma *conj-CRR-closed* [closure]:
 $\llbracket P \text{ is CRR}; Q \text{ is CRR} \rrbracket \implies (P \wedge Q) \text{ is CRR}$
 by (simp-all add: CRR-intro closure unrest)

lemma *disj-CRR-closed* [closure]:
 $\llbracket P \text{ is CRR}; Q \text{ is CRR} \rrbracket \implies (P \vee Q) \text{ is CRR}$
 by (rule CRR-intro, simp-all add: unrest closure)

lemma *rea-not-CRR-closed* [closure]:
 $P \text{ is CRR} \implies (\neg_r P) \text{ is CRR}$

using *false-CRR rea-implies-CRR-closed* **by** *fastforce*

lemma *disj-R1-closed* [closure]: $\llbracket P \text{ is } R1; Q \text{ is } R1 \rrbracket \implies (P \vee Q) \text{ is } R1$
by (*rel-blast*)

lemma *st-cond-R1-closed* [closure]: $\llbracket P \text{ is } R1; Q \text{ is } R1 \rrbracket \implies (P \triangleleft b \triangleright_R Q) \text{ is } R1$
by (*rel-blast*)

lemma *cond-st-RR-closed* [closure]:
assumes $P \text{ is } RR \ Q \text{ is } RR$
shows $(P \triangleleft b \triangleright_R Q) \text{ is } RR$
apply (*rule RR-intro, simp-all add: unrest closure assms, simp add: Healthy-def R2c-condr*)
apply (*simp add: Healthy-if assms RR-implies-R2c*)
apply (*rel-auto*)
done

lemma *cond-st-CRR-closed* [closure]:
 $\llbracket P \text{ is } CRR; Q \text{ is } CRR \rrbracket \implies (P \triangleleft b \triangleright_R Q) \text{ is } CRR$
by (*simp-all add: CRR-intro closure unrest*)

lemma *tr-extend-seqr-lit* [rdes]:
fixes $P :: ('s, 'e) \text{ action}$
assumes $\$ok \# P \ \$wait \# P \ \$ref \# P$
shows $(\$tr' =_u \$tr \hat{\ }_u \langle \llbracket a \rrbracket \rangle \wedge \$st' =_u \$st) ;; P = P[\$tr \hat{\ }_u \langle \llbracket a \rrbracket \rangle / \$tr]$
using *assms* **by** (*rel-auto, meson*)

lemma *tr-assign-comp* [rdes]:
fixes $P :: ('s, 'e) \text{ action}$
assumes $\$ok \# P \ \$wait \# P \ \$ref \# P$
shows $(\$tr' =_u \$tr \wedge \lceil \langle \sigma \rangle_a \rceil_s) ;; P = \lceil \sigma \rceil_{s\sigma} \dagger P$
using *assms* **by** (*rel-auto, meson*)

lemma *RR-msubst-tt*: $RR((P \ t) \llbracket t \rightarrow \&tt \rrbracket) = (RR \ (P \ t)) \llbracket t \rightarrow \&tt \rrbracket$
by (*rel-auto*)

lemma *RR-msubst-ref'*: $RR((P \ r) \llbracket r \rightarrow \$ref' \rrbracket) = (RR \ (P \ r)) \llbracket r \rightarrow \$ref' \rrbracket$
by (*rel-auto*)

lemma *msubst-tt-RR* [closure]: $\llbracket \bigwedge t. P \ t \text{ is } RR \rrbracket \implies (P \ t) \llbracket t \rightarrow \&tt \rrbracket \text{ is } RR$
by (*simp add: Healthy-def RR-msubst-tt*)

lemma *msubst-ref'-RR* [closure]: $\llbracket \bigwedge r. P \ r \text{ is } RR \rrbracket \implies (P \ r) \llbracket r \rightarrow \$ref' \rrbracket \text{ is } RR$
by (*simp add: Healthy-def RR-msubst-ref'*)

lemma *conj-less-tr-RR-closed* [closure]:
assumes $P \text{ is } CRR$
shows $(P \wedge \$tr <_u \$tr') \text{ is } CRR$
proof –
have $CRR(CRR(P) \wedge \$tr <_u \$tr') = (CRR(P) \wedge \$tr <_u \$tr')$
apply (*rel-auto, blast+*)
using *less-le* **apply** *fastforce+*
done
thus *?thesis*
by (*metis Healthy-def assms*)
qed

lemma *conj-eq-tr-RR-closed* [closure]:
assumes P is CRR
shows $(P \wedge \$tr' =_u \$tr)$ is CRR
proof –
have $CRR(CRR(P) \wedge \$tr' =_u \$tr) = (CRR(P) \wedge \$tr' =_u \$tr)$
by (*rel-auto*, *blast+*)
thus ?thesis
by (*metis Healthy-def assms*)
qed

3.3 Introduction laws

Extensionality principles for introducing refinement and equality of Circus reactive relations. It is necessary only to consider a subset of the variables that are present.

lemma *CRR-refine-ext*:
assumes
 P is CRR Q is CRR
 $\bigwedge t s s' r'. P[\langle \rangle, \langle t \rangle, \langle s \rangle, \langle s' \rangle, \langle r' \rangle / \$tr, \$tr', \$st, \$st', \$ref'] \sqsubseteq Q[\langle \rangle, \langle t \rangle, \langle s \rangle, \langle s' \rangle, \langle r' \rangle / \$tr, \$tr', \$st, \$st', \$ref']$
shows $P \sqsubseteq Q$
proof –
have $\bigwedge t s s' r'. (CRR P)[\langle \rangle, \langle t \rangle, \langle s \rangle, \langle s' \rangle, \langle r' \rangle / \$tr, \$tr', \$st, \$st', \$ref']$
 $\sqsubseteq (CRR Q)[\langle \rangle, \langle t \rangle, \langle s \rangle, \langle s' \rangle, \langle r' \rangle / \$tr, \$tr', \$st, \$st', \$ref']$
by (*simp add: assms Healthy-if*)
hence $CRR P \sqsubseteq CRR Q$
by (*rel-auto*)
thus ?thesis
by (*metis Healthy-if assms(1) assms(2)*)
qed

lemma *CRR-eq-ext*:
assumes
 P is CRR Q is CRR
 $\bigwedge t s s' r'. P[\langle \rangle, \langle t \rangle, \langle s \rangle, \langle s' \rangle, \langle r' \rangle / \$tr, \$tr', \$st, \$st', \$ref'] = Q[\langle \rangle, \langle t \rangle, \langle s \rangle, \langle s' \rangle, \langle r' \rangle / \$tr, \$tr', \$st, \$st', \$ref']$
shows $P = Q$
proof –
have $\bigwedge t s s' r'. (CRR P)[\langle \rangle, \langle t \rangle, \langle s \rangle, \langle s' \rangle, \langle r' \rangle / \$tr, \$tr', \$st, \$st', \$ref']$
 $= (CRR Q)[\langle \rangle, \langle t \rangle, \langle s \rangle, \langle s' \rangle, \langle r' \rangle / \$tr, \$tr', \$st, \$st', \$ref']$
by (*simp add: assms Healthy-if*)
hence $CRR P = CRR Q$
by (*rel-auto*)
thus ?thesis
by (*metis Healthy-if assms(1) assms(2)*)
qed

lemma *CRR-refine-impl-prop*:
assumes P is CRR Q is CRR
 $\bigwedge t s s' r'. 'Q[\langle r' \rangle, \langle s \rangle, \langle s' \rangle, \langle \rangle, \langle t \rangle / \$ref', \$st, \$st', \$tr, \$tr']' \implies 'P[\langle r' \rangle, \langle s \rangle, \langle s' \rangle, \langle \rangle, \langle t \rangle / \$ref', \$st, \$st', \$tr, \$tr']'$
shows $P \sqsubseteq Q$
by (*rule CRR-refine-ext, simp-all add: assms closure unrest usubst*)
(rule refine-prop-intro, simp-all add: unrest unrest-all-circus-vars closure assms)

3.4 Trace Substitution

definition *trace-subst* $(-[\cdot]_t [999, 0] 999)$

where $[upred-defs]: P\llbracket v \rrbracket_t = (P\llbracket \&tt - \lceil v \rceil_{S<} / \&tt \rrbracket \wedge \$tr + \lceil v \rceil_{S<} \leq_u \$tr')$

lemma *unrest-trace-subst* $[unrest]$:

$\llbracket mwb-lens\ x; x \bowtie (\$tr)_v; x \bowtie (\$tr')_v; x \bowtie (\$st)_v; x \# P \rrbracket \implies x \# P\llbracket v \rrbracket_t$
by (*simp add: trace-subst-def lens-indep-sym unrest*)

lemma *trace-subst-RR-closed* $[closure]$:

assumes P is *RR*

shows $P\llbracket v \rrbracket_t$ is *RR*

proof –

have $(RR\ P)\llbracket v \rrbracket_t$ is *RR*

apply (*rel-auto*)

apply (*metis diff-add-cancel-left' trace-class.add-left-mono*)

apply (*metis le-add minus-cancel-le trace-class.add-diff-cancel-left*)

using *le-add order-trans* **apply** *blast*

done

thus *?thesis*

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

qed

lemma *trace-subst-CRR-closed* $[closure]$:

assumes P is *CRR*

shows $P\llbracket v \rrbracket_t$ is *CRR*

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

lemma *tsubst-nil* $[usubst]$:

assumes P is *CRR*

shows $P\llbracket \langle \rangle \rrbracket_t = P$

proof –

have $(CRR\ P)\llbracket \langle \rangle \rrbracket_t = CRR\ P$

by (*rel-auto*)

thus *?thesis*

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

qed

lemma *tsubst-false* $[usubst]$: $false\llbracket y \rrbracket_t = false$

by *rel-auto*

lemma *cond-rea-tt-subst* $[usubst]$:

$(P \triangleleft b \triangleright_R Q)\llbracket v \rrbracket_t = (P\llbracket v \rrbracket_t \triangleleft b \triangleright_R Q\llbracket v \rrbracket_t)$

by (*rel-auto*)

lemma *tsubst-conj* $[usubst]$: $(P \wedge Q)\llbracket v \rrbracket_t = (P\llbracket v \rrbracket_t \wedge Q\llbracket v \rrbracket_t)$

by (*rel-auto*)

lemma *tsubst-disj* $[usubst]$: $(P \vee Q)\llbracket v \rrbracket_t = (P\llbracket v \rrbracket_t \vee Q\llbracket v \rrbracket_t)$

by (*rel-auto*)

lemma *rea-subst-R1-closed* $[closure]$: $P\llbracket v \rrbracket_t$ is *R1*

apply (*rel-auto*) **using** *le-add order.trans* **by** *blast*

lemma *tsubst-UINF-ind* $[usubst]$: $(\bigcap i \cdot P(i))\llbracket v \rrbracket_t = (\bigcap i \cdot (P(i))\llbracket v \rrbracket_t)$

by (*rel-auto*)

3.5 Initial Interaction

definition $rea-init :: 's \text{ upred} \Rightarrow ('t::trace, 's) \text{ uexpr} \Rightarrow ('s, 't, 'a, 'b) \text{ rel-rsp } (\mathcal{I}'(-, -))$ **where**
 $[upred-defs]: \mathcal{I}(s, t) = ([s]_{S<} \wedge \$tr + [t]_{S<} \leq_u \$tr')$

$\mathcal{I}(s, t)$ is a predicate stating that, if the initial state satisfies state predicate s , then the trace t is an initial trace.

lemma $unrest-rea-init$ $[unrest]:$
 $\llbracket x \bowtie (\$tr)_v; x \bowtie (\$tr')_v; x \bowtie (\$st)_v \rrbracket \implies x \# \mathcal{I}(s, t)$
by ($simp$ $add: rea-init-def$ $unrest$ $lens-indep-sym$)

lemma $rea-init-R1$ $[closure]: \mathcal{I}(s, t)$ *is* $R1$
apply ($rel-auto$) **using** $dual-order.trans$ $le-add$ **by** $blast$

lemma $rea-init-R2c$ $[closure]: \mathcal{I}(s, t)$ *is* $R2c$
apply ($rel-auto$)
apply ($metis$ $diff-add-cancel-left'$ $trace-class.add-left-mono$)
apply ($metis$ $le-add$ $minus-cancel-le$ $trace-class.add-diff-cancel-left$)
done

lemma $rea-init-R2$ $[closure]: \mathcal{I}(s, t)$ *is* $R2$
by ($metis$ $Healthy-def$ $R1-R2c-is-R2$ $rea-init-R1$ $rea-init-R2c$)

lemma $csp-init-RR$ $[closure]: \mathcal{I}(s, t)$ *is* RR
apply ($rel-auto$)
apply ($metis$ $diff-add-cancel-left'$ $trace-class.add-left-mono$)
apply ($metis$ $le-add$ $minus-cancel-le$ $trace-class.add-diff-cancel-left$)
apply ($metis$ $le-add$ $less-le$ $less-le-trans$)
done

lemma $csp-init-CRR$ $[closure]: \mathcal{I}(s, t)$ *is* CRR
by ($rule$ $CRR-intro$, $simp-all$ $add: unrest$ $closure$)

lemma $rea-init-impl-st$ $[closure]: (\mathcal{I}(b, t) \Rightarrow_r [c]_{S<})$ *is* RC
apply ($rule$ $RC-intro$)
apply ($simp$ $add: closure$)
apply ($rel-auto$)
using $order-trans$ **by** $auto$

lemma $rea-init-RC1:$
 $\neg_r \mathcal{I}(P, t)$ *is* $RC1$
apply ($rel-auto$) **using** $dual-order.trans$ **by** $blast$

lemma $init-acts-empty$ $[rpred]: \mathcal{I}(true, \langle \rangle) = true_r$
by ($rel-auto$)

lemma $rea-not-init$ $[rpred]:$
 $(\neg_r \mathcal{I}(P, \langle \rangle)) = \mathcal{I}(\neg P, \langle \rangle)$
by ($rel-auto$)

lemma $rea-init-conj$ $[rpred]:$
 $(\mathcal{I}(P, t) \wedge \mathcal{I}(Q, t)) = \mathcal{I}(P \wedge Q, t)$
by ($rel-auto$)

lemma $rea-init-empty-trace$ $[rpred]: \mathcal{I}(s, \langle \rangle) = [s]_{S<}$
by ($rel-auto$)

lemma *rea-init-disj-same* [*rpred*]: $(\mathcal{I}(s_1, t) \vee \mathcal{I}(s_2, t)) = \mathcal{I}(s_1 \vee s_2, t)$
by (*rel-auto*)

lemma *rea-init-impl-same* [*rpred*]: $(\mathcal{I}(s_1, t) \Rightarrow_r \mathcal{I}(s_2, t)) = (\mathcal{I}(s_1, t) \Rightarrow_r [s_2]_{S<})$
apply (*rel-auto*) **using** *dual-order.trans le-add* **by** *blast+*

lemma *tsubst-st-cond* [*usubst*]: $[P]_{S<} \llbracket t \rrbracket_t = \mathcal{I}(P, t)$
by (*rel-auto*)

lemma *tsubst-rea-init* [*usubst*]: $(\mathcal{I}(s, x)) \llbracket y \rrbracket_t = \mathcal{I}(s, y+x)$
apply (*rel-auto*)
apply (*metis add.assoc diff-add-cancel-left' trace-class.add-le-imp-le-left trace-class.add-left-mono*)
apply (*metis add.assoc diff-add-cancel-left' le-add trace-class.add-le-imp-le-left trace-class.add-left-mono*)
done

lemma *tsubst-rea-not* [*usubst*]: $(\neg_r P) \llbracket v \rrbracket_t = ((\neg_r P \llbracket v \rrbracket_t) \wedge \mathcal{I}(\text{true}, v))$
apply (*rel-auto*)
using *le-add order-trans* **by** *blast*

lemma *tsubst-true* [*usubst*]: $\text{true}_r \llbracket v \rrbracket_t = \mathcal{I}(\text{true}, v)$
by (*rel-auto*)

3.6 Enabled Events

definition *csp-enable* :: $'s \text{ upred} \Rightarrow ('e \text{ list}, 's) \text{ uexpr} \Rightarrow ('e \text{ set}, 's) \text{ uexpr} \Rightarrow ('s, 'e) \text{ action } (\mathcal{E}'(-, -, -'))$
where
 $[upred\text{-defs}]: \mathcal{E}(s, t, E) = ([s]_{S<} \wedge \$tr' =_u \$tr \wedge_u [t]_{S<} \wedge (\forall e \in [E]_{S<} \cdot \llbracket e \rrbracket \notin_u \$ref'))$

Predicate $\mathcal{E}(s, t, E)$ states that, if the initial state satisfies predicate s , then t is a possible (failure) trace, such that the events in the set E are enabled after the given interaction.

lemma *csp-enable-R1-closed* [*closure*]: $\mathcal{E}(s, t, E)$ is *R1*
by (*rel-auto*)

lemma *csp-enable-R2-closed* [*closure*]: $\mathcal{E}(s, t, E)$ is *R2c*
by (*rel-auto*)

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

lemma *tsubst-csp-enable* [*usubst*]: $\mathcal{E}(s, t_2, e) \llbracket t_1 \rrbracket_t = \mathcal{E}(s, t_1 \wedge_u t_2, e)$
apply (*rel-auto*)
apply (*metis append.assoc less-eq-list-def prefix-concat-minus*)
apply (*simp add: list-concat-minus-list-concat*)
done

lemma *csp-enable-unrests* [*unrest*]:
 $\llbracket x \bowtie (\$tr)_v; x \bowtie (\$tr')_v; x \bowtie (\$st)_v; x \bowtie (\$ref')_v \rrbracket \Longrightarrow x \# \mathcal{E}(s, t, e)$
by (*simp add: csp-enable-def R1-def lens-indep-sym unrest*)

lemma *csp-enable-tr'-eq-tr* [*rpred*]:
 $\mathcal{E}(s, \langle \rangle, r) \triangleleft \$tr' =_u \$tr \triangleright \text{false} = \mathcal{E}(s, \langle \rangle, r)$
by (*rel-auto*)

lemma *csp-enable-st-pred* [*rpred*]:

$([s_1]_{S<} \wedge \mathcal{E}(s_2, t, E)) = \mathcal{E}(s_1 \wedge s_2, t, E)$
by (*rel-auto*)

lemma *csp-enable-tr-empty*: $\mathcal{E}(\text{true}, \langle \rangle, \{v\}_u) = (\$tr' =_u \$tr \wedge [v]_{S<} \notin_u \$ref')$
by (*rel-auto*)

lemma *csp-enable-nothing*: $\mathcal{E}(\text{true}, \langle \rangle, \{\}_u) = (\$tr' =_u \$tr)$
by (*rel-auto*)

lemma *msubst-nil-csp-enable* [*usubst*]:
 $\mathcal{E}(s(x), t(x), E(x)) \llbracket x \rightarrow \langle \rangle \rrbracket = \mathcal{E}(s(x) \llbracket x \rightarrow \langle \rangle \rrbracket, t(x) \llbracket x \rightarrow \langle \rangle \rrbracket, E(x) \llbracket x \rightarrow \langle \rangle \rrbracket)$
by (*pred-auto*)

lemma *msubst-csp-enable* [*usubst*]:
 $\mathcal{E}(s(x), t(x), E(x)) \llbracket x \rightarrow [v]_{S\leftarrow} \rrbracket = \mathcal{E}(s(x) \llbracket x \rightarrow v \rrbracket, t(x) \llbracket x \rightarrow v \rrbracket, E(x) \llbracket x \rightarrow v \rrbracket)$
by (*rel-auto*)

lemma *csp-enable-false* [*rpred*]: $\mathcal{E}(\text{false}, t, E) = \text{false}$
by (*rel-auto*)

lemma *USUP-csp-enable* [*rpred*]:
 $(\bigsqcup x \cdot \mathcal{E}(s, t, A(x))) = \mathcal{E}(s, t, (\bigvee x \cdot A(x)))$
by (*rel-auto*)

3.7 Completed Trace Interaction

definition *csp-do* :: $'s \text{ upred} \Rightarrow ('s \Rightarrow 's) \Rightarrow ('e \text{ list}, 's) \text{ uexpr} \Rightarrow ('s, 'e) \text{ action } (\Phi'(-, -, -))$ **where**
 $[upred\text{-defs}]: \Phi(s, \sigma, t) = ([s]_{S<} \wedge \$tr' =_u \$tr \hat{^}_u [t]_{S<} \wedge [\langle \sigma \rangle_a]_S)$

Predicate $\Phi(s, \sigma, t)$ states that if the initial state satisfies s , and the trace t is performed, then afterwards the state update σ is executed.

lemma *unrest-csp-do* [*unrest*]:
 $\llbracket x \bowtie (\$tr)_v; x \bowtie (\$tr')_v; x \bowtie (\$st)_v; x \bowtie (\$st')_v \rrbracket \Longrightarrow x \# \Phi(s, \sigma, t)$
by (*simp-all add: csp-do-def alpha-in-var alpha-out-var prod-as-plus unrest lens-indep-sym*)

lemma *csp-do-CRR* [*closure*]: $\Phi(s, \sigma, t)$ *is CRR*
by (*rel-auto*)

lemma *trea-subst-csp-do* [*usubst*]:
 $(\Phi(s, \sigma, t_2)) \llbracket t_1 \rrbracket_t = \Phi(s, \sigma, t_1 \hat{^}_u t_2)$
apply (*rel-auto*)
apply (*metis append.assoc less-eq-list-def prefix-concat-minus*)
apply (*simp add: list-concat-minus-list-concat*)
done

lemma *st-subst-csp-do* [*usubst*]:
 $[\sigma]_{S\sigma} \dagger \Phi(s, \varrho, t) = \Phi(\sigma \dagger s, \varrho \circ \sigma, \sigma \dagger t)$
by (*rel-auto*)

lemma *csp-init-do* [*rpred*]: $(\mathcal{I}(s1, t) \wedge \Phi(s2, \sigma, t)) = \Phi(s1 \wedge s2, \sigma, t)$
by (*rel-auto*)

lemma *csp-do-false* [*rpred*]: $\Phi(\text{false}, s, t) = \text{false}$
by (*rel-auto*)

lemma *csp-do-assign* [*rpred*]:

assumes *P* is *CRR*

shows $\Phi(s, \sigma, t) ;; P = ([s]_{S<} \wedge ([\sigma]_{S\sigma} \dagger P))\llbracket t \rrbracket_t$

proof –

have $\Phi(s, \sigma, t) ;; CRR(P) = ([s]_{S<} \wedge ([\sigma]_{S\sigma} \dagger CRR(P)))\llbracket t \rrbracket_t$

by (*rel-blast*)

thus *?thesis*

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

qed

lemma *subst-state-csp-enable* [*usubst*]:

$[\sigma]_{S\sigma} \dagger \mathcal{E}(s, t_2, e) = \mathcal{E}(\sigma \dagger s, \sigma \dagger t_2, \sigma \dagger e)$

by (*rel-auto*)

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

$\Phi(s_1, \sigma, t_1) ;; \mathcal{E}(s_2, t_2, e) = \mathcal{E}(s_1 \wedge \sigma \dagger s_2, t_1 \hat{^}_u(\sigma \dagger t_2), (\sigma \dagger e))$

by (*simp add: rpred closure usubst*)

lemma *csp-do-assign-do* [*rpred*]:

$\Phi(s_1, \sigma, t_1) ;; \Phi(s_2, \varrho, t_2) = \Phi(s_1 \wedge (\sigma \dagger s_2), \varrho \circ \sigma, t_1 \hat{^}_u(\sigma \dagger t_2))$

by (*rel-auto*)

lemma *csp-do-skip* [*rpred*]:

assumes *P* is *CRR*

shows $\Phi(\text{true}, \text{id}, t) ;; P = P\llbracket t \rrbracket_t$

proof –

have $\Phi(\text{true}, \text{id}, t) ;; CRR(P) = (CRR P)\llbracket t \rrbracket_t$

by (*rel-auto*)

thus *?thesis*

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

qed

lemma *wp-rea-csp-do-lemma*:

fixes *P* :: (*'σ*, *'φ*) action

assumes $\$ok \# P \$wait \# P \$ref \# P$

shows $([\langle \sigma \rangle_a]_S \wedge \$tr' \hat{^}_u \$tr \hat{^}_u [t]_{S<}) ;; P = ([\sigma]_{S\sigma} \dagger P)\llbracket \$tr \hat{^}_u [t]_{S<} / \$tr \rrbracket$

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

lemma *wp-rea-csp-do* [*wp*]:

fixes *P* :: (*'σ*, *'φ*) action

assumes *P* is *CRR*

shows $\Phi(s, \sigma, t) \text{ wp}_r P = (\mathcal{I}(s, t) \Rightarrow_r ([\sigma]_{S\sigma} \dagger P))\llbracket t \rrbracket_t$

proof –

have $\Phi(s, \sigma, t) \text{ wp}_r CRR(P) = (\mathcal{I}(s, t) \Rightarrow_r ([\sigma]_{S\sigma} \dagger CRR(P)))\llbracket t \rrbracket_t$

by (*rel-blast*)

thus *?thesis*

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

qed

lemma *csp-do-power-Suc* [*rpred*]:

$\Phi(\text{true}, \text{id}, t) \hat{^} (\text{Suc } i) = \Phi(\text{true}, \text{id}, \text{iter}[\text{Suc } i](t))$

by (*induct i, (rel-auto)+*)

lemma *csp-power-do-comp* [*rpred*]:

assumes *P* is *CRR*

shows $\Phi(\text{true}, \text{id}, t) \wedge i \;; P = \Phi(\text{true}, \text{id}, \text{iter}[i](t)) \;; P$
apply (*cases i*)
apply (*simp-all add: rpred usubst assms closure*)
apply (*metis assms csp-do-power-Suc csp-do-skip upred-semiring.power-Suc*)
done

lemma *wp-rea-csp-do-skip* [*wp*]:
fixes $Q \:: ('σ, 'φ) \text{ action}$
assumes $P \text{ is CRR}$
shows $\Phi(s, \text{id}, t) \text{ wp}_r P = (\mathcal{I}(s, t) \Rightarrow_r P \llbracket t \rrbracket_t)$
proof –
have $\Phi(s, \text{id}, t) \text{ wp}_r P = \Phi(s, \text{id}, t) \text{ wp}_r P$
by (*simp add: skip-r-def*)
thus ?thesis **by** (*simp add: wp assms usubst alpha*)
qed

lemma *msubst-csp-do* [*usubst*]:
 $\Phi(s(x), \sigma, t(x)) \llbracket x \rightarrow \lceil v \rceil_{S \leftarrow} \rrbracket = \Phi(s(x) \llbracket x \rightarrow v \rrbracket, \sigma, t(x) \llbracket x \rightarrow v \rrbracket)$
by (*rel-auto*)

end

4 Circus and CSP Healthiness Conditions

theory *utp-circus-healths*
imports *utp-circus-rel*
begin

5 Definitions

We here define extra healthiness conditions for Circus / CSP processes.

abbreviation $CSP1 \:: (('σ, 'φ) \text{ st-csp} \times ('σ, 'φ) \text{ st-csp}) \text{ health}$
where $CSP1(P) \equiv RD1(P)$

abbreviation $CSP2 \:: (('σ, 'φ) \text{ st-csp} \times ('σ, 'φ) \text{ st-csp}) \text{ health}$
where $CSP2(P) \equiv RD2(P)$

abbreviation $CSP \:: (('σ, 'φ) \text{ st-csp} \times ('σ, 'φ) \text{ st-csp}) \text{ health}$
where $CSP(P) \equiv SRD(P)$

definition $STOP \:: 'φ \text{ rel-csp where}$
 $[upred-defs]: STOP = CSP1(\$ok' \wedge R3c(\$tr' =_u \$tr \wedge \$wait'))$

definition $SKIP \:: 'φ \text{ rel-csp where}$
 $[upred-defs]: SKIP = \mathbf{R}_s(\exists \$ref \cdot CSP1(\$ref \cdot II))$

definition $Stop \:: ('σ, 'φ) \text{ action where}$
 $[upred-defs]: Stop = \mathbf{R}_s(\text{true} \vdash (\$tr' =_u \$tr \wedge \$wait'))$

definition $Skip \:: ('σ, 'φ) \text{ action where}$
 $[upred-defs]: Skip = \mathbf{R}_s(\text{true} \vdash (\$tr' =_u \$tr \wedge \neg \$wait' \wedge \$st' =_u \$st))$

definition $CSP3 \:: (('σ, 'φ) \text{ st-csp} \times ('σ, 'φ) \text{ st-csp}) \text{ health where}$
 $[upred-defs]: CSP3(P) = (Skip \;; P)$

definition $CSP_4 :: ((\sigma, \varphi) \text{ st-csp} \times (\sigma, \varphi) \text{ st-csp}) \text{ health}$ **where**
 $[upred-defs]: CSP_4(P) = (P \mathrel{::} Skip)$

definition $NCSP :: ((\sigma, \varphi) \text{ st-csp} \times (\sigma, \varphi) \text{ st-csp}) \text{ health}$ **where**
 $[upred-defs]: NCSP = CSP_3 \circ CSP_4 \circ CSP$

Productive and normal processes

abbreviation $PCSP \equiv Productive \circ NCSP$

Instantaneous and normal processes

abbreviation $ICSP \equiv ISRD1 \circ NCSP$

5.1 Healthiness condition properties

$SKIP$ is the same as $Skip$, and $STOP$ is the same as $Stop$, when we consider stateless CSP processes. This is because any reference to the st variable degenerates when the alphabet type coerces its type to be empty. We therefore need not consider $SKIP$ and $STOP$ actions.

