Circus Modelling Language in Isabelle/UTP

Simon Foster James Baxter Ana Cavalcanti Jim Woodcock
Samuel Canham

March 8, 2018

Contents

1 Introduction **2**

2 Circus Core Types **2**
2.1 Circus Alphabet . . . 3
2.2 Basic laws . . . 3
2.3 Unrestriction laws . . . 4

3 Circus Reactive Relations **5**
3.1 Healthiness Conditions . . . 5
3.2 Closure Properties . . . 6
3.3 Introduction laws . . . 10
3.4 Trace Substitution . . . 10
3.5 Initial Interaction . . . 12
3.6 Enabled Events . . . 13
3.7 Completed Trace Interaction . . . 14

4 Circus and CSP Healthiness Conditions **16**

5 Definitions **16**
5.1 Healthiness condition properties . . . 17
5.2 CSP theories . . . 28
5.3 Algebraic laws . . . 29

6 Reactive Contracts for CSP/Circus with refusals **30**

7 External Choice **31**
7.1 Definitions and syntax . . . 31
7.2 Basic laws . . . 32
7.3 Algebraic laws . . . 32
7.4 Reactive design calculations . . . 32
7.5 Productivity and Guardedness . . . 40
7.6 Algebraic laws . . . 41

8 Circus and CSP Actions **44**
8.1 Conditionals . . . 44
8.2 Guarded commands . . . 44
8.3 Alternation . . . 44
8.4 While Loops . . . 44
8.5 Assignment . . . 45
8.6 Assignment with update . . . 46
8.7 State abstraction . . . 46
8.8 Assumptions . . . 47
8.9 Guards . . . 47
8.10 Basic events . . . 50
8.11 Event prefix . . . 52
8.12 Guarded external choice . . . 54
8.13 Input prefix . . . 55
8.14 Algebraic laws . . . 56

9 Syntax and Translations for Event Prefix **58**

10 Recursion in Circus **60**
10.1 Fixed-points . . . 61
10.2 Example action expansion . . . 62

11 Circus Trace Merge **62**
11.1 Function Definition . . . 62
11.2 Lifted Trace Merge . . . 63
11.3 Trace Merge Lemmas . . . 63

12 Circus Parallel Composition **64**
12.1 Merge predicates . . . 64
12.2 Parallel operator . . . 74
12.3 Parallel Laws . . . 76

13 Linking to the Failures-Divergences Model **79**
13.1 Failures-Divergences Semantics . . . 80
13.2 Circus Operators . . . 81
13.3 Deadlock Freedom . . . 89

14 Meta theory for Circus **89**

1 Introduction

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

2 Circus Core Types

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

2.1 Circus Alphabet

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

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

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

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

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

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

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

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

translations
(*type*) $('\sigma,'\varphi)$ *st-csp* <= (*type*) $('\sigma, '\varphi$ *list*, $'\varphi 1$ *csp-vars*) *rsp*
(*type*) $('\sigma,'\varphi)$ *action* <= (*type*) $('\sigma, '\varphi)$ *st-csp hrel*

notation *csp-vars-child-lens*$_a$ (Σ_c)
notation *csp-vars-child-lens* (Σ_C)

2.2 Basic laws

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

by (*rel-auto*)

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

2.3 Unrestriction laws

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

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

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

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

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

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

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

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

lemma *R1-ref'-unrest* [*unrest*]: $\$ref´ \sharp P \Longrightarrow \$ref´ \sharp R1(P)$
by (*simp add*: *R1-def unrest*)

lemma *R2c-ref'-unrest* [*unrest*]: $\$ref´ \sharp P \Longrightarrow \$ref´ \sharp R2c(P)$
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 :: ($'e$, $'t$) *action*
assumes
$\$ok \sharp P$ $\$ok´ \sharp P$ $\$wait \sharp P$ $\$wait´ \sharp P$ $\$tr \sharp P$
$\$tr´ \sharp P$ $\$st \sharp P$ $\$st´ \sharp P$ $\$ref \sharp P$ $\$ref´ \sharp P$
shows $\Sigma \sharp P$
by (*rule bij-lens-unrest-all*[*OF circus-alpha-bij-lens*], *simp add*: *unrest assms*)

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

lemma *unrest-any-circus-var*:
fixes P :: ($'s$, $'e$) *action*
assumes $\$ok \sharp P$ $\$ok´ \sharp P$ $\$wait \sharp P$ $\$wait´ \sharp P$ $\$ref \sharp P$ $\$ref´ \sharp P$ $\Sigma \sharp s$ $\Sigma \sharp s'$ $\Sigma \sharp t$ $\Sigma \sharp t'$
shows $x \sharp [\$st \mapsto_s s, \$st´ \mapsto_s s', \$tr \mapsto_s t, \$tr´ \mapsto_s t'] \dagger P$
by (*simp add*: *unrest-all-var unrest-all-circus-vars-st-st′ assms*)

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

end

3 Circus Reactive Relations

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

3.1 Healthiness Conditions

CSP Reactive Relations

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

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

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

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

CSP Reactive Conditions

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

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

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

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

lemma *ref-unrest-RC* [*unrest*]: $\$ref \sharp P \Longrightarrow \$ref \sharp RC\ P$
  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 $(\exists\ \$ref\acute{} \cdot P)$ *is RR*
proof −
  have $RR\ (\exists\ \$ref\acute{} \cdot RR(P)) = (\exists\ \$ref\acute{} \cdot RR(P))$
    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-implies-RR* [*closure*]:
  assumes *P is CRC*
  shows *P is RR*
proof –
  have $RR(CRC(P)) = CRC(P)$
    by (*rel-auto*)
      (*metis* (*no-types*, *lifting*) *Prefix-Order.prefixE Prefix-Order.prefixI append.assoc append-minus*)+
  thus *?thesis*
    by (*metis Healthy-def assms*)
qed

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

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

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

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

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

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

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

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

by (*rel-auto*)

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

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

lemma *conj-CRC-closed* [*closure*]:
⟦ P *is CRC*; Q *is CRC* ⟧ $\Longrightarrow$ $(P \wedge Q)$ *is CRC*
by (*rule CRC-intro*, *simp-all add*: *unrest closure*)

lemma *disj-CRC-closed* [*closure*]:
⟦ P *is CRC*; Q *is CRC* ⟧ $\Longrightarrow$ $(P \vee Q)$ *is CRC*
by (*rule CRC-intro*, *simp-all add*: *unrest closure*)

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

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

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

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

lemma *rea-implies-CRR-closed* [*closure*]:
⟦ P *is CRR*; Q *is CRR* ⟧ $\Longrightarrow$ $(P \Rightarrow_r Q)$ *is CRR*
by (*simp-all add*: *CRR-intro closure unrest*)

lemma *conj-CRR-closed* [*closure*]:
⟦ P *is CRR*; Q *is CRR* ⟧ $\Longrightarrow$ $(P \wedge Q)$ *is CRR*
by (*simp-all add*: *CRR-intro closure unrest*)

lemma *disj-CRR-closed* [*closure*]:
⟦ P *is CRR*; Q *is CRR* ⟧ $\Longrightarrow$ $(P \vee Q)$ *is CRR*
by (*rule CRR-intro*, *simp-all add*: *unrest closure*)

lemma *rea-not-CRR-closed* [*closure*]:
P *is CRR* $\Longrightarrow$ $(\neg_r\ P)$ *is CRR*

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

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

lemma *st-cond-R1-closed* [*closure*]: $\llbracket$ *P is R1*; *Q is R1* $\rrbracket \Longrightarrow (P \triangleleft b \triangleright_R Q)$ *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 is CRR*; *Q is CRR* $\rrbracket \Longrightarrow (P \triangleleft b \triangleright_R Q)$ *is CRR*
by (*simp-all add*: *CRR-intro closure unrest*)

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

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

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

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

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

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

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

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

3.3 Introduction laws

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

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

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

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

3.4 Trace Substitution

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

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

lemma *unrest-trace-subst* [*unrest*]:
  $[\![$ *mwb-lens* x; $x \bowtie (\$tr)_v$; $x \bowtie (\$tr')_v$; $x \bowtie (\$st)_v$; $x \sharp P$ $]\!] \Longrightarrow x \sharp P[\![v]\!]_t$
  by (*simp add*: *trace-subst-def lens-indep-sym unrest*)

lemma *trace-subst-RR-closed* [*closure*]:
  assumes *P is RR*
  shows $P[\![v]\!]_t$ *is RR*
proof −
  have $(RR\ P)[\![v]\!]_t$ *is RR*
    apply (*rel-auto*)
    apply (*metis diff-add-cancel-left′ trace-class.add-left-mono*)
    apply (*metis le-add minus-cancel-le trace-class.add-diff-cancel-left*)
    using *le-add order-trans* **apply** *blast*
  done
  thus *?thesis*
    by (*simp add*: *Healthy-if assms*)
qed

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

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

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

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

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

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

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

lemma *tsubst-UINF-ind* [*usubst*]: $(\bigsqcap\ i \cdot P(i))[\![v]\!]_t = (\bigsqcap\ i \cdot (P(i))[\![v]\!]_t)$
  by (*rel-auto*)

3.5 Initial Interaction

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

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

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

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

lemma *rea-init-R2c* [*closure*]: $\mathcal{I}(s,t)$ *is R2c*
  apply (*rel-auto*)
  apply (*metis diff-add-cancel-left′ trace-class.add-left-mono*)
  apply (*metis le-add minus-cancel-le trace-class.add-diff-cancel-left*)
done

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

lemma *csp-init-RR* [*closure*]: $\mathcal{I}(s,t)$ *is RR*
  apply (*rel-auto*)
  apply (*metis diff-add-cancel-left′ trace-class.add-left-mono*)
  apply (*metis le-add minus-cancel-le trace-class.add-diff-cancel-left*)
  apply (*metis le-add less-le less-le-trans*)
done

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

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

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

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

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

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

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

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

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

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

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

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

lemma *tsubst-true* [*usubst*]: $true_r[\![v]\!]_t = \mathcal{I}(true,v)$
  by (*rel-auto*)

3.6 Enabled Events

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

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*]: $\mathcal{E}(s,t_2,e)[\![t_1]\!]_t = \mathcal{E}(s,t_1\ \hat{}_u t_2,e)$
  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*]:
  $[\![\ x \bowtie (\$tr)_v;\ x \bowtie (\$tr')_v;\ x \bowtie (\$st)_v;\ x \bowtie (\$ref')_v\ ]\!] \Longrightarrow x \sharp \mathcal{E}(s,t,e)$
  by (*simp add*: *csp-enable-def R1-def lens-indep-sym unrest*)

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

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

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

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

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

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

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

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

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

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

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

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

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

3.7 Completed Trace Interaction

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

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

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

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

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

$\Phi(b,\sigma,bop\ Cons\ x\ xs)$ *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*]:
$(\Phi(s,\sigma,t_2))[\![t_1]\!]_t = \Phi(s,\sigma,t_1\ \hat{}_u\ t_2)$
apply (*rel-auto*)
apply (*metis append.assoc less-eq-list-def prefix-concat-minus*)
apply (*simp add*: *list-concat-minus-list-concat*)
done

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

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

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

lemma *csp-do-assign* [*rpred*]:
assumes *P is CRR*
shows $\Phi(s,\ \sigma,\ t)\ ;;\ P = ([s]_{S<} \wedge (\lceil\sigma\rceil_{S\sigma} \dagger P)[\![t]\!]_t)$
proof −
have $\Phi(s,\sigma,t)\ ;;\ CRR(P) = ([s]_{S<} \wedge (\lceil\sigma\rceil_{S\sigma} \dagger CRR(P))[\![t]\!]_t)$
by (*rel-blast*)
thus *?thesis*
by (*simp add*: *Healthy-if assms*)
qed

lemma *subst-state-csp-enable* [*usubst*]:
$\lceil\sigma\rceil_{S\sigma} \dagger \mathcal{E}(s,t_2,e) = \mathcal{E}(\sigma \dagger s,\ \sigma \dagger t_2,\ \sigma \dagger e)$
by (*rel-auto*)

lemma *csp-do-assign-enable* [*rpred*]:
$\Phi(s_1,\sigma,t_1)\ ;;\ \mathcal{E}(s_2,t_2,e) = \mathcal{E}(s_1 \wedge \sigma \dagger s_2,\ t_1\ \hat{}_u(\sigma \dagger t_2),\ (\sigma \dagger e))$
by (*simp add*: *rpred closure usubst*)

lemma *csp-do-assign-do* [*rpred*]:
$\Phi(s_1,\sigma,t_1)\ ;;\ \Phi(s_2,\varrho,t_2) = \Phi(s_1 \wedge (\sigma \dagger s_2),\ \varrho \circ \sigma,\ t_1\ \hat{}_u(\sigma \dagger t_2))$
by (*rel-auto*)

lemma *csp-do-skip* [*rpred*]:
assumes *P is CRR*
shows $\Phi(true,id,t)\ ;;\ P = P[\![t]\!]_t$
proof −
have $\Phi(true,id,t)\ ;;\ CRR(P) = (CRR\ P)[\![t]\!]_t$
by (*rel-auto*)
thus *?thesis*
by (*simp add*: *Healthy-if assms*)
qed

lemma *wp-rea-csp-do-lemma*:
  fixes P :: $('\sigma,\ '\varphi)$ *action*
  assumes $\$ok \sharp P$ $\$wait \sharp P$ $\$ref \sharp P$
  shows $(\lceil \langle\sigma\rangle_a \rceil_S \wedge \$tr' =_u \$tr\ \hat{}_u\ \lceil t \rceil_{S<})\ ;;\ P = (\lceil\sigma\rceil_{S\sigma} \dagger P)[\![\$tr\ \hat{}_u\ \lceil t\rceil_{S<}/\$tr]\!]$
  using *assms* **by** (*rel-auto*, *meson*)

lemma *wp-rea-csp-do* [*wp*]:
  fixes P :: $('\sigma,\ '\varphi)$ *action*
  assumes P *is CRR*
  shows $\Phi(s,\sigma,t)\ wp_r\ P = (\mathcal{I}(s,t) \Rightarrow_r (\lceil\sigma\rceil_{S\sigma} \dagger P)[\![t]\!]_t)$
proof −
  have $\Phi(s,\sigma,t)\ wp_r\ CRR(P) = (\mathcal{I}(s,t) \Rightarrow_r (\lceil\sigma\rceil_{S\sigma} \dagger CRR(P))[\![t]\!]_t)$
    by (*rel-blast*)
  thus *?thesis*
    by (*simp add*: *assms Healthy-if*)
qed

