

# Stateful-Failure Reactive Designs in Isabelle/UTP

Simon Foster      James Baxter      Ana Cavalcanti      Jim Woodcock

March 27, 2020

## 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

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Stateful-Failure Core Types</b>	<b>3</b>
2.1	SFRD Alphabet . . . . .	3
2.2	Basic laws . . . . .	3
2.3	Unrestriction laws . . . . .	4
<b>3</b>	<b>Stateful-Failure Reactive Relations</b>	<b>5</b>
3.1	Healthiness Conditions . . . . .	5
3.2	Closure Properties . . . . .	8
3.3	Introduction laws . . . . .	14
3.4	UTP Theory . . . . .	14
3.5	Weakest Precondition . . . . .	15
3.6	Trace Substitution . . . . .	17
3.7	Initial Interaction . . . . .	18
3.8	Enabled Events . . . . .	19
3.9	Completed Trace Interaction . . . . .	20
3.10	Assumptions . . . . .	25
3.11	Downward closure of refusals . . . . .	25
3.12	Renaming . . . . .	27
<b>4</b>	<b>Stateful-Failure Healthiness Conditions</b>	<b>28</b>

<b>5</b>	<b>Definitions</b>	<b>29</b>
5.1	Healthiness condition properties . . . . .	29
5.2	CSP theories . . . . .	41
5.3	Algebraic laws . . . . .	42
<b>6</b>	<b>Stateful-Failure Reactive Contracts</b>	<b>42</b>
<b>7</b>	<b>External Choice</b>	<b>44</b>
7.1	Definitions and syntax . . . . .	44
7.2	Basic laws . . . . .	45
7.3	Algebraic laws . . . . .	45
7.4	Reactive design calculations . . . . .	45
7.5	Productivity and Guardedness . . . . .	51
7.6	Algebraic laws . . . . .	53
<b>8</b>	<b>Stateful-Failure Programs</b>	<b>56</b>
8.1	Conditionals . . . . .	56
8.2	Guarded commands . . . . .	56
8.3	Alternation . . . . .	56
8.4	Specification Statement . . . . .	57
8.5	Assumptions . . . . .	57
8.6	While Loops . . . . .	57
8.7	Iteration Construction . . . . .	58
8.8	Assignment . . . . .	61
8.9	Assignment with update . . . . .	62
8.10	State abstraction . . . . .	63
8.11	Guards . . . . .	63
8.12	Basic events . . . . .	67
8.13	Event prefix . . . . .	69
8.14	Guarded external choice . . . . .	71
8.15	Input prefix . . . . .	71
8.16	Renaming . . . . .	72
8.17	Algebraic laws . . . . .	73
<b>9</b>	<b>Recursion in Stateful-Failures</b>	<b>75</b>
9.1	Fixed-points . . . . .	75
9.2	Example action expansion . . . . .	76
<b>10</b>	<b>Linking to the Failures-Divergences Model</b>	<b>77</b>
10.1	Failures-Divergences Semantics . . . . .	77
10.2	Circus Operators . . . . .	79
10.3	Deadlock Freedom . . . . .	85
<b>11</b>	<b>Meta-theory for Stateful-Failure Reactive Designs</b>	<b>85</b>

# 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 ('σ, 'φ) sfrd-vars = ('φ list, 'σ) rsp-vars +
  ref :: 'φ set
```

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

```
type-synonym ('σ, 'φ) sfrd = ('σ, 'φ) sfrd-vars
type-synonym ('σ, 'φ) action = ('σ, 'φ) sfrd hrel
type-synonym 'φ csp = (unit, 'φ) sfrd
type-synonym 'φ process = 'φ 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) ('σ, 'φ) sfrd <= (type) ('σ, 'φ) sfrd-vars
(type) ('σ, 'φ) action <= (type) ('σ, 'φ) sfrd hrel
(type) 'φ process <= (type) (unit, 'φ) action
```

```
notation sfrd-vars.moreL (ΣC)
```

```
declare des-vars.splits [alpha-splits del]
declare rp-vars.splits [alpha-splits del]
declare des-vars.splits [alpha-splits del]
declare rsp-vars.splits [alpha-splits del]
declare rsp-vars.splits [alpha-splits]
declare rp-vars.splits [alpha-splits]
declare des-vars.splits [alpha-splits]
```

### 2.2 Basic laws

```
term U($tr' = $tr @ [[a]S<])
```

```
lemma R2c-tr-ext: R2c (U($tr' = $tr @ [[a]S<])) = U($tr' = $tr @ [[a]S<])
  by (rel-auto)
```

```
lemma circus-alpha-bij-lens:
```

```
  bij-lens ({ $ok, $ok', $wait, $wait', $tr, $tr', $st, $st', $ref, $ref' }α :: - ⇒ ('s, 'e) sfrd × ('s, 'e) sfrd)
  by (unfold-locales, lens-simp+)
```

## 2.3 Unrestriction laws

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

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

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

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

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

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

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

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

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

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

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

**lemma** *unrest-circus-alpha*:  
**fixes**  $P :: ('e, 't) \text{ action}$   
**assumes**  
 $\$ok \# P \ \$ok' \# P \ \$wait \# P \ \$wait' \# P \ \$tr \# P$   
 $\$tr' \# P \ \$st \# P \ \$st' \# P \ \$ref \# P \ \$ref' \# P$   
**shows**  $\Sigma \# P$   
**by** (*rule bij-lens-unrest-all[OF circus-alpha-bij-lens], simp add: unrest assms*)

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

**lemma** *unrest-all-circus-vars-st-st'*:  
**fixes**  $P :: ('s, 'e) \text{ action}$   
**assumes**  $\$ok \# P \ \$ok' \# P \ \$wait \# P \ \$wait' \# P \ \$ref \# P \ \$ref' \# P \ \Sigma \# s \ \Sigma \# s' \ \Sigma \# t \ \Sigma \# t'$   
**shows**  $\Sigma \# [\$st \mapsto_s s, \$st' \mapsto_s s', \$tr \mapsto_s t, \$tr' \mapsto_s t'] \dagger P$

```

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

lemma unrest-any-circus-var:
  fixes  $P :: ('s, 'e) \text{ action}$ 
  assumes  $\$ok \# P \$ok' \# P \$wait \# P \$wait' \# P \$ref \# P \$ref' \# P \Sigma \# s \Sigma \# s' \Sigma \# t \Sigma \# t'$ 
  shows  $x \# [\$st \mapsto_s s, \$st' \mapsto_s s', \$tr \mapsto_s t, \$tr' \mapsto_s t'] \uparrow P$ 
  by (simp add: unrest-all-var unrest-all-circus-vars-st-st' assms)

lemma unrest-any-circus-var-st:
  fixes  $P :: ('s, 'e) \text{ action}$ 
  assumes  $\$ok \# P \$ok' \# P \$wait \# P \$wait' \# P \$ref \# P \$ref' \# P \$st' \# P \Sigma \# s \Sigma \# t \Sigma \# t'$ 
  shows  $x \# [\$st \mapsto_s s, \$tr \mapsto_s t, \$tr' \mapsto_s t'] \uparrow P$ 
  by (simp add: unrest-all-var unrest-all-circus-vars-st assms)

end

```

### 3 Stateful-Failure Reactive Relations

```

theory utp-sfrd-rel
  imports utp-sfrd-core
begin

```

#### 3.1 Healthiness Conditions

CSP Reactive Relations

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

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

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

**lemma** *Continuous-CRR* [*closure*]: *Continuous CRR*  
**by** (*rel-blast*)

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

**lemma** *CRR-form*:  $CRR(P) = (\exists \{\$ok, \$ok', \$wait, \$wait', \$ref\} \cdot (\exists tt_0 \cdot P[\ll[]\gg/\$tr][\ll tt_0 \gg/\$tr'] \wedge \$tr' =_u \$tr \hat{\phantom{u}}_{tt_0} \ll tt_0 \gg))$

by (rel-auto; fastforce)

**lemma** *CRR-seqr-form*:

$CRR(P) ;; CRR(Q) =$   
 $(\exists tt_1 \cdot \exists tt_2 \cdot ((\exists \{ \$ok, \$ok', \$wait, \$wait', \$ref \} \cdot P) [\llbracket \cdot \rrbracket / \$tr] [\llbracket tt_1 \rrbracket / \$tr'] ;;$   
 $(\exists \{ \$ok, \$ok', \$wait, \$wait', \$ref \} \cdot Q) [\llbracket \cdot \rrbracket / \$tr] [\llbracket tt_2 \rrbracket / \$tr'] \wedge \$tr' =_u \$tr \wedge_u$   
 $\llbracket tt_1 \rrbracket \wedge_u \llbracket tt_2 \rrbracket))$   
 by (simp add: CRR-form, rel-auto; fastforce)

CSP Reactive Finalisers

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

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

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

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

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

**lemma** *CRF-implies-CRR* [closure]:  
 assumes *P is CRF* shows *P is CRR*

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

qed

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

**lemma** *crel-skip-CRR* [closure]:  $II_c$  *is CRF*  
 by (rel-auto)

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

**lemma** *crel-skip-left-unit* [rpred]:  
 assumes *P is CRR*  
 shows  $II_c ;; P = P$

**proof** –  
 have  $II_c ;; CRR(P) = CRR(P)$  by (rel-auto)  
 thus ?thesis by (simp add: Healthy-if assms)  
 qed

**lemma** *crel-skip-right-unit* [rpred]:  
 assumes *P is CRF*

**shows**  $P \;; \; II_c = P$   
**proof** –  
**have**  $CRF(P) \;; \; II_c = CRF(P)$  **by** (*rel-auto*)  
**thus** *?thesis* **by** (*simp add: Healthy-if assms*)  
**qed**

## CSP Reactive Conditions

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

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

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

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

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

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

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

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

**lemma** *ex-ref'-RR-closed* [*closure*]:  
**assumes** *P is RR*  
**shows**  $(\exists \$ref' \cdot P)$  *is RR*

**proof** –  
**have**  $RR (\exists \$ref' \cdot RR(P)) = (\exists \$ref' \cdot RR(P))$   
**by** (*rel-auto*)  
**thus** *?thesis*  
**by** (*metis Healthy-def assms*)  
**qed**

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

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

### 3.2 Closure Properties

**lemma** *CRR-implies-RR* [closure]:

assumes *P* is CRR

shows *P* is RR

**proof** –

have  $RR(CRR(P)) = CRR(P)$

by (*rel-auto*)

thus ?thesis

by (*metis Healthy-def' assms*)

qed

**lemma** *CRC-intro''*:

assumes *P* is CRR *P* is RC1

shows *P* is CRC

by (*simp add: CRC-intro' CRR-implies-RR RC-intro' assms*)

**lemma** *CRC-implies-RR* [closure]:

assumes *P* is CRC

shows *P* is RR

**proof** –

have  $RR(CRC(P)) = CRC(P)$

by (*rel-auto*)

(*metis (no-types, lifting) Prefix-Order.prefixE Prefix-Order.prefixI append.assoc append-minus*) +

thus ?thesis

by (*metis Healthy-def assms*)

qed

**lemma** *CRC-implies-RC* [closure]:

assumes *P* is CRC

shows *P* is RC

**proof** –

have  $RC1(CRC(P)) = CRC(P)$

by (*rel-auto, meson dual-order.trans*)

thus ?thesis

by (*simp add: CRC-implies-RR Healthy-if RC1-def RC-intro assms*)

qed

**lemma** *CRR-unrest-ref* [unrest]:  $P \text{ is CRR} \implies \$ref \# P$

by (*metis CRR-def CRR-implies-RR Healthy-def in-var-uvar ref-vwb-lens unrest-as-exists*)

**lemma** *CRF-unrest-ref'* [unrest]:

assumes *P* is CRF

shows  $\$ref' \# P$

**proof** –

have  $\$ref' \# CRF(P)$  by (*simp add: CRF-def unrest*)

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

qed

**lemma** *CRC-implies-CRR* [closure]:

assumes *P* is CRC

shows *P* is CRR

apply (*rule CRR-intro*)

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

apply (*metis CRC-def CRC-implies-RC Healthy-def assms in-var-uvar ref-vwb-lens unrest-as-exists*)

done



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

**lemma** *rea-true-CRR* [*closure*]: *true<sub>r</sub> is CRR*  
 by (*rel-auto*)

**lemma** *rea-true-CRC* [*closure*]: *true<sub>r</sub> 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]<sub>S<</sub> is CRR*  
 by (*rel-auto*)

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

**lemma** *st-post-CRR* [*closure*]: *[b]<sub>S></sub> is CRR*  
 by (*rel-auto*)

**lemma** *st-cond-CRC* [*closure*]: *[P]<sub>S<</sub> is CRC*  
 by (*rel-auto*)

**lemma** *st-cond-CRF* [*closure*]: *[b]<sub>S<</sub> is CRF*  
 by (*rel-auto*)

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

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

**lemma** *st-subst-CRC-closed* [closure]:

assumes  $P$  is CRC

shows  $(\sigma \uparrow_S P)$  is CRC

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

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

$\llbracket P \text{ is CRC}; Q \text{ is CRC} \rrbracket \implies (P \wedge Q) \text{ is CRC}$

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

**lemma** *conj-CRF-closed* [closure]:  $\llbracket P \text{ is CRF}; Q \text{ is CRF} \rrbracket \implies (P \wedge Q) \text{ is CRF}$

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

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

$\llbracket P \text{ is CRC}; Q \text{ is CRC} \rrbracket \implies (P \vee Q) \text{ is CRC}$

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

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

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

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

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

by (rel-auto)

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

by (rel-auto)

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

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

by (rel-blast)

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

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

by (rel-blast)

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

assumes  $P$  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  $(\exists x. P(x))$  is CRR  
**proof** –  
 have  $CRR(\exists x. CRR(P(x))) = (\exists x. CRR(P(x)))$   
 by (rel-auto)  
 thus ?thesis  
 by (metis Healthy-def assms shEx-cong)  
**qed**

**lemma** *USUP-ind-CRR-closed* [closure]:  
 assumes  $\bigwedge i. P\ i$  is CRR  
 shows  $(\bigsqcup i. P(i))$  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  $(\bigcap i. P(i))$  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**  $P$  is  $RR$   $Q$  is  $RR$   
**shows**  $(P \triangleleft b \triangleright_R Q)$  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**  $P$  is  $CRR$   $Q$  is  $RR$   
**shows**  $(P ;; Q)$  is  $CRR$   
**by** (rule  $CRR$ -intro, simp-all add: unrest assms closure)

**lemma** *seq-CRF-closed* [closure]:  
**assumes**  $P$  is  $CRF$   $Q$  is  $CRF$   
**shows**  $(P ;; Q)$  is  $CRF$   
**by** (rule  $CRF$ -intro, simp-all add: unrest assms closure)

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

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

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

**lemma** *USUP-ind-CRC-closed* [closure]:  
 $\llbracket \bigwedge i. P \ i \text{ is } CRC \rrbracket \implies (\bigsqcup i. P \ i) \text{ is } CRC$   
**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**  $\$ok \# P \ \$wait \# P \ \$ref \# P$   
**shows**  $U(\$tr' = \$tr \ @ \llbracket \llbracket a \rrbracket \rrbracket \wedge \$st' = \$st) ;; P = P \llbracket U(\$tr \ @ \llbracket \llbracket a \rrbracket \rrbracket) / \$tr \rrbracket$   
**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**  $(\$tr' =_u \$tr \wedge \llbracket \langle \sigma \rangle_a \rrbracket_S) ;; P = \llbracket \sigma \rrbracket_{S\sigma} \dagger P$   
**using** assms **by** (rel-auto, meson)

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

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

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

by (simp add: Healthy-def RR-msubst-tt)

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

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

assumes *P* 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  $\implies R4(P)$  is CRR

by (simp add: R4-def conj-less-tr-RR-closed)

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

assumes *P* is CRR

shows *R5(P)* is CRR

**proof** –

have *R5(CRR(P))* is CRR

by (rel-auto; blast)

thus ?thesis

by (simp add: assms Healthy-if)

qed

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

assumes *P* is CRR

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

**proof** –

have  $CRR(CRR(P) \wedge \$tr' =_u \$tr) = (CRR(P) \wedge \$tr' =_u \$tr)$

by (rel-auto, blast+)

thus ?thesis

by (metis Healthy-def assms)

qed

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

*P* is CRC  $\implies (\forall \$ref \cdot P)$  is CRC

by (simp add: CRC-implies-CRR CRR-unrest-ref all-unrest)

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

*P* is CRR  $\implies (\exists \$ref \cdot P)$  is CRR

by (simp add: CRR-unrest-ref ex-unrest)

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

*P* is CRR  $\implies (\exists \$st' \cdot P)$  is CRR

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

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

*P* is CRR  $\implies (\exists \$ref' \cdot P)$  is CRR

using CRR-implies-RR CRR-intro CRR-unrest-ref ex-ref'-RR-closed out-in-indep unrest-ex-diff by

blast

### 3.3 Introduction laws

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

**lemma** *CRR-refine-ext*:

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

**lemma** *CRR-eq-ext*:

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

**lemma** *CRR-refine-impl-prop*:

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

### 3.4 UTP Theory

**interpretation** *crf-theory*: *utp-theory-kleene* CRF  $II_c$

**rewrites**  $P \in \text{carrier } crf\text{-theory.thy-order} \longleftrightarrow P \text{ is CRF}$

**and**  $le \text{ rrel-theory.thy-order} = (\sqsubseteq)$

**and**  $eq \text{ rrel-theory.thy-order} = (=)$

**and** *crf-top*: *crf-theory.utp-top* = *false*

**and** *crf-bottom*: *crf-theory.utp-bottom* = *true<sub>r</sub>*

**proof** –

**interpret** *utp-theory-continuous* CRF

**by** (*unfold-locals, simp-all add: add: CRF-idem Continuous-CRF*)

**show** *top:utp-top* = *false*