theorem $SKIP\text{-is-Skip}$: $SKIP = Skip$
by ($rel\text{-auto}$)

theorem $STOP\text{-is-Stop}$: $STOP = Stop$
by ($rel\text{-auto}$)

theorem $Skip\text{-UTP-form}$: $Skip = \mathbf{R}_s(\exists \$ref \cdot CSP1(II))$
by ($rel\text{-auto}$)

lemma $Skip\text{-is-CSP}$ [$closure$]:
 $Skip$ is CSP
by ($simp$ add : $Skip\text{-def}$ $RHS\text{-design-is-SRD}$ $unrest$)

lemma $Skip\text{-RHS-tri-design}$:
 $Skip = \mathbf{R}_s(true \vdash (false \diamond (\$tr' =_u \$tr \wedge \$st' =_u \$st)))$
by ($rel\text{-auto}$)

lemma $Skip\text{-RHS-tri-design}'$ [$rdes\text{-def}$]:
 $Skip = \mathbf{R}_s(true_r \vdash (false \diamond \Phi(true, id, \langle \rangle)))$
by ($rel\text{-auto}$)

lemma $Stop\text{-is-CSP}$ [$closure$]:
 $Stop$ is CSP
by ($simp$ add : $Stop\text{-def}$ $RHS\text{-design-is-SRD}$ $unrest$)

lemma $Stop\text{-RHS-tri-design}$: $Stop = \mathbf{R}_s(true \vdash (\$tr' =_u \$tr) \diamond false)$
by ($rel\text{-auto}$)

lemma $Stop\text{-RHS-rdes-def}$ [$rdes\text{-def}$]: $Stop = \mathbf{R}_s(true_r \vdash \mathcal{E}(true, \langle \rangle, \{\}_u) \diamond false)$
by ($rel\text{-auto}$)

lemma $preR\text{-Skip}$ [$rdes$]: $pre_R(Skip) = true_r$
by ($rel\text{-auto}$)

lemma $periR\text{-Skip}$ [$rdes$]: $peri_R(Skip) = false$
by ($rel\text{-auto}$)

lemma *postR-Skip* [*rdes*]: $\text{post}_R(\text{Skip}) = \Phi(\text{true}, \text{id}, \langle \rangle)$
by (*rel-auto*)

lemma *Productive-Stop* [*closure*]:
Stop is Productive
by (*simp add: Stop-RHS-tri-design Healthy-def Productive-RHS-design-form unrest*)

lemma *Skip-left-lemma*:
assumes *P is CSP*
shows $\text{Skip} ;; P = \mathbf{R}_s ((\forall \$\text{ref} \cdot \text{pre}_R P) \vdash (\exists \$\text{ref} \cdot \text{cmt}_R P))$
proof –
have $\text{Skip} ;; P =$
 $\mathbf{R}_s ((\$tr' =_u \$tr \wedge \$st' =_u \$st) \text{wp}_r \text{pre}_R P \vdash$
 $(\$tr' =_u \$tr \wedge \$st' =_u \$st) ;; \text{peri}_R P \diamond$
 $(\$tr' =_u \$tr \wedge \$st' =_u \$st) ;; \text{post}_R P)$
by (*simp add: SRD-composition-wp alpha rdes closure wp assms rpred C1, rel-auto*)
also have $\dots = \mathbf{R}_s ((\forall \$\text{ref} \cdot \text{pre}_R P) \vdash$
 $(\$tr' =_u \$tr \wedge \neg \$\text{wait}' \wedge \$st' =_u \$st) ;; ((\exists \$st \cdot \lceil II \rceil_D) \triangleleft \$\text{wait} \triangleright \text{cmt}_R P))$
by (*rule cong[of $\mathbf{R}_s \mathbf{R}_s$], simp, rel-auto*)
also have $\dots = \mathbf{R}_s ((\forall \$\text{ref} \cdot \text{pre}_R P) \vdash (\exists \$\text{ref} \cdot \text{cmt}_R P))$
by (*rule cong[of $\mathbf{R}_s \mathbf{R}_s$], simp, rel-auto*)
finally show *?thesis* .
qed

lemma *Skip-left-unit*:
assumes *P is CSP* $\$ref \# P \llbracket \text{false} / \$\text{wait} \rrbracket$
shows $\text{Skip} ;; P = P$
using *assms*
by (*simp add: Skip-left-lemma*)
(metis SRD-reactive-design-alt all-unrest cmt-unrest-ref cmt-wait-false ex-unrest pre-unrest-ref pre-wait-false)

lemma *CSP3-intro*:
 $\llbracket P \text{ is CSP}; \$ref \# P \llbracket \text{false} / \$\text{wait} \rrbracket \rrbracket \implies P \text{ is CSP3}$
by (*simp add: CSP3-def Healthy-def' Skip-left-unit*)

lemma *ref-unrest-RHS-design*:
assumes $\$ref \# P \ \$ref \# Q_1 \ \$ref \# Q_2$
shows $\$ref \# (\mathbf{R}_s(P \vdash Q_1 \diamond Q_2)) \text{f}$
by (*simp add: RHS-def R1-def R2c-def R2s-def R3h-def design-def unrest usubst assms*)

lemma *CSP3-SRD-intro*:
assumes *P is CSP* $\$ref \# \text{pre}_R(P) \ \$ref \# \text{peri}_R(P) \ \$ref \# \text{post}_R(P)$
shows *P is CSP3*
proof –
have $P: \mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \diamond \text{post}_R(P)) = P$
by (*simp add: SRD-reactive-design-alt assms(1) wait'-cond-peri-post-cmt[THEN sym]*)
have $\mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \diamond \text{post}_R(P)) \text{ is CSP3}$
by (*rule CSP3-intro, simp add: assms P, simp add: ref-unrest-RHS-design assms*)
thus *?thesis*
by (*simp add: P*)
qed

lemma *Skip-unrest-ref* [*unrest*]: $\$ref \# \text{Skip} \llbracket \text{false} / \$\text{wait} \rrbracket$
by (*simp add: Skip-def RHS-def R1-def R2c-def R2s-def R3h-def design-def usubst unrest*)

lemma *Skip-unrest-ref'* [*unrest*]: $\$ref' \# Skip \llbracket false/\$wait \rrbracket$
 by (*simp add: Skip-def RHS-def R1-def R2c-def R2s-def R3h-def design-def usubst unrest*)

lemma *CSP3-iff*:
 assumes *P is CSP*
 shows $P \text{ is } CSP3 \longleftrightarrow (\$ref \# P \llbracket false/\$wait \rrbracket)$

proof

assume 1: *P is CSP3*
 have $\$ref \# (Skip ;; P) \llbracket false/\$wait \rrbracket$
 by (*simp add: usubst unrest*)
 with 1 show $\$ref \# P \llbracket false/\$wait \rrbracket$
 by (*metis CSP3-def Healthy-def*)

next

assume 1: $\$ref \# P \llbracket false/\$wait \rrbracket$
 show *P is CSP3*
 by (*simp add: 1 CSP3-intro assms*)

qed

lemma *CSP3-unrest-ref* [*unrest*]:
 assumes *P is CSP* *P is CSP3*
 shows $\$ref \# pre_R(P) \ \$ref \# peri_R(P) \ \$ref \# post_R(P)$

proof –

have $a: (\$ref \# P \llbracket false/\$wait \rrbracket)$
 using *CSP3-iff assms* **by** *blast*
 from *a* show $\$ref \# pre_R(P)$
 by (*rel-blast*)
 from *a* show $\$ref \# peri_R(P)$
 by (*rel-blast*)
 from *a* show $\$ref \# post_R(P)$
 by (*rel-blast*)

qed

lemma *CSP3-rdes*:
 assumes *P is RR* *Q is RR* *R is RR*
 shows $CSP3(\mathbf{R}_s(P \vdash Q \diamond R)) = \mathbf{R}_s((\forall \$ref \cdot P) \vdash (\exists \$ref \cdot Q) \diamond (\exists \$ref \cdot R))$
 by (*simp add: CSP3-def Skip-left-lemma closure assms rdes, rel-auto*)

lemma *CSP3-form*:
 assumes *P is CSP*
 shows $CSP3(P) = \mathbf{R}_s((\forall \$ref \cdot pre_R(P)) \vdash (\exists \$ref \cdot peri_R(P)) \diamond (\exists \$ref \cdot post_R(P)))$
 by (*simp add: CSP3-def Skip-left-lemma assms, rel-auto*)

lemma *CSP3-Skip* [*closure*]:
Skip is CSP3
 by (*rule CSP3-intro, simp add: Skip-is-CSP, simp add: Skip-def unrest*)

lemma *CSP3-Stop* [*closure*]:
Stop is CSP3
 by (*rule CSP3-intro, simp add: Stop-is-CSP, simp add: Stop-def unrest*)

lemma *CSP3-Idempotent* [*closure*]: *Idempotent CSP3*
 by (*metis (no-types, lifting) CSP3-Skip CSP3-def Healthy-if Idempotent-def seqr-assoc*)

lemma *CSP3-Continuous*: *Continuous CSP3*

by (simp add: Continuous-def CSP3-def seq-Sup-distl)

lemma Skip-right-lemma:

assumes P is CSP

shows $P \;; \text{Skip} = \mathbf{R}_s ((\neg_r \text{pre}_R P) \text{wp}_r \text{false} \vdash ((\exists \$st' \cdot \text{cmt}_R P) \triangleleft \$wait' \triangleright (\exists \$ref' \cdot \text{cmt}_R P)))$

proof –

have $P \;; \text{Skip} = \mathbf{R}_s ((\neg_r \text{pre}_R P) \text{wp}_r \text{false} \vdash (\exists \$st' \cdot \text{peri}_R P) \diamond \text{post}_R P \;; (\$tr' =_u \$tr \wedge \$st' =_u \$st))$

by (simp add: SRD-composition-wp closure assms wp rdes rpred, rel-auto)

also have $\dots = \mathbf{R}_s ((\neg_r \text{pre}_R P) \text{wp}_r \text{false} \vdash ((\text{cmt}_R P \;; (\exists \$st \cdot [II]_D)) \triangleleft \$wait' \triangleright (\text{cmt}_R P \;; (\$tr' =_u \$tr \wedge \neg \$wait \wedge \$st' =_u \$st))))$

by (rule cong[of $\mathbf{R}_s \mathbf{R}_s$], simp, rel-auto)

also have $\dots = \mathbf{R}_s ((\neg_r \text{pre}_R P) \text{wp}_r \text{false} \vdash ((\exists \$st' \cdot \text{cmt}_R P) \triangleleft \$wait' \triangleright (\text{cmt}_R P \;; (\$tr' =_u \$tr \wedge \neg \$wait \wedge \$st' =_u \$st))))$

by (rule cong[of $\mathbf{R}_s \mathbf{R}_s$], simp, rel-auto)

also have $\dots = \mathbf{R}_s ((\neg_r \text{pre}_R P) \text{wp}_r \text{false} \vdash ((\exists \$st' \cdot \text{cmt}_R P) \triangleleft \$wait' \triangleright (\exists \$ref' \cdot \text{cmt}_R P)))$

by (rule cong[of $\mathbf{R}_s \mathbf{R}_s$], simp, rel-auto)

finally show ?thesis .

qed

lemma Skip-right-tri-lemma:

assumes P is CSP

shows $P \;; \text{Skip} = \mathbf{R}_s ((\neg_r \text{pre}_R P) \text{wp}_r \text{false} \vdash ((\exists \$st' \cdot \text{peri}_R P) \diamond (\exists \$ref' \cdot \text{post}_R P)))$

proof –

have $((\exists \$st' \cdot \text{cmt}_R P) \triangleleft \$wait' \triangleright (\exists \$ref' \cdot \text{cmt}_R P)) = ((\exists \$st' \cdot \text{peri}_R P) \diamond (\exists \$ref' \cdot \text{post}_R P))$

by (rel-auto)

thus ?thesis by (simp add: Skip-right-lemma[OF assms])

qed

lemma CSP4-intro:

assumes P is CSP $(\neg_r \text{pre}_R(P)) \;; R1(\text{true}) = (\neg_r \text{pre}_R(P))$

$\$st' \# (\text{cmt}_R P) \llbracket \text{true}/\$wait' \rrbracket \$ref' \# (\text{cmt}_R P) \llbracket \text{false}/\$wait' \rrbracket$

shows P is CSP4

proof –

have $\text{CSP4}(P) = \mathbf{R}_s ((\neg_r \text{pre}_R P) \text{wp}_r \text{false} \vdash ((\exists \$st' \cdot \text{cmt}_R P) \triangleleft \$wait' \triangleright (\exists \$ref' \cdot \text{cmt}_R P)))$

by (simp add: CSP4-def Skip-right-lemma assms(1))

also have $\dots = \mathbf{R}_s (\text{pre}_R(P) \vdash ((\exists \$st' \cdot \text{cmt}_R P) \llbracket \text{true}/\$wait' \rrbracket \triangleleft \$wait' \triangleright (\exists \$ref' \cdot \text{cmt}_R P) \llbracket \text{false}/\$wait' \rrbracket))$

by (simp add: wp-rea-def assms(2) rpred closure cond-var-subst-left cond-var-subst-right)

also have $\dots = \mathbf{R}_s (\text{pre}_R(P) \vdash ((\exists \$st' \cdot (\text{cmt}_R P) \llbracket \text{true}/\$wait' \rrbracket) \triangleleft \$wait' \triangleright (\exists \$ref' \cdot (\text{cmt}_R P) \llbracket \text{false}/\$wait' \rrbracket)))$

by (simp add: usubst unrest)

also have $\dots = \mathbf{R}_s (\text{pre}_R P \vdash ((\text{cmt}_R P) \llbracket \text{true}/\$wait' \rrbracket \triangleleft \$wait' \triangleright (\text{cmt}_R P) \llbracket \text{false}/\$wait' \rrbracket))$

by (simp add: ex-unrest assms)

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

by (simp add: cond-var-split)

also have $\dots = P$

by (simp add: SRD-reactive-design-alt assms(1))

finally show ?thesis

by (simp add: Healthy-def')

qed

lemma CSP4-RC-intro:

assumes P is CSP $pre_R(P)$ is RC
 $\$st' \# (cmt_R P) \llbracket true/\$wait' \rrbracket \$ref' \# (cmt_R P) \llbracket false/\$wait' \rrbracket$
shows P is CSP₄
proof –
have $(\neg_r pre_R(P)) \;; R1(true) = (\neg_r pre_R(P))$
by (metis (no-types, lifting) R1-seqr-closure assms(2) rea-not-R1 rea-not-false rea-not-not wp-rea-RC-false wp-rea-def)
thus ?thesis
by (simp add: CSP₄-intro assms)
qed

lemma CSP₄-rdes:
assumes P is RR Q is RR R is RR
shows $CSP_4(\mathbf{R}_s(P \vdash Q \diamond R)) = \mathbf{R}_s((\neg_r P) \text{ wp}_r \text{ false} \vdash ((\exists \$st' \cdot Q) \diamond (\exists \$ref' \cdot R)))$
by (simp add: CSP₄-def Skip-right-lemma closure assms rdes, rel-auto, blast+)

lemma CSP₄-form:
assumes P is CSP
shows $CSP_4(P) = \mathbf{R}_s((\neg_r pre_R P) \text{ wp}_r \text{ false} \vdash ((\exists \$st' \cdot peri_R P) \diamond (\exists \$ref' \cdot post_R P)))$
by (simp add: CSP₄-def Skip-right-tri-lemma assms)

lemma Skip-srdes-right-unit:
 $(Skip \:: (\sigma, \varphi) \text{ action}) \;; II_R = Skip$
by (rdes-simp)

lemma Skip-srdes-left-unit:
 $II_R \;; (Skip \:: (\sigma, \varphi) \text{ action}) = Skip$
by (rdes-eq)

lemma CSP₄-right-subsumes-RD3: $RD3(CSP_4(P)) = CSP_4(P)$
by (metis (no-types, hide-lams) CSP₄-def RD3-def Skip-srdes-right-unit seqr-assoc)

lemma CSP₄-implies-RD3: $P \text{ is } CSP_4 \implies P \text{ is } RD3$
by (metis CSP₄-right-subsumes-RD3 Healthy-def)

lemma CSP₄-tri-intro:
assumes P is CSP $(\neg_r pre_R(P)) \;; R1(true) = (\neg_r pre_R(P)) \$st' \# peri_R(P) \$ref' \# post_R(P)$
shows P is CSP₄
using assms
by (rule-tac CSP₄-intro, simp-all add: pre_R-def peri_R-def post_R-def usubst cmt_R-def)

lemma CSP₄-NSRD-intro:
assumes P is NSRD $\$ref' \# post_R(P)$
shows P is CSP₄
by (simp add: CSP₄-tri-intro NSRD-is-SRD NSRD-neg-pre-unit NSRD-st'-unrest-peri assms)

lemma CSP₃-commutes-CSP₄: $CSP_3(CSP_4(P)) = CSP_4(CSP_3(P))$
by (simp add: CSP₃-def CSP₄-def seqr-assoc)

lemma NCSP-implies-CSP [closure]: $P \text{ is } NCSP \implies P \text{ is } CSP$
by (metis (no-types, hide-lams) CSP₃-def CSP₄-def Healthy-def NCSP-def SRD-idem SRD-seqr-closure Skip-is-CSP comp-apply)

lemma NCSP-elim [RD-elim]:
 $\llbracket X \text{ is } NCSP; P(\mathbf{R}_s(pre_R(X) \vdash peri_R(X) \diamond post_R(X))) \rrbracket \implies P(X)$

by (*simp add: SRD-reactive-tri-design closure*)

lemma *NCSP-implies-CSP3* [closure]:
 $P \text{ is NCSP} \implies P \text{ is CSP3}$
by (*metis (no-types, lifting) CSP3-def Healthy-def' NCSP-def Skip-is-CSP Skip-left-unit Skip-unrest-ref comp-apply seqr-assoc*)

lemma *NCSP-implies-CSP4* [closure]:
 $P \text{ is NCSP} \implies P \text{ is CSP4}$
by (*metis (no-types, hide-lams) CSP3-commutes-CSP4 Healthy-def NCSP-def NCSP-implies-CSP NCSP-implies-CSP3 comp-apply*)

lemma *NCSP-implies-RD3* [closure]: $P \text{ is NCSP} \implies P \text{ is RD3}$
by (*metis CSP3-commutes-CSP4 CSP4-right-subsumes-RD3 Healthy-def NCSP-def comp-apply*)

lemma *NCSP-implies-NSRD* [closure]: $P \text{ is NCSP} \implies P \text{ is NSRD}$
by (*simp add: NCSP-implies-CSP NCSP-implies-RD3 SRD-RD3-implies-NSRD*)

lemma *NCSP-subset-implies-CSP* [closure]:
 $A \subseteq \llbracket \text{NCSP} \rrbracket_H \implies A \subseteq \llbracket \text{CSP} \rrbracket_H$
using *NCSP-implies-CSP* **by** *blast*

lemma *NCSP-subset-implies-NSRD* [closure]:
 $A \subseteq \llbracket \text{NCSP} \rrbracket_H \implies A \subseteq \llbracket \text{NSRD} \rrbracket_H$
using *NCSP-implies-NSRD* **by** *blast*

lemma *CSP-Healthy-subset-member*: $\llbracket P \in A; A \subseteq \llbracket \text{CSP} \rrbracket_H \rrbracket \implies P \text{ is CSP}$
by (*simp add: is-Healthy-subset-member*)

lemma *CSP3-Healthy-subset-member*: $\llbracket P \in A; A \subseteq \llbracket \text{CSP3} \rrbracket_H \rrbracket \implies P \text{ is CSP3}$
by (*simp add: is-Healthy-subset-member*)

lemma *CSP4-Healthy-subset-member*: $\llbracket P \in A; A \subseteq \llbracket \text{CSP4} \rrbracket_H \rrbracket \implies P \text{ is CSP4}$
by (*simp add: is-Healthy-subset-member*)

lemma *NCSP-Healthy-subset-member*: $\llbracket P \in A; A \subseteq \llbracket \text{NCSP} \rrbracket_H \rrbracket \implies P \text{ is NCSP}$
by (*simp add: is-Healthy-subset-member*)

lemma *NCSP-intro*:
assumes $P \text{ is CSP } P \text{ is CSP3 } P \text{ is CSP4}$
shows $P \text{ is NCSP}$
by (*metis Healthy-def NCSP-def assms comp-eq-dest-lhs*)

lemma *NCSP-NSRD-intro*:
assumes $P \text{ is NSRD } \$\text{ref} \# \text{pre}_R(P) \$\text{ref} \# \text{peri}_R(P) \$\text{ref} \# \text{post}_R(P) \$\text{ref}' \# \text{post}_R(P)$
shows $P \text{ is NCSP}$
by (*simp add: CSP3-SRD-intro CSP4-NSRD-intro NCSP-intro NSRD-is-SRD assms*)

lemma *CSP4-neg-pre-unit*:
assumes $P \text{ is CSP } P \text{ is CSP4}$
shows $(\neg_r \text{pre}_R(P)) ;; R1(\text{true}) = (\neg_r \text{pre}_R(P))$
by (*simp add: CSP4-implies-RD3 NSRD-neg-pre-unit SRD-RD3-implies-NSRD assms(1) assms(2)*)

lemma *NSRD-CSP4-intro*:
assumes $P \text{ is CSP } P \text{ is CSP4}$

shows P is NSRD

by (simp add: CSP4-implies-RD3 SRD-RD3-implies-NSRD assms(1) assms(2))

lemma NCSP-form:

$NCSP\ P = \mathbf{R}_s ((\forall \$ref \cdot (\neg_r\ pre_R(P))\ wp_r\ false) \vdash ((\exists \$ref \cdot \exists \$st' \cdot peri_R(P)) \diamond (\exists \$ref \cdot \exists \$ref' \cdot post_R(P))))$

proof –

have $NCSP\ P = CSP3\ (CSP4\ (NSRD\ P))$

by (metis (no-types, hide-lams) CSP4-def NCSP-def NSRD-alt-def RA1 RD3-def Skip-srdes-left-unit o-apply)

also

have ... = $\mathbf{R}_s ((\forall \$ref \cdot (\neg_r\ pre_R\ (NSRD\ P))\ wp_r\ false) \vdash ((\exists \$ref \cdot \exists \$st' \cdot peri_R\ (NSRD\ P)) \diamond (\exists \$ref \cdot \exists \$ref' \cdot post_R\ (NSRD\ P))))$

by (simp add: CSP3-form CSP4-form closure unrest rdes, rel-auto)

also have ... = $\mathbf{R}_s ((\forall \$ref \cdot (\neg_r\ pre_R(P))\ wp_r\ false) \vdash ((\exists \$ref \cdot \exists \$st' \cdot peri_R(P)) \diamond (\exists \$ref \cdot \exists \$ref' \cdot post_R(P))))$

by (simp add: NSRD-form rdes closure, rel-blast)

finally show ?thesis .

qed

lemma CSP4-st'-unrest-peri [unrest]:

assumes P is CSP P is CSP4

shows $\$st' \not\# peri_R(P)$

by (simp add: NSRD-CSP4-intro NSRD-st'-unrest-peri assms)

lemma CSP4-healthy-form:

assumes P is CSP P is CSP4

shows $P = \mathbf{R}_s((\neg_r\ pre_R\ P)\ wp_r\ false \vdash ((\exists \$st' \cdot peri_R(P)) \diamond (\exists \$ref' \cdot post_R(P))))$

proof –

have $P = \mathbf{R}_s((\neg_r\ pre_R\ P)\ wp_r\ false \vdash ((\exists \$st' \cdot cmt_R\ P) \triangleleft \$wait' \triangleright (\exists \$ref' \cdot cmt_R\ P)))$

by (metis CSP4-def Healthy-def Skip-right-lemma assms(1) assms(2))

also have ... = $\mathbf{R}_s((\neg_r\ pre_R\ P)\ wp_r\ false \vdash ((\exists \$st' \cdot cmt_R\ P) \llbracket true/\$wait' \rrbracket \triangleleft \$wait' \triangleright (\exists \$ref' \cdot cmt_R\ P) \llbracket false/\$wait' \rrbracket))$

by (metis (no-types, hide-lams) subst-wait'-left-subst subst-wait'-right-subst wait'-cond-def)

also have ... = $\mathbf{R}_s((\neg_r\ pre_R\ P)\ wp_r\ false \vdash ((\exists \$st' \cdot peri_R(P)) \diamond (\exists \$ref' \cdot post_R(P))))$

by (simp add: wait'-cond-def usubst peri_R-def post_R-def cmt_R-def unrest)

finally show ?thesis .

qed

lemma CSP4-ref'-unrest-pre [unrest]:

assumes P is CSP P is CSP4

shows $\$ref' \not\# pre_R(P)$

proof –

have $pre_R(P) = pre_R(\mathbf{R}_s((\neg_r\ pre_R\ P)\ wp_r\ false \vdash ((\exists \$st' \cdot peri_R(P)) \diamond (\exists \$ref' \cdot post_R(P))))$

using CSP4-healthy-form assms(1) assms(2) by fastforce

also have ... = $(\neg_r\ pre_R\ P)\ wp_r\ false$

by (simp add: rea-pre-RHS-design wp-rea-def usubst unrest

CSP4-neg-pre-unit R1-rea-not R2c-preR R2c-rea-not assms)

also have $\$ref' \not\# \dots$

by (simp add: wp-rea-def unrest)

finally show ?thesis .

qed

lemma NCSP-set-unrest-pre-wait':

assumes $A \subseteq \llbracket NCSP \rrbracket_H$
shows $\bigwedge P. P \in A \implies \$wait' \# pre_R(P)$
proof –
fix P
assume $P \in A$
hence P is NSRD
using *NCSP-implies-NSRD* *assms* **by** *auto*
thus $\$wait' \# pre_R(P)$
using *NSRD-wait'-unrest-pre* **by** *blast*
qed

lemma *CSP4-set-unrest-pre-st'*:
assumes $A \subseteq \llbracket CSP \rrbracket_H$ $A \subseteq \llbracket CSP4 \rrbracket_H$
shows $\bigwedge P. P \in A \implies \$st' \# pre_R(P)$
proof –
fix P
assume $P \in A$
hence P is NSRD
using *NSRD-CSP4-intro* *assms(1)* *assms(2)* **by** *blast*
thus $\$st' \# pre_R(P)$
using *NSRD-st'-unrest-pre* **by** *blast*
qed

lemma *CSP4-ref'-unrest-post* [*unrest*]:
assumes P is CSP P is CSP4
shows $\$ref' \# post_R(P)$
proof –
have $post_R(P) = post_R(\mathbf{R}_s((\neg_r pre_R P) wp_r false \vdash ((\exists \$st' \cdot peri_R(P)) \diamond (\exists \$ref' \cdot post_R(P)))))$
using *CSP4-healthy-form* *assms(1)* *assms(2)* **by** *fastforce*
also have $\dots = R1 (R2c ((\neg_r pre_R P) wp_r false \Rightarrow_r (\exists \$ref' \cdot post_R(P)))$
by (*simp add: rea-post-RHS-design* *usubst unrest wp-rea-def*)
also have $\$ref' \# \dots$
by (*simp add: R1-def R2c-def wp-rea-def unrest*)
finally show *?thesis* .
qed

lemma *CSP3-Chaos* [*closure*]: *Chaos* is CSP3
by (*simp add: Chaos-def, rule CSP3-intro, simp-all add: RHS-design-is-SRD unrest*)

lemma *CSP4-Chaos* [*closure*]: *Chaos* is CSP4
by (*rule CSP4-tri-intro, simp-all add: closure rdes unrest*)

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

lemma *CSP3-Miracle* [*closure*]: *Miracle* is CSP3
by (*simp add: Miracle-def, rule CSP3-intro, simp-all add: RHS-design-is-SRD unrest*)

lemma *CSP4-Miracle* [*closure*]: *Miracle* is CSP4
by (*rule CSP4-tri-intro, simp-all add: closure rdes unrest*)

lemma *NCSP-Miracle* [*closure*]: *Miracle* is NCSP
by (*simp add: NCSP-intro closure*)

lemma *NCSP-seqr-closure* [*closure*]:

assumes P is NCSP Q is NCSP
shows $P \leadsto Q$ is NCSP
by (*metis* (*no-types*, *lifting*) *CSP3-def CSP4-def Healthy-def' NCSP-implies-CSP NCSP-implies-CSP3*
NCSP-implies-CSP4 NCSP-intro SRD-seqr-closure assms(1) assms(2) seqr-assoc)

lemma *CSP4-Skip* [*closure*]: *Skip* is *CSP4*
apply (*rule CSP4-intro, simp-all add: Skip-is-CSP*)
apply (*simp-all add: Skip-def rea-pre-RHS-design rea-cmt-RHS-design usubst unrest R2c-true*)
done

lemma *NCSP-Skip* [*closure*]: *Skip* is NCSP
by (*metis CSP3-Skip CSP4-Skip Healthy-def NCSP-def Skip-is-CSP comp-apply*)

lemma *CSP4-Stop* [*closure*]: *Stop* is *CSP4*
apply (*rule CSP4-intro, simp-all add: Stop-is-CSP*)
apply (*simp-all add: Stop-def rea-pre-RHS-design rea-cmt-RHS-design usubst unrest R2c-true*)
done

lemma *NCSP-Stop* [*closure*]: *Stop* is NCSP
by (*metis CSP3-Stop CSP4-Stop Healthy-def NCSP-def Stop-is-CSP comp-apply*)

lemma *CSP4-Idempotent*: *Idempotent CSP4*
by (*metis* (*no-types*, *lifting*) *CSP3-Skip CSP3-def CSP4-def Healthy-if Idempotent-def seqr-assoc*)

lemma *CSP4-Continuous*: *Continuous CSP4*
by (*simp add: Continuous-def CSP4-def seq-Sup-distr*)

lemma *preR-Stop* [*rdes*]: $\text{pre}_R(\text{Stop}) = \text{true}_r$
by (*simp add: Stop-def Stop-is-CSP rea-pre-RHS-design unrest usubst R2c-true*)

lemma *periR-Stop* [*rdes*]: $\text{peri}_R(\text{Stop}) = \mathcal{E}(\text{true}, \langle \rangle, \{\}_u)$
by (*rel-auto*)

lemma *postR-Stop* [*rdes*]: $\text{post}_R(\text{Stop}) = \text{false}$
by (*rel-auto*)

lemma *cmtR-Stop* [*rdes*]: $\text{cmt}_R(\text{Stop}) = (\$tr' =_u \$tr \wedge \$wait')$
by (*rel-auto*)

lemma *NCSP-Idempotent* [*closure*]: *Idempotent NCSP*
by (*clarsimp simp add: NCSP-def Idempotent-def*)
(metis (*no-types*, *hide-lams*) *CSP3-Idempotent CSP3-def CSP4-Idempotent CSP4-def Healthy-def*
Idempotent-def SRD-idem SRD-seqr-closure Skip-is-CSP seqr-assoc)

lemma *NCSP-Continuous* [*closure*]: *Continuous NCSP*
by (*simp add: CSP3-Continuous CSP4-Continuous Continuous-comp NCSP-def SRD-Continuous*)

lemma *preR-CRR* [*closure*]: P is NCSP $\implies \text{pre}_R(P)$ is CRR
by (*rule CRR-intro, simp-all add: closure unrest*)

lemma *periR-CRR* [*closure*]: P is NCSP $\implies \text{peri}_R(P)$ is CRR
by (*rule CRR-intro, simp-all add: closure unrest*)

lemma *postR-CRR* [*closure*]: P is NCSP $\implies \text{post}_R(P)$ is CRR
by (*rule CRR-intro, simp-all add: closure unrest*)

lemma *NCSP-rdes-intro*:
assumes P is CRC Q is CRR R is CRR
 $\$st' \# Q \$ref' \# R$
shows $\mathbf{R}_s(P \vdash Q \diamond R)$ is NCSP
apply (rule NCSP-intro)
apply (simp-all add: closure assms)
apply (rule CSP3-SRD-intro)
apply (simp-all add: rdes closure assms unrest)
apply (rule CSP4-tri-intro)
apply (simp-all add: rdes closure assms unrest)
apply (metis (no-types, lifting) CRC-implies-RC R1-seqr-closure assms(1) rea-not-R1 rea-not-false
 rea-not-not wp-rea-RC-false wp-rea-def)
done

lemma *NCSP-preR-CRC [closure]*:
assumes P is NCSP
shows $\text{pre}_R(P)$ is CRC
by (rule CRC-intro, simp-all add: closure assms unrest)

lemma *CSP3-Sup-closure [closure]*:
 $A \subseteq \llbracket \text{CSP3} \rrbracket_H \implies (\bigwedge A)$ is CSP3
apply (auto simp add: CSP3-def Healthy-def seq-Sup-distl)
apply (rule cong[of Sup])
apply (simp)
using image-iff **apply** force
done

lemma *CSP4-Sup-closure [closure]*:
 $A \subseteq \llbracket \text{CSP4} \rrbracket_H \implies (\bigwedge A)$ is CSP4
apply (auto simp add: CSP4-def Healthy-def seq-Sup-distr)
apply (rule cong[of Sup])
apply (simp)
using image-iff **apply** force
done

lemma *NCSP-Sup-closure [closure]*: $\llbracket A \subseteq \llbracket \text{NCSP} \rrbracket_H; A \neq \{\} \rrbracket \implies (\bigwedge A)$ is NCSP
apply (rule NCSP-intro, simp-all add: closure)
apply (metis (no-types, lifting) Ball-Collect CSP3-Sup-closure NCSP-implies-CSP3)
apply (metis (no-types, lifting) Ball-Collect CSP4-Sup-closure NCSP-implies-CSP4)
done

lemma *NCSP-SUP-closure [closure]*: $\llbracket \bigwedge i. P(i) \text{ is NCSP}; A \neq \{\} \rrbracket \implies (\bigwedge i \in A. P(i))$ is NCSP
by (metis (mono-tags, lifting) Ball-Collect NCSP-Sup-closure image-iff image-is-empty)

lemma *PCSP-implies-NCSP [closure]*:
assumes P is PCSP
shows P is NCSP

proof –

have $P = \text{Productive}(\text{NCSP}(\text{NCSP } P))$
by (metis (no-types, hide-lams) Healthy-def' Idempotent-def NCSP-Idempotent assms comp-apply)

also have $\dots = \mathbf{R}_s((\forall \$ref \cdot (\neg_r \text{pre}_R(\text{NCSP } P)) \text{ wp}_r \text{ false}) \vdash$
 $(\exists \$ref \cdot \exists \$st' \cdot \text{peri}_R(\text{NCSP } P)) \diamond$
 $((\exists \$ref \cdot \exists \$ref' \cdot \text{post}_R(\text{NCSP } P)) \wedge \$tr <_u \$tr'))$

by (simp add: NCSP-form Productive-RHS-design-form unrest closure)
 also have ... is NCSP
 apply (rule NCSP-rdes-intro)
 apply (rule CRC-intro)
 apply (simp-all add: unrest ex-unrest all-unrest closure)
 done
 finally show ?thesis .
 qed

lemma PCSP-elim [RD-elim]:
 assumes X is PCSP P ($\mathbf{R}_s ((pre_R X) \vdash peri_R X \diamond (post_R X \wedge \$tr <_u \$tr'))$)
 shows $P X$
 by (metis Healthy-if NCSP-implies-CSP PCSP-implies-NCSP Productive-form assms comp-apply)

lemma ICSP-implies-NCSP [closure]:
 assumes P is ICSP
 shows P is NCSP
proof –
 have $P = ISRD1(NCSP(NCSP P))$
 by (metis (no-types, hide-lams) Healthy-def' Idempotent-def NCSP-Idempotent assms comp-apply)
 also have ... = $ISRD1 (\mathbf{R}_s ((\forall \$ref \cdot (\neg_r pre_R (NCSP P)) wp_r false) \vdash$
 $(\exists \$ref \cdot \exists \$st' \cdot peri_R (NCSP P)) \diamond$
 $(\exists \$ref \cdot \exists \$ref' \cdot post_R (NCSP P))))$
 by (simp add: NCSP-form)
 also have ... = $\mathbf{R}_s ((\forall \$ref \cdot (\neg_r pre_R (NCSP P)) wp_r false) \vdash$
 $false \diamond$
 $((\exists \$ref \cdot \exists \$ref' \cdot post_R (NCSP P)) \wedge \$tr' =_u \$tr))$
 by (simp-all add: ISRD1-RHS-design-form closure rdes unrest)
 also have ... is NCSP
 apply (rule NCSP-rdes-intro)
 apply (rule CRC-intro)
 apply (simp-all add: unrest ex-unrest all-unrest closure)
 done
 finally show ?thesis .
 qed

lemma ICSP-implies-ISRD [closure]:
 assumes P is ICSP
 shows P is ISRD
 by (metis (no-types, hide-lams) Healthy-def ICSP-implies-NCSP ISRD-def NCSP-implies-ISRD assms comp-apply)

lemma ICSP-elim [RD-elim]:
 assumes X is ICSP P ($\mathbf{R}_s ((pre_R X) \vdash false \diamond (post_R X \wedge \$tr' =_u \$tr))$)
 shows $P X$
 by (metis Healthy-if NCSP-implies-CSP ICSP-implies-NCSP ISRD1-form assms comp-apply)

lemma ICSP-Stop-right-zero-lemma:
 $(P \wedge (\$tr' =_u \$tr)) ;; true_r = true_r \implies (P \wedge (\$tr' =_u \$tr)) ;; (\$tr' =_u \$tr) = (\$tr' =_u \$tr)$
 by (rel-blast)

lemma ICSP-Stop-right-zero:
 assumes P is ICSP $pre_R(P) = true_r post_R(P) ;; true_r = true_r$
 shows $P ;; Stop = Stop$
proof –

from *assms*(3) **have** 1: ($\text{post}_R P \wedge \$tr' =_u \tr) ;; $\text{true}_r = \text{true}_r$
by (*rel-auto*, *metis* (*full-types*, *hide-lams*) *dual-order.antisym order-refl*)
show ?thesis
by (*rdes-simp cls: assms*(1), *simp add: csp-enable-nothing assms*(2) *ICSP-Stop-right-zero-lemma*[OF 1])
qed

lemma *ICSP-intro*: $\llbracket P \text{ is NCSP}; P \text{ is ISRD1} \rrbracket \implies P \text{ is ICSP}$
using *Healthy-comp* **by** *blast*

lemma *seq-ICSP-closed* [*closure*]:
assumes $P \text{ is ICSP } Q \text{ is ICSP}$
shows $P ;; Q \text{ is ICSP}$
by (*meson ICSP-implies-ISRD ICSP-implies-NCSP ICSP-intro ISRD-implies-ISRD1 NCSP-seqr-closure assms seq-ISRD-closed*)

lemma *Miracle-ICSP* [*closure*]: *Miracle is ICSP*
by (*rule ICSP-intro*, *simp add: closure*, *simp add: ISRD1-rdes-intro rdes-def closure*)