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

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

lemma *wp-rea-csp-do-skip* [*wp*]:
  fixes Q :: $('\sigma,\ '\varphi)$ *action*
  assumes P *is CRR*
  shows $\Phi(s,id,t)\ wp_r\ P = (\mathcal{I}(s,t) \Rightarrow_r P[\![t]\!]_t)$
proof −
  have $\Phi(s,id,t)\ wp_r\ P = \Phi(s,id,t)\ wp_r\ P$
    by (*simp add*: *skip-r-def*)
  thus *?thesis* **by** (*simp add*: *wp assms usubst alpha*)
qed

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

end

4 Circus and CSP Healthiness Conditions

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

5 Definitions

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

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

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

abbreviation CSP :: $(('\sigma, '\varphi)$ *st-csp* $\times$ $('\sigma, '\varphi)$ *st-csp*$)$ *health*
where $CSP(P) \equiv SRD(P)$

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

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

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

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

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

definition $CSP4$ :: $(('\sigma, '\varphi)$ *st-csp* $\times$ $('\sigma, '\varphi)$ *st-csp*$)$ *health* **where**
[*upred-defs*]: $CSP4(P) = (P\ ;;\ Skip)$

definition $NCSP$ :: $(('\sigma, '\varphi)$ *st-csp* $\times$ $('\sigma, '\varphi)$ *st-csp*$)$ *health* **where**
[*upred-defs*]: $NCSP = CSP3 \circ CSP4 \circ CSP$

Productive and normal processes

abbreviation $PCSP \equiv Productive \circ NCSP$

Instantaneous and normal processes

abbreviation $ICSP \equiv ISRD1 \circ NCSP$

5.1 Healthiness condition properties

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

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

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

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

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

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

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

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

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

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

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

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

lemma *postR-Skip* [*rdes*]: $post_R(Skip) = \Phi(true,id,\langle\rangle)$
  by (*rel-auto*)

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

lemma *Skip-left-lemma*:
  assumes *P is CSP*
  shows $Skip ;; P = \mathbf{R}_s ((\forall\ \$ref \cdot pre_R\ P) \vdash (\exists\ \$ref \cdot cmt_R\ P))$
proof −
  have $Skip ;; P =$
      $\mathbf{R}_s ((\$tr\acute{} =_u \$tr \wedge \$st\acute{} =_u \$st)\ wp_r\ pre_R\ P \vdash$
        $(\$tr\acute{} =_u \$tr \wedge \$st\acute{} =_u \$st) ;; peri_R\ P \diamond$
        $(\$tr\acute{} =_u \$tr \wedge \$st\acute{} =_u \$st) ;; post_R\ P)$
    by (*simp add*: *SRD-composition-wp alpha rdes closure wp assms rpred C1*, *rel-auto*)
  also have ... $= \mathbf{R}_s ((\forall\ \$ref \cdot pre_R\ P) \vdash$
        $(\$tr\acute{} =_u \$tr \wedge \neg\ \$wait\acute{} \wedge \$st\acute{} =_u \$st) ;; ((\exists\ \$st \cdot \lceil II \rceil_D) \triangleleft \$wait \triangleright cmt_R\ P))$
    by (*rule cong*[*of* $\mathbf{R}_s$ $\mathbf{R}_s$], *simp*, *rel-auto*)
  also have ... $= \mathbf{R}_s ((\forall\ \$ref \cdot pre_R\ P) \vdash (\exists\ \$ref \cdot cmt_R\ P))$
    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* $\$ref \sharp P[\![false/\$wait]\!]$
  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 is CSP*; $\$ref \sharp P\llbracket false/\$wait \rrbracket$ $\rrbracket \Longrightarrow$ *P is CSP3*
by (*simp add*: *CSP3-def Healthy-def′ Skip-left-unit-ref-unrest*)

lemma *ref-unrest-RHS-design*:
assumes $\$ref \sharp P$ $\$ref \sharp Q_1$ $\$ref \sharp Q_2$
shows $\$ref \sharp (\mathbf{R}_s(P \vdash Q_1 \diamond Q_2))\ _f$
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* $\$ref \sharp pre_R(P)$ $\$ref \sharp peri_R(P)$ $\$ref \sharp post_R(P)$
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*]: $\$ref \sharp Skip\llbracket false/\$wait \rrbracket$
by (*simp add*: *Skip-def RHS-def R1-def R2c-def R2s-def R3h-def design-def usubst unrest*)

lemma *Skip-unrest-ref′* [*unrest*]: $\$ref\,' \sharp Skip\llbracket false/\$wait \rrbracket$
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* $\longleftrightarrow$ $(\$ref \sharp P\llbracket false/\$wait \rrbracket)$
proof
assume *1*: *P is CSP3*
have $\$ref \sharp (Skip \;;;\; P)\llbracket false/\$wait \rrbracket$
by (*simp add*: *usubst unrest*)
with *1* **show** $\$ref \sharp P\llbracket false/\$wait \rrbracket$
by (*metis CSP3-def Healthy-def*)
next
assume *1*:$\$ref \sharp P\llbracket false/\$wait \rrbracket$
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 $\$ref \sharp pre_R(P)$ $\$ref \sharp peri_R(P)$ $\$ref \sharp post_R(P)$
proof −
have *a*:$(\$ref \sharp P\llbracket false/\$wait \rrbracket)$
using *CSP3-iff assms* **by** *blast*
from *a* **show** $\$ref \sharp pre_R(P)$
by (*rel-blast*)
from *a* **show** $\$ref \sharp peri_R(P)$
by (*rel-blast*)
from *a* **show** $\$ref \sharp post_R(P)$
by (*rel-blast*)
qed

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

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

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

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

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

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

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

lemma *Skip-right-tri-lemma*:
assumes P *is* CSP
shows $P\ ;;\ Skip = \mathbf{R}_s\ ((\neg_r\ pre_R\ P)\ wp_r\ false \vdash ((\exists\ \$st' \cdot peri_R\ P) \diamond (\exists\ \$ref' \cdot post_R\ P)))$
proof −
have $((\exists\ \$st' \cdot cmt_R\ P) \triangleleft \$wait' \triangleright (\exists\ \$ref' \cdot cmt_R\ P)) = ((\exists\ \$st' \cdot peri_R\ P) \diamond (\exists\ \$ref' \cdot post_R\ P))$
by (*rel-auto*)
thus *?thesis* **by** (*simp add*: *Skip-right-lemma*[*OF assms*])
qed

lemma *CSP4-intro*:

assumes P *is CSP* $(\neg_r\ pre_R(P))$;; $R1(true) = (\neg_r\ pre_R(P))$
$\$st' \sharp (cmt_R\ P)[\![true/\$wait']\!]$ $\$ref' \sharp (cmt_R\ P)[\![false/\$wait']\!]$
shows P *is CSP4*
proof –
have $CSP4(P) = \mathbf{R}_s\ ((\neg_r\ pre_R\ P)\ wp_r\ false \vdash ((\exists\ \$st' \cdot cmt_R\ P) \triangleleft \$wait' \triangleright (\exists\ \$ref' \cdot cmt_R\ P)))$
by (*simp add*: *CSP4-def Skip-right-lemma assms*(*1*))
also have ... $= \mathbf{R}_s\ (pre_R(P) \vdash ((\exists\ \$st' \cdot cmt_R\ P)[\![true/\$wait']\!] \triangleleft \$wait' \triangleright (\exists\ \$ref' \cdot cmt_R\ P)[\![false/\$wait']\!]))$
by (*simp add*: *wp-rea-def assms*(*2*) *rpred closure cond-var-subst-left cond-var-subst-right*)
also have ... $= \mathbf{R}_s\ (pre_R(P) \vdash ((\exists\ \$st' \cdot (cmt_R\ P)[\![true/\$wait']\!]) \triangleleft \$wait' \triangleright (\exists\ \$ref' \cdot (cmt_R\ P)[\![false/\$wait']\!])))$
by (*simp add*: *usubst unrest*)
also have ... $= \mathbf{R}_s\ (pre_R\ P \vdash ((cmt_R\ P)[\![true/\$wait']\!] \triangleleft \$wait' \triangleright (cmt_R\ P)[\![false/\$wait']\!]))$
by (*simp add*: *ex-unrest assms*)
also have ... $= \mathbf{R}_s\ (pre_R\ P \vdash cmt_R\ P)$
by (*simp add*: *cond-var-split*)
also have ... $= P$
by (*simp add*: *SRD-reactive-design-alt assms*(*1*))
finally show *?thesis*
by (*simp add*: *Healthy-def'*)
qed

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

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

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

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

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

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

lemma *CSP4-implies-RD3*: *P is CSP4* $\Longrightarrow$ *P is RD3*
by (*metis CSP4-right-subsumes-RD3 Healthy-def*)

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

lemma *CSP4-NSRD-intro*:
assumes *P is NSRD* $\$ref\acute{}\ \sharp\ post_R(P)$
shows *P is CSP4*
by (*simp add*: *CSP4-tri-intro NSRD-is-SRD NSRD-neg-pre-unit NSRD-st′-unrest-peri assms*)

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

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

lemma *NCSP-elim* [*RD-elim*]:
$[\![\ X\ is\ NCSP;\ P(\mathbf{R}_s(pre_R(X) \vdash peri_R(X) \diamond post_R(X)))\ ]\!] \Longrightarrow P(X)$
by (*simp add*: *SRD-reactive-tri-design closure*)

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

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

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

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

lemma *NCSP-subset-implies-CSP* [*closure*]:
$A \subseteq [\![NCSP]\!]_H \Longrightarrow A \subseteq [\![CSP]\!]_H$
using *NCSP-implies-CSP* **by** *blast*

lemma *NCSP-subset-implies-NSRD* [*closure*]:
$A \subseteq [\![NCSP]\!]_H \Longrightarrow A \subseteq [\![NSRD]\!]_H$
using *NCSP-implies-NSRD* **by** *blast*

lemma *CSP-Healthy-subset-member*: $[\![\ P \in A;\ A \subseteq [\![CSP]\!]_H\ ]\!] \Longrightarrow$ *P is CSP*
by (*simp add*: *is-Healthy-subset-member*)

lemma *CSP3-Healthy-subset-member*: $[\![\ P \in A;\ A \subseteq [\![CSP3]\!]_H\ ]\!] \Longrightarrow$ *P is CSP3*
by (*simp add*: *is-Healthy-subset-member*)

lemma *CSP4-Healthy-subset-member*: $[\![P \in A;\ A \subseteq [\![CSP4]\!]_H\ ]\!] \Longrightarrow P\ is\ CSP4$
  by (*simp add*: *is-Healthy-subset-member*)

lemma *NCSP-Healthy-subset-member*: $[\![P \in A;\ A \subseteq [\![NCSP]\!]_H\ ]\!] \Longrightarrow P\ is\ NCSP$
  by (*simp add*: *is-Healthy-subset-member*)

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

lemma *Skip-left-unit*: $P\ is\ NCSP \Longrightarrow Skip\ ;;\ P = P$
  by (*metis* (*full-types*) *CSP3-def Healthy-if NCSP-implies-CSP3*)

lemma *Skip-right-unit*: $P\ is\ NCSP \Longrightarrow P\ ;;\ Skip = P$
  by (*metis* (*full-types*) *CSP4-def Healthy-if NCSP-implies-CSP4*)

lemma *NCSP-NSRD-intro*:
  assumes $P\ is\ NSRD\ \$ref \sharp pre_R(P)\ \$ref \sharp peri_R(P)\ \$ref \sharp post_R(P)\ \$ref' \sharp post_R(P)$
  shows *P is NCSP*
  by (*simp add*: *CSP3-SRD-intro CSP4-NSRD-intro NCSP-intro NSRD-is-SRD assms*)

