

Stateful-Failure Reactive Designs in Isabelle/UTP

Simon Foster James Baxter Ana Cavalcanti Jim Woodcock

May 24, 2018

Abstract

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

Contents

1	Introduction	2
2	Stateful-Failure Core Types	2
2.1	SFRD Alphabet	3
2.2	Basic laws	4
2.3	Unrestriction laws	4
3	Stateful-Failure Reactive Relations	5
3.1	Healthiness Conditions	5
3.2	Closure Properties	6
3.3	Introduction laws	12
3.4	Weakest Precondition	13
3.5	Trace Substitution	14
3.6	Initial Interaction	15
3.7	Enabled Events	17
3.8	Completed Trace Interaction	19
3.9	Downward closure of refusals	21
3.10	Renaming	23
4	Stateful-Failure Healthiness Conditions	24
5	Definitions	24
5.1	Healthiness condition properties	25
5.2	CSP theories	36
5.3	Algebraic laws	37

6	Stateful-Failure Reactive Contracts	37
7	External Choice	39
7.1	Definitions and syntax	39
7.2	Basic laws	40
7.3	Algebraic laws	40
7.4	Reactive design calculations	40
7.5	Productivity and Guardedness	48
7.6	Algebraic laws	49
8	Stateful-Failure Programs	52
8.1	Conditionals	52
8.2	Guarded commands	52
8.3	Alternation	53
8.4	While Loops	53
8.5	Iteration Construction	53
8.6	Assignment	54
8.7	Assignment with update	55
8.8	State abstraction	56
8.9	Assumptions	56
8.10	Guards	56
8.11	Basic events	60
8.12	Event prefix	62
8.13	Guarded external choice	64
8.14	Input prefix	65
8.15	Renaming	66
8.16	Algebraic laws	67
9	Recursion in Stateful-Failures	69
9.1	Fixed-points	69
9.2	Example action expansion	70
10	Linking to the Failures-Divergences Model	70
10.1	Failures-Divergences Semantics	70
10.2	Circus Operators	72
10.3	Deadlock Freedom	78
11	Meta-theory for Stateful-Failure Reactive Designs	79

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 $'\varphi$ *csp-vars* = $'\sigma$ *rsp-vars* +
 $ref :: '\varphi$ *set*

declare *csp-vars.defs* [*lens-defs*]
declare *csp-vars.splits* [*alpha-splits*]

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

interpretation *alphabet-csp-prd*:
 $lens_interp \lambda(ok, wait, tr, m). (ok, wait, tr, ref_v m, more m)$
apply (*unfold-locales*)
apply (*rule injI*)
apply (*clarsimp*)
done

interpretation *alphabet-csp-rel*:
 $lens_interp \lambda(ok, ok', wait, wait', tr, tr', m, m').$
 $(ok, ok', wait, wait', tr, tr', ref_v m, ref_v m', more m, more m')$
apply (*unfold-locales*)
apply (*rule injI*)
apply (*clarsimp*)
done

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

type-synonym $(' \sigma, '\varphi)$ *st-csp* = $(' \sigma, '\varphi$ *list*, $(' \varphi, unit)$ *csp-vars-scheme*) *rsp*
type-synonym $(' \sigma, '\varphi)$ *action* = $(' \sigma, '\varphi)$ *st-csp hrel*
type-synonym $'\varphi$ *csp* = $(unit, '\varphi)$ *st-csp*
type-synonym $'\varphi$ *process* = $'\varphi$ *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
 $(type) (' \sigma, '\varphi)$ *st-csp* $\leq (type) (' \sigma, '\varphi$ *list*, $'\varphi 1$ *csp-vars*) *rsp*
 $(type) (' \sigma, '\varphi)$ *action* $\leq (type) (' \sigma, '\varphi)$ *st-csp hrel*
 $(type) '\varphi$ *process* $\leq (type) (unit, '\varphi)$ *action*

notation *csp-vars-child-lens_a* (Σ_c)
notation *csp-vars-child-lens* (Σ_C)

2.2 Basic laws

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

lemma *circus-alpha-bij-lens*:

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

2.3 Unrestriction laws

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

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

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

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

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

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

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

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

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

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

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

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

lemma *unrest-circus-alpha*:

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

assumes

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

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

shows $\Sigma \# P$

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

lemma *unrest-all-circus-vars*:

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

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

shows $\Sigma \# [\$ref' \mapsto_s r', \$st \mapsto_s s, \$st' \mapsto_s s', \$tr \mapsto_s t, \$tr' \mapsto_s t'] \dagger P$
using *assms*
by (*simp add: bij-lens-unrest-all-eq[OF circus-alpha-bij-lens] unrest-plus-split plus-vwb-lens*)
(simp add: unrest usubst closure)

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

fixes $P :: ('s, 'e) \text{ action}$
assumes $\$ok \# P \$ok' \# P \$wait \# P \$wait' \# P \$ref \# P \$ref' \# P \Sigma \# s \Sigma \# s' \Sigma \# t \Sigma \# t'$
shows $\Sigma \# [\$st \mapsto_s s, \$st' \mapsto_s s', \$tr \mapsto_s t, \$tr' \mapsto_s t'] \dagger P$
using *assms*
by (*simp add: bij-lens-unrest-all-eq[OF circus-alpha-bij-lens] unrest-plus-split plus-vwb-lens*)
(simp add: unrest usubst closure)

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

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 $\Sigma \# [\$st \mapsto_s s, \$tr \mapsto_s t, \$tr' \mapsto_s t'] \dagger P$
using *assms*
by (*simp add: bij-lens-unrest-all-eq[OF circus-alpha-bij-lens] unrest-plus-split plus-vwb-lens*)
(simp add: unrest usubst closure)

lemma *unrest-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 $x \# [\$st \mapsto_s s, \$st' \mapsto_s s', \$tr \mapsto_s t, \$tr' \mapsto_s t'] \dagger P$
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 $x \# [\$st \mapsto_s s, \$tr \mapsto_s t, \$tr' \mapsto_s t'] \dagger P$
by (*simp add: unrest-all-var unrest-all-circus-vars-st assms*)

end

3 Stateful-Failure Reactive Relations

theory *utp-sfrd-rel*

imports *utp-sfrd-core*

begin

3.1 Healthiness Conditions

CSP Reactive Relations

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

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

lemma *Idempotent-CRR [closure]*: *Idempotent CRR*
by (*simp add: CRR-idem Idempotent-def*)

lemma *CRR-intro*:

assumes $\$ref \# P$ P is RR
shows P is CRR
by (*simp add: CRR-def Healthy-def, simp add: Healthy-if assms ex-unrest*)

CSP Reactive Conditions

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

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

lemma $CRC\text{-intro}'$:
assumes P is CRR P is RC
shows P is CRC
by (*metis CRC-def CRR-def Healthy-def RC-implies-RR assms*)

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

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

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

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

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

lemma $ex\text{-ref}'\text{-}RR\text{-closed}$ [*closure*]:

assumes P is RR
shows $(\exists \$ref' \cdot P)$ is RR

proof –

have $RR (\exists \$ref' \cdot RR(P)) = (\exists \$ref' \cdot RR(P))$
by (*rel-auto*)

thus *?thesis*

by (*metis Healthy-def assms*)

qed

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

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

3.2 Closure Properties

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

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

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]:
 $P \text{ is CRC} \implies (\llbracket b \rrbracket_{S<} \Rightarrow_r P) \text{ is CRC}$
 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 is CRR
shows $P \oplus_r \text{map-st}_L[a]$ is CRR
by (*rule CRR-intro, simp-all add: closure unrest assms*)

lemma *map-st-ext-CRC-closed* [closure]:
assumes P is CRC
shows $P \oplus_r \text{map-st}_L[a]$ is CRC
by (*rule CRC-intro, simp-all add: closure unrest assms*)

lemma *rdes-frame-ext-CRR-closed* [closure]:
assumes P is CRR
shows $a:[P]_r^+$ 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 is CRC Q is CRC
shows $P \triangleleft b \triangleright_R Q$ is CRC
by (*rule CRC-intro, simp-all add: closure assms unrest*)

lemma *shEx-CRR-closed* [closure]:
assumes $\bigwedge x. P\ x$ is CRR
shows $(\exists x \cdot P(x))$ 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 $\bigwedge i. P\ i$ is CRR
shows $(\bigsqcup i \cdot P(i))$ is CRR
by (*rule CRR-intro, simp-all add: assms unrest closure*)

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

lemma *cond-tt-CRR-closed* [closure]:
assumes P is CRR Q is CRR
shows $P \triangleleft \$tr' =_u \$tr \triangleright Q$ 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 *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]:

$\llbracket \bigwedge i. P i \text{ is CRC} \rrbracket \implies (\bigsqcup i. P i) \text{ is CRC}$
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 $(\$tr' =_u \$tr \hat{^}_u \langle \ll a \gg \rangle \wedge \$st' =_u \$st) ;; P = P[\$tr \hat{^}_u \langle \ll a \gg \rangle / \$tr]$
using *assms* **by** (*rel-auto, meson*)

lemma *tr-assign-comp* [rdes]:

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

assumes $\$ok \# P \$wait \# P \$ref \# P$
shows $(\$tr' =_u \$tr \wedge \lceil \langle \sigma \rangle_a \rceil_s) ;; P = \lceil \sigma \rceil_{s\sigma} \dagger P$
using *assms* **by** (*rel-auto*, *meson*)

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

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

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

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

lemma *conj-less-tr-RR-closed [closure]*:

assumes $P\ is\ CRR$
shows $(P \wedge \$tr <_u \$tr')\ is\ CRR$

proof –

have $CRR(CRR(P) \wedge \$tr <_u \$tr') = (CRR(P) \wedge \$tr <_u \$tr')$
apply (*rel-auto*, *blast+*)
using *less-le* **apply** *fastforce+*
done

thus *?thesis*

by (*metis Healthy-def assms*)

qed

lemma *R4-CRR-closed [closure]*: $P\ is\ CRR \implies R_4(P)\ is\ CRR$
by (*simp add: R4-def conj-less-tr-RR-closed*)

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

assumes $P\ is\ CRR$
shows $R_5(P)\ is\ CRR$

proof –

have $R_5(CRR(P))\ is\ CRR$
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]:
 $P \text{ is CRR} \implies (\exists \$ref \cdot P) \text{ is CRR}$
by (*simp add: CRR-unrest-ref ex-unrest*)

lemma *ex-st'-CRR-closed* [closure]:
 $P \text{ is CRR} \implies (\exists \$st' \cdot P) \text{ is CRR}$
by (*rule CRR-intro, simp-all add: closure unrest*)

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

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

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

3.4 Weakest Precondition

lemma *nil-least* [simp]:

$\langle \rangle \leq_u x = \text{true}$ **by** *rel-auto*

lemma *minus-nil* [simp]:

$xs - \langle \rangle = xs$ **by** *rel-auto*

lemma *wp-rea-circus-lemma-1*:

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

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

proof –

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

thus *?thesis*

by (*simp add: Healthy-if assms(1) assms(2) ex-unrest*)

qed

lemma *wp-rea-circus-lemma-2*:

assumes P is CRR

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

proof –

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

thus *?thesis*

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

qed

The meaning of reactive weakest precondition for Circus. $P \text{ wp}_r Q$ means that, whenever P terminates in a state s_0 having done the interaction trace t_0 , which is a prefix of the overall trace, then Q must be satisfied. This in particular means that the remainder of the trace after t_0 must not be a divergent behaviour of Q .

lemma *wp-rea-circus-form*:

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

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

proof –

have $(P \text{ wp}_r Q) = (\neg_r (\exists t_0 \cdot P[\ll t_0 \gg / \$tr']) ; (\neg_r Q)[\ll t_0 \gg / \$tr] \wedge \ll t_0 \gg \leq_u \$tr')$
by (*simp-all add: wp-rea-def R2-tr-middle closure assms*)

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

by (*rel-blast*)

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

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

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

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

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

by (rel-auto)

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

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

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

by (rel-auto, blast+)

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

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

oops

3.5 Trace Substitution

definition trace-subst $(\ll \cdot \gg_t [999, 0] 999)$

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

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

$\ll mwb\text{-}lens\ x; x \bowtie (\$tr)_v; x \bowtie (\$tr')_v; x \bowtie (\$st)_v; x \# P \gg \implies x \# P[\ll v \gg_t]$

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

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

assumes P is RR

shows $P[\ll v \gg_t]$ is RR

proof –

have $(RR P)[\ll v \gg_t]$ 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)

```

3.6 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]: $\mathcal{I}(s, t) = (\lceil s \rceil_{S<} \wedge \$tr + \lceil t \rceil_{S<} \leq_u \$tr')$

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