5.2 CSP theories

typeddecl *TCSP*

abbreviation $TCSP \equiv UTHY(TCSP, ('\sigma, '\varphi) \text{ st-csp})$

overloading

tcspl-hcond == *utpl-hcond* :: $(TCSP, ('\sigma, '\varphi) \text{ st-csp}) \text{ uthy} \Rightarrow (('\sigma, '\varphi) \text{ st-csp} \times (''\sigma, ''\varphi) \text{ st-csp}) \text{ health}$

begin

definition *tcspl-hcond* :: $(TCSP, ('\sigma, '\varphi) \text{ st-csp}) \text{ uthy} \Rightarrow (('\sigma, '\varphi) \text{ st-csp} \times (''\sigma, ''\varphi) \text{ st-csp}) \text{ health}$ **where**
[*upred-defs*]: *tcspl-hcond* $T = NCSP$

end

interpretation *csp-theory*: *utpl-theory-continuous* $UTHY(TCSP, ('\sigma, '\varphi) \text{ st-csp})$

rewrites $\bigwedge P. P \in \text{carrier (uthy-order } TCSP) \longleftrightarrow P \text{ is NCSP}$

and $P \text{ is } \mathcal{H}_{TCSP} \longleftrightarrow P \text{ is NCSP}$

and $\text{carrier (uthy-order } TCSP) \rightarrow \text{carrier (uthy-order } TCSP) \equiv \llbracket NCSP \rrbracket_H \rightarrow \llbracket NCSP \rrbracket_H$

and $A \subseteq \text{carrier (uthy-order } TCSP) \longleftrightarrow A \subseteq \llbracket NCSP \rrbracket_H$

and $\text{le (uthy-order } TCSP) = \text{op} \sqsubseteq$

by (*unfold-locales*, *simp-all add: tcspl-hcond-def NCSP-Continuous Healthy-Idempotent Healthy-if NCSP-Idempotent*)

declare *csp-theory.top-healthy* [*simp del*]

declare *csp-theory.bottom-healthy* [*simp del*]

lemma *csp-bottom-Chaos*: $\perp_{TCSP} = \text{Chaos}$

proof –

have 1: $\perp_{TCSP} = \text{CSP3 (CSP4 (CSP true))}$

by (*simp add: csp-theory.healthy-bottom*, *simp add: tcspl-hcond-def NCSP-def*)

also have 2: $\dots = \text{CSP3 (CSP4 Chaos)}$

by (*metis srdes-hcond-def srdes-theory-continuous.healthy-bottom*)

also have 3: $\dots = \text{Chaos}$

by (*metis CSP3-Chaos CSP4-Chaos Healthy-def*)

finally show ?thesis .

qed

lemma *csp-top-Miracle*: $\top_{TCSP} = \text{Miracle}$

proof –

```

have 1:  $\top_{TCSP} = CSP3 \ (CSP4 \ (CSP \ false))$ 
  by (simp add: csp-theory.healthy-top, simp add: tcsp-hcond-def NCSP-def)
also have 2:... =  $CSP3 \ (CSP4 \ Miracle)$ 
  by (metis srdes-hcond-def srdes-theory-continuous.healthy-top)
also have 3:... =  $Miracle$ 
  by (metis CSP3-Miracle CSP4-Miracle Healthy-def)
finally show ?thesis .
qed

```

5.3 Algebraic laws

```

lemma Stop-left-zero:
  assumes  $P \text{ is } CSP$ 
  shows  $Stop \ ;\; P = Stop$ 
  by (simp add: NSRD-seq-post-false assms NCSP-implies-NSRD NCSP-Stop postR-Stop)

end

```

6 Reactive Contracts for CSP/Circus with refusals

```

theory utp-circus-contracts
  imports utp-circus-healths
begin

```

definition $mk\text{-}CRD :: 's \text{ upred} \Rightarrow ('e \text{ list} \Rightarrow 'e \text{ set} \Rightarrow 's \text{ upred}) \Rightarrow ('e \text{ list} \Rightarrow 's \text{ hrel}) \Rightarrow ('s, 'e) \text{ action}$
where

$mk\text{-}CRD \ P \ Q \ R = \mathbf{R}_s([P]_{S<} \vdash [Q \ x \ r]_{S<} \llbracket x \rightarrow \&tt \rrbracket \llbracket r \rightarrow \$ref \ ' \rrbracket \diamond [R(x)]_S \llbracket x \rightarrow \&tt \rrbracket)$

syntax

-ref-var :: *logic*
-mk-CRD :: *logic* \Rightarrow *logic* \Rightarrow *logic* \Rightarrow *logic* $([-/\vdash -/ \mid -]_C)$

parse-translation \ll

```

let
  fun ref-var-tr [] = Syntax.free refs
    | ref-var-tr - = raise Match;
in
  [(@{syntax-const -ref-var}, K ref-var-tr)]
end
 $\gg$ 

```

translations

$[P \vdash Q \mid R]_C \Rightarrow CONST \ mk\text{-}CRD \ P \ (\lambda \text{-trace-var} \text{-ref-var}. Q) \ (\lambda \text{-trace-var}. R)$
 $[P \vdash Q \mid R]_C \Leftarrow CONST \ mk\text{-}CRD \ P \ (\lambda x \ r. Q) \ (\lambda y. R)$

lemma $CSP\text{-}mk\text{-}CRD \ [closure]: [P \vdash Q \text{ trace refs} \mid R(\text{trace})]_C \text{ is } CSP$
 by (simp add: mk-CRD-def closure unrest)

lemma $preR\text{-}mk\text{-}CRD \ [rdes]: pre_R([P \vdash Q \text{ trace refs} \mid R(\text{trace})]_C) = [P]_{S<}$
 by (simp add: mk-CRD-def rea-pre-RHS-design usubst unrest R2c-not R2c-lift-state-pre rea-st-cond-def, rel-auto)

lemma $periR\text{-}mk\text{-}CRD \ [rdes]: peri_R([P \vdash Q \text{ trace refs} \mid R(\text{trace})]_C) = ([P]_{S<} \Rightarrow_r ([Q \text{ trace refs}]_{S<} \llbracket (\text{trace}, \text{refs}) \rightarrow (\&tt, \$ref) \rrbracket))$
 by (simp add: mk-CRD-def rea-peri-RHS-design usubst unrest R2c-not R2c-lift-state-pre
 impl-alt-def R2c-disj R2c-msubst-tt R1-disj, rel-auto)

lemma *postR-mk-CRD* [*rdes*]: $\text{post}_R([P \vdash Q \text{ trace refs} \mid R(\text{trace})]_C) = ([P]_{S<} \Rightarrow_r ([R(\text{trace})]_S) \llbracket \text{trace} \rightarrow \&tt \rrbracket)$
by (*simp add: mk-CRD-def rea-post-RHS-design usubst unrest R2c-not R2c-lift-state-pre*
impl-alt-def R2c-disj R2c-msubst-tt R1-disj, rel-auto)

Refinement introduction law for contracts

lemma *CRD-contract-refine*:

assumes
 $Q \text{ is CSP } '[P_1]_{S<} \Rightarrow \text{pre}_R Q'$
 $'[P_1]_{S<} \wedge \text{peri}_R Q \Rightarrow [P_2 \ t \ r]_{S<} \llbracket t \rightarrow \&tt \rrbracket \llbracket r \rightarrow \$ref' \rrbracket'$
 $'[P_1]_{S<} \wedge \text{post}_R Q \Rightarrow [P_3 \ x]_S \llbracket x \rightarrow \&tt \rrbracket'$
shows $[P_1 \vdash P_2 \text{ trace refs} \mid P_3(\text{trace})]_C \sqsubseteq Q$
proof –
have $[P_1 \vdash P_2 \text{ trace refs} \mid P_3(\text{trace})]_C \sqsubseteq \mathbf{R}_s(\text{pre}_R(Q) \vdash \text{peri}_R(Q) \diamond \text{post}_R(Q))$
using *assms by (simp add: mk-CRD-def, rule-tac sdes-tri-refine-intro, rel-auto+)*
thus *?thesis*
by (*simp add: SRD-reactive-tri-design assms(1)*)
qed

lemma *CRD-contract-refine'*:

assumes
 $Q \text{ is CSP } '[P_1]_{S<} \Rightarrow \text{pre}_R Q'$
 $[P_2 \ t \ r]_{S<} \llbracket t \rightarrow \&tt \rrbracket \llbracket r \rightarrow \$ref' \rrbracket \sqsubseteq ([P_1]_{S<} \wedge \text{peri}_R Q)$
 $[P_3 \ x]_S \llbracket x \rightarrow \&tt \rrbracket \sqsubseteq ([P_1]_{S<} \wedge \text{post}_R Q)$
shows $[P_1 \vdash P_2 \text{ trace refs} \mid P_3(\text{trace})]_C \sqsubseteq Q$
using *assms by (rule-tac CRD-contract-refine, simp-all add: refBy-order)*

lemma *CRD-refine-CRD*:

assumes
 $'[P_1]_{S<} \Rightarrow ([Q_1]_{S<} :: ('e, 's) \text{ action})'$
 $([P_2 \ x \ r]_{S<} \llbracket x \rightarrow \&tt \rrbracket \llbracket r \rightarrow \$ref' \rrbracket) \sqsubseteq ([P_1]_{S<} \wedge [Q_2 \ x \ r]_{S<} \llbracket x \rightarrow \&tt \rrbracket \llbracket r \rightarrow \$ref' \rrbracket :: ('e, 's) \text{ action})$
 $[P_3 \ x]_S \llbracket x \rightarrow \&tt \rrbracket \sqsubseteq ([P_1]_{S<} \wedge [Q_3 \ x]_S \llbracket x \rightarrow \&tt \rrbracket :: ('e, 's) \text{ action})$
shows $([P_1 \vdash P_2 \text{ trace refs} \mid P_3 \text{ trace}]_C :: ('e, 's) \text{ action}) \sqsubseteq [Q_1 \vdash Q_2 \text{ trace refs} \mid Q_3 \text{ trace}]_C$
using *assms*
by (*simp add: mk-CRD-def, rule-tac sdes-tri-refine-intro, rel-auto+*)

lemma *CRD-refine-rdes*:

assumes
 $'[P_1]_{S<} \Rightarrow Q_1'$
 $([P_2 \ x \ r]_{S<} \llbracket x \rightarrow \&tt \rrbracket \llbracket r \rightarrow \$ref' \rrbracket) \sqsubseteq ([P_1]_{S<} \wedge Q_2)$
 $[P_3 \ x]_S \llbracket x \rightarrow \&tt \rrbracket \sqsubseteq ([P_1]_{S<} \wedge Q_3)$
shows $([P_1 \vdash P_2 \text{ trace refs} \mid P_3 \text{ trace}]_C :: ('e, 's) \text{ action}) \sqsubseteq \mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3)$
using *assms*
by (*simp add: mk-CRD-def, rule-tac sdes-tri-refine-intro, rel-auto+*)

end

7 External Choice

theory *utp-circus-extchoice*

imports

utp-circus-healths

utp-circus-rel

begin

7.1 Definitions and syntax

definition *ExtChoice* ::

(σ, φ) action set $\Rightarrow (\sigma, \varphi)$ action **where**
 $[upred-defs]: ExtChoice\ A = \mathbf{R}_s((\bigsqcup P \in A \cdot pre_R(P)) \vdash ((\bigsqcup P \in A \cdot cmt_R(P)) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\bigsqcup P \in A \cdot cmt_R(P))))$

syntax

$-ExtChoice :: pttrn \Rightarrow 'a\ set \Rightarrow 'b \Rightarrow 'b\ ((\exists \square \square \cdot - / -) [0, 0, 10] 10)$
 $-ExtChoice-simp :: pttrn \Rightarrow 'b \Rightarrow 'b\ ((\exists \square \square \cdot - / -) [0, 10] 10)$

translations

$\square P \in A \cdot B \Rightarrow CONST\ ExtChoice\ ((\lambda P. B) \cdot A)$
 $\square P \cdot B \Rightarrow CONST\ ExtChoice\ (CONST\ range\ (\lambda P. B))$

definition *extChoice* ::

(σ, φ) action $\Rightarrow (\sigma, \varphi)$ action $\Rightarrow (\sigma, \varphi)$ action (**infixl** \square 65) **where**
 $[upred-defs]: P \square Q \equiv ExtChoice\ \{P, Q\}$

Small external choice as an indexed big external choice.

lemma *extChoice-alt-def*:

$P \square Q = (\square i :: nat \in \{0, 1\} \cdot P \triangleleft \ll i = 0 \gg \triangleright Q)$
by (*simp add: extChoice-def ExtChoice-def, unliteralise, simp*)

7.2 Basic laws

7.3 Algebraic laws

lemma *ExtChoice-empty*: $ExtChoice\ \{\} = Stop$

by (*simp add: ExtChoice-def cond-def Stop-def*)

lemma *ExtChoice-single*:

$P\ is\ CSP \Rightarrow ExtChoice\ \{P\} = P$
by (*simp add: ExtChoice-def usup-and uinf-or SRD-reactive-design-alt*)

7.4 Reactive design calculations

lemma *ExtChoice-rdes*:

assumes $\bigwedge i. \$ok' \nmid P(i) \ A \neq \{\}$
shows $(\square i \in A \cdot \mathbf{R}_s(P(i) \vdash Q(i))) = \mathbf{R}_s((\bigsqcup i \in A \cdot P(i)) \vdash ((\bigsqcup i \in A \cdot Q(i)) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\bigsqcup i \in A \cdot Q(i))))$

proof –

have $(\square i \in A \cdot \mathbf{R}_s(P(i) \vdash Q(i))) =$
 $\mathbf{R}_s((\bigsqcup i \in A \cdot pre_R(\mathbf{R}_s(P\ i \vdash Q\ i))) \vdash$
 $((\bigsqcup i \in A \cdot cmt_R(\mathbf{R}_s(P\ i \vdash Q\ i)))$
 $\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
 $(\bigsqcup i \in A \cdot cmt_R(\mathbf{R}_s(P\ i \vdash Q\ i))))$
by (*simp add: ExtChoice-def*)
also have ... =
 $\mathbf{R}_s((\bigsqcup i \in A \cdot R1(R2c(pre_s \dagger P(i)))) \vdash$
 $((\bigsqcup i \in A \cdot R1(R2c(cmt_s \dagger (P(i) \Rightarrow Q(i))))$
 $\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
 $(\bigsqcup i \in A \cdot R1(R2c(cmt_s \dagger (P(i) \Rightarrow Q(i))))))$
by (*simp add: rea-pre-RHS-design rea-cmt-RHS-design*)
also have ... =
 $\mathbf{R}_s((\bigsqcup i \in A \cdot R1(R2c(pre_s \dagger P(i)))) \vdash$

$R1(R2c$
 $((\sqcup i \in A \cdot R1(R2c(cmt_s \dagger (P(i) \Rightarrow Q(i))))$
 $\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
 $(\sqcap i \in A \cdot R1(R2c(cmt_s \dagger (P(i) \Rightarrow Q(i))))))$
by (*metis (no-types, lifting) RHS-design-export-R1 RHS-design-export-R2c*)
also have ... =
 $R_s ((\sqcup i \in A \cdot R1(R2c(pre_s \dagger P(i)))) \vdash$
 $R1(R2c$
 $((\sqcup i \in A \cdot (cmt_s \dagger (P(i) \Rightarrow Q(i))))$
 $\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
 $(\sqcap i \in A \cdot (cmt_s \dagger (P(i) \Rightarrow Q(i))))$
by (*simp add: R2c-UNIF R2c-condr R1-cond R1-idem R1-R2c-commute R2c-idem R1-UNIF assms*
R1-USUP R2c-USUP)
also have ... =
 $R_s ((\sqcup i \in A \cdot R1(R2c(pre_s \dagger P(i)))) \vdash$
 $cmt_s \dagger$
 $((\sqcup i \in A \cdot (cmt_s \dagger (P(i) \Rightarrow Q(i))))$
 $\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
 $(\sqcap i \in A \cdot (cmt_s \dagger (P(i) \Rightarrow Q(i))))$
by (*metis (no-types, lifting) RHS-design-export-R1 RHS-design-export-R2c rdes-export-cmt*)
also have ... =
 $R_s ((\sqcup i \in A \cdot R1(R2c(pre_s \dagger P(i)))) \vdash$
 $cmt_s \dagger$
 $((\sqcup i \in A \cdot (P(i) \Rightarrow Q(i))))$
 $\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
 $(\sqcap i \in A \cdot (P(i) \Rightarrow Q(i))))$
by (*simp add: usubst*)
also have ... =
 $R_s ((\sqcup i \in A \cdot R1(R2c(pre_s \dagger P(i)))) \vdash$
 $((\sqcup i \in A \cdot (P(i) \Rightarrow Q(i))) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\sqcap i \in A \cdot (P(i) \Rightarrow Q(i))))$
by (*simp add: rdes-export-cmt*)
also have ... =
 $R_s ((R1(R2c(\sqcup i \in A \cdot (pre_s \dagger P(i)))) \vdash$
 $((\sqcup i \in A \cdot (P(i) \Rightarrow Q(i))) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\sqcap i \in A \cdot (P(i) \Rightarrow Q(i))))$
by (*simp add: not-UNIF R1-UNIF R2c-UNIF assms*)
also have ... =
 $R_s ((R2c(\sqcup i \in A \cdot (pre_s \dagger P(i)))) \vdash$
 $((\sqcup i \in A \cdot (P(i) \Rightarrow Q(i))) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\sqcap i \in A \cdot (P(i) \Rightarrow Q(i))))$
by (*simp add: R1-design-R1-pre*)
also have ... =
 $R_s (((\sqcup i \in A \cdot (pre_s \dagger P(i)))) \vdash$
 $((\sqcup i \in A \cdot (P(i) \Rightarrow Q(i))) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\sqcap i \in A \cdot (P(i) \Rightarrow Q(i))))$
by (*metis (no-types, lifting) RHS-design-R2c-pre*)
also have ... =
 $R_s ([\$ok \mapsto_s true, \$wait \mapsto_s false] \dagger (\sqcup i \in A \cdot P(i)) \vdash$
 $((\sqcup i \in A \cdot (P(i) \Rightarrow Q(i))) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\sqcap i \in A \cdot (P(i) \Rightarrow Q(i))))$
proof –
from *assms* **have** $\bigwedge i. pre_s \dagger P(i) = [\$ok \mapsto_s true, \$wait \mapsto_s false] \dagger P(i)$
by (*rel-auto*)
thus *?thesis*
by (*simp add: usubst*)
qed
also have ... =
 $R_s ((\sqcup i \in A \cdot P(i)) \vdash ((\sqcup i \in A \cdot (P(i) \Rightarrow Q(i))) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\sqcap i \in A \cdot (P(i) \Rightarrow$
 $Q(i))))$

by (simp add: rdes-export-pre not-UINF)
also have ... = $\mathbf{R}_s ((\sqcup i \in A \cdot P(i)) \vdash ((\sqcup i \in A \cdot Q(i)) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\sqcap i \in A \cdot Q(i))))$
by (rule cong[of $\mathbf{R}_s \mathbf{R}_s$], simp, rel-auto, blast+)

finally show ?thesis .

qed

lemma ExtChoice-tri-rdes [rdes-def]:

assumes $\bigwedge i . \$ok' \# P_1(i) \ A \neq \{\}$

shows $(\sqcap i \in A \cdot \mathbf{R}_s(P_1(i) \vdash P_2(i) \diamond P_3(i))) =$

$\mathbf{R}_s ((\sqcup i \in A \cdot P_1(i)) \vdash (((\sqcup i \in A \cdot P_2(i)) \triangleleft \$tr' =_u \$tr \triangleright (\sqcap i \in A \cdot P_2(i))) \diamond (\sqcap i \in A \cdot P_3(i))))$

proof –

have $(\sqcap i \in A \cdot \mathbf{R}_s(P_1(i) \vdash P_2(i) \diamond P_3(i))) =$

$\mathbf{R}_s ((\sqcup i \in A \cdot P_1(i)) \vdash (((\sqcup i \in A \cdot P_2(i) \diamond P_3(i)) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\sqcap i \in A \cdot P_2(i) \diamond P_3(i))))$

by (simp add: ExtChoice-rdes assms)

also

have ... =

$\mathbf{R}_s ((\sqcup i \in A \cdot P_1(i)) \vdash (((\sqcup i \in A \cdot P_2(i) \diamond P_3(i)) \triangleleft \$wait' \wedge \$tr' =_u \$tr \triangleright (\sqcap i \in A \cdot P_2(i) \diamond P_3(i))))$

by (simp add: conj-comm)

also

have ... =

$\mathbf{R}_s ((\sqcup i \in A \cdot P_1(i)) \vdash (((\sqcup i \in A \cdot P_2(i) \diamond P_3(i)) \triangleleft \$tr' =_u \$tr \triangleright (\sqcap i \in A \cdot P_2(i) \diamond P_3(i))) \diamond (\sqcap i \in A \cdot P_2(i) \diamond P_3(i))))$

by (simp add: cond-conj wait'-cond-def)

also

have ... = $\mathbf{R}_s ((\sqcup i \in A \cdot P_1(i)) \vdash (((\sqcup i \in A \cdot P_2(i)) \triangleleft \$tr' =_u \$tr \triangleright (\sqcap i \in A \cdot P_2(i))) \diamond (\sqcap i \in A \cdot P_3(i))))$

by (rule cong[of $\mathbf{R}_s \mathbf{R}_s$], simp, rel-auto)

finally show ?thesis .

qed

lemma ExtChoice-tri-rdes-def [rdes-def]:

assumes $A \subseteq \llbracket CSP \rrbracket_H$

shows $ExtChoice\ A = \mathbf{R}_s ((\sqcup P \in A \cdot pre_R\ P) \vdash (((\sqcup P \in A \cdot peri_R\ P) \triangleleft \$tr' =_u \$tr \triangleright (\sqcap P \in A \cdot peri_R\ P)) \diamond (\sqcap P \in A \cdot post_R\ P)))$

proof –

have $((\sqcup P \in A \cdot cmt_R\ P) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\sqcap P \in A \cdot cmt_R\ P)) =$

$((\sqcup P \in A \cdot cmt_R\ P) \triangleleft \$tr' =_u \$tr \triangleright (\sqcap P \in A \cdot cmt_R\ P)) \diamond (\sqcap P \in A \cdot cmt_R\ P)$

by (rel-auto)

also have ... = $((\sqcup P \in A \cdot peri_R\ P) \triangleleft \$tr' =_u \$tr \triangleright (\sqcap P \in A \cdot peri_R\ P)) \diamond (\sqcap P \in A \cdot post_R\ P)$

by (rel-auto)

finally show ?thesis

by (simp add: ExtChoice-def)

qed

lemma extChoice-rdes:

assumes $\$ok' \# P_1 \ \$ok' \# Q_1$

shows $\mathbf{R}_s(P_1 \vdash P_2) \sqcap \mathbf{R}_s(Q_1 \vdash Q_2) = \mathbf{R}_s((P_1 \wedge Q_1) \vdash ((P_2 \wedge Q_2) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (P_2 \vee Q_2)))$

proof –

have $(\sqcap i :: nat \in \{0, 1\} \cdot \mathbf{R}_s(P_1 \vdash P_2) \triangleleft \ll i = 0 \gg \triangleright \mathbf{R}_s(Q_1 \vdash Q_2)) = (\sqcap i :: nat \in \{0, 1\} \cdot \mathbf{R}_s((P_1 \vdash P_2) \triangleleft \ll i = 0 \gg \triangleright (Q_1 \vdash Q_2)))$

by (*simp only: RHS-cond R2c-lit*)
 also have ... = ($\Box i :: \text{nat} \in \{0, 1\} \cdot \mathbf{R}_s ((P_1 \triangleleft \ll i = 0 \gg \triangleright Q_1) \vdash (P_2 \triangleleft \ll i = 0 \gg \triangleright Q_2))$)
 by (*simp add: design-condr*)
 also have ... = $\mathbf{R}_s ((P_1 \wedge Q_1) \vdash ((P_2 \wedge Q_2) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (P_2 \vee Q_2)))$
 apply (*subst ExtChoice-rdes, simp-all add: assms unrest*)
 apply *unliteralise*
 apply (*simp add: uinf-or usup-and*)
 done
 finally show ?thesis by (*simp add: extChoice-alt-def*)
 qed

lemma *extChoice-tri-rdes*:

assumes $\$ok' \# P_1 \$ok' \# Q_1$
 shows $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) \Box \mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3) =$
 $\mathbf{R}_s((P_1 \wedge Q_1) \vdash (((P_2 \wedge Q_2) \triangleleft \$tr' =_u \$tr \triangleright (P_2 \vee Q_2)) \diamond (P_3 \vee Q_3)))$
 proof –
 have $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) \Box \mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3) =$
 $\mathbf{R}_s((P_1 \wedge Q_1) \vdash ((P_2 \diamond P_3 \wedge Q_2 \diamond Q_3) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (P_2 \diamond P_3 \vee Q_2 \diamond Q_3)))$
 by (*simp add: extChoice-rdes assms*)
 also
 have ... = $\mathbf{R}_s((P_1 \wedge Q_1) \vdash ((P_2 \diamond P_3 \wedge Q_2 \diamond Q_3) \triangleleft \$wait' \wedge \$tr' =_u \$tr \triangleright (P_2 \diamond P_3 \vee Q_2 \diamond Q_3)))$
 by (*simp add: conj-comm*)
 also
 have ... = $\mathbf{R}_s((P_1 \wedge Q_1) \vdash$
 $((P_2 \diamond P_3 \wedge Q_2 \diamond Q_3) \triangleleft \$tr' =_u \$tr \triangleright (P_2 \diamond P_3 \vee Q_2 \diamond Q_3)) \diamond (P_2 \diamond P_3 \vee Q_2 \diamond Q_3)))$
 by (*simp add: cond-conj wait'-cond-def*)
 also
 have ... = $\mathbf{R}_s((P_1 \wedge Q_1) \vdash (((P_2 \wedge Q_2) \triangleleft \$tr' =_u \$tr \triangleright (P_2 \vee Q_2)) \diamond (P_3 \vee Q_3)))$
 by (*rule cong[of $\mathbf{R}_s \mathbf{R}_s$], simp, rel-auto*)
 finally show ?thesis .
 qed

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

assumes P_1 is *RR* Q_1 is *RR*
 shows $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) \Box \mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3) =$
 $\mathbf{R}_s((P_1 \wedge Q_1) \vdash (((P_2 \wedge Q_2) \triangleleft \$tr' =_u \$tr \triangleright (P_2 \vee Q_2)) \diamond (P_3 \vee Q_3)))$
 by (*subst extChoice-tri-rdes, simp-all add: assms unrest*)

lemma *CSP-ExtChoice* [*closure*]:

ExtChoice A is *CSP*
 by (*simp add: ExtChoice-def RHS-design-is-SRD unrest*)

lemma *CSP-extChoice* [*closure*]:

$P \Box Q$ is *CSP*
 by (*simp add: CSP-ExtChoice extChoice-def*)

lemma *preR-ExtChoice* [*rdes*]:

assumes $A \neq \{\}$ $A \subseteq \llbracket \text{CSP} \rrbracket_H$
 shows $\text{pre}_R(\text{ExtChoice } A) = (\bigsqcup P \in A \cdot \text{pre}_R(P))$

proof –

have $\text{pre}_R(\text{ExtChoice } A) = (R1 (R2c ((\bigsqcup P \in A \cdot \text{pre}_R(P))))$
 by (*simp add: ExtChoice-def rea-pre-RHS-design usubst unrest*)
 also from *assms* have ... = $(R1 (R2c (\bigsqcup P \in A \cdot (\text{pre}_R(\text{CSP}(P)))))$
 by (*metis USUP-healthy*)
 also from *assms* have ... = $(\bigsqcup P \in A \cdot (\text{pre}_R(\text{CSP}(P))))$

by (*rel-auto*)
 also from *assms* have ... = $(\bigsqcup P \in A \cdot (pre_R(P)))$
 by (*metis USUP-healthy*)
 finally show *?thesis* .
 qed

lemma *preR-ExtChoice-ind* [*rdes*]:
 assumes $A \neq \{\}$ \wedge $P. P \in A \implies F(P)$ is CSP
 shows $pre_R(\bigsqcup P \in A \cdot F(P)) = (\bigsqcup P \in A \cdot pre_R(F(P)))$
 using *assms* by (*subst preR-ExtChoice, auto*)

lemma *periR-ExtChoice* [*rdes*]:
 assumes $A \subseteq \llbracket NCSP \rrbracket_H$ $A \neq \{\}$
 shows $peri_R(ExtChoice\ A) = ((\bigsqcup P \in A \cdot pre_R(P)) \Rightarrow_r (\bigsqcup P \in A \cdot peri_R\ P)) \triangleleft \$tr' =_u \$tr \triangleright (\bigsqcap P \in A \cdot peri_R\ P)$

proof –

have $peri_R(ExtChoice\ A) = peri_R(\mathbf{R}_s((\bigsqcup P \in A \cdot pre_R\ P)) \vdash$
 $((\bigsqcup P \in A \cdot peri_R\ P) \triangleleft \$tr' =_u \$tr \triangleright (\bigsqcap P \in A \cdot peri_R\ P)) \diamond$
 $(\bigsqcap P \in A \cdot post_R\ P)))$
 by (*simp add: ExtChoice-tri-rdes-def assms closure*)

also have ... = $peri_R(\mathbf{R}_s((\bigsqcup P \in A \cdot pre_R(NCSP\ P)) \vdash$
 $((\bigsqcup P \in A \cdot peri_R(NCSP\ P)) \triangleleft \$tr' =_u \$tr \triangleright (\bigsqcap P \in A \cdot peri_R(NCSP\ P))) \diamond$
 $(\bigsqcap P \in A \cdot post_R\ P)))$
 by (*simp add: UINF-healthy[OF assms(1), THEN sym] USUP-healthy[OF assms(1), THEN sym]*)

also have ... = $R1\ (R2c\ ((\bigsqcup P \in A \cdot pre_R(NCSP\ P)) \Rightarrow_r$
 $(\bigsqcup P \in A \cdot peri_R(NCSP\ P))$
 $\triangleleft \$tr' =_u \$tr \triangleright$
 $(\bigsqcap P \in A \cdot peri_R(NCSP\ P))))$

proof –

have $(\bigsqcup P \in A \cdot [\$ok \mapsto_s true, \$ok' \mapsto_s true, \$wait \mapsto_s false, \$wait' \mapsto_s true] \dagger pre_R(NCSP\ P))$
 = $(\bigsqcup P \in A \cdot pre_R(NCSP\ P))$
 by (*rule USUP-cong, simp add: closure usubst unrest assms*)
 thus *?thesis*
 by (*simp add: rea-peri-RHS-design Healthy-Idempotent SRD-Idempotent usubst unrest assms*)

qed

also have ... = $R1\ ((\bigsqcup P \in A \cdot pre_R(NCSP\ P)) \Rightarrow_r$
 $(\bigsqcup P \in A \cdot peri_R(NCSP\ P))$
 $\triangleleft \$tr' =_u \$tr \triangleright$
 $(\bigsqcap P \in A \cdot peri_R(NCSP\ P)))$

by (*simp add: R2c-rea-impl R2c-cond R2c-UINF R2c-preR R2c-periR R2c-tr'-minus-tr R2c-USUP closure*)

also have ... = $((\bigsqcup P \in A \cdot pre_R(NCSP\ P)) \Rightarrow_r (\bigsqcup P \in A \cdot peri_R(NCSP\ P)))$
 $\triangleleft \$tr' =_u \$tr \triangleright$
 $((\bigsqcup P \in A \cdot pre_R(NCSP\ P)) \Rightarrow_r (\bigsqcap P \in A \cdot peri_R(NCSP\ P)))$

by (*simp add: R1-rea-impl R1-cond R1-USUP R1-UINF assms Healthy-if closure, rel-auto*)

also have ... = $((\bigsqcup P \in A \cdot pre_R(NCSP\ P)) \Rightarrow_r (\bigsqcup P \in A \cdot peri_R(NCSP\ P)))$
 $\triangleleft \$tr' =_u \$tr \triangleright$
 $((\bigsqcap P \in A \cdot pre_R(NCSP\ P)) \Rightarrow_r peri_R(NCSP\ P)))$

by (*simp add: UINF-rea-impl[THEN sym]*)

also have ... = $((\bigsqcup P \in A \cdot pre_R(NCSP\ P)) \Rightarrow_r (\bigsqcup P \in A \cdot peri_R(NCSP\ P)))$
 $\triangleleft \$tr' =_u \$tr \triangleright$
 $((\bigsqcap P \in A \cdot peri_R(NCSP\ P)))$

