

Stateful-Failure Reactive Designs in Isabelle/UTP

Simon Foster James Baxter Ana Cavalcanti Jim Woodcock

April 27, 2018

Abstract

Stateful-Failure Reactive Designs specialise reactive design contracts with failures traces, as present in languages like CSP and Circus. A failure trace consists of a sequence of events and a refusal set. It intuitively represents a quiescent observation, where certain events have previously occurred, and others are currently being accepted. Following the UTP book, we add an observational variable to represent refusal sets, and healthiness conditions that ensure their well-formedness. Using these, we also specialise our theory of reactive relations with operators to characterise both completed and quiescent interactions, and an accompanying equational theory. We use these to define the core operators — including assignment, event occurrence, and external choice — and specialise our proof strategy to support these. We also demonstrate a link with the CSP failures-divergences semantic model.

Contents

1	Introduction	2
2	Stateful-Failure Core Types	2
2.1	SFRD Alphabet	3
2.2	Basic laws	3
2.3	Unrestriction laws	4
3	Stateful-Failure Reactive Relations	5
3.1	Healthiness Conditions	5
3.2	Closure Properties	6
3.3	Introduction laws	10
3.4	Weakest Precondition	11
3.5	Trace Substitution	12
3.6	Initial Interaction	13
3.7	Enabled Events	15
3.8	Completed Trace Interaction	17
4	Stateful-Failure Healthiness Conditions	19
5	Definitions	19
5.1	Healthiness condition properties	20
5.2	CSP theories	31
5.3	Algebraic laws	32
6	Stateful-Failure Reactive Contracts	32

7	External Choice	34
7.1	Definitions and syntax	34
7.2	Basic laws	35
7.3	Algebraic laws	35
7.4	Reactive design calculations	35
7.5	Productivity and Guardedness	43
7.6	Algebraic laws	44
8	Stateful-Failure Programs	47
8.1	Conditionals	47
8.2	Guarded commands	47
8.3	Alternation	47
8.4	While Loops	48
8.5	Assignment	48
8.6	Assignment with update	49
8.7	State abstraction	50
8.8	Assumptions	50
8.9	Guards	50
8.10	Basic events	54
8.11	Event prefix	55
8.12	Guarded external choice	58
8.13	Input prefix	58
8.14	Algebraic laws	59
9	Recursion in Stateful-Failures	61
9.1	Fixed-points	61
9.2	Example action expansion	63
10	Linking to the Failures-Divergences Model	63
10.1	Failures-Divergences Semantics	63
10.2	Circus Operators	65
10.3	Deadlock Freedom	71
11	Meta-theory for Stateful-Failure Reactive Designs	71

1 Introduction

This document contains a mechanisation in Isabelle/UTP [1] of an specialisation of stateful reactive designs with refusal information, as present in languages like Circus [2].

2 Stateful-Failure Core Types

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

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

interpretation *alphabet-csp-rel*:
 $\text{lens-interp } \lambda(\text{ok}, \text{ok}', \text{wait}, \text{wait}', \text{tr}, \text{tr}', \text{m}, \text{m}').$
 $(\text{ok}, \text{ok}', \text{wait}, \text{wait}', \text{tr}, \text{tr}', \text{ref}_v \text{ m}, \text{ref}_v \text{ m}', \text{more m}, \text{more m}')$
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*: $R2c (\text{\$tr'} =_u \text{\$tr} \hat{\text{~}}_u \langle \lceil a \rceil_{S<} \rangle) = (\text{\$tr'} =_u \text{\$tr} \hat{\text{~}}_u \langle \lceil a \rceil_{S<} \rangle)$

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 Stateful-Failure Reactive Relations

theory *utp-sfrd-rel*

imports *utp-sfrd-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 $(\exists x \cdot P(x)) \text{ is CRR}$
proof –
have $\text{CRR}(\exists x \cdot \text{CRR}(P(x))) = (\exists x \cdot \text{CRR}(P(x)))$
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 $(\bigsqcup i \cdot P(i)) \text{ is CRR}$
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 $(\bigcap i \cdot P(i)) \text{ is CRR}$
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 *seq-CRR-closed* [closure]:
assumes $P \text{ is } CRR \ Q \text{ is } RR$
shows $(P ;; Q) \text{ is } CRR$
by (*rule CRR-intro, simp-all add: unrest assms closure*)

lemma *tr-extend-seqr-lit* [rdes]:
fixes $P :: ('s, 'e) \text{ action}$
assumes $\$ok \# P \ \$wait \# P \ \$ref \# P$
shows $(\$tr' =_u \$tr \hat{\ }_u \langle \ll a \gg \rangle \wedge \$st' =_u \$st) ;; P = P[\$tr \hat{\ }_u \langle \ll a \gg \rangle / \$tr]$
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 Weakest Precondition

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 $out\alpha \# P[\ll s_0 \gg, \ll t_0 \gg / \$st', \$tr']$

proof –

have $out\alpha \# (CRR (\exists \$ref' \cdot P))[\ll s_0 \gg, \ll t_0 \gg / \$st', \$tr']$

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 $in\alpha \# P[\ll s_0 \gg, \ll t_0 \gg / \$st, \$tr]$

proof –

have $in\alpha \# (CRR P)[\ll s_0 \gg, \ll t_0 \gg / \$st, \$tr]$

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 $(P \text{ wp}_r Q) = (\forall (s_0, t_0) \cdot \ll t_0 \gg \leq_u \$tr' \wedge P[\ll s_0 \gg, \ll t_0 \gg / \$st', \$tr']) \Rightarrow_r Q[\ll s_0 \gg, \ll t_0 \gg / \$st, \$tr])$

proof –

have $(P \text{ wp}_r Q) = (\neg_r (\exists t_0 \cdot P[\ll t_0 \gg / \$tr']) ; (\neg_r Q)[\ll t_0 \gg / \$tr] \wedge \ll t_0 \gg \leq_u \$tr')$

by (*simp-all add: wp-rea-def R2-tr-middle closure assms*)

also have $\dots = (\neg_r (\exists (s_0, t_0) \cdot P[\ll s_0 \gg, \ll t_0 \gg / \$st', \$tr']) ; (\neg_r Q)[\ll s_0 \gg, \ll t_0 \gg / \$st, \$tr] \wedge \ll t_0 \gg \leq_u \$tr')$

by (*rel-blast*)

also have $\dots = (\neg_r (\exists (s_0, t_0) \cdot P[\ll s_0 \gg, \ll t_0 \gg / \$st', \$tr']) \wedge (\neg_r Q)[\ll s_0 \gg, \ll t_0 \gg / \$st, \$tr] \wedge \ll t_0 \gg \leq_u \$tr')$

by (*simp add: seqr-to-conj add: wp-rea-circus-lemma-1 wp-rea-circus-lemma-2 assms closure conj-assoc*)

also have $\dots = (\forall (s_0, t_0) \cdot \neg_r P[\ll s_0 \gg, \ll t_0 \gg / \$st', \$tr']) \vee \neg_r (\neg_r Q)[\ll s_0 \gg, \ll t_0 \gg / \$st, \$tr] \vee \neg_r \ll t_0 \gg \leq_u \$tr')$

by (*rel-auto*)

also have $\dots = (\forall (s_0, t_0) \cdot \neg_r P[\ll s_0 \gg, \ll t_0 \gg / \$st', \$tr']) \vee \neg_r (\neg_r RR Q)[\ll s_0 \gg, \ll t_0 \gg / \$st, \$tr] \vee \neg_r \ll t_0 \gg \leq_u \$tr')$

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

also have $\dots = (\forall (s_0, t_0) \cdot \neg_r P[\ll s_0 \gg, \ll t_0 \gg / \$st', \$tr']) \vee (RR Q)[\ll s_0 \gg, \ll t_0 \gg / \$st, \$tr] \vee \neg_r \ll t_0 \gg \leq_u \$tr')$

by (*rel-auto*)

also have ... = $(\forall (s_0, t_0) \cdot \langle t_0 \rangle \leq_u \$tr' \wedge P[\langle s_0 \rangle, \langle t_0 \rangle / \$st', \$tr']) \Rightarrow_r (RR Q)[\langle s_0 \rangle, \langle t_0 \rangle / \$st, \$tr]$
 by (rel-auto)
 also have ... = $(\forall (s_0, t_0) \cdot \langle t_0 \rangle \leq_u \$tr' \wedge P[\langle s_0 \rangle, \langle t_0 \rangle / \$st', \$tr']) \Rightarrow_r Q[\langle s_0 \rangle, \langle t_0 \rangle / \$st, \$tr]$
 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 $(P \text{ wp}_r Q) = (\forall (s_0, t_0) \cdot \$tr \hat{=} \langle t_0 \rangle \leq_u \$tr' \wedge P[\langle s_0 \rangle, \langle \rangle, \langle t_0 \rangle / \$st', \$tr, \$tr'])$
 $\Rightarrow_r R1(Q[\langle s_0 \rangle, \langle \rangle, \&tt - \langle t_0 \rangle / \$st, \$tr, \$tr'])$

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 ... = $R2(\forall (s_0, tr_0) \cdot \langle tr_0 \rangle \leq_u \$tr' \wedge (RR P)[\langle s_0 \rangle, \langle tr_0 \rangle / \$st', \$tr']) \Rightarrow_r (RR Q)[\langle s_0 \rangle, \langle tr_0 \rangle / \$st, \$tr]$

by (simp add: wp-rea-circus-form assms closure Healthy-if)

also have ... = $(\exists tt_0 \cdot (\forall (s_0, tr_0) \cdot \langle tr_0 \rangle \leq_u \langle tt_0 \rangle \wedge (RR P)[\langle s_0 \rangle, \langle \rangle, \langle tr_0 \rangle / \$st', \$tr, \$tr'])$
 $\Rightarrow_r (RR Q)[\langle s_0 \rangle, \langle tr_0 \rangle, \langle tt_0 \rangle / \$st, \$tr, \$tr'])$
 $\wedge \$tr' =_u \$tr \hat{=} \langle tt_0 \rangle$

by (simp add: R2-form, rel-auto)

also have ... = $(\exists tt_0 \cdot (\forall (s_0, tr_0) \cdot \langle tr_0 \rangle \leq_u \langle tt_0 \rangle \wedge (RR P)[\langle s_0 \rangle, \langle \rangle, \langle tr_0 \rangle / \$st', \$tr, \$tr'])$
 $\Rightarrow_r (RR Q)[\langle s_0 \rangle, \langle \rangle, \langle tt_0 - tr_0 \rangle / \$st, \$tr, \$tr'])$
 $\wedge \$tr' =_u \$tr \hat{=} \langle tt_0 \rangle$

by (rel-auto)

also have ... = $(\exists tt_0 \cdot (\forall (s_0, tr_0) \cdot \$tr \hat{=} \langle tr_0 \rangle \leq_u \$tr' \wedge (RR P)[\langle s_0 \rangle, \langle \rangle, \langle tr_0 \rangle / \$st', \$tr, \$tr'])$
 $\Rightarrow_r (RR Q)[\langle s_0 \rangle, \langle \rangle, \&tt - \langle tr_0 \rangle / \$st, \$tr, \$tr'])$
 $\wedge \$tr' =_u \$tr \hat{=} \langle tt_0 \rangle$

by (rel-auto, (metis list-concat-minus-list-concat)+)

also have ... = $(\forall (s_0, tr_0) \cdot \$tr \hat{=} \langle tr_0 \rangle \leq_u \$tr' \wedge (RR P)[\langle s_0 \rangle, \langle \rangle, \langle tr_0 \rangle / \$st', \$tr, \$tr'])$
 $\Rightarrow_r R1((RR Q)[\langle s_0 \rangle, \langle \rangle, \&tt - \langle tr_0 \rangle / \$st, \$tr, \$tr'])$

by (rel-auto, blast+)

also have ... = $(\forall (s_0, t_0) \cdot \$tr \hat{=} \langle t_0 \rangle \leq_u \$tr' \wedge P[\langle s_0 \rangle, \langle \rangle, \langle t_0 \rangle / \$st', \$tr, \$tr'])$
 $\Rightarrow_r R1(Q[\langle s_0 \rangle, \langle \rangle, \&tt - \langle t_0 \rangle / \$st, \$tr, \$tr'])$

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 $(P \text{ wp}_r Q) = (\forall (s_0, t_0) \cdot \$tr \hat{=} \langle t_0 \rangle \leq_u \$tr' \wedge P[\langle s_0 \rangle, \langle \rangle, \langle t_0 \rangle / \$st', \$tr, \$tr'])$
 $\Rightarrow_r R1(Q[\langle s_0 \rangle, \langle \rangle, \&tt - \langle t_0 \rangle / \$st, \$tr, \$tr'])$

oops

3.5 Trace Substitution

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

where [upred-defs]: $P[\![v]\!]_t = (P[\![\&tt - [v]_{S<} / \&tt]\!] \wedge \$tr + [v]_{S<} \leq_u \$tr')$

lemma unrest-trace-subst [unrest]:

$\llbracket \text{mwb-lens } x; x \bowtie (\$tr)_v; x \bowtie (\$tr')_v; x \bowtie (\$st)_v; x \# P \rrbracket \Longrightarrow x \# P[\![v]\!]_t$

by (simp add: trace-subst-def lens-indep-sym unrest)

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

assumes P is RR

shows $P[\![v]\!]_t$ is RR

proof –

```

have (RR P)⟦v⟧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⟦v⟧t is CRR
  by (rule CRR-intro, simp-all add: closure assms unrest)

```

```

lemma tsubst-nil [usubst]:
  assumes P is CRR
  shows P⟦⟨⟩⟧t = P
proof -
  have (CRR P)⟦⟨⟩⟧t = CRR P
  by (rel-auto)
  thus ?thesis
  by (simp add: Healthy-if assms)
qed

```

```

lemma tsubst-false [usubst]: false⟦y⟧t = false
  by rel-auto

```

```

lemma cond-rea-tt-subst [usubst]:
  (P ◁ b ▷R Q)⟦v⟧t = (P⟦v⟧t ◁ b ▷R Q⟦v⟧t)
  by (rel-auto)

```

```

lemma tsubst-conj [usubst]: (P ∧ Q)⟦v⟧t = (P⟦v⟧t ∧ Q⟦v⟧t)
  by (rel-auto)

```

```

lemma tsubst-disj [usubst]: (P ∨ Q)⟦v⟧t = (P⟦v⟧t ∨ Q⟦v⟧t)
  by (rel-auto)

```

```

lemma rea-subst-R1-closed [closure]: P⟦v⟧t is R1
  apply (rel-auto) using le-add order.trans by blast

```

```

lemma tsubst-UINF-ind [usubst]: (⋀ i • P(i))⟦v⟧t = (⋀ i • (P(i))⟦v⟧t)
  by (rel-auto)