```

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

```

```

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

```

```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```


apply (*metis add.assoc diff-add-cancel-left' le-add trace-class.add-le-imp-le-left trace-class.add-left-mono*) +
done

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

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

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

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

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

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

3.7 Enabled Events

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

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

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

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

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

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

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

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

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

$$([s_1]_{S<} \wedge \mathcal{E}(s_2, t, E)) = \mathcal{E}(s_1 \wedge s_2, t, E)$$

by (*rel-auto*)

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

by (*rel-auto*)

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

by (*rel-auto*)

lemma *msubst-nil-csp-enable* [*usubst*]:

$$\mathcal{E}(s(x), t(x), E(x)) \llbracket x \rightarrow \langle \rangle \rrbracket = \mathcal{E}(s(x) \llbracket x \rightarrow \langle \rangle \rrbracket, t(x) \llbracket x \rightarrow \langle \rangle \rrbracket, E(x) \llbracket x \rightarrow \langle \rangle \rrbracket)$$

by (*pred-auto*)

lemma *msubst-csp-enable* [*usubst*]:

$$\mathcal{E}(s(x), t(x), E(x)) \llbracket x \rightarrow [v]_{S\leftarrow} \rrbracket = \mathcal{E}(s(x) \llbracket x \rightarrow v \rrbracket, t(x) \llbracket x \rightarrow v \rrbracket, E(x) \llbracket x \rightarrow v \rrbracket)$$

by (*rel-auto*)

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

by (*rel-auto*)

lemma *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*)

3.8 Completed Trace Interaction

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

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

lemma $unrest\text{-}csp\text{-}do$ $[unrest]:$

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

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

lemma $csp\text{-}do\text{-}CRR$ $[closure]: \Phi(s, \sigma, t)$ *is CRR*

by $(rel\text{-}auto)$

lemma $csp\text{-}do\text{-}R4\text{-}closed$ $[closure]:$

$\Phi(b, \sigma, bop \text{ Cons } x \text{ xs})$ *is R4*

by $(rel\text{-}auto, simp \text{ add: } Prefix\text{-}Order.strict\text{-}prefixI')$

lemma $st\text{-}pred\text{-}conj\text{-}csp\text{-}do$ $[rpred]:$

$([b]_{S<} \wedge \Phi(s, \sigma, t)) = \Phi(b \wedge s, \sigma, t)$

by $(rel\text{-}auto)$

lemma $trea\text{-}subst\text{-}csp\text{-}do$ $[usubst]:$

$(\Phi(s, \sigma, t_2)) \llbracket t_1 \rrbracket_t = \Phi(s, \sigma, t_1 \hat{\ }_u t_2)$

apply $(rel\text{-}auto)$

apply $(metis \text{ append.assoc } less\text{-}eq\text{-}list\text{-}def \text{ prefix-concat-minus})$

apply $(simp \text{ add: } list\text{-}concat\text{-}minus\text{-}list\text{-}concat)$

done

lemma $st\text{-}subst\text{-}csp\text{-}do$ $[usubst]:$

$[\sigma]_{S\sigma} \dagger \Phi(s, \varrho, t) = \Phi(\sigma \dagger s, \varrho \circ \sigma, \sigma \dagger t)$

by $(rel\text{-}auto)$

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

by $(rel\text{-}auto)$

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

by $(rel\text{-}auto)$

lemma $csp\text{-}do\text{-}assign$ $[rpred]:$

assumes P *is CRR*

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

proof –

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

by $(rel\text{-}blast)$

thus *?thesis*

by $(simp \text{ add: } Healthy\text{-}if \text{ assms})$

qed

lemma $subst\text{-}state\text{-}csp\text{-}enable$ $[usubst]:$

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

by $(rel\text{-}auto)$

lemma $csp\text{-}do\text{-}assign\text{-}enable$ $[rpred]:$

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

by $(simp \text{ add: } rpred \text{ closure } usubst)$

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

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

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

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

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

$[b]^\top_r ;; \Phi(s, \sigma, t) = \Phi(b \wedge s, \sigma, t)$
by (*rel-auto*)

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

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

proof –

have $\Phi(\text{true}, \text{id}, t) ;; \text{CRR}(P) = (\text{CRR } P) \llbracket t \rrbracket_t$
by (*rel-auto*)

thus *?thesis*

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

qed

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

fixes *P* :: (*'σ*, *'φ*) *action*

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

shows $(\llbracket \langle \sigma \rangle_a \rrbracket_S \wedge \llbracket \text{str}' \rrbracket_u = \llbracket \text{str} \rrbracket_u \llbracket t \rrbracket_{S<} ; P = (\llbracket \sigma \rrbracket_{S\sigma} \dagger P) \llbracket \text{str} \hat{}_u \llbracket t \rrbracket_{S<} / \text{str} \rrbracket$

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

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

fixes *P* :: (*'σ*, *'φ*) *action*

assumes *P* is *CRR*

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

proof –

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

by (*rel-blast*)

thus *?thesis*

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

qed

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

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

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

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

assumes *P* is *CRR*

shows $\Phi(\text{true}, \text{id}, t) \hat{}_u i ;; P = \Phi(\text{true}, \text{id}, \text{iter}[i](t)) ;; P$

apply (*cases i*)

apply (*simp-all add: rpred usubst assms closure*)

done

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

fixes *Q* :: (*'σ*, *'φ*) *action*

assumes *P* is *CRR*

shows $\Phi(s, \text{id}, t) \text{wp}_r P = (\mathcal{I}(s, t) \Rightarrow_r P \llbracket t \rrbracket_t)$

proof –

have $\Phi(s, id, t) \text{ } wp_r \text{ } P = \Phi(s, id, t) \text{ } wp_r \text{ } P$
 by (*simp add: skip-r-def*)
 thus ?thesis **by** (*simp add: wp assms usubst alpha*)

qed

lemma *msubst-csp-do* [*usubst*]:

$\Phi(s(x), \sigma, t(x)) \llbracket x \rightarrow [v]_{S \leftarrow} \rrbracket = \Phi(s(x) \llbracket x \rightarrow v \rrbracket, \sigma, t(x) \llbracket x \rightarrow v \rrbracket)$
 by (*rel-auto*)

3.9 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*]: $CDC(P) = (\exists \text{ } ref_0 \cdot P \llbracket \llbracket ref_0 \rrbracket / \$ref' \rrbracket \wedge \$ref' \subseteq_u \llbracket ref_0 \rrbracket)$

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*]: $P \text{ is } RR \Longrightarrow CDC(P) \text{ is } RR$

by (*metis CDC-RR-commute Healthy-def*)

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

assumes $P \text{ is } CRR$

shows $CDC(P) \text{ is } CRR$

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

lemma *CDC-unrest* [*unrest*]: $\llbracket vwb\text{-}lens \text{ } x; (\$ref')_v \bowtie x; x \# P \rrbracket \Longrightarrow x \# CDC(P)$

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*]: $P \text{ is } CDC \Longrightarrow R4(P) \text{ is } CDC$

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*]: $P \text{ is } CDC \Longrightarrow R5(P) \text{ is } CDC$

by (*simp add: CDC-R5-commute Healthy-def*)

lemma *rea-true-CDC* [*closure*]: $true_r \text{ is } CDC$

by (*rel-auto*)

lemma *false-CDC* [*closure*]: $false \text{ is } CDC$

by (*rel-auto*)

lemma *CDC-UNF-closed* [*closure*]:

assumes $\bigwedge i. i \in I \Longrightarrow P \text{ } i \text{ is } CDC$

shows $(\bigcap i \in I \cdot P \text{ } i) \text{ is } CDC$

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 *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

3.10 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 $((-)\langle\!\rangle_c [999, 0] 999)$ **where**
[upred-defs]: $P\langle\!\rangle_c = R2((\$tr' =_u \langle\!\rangle \wedge \$st' =_u \$st) ;; P ;; (\$tr' =_u map_u \ll f \gg \$tr \wedge \$st' =_u \$st \wedge \ll pre-image f \gg (\$ref')_a \subseteq_u \$ref))$

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

assumes *P is CRR*

shows $P\langle\!\rangle_c$ *is CRR*

proof –

have $(CRR P)\langle\!\rangle_c$ *is CRR*

by (*rel-auto*)

thus ?thesis **by** (*simp add: assms Healthy-if*)

qed

lemma *csp-rename-disj* [*rpred*]: $(P \vee Q)\langle\!\rangle_c = (P\langle\!\rangle_c \vee Q\langle\!\rangle_c)$

by (*rel-blast*)

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

by (*rel-blast*)

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

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 $(P \wedge Q)\langle\!\rangle_c = (P\langle\!\rangle_c \wedge Q\langle\!\rangle_c)$

proof –

from *assms(1)* **have** $((CDC (CRR P)) \wedge (CDC (CRR Q)))\langle\!\rangle_c = ((CDC (CRR P))\langle\!\rangle_c \wedge (CDC (CRR Q))\langle\!\rangle_c)$

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 $(P ;; Q)\langle\!\rangle_c = P\langle\!\rangle_c ;; Q\langle\!\rangle_c$

oops

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

$(R4(P))\langle\!\rangle_c = R4(P\langle\!\rangle_c)$

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*]:

$(R5(P))(\llbracket f \rrbracket)_c = R5(P(\llbracket f \rrbracket)_c)$
apply (*rel-auto*, *blast*)
using *less-le* **apply** *fastforce*
done

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

lemma *csp-rename-enable* [*rpred*]: $\mathcal{E}(s, t, E)(\llbracket f \rrbracket)_c = \mathcal{E}(s, \text{map}_u \llbracket f \rrbracket t, \llbracket \text{image } f \rrbracket (E)_a)$
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*)

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* :: $((\sigma', \varphi) \text{ st-csp} \times (\sigma', \varphi) \text{ st-csp}) \text{ health}$
where $CSP1(P) \equiv RD1(P)$

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

abbreviation *CSP* :: $((\sigma', \varphi) \text{ st-csp} \times (\sigma', \varphi) \text{ st-csp}) \text{ health}$
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* :: $((\sigma', \varphi) \text{ st-csp} \times (\sigma', \varphi) \text{ st-csp}) \text{ health}$ **where**

[upred-defs]: $CSP3(P) = (Skip \mathrel{;;} P)$

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

definition $NCSP :: ((\sigma, \varphi) \text{ st-csp} \times (\sigma, \varphi) \text{ st-csp}) \text{ health}$ **where**
 [upred-defs]: $NCSP = CSP3 \circ CSP_4 \circ CSP$

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

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

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

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

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

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

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

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

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

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

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

by (rel-auto)

lemma *periR-Skip* [rdes]: $\text{peri}_R(\text{Skip}) = \text{false}$
 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 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*: $P \text{ is } CSP4 \implies P \text{ is } RD3$

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

lemma *CSP4-tri-intro*:

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

lemma *CSP4-NSRD-intro*:

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

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

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

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

by (metis (no-types, hide-lams) CSP3-def CSP4-def Healthy-def NCSP-def SRD-idem SRD-seqr-closure)

Skip-is-CSP comp-apply)

lemma *NCSP-elim* [*RD-elim*]:

$\llbracket X \text{ is NCSP}; P(\mathbf{R}_s(\text{pre}_R(X) \vdash \text{peri}_R(X) \diamond \text{post}_R(X))) \rrbracket \implies P(X)$
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 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*:

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

proof –

have $\text{NCSP } P = \text{CSP3 } (\text{CSP4 } (\text{NSRD } P))$

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

also

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

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

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

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

proof –

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

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

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

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

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

by (*simp add: 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 $\text{pre}_R(P) = \text{pre}_R(\mathbf{R}_s ((\neg_r \text{pre}_R P) \text{wp}_r \text{false} \vdash ((\exists \$st' \cdot \text{peri}_R(P)) \diamond (\exists \$ref' \cdot \text{post}_R(P))))$

using *CSP4-healthy-form* *assms(1)* *assms(2)* by *fastforce*
 also have ... = $(\neg_r \text{pre}_R P) \text{wp}_r \text{false}$
 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 $A \subseteq \llbracket \text{NCSP} \rrbracket_H$
 shows $\bigwedge P. P \in A \implies \$wait' \# \text{pre}_R(P)$
proof –
 fix P
 assume $P \in A$
 hence P is *NSRD*
 using *NCSP-implies-NSRD assms* by *auto*
 thus $\$wait' \# \text{pre}_R(P)$
 using *NSRD-wait'-unrest-pre* by *blast*
 qed