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

also have ... = $((\bigsqcup P \in A \cdot \text{pre}_R P) \Rightarrow_r (\bigsqcup P \in A \cdot \text{peri}_R P))$
 $\triangleleft \$tr' =_u \$tr \triangleright$
 $((\bigsqcup P \in A \cdot \text{peri}_R P))$
 by (simp add: UINF-healthy[OF assms(1), THEN sym] USUP-healthy[OF assms(1), THEN sym])
 finally show ?thesis .
 qed

lemma *periR-ExtChoice-ind* [rdes]:
 assumes $\bigwedge P. P \in A \implies F(P)$ is NCSP $A \neq \{\}$
 shows $\text{peri}_R(\bigsqcup P \in A \cdot F(P)) = ((\bigsqcup P \in A \cdot \text{pre}_R(F P)) \Rightarrow_r (\bigsqcup P \in A \cdot \text{peri}_R(F P))) \triangleleft \$tr' =_u \$tr$
 $\triangleright (\bigsqcup P \in A \cdot \text{peri}_R(F P))$
 using assms by (subst periR-ExtChoice, auto simp add: closure unrest)

lemma *postR-ExtChoice* [rdes]:
 assumes $A \subseteq \llbracket \text{NCSP} \rrbracket_H A \neq \{\}$
 shows $\text{post}_R(\text{ExtChoice } A) = (\bigsqcup P \in A \cdot \text{post}_R P)$

proof –

have $\text{post}_R(\text{ExtChoice } A) = \text{post}_R(\mathbf{R}_s((\bigsqcup P \in A \cdot \text{pre}_R P)) \vdash$
 $((\bigsqcup P \in A \cdot \text{peri}_R P) \triangleleft \$tr' =_u \$tr \triangleright (\bigsqcup P \in A \cdot \text{peri}_R P)) \diamond$
 $(\bigsqcup P \in A \cdot \text{post}_R P))$
 by (simp add: ExtChoice-tri-rdes-def closure assms)

also have ... = $\text{post}_R(\mathbf{R}_s((\bigsqcup P \in A \cdot \text{pre}_R(\text{NCSP } P)) \vdash$
 $((\bigsqcup P \in A \cdot \text{peri}_R P) \triangleleft \$tr' =_u \$tr \triangleright (\bigsqcup P \in A \cdot \text{peri}_R P)) \diamond$
 $(\bigsqcup P \in A \cdot \text{post}_R(\text{NCSP } P))))$
 by (simp add: UINF-healthy[OF assms(1), THEN sym] USUP-healthy[OF assms(1), THEN sym])

also have ... = $R1(R2c((\bigsqcup P \in A \cdot \text{pre}_R(\text{NCSP } P)) \Rightarrow_r (\bigsqcup P \in A \cdot \text{post}_R(\text{NCSP } P))))$

proof –

have $(\bigsqcup P \in A \cdot [\$ok \mapsto_s \text{true}, \$ok' \mapsto_s \text{true}, \$wait \mapsto_s \text{false}, \$wait' \mapsto_s \text{false}] \dagger \text{pre}_R(\text{NCSP } P))$
 = $(\bigsqcup P \in A \cdot \text{pre}_R(\text{NCSP } P))$
 by (rule USUP-cong, simp add: usubst closure unrest assms)
 thus ?thesis
 by (simp add: rea-post-RHS-design Healthy-Idempotent SRD-Idempotent usubst unrest assms)

qed

also have ... = $R1((\bigsqcup P \in A \cdot \text{pre}_R(\text{NCSP } P)) \Rightarrow_r (\bigsqcup P \in A \cdot \text{post}_R(\text{NCSP } P)))$

by (simp add: R2c-rea-impl R2c-cond R2c-UINF R2c-preR R2c-postR
 R2c-tr'-minus-tr R2c-USUP closure)

also from assms(2) have ... = $((\bigsqcup P \in A \cdot \text{pre}_R(\text{NCSP } P)) \Rightarrow_r (\bigsqcup P \in A \cdot \text{post}_R(\text{NCSP } P)))$

by (simp add: R1-rea-impl R1-cond R1-USUP R1-UINF assms Healthy-if closure)

also have ... = $(\bigsqcup P \in A \cdot \text{pre}_R(\text{NCSP } P) \Rightarrow_r \text{post}_R(\text{NCSP } P))$

by (simp add: UINF-rea-impl)

also have ... = $(\bigsqcup P \in A \cdot \text{post}_R(\text{NCSP } P))$

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

finally show ?thesis

by (simp add: UINF-healthy[OF assms(1), THEN sym] USUP-healthy[OF assms(1), THEN sym])

qed

lemma *postR-ExtChoice-ind* [rdes]:
 assumes $\bigwedge P. P \in A \implies F(P)$ is NCSP $A \neq \{\}$
 shows $\text{post}_R(\bigsqcup P \in A \cdot F(P)) = (\bigsqcup P \in A \cdot \text{post}_R(F(P)))$
 using assms by (subst postR-ExtChoice, auto simp add: closure unrest)

lemma *preR-extChoice*:

assumes P is CSP Q is CSP $\$wait' \# \text{pre}_R(P) \# \text{pre}_R(Q)$

shows $pre_R(P \sqcap Q) = (pre_R(P) \wedge pre_R(Q))$
by (*simp add: extChoice-def preR-ExtChoice assms usup-and*)

lemma *preR-extChoice' [rdes]*:
assumes P is NCSP Q is NCSP
shows $pre_R(P \sqcap Q) = (pre_R(P) \wedge pre_R(Q))$
by (*simp add: preR-extChoice closure assms unrest*)

lemma *periR-extChoice [rdes]*:
assumes P is NCSP Q is NCSP
shows $peri_R(P \sqcap Q) = ((pre_R(P) \wedge pre_R(Q) \Rightarrow_r peri_R(P) \wedge peri_R(Q)) \triangleleft \$tr' =_u \$tr \triangleright (peri_R(P) \vee peri_R(Q)))$
using *assms*
by (*simp add: extChoice-def, subst periR-ExtChoice, auto simp add: usup-and uinf-or*)

lemma *postR-extChoice [rdes]*:
assumes P is NCSP Q is NCSP
shows $post_R(P \sqcap Q) = (post_R(P) \vee post_R(Q))$
using *assms*
by (*simp add: extChoice-def, subst postR-ExtChoice, auto simp add: usup-and uinf-or*)

lemma *ExtChoice-cong*:
assumes $\bigwedge P. P \in A \implies F(P) = G(P)$
shows $(\sqcap P \in A \cdot F(P)) = (\sqcap P \in A \cdot G(P))$
using *assms image-cong* **by** *force*

lemma *ref-unrest-ExtChoice*:
assumes
 $\bigwedge P. P \in A \implies \$ref \# pre_R(P)$
 $\bigwedge P. P \in A \implies \$ref \# cmt_R(P)$
shows $\$ref \# (ExtChoice\ A) \llbracket false / \$wait \rrbracket$
proof –
have $\bigwedge P. P \in A \implies \$ref \# pre_R(P \llbracket 0 / \$tr \rrbracket)$
using *assms* **by** (*rel-blast*)
with *assms* **show** *?thesis*
by (*simp add: ExtChoice-def RHS-def R1-def R2c-def R2s-def R3h-def design-def usubst unrest*)
qed

lemma *CSP4-ExtChoice*:
assumes $A \subseteq \llbracket NCSP \rrbracket_H$
shows *ExtChoice A* is CSP4
proof (*cases A = {}*)
case *True* **thus** *?thesis*
by (*simp add: ExtChoice-empty Healthy-def CSP4-def, simp add: Skip-is-CSP Stop-left-zero*)
next
case *False*
have $1: (\neg_r (\neg_r pre_R (ExtChoice\ A)) ;;_h R1\ true) = pre_R (ExtChoice\ A)$
proof –
have $\bigwedge P. P \in A \implies (\neg_r pre_R(P)) ;; R1\ true = (\neg_r pre_R(P))$
by (*simp add: NCSP-Healthy-subset-member NCSP-implies-NSRD NSRD-neg-pre-unit assms*)
thus *?thesis*
apply (*simp add: False preR-ExtChoice closure NCSP-set-unrest-pre-wait' assms not-UINF seq-UINF-distr not-USUP*)
apply (*rule USUP-cong*)
apply (*simp add: rpred assms closure*)

```

    done
  qed
  have 2:  $\$st' \# \text{peri}_R (\text{ExtChoice } A)$ 
  proof -
    have a:  $\bigwedge P. P \in A \implies \$st' \# \text{pre}_R(P)$ 
      by (simp add: NCSP-Healthy-subset-member NCSP-implies-NSRD NSRD-st'-unrest-pre assms)
    have b:  $\bigwedge P. P \in A \implies \$st' \# \text{peri}_R(P)$ 
      by (simp add: NCSP-Healthy-subset-member NCSP-implies-NSRD NSRD-st'-unrest-peri assms)
    from a b show ?thesis
      apply (subst periR-ExtChoice)
      apply (simp-all add: assms closure unrest CSP4-set-unrest-pre-st' NCSP-set-unrest-pre-wait'
False)
    done
  qed
  have 3:  $\$ref' \# \text{post}_R (\text{ExtChoice } A)$ 
  proof -
    have a:  $\bigwedge P. P \in A \implies \$ref' \# \text{pre}_R(P)$ 
      by (simp add: CSP4-ref'-unrest-pre CSP-Healthy-subset-member NCSP-Healthy-subset-member
NCSP-implies-CSP4 NCSP-subset-implies-CSP assms)
    have b:  $\bigwedge P. P \in A \implies \$ref' \# \text{post}_R(P)$ 
      by (simp add: CSP4-ref'-unrest-post CSP-Healthy-subset-member NCSP-Healthy-subset-member
NCSP-implies-CSP4 NCSP-subset-implies-CSP assms)
    from a b show ?thesis
      by (subst postR-ExtChoice, simp-all add: assms CSP4-set-unrest-pre-st' NCSP-set-unrest-pre-wait'
unrest False)
    qed
  show ?thesis
    by (rule CSP4-tri-intro, simp-all add: 1 2 3 assms closure)
    (metis 1 R1-seqr-closure rea-not-R1 rea-not-not rea-true-R1)
  qed

lemma CSP4-extChoice [closure]:
  assumes  $P \text{ is NCSP } Q \text{ is NCSP}$ 
  shows  $P \sqcap Q \text{ is CSP4}$ 
  by (simp add: extChoice-def, rule CSP4-ExtChoice, simp-all add: assms)

lemma NCSP-ExtChoice [closure]:
  assumes  $A \subseteq \llbracket \text{NCSP} \rrbracket_H$ 
  shows  $\text{ExtChoice } A \text{ is NCSP}$ 
proof (cases  $A = \{\}$ )
  case True
  then show ?thesis by (simp add: ExtChoice-empty closure)
next
  case False
  show ?thesis
  proof (rule NCSP-intro)
    from assms have cls:  $A \subseteq \llbracket \text{CSP} \rrbracket_H \ A \subseteq \llbracket \text{CSP3} \rrbracket_H \ A \subseteq \llbracket \text{CSP4} \rrbracket_H$ 
      using NCSP-implies-CSP NCSP-implies-CSP3 NCSP-implies-CSP4 by blast+
    have wu:  $\bigwedge P. P \in A \implies \$wait' \# \text{pre}_R(P)$ 
      using NCSP-implies-NSRD NSRD-wait'-unrest-pre assms by force
    show 1:  $\text{ExtChoice } A \text{ is CSP}$ 
      by (metis (mono-tags) Ball-Collect CSP-ExtChoice NCSP-implies-CSP assms)
    from cls show  $\text{ExtChoice } A \text{ is CSP3}$ 
      by (rule-tac CSP3-SRD-intro, simp-all add: CSP-Healthy-subset-member CSP3-Healthy-subset-member
closure rdes unrest wu assms 1 False)

```

from *cls* show *ExtChoice A is CSP4*
 by (*simp add: CSP4-ExtChoice assms*)
 qed
 qed

lemma *NCSP-extChoice [closure]*:
 assumes *P is NCSP Q is NCSP*
 shows *P \sqsubseteq Q is NCSP*
 by (*simp add: NCSP-ExtChoice assms extChoice-def*)

7.5 Productivity and Guardedness

lemma *Productive-ExtChoice [closure]*:
 assumes *A $\neq \{\}$ A $\subseteq \llbracket NCSP \rrbracket_H$ A $\subseteq \llbracket Productive \rrbracket_H$*
 shows *ExtChoice A is Productive*

proof –

have *1: $\bigwedge P. P \in A \implies \$wait' \nmid pre_R(P)$*
 using *NCSP-implies-NSRD NSRD-wait'-unrest-pre assms(2)* by *blast*
 show *?thesis*

proof (*rule Productive-intro, simp-all add: assms closure rdes 1 unrest*)

have $((\bigcup P \in A \cdot pre_R P) \wedge (\bigcap P \in A \cdot post_R P)) =$
 $((\bigcup P \in A \cdot pre_R P) \wedge (\bigcap P \in A \cdot (pre_R P \wedge post_R P)))$
 by (*rel-auto*)

moreover have $(\bigcap P \in A \cdot (pre_R P \wedge post_R P)) = (\bigcap P \in A \cdot ((pre_R P \wedge post_R P) \wedge \$tr <_u \$tr'))$

by (*rule UINF-cong, metis (no-types, lifting) 1 Ball-Collect NCSP-implies-CSP Productive-post-refines-tr-increase assms utp-pred-laws.inf.absorb1*)

ultimately show $(\$tr' >_u \$tr) \sqsubseteq ((\bigcup P \in A \cdot pre_R P) \wedge (\bigcap P \in A \cdot post_R P))$
 by (*rel-auto*)

qed
 qed

lemma *Productive-extChoice [closure]*:
 assumes *P is NCSP Q is NCSP P is Productive Q is Productive*
 shows *P \sqsubseteq Q is Productive*
 by (*simp add: extChoice-def Productive-ExtChoice assms*)

lemma *ExtChoice-Guarded [closure]*:
 assumes $\bigwedge P. P \in A \implies Guarded P$
 shows *Guarded $(\lambda X. \bigcap P \in A \cdot P(X))$*

proof (*rule GuardedI*)

fix *X n*

have $\bigwedge Y. ((\bigcap P \in A \cdot P Y) \wedge gvirt(n+1)) = ((\bigcap P \in A \cdot (P Y \wedge gvirt(n+1))) \wedge gvirt(n+1))$

proof –

fix *Y*

let *?lhs* = $((\bigcap P \in A \cdot P Y) \wedge gvirt(n+1))$ and *?rhs* = $((\bigcap P \in A \cdot (P Y \wedge gvirt(n+1))) \wedge gvirt(n+1))$

have *a: ?lhs* $\llbracket false/\$ok \rrbracket = ?rhs \llbracket false/\$ok \rrbracket$

by (*rel-auto*)

have *b: ?lhs* $\llbracket true/\$ok \rrbracket \llbracket true/\$wait \rrbracket = ?rhs \llbracket true/\$ok \rrbracket \llbracket true/\$wait \rrbracket$

by (*rel-auto*)

have *c: ?lhs* $\llbracket true/\$ok \rrbracket \llbracket false/\$wait \rrbracket = ?rhs \llbracket true/\$ok \rrbracket \llbracket false/\$wait \rrbracket$

by (*simp add: ExtChoice-def RHS-def R1-def R2c-def R2s-def R3h-def design-def usubst unrest, rel-blast*)

show *?lhs = ?rhs*

using *a b c*

```

    by (rule-tac bool-eq-splitI[of in-var ok], simp, rule-tac bool-eq-splitI[of in-var wait], simp-all)
  qed
  moreover have  $((\Box P \in A \cdot (P \ X \ \wedge \ gvirt(n+1))) \wedge gvirt(n+1)) = ((\Box P \in A \cdot (P \ (X \ \wedge \ gvirt(n)) \wedge gvirt(n+1))) \wedge gvirt(n+1))$ 
  proof -
    have  $(\Box P \in A \cdot (P \ X \ \wedge \ gvirt(n+1))) = (\Box P \in A \cdot (P \ (X \ \wedge \ gvirt(n)) \wedge gvirt(n+1)))$ 
    proof (rule ExtChoice-cong)
      fix P assume  $P \in A$ 
      thus  $(P \ X \ \wedge \ gvirt(n+1)) = (P \ (X \ \wedge \ gvirt(n)) \wedge gvirt(n+1))$ 
      using Guarded-def assms by blast
    qed
    thus ?thesis by simp
  qed
  ultimately show  $((\Box P \in A \cdot P \ X) \wedge gvirt(n+1)) = ((\Box P \in A \cdot (P \ (X \ \wedge \ gvirt(n)))) \wedge gvirt(n+1))$ 
  by simp
qed

```

```

lemma extChoice-Guarded [closure]:
  assumes Guarded P Guarded Q
  shows Guarded  $(\lambda X. P(X) \Box Q(X))$ 
proof -
  have Guarded  $(\lambda X. \Box F \in \{P, Q\} \cdot F(X))$ 
  by (rule ExtChoice-Guarded, auto simp add: assms)
  thus ?thesis
  by (simp add: extChoice-def)
qed

```

7.6 Algebraic laws

```

lemma extChoice-comm:
   $P \Box Q = Q \Box P$ 
  by (unfold extChoice-def, simp add: insert-commute)

```

```

lemma extChoice-idem:
   $P \text{ is CSP} \implies P \Box P = P$ 
  by (unfold extChoice-def, simp add: ExtChoice-single)

```

```

lemma extChoice-assoc:
  assumes  $P \text{ is CSP } Q \text{ is CSP } R \text{ is CSP}$ 
  shows  $P \Box Q \Box R = P \Box (Q \Box R)$ 
proof -
  have  $P \Box Q \Box R = \mathbf{R}_s(\text{pre}_R(P) \vdash \text{cmt}_R(P)) \Box \mathbf{R}_s(\text{pre}_R(Q) \vdash \text{cmt}_R(Q)) \Box \mathbf{R}_s(\text{pre}_R(R) \vdash \text{cmt}_R(R))$ 
  by (simp add: SRD-reactive-design-alt assms(1) assms(2) assms(3))
  also have ... =
     $\mathbf{R}_s(((\text{pre}_R P \wedge \text{pre}_R Q) \wedge \text{pre}_R R) \vdash$ 
       $((\text{cmt}_R P \wedge \text{cmt}_R Q) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\text{cmt}_R P \vee \text{cmt}_R Q) \wedge \text{cmt}_R R)$ 
       $\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$ 
       $((\text{cmt}_R P \wedge \text{cmt}_R Q) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\text{cmt}_R P \vee \text{cmt}_R Q) \vee \text{cmt}_R R)))$ 
  by (simp add: extChoice-rdes unrest)
  also have ... =
     $\mathbf{R}_s(((\text{pre}_R P \wedge \text{pre}_R Q) \wedge \text{pre}_R R) \vdash$ 
       $((\text{cmt}_R P \wedge \text{cmt}_R Q) \wedge \text{cmt}_R R)$ 
       $\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$ 
       $((\text{cmt}_R P \vee \text{cmt}_R Q) \vee \text{cmt}_R R)))$ 
  by (rule cong[of  $\mathbf{R}_s \ \mathbf{R}_s$ ], simp, rel-auto)
  also have ... =

```


$\mathbf{R}_s ((pre_R P \wedge pre_R Q \wedge pre_R R) \vdash$
 $((cmt_R P \wedge (cmt_R Q \wedge cmt_R R))$
 $\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
 $(cmt_R P \vee (cmt_R Q \vee cmt_R R))))$
 by (simp add: conj-assoc disj-assoc)
 also have ... =
 $\mathbf{R}_s ((pre_R P \wedge pre_R Q \wedge pre_R R) \vdash$
 $((cmt_R P \wedge (cmt_R Q \wedge cmt_R R) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (cmt_R Q \vee cmt_R R))$
 $\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
 $(cmt_R P \vee (cmt_R Q \wedge cmt_R R) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (cmt_R Q \vee cmt_R R))))$
 by (rule cong[of $\mathbf{R}_s \mathbf{R}_s$], simp, rel-auto)
 also have ... = $\mathbf{R}_s(pre_R(P) \vdash cmt_R(P)) \sqcap (\mathbf{R}_s(pre_R(Q) \vdash cmt_R(Q)) \sqcap \mathbf{R}_s(pre_R(R) \vdash cmt_R(R)))$
 by (simp add: extChoice-rdes unrest)
 also have ... = $P \sqcap (Q \sqcap R)$
 by (simp add: SRD-reactive-design-alt assms(1) assms(2) assms(3))
 finally show ?thesis .
 qed

lemma extChoice-Stop:
 assumes Q is CSP
 shows $Stop \sqcap Q = Q$
 using assms
proof –
 have $Stop \sqcap Q = \mathbf{R}_s (true \vdash (\$tr' =_u \$tr \wedge \$wait')) \sqcap \mathbf{R}_s(pre_R(Q) \vdash cmt_R(Q))$
 by (simp add: Stop-def SRD-reactive-design-alt assms)
 also have ... = $\mathbf{R}_s (pre_R Q \vdash (((\$tr' =_u \$tr \wedge \$wait') \wedge cmt_R Q) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\$tr' =_u \$tr \wedge \$wait' \vee cmt_R Q)))$
 by (simp add: extChoice-rdes unrest)
 also have ... = $\mathbf{R}_s (pre_R Q \vdash (cmt_R Q \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright cmt_R Q))$
 by (metis (no-types, lifting) cond-def eq-upred-sym neg-conj-cancel1 utp-pred-laws.inf.left-idem)
 also have ... = $\mathbf{R}_s (pre_R Q \vdash cmt_R Q)$
 by (simp add: cond-idem)
 also have ... = Q
 by (simp add: SRD-reactive-design-alt assms)
 finally show ?thesis .
 qed

lemma extChoice-Chaos:
 assumes Q is CSP
 shows $Chaos \sqcap Q = Chaos$
proof –
 have $Chaos \sqcap Q = \mathbf{R}_s (false \vdash true) \sqcap \mathbf{R}_s(pre_R(Q) \vdash cmt_R(Q))$
 by (simp add: Chaos-def SRD-reactive-design-alt assms)
 also have ... = $\mathbf{R}_s (false \vdash (cmt_R Q \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright true))$
 by (simp add: extChoice-rdes unrest)
 also have ... = $\mathbf{R}_s (false \vdash true)$
 by (rule cong[of $\mathbf{R}_s \mathbf{R}_s$], simp, rel-auto)
 also have ... = $Chaos$
 by (simp add: Chaos-def)
 finally show ?thesis .
 qed

lemma extChoice-Dist:
 assumes P is CSP $S \subseteq \llbracket CSP \rrbracket_H S \neq \{\}$
 shows $P \sqcap (\bigsqcap S) = (\bigsqcap_{Q \in S} P \sqcap Q)$

proof –

let $?S1 = pre_R \text{ ' } S$ and $?S2 = cmt_R \text{ ' } S$
 have $P \sqcap (\bigsqcap S) = P \sqcap (\bigsqcap_{Q \in S} \mathbf{R}_s(pre_R(Q) \vdash cmt_R(Q)))$
 by (simp add: SRD-as-reactive-design[THEN sym] Healthy-SUPREMUM UINF-as-Sup-collect assms)
 also have $\dots = \mathbf{R}_s(pre_R(P) \vdash cmt_R(P)) \sqcap \mathbf{R}_s(\bigsqcap_{Q \in S} pre_R(Q) \vdash (\bigsqcap_{Q \in S} cmt_R(Q)))$
 by (simp add: RHS-design-USUP SRD-reactive-design-alt assms)
 also have $\dots = \mathbf{R}_s((pre_R(P) \wedge (\bigsqcap_{Q \in S} pre_R(Q))) \vdash$
 $((cmt_R(P) \wedge (\bigsqcap_{Q \in S} cmt_R(Q)))$
 $\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
 $(cmt_R(P) \vee (\bigsqcap_{Q \in S} cmt_R(Q))))$
 by (simp add: extChoice-rdes unrest)
 also have $\dots = \mathbf{R}_s((\bigsqcap_{Q \in S} pre_R P \wedge pre_R Q) \vdash$
 $(\bigsqcap_{Q \in S} (cmt_R P \wedge cmt_R Q) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (cmt_R P \vee cmt_R Q)))$
 by (simp add: conj-USUP-dist conj-UINF-dist disj-UINF-dist cond-UINF-dist assms)
 also have $\dots = (\bigsqcap_{Q \in S} \mathbf{R}_s((pre_R P \wedge pre_R Q) \vdash$
 $((cmt_R P \wedge cmt_R Q) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (cmt_R P \vee cmt_R Q))))$
 by (simp add: assms RHS-design-USUP)
 also have $\dots = (\bigsqcap_{Q \in S} \mathbf{R}_s(pre_R(P) \vdash cmt_R(P)) \sqcap \mathbf{R}_s(pre_R(Q) \vdash cmt_R(Q)))$
 by (simp add: extChoice-rdes unrest)
 also have $\dots = (\bigsqcap_{Q \in S} P \sqcap CSP(Q))$
 by (simp add: UINF-as-Sup-collect, metis (no-types, lifting) Healthy-if SRD-as-reactive-design
 assms(1))
 also have $\dots = (\bigsqcap_{Q \in S} P \sqcap Q)$
 by (rule SUP-cong, simp-all add: Healthy-subset-member[OF assms(2)])
 finally show $?thesis$.
qed

lemma *extChoice-dist*:

assumes P is CSP Q is CSP R is CSP
 shows $P \sqcap (Q \sqcap R) = (P \sqcap Q) \sqcap (P \sqcap R)$
 using assms extChoice-Dist[of $P \{Q, R\}$] by simp

lemma *ExtChoice-seq-distr*:

assumes $\bigwedge i. i \in A \implies P i$ is PCSP Q is NCSP
 shows $(\bigsqcap_{i \in A} P i) ;; Q = (\bigsqcap_{i \in A} P i ;; Q)$

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

case True

then show $?thesis$

by (simp add: ExtChoice-empty NCSP-implies-CSP Stop-left-zero assms(2))

next

case False

show $?thesis$

proof –

have $1: (\bigsqcap_{i \in A} P i) = (\bigsqcap_{i \in A} (\mathbf{R}_s((pre_R(P i)) \vdash peri_R(P i) \diamond (post_R(P i) \wedge \$tr <_u \$tr'))))$
 (is $?X = ?Y$)

by (rule ExtChoice-cong, metis (no-types, hide-lams) Healthy-if NCSP-implies-CSP PCSP-implies-NCSP Productive-form assms(1) comp-apply)

have $2: (\bigsqcap_{i \in A} P i ;; Q) = (\bigsqcap_{i \in A} (\mathbf{R}_s((pre_R(P i)) \vdash peri_R(P i) \diamond (post_R(P i) \wedge \$tr <_u \$tr')))) ;; Q$
 (is $?X = ?Y$)

by (rule ExtChoice-cong, metis (no-types, hide-lams) Healthy-if NCSP-implies-CSP PCSP-implies-NCSP Productive-form assms(1) comp-apply)

show $?thesis$

apply (simp add: 1 2)

apply (rds-simp cls: assms False cong: ExtChoice-cong)

```

    apply (rule-tac srdes-tri-eq-intro)
    apply (rel-blast)
    apply (rel-auto; blast)
    apply (rel-auto; blast)
  done
qed
qed

lemma extChoice-seq-distr:
  assumes P is PCSP Q is PCSP R is NCSP
  shows  $(P \sqcap Q) ;; R = (P ;; R \sqcap Q ;; R)$ 
  by (rdes-eq cls: assms)

lemma extChoice-seq-distl:
  assumes P is ICSP Q is ICSP R is NCSP
  shows  $P ;; (Q \sqcap R) = (P ;; Q \sqcap P ;; R)$ 
  by (rdes-eq cls: assms)

end

```

8 Circus and CSP Actions

```

theory utp-circus-actions
  imports
    utp-circus-extchoice
begin

```

8.1 Conditionals

```

lemma NCSP-cond-srea [closure]:
  assumes P is NCSP Q is NCSP
  shows  $P \triangleleft b \triangleright_R Q \text{ is NCSP}$ 
  by (rule NCSP-NSRD-intro, simp-all add: closure rdes assms unrest)

```

8.2 Guarded commands

```

lemma GuardedCommR-NCSP-closed [closure]:
  assumes P is NCSP
  shows  $g \rightarrow_R P \text{ is NCSP}$ 
  by (simp add: gcmd-def closure assms)

```

8.3 Alternation

```

lemma AlternateR-NCSP-closed [closure]:
  assumes  $\bigwedge i. i \in A \implies P(i) \text{ is NCSP } Q \text{ is NCSP}$ 
  shows  $(\text{if}_R i \in A \cdot g(i) \rightarrow P(i) \text{ else } Q \text{ fi}) \text{ is NCSP}$ 
proof (cases  $A = \{\}$ )
  case True
  then show ?thesis
    by (simp add: assms)
next
  case False
  then show ?thesis
    by (simp add: AlternateR-def closure assms)
qed

```

8.4 Assignment

definition *AssignsCSP* :: ' σ *usubst* \Rightarrow (' σ , ' φ) *action* ($\langle \cdot \rangle_C$) **where**
 $[upred-defs]: \text{AssignsCSP } \sigma = \mathbf{R}_s(\text{true} \vdash \text{false} \diamond (\$tr' =_u \$tr \wedge \lceil \langle \sigma \rangle_a \rceil_S))$

syntax

-*assigns-csp* :: *svids* \Rightarrow *uexprs* \Rightarrow *logic* ('(\cdot) :=_C '(\cdot))
 -*assigns-csp* :: *svids* \Rightarrow *uexprs* \Rightarrow *logic* (**infixr** :=_C 90)

translations

-*assigns-csp* *xs vs* \Rightarrow *CONST AssignsCSP* (-*mk-usubst* (*CONST id*) *xs vs*)
 -*assigns-csp* *x v* \Leftarrow *CONST AssignsCSP* (*CONST subst-upd* (*CONST id*) *x v*)
 -*assigns-csp* *x v* \Leftarrow -*assigns-csp* (-*spvar* *x*) *v*
 $x, y :=_C u, v \Leftarrow \text{CONST AssignsCSP} (\text{CONST subst-upd} (\text{CONST subst-upd} (\text{CONST id}) (\text{CONST svar } x) u) (\text{CONST svar } y) v)$

lemma *preR-AssignsCSP* [*rdes*]: $\text{pre}_R(\langle \sigma \rangle_C) = \text{true}_r$
 by (*rel-auto*)

lemma *periR-AssignsCSP* [*rdes*]: $\text{peri}_R(\langle \sigma \rangle_C) = \text{false}$
 by (*rel-auto*)

lemma *postR-AssignsCSP* [*rdes*]: $\text{post}_R(\langle \sigma \rangle_C) = \Phi(\text{true}, \sigma, \langle \cdot \rangle)$
 by (*rel-auto*)

lemma *AssignsCSP-rdes-def* [*rdes-def*]: $\langle \sigma \rangle_C = \mathbf{R}_s(\text{true}_r \vdash \text{false} \diamond \Phi(\text{true}, \sigma, \langle \cdot \rangle))$
 by (*rel-auto*)

lemma *AssignsCSP-CSP* [*closure*]: $\langle \sigma \rangle_C$ is *CSP*
 by (*simp add: AssignsCSP-def RHS-tri-design-is-SRD unrest*)

lemma *AssignsCSP-CSP3* [*closure*]: $\langle \sigma \rangle_C$ is *CSP3*
 by (*rule CSP3-intro, simp add: closure, rel-auto*)

lemma *AssignsCSP-CSP4* [*closure*]: $\langle \sigma \rangle_C$ is *CSP4*
 by (*rule CSP4-intro, simp add: closure, rel-auto+*)

lemma *AssignsCSP-NCSP* [*closure*]: $\langle \sigma \rangle_C$ is *NCSP*
 by (*simp add: AssignsCSP-CSP AssignsCSP-CSP3 AssignsCSP-CSP4 NCSP-intro*)

lemma *AssignsCSP-ICSP* [*closure*]: $\langle \sigma \rangle_C$ is *ICSP*
 apply (*rule ICSP-intro, simp add: closure, simp add: rdes-def*)
 apply (*rule ISRD1-rdes-intro*)
 apply (*simp-all add: closure*)
 apply (*rel-auto*)
 done

8.5 Assignment with update

There are different collections that we would like to assign to, but they all have different types and perhaps more importantly different conditions on the update being well defined. For example, for a list well-definedness equates to the index being less than the length etc. Thus we here set up a polymorphic constant for CSP assignment updates that can be specialised to different types.

definition *AssignCSP-update* ::

$(f \Rightarrow k \text{ set}) \Rightarrow (f \Rightarrow k \Rightarrow v \Rightarrow f) \Rightarrow (f \Rightarrow \sigma) \Rightarrow$
 $(k, \sigma) \text{ uexpr} \Rightarrow (v, \sigma) \text{ uexpr} \Rightarrow (\sigma, \varphi) \text{ action where}$
 $[upred-defs, rdes-def]: \text{AssignCSP-update domf updatef } x \ k \ v =$
 $\mathbf{R}_s([k \in_u \text{uop domf } (\&x)]_{S<} \vdash \text{false} \diamond \Phi(\text{true}, [x \mapsto_s \text{trop updatef } (\&x) \ k \ v], \langle \rangle))$

All different assignment updates have the same syntax; the type resolves which implementation to use.