```

  by (simp add: healthy-top, rel-auto)
show bot:utp-bottom = truer
  by (simp add: healthy-bottom, rel-auto)
show utp-theory-kleene CRF  $II_c$ 
  by (unfold-locales, simp-all add: closure rpred top)
qed (simp-all)

```

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

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

**lemma**  $crf\text{-}star\text{-}inductl$ :  $R$  is CRR  $\implies Q \sqsubseteq (P ;; Q) \sqcap R \implies Q \sqsubseteq P^{*c} ;; R$   
 by (simp add: crel-skip-left-unit crf-theory.utp-star-def upred-semiring.mult-assoc urel-ka.star-inductl)

### 3.5 Weakest Precondition

**lemma**  $nil\text{-}least$  [simp]:  
 $\llbracket \cdot \rrbracket \leq_u x = true$  **by** rel-auto

**lemma**  $minus\text{-}nil$  [simp]:  
 $xs - \llbracket \cdot \rrbracket = xs$  **by** rel-auto

**lemma**  $wp\text{-}rea\text{-}circus\text{-}lemma\text{-}1$ :  
 assumes  $P$  is CRR  $\$ref' \# P$   
 shows  $out\alpha \# P[\llbracket s_0 \rrbracket, \llbracket t_0 \rrbracket / \$st', \$tr']$   
**proof** –  
 have  $out\alpha \# (CRR (\exists \$ref' \cdot P))[\llbracket s_0 \rrbracket, \llbracket t_0 \rrbracket / \$st', \$tr']$   
 by (rel-auto)  
 thus ?thesis  
 by (simp add: Healthy-if assms(1) assms(2) ex-unrest)  
**qed**

**lemma**  $wp\text{-}rea\text{-}circus\text{-}lemma\text{-}2$ :  
 assumes  $P$  is CRR  
 shows  $in\alpha \# P[\llbracket s_0 \rrbracket, \llbracket t_0 \rrbracket / \$st, \$tr]$   
**proof** –  
 have  $in\alpha \# (CRR P)[\llbracket s_0 \rrbracket, \llbracket t_0 \rrbracket / \$st, \$tr]$   
 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**  $wp\text{-}rea\text{-}circus\text{-}form$ :  
 assumes  $P$  is CRR  $\$ref' \# P$   $Q$  is CRC  
 shows  $(P \text{ wp}_r Q) = (\forall (s_0, t_0) \cdot \llbracket t_0 \rrbracket \leq_u \$tr' \wedge P[\llbracket s_0 \rrbracket, \llbracket t_0 \rrbracket / \$st', \$tr']) \Rightarrow_r Q[\llbracket s_0 \rrbracket, \llbracket t_0 \rrbracket / \$st, \$tr])$   
**proof** –  
 have  $(P \text{ wp}_r Q) = (\neg_r (\exists t_0 \cdot P[\llbracket t_0 \rrbracket / \$tr']) ;; (\neg_r Q)[\llbracket t_0 \rrbracket / \$tr]) \wedge \llbracket t_0 \rrbracket \leq_u \$tr')$   
 by (simp-all add: wp-rea-def R2-tr-middle closure assms)

also have ... =  $(\neg_r (\exists (s_0, t_0) \cdot P[\llbracket s_0 \rrbracket, \llbracket t_0 \rrbracket / \$st', \$tr'] ; (\neg_r Q)[\llbracket s_0 \rrbracket, \llbracket t_0 \rrbracket / \$st, \$tr] \wedge \llbracket t_0 \rrbracket \leq_u \$tr'))$   
 by (rel-blast)  
 also have ... =  $(\neg_r (\exists (s_0, t_0) \cdot P[\llbracket s_0 \rrbracket, \llbracket t_0 \rrbracket / \$st', \$tr'] \wedge (\neg_r Q)[\llbracket s_0 \rrbracket, \llbracket t_0 \rrbracket / \$st, \$tr] \wedge \llbracket t_0 \rrbracket \leq_u \$tr'))$   
 by (simp add: seqr-to-conj add: wp-rea-circus-lemma-1 wp-rea-circus-lemma-2 assms closure conj-assoc)  
 also have ... =  $(\forall (s_0, t_0) \cdot \neg_r P[\llbracket s_0 \rrbracket, \llbracket t_0 \rrbracket / \$st', \$tr'] \vee \neg_r (\neg_r Q)[\llbracket s_0 \rrbracket, \llbracket t_0 \rrbracket / \$st, \$tr] \vee \neg_r \llbracket t_0 \rrbracket \leq_u \$tr')$   
 by (rel-auto)  
 also have ... =  $(\forall (s_0, t_0) \cdot \neg_r P[\llbracket s_0 \rrbracket, \llbracket t_0 \rrbracket / \$st', \$tr'] \vee \neg_r (\neg_r RR Q)[\llbracket s_0 \rrbracket, \llbracket t_0 \rrbracket / \$st, \$tr] \vee \neg_r \llbracket t_0 \rrbracket \leq_u \$tr')$   
 by (simp add: Healthy-if assms closure)  
 also have ... =  $(\forall (s_0, t_0) \cdot \neg_r P[\llbracket s_0 \rrbracket, \llbracket t_0 \rrbracket / \$st', \$tr'] \vee (RR Q)[\llbracket s_0 \rrbracket, \llbracket t_0 \rrbracket / \$st, \$tr] \vee \neg_r \llbracket t_0 \rrbracket \leq_u \$tr')$   
 by (rel-auto)  
 also have ... =  $(\forall (s_0, t_0) \cdot \llbracket t_0 \rrbracket \leq_u \$tr' \wedge P[\llbracket s_0 \rrbracket, \llbracket t_0 \rrbracket / \$st', \$tr'] \Rightarrow_r (RR Q)[\llbracket s_0 \rrbracket, \llbracket t_0 \rrbracket / \$st, \$tr])$   
 by (rel-auto)  
 also have ... =  $(\forall (s_0, t_0) \cdot \llbracket t_0 \rrbracket \leq_u \$tr' \wedge P[\llbracket s_0 \rrbracket, \llbracket t_0 \rrbracket / \$st', \$tr'] \Rightarrow_r Q[\llbracket s_0 \rrbracket, \llbracket t_0 \rrbracket / \$st, \$tr])$   
 by (simp add: Healthy-if assms closure)  
 finally show ?thesis .  
 qed

lemma wp-rea-circus-form-alt:

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

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

proof –

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

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

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

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

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

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

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

by (rel-auto)

also have ... =  $(\exists tt_0 \cdot (\forall (s_0, tr_0) \cdot \$tr \hat{=}^u \llbracket tr_0 \rrbracket \leq_u \$tr' \wedge (RR P)[\llbracket s_0 \rrbracket, \llbracket \cdot \rrbracket, \llbracket tr_0 \rrbracket / \$st', \$tr, \$tr'])$

$\Rightarrow_r (RR Q)[\llbracket s_0 \rrbracket, \llbracket \cdot \rrbracket, (\&tt - \llbracket tr_0 \rrbracket) / \$st, \$tr, \$tr'])$   
 $\wedge \$tr' =_u \$tr \hat{=}^u \llbracket tt_0 \rrbracket)$

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

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

by (rel-auto, blast+)

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

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

### 3.6 Trace Substitution

**definition** *trace-subst*  $(-\llbracket \cdot \rrbracket_t [999, 0] 999)$   
**where**  $[upred-defs]: P\llbracket v \rrbracket_t = (P[\llbracket \&tt - \lceil v \rceil_{S<} \rrbracket / \&tt] \wedge \$tr + \lceil v \rceil_{S<} \leq_u \$tr')$

**lemma** *unrest-trace-subst*  $[unrest]:$   
 $\llbracket mwb\text{-}lens\ x; x \bowtie (\$tr)_v; x \bowtie (\$tr')_v; x \bowtie (\$st)_v; x \# P \rrbracket \Longrightarrow x \# P\llbracket v \rrbracket_t$   
**by** (*simp add: trace-subst-def lens-indep-sym unrest*)

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

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

**proof** –

**have**  $(RR\ P)\llbracket v \rrbracket_t$  is *RR*  
**apply** (*rel-auto*)  
**apply** (*metis diff-add-cancel-left' trace-class.add-left-mono*)  
**apply** (*metis le-add minus-cancel-le trace-class.add-diff-cancel-left*)  
**using** *le-add order-trans* **apply** *blast*  
**done**  
**thus** *?thesis*  
**by** (*simp add: Healthy-if assms*)

**qed**

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

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

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

**assumes**  $P$  is *CRR*  
**shows**  $P[\llbracket \cdot \rrbracket_t] = P$

**proof** –

**have**  $(CRR\ P)[\llbracket \cdot \rrbracket_t] = CRR\ P$   
**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* [*ustbst*]:  $(\prod i \cdot P(i))\llbracket v \rrbracket_t = (\prod i \cdot (P(i))\llbracket v \rrbracket_t)$   
**by** (*rel-auto*)

### 3.7 Initial Interaction

**definition** *rea-init* :: '*s upred*  $\Rightarrow$  (*t::trace*, '*s*) *uepr*  $\Rightarrow$  (*s*, '*t*, ' $\alpha$ , ' $\beta$ ) *rel-rsp* (*I*'(-,-')) **where**  
*[upred-defs]*:  $\mathcal{I}(s, t) = (\lceil s \rceil_{S<} \Rightarrow_r \neg_r \$tr + \lceil t \rceil_{S<} \leq_u \$tr')$

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

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

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

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

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

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

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

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

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

**lemma** *rea-init-nil* [*rpred*]:  $\mathcal{I}(s, \llbracket \gg \rrbracket) = [\neg s]_{S<}$   
**by** (*rel-auto*)

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

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

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

by (rel-auto)

### 3.8 Enabled Events

**definition** *csp-enable* :: 's upred  $\Rightarrow$  ('e list, 's) uexpr  $\Rightarrow$  ('e set, 's) uexpr  $\Rightarrow$  ('s, 'e) action ( $\mathcal{E}'(-, -, -)$ )  
**where**

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

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

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

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

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

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

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

**lemma** *st-unrest-csp-enable* [unrest]:  $\llbracket \&\mathbf{v} \# s; \&\mathbf{v} \# t; \&\mathbf{v} \# E \rrbracket \Longrightarrow \$st \# \mathcal{E}(s, t, E)$   
 by (simp add: csp-enable-def unrest)

**lemma** *csp-enable-tr'-eq-tr* [rpred]:  
 $\mathcal{E}(s, \llbracket \gg, r \rrbracket \triangleleft \$tr' =_u \$tr \triangleright \text{false} = \mathcal{E}(s, \llbracket \gg, r \rrbracket, r)$   
 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*:  $\mathcal{E}(\text{true}, \llbracket \gg, \{v\}_u \rrbracket) = (\$tr' =_u \$tr \wedge [v]_{S<} \notin_u \$ref')$   
 by (rel-auto)

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

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

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

**lemma** *csp-enable-false* [*rpred*]:  $\mathcal{E}(\text{false}, t, E) = \text{false}$   
**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** *refine-csp-enable*:  $\mathcal{E}(b_1, t, E_1) \sqsubseteq \mathcal{E}(b_2, t, E_2) \longleftrightarrow ('b_2 \Rightarrow b_1 \wedge E_1 \subseteq_u E_2')$   
**by** (*rel-blast*)

**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*]:  
 $R4(\mathcal{E}(s, \llbracket \cdot \rrbracket, E)) = \text{false}$   
**by** (*rel-auto*)

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

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

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

**lemma** *rel-aext-csp-enable* [*alpha*]:  
 $\text{vwb-lens } a \Longrightarrow \mathcal{E}(s, t, E) \oplus_r \text{map-st}_L[a] = \mathcal{E}(s \oplus_p a, t \oplus_p a, E \oplus_p a)$   
**by** (*rel-auto*)

### 3.9 Completed Trace Interaction

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

**lemma** *csp-do-eq-intro*:  
**assumes**  $s_1 = s_2 \ \sigma_1 = \sigma_2 \ t_1 = t_2$   
**shows**  $\Phi(s_1, \sigma_1, t_1) = \Phi(s_2, \sigma_2, t_2)$   
**by** (*simp add: assms*)

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

**lemma** *unrest-csp-do* [*unrest*]:

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

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

**by** (*rel-auto*)

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

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

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

**lemma** *st-pred-conj-csp-do* [*rpred*]:

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

**by** (*rel-auto*)

**lemma** *trea-subst-csp-do* [*usubst*]:

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

**apply** (*rel-auto*)

**apply** (*metis append.assoc less-eq-list-def prefix-concat-minus*)

**apply** (*simp add: list-concat-minus-list-concat*)

**done**

**lemma** *st-subst-csp-do* [*usubst*]:

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

**by** (*rel-auto*)

**lemma** *csp-do-nothing*:  $\Phi(\text{true}, id_s, \llbracket \phantom{x} \rrbracket) = II_c$

**by** (*rel-auto*)

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

**by** (*rel-auto*)

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

**by** (*rel-auto*)

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

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

**by** (*rel-auto*)

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

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

**by** (*rel-auto*)

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

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

**by** (*rel-auto*)

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

$\Phi(s_1, \sigma, t_1) \triangleleft b \triangleright_R \Phi(s_2, \varrho, t_2) = \Phi(s_1 \triangleleft b \triangleright s_2, \sigma \triangleleft b \triangleright \varrho, t_1 \triangleleft b \triangleright t_2)$

**by** (*rel-auto*)

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

$[b]^\top_r ;; \Phi(s, \sigma, t) = \Phi(b \wedge s, \sigma, t)$

**by** (*rel-auto*)

**lemma** *csp-do-comp*:  
**assumes**  $P$  is CRR  
**shows**  $\Phi(s, \sigma, t) ;; P = ([s]_{s<} \wedge (\sigma \uparrow_S P)) \llbracket t \rrbracket_t$   
**proof** –  
**have**  $\Phi(s, \sigma, t) ;; (CRR\ P) = ([s]_{s<} \wedge ((\sigma \uparrow_S CRR\ P)) \llbracket t \rrbracket_t)$   
**by** (*rel-auto*; *blast*)  
**thus** *?thesis*  
**by** (*simp add: Healthy-if assms*)  
**qed**

**lemma** *wp-rea-csp-do-lemma*:  
**fixes**  $P :: ('\sigma, '\varphi)$  action  
**assumes**  $\$ok \# P \$wait \# P \$ref \# P$   
**shows**  $(\llbracket \langle \sigma \rangle_a \rrbracket_S \wedge \$tr' =_u \$tr \hat{\ }_u \llbracket t \rrbracket_{s<}) ;; P = (\llbracket \sigma \rrbracket_{S\sigma} \uparrow P) \llbracket \$tr \hat{\ }_u \llbracket t \rrbracket_{s<} / \$tr \rrbracket$   
**using** *assms* **by** (*rel-auto*, *meson*)

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

**lemma** *wp-rea-csp-do [wp]*:  
 $\Phi(s_1, \sigma, t_1) \text{ wp}_r \mathcal{I}(s_2, t_2) = \mathcal{I}(s_1 \wedge \sigma \uparrow s_2, t_1 \hat{\ }_u \sigma \uparrow t_2)$   
**by** (*rel-auto*)

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

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

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

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

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

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

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

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

$P$  is CRR  $\implies \Phi(b, id_s, \llbracket \cdot \rrbracket) \;; P = ([b]_{S<} \wedge P)$   
**by** (simp add: csp-do-comp usubst)

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

$P$  is CRR  $\implies \Phi(b, id_s, \llbracket \cdot \rrbracket) \wp_r P = ([b]_{S<} \Rightarrow_r P)$   
**by** (metis (no-types, lifting) CRR-implies-RR RR-implies-R1 csp-do-id rea-impl-conj rea-impl-false  
 rea-not-CRR-closed rea-not-not wp-rea-def)

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

**by** (rel-auto)

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

**fixes**  $Q :: ('s, 's) \text{ action}$   
**assumes**  $P$  is CRR  
**shows**  $\Phi(s, \sigma, t) \wp_r P = (\mathcal{I}(s, t) \wedge (\sigma \dagger_S P) \llbracket t \rrbracket_t)$   
**apply** (simp add: wp-rea-def)  
**apply** (subst csp-do-comp)  
**apply** (simp-all add: closure assms usubst)  
**oops**

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

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

**lemma** *rea-frame-ext-csp-do* [frame]:

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

**lemma** *R5-csp-do-nil* [rpred]:  $R5(\Phi(s, \sigma, \llbracket \cdot \rrbracket)) = \Phi(s, \sigma, \llbracket \cdot \rrbracket)$

**by** (rel-auto)

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

**by** (rel-auto)

Iterated do relations

**fun** *titr* ::  $nat \Rightarrow 's \text{ usubst} \Rightarrow ('a \text{ list}, 's) \text{ uexpr} \Rightarrow ('a \text{ list}, 's) \text{ uexpr}$  **where**

*titr* 0  $\sigma$   $t = 0$  |

*titr* (Suc  $n$ )  $\sigma$   $t = (\text{titr } n \ \sigma \ t) + (\sigma \hat{\ }_s n) \dagger t$

**lemma** *titr-as-list-sum*:  $\text{titr } n \ \sigma \ t = \text{list-sum } (\text{map } (\lambda i. (\sigma \hat{\ }_s i) \dagger t) [0..<n])$

**apply** (induct  $n$ )

**apply** (auto simp add: usubst fold-plus-sum-list-rev foldr-conv-fold)

**done**

**lemma** *titr-as-foldr*:  $\text{titr } n \ \sigma \ t = \text{foldr } (\lambda i \ e. (\sigma \hat{\ }_s i) \dagger t + e) [0..<n] \ 0$

**by** (simp add: titr-as-list-sum foldr-map comp-def)

**lemma** *list-sum-uexpr-rep-eq*:  $\llbracket \text{list-sum } xs \rrbracket_e s = \text{list-sum } (\text{map } (\lambda e. \llbracket e \rrbracket_e s) xs)$

```

apply (induct xs)
apply (simp-all)
apply (pred-simp+)
done

```

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

#### update-uexpr-rep-eq-thms

**lemma** *titr-lemma*:

$t + (\sigma \dagger \text{titr } n \ \sigma \ t) + (\sigma \hat{\ }_s n \circ_s \sigma) \dagger t = (\text{titr } n \ \sigma \ t + (\sigma \hat{\ }_s n) \dagger t) + (\sigma \circ_s \sigma \hat{\ }_s n) \dagger t$   
**by** (induct n, simp-all add: usubst add.assoc, metis subst-monoid.power-Suc subst-monoid.power-Suc2)

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

$\Phi(s, \sigma, t)^\wedge (\text{Suc } n) = \Phi(\bigwedge i \in \{0..n\} \cdot (\sigma \hat{\ }_s i) \dagger s, \sigma \hat{\ }_s \text{Suc } n, \text{titr } (\text{Suc } n) \ \sigma \ t)$   
**apply** (induct n)  
**apply** (rel-auto)  
**apply** (simp add: power.power.power-Suc rpred usubst)  
**apply** (thin-tac -)  
**apply** (rule csp-do-eq-intro)  
**apply** (rel-auto)  
**apply** (case-tac x=0)  
**apply** (simp-all add: titr-lemma)  
**apply** (metis Suc-le-mono funpow-simps-right(2) gr0-implies-Suc o-def)  
**apply** force  
**apply** (metis Suc-leI funpow-simps-right(2) less-Suc-eq-le o-apply)  
**apply** (metis subst-monoid.power-Suc subst-monoid.power-Suc2)  
**apply** (metis add.assoc plus-list-def plus-uexpr-def titr-lemma)  
**done**

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

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

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

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

**proof** –

**have** ?lhs =  $II_c \sqcap (\bigcap n \cdot \Phi(\bigwedge i \in \{0..n\} \cdot (\sigma \hat{\ }_s i) \dagger s, \sigma \hat{\ }_s \text{Suc } n, \text{titr } (\text{Suc } n) \ \sigma \ t))$   
(is - =  $II_c \sqcap (\bigcap n \cdot ?F(n))$ )  
**by** (simp add: crf-theory.Star-alt-def closure uplus-power-def rpred)  
**also have** ... =  $II_c \sqcap (\bigcap n \in \{1..\} \cdot ?F(n - 1))$   
**by** (simp add: UINF-atLeast-Suc)  
**also have** ... =  $II_c \sqcap (\bigcap n \in \{1..\} \cdot \Phi(\bigsqcup i \in \{0..<n\} \cdot (\sigma \hat{\ }_s i) \dagger s, \sigma \hat{\ }_s n, \text{titr } n \ \sigma \ t))$

**proof** –

**have**  $(\bigcap n \in \{1..\} \cdot ?F(n - 1)) = (\bigcap n \in \{1..\} \cdot ?G(n))$   
**by** (rule UINF-cong, simp, metis (no-types, lifting) Suc-diff-le atLeastLessThanSuc-atLeastAtMost  
cancel-comm-monoid-add-class.diff-zero diff-Suc-Suc)

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

**qed**

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

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

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

**by** (simp)



also have ... = ( $\prod$   $n \cdot ?G(n)$ )  
**proof** –  
 have *insert* ( $0::nat$ )  $\{1..\}$  =  $\{0..\}$  **by** *auto*  
 thus *?thesis*  
 by *simp*  
**qed**  
 finally show *?thesis* .  
**qed**

### 3.10 Assumptions

**abbreviation** *crf-assume* ::  $'s \text{ upred} \Rightarrow ('s, 'e) \text{ action } ([\cdot]_c)$  **where**  
 $[b]_c \equiv \Phi(b, id_s, \llbracket \cdot \rrbracket)$

**lemma** *crf-assume-true* [*rpred*]:  $P \text{ is } CRR \Longrightarrow [true]_c ;; P = P$   
**by** (*simp add: crel-skip-left-unit csp-do-nothing*)

### 3.11 Downward closure of refusals

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

**definition** *CDC* ::  $('s, 'e) \text{ action} \Rightarrow ('s, 'e) \text{ action}$  **where**  
 $[upred-defs]: CDC(P) = (\exists \text{ ref}_0 \cdot P \llbracket \llbracket \text{ref}_0 \rrbracket / \$\text{ref}' \rrbracket \wedge \$\text{ref}' \subseteq_u \llbracket \text{ref}_0 \rrbracket)$

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

**lemma** *CDC-Continuous* [*closure*]: *Continuous CDC*  
**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-commute*:  $CDC(CRR P) = CRR(CDC P)$   
**by** (*rel-blast*)

**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*]:  $\llbracket vwb\text{-lens } x; (\$ref')_v \bowtie x; x \# P \rrbracket \Longrightarrow x \# CDC(P)$   
**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]:  $P$  is CDC  $\implies R5(P)$  is CDC  
 by (simp add: CDC-R5-commute Healthy-def)

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

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

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

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

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

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

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

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

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

**lemma** *st-subst-CDC-closed* [closure]:  
 assumes  $P$  is CDC

shows  $(\sigma \uparrow_S P)$  is CDC  
**proof** –  
 have  $(\sigma \uparrow_S \text{CDC } P)$  is CDC  
 by (rel-auto)  
 thus ?thesis  
 by (simp add: assms Healthy-if)  
**qed**

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

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

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

### 3.12 Renaming

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

**definition** *csp-rename* ::  $('s, 'e) \text{ action} \Rightarrow ('e \Rightarrow 'f) \Rightarrow ('s, 'f) \text{ action}$   $((-) \Downarrow_c [999, 0] 999)$  **where**  
 $[upred\text{-}defs]: P \Downarrow_c = R2((\$tr' =_u \ll \gg \wedge \$st' =_u \$st) ;; P ;; (\$tr' =_u \text{map}_u \ll f \gg \$tr \wedge \$st' =_u \$st$   
 $\wedge uop (\text{pre-image } f) \$ref' \subseteq_u \$ref))$

**lemma** *csp-rename-CRR-closed* [closure]:  
 assumes  $P$  is CRR  
 shows  $P \Downarrow_c$  is CRR  
**proof** –  
 have  $(\text{CRR } P) \Downarrow_c$  is CRR  
 by (rel-auto)  
 thus ?thesis by (simp add: assms Healthy-if)  
**qed**

**lemma** *csp-rename-disj* [rpred]:  $(P \vee Q) \Downarrow_c = (P \Downarrow_c \vee Q \Downarrow_c)$   
 by (rel-blast)

**lemma** *csp-rename-UINF-ind* [rpred]:  $(\bigcap i \cdot P i) \Downarrow_c = (\bigcap i \cdot (P i) \Downarrow_c)$   
 by (rel-blast)

**lemma** *csp-rename-UINF-mem* [rpred]:  $(\bigcap i \in A \cdot P i) \Downarrow_c = (\bigcap i \in A \cdot (P i) \Downarrow_c)$   
 by (rel-blast)

Renaming distributes through conjunction only when both sides are downward closed

**lemma** *csp-rename-conj* [rpred]:  
 assumes  $\text{inj } f$   $P$  is CRR  $Q$  is CRR  $P$  is CDC  $Q$  is CDC  
 shows  $(P \wedge Q) \Downarrow_c = (P \Downarrow_c \wedge Q \Downarrow_c)$   
**proof** –