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

lemma *CSP4-ref'-unrest-post [unrest]*:
 assumes P is *CSP* P is *CSP4*
 shows $\$ref' \# \text{post}_R(P)$
proof –
 have $\text{post}_R(P) = \text{post}_R(\mathbf{R}_s((\neg_r \text{pre}_R P) \text{wp}_r \text{false} \vdash ((\exists \$st' \cdot \text{peri}_R(P)) \diamond (\exists \$ref' \cdot \text{post}_R(P))))))$
 using *CSP4-healthy-form assms(1) assms(2)* by *fastforce*
 also have ... = $R1 (R2c ((\neg_r \text{pre}_R P) \text{wp}_r \text{false} \Rightarrow_r (\exists \$ref' \cdot \text{post}_R P)))$
 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 pre_R(P) is CRR*
 by (rule CRR-intro, simp-all add: closure unrest)

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

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

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

lemma *NCSP-preR-CRC* [closure]:
 assumes *P is NCSP*
 shows *pre_R(P) is CRC*
 by (rule CRC-intro, simp-all add: closure assms unrest)

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

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

lemma *NCSP-Sup-closure* [closure]: $\llbracket A \subseteq \llbracket NCSP \rrbracket_H; A \neq \{\} \rrbracket \implies (\bigwedge A)$ is NCSP
 apply (rule NCSP-intro, simp-all add: closure)

apply (*metis* (*no-types*, *lifting*) *Ball-Collect CSP3-Sup-closure NCSP-implies-CSP3*)
apply (*metis* (*no-types*, *lifting*) *Ball-Collect CSP4-Sup-closure NCSP-implies-CSP4*)
done

lemma *NCSP-SUP-closure* [*closure*]: $\llbracket \bigwedge i. P(i) \text{ is NCSP}; A \neq \{\} \rrbracket \implies (\bigcap_{i \in A}. P(i)) \text{ is NCSP}$
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 $\dots \text{ is NCSP}$

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 $\dots \text{ is NCSP}$

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

typeddecl *TCSP*

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

overloading

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

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

begin

definition $tcsp\text{-}hcond :: (TCSP, ('\sigma, '\varphi) \text{ st-csp}) \text{ uthy} \Rightarrow ((' \sigma, '\varphi) \text{ st-csp} \times (' \sigma, '\varphi) \text{ st-csp}) \text{ health}$ **where**

[*upred-defs*]: $tcsp\text{-}hcond T = NCSP$

definition $tcsp\text{-}unit :: (TCSP, ('\sigma, '\varphi) \text{ st-csp}) \text{ uthy} \Rightarrow (' \sigma, '\varphi) \text{ action}$ **where**

[*upred-defs*]: $tcsp\text{-}unit T = Skip$

end

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

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

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

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

and $\top_{TCSP} = \text{Miracle}$
and $\text{carrier } (\text{uthy-order } TCSP) \rightarrow \text{carrier } (\text{uthy-order } TCSP) \equiv \llbracket NCSP \rrbracket_H \rightarrow \llbracket NCSP \rrbracket_H$
and $A \subseteq \text{carrier } (\text{uthy-order } TCSP) \longleftrightarrow A \subseteq \llbracket NCSP \rrbracket_H$
and $\text{le } (\text{uthy-order } TCSP) = \text{op} \sqsubseteq$
proof –
interpret $\text{lat: utp-theory-continuous } UTHY(TCSP, ('σ, 'φ) \text{ st-csp})$
by $(\text{unfold-locales, simp-all add: tcsp-hcond-def closure Healthy-if})$
show $1: \top_{TCSP} = (\text{Miracle} :: ('σ, 'φ) \text{ action})$
by $(\text{metis NCSP-Miracle NCSP-implies-CSP lat.top-healthy lat.utp-theory-continuous-axioms srdes-theory-continuous. tcsp-hcond-def upred-semiring.add-commute utp-theory-continuous.meet-top})$

thus $\text{utp-theory-kleene } UTHY(TCSP, ('σ, 'φ) \text{ st-csp})$
by $(\text{unfold-locales, simp-all add: tcsp-hcond-def tcsp-unit-def Skip-left-unit Skip-right-unit closure Healthy-if Miracle-left-zero })$
qed $(\text{simp-all add: tcsp-hcond-def tcsp-unit-def closure Healthy-if})$

declare $\text{csp-theory.top-healthy [simp del]}$
declare $\text{csp-theory.bottom-healthy [simp del]}$

abbreviation $\text{TestC } (test_C) \text{ where}$
 $\text{test}_C P \equiv \text{utest } TCSP P$

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

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

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

5.3 Algebraic laws

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

end

6 Stateful-Failure Reactive Contracts

theory $\text{utp-sfrd-contracts}$
imports utp-sfrd-healths
begin

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

syntax

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

parse-translation \ll

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

translations

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

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

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

Refinement introduction law for contracts

lemma *CRD-contract-refine*:

assumes
 $Q \text{ is CSP } '[P_1]_{S<} \Rightarrow \text{pre}_R Q'$
 $'[P_1]_{S<} \wedge \text{peri}_R Q \Rightarrow [P_2 \ t \ r]_{S<} \llbracket t \rightarrow \&tt \rrbracket [r \rightarrow \$ref']'$
 $'[P_1]_{S<} \wedge \text{post}_R Q \Rightarrow [P_3 \ x]_S \llbracket x \rightarrow \&tt \rrbracket'$
shows $[P_1 \vdash P_2 \text{ trace refs} \mid P_3(\text{trace})]_C \sqsubseteq Q$

proof –

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

qed

lemma *CRD-contract-refine'*:

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

lemma *CRD-refine-CRD*:

assumes
 $'[P_1]_{S<} \Rightarrow ([Q_1]_{S<} :: ('e, 's) \text{ action})'$
 $([P_2 \ x \ r]_{S<} \llbracket x \rightarrow \&tt \rrbracket [r \rightarrow \$ref']) \sqsubseteq ([P_1]_{S<} \wedge [Q_2 \ x \ r]_{S<} \llbracket x \rightarrow \&tt \rrbracket [r \rightarrow \$ref'] :: ('e, 's) \text{ action})$
 $[P_3 \ x]_S \llbracket x \rightarrow \&tt \rrbracket \sqsubseteq ([P_1]_{S<} \wedge [Q_3 \ x]_S \llbracket x \rightarrow \&tt \rrbracket :: ('e, 's) \text{ action})$
shows $([P_1 \vdash P_2 \text{ trace refs} \mid P_3 \text{ trace}]_C :: ('e, 's) \text{ action}) \sqsubseteq [Q_1 \vdash Q_2 \text{ trace refs} \mid Q_3 \text{ trace}]_C$

using *assms*
by (*simp add: mk-CRD-def, rule-tac sdes-tri-refine-intro, rel-auto+*)

lemma *CRD-refine-rdes*:

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

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

$\triangleright (\prod_{P \in A} \cdot \text{cmt}_R(P)))$

syntax

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

translations

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

definition *extChoice* ::

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

Small external choice as an indexed big external choice.

lemma *extChoice-alt-def*:

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

7.2 Basic laws

7.3 Algebraic laws

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

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

lemma *ExtChoice-single*:

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

7.4 Reactive design calculations

lemma *ExtChoice-rdes*:

assumes $\bigwedge i. \$ok' \nmid P(i) \ A \neq \{\}$

shows $(\square i \in A \cdot \mathbf{R}_s(P(i) \vdash Q(i))) = \mathbf{R}_s((\square i \in A \cdot P(i)) \vdash ((\square i \in A \cdot Q(i)) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\square i \in A \cdot Q(i))))$

proof –

have $(\square i \in A \cdot \mathbf{R}_s(P(i) \vdash Q(i))) =$
 $\mathbf{R}_s((\square i \in A \cdot \text{pre}_R(\mathbf{R}_s(P(i) \vdash Q(i)))) \vdash$
 $((\square i \in A \cdot \text{cmt}_R(\mathbf{R}_s(P(i) \vdash Q(i))))$
 $\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
 $(\square i \in A \cdot \text{cmt}_R(\mathbf{R}_s(P(i) \vdash Q(i))))))$

by (*simp add: ExtChoice-def*)

also have ... =

$\mathbf{R}_s((\square i \in A \cdot R1(R2c(\text{pre}_s \dagger P(i)))) \vdash$
 $((\square i \in A \cdot R1(R2c(\text{cmt}_s \dagger (P(i) \Rightarrow Q(i))))))$
 $\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
 $(\square i \in A \cdot R1(R2c(\text{cmt}_s \dagger (P(i) \Rightarrow Q(i))))))$

by (*simp add: rea-pre-RHS-design rea-cmt-RHS-design*)

also have ... =

$\mathbf{R}_s((\square i \in A \cdot R1(R2c(\text{pre}_s \dagger P(i)))) \vdash$
 $R1(R2c$
 $((\square i \in A \cdot R1(R2c(\text{cmt}_s \dagger (P(i) \Rightarrow Q(i))))))$
 $\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
 $(\square i \in A \cdot R1(R2c(\text{cmt}_s \dagger (P(i) \Rightarrow Q(i))))))$

by (*metis (no-types, lifting) RHS-design-export-R1 RHS-design-export-R2c*)

also have ... =

$$\begin{aligned} & \mathbf{R}_s ((\bigsqcup i \in A \cdot R1 (R2c (pre_s \dagger P(i)))) \vdash \\ & \quad R1(R2c \\ & \quad ((\bigsqcup i \in A \cdot (cmt_s \dagger (P(i) \Rightarrow Q(i)))) \\ & \quad \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright \\ & \quad (\bigsqcup i \in A \cdot (cmt_s \dagger (P(i) \Rightarrow Q(i))))) \\ & \text{by (simp add: R2c-UNINF R2c-condr R1-cond R1-idem R1-R2c-commute R2c-idem R1-UNINF assms} \\ & \text{R1-USUP R2c-USUP)} \end{aligned}$$

also have ... =

$$\begin{aligned} & \mathbf{R}_s ((\bigsqcup i \in A \cdot R1 (R2c (pre_s \dagger P(i)))) \vdash \\ & \quad cmt_s \dagger \\ & \quad ((\bigsqcup i \in A \cdot (cmt_s \dagger (P(i) \Rightarrow Q(i)))) \\ & \quad \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright \\ & \quad (\bigsqcup i \in A \cdot (cmt_s \dagger (P(i) \Rightarrow Q(i))))) \\ & \text{by (metis (no-types, lifting) RHS-design-export-R1 RHS-design-export-R2c rdes-export-cmt)} \end{aligned}$$

also have ... =

$$\begin{aligned} & \mathbf{R}_s ((\bigsqcup i \in A \cdot R1 (R2c (pre_s \dagger P(i)))) \vdash \\ & \quad cmt_s \dagger \\ & \quad ((\bigsqcup i \in A \cdot (P(i) \Rightarrow Q(i))) \\ & \quad \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright \\ & \quad (\bigsqcup i \in A \cdot (P(i) \Rightarrow Q(i))))) \\ & \text{by (simp add: usubst)} \end{aligned}$$

also have ... =

$$\begin{aligned} & \mathbf{R}_s ((\bigsqcup i \in A \cdot R1 (R2c (pre_s \dagger P(i)))) \vdash \\ & \quad ((\bigsqcup i \in A \cdot (P(i) \Rightarrow Q(i))) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\bigsqcup i \in A \cdot (P(i) \Rightarrow Q(i))))) \\ & \text{by (simp add: rdes-export-cmt)} \end{aligned}$$

also have ... =

$$\begin{aligned} & \mathbf{R}_s ((R1(R2c(\bigsqcup i \in A \cdot (pre_s \dagger P(i)))) \vdash \\ & \quad ((\bigsqcup i \in A \cdot (P(i) \Rightarrow Q(i))) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\bigsqcup i \in A \cdot (P(i) \Rightarrow Q(i))))) \\ & \text{by (simp add: not-UNINF R1-UNINF R2c-UNINF assms)} \end{aligned}$$

also have ... =

$$\begin{aligned} & \mathbf{R}_s ((R2c(\bigsqcup i \in A \cdot (pre_s \dagger P(i)))) \vdash \\ & \quad ((\bigsqcup i \in A \cdot (P(i) \Rightarrow Q(i))) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\bigsqcup i \in A \cdot (P(i) \Rightarrow Q(i))))) \\ & \text{by (simp add: R1-design-R1-pre)} \end{aligned}$$

also have ... =

$$\begin{aligned} & \mathbf{R}_s (((\bigsqcup i \in A \cdot (pre_s \dagger P(i)))) \vdash \\ & \quad ((\bigsqcup i \in A \cdot (P(i) \Rightarrow Q(i))) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\bigsqcup i \in A \cdot (P(i) \Rightarrow Q(i))))) \\ & \text{by (metis (no-types, lifting) RHS-design-R2c-pre)} \end{aligned}$$

also have ... =

$$\begin{aligned} & \mathbf{R}_s ([\$ok \mapsto_s true, \$wait \mapsto_s false] \dagger (\bigsqcup i \in A \cdot P(i))) \vdash \\ & \quad ((\bigsqcup i \in A \cdot (P(i) \Rightarrow Q(i))) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\bigsqcup i \in A \cdot (P(i) \Rightarrow Q(i))))) \end{aligned}$$

proof –

$$\begin{aligned} & \text{from assms have } \bigwedge i. pre_s \dagger P(i) = [\$ok \mapsto_s true, \$wait \mapsto_s false] \dagger P(i) \\ & \quad \text{by (rel-auto)} \\ & \text{thus ?thesis} \\ & \quad \text{by (simp add: usubst)} \end{aligned}$$

qed

also have ... =

$$\begin{aligned} & \mathbf{R}_s ((\bigsqcup i \in A \cdot P(i)) \vdash ((\bigsqcup i \in A \cdot (P(i) \Rightarrow Q(i))) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\bigsqcup i \in A \cdot (P(i) \Rightarrow Q(i))))) \\ & \text{by (simp add: rdes-export-pre not-UNINF)} \\ & \text{also have ... = } \mathbf{R}_s ((\bigsqcup i \in A \cdot P(i)) \vdash ((\bigsqcup i \in A \cdot Q(i)) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\bigsqcup i \in A \cdot Q(i)))) \\ & \quad \text{by (rule cong[of } \mathbf{R}_s \mathbf{R}_s], \text{ simp, rel-auto, blast+)} \end{aligned}$$

finally show ?thesis .