syntax

$\text{-csp-assign-upd} :: \text{svid} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic} \ (-[-] :=_C - [0, 0, 72] \ 72)$

translations

$x[k] :=_C v == \text{CONST AssignCSP-update } (\text{CONST udom}) (\text{CONST uupd}) \ x \ k \ v$

lemma *AssignCSP-update-CSP [closure]:*

AssignCSP-update domf updatef x k v is CSP

by (*simp add: AssignCSP-update-def RHS-tri-design-is-SRD unrest*)

lemma *preR-AssignCSP-update [rdes]:*

pre_R(AssignCSP-update domf updatef x k v) = [k ∈_u uop domf (&x)]_{S<}

by (*rel-auto*)

lemma *periR-AssignCSP-update [rdes]:*

peri_R(AssignCSP-update domf updatef x k v) = [k ∉_u uop domf (&x)]_{S<}

by (*rel-simp*)

lemma *post-AssignCSP-update [rdes]:*

post_R(AssignCSP-update domf updatef x k v) =

(Φ(true, [x ↦_s trop updatef (&x) k v], ⟨⟩) ◁ k ∈_u uop domf (&x) ▷_R R1(true))

by (*rel-auto*)

lemma *AssignCSP-update-NCSP [closure]:*

(AssignCSP-update domf updatef x k v) is NCSP

proof (*rule NCSP-intro*)

show *(AssignCSP-update domf updatef x k v) is CSP*

by (*simp add: closure*)

show *(AssignCSP-update domf updatef x k v) is CSP3*

by (*rule CSP3-SRD-intro, simp-all add: csp-do-def closure rdes unrest*)

show *(AssignCSP-update domf updatef x k v) is CSP4*

by (*rule CSP4-tri-intro, simp-all add: csp-do-def closure rdes unrest, rel-auto*)

qed

8.6 State abstraction

lemma *ref-unrest-abs-st [unrest]:*

$\$ref \# P \Longrightarrow \$ref \# \langle P \rangle_S$

$\$ref' \# P \Longrightarrow \$ref' \# \langle P \rangle_S$

by (*rel-simp*)⁺

lemma *NCSP-state-srea [closure]: P is NCSP ⇒ state 'a · P is NCSP*

apply (*rule NCSP-NSRD-intro*)

apply (*simp-all add: closure rdes*)

apply (*simp-all add: state-srea-def unrest closure*)

done

8.7 Assumptions

definition *AssumeCircus* ($\{-\}_C$) **where**
 $[rdes-def]: \{b\}_C = \mathbf{R}_s(\mathcal{I}(b, \langle \rangle)) \vdash (false \diamond \Phi(true, id, \langle \rangle))$

8.8 Guards

definition *GuardCSP* ::

$'\sigma \text{ cond} \Rightarrow$
 $('\sigma, '\varphi) \text{ action} \Rightarrow$
 $('\sigma, '\varphi) \text{ action} \text{ (infixr } \&_u \text{ 70) where}$
 $[upred-defs]: g \&_u A = \mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r pre_R(A)) \vdash ((\lceil g \rceil_{S<} \wedge cmt_R(A)) \vee (\lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$

lemma *Guard-tri-design*:

$g \&_u P = \mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r pre_R P) \vdash (peri_R(P) \triangleleft \lceil g \rceil_{S<} \triangleright (\$tr' =_u \$tr)) \diamond (\lceil g \rceil_{S<} \wedge post_R(P)))$
proof –
have $(\lceil g \rceil_{S<} \wedge cmt_R P \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait') = (peri_R(P) \triangleleft \lceil g \rceil_{S<} \triangleright (\$tr' =_u \$tr)) \diamond (\lceil g \rceil_{S<} \wedge post_R(P))$
by (*rel-auto*)
thus *?thesis* **by** (*simp add: GuardCSP-def*)
qed

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

assumes $P \text{ is } RR \ Q \text{ is } RR \ R \text{ is } RR$
shows $g \&_u \mathbf{R}_s(P \vdash Q \diamond R) = \mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r P) \vdash (Q \triangleleft g \triangleright_R (\$tr' =_u \$tr)) \diamond (\lceil g \rceil_{S<} \wedge R))$
by (*simp add: Guard-tri-design rdes assms, rel-auto*)

lemma *Guard-rdes-def'*:

assumes $\$ok' \nmid P$
shows $g \&_u (\mathbf{R}_s(P \vdash Q)) = \mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r P) \vdash (\lceil g \rceil_{S<} \wedge Q \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
proof –
have $g \&_u (\mathbf{R}_s(P \vdash Q)) = \mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r pre_R (\mathbf{R}_s(P \vdash Q))) \vdash (\lceil g \rceil_{S<} \wedge cmt_R (\mathbf{R}_s(P \vdash Q)) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
by (*simp add: GuardCSP-def*)
also have $\dots = \mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r R1(R2c(pre_s \dagger P))) \vdash (\lceil g \rceil_{S<} \wedge R1(R2c(cmt_s \dagger (P \Rightarrow Q))) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
by (*simp add: rea-pre-RHS-design rea-cmt-RHS-design*)
also have $\dots = \mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r R1(R2c(pre_s \dagger P))) \vdash R1(R2c(\lceil g \rceil_{S<} \wedge R1(R2c(cmt_s \dagger (P \Rightarrow Q)))) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
by (*metis (no-types, lifting) RHS-design-export-R1 RHS-design-export-R2c*)
also have $\dots = \mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r R1(R2c(pre_s \dagger P))) \vdash R1(R2c(\lceil g \rceil_{S<} \wedge (cmt_s \dagger (P \Rightarrow Q)) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
by (*simp add: R1-R2c-commute R1-disj R1-extend-conj' R1-idem R2c-and R2c-disj R2c-idem*)
also have $\dots = \mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r R1(R2c(pre_s \dagger P))) \vdash (\lceil g \rceil_{S<} \wedge (cmt_s \dagger (P \Rightarrow Q)) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
by (*metis (no-types, lifting) RHS-design-export-R1 RHS-design-export-R2c*)
also have $\dots = \mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r R1(R2c(pre_s \dagger P))) \vdash cmt_s \dagger (\lceil g \rceil_{S<} \wedge (cmt_s \dagger (P \Rightarrow Q)) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
by (*simp add: rdes-export-cmt*)
also have $\dots = \mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r R1(R2c(pre_s \dagger P))) \vdash cmt_s \dagger (\lceil g \rceil_{S<} \wedge (P \Rightarrow Q) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
by (*simp add: usubst*)
also have $\dots = \mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r R1(R2c(pre_s \dagger P))) \vdash (\lceil g \rceil_{S<} \wedge (P \Rightarrow Q) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
by (*simp add: rdes-export-cmt*)

also from *assms* **have** ... = $\mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r (pre_s \dagger P)) \vdash (\lceil g \rceil_{S<} \wedge (P \Rightarrow Q) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
by (*rel-auto*)
also have ... = $\mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r pre_s \dagger P) \llbracket true, false / \$ok, \$wait \rrbracket \vdash (\lceil g \rceil_{S<} \wedge (P \Rightarrow Q) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
by (*simp add: rdes-export-pre*)
also from *assms* **have** ... = $\mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r P) \llbracket true, false / \$ok, \$wait \rrbracket \vdash (\lceil g \rceil_{S<} \wedge (P \Rightarrow Q) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
by (*rel-auto*)
also from *assms* **have** ... = $\mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r P) \vdash (\lceil g \rceil_{S<} \wedge (P \Rightarrow Q) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
by (*simp add: rdes-export-pre*)
also have ... = $\mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r P) \vdash (\lceil g \rceil_{S<} \wedge Q \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
by (*rule cong[of \mathbf{R}_s \mathbf{R}_s], simp, rel-auto*)
finally show ?thesis .
qed

lemma *CSP-Guard [closure]*: $b \&_u P$ is CSP

by (*simp add: GuardCSP-def, rule RHS-design-is-SRD, simp-all add: unrest*)

lemma *preR-Guard [rdes]*: P is CSP $\implies pre_R(b \&_u P) = (\lceil b \rceil_{S<} \Rightarrow_r pre_R P)$

by (*simp add: Guard-tri-design rea-pre-RHS-design usubst unrest R2c-preR R2c-lift-state-pre R2c-rea-impl R1-rea-impl R1-preR Healthy-if, rel-auto*)

lemma *periR-Guard [rdes]*:

assumes P is NCSP

shows $peri_R(b \&_u P) = (peri_R P \triangleleft b \triangleright_R \mathcal{E}(true, \langle \rangle, \{ \}_u))$

proof –

have $peri_R(b \&_u P) = ((\lceil b \rceil_{S<} \Rightarrow_r pre_R P) \Rightarrow_r (peri_R P \triangleleft \lceil b \rceil_{S<} \triangleright (\$tr' =_u \$tr)))$

by (*simp add: assms Guard-tri-design rea-peri-RHS-design usubst unrest R1-rea-impl R2c-rea-not R2c-rea-impl R2c-preR R2c-periR R2c-tr'-minus-tr R2c-lift-state-pre R2c-condr closure Healthy-if R1-cond R1-tr'-eq-tr*)

also have ... = $((pre_R P \Rightarrow_r peri_R P) \triangleleft \lceil b \rceil_{S<} \triangleright (\$tr' =_u \$tr))$

by (*rel-auto*)

also have ... = $(peri_R P \triangleleft \lceil b \rceil_{S<} \triangleright (\$tr' =_u \$tr))$

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

finally show ?thesis

by *rel-auto*

qed

lemma *postR-Guard [rdes]*:

assumes P is NCSP

shows $post_R(b \&_u P) = (\lceil b \rceil_{S<} \wedge post_R P)$

proof –

have $post_R(b \&_u P) = ((\lceil b \rceil_{S<} \Rightarrow_r pre_R P) \Rightarrow_r (\lceil b \rceil_{S<} \wedge post_R P))$

by (*simp add: Guard-tri-design rea-post-RHS-design usubst unrest R2c-rea-not R2c-and R2c-rea-impl R2c-preR R2c-postR R2c-tr'-minus-tr R2c-lift-state-pre R2c-condr R1-rea-impl R1-extend-conj' R1-post-SRD closure assms*)

also have ... = $(\lceil b \rceil_{S<} \wedge (pre_R P \Rightarrow_r post_R P))$

by (*rel-auto*)

also have ... = $(\lceil b \rceil_{S<} \wedge post_R P)$

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

also have ... = $(\lceil b \rceil_{S<} \wedge post_R P)$

by (*metis CSP-Guard R1-extend-conj R1-post-SRD calculation rea-st-cond-def*)

finally show ?thesis .

qed

lemma *CSP3-Guard* [closure]:
 assumes *P is CSP P is CSP3*
 shows *b &_u P is CSP3*

proof –

from *assms* have *1:\$ref # P[false/\$wait]*
 by (*simp add: CSP-Guard CSP3-iff*)
 hence *\$ref # pre_R (P[0/\$tr]) \$ref # pre_R P \$ref # cmt_R P*
 by (*pred-blast*)+
 hence *\$ref # (b &_u P)[false/\$wait]*
 by (*simp add: CSP3-iff GuardCSP-def RHS-def R1-def R2c-def R2s-def R3h-def design-def unrest*
usubst)
 thus ?thesis
 by (*metis CSP3-intro CSP-Guard*)

qed

lemma *CSP4-Guard* [closure]:
 assumes *P is NCSP*
 shows *b &_u P is CSP4*

proof (*rule CSP4-tri-intro[OF CSP-Guard]*)

show $(\neg_r \text{pre}_R (b \&_u P)) \;; R1 \text{ true} = (\neg_r \text{pre}_R (b \&_u P))$

proof –

have *a:($\neg_r \text{pre}_R P$) ;; R1 true = ($\neg_r \text{pre}_R P$)*
 by (*simp add: CSP4-neg-pre-unit assms closure*)
 have $(\neg_r ([b]_{S<} \Rightarrow_r \text{pre}_R P)) \;; R1 \text{ true} = (\neg_r ([b]_{S<} \Rightarrow_r \text{pre}_R P))$

proof –

have *1:($\neg_r ([b]_{S<} \Rightarrow_r \text{pre}_R P) = ([b]_{S<} \wedge (\neg_r \text{pre}_R P))$)*
 by (*rel-auto*)
 also have *2:... = ([b]_{S<} ∧ (($\neg_r \text{pre}_R P$) ;; R1 true))*
 by (*simp add: a*)
 also have *3:... = ($\neg_r ([b]_{S<} \Rightarrow_r \text{pre}_R P) \;; R1 \text{ true}$)*
 by (*rel-auto*)
 finally show ?thesis ..

qed

thus ?thesis

by (*simp add: preR-Guard periR-Guard NSRD-CSP4-intro closure assms unrest*)

qed

show *\$st' # peri_R (b &_u P)*

by (*simp add: preR-Guard periR-Guard NSRD-CSP4-intro closure assms unrest*)

show *\$ref' # post_R (b &_u P)*

by (*simp add: preR-Guard postR-Guard NSRD-CSP4-intro closure assms unrest*)

qed

lemma *NCSP-Guard* [closure]:
 assumes *P is NCSP*
 shows *b &_u P is NCSP*

proof –

have *P is CSP*

using *NCSP-implies-CSP assms* by *blast*

then show ?thesis

by (*metis (no-types) CSP3-Guard CSP3-commutes-CSP4 CSP4-Guard CSP4-Idempotent CSP-Guard*
Healthy-Idempotent Healthy-def NCSP-def assms comp-apply)

qed

lemma *Productive-Guard* [closure]:

assumes P is CSP P is Productive $\$wait' \# pre_R(P)$

shows $b \&_u P$ is Productive

proof –

have $b \&_u P = b \&_u \mathbf{R}_s(pre_R(P) \vdash peri_R(P) \diamond (post_R(P) \wedge \$tr <_u \$tr'))$

by (*metis Healthy-def Productive-form assms(1) assms(2)*)

also have ... =

$\mathbf{R}_s((\lceil b \rceil_{S<} \Rightarrow_r pre_R P) \vdash ((pre_R P \Rightarrow_r peri_R P) \triangleleft \lceil b \rceil_{S<} \triangleright (\$tr' =_u \$tr)) \diamond (\lceil b \rceil_{S<} \wedge (pre_R P \Rightarrow_r post_R P \wedge \$tr' >_u \$tr)))$

by (*simp add: Guard-tri-design rea-pre-RHS-design rea-peri-RHS-design rea-post-RHS-design unrest assms*)

usubst R1-preR Healthy-if R1-rea-impl R1-peri-SRD R1-extend-conj' R2c-preR R2c-not R2c-rea-impl

R2c-periR R2c-postR R2c-and R2c-tr-less-tr' R1-tr-less-tr'

also have ... = $\mathbf{R}_s((\lceil b \rceil_{S<} \Rightarrow_r pre_R P) \vdash (peri_R P \triangleleft \lceil b \rceil_{S<} \triangleright (\$tr' =_u \$tr)) \diamond ((\lceil b \rceil_{S<} \wedge post_R P) \wedge \$tr' >_u \$tr))$

by (*rel-auto*)

also have ... = *Productive*($b \&_u P$)

by (*simp add: Productive-def Guard-tri-design RHS-tri-design-par unrest*)

finally show *?thesis*

by (*simp add: Healthy-def'*)

qed

8.9 Basic events

definition $do_u ::$

$(\varphi, \sigma) uexpr \Rightarrow (\sigma, \varphi)$ action **where**

[*upred-defs*]: $do_u e = ((\$tr' =_u \$tr \wedge \lceil e \rceil_{S<} \notin_u \$ref') \triangleleft \$wait' \triangleright (\$tr' =_u \$tr \hat{\wedge}_u \langle \lceil e \rceil_{S<} \rangle \wedge \$st' =_u \$st))$

definition $DoCSP :: (\varphi, \sigma) uexpr \Rightarrow (\sigma, \varphi)$ action (do_C) **where**

[*upred-defs*]: $DoCSP a = \mathbf{R}_s(true \vdash do_u a)$

lemma *R1-DoAct*: $R1(do_u(a)) = do_u(a)$

by (*rel-auto*)

lemma *R2c-DoAct*: $R2c(do_u(a)) = do_u(a)$

by (*rel-auto*)

lemma *DoCSP-alt-def*: $do_C(a) = R3h(CSP1(\$ok' \wedge do_u(a)))$

apply (*simp add: DoCSP-def RHS-def design-def impl-alt-def R1-R3h-commute R2c-R3h-commute R2c-disj*)

R2c-not R2c-ok R2c-ok' R2c-and R2c-DoAct R1-disj R1-extend-conj' R1-DoAct

apply (*rel-auto*)

done

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

$\$ok \# do_u(a) \$wait \# do_u(a)$

by (*pred-auto*)⁺

lemma *DoCSP-RHS-tri* [*rdes-def*]:

$do_C(a) = \mathbf{R}_s(true_r \vdash (\mathcal{E}(true, \langle \rangle, \{a\}_u) \diamond \Phi(true, id, \langle a \rangle)))$

by (*simp add: DoCSP-def do_u-def wait'-cond-def, rel-auto*)

lemma *CSP-DoCSP* [closure]: $do_C(a)$ is CSP

by (simp add: DoCSP-def do_u-def RHS-design-is-SRD unrest)

lemma *preR-DoCSP* [rdes]: $\text{pre}_R(\text{do}_C(a)) = \text{true}_r$
 by (simp add: DoCSP-def rea-pre-RHS-design unrest usubst R2c-true)

lemma *periR-DoCSP* [rdes]: $\text{peri}_R(\text{do}_C(a)) = \mathcal{E}(\text{true}, \langle \rangle, \{a\}_u)$
 by (rel-auto)

lemma *postR-DoCSP* [rdes]: $\text{post}_R(\text{do}_C(a)) = \Phi(\text{true}, \text{id}, \langle a \rangle)$
 by (rel-auto)

lemma *CSP3-DoCSP* [closure]: $\text{do}_C(a)$ is CSP3
 by (rule CSP3-intro[OF CSP-DoCSP])
 (simp add: DoCSP-def do_u-def RHS-def design-def R1-def R2c-def R2s-def R3h-def unrest usubst)

lemma *CSP4-DoCSP* [closure]: $\text{do}_C(a)$ is CSP4
 by (rule CSP4-tri-intro[OF CSP-DoCSP], simp-all add: preR-DoCSP periR-DoCSP postR-DoCSP unrest)

lemma *NCSP-DoCSP* [closure]: $\text{do}_C(a)$ is NCSP
 by (metis CSP3-DoCSP CSP4-DoCSP CSP-DoCSP Healthy-def NCSP-def comp-apply)

lemma *Productive-DoCSP* [closure]:
 ($\text{do}_C a :: ('σ, 'ψ) \text{ action}$) is Productive
proof –
 have $((\Phi(\text{true}, \text{id}, \langle a \rangle) \wedge \$tr' >_u \$tr) :: ('σ, 'ψ) \text{ action})$
 $= (\Phi(\text{true}, \text{id}, \langle a \rangle))$
 by (rel-auto, simp add: Prefix-Order.strict-prefixI')
 hence $\text{Productive}(\text{do}_C a) = \text{do}_C a$
 by (simp add: Productive-RHS-design-form DoCSP-RHS-tri unrest)
 thus ?thesis
 by (simp add: Healthy-def)
qed

lemma *PCSP-DoCSP* [closure]:
 ($\text{do}_C a :: ('σ, 'ψ) \text{ action}$) is PCSP
 by (simp add: Healthy-comp NCSP-DoCSP Productive-DoCSP)

lemma *wp-rea-DoCSP-lemma*:
 fixes $P :: ('σ, 'φ) \text{ action}$
 assumes $\$ok \# P \ \$wait \# P$
 shows $(\$tr' =_u \$tr \wedge_u \langle [a]_{S<} \rangle \wedge \$st' =_u \$st) ;; P = (\exists \$ref \cdot P[\$tr \wedge_u \langle [a]_{S<} \rangle / \$tr])$
 using *assms*
 by (rel-auto, meson)

lemma *wp-rea-DoCSP*:
 assumes P is NCSP
 shows $(\$tr' =_u \$tr \wedge_u \langle [a]_{S<} \rangle \wedge \$st' =_u \$st) \text{ wp}_r \text{ pre}_R P =$
 $(\neg_r (\neg_r \text{pre}_R P)[\$tr \wedge_u \langle [a]_{S<} \rangle / \$tr])$
 by (simp add: wp-rea-def wp-rea-DoCSP-lemma unrest usubst ex-unrest assms closure)

lemma *wp-rea-DoCSP-alt*:
 assumes P is NCSP
 shows $(\$tr' =_u \$tr \wedge_u \langle [a]_{S<} \rangle \wedge \$st' =_u \$st) \text{ wp}_r \text{ pre}_R P =$
 $(\$tr' \geq_u \$tr \wedge_u \langle [a]_{S<} \rangle \Rightarrow_r (\text{pre}_R P)[\$tr \wedge_u \langle [a]_{S<} \rangle / \$tr])$

by (simp add: wp-rea-DoCSP assms rea-not-def R1-def usubst unrest, rel-auto)

8.10 Event prefix

definition *PrefixCSP* ::

(φ, σ) *uexpr* \Rightarrow
 (σ, φ) *action* \Rightarrow
 (σ, φ) *action* ($- \rightarrow_C - [81, 80] 80$) **where**

[upred-defs]: *PrefixCSP* $a \ P = (do_C(a) ;; CSP(P))$

abbreviation *OutputCSP* $c \ v \ P \equiv PrefixCSP \ (c.v)_u \ P$

lemma *CSP-PrefixCSP [closure]*: *PrefixCSP* $a \ P$ is *CSP*

by (simp add: *PrefixCSP-def closure*)

lemma *CSP3-PrefixCSP [closure]*:

PrefixCSP $a \ P$ is *CSP3*

by (metis (no-types, hide-lams) *CSP3-DoCSP CSP3-def Healthy-def PrefixCSP-def seqr-assoc*)

lemma *CSP4-PrefixCSP [closure]*:

assumes P is *CSP* P is *CSP4*

shows *PrefixCSP* $a \ P$ is *CSP4*

by (metis (no-types, hide-lams) *CSP4-def Healthy-def PrefixCSP-def assms(1) assms(2) seqr-assoc*)

lemma *NCSP-PrefixCSP [closure]*:

assumes P is *NCSP*

shows *PrefixCSP* $a \ P$ is *NCSP*

by (metis (no-types, hide-lams) *CSP3-PrefixCSP CSP3-commutes-CSP4 CSP4-Idempotent CSP4-PrefixCSP CSP-PrefixCSP Healthy-Idempotent Healthy-def NCSP-def NCSP-implies-CSP assms comp-apply*)

lemma *Productive-PrefixCSP [closure]*: P is *NCSP* $\implies PrefixCSP \ a \ P$ is *Productive*

by (simp add: *Healthy-if NCSP-DoCSP NCSP-implies-NSRD NSRD-is-SRD PrefixCSP-def Productive-DoCSP Productive-seq-1*)

lemma *PCSP-PrefixCSP [closure]*: P is *NCSP* $\implies PrefixCSP \ a \ P$ is *PCSP*

by (simp add: *Healthy-comp NCSP-PrefixCSP Productive-PrefixCSP*)

lemma *PrefixCSP-Guarded [closure]*: *Guarded* (*PrefixCSP* a)

proof –

have *PrefixCSP* $a = (\lambda X. do_C(a) ;; CSP(X))$

by (simp add: *fun-eq-iff PrefixCSP-def*)

thus ?thesis

using *Guarded-if-Productive NCSP-DoCSP NCSP-implies-NSRD Productive-DoCSP* **by** auto
 qed

lemma *PrefixCSP-type [closure]*: *PrefixCSP* $a \in \llbracket H \rrbracket_H \rightarrow \llbracket CSP \rrbracket_H$

using *CSP-PrefixCSP* **by** blast

lemma *PrefixCSP-Continuous [closure]*: *Continuous* (*PrefixCSP* a)

by (simp add: *Continuous-def PrefixCSP-def ContinuousD[OF SRD-Continuous] seq-Sup-distl*)

lemma *PrefixCSP-RHS-tri-lemma1*:

$R1 \ (R2s \ (\$tr' =_u \$tr \hat{^}_u \langle [a]_{S<} \rangle \wedge [II]_R)) = (\$tr' =_u \$tr \hat{^}_u \langle [a]_{S<} \rangle \wedge [II]_R)$

by (rel-auto)

lemma *PrefixCSP-RHS-tri-lemma2*:

fixes $P :: ('σ, 'φ) \text{ action}$
assumes $\$ok \# P \$wait \# P$
shows $((\$tr' =_u \$tr \hat{\ }_u \langle [a]_{S<} \rangle \wedge \$st' =_u \$st) \wedge \neg \$wait') ;; P = (\exists \$ref \cdot P[\$tr \hat{\ }_u \langle [a]_{S<} \rangle / \$tr])$
using *assms*
by (*rel-auto, meson, fastforce*)

lemma *tr-extend-seqr*:

fixes $P :: ('σ, 'φ) \text{ action}$
assumes $\$ok \# P \$wait \# P \$ref \# P$
shows $(\$tr' =_u \$tr \hat{\ }_u \langle [a]_{S<} \rangle \wedge \$st' =_u \$st) ;; P = P[\$tr \hat{\ }_u \langle [a]_{S<} \rangle / \$tr]$
using *assms* **by** (*simp add: wp-rea-DoCSP-lemma assms unrest ex-unrest*)

lemma *trace-ext-R1-closed [closure]*: $P \text{ is } R1 \implies P[\$tr \hat{\ }_u e / \$tr] \text{ is } R1$
by (*rel-blast*)

lemma *preR-PrefixCSP-NCSP [rdes]*:

assumes $P \text{ is } NCSP$
shows $pre_R(PrefixCSP \ a \ P) = (\mathcal{I}(true, \langle a \rangle) \Rightarrow_r (pre_R \ P)[\langle a \rangle]_t)$
by (*simp add: PrefixCSP-def assms closure rdes rpred Healthy-if wp usubst unrest*)

lemma *periR-PrefixCSP [rdes]*:

assumes $P \text{ is } NCSP$
shows $peri_R(PrefixCSP \ a \ P) = (\mathcal{E}(true, \langle \rangle, \{a\}_u) \vee (peri_R \ P)[\langle a \rangle]_t)$

proof –

have $peri_R(PrefixCSP \ a \ P) = peri_R(do_C \ a \ ; \ P)$
by (*simp add: PrefixCSP-def closure assms Healthy-if*)
also have $\dots = ((\mathcal{I}(true, \langle a \rangle) \Rightarrow_r pre_R \ P[\langle a \rangle]_t) \Rightarrow_r \$tr' =_u \$tr \wedge [a]_{S<} \notin_u \$ref' \vee peri_R \ P[\langle a \rangle]_t)$
by (*simp add: assms NSRD-CSP4-intro csp-enable-tr-empty closure rdes unrest ex-unrest usubst rpred wp*)
also have $\dots = (\mathcal{E}(true, \langle \rangle, \{a\}_u) \vee ((\mathcal{I}(true, \langle a \rangle) \Rightarrow_r pre_R \ P[\langle a \rangle]_t) \Rightarrow_r peri_R \ P[\langle a \rangle]_t))$
by (*rel-auto*)
also have $\dots = (\mathcal{E}(true, \langle \rangle, \{a\}_u) \vee ((pre_R(P) \Rightarrow_r peri_R \ P)[\langle a \rangle]_t))$
by (*rel-auto*)
also have $\dots = (\mathcal{E}(true, \langle \rangle, \{a\}_u) \vee (peri_R \ P)[\langle a \rangle]_t)$
by (*simp add: SRD-peri-under-pre assms closure unrest*)
finally show *?thesis* .

qed

lemma *postR-PrefixCSP [rdes]*:

assumes $P \text{ is } NCSP$
shows $post_R(PrefixCSP \ a \ P) = (post_R \ P)[\langle a \rangle]_t$

proof –

have $post_R(PrefixCSP \ a \ P) = ((\mathcal{I}(true, \langle a \rangle) \Rightarrow_r (pre_R \ P)[\langle a \rangle]_t) \Rightarrow_r (post_R \ P)[\langle a \rangle]_t)$
by (*simp add: PrefixCSP-def assms Healthy-if*)
(simp add: assms Healthy-if wp closure rdes rpred wp-rea-DoCSP-lemma unrest ex-unrest usubst)
also have $\dots = (\mathcal{I}(true, \langle a \rangle) \wedge (pre_R \ P \Rightarrow_r post_R \ P)[\langle a \rangle]_t)$
by (*rel-auto*)
also have $\dots = (\mathcal{I}(true, \langle a \rangle) \wedge (post_R \ P)[\langle a \rangle]_t)$
by (*simp add: SRD-post-under-pre assms closure unrest*)
also have $\dots = (post_R \ P)[\langle a \rangle]_t$
by (*rel-auto*)
finally show *?thesis* .

qed

lemma *PrefixCSP-RHS-tri*:

assumes P is NCSP
shows $\text{PrefixCSP } a \ P = \mathbf{R}_s ((\mathcal{I}(\text{true}, \langle a \rangle) \Rightarrow_r \text{pre}_R P \llbracket \langle a \rangle \rrbracket_t) \vdash (\mathcal{E}(\text{true}, \langle \rangle, \{a\}_u) \vee \text{peri}_R P \llbracket \langle a \rangle \rrbracket_t) \diamond \text{post}_R P \llbracket \langle a \rangle \rrbracket_t)$
by (*simp add: PrefixCSP-def Healthy-if unrest assms closure NSRD-composition-wp rdes rpred usubst wp*)

For prefix, we can chose whether to propagate the assumptions or not, hence there are two laws.

lemma *PrefixCSP-rdes-def-1* [rdes-def]:

assumes P is CRC Q is CRR R is CRR
 $\$st' \# Q \$ref' \# R$
shows $\text{PrefixCSP } a \ (\mathbf{R}_s(P \vdash Q \diamond R)) = \mathbf{R}_s((\mathcal{I}(\text{true}, \langle a \rangle) \Rightarrow_r P \llbracket \langle a \rangle \rrbracket_t) \vdash (\mathcal{E}(\text{true}, \langle \rangle, \{a\}_u) \vee Q \llbracket \langle a \rangle \rrbracket_t) \diamond R \llbracket \langle a \rangle \rrbracket_t)$
apply (*subst PrefixCSP-RHS-tri*)
apply (*rule NCSP-rdes-intro*)
apply (*simp-all add: assms rdes closure*)
apply (*rel-auto*)
done

lemma *PrefixCSP-rdes-def-2*:

assumes P is CRC Q is CRR R is CRR
 $\$st' \# Q \$ref' \# R$
shows $\text{PrefixCSP } a \ (\mathbf{R}_s(P \vdash Q \diamond R)) = \mathbf{R}_s((\mathcal{I}(\text{true}, \langle a \rangle) \Rightarrow_r P \llbracket \langle a \rangle \rrbracket_t) \vdash (\mathcal{E}(\text{true}, \langle \rangle, \{a\}_u) \vee (P \wedge Q) \llbracket \langle a \rangle \rrbracket_t) \diamond (P \wedge R) \llbracket \langle a \rangle \rrbracket_t)$
apply (*subst PrefixCSP-RHS-tri*)
apply (*rule NCSP-rdes-intro*)
apply (*simp-all add: assms rdes closure*)
apply (*rel-auto*)
done

8.11 Guarded external choice

abbreviation *GuardedChoiceCSP* :: $'\vartheta$ set $\Rightarrow (' \vartheta \Rightarrow (' \sigma, ' \vartheta)$ action) $\Rightarrow (' \sigma, ' \vartheta)$ action **where**
 $\text{GuardedChoiceCSP } A \ P \equiv (\Box x \in A \cdot \text{PrefixCSP } \llbracket x \rrbracket (P(x)))$

syntax

-GuardedChoiceCSP :: *logic* \Rightarrow *logic* \Rightarrow *logic* \Rightarrow *logic* (\Box - \in - \rightarrow - [0,0,85] 86)

translations

$\Box x \in A \rightarrow P == \text{CONST } \text{GuardedChoiceCSP } A \ (\lambda x. P)$

lemma *GuardedChoiceCSP* [rdes-def]:

assumes $\bigwedge x. P(x)$ is NCSP $A \neq \{\}$
shows $(\Box x \in A \rightarrow P(x)) =$
 $\mathbf{R}_s ((\Box x \in A \cdot \mathcal{I}(\text{true}, \langle \llbracket x \rrbracket \rangle)) \Rightarrow_r \text{pre}_R (P \ x) \llbracket \langle \llbracket x \rrbracket \rangle \rrbracket_t) \vdash$
 $((\Box x \in A \cdot \mathcal{E}(\text{true}, \langle \rangle, \{\llbracket x \rrbracket\}_u)) \triangleleft \$tr' =_u \$tr \triangleright (\Box x \in A \cdot \text{peri}_R (P \ x) \llbracket \langle \llbracket x \rrbracket \rangle \rrbracket_t)) \diamond$
 $(\Box x \in A \cdot \text{post}_R (P \ x) \llbracket \langle \llbracket x \rrbracket \rangle \rrbracket_t)$
by (*simp add: PrefixCSP-RHS-tri assms ExtChoice-tri-rdes closure unrest, rel-auto*)

8.12 Input prefix

definition *InputCSP* ::

$('a, ' \vartheta)$ chan $\Rightarrow ('a \Rightarrow ' \sigma \text{ upred}) \Rightarrow ('a \Rightarrow (' \sigma, ' \vartheta)$ action) $\Rightarrow (' \sigma, ' \vartheta)$ action **where**
[upred-defs]: $\text{InputCSP } c \ A \ P = (\Box x \in \text{UNIV} \cdot A(x) \ \&_u \ \text{PrefixCSP } (c \cdot \llbracket x \rrbracket)_u (P \ x))$

definition *InputVarCSP* :: $('a, ' \vartheta)$ chan $\Rightarrow ('a \Rightarrow ' \sigma \text{ upred}) \Rightarrow ('a \Rightarrow ' \sigma) \Rightarrow (' \sigma, ' \vartheta)$ action $\Rightarrow (' \sigma, ' \vartheta)$ action **where**