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

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

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

lemma *CSP4-st′-unrest-peri* [*unrest*]:
  assumes *P is CSP P is CSP4*
  shows $\$st' \sharp peri_R(P)$
  by (*simp add*: *NSRD-CSP4-intro NSRD-st′-unrest-peri assms*)

lemma *CSP4-healthy-form*:
assumes *P is CSP P is CSP4*
shows $P = \mathbf{R}_s((\neg_r\ pre_R\ P)\ wp_r\ false \vdash ((\exists\ \$st' \cdot peri_R(P)) \diamond (\exists\ \$ref' \cdot post_R(P))))$
proof −
have $P = \mathbf{R}_s\ ((\neg_r\ pre_R\ P)\ wp_r\ false \vdash ((\exists\ \$st' \cdot cmt_R\ P) \triangleleft \$wait' \triangleright (\exists\ \$ref' \cdot cmt_R\ P)))$
by (*metis CSP4-def Healthy-def Skip-right-lemma assms*(*1*) *assms*(*2*))
also have ... $= \mathbf{R}_s\ ((\neg_r\ pre_R\ P)\ wp_r\ false \vdash ((\exists\ \$st' \cdot cmt_R\ P)[\![true/\$wait']\!] \triangleleft \$wait' \triangleright (\exists\ \$ref' \cdot cmt_R\ P)[\![false/\$wait']\!]))$
by (*metis* (*no-types*, *hide-lams*) *subst-wait′-left-subst subst-wait′-right-subst wait′-cond-def*)
also have ... $= \mathbf{R}_s((\neg_r\ pre_R\ P)\ wp_r\ false \vdash ((\exists\ \$st' \cdot peri_R(P)) \diamond (\exists\ \$ref' \cdot post_R(P))))$
by (*simp add*: *wait′-cond-def usubst* $peri_R$*-def* $post_R$*-def* cmt_R*-def unrest*)
finally show *?thesis* **.**
qed

lemma *CSP4-ref′-unrest-pre* [*unrest*]:
assumes *P is CSP P is CSP4*
shows $\$ref' \sharp\ pre_R(P)$
proof −
have $pre_R(P) = pre_R(\mathbf{R}_s((\neg_r\ pre_R\ P)\ wp_r\ false \vdash ((\exists\ \$st' \cdot peri_R(P)) \diamond (\exists\ \$ref' \cdot post_R(P)))))$
using *CSP4-healthy-form assms*(*1*) *assms*(*2*) **by** *fastforce*
also have ... $= (\neg_r\ pre_R\ P)\ wp_r\ false$
by (*simp add*: *rea-pre-RHS-design wp-rea-def usubst unrest*
CSP4-neg-pre-unit R1-rea-not R2c-preR R2c-rea-not assms)
also have $\$ref' \sharp$ ...
by (*simp add*: *wp-rea-def unrest*)
finally show *?thesis* **.**
qed

lemma *NCSP-set-unrest-pre-wait′*:
assumes $A \subseteq [\![NCSP]\!]_H$
shows $\bigwedge P.\ P \in A \Longrightarrow \$wait' \sharp\ pre_R(P)$
proof −
fix *P*
assume $P \in A$
hence *P is NSRD*
using *NCSP-implies-NSRD assms* **by** *auto*
thus $\$wait' \sharp\ pre_R(P)$
using *NSRD-wait′-unrest-pre* **by** *blast*
qed

lemma *CSP4-set-unrest-pre-st′*:
assumes $A \subseteq [\![CSP]\!]_H$ $A \subseteq [\![CSP4]\!]_H$
shows $\bigwedge P.\ P \in A \Longrightarrow \$st' \sharp\ pre_R(P)$
proof −
fix *P*
assume $P \in A$
hence *P is NSRD*
using *NSRD-CSP4-intro assms*(*1*) *assms*(*2*) **by** *blast*
thus $\$st' \sharp\ pre_R(P)$
using *NSRD-st′-unrest-pre* **by** *blast*
qed

lemma *CSP4-ref′-unrest-post* [*unrest*]:
assumes *P is CSP P is CSP4*
shows $\$ref' \sharp\ post_R(P)$

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

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

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

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

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

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

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

lemma *NCSP-seqr-closure* [*closure*]:
assumes *P is NCSP Q is NCSP*
shows *P* ;; *Q is NCSP*
by (*metis* (*no-types*, *lifting*) *CSP3-def CSP4-def Healthy-def' NCSP-implies-CSP NCSP-implies-CSP3 NCSP-implies-CSP4 NCSP-intro SRD-seqr-closure assms(1) assms(2) seqr-assoc*)

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

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

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

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

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

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

lemma *preR-Stop* [*rdes*]: $pre_R(Stop) = true_r$
by (*simp add*: *Stop-def Stop-is-CSP rea-pre-RHS-design unrest usubst R2c-true*)

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

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

lemma *cmtR-Stop* [*rdes*]: $cmt_R(Stop) = (\$tr' =_u \$tr \wedge \$wait')$
by (*rel-auto*)

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

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

lemma *preR-CRR* [*closure*]: P *is NCSP* $\Longrightarrow pre_R(P)$ *is CRR*
by (*rule CRR-intro*, *simp-all add*: *closure unrest*)

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

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

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

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

lemma *CSP3-Sup-closure* [*closure*]:
$A \subseteq [\![CSP3]\!]_H \Longrightarrow (\bigsqcap A)$ *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*]:
$A \subseteq [\![CSP4]\!]_H \Longrightarrow (\bigsqcap\ A)$ *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*]: $[\![\ A \subseteq [\![NCSP]\!]_H;\ A \neq \{\}\ ]\!] \Longrightarrow (\bigsqcap\ A)$ *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*]: $[\![\ \bigwedge\ i.\ P(i)$ *is NCSP*; $A \neq \{\}\ ]\!] \Longrightarrow (\bigsqcap\ i{\in}A.\ P(i))$ *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 ... $= \mathbf{R}_s\ ((\forall\ \$ref \cdot (\neg_r\ pre_R(NCSP\ P))\ wp_r\ false) \vdash$
$(\exists\ \$ref \cdot \exists\ \$st' \cdot peri_R(NCSP\ P)) \diamond$
$((\exists\ \$ref \cdot \exists\ \$ref' \cdot post_R\ (NCSP\ P)) \wedge \$tr <_u \$tr'))$
by (*simp add*: *NCSP-form Productive-RHS-design-form unrest closure*)
also have ... *is NCSP*
apply (*rule NCSP-rdes-intro*)
apply (*rule CRC-intro*)
apply (*simp-all add*: *unrest ex-unrest all-unrest closure*)
done
finally show *?thesis* **.**
qed

lemma *PCSP-elim* [*RD-elim*]:
assumes *X is PCSP P* $(\mathbf{R}_s\ ((pre_R\ X) \vdash peri_R\ X \diamond (R4(post_R\ X))))$
shows *P X*
by (*metis R4-def Healthy-if NCSP-implies-CSP PCSP-implies-NCSP Productive-form assms comp-apply*)

lemma *ICSP-implies-NCSP* [*closure*]:
assumes *P is ICSP*
shows *P is NCSP*
proof –
have $P = ISRD1(NCSP(NCSP\ P))$
by (*metis* (*no-types*, *hide-lams*) *Healthy-def′ Idempotent-def NCSP-Idempotent assms comp-apply*)
also have ... $= ISRD1\ (\mathbf{R}_s\ ((\forall\ \$ref \cdot (\neg_r\ pre_R\ (NCSP\ P))\ wp_r\ false) \vdash$
$(\exists\ \$ref \cdot \exists\ \$st' \cdot peri_R\ (NCSP\ P)) \diamond$
$(\exists\ \$ref \cdot \exists\ \$ref' \cdot post_R\ (NCSP\ P))))$
by (*simp add*: *NCSP-form*)
also have ... $= \mathbf{R}_s\ ((\forall\ \$ref \cdot (\neg_r\ pre_R(NCSP\ P))\ wp_r\ false) \vdash$
$false \diamond$

$((\exists \ \$ref \cdot \exists \ \$ref' \cdot post_R \ (NCSP \ P)) \wedge \$tr' =_u \$tr))$
by (*simp-all add*: *ISRD1-RHS-design-form closure rdes unrest*)
also have ... *is NCSP*
apply (*rule NCSP-rdes-intro*)
apply (*rule CRC-intro*)
apply (*simp-all add*: *unrest ex-unrest all-unrest closure*)
done
finally show *?thesis* **.**
qed

lemma *ICSP-implies-ISRD* [*closure*]:
assumes *P is ICSP*
shows *P is ISRD*
by (*metis* (*no-types*, *hide-lams*) *Healthy-def ICSP-implies-NCSP ISRD-def NCSP-implies-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*:
$(P \wedge (\$tr' =_u \$tr)) \ ;; \ true_r = true_r \Longrightarrow (P \wedge (\$tr' =_u \$tr)) \ ;; \ (\$tr' =_u \$tr) = (\$tr' =_u \$tr)$
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*: ⟦ *P is NCSP*; *P is ISRD1* ⟧ ⟹ *P is ICSP*
using *Healthy-comp* **by** *blast*

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

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

5.2 CSP theories

typedecl *TCSP*

abbreviation *TCSP* ≡ *UTHY*(*TCSP*, $('\sigma,'\varphi)$ *st-csp*)

overloading
tcsp-hcond == *utp-hcond* :: (*TCSP*, $('\sigma,'\varphi)$ *st-csp*) *uthy* ⇒ (($('\sigma,'\varphi)$ *st-csp* × $('\sigma,'\varphi)$ *st-csp*) *health*

tcsp-unit == *utp-unit* :: (*TCSP*, ($'\sigma$,$'\varphi$) *st-csp*) *uthy* ⇒ ($'\sigma$, $'\varphi$) *action*
begin
definition *tcsp-hcond* :: (*TCSP*, ($'\sigma$,$'\varphi$) *st-csp*) *uthy* ⇒ (($'\sigma$,$'\varphi$) *st-csp* × ($'\sigma$,$'\varphi$) *st-csp*) *health* **where**
[*upred-defs*]: *tcsp-hcond T* = *NCSP*
definition *tcsp-unit* :: (*TCSP*, ($'\sigma$,$'\varphi$) *st-csp*) *uthy* ⇒ ($'\sigma$, $'\varphi$) *action* **where**
[*upred-defs*]: *tcsp-unit T* = *Skip*
end

interpretation *csp-theory*: *utp-theory-kleene UTHY*(*TCSP*, ($'\sigma$,$'\varphi$) *st-csp*)
rewrites ⋀ *P*. *P* ∈ *carrier* (*uthy-order TCSP*) ⟷ *P is NCSP*
and *P is* $\mathcal{H}_{TCSP}$ ⟷ *P is NCSP*
and $\mathcal{II}_{TCSP}$ = *Skip*
and *carrier* (*uthy-order TCSP*) → *carrier* (*uthy-order TCSP*) ≡ $[\![NCSP]\!]_H \to [\![NCSP]\!]_H$
and *A* ⊆ *carrier* (*uthy-order TCSP*) ⟷ *A* ⊆ $[\![NCSP]\!]_H$
and *le* (*uthy-order TCSP*) = *op* ⊑
by (*unfold-locales*, *simp-all add*: *tcsp-hcond-def tcsp-unit-def Skip-left-unit Skip-right-unit Healthy-if closure*)

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

abbreviation *TestC* ($test_C$) **where**
$test_C$ *P* ≡ *utest TCSP P*

abbreviation *StarC* :: ($'\sigma$, $'\varphi$) *action* ⇒ ($'\sigma$, $'\varphi$) *action* (-$^{\star C}$ [*999*] *999*) **where**
StarC P ≡ $P\star_{TCSP}$

lemma *csp-bottom-Chaos*: $\bot_{TCSP}$ = *Chaos*
proof −
have *1*: $\bot_{TCSP}$ = *CSP3* (*CSP4* (*CSP true*))
by (*simp add*: *csp-theory.healthy-bottom*, *simp add*: *tcsp-hcond-def NCSP-def*)
also have *2*:... = *CSP3* (*CSP4 Chaos*)
by (*metis srdes-hcond-def srdes-theory-continuous.healthy-bottom*)
also have *3*:... = *Chaos*
by (*metis CSP3-Chaos CSP4-Chaos Healthy-def'*)
finally show *?thesis* **.**
qed

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

5.3 Algebraic laws

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

end

6 Reactive Contracts for CSP/Circus with refusals

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

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

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

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
  $[P \vdash Q \mid R]_C$ => *CONST mk-CRD P* (λ *-trace-var -ref-var. Q*) (λ *-trace-var. R*)
  $[P \vdash Q \mid R]_C$ <= *CONST mk-CRD P* (λ *x r. Q*) (λ *y. R*)

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

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

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

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

Refinement introduction law for contracts

lemma *CRD-contract-refine*:
  assumes
    Q is CSP ‘$\lceil P_1 \rceil_{S<} \Rightarrow pre_R\ Q$‘
    ‘$\lceil P_1 \rceil_{S<} \wedge peri_R\ Q \Rightarrow \lceil P_2\ t\ r \rceil_{S<}[\![t{\to}\&tt]\!][\![r{\to}\$ref']\!]$‘
    ‘$\lceil P_1 \rceil_{S<} \wedge post_R\ Q \Rightarrow \lceil P_3\ x \rceil_S[\![x{\to}\&tt]\!]$‘
  shows $[P_1 \vdash P_2\ trace\ refs \mid P_3(trace)]_C \sqsubseteq Q$
proof −
  have $[P_1 \vdash P_2\ trace\ refs \mid P_3(trace)]_C \sqsubseteq \mathbf{R}_s(pre_R(Q) \vdash peri_R(Q) \diamond post_R(Q))$
    using *assms* **by** (*simp add*: *mk-CRD-def*, *rule-tac srdes-tri-refine-intro*, *rel-auto*+)
  thus *?thesis*