qed

lemma *ExtChoice-tri-rdes*:

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

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

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

proof –

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

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

by (*simp add: ExtChoice-rdes assms*)

also

have ... =

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

by (*simp add: conj-comm*)

also

have ... =

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

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

also

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

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

finally show *?thesis* .

qed

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

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

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

$\mathbf{R}_s((\Box_{i \in A} \cdot P_1(i)) \vdash (((\Box_{i \in A} \cdot R5(P_2(i))) \vee (\Box_{i \in A} \cdot R4(P_2(i)))) \diamond (\Box_{i \in A} \cdot P_3(i))))$

by (*simp add: ExtChoice-tri-rdes assms, rel-auto, simp-all add: less-le assms*)

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

assumes $A \subseteq \llbracket CSP \rrbracket_H$

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

proof –

have $((\Box_{P \in A} \cdot cmt_R \ P) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\Box_{P \in A} \cdot cmt_R \ P)) =$

$((\Box_{P \in A} \cdot cmt_R \ P) \triangleleft \$tr' =_u \$tr \triangleright (\Box_{P \in A} \cdot cmt_R \ P)) \diamond (\Box_{P \in A} \cdot cmt_R \ P)$

by (*rel-auto*)

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

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 ... = $\mathbf{R}_s ((P_1 \wedge Q_1) \vdash ((P_2 \wedge Q_2) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (P_2 \vee Q_2)))$
 apply (*subst ExtChoice-rdes, simp-all add: assms unrest*)
 apply *unliteralise*
 apply (*simp add: uinf-or usup-and*)
 done
 finally show ?thesis by (*simp add: extChoice-alt-def*)
 qed

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

lemma *extChoice-rdes-def*:
 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]*:
 $ExtChoice \ A \text{ is } CSP$
 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 $pre_R(ExtChoice\ A) = (\bigsqcup P \in A \cdot pre_R(P))$
proof –
 have $pre_R(ExtChoice\ A) = (R1\ (R2c\ ((\bigsqcup P \in A \cdot pre_R(P))))$
 by (*simp add: ExtChoice-def rea-pre-RHS-design usubst unrest*)
 also from *assms* have $\dots = (R1\ (R2c\ (\bigsqcup P \in A \cdot (pre_R(CSP(P)))))$
 by (*metis USUP-healthy*)
 also from *assms* have $\dots = (\bigsqcup P \in A \cdot (pre_R(CSP(P))))$
 by (*rel-auto*)
 also from *assms* have $\dots = (\bigsqcup P \in A \cdot (pre_R(P)))$
 by (*metis USUP-healthy*)
 finally show *?thesis* .
qed

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

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

proof –
 have $peri_R(ExtChoice\ A) = peri_R(\mathbf{R}_s((\bigsqcup P \in A \cdot pre_R\ P)) \vdash$
 $((\bigsqcup P \in A \cdot peri_R\ P) \triangleleft \$tr' =_u \$tr \triangleright (\bigsqcup P \in A \cdot peri_R\ P)) \diamond$
 $(\bigsqcup P \in A \cdot post_R\ P)))$
 by (*simp add: ExtChoice-tri-rdes-def assms closure*)
 also have $\dots = peri_R(\mathbf{R}_s((\bigsqcup P \in A \cdot pre_R(NCSP\ P)) \vdash$
 $((\bigsqcup P \in A \cdot peri_R(NCSP\ P)) \triangleleft \$tr' =_u \$tr \triangleright (\bigsqcup P \in A \cdot peri_R(NCSP\ P))) \diamond$
 $(\bigsqcup P \in A \cdot post_R\ P)))$
 by (*simp add: UINF-healthy[OF assms(1), THEN sym] USUP-healthy[OF assms(1), THEN sym]*)
 also have $\dots = R1\ (R2c\ ((\bigsqcup P \in A \cdot pre_R(NCSP\ P)) \Rightarrow_r$
 $(\bigsqcup P \in A \cdot peri_R(NCSP\ P))$
 $\triangleleft \$tr' =_u \$tr \triangleright$
 $(\bigsqcup P \in A \cdot peri_R(NCSP\ P))))$

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

also have $\dots = R1\ ((\bigsqcup P \in A \cdot pre_R(NCSP\ P)) \Rightarrow_r$
 $(\bigsqcup P \in A \cdot peri_R(NCSP\ P))$
 $\triangleleft \$tr' =_u \$tr \triangleright$
 $(\bigsqcup P \in A \cdot peri_R(NCSP\ P)))$
 by (*simp add: R2c-rea-impl R2c-condr R2c-UINF R2c-preR R2c-periR R2c-tr'-minus-tr R2c-USUP closure*)
 also have $\dots = (((\bigsqcup P \in A \cdot pre_R(NCSP\ P)) \Rightarrow_r (\bigsqcup P \in A \cdot peri_R(NCSP\ P)))$
 $\triangleleft \$tr' =_u \$tr \triangleright$
 $((\bigsqcup P \in A \cdot pre_R(NCSP\ P)) \Rightarrow_r (\bigsqcup P \in A \cdot peri_R(NCSP\ P))))$
 by (*simp add: R1-rea-impl R1-cond R1-USUP R1-UINF assms Healthy-if closure, rel-auto*)
 also have $\dots = (((\bigsqcup P \in A \cdot pre_R(NCSP\ P)) \Rightarrow_r (\bigsqcup P \in A \cdot peri_R(NCSP\ P)))$

$\triangleleft \$tr' =_u \$tr \triangleright$
 $((\prod P \in A \cdot pre_R (NCSP P)) \Rightarrow_r peri_R (NCSP P)))$
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 ... = $R1 ((\sqcup P \in A \cdot pre_R (NCSP P)) \Rightarrow_r (\sqcap P \in A \cdot post_R (NCSP P)))$
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** ... = $((\sqcup P \in A \cdot pre_R (NCSP P)) \Rightarrow_r (\sqcap P \in A \cdot post_R (NCSP P)))$
by (*simp add: R1-rea-impl R1-cond R1-USUP R1-UINF assms Healthy-if closure*)
also have ... = $(\sqcap P \in A \cdot pre_R (NCSP P) \Rightarrow_r post_R (NCSP P))$
by (*simp add: UINF-rea-impl*)
also have ... = $(\sqcap P \in A \cdot post_R (NCSP P))$
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 $post_R(\sqcap P \in A \cdot F(P)) = (\sqcap P \in A \cdot post_R(F(P)))$
using *assms* **by** (*subst postR-ExtChoice, auto simp add: closure unrest*)

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

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

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

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

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

lemma *ref-unrest-ExtChoice*:
assumes
 $\bigwedge P. P \in A \implies \$ref \# pre_R(P)$
 $\bigwedge P. P \in A \implies \$ref \# cmt_R(P)$
shows $\$ref \# (ExtChoice A) \llbracket false / \$wait \rrbracket$
proof –
have $\bigwedge P. P \in A \implies \$ref \# pre_R(P \llbracket 0 / \$tr \rrbracket)$
using *assms* **by** (*rel-blast*)
with *assms* **show** *?thesis*

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

lemma CSP4-ExtChoice:

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

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

proof -

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

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 $A \subseteq \llbracket NCSP \rrbracket_H$
shows *ExtChoice* A is *NCSP*
proof (*cases* $A = \{\}$)
 case *True*
 then show ?thesis **by** (*simp add: ExtChoice-empty closure*)
next
 case *False*
 show ?thesis
proof (*rule NCSP-intro*)
 from *assms* **have** $cls: A \subseteq \llbracket CSP \rrbracket_H \ A \subseteq \llbracket CSP^3 \rrbracket_H \ A \subseteq \llbracket CSP^4 \rrbracket_H$
 using *NCSP-implies-CSP NCSP-implies-CSP^3 NCSP-implies-CSP^4* **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 $1: ExtChoice \ A \text{ is } CSP$
 by (*metis (mono-tags) Ball-Collect CSP-ExtChoice NCSP-implies-CSP assms*)
 from cls **show** *ExtChoice* A is *CSP^3*
 by (*rule-tac CSP^3-SRD-intro, simp-all add: CSP-Healthy-subset-member CSP^3-Healthy-subset-member*
closure rdes unrest wu assms 1 False)
 from cls **show** *ExtChoice* A is *CSP^4*
 by (*simp add: CSP^4-ExtChoice assms*)
qed
qed

lemma *ExtChoice-NCSP-closed* [closure]:
assumes $\bigwedge i. i \in I \implies P(i) \text{ is } NCSP$
shows $(\bigcap_{i \in I} P(i)) \text{ is } NCSP$
by (*simp add: NCSP-ExtChoice assms image-subset-iff*)

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

7.5 Productivity and Guardedness

lemma *Productive-ExtChoice* [closure]:
assumes $A \neq \{\} \ A \subseteq \llbracket NCSP \rrbracket_H \ A \subseteq \llbracket Productive \rrbracket_H$
shows *ExtChoice* A is *Productive*
proof –
 have $1: \bigwedge P. P \in A \implies \$wait' \nmid pre_R(P)$
 using *NCSP-implies-NSRD NSRD-wait'-unrest-pre assms(2)* **by** *blast*
 show ?thesis
proof (*rule Productive-intro, simp-all add: assms closure rdes 1 unrest*)
 have $((\bigcup P \in A \cdot pre_R P) \wedge (\bigcap P \in A \cdot post_R P)) =$
 $((\bigcup P \in A \cdot pre_R P) \wedge (\bigcap P \in A \cdot (pre_R P \wedge post_R P)))$
 by (*rel-auto*)
 moreover have $(\bigcap P \in A \cdot (pre_R P \wedge post_R P)) = (\bigcap P \in A \cdot ((pre_R P \wedge post_R P) \wedge \$tr <_u$
 $\$tr'))$
 by (*rule UINF-cong, metis (no-types, lifting) 1 Ball-Collect NCSP-implies-CSP Productive-post-refines-tr-increase*
assms utp-pred-laws.inf.absorb1)