```

3.6 Initial Interaction

definition *rea-init* :: 's upred ⇒ ('t::trace, 's) uexpr ⇒ ('s, 't, 'α, 'β) rel-rsp (I'(-,-)) **where**
[upred-defs]: $\mathcal{I}(s, t) = ([s]_{S<} \wedge \$tr + \lceil t \rceil_{S<} \leq_u \$tr')$

$\mathcal{I}(s, t)$ 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]:
  ⟦ x ⋈ ($tr)v; x ⋈ ($tr')v; x ⋈ ($st)v ⟧ ⇒ x # 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*]: $\mathcal{I}(\text{true}, \langle \rangle) = \text{true}_r$
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* [*ustbst*]: $[P]_{S<} \llbracket t \rrbracket_t = \mathcal{I}(P, t)$
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*)

lemma *R4-csp-init* [*rpred*]: $R4(\mathcal{I}(s, \text{bop Cons } x \text{ } xs)) = \mathcal{I}(s, \text{bop Cons } x \text{ } xs)$
using *less-list-def* **by** (*rel-blast*)

lemma *R5-csp-init* [*rpred*]: $R5(\mathcal{I}(s, \text{bop Cons } x \text{ } xs)) = \text{false}$
by (*rel-auto*)

lemma *R4-trace-subst* [*rpred*]:
 $R4(P\llbracket \text{bop Cons } x \text{ } xs \rrbracket_t) = P\llbracket \text{bop Cons } x \text{ } xs \rrbracket_t$
using *le-imp-less-or-eq* **by** (*rel-blast*)

lemma *R5-trace-subst* [*rpred*]:
 $R5(P\llbracket \text{bop Cons } x \text{ } xs \rrbracket_t) = \text{false}$
by (*rel-auto*)

3.7 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
 $[\text{upred-defs}]: \mathcal{E}(s, t, E) = ([s]_{S<} \wedge \$tr' =_u \$tr \hat{_u} [t]_{S<} \wedge (\forall e \in [E]_{S<} \cdot \llbracket e \rrbracket \notin_u \$ref'))$

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*]: $\mathcal{E}(s, t_2, e)\llbracket t_1 \rrbracket_t = \mathcal{E}(s, t_1 \hat{_u} t_2, e)$
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*]:

$\mathcal{E}(s, \langle \rangle, r) \triangleleft \$tr' =_u \$tr \triangleright false = \mathcal{E}(s, \langle \rangle, r)$
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-conj* [*rpred*]:
 $(\mathcal{E}(s, t, E_1) \wedge \mathcal{E}(s, t, E_2)) = \mathcal{E}(s, t, E_1 \cup_u E_2)$
by (*rel-auto*)

lemma *csp-enable-cond* [*rpred*]:
 $\mathcal{E}(s_1, t_1, E_1) \triangleleft b \triangleright_R \mathcal{E}(s_2, t_2, E_2) = \mathcal{E}(s_1 \triangleleft b \triangleright s_2, t_1 \triangleleft b \triangleright t_2, E_1 \triangleleft b \triangleright E_2)$
by (*rel-auto*)

lemma *csp-enable-rea-assm* [*rpred*]:
 $[b]^\top_r ;; \mathcal{E}(s, t, E) = \mathcal{E}(b \wedge s, t, E)$
by (*rel-auto*)

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

lemma *csp-enable-nothing*: $\mathcal{E}(true, \langle \rangle, \{\}_u) = (\$tr' =_u \$tr)$
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*]:
 $\mathcal{E}(s(x), t(x), E(x)) \llbracket x \rightarrow \lceil v \rceil_{S\leftarrow} \rrbracket = \mathcal{E}(s(x) \llbracket x \rightarrow v \rrbracket, t(x) \llbracket x \rightarrow v \rrbracket, E(x) \llbracket x \rightarrow v \rrbracket)$
by (*rel-auto*)

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

lemma *conj-csp-enable* [*rpred*]: $(\mathcal{E}(b_1, t, E_1) \wedge \mathcal{E}(b_2, t, E_2)) = \mathcal{E}(b_1 \wedge b_2, t, E_1 \cup_u E_2)$
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*)

lemma *R4-csp-enable-nil* [*rpred*]:
 $R4(\mathcal{E}(s, \langle \rangle, E)) = false$
by (*rel-auto*)

lemma *R5-csp-enable-nil* [*rpred*]:
 $R5(\mathcal{E}(s, \langle \rangle, E)) = \mathcal{E}(s, \langle \rangle, E)$
by (*rel-auto*)

lemma *R4-csp-enable-Cons* [*rpred*]:
 $R4(\mathcal{E}(s, \text{bop Cons } x \text{ } xs, E)) = \mathcal{E}(s, \text{bop Cons } x \text{ } xs, E)$
by (*rel-auto*, *simp add: Prefix-Order.strict-prefixI'*)

lemma *R5-csp-enable-Cons* [rpred]:
 $R5(\mathcal{E}(s, \text{bop } \text{Cons } x \text{ } xs, E)) = \text{false}$
by (*rel-auto*)

3.8 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-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 *csp-do-R4-closed* [closure]:
 $\Phi(b, \sigma, \text{bop } \text{Cons } x \text{ } xs)$ *is R4*
by (*rel-auto, simp add: Prefix-Order.strict-prefixI'*)

lemma *st-pred-conj-csp-do* [rpred]:
 $([b]_{S<} \wedge \Phi(s, \sigma, t)) = \Phi(b \wedge s, \sigma, t)$
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]:

$\Phi(s_1, \sigma, t_1) ;; \mathcal{E}(s_2, t_2, e) = \mathcal{E}(s_1 \wedge \sigma \dagger s_2, t_1 \hat{\wedge}_u(\sigma \dagger t_2), (\sigma \dagger e))$
by (simp add: rpred closure usubst)

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

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

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

$\Phi(s_1, \sigma, t_1) \triangleleft b \triangleright_R \Phi(s_2, \varrho, t_2) = \Phi(s_1 \triangleleft b \triangleright s_2, \sigma \triangleleft b \triangleright_s \varrho, t_1 \triangleleft b \triangleright t_2)$
by (rel-auto)

lemma *rea-assm-csp-do* [rpred]:

$[b]^\top_r ;; \Phi(s, \sigma, t) = \Phi(b \wedge s, \sigma, t)$
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 $\Phi(\text{true}, \text{id}, t) ;; \text{CRR}(P) = (\text{CRR } P) \llbracket t \rrbracket_t$
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 $(\llbracket \langle \sigma \rangle_a \rrbracket_S \wedge \text{\$tr}' =_u \text{\$tr} \hat{\wedge}_u \llbracket t \rrbracket_{S<}) ;; P = (\llbracket \sigma \rrbracket_{S\sigma} \dagger P) \llbracket \text{\$tr} \hat{\wedge}_u \llbracket t \rrbracket_{S<} / \text{\$tr} \rrbracket$
using assms by (rel-auto, meson)

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

fixes *P* :: ('σ, 'φ) action
assumes *P* is CRR
shows $\Phi(s, \sigma, t) \text{ wp}_r P = (\mathcal{I}(s, t) \Rightarrow_r (\llbracket \sigma \rrbracket_{S\sigma} \dagger P) \llbracket t \rrbracket_t)$

proof –

have $\Phi(s, \sigma, t) \text{ wp}_r \text{CRR}(P) = (\mathcal{I}(s, t) \Rightarrow_r (\llbracket \sigma \rrbracket_{S\sigma} \dagger \text{CRR}(P)) \llbracket t \rrbracket_t)$
by (rel-blast)

thus ?thesis

by (simp add: assms Healthy-if)

qed

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

$\Phi(\text{true}, \text{id}, t) \hat{\wedge} (\text{Suc } i) = \Phi(\text{true}, \text{id}, \text{iter}[\text{Suc } i](t))$
by (induct i, (rel-auto)+)

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

assumes *P* is CRR
shows $\Phi(\text{true}, \text{id}, t) \hat{\wedge} i ;; P = \Phi(\text{true}, \text{id}, \text{iter}[i](t)) ;; P$
apply (cases i)
apply (simp-all add: rpred usubst assms closure)
done

lemma *wp-rea-csp-do-skip* [*wp*]:
fixes $Q :: ('σ, 'φ)$ *action*
assumes P *is CRR*
shows $\Phi(s, id, t) \text{ wp}_r P = (\mathcal{I}(s, t) \Rightarrow_r P \llbracket t \rrbracket_t)$
proof –
have $\Phi(s, id, t) \text{ wp}_r P = \Phi(s, id, t) \text{ wp}_r P$
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 Stateful-Failure Healthiness Conditions

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

5 Definitions

We here define extra healthiness conditions for stateful-failure reactive designs.

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*]: $SKIP = \mathbf{R}_s(\exists \$ref \cdot CSP1(II))$

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

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

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

definition $CSP4 :: (('σ, 'φ) \text{ st-csp} \times ('σ, 'φ) \text{ st-csp}) \text{ health}$ **where**
[*upred-defs*]: $CSP4(P) = (P ;; Skip)$

definition $NCSP :: (('σ, 'φ) \text{ st-csp} \times ('σ, 'φ) \text{ st-csp}) \text{ health}$ **where**

[upred-defs]: $NCSP = CSP3 \circ CSP4 \circ CSP$

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-is-Skip*: $SKIP = Skip$
by (*rel-auto*)

theorem *STOP-is-Stop*: $STOP = Stop$
by (*rel-auto*)

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

lemma *Skip-is-CSP* [*closure*]:
Skip is CSP
by (*simp add: Skip-def RHS-design-is-SRD unrest*)

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

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

lemma *Stop-is-CSP* [*closure*]:
Stop is CSP
by (*simp add: Stop-def RHS-design-is-SRD unrest*)

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

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

lemma *preR-Skip* [*rdes*]: $pre_R(Skip) = true_r$
by (*rel-auto*)

lemma *periR-Skip* [*rdes*]: $peri_R(Skip) = false$
by (*rel-auto*)

lemma *postR-Skip* [*rdes*]: $post_R(Skip) = \Phi(true, id, \langle \rangle)$
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 *Skip* ;; $P = \mathbf{R}_s ((\forall \$ref \cdot pre_R P) \vdash (\exists \$ref \cdot cmt_R P))$

proof –

have *Skip* ;; $P =$

$\mathbf{R}_s ((\$tr' =_u \$tr \wedge \$st' =_u \$st) \wp_r pre_R P \vdash$
 $(\$tr' =_u \$tr \wedge \$st' =_u \$st) ;; peri_R P \diamond$
 $(\$tr' =_u \$tr \wedge \$st' =_u \$st) ;; post_R P)$

by (simp add: SRD-composition-wp alpha rdes closure wp assms rpred C1, rel-auto)

also have ... = $\mathbf{R}_s ((\forall \$ref \cdot pre_R P) \vdash$

$(\$tr' =_u \$tr \wedge \neg \$wait' \wedge \$st' =_u \$st) ;; ((\exists \$st \cdot [II]_D) \triangleleft \$wait \triangleright cmt_R P))$

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

also have ... = $\mathbf{R}_s ((\forall \$ref \cdot pre_R P) \vdash (\exists \$ref \cdot cmt_R P))$

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

finally show ?thesis .

qed

lemma *Skip-left-unit-ref-unrest:*

assumes *P is CSP* $\$ref \# P \llbracket false/\$wait \rrbracket$

shows *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:*

$\llbracket P \text{ is CSP}; \$ref \# P \llbracket false/\$wait \rrbracket \rrbracket \implies P \text{ is CSP3}$

by (simp add: CSP3-def Healthy-def' Skip-left-unit-ref-unrest)

lemma *ref-unrest-RHS-design:*

assumes $\$ref \# P \ \$ref \# Q_1 \ \$ref \# Q_2$

shows $\$ref \# (\mathbf{R}_s(P \vdash Q_1 \diamond Q_2)) \ f$

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* $\$ref \# pre_R(P) \ \$ref \# peri_R(P) \ \$ref \# post_R(P)$

shows *P is CSP3*

proof –

have $P: \mathbf{R}_s(pre_R(P) \vdash peri_R(P) \diamond post_R(P)) = P$

by (simp add: SRD-reactive-design-alt assms(1) wait'-cond-peri-post-cmt[THEN sym])

have $\mathbf{R}_s(pre_R(P) \vdash peri_R(P) \diamond post_R(P))$ 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]:* $\$ref \# Skip \llbracket false/\$wait \rrbracket$

by (simp add: Skip-def RHS-def R1-def R2c-def R2s-def R3h-def design-def usubst unrest)

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

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 $\text{pre}_R(P)$ is RC

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

shows P is CSP4

proof –

have $(\neg_r \text{pre}_R(P)) \;; R1(\text{true}) = (\neg_r \text{pre}_R(P))$

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 $CSP_4(P) = \mathbf{R}_s((\neg_r \text{pre}_R P) \text{ wp}_r \text{ false} \vdash ((\exists \$st' \cdot \text{peri}_R P) \diamond (\exists \$ref' \cdot \text{post}_R P)))$
by (*simp* *add*: *CSP4-def* *Skip-right-tri-lemma* *assms*)

lemma *Skip-srdes-right-unit*:

(Skip :: ('σ, 'φ) action) ;; II_R = Skip
by (*rdes-simp*)

lemma *Skip-srdes-left-unit*:

II_R ;; (Skip :: ('σ, 'φ) 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* is *CSP4* \implies *P* is *RD3*

by (*metis* *CSP4-right-subsumes-RD3* *Healthy-def*)

lemma *CSP4-tri-intro*:

assumes *P* is *CSP* $(\neg_r \text{pre}_R(P)) ;; R1(\text{true}) = (\neg_r \text{pre}_R(P)) \$st' \# \text{peri}_R(P) \$ref' \# \text{post}_R(P)$
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* $\$ref' \# \text{post}_R(P)$
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*: $CSP3(CSP_4(P)) = CSP_4(CSP3(P))$

by (*simp* *add*: *CSP3-def* *CSP4-def* *seqr-assoc*)

lemma *NCSP-implies-CSP* [*closure*]: *P* is *NCSP* \implies *P* 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*]:

$\llbracket X \text{ is } NCSP; P(\mathbf{R}_s(\text{pre}_R(X) \vdash \text{peri}_R(X) \diamond \text{post}_R(X))) \rrbracket \implies P(X)$
by (*simp* *add*: *SRD-reactive-tri-design* *closure*)

lemma *NCSP-implies-CSP3* [*closure*]:

P is *NCSP* \implies *P* is *CSP3*
by (*metis* (*no-types*, *lifting*) *CSP3-def* *Healthy-def'* *NCSP-def* *Skip-is-CSP* *Skip-left-unit-ref-unrest*)

Skip-unrest-ref comp-apply seqr-assoc)

lemma *NCSP-implies-CSP4* [closure]:

P is NCSP \implies P 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 is NCSP \implies P is RD3*

by (*metis* *CSP3-commutes-CSP4 CSP4-right-subsumes-RD3 Healthy-def NCSP-def comp-apply*)

lemma *NCSP-implies-NSRD* [closure]: *P is NCSP \implies P 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 is CSP P is CSP3 P is CSP4*

shows *P is NCSP*

by (*metis* *Healthy-def NCSP-def assms comp-eq-dest-lhs*)

lemma *Skip-left-unit*: *P is NCSP \implies Skip ;; P = P*

by (*metis* (*full-types*) *CSP3-def Healthy-if NCSP-implies-CSP3*)

lemma *Skip-right-unit*: *P is NCSP \implies P ;; Skip = P*

by (*metis* (*full-types*) *CSP4-def Healthy-if NCSP-implies-CSP4*)

lemma *NCSP-NSRD-intro*:

assumes *P is NSRD \$ref \# pre_R(P) \$ref \# peri_R(P) \$ref \# post_R(P) \$ref' \# post_R(P)*

shows *P is NCSP*

by (*simp add: CSP3-SRD-intro CSP4-NSRD-intro NCSP-intro NSRD-is-SRD assms*)

lemma *CSP4-neg-pre-unit*:

assumes *P is CSP P is CSP4*

shows *($\neg_r pre_R(P)$) ;; R1(true) = ($\neg_r pre_R(P)$)*

by (*simp add: CSP4-implies-RD3 NSRD-neg-pre-unit SRD-RD3-implies-NSRD assms(1) assms(2)*)

lemma *NSRD-CSP4-intro*:

assumes P is CSP P 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 $A \subseteq \llbracket \text{NCSP} \rrbracket_H$
 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 $A \subseteq \llbracket \text{CSP} \rrbracket_H$ $A \subseteq \llbracket \text{CSP}_4 \rrbracket_H$
 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 ;; 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]: *pre_R(Stop) = true_r*
by (simp add: *Stop-def Stop-is-CSP rea-pre-RHS-design unrest usubst R2c-true*)

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

lemma *postR-Stop* [rdes]: *post_R(Stop) = false*
by (rel-auto)

lemma *cmtR-Stop* [rdes]: *cmt_R(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 pre_R(P) is CRR*
by (rule *CRR-intro, simp-all add: closure unrest*)

lemma *periR-CRR* [closure]: *P is NCSP \implies peri_R(P) is CRR*
by (rule *CRR-intro, simp-all add: closure unrest*)

lemma *postR-CRR* [closure]: *P is NCSP \implies post_R(P) is CRR*

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

lemma NCSP-rdes-intro [closure]:

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]:

$A \subseteq \llbracket \text{CSP3} \rrbracket_H \implies (\bigcap A)$ 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]:

$A \subseteq \llbracket \text{CSP4} \rrbracket_H \implies (\bigcap A)$ 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]: $\llbracket A \subseteq \llbracket \text{NCSP} \rrbracket_H; A \neq \{\} \rrbracket \implies (\bigcap A)$ 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]: $\llbracket \bigwedge i. P(i) \text{ is NCSP}; A \neq \{\} \rrbracket \implies (\bigcap i \in A. P(i))$ 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 $\dots = \mathbf{R}_s((\forall \$ref \cdot (\neg_r \text{pre}_R(\text{NCSP } P)) \text{ wp}_r \text{ false}) \vdash$

$(\exists \$ref \cdot \exists \$st' \cdot \text{peri}_R(\text{NCSP } P)) \diamond$

$((\exists \$ref \cdot \exists \$ref' \cdot post_R (NCSP P)) \wedge \$tr <_u \$tr')$
 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 ($\mathbf{R}_s ((pre_R X) \vdash peri_R X \diamond (R4(post_R X)))$)
 shows $P X$
 by (metis *R4-def 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** $1:(post_R P \wedge \$tr' =_u \$tr) ;; true_r = true_r$
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

tcsp-hcond == *utp-hcond* :: $(TCSP, ('\sigma, '\varphi) \text{ st-csp}) \text{ uthy} \Rightarrow (('\sigma, '\varphi) \text{ st-csp} \times (''\sigma, ''\varphi) \text{ st-csp}) \text{ health}$
tcsp-unit == *utp-unit* :: $(TCSP, ('\sigma, '\varphi) \text{ st-csp}) \text{ uthy} \Rightarrow (''\sigma, ''\varphi) \text{ action}$

begin

definition *tcsp-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*]: *tcsp-hcond* $T = NCSP$

definition *tcsp-unit* :: $(TCSP, ('\sigma, '\varphi) \text{ st-csp}) \text{ uthy} \Rightarrow (''\sigma, ''\varphi) \text{ action}$ **where**
[*upred-defs*]: *tcsp-unit* $T = Skip$

end

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

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

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

and $\mathcal{IL}_{TCSP} = Skip$

and $\top_{TCSP} = Miracle$

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

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

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

proof –

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

by (*unfold-locales*, *simp-all* *add*: *tcsp-hcond-def* *closure* *Healthy-if*)

show $1: \top_{TCSP} = (Miracle :: ('\sigma, '\varphi) \text{ action})$

by (*metis* *NCSP-Miracle* *NCSP-implies-CSP* *lat.top-healthy* *lat.utp-theory-continuous-axioms* *srdes-theory-continuous*.
tcsp-hcond-def *upred-semiring.add-commute* *utp-theory-continuous.meet-top*)

thus *utp-theory-kleene* $UTHY(TCSP, ('\sigma, '\varphi) \text{ st-csp})$

by (*unfold-locales*, *simp-all* *add*: *tcsp-hcond-def* *tcsp-unit-def* *Skip-left-unit* *Skip-right-unit* *closure* *Healthy-if* *Miracle-left-zero*)

qed (*simp-all* *add*: *tcsp-hcond-def* *tcsp-unit-def* *closure* *Healthy-if*)

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

abbreviation *TestC* (*test_C*) **where**
test_C *P* \equiv *utest TCSP P*

abbreviation *StarC* :: (*'σ*, *'φ*) *action* \Rightarrow (*'σ*, *'φ*) *action* (*-^{*C}* [999] 999) **where**
StarC P \equiv *P★_{TCSP}*

lemma *csp-bottom-Chaos*: $\perp_{TCSP} = \text{Chaos}$
using *NCSP-Chaos NCSP-implies-CSP* **by** *auto*

lemma *csp-top-Miracle*: $\top_{TCSP} = \text{Miracle}$
by (*simp add: csp-theory.healthy-top csp-theory.utp-theory-mono-axioms utp-theory-mono.healthy-top*)

5.3 Algebraic laws

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

end

6 Stateful-Failure Reactive Contracts

theory *utp-sfrd-contracts*
imports *utp-sfrd-healths*
begin

definition *mk-CRD* :: *'s upred* \Rightarrow (*'e list* \Rightarrow *'e set* \Rightarrow *'s upred*) \Rightarrow (*'e list* \Rightarrow *'s hrel*) \Rightarrow (*'s*, *'e*) *action*
where
 $[rdes-def]: mk-CRD\ P\ Q\ R = \mathbf{R}_s([P]_{S<} \vdash [Q\ x\ r]_{S<} \llbracket x \rightarrow \&tt \rrbracket [r \rightarrow \$ref\ ']] \diamond [R(x)]_S' \llbracket x \rightarrow \&tt \rrbracket$

syntax
-ref-var :: *logic*
-mk-CRD :: *logic* \Rightarrow *logic* \Rightarrow *logic* \Rightarrow *logic* (*[-/* \vdash *-/* $|$ *-]*_{*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
 $[P \vdash Q \mid R]_C \Rightarrow CONST\ mk-CRD\ P\ (\lambda\ -trace-var\ -ref-var.\ Q)\ (\lambda\ -trace-var.\ R)$
 $[P \vdash Q \mid R]_C \Leftarrow CONST\ mk-CRD\ P\ (\lambda\ x\ r.\ Q)\ (\lambda\ y.\ R)$

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

lemma *preR-mk-CRD [rdes]*: $pre_R([P \vdash Q\ trace\ refs \mid R(trace)]_C) = [P]_{S<}$

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-mk-CRD* [rdes]: $\text{peri}_R([P \vdash Q \text{ trace refs} \mid R(\text{trace})]_C) = ([P]_{S<} \Rightarrow_r ([Q \text{ trace refs}]_{S<}))[(\text{trace}, \text{refs}) \rightarrow (\&tt, \$r)]$
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]: $\text{post}_R([P \vdash Q \text{ trace refs} \mid R(\text{trace})]_C) = ([P]_{S<} \Rightarrow_r ([R(\text{trace})]_S))[\text{trace} \rightarrow \&tt]$
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
 $Q \text{ is CSP } '[P_1]_{S<} \Rightarrow \text{pre}_R Q'$
 $'[P_1]_{S<} \wedge \text{peri}_R Q \Rightarrow [P_2 \ t \ r]_{S<}[\![t \rightarrow \&tt]\!][r \rightarrow \$ref']'$
 $'[P_1]_{S<} \wedge \text{post}_R Q \Rightarrow [P_3 \ x]_S[\![x \rightarrow \&tt]\!]'$
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
 $Q \text{ is CSP } '[P_1]_{S<} \Rightarrow \text{pre}_R Q'$
 $[P_2 \ t \ r]_{S<}[\![t \rightarrow \&tt]\!][r \rightarrow \$ref'] \sqsubseteq ([P_1]_{S<} \wedge \text{peri}_R Q)$
 $[P_3 \ x]_S[\![x \rightarrow \&tt]\!] \sqsubseteq ([P_1]_{S<} \wedge \text{post}_R Q)$
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
 $'[P_1]_{S<} \Rightarrow ([Q_1]_{S<} :: ('e, 's) \text{ action})'$
 $([P_2 \ x \ r]_{S<}[\![x \rightarrow \&tt]\!][r \rightarrow \$ref']) \sqsubseteq ([P_1]_{S<} \wedge [Q_2 \ x \ r]_{S<}[\![x \rightarrow \&tt]\!][r \rightarrow \$ref']) :: ('e, 's) \text{ action}$
 $[P_3 \ x]_S[\![x \rightarrow \&tt]\!] \sqsubseteq ([P_1]_{S<} \wedge [Q_3 \ x]_S[\![x \rightarrow \&tt]\!] :: ('e, 's) \text{ action})$
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
 $'[P_1]_{S<} \Rightarrow Q_1'$
 $([P_2 \ x \ r]_{S<}[\![x \rightarrow \&tt]\!][r \rightarrow \$ref']) \sqsubseteq ([P_1]_{S<} \wedge Q_2)$
 $[P_3 \ x]_S[\![x \rightarrow \&tt]\!] \sqsubseteq ([P_1]_{S<} \wedge Q_3)$
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+*)

lemma *CRD-refine-rdes'*:

assumes
 $Q_2 \text{ is RR}$

Q_3 is RR
 $\text{'}[P_1]_{S<} \Rightarrow Q_1 \text{'}$
 $\bigwedge t. ([P_2 \ t \ r]_{S<} \llbracket r \rightarrow \$ref' \rrbracket) \sqsubseteq ([P_1]_{S<} \wedge Q_2 \llbracket \langle \rangle, \ll t \gg / \$tr, \$tr' \rrbracket)$
 $\bigwedge t. [P_3 \ t]_{S'} \sqsubseteq ([P_1]_{S<} \wedge Q_3 \llbracket \langle \rangle, \ll t \gg / \$tr, \$tr' \rrbracket)$
shows $([P_1 \vdash P_2 \text{ trace refs} \mid P_3 \text{ trace}]_C :: ('e, 's) \text{ action}) \sqsubseteq$
 $\mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3)$
proof (*simp add: mk-CRD-def, rule srdes-tri-refine-intro*)
show $\text{'}[P_1]_{S<} \Rightarrow Q_1 \text{'}$ **by** (*fact assms(3)*)

have $\bigwedge t. ([P_2 \ t \ r]_{S<} \llbracket r \rightarrow \$ref' \rrbracket) \sqsubseteq ([P_1]_{S<} \wedge (RR \ Q_2) \llbracket \langle \rangle, \ll t \gg / \$tr, \$tr' \rrbracket)$
by (*simp add: assms Healthy-if*)
hence $\text{'}[P_1]_{S<} \wedge RR(Q_2) \Rightarrow [P_2 \ x \ r]_{S<} \llbracket x \rightarrow \&tt \rrbracket \llbracket r \rightarrow \$ref' \rrbracket \text{'}$
by (*rel-simp; meson*)
thus $\text{'}[P_1]_{S<} \wedge Q_2 \Rightarrow [P_2 \ x \ r]_{S<} \llbracket x \rightarrow \&tt \rrbracket \llbracket r \rightarrow \$ref' \rrbracket \text{'}$
by (*simp add: Healthy-if assms*)

have $\bigwedge t. [P_3 \ t]_{S'} \sqsubseteq ([P_1]_{S<} \wedge (RR \ Q_3) \llbracket \langle \rangle, \ll t \gg / \$tr, \$tr' \rrbracket)$
by (*simp add: assms Healthy-if*)
hence $\text{'}[P_1]_{S<} \wedge (RR \ Q_3) \Rightarrow [P_3 \ x]_{S'} \llbracket x \rightarrow \&tt \rrbracket \text{'}$
by (*rel-simp; meson*)
thus $\text{'}[P_1]_{S<} \wedge Q_3 \Rightarrow [P_3 \ x]_{S'} \llbracket x \rightarrow \&tt \rrbracket \text{'}$
by (*simp add: Healthy-if assms*)
qed
end

7 External Choice

theory *utp-sfrd-extchoice*
imports
utp-sfrd-healths
utp-sfrd-rel
begin

7.1 Definitions and syntax

definition *ExtChoice* ::

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

syntax

$-ExtChoice :: ptnr \Rightarrow 'a \text{ set} \Rightarrow 'b \Rightarrow 'b \ ((\exists \square \ - \in \ - \cdot / \ -) [0, 0, 10] \ 10)$
 $-ExtChoice-simp :: ptnr \Rightarrow 'b \Rightarrow 'b \ ((\exists \square \ - \cdot / \ -) [0, 10] \ 10)$

translations

$\square P \in A \cdot B \quad \Rightarrow \text{CONST } ExtChoice \ ((\lambda P. B) \text{ ' } A)$
 $\square P \cdot B \quad \Rightarrow \text{CONST } ExtChoice \ (\text{CONST range } (\lambda P. B))$

definition *extChoice* ::

$(\sigma, \varphi) \text{ action} \Rightarrow (\sigma, \varphi) \text{ action} \Rightarrow (\sigma, \varphi) \text{ action}$ (**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*:

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

7.2 Basic laws

7.3 Algebraic laws

lemma *ExtChoice-empty*: $\text{ExtChoice } \{\} = \text{Stop}$
 by (simp add: ExtChoice-def cond-def Stop-def)

lemma *ExtChoice-single*:
 $P \text{ is CSP} \implies \text{ExtChoice } \{P\} = P$
 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 $(\sqcap i \in A \cdot \mathbf{R}_s(P(i) \vdash Q(i))) = \mathbf{R}_s((\sqcap i \in A \cdot P(i)) \vdash ((\sqcap i \in A \cdot Q(i)) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\sqcap i \in A \cdot Q(i))))$
proof –
 have $(\sqcap i \in A \cdot \mathbf{R}_s(P(i) \vdash Q(i))) =$
 $\mathbf{R}_s((\sqcap i \in A \cdot \text{pre}_R(\mathbf{R}_s(P(i) \vdash Q(i)))) \vdash$
 $((\sqcap i \in A \cdot \text{cmt}_R(\mathbf{R}_s(P(i) \vdash Q(i))))$
 $\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
 $(\sqcap i \in A \cdot \text{cmt}_R(\mathbf{R}_s(P(i) \vdash Q(i))))))$
 by (simp add: ExtChoice-def)
 also have ... =
 $\mathbf{R}_s((\sqcap i \in A \cdot R1(R2c(\text{pre}_s \dagger P(i)))) \vdash$
 $((\sqcap i \in A \cdot R1(R2c(\text{cmt}_s \dagger (P(i) \Rightarrow Q(i))))$
 $\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
 $(\sqcap i \in A \cdot R1(R2c(\text{cmt}_s \dagger (P(i) \Rightarrow Q(i)))))))$
 by (simp add: rea-pre-RHS-design rea-cmt-RHS-design)
 also have ... =
 $\mathbf{R}_s((\sqcap i \in A \cdot R1(R2c(\text{pre}_s \dagger P(i)))) \vdash$
 $R1(R2c$
 $((\sqcap i \in A \cdot R1(R2c(\text{cmt}_s \dagger (P(i) \Rightarrow Q(i))))$
 $\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
 $(\sqcap i \in A \cdot R1(R2c(\text{cmt}_s \dagger (P(i) \Rightarrow Q(i)))))))$
 by (metis (no-types, lifting) RHS-design-export-R1 RHS-design-export-R2c)
 also have ... =
 $\mathbf{R}_s((\sqcap i \in A \cdot R1(R2c(\text{pre}_s \dagger P(i)))) \vdash$
 $R1(R2c$
 $((\sqcap i \in A \cdot (\text{cmt}_s \dagger (P(i) \Rightarrow Q(i))))$
 $\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
 $(\sqcap i \in A \cdot (\text{cmt}_s \dagger (P(i) \Rightarrow Q(i))))))$
 by (simp add: R2c-UNIF R2c-cond R1-cond R1-idem R1-R2c-commute R2c-idem R1-UNIF assms R1-USUP R2c-USUP)
 also have ... =
 $\mathbf{R}_s((\sqcap i \in A \cdot R1(R2c(\text{pre}_s \dagger P(i)))) \vdash$
 $\text{cmt}_s \dagger$
 $((\sqcap i \in A \cdot (\text{cmt}_s \dagger (P(i) \Rightarrow Q(i))))$
 $\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
 $(\sqcap i \in A \cdot (\text{cmt}_s \dagger (P(i) \Rightarrow Q(i))))))$
 by (metis (no-types, lifting) RHS-design-export-R1 RHS-design-export-R2c rdes-export-cmt)
 also have ... =

$\mathbf{R}_s ((\sqcup i \in A \cdot R1 (R2c (pre_s \dagger P(i)))) \vdash$
 $\quad cmt_s \dagger$
 $\quad ((\sqcup i \in A \cdot (P(i) \Rightarrow Q(i)))$
 $\quad \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
 $\quad (\sqcap i \in A \cdot (P(i) \Rightarrow Q(i))))$
 by (simp add: usubst)
 also have ... =
 $\mathbf{R}_s ((\sqcup i \in A \cdot R1 (R2c (pre_s \dagger P(i)))) \vdash$
 $\quad ((\sqcup i \in A \cdot (P(i) \Rightarrow Q(i))) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\sqcap i \in A \cdot (P(i) \Rightarrow Q(i))))$
 by (simp add: rdes-export-cmt)
 also have ... =
 $\mathbf{R}_s ((R1(R2c(\sqcup i \in A \cdot (pre_s \dagger P(i)))) \vdash$
 $\quad ((\sqcup i \in A \cdot (P(i) \Rightarrow Q(i))) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\sqcap i \in A \cdot (P(i) \Rightarrow Q(i))))$
 by (simp add: not-UINF R1-UINF R2c-UINF assms)
 also have ... =
 $\mathbf{R}_s ((R2c(\sqcup i \in A \cdot (pre_s \dagger P(i)))) \vdash$
 $\quad ((\sqcup i \in A \cdot (P(i) \Rightarrow Q(i))) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\sqcap i \in A \cdot (P(i) \Rightarrow Q(i))))$
 by (simp add: R1-design-R1-pre)
 also have ... =
 $\mathbf{R}_s (((\sqcup i \in A \cdot (pre_s \dagger P(i)))) \vdash$
 $\quad ((\sqcup i \in A \cdot (P(i) \Rightarrow Q(i))) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\sqcap i \in A \cdot (P(i) \Rightarrow Q(i))))$
 by (metis (no-types, lifting) RHS-design-R2c-pre)
 also have ... =
 $\mathbf{R}_s ([\$ok \mapsto_s true, \$wait \mapsto_s false] \dagger (\sqcup i \in A \cdot P(i))) \vdash$
 $\quad ((\sqcup i \in A \cdot (P(i) \Rightarrow Q(i))) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\sqcap i \in A \cdot (P(i) \Rightarrow Q(i))))$
 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 ... =
 $\mathbf{R}_s ((\sqcup i \in A \cdot P(i)) \vdash ((\sqcup i \in A \cdot (P(i) \Rightarrow Q(i))) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\sqcap i \in A \cdot (P(i) \Rightarrow Q(i))))$
 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*:

assumes $\bigwedge i. \$ok' \nmid P_1(i) \ A \neq \{\}$
 shows $(\sqcap i \in A \cdot \mathbf{R}_s(P_1(i) \vdash P_2(i) \diamond P_3(i))) =$
 $\mathbf{R}_s ((\sqcap i \in A \cdot P_1(i)) \vdash (((\sqcap i \in A \cdot P_2(i)) \triangleleft \$tr' =_u \$tr \triangleright (\sqcap i \in A \cdot P_2(i))) \diamond (\sqcap i \in A \cdot P_3(i))))$
 proof –
 have $(\sqcap i \in A \cdot \mathbf{R}_s(P_1(i) \vdash P_2(i) \diamond P_3(i))) =$
 $\mathbf{R}_s ((\sqcap i \in A \cdot P_1(i)) \vdash ((\sqcap i \in A \cdot P_2(i) \diamond P_3(i)) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\sqcap i \in A \cdot P_2(i) \diamond P_3(i))))$
 by (simp add: ExtChoice-rdes assms)
 also
 have ... =
 $\mathbf{R}_s ((\sqcap i \in A \cdot P_1(i)) \vdash ((\sqcap i \in A \cdot P_2(i) \diamond P_3(i)) \triangleleft \$wait' \wedge \$tr' =_u \$tr \triangleright (\sqcap i \in A \cdot P_2(i) \diamond P_3(i))))$

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' [rdes-def]:
 assumes $\bigwedge i . \$ok' \# P_1(i) \ A \neq \{\}$
 shows $(\sqcap i \in A \cdot \mathbf{R}_s(P_1(i) \vdash P_2(i) \diamond P_3(i))) =$
 $\mathbf{R}_s ((\sqcup i \in A \cdot P_1(i)) \vdash (((\sqcup i \in A \cdot R5(P_2(i))) \vee (\sqcap i \in A \cdot R4(P_2(i)))) \diamond (\sqcap i \in A \cdot P_3(i))))$
 by (simp add: ExtChoice-tri-rdes assms, rel-auto, simp-all add: less-le assms)

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 ... = $(\sqcap i :: nat \in \{0, 1\} \cdot \mathbf{R}_s((P_1 \triangleleft \ll i = 0 \gg \triangleright Q_1) \vdash (P_2 \triangleleft \ll i = 0 \gg \triangleright Q_2)))$
 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 $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) \sqcap \mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3) =$
 $\mathbf{R}_s((P_1 \wedge Q_1) \vdash (((P_2 \wedge Q_2) \triangleleft \$tr' =_u \$tr \triangleright (P_2 \vee Q_2)) \diamond (P_3 \vee Q_3)))$

proof –

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

qed

lemma *extChoice-rdes-def*:

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

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

assumes P_1 is *RR* Q_1 is *RR*
shows $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) \sqcap \mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3) =$
 $\mathbf{R}_s((P_1 \wedge Q_1) \vdash ((R5(P_2 \wedge Q_2) \vee R4(P_2 \vee Q_2)) \diamond (P_3 \vee Q_3)))$
by (*simp add: extChoice-rdes-def assms, rel-auto, simp-all add: less-le*)

lemma *CSP-ExtChoice [closure]*:

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

lemma *CSP-extChoice [closure]*:

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

lemma *preR-ExtChoice [rdes]*:

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

proof –

have $pre_R(ExtChoice A) = (R1 (R2c ((\bigsqcup P \in A \cdot pre_R P))))$
by (*simp add: ExtChoice-def rea-pre-RHS-design usubst unrest*)
also from *assms* **have** $\dots = (R1 (R2c (\bigsqcup P \in A \cdot (pre_R(CSP(P)))))$
by (*metis USUP-healthy*)
also from *assms* **have** $\dots = (\bigsqcup P \in A \cdot (pre_R(CSP(P))))$
by (*rel-auto*)
also from *assms* **have** $\dots = (\bigsqcup P \in A \cdot (pre_R(P)))$
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 $peri_R(ExtChoice\ A) = ((\bigsqcup P \in A \cdot pre_R(P)) \Rightarrow_r (\bigsqcup P \in A \cdot peri_R\ P)) \triangleleft \$tr' =_u \$tr \triangleright (\bigsqcup P \in A \cdot peri_R\ P)$

proof –

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

by (*simp add: ExtChoice-tri-rdes-def assms closure*)

also have $\dots = peri_R(\mathbf{R}_s((\bigsqcup P \in A \cdot pre_R(NCSP\ P)) \vdash ((\bigsqcup P \in A \cdot peri_R(NCSP\ P)) \triangleleft \$tr' =_u \$tr \triangleright (\bigsqcup P \in A \cdot peri_R(NCSP\ P)) \diamond (\bigsqcup P \in A \cdot post_R\ P))))$

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

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

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

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

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

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

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

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

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

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

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

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

finally show *?thesis* .

qed

lemma *periR-ExtChoice'*:

assumes $A \subseteq \llbracket NCSP \rrbracket_H$ $A \neq \{\}$

shows $\text{peri}_R(\text{ExtChoice } A) = (R5((\bigsqcup P \in A \cdot \text{pre}_R(P)) \Rightarrow_r (\bigsqcup P \in A \cdot \text{peri}_R P)) \vee (\prod P \in A \cdot R4(\text{peri}_R P)))$
using $\text{assms}(2)$
by ($\text{simp add: periR-ExtChoice assms}(1), \text{rel-auto}$)

lemma $\text{periR-ExtChoice-ind}$ [rdes]:

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

lemma $\text{periR-ExtChoice-ind'}$:

assumes $\bigwedge P. P \in A \implies F(P)$ is NCSP $A \neq \{\}$
shows $\text{peri}_R(\bigsqcup P \in A \cdot F(P)) = (R5((\bigsqcup P \in A \cdot \text{pre}_R(F P)) \Rightarrow_r (\bigsqcup P \in A \cdot \text{peri}_R (F P))) \vee (\prod P \in A \cdot R4(\text{peri}_R (F P))))$
using assms **by** ($\text{subst periR-ExtChoice'}, \text{auto simp add: closure unrest}$)

lemma postR-ExtChoice [rdes]:

assumes $A \subseteq \llbracket \text{NCSP} \rrbracket_H A \neq \{\}$
shows $\text{post}_R(\text{ExtChoice } A) = (\prod P \in A \cdot \text{post}_R P)$

proof –

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

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

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

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 ($\text{rule USUP-cong}, \text{simp add: usubst closure unrest assms}$)
thus $?thesis$
by ($\text{simp add: rea-post-RHS-design Healthy-Idempotent SRD-Idempotent usubst unrest assms}$)

qed

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

by ($\text{simp add: R2c-rea-impl R2c-condr R2c-UINF R2c-preR R2c-postR}$
 $R2c-tr'-minus-tr R2c-USUP \text{ closure}$)

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

by ($\text{simp add: R1-rea-impl R1-cond R1-USUP R1-UINF assms Healthy-if closure}$)

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

by ($\text{simp add: UINF-rea-impl}$)