$InputVarCSP\ c\ A\ x\ P = InputCSP\ c\ A\ (\lambda\ v.\ \langle [x \mapsto_s \ll v \gg] \rangle_C) \ ::\ CSP(P)$

definition $do_I ::$

$(\text{'}a, \text{'}\vartheta) \text{ chan} \Rightarrow$
 $(\text{'}a \Rightarrow (\text{'}\sigma, \text{'}\vartheta) \text{ st-csp}) \Rightarrow$
 $(\text{'}a \Rightarrow (\text{'}\sigma, \text{'}\vartheta) \text{ action}) \Rightarrow$
 $(\text{'}\sigma, \text{'}\vartheta) \text{ action where}$
 $do_I\ c\ x\ P = ($
 $(\$tr' =_u \$tr \wedge \{e : \ll \delta_u(c) \gg \mid P(e) \cdot (c \cdot \ll e \gg)_u\}_u \cap_u \$ref' =_u \{\}_u)$
 $\triangleleft \$wait' \triangleright$
 $((\$tr' - \$tr) \in_u \{e : \ll \delta_u(c) \gg \mid P(e) \cdot \langle (c \cdot \ll e \gg)_u \rangle \}_u \wedge (c \cdot \$x')_u =_u last_u(\$tr')))$

lemma $InputCSP\text{-}CSP\ [closure]: InputCSP\ c\ A\ P\ \text{is}\ CSP$
by (*simp add: CSP-ExtChoice InputCSP-def*)

lemma $InputCSP\text{-}NCSP\ [closure]: \ll \bigwedge v. P(v) \text{ is } NCSP \gg \Rightarrow InputCSP\ c\ A\ P\ \text{is } NCSP$
apply (*simp add: InputCSP-def*)
apply (*rule NCSP-ExtChoice*)
apply (*simp add: NCSP-Guard NCSP-PrefixCSP image-Collect-subsetI top-set-def*)
done

lemma $Productive\text{-}InputCSP\ [closure]:$
 $\ll \bigwedge v. P(v) \text{ is } NCSP \gg \Rightarrow InputCSP\ x\ A\ P\ \text{is } Productive$
by (*auto simp add: InputCSP-def unrest closure intro: Productive-ExtChoice*)

lemma $preR\text{-}InputCSP\ [rdes]:$
assumes $\bigwedge v. P(v) \text{ is } NCSP$
shows $pre_R(InputCSP\ a\ A\ P) = (\bigsqcup v \cdot [A(v)]_{S<} \Rightarrow_r \mathcal{I}(true, \langle (a \cdot \ll v \gg)_u \rangle) \Rightarrow_r (pre_R(P(v))) \ll \langle (a \cdot \ll v \gg)_u \rangle_t)$
by (*simp add: InputCSP-def rdes closure assms alpha usubst unrest*)

lemma $periR\text{-}InputCSP\ [rdes]:$
assumes $\bigwedge v. P(v) \text{ is } NCSP$
shows $peri_R(InputCSP\ a\ A\ P) =$
 $((\bigsqcup x \cdot [A(x)]_{S<} \Rightarrow_r \mathcal{E}(true, \langle \rangle, \{(a \cdot \ll x \gg)_u\}_u))$
 $\triangleleft \$tr' =_u \$tr \triangleright$
 $(\bigcap x \cdot [A(x)]_{S<} \wedge (peri_R(P\ x)) \ll \langle (a \cdot \ll x \gg)_u \rangle_t)$
by (*simp add: InputCSP-def rdes closure assms, rel-auto*)

lemma $postR\text{-}InputCSP\ [rdes]:$
assumes $\bigwedge v. P(v) \text{ is } NCSP$
shows $post_R(InputCSP\ a\ A\ P) =$
 $(\bigcap x \cdot [A\ x]_{S<} \wedge post_R(P\ x) \ll \langle (a \cdot \ll x \gg)_u \rangle_t)$
using *assms* **by** (*simp add: InputCSP-def rdes closure assms usubst unrest*)

lemma $InputCSP\text{-}rdes\text{-}def\ [rdes\text{-}def]:$
assumes $\bigwedge v. P(v) \text{ is } CRC \bigwedge v. Q(v) \text{ is } CRR \bigwedge v. R(v) \text{ is } CRR$
 $\bigwedge v. \$st' \nmid Q(v) \bigwedge v. \$ref' \nmid R(v)$
shows $InputCSP\ a\ A\ (\lambda v. \mathbf{R}_s(P(v) \vdash Q(v) \diamond R(v))) =$
 $\mathbf{R}_s(\ (\bigsqcup v \cdot ([A(v)]_{S<} \Rightarrow_r \mathcal{I}(true, \langle (a \cdot \ll v \gg)_u \rangle) \Rightarrow_r (P\ v)) \ll \langle (a \cdot \ll v \gg)_u \rangle_t))$
 $\vdash ((\bigsqcup x \cdot [A(x)]_{S<} \Rightarrow_r \mathcal{E}(true, \langle \rangle, \{(a \cdot \ll x \gg)_u\}_u))$
 $\triangleleft \$tr' =_u \$tr \triangleright$
 $(\bigcap x \cdot [A(x)]_{S<} \wedge (P\ x \wedge Q\ x) \ll \langle (a \cdot \ll x \gg)_u \rangle_t))$
 $\diamond (\bigcap x \cdot [A\ x]_{S<} \wedge (P\ x \wedge R\ x) \ll \langle (a \cdot \ll x \gg)_u \rangle_t))$ (**is** *?lhs = ?rhs*)

proof –

have $1: pre_R(?lhs) = (\bigsqcup v \cdot [A\ v]_{S<} \Rightarrow_r \mathcal{I}(true, \langle (a \cdot \ll v \gg)_u \rangle) \Rightarrow_r P\ v \ll \langle (a \cdot \ll v \gg)_u \rangle_t)$ (**is** *- = ?A*)

by (simp add: rdes NCSP-rdes-intro assms conj-comm closure)
 have $2: \text{peri}_R(?lhs) = (\bigsqcup x \cdot [A\ x]_{S<} \Rightarrow_r \mathcal{E}(\text{true}, \langle \rangle, \{(a \cdot \langle x \rangle)_u\}_u)) \triangleleft \$tr' =_u \$tr \triangleright (\bigsqcup x \cdot [A\ x]_{S<} \wedge (P\ x \Rightarrow_r Q\ x) \llbracket (a \cdot \langle x \rangle)_u \rrbracket_t)$
 (is - = ?B)
 by (simp add: rdes NCSP-rdes-intro assms closure)
 have $3: \text{post}_R(?lhs) = (\bigsqcup x \cdot [A\ x]_{S<} \wedge (P\ x \Rightarrow_r R\ x) \llbracket (a \cdot \langle x \rangle)_u \rrbracket_t)$
 (is - = ?C)
 by (simp add: rdes NCSP-rdes-intro assms closure)
 have $?lhs = \mathbf{R}_s(?A \vdash ?B \diamond ?C)$
 by (subst SRD-reactive-tri-design[THEN sym], simp-all add: closure 1 2 3)
 also have ... = ?rhs
 by (rel-auto)
 finally show ?thesis .
 qed

8.13 Algebraic laws

lemma *AssignCSP-conditional:*

assumes *vwb-lens* x
 shows $x :=_C e \triangleleft b \triangleright_R x :=_C f = x :=_C (e \triangleleft b \triangleright f)$
 by (rdes-eq cls: assms)

lemma *AssignsCSP-id:* $\langle id \rangle_C = \text{Skip}$

by (rel-auto)

lemma *Guard-comp:*

$g \&_u h \&_u P = (g \wedge h) \&_u P$
 by (rule antisym, rel-blast, rel-blast)

lemma *Guard-false* [simp]: $\text{false} \&_u P = \text{Stop}$

by (simp add: GuardCSP-def Stop-def rpred closure alpha R1-design-R1-pre)

lemma *Guard-true* [simp]:

$P \text{ is CSP} \implies \text{true} \&_u P = P$
 by (simp add: GuardCSP-def alpha SRD-reactive-design-alt closure rpred)

lemma *Guard-conditional:*

assumes $P \text{ is NCSP}$
 shows $b \&_u P = P \triangleleft b \triangleright_R \text{Stop}$
 by (rdes-eq cls: assms)

lemma *Guard-expansion:*

assumes $P \text{ is NCSP}$
 shows $(g_1 \vee g_2) \&_u P = (g_1 \&_u P) \sqcap (g_2 \&_u P)$
 by (rdes-eq cls: assms)

lemma *Conditional-as-Guard:*

assumes $P \text{ is NCSP}$ $Q \text{ is NCSP}$
 shows $P \triangleleft b \triangleright_R Q = b \&_u P \sqcap (\neg b) \&_u Q$
 by (rdes-eq cls: assms)

lemma *PrefixCSP-dist:*

$\text{PrefixCSP } a\ (P \sqcap Q) = (\text{PrefixCSP } a\ P) \sqcap (\text{PrefixCSP } a\ Q)$
 using Continuous-Disjunctous Disjunctuous-def PrefixCSP-Continuous by auto

lemma *DoCSP-is-Prefix:*

$do_C(a) = \text{PrefixCSP } a \text{ Skip}$
by (*simp add: PrefixCSP-def Healthy-if closure, metis CSP4-DoCSP CSP4-def Healthy-def*)

lemma *PrefixCSP-seq*:
assumes P is CSP Q is CSP
shows $(\text{PrefixCSP } a \ P) \ ;\ ;\ Q = (\text{PrefixCSP } a \ (P \ ;\ ;\ Q))$
by (*simp add: PrefixCSP-def seqr-assoc Healthy-if assms closure*)

lemma *PrefixCSP-extChoice-dist*:
assumes P is NCSP Q is NCSP R is NCSP
shows $((a \rightarrow_C P) \sqcap (b \rightarrow_C Q)) \ ;\ ;\ R = (a \rightarrow_C P \ ;\ ;\ R) \sqcap (b \rightarrow_C Q \ ;\ ;\ R)$
by (*simp add: PCSP-PrefixCSP assms(1) assms(2) assms(3) extChoice-seq-distr*)

lemma *GuardedChoiceCSP-dist*:
assumes $\bigwedge i. i \in A \implies P(i)$ is NCSP Q is NCSP
shows $\square x \in A \rightarrow P(x) \ ;\ ;\ Q = \square x \in A \rightarrow (P(x) \ ;\ ;\ Q)$
by (*simp add: ExtChoice-seq-distr PrefixCSP-seq closure assms cong: ExtChoice-cong*)

Alternation can be re-expressed as an external choice when the guards are disjoint

lemma *AlternateR-as-ExtChoice*:
assumes
 $\bigwedge i. i \in A \implies P(i)$ is NCSP Q is NCSP
 $\bigwedge i j. \llbracket i \in A; j \in A; i \neq j \rrbracket \implies (g \ i \wedge g \ j) = \text{false}$
shows $(\text{if}_R i \in A \cdot g(i) \rightarrow P(i) \text{ else } Q \text{ fi}) =$
 $(\square i \in A \cdot g(i) \ \&_u \ P(i)) \sqcap (\bigwedge i \in A \cdot \neg g(i) \ \&_u \ Q)$
proof (*cases A = {}*)
case *True*
then show *?thesis* **by** (*simp add: ExtChoice-empty extChoice-Stop closure assms*)
next
case *False*
show *?thesis*
proof –
have $1: (\bigwedge i \in A \cdot g \ i \rightarrow_R P \ i) = (\bigwedge i \in A \cdot g \ i \rightarrow_R \mathbf{R}_s(\text{pre}_R(P \ i) \vdash \text{peri}_R(P \ i) \diamond \text{post}_R(P \ i)))$
by (*rule UINF-cong, simp add: NCSP-implies-CSP SRD-reactive-tri-design assms(1)*)
have $2: (\square i \in A \cdot g(i) \ \&_u \ P(i)) = (\square i \in A \cdot g(i) \ \&_u \ \mathbf{R}_s(\text{pre}_R(P \ i) \vdash \text{peri}_R(P \ i) \diamond \text{post}_R(P \ i)))$
by (*rule ExtChoice-cong, simp add: NCSP-implies-NSRD NSRD-is-SRD SRD-reactive-tri-design assms(1)*)
from *assms(3)* **show** *?thesis*
by (*simp add: AlternateR-def 1 2*)
 $(\text{rdes-eq cls: assms(1-2) simps: False cong: UINF-cong ExtChoice-cong})$
qed
qed
end

9 Syntax and Translations for Event Prefix

theory *utp-circus-prefix*
imports *utp-circus-actions*
begin

syntax
 $\text{-simple-prefix} :: \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic} \ (- \rightarrow - \ [81, 80] \ 80)$

translations

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

We next configure a syntax for mixed prefixes.

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

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

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

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

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

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

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

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

syntax *-output-prefix* :: ('*a*', ' σ ') *uexpr* \Rightarrow *prefix-elem'* (!'(-'))

syntax *-output-prefix* :: ('*a*', ' σ ') *uexpr* \Rightarrow *prefix-elem'* ('(-'))

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

syntax

-prefix-aux :: *pttrn* \Rightarrow *logic* \Rightarrow *prefix-elem'*

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

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

syntax

-prefix-action ::

('*a*', ' ε ') *chan* \Rightarrow *mixed-prefix'* \Rightarrow (' σ ', ' ε ') *action* \Rightarrow (' σ ', ' ε ') *action*

((-- \rightarrow / -) [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* \equiv *-input-prefix* *x* *true*

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

-prefix-aux (*-pattern* *x* *y*) (($\lambda x. P$) \wedge_l *Q*)

-mixed-prefix (*-output-prefix* *P*) (*-prefix-aux* *y* *Q*) \rightarrow

-prefix-aux (*-pattern -iddummy* *y*) ((*CONST outp-constraint* *P*) \wedge_l *Q*)

-end-prefix (*-input-prefix* *x* *P*) \rightarrow *-prefix-aux* *x* ($\lambda x. P$)

-end-prefix (*-output-prefix* *P*) \rightarrow *-prefix-aux -iddummy* (*CONST outp-constraint* *P*)

-prefix-action *c* (*-prefix-aux* *x* *P*) *A* \equiv (*CONST InputCSP*) *c* *P* ($\lambda x. A$)

Basic print translations; more work needed

translations

-simple-input-prefix *x* \leq *-input-prefix* *x* *true*

-output-prefix *v* \leq *-prefix-aux* *p* (*CONST outp-constraint* *v*)

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

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

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

Basic translations for state variable communications

syntax

$\text{-csp-input-var} :: \text{logic} \Rightarrow \text{id} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic} \ (-?\$:- \rightarrow - [81, 0, 0, 80] \ 80)$
 $\text{-csp-inputu-var} :: \text{logic} \Rightarrow \text{id} \Rightarrow \text{logic} \Rightarrow \text{logic} \ (-?\$- \rightarrow - [81, 0, 80] \ 80)$

translations

$c?\$x:A \rightarrow P \equiv \text{CONST InputVarCSP } c \ x \ A \ P$
 $c?\$x \rightarrow P \rightarrow \text{CONST InputVarCSP } c \ x \ (\text{CONST UNIV}) \ P$
 $c?\$x \rightarrow P \leq c?\$x:\text{true} \rightarrow P$

lemma *outp-constraint-prod*:

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

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

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

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

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

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

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

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

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

proof –

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

thus *?thesis*

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

qed

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

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

proof –
have $(\prod y \cdot [v =_o y]_{S<} \wedge RR(P(y))) = RR(P(y))[[y \rightarrow [v]_{S\leftarrow}]]$
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 $x!(P) \rightarrow A = \text{OutputCSP } x P A$ (**is** *?lhs = ?rhs*)
by (*rule SRD-eq-intro*, *simp-all add: assms closure rdes, rel-auto+*)

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

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

end

10 Recursion in Circus

theory *utp-circus-recursion*
imports *utp-circus-prefix utp-circus-contracts*
begin

10.1 Fixed-points

The CSP weakest fixed-point is obtained simply by precomposing the body with the CSP healthiness condition.

abbreviation *mu-CSP* :: $((\sigma, \varphi) \text{ action} \Rightarrow (\sigma, \varphi) \text{ action}) \Rightarrow (\sigma, \varphi) \text{ action } (\mu_C)$ **where**
 $\mu_C F \equiv \mu (F \circ \text{CSP})$

syntax
 $\text{-mu-CSP} :: \text{pttrn} \Rightarrow \text{logic} \Rightarrow \text{logic } (\mu_C \text{ - } \cdot \text{ - } [0, 10] \text{ } 10)$

translations
 $\mu_C X \cdot P == \text{CONST } \text{mu-CSP } (\lambda X. P)$

lemma *mu-CSP-equiv*:
assumes *Monotonic F* $F \in \llbracket \text{CSP} \rrbracket_H \rightarrow \llbracket \text{CSP} \rrbracket_H$
shows $(\mu_R F) = (\mu_C F)$
by (*simp add: srd-mu-equiv assms comp-def*)

lemma *mu-CSP-unfold*:
 $P \text{ is CSP} \implies (\mu_C X \cdot P ;; X) = P ;; (\mu_C X \cdot P ;; X)$
apply (*subst gfp-unfold*)
apply (*simp-all add: closure Healthy-if*)
done

lemma *mu-csp-expand* [rdes]: $(\mu_C (op ;; Q)) = (\mu X \cdot Q ;; CSP X)$
by (*simp add: comp-def*)

lemma *mu-csp-basic-refine*:

assumes

P is CSP Q is NCSP Q is Productive $pre_R(P) = true_r$ $pre_R(Q) = true_r$

$peri_R P \sqsubseteq peri_R Q$

$peri_R P \sqsubseteq post_R Q ;; peri_R P$

shows $P \sqsubseteq (\mu_C X \cdot Q ;; X)$

proof (*rule SRD-refine-intro', simp-all add: closure usubst alpha rpred rdes unrest wp seq-UINF-distr assms*)

show $peri_R P \sqsubseteq (\bigcap i \cdot post_R Q \wedge i ;; peri_R Q)$

proof (*rule UINF-refines'*)

fix *i*

show $peri_R P \sqsubseteq post_R Q \wedge i ;; peri_R Q$

proof (*induct i*)

case 0

then show ?case **by** (*simp add: assms*)

next

case (*Suc i*)

then show ?case

by (*meson assms(6) assms(7) semilattice-sup-class.le-sup-iff upower-inductl*)

qed

qed

qed

lemma *CRD-mu-basic-refine*:

fixes *P* :: 'e list \Rightarrow 'e set \Rightarrow 's upred

assumes

Q is NCSP Q is Productive $pre_R(Q) = true_r$

$[P \ t \ r]_{S<} \llbracket (t, r) \rightarrow (\&tt, \$ref')_u \rrbracket \sqsubseteq peri_R Q$

$[P \ t \ r]_{S<} \llbracket (t, r) \rightarrow (\&tt, \$ref')_u \rrbracket \sqsubseteq post_R Q ;;_h [P \ t \ r]_{S<} \llbracket (t, r) \rightarrow (\&tt, \$ref')_u \rrbracket$

shows $[true \vdash P \ trace \ refs \mid R]_C \sqsubseteq (\mu_C X \cdot Q ;; X)$

proof (*rule mu-csp-basic-refine, simp-all add: msubst-pair assms closure alpha rdes rpred Healthy-if R1-false*)

show $[P \ trace \ refs]_{S<} \llbracket trace \rightarrow \&tt \rrbracket \llbracket refs \rightarrow \$ref' \rrbracket \sqsubseteq peri_R Q$

using *assms* **by** (*simp add: msubst-pair*)

show $[P \ trace \ refs]_{S<} \llbracket trace \rightarrow \&tt \rrbracket \llbracket refs \rightarrow \$ref' \rrbracket \sqsubseteq post_R Q ;; [P \ trace \ refs]_{S<} \llbracket trace \rightarrow \&tt \rrbracket \llbracket refs \rightarrow \$ref' \rrbracket$

using *assms* **by** (*simp add: msubst-pair*)

qed

10.2 Example action expansion

lemma *mu-example1*: $(\mu X \cdot a \rightarrow X) = (\bigcap i \cdot do_C(\llbracket a \rrbracket) \wedge (i+1)) ;; Miracle$
by (*simp add: PrefixCSP-def mu-csp-form-1 closure*)

lemma *preR-mu-example1* [rdes]: $pre_R(\mu X \cdot a \rightarrow X) = true_r$
by (*simp add: mu-example1 rdes closure unrest wp*)

lemma *periR-mu-example1* [rdes]:

$peri_R(\mu X \cdot a \rightarrow X) = (\bigcap i \cdot \mathcal{E}(true, iter[i](\llbracket a \rrbracket), \{\llbracket a \rrbracket\}_u))$

by (*simp add: mu-example1 rdes rpred closure unrest wp seq-UINF-distr alpha usubst*)

lemma *postR-mu-example1* [rdes]:

$post_R(\mu X \cdot a \rightarrow X) = false$

by (simp add: mu-example1 rdes closure unrest wp)

end

11 Circus Trace Merge

theory utp-circus-traces
imports utp-circus-core
begin

11.1 Function Definition

fun *tr-par* ::
 $\text{'}\vartheta \text{ set} \Rightarrow \text{'}\vartheta \text{ list} \Rightarrow \text{'}\vartheta \text{ list} \Rightarrow \text{'}\vartheta \text{ list set}$ **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]} ∩ (*tr-par* *cs* *t*₁ *t*₂)
 else
 ({[*e*₁]} ∩ (*tr-par* *cs* *t*₁ (*e*₂ # *t*₂))) ∪
 ({[*e*₂]} ∩ (*tr-par* *cs* (*e*₁ # *t*₁) *t*₂))
 else
 if *e*₁ ∈ *cs* then
 if *e*₂ ∈ *cs* then {}
 else
 {[*e*₂]} ∩ (*tr-par* *cs* (*e*₁ # *t*₁) *t*₂)
 else
 if *e*₂ ∈ *cs* then
 {[*e*₁]} ∩ (*tr-par* *cs* *t*₁ (*e*₂ # *t*₂))
 else
 ({[*e*₁]} ∩ (*tr-par* *cs* *t*₁ (*e*₂ # *t*₂))) ∪
 {[*e*₂]} ∩ (*tr-par* *cs* (*e*₁ # *t*₁) *t*₂))

abbreviation *tr-inter* :: $\text{'}\vartheta \text{ list} \Rightarrow \text{'}\vartheta \text{ list} \Rightarrow \text{'}\vartheta \text{ list set}$ (**infixr** |||_{*t*} 100) **where**
 $x \text{ |||}_t y \equiv \text{tr-par } \{ \} x y$

11.2 Lifted Trace Merge

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

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

translations

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

11.3 Trace Merge Lemmas

lemma *tr-par-empty*:

$\text{tr-par } \text{cs } t1 \text{ } [] = \{\text{takeWhile } (\lambda x. x \notin \text{cs}) t1\}$
 $\text{tr-par } \text{cs } [] \text{ } t2 = \{\text{takeWhile } (\lambda x. x \notin \text{cs}) t2\}$

— Subgoal 1

apply (*induct t1; simp*)
 — Subgoal 2
apply (*induct t2; simp*)
done

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

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

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

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

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

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

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

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

end

12 Circus Parallel Composition

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

begin

12.1 Merge predicates

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

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

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

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

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

definition $CSPInterMerge \text{ where}$

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

definition $CSPFinalMerge \text{ where}$

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

syntax

$$\begin{aligned} -cinter-merge &:: logic \Rightarrow salpha \Rightarrow logic \Rightarrow salpha \Rightarrow logic \Rightarrow logic \ (- \ll -|-|- \gg^I - [85,0,0,0,86] \ 86) \\ -cfinal-merge &:: logic \Rightarrow salpha \Rightarrow logic \Rightarrow salpha \Rightarrow logic \Rightarrow logic \ (- \ll -|-|- \gg^F - [85,0,0,0,86] \ 86) \\ -wrC &:: logic \Rightarrow salpha \Rightarrow logic \Rightarrow salpha \Rightarrow logic \Rightarrow logic \ (- \ wr[-|-|-]_C - [85,0,0,0,86] \ 86) \end{aligned}$$

translations

$$\begin{aligned} -cinter-merge \ P \ ns1 \ cs \ ns2 \ Q &== CONST \ CSPInterMerge \ P \ ns1 \ cs \ ns2 \ Q \\ -cfinal-merge \ P \ ns1 \ cs \ ns2 \ Q &== CONST \ CSPFinalMerge \ P \ ns1 \ cs \ ns2 \ Q \\ -wrC \ P \ ns1 \ cs \ ns2 \ Q &== P \ wr_R(N_C \ ns1 \ cs \ ns2) \ Q \end{aligned}$$

lemma $CSPInnerMerge-R2m \ [closure]: N_C \ ns1 \ cs \ ns2 \text{ is } R2m$

by (rel-auto)

lemma $CSPInnerMerge-RDM \ [closure]: N_C \ ns1 \ cs \ ns2 \text{ is } RDM$

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

lemma $ex-ref'-R2m-closed \ [closure]:$

assumes $P \text{ is } R2m$

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

proof –

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

by (rel-auto)

thus ?thesis

by (metis Healthy-def' assms)

qed

lemma $CSPInnerMerge-unrests \ [unrest]:$

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

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

lemma *CSPInterMerge-unrest-st'* [unrest]:
 $\$st' \# P \llbracket ns1 | cs | ns2 \rrbracket^I Q$
by (rel-auto)

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

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

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

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

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

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

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

lemma *SymMerge-CSPInnerInterleave* [closure]:
 AssocMerge $(N_I \text{ } 0_L \text{ } 0_L)$
apply (rel-auto)
apply (rename-tac tr tr₂' ref₀ tr₀' ref₀' tr₁' ref₁' tr' ref₂' tr_i' ref₃')
oops

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

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

by (simp add: CSPFinalMerge-def)

lemma CSPInterMerge-commute:

assumes $ns1 \bowtie ns2$

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

proof –

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

by (simp add: CSPInterMerge-def)

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

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

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

by (simp add: seqr-exists-right)

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

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

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

by (simp add: CSPInterMerge-def)

finally show ?thesis .

qed

lemma CSPFinalMerge-commute:

assumes $ns1 \bowtie ns2$

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

proof –

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

by (simp add: CSPFinalMerge-def)

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

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

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

by (simp add: seqr-exists-right)

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

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

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

by (simp add: CSPFinalMerge-def)

finally show ?thesis .

qed

Important theorem that shows the form of a parallel process

lemma CSPInnerMerge-form:

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

assumes $vwb\text{-}lens\ ns1\ vwb\text{-}lens\ ns2\ P\ is\ RR\ Q\ is\ RR$

shows

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

$(\exists (ref_0, ref_1, st_0, st_1, tt_0, tt_1) \cdot$

$P \llbracket \langle ref_0 \rangle, \langle st_0 \rangle, \langle \rangle, \langle tt_0 \rangle / \$ref', \$st', \$tr, \$tr' \rrbracket \wedge Q \llbracket \langle ref_1 \rangle, \langle st_1 \rangle, \langle \rangle, \langle tt_1 \rangle / \$ref', \$st', \$tr, \$tr' \rrbracket$

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

$\wedge \$tr \leq_u \tr'

$\wedge \&tt \in_u \langle tt_0 \rangle \star_{\langle cs \rangle} \langle tt_1 \rangle$

$\wedge \langle tt_0 \rangle \upharpoonright_u \langle cs \rangle =_u \langle tt_1 \rangle \upharpoonright_u \langle cs \rangle$

$\wedge \$st' =_u (\$st \oplus \langle st_0 \rangle\ on\ \&ns1) \oplus \langle st_1 \rangle\ on\ \&ns2)$

(is ?lhs = ?rhs)

proof –

have $P : (\exists \{ \$ok', \$wait' \} \cdot R2(P)) = P$ (is ?P' = -)

by (simp add: ex-unrest ex-plus Healthy-if assms RR-implies-R2 unrest closure)

have $Q : (\exists \{ \$ok', \$wait' \} \cdot R2(Q)) = Q$ (is ?Q' = -)

by (simp add: ex-unrest ex-plus Healthy-if assms RR-implies-R2 unrest closure)
 from assms(1,2)
 have $?P' \parallel_{N_C} ns1\ cs\ ns2\ ?Q' =$
 $(\exists (ref_0, ref_1, st_0, st_1, tt_0, tt_1) \cdot$
 $?P' \llbracket \langle ref_0 \rangle, \langle st_0 \rangle, \langle \rangle, \langle tt_0 \rangle / \$ref', \$st', \$tr, \$tr' \rrbracket \wedge ?Q' \llbracket \langle ref_1 \rangle, \langle st_1 \rangle, \langle \rangle, \langle tt_1 \rangle / \$ref', \$st', \$tr, \$tr' \rrbracket$
 $\wedge \$ref' \subseteq_u ((\langle ref_0 \rangle \cup_u \langle ref_1 \rangle) \cap_u \langle cs \rangle) \cup_u ((\langle ref_0 \rangle \cap_u \langle ref_1 \rangle) - \langle cs \rangle)$
 $\wedge \$tr \leq_u \tr'
 $\wedge \&tt \in_u \langle tt_0 \rangle \star_{\langle cs \rangle} \langle tt_1 \rangle$
 $\wedge \langle tt_0 \rangle \upharpoonright_u \langle cs \rangle =_u \langle tt_1 \rangle \upharpoonright_u \langle cs \rangle$
 $\wedge \$st' =_u (\$st \oplus \langle st_0 \rangle \text{ on } \&ns1) \oplus \langle st_1 \rangle \text{ on } \&ns2)$
 apply (simp add: par-by-merge-alt-def, rel-auto, blast)
 apply (rename-tac ok wait tr st ref tr' ref' ref_0 ref_1 st_0 st_1 tr_0 ok_0 tr_1 wait_0 ok_1 wait_1)
 apply (rule-tac x=ok in exI)
 apply (rule-tac x=wait in exI)
 apply (rule-tac x=tr in exI)
 apply (rule-tac x=st in exI)
 apply (rule-tac x=ref in exI)
 apply (rule-tac x=tr @ tr_0 in exI)
 apply (rule-tac x=st_0 in exI)
 apply (rule-tac x=ref_0 in exI)
 apply (auto)
 apply (metis Prefix-Order.prefixI append-minus)
 done
 thus ?thesis
 by (simp add: P Q)
 qed

lemma CSPInterMerge-form:

fixes $P\ Q :: ('s, 't) \text{ action}$

assumes $vwb\text{-lens}\ ns1\ vwb\text{-lens}\ ns2\ P\ \text{is}\ RR\ Q\ \text{is}\ RR$

shows

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

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

(is ?lhs = ?rhs)

proof –

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

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

also have ... =

$(\exists \$st' \cdot$

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

by (simp add: CSPInnerMerge-form assms)

also have ... = ?rhs

by (rel-blast)

finally show ?thesis .

qed

lemma *CSPFinalMerge-form*:

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

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

shows

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

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

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

$\wedge \$tr \leq_u \tr'

$\wedge \&tt \in_u \langle tt_0 \rangle \star_{\langle cs \rangle} \langle tt_1 \rangle$

$\wedge \langle tt_0 \rangle \upharpoonright_u \langle cs \rangle =_u \langle tt_1 \rangle \upharpoonright_u \langle cs \rangle$

$\wedge \$st' =_u (\$st \oplus \langle st_0 \rangle \text{ on } \&ns1) \oplus \langle st_1 \rangle \text{ on } \&ns2)$

(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 ... =

$(\exists \$ref' \cdot$

$(\exists (ref_0, ref_1, st_0, st_1, tt_0, tt_1) \cdot$

$P \llbracket \langle ref_0 \rangle, \langle st_0 \rangle, \langle \rangle, \langle tt_0 \rangle / \$ref', \$st', \$tr, \$tr' \rrbracket \wedge Q \llbracket \langle ref_1 \rangle, \langle st_1 \rangle, \langle \rangle, \langle tt_1 \rangle / \$ref', \$st', \$tr, \$tr' \rrbracket$

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

$\wedge \$tr \leq_u \tr'

$\wedge \&tt \in_u \langle tt_0 \rangle \star_{\langle cs \rangle} \langle tt_1 \rangle$

$\wedge \langle tt_0 \rangle \upharpoonright_u \langle cs \rangle =_u \langle tt_1 \rangle \upharpoonright_u \langle cs \rangle$

$\wedge \$st' =_u (\$st \oplus \langle st_0 \rangle \text{ on } \&ns1) \oplus \langle st_1 \rangle \text{ on } \&ns2))$

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

also have ... =

$(\exists \$ref' \cdot$

$(\exists (ref_0, ref_1, st_0, st_1, tt_0, tt_1) \cdot$

$(\exists \$ref' \cdot P) \llbracket \langle ref_0 \rangle, \langle st_0 \rangle, \langle \rangle, \langle tt_0 \rangle / \$ref', \$st', \$tr, \$tr' \rrbracket \wedge (\exists \$ref' \cdot Q) \llbracket \langle ref_1 \rangle, \langle st_1 \rangle, \langle \rangle, \langle tt_1 \rangle / \$ref', \$st', \$tr, \$tr' \rrbracket$

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

$\wedge \$tr \leq_u \tr'

$\wedge \&tt \in_u \langle tt_0 \rangle \star_{\langle cs \rangle} \langle tt_1 \rangle$

$\wedge \langle tt_0 \rangle \upharpoonright_u \langle cs \rangle =_u \langle tt_1 \rangle \upharpoonright_u \langle cs \rangle$

$\wedge \$st' =_u (\$st \oplus \langle st_0 \rangle \text{ on } \&ns1) \oplus \langle st_1 \rangle \text{ on } \&ns2))$

by (*simp add: ex-unrest assms*)

also have ... =

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

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

$\wedge \$tr \leq_u \tr'

$\wedge \&tt \in_u \langle tt_0 \rangle \star_{\langle cs \rangle} \langle tt_1 \rangle$

$\wedge \langle tt_0 \rangle \upharpoonright_u \langle cs \rangle =_u \langle tt_1 \rangle \upharpoonright_u \langle cs \rangle$

$\wedge \$st' =_u (\$st \oplus \langle st_0 \rangle \text{ on } \&ns1) \oplus \langle st_1 \rangle \text{ on } \&ns2)$

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 $\Phi(s_0, \sigma_0, t_0) \parallel_{N_C} ns1 \ cs \ ns2 \ P =$