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

qed

lemma *Productive-extChoice* [closure]:
 assumes *P* is NCSP *Q* is NCSP *P* is Productive *Q* is Productive
 shows $P \sqcap Q$ 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 $\bigwedge Y. ((\sqcap P \in A \cdot P Y) \wedge \text{gvert}(n+1)) = ((\sqcap P \in A \cdot (P Y \wedge \text{gvert}(n+1))) \wedge \text{gvert}(n+1))$
proof –
 fix *Y*
 let *?lhs* = $((\sqcap P \in A \cdot P Y) \wedge \text{gvert}(n+1))$ and *?rhs* = $((\sqcap P \in A \cdot (P Y \wedge \text{gvert}(n+1))) \wedge \text{gvert}(n+1))$
 have *a*: *?lhs*[[false/\$ok]] = *?rhs*[[false/\$ok]]
 by (rel-auto)
 have *b*: *?lhs*[[true/\$ok]][[true/\$wait]] = *?rhs*[[true/\$ok]][[true/\$wait]]
 by (rel-auto)
 have *c*: *?lhs*[[true/\$ok]][[false/\$wait]] = *?rhs*[[true/\$ok]][[false/\$wait]]
 by (simp add: ExtChoice-def RHS-def R1-def R2c-def R2s-def R3h-def design-def usubst unrest, rel-blast)
 show *?lhs* = *?rhs*
 using *a b c*
 by (rule-tac bool-eq-splitI[of in-var ok], simp, rule-tac bool-eq-splitI[of in-var wait], simp-all)
 qed
 moreover have $((\sqcap P \in A \cdot (P X \wedge \text{gvert}(n+1))) \wedge \text{gvert}(n+1)) = ((\sqcap P \in A \cdot (P (X \wedge \text{gvert}(n))) \wedge \text{gvert}(n+1))) \wedge \text{gvert}(n+1))$
proof –
 have $(\sqcap P \in A \cdot (P X \wedge \text{gvert}(n+1))) = (\sqcap P \in A \cdot (P (X \wedge \text{gvert}(n)) \wedge \text{gvert}(n+1)))$
proof (rule *ExtChoice-cong*)
 fix *P* assume $P \in A$
 thus $(P X \wedge \text{gvert}(n+1)) = (P (X \wedge \text{gvert}(n)) \wedge \text{gvert}(n+1))$
 using *Guarded-def* assms by blast
 qed
 thus *?thesis* by simp
 qed
 ultimately show $((\sqcap P \in A \cdot P X) \wedge \text{gvert}(n+1)) = ((\sqcap P \in A \cdot (P (X \wedge \text{gvert}(n)))) \wedge \text{gvert}(n+1))$
 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 ... =

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

by (simp add: extChoice-rdes unrest)

also have ... =

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

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

also have ... =

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

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

also have ... =

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

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

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

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

by (simp add: extChoice-rdes unrest)

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

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

also have ... = $\mathbf{R}_s (pre_R Q \vdash cmt_R Q)$
 by (simp add: cond-idem)
 also have ... = Q
 by (simp add: SRD-reactive-design-alt assms)
 finally show ?thesis .
 qed

lemma extChoice-Chaos:
 assumes Q is CSP
 shows $Chaos \sqcap Q = Chaos$

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

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

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

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

end

```

8 Stateful-Failure Programs

```

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

```

8.1 Conditionals

```

lemma NCSP-cond-srea [closure]:
  assumes P 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 $g \rightarrow_R P$ is NCSP
by (simp add: gcnd-def closure assms)

8.3 Alternation

lemma *AlternateR-NCSP-closed* [closure]:
assumes $\bigwedge i. i \in A \implies P(i)$ is NCSP Q is NCSP
shows $(\text{if}_R i \in A \cdot g(i) \rightarrow P(i) \text{ else } Q \text{ fi})$ 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 $\bigwedge b P. (b, P) \in \text{set } A \implies P$ is NCSP Q is NCSP
shows $(\text{AlternateR-list } A \ Q)$ 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

8.4 While Loops

lemma *NSRD-coerce-NCSP*:
 P is NSRD $\implies \text{Skip} ;; P ;; \text{Skip}$ is NCSP
by (metis (no-types, hide-lams) CSP3-Skip CSP3-def CSP4-def Healthy-def NCSP-Skip NCSP-implies-CSP
 NCSP-intro NSRD-is-SRD RA1 SRD-seqr-closure)

definition *WhileC* :: $'s \text{ upred} \Rightarrow ('s, 'e) \text{ action} \Rightarrow ('s, 'e) \text{ action}$ (*while_C - do - od*) **where**
 [rdes-def]: $\text{while}_C b \text{ do } P \text{ od} = \text{Skip} ;; \text{while}_R b \text{ do } P \text{ od} ;; \text{Skip}$

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

8.5 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 :: pttm \Rightarrow logic \Rightarrow logic \Rightarrow logic \Rightarrow logic (do_C - \in - \cdot - \rightarrow - od)
 -iter-gcommC :: gcomms \Rightarrow logic (do_C / - / od)

translations

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

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

$\bigwedge i. i \in I \Rightarrow P(i)$ 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

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

$\bigwedge b P. (b, P) \in \text{set } A \Rightarrow P$ 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)

8.6 Assignment

definition AssignsCSP :: ' σ usubst \Rightarrow (' σ , ' φ) action ($\langle \cdot \rangle_C$) **where**

[upred-defs]: AssignsCSP $\sigma = \mathbf{R}_s(\text{true} \vdash \text{false} \diamond (\$tr' =_u \$tr \wedge [\langle \sigma \rangle_a]_S))$

syntax

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

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

translations

-assigns-csp xs vs \Rightarrow CONST AssignsCSP (-mk-usubst (CONST id) xs vs)

$-assigns-csp\ x\ v\ \leq\ CONST\ AssignsCSP\ (CONST\ subst-upd\ (CONST\ id)\ x\ v)$
 $-assigns-csp\ x\ v\ \leq\ -assigns-csp\ (-spvar\ x)\ v$
 $x, y :=_C\ u, v\ \leq\ CONST\ AssignsCSP\ (CONST\ subst-upd\ (CONST\ subst-upd\ (CONST\ id)\ (CONST\ svar\ x)\ u)\ (CONST\ svar\ y)\ v)$

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 ISRD1-rdes-intro*)
apply (*simp-all add: closure*)
apply (*rel-auto*)
done

8.7 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* ::

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

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

syntax

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

translations

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

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

AssignCSP-update domf updatef x k v is CSP

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

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

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

by (*rel-auto*)

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

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

by (*rel-simp*)

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

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

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

by (*rel-auto*)

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

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

proof (*rule NCSP-intro*)

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

by (*simp add: closure*)

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

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

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

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

qed

8.8 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]: $P \text{ is NCSP} \implies \text{state } 'a \cdot P \text{ is NCSP}$

apply (*rule NCSP-NSRD-intro*)

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

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

done

8.9 Assumptions

definition *AssumeCircus* ($\{-\}_C$) **where**

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

8.10 Guards

definition *GuardCSP* ::

$'\sigma \text{ cond} \Rightarrow$

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

$('\sigma, '\varphi) \text{ action } (\text{infixr } \&_u \ 70) \text{ where}$

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

lemma *Guard-tri-design*:

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

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 $g \&_u \mathbf{R}_s(P \vdash Q \diamond R) = \mathbf{R}_s((\mathcal{I}(g, \langle \rangle) \Rightarrow_r P) \vdash ((\Phi(g, id, \langle \rangle) ;; Q) \vee \mathcal{E}(\neg g, \langle \rangle, \{ \}_u)) \diamond (\Phi(g, id, \langle \rangle) ;; R))$

(**is** *?lhs = ?rhs*)

proof –

have $?lhs = \mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r P) \vdash ((P \Rightarrow_r Q) \triangleleft \lceil g \rceil_{S<} \triangleright (\$tr' =_u \$tr)) \diamond (\lceil g \rceil_{S<} \wedge (P \Rightarrow_r R)))$

by (*simp add: Guard-tri-design rdes assms closure*)

also have $\dots = \mathbf{R}_s((\mathcal{I}(g, \langle \rangle) \Rightarrow_r P) \vdash ((\lceil g \rceil_{S<} \wedge Q) \vee \mathcal{E}(\neg g, \langle \rangle, \{ \}_u)) \diamond (\lceil g \rceil_{S<} \wedge R))$

by (*rel-auto*)

also have $\dots = \mathbf{R}_s((\mathcal{I}(g, \langle \rangle) \Rightarrow_r P) \vdash ((\Phi(g, id, \langle \rangle) ;; Q) \vee \mathcal{E}(\neg g, \langle \rangle, \{ \}_u)) \diamond (\Phi(g, id, \langle \rangle) ;; R))$

by (*simp add: assms(2) assms(3) csp-do-cond-conj*)

finally show *?thesis* .

qed

lemma *Guard-rdes-def'*:

assumes $\$ok' \nmid P$

shows $g \&_u (\mathbf{R}_s(P \vdash Q)) = \mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r P) \vdash (\lceil g \rceil_{S<} \wedge Q \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$

proof –

have $g \&_u (\mathbf{R}_s(P \vdash Q)) = \mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r \text{pre}_R(\mathbf{R}_s(P \vdash Q))) \vdash (\lceil g \rceil_{S<} \wedge \text{cmt}_R(\mathbf{R}_s(P \vdash Q)) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$

by (*simp add: GuardCSP-def*)

also have $\dots = \mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r R1(R2c(\text{pre}_s \dagger P))) \vdash (\lceil g \rceil_{S<} \wedge R1(R2c(\text{cmt}_s \dagger (P \Rightarrow Q))) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$

by (*simp add: rea-pre-RHS-design rea-cmt-RHS-design*)

also have $\dots = \mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r R1(R2c(\text{pre}_s \dagger P))) \vdash R1(R2c(\lceil g \rceil_{S<} \wedge R1(R2c(\text{cmt}_s \dagger (P \Rightarrow Q)))) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$

by (*metis (no-types, lifting) RHS-design-export-R1 RHS-design-export-R2c*)

also have $\dots = \mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r R1(R2c(\text{pre}_s \dagger P))) \vdash R1(R2c(\lceil g \rceil_{S<} \wedge (\text{cmt}_s \dagger (P \Rightarrow Q)) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$

by (*simp add: R1-R2c-commute R1-disj R1-extend-conj' R1-idem R2c-and R2c-disj R2c-idem*)

also have $\dots = \mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r R1(R2c(\text{pre}_s \dagger P))) \vdash (\lceil g \rceil_{S<} \wedge (\text{cmt}_s \dagger (P \Rightarrow Q)) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$

by (metis (no-types, lifting) RHS-design-export-R1 RHS-design-export-R2c)
 also have ... = $\mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r R1(R2c(pre_s \dagger P))) \vdash cmt_s \dagger (\lceil g \rceil_{S<} \wedge (cmt_s \dagger (P \Rightarrow Q)) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
 by (simp add: rdes-export-cmt)
 also have ... = $\mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r R1(R2c(pre_s \dagger P))) \vdash cmt_s \dagger (\lceil g \rceil_{S<} \wedge (P \Rightarrow Q) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
 by (simp add: usubst)
 also have ... = $\mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r R1(R2c(pre_s \dagger P))) \vdash (\lceil g \rceil_{S<} \wedge (P \Rightarrow Q) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
 by (simp add: rdes-export-cmt)
 also from assms have ... = $\mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r (pre_s \dagger P)) \vdash (\lceil g \rceil_{S<} \wedge (P \Rightarrow Q) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
 by (rel-auto)
 also have ... = $\mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r pre_s \dagger P) \llbracket true, false / \$ok, \$wait \rrbracket \vdash (\lceil g \rceil_{S<} \wedge (P \Rightarrow Q) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
 by (simp add: rdes-export-pre)
 also from assms have ... = $\mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r P) \llbracket true, false / \$ok, \$wait \rrbracket \vdash (\lceil g \rceil_{S<} \wedge (P \Rightarrow Q) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
 by (rel-auto)
 also from assms have ... = $\mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r P) \vdash (\lceil g \rceil_{S<} \wedge (P \Rightarrow Q) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
 by (simp add: rdes-export-pre)
 also have ... = $\mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r P) \vdash (\lceil g \rceil_{S<} \wedge Q \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
 by (rule cong[of \mathbf{R}_s \mathbf{R}_s], simp, rel-auto)
 finally show ?thesis .
 qed

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

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

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

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

lemma periR-Guard [rdes]:

assumes P is NCSP

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

proof –

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

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

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

by (rel-auto)

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

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

finally show ?thesis

by rel-auto

qed

lemma postR-Guard [rdes]:

assumes P is NCSP

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

proof –

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

by (simp add: Guard-tri-design rea-post-RHS-design usubst unrest R2c-rea-not R2c-and R2c-rea-impl
 R2c-preR R2c-postR R2c-tr'-minus-tr R2c-lift-state-pre R2c-condR R1-rea-impl R1-extend-conj'
 R1-post-SRD closure assms)
 also have ... = ($\lceil b \rceil_{S<} \wedge (\text{pre}_R P \Rightarrow_r \text{post}_R P)$)
 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 $b \&_u P$ is CSP3

proof –

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

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

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

by (pred-blast)+

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

by (simp add: CSP3-iff GuardCSP-def RHS-def R1-def R2c-def R2s-def R3h-def design-def unrest
 usubst)

thus ?thesis

by (metis CSP3-intro CSP-Guard)

qed

lemma CSP4-Guard [closure]:

assumes P is NCSP

shows $b \&_u P$ is CSP4

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

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

proof –

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

by (simp add: CSP4-neg-pre-unit assms closure)

have $(\neg_r ([b]_{S<} \Rightarrow_r \text{pre}_R P)) ;; R1 \text{ true} = (\neg_r ([b]_{S<} \Rightarrow_r \text{pre}_R P))$

proof –

have 1: $(\neg_r ([b]_{S<} \Rightarrow_r \text{pre}_R P)) = ([b]_{S<} \wedge (\neg_r \text{pre}_R P))$

by (rel-auto)

also have 2:... = $([b]_{S<} \wedge ((\neg_r \text{pre}_R P) ;; R1 \text{ true}))$

by (simp add: a)

also have 3:... = $(\neg_r ([b]_{S<} \Rightarrow_r \text{pre}_R P)) ;; R1 \text{ true}$

by (rel-auto)

finally show ?thesis ..

