

Stateful-Failure Reactive Designs in Isabelle/UTP

Simon Foster James Baxter Ana Cavalcanti Jim Woodcock

April 18, 2019

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	3
3	Stateful-Failure Reactive Relations	5
3.1	Healthiness Conditions	5
3.2	Closure Properties	7
3.3	Introduction laws	13
3.4	UTP Theory	14
3.5	Weakest Precondition	15
3.6	Trace Substitution	16
3.7	Initial Interaction	17
3.8	Enabled Events	18
3.9	Completed Trace Interaction	20
3.10	Downward closure of refusals	24
3.11	Renaming	26
4	Stateful-Failure Healthiness Conditions	28
5	Definitions	28
5.1	Healthiness condition properties	29
5.2	CSP theories	40
5.3	Algebraic laws	41

6	Stateful-Failure Reactive Contracts	41
7	External Choice	43
7.1	Definitions and syntax	43
7.2	Basic laws	44
7.3	Algebraic laws	44
7.4	Reactive design calculations	44
7.5	Productivity and Guardedness	52
7.6	Algebraic laws	53
8	Stateful-Failure Programs	56
8.1	Conditionals	57
8.2	Guarded commands	57
8.3	Assumptions	57
8.4	Alternation	57
8.5	While Loops	58
8.6	Iteration Construction	58
8.7	Assignment	61
8.8	Assignment with update	62
8.9	State abstraction	63
8.10	Guards	63
8.11	Basic events	67
8.12	Event prefix	69
8.13	Guarded external choice	70
8.14	Input prefix	71
8.15	Renaming	72
8.16	Algebraic laws	73
9	Recursion in Stateful-Failures	74
9.1	Fixed-points	75
9.2	Example action expansion	76
10	Linking to the Failures-Divergences Model	76
10.1	Failures-Divergences Semantics	76
10.2	Circus Operators	78
10.3	Deadlock Freedom	84
11	Meta-theory for Stateful-Failure Reactive Designs	84

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 $(\sigma, \varphi) \text{ sfrd-vars} = (\varphi \text{ list}, \sigma) \text{ rsp-vars} +$
 $\text{ref} :: \varphi \text{ set}$

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.

type-synonym $(\sigma, \varphi) \text{ sfrd} = (\sigma, \varphi) \text{ sfrd-vars}$
type-synonym $(\sigma, \varphi) \text{ action} = (\sigma, \varphi) \text{ sfrd hrel}$
type-synonym $\varphi \text{ csp} = (\text{unit}, \varphi) \text{ sfrd}$
type-synonym $\varphi \text{ process} = \varphi \text{ csp hrel}$

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

$(\text{type}) (\sigma, \varphi) \text{ sfrd} \leq (\text{type}) (\sigma, \varphi) \text{ sfrd-vars}$
 $(\text{type}) (\sigma, \varphi) \text{ action} \leq (\text{type}) (\sigma, \varphi) \text{ sfrd hrel}$
 $(\text{type}) \varphi \text{ process} \leq (\text{type}) (\text{unit}, \varphi) \text{ action}$

notation $\text{sfrd-vars.more}_L (\Sigma_C)$

declare $\text{des-vars.splits} [\text{alpha-splits del}]$
declare $\text{rp-vars.splits} [\text{alpha-splits del}]$
declare $\text{des-vars.splits} [\text{alpha-splits del}]$
declare $\text{rsp-vars.splits} [\text{alpha-splits del}]$
declare $\text{rsp-vars.splits} [\text{alpha-splits}]$
declare $\text{rp-vars.splits} [\text{alpha-splits}]$
declare $\text{des-vars.splits} [\text{alpha-splits}]$

2.2 Basic laws

lemma $R2c\text{-tr-ext}: R2c (\$tr' =_u \$tr \hat{~}_u \langle [a]_{S<} \rangle) = (\$tr' =_u \$tr \hat{~}_u \langle [a]_{S<} \rangle)$
by (*rel-auto*)

lemma *circus-alpha-bij-lens*:

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

2.3 Unrestriction laws

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

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

lemma *post-unrest-ref* [*unrest*]: $\$ref \# P \implies \$ref \# \text{post}_R(P)$
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*]: $\$ref' \# [b]_{S<}$
 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 $\Sigma \# [\$ref' \mapsto_s r', \$st \mapsto_s s, \$st' \mapsto_s s', \$tr \mapsto_s t, \$tr' \mapsto_s t'] \uparrow P$
 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-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 *Continuous-CRR* [closure]: *Continuous CRR*

by (rel-blast)

lemma *CRR-intro*:

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

shows P is *CRR*

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

lemma *CRR-form*: $CRR(P) = (\exists \{\$ok, \$ok', \$wait, \$wait', \$ref\} \cdot (\exists tt_0 \cdot P[\langle \rangle / \$tr][\langle \langle tt_0 \rangle \rangle / \$tr'] \wedge \$tr' =_u \$tr \hat{^}_u \langle \langle tt_0 \rangle \rangle))$

by (rel-auto; fastforce)

lemma *CRR-seqr-form*:

$CRR(P) ;; CRR(Q) =$

$(\exists tt_1 \cdot \exists tt_2 \cdot ((\exists \{\$ok, \$ok', \$wait, \$wait', \$ref\} \cdot P)[\langle \rangle / \$tr][\langle \langle tt_1 \rangle \rangle / \$tr'] ;;$

$(\exists \{\$ok, \$ok', \$wait, \$wait', \$ref\} \cdot Q)[\langle \rangle / \$tr][\langle \langle tt_2 \rangle \rangle / \$tr'] \wedge \$tr' =_u \$tr \hat{^}_u \langle \langle tt_1 \rangle \rangle \hat{^}_u \langle \langle tt_2 \rangle \rangle))$

by (simp add: CRR-form, rel-auto; fastforce)

CSP Reactive Finalisers

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

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

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

lemma $Continuous-CRF [closure]: Continuous CRF$
by $(rel-blast)$

lemma $CRF-intro:$
assumes $\$ref \# P \ \$ref' \# P \text{ is } RR$
shows $P \text{ is } CRF$
by $(simp \text{ add: } CRF-def \ CRR-def \ Healthy-def, \ simp \text{ add: } Healthy-if \ assms \ ex-unrest)$

lemma $CRF-implies-CRR [closure]:$
assumes $P \text{ is } CRF$ **shows** $P \text{ is } CRR$

proof –
have $CRR(CRF(P)) = CRF(P)$
by $(rel-auto)$
thus $?thesis$
by $(metis \ Healthy-def \ assms)$
qed

definition $crel-skip :: ('s, 'e) \text{ action} (II_c)$ **where**
 $[upred-defs]: crel-skip = (\$tr' =_u \$tr \wedge \$st' =_u \$st)$

lemma $crel-skip-CRR [closure]: II_c \text{ is } CRR$
by $(rel-auto)$

lemma $crel-skip-via-rrel: II_c = CRR(II_r)$
by $(rel-auto)$

lemma $crel-skip-left-unit [rpred]:$
assumes $P \text{ is } CRR$
shows $II_c ;; P = P$
proof –
have $II_c ;; CRR(P) = CRR(P)$ **by** $(rel-auto)$
thus $?thesis$ **by** $(simp \text{ add: } Healthy-if \ assms)$
qed

lemma $crel-skip-right-unit [rpred]:$
assumes $P \text{ is } CRF$
shows $P ;; II_c = P$
proof –
have $CRF(P) ;; II_c = CRF(P)$ **by** $(rel-auto)$
thus $?thesis$ **by** $(simp \text{ add: } Healthy-if \ assms)$
qed

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-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 *CRC-intro'*:
 assumes P is *CRR* P is *RC*
 shows P is *CRC*
 by (metis *CRC-def CRR-def Healthy-def RC-implies-RR assms*)

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

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

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

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

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

lemma *ex-ref'-RR-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-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-CRC* [*closure*]: *Idempotent CRC*
 by (simp add: *CRC-idem Idempotent-def*)

3.2 Closure Properties

lemma *CRR-implies-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-intro'':
  assumes  $P$  is CRR  $P$  is RC1
  shows  $P$  is CRC
  by (simp add: CRC-intro' CRR-implies-RR RC-intro' assms)

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 CRF-unrest-ref' [unrest]:
  assumes  $P$  is CRF
  shows  $\$ref' \# P$ 
proof -
  have  $\$ref' \# CRF(P)$  by (simp add: CRF-def unrest)
  thus ?thesis by (simp add: assms Healthy-if)
qed

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<} is CRR*
by (*rel-auto*)

lemma *st-post-unrest-ref'* [unrest]: *\$ref' # [b]_{S>}*
by (*rel-auto*)

lemma *st-post-CRR* [closure]: *[b]_{S>} is CRR*
by (*rel-auto*)

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

lemma *rea-rename-CRR-closed* [closure]:
assumes *P is CRR*
shows *P($\lfloor f \rfloor_r$) is CRR*
proof –
have *\$ref # (CRR P)($\lfloor f \rfloor_r$)*
by (*rel-auto*)
thus *?thesis*
by (*rule-tac CRR-intro, simp-all add: closure Healthy-if assms*)
qed

lemma *st-subst-CRR-closed* [closure]:
assumes *P is CRR*
shows *($\sigma \uparrow_S P$) is CRR*
by (*rule CRR-intro, simp-all add: unrest closure assms*)

lemma *st-subst-CRC-closed* [closure]:
assumes *P is CRC*
shows *($\sigma \uparrow_S P$) is CRC*
by (*rule CRC-intro, simp-all add: closure assms unrest*)

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 *st-cond-left-impl-CRC-closed* [closure]:

$P \text{ is CRC} \implies ([b]_{S<} \Rightarrow_r P) \text{ is CRC}$

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

lemma *unrest-ref-map-st* [unrest]: $\$ref \# P \implies \$ref \# P \oplus_r \text{map-st}_L[a]$

by (rel-auto)

lemma *unrest-ref'-map-st* [unrest]: $\$ref' \# P \implies \$ref' \# P \oplus_r \text{map-st}_L[a]$

by (rel-auto)

lemma *unrest-ref-rdes-frame-ext* [unrest]:

$\$ref \# P \implies \$ref \# a:[P]_r^+$

by (rel-blast)

lemma *unrest-ref'-rdes-frame-ext* [unrest]:

$\$ref' \# P \implies \$ref' \# a:[P]_r^+$

by (rel-blast)

lemma *map-st-ext-CRR-closed* [closure]:

assumes $P \text{ is CRR}$

shows $P \oplus_r \text{map-st}_L[a] \text{ is CRR}$

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

lemma *map-st-ext-CRC-closed* [closure]:

assumes $P \text{ is CRC}$

shows $P \oplus_r \text{map-st}_L[a] \text{ is CRC}$

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

lemma *rdes-frame-ext-CRR-closed* [closure]:

assumes $P \text{ is CRR}$

shows $a:[P]_r^+ \text{ is CRR}$

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

lemma *USUP-CRC-closed* [closure]: $\llbracket A \neq \{\}; \bigwedge i. i \in A \implies P i \text{ is CRC} \rrbracket \implies (\bigsqcup i \in A \cdot P i) \text{ is CRC}$

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

lemma *UINF-CRR-closed* [closure]: $\llbracket \bigwedge i. i \in A \implies P i \text{ is CRR} \rrbracket \implies (\bigsqcap i \in A \cdot P i) \text{ is CRR}$

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

lemma *cond-CRC-closed* [closure]:

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

shows $P \triangleleft b \triangleright_R Q \text{ is CRC}$

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

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 $(\bigsqcup i. P(i))\ \text{is}\ CRR$
 by (rule *CRR-intro*, simp-all add: *assms unrest closure*)

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

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

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

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

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

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

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

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

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

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

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

assumes P is CRF Q is CRF
shows $(P \parallel Q)$ is CRF
by (rule CRF-intro, simp-all add: unrest assms closure)

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

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

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

lemma *USUP-ind-CRC-closed* [closure]:
 $\llbracket \bigwedge i. P \text{ i is CRC} \rrbracket \implies (\bigsqcup i. P \text{ i}) \text{ is CRC}$
by (metis CRC-implies-CRR CRC-implies-RC USUP-ind-CRR-closed USUP-ind-RC-closed false-CRC rea-not-CRR-closed wp-rea-CRC wp-rea-RC-false)

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

lemma *tr-assign-comp* [rdes]:
fixes $P :: ('s, 'e) \text{ action}$
assumes $\$ok \nmid P \ \$wait \nmid P \ \$ref \nmid P$
shows $(\$tr' =_u \$tr \wedge \lceil \langle \sigma \rangle_a \rceil_S) \parallel P = \lceil \sigma \rceil_{S\sigma} \dagger P$
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 *R4-CRR-closed* [closure]: P is CRR $\implies R4(P)$ is CRR
 by (simp add: R4-def conj-less-tr-RR-closed)

lemma *R5-CRR-closed* [closure]:

assumes P is CRR
 shows $R5(P)$ is CRR

proof –

have $R5(CRR(P))$ is CRR
 by (rel-auto; blast)
 thus ?thesis
 by (simp add: assms Healthy-if)

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

lemma *all-ref-CRC-closed* [closure]:

P is CRC $\implies (\forall \$ref \cdot P)$ is CRC
 by (simp add: CRC-implies-CRR CRR-unrest-ref all-unrest)

lemma *ex-ref-CRR-closed* [closure]:

P is CRR $\implies (\exists \$ref \cdot P)$ is CRR
 by (simp add: CRR-unrest-ref ex-unrest)

lemma *ex-st'-CRR-closed* [closure]:

P is CRR $\implies (\exists \$st' \cdot P)$ is CRR
 by (rule CRR-intro, simp-all add: closure unrest)

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

P is CRR $\implies (\exists \$ref' \cdot P)$ is CRR
 using CRR-implies-RR CRR-intro CRR-unrest-ref ex-ref'-RR-closed out-in-indep unrest-ex-diff by blast

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 $\bigwedge t s s' r'. (CRR P)[\langle \rangle, \langle t \rangle, \langle s \rangle, \langle s' \rangle, \langle r' \rangle / \$tr, \$tr', \$st, \$st', \$ref']$
 $\sqsubseteq (CRR Q)[\langle \rangle, \langle t \rangle, \langle s \rangle, \langle s' \rangle, \langle r' \rangle / \$tr, \$tr', \$st, \$st', \$ref']$
 using assms by (simp add: 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 $\bigwedge t s s' r'. (CRR P)[\langle \rangle, \langle t \rangle, \langle s \rangle, \langle s' \rangle, \langle r' \rangle / \$tr, \$tr', \$st, \$st', \$ref']$
 $= (CRR Q)[\langle \rangle, \langle t \rangle, \langle s \rangle, \langle s' \rangle, \langle r' \rangle / \$tr, \$tr', \$st, \$st', \$ref']$
 using assms by (simp add: 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 UTP Theory

interpretation crf-theory: utp-theory-kleene CRF II_c
 rewrites $P \in \text{carrier crf-theory.thy-order} \longleftrightarrow P \text{ is CRF}$
 and le rrel-theory.thy-order = (\sqsubseteq)
 and eq rrel-theory.thy-order = (=)
 and crf-top: crf-theory.utp-top = false
 and crf-bottom: crf-theory.utp-bottom = true_r

proof –
 interpret utp-theory-continuous CRF
 by (unfold-locales, simp-all add: add: CRF-idem Continuous-CRF)
 show top:utp-top = false
 by (simp add: healthy-top, rel-auto)
 show bot:utp-bottom = true_r
 by (simp add: healthy-bottom, rel-auto)
 show utp-theory-kleene CRF II_c
 by (unfold-locales, simp-all add: closure rpred top)
 qed (simp-all)

abbreviation crf-star :: $- \Rightarrow -$ ($-^{*c}$ [999] 999) where
 $P^{*c} \equiv \text{crf-theory.utp-star } P$

lemma crf-star-as-rea-star:

$P \text{ is CRF} \implies P^{*c} = P^{*r} ;; II_c$
 by (simp add: crf-theory.Star-alt-def rrel-theory.Star-alt-def closure rpred unrest)

lemma crf-star-inductl: $R \text{ is CRR} \implies Q \sqsubseteq (P ;; Q) \sqcap R \implies Q \sqsubseteq P^{*c} ;; R$

by (simp add: crel-skip-left-unit crf-theory.utp-star-def upred-semiring.mult-assoc urel-ka.star-inductl)

3.5 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: segr-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 $\dots = (\forall (s_0, t_0) \cdot \ll t_0 \gg \leq_u \$tr' \wedge P[\ll s_0 \gg, \ll t_0 \gg / \$st', \$tr']) \Rightarrow_r (RR Q)[\ll s_0 \gg, \ll t_0 \gg / \$st, \$tr])$
by (*rel-auto*)
also have $\dots = (\forall (s_0, t_0) \cdot \ll t_0 \gg \leq_u \$tr' \wedge P[\ll s_0 \gg, \ll t_0 \gg / \$st', \$tr']) \Rightarrow_r Q[\ll s_0 \gg, \ll t_0 \gg / \$st, \$tr])$

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

lemma wp-rea-circus-form-alt:

assumes P is CRR $\$ref' \# P$ Q is CRC

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

proof –

have $(P \text{ wp}_r Q) = R2(P \text{ wp}_r Q)$

by (simp add: CRC-implies-RR CRR-implies-RR Healthy-if RR-implies-R2 assms wp-rea-R2-closed)

also have $\dots = R2(\forall (s_0, tr_0) \cdot \ll tr_0 \gg \leq_u \$tr' \wedge (RR P)[\ll s_0 \gg, \ll tr_0 \gg / \$st', \$tr']) \Rightarrow_r (RR Q)[\ll s_0 \gg, \ll tr_0 \gg / \$st, \$tr])$

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

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

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

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

by (rel-auto)

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

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

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

by (rel-auto, blast+)

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

by (simp add: Healthy-if assms closure)

finally show ?thesis .

qed

lemma wp-rea-circus-form-alt:

assumes P is CRR $\$ref' \# P$ Q is CRC

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

oops

3.6 Trace Substitution

definition trace-subst $(\llbracket - \rrbracket_t [999, 0] 999)$

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

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

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

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

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

assumes P is RR

shows $P[\llbracket v \rrbracket_t]$ is RR

proof –

have $(RR P)[\llbracket v \rrbracket_t]$ is RR

apply (rel-auto)

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

apply (*metis le-add minus-cancel-le trace-class.add-diff-cancel-left*)
using *le-add order-trans* **apply** *blast*
done
thus *?thesis*
by (*simp add: Healthy-if assms*)
qed

lemma *trace-subst-CRR-closed* [*closure*]:
assumes *P is CRR*
shows $P\llbracket v \rrbracket_t$ *is CRR*
by (*rule CRR-intro, simp-all add: closure assms unrest*)

lemma *tsubst-nil* [*usubst*]:
assumes *P is CRR*
shows $P\llbracket \langle \rangle \rrbracket_t = P$
proof –
have $(CRR\ P)\llbracket \langle \rangle \rrbracket_t = CRR\ P$
by (*rel-auto*)
thus *?thesis*
by (*simp add: Healthy-if assms*)
qed

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

lemma *cond-rea-tt-subst* [*usubst*]:
 $(P \triangleleft b \triangleright_R Q)\llbracket v \rrbracket_t = (P\llbracket v \rrbracket_t \triangleleft b \triangleright_R Q\llbracket v \rrbracket_t)$
by (*rel-auto*)

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

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

lemma *rea-subst-R1-closed* [*closure*]: $P\llbracket v \rrbracket_t$ *is R1*
apply (*rel-auto*) **using** *le-add order.trans* **by** *blast*

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

3.7 Initial Interaction

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

lemma *usubst-rea-init'* [*usubst*]:
 $\sigma \dagger_S \mathcal{I}(s, t) = \mathcal{I}(\sigma \dagger_S s, \sigma \dagger_S t)$
by (*rel-auto*)

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

lemma *rea-init-R1* [*closure*]: $\mathcal{I}(s, t)$ *is R1*
by (*rel-auto*)

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

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

lemma *rea-init-RC* [closure]: $\mathcal{I}(s, t)$ is *CRC*
apply (*rel-auto*) **by** *fastforce*

lemma *rea-init-false* [*rpred*]: $\mathcal{I}(\text{false}, t) = \text{true}_r$
by (*rel-auto*)

lemma *rea-init-nil* [*rpred*]: $\mathcal{I}(s, \langle \rangle) = [\neg s]_{S<}$
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*]:
 $(\mathcal{I}(s_1, t) \wedge \mathcal{I}(s_2, t)) = \mathcal{I}(s_1 \vee s_2, t)$
by (*rel-auto*)