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

lemma *CRD-contract-refine*′:
assumes
Q *is CSP* ‘$\lceil P_1 \rceil_{S<} \Rightarrow pre_R\ Q$‘
$\lceil P_2\ t\ r \rceil_{S<}[\![t \rightarrow \&tt]\!][\![r \rightarrow \$ref']\!] \sqsubseteq (\lceil P_1 \rceil_{S<} \wedge peri_R\ Q)$
$\lceil P_3\ x \rceil_S[\![x \rightarrow \&tt]\!] \sqsubseteq (\lceil P_1 \rceil_{S<} \wedge post_R\ Q)$
shows $[P_1 \vdash P_2\ trace\ refs \mid P_3(trace)]_C \sqsubseteq Q$
using *assms* **by** (*rule-tac CRD-contract-refine*, *simp-all add*: *refBy-order*)

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

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

end

7 External Choice

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

7.1 Definitions and syntax

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

syntax
-ExtChoice :: *pttrn* $\Rightarrow$ *'a set* $\Rightarrow$ *'b* $\Rightarrow$ *'b* ((*3*□ -∈- ·/ -) [*0*, *0*, *10*] *10*)
-ExtChoice-simp :: *pttrn* $\Rightarrow$ *'b* $\Rightarrow$ *'b* ((*3*□ - ·/ -) [*0*, *10*] *10*)

translations
$\Box P{\in}A \cdot B \rightleftharpoons$ *CONST ExtChoice* $((\lambda P.\ B)$ ‘ $A)$
$\Box P \cdot B \rightleftharpoons$ *CONST ExtChoice* (*CONST range* $(\lambda P.\ B)$)

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

Small external choice as an indexed big external choice.

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

7.2 Basic laws

7.3 Algebraic laws

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

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

7.4 Reactive design calculations

lemma *ExtChoice-rdes*:
assumes $\bigwedge i.\ \$ok' \sharp P(i)\ A \neq \{\}$
shows $(\Box i{\in}A \cdot \mathbf{R}_s(P(i) \vdash Q(i))) = \mathbf{R}_s((\bigsqcup i{\in}A \cdot P(i)) \vdash ((\bigsqcup i{\in}A \cdot Q(i)) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\bigsqcap i{\in}A \cdot Q(i))))$
proof −
have $(\Box i{\in}A \cdot \mathbf{R}_s(P(i) \vdash Q(i))) =$
$\mathbf{R}_s\ ((\bigsqcup i{\in}A \cdot pre_R\ (\mathbf{R}_s\ (P\ i \vdash Q\ i))) \vdash$
$((\bigsqcup i{\in}A \cdot cmt_R\ (\mathbf{R}_s\ (P\ i \vdash Q\ i)))$
$\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
$(\bigsqcap i{\in}A \cdot cmt_R\ (\mathbf{R}_s\ (P\ i \vdash Q\ i)))))$
by (*simp add*: *ExtChoice-def*)
also have ... =
$\mathbf{R}_s\ ((\bigsqcup i{\in}A \cdot R1\ (R2c\ (pre_s \dagger P(i)))) \vdash$
$((\bigsqcup i{\in}A \cdot R1(R2c(cmt_s \dagger (P(i) \Rightarrow Q(i)))))$
$\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
$(\bigsqcap i{\in}A \cdot R1(R2c(cmt_s \dagger (P(i) \Rightarrow Q(i)))))))$
by (*simp add*: *rea-pre-RHS-design rea-cmt-RHS-design*)
also have ... =
$\mathbf{R}_s\ ((\bigsqcup i{\in}A \cdot R1\ (R2c\ (pre_s \dagger P(i)))) \vdash$
$R1(R2c$
$((\bigsqcup i{\in}A \cdot R1(R2c(cmt_s \dagger (P(i) \Rightarrow Q(i)))))$
$\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
$(\bigsqcap i{\in}A \cdot R1(R2c(cmt_s \dagger (P(i) \Rightarrow Q(i)))))))))$
by (*metis* (*no-types*, *lifting*) *RHS-design-export-R1 RHS-design-export-R2c*)
also have ... =
$\mathbf{R}_s\ ((\bigsqcup i{\in}A \cdot R1\ (R2c\ (pre_s \dagger P(i)))) \vdash$
$R1(R2c$
$((\bigsqcup i{\in}A \cdot (cmt_s \dagger (P(i) \Rightarrow Q(i))))$
$\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
$(\bigsqcap i{\in}A \cdot (cmt_s \dagger (P(i) \Rightarrow Q(i)))))))$
by (*simp add*: *R2c-UINF R2c-condr 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 †
$((\bigsqcup i{\in}A \cdot (cmt_s \dagger (P(i) \Rightarrow Q(i))))$
$\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
$(\bigsqcap i{\in}A \cdot (cmt_s \dagger (P(i) \Rightarrow Q(i))))))$
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 †
$((\bigsqcup i{\in}A \cdot (P(i) \Rightarrow Q(i)))$
$\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
$(\bigsqcap i{\in}A \cdot (P(i) \Rightarrow Q(i)))))$
by (*simp add*: *usubst*)
also have ... =
$\mathbf{R}_s$ $((\bigsqcup i{\in}A \cdot R1\ (R2c\ (pre_s \dagger P(i)))) \vdash$
$((\bigsqcup i{\in}A \cdot (P(i) \Rightarrow Q(i))) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\bigsqcap i{\in}A \cdot (P(i) \Rightarrow Q(i)))))$
by (*simp add*: *rdes-export-cmt*)
also have ... =
$\mathbf{R}_s$ $((R1(R2c(\bigsqcup i{\in}A \cdot (pre_s \dagger P(i))))) \vdash$
$((\bigsqcup i{\in}A \cdot (P(i) \Rightarrow Q(i))) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\bigsqcap i{\in}A \cdot (P(i) \Rightarrow Q(i)))))$
by (*simp add*: *not-UINF R1-UINF R2c-UINF assms*)
also have ... =
$\mathbf{R}_s$ $((R2c(\bigsqcup i{\in}A \cdot (pre_s \dagger P(i)))) \vdash$
$((\bigsqcup i{\in}A \cdot (P(i) \Rightarrow Q(i))) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\bigsqcap i{\in}A \cdot (P(i) \Rightarrow Q(i)))))$
by (*simp add*: *R1-design-R1-pre*)
also have ... =
$\mathbf{R}_s$ $(((\bigsqcup i{\in}A \cdot (pre_s \dagger P(i)))) \vdash$
$((\bigsqcup i{\in}A \cdot (P(i) \Rightarrow Q(i))) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\bigsqcap i{\in}A \cdot (P(i) \Rightarrow Q(i)))))$
by (*metis* (*no-types*, *lifting*) *RHS-design-R2c-pre*)
also have ... =
$\mathbf{R}_s$ $(([\$ok \mapsto_s true, \$wait \mapsto_s false] \dagger (\bigsqcup i{\in}A \cdot P(i))) \vdash$
$((\bigsqcup i{\in}A \cdot (P(i) \Rightarrow Q(i))) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\bigsqcap i{\in}A \cdot (P(i) \Rightarrow Q(i)))))$
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 ... =
$\mathbf{R}_s$ $((\bigsqcup i{\in}A \cdot P(i)) \vdash ((\bigsqcup i{\in}A \cdot (P(i) \Rightarrow Q(i))) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\bigsqcap i{\in}A \cdot (P(i) \Rightarrow Q(i)))))$
by (*simp add*: *rdes-export-pre not-UINF*)
also have ... = $\mathbf{R}_s$ $((\bigsqcup i{\in}A \cdot P(i)) \vdash ((\bigsqcup i{\in}A \cdot Q(i)) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\bigsqcap i{\in}A \cdot Q(i))))$
by (*rule cong*[*of* $\mathbf{R}_s$ $\mathbf{R}_s$], *simp*, *rel-auto*, *blast+*)

finally show *?thesis* .
qed

lemma *ExtChoice-tri-rdes*:
assumes $\bigwedge\ i\ .\ \$ok' \sharp P_1(i)$ $A \neq \{\}$
shows $(\Box\ i{\in}A \cdot \mathbf{R}_s(P_1(i) \vdash P_2(i) \diamond P_3(i))) =$
$\mathbf{R}_s$ $((\bigsqcup\ i{\in}A \cdot P_1(i)) \vdash (((\bigsqcup\ i{\in}A \cdot P_2(i)) \triangleleft \$tr' =_u \$tr \triangleright (\bigsqcap\ i{\in}A \cdot P_2(i))) \diamond (\bigsqcap\ i{\in}A \cdot P_3(i))))$
proof −
have $(\Box\ i{\in}A \cdot \mathbf{R}_s(P_1(i) \vdash P_2(i) \diamond P_3(i))) =$
$\mathbf{R}_s$ $((\bigsqcup\ i{\in}A \cdot P_1(i)) \vdash ((\bigsqcup\ i{\in}A \cdot P_2(i) \diamond P_3(i)) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\bigsqcap\ i{\in}A \cdot P_2(i) \diamond$

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

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

lemma *ExtChoice-tri-rdes-def* [*rdes-def*]:
assumes $A \subseteq [\![CSP]\!]_H$
shows *ExtChoice* $A = \mathbf{R}_s$ $((\bigsqcup\ P{\in}A \cdot pre_R\ P) \vdash (((\bigsqcup\ P{\in}A \cdot peri_R\ P) \triangleleft \$tr' =_u \$tr \triangleright (\bigsqcap\ P{\in}A \cdot peri_R\ P)) \diamond (\bigsqcap\ P{\in}A \cdot post_R\ P)))$
proof −
have $((\bigsqcup\ P{\in}A \cdot cmt_R\ P) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\bigsqcap\ P{\in}A \cdot cmt_R\ P)) =$
$(((\bigsqcup\ P{\in}A \cdot cmt_R\ P) \triangleleft \$tr' =_u \$tr \triangleright (\bigsqcap\ P{\in}A \cdot cmt_R\ P)) \diamond (\bigsqcap\ P{\in}A \cdot cmt_R\ P))$
by (*rel-auto*)
also have ... = $(((\bigsqcup\ P{\in}A \cdot peri_R\ P) \triangleleft \$tr' =_u \$tr \triangleright (\bigsqcap\ P{\in}A \cdot peri_R\ P)) \diamond (\bigsqcap\ P{\in}A \cdot post_R\ P))$
by (*rel-auto*)
finally show *?thesis*
by (*simp add*: *ExtChoice-def*)
qed

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

qed

lemma *extChoice-tri-rdes*:
assumes $\$ok' \sharp P_1$ $\$ok' \sharp Q_1$
shows $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) \square \mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3) =$
$\mathbf{R}_s\ ((P_1 \wedge Q_1) \vdash (((P_2 \wedge Q_2) \triangleleft \$tr' =_u \$tr \triangleright (P_2 \vee Q_2)) \diamond (P_3 \vee Q_3)))$
proof −
have $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) \square \mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3) =$
$\mathbf{R}_s\ ((P_1 \wedge Q_1) \vdash ((P_2 \diamond P_3 \wedge Q_2 \diamond Q_3) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (P_2 \diamond P_3 \vee Q_2 \diamond Q_3)))$
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 $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) \square \mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3) =$
$\mathbf{R}_s\ ((P_1 \wedge Q_1) \vdash (((P_2 \wedge Q_2) \triangleleft \$tr' =_u \$tr \triangleright (P_2 \vee Q_2)) \diamond (P_3 \vee Q_3)))$
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 $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) \square \mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3) =$
$\mathbf{R}_s\ ((P_1 \wedge Q_1) \vdash ((R5(P_2 \wedge Q_2) \vee R4(P_2 \vee Q_2)) \diamond (P_3 \vee Q_3)))$
by (*simp add*: *extChoice-rdes-def assms*, *rel-auto*, *simp-all add*: *less-le*)