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

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

$$\begin{aligned}
& [\$ref' \mapsto_s \langle ref_1 \rangle, \$st' \mapsto_s \langle st_1 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_1 \rangle] \dagger P \wedge \\
& \$ref' \subseteq_u \langle cs \rangle \cup_u (\langle ref_1 \rangle - \langle cs \rangle) \wedge \\
& [\langle trace \rangle \in_u t_0 \star \langle cs \rangle \langle tt_1 \rangle \wedge t_0 \upharpoonright_u \langle cs \rangle =_u \langle tt_1 \rangle \upharpoonright_u \langle cs \rangle]_t \wedge \\
& \$st' =_u \$st \oplus \langle \sigma_0 \rangle (\$st)_a \text{ on } \&ns1 \oplus \langle st_1 \rangle \text{ on } \&ns2) \\
& (\text{is } ?lhs = ?rhs) \\
\text{proof } - \\
& \text{have } ?lhs = \\
& (\exists (ref_0, ref_1, st_0, st_1, tt_0, tt_1) \cdot \\
& [\$ref' \mapsto_s \langle ref_0 \rangle, \$st' \mapsto_s \langle st_0 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_0 \rangle] \dagger \Phi(s_0, \sigma_0, t_0) \wedge \\
& [\$ref' \mapsto_s \langle ref_1 \rangle, \$st' \mapsto_s \langle st_1 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_1 \rangle] \dagger P \wedge \\
& \$ref' \subseteq_u (\langle ref_0 \rangle \cup_u \langle ref_1 \rangle) \cap_u \langle cs \rangle \cup_u (\langle ref_0 \rangle \cap_u \langle ref_1 \rangle - \langle cs \rangle) \wedge \\
& \$tr \leq_u \$tr' \wedge \\
& \&tt \in_u \langle tt_0 \rangle \star \langle cs \rangle \langle tt_1 \rangle \wedge \langle tt_0 \rangle \upharpoonright_u \langle cs \rangle =_u \langle tt_1 \rangle \upharpoonright_u \langle cs \rangle \wedge \$st' =_u \$st \oplus \langle st_0 \rangle \text{ on } \&ns1 \\
& \oplus \langle st_1 \rangle \text{ on } \&ns2) \\
& \text{by (simp add: CSPInnerMerge-form assms closure)} \\
& \text{also have ... =} \\
& (\exists (ref_1, st_1, tt_1) \cdot \\
& [s_0]_{S<} \wedge \\
& [\$ref' \mapsto_s \langle ref_1 \rangle, \$st' \mapsto_s \langle st_1 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_1 \rangle] \dagger P \wedge \\
& \$ref' \subseteq_u \langle cs \rangle \cup_u (\langle ref_1 \rangle - \langle cs \rangle) \wedge \\
& [\langle trace \rangle \in_u t_0 \star \langle cs \rangle \langle tt_1 \rangle \wedge t_0 \upharpoonright_u \langle cs \rangle =_u \langle tt_1 \rangle \upharpoonright_u \langle cs \rangle]_t \wedge \\
& \$st' =_u \$st \oplus \langle \sigma_0 \rangle (\$st)_a \text{ on } \&ns1 \oplus \langle st_1 \rangle \text{ on } \&ns2) \\
& \text{by (rel-blast)} \\
& \text{finally show ?thesis .} \\
\text{qed}
\end{aligned}$$

lemma *merge-csp-do-right:*

assumes *vwb-lens ns1 vwb-lens ns2 ns1 \bowtie ns2 P is RR*
shows $P \parallel_{N_C} ns1 \text{ cs } ns2 \Phi(s_1, \sigma_1, t_1) =$

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

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

lemma *FinalMerge-csp-do-left:*

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

shows $\Phi(s_0, \sigma_0, t_0) \llbracket ns1 | cs | ns2 \rrbracket^F P =$
 $(\exists (st_1, t_1) \cdot$
 $\quad [s_0]_{S<} \wedge$
 $\quad [\$st' \mapsto_s \langle st_1 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle t_1 \rangle] \dagger P \wedge$
 $\quad [\langle \langle trace \rangle \rangle \in_u t_0 \star \langle cs \rangle \langle t_1 \rangle \wedge t_0 \upharpoonright_u \langle cs \rangle =_u \langle t_1 \rangle \upharpoonright_u \langle cs \rangle]_t \wedge$
 $\quad \$st' =_u \$st \oplus \langle \sigma_0 \rangle (\$st)_a \text{ on } \&ns1 \oplus \langle st_1 \rangle \text{ on } \&ns2)$
(is ?lhs = ?rhs)
proof –
have ?lhs =
 $(\exists (st_0, st_1, tt_0, tt_1) \cdot$
 $\quad [\$st' \mapsto_s \langle st_0 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_0 \rangle] \dagger \Phi(s_0, \sigma_0, t_0) \wedge$
 $\quad [\$st' \mapsto_s \langle st_1 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_1 \rangle] \dagger RR(\exists \$ref' \cdot P) \wedge$
 $\quad \$tr \leq_u \$tr' \wedge \&tt \in_u \langle tt_0 \rangle \star \langle cs \rangle \langle tt_1 \rangle \wedge \langle tt_0 \rangle \upharpoonright_u \langle cs \rangle =_u \langle tt_1 \rangle \upharpoonright_u \langle cs \rangle \wedge$
 $\quad \$st' =_u \$st \oplus \langle st_0 \rangle \text{ on } \&ns1 \oplus \langle st_1 \rangle \text{ on } \&ns2)$
by (simp add: CSPFinalMerge-form ex-unrest Healthy-if unrest closure assms)
also have ... =
 $(\exists (st_1, tt_1) \cdot$
 $\quad [s_0]_{S<} \wedge$
 $\quad [\$st' \mapsto_s \langle st_1 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_1 \rangle] \dagger RR(\exists \$ref' \cdot P) \wedge$
 $\quad [\langle \langle trace \rangle \rangle \in_u t_0 \star \langle cs \rangle \langle tt_1 \rangle \wedge t_0 \upharpoonright_u \langle cs \rangle =_u \langle tt_1 \rangle \upharpoonright_u \langle cs \rangle]_t \wedge$
 $\quad \$st' =_u \$st \oplus \langle \sigma_0 \rangle (\$st)_a \text{ on } \&ns1 \oplus \langle st_1 \rangle \text{ on } \&ns2)$
by (rel-blast)
also have ... =
 $(\exists (st_1, t_1) \cdot$
 $\quad [s_0]_{S<} \wedge$
 $\quad [\$st' \mapsto_s \langle st_1 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle t_1 \rangle] \dagger P \wedge$
 $\quad [\langle \langle trace \rangle \rangle \in_u t_0 \star \langle cs \rangle \langle t_1 \rangle \wedge t_0 \upharpoonright_u \langle cs \rangle =_u \langle t_1 \rangle \upharpoonright_u \langle cs \rangle]_t \wedge$
 $\quad \$st' =_u \$st \oplus \langle \sigma_0 \rangle (\$st)_a \text{ on } \&ns1 \oplus \langle st_1 \rangle \text{ on } \&ns2)$
by (simp add: ex-unrest Healthy-if unrest closure assms)
finally show ?thesis .
qed

lemma *FinalMerge-csp-do-right:*

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

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

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

proof –

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

by (simp add: assms CSPFinalMerge-commute)

also have ... = ?rhs

apply (simp add: FinalMerge-csp-do-left assms lens-indep-sym conj-comm)

apply (rel-auto)

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

done

finally show ?thesis .

qed

lemma *FinalMerge-csp-do:*

assumes *vwb-lens ns1 vwb-lens ns2 ns1 \bowtie ns2*

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

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

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

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

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

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

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

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

lemma *InterMerge-csp-enable*:

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

proof –

have ?lhs =
 ($\exists (ref_0, ref_1, st_0, st_1, tt_0, tt_1) \cdot$
 $[\$ref' \mapsto_s \llbracket ref_0 \rrbracket, \$st' \mapsto_s \llbracket st_0 \rrbracket, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \llbracket tt_0 \rrbracket] \dagger \mathcal{E}(s_1, t_1, E_1) \wedge$
 $[\$ref' \mapsto_s \llbracket ref_1 \rrbracket, \$st' \mapsto_s \llbracket st_1 \rrbracket, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \llbracket tt_1 \rrbracket] \dagger \mathcal{E}(s_2, t_2, E_2) \wedge$
 $\$ref' \subseteq_u (\llbracket ref_0 \rrbracket \cup_u \llbracket ref_1 \rrbracket) \cap_u \llbracket cs \rrbracket \cup_u (\llbracket ref_0 \rrbracket \cap_u \llbracket ref_1 \rrbracket - \llbracket cs \rrbracket) \wedge$
 $\$tr \leq_u \$tr' \wedge \&tt \in_u \llbracket tt_0 \rrbracket \star_{\llbracket cs \rrbracket} \llbracket tt_1 \rrbracket \wedge \llbracket tt_0 \rrbracket \downarrow_u \llbracket cs \rrbracket =_u \llbracket tt_1 \rrbracket \downarrow_u \llbracket cs \rrbracket)$
 by (simp add: CSPInterMerge-form unrest closure assms)
 also have ... =
 ($\exists (ref_0, ref_1, tt_0, tt_1) \cdot$
 $[\$ref' \mapsto_s \llbracket ref_0 \rrbracket, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \llbracket tt_0 \rrbracket] \dagger \mathcal{E}(s_1, t_1, E_1) \wedge$
 $[\$ref' \mapsto_s \llbracket ref_1 \rrbracket, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \llbracket tt_1 \rrbracket] \dagger \mathcal{E}(s_2, t_2, E_2) \wedge$
 $\$ref' \subseteq_u (\llbracket ref_0 \rrbracket \cup_u \llbracket ref_1 \rrbracket) \cap_u \llbracket cs \rrbracket \cup_u (\llbracket ref_0 \rrbracket \cap_u \llbracket ref_1 \rrbracket - \llbracket cs \rrbracket) \wedge$
 $\$tr \leq_u \$tr' \wedge \&tt \in_u \llbracket tt_0 \rrbracket \star_{\llbracket cs \rrbracket} \llbracket tt_1 \rrbracket \wedge \llbracket tt_0 \rrbracket \downarrow_u \llbracket cs \rrbracket =_u \llbracket tt_1 \rrbracket \downarrow_u \llbracket cs \rrbracket)$
 by (rel-auto)

also have ... =

$$\begin{aligned} & ([s_1 \wedge s_2]_{S<} \wedge \\ & (\forall e \in [(E_1 \cap_u E_2 \cap_u \ll cs \gg) \cup_u ((E_1 \cup_u E_2) - \ll cs \gg)]_{S<} \cdot \ll e \gg \notin_u \$ref') \wedge \\ & [\ll trace \gg \in_u t_1 \star_{\ll cs \gg} t_2 \wedge t_1 \upharpoonright_u \ll cs \gg =_u t_2 \upharpoonright_u \ll cs \gg]_t \\ &) \\ & \text{apply } (rel-auto) \\ & \text{apply } (rename-tac \ tr \ st \ tr' \ ref') \\ & \text{apply } (rule-tac \ x = - \ll E_1 \gg_e \ st \ \text{in} \ exI) \\ & \text{apply } (simp) \\ & \text{apply } (rule-tac \ x = - \ll E_2 \gg_e \ st \ \text{in} \ exI) \\ & \text{apply } (auto) \\ & \text{done} \\ & \text{finally show } ?thesis . \\ & \text{qed} \end{aligned}$$

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

lemma *InterMerge-csp-enable-csp-do*:
assumes *vwb-lens ns1 vwb-lens ns2 ns1* \bowtie *ns2*
shows $\mathcal{E}(s_1, t_1, E_1) \llbracket ns1 | cs | ns2 \rrbracket^I \Phi(s_2, \sigma_2, t_2) =$
 $([s_1 \wedge s_2]_{S<} \wedge (\forall e \in [(E_1 - \ll cs \gg)]_{S<} \cdot \ll e \gg \notin_u \$ref') \wedge$
 $[\ll trace \gg \in_u t_1 \star_{\ll cs \gg} t_2 \wedge t_1 \upharpoonright_u \ll cs \gg =_u t_2 \upharpoonright_u \ll cs \gg]_t)$
(is ?lhs = ?rhs)
proof –
have *?lhs* =
 $(\exists (ref_0, ref_1, st_0, st_1, tt_0, tt_1) \cdot$
 $[\$ref' \mapsto_s \ll ref_0 \gg, \$st' \mapsto_s \ll st_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll tt_0 \gg] \dagger \mathcal{E}(s_1, t_1, E_1) \wedge$
 $[\$ref' \mapsto_s \ll ref_1 \gg, \$st' \mapsto_s \ll st_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll tt_1 \gg] \dagger \Phi(s_2, \sigma_2, t_2) \wedge$
 $\$ref' \subseteq_u (\ll ref_0 \gg \cup_u \ll ref_1 \gg) \cap_u \ll cs \gg \cup_u (\ll ref_0 \gg \cap_u \ll ref_1 \gg - \ll cs \gg) \wedge$
 $\$tr \leq_u \$tr' \wedge \&tt \in_u \ll tt_0 \gg \star_{\ll cs \gg} \ll tt_1 \gg \wedge \ll tt_0 \gg \upharpoonright_u \ll cs \gg =_u \ll tt_1 \gg \upharpoonright_u \ll cs \gg)$
by (*simp add: CSPInterMerge-form unrest closure assms*)

also have ... =
 $(\exists (ref_0, ref_1, tt_0) \cdot$
 $[\$ref' \mapsto_s \ll ref_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll tt_0 \gg] \dagger \mathcal{E}(s_1, t_1, E_1) \wedge$
 $[s_2]_{S<} \wedge$
 $\$ref' \subseteq_u (\ll ref_0 \gg \cup_u \ll ref_1 \gg) \cap_u \ll cs \gg \cup_u (\ll ref_0 \gg \cap_u \ll ref_1 \gg - \ll cs \gg) \wedge$
 $[\ll trace \gg \in_u t_1 \star_{\ll cs \gg} t_2 \wedge t_1 \upharpoonright_u \ll cs \gg =_u t_2 \upharpoonright_u \ll cs \gg]_t)$
by (*rel-auto*)
also have ... = $([s_1 \wedge s_2]_{S<} \wedge (\forall e \in [(E_1 - \ll cs \gg)]_{S<} \cdot \ll e \gg \notin_u \$ref') \wedge$
 $[\ll trace \gg \in_u t_1 \star_{\ll cs \gg} t_2 \wedge t_1 \upharpoonright_u \ll cs \gg =_u t_2 \upharpoonright_u \ll cs \gg]_t)$
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 $\mathcal{E}(s_1, t_1, E_1) \llbracket ns1 | cs | ns2 \rrbracket^I \Phi(s_2, \sigma_2, t_2) =$
 $(\bigcap \text{trace} \mid \ll trace \gg \in_u [t_1 \star_{\ll cs \gg} t_2]_{S<} \cdot$

$\mathcal{E}(s_1 \wedge s_2 \wedge t_1 \upharpoonright_u \ll cs \gg =_u t_2 \upharpoonright_u \ll cs \gg, \ll trace \gg, E_1 - \ll cs \gg)$
 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 $\Phi(s_1, \sigma_1, t_1) \llbracket ns1 | cs | ns2 \rrbracket^I \mathcal{E}(s_2, t_2, E_2) =$
 $([s_1 \wedge s_2]_{S<} \wedge (\forall e \in [(E_2 - \ll cs \gg)]_{S<} \cdot \ll e \gg \notin_u \$ref')) \wedge$
 $[\ll trace \gg \in_u t_1 \star_{\ll cs \gg} t_2 \wedge t_1 \upharpoonright_u \ll cs \gg =_u t_2 \upharpoonright_u \ll cs \gg]_t)$
(is ?lhs = ?rhs)

proof –

have $\Phi(s_1, \sigma_1, t_1) \llbracket ns1 | cs | ns2 \rrbracket^I \mathcal{E}(s_2, t_2, E_2) = \mathcal{E}(s_2, t_2, E_2) \llbracket ns2 | cs | ns1 \rrbracket^I \Phi(s_1, \sigma_1, t_1)$
by (simp add: CSPInterMerge-commute assms)

also have ... = ?rhs

by (simp add: InterMerge-csp-enable-csp-do assms lens-indep-sym trace-merge-commute conj-comm eq-upred-sym)

finally show ?thesis .

qed

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

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

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

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

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

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

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

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

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

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

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

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

lemma *merge-csp-do-right*:

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

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

(is ?lhs = ?rhs)

proof –

have ?lhs =

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

$\$st' =_u \$st \oplus \langle st_0 \rangle \text{ on } \&ns1 \oplus \langle \sigma_1 \rangle (\$st)_a \text{ on } \&ns2) ;; R1 \text{ true})$
 by (simp add: wrR-def par-by-merge-seq-remove merge-csp-do-right closure assms Healthy-if rpred)
 also have ... =
 $(\neg_r (\exists (ref_0, st_0, tt_0) \cdot$
 $[\$ref' \mapsto_s \langle ref_0 \rangle, \$st' \mapsto_s \langle st_0 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tt_0 \rangle] \dagger (\neg_r RC(P)) \wedge$
 $[s_1]_{S<} \wedge$
 $\$ref' \subseteq_u \langle cs \rangle \cup_u (\langle ref_0 \rangle - \langle cs \rangle) \wedge$
 $[\langle trace \rangle \in_u \langle tt_0 \rangle \star \langle cs \rangle t_1 \wedge \langle tt_0 \rangle \upharpoonright_u \langle cs \rangle =_u t_1 \upharpoonright_u \langle cs \rangle]_t ;; true_r \wedge$
 $\$st' =_u \$st \oplus \langle st_0 \rangle \text{ on } \&ns1 \oplus \langle \sigma_1 \rangle (\$st)_a \text{ on } \&ns2))$
 apply (rel-auto)

oops

12.2 Parallel operator

syntax

$-par-circus :: logic \Rightarrow salpha \Rightarrow logic \Rightarrow salpha \Rightarrow logic \Rightarrow logic \quad (- \llbracket - \rrbracket - [75, 0, 0, 76] \ 76)$
 $-par-csp :: logic \Rightarrow logic \Rightarrow logic \Rightarrow logic \quad (- \llbracket - \rrbracket_C - [75, 0, 76] \ 76)$
 $-inter-circus :: logic \Rightarrow salpha \Rightarrow salpha \Rightarrow logic \Rightarrow logic \quad (- \llbracket - \rrbracket - [75, 0, 0, 76] \ 76)$
 $-inter-csp :: logic \Rightarrow logic \Rightarrow logic \quad (\mathbf{infixr} \ ||| \ 75)$

translations

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

definition $CSP5 :: ('\sigma, '\varphi) \text{ action} \Rightarrow (''\sigma, '\varphi) \text{ action}$ **where**
 $[upred-defs]: CSP5(P) = (P \ ||| \ Skip)$

definition $C2 :: (''\sigma, '\varphi) \text{ action} \Rightarrow (''\sigma, '\varphi) \text{ action}$ **where**
 $[upred-defs]: C2(P) = (P \ \llbracket \Sigma \rrbracket \{\} \llbracket \emptyset \rrbracket \ Skip)$

lemma *Skip-right-form*:

assumes $P_1 \text{ is } RC \ P_2 \text{ is } RR \ P_3 \text{ is } RR \ \$st' \ \# \ P_2$
 shows $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) ;; Skip = \mathbf{R}_s(P_1 \vdash P_2 \diamond (\exists \$ref' \cdot P_3))$

proof –

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

show ?thesis

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

qed

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

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

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

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

$ns1 \bowtie ns2$

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

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

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

12.3 Parallel Laws

lemma *ParCSP-expand*:

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

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

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

proof –

have $(P \parallel_{MR(N_C \ ns1 \ cs \ ns2)} Q)$ is CSP

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

lemma parallel-is-CSP3 [closure]:
 assumes $P \text{ is CSP } P \text{ is CSP3 } Q \text{ is CSP } Q \text{ is CSP3}$
 shows $(P \llbracket \text{ns1} \parallel \text{cs} \parallel \text{ns2} \rrbracket Q) \text{ is CSP3}$
proof –
 have $(P \parallel_{M_R(N_C \text{ ns1 cs ns2})} Q) \text{ is CSP}$
 by (simp add: closure assms)
 hence $(P \parallel_{M_R(N_C \text{ ns1 cs ns2})} Q) ;; \text{Skip is CSP}$
 by (simp add: closure)
 thus ?thesis
 oops

theorem parallel-commutative:
 assumes $\text{ns1} \bowtie \text{ns2}$
 shows $(P \llbracket \text{ns1} \parallel \text{cs} \parallel \text{ns2} \rrbracket Q) = (Q \llbracket \text{ns2} \parallel \text{cs} \parallel \text{ns1} \rrbracket P)$
proof –
 have $(P \llbracket \text{ns1} \parallel \text{cs} \parallel \text{ns2} \rrbracket Q) = P \parallel_{\text{swap}_m} ;; (M_C \text{ ns2 cs ns1}) Q$
 by (simp add: CSPMerge-def seqr-assoc[THEN sym] swap-merge-rd swap-CSPInnerMerge lens-indep-sym assms)
 also have $\dots = Q \llbracket \text{ns2} \parallel \text{cs} \parallel \text{ns1} \rrbracket P$
 by (metis par-by-merge-commute-swap)
 finally show ?thesis .
 qed

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

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

lemma C2-form:
 assumes $P \text{ is NCSP}$
 shows $C2(P) = \mathbf{R}_s (\text{pre}_R P \vdash (\exists \text{ref}_0 \cdot \text{peri}_R P [\llbracket \llbracket \text{ref}_0 \rrbracket / \$\text{ref}' \rrbracket \wedge \$\text{ref}' \subseteq_u \llbracket \text{ref}_0 \rrbracket \diamond \text{post}_R P])$
proof –
 have $1:\Phi(\text{true}, \text{id}, \langle \rangle) \text{ wr}[\Sigma|\{\}|\emptyset]_C \text{pre}_R P = \text{pre}_R P \text{ (is ?lhs = ?rhs)}$
proof –
 have ?lhs = $(\neg_r (\exists (\text{ref}_0, \text{st}_0, \text{tt}_0) \cdot$
 $[\$ \text{ref}' \mapsto_s \llbracket \text{ref}_0 \rrbracket, \$ \text{st}' \mapsto_s \llbracket \text{st}_0 \rrbracket, \$ \text{tr} \mapsto_s \langle \rangle, \$ \text{tr}' \mapsto_s \llbracket \text{tt}_0 \rrbracket] \dagger (\exists \$ \text{ref}'; \$ \text{st}' \cdot \text{RR}(\neg_r$
 $\text{pre}_R P))) \wedge$
 $\$ \text{ref}' \subseteq_u \llbracket \text{ref}_0 \rrbracket \wedge [\llbracket \text{trace} \rrbracket =_u \llbracket \text{tt}_0 \rrbracket]_t \wedge$
 $\$ \text{st}' =_u \$ \text{st} \oplus \llbracket \text{st}_0 \rrbracket \text{ on } \Sigma \oplus \llbracket \text{id} \rrbracket (\$ \text{st})_a \text{ on } \emptyset) ;; R1 \text{ true}$
 by (simp add: wrR-def par-by-merge-seq-remove rpred merge-csp-do-right ex-unrest Healthy-if pr-var-def closure assms unrest usubst)
 also have $\dots = (\neg_r (\exists \$ \text{ref}'; \$ \text{st}' \cdot \text{RR}(\neg_r \text{pre}_R P))) ;; R1 \text{ true}$
 by (rel-auto)
 also have $\dots = (\neg_r (\neg_r \text{pre}_R P)) ;; R1 \text{ true}$
 by (simp add: Healthy-if closure ex-unrest unrest assms)
 also have $\dots = \text{pre}_R P$
 by (simp add: NCSP-implies-NSRD NSRD-neg-pre-unit R1-preR assms rea-not-not)
 finally show ?thesis .

qed

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

proof –

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

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

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

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

also have ... =

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

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

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

also have ... =

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

by (rel-auto)

also have ... = ?rhs

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

finally show ?thesis .

qed

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

proof –

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

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

also have ... = $(\exists (st_0, t_0) \cdot$

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

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

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

by (rel-auto)

finally show ?thesis

by (simp add: Healthy-if assms closure)

qed

show ?thesis

proof –

have $C2(P) = \mathbf{R}_s (\Phi(true, id, \langle \rangle) \text{ wr } [\Sigma | \{ \} | \emptyset]_C pre_R P \vdash$

$(pre_R P \Rightarrow_r peri_R P) \llbracket \Sigma | \{ \} | \emptyset \rrbracket^I \Phi(true, id, \langle \rangle) \diamond (pre_R P \Rightarrow_r post_R P) \llbracket \Sigma | \{ \} | \emptyset \rrbracket^F \Phi(true, id, \langle \rangle))$

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

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

by (simp add: 1 2 3)

finally show ?thesis .

qed

qed

lemma Skip-C2-closed [closure]:

Skip is C2

apply (simp add: Healthy-def C2-form)

apply (simp add: C2-form closure rdes usubst)

apply (simp add: rdes-def)

done

lemma ref-down-CRR [closure]:

assumes P is NCSP

shows $(\exists ref_0 \cdot peri_R P \llbracket \langle ref_0 \rangle / \$ref' \rrbracket \wedge \$ref' \subseteq_u \langle ref_0 \rangle)$ is CRR

proof –

have $(\exists \text{ ref}_0 \cdot \text{peri}_R P[\llbracket \langle \text{ref}_0 \rangle \rrbracket / \$\text{ref}'] \wedge \$\text{ref}' \subseteq_u \langle \text{ref}_0 \rangle) =$
 $(\exists \text{ ref}_0 \cdot (\text{CRR}(\text{peri}_R P))[\llbracket \langle \text{ref}_0 \rangle \rrbracket / \$\text{ref}'] \wedge \$\text{ref}' \subseteq_u \langle \text{ref}_0 \rangle)$
by (*simp add: Healthy-if assms closure*)
also have $\dots = \text{CRR}(\exists \text{ ref}_0 \cdot \text{peri}_R P[\llbracket \langle \text{ref}_0 \rangle \rrbracket / \$\text{ref}'] \wedge \$\text{ref}' \subseteq_u \langle \text{ref}_0 \rangle)$
by (*rel-auto*)
finally show *?thesis*
by (*simp add: Healthy-def'*)
qed

lemma *C2-idem*:

assumes *P is NCSP*

shows $\text{C2}(\text{C2}(P)) = \text{C2}(P)$ (**is** *?lhs = ?rhs*)

proof –

have $?lhs = \mathbf{R}_s (\text{pre}_R P \vdash (\exists \text{ ref}_0 \cdot (\text{pre}_R P \Rightarrow_r (\exists \text{ ref}_0' \cdot \text{peri}_R P[\llbracket \langle \text{ref}_0' \rangle \rrbracket / \$\text{ref}'] \wedge \langle \text{ref}_0 \rangle \subseteq_u \langle \text{ref}_0' \rangle)) \wedge \$\text{ref}' \subseteq_u \langle \text{ref}_0 \rangle) \diamond \text{post}_R P)$

by (*simp add: C2-form closure unrest rdes SRD-post-under-pre SRD-peri-under-pre usubst NCSP-rdes-intro assms*)

also have

$\dots = \mathbf{R}_s (\text{pre}_R P \vdash (\exists \text{ ref}_0 \cdot (\exists \text{ ref}_0' \cdot \text{peri}_R P[\llbracket \langle \text{ref}_0' \rangle \rrbracket / \$\text{ref}'] \wedge \langle \text{ref}_0 \rangle \subseteq_u \langle \text{ref}_0' \rangle) \wedge \$\text{ref}' \subseteq_u \langle \text{ref}_0 \rangle) \diamond \text{post}_R P)$

by (*rel-auto*)

also have

$\dots = \mathbf{R}_s (\text{pre}_R P \vdash (\exists \text{ ref}_0 \cdot \text{peri}_R P[\llbracket \langle \text{ref}_0 \rangle \rrbracket / \$\text{ref}'] \wedge \$\text{ref}' \subseteq_u \langle \text{ref}_0 \rangle) \diamond \text{post}_R P)$

by (*rel-auto*)

also have $\dots = \text{C2}(P)$

by (*simp add: C2-form closure unrest assms*)

finally show *?thesis* .

qed

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

Stop is C2

apply (*simp add: Healthy-def C2-form*)

apply (*simp add: C2-form closure rdes usubst*)

apply (*rel-auto*)

done

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

Miracle is C2

apply (*simp add: Healthy-def C2-form*)

apply (*simp add: C2-form closure rdes usubst*)

apply (*simp add: rdes-def*)

done

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

Chaos is C2

apply (*simp add: Healthy-def C2-form*)

apply (*simp add: C2-form closure rdes usubst unrest*)

apply (*simp add: rdes-def*)

apply (*rel-auto*)

done

lemma

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

shows $P \text{ wr}[ns1|cs|ns2]_C \text{ false} = \text{undefined}$ (is ?lhs = ?rhs)

proof –

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

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

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

by (rel-blast)

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

by (simp add: Healthy-if assms)

oops

end

13 Linking to the Failures-Divergences Model

theory utp-circus-fdsem
 imports utp-circus-parallel utp-circus-recursion
 begin

13.1 Failures-Divergences Semantics

The following functions play a similar role to those in Roscoe's CSP semantics, and are calculated from the Circus reactive design semantics. A major difference is that these three functions account for state. Each divergence, trace, and failure is subject to an initial state. Moreover, the traces are terminating traces, and therefore also provide a final state following the given interaction. A more subtle difference from the Roscoe semantics is that the set of traces do not include the divergences. The same semantic information is present, but we construct a direct analogy with the pre-, peri- and postconditions of our reactive designs.

definition *divergences* :: $('σ, 'φ) \text{ action} \Rightarrow 'σ \Rightarrow 'φ \text{ list set } (dv[-] - [0,100] \ 100)$ **where**
 $[upred-defs]: \text{divergences } P \ s = \{t \mid t. '(\neg_r \text{pre}_R(P)) \llbracket \langle\langle s \rangle\rangle, \langle\rangle, \langle\langle t \rangle\rangle / \$st, \$tr, \$tr' \rrbracket '\}$

definition *traces* :: $('σ, 'φ) \text{ action} \Rightarrow 'σ \Rightarrow ('φ \text{ list} \times 'σ) \text{ set } (tr[-] - [0,100] \ 100)$ **where**
 $[upred-defs]: \text{traces } P \ s = \{(t, s') \mid t \ s'. '(pre_R(P) \wedge post_R(P)) \llbracket \langle\langle s \rangle\rangle, \langle\langle s' \rangle\rangle, \langle\rangle, \langle\langle t \rangle\rangle / \$st, \$st', \$tr, \$tr' \rrbracket '\}$

definition *failures* :: $('σ, 'φ) \text{ action} \Rightarrow 'σ \Rightarrow ('φ \text{ list} \times 'φ \text{ set}) \text{ set } (fl[-] - [0,100] \ 100)$ **where**
 $[upred-defs]: \text{failures } P \ s = \{(t, r) \mid t \ r. '(pre_R(P) \wedge peri_R(P)) \llbracket \langle\langle r \rangle\rangle, \langle\langle s \rangle\rangle, \langle\rangle, \langle\langle t \rangle\rangle / \$ref', \$st, \$tr, \$tr' \rrbracket '\}$

lemma *trace-divergence-disj*:

assumes $P \text{ is NCSP } (t, s') \in tr[P] \ s \ t \in dv[P] \ s$
 shows *False*

using *assms*(2,3)
 by (*simp add: traces-def divergences-def, rdes-simp cls:assms, rel-auto*)

lemma *preR-refine-divergences*:

assumes *P* is NCSP *Q* is NCSP $\wedge s. dv[P]s \subseteq dv[Q]s$

shows $pre_R(P) \sqsubseteq pre_R(Q)$

proof (*rule CRR-refine-impl-prop, simp-all add: assms closure usubst unrest*)

fix *t s*

assume *a*: ‘ $[\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg] \dagger pre_R Q$ ’

with *a* show ‘ $[\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg] \dagger pre_R P$ ’

proof (*rule-tac ccontr*)

from *assms*(3)[*of s*] have *b*: $t \in dv[P]s \implies t \in dv[Q]s$

by (*auto*)

assume \neg ‘ $[\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg] \dagger pre_R P$ ’

hence \neg ‘ $[\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg] \dagger CRC(pre_R P)$ ’

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

hence ‘ $[\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg] \dagger (\neg_r CRC(pre_R P))$ ’

by (*rel-auto*)

hence ‘ $[\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg] \dagger (\neg_r pre_R P)$ ’