```

from assms(1) have ((CDC (CRR P))  $\wedge$  (CDC (CRR Q)))( $\downarrow f$ )c = ((CDC (CRR P))( $\downarrow f$ )c  $\wedge$  (CDC
(CRR Q))( $\downarrow f$ )c)
  apply (rel-auto)
  apply blast
  apply blast
  apply (meson order-refl order-trans)
  done
thus ?thesis
  by (simp add: assms Healthy-if)
qed

```

```

lemma csp-rename-seq [rpred]:
  assumes P is CRR Q is CRR
  shows (P ;; Q)( $\downarrow f$ )c = P( $\downarrow f$ )c ;; Q( $\downarrow f$ )c
  oops

```

```

lemma csp-rename-R4 [rpred]:
  (R4(P))( $\downarrow f$ )c = R4(P( $\downarrow f$ )c)
  apply (rel-auto, blast)
  using less-le apply fastforce
  apply (metis (mono-tags, lifting) Prefix-Order.Nil-prefix append-Nil2 diff-add-cancel-left' less-le list.simps(8)
plus-list-def)
  done

```

```

lemma csp-rename-R5 [rpred]:
  (R5(P))( $\downarrow f$ )c = R5(P( $\downarrow f$ )c)
  apply (rel-auto, blast)
  using less-le apply fastforce
  done

```

```

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

```

```

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

```

```

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

```

```

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

```

```

lemma csp-rename-CDC-closed [closure]:
  P is CDC  $\implies P(\downarrow f)_c$  is CDC
  by (rel-blast)

```

```

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

```

**end**

## 4 Stateful-Failure Healthiness Conditions