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

3.8 Enabled Events

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

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

lemma *st-unrest-csp-enable* [*unrest*]: $\llbracket \&\mathbf{v} \# s; \&\mathbf{v} \# t; \&\mathbf{v} \# E \rrbracket \implies \$st \# \mathcal{E}(s, t, E)$
by (*simp add: csp-enable-def 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*: $\mathcal{E}(true, \langle \rangle, \{v\}_u) = (\$tr' =_u \$tr \wedge [v]_{S<} \notin_u \$ref')$
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*]:
 $\mathcal{E}(s(x), t(x), E(x)) \llbracket x \rightarrow [v]_{S\leftarrow} \rrbracket = \mathcal{E}(s(x) \llbracket x \rightarrow v \rrbracket, t(x) \llbracket x \rightarrow v \rrbracket, E(x) \llbracket x \rightarrow v \rrbracket)$
by (*rel-auto*)

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

$R4(\mathcal{E}(s, \langle \rangle, E)) = \text{false}$

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]:

$R5(\mathcal{E}(s, \text{bop Cons } x \text{ } xs, E)) = \text{false}$

by (rel-auto)

lemma *rel-aext-csp-enable* [alpha]:

$\text{vwb-lens } a \implies \mathcal{E}(s, t, E) \oplus_r \text{map-st}_L[a] = \mathcal{E}(s \oplus_p a, t \oplus_p a, E \oplus_p a)$

by (rel-auto)

3.9 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]: $\Phi(s, \sigma, t) = ([s]_{S<} \wedge \$tr' =_u \$tr \hat{^}_u [t]_{S<} \wedge [\langle \sigma \rangle_a]_S)$

lemma *csp-do-eq-intro*:

assumes $s_1 = s_2 \ \sigma_1 = \sigma_2 \ t_1 = t_2$

shows $\Phi(s_1, \sigma_1, t_1) = \Phi(s_2, \sigma_2, t_2)$

by (simp add: assms)

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]:

$\llbracket x \bowtie (\$tr)_v; x \bowtie (\$tr')_v; x \bowtie (\$st)_v; x \bowtie (\$st')_v \rrbracket \implies x \# \Phi(s, \sigma, t)$

by (simp-all add: csp-do-def alpha-in-var alpha-out-var prod-as-plus unrest lens-indep-sym)

lemma *csp-do-CRF* [closure]: $\Phi(s, \sigma, t)$ is CRF

by (rel-auto)

lemma *csp-do-R4-closed* [closure]:

$\Phi(b, \sigma, \text{bop Cons } x \text{ } xs)$ 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*]:

$[\sigma]_{S\sigma} \uparrow \Phi(s, \varrho, t) = \Phi(\sigma \uparrow s, \varrho \circ \sigma, \sigma \uparrow t)$
by (*rel-auto*)

lemma *csp-do-nothing*: $\Phi(\text{true}, \text{id}, \langle \rangle) = II_c$

by (*rel-auto*)

lemma *csp-do-nothing-0*: $\Phi(\text{true}, \text{id}, 0) = II_c$

by (*rel-auto*)

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

by (*rel-auto*)

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

$[\sigma]_{S\sigma} \uparrow \mathcal{E}(s, t_2, e) = \mathcal{E}(\sigma \uparrow s, \sigma \uparrow t_2, \sigma \uparrow e)$
by (*rel-auto*)

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

$\Phi(s_1, \sigma, t_1) ;; \mathcal{E}(s_2, t_2, e) = \mathcal{E}(s_1 \wedge \sigma \uparrow s_2, t_1 \hat{^}_u(\sigma \uparrow t_2), (\sigma \uparrow e))$
by (*rel-auto*)

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

$\Phi(s_1, \sigma, t_1) ;; \Phi(s_2, \varrho, t_2) = \Phi(s_1 \wedge (\sigma \uparrow s_2), \varrho \circ \sigma, t_1 \hat{^}_u(\sigma \uparrow t_2))$
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-comp*:

assumes *P is CRR*

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

proof –

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

by (*rel-auto*; *blast*)

thus *?thesis*

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

qed

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

fixes *P :: ('σ, 'φ) action*

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

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

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

This operator sets an upper bound on the permissible traces, when starting from a particular state

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

$\Phi(s_1, \sigma, t_1) \text{ wp}_r \mathcal{I}(s_2, t_2) = \mathcal{I}(s_1 \wedge \sigma \uparrow s_2, t_1 \hat{^}_u \sigma \uparrow t_2)$

by (*rel-auto*)

lemma *wp-rea-csp-do-false'* [wp]:
 $\Phi(s_1, \sigma, t_1) \text{ wp}_r \text{ false} = \mathcal{I}(s_1, t_1)$
by (*rel-auto*)

lemma *st-pred-impl-csp-do-wp* [rpred]:
 $([s_1]_{S<} \Rightarrow_r \Phi(s_2, \sigma, t) \text{ wp}_r P) = \Phi(s_1 \wedge s_2, \sigma, t) \text{ wp}_r P$
by (*rel-auto*)

lemma *csp-do-seq-USUP-distl* [rpred]:
assumes $\bigwedge i. i \in A \implies P(i) \text{ is CRR } A \neq \{\}$
shows $\Phi(s, \sigma, t) ;; (\bigwedge i \in A \cdot P(i)) = (\bigwedge i \in A \cdot \Phi(s, \sigma, t) ;; P(i))$
proof –
from *assms(2)* **have** $\Phi(s, \sigma, t) ;; (\bigsqcup i \in A \cdot \text{CRR}(P(i))) = (\bigsqcup i \in A \cdot \Phi(s, \sigma, t) ;; \text{CRR}(P(i)))$
by (*rel-blast*)
thus *?thesis*
by (*simp cong: USUP-cong add: assms(1) Healthy-if*)
qed

lemma *csp-do-seq-conj-distl*:
assumes *P is CRR Q is CRR*
shows $\Phi(s, \sigma, t) ;; (P \wedge Q) = (\Phi(s, \sigma, t) ;; P \wedge \Phi(s, \sigma, t) ;; Q)$
proof –
have $\Phi(s, \sigma, t) ;; (\text{CRR}(P) \wedge \text{CRR}(Q)) = ((\Phi(s, \sigma, t) ;; (\text{CRR } P)) \wedge (\Phi(s, \sigma, t) ;; (\text{CRR } Q)))$
by (*rel-blast*)
thus *?thesis*
by (*simp add: assms Healthy-if*)
qed

lemma *csp-do-power-Suc* [rpred]:
 $\Phi(\text{true}, \text{id}, t) \wedge (\text{Suc } i) = \Phi(\text{true}, \text{id}, \text{iter}[\text{Suc } i](t))$
by (*induct i, (rel-auto)+*)

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

lemma *csp-do-id* [rpred]:
 $P \text{ is CRR} \implies \Phi(b, \text{id}, \langle \rangle) ;; P = ([b]_{S<} \wedge P)$
by (*simp add: csp-do-comp usubst*)

lemma *csp-do-id-wp* [wp]:
 $P \text{ is CRR} \implies \Phi(b, \text{id}, \langle \rangle) \text{ wp}_r P = ([b]_{S<} \Rightarrow_r P)$
by (*metis (no-types, lifting) CRR-implies-RR RR-implies-R1 csp-do-id rea-impl-conj rea-impl-false rea-not-CRR-closed rea-not-not wp-rea-def*)

lemma *wp-rea-csp-do-st-pre* [wp]: $\Phi(s_1, \sigma, t_1) \text{ wp}_r [s_2]_{S<} = \mathcal{I}(s_1 \wedge \neg \sigma \dagger s_2, t_1)$
by (*rel-auto*)

lemma *wp-rea-csp-do-skip* [wp]:
fixes $Q :: ('s, 't) \text{ action}$
assumes *P is CRR*

shows $\Phi(s, \sigma, t) \text{ wp}_r P = (\mathcal{I}(s, t) \wedge (\sigma \uparrow_S P) \llbracket t \rrbracket_t)$
apply (*simp add: wp-rea-def*)
apply (*subst csp-do-comp*)
apply (*simp-all add: closure assms usubst*)
oops

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*)

lemma *rea-frame-ext-csp-do* [*frame*]:
 $\text{vwb-lens } a \implies a : [\Phi(s, \sigma, t)]_r^+ = \Phi(s \oplus_p a, \sigma \oplus_s a, t \oplus_p a)$
by (*rel-auto*)

lemma *R5-csp-do-nil* [*rpred*]: $R5(\Phi(s, \sigma, \langle \rangle)) = \Phi(s, \sigma, \langle \rangle)$
by (*rel-auto*)

lemma *R5-csp-do-Cons* [*rpred*]: $R5(\Phi(s, \sigma, x \#_u xs)) = \text{false}$
by (*rel-auto*)

Iterated do relations

fun *titr* :: $\text{nat} \Rightarrow 's \text{ usubst} \Rightarrow ('a \text{ list}, 's) \text{ uexpr} \Rightarrow ('a \text{ list}, 's) \text{ uexpr}$ **where**
titr 0 σ $t = 0$ |
titr (*Suc* n) σ $t = (\text{titr } n \ \sigma \ t) + (\sigma \hat{\ } n) \dagger t$

lemma *titr-as-list-sum*: $\text{titr } n \ \sigma \ t = \text{list-sum } (\text{map } (\lambda i. (\sigma \hat{\ } i) \dagger t) [0..<n])$
apply (*induct n*)
apply (*auto simp add: usubst fold-plus-sum-list-rev foldr-conv-fold*)
done

lemma *titr-as-foldr*: $\text{titr } n \ \sigma \ t = \text{foldr } (\lambda i \ e. (\sigma \hat{\ } i) \dagger t + e) [0..<n] \ 0$
by (*simp add: titr-as-list-sum foldr-map comp-def*)

lemma *list-sum-uexpr-rep-eq*: $\llbracket \text{list-sum } xs \rrbracket_e s = \text{list-sum } (\text{map } (\lambda e. \llbracket e \rrbracket_e s) xs)$
apply (*induct xs*)
apply (*simp-all*)
apply (*pred-simp+*)
done

lemma *titr-rep-eq*: $\llbracket \text{titr } n \ \sigma \ t \rrbracket_e s = \text{foldr } (@) (\text{map } (\lambda x. \llbracket t \rrbracket_e ((\sigma \hat{\ } x) s)) [0..<n]) []$
by (*simp add: titr-as-list-sum list-sum-uexpr-rep-eq comp-def, rel-auto*)

update-uexpr-rep-eq-thms

lemma *funpow-lemma*: $(\lambda x. (f \hat{\ } n) (f x)) = (f \hat{\ } n) \circ f$
by (*simp add: fun-eq-iff funpow-swap1*)

lemma *titr-lemma*:
 $t + (\sigma \dagger \text{titr } n \ \sigma \ t) + (\sigma \hat{\ } n \circ \sigma) \dagger t = (\text{titr } n \ \sigma \ t + (\sigma \hat{\ } n) \dagger t) + (\sigma \circ \sigma \hat{\ } n) \dagger t$
by (*induct n, simp-all add: usubst funpow-lemma add.assoc funpow-swap1*)

lemma *csp-do-power* [*rpred*]:
 $\Phi(s, \sigma, t) \wedge (\text{Suc } n) = \Phi(\bigwedge i \in \{0..n\} \cdot (\sigma \hat{\ } i) \dagger s, \sigma \hat{\ } \text{Suc } n, \text{titr } (\text{Suc } n) \ \sigma \ t)$
apply (*induct n*)
apply (*rel-auto*)

```

apply (simp add: power.power.power-Suc rpred usubst)
apply (thin-tac -)
apply (rule csp-do-eq-intro)
  apply (rel-auto)
  apply (case-tac x=0)
apply (simp-all add: titr-lemma)
apply (metis Suc-le-mono funpow-simps-right(2) gr0-implies-Suc o-def)
apply force
apply (metis Suc-leI funpow-simps-right(2) less-Suc-eq-le o-apply)
apply (metis comp-apply funpow-swap1)
apply (metis add.assoc plus-list-def plus-ueqpr-def titr-lemma)
done

```

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

```

 $\Phi(s, \sigma, t)^{*r} = II_r \sqcap (\bigsqcap n \cdot \Phi(\bigwedge i \in \{0..n\} \cdot (\sigma \hat{=} i) \dagger s, \sigma \hat{=} \text{Suc } n, \text{titr } (\text{Suc } n) \sigma t))$ 
by (simp add: rrel-theory.Star-alt-def closure uplus-power-def rpred)

```

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

```

 $\Phi(s, \sigma, t)^{*c} = (\bigsqcap n \cdot \Phi(\bigsqcup i \in \{0..<n\} \cdot (\sigma \hat{=} i) \dagger s, \sigma \hat{=} n, \text{titr } n \sigma t))$ 
(is ?lhs = ( $\bigsqcap n \cdot ?G(n)$ ))

```

proof –

```

have ?lhs =  $II_c \sqcap (\bigsqcap n \cdot \Phi(\bigwedge i \in \{0..n\} \cdot (\sigma \hat{=} i) \dagger s, \sigma \hat{=} \text{Suc } n, \text{titr } (\text{Suc } n) \sigma t))$ 
  (is - =  $II_c \sqcap (\bigsqcap n \cdot ?F(n))$ )

```

```

by (simp add: crf-theory.Star-alt-def closure uplus-power-def rpred)

```

```

also have ... =  $II_c \sqcap (\bigsqcap n \in \{1..\} \cdot ?F(n - 1))$ 

```

```

by (simp add: UINF-atLeast-Suc)

```

```

also have ... =  $II_c \sqcap (\bigsqcap n \in \{1..\} \cdot \Phi(\bigsqcup i \in \{0..<n\} \cdot (\sigma \hat{=} i) \dagger s, \sigma \hat{=} n, \text{titr } n \sigma t))$ 

```

proof –

```

have ( $\bigsqcap n \in \{1..\} \cdot ?F(n - 1)$ ) = ( $\bigsqcap n \in \{1..\} \cdot ?G(n)$ )

```

```

by (rule UINF-cong, simp, metis Suc-pred atLeastLessThanSuc-atLeastAtMost diff-is-0-eq not0-implies-Suc
not-less-eq-eq zero-less-Suc)

```

```

thus ?thesis by simp

```

qed

```

also have ... =  $?G(0) \sqcap (\bigsqcap n \in \{1..\} \cdot ?G(n))$ 

```

```

by (simp add: usubst csp-do-nothing-0)

```

```

also have ... = ( $\bigsqcap n \in \text{insert } 0 \{1..\} \cdot ?G(n)$ )

```

```

by (simp)

```

```

also have ... = ( $\bigsqcap n \cdot ?G(n)$ )

```

proof –

```

have  $\text{insert } (0::\text{nat}) \{1..\} = \{0..\}$  by auto

```

```

thus ?thesis

```

```

by simp

```

qed

```

finally show ?thesis .

```

qed

3.10 Downward closure of refusals

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

definition *CDC* :: (*'s, 'e*) *action* \Rightarrow (*'s, 'e*) *action* **where**

[upred-defs]: $CDC(P) = (\exists \text{ref}_0 \cdot P \llbracket \llbracket \text{ref}_0 \rrbracket / \$\text{ref}' \rrbracket \wedge \$\text{ref}' \subseteq_u \llbracket \text{ref}_0 \rrbracket)$

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

```

by (rel-auto)

```


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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma *st-subst-CDC-closed* [closure]:
assumes P is CDC
shows $(\sigma \uparrow_S P)$ is CDC
proof –
have $(\sigma \uparrow_S CDC P)$ is CDC
by (*rel-auto*)
thus ?thesis
by (*simp add: assms Healthy-if*)
qed

lemma *rea-st-cond-CDC* [closure]: $[g]_{S<}$ is CDC
by (*rel-auto*)

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

lemma *state-srea-CDC-closed* [closure]:
assumes P is CDC
shows *state* $'a \cdot P$ is CDC
proof –
have *state* $'a \cdot CDC(P)$ is CDC
by (*rel-blast*)
thus ?thesis
by (*simp add: Healthy-if assms*)
qed

3.11 Renaming

abbreviation *pre-image* $f B \equiv \{x. f(x) \in B\}$

definition *csp-rename* :: (s, e) action $\Rightarrow (e \Rightarrow f) \Rightarrow (s, f)$ action $((-)\llbracket - \rrbracket_c [999, 0] 999)$ **where**
 $[upred-defs]: P \llbracket f \rrbracket_c = R2((\$tr' =_u \langle \rangle \wedge \$st' =_u \$st) ;; P ;; (\$tr' =_u \text{map}_u \llbracket f \rrbracket \$tr \wedge \$st' =_u \$st \wedge$

$uop \ (pre\text{-}image \ f) \ \$ref' \subseteq_u \ \$ref))$

lemma *csp-rename-CRR-closed* [closure]:

assumes P is CRR
shows $P(f)_c$ is CRR

proof –

have $(CRR \ P)(f)_c$ is CRR
by (*rel-auto*)
thus ?thesis **by** (*simp add: assms Healthy-if*)

qed

lemma *csp-rename-disj* [rpred]: $(P \vee Q)(f)_c = (P(f)_c \vee Q(f)_c)$

by (*rel-blast*)

lemma *csp-rename-UINF-ind* [rpred]: $(\bigcap i \cdot P \ i)(f)_c = (\bigcap i \cdot (P \ i)(f)_c)$

by (*rel-blast*)

lemma *csp-rename-UINF-mem* [rpred]: $(\bigcap i \in A \cdot P \ i)(f)_c = (\bigcap i \in A \cdot (P \ i)(f)_c)$

by (*rel-blast*)

Renaming distributes through conjunction only when both sides are downward closed

lemma *csp-rename-conj* [rpred]:

assumes $inj \ f$ P is CRR Q is CRR P is CDC Q is CDC
shows $(P \wedge Q)(f)_c = (P(f)_c \wedge Q(f)_c)$

proof –

from *assms*(1) **have** $((CDC \ (CRR \ P)) \wedge (CDC \ (CRR \ Q)))(f)_c = ((CDC \ (CRR \ P))(f)_c \wedge (CDC \ (CRR \ Q))(f)_c)$

apply (*rel-auto*)

apply *blast*

apply *blast*

apply (*meson order-refl order-trans*)

done

thus ?thesis

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

qed

lemma *csp-rename-seq* [rpred]:

assumes P is CRR Q is CRR

shows $(P ;; Q)(f)_c = P(f)_c ;; Q(f)_c$

oops

lemma *csp-rename-R4* [rpred]:

$(R4(P))(f)_c = R4(P(f)_c)$

apply (*rel-auto, blast*)

using *less-le* **apply** *fastforce*

apply (*metis (mono-tags, lifting) Prefix-Order.Nil-prefix append-Nil2 diff-add-cancel-left' less-le list.simps(8) plus-list-def*)

done

lemma *csp-rename-R5* [rpred]:

$(R5(P))(f)_c = R5(P(f)_c)$

apply (*rel-auto, blast*)

using *less-le* **apply** *fastforce*

done

lemma *csp-rename-do* [*rpred*]: $\Phi(s, \sigma, t) \llbracket f \rrbracket_c = \Phi(s, \sigma, \text{map}_u \llbracket f \rrbracket \gg t)$
by (*rel-auto*)

lemma *csp-rename-enable* [*rpred*]: $\mathcal{E}(s, t, E) \llbracket f \rrbracket_c = \mathcal{E}(s, \text{map}_u \llbracket f \rrbracket \gg t, \text{uop}(\text{image } f) E)$
by (*rel-auto*)

lemma *st'-unrest-csp-rename* [*unrest*]: $\$st' \# P \implies \$st' \# P \llbracket f \rrbracket_c$
by (*rel-blast*)

lemma *ref'-unrest-csp-rename* [*unrest*]: $\$ref' \# P \implies \$ref' \# P \llbracket f \rrbracket_c$
by (*rel-blast*)

lemma *csp-rename-CDC-closed* [*closure*]:
 $P \text{ is } CDC \implies P \llbracket f \rrbracket_c \text{ is } CDC$
by (*rel-blast*)

lemma *csp-do-CDC* [*closure*]: $\Phi(s, \sigma, t) \text{ is } CDC$
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* :: $((\sigma', \varphi) \text{ sfrd} \times (\sigma', \varphi) \text{ sfrd}) \text{ health}$
where $CSP1(P) \equiv RD1(P)$

abbreviation *CSP2* :: $((\sigma', \varphi) \text{ sfrd} \times (\sigma', \varphi) \text{ sfrd}) \text{ health}$
where $CSP2(P) \equiv RD2(P)$

abbreviation *CSP* :: $((\sigma', \varphi) \text{ sfrd} \times (\sigma', \varphi) \text{ sfrd}) \text{ health}$
where $CSP(P) \equiv SRD(P)$