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

by ($\text{simp add: SRD-post-under-pre closure assms unrest}$)

finally show $?thesis$

by ($\text{simp add: UINF-healthy[OF assms}(1), \text{THEN sym}] \text{USUP-healthy[OF assms}(1), \text{THEN sym}]$)

qed

lemma $\text{postR-ExtChoice-ind}$ [rdes]:

assumes $\bigwedge P. P \in A \implies F(P)$ is NCSP $A \neq \{\}$
shows $\text{post}_R(\bigsqcup P \in A \cdot F(P)) = (\prod P \in A \cdot \text{post}_R(F(P)))$

using *assms* **by** (*subst postR-ExtChoice*, *auto simp add: closure unrest*)

lemma *preR-extChoice*:
assumes *P is CSP Q is CSP \$wait' # pre_R(P) \$wait' # 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 $1 : (\neg_r (\neg_r pre_R (ExtChoice A)) ;;_h R1 \text{ true}) = pre_R (ExtChoice A)$
proof –
have $\bigwedge P. P \in A \implies (\neg_r pre_R(P)) ;; R1 \text{ true} = (\neg_r pre_R(P))$
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' \# peri_R (ExtChoice A)
proof -
  have a:  $\bigwedge P. P \in A \implies \$st' \# pre_R(P)$ 
    by (simp add: NCSP-Healthy-subset-member NCSP-implies-NSRD NSRD-st'-unrest-pre assms)
  have b:  $\bigwedge P. P \in A \implies \$st' \# peri_R(P)$ 
    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' \# post_R (ExtChoice A)
proof -
  have a:  $\bigwedge P. P \in A \implies \$ref' \# pre_R(P)$ 
    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:  $\bigwedge P. P \in A \implies \$ref' \# post_R(P)$ 
    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 is NCSP Q is NCSP
  shows P  $\square$  Q is CSP4
  by (simp add: extChoice-def, rule CSP4-ExtChoice, simp-all add: assms)

lemma NCSP-ExtChoice [closure]:
  assumes A  $\subseteq \llbracket NCSP \rrbracket_H$ 
  shows ExtChoice A 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  $\subseteq \llbracket CSP \rrbracket_H$  A  $\subseteq \llbracket CSP3 \rrbracket_H$  A  $\subseteq \llbracket CSP4 \rrbracket_H$ 
    using NCSP-implies-CSP NCSP-implies-CSP3 NCSP-implies-CSP4 by blast+
    have wu:  $\bigwedge P. P \in A \implies \$wait' \# pre_R(P)$ 
    using NCSP-implies-NSRD NSRD-wait'-unrest-pre assms by force
    show 1:ExtChoice A is CSP

```

by (metis (mono-tags) Ball-Collect CSP-ExtChoice NCSP-implies-CSP assms)
 from cls show ExtChoice A 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 ExtChoice-NCSP-closed [closure]:
 assumes $\bigwedge i. i \in I \implies P(i)$ is NCSP
 shows $(\bigwedge i \in I. P(i))$ is NCSP
 by (simp add: NCSP-ExtChoice assms image-subset-iff)

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

7.5 Productivity and Guardedness

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

proof –

have 1: $\bigwedge P. P \in A \implies \$wait' \# pre_R(P)$
 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 $((\bigwedge P \in A. pre_R P) \wedge (\bigwedge P \in A. post_R P)) =$
 $((\bigwedge P \in A. pre_R P) \wedge (\bigwedge P \in A. (pre_R P \wedge post_R P)))$
 by (rel-auto)

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

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 $(\$tr' >_u \$tr) \sqsubseteq ((\bigwedge P \in A. pre_R P) \wedge (\bigwedge P \in A. post_R P))$
 by (rel-auto)

qed
 qed

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

lemma ExtChoice-Guarded [closure]:
 assumes $\bigwedge P. P \in A \implies \text{Guarded } P$
 shows Guarded $(\lambda X. \bigwedge P \in A. P(X))$

proof (rule GuardedI)

fix X n

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

proof –

fix Y

let ?lhs = $((\bigwedge P \in A. P Y) \wedge gvirt(n+1))$ and ?rhs = $((\bigwedge P \in A. (P Y \wedge gvirt(n+1))) \wedge gvirt(n+1))$

```

have a: ?lhs[false/$ok] = ?rhs[false/$ok]
  by (rel-auto)
have b: ?lhs[true/$ok][true/$wait] = ?rhs[true/$ok][true/$wait]
  by (rel-auto)
have c: ?lhs[true/$ok][false/$wait] = ?rhs[true/$ok][false/$wait]
  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 is CSP  $\implies P \Box P = P$ 
  by (unfold extChoice-def, simp add: ExtChoice-single)

```