```

theory utp-sfrd-healths
  imports utp-sfrd-rel

```

begin

## 5 Definitions

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

**abbreviation**  $CSP1 :: ((\sigma, \varphi) \text{ sfrd} \times (\sigma, \varphi) \text{ sfrd}) \text{ health}$   
**where**  $CSP1(P) \equiv RD1(P)$

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

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

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

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

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

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

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

**definition**  $CSP4 :: ((\sigma, \varphi) \text{ sfrd} \times (\sigma, \varphi) \text{ sfrd}) \text{ health where}$   
 $[upred-defs]: CSP4(P) = (P ;; Skip)$

**definition**  $NCSP :: ((\sigma, \varphi) \text{ sfrd} \times (\sigma, \varphi) \text{ sfrd}) \text{ health where}$   
 $[upred-defs]: NCSP = CSP3 \circ CSP4 \circ CSP$

Productive and normal processes

**abbreviation**  $PCSP \equiv Productive \circ NCSP$

Instantaneous and normal processes

**abbreviation**  $ICSP \equiv ISRD1 \circ NCSP$

### 5.1 Healthiness condition properties

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

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

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

**theorem**  $Skip\text{-UTP-form}$ :  $Skip = \mathbf{R}_s(\exists \$ref \cdot CSP1(H))$

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]:  
 $Skip = \mathbf{R}_s(true_r \vdash (false \diamond \Phi(true, id_s, \llbracket \gg \rrbracket)))$   
 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]:  $Stop = \mathbf{R}_s(true_r \vdash \mathcal{E}(true, \llbracket \gg \rrbracket, \{\}_u) \diamond false)$   
 by (rel-auto)

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

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

**lemma** *postR-Skip* [rdes]:  $post_R(Skip) = \Phi(true, id_s, \llbracket \gg \rrbracket)$   
 by (rel-auto)

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

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

qed

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

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

**shows**  $Skip \;; P = P$

**using** *assms*

**by** (*simp add: Skip-left-lemma*)

(*metis SRD-reactive-design-alt all-unrest cmt-unrest-ref cmt-wait-false ex-unrest pre-unrest-ref pre-wait-false*)

**lemma** *CSP3-intro:*

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

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

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

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

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

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

**lemma** *CSP3-SRD-intro:*

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

**shows**  $P$  is CSP3

**proof** –

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

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

**have**  $\mathbf{R}_s(pre_R(P) \vdash peri_R(P) \diamond post_R(P))$  is CSP3

**by** (*rule CSP3-intro, simp add: assms P, simp add: ref-unrest-RHS-design assms*)

**thus** *?thesis*

**by** (*simp add: P*)

qed

**lemma** *Skip-unrest-ref [unrest]:*  $\$ref \# Skip \llbracket false/\$wait \rrbracket$

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

**lemma** *Skip-unrest-ref' [unrest]:*  $\$ref' \# Skip \llbracket false/\$wait \rrbracket$

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

**lemma** *CSP3-iff:*

**assumes**  $P$  is CSP

**shows**  $P$  is CSP3  $\longleftrightarrow (\$ref \# P \llbracket false/\$wait \rrbracket)$

**proof**

**assume**  $1: P$  is CSP3

**have**  $\$ref \# (Skip \;; P) \llbracket false/\$wait \rrbracket$

**by** (*simp add: usubst unrest*)

**with**  $1$  **show**  $\$ref \# P \llbracket false/\$wait \rrbracket$

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

**next**

**assume**  $1: \$ref \# P \llbracket false/\$wait \rrbracket$

**show**  $P$  is CSP3

**by** (*simp add: 1 CSP3-intro assms*)

qed

**lemma** *CSP3-unrest-ref [unrest]:*

**assumes**  $P$  is CSP  $P$  is CSP3

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

**proof** –

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

**lemma**  $CSP3\text{-}rdes$ :

**assumes**  $P \text{ is } RR \ Q \text{ is } RR \ R \text{ is } RR$   
**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**  $(simp \text{ add: } CSP3\text{-}def \text{ Skip-left-lemma closure assms rdes, rel-auto})$

**lemma**  $CSP3\text{-}form$ :

**assumes**  $P \text{ is } CSP$   
**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**  $(simp \text{ add: } CSP3\text{-}def \text{ Skip-left-lemma assms, rel-auto})$

**lemma**  $CSP3\text{-}Skip \text{ [closure]}$ :

$Skip \text{ is } CSP3$   
**by**  $(rule \text{ } CSP3\text{-}intro, simp \text{ add: } Skip\text{-is-CSP}, simp \text{ add: } Skip\text{-def unrest})$

**lemma**  $CSP3\text{-}Stop \text{ [closure]}$ :

$Stop \text{ is } CSP3$   
**by**  $(rule \text{ } CSP3\text{-}intro, simp \text{ add: } Stop\text{-is-CSP}, simp \text{ add: } Stop\text{-def unrest})$

**lemma**  $CSP3\text{-}Idempotent \text{ [closure]}$ :  $Idempotent \text{ } CSP3$

**by**  $(metis \text{ (no-types, lifting) } CSP3\text{-}Skip \text{ } CSP3\text{-}def \text{ Healthy-if Idempotent-def seqr-assoc})$

**lemma**  $CSP3\text{-}Continuous$ :  $Continuous \text{ } CSP3$

**by**  $(simp \text{ add: } Continuous\text{-def } CSP3\text{-}def \text{ seq-Sup-distl})$

**lemma**  $Skip\text{-right-lemma}$ :

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

**proof** –

**have**  $P ;; Skip = \mathbf{R}_s((\neg_r pre_R P) wp_r false \vdash (\exists \$st' \cdot peri_R P) \diamond post_R P ;; (\$tr' =_u \$tr \wedge \$st' =_u \$st))$

**by**  $(simp \text{ add: } SRD\text{-composition-wp closure assms wp rdes rpred, rel-auto})$

**also have**  $\dots = \mathbf{R}_s((\neg_r pre_R P) wp_r false \vdash ((cmt_R P ;; (\exists \$st \cdot [II]_D)) \triangleleft \$wait' \triangleright (cmt_R P ;; (\$tr' =_u \$tr \wedge \neg \$wait \wedge \$st' =_u \$st))))$

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

**also have**  $\dots = \mathbf{R}_s((\neg_r pre_R P) wp_r false \vdash ((\exists \$st' \cdot cmt_R P) \triangleleft \$wait' \triangleright (cmt_R P ;; (\$tr' =_u \$tr \wedge \neg \$wait \wedge \$st' =_u \$st))))$

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

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

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

**finally show**  $?thesis$  .

**qed**

**lemma**  $Skip\text{-right-tri-lemma}$ :

**assumes**  $P \text{ is } CSP$



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  $P \text{ is CSP } (\neg_r \text{pre}_R(P)) \;; R1(\text{true}) = (\neg_r \text{pre}_R(P))$   
 $\$st' \# (\text{cmt}_R P) \llbracket \text{true}/\$wait' \rrbracket \$ref' \# (\text{cmt}_R P) \llbracket \text{false}/\$wait' \rrbracket$   
 shows  $P \text{ is CSP}_4$   
**proof** –  
 have  $\text{CSP}_4(P) = \mathbf{R}_s ((\neg_r \text{pre}_R P) \text{wp}_r \text{false} \vdash ((\exists \$st' \cdot \text{cmt}_R P) \triangleleft \$wait' \triangleright (\exists \$ref' \cdot \text{cmt}_R P)))$   
 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  $P \text{ is CSP } \text{pre}_R(P) \text{ is RC}$   
 $\$st' \# (\text{cmt}_R P) \llbracket \text{true}/\$wait' \rrbracket \$ref' \# (\text{cmt}_R P) \llbracket \text{false}/\$wait' \rrbracket$   
 shows  $P \text{ is CSP}_4$   
**proof** –  
 have  $(\neg_r \text{pre}_R(P)) \;; R1(\text{true}) = (\neg_r \text{pre}_R(P))$   
 by (*metis (no-types, lifting) R1-seqr-closure assms(2) rea-not-R1 rea-not-false rea-not-not wp-rea-RC-false wp-rea-def*)  
 thus ?thesis  
 by (*simp add: CSP4-intro assms*)  
**qed**

**lemma CSP4-rdes:**

assumes  $P \text{ is RR } Q \text{ is RR } R \text{ is RR}$   
 shows  $\text{CSP}_4(\mathbf{R}_s(P \vdash Q \diamond R)) = \mathbf{R}_s ((\neg_r P) \text{wp}_r \text{false} \vdash ((\exists \$st' \cdot Q) \diamond (\exists \$ref' \cdot R)))$   
 by (*simp add: CSP4-def Skip-right-lemma closure assms rdes, rel-auto, blast+*)

**lemma CSP4-form:**

assumes  $P \text{ is CSP}$   
 shows  $\text{CSP}_4(P) = \mathbf{R}_s ((\neg_r \text{pre}_R P) \text{wp}_r \text{false} \vdash ((\exists \$st' \cdot \text{peri}_R P) \diamond (\exists \$ref' \cdot \text{post}_R P)))$   
 by (*simp add: CSP4-def Skip-right-tri-lemma assms*)

**lemma Skip-srdes-right-unit:**

$(\text{Skip} :: ('\sigma, '\varphi) \text{ action}) \;; \text{II}_R = \text{Skip}$

by (*rdes-simp*)

**lemma** *Skip-srdes-left-unit*:  
 $\Pi_R \;; (Skip \:: (' \sigma, ' \varphi) \text{ action}) = Skip$   
 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  $P \text{ is } CSP (\neg_r \text{ pre}_R(P)) \;; R1(true) = (\neg_r \text{ pre}_R(P)) \$st' \# \text{peri}_R(P) \$ref' \# \text{post}_R(P)$   
 shows  $P \text{ is } CSP4$   
 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  $P \text{ is } NSRD \$ref' \# \text{post}_R(P)$   
 shows  $P \text{ is } CSP4$   
 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 seqr-assoc*)

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

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

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

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

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

$A \subseteq \llbracket NCSP \rrbracket_H \implies A \subseteq \llbracket NSRD \rrbracket_H$   
**using** *NCSP-implies-NSRD* **by** *blast*

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

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

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

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

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

**lemma** *Skip-left-unit*:  $P \text{ is } NCSP \implies \text{Skip} ;; P = P$   
**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**  $P \text{ is } NSRD \ \$ref \ \# \ pre_R(P) \ \$ref \ \# \ peri_R(P) \ \$ref \ \# \ post_R(P) \ \$ref' \ \# \ post_R(P)$   
**shows** *P is NCSP*  
**by** (*simp add: CSP3-SRD-intro CSP4-NSRD-intro NCSP-intro NSRD-is-SRD assms*)

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

**lemma** *NSRD-CSP4-intro*:  
**assumes** *P is CSP P is CSP4*  
**shows** *P is NSRD*  
**by** (*simp add: CSP4-implies-RD3 SRD-RD3-implies-NSRD assms(1) assms(2)*)

**lemma** *NCSP-form*:  
 $NCSP \ P = \mathbf{R}_s ((\forall \ \$ref \cdot (\neg_r \ pre_R(P)) \ wp_r \ false) \vdash ((\exists \ \$ref \cdot \exists \ \$st' \cdot peri_R(P)) \diamond (\exists \ \$ref \cdot \exists \ \$ref' \cdot post_R(P))))$   
**proof** –  
**have**  $NCSP \ P = CSP3 \ (CSP4 \ (NSRD \ P))$   
**by** (*metis (no-types, hide-lams) CSP4-def NCSP-def NSRD-alt-def RA1 RD3-def Skip-srdes-left-unit o-apply*)  
**also**  
**have**  $\dots = \mathbf{R}_s ((\forall \ \$ref \cdot (\neg_r \ pre_R \ (NSRD \ P)) \ wp_r \ false) \vdash ((\exists \ \$ref \cdot \exists \ \$st' \cdot peri_R \ (NSRD \ P)) \diamond ((\exists \ \$ref \cdot \exists \ \$ref' \cdot post_R \ (NSRD \ P))))$   
**by** (*simp add: CSP3-form CSP4-form closure unrest rdes, rel-auto*)  
**also have**  $\dots = \mathbf{R}_s ((\forall \ \$ref \cdot (\neg_r \ pre_R(P)) \ wp_r \ false) \vdash ((\exists \ \$ref \cdot \exists \ \$st' \cdot peri_R(P)) \diamond (\exists \ \$ref \cdot \exists \ \$ref' \cdot post_R(P))))$

by (simp add: NSRD-form rdes closure, rel-blast)  
 finally show ?thesis .  
 qed

lemma *CSP4-st'-unrest-peri* [unrest]:  
 assumes *P is CSP P is CSP4*  
 shows  $\$st' \# \text{peri}_R(P)$   
 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  $\$ref' \# \text{pre}_R(P)$   
 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  $\dots = (\neg_r \text{pre}_R P) \text{wp}_r \text{false}$   
 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-CSP<sub>4</sub>-intro *assms(1)* *assms(2)* by blast  
 thus  $\$st' \# pre_R(P)$   
 using NSRD- $st'$ -unrest-pre by blast  
 qed

**lemma** CSP<sub>4</sub>-ref'-unrest-post [unrest]:  
 assumes  $P$  is CSP  $P$  is CSP<sub>4</sub>  
 shows  $\$ref' \# post_R(P)$   
**proof** –  
 have  $post_R(P) = post_R(\mathbf{R}_s((\neg_r pre_R P) wp_r false \vdash ((\exists \$st' \cdot peri_R(P)) \diamond (\exists \$ref' \cdot post_R(P))))$   
 using CSP<sub>4</sub>-healthy-form *assms(1)* *assms(2)* by fastforce  
 also have  $\dots = R1 (R2c ((\neg_r pre_R P) wp_r false \Rightarrow_r (\exists \$ref' \cdot post_R P)))$   
 by (simp add: rea-post-RHS-design usubst unrest wp-rea-def)  
 also have  $\$ref' \# \dots$   
 by (simp add: R1-def R2c-def wp-rea-def unrest)  
 finally show ?thesis .  
 qed

**lemma** CSP<sub>3</sub>-Chaos [closure]: Chaos is CSP<sub>3</sub>  
 by (simp add: Chaos-def, rule CSP<sub>3</sub>-intro, simp-all add: RHS-design-is-SRD unrest)

**lemma** CSP<sub>4</sub>-Chaos [closure]: Chaos is CSP<sub>4</sub>  
 by (rule CSP<sub>4</sub>-tri-intro, simp-all add: closure rdes unrest)

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

**lemma** CSP<sub>3</sub>-Miracle [closure]: Miracle is CSP<sub>3</sub>  
 by (simp add: Miracle-def, rule CSP<sub>3</sub>-intro, simp-all add: RHS-design-is-SRD unrest)

**lemma** CSP<sub>4</sub>-Miracle [closure]: Miracle is CSP<sub>4</sub>  
 by (rule CSP<sub>4</sub>-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) CSP<sub>3</sub>-def CSP<sub>4</sub>-def Healthy-def' NCSP-implies-CSP NCSP-implies-CSP<sub>3</sub>  
 NCSP-implies-CSP<sub>4</sub> NCSP-intro SRD-seqr-closure *assms(1)* *assms(2)* seqr-assoc)

**lemma** CSP<sub>4</sub>-Skip [closure]: Skip is CSP<sub>4</sub>  
 apply (rule CSP<sub>4</sub>-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 CSP<sub>3</sub>-Skip CSP<sub>4</sub>-Skip Healthy-def NCSP-def Skip-is-CSP comp-apply)

**lemma** CSP<sub>4</sub>-Stop [closure]: Stop is CSP<sub>4</sub>  
 apply (rule CSP<sub>4</sub>-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]:  $\text{pre}_R(\text{Stop}) = \text{true}_r$   
 by (simp add: *Stop-def Stop-is-CSP rea-pre-RHS-design unrest usubst R2c-true*)

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

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

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

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

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

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

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

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

**lemma** *NCSP-rdes-intro* [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<sub>R</sub>(P) is CRC*  
**by** (*rule CRC-intro, simp-all add: closure assms unrest*)

**lemma** *NCSP-postR-CRF [closure]: P is NCSP  $\implies$  post<sub>R</sub> P is CRF*  
**by** (*rule CRF-intro, simp-all add: unrest closure*)

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

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

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

**lemma** *NCSP-SUP-closure [closure]:  $\llbracket \bigwedge i. P(i) \text{ is NCSP}; A \neq \{\} \rrbracket \implies (\bigcap_{i \in A} P(i))$  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 = Productive(NCSP(NCSP P))*  
**by** (*metis (no-types, hide-lams) Healthy-def' Idempotent-def NCSP-Idempotent assms comp-apply*)

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

**by** (*simp add: NCSP-form Productive-RHS-design-form unrest closure*)

**also have**  $\dots$  *is NCSP*  
**apply** (*rule NCSP-rdes-intro*)  
**apply** (*rule CRC-intro*)  
**apply** (*simp-all add: unrest ex-unrest all-unrest closure*)  
**done**

**finally show** *?thesis* .

qed

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

**assumes** *X is PCSP P* ( $\mathbf{R}_s ((pre_R X) \vdash peri_R X \diamond (R4(post_R X))))$ )

**shows** *P X*

**by** (*metis R4-def Healthy-if NCSP-implies-CSP PCSP-implies-NCSP Productive-form assms comp-apply*)

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

**assumes** *P is ICSP*

**shows** *P is NCSP*

**proof** –

**have**  $P = ISRD1(NCSP(NCSP P))$

**by** (*metis (no-types, hide-lams) Healthy-def' Idempotent-def NCSP-Idempotent assms comp-apply*)

**also have**  $\dots = ISRD1(\mathbf{R}_s ((\forall \$ref \cdot (\neg_r pre_R (NCSP P)) wp_r false) \vdash$   
 $(\exists \$ref \cdot \exists \$st' \cdot peri_R (NCSP P)) \diamond$   
 $(\exists \$ref \cdot \exists \$ref' \cdot post_R (NCSP P))))$

**by** (*simp add: NCSP-form*)

**also have**  $\dots = \mathbf{R}_s ((\forall \$ref \cdot (\neg_r pre_R (NCSP P)) wp_r false) \vdash$   
 $false \diamond$

$((\exists \$ref \cdot \exists \$ref' \cdot post_R (NCSP P)) \wedge \$tr' =_u \$tr))$

**by** (*simp-all add: ISRD1-RHS-design-form closure rdes unrest*)

**also have**  $\dots$  *is NCSP*

**apply** (*rule NCSP-rdes-intro*)

**apply** (*rule CRC-intro*)

**apply** (*simp-all add: unrest ex-unrest all-unrest closure*)

**done**

**finally show** *?thesis* .

qed

**lemma** *ICSP-implies-ISRD* [closure]:

**assumes** *P is ICSP*

**shows** *P is ISRD*

**by** (*metis (no-types, hide-lams) Healthy-def ICSP-implies-NCSP ISRD-def NCSP-implies-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**  $P \text{ is ICSP } Q \text{ is ICSP}$   
**shows**  $P ;; Q \text{ is ICSP}$   
**by** (*meson ICSP-implies-ISRD ICSP-implies-NCSP ICSP-intro ISRD-implies-ISRD1 NCSP-seqr-closure assms seq-ISRD-closed*)

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

## 5.2 CSP theories

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

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

**interpretation** *csp-theory*: *utp-theory-kleene NCSP Skip*  
**rewrites**  $P \in \text{carrier csp-theory.thy-order} \longleftrightarrow P \text{ is NCSP}$   
**and**  $\text{carrier csp-theory.thy-order} \rightarrow \text{carrier csp-theory.thy-order} \equiv \llbracket NCSP \rrbracket_H \rightarrow \llbracket NCSP \rrbracket_H$   
**and**  $\text{le csp-theory.thy-order} = (\sqsubseteq)$   
**and**  $\text{eq csp-theory.thy-order} = (=)$   
**and** *csp-top*:  $\text{csp-theory.utp-top} = \text{Miracle}$   
**and** *csp-bottom*:  $\text{csp-theory.utp-bottom} = \text{Chaos}$

**proof** –

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

**abbreviation** *TestC* (*test<sub>C</sub>*) **where**  
 $\text{test}_C P \equiv \text{csp-theory.utp-test } P$

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

**lemma** *StarC-unfold*:  $P \text{ is NCSP} \implies P^{*C} = \text{Skip} \sqcap (P ;; P^{*C})$   
**by** (*simp add: StarC-def csp-theory.Star-unfoldl-eq*)

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

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

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

### 5.3 Algebraic laws

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

## 6 Stateful-Failure Reactive Contracts

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

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

**syntax**  
*-ref-var* :: *logic*  
*-mk-CRD* :: *logic*  $\Rightarrow$  *logic*  $\Rightarrow$  *logic*  $\Rightarrow$  *logic* (*[-/*  $\vdash$  *-/*  $|$  *-]*<sub>*C*</sub>)

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

**translations**  
*-mk-CRD* *P* *Q* *R*  $\Rightarrow$  *CONST mk-CRD* *P* ( $\lambda$  *-trace-var -ref-var.* *Q*) ( $\lambda$  *-trace-var.* *R*)  
*-mk-CRD* *P* *Q* *R*  $\Leftarrow$  *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]*:  $\text{peri}_R([P \vdash Q \text{ trace refs} \mid R(\text{trace})]_C) = ([P]_{S<} \Rightarrow_r ([Q \text{ trace refs}]_{S<} \llbracket (\text{trace}, \text{refs}) \rightarrow (\&tt, \$n$

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**

$Q \text{ is CSP } \llbracket P_1 \rrbracket_{S<} \Rightarrow \text{pre}_R Q$   
 $\llbracket P_1 \rrbracket_{S<} \wedge \text{peri}_R Q \Rightarrow \llbracket P_2 \ t \ r \rrbracket_{S<} \llbracket t \rightarrow \&tt \rrbracket \llbracket r \rightarrow \$ref' \rrbracket$   
 $\llbracket P_1 \rrbracket_{S<} \wedge \text{post}_R Q \Rightarrow \llbracket P_3 \ x \rrbracket_S \llbracket x \rightarrow \&tt \rrbracket$

**shows**  $[P_1 \vdash P_2 \text{ trace refs} \mid P_3(\text{trace})]_C \sqsubseteq Q$

**proof** –

**have**  $[P_1 \vdash P_2 \text{ trace refs} \mid P_3(\text{trace})]_C \sqsubseteq \mathbf{R}_s(\text{pre}_R(Q) \vdash \text{peri}_R(Q) \diamond \text{post}_R(Q))$

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

**thus** *?thesis*

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

**qed**

**lemma** *CRD-contract-refine'*:

**assumes**

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

**shows**  $[P_1 \vdash P_2 \text{ trace refs} \mid P_3(\text{trace})]_C \sqsubseteq Q$

**using** *assms* **by** (rule-tac CRD-contract-refine, simp-all add: refBy-order)

**lemma** *CRD-refine-CRD*:

**assumes**

$\llbracket P_1 \rrbracket_{S<} \Rightarrow (\llbracket Q_1 \rrbracket_{S<} :: ('e, 's) \text{ action})$   
 $(\llbracket P_2 \ x \ r \rrbracket_{S<} \llbracket x \rightarrow \&tt \rrbracket \llbracket r \rightarrow \$ref' \rrbracket) \sqsubseteq (\llbracket P_1 \rrbracket_{S<} \wedge \llbracket Q_2 \ x \ r \rrbracket_{S<} \llbracket x \rightarrow \&tt \rrbracket \llbracket r \rightarrow \$ref' \rrbracket :: ('e, 's) \text{ action})$   
 $\llbracket P_3 \ x \rrbracket_S \llbracket x \rightarrow \&tt \rrbracket \sqsubseteq (\llbracket P_1 \rrbracket_{S<} \wedge \llbracket Q_3 \ x \rrbracket_S \llbracket x \rightarrow \&tt \rrbracket :: ('e, 's) \text{ action})$

**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**

$\llbracket P_1 \rrbracket_{S<} \Rightarrow Q_1$   
 $(\llbracket P_2 \ x \ r \rrbracket_{S<} \llbracket x \rightarrow \&tt \rrbracket \llbracket r \rightarrow \$ref' \rrbracket) \sqsubseteq (\llbracket P_1 \rrbracket_{S<} \wedge Q_2)$   
 $\llbracket P_3 \ x \rrbracket_S \llbracket x \rightarrow \&tt \rrbracket \sqsubseteq (\llbracket P_1 \rrbracket_{S<} \wedge Q_3)$

**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**

$Q_2 \text{ is RR}$

$Q_3 \text{ is RR}$

$\llbracket P_1 \rrbracket_{S<} \Rightarrow Q_1$

$\bigwedge t. (\llbracket P_2 \ t \ r \rrbracket_{S<} \llbracket r \rightarrow \$ref' \rrbracket) \sqsubseteq (\llbracket P_1 \rrbracket_{S<} \wedge Q_2 \llbracket \llbracket \rrbracket, \llbracket t \rrbracket / \$tr, \$tr')$

$\bigwedge t. \llbracket P_3 \ t \rrbracket_{S'} \sqsubseteq (\llbracket P_1 \rrbracket_{S<} \wedge Q_3 \llbracket \llbracket \rrbracket, \llbracket t \rrbracket / \$tr, \$tr')$

**shows**  $([P_1 \vdash P_2 \text{ trace refs} \mid P_3 \text{ trace}]_C :: ('e, 's) \text{ action}) \sqsubseteq$   
 $\mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3)$   
**proof** (*simp add: mk-CRD-def, rule srdes-tri-refine-intro*)  
**show**  $'[P_1]_{S<} \Rightarrow Q_1'$  **by** (*fact assms(3)*)  
  
**have**  $\bigwedge t. ([P_2 \ t \ r]_{S<} \llbracket r \rightarrow \$ref' \rrbracket) \sqsubseteq ([P_1]_{S<} \wedge (RR \ Q_2) \llbracket \llbracket \cdot \rrbracket, \llbracket t \rrbracket / \$tr, \$tr' \rrbracket)$   
**by** (*simp add: assms Healthy-if*)  
**hence**  $'[P_1]_{S<} \wedge RR(Q_2) \Rightarrow [P_2 \ x \ r]_{S<} \llbracket x \rightarrow \&tt \rrbracket \llbracket r \rightarrow \$ref' \rrbracket'$   
**by** (*rel-simp; meson*)  
**thus**  $'[P_1]_{S<} \wedge Q_2 \Rightarrow [P_2 \ x \ r]_{S<} \llbracket x \rightarrow \&tt \rrbracket \llbracket r \rightarrow \$ref' \rrbracket'$   
**by** (*simp add: Healthy-if assms*)  
  
**have**  $\bigwedge t. [P_3 \ t]_{S'} \sqsubseteq ([P_1]_{S<} \wedge (RR \ Q_3) \llbracket \llbracket \cdot \rrbracket, \llbracket t \rrbracket / \$tr, \$tr' \rrbracket)$   
**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*  $:: 'a \text{ set} \Rightarrow ('a \Rightarrow (' \sigma, ' \varphi) \text{ action}) \Rightarrow (' \sigma, ' \varphi) \text{ action}$  **where**  
*ExtChoice-def* [*upred-defs*]:  $EXTCHOICE \ A \ F = \mathbf{R}_s((\bigsqcup P \in A \cdot pre_R(F \ P)) \vdash ((\bigsqcup P \in A \cdot cmt_R(F \ P)) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\bigsqcap P \in A \cdot cmt_R(F \ P))))$

**abbreviation** *ExtChoice*  $:: (' \sigma, ' \varphi) \text{ action set} \Rightarrow (' \sigma, ' \varphi) \text{ action}$  **where**  
*ExtChoice*  $A \equiv EXTCHOICE \ A \ id$

**syntax**

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

**translations**

$\square P \in A \cdot B \quad \Rightarrow \quad CONST \ EXTCHOICE \ A \ (\lambda P. \ B)$   
 $\square P \cdot B \quad \Rightarrow \quad CONST \ EXTCHOICE \ (CONST \ UNIV) \ (\lambda P. \ B)$

**definition** *extChoice*  $::$

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

Small external choice as an indexed big external choice.

**lemma** *extChoice-alt-def*:

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

## 7.2 Basic laws

## 7.3 Algebraic laws

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

**lemma** *ExtChoice-single*:  
 $P \text{ is CSP} \implies ExtChoice \ \{P\} = P$   
 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  $(\bigsqcup_{i \in A} \mathbf{R}_s(P(i) \vdash Q(i))) = \mathbf{R}_s((\bigsqcup_{i \in A} P(i)) \vdash ((\bigsqcup_{i \in A} Q(i)) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\bigsqcup_{i \in A} Q(i))))$

**proof** –

have  $(\bigsqcup_{i \in A} \mathbf{R}_s(P(i) \vdash Q(i))) =$   
 $\mathbf{R}_s((\bigsqcup_{i \in A} \cdot pre_R(\mathbf{R}_s(P(i) \vdash Q(i))) \vdash$   
 $((\bigsqcup_{i \in A} \cdot cmt_R(\mathbf{R}_s(P(i) \vdash Q(i)))$   
 $\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$   
 $(\bigsqcup_{i \in A} \cdot cmt_R(\mathbf{R}_s(P(i) \vdash Q(i))))))$   
 by (simp add: ExtChoice-def)

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

also have ... =  
 $\mathbf{R}_s((\bigsqcup_{i \in A} \cdot R1(R2c(pre_s \dagger P(i)))) \vdash$   
 $R1(R2c$   
 $((\bigsqcup_{i \in A} \cdot R1(R2c(cmt_s \dagger (P(i) \Rightarrow Q(i))))$   
 $\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$   
 $(\bigsqcup_{i \in A} \cdot R1(R2c(cmt_s \dagger (P(i) \Rightarrow Q(i))))))$   
 by (metis (no-types, lifting) RHS-design-export-R1 RHS-design-export-R2c)

also have ... =  
 $\mathbf{R}_s((\bigsqcup_{i \in A} \cdot R1(R2c(pre_s \dagger P(i)))) \vdash$   
 $R1(R2c$   
 $((\bigsqcup_{i \in A} \cdot (cmt_s \dagger (P(i) \Rightarrow Q(i))))$   
 $\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$   
 $(\bigsqcup_{i \in A} \cdot (cmt_s \dagger (P(i) \Rightarrow Q(i))))$   
 by (simp add: R2c-UINF R2c-cond R1-cond R1-idem R1-R2c-commute R2c-idem R1-UINF assms R1-USUP R2c-USUP)

also have ... =  
 $\mathbf{R}_s((\bigsqcup_{i \in A} \cdot R1(R2c(pre_s \dagger P(i)))) \vdash$   
 $cmt_s \dagger$   
 $((\bigsqcup_{i \in A} \cdot (cmt_s \dagger (P(i) \Rightarrow Q(i))))$   
 $\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$   
 $(\bigsqcup_{i \in A} \cdot (cmt_s \dagger (P(i) \Rightarrow Q(i))))$   
 by (metis (no-types, lifting) RHS-design-export-R1 RHS-design-export-R2c rdes-export-cmt)

also have ... =  
 $\mathbf{R}_s((\bigsqcup_{i \in A} \cdot R1(R2c(pre_s \dagger P(i)))) \vdash$   
 $cmt_s \dagger$   
 $((\bigsqcup_{i \in A} \cdot (P(i) \Rightarrow Q(i))))$

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

$\mathbf{R}_s ((\sqcup i \in A \cdot P_1(i)) \vdash (((\sqcup i \in A \cdot P_2(i) \diamond P_3(i)) \triangleleft \$tr' =_u \$tr \triangleright (\sqcap i \in A \cdot P_2(i) \diamond P_3(i))) \diamond (\sqcap i \in A \cdot P_2(i) \diamond P_3(i))))$   
 by (simp add: cond-conj wait'-cond-def)  
 also  
 have ... =  $\mathbf{R}_s ((\sqcup i \in A \cdot P_1(i)) \vdash (((\sqcup i \in A \cdot P_2(i)) \triangleleft \$tr' =_u \$tr \triangleright (\sqcap i \in A \cdot P_2(i))) \diamond (\sqcap i \in A \cdot P_3(i))))$   
 by (rule cong[of  $\mathbf{R}_s \mathbf{R}_s$ ], simp, rel-auto)  
 finally show ?thesis .  
 qed

**lemma** *ExtChoice-tri-rdes'* [rdes-def]:  
 assumes  $\bigwedge i. \$ok' \# P_1(i) \ A \neq \{\}$   
 shows  $(\sqcap i \in A \cdot \mathbf{R}_s(P_1(i) \vdash P_2(i) \diamond P_3(i))) =$   
 $\mathbf{R}_s ((\sqcup i \in A \cdot P_1(i)) \vdash (((\sqcup i \in A \cdot R_5(P_2(i))) \vee (\sqcap i \in A \cdot R_4(P_2(i)))) \diamond (\sqcap i \in A \cdot P_3(i))))$   
 by (simp add: ExtChoice-tri-rdes assms, rel-auto, simp-all add: less-le assms)

**lemma** *ExtChoice-tri-rdes-def*:  
 assumes  $\bigwedge i. i \in A \implies F \ i \text{ is CSP}$   
 shows  $(\sqcap i \in A \cdot F \ i) = \mathbf{R}_s ((\sqcup P \in A \cdot pre_R (F \ P)) \vdash (((\sqcup P \in A \cdot peri_R (F \ P)) \triangleleft \$tr' =_u \$tr \triangleright (\sqcap P \in A \cdot peri_R (F \ P))) \diamond (\sqcap P \in A \cdot post_R (F \ P))))$   
**proof** –  
 have  $((\sqcup P \in A \cdot cmt_R (F \ P)) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\sqcap P \in A \cdot cmt_R (F \ P))) =$   
 $((\sqcup P \in A \cdot cmt_R (F \ P)) \triangleleft \$tr' =_u \$tr \triangleright (\sqcap P \in A \cdot cmt_R (F \ P))) \diamond (\sqcap P \in A \cdot cmt_R (F \ P))$   
 by (rel-auto)  
 also have ... =  $((\sqcup P \in A \cdot peri_R (F \ P)) \triangleleft \$tr' =_u \$tr \triangleright (\sqcap P \in A \cdot peri_R (F \ P))) \diamond (\sqcap P \in A \cdot post_R (F \ P))$   
 by (rel-auto)  
 finally show ?thesis  
 by (simp add: ExtChoice-def)  
 qed

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

**lemma** *extChoice-tri-rdes*:  
 assumes  $\$ok' \# P_1 \ \$ok' \# Q_1$   
 shows  $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) \sqcap \mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3) =$   
 $\mathbf{R}_s ((P_1 \wedge Q_1) \vdash (((P_2 \wedge Q_2) \triangleleft \$tr' =_u \$tr \triangleright (P_2 \vee Q_2)) \diamond (P_3 \vee Q_3)))$   
**proof** –  
 have  $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) \sqcap \mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3) =$   
 $\mathbf{R}_s ((P_1 \wedge Q_1) \vdash ((P_2 \diamond P_3 \wedge Q_2 \diamond Q_3) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (P_2 \diamond P_3 \vee Q_2 \diamond Q_3)))$   
 by (simp add: extChoice-rdes assms)  
 also

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

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

**lemma** *CSP-ExtChoice [closure]*:  
 $EXTCHOICE A F$  is *CSP*  
**by** (*simp add: ExtChoice-def RHS-design-is-SRD unrest*)

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

**lemma** *preR-EXTCHOICE [rdes]*:  
**assumes**  $A \neq \{\}$   $\bigwedge i. i \in A \implies F i$  is *NCSP*  
**shows**  $pre_R(EXTCHOICE A F) = (\bigsqcup_{P \in A} P \cdot pre_R(F P))$   
**by** (*simp add: ExtChoice-tri-rdes-def closure rdes assms*)

**lemma** *preR-ExtChoice*:  
**assumes**  $A \neq \{\}$   $\forall P \in A. P$  is *NCSP*  
**shows**  $pre_R(ExtChoice A) = (\bigsqcup_{P \in A} P \cdot pre_R(P))$   
**using** *assms* **by** (*auto simp add: preR-EXTCHOICE*)

**lemma** *periR-ExtChoice [rdes]*:  
**assumes**  $A \neq \{\}$   $\bigwedge i. i \in A \implies F i$  is *NCSP*  
**shows**  $peri_R(EXTCHOICE A F) = (((\bigsqcup_{P \in A} P \cdot pre_R(F P)) \Rightarrow_r (\bigsqcup_{P \in A} P \cdot peri_R(F P))) \triangleleft U(\$tr' = \$tr) \triangleright (\bigsqcup_{P \in A} P \cdot peri_R(F P)))$   
**(is ?lhs = ?rhs)**

**proof** –

**have** *?lhs* =  $((\bigsqcup_{P \in A} P \cdot pre_R(F P)) \Rightarrow_r (\bigsqcup_{P \in A} P \cdot peri_R(F P)) \triangleleft U(\$tr' = \$tr) \triangleright (\bigsqcup_{P \in A} P \cdot peri_R(F P)))$   
**by** (*simp add: ExtChoice-tri-rdes-def closure rdes assms*)  
**also have** ... =  $((\bigsqcup_{P \in A} P \cdot pre_R(F P)) \Rightarrow_r (\bigsqcup_{P \in A} P \cdot pre_R(F P) \Rightarrow_r peri_R(F P)) \triangleleft U(\$tr' = \$tr) \triangleright (\bigsqcup_{P \in A} P \cdot pre_R(F P) \Rightarrow_r peri_R(F P)))$   
**by** (*simp add: NSRD-peri-under-pre assms closure cong: UINF-cong USUP-cong*)  
**also have** ... =  $((\bigsqcup_{P \in A} P \cdot RR(pre_R(F P))) \Rightarrow_r (\bigsqcup_{P \in A} P \cdot RR(pre_R(F P)) \Rightarrow_r RR(peri_R(F P)))$



$\triangleleft U(\$tr' = \$tr) \triangleright ((\prod P \in A \cdot RR(pre_R (F P)) \Rightarrow_r RR(peri_R (F P))))$   
 by (simp add: Healthy-if assms closure cong: UINF-cong USUP-cong)  
 also from assms(1) have ... =  $((\prod P \in A \cdot RR(pre_R (F P))) \Rightarrow_r (\prod P \in A \cdot RR(pre_R (F P)) \Rightarrow_r RR(peri_R (F P)))) \triangleleft U(\$tr' = \$tr) \triangleright ((\prod P \in A \cdot RR(pre_R (F P)) \Rightarrow_r RR(peri_R (F P))))$   
 by (rel-auto)  
 finally show ?thesis  
 by (simp add: Healthy-if NSRD-peri-under-pre assms closure cong: UINF-cong USUP-cong)  
 qed

**lemma** *periR-ExtChoice'*:

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

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

assumes  $A \neq \{\}$   $\bigwedge i. i \in A \implies F i$  is NCSP  
 shows  $post_R(EXTCHOICE A F) = (\prod P \in A \cdot post_R (F P))$   
 (is ?lhs = ?rhs)

**proof** –

have ?lhs =  $((\prod P \in A \cdot pre_R (F P)) \Rightarrow_r (\prod P \in A \cdot post_R (F P)))$   
 by (simp add: ExtChoice-tri-rdes-def closure rdes assms)  
 also have ... =  $((\prod P \in A \cdot pre_R (F P)) \Rightarrow_r (\prod P \in A \cdot pre_R (F P) \Rightarrow_r post_R (F P)))$   
 by (simp add: NSRD-post-under-pre assms closure cong: UINF-cong)  
 also have ... =  $(\prod P \in A \cdot pre_R (F P) \Rightarrow_r post_R (F P))$   
 by (rel-auto)  
 finally show ?thesis  
 by (simp add: NSRD-post-under-pre assms closure cong: UINF-cong)  
 qed

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

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

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

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

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

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

**lemma** *ExtChoice-cong*:

assumes  $\bigwedge P. P \in A \implies F(P) = G(P)$   
 shows  $(\prod P \in A \cdot F(P)) = (\prod P \in A \cdot G(P))$   
 by (simp add: ExtChoice-def assms cong: UINF-cong USUP-cong)

**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  $\bigwedge i. i \in A \implies F\ i\ \text{is}\ NCSP$ 
  shows EXTCHOICE A F is CSP4
proof (cases A = {})
  case True thus ?thesis
    by (simp add: ExtChoice-empty Healthy-def CSP4-def, simp add: Skip-is-CSP Stop-left-zero)
next
  case False
  have 1:  $(\neg_r (\neg_r pre_R (EXTCHOICE\ A\ F)) \;;_h R1\ true) = pre_R (EXTCHOICE\ A\ F)$ 
  proof -
    have  $\bigwedge P. P \in A \implies (\neg_r pre_R(F\ P)) \;; R1\ true = (\neg_r pre_R(F\ P))$ 
      by (simp add: NCSP-Healthy-subset-member NCSP-implies-NSRD NSRD-neg-pre-unit assms)
    thus ?thesis
      apply (simp add: False preR-EXTCHOICE closure NCSP-set-unrest-pre-wait' assms not-UINF
        seq-UINF-distr not-USUP)
      apply (rule USUP-cong)
      apply (simp add: rpred assms closure)
      done
  qed
  have 2:  $\$st' \# peri_R (EXTCHOICE\ A\ F)$ 
  proof -
    have a:  $\bigwedge P. P \in A \implies \$st' \# pre_R(F\ P)$ 
      by (simp add: NCSP-Healthy-subset-member NCSP-implies-NSRD NSRD-st'-unrest-pre assms)
    have b:  $\bigwedge P. P \in A \implies \$st' \# peri_R(F\ P)$ 
      by (simp add: NCSP-Healthy-subset-member NCSP-implies-NSRD NSRD-st'-unrest-peri assms)
    from a b show ?thesis
      apply (subst periR-ExtChoice)
      apply (simp-all add: assms closure unrest CSP4-set-unrest-pre-st' NCSP-set-unrest-pre-wait'
        False)
      done
  qed
  have 3:  $\$ref' \# post_R (EXTCHOICE\ A\ F)$ 
  proof -
    have a:  $\bigwedge P. P \in A \implies \$ref' \# pre_R(F\ P)$ 
      by (simp add: CSP4-ref'-unrest-pre assms closure)
    have b:  $\bigwedge P. P \in A \implies \$ref' \# post_R(F\ P)$ 
      by (simp add: CSP4-ref'-unrest-post assms closure)
    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, auto simp add: assms)

lemma NCSP-ExtChoice [closure]:
  assumes  $\bigwedge i. i \in A \implies F\ i$  is NCSP
  shows EXTCHOICE  $A\ F$  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)
  show 1: EXTCHOICE  $A\ F$  is CSP
    by (metis (mono-tags) CSP-ExtChoice)
  show EXTCHOICE  $A\ F$  is CSP3
    by (rule-tac CSP3-SRD-intro, simp-all add: CSP-Healthy-subset-member CSP3-Healthy-subset-member
        closure rdes unrest assms 1 False)
  show EXTCHOICE  $A\ F$  is CSP4
    by (simp add: CSP4-ExtChoice assms)
qed
qed