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

definition *SKIP* :: $\varphi \text{ process}$ **where**
[*upred-defs*]: $SKIP = \mathbf{R}_s(\exists \$ref \cdot CSP1(II))$

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

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

definition *CSP3* :: $((\sigma', \varphi) \text{ sfrd} \times (\sigma', \varphi) \text{ sfrd}) \text{ health}$ **where**
[*upred-defs*]: $CSP3(P) = (Skip ;; P)$

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

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

Productive and normal processes

abbreviation $PCSP \equiv Productive \circ NCSP$

Instantaneous and normal processes

abbreviation $ICSP \equiv ISRD1 \circ NCSP$

5.1 Healthiness condition properties

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

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

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

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

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

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

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

lemma $Skip\text{-frame}$ $[frame]$: $vwb\text{-lens } a \implies a:[Skip]_R^+ = Skip$
by $(rdes\text{-eq})$

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

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

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

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

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

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

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

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

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

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

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

lemma *CSP3-SRD-intro*:
 assumes P is CSP $\$ref \# \text{pre}_R(P) \ \$ref \# \text{peri}_R(P) \ \$ref \# \text{post}_R(P)$
 shows P is CSP3
proof –
 have $P: \mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \diamond \text{post}_R(P)) = P$
 by (simp add: SRD-reactive-design-alt assms(1) wait'-cond-peri-post-cmt[THEN sym])
 have $\mathbf{R}_s(\text{pre}_R(P) \vdash \text{peri}_R(P) \diamond \text{post}_R(P))$ 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 $P \text{ is } CSP3 \longleftrightarrow (\$ref \# P\llbracket false/\$wait \rrbracket)$

proof

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

next

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

qed

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

proof –

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

qed

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

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

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

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

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

lemma *CSP3-Continuous: Continuous CSP3*

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

lemma *Skip-right-lemma:*

assumes *P is CSP*

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

proof –

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

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

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

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

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

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

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

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

finally show *?thesis* .

qed

lemma *Skip-right-tri-lemma:*

assumes *P is CSP*

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

proof –

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

by (*rel-auto*)

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

qed

lemma *CSP4-intro:*

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

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

shows *P is CSP4*

proof –

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

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

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

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

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

by (*simp add: usubst unrest*)

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

by (*simp add: ex-unrest assms*)

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

by (*simp add: cond-var-split*)

also have $\dots = P$

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

finally show *?thesis*

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

qed

lemma *CSP4-RC-intro*:

assumes P is CSP $pre_R(P)$ is RC
 $\$st' \# (cmt_R P) \llbracket true/\$wait' \rrbracket \$ref' \# (cmt_R P) \llbracket false/\$wait' \rrbracket$
shows P is CSP4

proof –

have $(\neg_r pre_R(P)) \;; R1(true) = (\neg_r pre_R(P))$
by (metis (no-types, lifting) R1-seqr-closure assms(2) rea-not-R1 rea-not-false rea-not-not wp-rea-RC-false wp-rea-def)
thus ?thesis
by (simp add: CSP4-intro assms)
qed

lemma *CSP4-rdes*:

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

lemma *CSP4-form*:

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

lemma *Skip-srdes-right-unit*:

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

lemma *Skip-srdes-left-unit*:

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

lemma *CSP4-right-subsumes-RD3*: $RD3(CSP4(P)) = CSP4(P)$

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

lemma *CSP4-NSRD-intro*:

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

lemma *CSP3-commutes-CSP4*: $CSP3(CSP4(P)) = CSP4(CSP3(P))$

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*]:

$P \text{ is NCSP} \implies P \text{ is CSP3}$

by (*metis (no-types, lifting) CSP3-def Healthy-def' NCSP-def Skip-is-CSP Skip-left-unit-ref-unrest Skip-unrest-ref comp-apply segr-assoc*)

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

$P \text{ is NCSP} \implies P \text{ is CSP4}$

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

lemma *NCSP-implies-RD3* [*closure*]: $P \text{ is NCSP} \implies P \text{ is RD3}$

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

lemma *NCSP-implies-NSRD* [*closure*]: $P \text{ is NCSP} \implies P \text{ is NSRD}$

by (*simp add: NCSP-implies-CSP NCSP-implies-RD3 SRD-RD3-implies-NSRD*)

lemma *NCSP-subset-implies-CSP* [*closure*]:

$A \subseteq \llbracket \text{NCSP} \rrbracket_H \implies A \subseteq \llbracket \text{CSP} \rrbracket_H$

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

lemma *NCSP-subset-implies-NSRD* [*closure*]:

$A \subseteq \llbracket \text{NCSP} \rrbracket_H \implies A \subseteq \llbracket \text{NSRD} \rrbracket_H$

using *NCSP-implies-NSRD* **by** *blast*

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

by (*simp add: is-Healthy-subset-member*)

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

by (*simp add: is-Healthy-subset-member*)

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

by (*simp add: is-Healthy-subset-member*)

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

by (*simp add: is-Healthy-subset-member*)

lemma *NCSP-intro*:

assumes $P \text{ is CSP } P \text{ is CSP3 } P \text{ is CSP4}$

shows $P \text{ is NCSP}$

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

lemma *Skip-left-unit*: $P \text{ is NCSP} \implies \text{Skip} ;; P = P$

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

lemma *Skip-right-unit*: $P \text{ is NCSP} \implies P ;; \text{Skip} = P$

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

lemma *NCSP-NSRD-intro*:

assumes $P \text{ is NSRD } \$\text{ref} \# \text{pre}_R(P) \$\text{ref} \# \text{peri}_R(P) \$\text{ref} \# \text{post}_R(P) \$\text{ref}' \# \text{post}_R(P)$

shows $P \text{ is NCSP}$

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

lemma *CSP4-neg-pre-unit*:

assumes *P is CSP P is CSP4*

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

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

lemma *NSRD-CSP4-intro*:

assumes *P 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 = R_s ((\forall \$ref · ($\neg_r \text{pre}_R(P)$) wp_r false) \vdash ((\exists \$ref · \exists \$st' · peri_R(P)) \diamond (\exists \$ref · \exists \$ref' · 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 ... = *R_s ((\forall \$ref · ($\neg_r \text{pre}_R$ (NSRD P)) wp_r false) \vdash ((\exists \$ref · \exists \$st' · peri_R (NSRD P)) \diamond (\exists \$ref · \exists \$ref' · post_R (NSRD P))))*

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

also have ... = *R_s ((\forall \$ref · ($\neg_r \text{pre}_R(P)$) wp_r false) \vdash ((\exists \$ref · \exists \$st' · peri_R(P)) \diamond (\exists \$ref · \exists \$ref' · 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 $\$st' \not\# \text{peri}_R(P)$

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

lemma *CSP4-healthy-form*:

assumes *P is CSP P is CSP4*

shows *P = R_s(($\neg_r \text{pre}_R P$) wp_r false \vdash ((\exists \$st' · peri_R(P)) \diamond (\exists \$ref' · post_R(P))))*

proof –

have *P = R_s (($\neg_r \text{pre}_R P$) wp_r false \vdash ((\exists \$st' · cmt_R P) \triangleleft \$wait' \triangleright (\exists \$ref' · cmt_R P)))*

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

also have ... = *R_s (($\neg_r \text{pre}_R P$) wp_r false \vdash ((\exists \$st' · cmt_R P) $\llbracket \text{true}/\$wait' \rrbracket \triangleleft \$wait' \triangleright$ (\exists \$ref' · cmt_R P) $\llbracket \text{false}/\$wait' \rrbracket$))*

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

also have ... = *R_s(($\neg_r \text{pre}_R P$) wp_r false \vdash ((\exists \$st' · peri_R(P)) \diamond (\exists \$ref' · 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 $\$ref' \not\# \text{pre}_R(P)$

proof –

have *pre_R(P) = pre_R(R_s(($\neg_r \text{pre}_R P$) wp_r false \vdash ((\exists \$st' · peri_R(P)) \diamond (\exists \$ref' · post_R(P))))*

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

also have ... = *($\neg_r \text{pre}_R P$) 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 $\$ref' \# \dots$
 by (simp add: wp-rea-def unrest)
 finally show ?thesis .
 qed

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

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

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

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

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

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

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

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

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

lemma *NCSP-seqr-closure* [closure]:
 assumes *P is NCSP Q is NCSP*
 shows *P ;; 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 *rdes-frame-ext-NCSP-closed* [closure]:
 assumes *vwb-lens a P is NCSP*
 shows *a:[P]_R⁺ is NCSP*
 by (*metis (no-types, lifting) CSP3-def CSP4-def Healthy-intro NCSP-Skip NCSP-implies-NSRD NCSP-intro*
NSRD-is-SRD Skip-frame Skip-left-unit Skip-right-unit assms(1) assms(2) rdes-frame-ext-NSRD-closed
seq-srea-frame)

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 $\text{pre}_R(P)$ is CRR
 by (rule CRR-intro, simp-all add: closure unrest)

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

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

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

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

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

assumes *P is PCSP*
shows *P is NCSP*

proof –

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

also have $\dots = \mathbf{R}_s ((\forall \$ref \cdot (\neg_r \text{pre}_R(\text{NCSP } P)) \text{wp}_r \text{false}) \vdash$
 $(\exists \$ref \cdot \exists \$st' \cdot \text{peri}_R(\text{NCSP } P)) \diamond$
 $((\exists \$ref \cdot \exists \$ref' \cdot \text{post}_R(\text{NCSP } P)) \wedge \$tr <_u \$tr'))$
by (simp add: NCSP-form Productive-RHS-design-form unrest closure)

also have \dots *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* $P (\mathbf{R}_s ((\text{pre}_R X) \vdash \text{peri}_R X \diamond (R_4(\text{post}_R X))))$
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 $P = \text{ISRDI}(\text{NCSP}(\text{NCSP } P))$
by (metis (no-types, hide-lams) Healthy-def' Idempotent-def NCSP-Idempotent assms comp-apply)

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

by (simp add: NCSP-form)

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

by (simp-all add: ISRDI-RHS-design-form closure rdes unrest)

also have \dots *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-ISRDI* [closure]:

assumes P is ICSP
shows P is ISRD
by (*metis* (*no-types*, *hide-lams*) *Healthy-def ICSP-implies-NCSP ISRD-def NCSP-implies-NSRD 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 is ICSP Q is ICSP
shows $P ;; Q$ 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

lemma *NCSP-false*: $NCSP \text{ false} = \text{Miracle}$
by (*simp add: NCSP-def srdes-theory.healthy-top[THEN sym]*, *simp add: closure Healthy-if*)

lemma *NCSP-true*: $NCSP \text{ true} = \text{Chaos}$
by (*simp add: NCSP-def srdes-theory.healthy-bottom[THEN sym]*, *simp add: closure Healthy-if*)

interpretation *csp-theory*: *utp-theory-kleene NCSP Skip*
rewrites $P \in \text{carrier } csp\text{-theory.thy-order} \longleftrightarrow P \text{ is NCSP}$
and $\text{carrier } csp\text{-theory.thy-order} \rightarrow \text{carrier } csp\text{-theory.thy-order} \equiv \llbracket NCSP \rrbracket_H \rightarrow \llbracket NCSP \rrbracket_H$
and $le \text{ } csp\text{-theory.thy-order} = (\sqsubseteq)$
and $eq \text{ } csp\text{-theory.thy-order} = (=)$
and *csp-top*: $csp\text{-theory.utp-top} = \text{Miracle}$
and *csp-bottom*: $csp\text{-theory.utp-bottom} = \text{Chaos}$
proof –
have *utp-theory-continuous NCSP*
by (*unfold-locales*, *simp-all add: Healthy-Idempotent Healthy-if NCSP-Idempotent NCSP-Continuous*)
then interpret *utp-theory-continuous NCSP*

by *simp*
 show $t: \text{utp-top} = \text{Miracle}$ and $b: \text{utp-bottom} = \text{Chaos}$
 by (*simp-all* add: *healthy-top healthy-bottom NCSP-false NCSP-true*)
 show *utp-theory-kleene NCSP Skip*
 by (*unfold-locales, simp-all* add: *closure Skip-left-unit Skip-right-unit Miracle-left-zero t*)
 qed (*simp-all*)

abbreviation *TestC* (*test_C*) **where**
test_C $P \equiv \text{csp-theory.utp-test } P$

definition *StarC* :: (σ, φ) *action* \Rightarrow (σ, φ) *action* (*C [999] 999) **where**
StarC $P \equiv \text{csp-theory.utp-star } P$

lemma *sfrd-star-as-rdes-star*:
 $P \text{ is NCSP} \Rightarrow P^{*R} ;; \text{Skip} = P^{*C}$
 by (*simp* add: *csp-theory.Star-alt-def nsrdes-theory.Star-alt-def StarC-def StarR-def closure unrest*
Skip-srdes-left-unit csp-theory.Unit-Right)

lemma *sfrd-star-as-rdes-star'*:
 $P \text{ is NCSP} \Rightarrow \text{Skip} ;; P^{*R} = P^{*C}$
 by (*simp* add: *csp-theory.Star-alt-def nsrdes-theory.Star-alt-def StarC-def StarR-def closure unrest*
Skip-srdes-right-unit csp-theory.Unit-Left upred-semiring.distrib-left)

theorem *csp-star-rdes-def* [*rdes-def*]:
 assumes $P \text{ is CRC } Q \text{ is CRR } R \text{ is CRF } \$st' \nmid Q$
 shows $(\mathbf{R}_s(P \vdash Q \diamond R))^{*C} = \mathbf{R}_s(R^{*c} \text{ wp}_r P \vdash (R^{*c} ;; Q) \diamond R^{*c})$
 apply (*simp* add: *wp-rea-def sfrd-star-as-rdes-star[THEN sym] crf-star-as-rea-star assms seqr-assoc*
rpred closure unrest StarR-rdes-def)
 apply (*simp* add: *rdes-def assms closure unrest wp-rea-def[THEN sym]*)
 apply (*simp* add: *wp rpred assms closure*)
 apply (*simp* add: *csp-do-nothing*)
 done

5.3 Algebraic laws

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

syntax
 $\text{-ref-var} :: \text{logic}$
 $\text{-mk-CRD} :: \text{uexp} \Rightarrow \text{uexp} \Rightarrow \text{logic} \Rightarrow \text{logic} \ ([\text{-} / \vdash \text{-} / \mid \text{-}]_C)$

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

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

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

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

lemma *periR-mk-CRD* [*rdes*]: $\text{peri}_R([P \vdash Q \text{ trace refs} \mid R(\text{trace})]_C) = ([P]_{S<} \Rightarrow_r ([Q \text{ trace refs}]_{S<} \ll ((\text{trace}, \text{refs}) \rightarrow (\&tt, \$r))))$
 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*]: $\text{post}_R([P \vdash Q \text{ trace refs} \mid R(\text{trace})]_C) = ([P]_{S<} \Rightarrow_r ([R(\text{trace})]_S') \ll (\text{trace} \rightarrow \&tt))$
 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
 $Q \text{ is CSP } \llbracket P_1 \rrbracket_{S<} \Rightarrow \text{pre}_R Q$
 $\llbracket P_1 \rrbracket_{S<} \wedge \text{peri}_R Q \Rightarrow \llbracket P_2 \ t \ r \rrbracket_{S<} \llbracket t \rightarrow \&tt \rrbracket \llbracket r \rightarrow \$ref' \rrbracket$
 $\llbracket P_1 \rrbracket_{S<} \wedge \text{post}_R Q \Rightarrow \llbracket P_3 \ x \rrbracket_S \llbracket x \rightarrow \&tt \rrbracket$
shows $[P_1 \vdash P_2 \text{ trace refs} \mid P_3(\text{trace})]_C \sqsubseteq Q$

proof –

have $[P_1 \vdash P_2 \text{ trace refs} \mid P_3(\text{trace})]_C \sqsubseteq \mathbf{R}_s(\text{pre}_R(Q) \vdash \text{peri}_R(Q) \diamond \text{post}_R(Q))$
using *assms* **by** (*simp add: mk-CRD-def, rule-tac srdes-tri-refine-intro, rel-auto+*)
thus *?thesis*
by (*simp add: SRD-reactive-tri-design assms(1)*)

qed

lemma *CRD-contract-refine'*:

assumes
 $Q \text{ is CSP } \llbracket P_1 \rrbracket_{S<} \Rightarrow \text{pre}_R Q$
 $\llbracket P_2 \ t \ r \rrbracket_{S<} \llbracket t \rightarrow \&tt \rrbracket \llbracket r \rightarrow \$ref' \rrbracket \sqsubseteq (\llbracket P_1 \rrbracket_{S<} \wedge \text{peri}_R Q)$
 $\llbracket P_3 \ x \rrbracket_S \llbracket x \rightarrow \&tt \rrbracket \sqsubseteq (\llbracket P_1 \rrbracket_{S<} \wedge \text{post}_R Q)$
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
 $\llbracket P_1 \rrbracket_{S<} \Rightarrow (\llbracket Q_1 \rrbracket_{S<} :: ('e, 's) \text{ action})$
 $(\llbracket P_2 \ x \ r \rrbracket_{S<} \llbracket x \rightarrow \&tt \rrbracket \llbracket r \rightarrow \$ref' \rrbracket) \sqsubseteq (\llbracket P_1 \rrbracket_{S<} \wedge \llbracket Q_2 \ x \ r \rrbracket_{S<} \llbracket x \rightarrow \&tt \rrbracket \llbracket r \rightarrow \$ref' \rrbracket :: ('e, 's) \text{ action})$

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

lemma *CRD-refine-rdes*:

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

lemma *CRD-refine-rdes'*:

assumes
 $Q_2 \text{ is } RR$
 $Q_3 \text{ is } RR$
 $\lceil P_1 \rceil_{S<} \Rightarrow Q_1$
 $\bigwedge t. ([P_2 \ t \ r]_{S<} \llbracket r \rightarrow \$ref' \rrbracket) \sqsubseteq ([P_1]_{S<} \wedge Q_2 \llbracket \langle \rangle, \ll t \gg / \$tr, \$tr' \rrbracket)$
 $\bigwedge t. [P_3 \ t]_{S'} \sqsubseteq ([P_1]_{S<} \wedge Q_3 \llbracket \langle \rangle, \ll t \gg / \$tr, \$tr' \rrbracket)$
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 sdes-tri-refine-intro*)
show $\lceil P_1 \rceil_{S<} \Rightarrow Q_1$ **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 $\lceil P_1 \rceil_{S<} \wedge RR(Q_2) \Rightarrow [P_2 \ x \ r]_{S<} \llbracket x \rightarrow \&tt \rrbracket \llbracket r \rightarrow \$ref' \rrbracket$
by (*rel-simp; meson*)
thus $\lceil P_1 \rceil_{S<} \wedge Q_2 \Rightarrow [P_2 \ x \ r]_{S<} \llbracket x \rightarrow \&tt \rrbracket \llbracket r \rightarrow \$ref' \rrbracket$
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 $\lceil P_1 \rceil_{S<} \wedge (RR \ Q_3) \Rightarrow [P_3 \ x]_S' \llbracket x \rightarrow \&tt \rrbracket$
by (*rel-simp; meson*)
thus $\lceil P_1 \rceil_{S<} \wedge Q_3 \Rightarrow [P_3 \ x]_S' \llbracket x \rightarrow \&tt \rrbracket$
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* ::

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

syntax

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

translations

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

definition $extChoice ::$

(σ, φ) action $\Rightarrow (\sigma, \varphi)$ action $\Rightarrow (\sigma, \varphi)$ action (**infixl** \square 59) **where**
 $[upred-defs]: P \square Q \equiv ExtChoice\ \{P, Q\}$

Small external choice as an indexed big external choice.

lemma $extChoice-alt-def$:

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

7.2 Basic laws

7.3 Algebraic laws

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

lemma $ExtChoice-single$:

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

7.4 Reactive design calculations

lemma $ExtChoice-rdes$:

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

proof –

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

$(\prod_{i \in A} \cdot R1(R2c(cmt_s \uparrow (P(i) \Rightarrow Q(i))))))$
 by (*metis (no-types, lifting) RHS-design-export-R1 RHS-design-export-R2c*)
 also have ... =
 $\mathbf{R}_s ((\prod_{i \in A} \cdot R1 (R2c (pre_s \uparrow P(i)))) \vdash$
 $R1(R2c$
 $((\prod_{i \in A} \cdot (cmt_s \uparrow (P(i) \Rightarrow Q(i))))$
 $\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
 $(\prod_{i \in A} \cdot (cmt_s \uparrow (P(i) \Rightarrow Q(i))))))$
 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 ... =
 $\mathbf{R}_s ((\prod_{i \in A} \cdot R1 (R2c (pre_s \uparrow P(i)))) \vdash$
 $cmt_s \uparrow$
 $((\prod_{i \in A} \cdot (cmt_s \uparrow (P(i) \Rightarrow Q(i))))$
 $\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
 $(\prod_{i \in A} \cdot (cmt_s \uparrow (P(i) \Rightarrow Q(i))))))$
 by (*metis (no-types, lifting) RHS-design-export-R1 RHS-design-export-R2c rdes-export-cmt*)
 also have ... =
 $\mathbf{R}_s ((\prod_{i \in A} \cdot R1 (R2c (pre_s \uparrow P(i)))) \vdash$
 $cmt_s \uparrow$
 $((\prod_{i \in A} \cdot (P(i) \Rightarrow Q(i))))$
 $\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
 $(\prod_{i \in A} \cdot (P(i) \Rightarrow Q(i))))$
 by (*simp add: usubst*)
 also have ... =
 $\mathbf{R}_s ((\prod_{i \in A} \cdot R1 (R2c (pre_s \uparrow P(i)))) \vdash$
 $((\prod_{i \in A} \cdot (P(i) \Rightarrow Q(i))) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\prod_{i \in A} \cdot (P(i) \Rightarrow Q(i))))$
 by (*simp add: rdes-export-cmt*)
 also have ... =
 $\mathbf{R}_s ((R1(R2c(\prod_{i \in A} \cdot (pre_s \uparrow P(i)))) \vdash$
 $((\prod_{i \in A} \cdot (P(i) \Rightarrow Q(i))) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\prod_{i \in A} \cdot (P(i) \Rightarrow Q(i))))$
 by (*simp add: not-UNIF R1-UNIF R2c-UNIF assms*)
 also have ... =
 $\mathbf{R}_s ((R2c(\prod_{i \in A} \cdot (pre_s \uparrow P(i)))) \vdash$
 $((\prod_{i \in A} \cdot (P(i) \Rightarrow Q(i))) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\prod_{i \in A} \cdot (P(i) \Rightarrow Q(i))))$
 by (*simp add: R1-design-R1-pre*)
 also have ... =
 $\mathbf{R}_s (((\prod_{i \in A} \cdot (pre_s \uparrow P(i)))) \vdash$
 $((\prod_{i \in A} \cdot (P(i) \Rightarrow Q(i))) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\prod_{i \in A} \cdot (P(i) \Rightarrow Q(i))))$
 by (*metis (no-types, lifting) RHS-design-R2c-pre*)
 also have ... =
 $\mathbf{R}_s ([\$ok \mapsto_s true, \$wait \mapsto_s false] \uparrow (\prod_{i \in A} \cdot P(i)) \vdash$
 $((\prod_{i \in A} \cdot (P(i) \Rightarrow Q(i))) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\prod_{i \in A} \cdot (P(i) \Rightarrow Q(i))))$
 proof –
 from *assms* have $\bigwedge i. pre_s \uparrow P(i) = [\$ok \mapsto_s true, \$wait \mapsto_s false] \uparrow P(i)$
 by (*rel-auto*)
 thus ?thesis
 by (*simp add: usubst*)
 qed
 also have ... =
 $\mathbf{R}_s ((\prod_{i \in A} \cdot P(i)) \vdash ((\prod_{i \in A} \cdot (P(i) \Rightarrow Q(i))) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\prod_{i \in A} \cdot (P(i) \Rightarrow$
 $Q(i))))$
 by (*simp add: rdes-export-pre not-UNIF*)
 also have ... = $\mathbf{R}_s ((\prod_{i \in A} \cdot P(i)) \vdash ((\prod_{i \in A} \cdot Q(i)) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\prod_{i \in A} \cdot Q(i))))$
 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 $(\Box i \in A . \mathbf{R}_s(P_1(i) \vdash P_2(i) \diamond P_3(i))) =$
 $\mathbf{R}_s ((\Box i \in A . P_1(i)) \vdash (((\Box i \in A . P_2(i)) \triangleleft \$tr' =_u \$tr \triangleright (\Box i \in A . P_2(i))) \diamond (\Box i \in A . P_3(i))))$
proof –
have $(\Box i \in A . \mathbf{R}_s(P_1(i) \vdash P_2(i) \diamond P_3(i))) =$
 $\mathbf{R}_s ((\Box i \in A . P_1(i)) \vdash ((\Box i \in A . P_2(i) \diamond P_3(i)) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\Box i \in A . P_2(i) \diamond P_3(i))))$
by (*simp add: ExtChoice-rdes assms*)
also
have ... =
 $\mathbf{R}_s ((\Box i \in A . P_1(i)) \vdash ((\Box i \in A . P_2(i) \diamond P_3(i)) \triangleleft \$wait' \wedge \$tr' =_u \$tr \triangleright (\Box i \in A . P_2(i) \diamond P_3(i))))$
by (*simp add: conj-comm*)
also
have ... =
 $\mathbf{R}_s ((\Box i \in A . P_1(i)) \vdash (((\Box i \in A . P_2(i) \diamond P_3(i)) \triangleleft \$tr' =_u \$tr \triangleright (\Box i \in A . P_2(i) \diamond P_3(i))) \diamond (\Box i \in A . P_2(i) \diamond P_3(i))))$
by (*simp add: cond-conj wait'-cond-def*)
also
have ... = $\mathbf{R}_s ((\Box i \in A . P_1(i)) \vdash (((\Box i \in A . P_2(i)) \triangleleft \$tr' =_u \$tr \triangleright (\Box i \in A . P_2(i))) \diamond (\Box i \in A . P_3(i))))$
by (*rule cong[of $\mathbf{R}_s \ \mathbf{R}_s$], simp, rel-auto*)
finally show *?thesis* .
qed

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

assumes $\bigwedge i . \$ok' \nmid P_1(i) \ A \neq \{\}$
shows $(\Box i \in A . \mathbf{R}_s(P_1(i) \vdash P_2(i) \diamond P_3(i))) =$
 $\mathbf{R}_s ((\Box i \in A . P_1(i)) \vdash (((\Box i \in A . R5(P_2(i))) \vee (\Box i \in A . R4(P_2(i)))) \diamond (\Box i \in A . P_3(i))))$
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 $ExtChoice \ A = \mathbf{R}_s ((\Box P \in A . pre_R \ P) \vdash (((\Box P \in A . peri_R \ P) \triangleleft \$tr' =_u \$tr \triangleright (\Box P \in A . post_R \ P)))$
proof –
have $((\Box P \in A . cmt_R \ P) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\Box P \in A . cmt_R \ P)) =$
 $((\Box P \in A . cmt_R \ P) \triangleleft \$tr' =_u \$tr \triangleright (\Box P \in A . cmt_R \ P)) \diamond (\Box P \in A . cmt_R \ P)$
by (*rel-auto*)
also have ... = $((\Box P \in A . peri_R \ P) \triangleleft \$tr' =_u \$tr \triangleright (\Box P \in A . peri_R \ P)) \diamond (\Box P \in A . post_R \ P)$
by (*rel-auto*)
finally show *?thesis*
by (*simp add: ExtChoice-def*)
qed

lemma *extChoice-rdes*:

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

proof –

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

lemma *extChoice-tri-rdes*:

assumes $\$ok' \# P_1 \ \$ok' \# Q_1$
 shows $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) \Box \mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3) =$
 $\mathbf{R}_s((P_1 \wedge Q_1) \vdash (((P_2 \wedge Q_2) \triangleleft \$tr' =_u \$tr \triangleright (P_2 \vee Q_2)) \diamond (P_3 \vee Q_3)))$

proof –

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

qed

lemma *extChoice-rdes-def*:

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

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

assumes $P_1 \text{ is } RR \ Q_1 \text{ is } RR$
 shows $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) \Box \mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3) =$
 $\mathbf{R}_s((P_1 \wedge Q_1) \vdash ((R5(P_2 \wedge Q_2) \vee R4(P_2 \vee Q_2)) \diamond (P_3 \vee Q_3)))$
 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]*:

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

lemma *preR-ExtChoice [rdes]*:

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

proof –

have $pre_R (ExtChoice A) = (R1 (R2c ((\bigcup P \in A \cdot pre_R P))))$
by (*simp add: ExtChoice-def rea-pre-RHS-design usubst unrest*)
also from *assms* **have** $\dots = (R1 (R2c (\bigcup P \in A \cdot (pre_R(CSP(P)))))$
by (*metis USUP-healthy*)
also from *assms* **have** $\dots = (\bigcup P \in A \cdot (pre_R(CSP(P))))$
by (*rel-auto*)
also from *assms* **have** $\dots = (\bigcup P \in A \cdot (pre_R(P)))$
by (*metis USUP-healthy*)
finally show *?thesis* .

qed

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

assumes $A \neq \{\}$ $\bigwedge P. P \in A \implies F(P)$ *is CSP*
shows $pre_R(\bigcup P \in A \cdot F(P)) = (\bigcup P \in A \cdot pre_R(F(P)))$
using *assms* **by** (*subst preR-ExtChoice, auto*)

lemma *periR-ExtChoice [rdes]*:

assumes $A \subseteq \llbracket NCSP \rrbracket_H$ $A \neq \{\}$
shows $peri_R(ExtChoice A) = ((\bigcup P \in A \cdot pre_R(P)) \Rightarrow_r (\bigcup P \in A \cdot peri_R P)) \triangleleft \$tr' =_u \$tr \triangleright (\bigcap P \in A \cdot peri_R P)$

proof –

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

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

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

proof –

have $(\bigcup P \in A \cdot [\$ok \mapsto_s true, \$ok' \mapsto_s true, \$wait \mapsto_s false, \$wait' \mapsto_s true] \dagger pre_R (NCSP P))$
 $= (\bigcup P \in A \cdot pre_R (NCSP P))$
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 $\dots = R1 ((\bigcup P \in A \cdot pre_R (NCSP P)) \Rightarrow_r$
 $(\bigcup P \in A \cdot peri_R (NCSP P))$
 $\triangleleft \$tr' =_u \$tr \triangleright$
 $(\bigcap P \in A \cdot peri_R (NCSP P)))$
by (*simp add: R2c-rea-impl R2c-condr R2c-UINF R2c-preR R2c-periR R2c-tr'-minus-tr R2c-USUP closure*)

also have $\dots = (((\bigcup P \in A \cdot pre_R (NCSP P)) \Rightarrow_r (\bigcup P \in A \cdot peri_R (NCSP P)))$
 $\triangleleft \$tr' =_u \$tr \triangleright$
 $((\bigcup P \in A \cdot pre_R (NCSP P)) \Rightarrow_r (\bigcap P \in A \cdot peri_R (NCSP P))))$
by (*simp add: R1-rea-impl R1-cond R1-USUP R1-UINF assms Healthy-if closure, rel-auto*)
also have $\dots = (((\bigcup P \in A \cdot pre_R (NCSP P)) \Rightarrow_r (\bigcup P \in A \cdot peri_R (NCSP P)))$
 $\triangleleft \$tr' =_u \$tr \triangleright$

$((\prod P \in A \cdot pre_R (NCSP P)) \Rightarrow_r peri_R (NCSP P)))$
 by (simp add: UINF-rea-impl[THEN sym])
 also have ... = $((\prod P \in A \cdot pre_R (NCSP P)) \Rightarrow_r (\prod P \in A \cdot peri_R (NCSP P)))$
 $\triangleleft \$tr' =_u \$tr \triangleright$
 $((\prod P \in A \cdot peri_R (NCSP P)))$
 by (simp add: SRD-peri-under-pre closure assms unrest)
 also have ... = $((\prod P \in A \cdot pre_R P) \Rightarrow_r (\prod P \in A \cdot peri_R P))$
 $\triangleleft \$tr' =_u \$tr \triangleright$
 $((\prod P \in A \cdot peri_R P))$
 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 $peri_R(ExtChoice A) = (R5((\prod P \in A \cdot pre_R(P)) \Rightarrow_r (\prod P \in A \cdot peri_R P)) \vee (\prod P \in A \cdot R4(peri_R P)))$
 using assms(2)
 by (simp add: periR-ExtChoice assms(1), rel-auto)

lemma *periR-ExtChoice-ind* [rdes]:

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

lemma *periR-ExtChoice-ind'*:

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

lemma *postR-ExtChoice* [rdes]:

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

proof –

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

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

also have ... = $R1(R2c((\prod P \in A \cdot pre_R (NCSP P)) \Rightarrow_r (\prod P \in A \cdot post_R (NCSP P))))$

proof –

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

qed

also have ... = $R1((\prod P \in A \cdot pre_R (NCSP P)) \Rightarrow_r (\prod P \in A \cdot post_R (NCSP P)))$

by (simp add: R2c-rea-impl R2c-cond R2c-UINF R2c-preR R2c-postR
 R2c-tr'-minus-tr R2c-USUP closure)
 also from assms(2) have ... = $((\bigwedge P \in A \cdot \text{pre}_R(\text{NCSP } P)) \Rightarrow_r (\bigwedge P \in A \cdot \text{post}_R(\text{NCSP } P)))$
 by (simp add: R1-rea-impl R1-cond R1-USUP R1-UINF assms Healthy-if closure)
 also have ... = $(\bigwedge P \in A \cdot \text{pre}_R(\text{NCSP } P) \Rightarrow_r \text{post}_R(\text{NCSP } P))$
 by (simp add: UINF-rea-impl)
 also have ... = $(\bigwedge P \in A \cdot \text{post}_R(\text{NCSP } P))$
 by (simp add: SRD-post-under-pre closure assms unrest)
 finally show ?thesis
 by (simp add: UINF-healthy[OF assms(1), THEN sym] USUP-healthy[OF assms(1), THEN sym])
 qed

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

lemma *preR-extChoice*:
 assumes $P \text{ is CSP } Q \text{ is CSP } \$wait' \# \text{pre}_R(P) \$wait' \# \text{pre}_R(Q)$
 shows $\text{pre}_R(P \sqcap Q) = (\text{pre}_R(P) \wedge \text{pre}_R(Q))$
 by (simp add: extChoice-def preR-ExtChoice assms usup-and)

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

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

lemma *postR-extChoice* [rdes]:
 assumes $P \text{ is NCSP } Q \text{ is NCSP}$
 shows $\text{post}_R(P \sqcap Q) = (\text{post}_R(P) \vee \text{post}_R(Q))$
 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 $(\bigwedge P \in A \cdot F(P)) = (\bigwedge P \in A \cdot G(P))$
 using assms image-cong by force

lemma *ref-unrest-ExtChoice*:
 assumes
 $\bigwedge P. P \in A \implies \$ref \# \text{pre}_R(P)$
 $\bigwedge P. P \in A \implies \$ref \# \text{cmt}_R(P)$
 shows $\$ref \# (\text{ExtChoice } A) \llbracket \text{false} / \$wait \rrbracket$
proof –
 have $\bigwedge P. P \in A \implies \$ref \# \text{pre}_R(P \llbracket 0 / \$tr \rrbracket)$
 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 $A \subseteq \llbracket \text{NCSP} \rrbracket_H$

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: $(\neg_r (\neg_r \text{pre}_R (\text{ExtChoice } A)) \text{ ;;}_h R1 \text{ true}) = \text{pre}_R (\text{ExtChoice } A)$

proof –

have $\bigwedge P. P \in A \implies (\neg_r \text{pre}_R(P)) \text{ ;; } R1 \text{ true} = (\neg_r \text{pre}_R(P))$

by (*simp add: NCSP-Healthy-subset-member NCSP-implies-NSRD NSRD-neg-pre-unit assms*)

thus *?thesis*

apply (*simp add: False preR-ExtChoice closure NCSP-set-unrest-pre-wait' assms not-UINF seq-UINF-distr not-USUP*)

apply (*rule USUP-cong*)

apply (*simp add: rpred assms closure*)

done

qed

have 2: $\$st' \# \text{peri}_R (\text{ExtChoice } A)$

proof –

have $a: \bigwedge P. P \in A \implies \$st' \# \text{pre}_R(P)$

by (*simp add: NCSP-Healthy-subset-member NCSP-implies-NSRD NSRD-st'-unrest-pre assms*)

have $b: \bigwedge P. P \in A \implies \$st' \# \text{peri}_R(P)$

by (*simp add: NCSP-Healthy-subset-member NCSP-implies-NSRD NSRD-st'-unrest-peri assms*)

from $a \ b$ **show** *?thesis*

apply (*subst periR-ExtChoice*)

apply (*simp-all add: assms closure unrest CSP4-set-unrest-pre-st' NCSP-set-unrest-pre-wait'*

False)

done

qed

have 3: $\$ref' \# \text{post}_R (\text{ExtChoice } A)$

proof –

have $a: \bigwedge P. P \in A \implies \$ref' \# \text{pre}_R(P)$

by (*simp add: CSP4-ref'-unrest-pre CSP-Healthy-subset-member NCSP-Healthy-subset-member NCSP-implies-CSP4 NCSP-subset-implies-CSP assms*)

have $b: \bigwedge P. P \in A \implies \$ref' \# \text{post}_R(P)$

by (*simp add: CSP4-ref'-unrest-post CSP-Healthy-subset-member NCSP-Healthy-subset-member NCSP-implies-CSP4 NCSP-subset-implies-CSP assms*)

from $a \ b$ **show** *?thesis*

by (*subst postR-ExtChoice, simp-all add: assms CSP4-set-unrest-pre-st' NCSP-set-unrest-pre-wait' unrest False*)

qed

show *?thesis*

by (*rule CSP4-tri-intro, simp-all add: 1 2 3 assms closure*)

(*metis 1 R1-seqr-closure rea-not-R1 rea-not-not rea-true-R1*)

qed

lemma *CSP4-extChoice [closure]*:

assumes P is NCSP Q is NCSP

shows $P \sqcap Q$ is CSP4

by (*simp add: extChoice-def, rule CSP4-ExtChoice, simp-all add: assms*)

lemma *NCSP-ExtChoice* [closure]:
assumes $A \subseteq \llbracket \text{NCSP} \rrbracket_H$
shows *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** $cls: A \subseteq \llbracket \text{CSP} \rrbracket_H \ A \subseteq \llbracket \text{CSP}^3 \rrbracket_H \ A \subseteq \llbracket \text{CSP}^4 \rrbracket_H$
 using *NCSP-implies-CSP NCSP-implies-CSP³ NCSP-implies-CSP⁴* **by** blast+
 have $wu: \bigwedge P. P \in A \implies \$wait' \nmid pre_R(P)$
 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: $\bigwedge P. P \in A \implies \$wait' \nmid pre_R(P)$
 using *NCSP-implies-NSRD NSRD-wait'-unrest-pre assms(2)* **by** blast
 show ?thesis
proof (rule *Productive-intro, simp-all add: assms closure rdes 1 unrest*)
 have $((\bigwedge P \in A. pre_R P) \wedge (\bigwedge P \in A. post_R P)) =$
 $((\bigwedge P \in A. pre_R P) \wedge (\bigwedge P \in A. (pre_R P \wedge post_R P)))$
 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* $(\lambda X. \sqcap P \in A \cdot P(X))$

proof (*rule GuardedI*)

fix $X \ n$

have $\bigwedge Y. ((\sqcap P \in A \cdot P \ Y) \wedge \text{gvr}(n+1)) = ((\sqcap P \in A \cdot (P \ Y \wedge \text{gvr}(n+1))) \wedge \text{gvr}(n+1))$

proof –

fix Y

let $?lhs = ((\sqcap P \in A \cdot P \ Y) \wedge \text{gvr}(n+1))$ **and** $?rhs = ((\sqcap P \in A \cdot (P \ Y \wedge \text{gvr}(n+1))) \wedge \text{gvr}(n+1))$

have $a: ?lhs \llbracket \text{false}/\$ok \rrbracket = ?rhs \llbracket \text{false}/\$ok \rrbracket$

by (*rel-auto*)

have $b: ?lhs \llbracket \text{true}/\$ok \rrbracket \llbracket \text{true}/\$wait \rrbracket = ?rhs \llbracket \text{true}/\$ok \rrbracket \llbracket \text{true}/\$wait \rrbracket$

by (*rel-auto*)