```

lemma extChoice-assoc:
  assumes P is CSP Q is CSP R 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 ... =
     $\mathbf{R}_s(((\text{pre}_R P \wedge \text{pre}_R Q) \wedge \text{pre}_R R) \vdash$ 
       $((\text{cmt}_R P \wedge \text{cmt}_R Q) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\text{cmt}_R P \vee \text{cmt}_R Q) \wedge \text{cmt}_R R)$ 
       $\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$ 

```

$((cmt_R P \wedge cmt_R Q) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (cmt_R P \vee cmt_R Q) \vee cmt_R R))$
 by (simp add: extChoice-rdes unrest)
 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 (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 $S \subseteq \llbracket \text{CSP} \rrbracket_H S \neq \{\}$
shows $P \sqcap (\sqcap S) = (\sqcap_{Q \in S} P \sqcap Q)$

proof –

let ?S1 = $\text{pre}_R \text{ ` } S$ and ?S2 = $\text{cmt}_R \text{ ` } S$
have $P \sqcap (\sqcap S) = P \sqcap (\sqcap_{Q \in S} \mathbf{R}_s(\text{pre}_R(Q) \vdash \text{cmt}_R(Q)))$
by (simp add: SRD-as-reactive-design[THEN sym] Healthy-SUPREMUM UINF-as-Sup-collect assms)
also have ... = $\mathbf{R}_s(\text{pre}_R(P) \vdash \text{cmt}_R(P)) \sqcap \mathbf{R}_s((\sqcap_{Q \in S} \text{pre}_R(Q)) \vdash (\sqcap_{Q \in S} \text{cmt}_R(Q)))$
by (simp add: RHS-design-USUP SRD-reactive-design-alt assms)
also have ... = $\mathbf{R}_s((\text{pre}_R(P) \wedge (\sqcap_{Q \in S} \text{pre}_R(Q))) \vdash$
 $((\text{cmt}_R(P) \wedge (\sqcap_{Q \in S} \text{cmt}_R(Q)))$
 $\triangleleft \text{\$tr}' =_u \text{\$tr} \wedge \text{\$wait}' \triangleright$
 $(\text{cmt}_R(P) \vee (\sqcap_{Q \in S} \text{cmt}_R(Q))))$
by (simp add: extChoice-rdes unrest)
also have ... = $\mathbf{R}_s((\sqcap_{Q \in S} \text{pre}_R P \wedge \text{pre}_R Q) \vdash$
 $(\sqcap_{Q \in S} (\text{cmt}_R P \wedge \text{cmt}_R Q) \triangleleft \text{\$tr}' =_u \text{\$tr} \wedge \text{\$wait}' \triangleright (\text{cmt}_R P \vee \text{cmt}_R Q)))$
by (simp add: conj-USUP-dist conj-UINF-dist disj-UINF-dist cond-UINF-dist assms)
also have ... = $(\sqcap_{Q \in S} \mathbf{R}_s((\text{pre}_R P \wedge \text{pre}_R Q) \vdash$
 $((\text{cmt}_R P \wedge \text{cmt}_R Q) \triangleleft \text{\$tr}' =_u \text{\$tr} \wedge \text{\$wait}' \triangleright (\text{cmt}_R P \vee \text{cmt}_R Q))))$
by (simp add: assms RHS-design-USUP)
also have ... = $(\sqcap_{Q \in S} \mathbf{R}_s(\text{pre}_R(P) \vdash \text{cmt}_R(P)) \sqcap \mathbf{R}_s(\text{pre}_R(Q) \vdash \text{cmt}_R(Q)))$
by (simp add: extChoice-rdes unrest)
also have ... = $(\sqcap_{Q \in S} P \sqcap \text{CSP}(Q))$
by (simp add: UINF-as-Sup-collect, metis (no-types, lifting) Healthy-if SRD-as-reactive-design
assms(1))
also have ... = $(\sqcap_{Q \in S} P \sqcap Q)$
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 $(\sqcap_{i \in A} P i) ;; Q = (\sqcap_{i \in A} P i ;; Q)$

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 $1: (\sqcap_{i \in A} P i) = (\sqcap_{i \in A} (\mathbf{R}_s((\text{pre}_R(P i)) \vdash \text{peri}_R(P i) \diamond (R4(\text{post}_R(P i)))))$
(is ?X = ?Y)

by (rule ExtChoice-cong, metis (no-types, hide-lams) R4-def Healthy-if NCSP-implies-CSP PCSP-implies-NCSP

Productive-form assms(1) comp-apply
have $2: (\Box i \in A \cdot P \ i \ ; \ Q) = (\Box i \in A \cdot (\mathbf{R}_s ((pre_R (P \ i)) \vdash peri_R (P \ i) \diamond (R4(post_R (P \ i))))) \ ; \ Q)$
(is $?X = ?Y$ **)**
by (*rule ExtChoice-cong, metis (no-types, hide-lams) R4-def Healthy-if NCSP-implies-CSP PCSP-implies-NCSP*
Productive-form assms(1) comp-apply)
show $?thesis$
by (*simp add: 1 2, rdes-eq cls: assms False cong: ExtChoice-cong USUP-cong*)
qed
qed

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

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

end

8 Stateful-Failure Programs

theory *utp-sfrd-prog*
imports
utp-sfrd-extchoice
begin

8.1 Conditionals

lemma *NCSP-cond-srea [closure]*:
assumes $P \text{ is NCSP } Q \text{ is NCSP}$
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 $P \text{ is NCSP}$
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 $(if_R i \in A \cdot g(i) \rightarrow P(i) \text{ else } Q \text{ fi}) \text{ is NCSP}$
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 While Loops

lemma *NSRD-coerce-NCSP*:

P is NSRD \implies Skip ;; P ;; Skip is NCSP

by (metis (no-types, hide-lams) CSP3-Skip CSP3-def CSP4-def Healthy-def NCSP-Skip NCSP-implies-CSP
NCSP-intro NSRD-is-SRD RA1 SRD-seqr-closure)

definition *WhileC* :: 's upred \Rightarrow ('s, 'e) action \Rightarrow ('s, 'e) action (while_C - do - od) **where**
while_C b do P od = Skip ;; while_R b do P od ;; Skip

lemma *WhileC-NCSP-closed* [closure]:

assumes *P is NCSP P is Productive*

shows while_C b do P od is NCSP

by (simp add: WhileC-def NSRD-coerce-NCSP assms closure)

8.5 Assignment

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

syntax

-assigns-csp :: svids \Rightarrow uexprs \Rightarrow logic ('(-) :=_C '(-))

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

translations

-assigns-csp xs vs => **CONST** *AssignsCSP* (-mk-usubst (CONST id) xs vs)

-assigns-csp x v <= **CONST** *AssignsCSP* (CONST subst-upd (CONST id) x v)

-assigns-csp x v <= -assigns-csp (-spvar x) v

x,y :=_C u,v <= **CONST** *AssignsCSP* (CONST subst-upd (CONST subst-upd (CONST id) (CONST svar x) u) (CONST svar y) v)

lemma *preR-AssignsCSP* [rdes]: pre_R($\langle \sigma \rangle_C$) = true_r
by (rel-auto)

lemma *periR-AssignsCSP* [rdes]: peri_R($\langle \sigma \rangle_C$) = false
by (rel-auto)

lemma *postR-AssignsCSP* [rdes]: post_R($\langle \sigma \rangle_C$) = Φ(true,σ,⟨⟩)
by (rel-auto)

lemma *AssignsCSP-rdes-def* [rdes-def] : $\langle \sigma \rangle_C$ = **R**_s(true_r \vdash false \diamond Φ(true,σ,⟨⟩))
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

8.6 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* ::

$(f \Rightarrow k \text{ set}) \Rightarrow (f \Rightarrow k \Rightarrow v \Rightarrow f) \Rightarrow (f \Rightarrow \sigma) \Rightarrow$
 $(k, \sigma) \text{ uexpr} \Rightarrow (v, \sigma) \text{ uexpr} \Rightarrow (\sigma, \varphi) \text{ action} \textbf{ where}$

[upred-defs, rdes-def]: *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

-csp-assign-upd :: *svid* \Rightarrow *logic* \Rightarrow *logic* \Rightarrow *logic* (-[-] :=_C - [0,0,72] 72)

translations

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

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]:

$\text{pre}_R(\text{AssignCSP-update domf updatef } x k v) = [k \in_u \text{uop domf } (\&x)]_{S<}$

by (rel-auto)

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

$\text{peri}_R(\text{AssignCSP-update domf updatef } x k v) = [k \notin_u \text{uop domf } (\&x)]_{S<}$

by (rel-simp)

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

$\text{post}_R(\text{AssignCSP-update domf updatef } x k v) =$

$(\Phi(\text{true}, [x \mapsto_s \text{trop updatef } (\&x) k v], \langle \rangle) \triangleleft k \in_u \text{uop domf } (\&x) \triangleright_R R1(\text{true}))$

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

8.7 State abstraction

lemma *ref-unrest-abs-st* [*unrest*]:
 $\$ref \# P \implies \$ref \# \langle P \rangle_S$
 $\$ref' \# P \implies \$ref' \# \langle P \rangle_S$
by (*rel-simp*)⁺

lemma *NCSP-state-srea* [*closure*]: *P* is *NCSP* \implies *state 'a* \cdot *P* is *NCSP*
apply (*rule NCSP-NSRD-intro*)
apply (*simp-all add: closure rdes*)
apply (*simp-all add: state-srea-def unrest closure*)
done

8.8 Assumptions

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

8.9 Guards

definition *GuardCSP* ::

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

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 *csp-do-cond-conj*:

assumes *P* is *CRR*
shows $(\lceil b \rceil_{S<} \wedge P) = \Phi(b, id, \langle \rangle) ;; P$
proof –
have $(\lceil b \rceil_{S<} \wedge CRR(P)) = \Phi(b, id, \langle \rangle) ;; CRR(P)$
by (*rel-auto*)
thus *?thesis*
by (*simp add: Healthy-if assms*)
qed

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

assumes *P* is *RR* *Q* is *CRR* *R* is *CRR*
shows $g \ \&_u \ \mathbf{R}_s(P \vdash Q \diamond R) = \mathbf{R}_s((\mathcal{I}(g, \langle \rangle) \Rightarrow_r P) \vdash ((\Phi(g, id, \langle \rangle) ;; Q) \vee \mathcal{E}(\neg g, \langle \rangle, \{\}_u)) \diamond (\Phi(g, id, \langle \rangle) ;; R))$
(is ?lhs = ?rhs)
proof –

have ?lhs = $\mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r P) \vdash ((P \Rightarrow_r Q) \triangleleft \lceil g \rceil_{S<} \triangleright (\$tr' =_u \$tr)) \diamond (\lceil g \rceil_{S<} \wedge (P \Rightarrow_r R)))$
 by (simp add: Guard-tri-design rdes assms closure)
 also have ... = $\mathbf{R}_s((\mathcal{I}(g, \langle \rangle) \Rightarrow_r P) \vdash ((\lceil g \rceil_{S<} \wedge Q) \vee \mathcal{E}(\neg g, \langle \rangle, \{\}_u)) \diamond (\lceil g \rceil_{S<} \wedge R))$
 by (rel-auto)
 also have ... = $\mathbf{R}_s((\mathcal{I}(g, \langle \rangle) \Rightarrow_r P) \vdash ((\Phi(g, id, \langle \rangle) ;; Q) \vee \mathcal{E}(\neg g, \langle \rangle, \{\}_u)) \diamond (\Phi(g, id, \langle \rangle) ;; R))$
 by (simp add: assms(2) assms(3) csp-do-cond-conj)
 finally show ?thesis .
 qed

lemma Guard-rdes-def':

assumes $\$ok' \# P$
 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 ... = $\mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r R1(R2c(pre_s \dagger P))) \vdash (\lceil g \rceil_{S<} \wedge R1(R2c(cmt_s \dagger (P \Rightarrow Q))) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
 by (simp add: rea-pre-RHS-design rea-cmt-RHS-design)
 also have ... = $\mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r R1(R2c(pre_s \dagger P))) \vdash R1(R2c(\lceil g \rceil_{S<} \wedge R1(R2c(cmt_s \dagger (P \Rightarrow Q)))) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
 by (metis (no-types, lifting) RHS-design-export-R1 RHS-design-export-R2c)
 also have ... = $\mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r R1(R2c(pre_s \dagger P))) \vdash R1(R2c(\lceil g \rceil_{S<} \wedge (cmt_s \dagger (P \Rightarrow Q)) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
 by (simp add: R1-R2c-commute R1-disj R1-extend-conj' R1-idem R2c-and R2c-disj R2c-idem)
 also have ... = $\mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r R1(R2c(pre_s \dagger P))) \vdash (\lceil g \rceil_{S<} \wedge (cmt_s \dagger (P \Rightarrow Q)) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
 by (metis (no-types, lifting) RHS-design-export-R1 RHS-design-export-R2c)
 also have ... = $\mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r R1(R2c(pre_s \dagger P))) \vdash cmt_s \dagger (\lceil g \rceil_{S<} \wedge (cmt_s \dagger (P \Rightarrow Q)) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
 by (simp add: rdes-export-cmt)
 also have ... = $\mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r R1(R2c(pre_s \dagger P))) \vdash cmt_s \dagger (\lceil g \rceil_{S<} \wedge (P \Rightarrow Q) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
 by (simp add: usubst)
 also have ... = $\mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r R1(R2c(pre_s \dagger P))) \vdash (\lceil g \rceil_{S<} \wedge (P \Rightarrow Q) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
 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 $\implies \text{pre}_R(b \&_u P) = ([b]_{S<} \Rightarrow_r \text{pre}_R P)$
 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 $\text{peri}_R(b \&_u P) = (\text{peri}_R P \triangleleft b \triangleright_R \mathcal{E}(\text{true}, \langle \rangle, \{\}_u))$

proof –

have $\text{peri}_R(b \&_u P) = (([b]_{S<} \Rightarrow_r \text{pre}_R P) \Rightarrow_r (\text{peri}_R P \triangleleft [b]_{S<} \triangleright (\$tr' =_u \$tr)))$

by (simp add: asms 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 $\dots = ((\text{pre}_R P \Rightarrow_r \text{peri}_R P) \triangleleft [b]_{S<} \triangleright (\$tr' =_u \$tr))$

by (rel-auto)

also have $\dots = (\text{peri}_R P \triangleleft [b]_{S<} \triangleright (\$tr' =_u \$tr))$

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

finally show ?thesis

by rel-auto

qed

lemma *postR-Guard* [rdes]:

assumes P is NCSP

shows $\text{post}_R(b \&_u P) = ([b]_{S<} \wedge \text{post}_R P)$

proof –

have $\text{post}_R(b \&_u P) = (([b]_{S<} \Rightarrow_r \text{pre}_R P) \Rightarrow_r ([b]_{S<} \wedge \text{post}_R P))$

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 asms)

also have $\dots = ([b]_{S<} \wedge (\text{pre}_R P \Rightarrow_r \text{post}_R P))$

by (rel-auto)

also have $\dots = ([b]_{S<} \wedge \text{post}_R P)$

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

also have $\dots = ([b]_{S<} \wedge \text{post}_R P)$

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 asms have $1:\$ref \# P \llbracket \text{false}/\$wait \rrbracket$

by (simp add: CSP-Guard CSP3-iff)

hence $\$ref \# \text{pre}_R (P \llbracket 0/\$tr \rrbracket) \ \$ref \# \text{pre}_R P \ \$ref \# \text{cmt}_R P$

by (pred-blast)+

hence $\$ref \# (b \&_u P) \llbracket \text{false}/\$wait \rrbracket$

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  $a: (\neg_r \text{pre}_R P) \;; R1 \text{ true} = (\neg_r \text{pre}_R P)$ 
      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  $2: \dots = ([b]_{S<} \wedge ((\neg_r \text{pre}_R P) \;; R1 \text{ true}))$ 
        by (simp add: a)
      also have  $3: \dots = (\neg_r ([b]_{S<} \Rightarrow_r \text{pre}_R P)) \;; R1 \text{ true}$ 
        by (rel-auto)
      finally show ?thesis ..
    qed
  thus ?thesis
    by (simp add: preR-Guard periR-Guard NSRD-CSP4-intro closure assms unrest)
  qed
show  $\$st' \# \text{peri}_R (b \&_u P)$ 
  by (simp add: preR-Guard periR-Guard NSRD-CSP4-intro closure assms unrest)
show  $\$ref' \# \text{post}_R (b \&_u P)$ 
  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  $\$wait' \# \text{pre}_R(P)$ 
  shows  $b \&_u P$  is Productive
proof –
  have  $b \&_u P = b \&_u \mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \diamond (\text{post}_R(P) \wedge \$tr <_u \$tr'))$ 
    by (metis Healthy-def Productive-form assms(1) assms(2))
  also have  $\dots =$ 
     $\mathbf{R}_s (([b]_{S<} \Rightarrow_r \text{pre}_R P) \vdash$ 
       $((\text{pre}_R P \Rightarrow_r \text{peri}_R P) \triangleleft [b]_{S<} \triangleright (\$tr' =_u \$tr)) \diamond ([b]_{S<} \wedge (\text{pre}_R P \Rightarrow_r \text{post}_R P \wedge \$tr' >_u$ 
       $\$tr)))$ 
    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  $\dots = \mathbf{R}_s (([b]_{S<} \Rightarrow_r \text{pre}_R P) \vdash (\text{peri}_R P \triangleleft [b]_{S<} \triangleright (\$tr' =_u \$tr)) \diamond (([b]_{S<} \wedge \text{post}_R P)$ 
     $\wedge \$tr' >_u \$tr))$ 
    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

8.10 Basic events

definition $do_u ::$

$(\sigma, \sigma') \text{ uexpr} \Rightarrow (\sigma, \sigma') \text{ action}$ **where**
 $[upred-defs]: do_u e = ((\sigma' =_u \sigma \wedge \lceil e \rceil_{S<} \notin_u \sigma') \triangleleft \sigma' \triangleright (\sigma' =_u \sigma \wedge \lceil e \rceil_{S<} \wedge \sigma' =_u \sigma))$

definition $DoCSP :: (\sigma, \sigma') \text{ uexpr} \Rightarrow (\sigma, \sigma') \text{ action}$ (do_C) **where**

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

lemma $R1\text{-}DoAct: R1(do_u(a)) = do_u(a)$

by (*rel-auto*)

lemma $R2c\text{-}DoAct: R2c(do_u(a)) = do_u(a)$

by (*rel-auto*)

lemma $DoCSP\text{-}alt\text{-}def: do_C(a) = R3h(CSP1(\sigma' \wedge do_u(a)))$

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

$R2c\text{-}not\ R2c\text{-}ok\ R2c\text{-}ok'\ R2c\text{-}and\ R2c\text{-}DoAct\ R1\text{-}disj\ R1\text{-}extend\text{-}conj'\ R1\text{-}DoAct$

apply (*rel-auto*)

done

lemma $DoAct\text{-}unrests\ [unrest]:$

$\sigma \not\# do_u(a)\ \sigma' \not\# do_u(a)$

by (*pred-auto*) $+$

lemma $DoCSP\text{-}RHS\text{-}tri\ [rdes\text{-}def]:$

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

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

lemma $CSP\text{-}DoCSP\ [closure]: do_C(a)$ is CSP

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

lemma $preR\text{-}DoCSP\ [rdes]: pre_R(do_C(a)) = \text{true}_r$

by (*simp add: DoCSP-def rea-pre-RHS-design unrest usubst R2c-true*)

lemma $periR\text{-}DoCSP\ [rdes]: peri_R(do_C(a)) = \mathcal{E}(\text{true}, \langle \rangle, \{a\}_u)$

by (*rel-auto*)

lemma $postR\text{-}DoCSP\ [rdes]: post_R(do_C(a)) = \Phi(\text{true}, id, \langle a \rangle)$

by (*rel-auto*)

lemma $CSP3\text{-}DoCSP\ [closure]: do_C(a)$ 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 $CSP4\text{-}DoCSP\ [closure]: do_C(a)$ is $CSP4$

by (*rule CSP4-tri-intro[OF CSP-DoCSP], simp-all add: preR-DoCSP periR-DoCSP postR-DoCSP unrest*)

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

lemma *Productive-DoCSP* [closure]:
 $(do_C a :: ('\sigma, '\psi) \text{ action})$ is Productive

proof –

have $((\Phi(true, id, \langle a \rangle) \wedge \$tr' >_u \$tr) :: ('\sigma, '\psi) \text{ action})$
 $= (\Phi(true, id, \langle a \rangle))$
 by (rel-auto, simp add: Prefix-Order.strict-prefixI')
 hence $Productive(do_C a) = do_C a$
 by (simp add: Productive-RHS-design-form DoCSP-RHS-tri unrest)
 thus ?thesis
 by (simp add: Healthy-def)

qed

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

lemma *wp-rea-DoCSP-lemma*:

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

lemma *wp-rea-DoCSP*:

assumes P is NCSP
 shows $(\$tr' =_u \$tr \hat{^}_u \langle [a]_{S<} \rangle \wedge \$st' =_u \$st) \text{ wp}_r \text{ pre}_R P =$
 $(\neg_r (\neg_r \text{pre}_R P)[\$tr \hat{^}_u \langle [a]_{S<} \rangle / \$tr])$
 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 $(\$tr' =_u \$tr \hat{^}_u \langle [a]_{S<} \rangle \wedge \$st' =_u \$st) \text{ wp}_r \text{ pre}_R P =$
 $(\$tr' \geq_u \$tr \hat{^}_u \langle [a]_{S<} \rangle \Rightarrow_r (\text{pre}_R P)[\$tr \hat{^}_u \langle [a]_{S<} \rangle / \$tr])$
 by (simp add: wp-rea-DoCSP assms rea-not-def R1-def usubst unrest, rel-auto)

8.11 Event prefix

definition *PrefixCSP* ::

$('\varphi, '\sigma) \text{ uexpr} \Rightarrow$
 $('\sigma, '\varphi) \text{ action} \Rightarrow$
 $('\sigma, '\varphi) \text{ action} (- \rightarrow_C - [81, 80] 80) \text{ where}$
 $[upred-defs]: \text{PrefixCSP } a \text{ } P = (do_C(a) ;; \text{CSP}(P))$

abbreviation *OutputCSP* $c \text{ } v \text{ } P \equiv \text{PrefixCSP } (c \cdot v)_u \text{ } P$

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

by (simp add: PrefixCSP-def closure)

lemma *CSP3-PrefixCSP* [closure]:

PrefixCSP $a \text{ } P$ is CSP3
 by (metis (no-types, hide-lams) *CSP3-DoCSP CSP3-def Healthy-def PrefixCSP-def segr-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 $P :: ('σ, 'φ)$ 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 $P :: ('σ, 'φ)$ 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 is R1 $\implies P[\$tr \hat{^}_u e / \$tr]$ is R1
by (rel-blast)

lemma *preR-PrefixCSP-NCSP* [rdes]:
assumes P is NCSP
shows $pre_R(PrefixCSP \ a \ P) = (\mathcal{I}(true, \langle a \rangle) \Rightarrow_r (pre_R \ P) \llbracket \langle a \rangle \rrbracket_t)$

by (simp add: PrefixCSP-def assms closure rdes rpred Healthy-if wp usubst unrest)

lemma *periR-PrefixCSP* [rdes]:

assumes *P* is NCSP

shows $\text{peri}_R(\text{PrefixCSP } a \ P) = (\mathcal{E}(\text{true}, \langle \rangle, \{a\}_u) \vee (\text{peri}_R \ P) \llbracket \langle a \rangle \rrbracket_t)$

proof –

have $\text{peri}_R(\text{PrefixCSP } a \ P) = \text{peri}_R(\text{do}_C \ a \ ; \ P)$

by (simp add: PrefixCSP-def closure assms Healthy-if)

also have $\dots = ((\mathcal{I}(\text{true}, \langle a \rangle)) \Rightarrow_r \text{pre}_R \ P \llbracket \langle a \rangle \rrbracket_t) \Rightarrow_r \$tr' =_u \$tr \wedge \lceil a \rceil_{S<} \notin_u \$ref' \vee \text{peri}_R \ P \llbracket \langle a \rangle \rrbracket_t)$

by (simp add: assms NSRD-CSP₄-intro csp-enable-tr-empty closure rdes unrest ex-unrest usubst rpred wp)

also have $\dots = (\mathcal{E}(\text{true}, \langle \rangle, \{a\}_u) \vee ((\mathcal{I}(\text{true}, \langle a \rangle)) \Rightarrow_r \text{pre}_R \ P \llbracket \langle a \rangle \rrbracket_t) \Rightarrow_r \text{peri}_R \ P \llbracket \langle a \rangle \rrbracket_t)$

by (rel-auto)

also have $\dots = (\mathcal{E}(\text{true}, \langle \rangle, \{a\}_u) \vee ((\text{pre}_R(P) \Rightarrow_r \text{peri}_R \ P) \llbracket \langle a \rangle \rrbracket_t))$

by (rel-auto)

also have $\dots = (\mathcal{E}(\text{true}, \langle \rangle, \{a\}_u) \vee (\text{peri}_R \ P) \llbracket \langle a \rangle \rrbracket_t)$

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

finally show ?thesis .

qed

lemma *postR-PrefixCSP* [rdes]:

assumes *P* is NCSP

shows $\text{post}_R(\text{PrefixCSP } a \ P) = (\text{post}_R \ P) \llbracket \langle a \rangle \rrbracket_t$

proof –

have $\text{post}_R(\text{PrefixCSP } a \ P) = ((\mathcal{I}(\text{true}, \langle a \rangle)) \Rightarrow_r (\text{pre}_R \ P) \llbracket \langle a \rangle \rrbracket_t) \Rightarrow_r (\text{post}_R \ P) \llbracket \langle a \rangle \rrbracket_t)$

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 $\dots = (\mathcal{I}(\text{true}, \langle a \rangle)) \wedge (\text{pre}_R \ P \Rightarrow_r \text{post}_R \ P) \llbracket \langle a \rangle \rrbracket_t)$

by (rel-auto)

also have $\dots = (\mathcal{I}(\text{true}, \langle a \rangle)) \wedge (\text{post}_R \ P) \llbracket \langle a \rangle \rrbracket_t)$

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

also have $\dots = (\text{post}_R \ P) \llbracket \langle a \rangle \rrbracket_t$

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 $PrefixCSP\ a\ (\mathbf{R}_s(P \vdash Q \diamond R)) = \mathbf{R}_s((\mathcal{I}(true, \langle a \rangle) \Rightarrow_r P \llbracket \langle a \rangle \rrbracket_t) \vdash (\mathcal{E}(true, \langle \rangle, \{a\}_u) \vee (P \wedge Q) \llbracket \langle a \rangle \rrbracket_t) \diamond (P \wedge R) \llbracket \langle a \rangle \rrbracket_t)$

apply (*subst PrefixCSP-RHS-tri*)

apply (*rule NCSP-rdes-intro*)

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

apply (*rel-auto*)

done

8.12 Guarded external choice

abbreviation *GuardedChoiceCSP* :: $'\vartheta$ set $\Rightarrow (' \vartheta \Rightarrow (' \sigma, ' \vartheta)$ action) $\Rightarrow (' \sigma, ' \vartheta)$ action **where**
GuardedChoiceCSP $A\ P \equiv (\Box x \in A \cdot PrefixCSP \ll x \gg (P(x)))$

syntax

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

translations

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

lemma *GuardedChoiceCSP [rdes-def]*:

assumes $\bigwedge x. P(x)$ is NCSP $A \neq \{\}$

shows $(\Box x \in A \rightarrow P(x)) =$

$\mathbf{R}_s((\bigcup x \in A \cdot \mathcal{I}(true, \langle \ll x \gg \rangle) \Rightarrow_r pre_R(P\ x) \llbracket \langle \ll x \gg \rangle \rrbracket_t) \vdash$
 $((\bigcup x \in A \cdot \mathcal{E}(true, \langle \rangle, \{\ll x \gg\}_u)) \triangleleft \$tr' =_u \$tr \triangleright (\bigcap x \in A \cdot peri_R(P\ x) \llbracket \langle \ll x \gg \rangle \rrbracket_t)) \diamond$
 $(\bigcap x \in A \cdot post_R(P\ x) \llbracket \langle \ll x \gg \rangle \rrbracket_t))$

by (*simp add: PrefixCSP-RHS-tri assms ExtChoice-tri-rdes closure unrest, rel-auto*)

8.13 Input prefix

definition *InputCSP* ::

$('a, ' \vartheta)$ chan $\Rightarrow ('a \Rightarrow ' \sigma\ upred) \Rightarrow ('a \Rightarrow (' \sigma, ' \vartheta)$ action) $\Rightarrow (' \sigma, ' \vartheta)$ action **where**

[*upred-defs*]: *InputCSP* $c\ A\ P = (\Box x \in UNIV \cdot A(x) \ \&_u\ PrefixCSP\ (c \cdot \ll x \gg)_u\ (P\ x))$

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

[*upred-defs, rdes-def*]: *InputVarCSP* $c\ x\ A\ P = InputCSP\ c\ A\ (\lambda v. \langle [x \mapsto_s \ll v \gg] \rangle_C) ;; P$

definition *do_I* ::

$('a, ' \vartheta)$ chan \Rightarrow

$('a \Longrightarrow (' \sigma, ' \vartheta)$ st-csp) \Rightarrow

$('a \Rightarrow (' \sigma, ' \vartheta)$ action) \Rightarrow

$(' \sigma, ' \vartheta)$ 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-CSP [closure]*: *InputCSP* $c\ A\ P$ is CSP

by (*simp add: CSP-ExtChoice InputCSP-def*)

lemma *InputCSP-NCSP [closure]*: $\ll \bigwedge v. P(v)$ is NCSP $\rrbracket \Longrightarrow InputCSP\ c\ A\ P$ 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-InputCSP* [closure]:

$\llbracket \bigwedge v. P(v) \text{ is NCSP} \rrbracket \implies \text{InputCSP } x \ A \ P \text{ is Productive}$

by (*auto simp add: InputCSP-def unrest closure intro: Productive-ExtChoice*)

lemma *preR-InputCSP* [rdes]:

assumes $\bigwedge v. P(v) \text{ is NCSP}$

shows $\text{pre}_R(\text{InputCSP } a \ A \ P) = (\bigsqcup v \cdot [A(v)]_{S<} \Rightarrow_r \mathcal{I}(\text{true}, \langle (a \cdot \ll v \gg)_u \rangle) \Rightarrow_r (\text{pre}_R(P(v))) \llbracket \langle (a \cdot \ll v \gg)_u \rangle \rrbracket_t)$

by (*simp add: InputCSP-def rdes closure assms alpha usubst unrest*)

lemma *periR-InputCSP* [rdes]:

assumes $\bigwedge v. P(v) \text{ is NCSP}$

shows $\text{peri}_R(\text{InputCSP } a \ A \ P) =$

$((\bigsqcup x \cdot [A(x)]_{S<} \Rightarrow_r \mathcal{E}(\text{true}, \langle \rangle, \{(a \cdot \ll x \gg)_u\}_u)) \triangleleft \$tr' =_u \$tr \triangleright$

$(\bigsqcap x \cdot [A(x)]_{S<} \wedge (\text{peri}_R(P \ x)) \llbracket \langle (a \cdot \ll x \gg)_u \rangle \rrbracket_t)$

by (*simp add: InputCSP-def rdes closure assms, rel-auto*)

lemma *postR-InputCSP* [rdes]:

assumes $\bigwedge v. P(v) \text{ is NCSP}$

shows $\text{post}_R(\text{InputCSP } a \ A \ P) =$

$(\bigsqcap x \cdot [A \ x]_{S<} \wedge \text{post}_R(P \ x) \llbracket \langle (a \cdot \ll x \gg)_u \rangle \rrbracket_t)$

using *assms* **by** (*simp add: InputCSP-def rdes closure assms usubst unrest*)

lemma *InputCSP-rdes-def* [rdes-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 $\text{InputCSP } a \ A \ (\lambda v. \mathbf{R}_s(P(v) \vdash Q(v) \diamond R(v))) =$

$\mathbf{R}_s((\bigsqcup v \cdot ([A(v)]_{S<} \Rightarrow_r \mathcal{I}(\text{true}, \langle (a \cdot \ll v \gg)_u \rangle) \Rightarrow_r (P \ v)) \llbracket \langle (a \cdot \ll v \gg)_u \rangle \rrbracket_t) \vdash (((\bigsqcup x \cdot R5([A(x)]_{S<} \Rightarrow_r \mathcal{E}(\text{true}, \langle \rangle, \{(a \cdot \ll x \gg)_u\}_u)))$

\vee

$(\bigsqcap x \cdot [A(x)]_{S<} \wedge (P \ x \wedge Q \ x) \llbracket \langle (a \cdot \ll x \gg)_u \rangle \rrbracket_t))$

$\diamond (\bigsqcap x \cdot [A \ x]_{S<} \wedge (P \ x \wedge R \ x) \llbracket \langle (a \cdot \ll x \gg)_u \rangle \rrbracket_t))$ (**is** *?lhs* = *?rhs*)

proof –

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

by (*simp add: rdes NCSP-rdes-intro assms conj-comm closure*)

have *2:peri_R(?lhs)* = $(\bigsqcup x \cdot [A \ x]_{S<} \Rightarrow_r \mathcal{E}(\text{true}, \langle \rangle, \{(a \cdot \ll x \gg)_u\}_u)) \triangleleft \$tr' =_u \$tr \triangleright (\bigsqcap x \cdot [A \ x]_{S<} \wedge (P \ x \Rightarrow_r Q \ x) \llbracket \langle (a \cdot \ll x \gg)_u \rangle \rrbracket_t)$

$\wedge (P \ x \Rightarrow_r Q \ x) \llbracket \langle (a \cdot \ll x \gg)_u \rangle \rrbracket_t$

(**is** - = *?B*)

by (*simp add: rdes NCSP-rdes-intro assms closure*)

have *3:post_R(?lhs)* = $(\bigsqcap x \cdot [A \ x]_{S<} \wedge (P \ x \Rightarrow_r R \ x) \llbracket \langle (a \cdot \ll x \gg)_u \rangle \rrbracket_t)$

(**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

8.14 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*: $\langle id \rangle_C = Skip$
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*]: $false \&_u P = Stop$
by (*simp add: GuardCSP-def Stop-def rpred closure alpha R1-design-R1-pre*)

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

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

lemma *Guard-expansion*:
 $(g_1 \vee g_2) \&_u P = (g_1 \&_u P) \sqcap (g_2 \&_u P)$
by (*rel-auto*)

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; simp add: le-less*)

lemma *PrefixCSP-dist*:
 $PrefixCSP a (P \sqcap Q) = (PrefixCSP a P) \sqcap (PrefixCSP a Q)$
using *Continuous-Disjunctous Disjunctuous-def PrefixCSP-Continuous* **by** *auto*

lemma *DoCSP-is-Prefix*:
 $do_C(a) = PrefixCSP a Skip$
by (*simp add: PrefixCSP-def Healthy-if closure, metis CSP4-DoCSP CSP4-def Healthy-def*)

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

lemma *PrefixCSP-extChoice-dist*:
assumes $P \text{ is NCSP } Q \text{ is NCSP } R \text{ is NCSP}$
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 $\bigwedge i. i \in A \implies P(i) \text{ is NCSP } Q \text{ is NCSP}$
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