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

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

lemma *preR-ExtChoice* [*rdes*]:
assumes $A \neq \{\}$ $A \subseteq [\![CSP]\!]_H$
shows $pre_R(ExtChoice\ A) = (\bigsqcup\ P{\in}A \cdot pre_R(P))$
proof −
have $pre_R\ (ExtChoice\ A) = (R1\ (R2c\ ((\bigsqcup\ P{\in}A \cdot pre_R\ P))))$
by (*simp add*: *ExtChoice-def rea-pre-RHS-design usubst unrest*)
also from *assms* **have** ... $= (R1\ (R2c\ (\bigsqcup\ P{\in}A \cdot (pre_R(CSP(P))))))$
by (*metis USUP-healthy*)
also from *assms* **have** ... $= (\bigsqcup\ P{\in}A \cdot (pre_R(CSP(P))))$
by (*rel-auto*)
also from *assms* **have** ... $= (\bigsqcup\ P{\in}A \cdot (pre_R(P)))$
by (*metis USUP-healthy*)

finally show *?thesis* **.**
qed

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

lemma *periR-ExtChoice* [*rdes*]:
assumes $A \subseteq [\![NCSP]\!]_H$ $A \neq \{\}$
shows $peri_R(ExtChoice\ A) = ((\bigsqcup\ P{\in}A \cdot pre_R(P)) \Rightarrow_r (\bigsqcup\ P{\in}A \cdot peri_R\ P)) \triangleleft \$tr' =_u \$tr \triangleright (\bigsqcap\ P{\in}A \cdot peri_R\ P)$
proof –
have $peri_R\ (ExtChoice\ A) = peri_R\ (\mathbf{R}_s\ ((\bigsqcup\ P \in A \cdot pre_R\ P) \vdash$
$((\bigsqcup\ P \in A \cdot peri_R\ P) \triangleleft \$tr' =_u \$tr \triangleright (\bigsqcap\ P \in A \cdot peri_R\ P)) \diamond$
$(\bigsqcap\ P \in A \cdot post_R\ P)))$
by (*simp add*: *ExtChoice-tri-rdes-def assms closure*)

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

also have ... $= R1\ (R2c\ ((\bigsqcup\ P{\in}A \cdot pre_R\ (NCSP\ P)) \Rightarrow_r$
$(\bigsqcup\ P{\in}A \cdot peri_R\ (NCSP\ P))$
$\triangleleft \$tr' =_u \$tr \triangleright$
$(\bigsqcap\ P{\in}A \cdot peri_R\ (NCSP\ P))))$
proof –
have $(\bigsqcup\ P{\in}A \cdot [\$ok \mapsto_s true, \$ok' \mapsto_s true, \$wait \mapsto_s false, \$wait' \mapsto_s true] \dagger pre_R\ (NCSP\ P))$
$= (\bigsqcup\ P{\in}A \cdot pre_R\ (NCSP\ P))$
by (*rule USUP-cong*, *simp add*: *closure usubst unrest assms*)
thus *?thesis*
by (*simp add*: *rea-peri-RHS-design Healthy-Idempotent SRD-Idempotent usubst unrest assms*)
qed
also have ... $= R1\ ((\bigsqcup\ P{\in}A \cdot pre_R\ (NCSP\ P)) \Rightarrow_r$
$(\bigsqcup\ P{\in}A \cdot peri_R\ (NCSP\ P))$
$\triangleleft \$tr' =_u \$tr \triangleright$
$(\bigsqcap\ P{\in}A \cdot peri_R\ (NCSP\ P)))$
by (*simp add*: *R2c-rea-impl R2c-condr R2c-UINF R2c-preR R2c-periR R2c-tr′-minus-tr R2c-USUP closure*)
also have ... $= (((\bigsqcup\ P{\in}A \cdot pre_R\ (NCSP\ P)) \Rightarrow_r (\bigsqcup\ P{\in}A \cdot peri_R\ (NCSP\ P)))$
$\triangleleft \$tr' =_u \$tr \triangleright$
$((\bigsqcup\ P{\in}A \cdot pre_R\ (NCSP\ P)) \Rightarrow_r (\bigsqcap\ P{\in}A \cdot peri_R\ (NCSP\ P))))$
by (*simp add*: *R1-rea-impl R1-cond R1-USUP R1-UINF assms Healthy-if closure*, *rel-auto*)
also have ... $= (((\bigsqcup\ P{\in}A \cdot pre_R\ (NCSP\ P)) \Rightarrow_r (\bigsqcup\ P{\in}A \cdot peri_R\ (NCSP\ P)))$
$\triangleleft \$tr' =_u \$tr \triangleright$
$((\bigsqcap\ P{\in}A \cdot pre_R\ (NCSP\ P) \Rightarrow_r peri_R\ (NCSP\ P))))$
by (*simp add*: *UINF-rea-impl*[*THEN sym*])
also have ... $= (((\bigsqcup\ P{\in}A \cdot pre_R\ (NCSP\ P)) \Rightarrow_r (\bigsqcup\ P{\in}A \cdot peri_R\ (NCSP\ P)))$
$\triangleleft \$tr' =_u \$tr \triangleright$
$((\bigsqcap\ P{\in}A \cdot peri_R\ (NCSP\ P))))$
by (*simp add*: *SRD-peri-under-pre closure assms unrest*)
also have ... $= (((\bigsqcup\ P{\in}A \cdot pre_R\ P) \Rightarrow_r (\bigsqcup\ P{\in}A \cdot peri_R\ P))$
$\triangleleft \$tr' =_u \$tr \triangleright$
$((\bigsqcap\ P{\in}A \cdot peri_R\ P)))$

by (*simp add*: *UINF-healthy*[*OF assms*(*1*), *THEN sym*] *USUP-healthy*[*OF assms*(*1*), *THEN sym*])
finally show *?thesis* **.**
qed

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

lemma *postR-ExtChoice* [*rdes*]:
assumes $A \subseteq [\![NCSP]\!]_H$ $A \neq \{\}$
shows $post_R(ExtChoice\ A) = (\bigsqcap\ P{\in}A \cdot post_R\ P)$
proof −
have $post_R\ (ExtChoice\ A) = post_R\ (\mathbf{R}_s\ ((\bigsqcup\ P \in A \cdot pre_R\ P) \vdash$
$((\bigsqcup\ P \in A \cdot peri_R\ P) \triangleleft \$tr' =_u \$tr \triangleright (\bigsqcap\ P \in A \cdot peri_R\ P)) \diamond$
$(\bigsqcap\ P \in A \cdot post_R\ P)))$
by (*simp add*: *ExtChoice-tri-rdes-def closure assms*)

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

also have ... $= R1\ (R2c\ ((\bigsqcup\ P{\in}A \cdot pre_R\ (NCSP\ P)) \Rightarrow_r (\bigsqcap\ P{\in}A \cdot post_R\ (NCSP\ P))))$
proof −
have $(\bigsqcup\ P{\in}A \cdot [\$ok \mapsto_s true, \$ok' \mapsto_s true, \$wait \mapsto_s false, \$wait' \mapsto_s false] \dagger pre_R\ (NCSP\ P))$
$= (\bigsqcup\ P{\in}A \cdot pre_R\ (NCSP\ P))$
by (*rule USUP-cong*, *simp add*: *usubst closure unrest assms*)
thus *?thesis*
by (*simp add*: *rea-post-RHS-design Healthy-Idempotent SRD-Idempotent usubst unrest assms*)
qed
also have ... $= R1\ ((\bigsqcup\ P{\in}A \cdot pre_R\ (NCSP\ P)) \Rightarrow_r (\bigsqcap\ P{\in}A \cdot post_R\ (NCSP\ P)))$
by (*simp add*: *R2c-rea-impl R2c-condr R2c-UINF R2c-preR R2c-postR*
R2c-tr′-minus-tr R2c-USUP closure)
also from *assms*(*2*) **have** ... $= ((\bigsqcup\ P{\in}A \cdot pre_R\ (NCSP\ P)) \Rightarrow_r (\bigsqcap\ P{\in}A \cdot post_R\ (NCSP\ P)))$
by (*simp add*: *R1-rea-impl R1-cond R1-USUP R1-UINF assms Healthy-if closure*)
also have ... $= (\bigsqcap\ P{\in}A \cdot pre_R\ (NCSP\ P) \Rightarrow_r post_R\ (NCSP\ P))$
by (*simp add*: *UINF-rea-impl*)
also have ... $= (\bigsqcap\ P{\in}A \cdot post_R\ (NCSP\ P))$
by (*simp add*: *SRD-post-under-pre closure assms unrest*)
finally show *?thesis*
by (*simp add*: *UINF-healthy*[*OF assms*(*1*), *THEN sym*] *USUP-healthy*[*OF assms*(*1*), *THEN sym*])
qed

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

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

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

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

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

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

lemma *CSP4-ExtChoice*:
  assumes $A \subseteq [\![NCSP]\!]_H$
  shows *ExtChoice A is CSP4*
proof (*cases* $A = \{\}$)
  case *True* **thus** *?thesis*
    by (*simp add*: *ExtChoice-empty Healthy-def CSP4-def*, *simp add*: *Skip-is-CSP Stop-left-zero*)
next
  case *False*
  have *1*:$(\neg_r\ (\neg_r\ pre_R\ (ExtChoice\ A))\ ;;_h\ R1\ true) = pre_R\ (ExtChoice\ A)$
  proof −
    have $\bigwedge P.\ P \in A \Longrightarrow (\neg_r\ pre_R(P))\ ;;\ R1\ true = (\neg_r\ pre_R(P))$
      by (*simp add*: *NCSP-Healthy-subset-member NCSP-implies-NSRD NSRD-neg-pre-unit assms*)
    thus *?thesis*
     apply (*simp add*: *False preR-ExtChoice closure NCSP-set-unrest-pre-wait′ assms not-UINF seq-UINF-distr not-USUP*)
      apply (*rule USUP-cong*)
      apply (*simp add*: *rpred assms closure*)
      done
  qed
  have *2*: $\$st' \sharp peri_R\ (ExtChoice\ A)$

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

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

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

qed

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

7.5 Productivity and Guardedness

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

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

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

lemma *ExtChoice-Guarded* [*closure*]:
assumes $\bigwedge P.\ P \in A \Longrightarrow$ *Guarded P*
shows *Guarded* $(\lambda\ X.\ \square P{\in}A \cdot P(X))$
proof (*rule GuardedI*)
fix *X n*
have $\bigwedge\ Y.\ ((\square P{\in}A \cdot P\ Y) \wedge gvrt(n{+}1)) = ((\square P{\in}A \cdot (P\ Y \wedge gvrt(n{+}1))) \wedge gvrt(n{+}1))$
proof −
fix *Y*
let $?lhs = ((\square P{\in}A \cdot P\ Y) \wedge gvrt(n{+}1))$ **and** $?rhs = ((\square P{\in}A \cdot (P\ Y \wedge gvrt(n{+}1))) \wedge gvrt(n{+}1))$
have $a{:}?lhs[\![false/\$ok]\!] = ?rhs[\![false/\$ok]\!]$
by (*rel-auto*)
have $b{:}?lhs[\![true/\$ok]\!][\![true/\$wait]\!] = ?rhs[\![true/\$ok]\!][\![true/\$wait]\!]$
by (*rel-auto*)
have $c{:}?lhs[\![true/\$ok]\!][\![false/\$wait]\!] = ?rhs[\![true/\$ok]\!][\![false/\$wait]\!]$
by (*simp add*: *ExtChoice-def RHS-def R1-def R2c-def R2s-def R3h-def design-def usubst unrest*, *rel-blast*)
show $?lhs = ?rhs$
using *a b c*
by (*rule-tac bool-eq-splitI*[*of in-var ok*], *simp*, *rule-tac bool-eq-splitI*[*of in-var wait*], *simp-all*)
qed
moreover have $((\square P{\in}A \cdot (P\ X \wedge gvrt(n{+}1))) \wedge gvrt(n{+}1)) = ((\square P{\in}A \cdot (P\ (X \wedge gvrt(n)) \wedge$

gvrt(*n*+*1*))) ∧ *gvrt*(*n*+*1*))
  proof −
    have (□*P*∈*A* · (*P X* ∧ *gvrt*(*n*+*1*))) = (□*P*∈*A* · (*P* (*X* ∧ *gvrt*(*n*)) ∧ *gvrt*(*n*+*1*)))
    proof (*rule ExtChoice-cong*)
      fix *P* **assume** *P* ∈ *A*
      thus (*P X* ∧ *gvrt*(*n*+*1*)) = (*P* (*X* ∧ *gvrt*(*n*)) ∧ *gvrt*(*n*+*1*))
        using *Guarded-def assms* **by** *blast*
    qed
    thus *?thesis* **by** *simp*
  qed
  ultimately show ((□*P*∈*A* · *P X*) ∧ *gvrt*(*n*+*1*)) = ((□*P*∈*A* · (*P* (*X* ∧ *gvrt*(*n*)))) ∧ *gvrt*(*n*+*1*))
    by *simp*
qed

lemma *extChoice-Guarded* [*closure*]:
  assumes *Guarded P Guarded Q*
  shows *Guarded* (λ *X*. *P*(*X*) □ *Q*(*X*))
proof −
  have *Guarded* (λ *X*. □*F*∈{*P*,*Q*} · *F*(*X*))
    by (*rule ExtChoice-Guarded*, *auto simp add*: *assms*)
  thus *?thesis*
    by (*simp add*: *extChoice-def*)
qed