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

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

show $?lhs = ?rhs$

using $a \ b \ c$

by (*rule-tac bool-eq-splitI[of in-var ok], simp, rule-tac bool-eq-splitI[of in-var wait], simp-all*)

qed

moreover have $((\sqcap P \in A \cdot (P \ X \wedge \text{gvr}(n+1))) \wedge \text{gvr}(n+1)) = ((\sqcap P \in A \cdot (P \ (X \wedge \text{gvr}(n)) \wedge \text{gvr}(n+1))) \wedge \text{gvr}(n+1))$

proof –

have $(\sqcap P \in A \cdot (P \ X \wedge \text{gvr}(n+1))) = (\sqcap P \in A \cdot (P \ (X \wedge \text{gvr}(n)) \wedge \text{gvr}(n+1)))$

proof (*rule ExtChoice-cong*)

fix P **assume** $P \in A$

thus $(P \ X \wedge \text{gvr}(n+1)) = (P \ (X \wedge \text{gvr}(n)) \wedge \text{gvr}(n+1))$

using *Guarded-def assms* **by** *blast*

qed

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

qed

ultimately show $((\sqcap P \in A \cdot P \ X) \wedge \text{gvr}(n+1)) = ((\sqcap P \in A \cdot (P \ (X \wedge \text{gvr}(n)))) \wedge \text{gvr}(n+1))$

by *simp*

qed

lemma *extChoice-Guarded* [closure]:

assumes *Guarded* P *Guarded* Q

shows *Guarded* $(\lambda X. P(X) \sqcap Q(X))$

proof –

have *Guarded* $(\lambda X. \sqcap F \in \{P, Q\} \cdot F(X))$

by (*rule ExtChoice-Guarded, auto simp add: assms*)

thus *?thesis*

by (*simp add: extChoice-def*)

qed

7.6 Algebraic laws

lemma *extChoice-comm*:

$P \sqcap Q = Q \sqcap P$

by (*unfold extChoice-def, simp add: insert-commute*)

lemma *extChoice-idem*:

P is CSP $\implies P \sqcap P = P$

by (*unfold extChoice-def*, *simp add: ExtChoice-single*)

lemma *extChoice-assoc*:

assumes P is CSP Q is CSP R is CSP

shows $P \sqcap Q \sqcap R = P \sqcap (Q \sqcap R)$

proof –

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

by (*simp add: SRD-reactive-design-alt assms(1) assms(2) assms(3)*)

also have ... =

$\mathbf{R}_s(((\text{pre}_R P \wedge \text{pre}_R Q) \wedge \text{pre}_R R) \vdash$
 $((\text{cmt}_R P \wedge \text{cmt}_R Q) \triangleleft \text{\$tr}' =_u \text{\$tr} \wedge \text{\$wait}' \triangleright (\text{cmt}_R P \vee \text{cmt}_R Q) \wedge \text{cmt}_R R)$
 $\triangleleft \text{\$tr}' =_u \text{\$tr} \wedge \text{\$wait}' \triangleright$
 $((\text{cmt}_R P \wedge \text{cmt}_R Q) \triangleleft \text{\$tr}' =_u \text{\$tr} \wedge \text{\$wait}' \triangleright (\text{cmt}_R P \vee \text{cmt}_R Q) \vee \text{cmt}_R R)))$

by (*simp add: extChoice-rdes unrest*)

also have ... =

$\mathbf{R}_s(((\text{pre}_R P \wedge \text{pre}_R Q) \wedge \text{pre}_R R) \vdash$
 $((\text{cmt}_R P \wedge \text{cmt}_R Q) \wedge \text{cmt}_R R)$
 $\triangleleft \text{\$tr}' =_u \text{\$tr} \wedge \text{\$wait}' \triangleright$
 $((\text{cmt}_R P \vee \text{cmt}_R Q) \vee \text{cmt}_R R)))$

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

also have ... =

$\mathbf{R}_s((\text{pre}_R P \wedge \text{pre}_R Q \wedge \text{pre}_R R) \vdash$
 $((\text{cmt}_R P \wedge (\text{cmt}_R Q \wedge \text{cmt}_R R))$
 $\triangleleft \text{\$tr}' =_u \text{\$tr} \wedge \text{\$wait}' \triangleright$
 $(\text{cmt}_R P \vee (\text{cmt}_R Q \vee \text{cmt}_R R))))$

by (*simp add: conj-assoc disj-assoc*)

also have ... =

$\mathbf{R}_s((\text{pre}_R P \wedge \text{pre}_R Q \wedge \text{pre}_R R) \vdash$
 $((\text{cmt}_R P \wedge (\text{cmt}_R Q \wedge \text{cmt}_R R) \triangleleft \text{\$tr}' =_u \text{\$tr} \wedge \text{\$wait}' \triangleright (\text{cmt}_R Q \vee \text{cmt}_R R))$
 $\triangleleft \text{\$tr}' =_u \text{\$tr} \wedge \text{\$wait}' \triangleright$
 $(\text{cmt}_R P \vee (\text{cmt}_R Q \wedge \text{cmt}_R R) \triangleleft \text{\$tr}' =_u \text{\$tr} \wedge \text{\$wait}' \triangleright (\text{cmt}_R Q \vee \text{cmt}_R R))))$

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

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

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 $\text{Stop} \sqcap Q = Q$

using *assms*

proof –

have $\text{Stop} \sqcap Q = \mathbf{R}_s(\text{true} \vdash (\text{\$tr}' =_u \text{\$tr} \wedge \text{\$wait}')) \sqcap \mathbf{R}_s(\text{pre}_R(Q) \vdash \text{cmt}_R(Q))$

by (*simp add: Stop-def SRD-reactive-design-alt assms*)

also have ... = $\mathbf{R}_s(\text{pre}_R Q \vdash (((\text{\$tr}' =_u \text{\$tr} \wedge \text{\$wait}') \wedge \text{cmt}_R Q) \triangleleft \text{\$tr}' =_u \text{\$tr} \wedge \text{\$wait}' \triangleright (\text{\$tr}' =_u \text{\$tr} \wedge \text{\$wait}' \vee \text{cmt}_R Q)))$

by (*simp add: extChoice-rdes unrest*)

also have ... = $\mathbf{R}_s(\text{pre}_R Q \vdash (\text{cmt}_R Q \triangleleft \text{\$tr}' =_u \text{\$tr} \wedge \text{\$wait}' \triangleright \text{cmt}_R Q))$

by (*metis (no-types, lifting) cond-def eq-upred-sym neg-conj-cancel1 utp-pred-laws.inf.left-idem*)

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

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 $\text{Chaos} \sqcap Q = \text{Chaos}$

proof –
 have $\text{Chaos} \sqcap Q = \mathbf{R}_s(\text{false} \vdash \text{true}) \sqcap \mathbf{R}_s(\text{pre}_R(Q) \vdash \text{cmt}_R(Q))$
 by (simp add: Chaos-def SRD-reactive-design-alt assms)
 also have ... = $\mathbf{R}_s(\text{false} \vdash (\text{cmt}_R Q \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright \text{true}))$
 by (simp add: extChoice-rdes unrest)
 also have ... = $\mathbf{R}_s(\text{false} \vdash \text{true})$
 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 $P \sqcap (\bigsqcup S) = (\bigsqcup_{Q \in S} P \sqcap Q)$

proof –
 let ?S1 = $\text{pre}_R S$ and ?S2 = $\text{cmt}_R S$
 have $P \sqcap (\bigsqcup S) = P \sqcap (\bigsqcup_{Q \in S} \mathbf{R}_s(\text{pre}_R(Q) \vdash \text{cmt}_R(Q)))$
 by (simp add: SRD-as-reactive-design[THEN sym] Healthy-SUPREMUM UINF-as-Sup-collect assms)
 also have ... = $\mathbf{R}_s(\text{pre}_R(P) \vdash \text{cmt}_R(P)) \sqcap \mathbf{R}_s((\bigsqcup_{Q \in S} \text{pre}_R(Q)) \vdash (\bigsqcup_{Q \in S} \text{cmt}_R(Q)))$
 by (simp add: RHS-design-USUP SRD-reactive-design-alt assms)
 also have ... = $\mathbf{R}_s((\text{pre}_R(P) \wedge (\bigsqcup_{Q \in S} \text{pre}_R(Q))) \vdash$
 $((\text{cmt}_R(P) \wedge (\bigsqcup_{Q \in S} \text{cmt}_R(Q)))$
 $\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
 $(\text{cmt}_R(P) \vee (\bigsqcup_{Q \in S} \text{cmt}_R(Q))))$
 by (simp add: extChoice-rdes unrest)
 also have ... = $\mathbf{R}_s((\bigsqcup_{Q \in S} \text{pre}_R P \wedge \text{pre}_R Q) \vdash$
 $(\bigsqcup_{Q \in S} (\text{cmt}_R P \wedge \text{cmt}_R Q) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\text{cmt}_R P \vee \text{cmt}_R Q)))$
 by (simp add: conj-USUP-dist conj-UINF-dist disj-UINF-dist cond-UINF-dist assms)
 also have ... = $(\bigsqcup_{Q \in S} \mathbf{R}_s((\text{pre}_R P \wedge \text{pre}_R Q) \vdash$
 $((\text{cmt}_R P \wedge \text{cmt}_R Q) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\text{cmt}_R P \vee \text{cmt}_R Q))))$
 by (simp add: assms RHS-design-USUP)
 also have ... = $(\bigsqcup_{Q \in S} \mathbf{R}_s(\text{pre}_R(P) \vdash \text{cmt}_R(P)) \sqcap \mathbf{R}_s(\text{pre}_R(Q) \vdash \text{cmt}_R(Q)))$
 by (simp add: extChoice-rdes unrest)
 also have ... = $(\bigsqcup_{Q \in S} P \sqcap \text{CSP}(Q))$
 by (simp add: UINF-as-Sup-collect, metis (no-types, lifting) Healthy-if SRD-as-reactive-design
 assms(1))
 also have ... = $(\bigsqcup_{Q \in S} P \sqcap Q)$
 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  $\bigwedge i. i \in A \implies P\ i \text{ is PCSP } Q \text{ is NCSP}$ 
  shows  $(\Box i \in A \cdot P\ i) ;; Q = (\Box i \in A \cdot P\ i ;; Q)$ 
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  $1: (\Box i \in A \cdot P\ i) = (\Box i \in A \cdot (\mathbf{R}_s ((pre_R (P\ i)) \vdash peri_R (P\ i) \diamond (R4(post_R (P\ i)))))$ 
      (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)

lemma extchoice-StateInvR-refine:
  assumes
     $P \text{ is NCSP } Q \text{ is NCSP}$ 
     $sinv_R(b) \sqsubseteq P\ sinv_R(b) \sqsubseteq Q$ 
  shows  $sinv_R(b) \sqsubseteq P \Box Q$ 
proof –
  have  $1:$ 
     $pre_R P \sqsubseteq [b]_{S<} [b]_{S>} \sqsubseteq ([b]_{S<} \wedge post_R P)$ 
     $pre_R Q \sqsubseteq [b]_{S<} [b]_{S>} \sqsubseteq ([b]_{S<} \wedge post_R Q)$ 
  by (metis (no-types, lifting) CRR-implies-RR NCSP-implies-CSP RHS-tri-design-refine SRD-reactive-tri-design
StateInvR-def assms periR-RR postR-RR preR-CRR rea-st-cond-RR rea-true-RR refBy-order st-post-CRR)
  show ?thesis
  by (rdes-refine-split cls: assms(1–2), simp-all add: 1 closure assms truer-bottom-rpred utp-pred-laws.inf-sup-distrib1)
qed

end

```

8 Stateful-Failure Programs

theory *utp-sfrd-prog*


```

imports
  UTP.utp-full
  utp-sfrd-extchoice
begin

```

8.1 Conditionals

```

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

```

8.2 Guarded commands

```

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

```

8.3 Assumptions

```

definition AssumeCircus ( $[-]_C$ ) where
  [rdes-def]:  $[b]_C = b \rightarrow_R \text{Skip}$ 

```

```

lemma AssumeCircus-NCSP [closure]:  $[b]_C \text{ is NCSP}$ 
  by (simp add: AssumeCircus-def GuardedCommR-NCSP-closed NCSP-Skip)

```

```

lemma AssumeCircus-AssumeR:  $\text{Skip} ;; [b]^\top_R = [b]_C [b]^\top_R ;; \text{Skip} = [b]_C$ 
  by (rdes-eq)+

```

```

lemma AssumeR-comp-AssumeCircus:  $P \text{ is NCSP} \implies P ;; [b]^\top_R = P ;; [b]_C$ 
  by (metis (no-types, hide-lams) AssumeCircus-AssumeR(1) RA1 Skip-right-unit)

```

```

lemma gcmd-AssumeCircus:
   $P \text{ is NCSP} \implies b \rightarrow_R P = [b]_C ;; P$ 
  by (simp add: AssumeCircus-def NCSP-implies-NSRD Skip-left-unit gcmd-seq-distr)

```

```

lemma rdes-assume-pre-refine:
  assumes P is NCSP
  shows  $P \sqsubseteq [b]_C ;; P$ 
  by (rdes-refine cls: assms)

```

8.4 Alternation

```

lemma AlternateR-NCSP-closed [closure]:
  assumes  $\bigwedge i. i \in A \implies P(i) \text{ is NCSP } Q \text{ is NCSP}$ 
  shows (ifR i ∈ A · g(i) → P(i) else Q fi) 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

```

lemma *AlternateR-list-NCSP-closed* [closure]:
assumes $\bigwedge b P. (b, P) \in \text{set } A \implies P \text{ is NCSP } Q \text{ is NCSP}$
shows $(\text{AlternateR-list } A \ Q) \text{ is NCSP}$
apply (simp add: AlternateR-list-def)
apply (rule AlternateR-NCSP-closed)
apply (auto simp add: assms)
apply (metis assms(1) eq-snd-iff nth-mem)
done

8.5 While Loops

lemma *NSRD-coerce-NCSP*:
 $P \text{ is NSRD} \implies \text{Skip} ;; P ;; \text{Skip} \text{ is NCSP}$
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* :: $'s \text{ upred} \Rightarrow ('s, 'e) \text{ action} \Rightarrow ('s, 'e) \text{ action}$ (*while_C* - do - od) **where**
 $\text{while}_C \ b \ \text{do } P \ \text{od} = \text{Skip} ;; \text{while}_R \ b \ \text{do } P \ \text{od} ;; \text{Skip}$

lemma *WhileC-NCSP-closed* [closure]:
assumes $P \text{ is NCSP } P \text{ is Productive}$
shows $\text{while}_C \ b \ \text{do } P \ \text{od} \text{ is NCSP}$
by (simp add: WhileC-def NSRD-coerce-NCSP assms closure)

lemma *WhileC-false*:
 $P \text{ is NCSP} \implies \text{WhileC} \ \text{false} \ P = \text{Skip}$
by (simp add: NCSP-implies-NSRD Skip-srdes-left-unit WhileC-def WhileR-false)

theorem *WhileC-iter-form* [rdes-def]:
assumes $P \text{ is NCSP } P \text{ is Productive}$
shows $\text{while}_C \ b \ \text{do } P \ \text{od} = ([b]_C ;; P)^{*C} ;; [\neg b]_C$
by (simp add: WhileC-def WhileR-iter-form assms closure seqr-assoc AssumeCircus-AssumeR)
(metis (no-types, lifting) AssumeCircus-AssumeR(2) AssumeCircus-NCSP RA1 assms(1) csp-theory.Healthy-Sequenc
csp-theory.Unit-Left sfrd-star-as-rdes-star')

8.6 Iteration Construction

definition *IterateC* :: $'a \text{ set} \Rightarrow ('a \Rightarrow 's \text{ upred}) \Rightarrow ('a \Rightarrow ('s, 'e) \text{ action}) \Rightarrow ('s, 'e) \text{ action}$
where [upred-defs, ndes-simp]: $\text{IterateC } A \ g \ P = \text{while}_C \ (\bigvee i \in A \cdot g(i)) \ \text{do } (\text{if}_R \ i \in A \cdot g(i) \rightarrow P(i) \ \text{fi})$
od

lemma *IterateC-IterateR-def*: $\text{IterateC } A \ g \ P = \text{Skip} ;; \text{IterateR } A \ g \ P ;; \text{Skip}$
by (simp add: IterateC-def IterateR-def WhileC-def)

definition *IterateC-list* :: $('s \text{ upred} \times ('s, 'e) \text{ action}) \text{ list} \Rightarrow ('s, 'e) \text{ action}$ **where**
[upred-defs, ndes-simp]:
 $\text{IterateC-list } xs = \text{IterateC} \ \{0..<\text{length } xs\} \ (\lambda i. \text{map fst } xs \ ! \ i) \ (\lambda i. \text{map snd } xs \ ! \ i)$

syntax

-iter-C :: $\text{pttrn} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic}$ (*do_C* -∈- · - → - od)
-iter-gcommC :: $\text{gcomms} \Rightarrow \text{logic}$ (*do_C* / - /od)

translations

-iter-C $x \ A \ g \ P \Rightarrow \text{CONST } \text{IterateC } A \ (\lambda x. g) \ (\lambda x. P)$
-iter-C $x \ A \ g \ P \Leftarrow \text{CONST } \text{IterateC } A \ (\lambda x. g) \ (\lambda x'. P)$

-iter-gcommC cs \rightarrow CONST IterateC-list cs
 -iter-gcommC (-gcomm-show cs) \leftarrow CONST IterateC-list cs

lemma IterateC-NCSP-closed [closure]:

assumes

$\bigwedge i. i \in I \implies P(i)$ is NCSP

$\bigwedge i. i \in I \implies P(i)$ is Productive

shows $do_C i \in I \cdot g(i) \rightarrow P(i)$ od is NCSP

by (simp add: IterateC-IterateR-def IterateR-NSRD-closed NCSP-implies-NSRD NSRD-coerce-NCSP
 assms(1) assms(2))

lemma IterateC-list-NCSP-closed [closure]:

assumes

$\bigwedge b P. (b, P) \in \text{set } A \implies P$ is NCSP

$\bigwedge b P. (b, P) \in \text{set } A \implies P$ is Productive

shows IterateC-list A is NCSP

apply (simp add: IterateC-list-def, rule IterateC-NCSP-closed)

apply (metis assms atLeastLessThan-iff nth-map nth-mem prod.collapse)+
done

lemma IterateC-list-alt-def:

IterateC-list xs = while_C ($\bigvee b \in \text{set}(\text{map fst xs}) \cdot b$) do AlternateR-list xs Chaos od

proof –

have ($\bigvee i \in \{0..<\text{length}(xs)\} \cdot (\text{map fst xs}) ! i$) = ($\bigvee b \in \text{set}(\text{map fst xs}) \cdot b$)

by (rel-auto, metis nth-mem prod.collapse, metis fst-conv in-set-conv-nth nth-map)

thus ?thesis

by (simp add: IterateC-list-def IterateC-def AlternateR-list-def)

qed

lemma IterateC-empty:

$do_C i \in \{\} \cdot g(i) \rightarrow P(i)$ od = Skip

by (simp add: IterateC-IterateR-def IterateR-empty closure Skip-srdes-left-unit)

lemma IterateC-singleton:

assumes P k is NCSP P k is Productive

shows $do_C i \in \{k\} \cdot g(i) \rightarrow P(i)$ od = while_C g(k) do P(k) od (**is** ?lhs = ?rhs)

by (simp add: IterateC-IterateR-def IterateR-singleton NCSP-implies-NSRD WhileC-def assms)

lemma IterateC-outer-refine-intro:

assumes $I \neq \{\} \bigwedge i. i \in I \implies P i$ is NCSP $\bigwedge i. i \in I \implies P i$ is Productive

$\bigwedge i. i \in I \implies S \sqsubseteq (b i \rightarrow_R P i ;; S)$ S is NCSP

$S \sqsubseteq [\neg (\bigwedge i \in I \cdot b i)]^\top_R$

shows $S \sqsubseteq do_C i \in I \cdot b(i) \rightarrow P(i)$ od

proof –

have $S \sqsubseteq do_R i \in I \cdot b(i) \rightarrow P(i)$ od

by (simp add: IterateR-outer-refine-intro NCSP-implies-NSRD assms)

thus ?thesis

unfolding IterateC-IterateR-def

by (metis (full-types) Skip-left-unit Skip-right-unit assms(5) urel-diod.mult-isol urel-diod.mult-isor)

qed

lemma IterateC-outer-refine-init-intro:

assumes

$\bigwedge i. i \in A \implies P i$ is NCSP

$\bigwedge i. i \in A \implies P i$ is Productive

S is NCSP I is NCSP
 $S \sqsubseteq I$;; $[\neg (\bigcap i \in A \cdot b\ i)]^\top_R$
 $\bigwedge i. i \in A \implies S \sqsubseteq S$;; $b\ i \rightarrow_R P\ i$
 $\bigwedge i. i \in A \implies S \sqsubseteq I$;; $b\ i \rightarrow_R P\ i$
shows $S \sqsubseteq I$;; $do_C\ i \in A \cdot b(i) \rightarrow P(i)$ *od*
proof (*cases* $A = \{\}$)
 case *True*
 with *assms*(5) **show** *?thesis*
 by (*simp add: IterateC-empty assms closure Skip-right-unit AssumeR-true NSRD-right-unit*)
next
 case *False*
 have $S \sqsubseteq I$;; $do_R\ i \in A \cdot b(i) \rightarrow P(i)$ *od*
 by (*simp add: IterateR-outer-refine-init-intro NCSP-implies-NSRD assms False*)
 thus *?thesis*
 unfolding *IterateC-IterateR-def*
 by (*metis (no-types, hide-lams) RA1 Skip-right-unit assms(3) assms(4) urel-dioid.mult-isor*)
qed

lemma *IterateC-list-outer-refine-intro:*

assumes
 $A \neq []$ S is NCSP
 $\bigwedge b\ P. (b, P) \in \text{set } A \implies P$ is NCSP
 $\bigwedge b\ P. (b, P) \in \text{set } A \implies P$ is Productive
 $\bigwedge b\ P. (b, P) \in \text{set } A \implies S \sqsubseteq (b \rightarrow_R P$;; $S)$
 $S \sqsubseteq [\neg (\bigcap (b, P) \in \text{set } A \cdot b)]^\top_R$
shows $S \sqsubseteq \text{IterateC-list } A$
proof –
have $(\bigcap i \in \{0..<\text{length}(A)\} \cdot (\text{map fst } A) ! i) = (\bigcap (b, P) \in \text{set } A \cdot b)$
 by (*rel-auto, metis nth-mem prod.exhaust-sel, metis fst-conv in-set-conv-nth nth-map*)
thus *?thesis*
 apply (*simp add: IterateC-list-def*)
 apply (*rule IterateC-outer-refine-intro*)
 apply (*simp-all add: closure assms*)
 apply (*metis assms(3) nth-mem prod.collapse*)
 apply (*metis assms(4) nth-mem prod.collapse*)
 done
qed

lemma *IterateC-list-outer-refine-init-intro:*

assumes
 S is NCSP I is NCSP
 $\bigwedge b\ P. (b, P) \in \text{set } A \implies P$ is NCSP
 $\bigwedge b\ P. (b, P) \in \text{set } A \implies P$ is Productive
 $S \sqsubseteq I$;; $[\neg (\bigcap (b, P) \in \text{set } A \cdot b)]^\top_R$
 $\bigwedge b\ P. (b, P) \in \text{set } A \implies S \sqsubseteq S$;; $b \rightarrow_R P$
 $\bigwedge b\ P. (b, P) \in \text{set } A \implies S \sqsubseteq I$;; $b \rightarrow_R P$
shows $S \sqsubseteq I$;; $\text{IterateC-list } A$
proof –
have $(\bigcap i \in \{0..<\text{length}(A)\} \cdot (\text{map fst } A) ! i) = (\bigcap (b, P) \in \text{set } A \cdot b)$
 by (*rel-auto, metis nth-mem prod.exhaust-sel, metis fst-conv in-set-conv-nth nth-map*)
thus *?thesis*
 apply (*simp add: IterateC-list-def*)
 apply (*rule IterateC-outer-refine-init-intro*)
 apply (*simp-all add: closure assms*)