```

declare ExtChoice-tri-rdes [rdes-def]
declare ExtChoice-tri-rdes' [rdes-def del]

declare extChoice-rdes-def [rdes-def]
declare extChoice-rdes-def' [rdes-def del]

lemma AlternateR-as-ExtChoice:
  assumes
     $\bigwedge i. i \in A \implies P(i) \text{ is NCSP } Q \text{ is NCSP}$ 
     $\bigwedge i j. \llbracket i \in A; j \in A; i \neq j \rrbracket \implies (g i \wedge g j) = \text{false}$ 
  shows  $(\text{if}_R i \in A \cdot g(i) \rightarrow P(i) \text{ else } Q \text{ fi}) =$ 
     $(\Box i \in A \cdot g(i) \ \&_u \ P(i)) \Box (\bigwedge i \in A \cdot \neg g(i) \ \&_u \ Q)$ 
proof (cases  $A = \{\}$ )
  case True
    then show ?thesis by (simp add: ExtChoice-empty extChoice-Stop closure assms)
  next
    case False
    show ?thesis

  proof –
    have  $1: (\Box i \in A \cdot g i \rightarrow_R P i) = (\Box i \in A \cdot g i \rightarrow_R \mathbf{R}_s(\text{pre}_R(P i) \vdash \text{peri}_R(P i) \diamond \text{post}_R(P i)))$ 
      by (rule UINF-cong, simp add: NCSP-implies-CSP SRD-reactive-tri-design assms(1))
    have  $2: (\Box i \in A \cdot g(i) \ \&_u \ P(i)) = (\Box i \in A \cdot g(i) \ \&_u \ \mathbf{R}_s(\text{pre}_R(P i) \vdash \text{peri}_R(P i) \diamond \text{post}_R(P i)))$ 
      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)
    (rdes-eq cls: assms(1-2)_simps: False cong: UINF-cong ExtChoice-cong)
  qed
qed

declare ExtChoice-tri-rdes [rdes-def del]
declare ExtChoice-tri-rdes' [rdes-def]

declare extChoice-rdes-def [rdes-def del]
declare extChoice-rdes-def' [rdes-def]