7.6 Algebraic laws

lemma *extChoice-comm*:
  P □ *Q* = *Q* □ *P*
  by (*unfold extChoice-def*, *simp add*: *insert-commute*)

lemma *extChoice-idem*:
  P is CSP ⟹ *P* □ *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* □ *Q* □ *R* = *P* □ (*Q* □ *R*)
proof −
  have $P \Box Q \Box R = \mathbf{R}_s(pre_R(P) \vdash cmt_R(P)) \Box \mathbf{R}_s(pre_R(Q) \vdash cmt_R(Q)) \Box \mathbf{R}_s(pre_R(R) \vdash cmt_R(R))$
    by (*simp add*: *SRD-reactive-design-alt assms*(*1*) *assms*(*2*) *assms*(*3*))
  also have ... =
    $\mathbf{R}_s$ (((*pre*$_R$ *P* ∧ *pre*$_R$ *Q*) ∧ *pre*$_R$ *R*) ⊢
        (((*cmt*$_R$ *P* ∧ *cmt*$_R$ *Q*) ◁ \$*tr*´ $=_u$ \$*tr* ∧ \$*wait*´ ▷ (*cmt*$_R$ *P* ∨ *cmt*$_R$ *Q*) ∧ *cmt*$_R$ *R*)
            ◁ \$*tr*´ $=_u$ \$*tr* ∧ \$*wait*´ ▷
          ((*cmt*$_R$ *P* ∧ *cmt*$_R$ *Q*) ◁ \$*tr*´ $=_u$ \$*tr* ∧ \$*wait*´ ▷ (*cmt*$_R$ *P* ∨ *cmt*$_R$ *Q*) ∨ *cmt*$_R$ *R*)))
    by (*simp add*: *extChoice-rdes unrest*)
  also have ... =
    $\mathbf{R}_s$ (((*pre*$_R$ *P* ∧ *pre*$_R$ *Q*) ∧ *pre*$_R$ *R*) ⊢
        (((*cmt*$_R$ *P* ∧ *cmt*$_R$ *Q*) ∧ *cmt*$_R$ *R*)
            ◁ \$*tr*´ $=_u$ \$*tr* ∧ \$*wait*´ ▷
          ((*cmt*$_R$ *P* ∨ *cmt*$_R$ *Q*) ∨ *cmt*$_R$ *R*)))
    by (*rule cong*[*of* $\mathbf{R}_s$ $\mathbf{R}_s$], *simp*, *rel-auto*)
  also have ... =
    $\mathbf{R}_s$ ((*pre*$_R$ *P* ∧ *pre*$_R$ *Q* ∧ *pre*$_R$ *R*) ⊢
        ((*cmt*$_R$ *P* ∧ (*cmt*$_R$ *Q* ∧ *cmt*$_R$ *R*))
            ◁ \$*tr*´ $=_u$ \$*tr* ∧ \$*wait*´ ▷

$(cmt_R\ P \vee (cmt_R\ Q \vee cmt_R\ R))))$
by (*simp add*: *conj-assoc disj-assoc*)
also have ... =
$\mathbf{R}_s\ ((pre_R\ P \wedge pre_R\ Q \wedge pre_R\ R) \vdash$
$((cmt_R\ P \wedge (cmt_R\ Q \wedge cmt_R\ R) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (cmt_R\ Q \vee cmt_R\ R))$
$\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
$(cmt_R\ P \vee (cmt_R\ Q \wedge cmt_R\ R) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (cmt_R\ Q \vee cmt_R\ R))))$
by (*rule cong*[*of* $\mathbf{R}_s$ $\mathbf{R}_s$], *simp*, *rel-auto*)
also have ... = $\mathbf{R}_s(pre_R(P) \vdash cmt_R(P)) \Box (\mathbf{R}_s(pre_R(Q) \vdash cmt_R(Q)) \Box \mathbf{R}_s(pre_R(R) \vdash cmt_R(R)))$
by (*simp add*: *extChoice-rdes unrest*)
also have ... = $P \Box (Q \Box R)$
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* $\Box$ $Q = Q$
using *assms*
proof −
have *Stop* $\Box\ Q = \mathbf{R}_s\ (true \vdash (\$tr' =_u \$tr \wedge \$wait')) \Box \mathbf{R}_s(pre_R(Q) \vdash cmt_R(Q))$
by (*simp add*: *Stop-def SRD-reactive-design-alt assms*)
also have ... = $\mathbf{R}_s\ (pre_R\ Q \vdash (((\$tr' =_u \$tr \wedge \$wait') \wedge cmt_R\ Q) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (\$tr'$
$=_u \$tr \wedge \$wait' \vee cmt_R\ Q)))$
by (*simp add*: *extChoice-rdes unrest*)
also have ... = $\mathbf{R}_s\ (pre_R\ Q \vdash (cmt_R\ Q \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright cmt_R\ Q))$
by (*metis* (*no-types*, *lifting*) *cond-def eq-upred-sym neg-conj-cancel1 utp-pred-laws.inf.left-idem*)
also have ... = $\mathbf{R}_s\ (pre_R\ Q \vdash cmt_R\ Q)$
by (*simp add*: *cond-idem*)
also have ... = Q
by (*simp add*: *SRD-reactive-design-alt assms*)
finally show *?thesis* **.**
qed

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

lemma *extChoice-Dist*:
assumes *P is CSP* $S \subseteq [\![CSP]\!]_H$ $S \neq \{\}$
shows $P \Box (\bigsqcap\ S) = (\bigsqcap\ Q{\in}S.\ P \Box Q)$
proof −
let $?S1 = pre_R$ ‘ S **and** $?S2 = cmt_R$ ‘ S
have $P \Box (\bigsqcap\ S) = P \Box (\bigsqcap\ Q{\in}S \cdot \mathbf{R}_s(pre_R(Q) \vdash cmt_R(Q)))$

by (*simp add*: *SRD-as-reactive-design*[*THEN sym*] *Healthy-SUPREMUM UINF-as-Sup-collect assms*)
also have ... = $\mathbf{R}_s(pre_R(P) \vdash cmt_R(P)) \Box \mathbf{R}_s((\bigsqcup\ Q \in S \cdot pre_R(Q)) \vdash (\bigsqcap\ Q \in S \cdot cmt_R(Q)))$
by (*simp add*: *RHS-design-USUP SRD-reactive-design-alt assms*)
also have ... = $\mathbf{R}_s\ ((pre_R(P) \wedge (\bigsqcup\ Q \in S \cdot pre_R(Q))) \vdash$
$((cmt_R(P) \wedge (\bigsqcap\ Q \in S \cdot cmt_R(Q)))$
$\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
$(cmt_R(P) \vee (\bigsqcap\ Q \in S \cdot cmt_R(Q)))))$
by (*simp add*: *extChoice-rdes unrest*)
also have ... = $\mathbf{R}_s\ ((\bigsqcup\ Q{\in}S \cdot pre_R\ P \wedge pre_R\ Q) \vdash$
$(\bigsqcap\ Q{\in}S \cdot (cmt_R\ P \wedge cmt_R\ Q) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (cmt_R\ P \vee cmt_R\ Q)))$
by (*simp add*: *conj-USUP-dist conj-UINF-dist disj-UINF-dist cond-UINF-dist assms*)
also have ... = $(\bigsqcap\ Q \in S \cdot \mathbf{R}_s\ ((pre_R\ P \wedge pre_R\ Q) \vdash$
$((cmt_R\ P \wedge cmt_R\ Q) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (cmt_R\ P \vee cmt_R\ Q))))$
by (*simp add*: *assms RHS-design-USUP*)
also have ... = $(\bigsqcap\ Q{\in}S \cdot \mathbf{R}_s(pre_R(P) \vdash cmt_R(P)) \Box \mathbf{R}_s(pre_R(Q) \vdash cmt_R(Q)))$
by (*simp add*: *extChoice-rdes unrest*)
also have ... = $(\bigsqcap\ Q{\in}S.\ P \Box CSP(Q))$
by (*simp add*: *UINF-as-Sup-collect*, *metis* (*no-types*, *lifting*) *Healthy-if SRD-as-reactive-design assms*(*1*))
also have ... = $(\bigsqcap\ Q{\in}S.\ P \Box Q)$
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 $P \Box (Q \sqcap R) = (P \Box Q) \sqcap (P \Box R)$
using *assms extChoice-Dist*[*of P* $\{Q, R\}$] **by** *simp*

lemma *ExtChoice-seq-distr*:
assumes $\bigwedge\ i.\ i \in A \Longrightarrow P\ i$ *is PCSP Q is NCSP*
shows $(\Box\ i{\in}A \cdot P\ i)\ ;;\ Q = (\Box\ i{\in}A \cdot P\ i\ ;;\ Q)$
proof (*cases* $A = \{\}$)
case *True*
then show *?thesis*
by (*simp add*: *ExtChoice-empty NCSP-implies-CSP Stop-left-zero assms*(*2*))
next
case *False*
show *?thesis*
proof −
have *1*:$(\Box\ i{\in}A \cdot P\ i) = (\Box\ i{\in}A \cdot (\mathbf{R}_s\ ((pre_R\ (P\ i)) \vdash peri_R\ (P\ i) \diamond (R4(post_R\ (P\ i))))))$
(**is** *?X* = *?Y*)
by (*rule ExtChoice-cong*, *metis* (*no-types*, *hide-lams*) *R4-def Healthy-if NCSP-implies-CSP PCSP-implies-NCSP Productive-form assms*(*1*) *comp-apply*)
have *2*:$(\Box\ i{\in}A \cdot P\ i\ ;;\ Q) = (\Box\ i{\in}A \cdot (\mathbf{R}_s\ ((pre_R\ (P\ i)) \vdash peri_R\ (P\ i) \diamond (R4(post_R\ (P\ i)))))\ ;;\ Q)$
(**is** *?X* = *?Y*)
by (*rule ExtChoice-cong*, *metis* (*no-types*, *hide-lams*) *R4-def Healthy-if NCSP-implies-CSP PCSP-implies-NCSP Productive-form assms*(*1*) *comp-apply*)
show *?thesis*
by (*simp add*: *1 2*, *rdes-eq cls*: *assms False cong*: *ExtChoice-cong USUP-cong*)
qed
qed

lemma *extChoice-seq-distr*:
assumes *P is PCSP Q is PCSP R is NCSP*

shows $(P \Box Q) ;; R = (P ;; R \Box Q ;; R)$
by (*rdes-eq cls*: *assms*)

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

end

8 Circus and CSP Actions

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

8.1 Conditionals

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

8.2 Guarded commands

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

8.3 Alternation

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

8.4 While Loops

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

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

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

8.5 Assignment

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

syntax
  -assigns-csp :: *svids* $\Rightarrow$ *uexprs* $\Rightarrow$ *logic* $('(\text{-}') :=_C '(\text{-}'))$
  -assigns-csp :: *svids* $\Rightarrow$ *uexprs* $\Rightarrow$ *logic* (**infixr** $:=_C$ *90*)

translations
  -assigns-csp xs vs => *CONST AssignsCSP* (*-mk-usubst* (*CONST id*) *xs vs*)
  -assigns-csp x v <= *CONST AssignsCSP* (*CONST subst-upd* (*CONST id*) *x v*)
  -assigns-csp x v <= *-assigns-csp* (*-spvar x*) *v*
  x,*y* $:=_C$ *u*,*v* <= *CONST AssignsCSP* (*CONST subst-upd* (*CONST subst-upd* (*CONST id*) (*CONST svar x*) *u*) (*CONST svar y*) *v*)

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

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

lemma *postR-AssignsCSP* [*rdes*]: $post_R(\langle\sigma\rangle_C) = \Phi(true,\sigma,\langle\rangle)$
  by (*rel-auto*)

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

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

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

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

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

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

8.6 Assignment with update

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

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

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

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

translations
  $x[k] :=_C v$ == *CONST AssignCSP-update* (*CONST udom*) (*CONST uupd*) $x\ k\ v$

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

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

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

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

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

8.7 State abstraction

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

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

8.8 Assumptions

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

8.9 Guards

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

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

lemma *Guard-rdes-def* [*rdes-def*]:
  assumes P *is RR* Q *is RR* R *is RR*
  shows $g \&_u \mathbf{R}_s(P \vdash Q \diamond R) = \mathbf{R}_s(([g]_{S<} \Rightarrow_r P) \vdash (Q \triangleleft g \triangleright_R (\mathcal{E}(true,\langle\rangle,\{\}_u))) \diamond ([g]_{S<} \wedge R))$
  by (*simp add*: *Guard-tri-design rdes assms*, *rel-auto*)