```

    apply (metis assms(3) nth-mem prod.collapse)
    apply (metis assms(4) nth-mem prod.collapse)
  done
qed

```

8.7 Assignment

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

syntax

```

-assigns-csp :: svids  $\Rightarrow$  uexprs  $\Rightarrow$  logic ('(-) :=C '(-))
-assigns-csp :: svids  $\Rightarrow$  uexprs  $\Rightarrow$  logic (infixr :=C 64)

```

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]: $post_R(\langle \sigma \rangle_C) = \Phi(true, \sigma, \langle \rangle)$
by (rel-auto)

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

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

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

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

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

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

lemma *AssignsCSP-as-AssignsR*: $\langle \sigma \rangle_R ; Skip = \langle \sigma \rangle_C$
by (rdes-eq)

lemma *AssignC-init-refine-intro*:

assumes

$vwb\text{-}lens\ x\ \$st:x\ \# \ P_2\ \$st:x\ \# \ P_3$

$P_2\ is\ RR\ P_3\ is\ RR\ Q\ is\ NCSP$

$\mathbf{R}_s([\&x =_u \ll k \gg]_{S<} \vdash P_2 \diamond P_3) \sqsubseteq Q$

shows $\mathbf{R}_s(true_r \vdash P_2 \diamond P_3) \sqsubseteq (x :=_C \ll k \gg) ;; Q$

by (*simp add: AssignsCSP-as-AssignsR[THEN sym] assms seqr-assoc Skip-left-unit AssignR-init-refine-intro closure*)

lemma *AssignsCSP-refines-sinv*:

assumes $\sigma \uparrow b$

shows $sinv_R(b) \sqsubseteq \langle \sigma \rangle_C$

apply (*rdes-refine-split*)

apply (*simp-all*)

apply (*metis rea-st-cond-true st-cond-conj utp-pred-laws.inf.absorb-iff2 utp-pred-laws.inf-top-left*)

using *assms* **apply** (*rel-auto*)

done

8.8 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* ::

$(f \Rightarrow 'k\ set) \Rightarrow (f \Rightarrow 'k \Rightarrow 'v \Rightarrow 'f) \Rightarrow (f \Rightarrow 'f) \Rightarrow$

$(k, 'v) \Rightarrow uexpr \Rightarrow ('v, 'v) \Rightarrow uexpr \Rightarrow ('v, 'v) \Rightarrow action\ \mathbf{where}$

[*upred-defs, rdes-def*]: *AssignCSP-update* *domf* *updatef* $x\ k\ v =$

$\mathbf{R}_s([k \in_u uop\ domf\ (\&x)]_{S<} \vdash false \diamond \Phi(true, [x \mapsto_s trop\ updatef\ (\&x)\ k\ v], \langle \rangle))$

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

syntax

-csp-assign-upd :: *svld* $\Rightarrow uexp \Rightarrow uexp \Rightarrow logic\ (-[-] :=_C - [61, 0, 62]\ 62)$

translations

-csp-assign-upd $x\ k\ v == CONST\ AssignCSP\ update\ (CONST\ udom)\ (CONST\ uupd)\ x\ k\ v$

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

AssignCSP-update *domf* *updatef* $x\ k\ v\ is\ CSP$

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

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

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

by (*rel-auto*)

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

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

by (*rel-simp*)

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

$post_R(AssignCSP\ update\ domf\ updatef\ x\ k\ v) =$

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

by (rel-auto)

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

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

proof (rule NCSP-intro)

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

by (simp add: closure)

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

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

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

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

qed

8.9 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

8.10 Guards

definition *GuardCSP* ::

' σ cond \Rightarrow

(' σ , ' φ) action \Rightarrow

(' σ , ' φ) action **where**

[upred-defs]: *GuardCSP g A* = $\mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r \text{pre}_R(A)) \vdash ((\lceil g \rceil_{S<} \wedge \text{cmt}_R(A)) \vee (\lceil \neg g \rceil_{S<}) \wedge \$tr' =_u \$tr \wedge \$wait'))$

syntax

-*GuardCSP* :: *uexp* \Rightarrow *logic* \Rightarrow *logic* (**infixr** &_C 60)

translations

-*GuardCSP b P* == *CONST GuardCSP b P*

lemma *Guard-tri-design*:

$g \&_C P = \mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r \text{pre}_R(P)) \vdash (\text{peri}_R(P) \triangleleft \lceil g \rceil_{S<} \triangleright (\$tr' =_u \$tr)) \diamond (\lceil g \rceil_{S<} \wedge \text{post}_R(P)))$

proof –

have $(\lceil g \rceil_{S<} \wedge \text{cmt}_R P \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait') = (\text{peri}_R(P) \triangleleft \lceil g \rceil_{S<} \triangleright (\$tr' =_u \$tr)) \diamond (\lceil g \rceil_{S<} \wedge \text{post}_R(P))$

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 $(\lceil b \rceil_{S<} \wedge \text{CRR}(P)) = \Phi(b, id, \langle \rangle) ;; \text{CRR}(P)$

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

lemma *Guard-rdes-def'*:

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

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]: $b \&_C P$ is CSP*

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

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

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 $\text{peri}_R(b \&_C P) = (\text{peri}_R P \triangleleft b \triangleright_R \mathcal{E}(\text{true}, \langle \rangle, \{\}_u))$

proof –

have $\text{peri}_R(b \&_C P) = ((\lceil b \rceil_{S<} \Rightarrow_r \text{pre}_R P) \Rightarrow_r (\text{peri}_R P \triangleleft \lceil b \rceil_{S<} \triangleright (\$tr' =_u \$tr)))$
 by (simp add: assms Guard-tri-design rea-peri-RHS-design usubst unrest R1-rea-impl R2c-rea-not R2c-rea-impl R2c-preR R2c-periR R2c-tr'-minus-tr R2c-lift-state-pre R2c-condr closure Healthy-if R1-cond R1-tr'-eq-tr)

also have ... = $(\text{pre}_R P \Rightarrow_r \text{peri}_R P) \triangleleft \lceil b \rceil_{S<} \triangleright (\$tr' =_u \$tr)$

by (rel-auto)

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

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

finally show ?thesis

by rel-auto

qed

lemma *postR-Guard [rdes]:*

assumes *P is NCSP*

shows $\text{post}_R(b \&_C P) = (\lceil b \rceil_{S<} \wedge \text{post}_R P)$

proof –

have $\text{post}_R(b \&_C P) = ((\lceil b \rceil_{S<} \Rightarrow_r \text{pre}_R P) \Rightarrow_r (\lceil b \rceil_{S<} \wedge \text{post}_R P))$

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 assms)

also have ... = $(\lceil b \rceil_{S<} \wedge (\text{pre}_R P \Rightarrow_r \text{post}_R P))$

by (rel-auto)

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

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

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

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 $b \&_C P$ is CSP3

proof –

from *assms* have $1:\$ref \nmid P \llbracket \text{false}/\$wait \rrbracket$

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

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

lemma *CSP4-Guard [closure]*:

assumes P is NCSP
 shows $b \&_C P$ is CSP4
proof (rule CSP4-tri-intro[OF CSP-Guard])
 show $(\neg_r pre_R (b \&_C P)) ;; R1 \text{ true} = (\neg_r pre_R (b \&_C P))$
proof –
 have $a: (\neg_r pre_R P) ;; R1 \text{ true} = (\neg_r pre_R P)$
 by (simp add: CSP4-neg-pre-unit assms closure)
 have $(\neg_r ([b]_{S<} \Rightarrow_r pre_R P)) ;; R1 \text{ true} = (\neg_r ([b]_{S<} \Rightarrow_r pre_R P))$
proof –
 have $1: (\neg_r ([b]_{S<} \Rightarrow_r pre_R P)) = ([b]_{S<} \wedge (\neg_r pre_R P))$
 by (rel-auto)
 also have $2: \dots = ([b]_{S<} \wedge ((\neg_r pre_R P) ;; R1 \text{ true}))$
 by (simp add: a)
 also have $3: \dots = (\neg_r ([b]_{S<} \Rightarrow_r pre_R P)) ;; R1 \text{ true}$
 by (rel-auto)
 finally show ?thesis ..
 qed
 thus ?thesis
 by (simp add: preR-Guard periR-Guard NSRD-CSP4-intro closure assms unrest)
 qed
 show $\$st' \# peri_R (b \&_C P)$
 by (simp add: preR-Guard periR-Guard NSRD-CSP4-intro closure assms unrest)
 show $\$ref' \# post_R (b \&_C P)$
 by (simp add: preR-Guard postR-Guard NSRD-CSP4-intro closure assms unrest)
 qed

lemma *NCSP-Guard [closure]*:

assumes P is NCSP
 shows $b \&_C P$ is NCSP
proof –
 have P is CSP
 using NCSP-implies-CSP assms by blast
 then show ?thesis
 by (metis (no-types) CSP3-Guard CSP3-commutes-CSP4 CSP4-Guard CSP4-Idempotent CSP-Guard
 Healthy-Idempotent Healthy-def NCSP-def assms comp-apply)
 qed

lemma *Productive-Guard [closure]*:

assumes P is CSP P is Productive $\$wait' \# pre_R(P)$
 shows $b \&_C P$ is Productive
proof –
 have $b \&_C P = b \&_C \mathbf{R}_s(pre_R(P) \vdash peri_R(P) \diamond (post_R(P) \wedge \$tr <_u \$tr'))$
 by (metis Healthy-def Productive-form assms(1) assms(2))
 also have $\dots =$
 $\mathbf{R}_s (([b]_{S<} \Rightarrow_r pre_R P) \vdash$

$((pre_R P \Rightarrow_r peri_R P) \triangleleft [b]_{S<} \triangleright (\$tr' =_u \$tr)) \diamond ([b]_{S<} \wedge (pre_R P \Rightarrow_r post_R P \wedge \$tr' >_u \$tr)))$
by (*simp add: Guard-tri-design rea-pre-RHS-design rea-peri-RHS-design rea-post-RHS-design unrest*
assms
usubst R1-preR Healthy-if R1-rea-impl R1-peri-SRD R1-extend-conj' R2c-preR R2c-not R2c-rea-impl
R2c-periR R2c-postR R2c-and R2c-tr-less-tr' R1-tr-less-tr')
also have ... = $\mathbf{R}_s (([b]_{S<} \Rightarrow_r pre_R P) \vdash (peri_R P \triangleleft [b]_{S<} \triangleright (\$tr' =_u \$tr)) \diamond ([b]_{S<} \wedge post_R P) \wedge \$tr' >_u \$tr)$
by (*rel-auto*)
also have ... = *Productive*($b \ \&_C \ P$)
by (*simp add: Productive-def Guard-tri-design RHS-tri-design-par unrest*)
finally show *?thesis*
by (*simp add: Healthy-def'*)
qed

lemma *Guard-refines-sinv*:

assumes *P is NCSP* $sinv_R(b) \sqsubseteq P$

shows $sinv_R(b) \sqsubseteq g \ \&_C \ P$

proof –

from *assms*

have $\mathbf{R}_s([b]_{S<} \vdash R1 \ true \diamond [b]_{S>}) \sqsubseteq \mathbf{R}_s(pre_R(P) \vdash peri_R(P) \diamond post_R(P))$

by (*simp add: rdes-def NCSP-implies-CSP SRD-reactive-tri-design*)

thus *?thesis*

apply (*simp add: RHS-tri-design-refine' closure unrest assms*)

apply (*safe*)

apply (*rdes-refine cls: assms(1)*)

done

qed

8.11 Basic events

definition $do_u ::$

$(\sigma, \sigma') \text{ uexpr} \Rightarrow (\sigma, \sigma') \text{ action}$ **where**

$[upred-defs]: do_u \ e = ((\$tr' =_u \$tr \wedge [e]_{S<} \notin_u \$ref') \triangleleft \$wait' \triangleright (\$tr' =_u \$tr \wedge [e]_{S<} \wedge \$st' =_u \$st))$

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

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

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

by (*rel-auto*)

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

by (*rel-auto*)

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

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

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

apply (*rel-auto*)

done

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

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

by (*pred-auto*)**+**

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

$do_C(a) = \mathbf{R}_s(true_r \vdash (\mathcal{E}(true, \langle \rangle, \{a\}_u) \diamond \Phi(true, id, \langle a \rangle)))$
by (*simp add: DoCSP-def do_u-def wait'-cond-def, rel-auto*)

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

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

lemma *preR-DoCSP* [*rdes*]: $pre_R(do_C(a)) = true_r$

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

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

by (*rel-auto*)

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

by (*rel-auto*)

lemma *CSP3-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-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*]:

($do_C a :: ('σ, 'ψ) \text{ action}$) is *Productive*

proof –

have ($(\Phi(true, id, \langle a \rangle) \wedge \$tr' >_u \$tr) :: ('σ, 'ψ) \text{ action}$)
 $= (\Phi(true, id, \langle a \rangle))$

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*]:

($do_C a :: ('σ, 'ψ) \text{ action}$) is *PCSP*

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

lemma *wp-rea-DoCSP-lemma*:

fixes $P :: ('σ, 'φ) \text{ action}$

assumes $\$ok \nmid P \ \$wait \nmid P$

shows $(\$tr' =_u \$tr \wedge_u \langle [a]_{S<} \rangle \wedge \$st' =_u \$st) ;; P = (\exists \$ref \cdot P[\$tr \wedge_u \langle [a]_{S<} \rangle / \$tr])$

using *assms*

by (*rel-auto, meson*)

lemma *wp-rea-DoCSP*:

assumes P is *NCSP*

shows $(\$tr' =_u \$tr \wedge_u \langle [a]_{S<} \rangle \wedge \$st' =_u \$st) \text{ wp}_r \text{ pre}_R P =$

$(\neg_r (\neg_r \text{pre}_R P)[\$tr \wedge_u \langle [a]_{S<} \rangle / \$tr])$

by (simp add: wp-rea-def wp-rea-DoCSP-lemma unrest usubst ex-unrest assms closure)

lemma wp-rea-DoCSP-alt:

assumes P is NCSP

shows $(\$tr' =_u \$tr \hat{\ }_u \langle [a]_{S<} \rangle \wedge \$st' =_u \$st) \text{ wp}_r \text{ pre}_R P =$
 $(\$tr' \geq_u \$tr \hat{\ }_u \langle [a]_{S<} \rangle \Rightarrow_r (\text{pre}_R P) [\$tr \hat{\ }_u \langle [a]_{S<} \rangle / \$tr])$

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

lemma DoCSP-refine-sinv: $\text{sinv}_R(b) \sqsubseteq \text{do}_C(a)$

by (rdes-refine)

8.12 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]: PrefixCSP $a \ P = (\text{do}_C(a) ;; \text{CSP}(P))$

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

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

by (simp add: PrefixCSP-def closure)

lemma CSP3-PrefixCSP [closure]:

PrefixCSP $a \ P$ is CSP3

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

lemma CSP4-PrefixCSP [closure]:

assumes P is CSP P is CSP4

shows PrefixCSP $a \ P$ is CSP4

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

lemma NCSP-PrefixCSP [closure]:

assumes P is NCSP

shows PrefixCSP $a \ P$ is NCSP

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

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

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

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

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

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

proof –

have PrefixCSP $a = (\lambda X. \text{do}_C(a) ;; \text{CSP}(X))$

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 $a \in \llbracket H \rrbracket_H \rightarrow \llbracket \text{CSP} \rrbracket_H$

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:*
 $R1 \ (R2s \ (\$tr' =_u \$tr \hat{\ }_u \langle \lceil a \rceil_{S<} \rangle \wedge \lceil II \rceil_R)) = (\$tr' =_u \$tr \hat{\ }_u \langle \lceil a \rceil_{S<} \rangle \wedge \lceil II \rceil_R)$
by (*rel-auto*)

lemma *PrefixCSP-RHS-tri-lemma2:*
fixes $P :: ('σ, 'φ) \text{ action}$
assumes $\$ok \# P \ \$wait \# P$
shows $((\$tr' =_u \$tr \hat{\ }_u \langle \lceil a \rceil_{S<} \rangle \wedge \$st' =_u \$st) \wedge \neg \$wait') ;; P = (\exists \$ref \cdot P[\$tr \hat{\ }_u \langle \lceil a \rceil_{S<} \rangle / \$tr])$
using *assms*
by (*rel-auto, meson, fastforce*)

lemma *tr-extend-seqr:*
fixes $P :: ('σ, 'φ) \text{ action}$
assumes $\$ok \# P \ \$wait \# P \ \$ref \# P$
shows $(\$tr' =_u \$tr \hat{\ }_u \langle \lceil a \rceil_{S<} \rangle \wedge \$st' =_u \$st) ;; P = P[\$tr \hat{\ }_u \langle \lceil a \rceil_{S<} \rangle / \$tr]$
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 $P \text{ is NCSP}$
shows $pre_R(PrefixCSP \ a \ P) = (\Phi(true, id, \langle a \rangle) \ wp_r \ pre_R \ P)$
by (*simp add: PrefixCSP-def assms closure rdes rpred Healthy-if wp usubst unrest*)

lemma *PrefixCSP-RHS-tri:*
assumes $P \text{ is NCSP}$
shows $PrefixCSP \ a \ P = \mathbf{R}_s (\Phi(true, id, \langle a \rangle) \ wp_r \ pre_R \ P \vdash (\mathcal{E}(true, \langle \rangle, \{a\}_u) \vee \Phi(true, id, \langle a \rangle) ;; peri_R \ P) \diamond \Phi(true, id, \langle a \rangle) ;; post_R \ P)$
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 $P \text{ is CRC } Q \text{ is CRR } R \text{ is CRR}$
 $\$st' \# Q \ \$ref' \# R$
shows $PrefixCSP \ a \ (\mathbf{R}_s (P \vdash Q \diamond R)) = \mathbf{R}_s (\Phi(true, id, \langle a \rangle) \ wp_r \ P \vdash (\mathcal{E}(true, \langle \rangle, \{a\}_u) \vee \Phi(true, id, \langle a \rangle) ;; Q) \diamond \Phi(true, id, \langle a \rangle) ;; R)$
by (*simp add: PrefixCSP-def Healthy-if assms closure, rdes-simp cls: assms*)

8.13 Guarded external choice

abbreviation *GuardedChoiceCSP* $:: 'θ \text{ set} \Rightarrow ('θ \Rightarrow ('σ, 'φ) \text{ action}) \Rightarrow ('σ, 'φ) \text{ action}$ **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 $(\Box x \in A \rightarrow P(x)) =$
 $\mathbf{R}_s ((\Box x \in A \cdot \Phi(\text{true}, \text{id}, \langle \ll x \gg \rangle) \text{ wp}_r \text{ pre}_R (P x)) \vdash$
 $((\Box x \in A \cdot \mathcal{E}(\text{true}, \langle \rangle, \{\ll x \gg\}_u)) \triangleleft \$tr' =_u \$tr \triangleright (\Box x \in A \cdot \Phi(\text{true}, \text{id}, \langle \ll x \gg \rangle) ;; \text{peri}_R$
 $(P x))) \diamond$
 $(\Box x \in A \cdot \Phi(\text{true}, \text{id}, \langle \ll x \gg \rangle) ;; \text{post}_R (P x)))$
by (simp add: PrefixCSP-RHS-tri assms ExtChoice-tri-rdes closure unrest, rel-auto)

8.14 Input prefix

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

definition *InputVarCSP* :: $('a, ' \vartheta) \text{ chan} \Rightarrow ('a \Longrightarrow ' \sigma) \Rightarrow ('a \Rightarrow ' \sigma \text{ upred}) \Rightarrow (' \sigma, ' \vartheta) \text{ action}$ **where**
[upred-defs, rdes-def]: *InputVarCSP* $c x A = \text{InputCSP } c A (\lambda v. \langle [x \mapsto_s \ll v \gg] \rangle_C)$

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

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

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

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

lemma *R4-st-pred-conj-do* [rpred]:
 $((R4 [s_1]_{S<} \wedge \Phi(s_2, \sigma, t) ;; P) = R4(\Phi(s_1 \wedge s_2, \sigma, t) ;; P)$
by (rel-auto)

lemma *unrest-ref'-R4* [unrest]: $\$ref' \# P \Longrightarrow \$ref' \# R4(P)$
by (simp add: R4-def unrest)

lemma *st-pred-conj-seq* [rpred]:
 $\ll P \text{ is RR}; Q \text{ is RR} \rrangle \Longrightarrow ([s]_{S<} \wedge P ;; Q) = (([s]_{S<} \wedge P) ;; Q)$
by (metis (no-types, lifting) R1-seqr-closure RR-implies-R1 cond-st-distr cond-st-miracle seqr-left-zero)

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

assumes $\bigwedge v. P(v) \text{ is CRC} \bigwedge v. Q(v) \text{ is CRR} \bigwedge v. R(v) \text{ is CRR}$
 $\bigwedge v. \$st' \# Q(v) \bigwedge v. \$ref' \# R(v)$
shows $InputCSP\ a\ A\ (\lambda v. \mathbf{R}_s(P(v) \vdash Q(v) \diamond R(v))) =$
 $\mathbf{R}_s((\bigsqcup x \cdot \Phi(A\ x, id, \langle (a \cdot \ll x \gg)_u \rangle))\ wp_r\ P\ x) \vdash$
 $((\bigsqcup x \cdot \mathcal{E}(A\ x, \langle \rangle, \{(a \cdot \ll x \gg)_u\}_u) \vee \mathcal{E}(\neg A\ x, \langle \rangle, \{\}_u)) \vee (\bigsqcup x \cdot \Phi(A\ x, id, \langle (a \cdot \ll x \gg)_u \rangle) ;; Q\ x))$
 \diamond
 $(\bigsqcup x \cdot \Phi(A\ x, id, \langle (a \cdot \ll x \gg)_u \rangle) ;; R\ x))$
by (*simp add: InputCSP-def, rdes-simp cls: assms*)

8.15 Renaming

definition *RenameCSP* :: $(s, e) \text{ action} \Rightarrow (e \Rightarrow f) \Rightarrow (s, f) \text{ action}$ $((-)\downarrow_C [999, 0] 999)$ **where**
 $[upred-defs]: \text{RenameCSP}\ P\ f = \mathbf{R}_s((\neg_r (\neg_r\ pre_R(P))\downarrow_c ;; true_r) \vdash ((peri_R(P))\downarrow_c) \diamond ((post_R(P))\downarrow_c))$

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

assumes $P \text{ is CRC } Q \text{ is CRR } R \text{ is CRR}$
shows $(\mathbf{R}_s(P \vdash Q \diamond R))\downarrow_c = \mathbf{R}_s((\neg_r (\neg_r\ P)\downarrow_c ;; true_r) \vdash Q\downarrow_c \diamond R\downarrow_c)$ (**is** $?lhs = ?rhs$)
proof –
have $?lhs = \mathbf{R}_s((\neg_r (\neg_r\ P)\downarrow_c ;; true_r) \vdash (P \Rightarrow_r Q)\downarrow_c \diamond (P \Rightarrow_r R)\downarrow_c)$
by (*simp add: RenameCSP-def rdes closure assms*)
also have $\dots = \mathbf{R}_s((\neg_r (\neg_r\ CRC(P))\downarrow_c ;; true_r) \vdash (CRC(P) \Rightarrow_r CRR(Q))\downarrow_c \diamond (CRC(P) \Rightarrow_r CRR(R))\downarrow_c)$
by (*simp add: Healthy-if assms*)
also have $\dots = \mathbf{R}_s((\neg_r (\neg_r\ CRC(P))\downarrow_c ;; true_r) \vdash (CRR(Q))\downarrow_c \diamond (CRR(R))\downarrow_c)$
by (*rel-auto, (metis order-refl)+*)
also have $\dots = ?rhs$
by (*simp add: Healthy-if assms*)
finally show $?thesis$.
qed

lemma *RenameCSP-pre-CRC-closed*:

assumes $P \text{ is CRR}$
shows $\neg_r (\neg_r\ P)\downarrow_c ;; R1\ true \text{ is CRC}$
apply (*rule CRC-intro''*)
apply (*simp add: unrest closure assms*)
apply (*simp add: Healthy-def, simp add: RC1-def rpred closure CRC-idem assms segr-assoc*)
done

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

assumes $P \text{ is NCSP}$
shows $P\downarrow_C \text{ is NCSP}$
by (*simp add: RenameCSP-def RenameCSP-pre-CRC-closed closure assms unrest*)

lemma *cspr-rename-false* [*rpred*]:

$false\downarrow_c = false$
by (*rel-auto*)

lemma *umap-nil* [*simp*]: $map_u\ f\ \langle \rangle = \langle \rangle$

by (*rel-auto*)

lemma *rename-Skip*: $Skip\downarrow_C = Skip$

by (*rdes-eq*)

lemma *rename-Chaos*: $Chaos\downarrow_C = Chaos$

by (*rdes-eq-split; rel-simp; force*)

lemma *rename-Miracle*: $\text{Miracle}(\lfloor f \rfloor)_C = \text{Miracle}$
by (*rdes-eq*)

lemma *rename-DoCSP*: $(\text{do}_C(a))(\lfloor f \rfloor)_C = \text{do}_C(\llbracket f \rrbracket(a)_a)$
by (*rdes-eq*)

8.16 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*: $\langle \text{id} \rangle_C = \text{Skip}$
by (*rel-auto*)

lemma *Guard-comp*:
 $g \&_C h \&_C P = (g \wedge h) \&_C P$
by (*rule antisym, rel-blast, rel-blast*)

lemma *Guard-false* [*simp*]: $\text{false} \&_C P = \text{Stop}$
by (*simp add: GuardCSP-def Stop-def rpred closure alpha R1-design-R1-pre*)

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

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

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

lemma *Conditional-as-Guard*:
assumes $P \text{ is NCSP}$ $Q \text{ is NCSP}$
shows $P \triangleleft b \triangleright_R Q = b \&_C P \sqcap (\neg b) \&_C Q$
by (*rdes-eq cls: assms; simp add: le-less*)

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

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

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

lemma *PrefixCSP-extChoice-dist*:
assumes $P \text{ is NCSP}$ $Q \text{ is NCSP}$ $R \text{ is NCSP}$

shows $((a \rightarrow_C P) \sqcap (b \rightarrow_C Q)) \parallel R = (a \rightarrow_C P \parallel R) \sqcap (b \rightarrow_C Q \parallel R)$
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 $\sqcap x \in A \rightarrow P(x) \parallel Q = \sqcap x \in A \rightarrow (P(x) \parallel Q)$
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 $(\text{if}_R i \in A \cdot g(i) \rightarrow P(i) \text{ else } Q \text{ fi}) =$
 $(\sqcap i \in A \cdot g(i) \&_C P(i)) \sqcap (\bigwedge i \in A \cdot \neg g(i)) \&_C Q$
proof (*cases A = {}*)
case *True*
then show *?thesis* **by** (*simp add: ExtChoice-empty extChoice-Stop closure assms*)
next
case *False*
show *?thesis*
proof –
have $1: (\sqcap i \in A \cdot g i \rightarrow_R P i) = (\sqcap i \in A \cdot g i \rightarrow_R \mathbf{R}_s(\text{pre}_R(P i) \vdash \text{peri}_R(P i) \diamond \text{post}_R(P i)))$
by (*rule UINF-cong, simp add: NCSP-implies-CSP SRD-reactive-tri-design assms(1)*)
have $2: (\sqcap i \in A \cdot g(i) \&_C P(i)) = (\sqcap i \in A \cdot g(i) \&_C \mathbf{R}_s(\text{pre}_R(P i) \vdash \text{peri}_R(P i) \diamond \text{post}_R(P i)))$
by (*rule ExtChoice-cong, simp add: NCSP-implies-NSRD NSRD-is-SRD SRD-reactive-tri-design assms(1)*)
from *assms(3)* **show** *?thesis*
by (*simp add: AlternateR-def 1 2*)
 $(\text{rdes-eq cls: } \text{assms}(1-2) \text{_simps: False cong: UINF-cong ExtChoice-cong})$
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*]

find-theorems *R4*

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

syntax

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

translations

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

lemma $\mu\text{-CSP-equiv}$:

assumes $\text{Monotonic } F \ F \in \llbracket \text{CSP} \rrbracket_H \rightarrow \llbracket \text{CSP} \rrbracket_H$

shows $(\mu_R F) = (\mu_C F)$

by ($\text{simp add: srd-}\mu\text{-equiv assms comp-def}$)

lemma $\mu\text{-CSP-unfold}$:

$P \text{ is CSP} \implies (\mu_C X \cdot P ;; X) = P ;; (\mu_C X \cdot P ;; X)$

apply (subst gfp-unfold)

apply ($\text{simp-all add: closure Healthy-if}$)

done

lemma $\mu\text{-csp-expand } [rdes]$: $(\mu_C ((;;) Q)) = (\mu X \cdot Q ;; \text{CSP } X)$

by ($\text{simp add: comp-def}$)

lemma $\mu\text{-csp-basic-refine}$:

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 ($\text{rule SRD-refine-intro'}$, $\text{simp-all add: closure usubst alpha rpred rdes unrest wp seq-UINF-distr assms}$)

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

proof ($\text{rule UINF-refines'}$)

fix i

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

proof ($\text{induct } i$)

case 0

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

next

case ($\text{Suc } i$)

then show $?case$

by ($\text{meson assms}(6) \text{ assms}(7) \text{ semilattice-sup-class.le-sup-iff upower-inductl}$)

qed

qed

qed

lemma $\text{CRD-}\mu\text{-basic-refine}$:

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

assumes

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

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

$[P \ t \ r]_{S < \llbracket (t, r) \rightarrow (\&tt, \$ref')_u \rrbracket} \sqsubseteq post_R \ Q \ ;;_h [P \ t \ r]_{S < \llbracket (t, r) \rightarrow (\&tt, \$ref')_u \rrbracket}$
shows $[true \vdash P \ trace \ refs \mid R]_C \sqsubseteq (\mu_C \ X \cdot Q \ ;; \ X)$
proof (rule *mu-csp-basic-refine*, *simp-all* add: *msubst-pair* *assms* *closure* *alpha* *rdes* *rpred* *Healthy-if* *R1-false*)
show $[P \ trace \ refs]_{S < \llbracket trace \rightarrow \&tt \rrbracket \llbracket refs \rightarrow \$ref' \rrbracket} \sqsubseteq peri_R \ Q$
using *assms* **by** (*simp* add: *msubst-pair*)
show $[P \ trace \ refs]_{S < \llbracket trace \rightarrow \&tt \rrbracket \llbracket refs \rightarrow \$ref' \rrbracket} \sqsubseteq post_R \ Q \ ;; [P \ trace \ refs]_{S < \llbracket trace \rightarrow \&tt \rrbracket \llbracket refs \rightarrow \$ref' \rrbracket}$
using *assms* **by** (*simp* add: *msubst-pair*)
qed

9.2 Example action expansion

lemma *mu-example1*: $(\mu \ X \cdot \ll a \gg \rightarrow_C \ X) = (\bigcap i \cdot do_C(\ll a \gg) \wedge (i+1)) \ ;; \ Miracle$
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-UINF-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* :: $(' \sigma, ' \varphi) \ action \Rightarrow ' \sigma \Rightarrow ' \varphi \ list \ set \ (dv \llbracket - \rrbracket - [0,100] \ 100)$ **where**
 $[upred-defs]: \text{divergences } P \ s = \{t \mid t. '(\neg_r \ pre_R(P)) \llbracket \ll s \gg, \langle \rangle, \ll t \gg / \$st, \$tr, \$tr' \rrbracket '\}$

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

definition *failures* :: $(' \sigma, ' \varphi) \ action \Rightarrow ' \sigma \Rightarrow (' \varphi \ list \times ' \varphi \ set) \ set \ (fl \llbracket - \rrbracket - [0,100] \ 100)$ **where**
 $[upred-defs]: \text{failures } P \ s = \{(t, r) \mid t \ r. '(\pre_R(P) \wedge \text{peri}_R(P)) \llbracket \ll r \gg, \ll s \gg, \langle \rangle, \ll t \gg / \$ref', \$st, \$tr, \$tr' \rrbracket '\}$

lemma *trace-divergence-disj*:
assumes *P* is NCSP $(t, s') \in tr \llbracket P \rrbracket s \ t \in dv \llbracket P \rrbracket s$

shows *False*
 using *assms(2,3)*
 by (*simp add: traces-def divergences-def, rdes-simp cls:assms, rel-auto*)

lemma *preR-refine-divergences*:

assumes *P is NCSP Q is NCSP* $\wedge s. dv[P]s \subseteq dv[Q]s$
 shows $pre_R(P) \sqsubseteq pre_R(Q)$

proof (*rule CRR-refine-impl-prop, simp-all add: assms closure usubst unrest*)

fix *t s*

assume *a*: $['\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg] \dagger pre_R Q$

with *a* show $['\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg] \dagger pre_R P$

proof (*rule-tac ccontr*)

from *assms(3)[of s]* have *b*: $t \in dv[P]s \implies t \in dv[Q]s$

by (*auto*)

assume $\neg ['\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg] \dagger pre_R P$

hence $\neg ['\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg] \dagger CRC(pre_R P)$

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

hence $['\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg] \dagger (\neg_r CRC(pre_R P))$

by (*rel-auto*)

hence $['\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg] \dagger (\neg_r pre_R P)$

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

with *a b* show *False*

by (*rel-auto*)

qed

qed

lemma *preR-eq-divergences*:

assumes *P is NCSP Q is NCSP* $\wedge s. dv[P]s = dv[Q]s$

shows $pre_R(P) = pre_R(Q)$

by (*metis assms dual-order.antisym order-refl preR-refine-divergences*)

lemma *periR-refine-failures*:

assumes *P is NCSP Q is NCSP* $\wedge s. fl[Q]s \subseteq fl[P]s$

shows $(pre_R(P) \wedge peri_R(P)) \sqsubseteq (pre_R(Q) \wedge peri_R(Q))$

proof (*rule CRR-refine-impl-prop, simp-all add: assms closure unrest subst-unrest-3*)

fix *t s r'*

assume *a*: $['\$ref' \mapsto_s \ll r' \gg, \$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg] \dagger (pre_R Q \wedge peri_R Q)$

from *assms(3)[of s]* have *b*: $(t, r') \in fl[Q]s \implies (t, r') \in fl[P]s$

by (*auto*)

with *a* show $['\$ref' \mapsto_s \ll r' \gg, \$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg] \dagger (pre_R P \wedge peri_R P)$

by (*simp add: failures-def*)

qed

lemma *periR-eq-failures*:

assumes *P is NCSP Q is NCSP* $\wedge s. fl[P]s = fl[Q]s$

shows $(pre_R(P) \wedge peri_R(P)) = (pre_R(Q) \wedge peri_R(Q))$

by (*metis (full-types) assms dual-order.antisym order-refl periR-refine-failures*)

lemma *postR-refine-traces*:

assumes *P is NCSP Q is NCSP* $\wedge s. tr[Q]s \subseteq tr[P]s$

shows $(pre_R(P) \wedge post_R(P)) \sqsubseteq (pre_R(Q) \wedge post_R(Q))$

proof (*rule CRR-refine-impl-prop, simp-all add: assms closure unrest subst-unrest-5*)

fix *t s s'*

assume *a*: $['\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg] \dagger (pre_R Q \wedge post_R Q)$

from *assms(3)[of s]* have *b*: $(t, s') \in tr[Q]s \implies (t, s') \in tr[P]s$

by (auto)
 with a show ‘ $[\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg] \dagger (pre_R P \wedge post_R P)$ ’
 by (simp add: traces-def)
 qed

lemma *postR-eq-traces*:
 assumes P is NCSP Q is NCSP $\wedge s. tr[P]s = tr[Q]s$
 shows $(pre_R(P) \wedge post_R(P)) = (pre_R(Q) \wedge post_R(Q))$
 by (metis assms dual-order.antisym order-refl postR-refine-traces)

lemma *circus-fd-refine-intro*:
 assumes P is NCSP Q is NCSP $\wedge s. dv[Q]s \subseteq dv[P]s \wedge s. fl[Q]s \subseteq fl[P]s \wedge s. tr[Q]s \subseteq tr[P]s$
 shows $P \sqsubseteq Q$
proof (rule SRD-refine-intro', simp-all add: closure assms)
 show a: ‘ $pre_R P \Rightarrow pre_R Q$ ’
 using assms(1) assms(2) assms(3) preR-refine-divergences refBy-order by blast
 show $peri_R P \sqsubseteq (pre_R P \wedge peri_R Q)$
proof –
 have $peri_R P \sqsubseteq (pre_R Q \wedge peri_R Q)$
 by (metis (no-types) assms(1) assms(2) assms(4) periR-refine-failures utp-pred-laws.le-inf-iff)
 then show ?thesis
 by (metis a refBy-order utp-pred-laws.inf.order-iff utp-pred-laws.inf-assoc)
 qed
 show $post_R P \sqsubseteq (pre_R P \wedge post_R Q)$
proof –
 have $post_R P \sqsubseteq (pre_R Q \wedge post_R Q)$
 by (meson assms(1) assms(2) assms(5) postR-refine-traces utp-pred-laws.le-inf-iff)
 then show ?thesis
 by (metis a refBy-order utp-pred-laws.inf.absorb-iff1 utp-pred-laws.inf-assoc)
 qed
 qed

10.2 Circus Operators

lemma *traces-Skip*:
 $tr[Skip]s = \{(\[], s)\}$
 by (simp add: traces-def rdes alpha closure, rel-simp)

lemma *failures-Skip*:
 $fl[Skip]s = \{\}$
 by (simp add: failures-def, rdes-calc)

lemma *divergences-Skip*:
 $dv[Skip]s = \{\}$
 by (simp add: divergences-def, rdes-calc)

lemma *traces-Stop*:
 $tr[Stop]s = \{\}$
 by (simp add: traces-def, rdes-calc)

lemma *failures-Stop*:
 $fl[Stop]s = \{(\[], E) \mid E. True\}$
 by (simp add: failures-def, rdes-calc, rel-auto)

lemma *divergences-Stop*:
 $dv[Stop]s = \{\}$

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

lemma *traces-AssignsCSP*:
 $tr\llbracket \langle \sigma \rangle_C \rrbracket s = \{(\llbracket \cdot \rrbracket, \sigma(s))\}$
 by (simp add: traces-def rdes closure usubst alpha, rel-auto)

lemma *failures-AssignsCSP*:
 $fl\llbracket \langle \sigma \rangle_C \rrbracket s = \{\}$
 by (simp add: failures-def, rdes-calc)

lemma *divergences-AssignsCSP*:
 $dv\llbracket \langle \sigma \rangle_C \rrbracket s = \{\}$
 by (simp add: divergences-def, rdes-calc)

lemma *failures-Miracle*: $fl\llbracket Miracle \rrbracket s = \{\}$
 by (simp add: failures-def rdes closure usubst)

lemma *divergences-Miracle*: $dv\llbracket Miracle \rrbracket s = \{\}$
 by (simp add: divergences-def rdes closure usubst)

lemma *failures-Chaos*: $fl\llbracket Chaos \rrbracket s = \{\}$
 by (simp add: failures-def rdes, rel-auto)

lemma *divergences-Chaos*: $dv\llbracket Chaos \rrbracket s = UNIV$
 by (simp add: divergences-def rdes, rel-auto)

lemma *traces-Chaos*: $tr\llbracket Chaos \rrbracket s = \{\}$
 by (simp add: traces-def rdes closure usubst)

lemma *divergences-cond*:
 assumes P is NCSP Q is NCSP
 shows $dv\llbracket P \triangleleft b \triangleright_R Q \rrbracket s = (if (\llbracket b \rrbracket_e s) then dv\llbracket P \rrbracket s else dv\llbracket Q \rrbracket s)$
 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 $tr\llbracket P \triangleleft b \triangleright_R Q \rrbracket s = (if (\llbracket b \rrbracket_e s) then tr\llbracket P \rrbracket s else tr\llbracket Q \rrbracket s)$
 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 $fl\llbracket P \triangleleft b \triangleright_R Q \rrbracket s = (if (\llbracket b \rrbracket_e s) then fl\llbracket P \rrbracket s else fl\llbracket Q \rrbracket s)$
 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 $dv\llbracket g \&_C P \rrbracket s = (if (\llbracket g \rrbracket_e s) then dv\llbracket g \&_C P \rrbracket s else \{\})$
 by (rdes-simp cls: assms, simp add: divergences-def traces-def rdes closure rpred assms, rel-auto)

lemma *traces-do*: $tr\llbracket do_C(e) \rrbracket s = \{(\llbracket e \rrbracket_e s, s)\}$
 by (rdes-simp, simp add: traces-def rdes closure rpred, rel-auto)

lemma *failures-do*: $fl\llbracket do_C(e) \rrbracket s = \{(\llbracket \cdot \rrbracket, E) \mid E. \llbracket e \rrbracket_e s \notin E\}$
 by (rdes-simp, simp add: failures-def rdes closure rpred usubst, rel-auto)

lemma *divergences-do*: $dv\llbracket do_C(e) \rrbracket s = \{\}$
by (*rel-auto*)

lemma *divergences-seq*:
fixes $P :: ('s, 'e) \text{ action}$
assumes $P \text{ is NCSP } Q \text{ is NCSP}$
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 $tr\llbracket P ;; Q \rrbracket s =$

$$\{(t_1 @ t_2, s') \mid t_1 \ t_2 \ s_0 \ s'. (t_1, s_0) \in tr\llbracket P \rrbracket s \wedge (t_2, s') \in tr\llbracket Q \rrbracket s_0$$

$$\wedge (t_1 @ t_2) \notin dv\llbracket P \rrbracket s$$

$$\wedge (\forall (t, s_1) \in tr\llbracket P \rrbracket s. t \leq t_1 @ t_2 \longrightarrow (t_1 @ t_2) - t \notin dv\llbracket Q \rrbracket s_1) \}$$

(is ?lhs = ?rhs)

proof

show $?lhs \subseteq ?rhs$

proof (*rdes-expand cls: assms, simp add: traces-def divergences-def rdes closure assms rdes-def unrest rpred usubst, auto*)

fix $t :: 'e \text{ list}$ **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

$a1: '? \sigma \dagger (post_R P ;; post_R Q)'$ **and**

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

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

from $a1$ **have** $'? \sigma \dagger (\exists tr_0. ((post_R P) \llbracket \ll tr_0 \gg / \$tr' \rrbracket ;; (post_R Q) \llbracket \ll tr_0 \gg / \$tr \rrbracket) \wedge \ll tr_0 \gg \leq_u \$tr')$
by (*simp add: R2-tr-middle assms closure*)

then obtain tr_0 **where** $p1: '? \sigma \dagger ((post_R P) \llbracket \ll tr_0 \gg / \$tr' \rrbracket ;; (post_R Q) \llbracket \ll tr_0 \gg / \$tr \rrbracket)'$ **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** $'? \sigma \dagger (\exists st_0. (post_R P) \llbracket \ll tr_0 \gg / \$tr' \rrbracket \llbracket \ll st_0 \gg / \$st' \rrbracket ;; (post_R Q) \llbracket \ll tr_0 \gg / \$tr \rrbracket \llbracket \ll st_0 \gg / \$st \rrbracket)'$
by (*simp add: seqr-middle[of st, THEN sym]*)

then obtain s_0 **where** $'? \sigma \dagger ((post_R P) \llbracket \ll s_0 \gg, \ll tr_0 \gg / \$st', \$tr' \rrbracket ;; (post_R Q) \llbracket \ll s_0 \gg, \ll tr_0 \gg / \$st, \$tr \rrbracket)'$

apply (*simp add: usubst*)

apply (*erule taut-shEx-elim*)

apply (*simp add: unrest-all-circus-vars-st-st' closure unrest assms*)

apply (*rel-auto*)

done

hence $'(([\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll tr_0 \gg] \dagger post_R P) ;;$
 $([\$st \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \ll tr_0 \gg, \$tr' \mapsto_s \ll t \gg] \dagger post_R Q))'$

by (*rel-auto*)

hence $'(([\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll tr_0 \gg] \dagger post_R P) \wedge$
 $([\$st \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \ll tr_0 \gg, \$tr' \mapsto_s \ll t \gg] \dagger post_R Q))'$

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

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

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

by (*rel-auto*)

from $postQ'$ **have** $'[\$st \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg] \dagger [\$tr \mapsto_s \ll tr_0 \gg, \$tr' \mapsto_s \ll tr_0 \gg + (\ll t \gg -$

$\llbracket tr_0 \rrbracket \vdash post_R Q'$
using $tr0$ **by** (*rel-auto*)
hence $[\$st \mapsto_s \llbracket s_0 \rrbracket, \$st' \mapsto_s \llbracket s' \rrbracket] \vdash [\$tr \mapsto_s 0, \$tr' \mapsto_s \llbracket t \rrbracket - \llbracket tr_0 \rrbracket] \vdash post_R Q'$
by (*simp add: R2-subst-tr closure assms*)
hence $postQ: [\$st \mapsto_s \llbracket s_0 \rrbracket, \$st' \mapsto_s \llbracket s' \rrbracket, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \llbracket t - tr_0 \rrbracket] \vdash post_R Q'$
by (*rel-auto*)
have $preP: [\$st \mapsto_s \llbracket s \rrbracket, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \llbracket tr_0 \rrbracket] \vdash pre_R P'$
proof –
have $(pre_R P) \llbracket 0, \llbracket tr_0 \rrbracket / \$tr, \$tr' \rrbracket \sqsubseteq (pre_R P) \llbracket 0, \llbracket t \rrbracket / \$tr, \$tr' \rrbracket$
by (*simp add: RC-prefix-refine closure assms tr0*)
hence $[\$st \mapsto_s \llbracket s \rrbracket, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \llbracket tr_0 \rrbracket] \vdash pre_R P \sqsubseteq [\$st \mapsto_s \llbracket s \rrbracket, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \llbracket t \rrbracket] \vdash pre_R P$
by (*rel-auto*)
thus *?thesis*
by (*simp add: taut-refine-impl a2*)
qed

have $preQ: [\$st \mapsto_s \llbracket s_0 \rrbracket, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \llbracket t - tr_0 \rrbracket] \vdash pre_R Q'$
proof –
from $postP$ *a3* **have** $[\$st \mapsto_s \llbracket s_0 \rrbracket, \$tr \mapsto_s \llbracket tr_0 \rrbracket, \$tr' \mapsto_s \llbracket t \rrbracket] \vdash pre_R Q'$
apply (*simp add: wp-rea-def*)
apply (*rel-auto*)
using $tr0$ **apply** $blast+$
done
hence $[\$st \mapsto_s \llbracket s_0 \rrbracket] \vdash [\$tr \mapsto_s \llbracket tr_0 \rrbracket, \$tr' \mapsto_s \llbracket tr_0 \rrbracket + (\llbracket t \rrbracket - \llbracket tr_0 \rrbracket)] \vdash pre_R Q'$
by (*rel-auto*)

hence $[\$st \mapsto_s \llbracket s_0 \rrbracket] \vdash [\$tr \mapsto_s 0, \$tr' \mapsto_s \llbracket t \rrbracket - \llbracket tr_0 \rrbracket] \vdash pre_R Q'$
by (*simp add: R2-subst-tr closure assms*)
thus *?thesis*
by (*rel-auto*)
qed

from $a2$ **have** $ndiv: \neg [\$st \mapsto_s \llbracket s \rrbracket, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \llbracket t \rrbracket] \vdash (\neg_r pre_R P)'$
by (*rel-auto*)

have $t_minus_tr0: tr_0 @ (t - tr_0) = t$
using $append_minus$ $tr0$ **by** $blast$

from $a3$

have $wpr: \bigwedge t_0 s_1.$

$[\$st \mapsto_s \llbracket s \rrbracket, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \llbracket t_0 \rrbracket] \vdash pre_R P' \implies$
 $[\$st \mapsto_s \llbracket s \rrbracket, \$st' \mapsto_s \llbracket s_1 \rrbracket, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \llbracket t_0 \rrbracket] \vdash post_R P' \implies$
 $t_0 \leq t \implies [\$st \mapsto_s \llbracket s_1 \rrbracket, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \llbracket t - t_0 \rrbracket] \vdash (\neg_r pre_R Q)' \implies False$

proof –

fix $t_0 s_1$

assume $b:$

$[\$st \mapsto_s \llbracket s \rrbracket, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \llbracket t_0 \rrbracket] \vdash pre_R P'$
 $[\$st \mapsto_s \llbracket s \rrbracket, \$st' \mapsto_s \llbracket s_1 \rrbracket, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \llbracket t_0 \rrbracket] \vdash post_R P'$
 $t_0 \leq t$
 $[\$st \mapsto_s \llbracket s_1 \rrbracket, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \llbracket t - t_0 \rrbracket] \vdash (\neg_r pre_R Q)'$

from $a3$ **have** $c: \forall (s_0, t_0) \cdot \llbracket t_0 \rrbracket \leq_u \llbracket t \rrbracket$

$\wedge [\$st \mapsto_s \llbracket s \rrbracket, \$st' \mapsto_s \llbracket s_0 \rrbracket, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \llbracket t_0 \rrbracket] \vdash post_R P$
 $\implies [\$st \mapsto_s \llbracket s_0 \rrbracket, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \llbracket t \rrbracket - \llbracket t_0 \rrbracket] \vdash pre_R Q'$