```

```

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

```

```

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

```

## 7.5 Productivity and Guardedness

```

lemma Productive-ExtChoice [closure]:
  assumes  $\bigwedge i. i \in I \implies P(i)$  is NCSP  $\bigwedge i. i \in I \implies P(i)$  is Productive
  shows EXTCHOICE  $I\ P$  is Productive
proof (cases  $I = \{\}$ )
  case True
  then show ?thesis
    by (simp add: ExtChoice-empty Productive-Stop)
next
  case False
  have 1:  $\bigwedge i. i \in I \implies \$wait' \# pre_R(P\ i)$ 
    using NCSP-implies-NSRD NSRD-wait'-unrest-pre assms(1) by blast

  show ?thesis
proof (rule Productive-intro, simp-all add: assms closure rdes unrest 1 False)
  have  $((\bigsqcap_{i \in I} pre_R(P\ i)) \wedge (\bigsqcap_{i \in I} post_R(P\ i))) =$ 
     $((\bigsqcap_{i \in I} pre_R(P\ i)) \wedge (\bigsqcap_{i \in I} (pre_R(P\ i) \wedge post_R(P\ i))))$ 
    by (rel-auto)

```

**moreover have**  $(\prod_{i \in I} (pre_R (P \ i) \wedge post_R (P \ i))) = (\prod_{i \in I} ((pre_R (P \ i) \wedge post_R (P \ i)) \wedge \$tr <_u \$tr'))$   
**by** (rule UINF-cong, metis (no-types, lifting) 1 NCSP-implies-CSP Productive-post-refines-tr-increase  
 assms utp-pred-laws.inf.absorb1)

**ultimately show**  $U(\$tr < \$tr') \sqsubseteq ((\prod_{i \in I} pre_R (P \ i)) \wedge ((\prod_{i \in I} post_R (P \ i))))$   
**by** (rel-auto)  
 qed  
 qed

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

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

**lemma** *ExtChoice-image*: *ExtChoice*  $(P \ ' A) = \text{EXTCHOICE } A \ P$   
**by** (rel-auto)

**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** (*subst (asm) ExtChoice-image[THEN sym], 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:*

$P \text{ is CSP} \implies P \sqcap P = P$   
**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  $Stop \sqcap Q = Q$

using *assms*

**proof** –

have  $Stop \sqcap Q = \mathbf{R}_s (true \vdash (\$tr' =_u \$tr \wedge \$wait')) \sqcap \mathbf{R}_s (pre_R(Q) \vdash cmt_R(Q))$

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

also have  $\dots = \mathbf{R}_s (pre_R Q \vdash (((\$tr' =_u \$tr \wedge \$wait') \wedge cmt_R Q) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\$tr' =_u \$tr \wedge \$wait' \vee cmt_R Q)))$

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

also have  $\dots = \mathbf{R}_s (pre_R Q \vdash (cmt_R Q \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright cmt_R Q))$

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

also have  $\dots = \mathbf{R}_s (pre_R Q \vdash cmt_R Q)$

by (*simp add: cond-idem*)

also have  $\dots = Q$

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  $\dots = \mathbf{R}_s (false \vdash (cmt_R Q \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright true))$

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

also have  $\dots = \mathbf{R}_s (false \vdash true)$

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

also have  $\dots = Chaos$

by (*simp add: Chaos-def*)

finally show *?thesis* .

**qed**

**lemma** *extChoice-Dist*:

assumes *P* is CSP  $S \subseteq \llbracket CSP \rrbracket_H S \neq \{\}$

shows  $P \sqcap (\bigsqcap S) = (\bigsqcap_{Q \in S} P \sqcap Q)$

**proof** –

let *?S1* =  $pre_R ' S$  and *?S2* =  $cmt_R ' S$

have  $P \sqcap (\bigsqcap S) = P \sqcap (\bigsqcap_{Q \in S} Q \cdot \mathbf{R}_s (pre_R(Q) \vdash cmt_R(Q)))$

by (*simp add: SRD-as-reactive-design[THEN sym] Healthy-SUPREMUM UINF-as-Sup-collect assms*)

also have  $\dots = \mathbf{R}_s (pre_R(P) \vdash cmt_R(P)) \sqcap \mathbf{R}_s ((\bigsqcap_{Q \in S} Q \in S \cdot pre_R(Q)) \vdash (\bigsqcap_{Q \in S} Q \in S \cdot cmt_R(Q)))$

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

also have  $\dots = \mathbf{R}_s ((pre_R(P) \wedge (\bigsqcap_{Q \in S} Q \in S \cdot pre_R(Q))) \vdash ((cmt_R(P) \wedge (\bigsqcap_{Q \in S} Q \in S \cdot cmt_R(Q))) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (cmt_R(P) \vee (\bigsqcap_{Q \in S} Q \in S \cdot cmt_R(Q))))))$

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

also have  $\dots = \mathbf{R}_s ((\bigsqcap_{Q \in S} Q \in S \cdot pre_R P \wedge pre_R Q) \vdash$

$(\bigsqcap_{Q \in S} Q \in S \cdot (cmt_R P \wedge cmt_R Q) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (cmt_R P \vee cmt_R Q)))$

by (*simp add: conj-USUP-dist conj-UINF-dist disj-UINF-dist cond-UINF-dist assms*)

also have  $\dots = (\bigsqcap_{Q \in S} Q \in S \cdot \mathbf{R}_s ((pre_R P \wedge pre_R Q) \vdash$

$((cmt_R P \wedge cmt_R Q) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (cmt_R P \vee cmt_R Q))))$

by (*simp add: assms RHS-design-USUP*)

also have  $\dots = (\bigsqcap_{Q \in S} Q \in S \cdot \mathbf{R}_s (pre_R(P) \vdash cmt_R(P)) \sqcap \mathbf{R}_s (pre_R(Q) \vdash cmt_R(Q)))$

by (simp add: extChoice-rdes unrest)  
 also have ... = ( $\prod_{Q \in S} P \sqcap \text{CSP}(Q)$ )  
 by (simp add: UINF-as-Sup-collect, metis (no-types, lifting) Healthy-if SRD-as-reactive-design  
 assms(1))  
 also have ... = ( $\prod_{Q \in S} P \sqcap Q$ )  
 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  $\bigwedge i. i \in A \implies P \text{ is PCSP } Q \text{ is NCSP}$   
 shows  $(\sqcap_{i \in A} P \text{ i}) ;; Q = (\sqcap_{i \in A} P \text{ i} ;; Q)$   
**proof** (cases  $A = \{\}$ )  
 case True  
 then show ?thesis  
 by (simp add: ExtChoice-empty NCSP-implies-CSP Stop-left-zero assms(2))  
**next**  
 case False  
 show ?thesis  
**proof** –  
 have 1:  $(\sqcap_{i \in A} P \text{ i}) = (\sqcap_{i \in A} \cdot (\mathbf{R}_s ((\text{pre}_R (P \text{ i})) \vdash \text{peri}_R (P \text{ i}) \diamond (R4(\text{post}_R (P \text{ i}))))))$   
 (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:  $(\sqcap_{i \in A} P \text{ i} ;; Q) = (\sqcap_{i \in A} \cdot (\mathbf{R}_s ((\text{pre}_R (P \text{ i})) \vdash \text{peri}_R (P \text{ i}) \diamond (R4(\text{post}_R (P \text{ i})))))) ;; Q$   
 (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  $(P \sqcap Q) ;; R = (P ;; R \sqcap Q ;; R)$   
 by (rdes-eq' cls: assms)

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

**lemma** extchoice-StateInvR-refine:  
 assumes  
 $P$  is NCSP  $Q$  is NCSP  
 $\text{inv}_R(b) \sqsubseteq P \text{ inv}_R(b) \sqsubseteq Q$   
 shows  $\text{inv}_R(b) \sqsubseteq P \sqcap Q$   
**proof** –  
 have 1:

```

  preR P ⊆ [b]S< [b]S> ⊆ ([b]S< ∧ postR P)
  preR Q ⊆ [b]S< [b]S> ⊆ ([b]S< ∧ postR Q)
  by (metis (no-types, lifting) CRR-implies-RR NCSP-implies-CSP RHS-tri-design-refine SRD-reactive-tri-design
    StateInvR-def assms periR-RR postR-RR preR-CRR rea-st-cond-RR rea-true-RR refBy-order st-post-CRR)+
  show ?thesis
  by (rdes-refine-split cls: assms(1-2), simp-all add: 1 closure assms truer-bottom-rpred utp-pred-laws.inf-sup-distrib1)
qed

end

```

## 8 Stateful-Failure Programs

```

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

```

### 8.1 Conditionals

```

lemma NCSP-cond-srea [closure]:
  assumes P is NCSP Q is NCSP
  shows P ◁ b ▷R Q 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 →R P is NCSP
  by (simp add: gcmd-def closure assms)

```

### 8.3 Alternation

```

lemma AlternateR-NCSP-closed [closure]:
  assumes ⋀ i. i ∈ A ⇒ P(i) is NCSP Q is NCSP
  shows (ifR i ∈ A • g(i) → P(i) else Q fi) is NCSP
proof (cases A = {})
  case True
  then show ?thesis
    by (simp add: assms)
next
  case False
  then show ?thesis
    by (simp add: AlternateR-def closure assms)
qed