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

with *a b* show *False*

by (*rel-auto*)

qed

qed

lemma *preR-eq-divergences*:

assumes *P* is NCSP *Q* is NCSP $\wedge s. dv[P]s = dv[Q]s$

shows $pre_R(P) = pre_R(Q)$

by (*metis assms dual-order.antisym order-refl preR-refine-divergences*)

lemma *periR-refine-failures*:

assumes *P* is NCSP *Q* is NCSP $\wedge s. fl[Q]s \subseteq fl[P]s$

shows $(pre_R(P) \wedge peri_R(P)) \sqsubseteq (pre_R(Q) \wedge peri_R(Q))$

proof (*rule CRR-refine-impl-prop, simp-all add: assms closure unrest subst-unrest-3*)

fix *t s r'*

assume *a*: ‘ $[\$ref' \mapsto_s \ll r' \gg, \$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg] \dagger (pre_R Q \wedge peri_R Q)$ ’

from *assms*(3)[*of s*] have *b*: $(t, r') \in fl[Q]s \implies (t, r') \in fl[P]s$

by (*auto*)

with *a* show ‘ $[\$ref' \mapsto_s \ll r' \gg, \$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg] \dagger (pre_R P \wedge peri_R P)$ ’

by (*simp add: failures-def*)

qed

lemma *periR-eq-failures*:

assumes *P* is NCSP *Q* is NCSP $\wedge s. fl[P]s = fl[Q]s$

shows $(pre_R(P) \wedge peri_R(P)) = (pre_R(Q) \wedge peri_R(Q))$

by (*metis (full-types) assms dual-order.antisym order-refl periR-refine-failures*)

lemma *postR-refine-traces*:

assumes *P* is NCSP *Q* is NCSP $\wedge s. tr[Q]s \subseteq tr[P]s$

shows $(pre_R(P) \wedge post_R(P)) \sqsubseteq (pre_R(Q) \wedge post_R(Q))$

proof (*rule CRR-refine-impl-prop, simp-all add: assms closure unrest subst-unrest-5*)

fix *t s s'*

assume *a*: ‘ $[\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg] \dagger (pre_R Q \wedge post_R Q)$ ’

from *assms*(3)[*of s*] have *b*: $(t, s') \in tr[Q]s \implies (t, s') \in tr[P]s$

by (*auto*)

with a show $[\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg] \uparrow (pre_R P \wedge post_R P)$
by (*simp add: traces-def*)
qed

lemma *postR-eq-traces*:
assumes P is NCSP Q is NCSP $\wedge s. tr[P]s = tr[Q]s$
shows $(pre_R(P) \wedge post_R(P)) = (pre_R(Q) \wedge post_R(Q))$
by (*metis assms dual-order.antisym order-refl postR-refine-traces*)

lemma *circus-fd-refine-intro*:
assumes P is NCSP Q is NCSP $\wedge s. dv[Q]s \subseteq dv[P]s \wedge s. fl[Q]s \subseteq fl[P]s \wedge s. tr[Q]s \subseteq tr[P]s$
shows $P \sqsubseteq Q$
proof (*rule SRD-refine-intro', simp-all add: closure assms*)
show $a: pre_R P \Rightarrow pre_R Q$
using *assms(1) assms(2) assms(3) preR-refine-divergences refBy-order* **by** *blast*
show $peri_R P \sqsubseteq (pre_R P \wedge peri_R Q)$
proof –
have $peri_R P \sqsubseteq (pre_R Q \wedge peri_R Q)$
by (*metis (no-types) assms(1) assms(2) assms(4) periR-refine-failures utp-pred-laws.le-inf-iff*)
then show *?thesis*
by (*metis a refBy-order utp-pred-laws.inf.order-iff utp-pred-laws.inf-assoc*)
qed
show $post_R P \sqsubseteq (pre_R P \wedge post_R Q)$
proof –
have $post_R P \sqsubseteq (pre_R Q \wedge post_R Q)$
by (*meson assms(1) assms(2) assms(5) postR-refine-traces utp-pred-laws.le-inf-iff*)
then show *?thesis*
by (*metis a refBy-order utp-pred-laws.inf.absorb-iff1 utp-pred-laws.inf-assoc*)
qed
qed

13.2 Circus Operators

lemma *traces-Skip*:
 $tr[Skip]s = \{([], s)\}$
by (*simp add: traces-def rdes alpha closure, rel-simp*)

lemma *failures-Skip*:
 $fl[Skip]s = \{\}$
by (*simp add: failures-def, rdes-calc*)

lemma *divergences-Skip*:
 $dv[Skip]s = \{\}$
by (*simp add: divergences-def, rdes-calc*)

lemma *traces-Stop*:
 $tr[Stop]s = \{\}$
by (*simp add: traces-def, rdes-calc*)

lemma *failures-Stop*:
 $fl[Stop]s = \{([], E) \mid E. True\}$
by (*simp add: failures-def, rdes-calc, rel-auto*)

lemma *divergences-Stop*:
 $dv[Stop]s = \{\}$
by (*simp add: divergences-def, rdes-calc*)

lemma *traces-AssignsCSP*:
 $tr\llbracket \langle \sigma \rangle_C \rrbracket s = \{(\llbracket \cdot \rrbracket, \sigma(s))\}$
by (*simp add: traces-def rdes closure usubst alpha, rel-auto*)

lemma *failures-AssignsCSP*:
 $fl\llbracket \langle \sigma \rangle_C \rrbracket s = \{\}$
by (*simp add: failures-def, rdes-calc*)

lemma *divergences-AssignsCSP*:
 $dv\llbracket \langle \sigma \rangle_C \rrbracket s = \{\}$
by (*simp add: divergences-def, rdes-calc*)

lemma *failures-Miracle*: $fl\llbracket Miracle \rrbracket s = \{\}$
by (*simp add: failures-def rdes closure usubst*)

lemma *divergences-Miracle*: $dv\llbracket Miracle \rrbracket s = \{\}$
by (*simp add: divergences-def rdes closure usubst*)

lemma *failures-Chaos*: $fl\llbracket Chaos \rrbracket s = \{\}$
by (*simp add: failures-def rdes, rel-auto*)

lemma *divergences-Chaos*: $dv\llbracket Chaos \rrbracket s = UNIV$
by (*simp add: divergences-def rdes, rel-auto*)

lemma *traces-Chaos*: $tr\llbracket Chaos \rrbracket s = \{\}$
by (*simp add: traces-def rdes closure usubst*)

lemma *divergences-cond*:
assumes P is NCSP Q is NCSP
shows $dv\llbracket P \triangleleft b \triangleright_R Q \rrbracket s = (if\ (\llbracket b \rrbracket_e s) \text{ then } dv\llbracket P \rrbracket s \text{ else } dv\llbracket Q \rrbracket s)$
by (*rdes-simp cls: assms, simp add: divergences-def traces-def rdes closure rpred assms, rel-auto*)

lemma *traces-cond*:
assumes P is NCSP Q is NCSP
shows $tr\llbracket P \triangleleft b \triangleright_R Q \rrbracket s = (if\ (\llbracket b \rrbracket_e s) \text{ then } tr\llbracket P \rrbracket s \text{ else } tr\llbracket Q \rrbracket s)$
by (*rdes-simp cls: assms, simp add: divergences-def traces-def rdes closure rpred assms, rel-auto*)

lemma *failures-cond*:
assumes P is NCSP Q is NCSP
shows $fl\llbracket P \triangleleft b \triangleright_R Q \rrbracket s = (if\ (\llbracket b \rrbracket_e s) \text{ then } fl\llbracket P \rrbracket s \text{ else } fl\llbracket Q \rrbracket s)$
by (*rdes-simp cls: assms, simp add: divergences-def failures-def rdes closure rpred assms, rel-auto*)

lemma *divergences-guard*:
assumes P is NCSP
shows $dv\llbracket g \&_u P \rrbracket s = (if\ (\llbracket g \rrbracket_e s) \text{ then } dv\llbracket g \&_u P \rrbracket s \text{ else } \{\})$
by (*rdes-simp cls: assms, simp add: divergences-def traces-def rdes closure rpred assms, rel-auto*)

lemma *traces-do*: $tr\llbracket do_C(e) \rrbracket s = \{(\llbracket e \rrbracket_e s, s)\}$
by (*rdes-simp, simp add: traces-def rdes closure rpred, rel-auto*)

lemma *failures-do*: $fl\llbracket do_C(e) \rrbracket s = \{(\llbracket \cdot \rrbracket, E) \mid E. \llbracket e \rrbracket_e s \notin E\}$
by (*rdes-simp, simp add: failures-def rdes closure rpred usubst, rel-auto*)

lemma *divergences-do*: $dv\llbracket do_C(e) \rrbracket s = \{\}$

by (rel-auto)

lemma *nil-least* [simp]:
 $\langle \rangle \leq_u x = \text{true}$ **by** rel-auto

lemma *minus-nil* [simp]:
 $xs - \langle \rangle = xs$ **by** rel-auto

lemma *wp-rea-circus-lemma-1*:
 assumes P is CRR $\$ref' \# P$
 shows $out\alpha \# P[\llbracket \langle s_0 \rangle, \langle t_0 \rangle / \$st', \$tr' \rrbracket]$
proof –
 have $out\alpha \# (CRR (\exists \$ref' \cdot P))[\llbracket \langle s_0 \rangle, \langle t_0 \rangle / \$st', \$tr' \rrbracket]$
 by (rel-auto)
 thus ?thesis
 by (simp add: Healthy-if assms(1) assms(2) ex-unrest)
qed

lemma *wp-rea-circus-lemma-2*:
 assumes P is CRR
 shows $in\alpha \# P[\llbracket \langle s_0 \rangle, \langle t_0 \rangle / \$st, \$tr \rrbracket]$
proof –
 have $in\alpha \# (CRR P)[\llbracket \langle s_0 \rangle, \langle t_0 \rangle / \$st, \$tr \rrbracket]$
 by (rel-auto)
 thus ?thesis
 by (simp add: Healthy-if assms ex-unrest)
qed

The meaning of reactive weakest precondition for Circus. $P \text{ wp}_r Q$ means that, whenever P terminates in a state s_0 having done the interaction trace t_0 , which is a prefix of the overall trace, then Q must be satisfied. This in particular means that the remainder of the trace after t_0 must not be a divergent behaviour of Q .

lemma *wp-rea-circus-form*:
 assumes P is CRR $\$ref' \# P$ Q is CRC
 shows $(P \text{ wp}_r Q) = (\forall (s_0, t_0) \cdot \langle t_0 \rangle \leq_u \$tr' \wedge P[\llbracket \langle s_0 \rangle, \langle t_0 \rangle / \$st', \$tr' \rrbracket] \Rightarrow_r Q[\llbracket \langle s_0 \rangle, \langle t_0 \rangle / \$st, \$tr \rrbracket])$
proof –
 have $(P \text{ wp}_r Q) = (\neg_r (\exists t_0 \cdot P[\llbracket \langle t_0 \rangle / \$tr' \rrbracket] ; (\neg_r Q)[\llbracket \langle t_0 \rangle / \$tr \rrbracket] \wedge \langle t_0 \rangle \leq_u \$tr'))$
 by (simp-all add: wp-rea-def R2-tr-middle closure RR-implies-R2 assms)
 also have $\dots = (\neg_r (\exists (s_0, t_0) \cdot P[\llbracket \langle s_0 \rangle, \langle t_0 \rangle / \$st', \$tr' \rrbracket] ; (\neg_r Q)[\llbracket \langle s_0 \rangle, \langle t_0 \rangle / \$st, \$tr \rrbracket] \wedge \langle t_0 \rangle \leq_u \$tr'))$
 by (rel-blast)
 also have $\dots = (\neg_r (\exists (s_0, t_0) \cdot P[\llbracket \langle s_0 \rangle, \langle t_0 \rangle / \$st', \$tr' \rrbracket] \wedge (\neg_r Q)[\llbracket \langle s_0 \rangle, \langle t_0 \rangle / \$st, \$tr \rrbracket] \wedge \langle t_0 \rangle \leq_u \$tr'))$
 by (simp add: seqr-to-conj add: wp-rea-circus-lemma-1 wp-rea-circus-lemma-2 assms closure conj-assoc)
 also have $\dots = (\forall (s_0, t_0) \cdot \neg_r P[\llbracket \langle s_0 \rangle, \langle t_0 \rangle / \$st', \$tr' \rrbracket] \vee \neg_r (\neg_r Q)[\llbracket \langle s_0 \rangle, \langle t_0 \rangle / \$st, \$tr \rrbracket] \vee \neg_r \langle t_0 \rangle \leq_u \$tr')$
 by (rel-auto)
 also have $\dots = (\forall (s_0, t_0) \cdot \neg_r P[\llbracket \langle s_0 \rangle, \langle t_0 \rangle / \$st', \$tr' \rrbracket] \vee \neg_r (\neg_r RR Q)[\llbracket \langle s_0 \rangle, \langle t_0 \rangle / \$st, \$tr \rrbracket] \vee \neg_r \langle t_0 \rangle \leq_u \$tr')$
 by (simp add: Healthy-if assms closure)
 also have $\dots = (\forall (s_0, t_0) \cdot \neg_r P[\llbracket \langle s_0 \rangle, \langle t_0 \rangle / \$st', \$tr' \rrbracket] \vee (RR Q)[\llbracket \langle s_0 \rangle, \langle t_0 \rangle / \$st, \$tr \rrbracket] \vee \neg_r \langle t_0 \rangle \leq_u \$tr')$
 by (rel-auto)
 also have $\dots = (\forall (s_0, t_0) \cdot \langle t_0 \rangle \leq_u \$tr' \wedge P[\llbracket \langle s_0 \rangle, \langle t_0 \rangle / \$st', \$tr' \rrbracket] \Rightarrow_r (RR Q)[\llbracket \langle s_0 \rangle, \langle t_0 \rangle / \$st, \$tr \rrbracket])$
 by (rel-auto)

also have ... = $(\forall (s_0, t_0) \cdot \ll t_0 \gg \leq_u \$tr' \wedge P[\ll s_0 \gg, \ll t_0 \gg / \$st', \$tr']) \Rightarrow_r Q[\ll s_0 \gg, \ll t_0 \gg / \$st, \$tr])$
 by (simp add: Healthy-if assms closure)
 finally show ?thesis .
 qed

lemma wp-rea-circus-form-alt:

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

shows $(P \text{ wp}_r Q) = (\forall (s_0, t_0) \cdot \$tr \hat{=}_u \ll t_0 \gg \leq_u \$tr' \wedge P[\ll s_0 \gg, \langle \rangle, \ll t_0 \gg / \$st', \$tr, \$tr'])$
 $\Rightarrow_r R1(Q[\ll s_0 \gg, \langle \rangle, \&tt - \ll t_0 \gg / \$st, \$tr, \$tr'])$

proof –

have $(P \text{ wp}_r Q) = R2(P \text{ wp}_r Q)$

by (simp add: CRC-implies-RR CRR-implies-RR Healthy-if RR-implies-R2 assms wp-rea-R2-closed)

also have ... = $R2(\forall (s_0, tr_0) \cdot \ll tr_0 \gg \leq_u \$tr' \wedge (RR P)[\ll s_0 \gg, \ll tr_0 \gg / \$st', \$tr']) \Rightarrow_r (RR Q)[\ll s_0 \gg, \ll tr_0 \gg / \$st, \$tr])$

by (simp add: wp-rea-circus-form assms closure Healthy-if)

also have ... = $(\exists tt_0 \cdot (\forall (s_0, tr_0) \cdot \ll tr_0 \gg \leq_u \ll tt_0 \gg \wedge (RR P)[\ll s_0 \gg, \langle \rangle, \ll tr_0 \gg / \$st', \$tr, \$tr'])$
 $\Rightarrow_r (RR Q)[\ll s_0 \gg, \ll tr_0 \gg, \ll tt_0 \gg / \$st, \$tr, \$tr'])$
 $\wedge \$tr' =_u \$tr \hat{=}_u \ll tt_0 \gg)$

by (simp add: R2-form, rel-auto)

also have ... = $(\exists tt_0 \cdot (\forall (s_0, tr_0) \cdot \ll tr_0 \gg \leq_u \ll tt_0 \gg \wedge (RR P)[\ll s_0 \gg, \langle \rangle, \ll tr_0 \gg / \$st', \$tr, \$tr'])$
 $\Rightarrow_r (RR Q)[\ll s_0 \gg, \langle \rangle, \ll tt_0 - tr_0 \gg / \$st, \$tr, \$tr'])$
 $\wedge \$tr' =_u \$tr \hat{=}_u \ll tt_0 \gg)$

by (rel-auto)

also have ... = $(\exists tt_0 \cdot (\forall (s_0, tr_0) \cdot \$tr \hat{=}_u \ll tr_0 \gg \leq_u \$tr' \wedge (RR P)[\ll s_0 \gg, \langle \rangle, \ll tr_0 \gg / \$st', \$tr, \$tr'])$
 $\Rightarrow_r (RR Q)[\ll s_0 \gg, \langle \rangle, \&tt - \ll tr_0 \gg / \$st, \$tr, \$tr'])$
 $\wedge \$tr' =_u \$tr \hat{=}_u \ll tt_0 \gg)$

by (rel-auto, (metis list-concat-minus-list-concat)+)

also have ... = $(\forall (s_0, tr_0) \cdot \$tr \hat{=}_u \ll tr_0 \gg \leq_u \$tr' \wedge (RR P)[\ll s_0 \gg, \langle \rangle, \ll tr_0 \gg / \$st', \$tr, \$tr'])$
 $\Rightarrow_r R1((RR Q)[\ll s_0 \gg, \langle \rangle, \&tt - \ll tr_0 \gg / \$st, \$tr, \$tr'])$

by (rel-auto, blast+)

also have ... = $(\forall (s_0, t_0) \cdot \$tr \hat{=}_u \ll t_0 \gg \leq_u \$tr' \wedge P[\ll s_0 \gg, \langle \rangle, \ll t_0 \gg / \$st', \$tr, \$tr'])$
 $\Rightarrow_r R1(Q[\ll s_0 \gg, \langle \rangle, \&tt - \ll t_0 \gg / \$st, \$tr, \$tr'])$

by (simp add: Healthy-if assms closure)

finally show ?thesis .

qed

lemma divergences-seq:

fixes $P :: ('s, 'e)$ action

assumes P is NCSP Q is NCSP

shows $dv[P ;; Q]s = dv[P]s \cup \{t_1 @ t_2 \mid t_1 \ t_2 \ s_0. (t_1, s_0) \in tr[P]s \wedge t_2 \in dv[Q]s_0\}$

(is ?lhs = ?rhs)

oops

lemma traces-seq:

fixes $P :: ('s, 'e)$ action

assumes P is NCSP Q is NCSP

shows $tr[P ;; Q]s =$

$\{(t_1 @ t_2, s') \mid t_1 \ t_2 \ s_0 \ s'. (t_1, s_0) \in tr[P]s \wedge (t_2, s') \in tr[Q]s_0$
 $\wedge (t_1 @ t_2) \notin dv[P]s$
 $\wedge (\forall (t, s_1) \in tr[P]s. t \leq t_1 @ t_2 \longrightarrow (t_1 @ t_2) - t \notin dv[Q]s_1) \}$

(is ?lhs = ?rhs)

proof

show ?lhs \subseteq ?rhs

proof (rdes-expand cls: assms, simp add: traces-def divergences-def rdes closure assms rdes-def unrest
 rpred usubst, auto)

fix $t :: 'e$ list and $s' :: 's$

let $? \sigma = [\$st \mapsto_s \langle s \rangle, \$st' \mapsto_s \langle s' \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle t \rangle]$
 assume
 $a1: '? \sigma \vdash (post_R P ;; post_R Q)'$ and
 $a2: '[\$st \mapsto_s \langle s \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle t \rangle] \vdash pre_R P'$ and
 $a3: '[\$st \mapsto_s \langle s \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle t \rangle] \vdash (post_R P \text{ wp}_r pre_R Q)'$
 from $a1$ have $'? \sigma \vdash (\exists tr_0 \cdot ((post_R P) \llbracket \langle tr_0 \rangle / \$tr' \rrbracket ;; (post_R Q) \llbracket \langle tr_0 \rangle / \$tr \rrbracket) \wedge \langle tr_0 \rangle \leq_u \$tr)'$
 by (simp add: R2-tr-middle assms closure)
 then obtain tr_0 where $p1: '? \sigma \vdash ((post_R P) \llbracket \langle tr_0 \rangle / \$tr' \rrbracket ;; (post_R Q) \llbracket \langle tr_0 \rangle / \$tr \rrbracket)'$ and $tr0: tr_0 \leq t$
 apply (simp add: usubst)
 apply (erule taut-shEx-elim)
 apply (simp add: unrest-all-circus-vars-st-st' closure unrest assms)
 apply (rel-auto)
 done
 from $p1$ have $'? \sigma \vdash (\exists st_0 \cdot (post_R P) \llbracket \langle tr_0 \rangle / \$tr' \rrbracket \llbracket \langle st_0 \rangle / \$st' \rrbracket ;; (post_R Q) \llbracket \langle tr_0 \rangle / \$tr \rrbracket \llbracket \langle st_0 \rangle / \$st \rrbracket)'$
 by (simp add: seqr-middle[of st, THEN sym])
 then obtain s_0 where $'? \sigma \vdash ((post_R P) \llbracket \langle s_0 \rangle, \langle tr_0 \rangle / \$st', \$tr' \rrbracket ;; (post_R Q) \llbracket \langle s_0 \rangle, \langle tr_0 \rangle / \$st, \$tr \rrbracket)'$
 apply (simp add: usubst)
 apply (erule taut-shEx-elim)
 apply (simp add: unrest-all-circus-vars-st-st' closure unrest assms)
 apply (rel-auto)
 done
 hence $'(([\$st \mapsto_s \langle s \rangle, \$st' \mapsto_s \langle s_0 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tr_0 \rangle] \vdash post_R P) ;; ([\$st \mapsto_s \langle s_0 \rangle, \$st' \mapsto_s \langle s' \rangle, \$tr \mapsto_s \langle tr_0 \rangle, \$tr' \mapsto_s \langle t \rangle] \vdash post_R Q))'$
 by (rel-auto)
 hence $'(([\$st \mapsto_s \langle s \rangle, \$st' \mapsto_s \langle s_0 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tr_0 \rangle] \vdash post_R P) \wedge ([\$st \mapsto_s \langle s_0 \rangle, \$st' \mapsto_s \langle s' \rangle, \$tr \mapsto_s \langle tr_0 \rangle, \$tr' \mapsto_s \langle t \rangle] \vdash post_R Q))'$
 by (simp add: seqr-to-conj unrest-any-circus-var assms closure unrest)
 hence $postP: '([\$st \mapsto_s \langle s \rangle, \$st' \mapsto_s \langle s_0 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tr_0 \rangle] \vdash post_R P)'$ and
 $postQ': '([\$st \mapsto_s \langle s_0 \rangle, \$st' \mapsto_s \langle s' \rangle, \$tr \mapsto_s \langle tr_0 \rangle, \$tr' \mapsto_s \langle t \rangle] \vdash post_R Q)'$
 by (rel-auto)+
 from $postQ'$ have $'[\$st \mapsto_s \langle s_0 \rangle, \$st' \mapsto_s \langle s' \rangle] \vdash [\$tr \mapsto_s \langle tr_0 \rangle, \$tr' \mapsto_s \langle tr_0 \rangle + (\langle t \rangle - \langle tr_0 \rangle)] \vdash post_R Q'$
 using $tr0$ by (rel-auto)
 hence $'[\$st \mapsto_s \langle s_0 \rangle, \$st' \mapsto_s \langle s' \rangle] \vdash [\$tr \mapsto_s 0, \$tr' \mapsto_s \langle t \rangle - \langle tr_0 \rangle] \vdash post_R Q'$
 by (simp add: R2-subst-tr closure assms)
 hence $postQ: '([\$st \mapsto_s \langle s_0 \rangle, \$st' \mapsto_s \langle s' \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle t - tr_0 \rangle] \vdash post_R Q)'$
 by (rel-auto)
 have $preP: '[\$st \mapsto_s \langle s \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tr_0 \rangle] \vdash pre_R P'$
 proof -
 have $(pre_R P) \llbracket 0, \langle tr_0 \rangle / \$tr, \$tr' \rrbracket \sqsubseteq (pre_R P) \llbracket 0, \langle t \rangle / \$tr, \$tr' \rrbracket$
 by (simp add: RC-prefix-refine closure assms $tr0$)
 hence $[\$st \mapsto_s \langle s \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle tr_0 \rangle] \vdash pre_R P \sqsubseteq [\$st \mapsto_s \langle s \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle t \rangle] \vdash pre_R P$
 by (rel-auto)
 thus ?thesis
 by (simp add: taut-refine-impl $a2$)
 qed
 have $preQ: '[\$st \mapsto_s \langle s_0 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle t - tr_0 \rangle] \vdash pre_R Q'$
 proof -
 from $postP$ $a3$ have $'[\$st \mapsto_s \langle s_0 \rangle, \$tr \mapsto_s \langle tr_0 \rangle, \$tr' \mapsto_s \langle t \rangle] \vdash pre_R Q'$
 apply (simp add: wp-rea-def)
 apply (rel-auto)
 using $tr0$ apply blast+

done
 hence $['\$st \mapsto_s \ll s_0 \gg] \dagger ['\$tr \mapsto_s \ll tr_0 \gg, \$tr' \mapsto_s \ll tr_0 \gg + (\ll t \gg - \ll tr_0 \gg)] \dagger pre_R Q'$
 by (rel-auto)

 hence $['\$st \mapsto_s \ll s_0 \gg] \dagger ['$tr \mapsto_s 0, \$tr' \mapsto_s \ll t \gg - \ll tr_0 \gg] \dagger pre_R Q'$
 by (simp add: R2-subst-tr closure assms)
 thus ?thesis
 by (rel-auto)
 qed

 from a2 have ndiv: $\neg ['\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg] \dagger (\neg_r pre_R P)'$
 by (rel-auto)

 have t-minus-tr0: $tr_0 @ (t - tr_0) = t$
 using append-minus tr0 by blast

 from a3
 have wpr: $\bigwedge t_0 s_1.$
 $['\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_0 \gg] \dagger pre_R P' \implies$
 $['\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_0 \gg] \dagger post_R P' \implies$
 $t_0 \leq t \implies ['\$st \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t - t_0 \gg] \dagger (\neg_r pre_R Q)' \implies False$
 proof –
 fix $t_0 s_1$
 assume b:
 $['\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_0 \gg] \dagger pre_R P'$
 $['\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_0 \gg] \dagger post_R P'$
 $t_0 \leq t$
 $['\$st \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t - t_0 \gg] \dagger (\neg_r pre_R Q)'$

 from a3 have c: $\forall (s_0, t_0) \cdot \ll t_0 \gg \leq_u \ll t \gg$
 $\wedge ['\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_0 \gg] \dagger post_R P$
 $\implies ['\$st \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg - \ll t_0 \gg] \dagger pre_R Q'$
 by (simp add: wp-rea-circus-form-alt[of post_R P pre_R Q] closure assms unrest usubst)
 (rel-simp)

 from c b(2-4) show False
 by (rel-auto)
 qed

 show $\exists t_1 t_2.$
 $t = t_1 @ t_2 \wedge$
 $(\exists s_0. ['\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_1 \gg] \dagger pre_R P \wedge$
 $['\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_1 \gg] \dagger post_R P' \wedge$
 $['\$st \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_2 \gg] \dagger pre_R Q \wedge$
 $['\$st \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_2 \gg] \dagger post_R Q' \wedge$
 $\neg ['\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_1 @ t_2 \gg] \dagger (\neg_r pre_R P)' \wedge$
 $(\forall t_0 s_1. ['\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_0 \gg] \dagger pre_R P \wedge$
 $['\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_0 \gg] \dagger post_R P' \longrightarrow$
 $t_0 \leq t_1 @ t_2 \longrightarrow \neg ['\$st \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll (t_1 @ t_2) - t_0 \gg] \dagger (\neg_r$
 $pre_R Q)'))$
 apply (rule-tac $x=tr_0$ in exI)
 apply (rule-tac $x=(t - tr_0)$ in exI)
 apply (auto)
 using tr0 apply auto[1]
 apply (rule-tac $x=s_0$ in exI)

apply (*auto intro:wpr simp add: taut-conj preP preQ postP postQ ndiv wpr t-minus-tr0*)
done
qed

show $?rhs \subseteq ?lhs$

proof (*rdes-expand cls: assms, simp add: traces-def divergences-def rdes closure assms rdes-def unrest rpred usubst, auto*)

fix $t_1 t_2 :: 'e \text{ list}$ **and** $s_0 s' :: 's$

assume

$a1: \neg '[\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_1 @ t_2 \gg] \dagger (\neg_r \text{pre}_R P)'$ **and**
 $a2: '[\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_1 \gg] \dagger \text{pre}_R P'$ **and**
 $a3: '[\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_1 \gg] \dagger \text{post}_R P'$ **and**
 $a4: '[\$st \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_2 \gg] \dagger \text{pre}_R Q'$ **and**
 $a5: '[\$st \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_2 \gg] \dagger \text{post}_R Q'$ **and**
 $a6: \forall t s_1. '[\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg] \dagger \text{pre}_R P \wedge$
 $[\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg] \dagger \text{post}_R P' \longrightarrow$
 $t \leq t_1 @ t_2 \longrightarrow \neg '[\$st \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll (t_1 @ t_2) - t \gg] \dagger (\neg_r \text{pre}_R Q)'$

from $a1$ **have** $\text{pre}P: '[\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_1 @ t_2 \gg] \dagger (\text{pre}_R P)'$
by (*simp add: taut-not unrest-all-circus-vars-st assms closure unrest, rel-auto*)

have $'[\$st \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \ll t_1 \gg, \$tr' \mapsto_s \ll t_1 \gg + \ll t_2 \gg] \dagger \text{post}_R Q'$

proof –

have $[\$st \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_2 \gg] \dagger \text{post}_R Q =$
 $[\$st \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg] \dagger [\$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_2 \gg] \dagger \text{post}_R Q$

by *rel-auto*

also have $\dots = [\$st \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg] \dagger [\$tr \mapsto_s \ll t_1 \gg, \$tr' \mapsto_s \ll t_1 \gg + \ll t_2 \gg] \dagger \text{post}_R Q$

by (*simp add: R2-subst-tr assms closure, rel-auto*)

finally show $?thesis$ **using** $a5$

by (*rel-auto*)

qed

with $a3$

have $\text{post}PQ: '[\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_1 @ t_2 \gg] \dagger (\text{post}_R P ;; \text{post}_R Q)'$
by (*rel-auto, meson Prefix-Order.prefixI*)

have $'[\$st \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \ll t_1 \gg, \$tr' \mapsto_s \ll t_1 \gg + \ll t_2 \gg] \dagger \text{pre}_R Q'$

proof –

have $[\$st \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \ll t_1 \gg, \$tr' \mapsto_s \ll t_1 \gg + \ll t_2 \gg] \dagger \text{pre}_R Q =$
 $[\$st \mapsto_s \ll s_0 \gg] \dagger [\$tr \mapsto_s \ll t_1 \gg, \$tr' \mapsto_s \ll t_1 \gg + \ll t_2 \gg] \dagger \text{pre}_R Q$

by *rel-auto*

also have $\dots = [\$st \mapsto_s \ll s_0 \gg] \dagger [\$tr \mapsto_s 0, \$tr' \mapsto_s \ll t_2 \gg] \dagger \text{pre}_R Q$

by (*simp add: R2-subst-tr assms closure*)

finally show $?thesis$ **using** $a4$

by (*rel-auto*)

qed

from $a6$

have $a6': \bigwedge t s_1. \ll t \leq t_1 @ t_2; '[\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg] \dagger \text{pre}_R P'; '[\$st \mapsto_s \ll s \gg,$
 $\$st' \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg] \dagger \text{post}_R P' \rrbracket \implies$
 $'[\$st \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll (t_1 @ t_2) - t \gg] \dagger \text{pre}_R Q'$

apply (*subst (asm) taut-not*)

apply (*simp add: unrest-all-circus-vars-st assms closure unrest*)

apply (*rel-auto*)

done

```

have wpR: ‘ $\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_1 @ t_2 \gg$ ’  $\dagger (post_R P \text{ wp}_r pre_R Q)$ ‘
proof –
  have  $\bigwedge s_1 t_0. \llbracket t_0 \leq t_1 @ t_2; ‘\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_0 \gg$ ’  $\dagger post_R P$ ‘
  ]
     $\implies ‘\$st \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll (t_1 @ t_2) - t_0 \gg$ ’  $\dagger pre_R Q$ ‘
  proof –
    fix s1 t0
    assume c:  $t_0 \leq t_1 @ t_2$  ‘ $\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_0 \gg$ ’  $\dagger post_R P$ ‘

    have preP': ‘ $\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_0 \gg$ ’  $\dagger pre_R P$ ‘
    proof –
      have  $(pre_R P) \llbracket 0, \ll t_0 \gg / \$tr, \$tr' \rrbracket \sqsubseteq (pre_R P) \llbracket 0, \ll t_1 @ t_2 \gg / \$tr, \$tr' \rrbracket$ 
      by (simp add: RC-prefix-refine-closure-assms c)
      hence  $[\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_0 \gg] \dagger pre_R P \sqsubseteq [\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_1 @ t_2 \gg] \dagger pre_R P$ 
      by (rel-auto)
      thus ?thesis
      by (simp add: taut-refine-impl preP)
    qed

    with c a3 preP a6 '[of t0 s1] show ‘ $\$st \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll (t_1 @ t_2) - t_0 \gg$ ’  $\dagger pre_R$ 
    Q‘
      by (simp)
    qed

  thus ?thesis
  apply (simp-all add: wp-rea-circus-form-alt-assms-closure-unrest-usubst-rea-impl-alt-def)
  apply (simp add: R1-def-usubst-tcontr-alt-def)
  apply (auto intro!: taut-shAll-intro-2)
  apply (rule taut-impl-intro)
  apply (simp add: unrest-all-circus-vars-st-st'-unrest-closure-assms)
  apply (rel-simp)
  done
qed
show ‘ $([\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_1 @ t_2 \gg] \dagger pre_R P \wedge$ 
 $[\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_1 @ t_2 \gg] \dagger (post_R P \text{ wp}_r pre_R Q)) \wedge$ 
 $[\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_1 @ t_2 \gg] \dagger (post_R P ;; post_R Q)$ ‘
by (auto simp add: taut-conj preP postPQ wpR)
qed
qed

lemma Cons-minus [simp]:  $(a \# t) - [a] = t$ 
by (metis append-Cons append-Nil append-minus)

lemma traces-prefix:
  assumes  $P \text{ is NCSP}$ 
  shows  $tr \llbracket a \rightarrow P \rrbracket s = \{(a \# t, s') \mid t s'. (t, s') \in tr \llbracket P \rrbracket s\}$ 
  apply (auto simp add: PrefixCSP-def-traces-seq-traces-do-divergences-do-lit.rep-eq-assms-closure-Healthy-if-trace-divergence-disj)
  apply (meson assms-trace-divergence-disj)
  done

```

13.3 Deadlock Freedom

definition $DF :: 'e \text{ set} \Rightarrow ('s, 'e) \text{ action} \text{ where}$
 $DF(A) = (\mu_C X \cdot (\bigcap_{a \in A} a \rightarrow Skip) ;; X)$

lemma $DF\text{-}CSP \text{ [closure]: } A \neq \{\} \implies DF(A) \text{ is } CSP$
by (*simp add: DF-def closure unrest*)

end

14 Meta theory for Circus

theory *utp-circus*

imports

utp-circus-core

utp-circus-rel

utp-circus-healths

utp-circus-contracts

utp-circus-extchoice

utp-circus-actions

utp-circus-prefix

utp-circus-recursion

utp-circus-traces

utp-circus-parallel

utp-circus-fdsem

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.