qed

thus ?thesis

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

qed

show $\$st' \# \text{peri}_R (b \&_u P)$

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

show $\$ref' \# \text{post}_R (b \&_u P)$

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

qed

lemma NCSP-Guard [closure]:

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

lemma *Productive-Guard [closure]*:

assumes P is CSP P is Productive $\$wait' \# pre_R(P)$
shows $b \&_u P$ is Productive
proof –
have $b \&_u P = b \&_u \mathbf{R}_s(pre_R(P) \vdash peri_R(P) \diamond (post_R(P) \wedge \$tr <_u \$tr'))$
by (*metis* Healthy-def Productive-form *assms(1) assms(2)*)
also have ... =
 $\mathbf{R}_s(((\lceil b \rceil_{S<} \Rightarrow_r pre_R P) \vdash$
 $((pre_R P \Rightarrow_r peri_R P) \triangleleft \lceil b \rceil_{S<} \triangleright (\$tr' =_u \$tr)) \diamond (\lceil b \rceil_{S<} \wedge (pre_R P \Rightarrow_r post_R P \wedge \$tr' >_u$
 $\$tr)))$
by (*simp add: Guard-tri-design rea-pre-RHS-design rea-peri-RHS-design rea-post-RHS-design unrest*
assms
usubst R1-preR Healthy-if R1-rea-impl R1-peri-SRD R1-extend-conj' R2c-preR R2c-not R2c-rea-impl
R2c-periR R2c-postR R2c-and R2c-tr-less-tr' R1-tr-less-tr')
also have ... = $\mathbf{R}_s(((\lceil b \rceil_{S<} \Rightarrow_r pre_R P) \vdash (peri_R P \triangleleft \lceil b \rceil_{S<} \triangleright (\$tr' =_u \$tr)) \diamond ((\lceil b \rceil_{S<} \wedge post_R P)$
 $\wedge \$tr' >_u \$tr)))$
by (*rel-auto*)
also have ... = *Productive*($b \&_u P$)
by (*simp add: Productive-def Guard-tri-design RHS-tri-design-par unrest*)
finally show ?thesis
by (*simp add: Healthy-def'*)
qed

8.11 Basic events

definition $do_u ::$

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

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

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

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

by (*rel-auto*)

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

by (*rel-auto*)

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

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

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

apply (*rel-auto*)

done

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

$\$ok \# do_u(a) \$wait \# do_u(a)$
by (*pred-auto*)⁺

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

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

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

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

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

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*]:

($do_C a :: ('\sigma, '\psi)$ action) is Productive

proof –

have ($(\Phi(true, id, \langle a \rangle) \wedge \$tr' >_u \$tr) :: ('\sigma, '\psi)$ action)
 $= (\Phi(true, id, \langle a \rangle))$

by (*rel-auto, simp add: Prefix-Order.strict-prefixI'*)

hence $Productive(do_C a) = do_C a$

by (*simp add: Productive-RHS-design-form DoCSP-RHS-tri unrest*)

thus *?thesis*

by (*simp add: Healthy-def*)

qed

lemma *PCSP-DoCSP* [*closure*]:

($do_C a :: ('\sigma, '\psi)$ action) is PCSP

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

lemma *wp-rea-DoCSP-lemma*:

fixes $P :: ('\sigma, '\varphi)$ action

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

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

using *assms*

by (*rel-auto, meson*)

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

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

8.12 Event prefix

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

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

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

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

also have ... = $(\mathcal{I}(\text{true}, \langle a \rangle) \wedge (\text{pre}_R P \Rightarrow_r \text{post}_R P) \llbracket \langle a \rangle \rrbracket_t)$
 by (rel-auto)
 also have ... = $(\mathcal{I}(\text{true}, \langle a \rangle) \wedge (\text{post}_R P) \llbracket \langle a \rangle \rrbracket_t)$
 by (simp add: SRD-post-under-pre assms closure unrest)
 also have ... = $(\text{post}_R P) \llbracket \langle a \rangle \rrbracket_t$
 by (rel-auto)
 finally show ?thesis .
 qed

lemma *PrefixCSP-RHS-tri*:

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

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

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

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

lemma *PrefixCSP-rdes-def-2*:

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

8.13 Guarded external choice

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

syntax

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

translations

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

lemma *GuardedChoiceCSP* [rdes-def]:

assumes $\bigwedge x. P(x)$ is NCSP $A \neq \{\}$
 shows $(\Box x \in A \rightarrow P(x)) =$
 $\mathbf{R}_s ((\Box x \in A \cdot \mathcal{I}(\text{true}, \langle \llbracket x \rrbracket \rangle)) \Rightarrow_r \text{pre}_R (P \ x) \llbracket \langle \llbracket x \rrbracket \rangle \rrbracket_t) \vdash$
 $((\Box x \in A \cdot \mathcal{E}(\text{true}, \langle \rangle, \{\llbracket x \rrbracket\}_u)) \triangleleft \$tr' =_u \$tr \triangleright (\Box x \in A \cdot \text{peri}_R (P \ x) \llbracket \langle \llbracket x \rrbracket \rangle \rrbracket_t)) \diamond$
 $(\Box x \in A \cdot \text{post}_R (P \ x) \llbracket \langle \llbracket x \rrbracket \rangle \rrbracket_t)$

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

8.14 Input prefix

definition *InputCSP* ::

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

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

definition *do_I* ::

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

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

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

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

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

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

lemma *InputCSP-rdes-def [rdes-def]*:
 assumes $\bigwedge v. P(v) \text{ is CRC} \wedge v. Q(v) \text{ is CRR} \wedge v. R(v) \text{ is CRR}$
 $\wedge v. \$st' \# Q(v) \wedge v. \$ref' \# R(v)$

shows $\text{InputCSP } a \ A \ (\lambda \ v. \ \mathbf{R}_s(P(v) \vdash Q(v) \diamond R(v))) =$
 $\mathbf{R}_s(\ (\sqcup \ v \cdot [A(v)]_{S<} \Rightarrow_r \mathcal{I}(\text{true}, \langle (a \cdot \ll v \gg)_u \rangle) \Rightarrow_r (P \ v) \llbracket \langle (a \cdot \ll v \gg)_u \rangle \rrbracket_t))$
 $\vdash (((\sqcup \ x \cdot R5([A(x)]_{S<} \Rightarrow_r \mathcal{E}(\text{true}, \langle \rangle, \{(a \cdot \ll x \gg)_u\}_u)))$
 \vee
 $(\sqcap \ x \cdot [A(x)]_{S<} \wedge (P \ x \wedge Q \ x) \llbracket \langle (a \cdot \ll x \gg)_u \rangle \rrbracket_t))$
 $\diamond (\sqcap \ x \cdot [A \ x]_{S<} \wedge (P \ x \wedge R \ x) \llbracket \langle (a \cdot \ll x \gg)_u \rangle \rrbracket_t)) \ (\text{is } ?lhs = ?rhs)$
proof –
have $1:\text{pre}_R(?lhs) = (\sqcup \ v \cdot [A \ v]_{S<} \Rightarrow_r \mathcal{I}(\text{true}, \langle (a \cdot \ll v \gg)_u \rangle) \Rightarrow_r P \ v \llbracket \langle (a \cdot \ll v \gg)_u \rangle \rrbracket_t) \ (\text{is } - = ?A)$
by (*simp add: rdes NCSP-rdes-intro assms conj-comm closure*)
have $2:\text{peri}_R(?lhs) = (\sqcup \ x \cdot [A \ x]_{S<} \Rightarrow_r \mathcal{E}(\text{true}, \langle \rangle, \{(a \cdot \ll x \gg)_u\}_u)) \triangleleft \$tr' =_u \$tr \triangleright (\sqcap \ x \cdot [A \ x]_{S<} \wedge (P \ x \Rightarrow_r Q \ x) \llbracket \langle (a \cdot \ll x \gg)_u \rangle \rrbracket_t)$
 $\wedge (P \ x \Rightarrow_r Q \ x) \llbracket \langle (a \cdot \ll x \gg)_u \rangle \rrbracket_t)$
 $(\text{is } - = ?B)$
by (*simp add: rdes NCSP-rdes-intro assms closure*)
have $3:\text{post}_R(?lhs) = (\sqcap \ x \cdot [A \ x]_{S<} \wedge (P \ x \Rightarrow_r R \ x) \llbracket \langle (a \cdot \ll x \gg)_u \rangle \rrbracket_t)$
 $(\text{is } - = ?C)$
by (*simp add: rdes NCSP-rdes-intro assms closure*)
have $?lhs = \mathbf{R}_s(?A \vdash ?B \diamond ?C)$
by (*subst SRD-reactive-tri-design[THEN sym], simp-all add: closure 1 2 3*)
also have $\dots = ?rhs$
by (*rel-auto*)
finally show $?thesis$.
qed

8.15 Renaming

definition $\text{RenameCSP} :: ('s, 'e) \text{ action} \Rightarrow ('e \Rightarrow 'f) \Rightarrow ('s, 'f) \text{ action} \ ((-) \llbracket - \rrbracket_C [999, 0] 999)$ **where**
 $[\text{upred-defs}]: \text{RenameCSP } P \ f = \mathbf{R}_s((\neg_r (\neg_r \text{pre}_R(P)) \llbracket f \rrbracket_c ;; \text{true}_r) \vdash ((\text{peri}_R(P)) \llbracket f \rrbracket_c) \diamond ((\text{post}_R(P)) \llbracket f \rrbracket_c))$

lemma $\text{RenameCSP-rdes-def}$ [*rdes-def*]:

assumes P is CRC Q is CRR R is CRR

shows $(\mathbf{R}_s(P \vdash Q \diamond R)) \llbracket f \rrbracket_C = \mathbf{R}_s((\neg_r (\neg_r P) \llbracket f \rrbracket_c ;; \text{true}_r) \vdash Q \llbracket f \rrbracket_c \diamond R \llbracket f \rrbracket_c) \ (\text{is } ?lhs = ?rhs)$

proof –

have $?lhs = \mathbf{R}_s((\neg_r (\neg_r P) \llbracket f \rrbracket_c ;; \text{true}_r) \vdash (P \Rightarrow_r Q) \llbracket f \rrbracket_c \diamond (P \Rightarrow_r R) \llbracket f \rrbracket_c)$

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

also have $\dots = \mathbf{R}_s((\neg_r (\neg_r \text{CRC}(P)) \llbracket f \rrbracket_c ;; \text{true}_r) \vdash (\text{CRC}(P) \Rightarrow_r \text{CRR}(Q)) \llbracket f \rrbracket_c \diamond (\text{CRC}(P) \Rightarrow_r \text{CRR}(R)) \llbracket f \rrbracket_c)$

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

also have $\dots = \mathbf{R}_s((\neg_r (\neg_r \text{CRC}(P)) \llbracket f \rrbracket_c ;; \text{true}_r) \vdash (\text{CRR}(Q)) \llbracket f \rrbracket_c \diamond (\text{CRR}(R)) \llbracket f \rrbracket_c)$

by (*rel-auto, (metis order-refl)+*)

also have $\dots = ?rhs$

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

finally show $?thesis$.

qed

lemma $\text{RenameCSP-pre-CRC-closed}$:

assumes P is CRR

shows $\neg_r (\neg_r P) \llbracket f \rrbracket_c ;; R1$ true 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 seqr-assoc*)

done

lemma $\text{RenameCSP-NCSP-closed}$ [*closure*]:

assumes P is NCSP

shows $P \llbracket f \rrbracket_C$ is NCSP

by (*simp add: RenameCSP-def RenameCSP-pre-CRC-closed closure assms unrest*)

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

$\text{false}\langle f \rangle_c = \text{false}$

by (*rel-auto*)

lemma *umap-nil* [*simp*]: $\text{map}_u f \langle \rangle = \langle \rangle$

by (*rel-auto*)

lemma *rename-Skip*: $\text{Skip}\langle f \rangle_C = \text{Skip}$

by (*rdes-eq*)

lemma *rename-Chaos*: $\text{Chaos}\langle f \rangle_C = \text{Chaos}$

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

lemma *rename-Miracle*: $\text{Miracle}\langle f \rangle_C = \text{Miracle}$

by (*rdes-eq*)

lemma *rename-DoCSP*: $(\text{do}_C(a))\langle f \rangle_C = \text{do}_C(\llbracket f \rrbracket(a)_a)$