```

```

lemma AlternateR-list-NCSP-closed [closure]:
  assumes ⋀ b P. (b, P) ∈ set A ⇒ P is NCSP Q is NCSP
  shows (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 Specification Statement

**definition**  $\text{Spec}C :: ('a \Rightarrow 's) \Rightarrow 's \text{ upred} \Rightarrow 's \text{ upred} \Rightarrow ('s, 'e) \text{ action } (-:[-,-]_C [999,0,0] 999) \text{ where}$   
 $[\text{rdes-def}]: \text{Spec}C \text{ frm pre post} = \mathbf{R}_s([pre]_{S<} \vdash \text{false} \diamond [frm:[post>]]_S)$

**lemma**  $\text{Spec}C\text{-is-NCSP}$   $[\text{closure}]: \text{frm}:[pre,post]_C \text{ is NCSP}$

**apply**  $(\text{simp add: Spec}C\text{-def})$   
**apply**  $(\text{rule NCSP-rdes-intro})$   
**apply**  $(\text{simp-all add: closure unrest})$   
**apply**  $(\text{rel-auto})+$   
**done**

**lemma**  $\text{Spec}C\text{-skip}: \{\}_v:[true,true]_C = \text{Skip}$   
**by**  $(\text{rdes-eq})$

**lemma**  $\text{Spec}C\text{-false-pre}: a:[false,q]_C = \text{Chaos}$   
**by**  $(\text{rdes-eq})$

**lemma**  $\text{Spec}C\text{-false-post}: a:[true,false]_C = \text{Miracle}$   
**by**  $(\text{rdes-eq})$

**lemma**  $\text{Spec}C\text{-refine-seq}$ :  
 $\text{vwb-lens } a \Rightarrow a:[p,q]_C \sqsubseteq a:[p,r]_C ;; a:[r,q]_C$   
**by**  $((\text{rdes-refine-split}; \text{rel-simp}), \text{metis vwb-lens.put-eq})$

## 8.5 Assumptions

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

**lemma**  $\text{AssumeCircus-rdes-def}$   $[\text{rdes-def}]: [b]_C = \mathbf{R}_s(\text{true}_r \vdash \text{false} \diamond [b]_c)$   
**unfolding**  $\text{AssumeCircus-def}$  **by**  $\text{rdes-eq}$

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

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

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

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

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

## 8.6 While Loops

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

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**  
 $while_C \ b \ do \ P \ od = Skip \ ; \ while_R \ b \ do \ P \ od \ ; \ Skip$

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

**assumes**  $P$  is NCSP  $P$  is Productive

**shows**  $while_C \ b \ do \ P \ od$  is NCSP

**by** (simp add:  $WhileC\text{-}def$  NSRD-coerce-NCSP assms closure)

**theorem**  $WhileC\text{-}iter\text{-form}$ :

**assumes**  $P$  is NCSP  $P$  is Productive

**shows**  $while_C \ b \ do \ P \ od = ([b]_C \ ; \ P)^{*C} \ ; \ [\neg b]_C$

**by** (simp add:  $WhileC\text{-}def$   $WhileR\text{-}iter\text{-form}$  assms closure)

(metis (no-types, lifting)  $StarC\text{-}def$   $AssumeCircus\text{-}AssumeR(2)$   $AssumeCircus\text{-}NCSP$  RA1 assms(1)  $csp\text{-}theory.Healthy\text{-}Sequence$   $csp\text{-}theory.Star\text{-}Healthy$   $csp\text{-}theory.Unit\text{-}Left$   $sfrd\text{-}star\text{-}as\text{-}rdes\text{-}star$ )

**theorem**  $WhileC\text{-}rdes\text{-}def$  [rdes-def]:

**assumes**  $P$  is CRC  $Q$  is CRR  $R$  is CRF  $\$st' \ \# \ Q \ R$  is R4

**shows**  $while_C \ b \ do \ \mathbf{R}_s(P \vdash Q \diamond R) \ od =$

$\mathbf{R}_s \ (([b]_c \ ; \ R)^{*c} \ wp_r \ ([b]_{S<} \Rightarrow_r P) \vdash \ (([b]_c \ ; \ R)^{*c} \ ; \ [b]_c \ ; \ Q) \diamond \ (([b]_c \ ; \ R)^{*c} \ ; \ [\neg b]_c))$   
(is ?lhs = ?rhs)

**proof** –

**have** ?lhs =  $([b]_C \ ; \ \mathbf{R}_s(P \vdash Q \diamond R))^{*C} \ ; \ [\neg b]_C$

**by** (simp add:  $WhileC\text{-}iter\text{-form}$  assms closure unrest Productive-rdes-RR-intro)

**also have** ... = ?rhs

**by** (simp add: rdes-def assms closure unrest rpred wp del: rea-star-wp)

**finally show** ?thesis .

qed

**lemma**  $WhileC\text{-}false$ :

$P$  is NCSP  $\implies WhileC \ false \ P = Skip$

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

**lemma**  $WhileC\text{-}unfold$ :

**assumes**  $P$  is NCSP  $P$  is Productive

**shows**  $WhileC \ b \ P = (P \ ; \ WhileC \ b \ P) \triangleleft b \triangleright_R Skip$

**proof** –

**have**  $WhileC \ b \ P = (Skip \vee [b]_C \ ; \ P \ ; \ ([b]_C \ ; \ P)^{*C}) \ ; \ [\neg b]_C$

**by** (simp add:  $WhileC\text{-}iter\text{-form}$  assms closure)

(metis (no-types, lifting)  $AssumeCircus\text{-}NCSP$  RA1  $StarC\text{-}unfold$  assms(1)  $csp\text{-}theory.Healthy\text{-}Sequence$   $disj\text{-}upred\text{-}def$ )

**also have** ... =  $([\neg b]_C \vee [b]_C \ ; \ P \ ; \ ([b]_C \ ; \ P)^{*C}) \ ; \ [\neg b]_C$

**by** (metis (no-types, lifting)  $AssumeCircus\text{-}AssumeR(1)$  RA1  $csp\text{-}theory.Unit\text{-}self \text{ seqr-or-distl}$ )

**also have** ... =  $(P \ ; \ WhileC \ b \ P) \triangleleft b \triangleright_R Skip$

**by** (metis (no-types, lifting)  $AssumeCircus\text{-}AssumeR(2)$  NCSP-implies-NSRD RA1  $WhileC\text{-}NCSP\text{-closed}$   $WhileC\text{-}iter\text{-form}$  assms(1) assms(2)  $cond\text{-}srea\text{-}AssumeR\text{-}form$   $csp\text{-}theory.Healthy\text{-}Sequence$   $csp\text{-}theory.Healthy\text{-}Unit$   $csp\text{-}theory.Unit\text{-}Left$   $uinf\text{-}or \ utp\text{-}pred\text{-}laws.sup\text{-}commute$ )

**finally show** ?thesis .

qed

## 8.7 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]:  $IterateC \ A \ g \ P = while_C \ (\bigvee_{i \in A} g(i)) \ do \ (if_R \ i \in A \cdot g(i) \rightarrow P(i) \ fi)$   
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* ::  $(\text{'s upred} \times (\text{'s, 'e}) \text{ action}) \text{ list} \Rightarrow (\text{'s, 'e}) \text{ action}$  **where**  
*[upred-defs, ndes-simp]*:  
 $\text{IterateC-list } xs = \text{IterateC } \{0..<\text{length } xs\} \ (\lambda \ i. \ \text{map } \text{fst } xs \ ! \ i) \ (\lambda \ i. \ \text{map } \text{snd } xs \ ! \ i)$

**syntax**

*-iter-C* ::  $\text{pttrn} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic} \ (\text{do}_C \ - \in \cdot \rightarrow \cdot \text{ od})$   
*-iter-gcommC* ::  $\text{gcomms} \Rightarrow \text{logic} \ (\text{do}_C / \cdot / \text{od})$

**translations**

*-iter-C*  $x \ A \ g \ P \Rightarrow \text{CONST } \text{IterateC } A \ (\lambda \ x. \ g) \ (\lambda \ x. \ P)$   
*-iter-C*  $x \ A \ g \ P \Leftarrow \text{CONST } \text{IterateC } A \ (\lambda \ x. \ g) \ (\lambda \ x'. \ P)$   
*-iter-gcommC*  $cs \rightarrow \text{CONST } \text{IterateC-list } cs$   
*-iter-gcommC*  $(\text{-gcomm-show } cs) \leftarrow \text{CONST } \text{IterateC-list } cs$

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

**assumes**

$\bigwedge \ i. \ i \in I \Longrightarrow P(i) \text{ is NCSP}$

$\bigwedge \ i. \ i \in I \Longrightarrow P(i) \text{ is Productive}$

**shows**  $\text{do}_C \ i \in I \cdot g(i) \rightarrow P(i) \text{ od is NCSP}$

**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**

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

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

**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*:

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

**proof** –

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

**by** (*rel-auto, 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*:

$\text{do}_C \ i \in \{\} \cdot g(i) \rightarrow P(i) \text{ od} = \text{Skip}$

**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**  $\text{do}_C \ i \in \{k\} \cdot g(i) \rightarrow P(i) \text{ od} = \text{while}_C \ g(k) \ \text{do } P(k) \text{ od}$  (**is** *?lhs = ?rhs*)

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

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

**assumes**  $I \neq \{\} \bigwedge \ i. \ i \in I \Longrightarrow P \ i \text{ is NCSP} \bigwedge \ i. \ i \in I \Longrightarrow P \ i \text{ is Productive}$

$\bigwedge i. i \in I \implies S \sqsubseteq (b\ i \rightarrow_R P\ i \ ;\ S)\ S\ \text{is NCSP}$   
 $S \sqsubseteq [\neg (\bigcap i \in I \cdot b\ i)]^\top_R$   
**shows**  $S \sqsubseteq \text{do}_C\ i \in I \cdot b(i) \rightarrow P(i)\ \text{od}$   
**proof** –  
**have**  $S \sqsubseteq \text{do}_R\ i \in I \cdot b(i) \rightarrow P(i)\ \text{od}$   
**by** (*simp add: IterateR-outer-refine-intro NCSP-implies-NSRD assms*)  
**thus** ?thesis  
**unfolding** IterateC-IterateR-def  
**by** (*metis (full-types) Skip-left-unit Skip-right-unit assms(5) urel-dioid.mult-isol urel-dioid.mult-isor*)  
**qed**

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

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

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

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

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

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

## 8.8 Assignment

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

**abbreviation** *AssignCSP*  $x v \equiv \mathbf{R}_s([\&\mathbf{v} \in_u \ll \mathcal{S}_x \gg]_{S<} \vdash \text{false} \diamond \Phi(\text{true}, [x \mapsto_s v], \ll [] \gg))$

**syntax**

*-assigns-csp* :: *svids*  $\Rightarrow$  *uexprs*  $\Rightarrow$  *logic*  $(\langle \cdot \rangle :=_C \langle \cdot \rangle)$   
*-assigns-csp* :: *svids*  $\Rightarrow$  *uexprs*  $\Rightarrow$  *logic* **(infixr :=<sub>C</sub> 64)**

**translations**

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

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

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

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

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

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

**lemma** *AssignC-init-refine-intro*:  
 assumes  
    $vwb\text{-}lens\ x\ \$st:x\ \# P_2\ \$st:x\ \# P_3$   
    $P_2\ is\ RR\ P_3\ is\ RR\ Q\ is\ NCSP$   
    $\mathbf{R}_s([\&x =_u \ll k \gg]_{S<} \vdash P_2 \diamond P_3) \sqsubseteq Q$   
 shows  $\mathbf{R}_s(true_r \vdash P_2 \diamond P_3) \sqsubseteq (x :=_C \ll k \gg) ; Q$   
 by (simp add: *AssignsCSP-as-AssignsR[THEN sym]* assms seqr-assoc Skip-left-unit AssignR-init-refine-intro closure)

**lemma** *AssignsCSP-refines-sinv*:  
 assumes ' $\sigma \uparrow b$ '  
 shows  $sinv_R(b) \sqsubseteq \langle \sigma \rangle_C$   
 apply (rdes-refine-split)  
 apply (simp-all)  
 apply (metis rea-st-cond-true st-cond-conj utp-pred-laws.inf.absorb-iff2 utp-pred-laws.inf-top-left)  
 using assms apply (rel-auto)  
 done

## 8.9 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* ::  
 ( $'f \Rightarrow 'k\ set \Rightarrow ('f \Rightarrow 'k \Rightarrow 'v \Rightarrow 'f) \Rightarrow ('f \Rightarrow 'v) \Rightarrow$   
 ( $'k, 'v) uexpr \Rightarrow ('v, 'v) uexpr \Rightarrow ('v, 'v) action$  **where**  
 [*upred-defs, rdes-def*]: *AssignCSP-update* domf updatef  $x\ k\ v =$   
 $\mathbf{R}_s([k \in_u uop\ domf\ (\&x)]_{S<} \vdash false \diamond \Phi(true, [x \mapsto_s trop\ updatef\ (\&x)\ k\ v], \ll [] \gg))$

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

**syntax**

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

**translations**

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

**lemma** *AssignCSP-update-CSP* [closure]:  
*AssignCSP-update domf updatef x k v is CSP*  
**by** (*simp add: AssignCSP-update-def RHS-tri-design-is-SRD unrest*)

**lemma** *preR-AssignCSP-update* [rdes]:  
 $pre_R(AssignCSP-update\ domf\ updatef\ x\ k\ v) = [k \in_u uop\ domf\ (\&x)]_{S<}$   
**by** (*rel-auto*)

**lemma** *periR-AssignCSP-update* [rdes]:  
 $peri_R(AssignCSP-update\ domf\ updatef\ x\ k\ v) = [k \notin_u uop\ domf\ (\&x)]_{S<}$   
**by** (*rel-simp*)

**lemma** *post-AssignCSP-update* [rdes]:  
 $post_R(AssignCSP-update\ domf\ updatef\ x\ k\ v) =$   
 $(\Phi(true, [x \mapsto_s trop\ updatef\ (\&x)\ k\ v], \llbracket \rrbracket) \triangleleft (k \in_u uop\ domf\ (\&x)) \triangleright_R R1(true))$   
**by** (*rel-auto*)

**lemma** *AssignCSP-update-NCSP* [closure]:  
*(AssignCSP-update domf updatef x k v) is NCSP*  
**proof** (*rule NCSP-intro*)  
**show** *(AssignCSP-update domf updatef x k v) is CSP*  
**by** (*simp add: closure*)  
**show** *(AssignCSP-update domf updatef x k v) is CSP3*  
**by** (*rule CSP3-SRD-intro, simp-all add: csp-do-def closure rdes unrest*)  
**show** *(AssignCSP-update domf updatef x k v) is CSP4*  
**by** (*rule CSP4-tri-intro, simp-all add: csp-do-def closure rdes unrest, rel-auto*)  
**qed**

## 8.10 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*)<sup>+</sup>

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

## 8.11 Guards

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

**syntax**  
 $-GuardCSP :: logic \Rightarrow logic \Rightarrow logic\ (\mathbf{infixr}\ \&_C\ 60)$

**translations**  
 $-GuardCSP\ b\ P == CONST\ GuardCSP\ b\ P$

**lemma** *Guard-tri-design*:

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

**lemma** *csp-do-cond-conj*:

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

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

**assumes**  $P$  is RR  $Q$  is CRR  $R$  is CRR  
**shows**  $g \&_C \mathbf{R}_s(P \vdash Q \diamond R) = \mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r P) \vdash ((\Phi(g, id_s, \llbracket \gg \rrbracket) ;; Q) \vee \mathcal{E}(\neg g, \llbracket \gg \rrbracket, \{u\})) \diamond (\Phi(g, id_s, \llbracket \gg \rrbracket) ;; R))$   
*(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  $\dots = \mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r P) \vdash ((\lceil g \rceil_{S<} \wedge Q) \vee \mathcal{E}(\neg g, \llbracket \gg \rrbracket, \{u\})) \diamond (\lceil g \rceil_{S<} \wedge R))$   
 by *(rel-auto)*  
 also have  $\dots = \mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r P) \vdash ((\Phi(g, id_s, \llbracket \gg \rrbracket) ;; Q) \vee \mathcal{E}(\neg g, \llbracket \gg \rrbracket, \{u\})) \diamond (\Phi(g, id_s, \llbracket \gg \rrbracket) ;; R))$   
 by *(simp add: assms(2) assms(3) csp-do-cond-conj)*  
**finally show** *?thesis* .  
**qed**

**lemma** *Guard-rdes-def'*:

**assumes**  $\$ok' \nmid P$   
**shows**  $g \&_C (\mathbf{R}_s(P \vdash Q)) = \mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r P) \vdash (\lceil g \rceil_{S<} \wedge Q \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$   
**proof** –  
 have  $g \&_C (\mathbf{R}_s(P \vdash Q)) = \mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r \text{pre}_R (\mathbf{R}_s(P \vdash Q))) \vdash (\lceil g \rceil_{S<} \wedge \text{cmt}_R (\mathbf{R}_s(P \vdash Q)) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$   
 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]*:  $b \&_C P$  is CSP

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

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

**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**  $peri_R(b \&_C P) = (peri_R P \triangleleft b \triangleright_R \mathcal{E}(true, \llbracket \cdot \rrbracket, \{ \}_u))$

**proof** –

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

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

**proof** –

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

**by** (*simp add: Guard-tri-design rea-post-RHS-design usubst unrest R2c-rea-not R2c-and R2c-rea-impl*)

$R2c\text{-}preR$   $R2c\text{-}postR$   $R2c\text{-}tr'\text{-}minus\text{-}tr$   $R2c\text{-}lift\text{-}state\text{-}pre$   $R2c\text{-}condR$   $R1\text{-}rea\text{-}impl$   $R1\text{-}extend\text{-}conj'$   
 $R1\text{-}post\text{-}SRD$   $closure$   $assms$ )  
**also have** ... =  $([b]_{S<} \wedge (pre_R P \Rightarrow_r post_R P))$   
**by** (*rel-auto*)  
**also have** ... =  $([b]_{S<} \wedge post_R P)$   
**by** (*simp add: SRD-post-under-pre add: unrest closure assms*)  
**also have** ... =  $([b]_{S<} \wedge post_R P)$   
**by** (*metis CSP-Guard R1-extend-conj R1-post-SRD calculation rea-st-cond-def*)  
**finally show** *?thesis* .  
**qed**

**lemma** *CSP3-Guard* [*closure*]:  
**assumes**  $P$  is *CSP*  $P$  is *CSP3*  
**shows**  $b \ \&_C \ P$  is *CSP3*  
**proof** –  
**from** *assms* **have**  $1:\$ref \ \# \ P\llbracket false/\$wait \rrbracket$   
**by** (*simp add: CSP-Guard CSP3-iff*)  
**hence**  $\$ref \ \# \ pre_R (P\llbracket 0/\$tr \rrbracket) \ \$ref \ \# \ pre_R P \ \$ref \ \# \ cmt_R P$   
**by** (*pred-blast*)+  
**hence**  $\$ref \ \# \ (b \ \&_C \ P)\llbracket false/\$wait \rrbracket$   
**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**  $b \ \&_C \ P$  is *CSP4*  
**proof** (*rule CSP4-tri-intro[OF CSP-Guard]*)  
**show**  $(\neg_r pre_R (b \ \&_C \ P)) \ ; \ R1 \ true = (\neg_r pre_R (b \ \&_C \ P))$   
**proof** –  
**have**  $a:(\neg_r pre_R P) \ ; \ R1 \ true = (\neg_r pre_R P)$   
**by** (*simp add: CSP4-neg-pre-unit assms closure*)  
**have**  $(\neg_r ([b]_{S<} \Rightarrow_r pre_R P)) \ ; \ R1 \ true = (\neg_r ([b]_{S<} \Rightarrow_r pre_R P))$   
**proof** –  
**have**  $1:(\neg_r ([b]_{S<} \Rightarrow_r pre_R P)) = ([b]_{S<} \wedge (\neg_r pre_R P))$   
**by** (*rel-auto*)  
**also have**  $2:\dots = ([b]_{S<} \wedge ((\neg_r pre_R P) \ ; \ R1 \ true))$   
**by** (*simp add: a*)  
**also have**  $3:\dots = (\neg_r ([b]_{S<} \Rightarrow_r pre_R P)) \ ; \ R1 \ true$   
**by** (*rel-auto*)  
**finally show** *?thesis* ..  
**qed**  
**thus** *?thesis*  
**by** (*simp add: preR-Guard periR-Guard NSRD-CSP4-intro closure assms unrest*)  
**qed**  
**show**  $\$st' \ \# \ peri_R (b \ \&_C \ P)$   
**by** (*simp add: preR-Guard periR-Guard NSRD-CSP4-intro closure assms unrest*)  
**show**  $\$ref' \ \# \ post_R (b \ \&_C \ P)$   
**by** (*simp add: preR-Guard postR-Guard NSRD-CSP4-intro closure assms unrest*)  
**qed**

**lemma** *NCSP-Guard* [*closure*]:  
**assumes**  $P$  is *NCSP*

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

**lemma** Productive-Guard [closure]:  
 assumes  $P$  is CSP  $P$  is Productive  $\$wait' \# pre_R(P)$   
 shows  $b \&_C P$  is Productive  
**proof** –  
 have  $b \&_C P = b \&_C \mathbf{R}_s(pre_R(P) \vdash peri_R(P) \diamond (post_R(P) \wedge \$tr <_u \$tr'))$   
 by (metis Healthy-def Productive-form *assms*(1) *assms*(2))  
 also have ... =  
 $\mathbf{R}_s((\lceil b \rceil_{S<} \Rightarrow_r pre_R P) \vdash$   
 $((pre_R P \Rightarrow_r peri_R P) \triangleleft \lceil b \rceil_{S<} \triangleright (\$tr' =_u \$tr)) \diamond (\lceil b \rceil_{S<} \wedge (pre_R P \Rightarrow_r post_R P \wedge \$tr' >_u$   
 $\$tr)))$   
 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( $b \&_C P$ )  
 by (simp add: Productive-def Guard-tri-design RHS-tri-design-par unrest)  
 finally show ?thesis  
 by (simp add: Healthy-def')  
**qed**

**lemma** Guard-refines-sinv:  
 assumes  $P$  is NCSP  $sinv_R(b) \sqsubseteq P$   
 shows  $sinv_R(b) \sqsubseteq g \&_C P$   
**proof** –  
 from *assms*  
 have  $\mathbf{R}_s(\lceil b \rceil_{S<} \vdash R1 \text{ true} \diamond \lceil b \rceil_{S>}) \sqsubseteq \mathbf{R}_s(pre_R(P) \vdash peri_R(P) \diamond post_R(P))$   
 by (simp add: rdes-def NCSP-implies-CSP SRD-reactive-tri-design)  
 thus ?thesis  
 apply (simp add: RHS-tri-design-refine' closure unrest *assms*)  
 apply (safe)  
 apply (rdes-refine cls: *assms*(1))  
 done  
**qed**

## 8.12 Basic events

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

**definition**  $DoCSP :: (' \varphi, ' \sigma) \text{ uexpr} \Rightarrow (' \sigma, ' \varphi) \text{ action}$  ( $do_C$ ) **where**  
 $[upred-defs]: DoCSP a = \mathbf{R}_s(true \vdash do_u a)$

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

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

**lemma** *DoCSP-alt-def*:  $do_C(a) = R3h(CSP1(\$ok' \wedge do_u(a)))$   
**apply** (*simp add: DoCSP-def RHS-def design-def impl-alt-def R1-R3h-commute R2c-R3h-commute R2c-disj R2c-not R2c-ok R2c-ok' R2c-and R2c-DoAct R1-disj R1-extend-conj' R1-DoAct*)  
**apply** (*rel-auto*)  
**done**

**lemma** *DoAct-unrests* [*unrest*]:  
 $\$ok \# do_u(a) \ \$wait \# do_u(a)$   
**by** (*pred-auto*)**+**

**lemma** *DoCSP-RHS-tri* [*rdes-def*]:  
 $do_C(a) = \mathbf{R}_s(true_r \vdash (\mathcal{E}(true, \llbracket \gg, \{a\}_u) \diamond \Phi(true, id_s, U([a])))$   
**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*]:  $peri_R(do_C(a)) = \mathcal{E}(true, \llbracket \gg, \{a\}_u)$   
**by** (*rel-auto*)

**lemma** *postR-DoCSP* [*rdes*]:  $post_R(do_C(a)) = \Phi(true, id_s, U([a]))$   
**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*]:  
 $(do_C a :: ('\sigma, '\psi) \text{ action})$  is *Productive*

**proof** –

**have**  $((\Phi(true, id_s, U([a])) \wedge \$tr' >_u \$tr) :: ('\sigma, '\psi) \text{ action})$   
 $= (\Phi(true, id_s, U([a])))$

**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  $U(\$tr' = \$tr @ [[a]_{S<}] \wedge \$st' = \$st) ;; P = (\exists \$ref \cdot P[U(\$tr @ [[a]_{S<}])/\$tr])$

using *assms*

by (rel-auto, meson)

**lemma** *wp-rea-DoCSP*:

assumes  $P$  is NCSP

shows  $U(\$tr' = \$tr @ [[a]_{S<}] \wedge \$st' = \$st) wp_r pre_R P =$

$(\neg_r (\neg_r pre_R P)[U(\$tr @ [[a]_{S<}])/\$tr])$

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  $U(\$tr' = \$tr @ [[a]_{S<}] \wedge \$st' = \$st) wp_r pre_R P =$

$U(\$tr' \geq \$tr @ [[a]_{S<}] \Rightarrow_r (pre_R P)[\$tr @ [[a]_{S<}]/\$tr])$

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

**lemma** *DoCSP-refine-sinv*:  $sinv_R(b) \sqsubseteq do_C(a)$

by (rdes-refine)

### 8.13 Event prefix

**definition** *PrefixCSP* ::

$(''\varphi, '\sigma)$  uexpr  $\Rightarrow$

$(''\sigma, '\varphi)$  action  $\Rightarrow$

$(''\sigma, '\varphi)$  action  $(- \rightarrow_C - [81, 80] 80)$  **where**

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

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

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

by (simp add: PrefixCSP-def closure)

**lemma** *CSP3-PrefixCSP* [closure]:

*PrefixCSP*  $a P$  is CSP3

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

**lemma** *CSP4-PrefixCSP* [closure]:

assumes  $P$  is CSP  $P$  is CSP4

shows *PrefixCSP*  $a P$  is CSP4

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

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

assumes  $P$  is NCSP

shows *PrefixCSP*  $a P$  is NCSP

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

**lemma** *Productive-PrefixCSP* [closure]:  $P$  is NCSP  $\implies$  PrefixCSP  $a$   $P$  is Productive  
**by** (simp add: Healthy-if NCSP-DoCSP NCSP-implies-NSRD NSRD-is-SRD PrefixCSP-def Productive-DoCSP Productive-seq-1)

**lemma** *PCSP-PrefixCSP* [closure]:  $P$  is NCSP  $\implies$  PrefixCSP  $a$   $P$  is PCSP  
**by** (simp add: Healthy-comp NCSP-PrefixCSP Productive-PrefixCSP)

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

**proof** –

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

**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*:

$R1 \ (R2s \ (U(\$tr' = \$tr @ \llbracket a \rrbracket_{S<})) \wedge \llbracket II \rrbracket_R)) = (U(\$tr' = \$tr @ \llbracket a \rrbracket_{S<})) \wedge \llbracket II \rrbracket_R)$

**by** (rel-auto)

**lemma** *PrefixCSP-RHS-tri-lemma2*:

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

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

**shows**  $(U(\$tr' = \$tr @ \llbracket a \rrbracket_{S<})) \wedge \$st' = \$st \wedge \neg \$wait' ;; P = (\exists \$ref. P \llbracket U(\$tr @ \llbracket a \rrbracket_{S<})/\$tr \rrbracket)$

**using** assms

**by** (rel-auto, meson, fastforce)

**lemma** *tr-extend-seqr*:

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

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

**shows**  $U(\$tr' = \$tr @ \llbracket a \rrbracket_{S<})) \wedge \$st' = \$st ;; P = P \llbracket U(\$tr @ \llbracket a \rrbracket_{S<})/\$tr \rrbracket$

**using** assms **by** (simp add: wp-rea-DoCSP-lemma assms unrest ex-unrest)

**lemma** *trace-ext-R1-closed* [closure]:  $P$  is R1  $\implies P \llbracket \$tr \hat{\ }_u e / \$tr \rrbracket$  is R1

**by** (rel-blast)

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

**assumes**  $P$  is NCSP

**shows**  $\text{pre}_R(\text{PrefixCSP } a \ P) = (\Phi(\text{true}, id_s, U(\llbracket a \rrbracket)) \ \text{wp}_r \ \text{pre}_R \ P)$

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

**lemma** *PrefixCSP-RHS-tri*:

**assumes**  $P$  is NCSP

**shows**  $\text{PrefixCSP } a \ P = \mathbf{R}_s \ (\Phi(\text{true}, id_s, U(\llbracket a \rrbracket)) \ \text{wp}_r \ \text{pre}_R \ P \vdash (\mathcal{E}(\text{true}, \llbracket \cdot \rrbracket, \{a\}_u) \vee \Phi(\text{true}, id_s, U(\llbracket a \rrbracket))$   
 $;; \text{peri}_R \ P) \diamond \Phi(\text{true}, id_s, U(\llbracket a \rrbracket)) ;; \text{post}_R \ P)$

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

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

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

**assumes**  $P$  is CRC  $Q$  is CRR  $R$  is CRR

$\$st' \# Q \$ref' \# R$

**shows**  $PrefixCSP\ a\ (\mathbf{R}_s(P \vdash Q \diamond R)) =$

$\mathbf{R}_s(\Phi(true, id_s, U([a]))\ wp_r\ P \vdash (\mathcal{E}(true, \llbracket \cdot \rrbracket, \{a\}_u) \vee \Phi(true, id_s, U([a])) \;;\ Q) \diamond \Phi(true, id_s, U([a])) \;;\ R)$

**by** (*simp add: PrefixCSP-def Healthy-if assms closure, rdes-simp cls: assms*)

## 8.14 Guarded external choice

**abbreviation** *GuardedChoiceCSP* ::  $'\vartheta$  set  $\Rightarrow (' \vartheta \Rightarrow (' \sigma, ' \vartheta)$  action)  $\Rightarrow (' \sigma, ' \vartheta)$  action **where**

*GuardedChoiceCSP*  $A\ P \equiv (\Box x \in A \cdot PrefixCSP\ \llbracket x \rrbracket\ (P(x)))$

**syntax**

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

**translations**

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

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

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

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

$\mathbf{R}_s((\Box x \in A \cdot \Phi(true, id_s, \llbracket x \rrbracket)\ wp_r\ pre_R\ (P\ x)) \vdash ((\Box x \in A \cdot \mathcal{E}(true, \llbracket \cdot \rrbracket, \{\llbracket x \rrbracket\}_u)) \triangleleft \$tr' =_u \$tr \triangleright (\Box x \in A \cdot \Phi(true, id_s, \llbracket x \rrbracket) \;;\ peri_R\ (P\ x))) \diamond$

$(\Box x \in A \cdot \Phi(true, id_s, \llbracket x \rrbracket) \;;\ post_R\ (P\ x)))$

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

## 8.15 Input prefix

**definition** *InputCSP* ::

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

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

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

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

**definition** *do<sub>I</sub>* ::

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

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

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

$(' \sigma, ' \vartheta)$  action **where**

*do<sub>I</sub>*  $c\ x\ P =$

$(\$tr' =_u \$tr \wedge \{e : \llbracket e \rrbracket \mid P(e) \cdot (c \cdot \llbracket e \rrbracket)_u\} \cap_u \$ref' =_u \{\}_u)$

$\triangleleft \$wait' \triangleright$

$((\$tr' - \$tr) \in_u \{e : \llbracket e \rrbracket \mid P(e) \cdot U([\llbracket c \cdot \llbracket e \rrbracket \rrbracket])\} \wedge (c \cdot \$x')_u =_u last_u(\$tr'))$

**lemma** *InputCSP-CSP* [closure]: *InputCSP*  $c\ A\ P$  is CSP

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

**lemma** *InputCSP-NCSP* [closure]:  $\llbracket \bigwedge v. P(v) \text{ is NCSP} \rrbracket \Rightarrow InputCSP\ c\ A\ P \text{ is NCSP}$

**apply** (*simp add: InputCSP-def*)

**apply** (*rule NCSP-ExtChoice*)

**apply** (*simp add: NCSP-Guard NCSP-PrefixCSP image-Collect-subsetI top-set-def*)

**done**

**lemma** *InputVarCSP-NCSP [closure]: InputVarCSP c x A is NCSP*  
**by** (*simp add: AssignsCSP-NCSP InputCSP-NCSP InputVarCSP-def*)

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

**lemma** *Productive-InputVarCSP [closure]: InputVarCSP c x A is Productive*  
**by** (*simp add: InputVarCSP-def closure*)

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

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

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

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

## 8.16 Renaming

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

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

**lemma** *RenameCSP-pre-CRC-closed:*  
**assumes**  $P \text{ is CRR}$



**shows**  $\neg_r (\neg_r P)(\lfloor f \rfloor)_c \;; R1 \text{ true is CRC}$   
**apply** (rule CRC-intro")  
**apply** (simp add: unrest closure assms)  
**apply** (simp add: Healthy-def, simp add: RC1-def rpred closure CRC-idem assms segr-assoc)  
**done**

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

**assumes** *P is NCSP*

**shows**  $P(\lfloor f \rfloor)_C \text{ is NCSP}$

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

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

$\text{false}(\lfloor f \rfloor)_c = \text{false}$

**by** (rel-auto)

**lemma** *umap-nil* [simp]:  $\text{map}_u f \ll \square \gg = \ll \square \gg$

**by** (rel-auto)

**lemma** *rename-Skip*:  $\text{Skip}(\lfloor f \rfloor)_C = \text{Skip}$

**by** (rdes-eq)

**lemma** *rename-Chaos*:  $\text{Chaos}(\lfloor f \rfloor)_C = \text{Chaos}$

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

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

**by** (rdes-eq)

**lemma** *rename-DoCSP*:  $(\text{do}_C(a))(\lfloor f \rfloor)_C = \text{do}_C(\ll f \gg(a)_a)$

**by** (rdes-eq)

## 8.17 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*:  $\langle \text{id}_s \rangle_C = \text{Skip}$

**by** (rel-auto)

**lemma** *Guard-comp*:

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

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

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

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

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

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

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

**lemma** *Guard-conditional*:

**assumes** *P is NCSP*

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

**by** (rdes-eq cls: assms)

**lemma** *Guard-expansion:*

**assumes**  $P$  is NCSP  
**shows**  $(g_1 \vee g_2) \&_C P = (g_1 \&_C P) \sqcap (g_2 \&_C P)$   
**apply** (*rdes-eq-split cls: assms*)  
**apply** (*rel-simp'*, *fastforce simp add: dual-order.order-iff-strict*)  
**apply** (*rel-simp'*, *simp add: dual-order.order-iff-strict, fastforce*)  
**apply** (*rel-simp'*, *simp add: dual-order.order-iff-strict, fastforce*)  
**done**

**lemma** *Conditional-as-Guard:*

**assumes**  $P$  is NCSP  $Q$  is NCSP  
**shows**  $P \triangleleft b \triangleright_R Q = b \&_C P \sqcap (\neg b) \&_C Q$   
**by** (*rdes-eq' cls: assms; simp add: le-less*)

**lemma** *PrefixCSP-dist:*

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

**lemma** *PrefixCSP-extChoice-dist:*

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

**lemma** *GuardedChoiceCSP-dist:*

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

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

**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**  
 $\bigwedge i. i \in A \implies P(i)$  is NCSP  $Q$  is NCSP  
 $\bigwedge i j. \llbracket i \in A; j \in A; i \neq j \rrbracket \implies (g i \wedge g j) = \text{false}$   
**shows**  $(\text{if}_R i \in A \cdot g(i) \rightarrow P(i) \text{ else } Q \text{ fi}) =$   
 $(\square i \in A \cdot g(i) \&_C P(i)) \sqcap (\bigwedge i \in A \cdot \neg g(i)) \&_C Q$

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

**case** *True*

**then show** *?thesis* **by** (*simp add: ExtChoice-empty extChoice-Stop closure assms*)

**next**

**case** *False*