end

```

9 Recursion in Stateful-Failures

```

theory utp-sfrd-recursion
  imports utp-sfrd-contracts utp-sfrd-prog
begin

```

9.1 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

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

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*]: $(\mu_C \ (op ;; Q)) = (\mu \ X \cdot Q ;; \text{CSP } X)$

by (*simp add: comp-def*)

lemma *mu-csp-basic-refine*:

assumes

$P \text{ is CSP } Q \text{ is NCSP } Q \text{ is Productive } \text{pre}_R(P) = \text{true}_r \ \text{pre}_R(Q) = \text{true}_r$

$\text{peri}_R P \sqsubseteq \text{peri}_R Q$

$\text{peri}_R P \sqsubseteq \text{post}_R Q ;; \text{peri}_R P$

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 $\text{peri}_R P \sqsubseteq (\bigcap i \cdot \text{post}_R Q \wedge i ;; \text{peri}_R Q)$

proof (*rule UINF-refines'*)

fix i

show $\text{peri}_R P \sqsubseteq \text{post}_R Q \wedge i ;; \text{peri}_R Q$

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 $P :: 'e \text{ list} \Rightarrow 'e \text{ set} \Rightarrow 's \text{ upred}$

assumes

$Q \text{ is NCSP } Q \text{ is Productive } \text{pre}_R(Q) = \text{true}_r$

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

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

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

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

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

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

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

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

qed

9.2 Example action expansion

lemma *mu-example1*: $(\mu X \cdot \ll a \gg \rightarrow_C X) = (\bigcap i \cdot do_C(\ll a \gg) \wedge (i+1)) \;; \text{Miracle}$
by (*simp add: PrefixCSP-def mu-csp-form-1 closure*)

lemma *preR-mu-example1* [*rdes*]: $pre_R(\mu X \cdot \ll a \gg \rightarrow_C X) = true_r$
by (*simp add: mu-example1 rdes closure unrest wp*)

lemma *periR-mu-example1* [*rdes*]:
 $peri_R(\mu X \cdot \ll a \gg \rightarrow_C X) = (\bigcap i \cdot \mathcal{E}(true, iter[i](\ll a \gg), \{\ll a \gg\}_u))$
by (*simp add: mu-example1 rdes rpred closure unrest wp seq-UNF-distr alpha usubst*)

lemma *postR-mu-example1* [*rdes*]:
 $post_R(\mu X \cdot \ll a \gg \rightarrow_C X) = false$
by (*simp add: mu-example1 rdes closure unrest wp*)

end

10 Linking to the Failures-Divergences Model

theory *utp-sfrd-fdsem*
imports *utp-sfrd-recursion*
begin

10.1 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*]: *divergences* $P \ s = \{t \mid t. '(\neg_r \ pre_R(P)) \ll s \gg, \langle \rangle, \ll t \gg / \$st, \$str, \$tr '\}$

definition *traces* :: $('σ, 'φ) \text{ action} \Rightarrow 'σ \Rightarrow ('φ \text{ list} \times 'σ) \text{ set } (tr[-] - [0,100] \ 100)$ **where**
[*upred-defs*]: *traces* $P \ s = \{(t, s') \mid t \ s'. '(pre_R(P) \wedge post_R(P)) \ll s \gg, \ll s' \gg, \langle \rangle, \ll t \gg / \$st, \$st', \$tr, \$tr '\}$

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*]: *failures* $P \ s = \{(t, r) \mid t \ r. '(pre_R(P) \wedge peri_R(P)) \ll r \gg, \ll s \gg, \langle \rangle, \ll t \gg / \$ref', \$st, \$tr, \$tr '\}$

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 $P \text{ is NCSP } Q \text{ is NCSP } \bigwedge s. dv[P] \ s \subseteq dv[Q] \ s$
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 $assms(\mathcal{J})[of\ s]$ **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 $\bigwedge s. dv[P]s = dv[Q]s$
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 $\bigwedge s. fl[Q]s \subseteq fl[P]s$
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 $assms(\mathcal{J})[of\ s]$ **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 $\bigwedge s. fl[P]s = fl[Q]s$
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 $\bigwedge s. tr[Q]s \subseteq tr[P]s$
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 $assms(\mathcal{J})[of\ s]$ **have** b : $(t, s') \in tr[Q]s \implies (t, s') \in tr[P]s$
by (*auto*)
with a **show** $[\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg] \dagger (pre_R P \wedge post_R P)$
by (*simp add: traces-def*)
qed

lemma *postR-eq-traces*:

assumes P is NCSP Q is NCSP $\bigwedge s. tr[P]s = tr[Q]s$
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 $\wedge s. dv\llbracket Q \rrbracket s \subseteq dv\llbracket P \rrbracket s \wedge s. fl\llbracket Q \rrbracket s \subseteq fl\llbracket P \rrbracket s \wedge s. tr\llbracket Q \rrbracket s \subseteq tr\llbracket P \rrbracket s$
 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

10.2 Circus Operators

lemma *traces-Skip*:

$tr\llbracket Skip \rrbracket s = \{(\llbracket, s\rrbracket)\}$

by (simp add: *traces-def rdes alpha closure, rel-simp*)

lemma *failures-Skip*:

$fl\llbracket Skip \rrbracket s = \{\}$

by (simp add: *failures-def, rdes-calc*)

lemma *divergences-Skip*:

$dv\llbracket Skip \rrbracket s = \{\}$

by (simp add: *divergences-def, rdes-calc*)

lemma *traces-Stop*:

$tr\llbracket Stop \rrbracket s = \{\}$

by (simp add: *traces-def, rdes-calc*)

lemma *failures-Stop*:

$fl\llbracket Stop \rrbracket s = \{(\llbracket, E) \mid E. True\}$

by (simp add: *failures-def, rdes-calc, rel-auto*)

lemma *divergences-Stop*:

$dv\llbracket Stop \rrbracket s = \{\}$

by (simp add: *divergences-def, rdes-calc*)

lemma *traces-AssignsCSP*:

$tr\llbracket \langle \sigma \rangle_C \rrbracket s = \{(\llbracket, \sigma(s))\}$

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*: $fl\llbracket do_C(e) \rrbracket s = \{([], E) \mid E. \llbracket e \rrbracket_e s \notin E\}$
 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 *divergences-seq*:
 fixes $P :: ('s, 'e)$ action
 assumes P is NCSP Q is NCSP
 shows $dv\llbracket P ;; Q \rrbracket s = dv\llbracket P \rrbracket s \cup \{t_1 @ t_2 \mid t_1 \ t_2 \ s_0. (t_1, s_0) \in tr\llbracket P \rrbracket s \wedge t_2 \in dv\llbracket Q \rrbracket s_0\}$
 (is ?lhs = ?rhs)

oops

lemma *traces-seq*:

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

assumes $P \text{ is NCSP } Q \text{ is NCSP}$

shows $tr[P ;; Q]_s =$

$$\begin{aligned} & \{ (t_1 @ t_2, s') \mid t_1 \ t_2 \ s_0 \ s'. (t_1, s_0) \in tr[P]_s \wedge (t_2, s') \in tr[Q]_{s_0} \\ & \quad \wedge (t_1 @ t_2) \notin dv[P]_s \\ & \quad \wedge (\forall (t, s_1) \in tr[P]_s. t \leq t_1 @ t_2 \longrightarrow (t_1 @ t_2) - t \notin dv[Q]_{s_1}) \} \end{aligned}$$

(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 $t :: 'e \text{ list}$ and $s' :: 's$

let $\sigma = [\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg]$

assume

a1: $\sigma \vdash (post_R P ;; post_R Q)'$ and

a2: $[\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg] \vdash pre_R P'$ and

a3: $[\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg] \vdash (post_R P \text{ wr } pre_R Q)'$

from a1 have $\sigma \vdash (\exists tr_0. ((post_R P) \ll tr_0 \gg / \$tr') ;; (post_R Q) \ll tr_0 \gg / \$tr) \wedge \ll tr_0 \gg \leq_u \tr'

by (simp add: R2-tr-middle assms closure)

then obtain tr_0 where $p1: \sigma \vdash ((post_R P) \ll tr_0 \gg / \$tr') ;; (post_R Q) \ll tr_0 \gg / \tr and $tr_0: tr_0$

$\leq t$

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 $\sigma \vdash (\exists st_0. (post_R P) \ll tr_0 \gg / \$tr' \ll st_0 \gg / \$st' ;; (post_R Q) \ll tr_0 \gg / \$tr \ll st_0 \gg / \$st)'$

by (simp add: seqr-middle[of st, THEN sym])

then obtain s_0 where $\sigma \vdash ((post_R P) \ll s_0 \gg, \ll tr_0 \gg / \$st', \$tr') ;; (post_R Q) \ll s_0 \gg, \ll tr_0 \gg / \$st, \$tr)$

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 $(([\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll tr_0 \gg] \vdash post_R P) ;;$

$[\$st \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \ll tr_0 \gg, \$tr' \mapsto_s \ll t \gg] \vdash post_R Q))'$

by (rel-auto)

hence $(([\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll tr_0 \gg] \vdash post_R P) \wedge$

$[\$st \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \ll tr_0 \gg, \$tr' \mapsto_s \ll t \gg] \vdash post_R Q))'$

by (simp add: seqr-to-conj unrest-any-circus-var assms closure unrest)

hence $postP: ([\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll tr_0 \gg] \vdash post_R P)'$ and

$postQ': ([\$st \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \ll tr_0 \gg, \$tr' \mapsto_s \ll t \gg] \vdash post_R Q)'$

by (rel-auto)+

from $postQ'$ have $[\$st \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg] \vdash [\$tr \mapsto_s \ll tr_0 \gg, \$tr' \mapsto_s \ll tr_0 \gg + (\ll t \gg - \ll tr_0 \gg)] \vdash post_R Q'$

using tr_0 by (rel-auto)

hence $[\$st \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg] \vdash [\$tr \mapsto_s 0, \$tr' \mapsto_s \ll t \gg - \ll tr_0 \gg] \vdash post_R Q'$

by (simp add: R2-subst-tr closure assms)

hence $postQ: [\$st \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t - tr_0 \gg] \vdash post_R Q'$

by (rel-auto)

have $preP: [\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll tr_0 \gg] \vdash pre_R P'$

proof –

have $(pre_R P)[[0, \ll tr_0 \gg / \$tr, \$tr']] \sqsubseteq (pre_R P)[[0, \ll t \gg / \$tr, \$tr']]$
by $(simp \text{ add: } RC\text{-prefix-refine closure assms } tr0)$
hence $[\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll tr_0 \gg] \dagger pre_R P \sqsubseteq [\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll tr_0 \gg] \dagger pre_R P$
by $(rel\text{-auto})$
thus $?thesis$
by $(simp \text{ add: } taut\text{-refine-impl } a2)$
qed

have $preQ: '[\$st \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t - tr_0 \gg] \dagger pre_R Q'$

proof –

from $postP \ a3$ **have** $'[\$st \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \ll tr_0 \gg, \$tr' \mapsto_s \ll t \gg] \dagger pre_R Q'$
apply $(simp \text{ add: } wp\text{-rea-def})$
apply $(rel\text{-auto})$
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 $(rel\text{-auto})$

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 $(simp \text{ add: } R2\text{-subst-tr closure assms})$
thus $?thesis$
by $(rel\text{-auto})$
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 $(rel\text{-auto})$

have $t\text{-minus-}tr0: tr_0 @ (t - tr_0) = t$
using $append\text{-minus } tr0$ **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 $(simp \text{ add: } wp\text{-rea-circus-form-alt[of } post_R P \ pre_R Q] \text{ closure assms unrest usubst})$
 $(rel\text{-simp})$

from $c \ b(2-4)$ **show** $False$
by $(rel\text{-auto})$
qed

show $\exists t_1 \ t_2.$

$t = t_1 @ t_2 \wedge$
 $(\exists s_0. \text{'[}\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_1 \gg\text{' } \dagger \text{ } pre_R P \wedge$
 $\text{'[}\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_1 \gg\text{' } \dagger \text{ } post_R P' \wedge$
 $\text{'[}\$st \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_2 \gg\text{' } \dagger \text{ } pre_R Q \wedge$
 $\text{'[}\$st \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_2 \gg\text{' } \dagger \text{ } post_R Q' \wedge$
 $\neg \text{'[}\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_1 @ t_2 \gg\text{' } \dagger (\neg_r pre_R P)' \wedge$
 $(\forall t_0 s_1. \text{'[}\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_0 \gg\text{' } \dagger \text{ } pre_R P \wedge$
 $\text{'[}\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_0 \gg\text{' } \dagger \text{ } post_R P' \longrightarrow$
 $t_0 \leq t_1 @ t_2 \longrightarrow \neg \text{'[}\$st \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll (t_1 @ t_2) - t_0 \gg\text{' } \dagger (\neg_r$
 $pre_R Q)'$))
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

$a1: \neg \text{'[}\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_1 @ t_2 \gg\text{' } \dagger (\neg_r pre_R P)'$ **and**
 $a2: \text{'[}\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_1 \gg\text{' } \dagger \text{ } pre_R P'$ **and**
 $a3: \text{'[}\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_1 \gg\text{' } \dagger \text{ } post_R P'$ **and**
 $a4: \text{'[}\$st \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_2 \gg\text{' } \dagger \text{ } pre_R Q'$ **and**
 $a5: \text{'[}\$st \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_2 \gg\text{' } \dagger \text{ } post_R Q'$ **and**
 $a6: \forall t s_1. \text{'[}\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg\text{' } \dagger \text{ } pre_R P \wedge$
 $\text{'[}\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg\text{' } \dagger \text{ } post_R P' \longrightarrow$
 $t \leq t_1 @ t_2 \longrightarrow \neg \text{'[}\$st \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll (t_1 @ t_2) - t \gg\text{' } \dagger (\neg_r pre_R Q)'$

from $a1$ **have** $preP: \text{'[}\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_1 @ t_2 \gg\text{' } \dagger (pre_R P)'$

by (*simp add: taut-not unrest-all-circus-vars-st assms closure unrest, rel-auto*)

have $\text{'[}\$st \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \ll t_1 \gg, \$tr' \mapsto_s \ll t_1 \gg + \ll t_2 \gg\text{' } \dagger \text{ } post_R Q'$

proof –

have $\text{'[}\$st \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_2 \gg\text{' } \dagger \text{ } post_R Q =$

$\text{'[}\$st \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg\text{' } \dagger \text{'[}\$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_2 \gg\text{' } \dagger \text{ } post_R Q$

by *rel-auto*

also have $\dots = \text{'[}\$st \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg\text{' } \dagger \text{'[}\$tr \mapsto_s \ll t_1 \gg, \$tr' \mapsto_s \ll t_1 \gg + \ll t_2 \gg\text{' } \dagger \text{ } post_R Q$

by (*simp add: R2-subst-tr assms closure, rel-auto*)

finally show $?thesis$ **using** $a5$

by (*rel-auto*)

qed

with $a3$

have $postPQ: \text{'[}\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_1 @ t_2 \gg\text{' } \dagger (post_R P ;; post_R Q)'$

by (*rel-auto, meson Prefix-Order.prefixI*)

have $\text{'[}\$st \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \ll t_1 \gg, \$tr' \mapsto_s \ll t_1 \gg + \ll t_2 \gg\text{' } \dagger \text{ } pre_R Q'$

proof –

have $\text{'[}\$st \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \ll t_1 \gg, \$tr' \mapsto_s \ll t_1 \gg + \ll t_2 \gg\text{' } \dagger \text{ } pre_R Q =$

$\text{'[}\$st \mapsto_s \ll s_0 \gg\text{' } \dagger \text{'[}\$tr \mapsto_s \ll t_1 \gg, \$tr' \mapsto_s \ll t_1 \gg + \ll t_2 \gg\text{' } \dagger \text{ } pre_R Q$

```

    by rel-auto
  also have ... = [ $\$st \mapsto_s \ll s_0 \gg$ ]  $\dagger$  [ $\$tr \mapsto_s 0, \$tr' \mapsto_s \ll t_2 \gg$ ]  $\dagger$   $pre_R Q$ 
    by (simp add: R2-subst-tr assms closure)
  finally show ?thesis using a4
    by (rel-auto)
qed

from a6
have a6':  $\bigwedge t s_1. \llbracket t \leq t_1 @ t_2; '[\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg] \dagger pre_R P'; '[\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg] \dagger post_R P' \rrbracket \implies$ 
 $'[\$st \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll (t_1 @ t_2) - t \gg] \dagger pre_R Q'$ 
  apply (subst (asm) taut-not)
  apply (simp add: unrest-all-circus-vars-st assms closure unrest)
  apply (rel-auto)
  done

have wpR:  $'[\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_1 @ t_2 \gg] \dagger (post_R P wp_r pre_R Q)'$ 
proof -
  have  $\bigwedge s_1 t_0. \llbracket t_0 \leq t_1 @ t_2; '[\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_0 \gg] \dagger post_R P' \rrbracket$ 
 $\implies '[\$st \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll (t_1 @ t_2) - t_0 \gg] \dagger pre_R Q'$ 
  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  $'([\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_1 @ t_2 \gg] \dagger pre_R P \wedge$ 
 $[\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_1 @ t_2 \gg] \dagger (post_R P wp_r pre_R Q)) \wedge$ 
 $[\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_1 @ t_2 \gg] \dagger (post_R P ;; post_R Q)'$ 
  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* is NCSP
shows $tr\llbracket\langle a \rangle\rrbracket \rightarrow_C P\llbracket s \rrbracket = \{(a \# t, s') \mid t \ s'. (t, s') \in tr\llbracket P \rrbracket s\}$
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

10.3 Deadlock Freedom

The following is a specification for deadlock free actions. In any intermediate observation, there must be at least one enabled event.

definition *CDF* :: (s, e) action **where**
[*rdes-def*]: $CDF = \mathbf{R}_s(true_r \vdash (\bigwedge (s, t, E, e) \cdot \mathcal{E}(\langle s \rangle, \langle t \rangle, \langle insert\ e\ E \rangle)) \diamond true_r)$

lemma *CDF-NCSP* [*closure*]: *CDF* is NCSP
apply (*simp add: CDF-def*)
apply (*rule NCSP-rdes-intro*)
apply (*simp-all add: closure unrest*)
apply (*rel-auto*)
done

lemma *Skip-deadlock-free*: $CDF \sqsubseteq Skip$
by (*rdes-refine*)

end

11 Meta-theory for Stateful-Failure Reactive Designs

theory *utp-sf-rdes*
imports
 utp-sfrd-core
 utp-sfrd-rel
 utp-sfrd-healths
 utp-sfrd-contracts
 utp-sfrd-extchoice
 utp-sfrd-prog
 utp-sfrd-recursion
 utp-sfrd-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.