```

    by (simp add: wp-rea-circus-form-alt[of postR P preR Q] closure assms unrest usubst)
      (rel-simp)

from c b(2-4) show False
  by (rel-auto)
qed

show  $\exists t_1 t_2.$ 
   $t = t_1 @ t_2 \wedge$ 
   $(\exists s_0. \text{'[\$st} \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_1 \gg] \dagger pre_R P \wedge$ 
     $[\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_1 \gg] \dagger post_R P' \wedge$ 
     $\text{'[\$st} \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_2 \gg] \dagger pre_R Q \wedge$ 
     $[\$st \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_2 \gg] \dagger post_R Q' \wedge$ 
     $\neg \text{'[\$st} \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_1 @ t_2 \gg] \dagger (\neg_r pre_R P)' \wedge$ 
     $(\forall t_0 s_1. \text{'[\$st} \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_0 \gg] \dagger pre_R P \wedge$ 
       $[\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_0 \gg] \dagger post_R P' \longrightarrow$ 
       $t_0 \leq t_1 @ t_2 \longrightarrow \neg \text{'[\$st} \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll (t_1 @ t_2) - t_0 \gg] \dagger (\neg_r$ 
     $pre_R Q)'})$ 
  apply (rule-tac x=tr0 in exI)
  apply (rule-tac x=(t - tr0) in exI)
  apply (auto)
  using tr0 apply auto[1]
  apply (rule-tac x=s0 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 t1 t2 :: 'e list and s0 s' :: 's
  assume
    a1:  $\neg \text{'[\$st} \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_1 @ t_2 \gg] \dagger (\neg_r pre_R P)'$  and
    a2:  $\text{'[\$st} \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_1 \gg] \dagger pre_R P'$  and
    a3:  $\text{'[\$st} \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_1 \gg] \dagger post_R P'$  and
    a4:  $\text{'[\$st} \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_2 \gg] \dagger pre_R Q'$  and
    a5:  $\text{'[\$st} \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_2 \gg] \dagger post_R Q'$  and
    a6:  $\forall t s_1. \text{'[\$st} \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg] \dagger pre_R P \wedge$ 
       $[\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg] \dagger post_R P' \longrightarrow$ 
       $t \leq t_1 @ t_2 \longrightarrow \neg \text{'[\$st} \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll (t_1 @ t_2) - t \gg] \dagger (\neg_r pre_R Q)'$ 

  from a1 have preP:  $\text{'[\$st} \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_1 @ t_2 \gg] \dagger (pre_R P)'$ 
    by (simp add: taut-not unrest-all-circus-vars-st assms closure unrest, rel-auto)

  have  $\text{'[\$st} \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \ll t_1 \gg, \$tr' \mapsto_s \ll t_1 \gg + \ll t_2 \gg] \dagger post_R Q'$ 
  proof -
    have  $[\$st \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_2 \gg] \dagger post_R Q =$ 
       $[\$st \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg] \dagger [\$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_2 \gg] \dagger post_R Q$ 
    by rel-auto
    also have ... =  $[\$st \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg] \dagger [\$tr \mapsto_s \ll t_1 \gg, \$tr' \mapsto_s \ll t_1 \gg + \ll t_2 \gg] \dagger post_R Q$ 
    by (simp add: R2-subst-tr assms closure, rel-auto)
    finally show ?thesis using a5
    by (rel-auto)
  qed
with a3

```

have $postPQ$: $['\$st \mapsto_s \langle s \rangle, \$st' \mapsto_s \langle s' \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle t_1 @ t_2 \rangle] \dagger (post_R P ;; post_R Q)'$
by (*rel-auto*, *meson Prefix-Order.prefixI*)

have $['\$st \mapsto_s \langle s_0 \rangle, \$tr \mapsto_s \langle t_1 \rangle, \$tr' \mapsto_s \langle t_1 \rangle + \langle t_2 \rangle] \dagger pre_R Q'$
proof –
have $['\$st \mapsto_s \langle s_0 \rangle, \$tr \mapsto_s \langle t_1 \rangle, \$tr' \mapsto_s \langle t_1 \rangle + \langle t_2 \rangle] \dagger pre_R Q =$
 $['\$st \mapsto_s \langle s_0 \rangle] \dagger ['$tr \mapsto_s \langle t_1 \rangle, \$tr' \mapsto_s \langle t_1 \rangle + \langle t_2 \rangle] \dagger pre_R Q$
by *rel-auto*
also have $\dots = ['$st \mapsto_s \langle s_0 \rangle] \dagger ['$tr \mapsto_s 0, \$tr' \mapsto_s \langle t_2 \rangle] \dagger pre_R Q$
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 \langle s \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle t \rangle] \dagger pre_R P'; ['$st \mapsto_s \langle s \rangle,$
 $\$st' \mapsto_s \langle s_1 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle t \rangle] \dagger post_R P' \rrbracket \implies$
 $['$st \mapsto_s \langle s_1 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle (t_1 @ t_2) - t \rangle] \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 \langle s \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle t_1 @ t_2 \rangle] \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 \langle s \rangle, \$st' \mapsto_s \langle s_1 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle t_0 \rangle] \dagger post_R P'$
 \rrbracket
 $\implies ['$st \mapsto_s \langle s_1 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle (t_1 @ t_2) - t_0 \rangle] \dagger pre_R Q'$
proof –
fix $s_1 t_0$
assume $c: t_0 \leq t_1 @ t_2$ $['$st \mapsto_s \langle s \rangle, \$st' \mapsto_s \langle s_1 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle t_0 \rangle] \dagger post_R P'$

have $preP'$: $['$st \mapsto_s \langle s \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle t_0 \rangle] \dagger pre_R P'$
proof –
have $(pre_R P) \llbracket 0, \langle t_0 \rangle / \$tr, \$tr' \rrbracket \sqsubseteq (pre_R P) \llbracket 0, \langle t_1 @ t_2 \rangle / \$tr, \$tr' \rrbracket$
by (*simp add: RC-prefix-refine closure assms c*)
hence $['$st \mapsto_s \langle s \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle t_0 \rangle] \dagger pre_R P \sqsubseteq ['$st \mapsto_s \langle s \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s$
 $\langle t_1 @ t_2 \rangle] \dagger pre_R P$
by (*rel-auto*)
thus *?thesis*
by (*simp add: taut-refine-impl preP*)
qed

with c *a3 preP a6'* $[of\ t_0\ s_1]$ **show** $['$st \mapsto_s \langle s_1 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle (t_1 @ t_2) - t_0 \rangle] \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 '([ $\$st \mapsto_s \langle s \rangle$ ,  $\$tr \mapsto_s \langle \rangle$ ,  $\$tr' \mapsto_s \langle t_1 @ t_2 \rangle$ ]  $\dagger$   $pre_R P \wedge$ 
  [ $\$st \mapsto_s \langle s \rangle$ ,  $\$tr \mapsto_s \langle \rangle$ ,  $\$tr' \mapsto_s \langle t_1 @ t_2 \rangle$ ]  $\dagger$  ( $post_R P \wp_r pre_R Q$ ))  $\wedge$ 
  [ $\$st \mapsto_s \langle s \rangle$ ,  $\$st' \mapsto_s \langle s' \rangle$ ,  $\$tr \mapsto_s \langle \rangle$ ,  $\$tr' \mapsto_s \langle t_1 @ t_2 \rangle$ ]  $\dagger$  ( $post_R P ;; post_R Q$ )'
  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 $tr[\llbracket a \rrbracket s \rightarrow_C P]s = \{(a \# t, s') \mid t s'. (t, s') \in tr[P]s\}$
 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)
 done

lemma *CDF-seq-idem*: $CDF ;; CDF = CDF$
 by (rdes-eq)

lemma *CDF-refine-intro*: $CDF \sqsubseteq P \implies CDF \sqsubseteq (CDF ;; P)$
 by (metis CDF-seq-idem urel-diodid.mult-isol)

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

lemma *CDF-ext-st* [*alpha*]: $CDF \oplus_p abs-st_L = CDF$
 by (rdes-eq)

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.