```

show ?thesis

proof –
  have 1: ( $\bigcap i \in A \cdot g\ i \rightarrow_R P\ i$ ) = ( $\bigcap i \in A \cdot g\ i \rightarrow_R \mathbf{R}_s(\text{pre}_R(P\ i) \vdash \text{peri}_R(P\ i) \diamond \text{post}_R(P\ i))$ )
    by (rule UINF-cong, simp add: NCSP-implies-CSP SRD-reactive-tri-design assms(1))
  have 2: ( $\bigcap i \in A \cdot g(i) \&_C P(i)$ ) = ( $\bigcap i \in A \cdot g(i) \&_C \mathbf{R}_s(\text{pre}_R(P\ i) \vdash \text{peri}_R(P\ i) \diamond \text{post}_R(P\ i))$ )
    by (rule ExtChoice-cong, simp add: NCSP-implies-NSRD NSRD-is-SRD SRD-reactive-tri-design
    assms(1))
  from assms(3) show ?thesis
    by (simp add: AlternateR-def 1 2)
      (rdes-eq' cls: assms(1–2)_simps: False cong: UINF-cong USUP-cong ExtChoice-cong)
qed
qed

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

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

find-theorems R4

end

```

## 9 Recursion in Stateful-Failures

```

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

```

### 9.1 Fixed-points

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

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

```

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

```

```

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

```

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

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

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

**lemma** *mu-csp-basic-refine*:

**assumes**

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

$peri_R P \sqsubseteq peri_R Q$

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

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

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

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

**proof** (rule *UINF-refines'*)

**fix** *i*

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

**proof** (induct *i*)

**case** 0

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

**next**

**case** (Suc *i*)

**then show** ?case

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

**qed**

**qed**

**qed**

**lemma** *CRD-mu-basic-refine*:

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

**assumes**

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

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

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

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

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

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

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

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

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

**qed**

## 9.2 Example action expansion

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

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

**lemma** *periR-mu-example1* [rdes]:

$peri_R(\mu X \cdot \langle\langle a \rangle\rangle \rightarrow_C X) = (\bigcap i \cdot \mathcal{E}(true, iter[i](U([\langle\langle a \rangle\rangle])), \{\langle\langle a \rangle\rangle\}_u))$

**by** (simp add: mu-example1 rdes rpred closure unrest wp seq-UINF-distr alpha usubst)

**lemma** *postR-mu-example1* [rdes]:

$post_R(\mu X \cdot \langle\langle a \rangle\rangle \rightarrow_C X) = false$

**by** (simp add: mu-example1 rdes closure unrest wp)

end

## 10 Linking to the Failures-Divergences Model

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

### 10.1 Failures-Divergences Semantics

The following functions play a similar role to those in Roscoe's CSP semantics, and are calculated from the Circus reactive design semantics. A major difference is that these three functions account for state. Each divergence, trace, and failure is subject to an initial state. Moreover, the traces are terminating traces, and therefore also provide a final state following the given interaction. A more subtle difference from the Roscoe semantics is that the set of traces do not include the divergences. The same semantic information is present, but we construct a direct analogy with the pre-, peri- and postconditions of our reactive designs.

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

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

**lemma** *trace-divergence-disj*:

```
  assumes P is NCSP (t, s') ∈ tr[P]s t ∈ dv[P]s
  shows False
  using assms(2,3)
  by (simp add: traces-def divergences-def, rdes-simp cls:assms, rel-auto)
```

**lemma** *preR-refine-divergences*:

```
  assumes P is NCSP Q is NCSP ∧ s. dv[P]s ⊆ dv[Q]s
  shows pre_R(P) ⊆ pre_R(Q)
```

**proof** (rule CRR-refine-impl-prop, simp-all add: assms closure usubst unrest)

```
  fix t s
```

```
  assume a: '[$st ↦_s <s>, $tr ↦_s <[]>, $tr' ↦_s <t>] † pre_R Q'
```

```
  with a show '[$st ↦_s <s>, $tr ↦_s <[]>, $tr' ↦_s <t>] † pre_R P'
```

```
  proof (rule-tac ccontr)
```

```
    from assms(3)[of s] have b: t ∈ dv[P]s ⇒ t ∈ dv[Q]s
```

```
    by (auto)
```

```
    assume ¬ '[$st ↦_s <s>, $tr ↦_s <[]>, $tr' ↦_s <t>] † pre_R P'
```

```
    hence ¬ '[$st ↦_s <s>, $tr ↦_s <[]>, $tr' ↦_s <t>] † CRC(pre_R P)'
```

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

```
    hence '[$st ↦_s <s>, $tr ↦_s <[]>, $tr' ↦_s <t>] † (¬_r CRC(pre_R P))'
```

```
    by (rel-auto)
```

```
    hence '[$st ↦_s <s>, $tr ↦_s <[]>, $tr' ↦_s <t>] † (¬_r pre_R P)'
```

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