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*:

$g \&_u h \&_u P = (g \wedge h) \&_u P$

by (*rule antisym*, *rel-blast*, *rel-blast*)

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

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

lemma *Guard-true* [*simp*]:

$P \text{ is CSP} \implies \text{true} \&_u P = P$

by (*simp add: GuardCSP-def alpha SRD-reactive-design-alt closure rpred*)

lemma *Guard-conditional*:

assumes *P is NCSP*

shows $b \&_u P = P \triangleleft b \triangleright_R \text{Stop}$

by (*rdes-eq cls: assms*)

lemma *Guard-expansion*:

$(g_1 \vee g_2) \&_u P = (g_1 \&_u P) \sqcap (g_2 \&_u P)$

by (*rel-auto*)

lemma *Conditional-as-Guard*:

assumes *P is NCSP* *Q is NCSP*

shows $P \triangleleft b \triangleright_R Q = b \&_u P \sqcap (\neg b) \&_u Q$

by (*rdes-eq cls: assms*; *simp add: le-less*)

lemma *PrefixCSP-dist*:

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

lemma *DoCSP-is-Prefix*:
 $\text{do}_C(a) = \text{PrefixCSP } a \text{ Skip}$
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 $((a \rightarrow_C P) \sqcap (b \rightarrow_C Q)) ;; R = (a \rightarrow_C P ;; R) \sqcap (b \rightarrow_C Q ;; R)$
by (*simp add: PCSP-PrefixCSP assms(1) assms(2) assms(3) extChoice-seq-distr*)

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

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

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

```

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

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

```

```

end

```

9 Recursion in Stateful-Failures

```

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

```

9.1 Fixed-points

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

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

```

syntax
  -mu-CSP :: pttrn  $\Rightarrow$  logic  $\Rightarrow$  logic ( $\mu_C$  - - - [l, h] h)

```

```

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

```

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

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

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

lemma *mu-csp-basic-refine*:
assumes
 $P \text{ is CSP } Q \text{ is NCSP } Q \text{ is Productive } \text{pre}_R(P) = \text{true}_r \text{pre}_R(Q) = \text{true}_r$
 $\text{peri}_R P \sqsubseteq \text{peri}_R Q$
 $\text{peri}_R P \sqsubseteq \text{post}_R Q ;; \text{peri}_R P$
shows $P \sqsubseteq (\mu_C X \cdot Q ;; X)$

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

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

proof (*rule UINF-refines'*)

fix *i*

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

proof (*induct i*)

```

    case 0
    then show ?case by (simp add: assms)
next
case (Suc i)
then show ?case
  by (meson assms(6) assms(7) semilattice-sup-class.le-sup-iff upower-inductl)
qed
qed
qed

lemma CRD-mu-basic-refine:
  fixes P :: 'e list  $\Rightarrow$  'e set  $\Rightarrow$  's upred
  assumes
    Q is NCSP Q is Productive  $pre_R(Q) = true_r$ 
     $[P \ t \ r]_{S<} \llbracket (t, r) \rightarrow (\&tt, \$ref')_u \rrbracket \sqsubseteq peri_R \ Q$ 
     $[P \ t \ r]_{S<} \llbracket (t, r) \rightarrow (\&tt, \$ref')_u \rrbracket \sqsubseteq post_R \ Q \ ;;_h [P \ t \ r]_{S<} \llbracket (t, r) \rightarrow (\&tt, \$ref')_u \rrbracket$ 
  shows  $[true \vdash P \ trace \ refs \mid R]_C \sqsubseteq (\mu_C \ X \cdot Q \ ;; \ X)$ 
proof (rule mu-csp-basic-refine, simp-all add: msubst-pair assms closure alpha rdes rpred Healthy-if R1-false)
  show  $[P \ trace \ refs]_{S<} \llbracket trace \rightarrow \&tt \rrbracket \llbracket refs \rightarrow \$ref' \rrbracket \sqsubseteq peri_R \ Q$ 
  using assms by (simp add: msubst-pair)
  show  $[P \ trace \ refs]_{S<} \llbracket trace \rightarrow \&tt \rrbracket \llbracket refs \rightarrow \$ref' \rrbracket \sqsubseteq post_R \ Q \ ;; [P \ trace \ refs]_{S<} \llbracket trace \rightarrow \&tt \rrbracket \llbracket refs \rightarrow \$ref' \rrbracket$ 
  using assms by (simp add: msubst-pair)
qed

```

9.2 Example action expansion

```

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

```

```

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

```

```

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

```

```

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

```

end

10 Linking to the Failures-Divergences Model

```

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

```

10.1 Failures-Divergences Semantics

The following functions play a similar role to those in Roscoe's CSP semantics, and are calculated from the Circus reactive design semantics. A major difference is that these three functions account for state. Each divergence, trace, and failure is subject to an initial state. Moreover, the traces are terminating traces, and therefore also provide a final state following the given

interaction. A more subtle difference from the Roscoe semantics is that the set of traces do not include the divergences. The same semantic information is present, but we construct a direct analogy with the pre-, peri- and postconditions of our reactive designs.

definition *divergences* $:: ('σ, 'φ) \text{ action} \Rightarrow 'σ \Rightarrow 'φ \text{ list set } (dv[-] - [0,100] \ 100) \text{ where}$
 $[upred-defs]: \text{divergences } P \ s = \{t \mid t. '(\neg_r \text{ pre}_R(P)) \llbracket \langle s \rangle, \langle \rangle, \langle t \rangle \rrbracket / \$st, \$tr, \$tr' \}$

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

definition *failures* $:: ('σ, 'φ) \text{ action} \Rightarrow 'σ \Rightarrow ('φ \text{ list} \times 'φ \text{ set}) \text{ set } (fl[-] - [0,100] \ 100) \text{ where}$
 $[upred-defs]: \text{failures } P \ s = \{(t, r) \mid t \ r. '(pre_R(P) \wedge peri_R(P)) \llbracket \langle r \rangle, \langle s \rangle, \langle \rangle, \langle t \rangle \rrbracket / \$ref', \$st, \$tr, \$tr' \}$

lemma *trace-divergence-disj*:

assumes $P \text{ is NCSP } (t, s') \in tr[P] \ s \ t \in dv[P] \ s$
shows *False*
using *assms(2,3)*
by (*simp add: traces-def divergences-def, rdes-simp cls:assms, rel-auto*)

lemma *preR-refine-divergences*:

assumes $P \text{ is NCSP } Q \text{ is NCSP } \wedge s. dv[P] \ s \subseteq dv[Q] \ s$
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 \langle s \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle t \rangle] \dagger pre_R \ Q'$

with a **show** $'[$st \mapsto_s \langle s \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle t \rangle] \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 \langle s \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle t \rangle] \dagger pre_R \ P'$

hence $\neg '[$st \mapsto_s \langle s \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle t \rangle] \dagger CRC(pre_R \ P)'$

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

hence $'[$st \mapsto_s \langle s \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle t \rangle] \dagger (\neg_r \ CRC(pre_R \ P))'$

by (*rel-auto*)

hence $'[$st \mapsto_s \langle s \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle t \rangle] \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 $P \text{ is NCSP } Q \text{ is NCSP } \wedge s. dv[P] \ s = dv[Q] \ s$

shows $pre_R(P) = pre_R(Q)$

by (*metis assms dual-order.antisym order-refl preR-refine-divergences*)

lemma *periR-refine-failures*:

assumes $P \text{ is NCSP } Q \text{ is NCSP } \wedge s. fl[Q] \ s \subseteq fl[P] \ s$

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 \langle r' \rangle, \$st \mapsto_s \langle s \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle t \rangle] \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 \langle r' \rangle, \$st \mapsto_s \langle s \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle t \rangle] \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*:

$tr[Skip]s = \{(\square, s)\}$
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*:
 $dv\llbracket \text{Skip} \rrbracket s = \{\}$
by (*simp add: divergences-def, rdes-calc*)

lemma *traces-Stop*:
 $tr\llbracket \text{Stop} \rrbracket s = \{\}$
by (*simp add: traces-def, rdes-calc*)

lemma *failures-Stop*:
 $fl\llbracket \text{Stop} \rrbracket s = \{(\llbracket \cdot \rrbracket, E) \mid E. \text{True}\}$
by (*simp add: failures-def, rdes-calc, rel-auto*)

lemma *divergences-Stop*:
 $dv\llbracket \text{Stop} \rrbracket s = \{\}$
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*: $fl\llbracket \text{Miracle} \rrbracket s = \{\}$
by (*simp add: failures-def rdes closure usubst*)

lemma *divergences-Miracle*: $dv\llbracket \text{Miracle} \rrbracket s = \{\}$
by (*simp add: divergences-def rdes closure usubst*)

lemma *failures-Chaos*: $fl\llbracket \text{Chaos} \rrbracket s = \{\}$
by (*simp add: failures-def rdes, rel-auto*)

lemma *divergences-Chaos*: $dv\llbracket \text{Chaos} \rrbracket s = UNIV$
by (*simp add: divergences-def rdes, rel-auto*)

lemma *traces-Chaos*: $tr\llbracket \text{Chaos} \rrbracket s = \{\}$
by (*simp add: traces-def rdes closure usubst*)

lemma *divergences-cond*:
assumes P is NCSP Q is NCSP
shows $dv\llbracket P \triangleleft b \triangleright_R Q \rrbracket s = (\text{if } (\llbracket b \rrbracket_e s) \text{ then } dv\llbracket P \rrbracket s \text{ else } dv\llbracket Q \rrbracket s)$
by (*rdes-simp cls: assms, simp add: divergences-def traces-def rdes closure rpred assms, rel-auto*)

lemma *traces-cond*:
assumes P is NCSP Q is NCSP
shows $tr\llbracket P \triangleleft b \triangleright_R Q \rrbracket s = (\text{if } (\llbracket b \rrbracket_e s) \text{ then } tr\llbracket P \rrbracket s \text{ else } tr\llbracket Q \rrbracket s)$
by (*rdes-simp cls: assms, simp add: divergences-def traces-def rdes closure rpred assms, rel-auto*)

lemma *failures-cond*:
assumes P is NCSP Q is NCSP

shows $\text{fl}[P \triangleleft b \triangleright_R Q]s = (\text{if } (\llbracket b \rrbracket_e s) \text{ then } \text{fl}[P]s \text{ else } \text{fl}[Q]s)$
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 $\text{dv}[g \&_u P]s = (\text{if } (\llbracket g \rrbracket_e s) \text{ then } \text{dv}[g \&_u P]s \text{ else } \{\})$

by (*rdes-simp cls: assms, simp add: divergences-def traces-def rdes closure rpred assms, rel-auto*)

lemma *traces-do:* $\text{tr}[\text{do}_C(e)]s = \{(\llbracket e \rrbracket_e s, s)\}$

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

lemma *failures-do:* $\text{fl}[\text{do}_C(e)]s = \{(\llbracket \cdot \rrbracket, E) \mid E. \llbracket e \rrbracket_e s \notin E\}$

by (*rdes-simp, simp add: failures-def rdes closure rpred usubst, rel-auto*)

lemma *divergences-do:* $\text{dv}[\text{do}_C(e)]s = \{\}$

by (*rel-auto*)

lemma *divergences-seq:*

fixes $P :: ('s, 'e)$ action

assumes P is NCSP Q is NCSP

shows $\text{dv}[P ;; Q]s = \text{dv}[P]s \cup \{t_1 @ t_2 \mid t_1 \ t_2 \ s_0. (t_1, s_0) \in \text{tr}[P]s \wedge t_2 \in \text{dv}[Q]s_0\}$

(**is** ?lhs = ?rhs)

oops

lemma *traces-seq:*

fixes $P :: ('s, 'e)$ action

assumes P is NCSP Q is NCSP

shows $\text{tr}[P ;; Q]s =$

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

(**is** ?lhs = ?rhs)

proof

show ?lhs \subseteq ?rhs

proof (*rdes-expand cls: assms, simp add: traces-def divergences-def rdes closure assms rdes-def unrest rpred usubst, auto*)

fix $t :: 'e$ list **and** $s' :: 's$

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

assume

$a1: \sigma \vdash (\text{post}_R P ;; \text{post}_R Q)'$ **and**

$a2: [\$st \mapsto_s \llbracket s \rrbracket, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \llbracket t \rrbracket] \vdash \text{pre}_R P'$ **and**