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

by (*simp add*: *rdes-export-cmt*)
also have ... = $\mathbf{R}_s((\lceil g\rceil_{S<} \Rightarrow_r R1(R2c(pre_s \dagger P))) \vdash cmt_s \dagger (\lceil g\rceil_{S<} \wedge (P \Rightarrow Q) \vee \lceil \neg g\rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
by (*simp add*: *usubst*)
also have ... = $\mathbf{R}_s((\lceil g\rceil_{S<} \Rightarrow_r R1(R2c(pre_s \dagger P))) \vdash (\lceil g\rceil_{S<} \wedge (P \Rightarrow Q) \vee \lceil \neg g\rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
by (*simp add*: *rdes-export-cmt*)
also from *assms* **have** ... = $\mathbf{R}_s((\lceil g\rceil_{S<} \Rightarrow_r (pre_s \dagger P)) \vdash (\lceil g\rceil_{S<} \wedge (P \Rightarrow Q) \vee \lceil \neg g\rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
by (*rel-auto*)
also have ... = $\mathbf{R}_s((\lceil g\rceil_{S<} \Rightarrow_r pre_s \dagger P)\llbracket true,false/\$ok,\$wait\rrbracket \vdash (\lceil g\rceil_{S<} \wedge (P \Rightarrow Q) \vee \lceil \neg g\rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
by (*simp add*: *rdes-export-pre*)
also from *assms* **have** ... = $\mathbf{R}_s((\lceil g\rceil_{S<} \Rightarrow_r P)\llbracket true,false/\$ok,\$wait\rrbracket \vdash (\lceil g\rceil_{S<} \wedge (P \Rightarrow Q) \vee \lceil \neg g\rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
by (*rel-auto*)
also from *assms* **have** ... = $\mathbf{R}_s((\lceil g\rceil_{S<} \Rightarrow_r P) \vdash (\lceil g\rceil_{S<} \wedge (P \Rightarrow Q) \vee \lceil \neg g\rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
by (*simp add*: *rdes-export-pre*)
also have ... = $\mathbf{R}_s((\lceil g\rceil_{S<} \Rightarrow_r P) \vdash (\lceil g\rceil_{S<} \wedge Q \vee \lceil \neg g\rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
by (*rule cong*[*of* $\mathbf{R}_s$ $\mathbf{R}_s$], *simp*, *rel-auto*)
finally show *?thesis* **.**
qed

lemma *CSP-Guard* [*closure*]: $b \,\&_u\, P$ *is CSP*
by (*simp add*: *GuardCSP-def*, *rule RHS-design-is-SRD*, *simp-all add*: *unrest*)

lemma *preR-Guard* [*rdes*]: *P is CSP* $\Longrightarrow pre_R(b \,\&_u\, P) = ([b]_{S<} \Rightarrow_r pre_R\ P)$
by (*simp add*: *Guard-tri-design rea-pre-RHS-design usubst unrest R2c-preR R2c-lift-state-pre R2c-rea-impl R1-rea-impl R1-preR Healthy-if*, *rel-auto*)

lemma *periR-Guard* [*rdes*]:
assumes *P is NCSP*
shows $peri_R(b \,\&_u\, P) = (peri_R\ P \triangleleft b \triangleright_R \mathcal{E}(true,\langle\rangle,\{\}_u))$
proof −
have $peri_R(b \,\&_u\, P) = ((\lceil b\rceil_{S<} \Rightarrow_r pre_R\ P) \Rightarrow_r (peri_R\ P \triangleleft \lceil b\rceil_{S<} \triangleright (\$tr' =_u \$tr)))$
by (*simp add*: *assms Guard-tri-design rea-peri-RHS-design usubst unrest R1-rea-impl R2c-rea-not R2c-rea-impl R2c-preR R2c-periR R2c-tr′-minus-tr R2c-lift-state-pre R2c-condr closure Healthy-if R1-cond R1-tr′-eq-tr*)
also have ... = $((pre_R\ P \Rightarrow_r peri_R\ P) \triangleleft \lceil b\rceil_{S<} \triangleright (\$tr' =_u \$tr))$
by (*rel-auto*)
also have ... = $(peri_R\ P \triangleleft \lceil b\rceil_{S<} \triangleright (\$tr' =_u \$tr))$
by (*simp add*: *SRD-peri-under-pre add*: *unrest closure assms*)
finally show *?thesis*
by *rel-auto*
qed

lemma *postR-Guard* [*rdes*]:
assumes *P is NCSP*
shows $post_R(b \,\&_u\, P) = ([b]_{S<} \wedge post_R\ P)$
proof −
have $post_R(b \,\&_u\, P) = ((\lceil b\rceil_{S<} \Rightarrow_r pre_R\ P) \Rightarrow_r (\lceil b\rceil_{S<} \wedge post_R\ P))$
by (*simp add*: *Guard-tri-design rea-post-RHS-design usubst unrest R2c-rea-not R2c-and R2c-rea-impl R2c-preR R2c-postR R2c-tr′-minus-tr R2c-lift-state-pre R2c-condr R1-rea-impl R1-extend-conj′ R1-post-SRD closure assms*)

also have ... = $(\lceil b \rceil_{S<} \wedge (pre_R\ P \Rightarrow_r post_R\ P))$
  by (*rel-auto*)
also have ... = $(\lceil b \rceil_{S<} \wedge post_R\ P)$
  by (*simp add*: *SRD-post-under-pre add*: *unrest closure assms*)
also have ... = $([b]_{S<} \wedge post_R\ P)$
  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 $b\ \&_u\ P$ *is CSP3*
proof −
from *assms* **have** *1*:$\$ref \sharp P[\![false/\$wait]\!]$
  by (*simp add*: *CSP-Guard CSP3-iff*)
hence $\$ref \sharp pre_R\ (P[\![0/\$tr]\!])$ $\$ref \sharp pre_R\ P$ $\$ref \sharp cmt_R\ P$
  by (*pred-blast*)+
hence $\$ref \sharp (b\ \&_u\ P)[\![false/\$wait]\!]$
  by (*simp add*: *CSP3-iff GuardCSP-def RHS-def R1-def R2c-def R2s-def R3h-def design-def unrest usubst*)
thus *?thesis*
  by (*metis CSP3-intro CSP-Guard*)
qed

lemma *CSP4-Guard* [*closure*]:
assumes *P is NCSP*
shows $b\ \&_u\ P$ *is CSP4*
proof (*rule CSP4-tri-intro*[*OF CSP-Guard*])
show $(\neg_r\ pre_R\ (b\ \&_u\ P))$;; *R1 true* = $(\neg_r\ pre_R\ (b\ \&_u\ P))$
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*:... = $([b]_{S<} \wedge ((\neg_r\ pre_R\ P)$;; *R1 true*))
      by (*simp add*: *a*)
    also have *3*:... = $(\neg_r\ ([b]_{S<} \Rightarrow_r pre_R\ P))$;; *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 $\$st' \sharp peri_R\ (b\ \&_u\ P)$
  by (*simp add*: *preR-Guard periR-Guard NSRD-CSP4-intro closure assms unrest*)
show $\$ref' \sharp post_R\ (b\ \&_u\ P)$
  by (*simp add*: *preR-Guard postR-Guard NSRD-CSP4-intro closure assms unrest*)
qed

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

have P *is CSP*
using *NCSP-implies-CSP assms* **by** *blast*
then show *?thesis*
by (*metis* (*no-types*) *CSP3-Guard CSP3-commutes-CSP4 CSP4-Guard CSP4-Idempotent CSP-Guard Healthy-Idempotent Healthy-def NCSP-def assms comp-apply*)
qed

lemma *Productive-Guard* [*closure*]:
assumes P *is CSP* P *is Productive* $\$wait' \sharp pre_R(P)$
shows $b \; \&_u \; P$ *is Productive*
proof −
have $b \; \&_u \; P = b \; \&_u \; \mathbf{R}_s(pre_R(P) \vdash peri_R(P) \diamond (post_R(P) \wedge \$tr <_u \$tr'))$
by (*metis Healthy-def Productive-form assms*(*1*) *assms*(*2*))
also have ... =
$\mathbf{R}_s \; ((\lceil b \rceil_{S<} \Rightarrow_r pre_R \; P) \vdash$
$((pre_R \; P \Rightarrow_r peri_R \; P) \triangleleft \lceil b \rceil_{S<} \triangleright (\$tr' =_u \$tr)) \diamond (\lceil b \rceil_{S<} \wedge (pre_R \; P \Rightarrow_r post_R \; P \wedge \$tr' >_u \$tr)))$
by (*simp add*: *Guard-tri-design rea-pre-RHS-design rea-peri-RHS-design rea-post-RHS-design unrest assms*
usubst R1-preR Healthy-if R1-rea-impl R1-peri-SRD R1-extend-conj' R2c-preR R2c-not R2c-rea-impl

R2c-periR R2c-postR R2c-and R2c-tr-less-tr' R1-tr-less-tr')
also have ... = $\mathbf{R}_s \; ((\lceil b \rceil_{S<} \Rightarrow_r pre_R \; P) \vdash (peri_R \; P \triangleleft \lceil b \rceil_{S<} \triangleright (\$tr' =_u \$tr)) \diamond ((\lceil b \rceil_{S<} \wedge post_R \; P) \wedge \$tr' >_u \$tr))$
by (*rel-auto*)
also have ... = $Productive(b \; \&_u \; P)$
by (*simp add*: *Productive-def Guard-tri-design RHS-tri-design-par unrest*)
finally show *?thesis*
by (*simp add*: *Healthy-def'*)
qed

8.10 Basic events

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

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

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

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

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

lemma *DoAct-unrests* [*unrest*]:
$\$ok \sharp do_u(a)$ $\$wait \sharp do_u(a)$

by (*pred-auto*)+

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

lemma *CSP-DoCSP* [*closure*]: $do_C(a)$ *is CSP*
by (*simp add*: *DoCSP-def* do_u*-def RHS-design-is-SRD unrest*)

lemma *preR-DoCSP* [*rdes*]: $pre_R(do_C(a)) = true_r$
by (*simp add*: *DoCSP-def rea-pre-RHS-design unrest usubst R2c-true*)

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

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

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

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

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

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

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

lemma *wp-rea-DoCSP-lemma*:
fixes *P* :: ($'\sigma$, $'\varphi$) *action*
assumes $\$ok \sharp P$ $\$wait \sharp P$
shows $(\$tr' =_u \$tr\ \hat{}_u\ \langle\lceil a\rceil_{S<}\rangle \wedge \$st' =_u \$st)\ ;;\ P = (\exists\ \$ref \cdot P[\![\$tr\ \hat{}_u\ \langle\lceil a\rceil_{S<}\rangle/\$tr]\!])$
using *assms*
by (*rel-auto*, *meson*)

lemma *wp-rea-DoCSP*:
assumes *P is NCSP*
shows $(\$tr' =_u \$tr\ \hat{}_u\ \langle\lceil a\rceil_{S<}\rangle \wedge \$st' =_u \$st)\ wp_r\ pre_R\ P =$

$(\neg_r\ (\neg_r\ pre_R\ P)[\![\$tr\ \hat{}_u\ \langle\lceil a\rceil_{S<}\rangle/\$tr]\!])$
by (*simp add*: *wp-rea-def wp-rea-DoCSP-lemma unrest usubst ex-unrest assms closure*)

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

8.11 Event prefix

definition *PrefixCSP* ::
$('\varphi, '\sigma)$ *uexpr* $\Rightarrow$
$('\sigma, '\varphi)$ *action* $\Rightarrow$
$('\sigma, '\varphi)$ *action* (- $\to_C$ - [*81*, *80*] *80*) **where**
[*upred-defs*]: *PrefixCSP a P* = $(do_C(a)$;; $CSP(P))$

abbreviation *OutputCSP c v P* $\equiv$ *PrefixCSP* $(c{\cdot}v)_u$ *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* $\Longrightarrow$ *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* $\Longrightarrow$ *PrefixCSP a P is PCSP*
by (*simp add*: *Healthy-comp NCSP-PrefixCSP Productive-PrefixCSP*)

lemma *PrefixCSP-Guarded* [*closure*]: *Guarded* (*PrefixCSP a*)
proof −
have *PrefixCSP a* = $(\lambda\ X.\ do_C(a)$;; $CSP(X))$
by (*simp add*: *fun-eq-iff PrefixCSP-def*)
thus *?thesis*
using *Guarded-if-Productive NCSP-DoCSP NCSP-implies-NSRD Productive-DoCSP* **by** *auto*
qed

lemma *PrefixCSP-type* [*closure*]: *PrefixCSP a* $\in [\![H]\!]_H \to [\![CSP]\!]_H$
using *CSP-PrefixCSP* **by** *blast*

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

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

lemma *PrefixCSP-RHS-tri-lemma1*:
R1 (*R2s* ($\$tr' =_u \$tr \ \hat{}_u \ \langle\lceil a\rceil_{S<}\rangle \wedge \lceil II\rceil_R$)) = ($\$tr' =_u \$tr \ \hat{}_u \ \langle\lceil a\rceil_{S<}\rangle \wedge \lceil II\rceil_R$)
by (*rel-auto*)

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

lemma *tr-extend-seqr*:
fixes P :: ($'\sigma$, $'\varphi$) *action*
assumes $\$ok \sharp P$ $\$wait \sharp P$ $\$ref \sharp P$
shows ($\$tr' =_u \$tr \ \hat{}_u \ \langle\lceil a\rceil_{S<}\rangle \wedge \$st' =_u \$st$) ;; $P = P[\![\$tr \ \hat{}_u \ \langle\lceil a\rceil_{S<}\rangle/\$tr]\!]$
using *assms* **by** (*simp add*: *wp-rea-DoCSP-lemma assms unrest ex-unrest*)

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

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

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

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

by (*simp add*: *SRD-post-under-pre assms closure unrest*)
also have ... = $(post_R\ P)[\![\langle a\rangle]\!]_t$
by (*rel-auto*)
finally show *?thesis* **.**
qed

lemma *PrefixCSP-RHS-tri*:
assumes *P is NCSP*
shows *PrefixCSP a P* = $\mathbf{R}_s\ ((\mathcal{I}(true,\langle a\rangle) \Rightarrow_r pre_R\ P[\![\langle a\rangle]\!]_t) \vdash (\mathcal{E}(true,\langle\rangle,\{a\}_u) \vee peri_R\ P[\![\langle a\rangle]\!]_t) \diamond post_R\ P[\![\langle a\rangle]\!]_t)$
by (*simp add*: *PrefixCSP-def Healthy-if unrest assms closure NSRD-composition-wp rdes rpred usubst wp*)