```
    with a b show False
```

```
    by (rel-auto)
```

```
  qed
```

qed

**lemma** *preR-eq-divergences*:

**assumes**  $P$  is NCSP  $Q$  is NCSP  $\wedge s. dv[P]s = dv[Q]s$

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

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

**lemma** *periR-refine-failures*:

**assumes**  $P$  is NCSP  $Q$  is NCSP  $\wedge s. fl[Q]s \subseteq fl[P]s$

**shows**  $(pre_R(P) \wedge peri_R(P)) \sqsubseteq (pre_R(Q) \wedge peri_R(Q))$

**proof** (*rule* *CRR-refine-impl-prop*, *simp-all* *add: assms closure unrest subst-unrest-3*)

**fix**  $t s r'$

**assume**  $a: '[\$ref' \mapsto_s \ll r' \gg, \$st \mapsto_s \ll s \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t \gg] \dagger (pre_R Q \wedge peri_R Q)'$

**from** *assms*(3)[*of*  $s$ ] **have**  $b: (t, r') \in fl[Q]s \implies (t, r') \in fl[P]s$

**by** (*auto*)

**with**  $a$  **show**  $'[\$ref' \mapsto_s \ll r' \gg, \$st \mapsto_s \ll s \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t \gg] \dagger (pre_R P \wedge peri_R P)'$

**by** (*simp* *add: failures-def*)

**qed**

**lemma** *periR-eq-failures*:

**assumes**  $P$  is NCSP  $Q$  is NCSP  $\wedge s. fl[P]s = fl[Q]s$

**shows**  $(pre_R(P) \wedge peri_R(P)) = (pre_R(Q) \wedge peri_R(Q))$

**by** (*metis* (*full-types*) *assms* *dual-order.antisym* *order-refl* *periR-refine-failures*)

**lemma** *postR-refine-traces*:

**assumes**  $P$  is NCSP  $Q$  is NCSP  $\wedge s. tr[Q]s \subseteq tr[P]s$

**shows**  $(pre_R(P) \wedge post_R(P)) \sqsubseteq (pre_R(Q) \wedge post_R(Q))$

**proof** (*rule* *CRR-refine-impl-prop*, *simp-all* *add: assms closure unrest subst-unrest-5*)

**fix**  $t s s'$

**assume**  $a: '[$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \ll [] \gg, \$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 \ll [] \gg, \$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*:  
 $tr\llbracket Skip \rrbracket s = \{([], s)\}$   
 by (simp add: traces-def rdes alpha closure, rel-simp)

**lemma** *failures-Skip*:  
 $fl\llbracket Skip \rrbracket s = \{\}$   
 by (simp add: failures-def, rdes-calc)

**lemma** *divergences-Skip*:  
 $dv\llbracket Skip \rrbracket s = \{\}$   
 by (simp add: divergences-def, rdes-calc)

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

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

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

**lemma** *traces-AssignsCSP*:  
 $tr\llbracket \langle \sigma \rangle_C \rrbracket s = \{([], \llbracket \sigma \rrbracket_e s)\}$   
 by (simp add: traces-def rdes closure usubst alpha, rel-auto)

**lemma** *failures-AssignsCSP*:  
 $fl\llbracket \langle \sigma \rangle_C \rrbracket s = \{\}$   
 by (simp add: failures-def, rdes-calc)

**lemma** *divergences-AssignsCSP*:  
 $dv\llbracket \langle \sigma \rangle_C \rrbracket s = \{\}$   
 by (simp add: divergences-def, rdes-calc)

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

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

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

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

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

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

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

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

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

**lemma** *traces-do*:  $tr\llbracket do_C(e) \rrbracket s = \{(\llbracket e \rrbracket_e s, s)\}$   
**by** (*rdes-simp, simp add: traces-def rdes closure rpred, rel-auto*)

**lemma** *failures-do*:  $fl\llbracket do_C(e) \rrbracket s = \{([], E) \mid E. \llbracket e \rrbracket_e s \notin E\}$   
**by** (*rdes-simp, simp add: failures-def rdes closure rpred usubst, rel-auto*)

**lemma** *divergences-do*:  $dv\llbracket do_C(e) \rrbracket s = \{\}$   
**by** (*rel-auto*)

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

**lemma** *traces-seq*:  
**fixes**  $P :: ('s, 'e) \text{ action}$   
**assumes**  $P$  is NCSP  $Q$  is NCSP  
**shows**  $tr\llbracket P ;; Q \rrbracket s =$   
 $\{(t_1 @ t_2, s') \mid t_1 \ t_2 \ s_0 \ s'. (t_1, s_0) \in tr\llbracket P \rrbracket s \wedge (t_2, s') \in tr\llbracket Q \rrbracket s_0$   
 $\wedge (t_1 @ t_2) \notin dv\llbracket P \rrbracket s$   
 $\wedge (\forall (t, s_1) \in tr\llbracket P \rrbracket s. t \leq t_1 @ t_2 \longrightarrow (t_1 @ t_2) - t \notin dv\llbracket Q \rrbracket s_1) \}$   
**(is ?lhs = ?rhs)**

**proof**  
**show**  $?lhs \subseteq ?rhs$   
**proof** (*rdes-expand cls: assms, simp add: traces-def divergences-def rdes closure assms rdes-def unrest*  
*rpred usubst, auto*)  
**fix**  $t :: 'e \text{ list}$  **and**  $s' :: 's$   
**let**  $?s = [\$st \mapsto_s \llbracket s \rrbracket, \$st' \mapsto_s \llbracket s' \rrbracket, \$tr \mapsto_s \llbracket \rrbracket, \$tr' \mapsto_s \llbracket t \rrbracket]$   
**assume**



$a1: '?\sigma \uparrow (post_R P ;; post_R Q)'$  and  
 $a2: '[\$st \mapsto_s \langle s \rangle, \$tr \mapsto_s \langle [] \rangle, \$tr' \mapsto_s \langle t \rangle] \uparrow pre_R P'$  and  
 $a3: '[\$st \mapsto_s \langle s \rangle, \$tr \mapsto_s \langle [] \rangle, \$tr' \mapsto_s \langle t \rangle] \uparrow (post_R P \text{ wp}_r pre_R Q)'$   
**from**  $a1$  **have**  $'?\sigma \uparrow (\exists tr_0 \cdot ((post_R P) \llbracket \langle tr_0 \rangle / \$tr' \rrbracket ;; (post_R Q) \llbracket \langle tr_0 \rangle / \$tr \rrbracket) \wedge \langle tr_0 \rangle \leq_u \$tr')'$   
**by** (*simp add: R2-tr-middle assms closure*)  
**then obtain**  $tr_0$  **where**  $p1: '?\sigma \uparrow ((post_R P) \llbracket \langle tr_0 \rangle / \$tr' \rrbracket ;; (post_R Q) \llbracket \langle tr_0 \rangle / \$tr \rrbracket)'$  **and**  $tr0: tr_0 \leq t$   
**apply** (*simp add: usubst*)  
**apply** (*erule taut-shEx-elim*)  
**apply** (*simp add: unrest-all-circus-vars-st-st' closure unrest assms*)  
**apply** (*rel-auto*)  
**done**  
**from**  $p1$  **have**  $'?\sigma \uparrow (\exists st_0 \cdot (post_R P) \llbracket \langle tr_0 \rangle / \$tr' \rrbracket \llbracket \langle st_0 \rangle / \$st' \rrbracket ;; (post_R Q) \llbracket \langle tr_0 \rangle / \$tr \rrbracket \llbracket \langle st_0 \rangle / \$st \rrbracket)'$   
**by** (*simp add: seqr-middle[of st, THEN sym]*)  
**then obtain**  $s_0$  **where**  $'?\sigma \uparrow ((post_R P) \llbracket \langle s_0 \rangle, \langle tr_0 \rangle / \$st', \$tr' \rrbracket ;; (post_R Q) \llbracket \langle s_0 \rangle, \langle tr_0 \rangle / \$st, \$tr \rrbracket)'$   
**apply** (*simp add: usubst*)  
**apply** (*erule taut-shEx-elim*)  
**apply** (*simp add: unrest-all-circus-vars-st-st' closure unrest assms*)  
**apply** (*rel-auto*)  
**done**  
**hence**  $'(([\$st \mapsto_s \langle s \rangle, \$st' \mapsto_s \langle s_0 \rangle, \$tr \mapsto_s \langle [] \rangle, \$tr' \mapsto_s \langle tr_0 \rangle] \uparrow post_R P) ;; ([\$st \mapsto_s \langle s_0 \rangle, \$st' \mapsto_s \langle s' \rangle, \$tr \mapsto_s \langle tr_0 \rangle, \$tr' \mapsto_s \langle t \rangle] \uparrow post_R Q))'$   
**by** (*rel-auto*)  
**hence**  $'(([\$st \mapsto_s \langle s \rangle, \$st' \mapsto_s \langle s_0 \rangle, \$tr \mapsto_s \langle [] \rangle, \$tr' \mapsto_s \langle tr_0 \rangle] \uparrow post_R P) \wedge ([\$st \mapsto_s \langle s_0 \rangle, \$st' \mapsto_s \langle s' \rangle, \$tr \mapsto_s \langle tr_0 \rangle, \$tr' \mapsto_s \langle t \rangle] \uparrow post_R Q))'$   
**by** (*simp add: seqr-to-conj unrest-any-circus-var assms closure unrest*)  
**hence**  $postP: '([\$st \mapsto_s \langle s \rangle, \$st' \mapsto_s \langle s_0 \rangle, \$tr \mapsto_s \langle [] \rangle, \$tr' \mapsto_s \langle tr_0 \rangle] \uparrow post_R P)'$  **and**  
 $postQ': '([\$st \mapsto_s \langle s_0 \rangle, \$st' \mapsto_s \langle s' \rangle, \$tr \mapsto_s \langle tr_0 \rangle, \$tr' \mapsto_s \langle t \rangle] \uparrow post_R Q)'$   
**by** (*rel-auto*)  
**from**  $postQ'$  **have**  $'[\$st \mapsto_s \langle s_0 \rangle, \$st' \mapsto_s \langle s' \rangle] \uparrow [\$tr \mapsto_s \langle tr_0 \rangle, \$tr' \mapsto_s \langle tr_0 \rangle + (\langle t \rangle - \langle tr_0 \rangle)] \uparrow post_R Q'$   
**using**  $tr0$  **by** (*rel-auto*)  
**hence**  $'[\$st \mapsto_s \langle s_0 \rangle, \$st' \mapsto_s \langle s' \rangle] \uparrow [\$tr \mapsto_s 0, \$tr' \mapsto_s \langle t \rangle - \langle tr_0 \rangle] \uparrow post_R Q'$   
**by** (*simp add: R2-subst-tr closure assms*)  
**hence**  $postQ: '[\$st \mapsto_s \langle s_0 \rangle, \$st' \mapsto_s \langle s' \rangle, \$tr \mapsto_s \langle [] \rangle, \$tr' \mapsto_s \langle t - tr_0 \rangle] \uparrow post_R Q'$   
**by** (*rel-auto*)  
**have**  $preP: '[\$st \mapsto_s \langle s \rangle, \$tr \mapsto_s \langle [] \rangle, \$tr' \mapsto_s \langle tr_0 \rangle] \uparrow pre_R P'$   
**proof** –  
**have**  $(pre_R P) \llbracket 0, \langle tr_0 \rangle / \$tr, \$tr' \rrbracket \sqsubseteq (pre_R P) \llbracket 0, \langle t \rangle / \$tr, \$tr' \rrbracket$   
**by** (*simp add: RC-prefix-refine closure assms tr0*)  
**hence**  $[\$st \mapsto_s \langle s \rangle, \$tr \mapsto_s \langle [] \rangle, \$tr' \mapsto_s \langle tr_0 \rangle] \uparrow pre_R P \sqsubseteq [\$st \mapsto_s \langle s \rangle, \$tr \mapsto_s \langle [] \rangle, \$tr' \mapsto_s \langle t \rangle] \uparrow pre_R P$   
**by** (*rel-auto*)  
**thus** *?thesis*  
**by** (*simp add: taut-refine-impl a2*)  
**qed**  
  
**have**  $preQ: '[\$st \mapsto_s \langle s_0 \rangle, \$tr \mapsto_s \langle [] \rangle, \$tr' \mapsto_s \langle t - tr_0 \rangle] \uparrow pre_R Q'$   
**proof** –  
**from**  $postP$   $a3$  **have**  $'[\$st \mapsto_s \langle s_0 \rangle, \$tr \mapsto_s \langle tr_0 \rangle, \$tr' \mapsto_s \langle t \rangle] \uparrow pre_R Q'$   
**apply** (*simp add: wp-rea-def*)  
**apply** (*rel-auto*)  
**using**  $tr0$  **apply** *blast+*  
**done**  
**hence**  $'[\$st \mapsto_s \langle s_0 \rangle] \uparrow [\$tr \mapsto_s \langle tr_0 \rangle, \$tr' \mapsto_s \langle tr_0 \rangle + (\langle t \rangle - \langle tr_0 \rangle)] \uparrow pre_R Q'$

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:  $\neg ['\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t \gg] \dagger (\neg_r pre_R P)'$   
 by (rel-auto)

have t-minus-tr0:  $tr_0 @ (t - tr_0) = t$   
 using append-minus tr0 by blast

from a3  
 have wpr:  $\bigwedge t_0 s_1.$   
 $['\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t_0 \gg] \dagger pre_R P' \implies$   
 $['\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t_0 \gg] \dagger post_R P' \implies$   
 $t_0 \leq t \implies ['\$st \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t - t_0 \gg] \dagger (\neg_r pre_R Q)' \implies False$

proof –  
 fix  $t_0 s_1$   
 assume b:  
 $['\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t_0 \gg] \dagger pre_R P'$   
 $['\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t_0 \gg] \dagger post_R P'$   
 $t_0 \leq t$   
 $['\$st \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t - t_0 \gg] \dagger (\neg_r pre_R Q)'$

from a3 have c:  $\forall (s_0, t_0) \cdot \ll t_0 \gg \leq_u \ll t \gg$   
 $\wedge [\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t_0 \gg] \dagger post_R P$   
 $\implies [\$st \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t \gg - \ll t_0 \gg] \dagger pre_R Q'$   
 by (simp add: wp-rea-circus-form-alt[of post\_R P pre\_R Q] closure assms unrest usubst)  
 (rel-simp)

from c b(2-4) show False  
 by (rel-auto)

qed

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

apply (rule-tac x=tr0 in exI)  
 apply (rule-tac x=(t - tr0) in exI)  
 apply (auto)  
 using tr0 apply auto[1]  
 apply (rule-tac x=s0 in exI)  
 apply (auto intro:wpr simp add: taut-conj preP preQ postP postQ ndiv wpr t-minus-tr0)  
 done

qed

show  $?rhs \subseteq ?lhs$

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

fix  $t_1 t_2 :: 'e \text{ list}$  and  $s_0 s' :: 's$

assume

a1:  $\neg ' \$st \mapsto_s \ll s \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t_1 @ t_2 \gg ] \dagger (\neg_r \text{pre}_R P) ' \text{ and}$

a2:  $' \$st \mapsto_s \ll s \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t_1 \gg ] \dagger \text{pre}_R P ' \text{ and}$

a3:  $' \$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t_1 \gg ] \dagger \text{post}_R P ' \text{ and}$

a4:  $' \$st \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t_2 \gg ] \dagger \text{pre}_R Q ' \text{ and}$

a5:  $' \$st \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t_2 \gg ] \dagger \text{post}_R Q ' \text{ and}$

a6:  $\forall t s_1. ' \$st \mapsto_s \ll s \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t \gg ] \dagger \text{pre}_R P \wedge$

$[ \$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t \gg ] \dagger \text{post}_R P ' \longrightarrow$

$t \leq t_1 @ t_2 \longrightarrow \neg ' \$st \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll (t_1 @ t_2) - t \gg ] \dagger (\neg_r \text{pre}_R Q) ' \text{ and}$

from a1 have  $\text{pre}P: ' \$st \mapsto_s \ll s \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t_1 @ t_2 \gg ] \dagger (\text{pre}_R P) ' \text{ and}$

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

have  $' \$st \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \ll t_1 \gg, \$tr' \mapsto_s \ll t_1 \gg + \ll t_2 \gg ] \dagger \text{post}_R Q ' \text{ and}$

**proof** –

have  $[ \$st \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t_2 \gg ] \dagger \text{post}_R Q =$

$[ \$st \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg ] \dagger [ \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t_2 \gg ] \dagger \text{post}_R Q$

by *rel-auto*

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

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

finally show *?thesis* using a5

by (*rel-auto*)

qed

with a3

have  $\text{post}PQ: ' \$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t_1 @ t_2 \gg ] \dagger (\text{post}_R P ;; \text{post}_R Q) ' \text{ and}$

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

have  $' \$st \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \ll t_1 \gg, \$tr' \mapsto_s \ll t_1 \gg + \ll t_2 \gg ] \dagger \text{pre}_R Q ' \text{ and}$

**proof** –

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

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

by *rel-auto*

also have  $\dots = [ \$st \mapsto_s \ll s_0 \gg ] \dagger [ \$tr \mapsto_s 0, \$tr' \mapsto_s \ll t_2 \gg ] \dagger \text{pre}_R Q$

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

finally show *?thesis* using a4

by (*rel-auto*)

qed

from a6

have  $a6': \bigwedge t s_1. [ t \leq t_1 @ t_2; ' \$st \mapsto_s \ll s \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t \gg ] \dagger \text{pre}_R P'; ' \$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t \gg ] \dagger \text{post}_R P' ] \implies$

$' \$st \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll (t_1 @ t_2) - t \gg ] \dagger \text{pre}_R Q ' \text{ and}$

apply (*subst (asm) taut-not*)

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

apply (*rel-auto*)

done

have  $\text{wp}R: ' \$st \mapsto_s \ll s \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t_1 @ t_2 \gg ] \dagger (\text{post}_R P \text{wp}_r \text{pre}_R Q) ' \text{ and}$

```

proof –
  have  $\bigwedge s_1 t_0. [t_0 \leq t_1 @ t_2; '[\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t_0 \gg] \dagger post_R P']$ 
     $\implies '[\$st \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll (t_1 @ t_2) - t_0 \gg] \dagger pre_R Q'$ 
proof –
  fix  $s_1 t_0$ 
  assume  $c: t_0 \leq t_1 @ t_2$   $'[\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t_0 \gg] \dagger post_R P'$ 

  have  $preP': '[\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t_0 \gg] \dagger pre_R P'$ 
  proof –
    have  $(pre_R P)[[0, \ll t_0 \gg / \$tr, \$tr']] \sqsubseteq (pre_R P)[[0, \ll t_1 @ t_2 \gg / \$tr, \$tr']]$ 
    by (simp add: RC-prefix-refine closure assms c)
    hence  $[\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t_0 \gg] \dagger pre_R P \sqsubseteq [\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t_1 @ t_2 \gg] \dagger pre_R P$ 
    by (rel-auto)
    thus ?thesis
    by (simp add: taut-refine-impl preP)
  qed

  with  $c$  a3 preP a6 [of t0 s1] show  $'[\$st \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll (t_1 @ t_2) - t_0 \gg] \dagger pre_R Q'$ 
    by (simp)
  qed

thus ?thesis
  apply (simp-all add: wp-rea-circus-form-alt assms closure unrest usubst rea-impl-alt-def)
  apply (simp add: R1-def usubst tcontr-alt-def)
  apply (auto intro!: taut-shAll-intro-2)
  apply (rule taut-impl-intro)
  apply (simp add: unrest-all-circus-vars-st-st' unrest closure assms)
  apply (rel-simp)
done
qed
show  $'([\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t_1 @ t_2 \gg] \dagger pre_R P \wedge$ 
   $[\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t_1 @ t_2 \gg] \dagger (post_R P \wp_r pre_R Q)) \wedge$ 
   $[\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \ll [] \gg, \$tr' \mapsto_s \ll t_1 @ t_2 \gg] \dagger (post_R P ;; post_R Q)'$ 
  by (auto simp add: taut-conj preP postPQ wpR)
qed
qed

lemma Cons-minus [simp]:  $(a \# t) - [a] = t$ 
  by (metis append-Cons append-Nil append-minus)

lemma traces-prefix:
  assumes  $P$  is NCSP
  shows  $tr[\ll a \gg \rightarrow_C P]s = \{(a \# t, s') \mid t s'. (t, s') \in tr[P]s\}$ 
  apply (auto simp add: PrefixCSP-def traces-seq traces-do divergences-do lit.rep-eq assms closure Healthy-if trace-divergence-disj)
  apply (meson assms trace-divergence-disj)
done

```

### 10.3 Deadlock Freedom

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

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

$[rdes-def]: CDF = \mathbf{R}_s(true_r \vdash (\prod (s, t, E, e) \cdot \mathcal{E}(\ll s \gg, \ll t \gg, \ll insert\ e\ E \gg))) \diamond true_r)$

**lemma**  $CDF\text{-}NCSP$   $[closure]: CDF \text{ is } NCSP$

**apply** ( $simp\ add: CDF\text{-}def$ )

**apply** ( $rule\ NCSP\text{-}rdes\text{-}intro$ )

**apply** ( $simp\text{-}all\ add: closure\ unrest$ )

**done**

**lemma**  $CDF\text{-}seq\text{-}idem: CDF ;; CDF = CDF$

**by** ( $rdes\text{-}eq$ )

**lemma**  $CDF\text{-}refine\text{-}intro: CDF \sqsubseteq P \implies CDF \sqsubseteq (CDF ;; P)$

**by** ( $metis\ CDF\text{-}seq\text{-}idem\ urel\text{-}diod.\text{mult}\text{-}isol$ )

**lemma**  $Skip\text{-}deadlock\text{-}free: CDF \sqsubseteq Skip$

**by** ( $rdes\text{-}refine$ )

**lemma**  $CDF\text{-}ext\text{-}st$   $[alpha]: CDF \oplus_p abs\text{-}st_L = CDF$

**by** ( $rdes\text{-}eq$ )

**end**

## 11 Meta-theory for Stateful-Failure Reactive Designs

**theory**  $utp\text{-}sf\text{-}rdes$

**imports**

$utp\text{-}sfrd\text{-}core$

$utp\text{-}sfrd\text{-}rel$

$utp\text{-}sfrd\text{-}healths$

$utp\text{-}sfrd\text{-}contracts$

$utp\text{-}sfrd\text{-}extchoice$

$utp\text{-}sfrd\text{-}prog$

$utp\text{-}sfrd\text{-}recursion$

$utp\text{-}sfrd\text{-}fdsem$

**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.