$a3: [\$st \mapsto_s \llbracket s \rrbracket, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \llbracket t \rrbracket] \vdash (\text{post}_R P \text{ wp}_r \text{pre}_R Q)'$

from $a1$ **have** $\sigma \vdash (\exists tr_0. ((\text{post}_R P) \llbracket \llbracket tr_0 \rrbracket \rrbracket / \$tr') ;; (\text{post}_R Q) \llbracket \llbracket tr_0 \rrbracket \rrbracket / \$tr) \wedge \llbracket tr_0 \rrbracket \leq_u \tr'

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

then obtain tr_0 **where** $p1: \sigma \vdash ((\text{post}_R P) \llbracket \llbracket tr_0 \rrbracket \rrbracket / \$tr') ;; (\text{post}_R Q) \llbracket \llbracket tr_0 \rrbracket \rrbracket / \tr **and** $tr_0: tr_0$

$\leq t$

apply (*simp add: usubst*)

apply (*erule taut-shEx-elim*)

apply (*simp add: unrest-all-circus-vars-st-st' closure unrest assms*)

apply (*rel-auto*)

done

from $p1$ **have** $\sigma \vdash (\exists st_0. (\text{post}_R P) \llbracket \llbracket tr_0 \rrbracket \rrbracket / \$tr' \llbracket \llbracket st_0 \rrbracket \rrbracket / \$st' ;; (\text{post}_R Q) \llbracket \llbracket tr_0 \rrbracket \rrbracket / \$tr \llbracket \llbracket st_0 \rrbracket \rrbracket / \$st)'$

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

then obtain s_0 **where** $\sigma \vdash ((\text{post}_R P) \llbracket \llbracket s_0 \rrbracket \rrbracket, \llbracket tr_0 \rrbracket \rrbracket / \$st', \$tr') ;; (\text{post}_R Q) \llbracket \llbracket s_0 \rrbracket \rrbracket, \llbracket tr_0 \rrbracket \rrbracket / \$st, \$tr)$

apply (*simp add: usubst*)

apply (erule taut-shEx-elim)
 apply (simp add: unrest-all-circus-vars-st-st' closure unrest assms)
 apply (rel-auto)
 done
 hence '([$\$st \mapsto_s \ll s \gg$, $\$st' \mapsto_s \ll s_0 \gg$, $\$tr \mapsto_s \langle \rangle$, $\$tr' \mapsto_s \ll tr_0 \gg$] \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: segr-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 [$\$tr \mapsto_s \ll tr_0 \gg$, $\$tr' \mapsto_s \ll tr_0 \gg + (\ll t \gg - \ll tr_0 \gg)$] \dagger $post_R Q$ '
 using $tr0$ by (rel-auto)
 hence '[$\$st \mapsto_s \ll s_0 \gg$, $\$st' \mapsto_s \ll s' \gg$] \dagger [$\$tr \mapsto_s 0$, $\$tr' \mapsto_s \ll t \gg - \ll tr_0 \gg$] \dagger $post_R Q$ '
 by (simp add: R2-subst-tr closure assms)
 hence $postQ$: '[$\$st \mapsto_s \ll s_0 \gg$, $\$st' \mapsto_s \ll s' \gg$, $\$tr \mapsto_s \langle \rangle$, $\$tr' \mapsto_s \ll t - tr_0 \gg$] \dagger $post_R Q$ '
 by (rel-auto)
 have $preP$: '[$\$st \mapsto_s \ll s \gg$, $\$tr \mapsto_s \langle \rangle$, $\$tr' \mapsto_s \ll tr_0 \gg$] \dagger $pre_R P$ '
 proof -
 have $(pre_R P)[[0, \ll tr_0 \gg / \$tr, \$tr']] \sqsubseteq (pre_R P)[[0, \ll t \gg / \$tr, \$tr']]$
 by (simp add: RC-prefix-refine closure assms $tr0$)
 hence [$\$st \mapsto_s \ll s \gg$, $\$tr \mapsto_s \langle \rangle$, $\$tr' \mapsto_s \ll tr_0 \gg$] \dagger $pre_R P \sqsubseteq$ [$\$st \mapsto_s \ll s \gg$, $\$tr \mapsto_s \langle \rangle$, $\$tr' \mapsto_s \ll t \gg$] \dagger $pre_R P$
 by (rel-auto)
 thus ?thesis
 by (simp add: taut-refine-impl a2)
 qed
 have $preQ$: '[$\$st \mapsto_s \ll s_0 \gg$, $\$tr \mapsto_s \langle \rangle$, $\$tr' \mapsto_s \ll t - tr_0 \gg$] \dagger $pre_R Q$ '
 proof -
 from $postP$ a3 have '[$\$st \mapsto_s \ll s_0 \gg$, $\$tr \mapsto_s \ll tr_0 \gg$, $\$tr' \mapsto_s \ll t \gg$] \dagger $pre_R Q$ '
 apply (simp add: wp-rea-def)
 apply (rel-auto)
 using $tr0$ apply blast+
 done
 hence '[$\$st \mapsto_s \ll s_0 \gg$] \dagger [$\$tr \mapsto_s \ll tr_0 \gg$, $\$tr' \mapsto_s \ll tr_0 \gg + (\ll t \gg - \ll tr_0 \gg)$] \dagger $pre_R Q$ '
 by (rel-auto)
 hence '[$\$st \mapsto_s \ll s_0 \gg$] \dagger [$\$tr \mapsto_s 0$, $\$tr' \mapsto_s \ll t \gg - \ll tr_0 \gg$] \dagger $pre_R Q$ '
 by (simp add: R2-subst-tr closure assms)
 thus ?thesis
 by (rel-auto)
 qed
 from a2 have $ndiv$: \neg '[$\$st \mapsto_s \ll s \gg$, $\$tr \mapsto_s \langle \rangle$, $\$tr' \mapsto_s \ll t \gg$] \dagger $(\neg_r pre_R P)$ '
 by (rel-auto)
 have $t\text{-minus-}tr0$: $tr_0 @ (t - tr_0) = t$
 using append-minus $tr0$ by blast
 from a3
 have wpr : $\bigwedge t_0 s_1.$

$$\begin{aligned} & \text{'}[\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_0 \gg] \dagger pre_R P' \implies \\ & \text{'}[\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_0 \gg] \dagger post_R P' \implies \\ & t_0 \leq t \implies \text{'}[\$st \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t - t_0 \gg] \dagger (\neg_r pre_R Q)' \implies False \end{aligned}$$

proof –

fix $t_0 \ s_1$

assume b :

$$\begin{aligned} & \text{'}[\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_0 \gg] \dagger pre_R P' \\ & \text{'}[\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_0 \gg] \dagger post_R P' \\ & t_0 \leq t \\ & \text{'}[\$st \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t - t_0 \gg] \dagger (\neg_r pre_R Q)' \end{aligned}$$

from $a3$ **have** c : $\forall (s_0, t_0) \cdot \ll t_0 \gg \leq_u \ll t \gg$

$$\begin{aligned} & \wedge [\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t_0 \gg] \dagger post_R P \\ & \implies [\$st \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \ll t \gg - \ll t_0 \gg] \dagger pre_R Q' \end{aligned}$$

by (*simp add: wp-rea-circus-form-alt[of post_R P pre_R Q] closure assms unrest usubst*)

(*rel-simp*)

from $c \ b(2-4)$ **show** *False*

by (*rel-auto*)

qed

show $\exists t_1 \ t_2$.

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

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 $t_1 \ t_2 :: 'e \ list$ **and** $s_0 \ s' :: 's$

assume

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

from $a1$ **have** $preP$: ‘ $[\$st \mapsto_s \langle s \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle t_1 @ t_2 \rangle] \dagger (pre_R P)$ ’
by (*simp add: taut-not unrest-all-circus-vars-st assms closure unrest, rel-auto*)

have ‘ $[\$st \mapsto_s \langle s_0 \rangle, \$st' \mapsto_s \langle s' \rangle, \$tr \mapsto_s \langle t_1 \rangle, \$tr' \mapsto_s \langle t_1 \rangle + \langle t_2 \rangle] \dagger post_R Q$ ’
proof –
have $[\$st \mapsto_s \langle s_0 \rangle, \$st' \mapsto_s \langle s' \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle t_2 \rangle] \dagger post_R Q =$
 $[\$st \mapsto_s \langle s_0 \rangle, \$st' \mapsto_s \langle s' \rangle] \dagger [\$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle t_2 \rangle] \dagger post_R Q$
by *rel-auto*
also have $\dots = [\$st \mapsto_s \langle s_0 \rangle, \$st' \mapsto_s \langle s' \rangle] \dagger [\$tr \mapsto_s \langle t_1 \rangle, \$tr' \mapsto_s \langle t_1 \rangle + \langle t_2 \rangle] \dagger post_R Q$
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 $a6'$: $\bigwedge t s_1. [t \leq t_1 @ t_2; ' [\$st \mapsto_s \langle s \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle t \rangle] \dagger pre_R P'; ' [\$st \mapsto_s \langle s \rangle, \$st' \mapsto_s \langle s_1 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle t \rangle] \dagger post_R P'] \implies$
 $' [\$st \mapsto_s \langle s_1 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle (t_1 @ t_2) - t \rangle] \dagger pre_R Q'$
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 $\bigwedge s_1 t_0. [t_0 \leq t_1 @ t_2; ' [\$st \mapsto_s \langle s \rangle, \$st' \mapsto_s \langle s_1 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle t_0 \rangle] \dagger post_R P'$
 $\implies ' [\$st \mapsto_s \langle s_1 \rangle, \$tr \mapsto_s \langle \rangle, \$tr' \mapsto_s \langle (t_1 @ t_2) - t_0 \rangle] \dagger pre_R Q'$
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 $(pre_R P)[[0, \langle t_0 \rangle / \$tr, \$tr']] \sqsubseteq (pre_R P)[[0, \langle t_1 @ t_2 \rangle / \$tr, \$tr']]$
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 t0 s1] show '[ $\$st \mapsto_s \ll s_1 \gg$ ,  $\$tr \mapsto_s \langle \rangle$ ,  $\$tr' \mapsto_s \ll (t_1 @ t_2) - t_0 \gg$ ]  $\dagger$  preR
Q'
    by (simp)
qed

thus ?thesis
  apply (simp-all add: wp-rea-circus-form-alt assms closure unrest usubst rea-impl-alt-def)
  apply (simp add: R1-def usubst tcontr-alt-def)
  apply (auto intro!: taut-shAll-intro-2)
  apply (rule taut-impl-intro)
  apply (simp add: unrest-all-circus-vars-st-st' unrest closure assms)
  apply (rel-simp)
done
qed
show '([ $\$st \mapsto_s \ll s \gg$ ,  $\$tr \mapsto_s \langle \rangle$ ,  $\$tr' \mapsto_s \ll t_1 @ t_2 \gg$ ]  $\dagger$  preR P  $\wedge$ 
  [ $\$st \mapsto_s \ll s \gg$ ,  $\$tr \mapsto_s \langle \rangle$ ,  $\$tr' \mapsto_s \ll t_1 @ t_2 \gg$ ]  $\dagger$  (postR P wpr preR Q))  $\wedge$ 
  [ $\$st \mapsto_s \ll s \gg$ ,  $\$st' \mapsto_s \ll s' \gg$ ,  $\$tr \mapsto_s \langle \rangle$ ,  $\$tr' \mapsto_s \ll t_1 @ t_2 \gg$ ]  $\dagger$  (postR P ;; postR 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 $tr \ll a \gg \rightarrow_C P \mid s = \{(a \# t, s') \mid t s'. (t, s') \in tr \ll P \mid s\}$
 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]: $CDF = \mathbf{R}_s(\text{true}_r \vdash (\bigcap (s, t, E, e) \cdot \mathcal{E}(\ll s \gg, \ll t \gg, \ll \text{insert } e \ E \gg))) \diamond \text{true}_r$

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

lemma Skip-deadlock-free: $CDF \sqsubseteq \text{Skip}$
 by (rdes-refine)

end

11 Meta-theory for Stateful-Failure Reactive Designs

```
theory utp-sf-rdes
imports
  utp-sfrd-core
  utp-sfrd-rel
  utp-sfrd-healths
  utp-sfrd-contracts
  utp-sfrd-extchoice
  utp-sfrd-prog
  utp-sfrd-recursion
  utp-sfrd-fdsem
begin end
```

References

- [1] S. Foster, F. Zeyda, and J. Woodcock. Unifying heterogeneous state-spaces with lenses. In *ICTAC*, LNCS 9965. Springer, 2016.
- [2] M. V. M. Oliveira. *Formal Derivation of State-Rich Reactive Programs using Circus*. PhD thesis, Department of Computer Science - University of York, UK, 2006. YCST-2006-02.