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

lemma *PrefixCSP-rdes-def-1* [*rdes-def*]:
assumes *P is CRC Q is CRR R is CRR*
$\$st' \sharp Q$ $\$ref' \sharp R$
shows *PrefixCSP a* $(\mathbf{R}_s(P \vdash Q \diamond R)) = \mathbf{R}_s((\mathcal{I}(true,\langle a\rangle) \Rightarrow_r P[\![\langle a\rangle]\!]_t) \vdash (\mathcal{E}(true,\langle\rangle,\{a\}_u) \vee Q[\![\langle a\rangle]\!]_t) \diamond R[\![\langle a\rangle]\!]_t)$
apply (*subst PrefixCSP-RHS-tri*)
apply (*rule NCSP-rdes-intro*)
apply (*simp-all add*: *assms rdes closure*)
apply (*rel-auto*)
done

lemma *PrefixCSP-rdes-def-2*:
assumes *P is CRC Q is CRR R is CRR*
$\$st' \sharp Q$ $\$ref' \sharp R$
shows *PrefixCSP a* $(\mathbf{R}_s(P \vdash Q \diamond R)) = \mathbf{R}_s((\mathcal{I}(true,\langle a\rangle) \Rightarrow_r P[\![\langle a\rangle]\!]_t) \vdash (\mathcal{E}(true,\langle\rangle,\{a\}_u) \vee (P \wedge Q)[\![\langle a\rangle]\!]_t) \diamond (P \wedge R)[\![\langle a\rangle]\!]_t)$
apply (*subst PrefixCSP-RHS-tri*)
apply (*rule NCSP-rdes-intro*)
apply (*simp-all add*: *assms rdes closure*)
apply (*rel-auto*)
done

8.12 Guarded external choice

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

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

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

lemma *GuardedChoiceCSP* [*rdes-def*]:
assumes $\bigwedge x.\ P(x)$ *is NCSP* $A \neq \{\}$
shows $(\Box\ x{\in}A \rightarrow P(x))$ =
$\mathbf{R}_s\ ((\bigsqcup\ x \in A \cdot \mathcal{I}(true,\langle\ll x\gg\rangle) \Rightarrow_r pre_R\ (P\ x)[\![\langle\ll x\gg\rangle]\!]_t) \vdash$
$((\bigsqcup\ x \in A \cdot \mathcal{E}(true,\langle\rangle,\{\ll x\gg\}_u)) \triangleleft \$tr' =_u \$tr \triangleright (\bigsqcap\ x \in A \cdot peri_R\ (P\ x)[\![\langle\ll x\gg\rangle]\!]_t)) \diamond$
$(\bigsqcap\ x \in A \cdot post_R\ (P\ x)[\![\langle\ll x\gg\rangle]\!]_t))$
by (*simp add*: *PrefixCSP-RHS-tri assms ExtChoice-tri-rdes closure unrest*, *rel-auto*)

8.13 Input prefix

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

definition *InputVarCSP* :: $('a, '\vartheta)$ *chan* $\Rightarrow$ $('a \Rightarrow '\sigma$ *upred*$)$ $\Rightarrow$ $('a \Longrightarrow '\sigma)$ $\Rightarrow$ $('\sigma, '\vartheta)$ *action* $\Rightarrow$ $('\sigma, '\vartheta)$ *action* **where**
$InputVarCSP\ c\ A\ x\ P = InputCSP\ c\ A\ (\lambda\ v.\ \langle[x \mapsto_s \ll v\gg]\rangle_C)\ ;;\ CSP(P)$

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

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

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

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

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

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

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

lemma *InputCSP-rdes-def* [*rdes-def*]:
  assumes $\bigwedge\ v.\ P(v)$ *is CRC* $\bigwedge\ v.\ Q(v)$ *is CRR* $\bigwedge\ v.\ R(v)$ *is CRR*
    $\bigwedge\ v.\ \$st' \sharp\ Q(v)\ \bigwedge\ v.\ \$ref' \sharp\ R(v)$
  shows $InputCSP\ a\ A\ (\lambda\ v.\ \mathbf{R}_s(P(v) \vdash Q(v) \diamond R(v))) =$

$\mathbf{R}_s((\bigsqcup\ v \cdot ([A(v)]_{S<} \Rightarrow_r \mathcal{I}(true,\langle(a\cdot\ll v\gg)_u\rangle) \Rightarrow_r (P\ v)[\![\langle(a\cdot\ll v\gg)_u\rangle]\!]_t))$
$\vdash (((\bigsqcup\ x \cdot [A(x)]_{S<} \Rightarrow_r \mathcal{E}(true,\ \langle\rangle,\ \{(a\cdot\ll x\gg)_u\}_u)))$
$\triangleleft \$tr\acute{} =_u \$tr \triangleright$
$(\bigsqcap\ x \cdot [A(x)]_{S<} \wedge (P\ x \wedge Q\ x)[\![\langle(a\cdot\ll x\gg)_u\rangle]\!]_t))$
$\diamond (\bigsqcap\ x \cdot [A\ x]_{S<} \wedge (P\ x \wedge R\ x)[\![\langle(a\cdot\ll x\gg)_u\rangle]\!]_t))$ (**is** *?lhs* = *?rhs*)
proof –
have *1*:pre_R(*?lhs*) = $(\bigsqcup\ v \cdot [A\ v]_{S<} \Rightarrow_r \mathcal{I}(true,\langle(a\cdot\ll v\gg)_u\rangle) \Rightarrow_r P\ v[\![\langle(a\cdot\ll v\gg)_u\rangle]\!]_t)$ (**is** - = *?A*)
by (*simp add*: *rdes NCSP-rdes-intro assms conj-comm closure*)
have *2*:$peri_R$(*?lhs*) = $(\bigsqcup\ x \cdot [A\ x]_{S<} \Rightarrow_r \mathcal{E}(true,\langle\rangle,\ \{(a\cdot\ll x\gg)_u\}_u)) \triangleleft \$tr\acute{} =_u \$tr \triangleright (\bigsqcap\ x \cdot [A\ x]_{S<} \wedge (P\ x \Rightarrow_r Q\ x)[\![\langle(a\cdot\ll x\gg)_u\rangle]\!]_t)$
(**is** - = *?B*)
by (*simp add*: *rdes NCSP-rdes-intro assms closure*)
have *3*:$post_R$(*?lhs*) = $(\bigsqcap\ x \cdot [A\ x]_{S<} \wedge (P\ x \Rightarrow_r R\ x)[\![\langle(a\cdot\ll x\gg)_u\rangle]\!]_t)$
(**is** - = *?C*)
by (*simp add*: *rdes NCSP-rdes-intro assms closure*)
have *?lhs* = $\mathbf{R}_s$(*?A* ⊢ *?B* ⋄ *?C*)
by (*subst SRD-reactive-tri-design*[*THEN sym*], *simp-all add*: *closure 1 2 3*)
also have ... = *?rhs*
by (*rel-auto*)
finally show *?thesis* **.**
qed

8.14 Algebraic laws

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

lemma *AssignsCSP-id*: $\langle id\rangle_C = Skip$
by (*rel-auto*)

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

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

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

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

lemma *Guard-expansion*:
assumes *P is NCSP*
shows $(g_1 \vee g_2) \&_u P = (g_1 \&_u P) \Box (g_2 \&_u P)$
by (*rdes-eq cls*: *assms*; *simp add*: *le-less*)

lemma *Conditional-as-Guard*:
assumes *P is NCSP Q is NCSP*
shows $P \triangleleft b \triangleright_R Q = b \&_u P \Box (\neg\ b) \&_u Q$

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

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

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

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

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

lemma *GuardedChoiceCSP-dist*:
assumes $\bigwedge i.\ i{\in}A \Longrightarrow P(i)$ *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
$\bigwedge i.\ i \in A \Longrightarrow P(i)$ *is NCSP Q is NCSP*
$\bigwedge i\ j.\ [\![\ i \in A;\ j \in A;\ i \neq j\ ]\!] \Longrightarrow (g\ i \wedge g\ j) = false$
shows $(if_R\ i{\in}A \cdot g(i) \rightarrow P(i)\ else\ Q\ fi) =$
$(\square\ i{\in}A \cdot g(i)\ \&_u\ P(i)) \square (\bigwedge\ i{\in}A \cdot \neg\ g(i))\ \&_u\ Q$
proof (*cases* $A = \{\}$)
case *True*
then show *?thesis* **by** (*simp add*: *ExtChoice-empty extChoice-Stop closure assms*)
next
case *False*
show *?thesis*

proof −
have *1*:$(\bigsqcap\ i \in A \cdot g\ i \rightarrow_R P\ i) = (\bigsqcap\ i \in A \cdot g\ i \rightarrow_R \mathbf{R}_s(pre_R(P\ i) \vdash peri_R(P\ i) \diamond post_R(P\ i)))$
by (*rule UINF-cong*, *simp add*: *NCSP-implies-CSP SRD-reactive-tri-design assms*(*1*))
have *2*:$(\square\ i{\in}A \cdot g(i)\ \&_u\ P(i)) = (\square\ i{\in}A \cdot g(i)\ \&_u\ \mathbf{R}_s(pre_R(P\ i) \vdash peri_R(P\ i) \diamond post_R(P\ i)))$
by (*rule ExtChoice-cong*, *simp add*: *NCSP-implies-NSRD NSRD-is-SRD SRD-reactive-tri-design assms*(*1*))
from *assms*(*3*) **show** *?thesis*
by (*simp add*: *AlternateR-def 1 2*)
(*rdes-eq cls*: *assms*(*1*−*2*) *simps*: *False cong*: *UINF-cong ExtChoice-cong*)
qed

qed

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

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

end

9 Syntax and Translations for Event Prefix

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

syntax
  -simple-prefix :: *logic* ⇒ *logic* ⇒ *logic* (- ⟶ - [*81*, *80*] *80*)

translations
  a ⟶ *P* == *CONST PrefixCSP* ≪*a*≫ *P*

We next configure a syntax for mixed prefixes.

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

syntax *-end-prefix* :: *prefix-elem'* ⇒ *mixed-prefix'* (-)

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

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

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

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

Output Prefix: . . .![v]e

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

syntax *-output-prefix* :: ('a, 'σ) *uexpr* ⇒ *prefix-elem'* (!'(-'))
syntax *-output-prefix* :: ('a, 'σ) *uexpr* ⇒ *prefix-elem'* (.'(-'))

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

syntax
  -prefix-aux :: *pttrn* ⇒ *logic* ⇒ *prefix-elem'*

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

syntax *-mixed-prefix* :: *prefix-elem'* ⇒ *mixed-prefix'* ⇒ *mixed-prefix'* (--)

syntax
  -prefix-action ::
  ('a, 'ε) *chan* ⇒ *mixed-prefix'* ⇒ ('σ, 'ε) *action* ⇒ ('σ, 'ε) *action*
  ((-- ⟶/ -) [*81*, *81*, *80*] *80*)

Syntax translations

definition *lconj* :: $('a \Rightarrow '\alpha\ upred) \Rightarrow ('b \Rightarrow '\alpha\ upred) \Rightarrow ('a \times 'b \Rightarrow '\alpha\ upred)$ (**infixr** $\wedge_l$ *35*)
where [*upred-defs*]: $(P \wedge_l Q) \equiv (\lambda\ (x,y).\ P\ x \wedge Q\ y)$

definition *outp-constraint* (**infix** $=_o$ *60*) **where**
[*upred-defs*]: *outp-constraint* $v \equiv (\lambda\ x.\ \ll x \gg =_u v)$

translations
  -simple-input-prefix x ⇌ *-input-prefix x true*
  -mixed-prefix (*-input-prefix x P*) (*-prefix-aux y Q*) ⇀
  -prefix-aux (*-pattern x y*) $((\lambda\ x.\ P) \wedge_l Q)$
  -mixed-prefix (*-output-prefix P*) (*-prefix-aux y Q*) ⇀
  -prefix-aux (*-pattern -idtdummy y*) ((*CONST outp-constraint P*) $\wedge_l$ *Q*)
  -end-prefix (*-input-prefix x P*) ⇀ *-prefix-aux x* $(\lambda\ x.\ P)$
  -end-prefix (*-output-prefix P*) ⇀ *-prefix-aux -idtdummy* (*CONST outp-constraint P*)
  -prefix-action c (*-prefix-aux x P*) *A* == (*CONST InputCSP*) *c P* $(\lambda x.\ A)$

Basic print translations; more work needed

translations
  -simple-input-prefix x <= *-input-prefix x true*
  -output-prefix v <= *-prefix-aux p* (*CONST outp-constraint v*)
  -output-prefix u (*-output-prefix v*)
    <= *-prefix-aux p* (λ(*x1*, *y1*). *CONST outp-constraint u x2* ∧ *CONST outp-constraint v y2*)
  -input-prefix x P <= *-prefix-aux v* $(\lambda x.\ P)$
  $x!(v) \rightarrow P$ <= *CONST OutputCSP x v P*

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

Basic translations for state variable communications

syntax
  -csp-input-var :: *logic* ⇒ *id* ⇒ *logic* ⇒ *logic* ⇒ *logic* (-**?$**-:- ⟶ - [*81*, *0*, *0*, *80*] *80*)
  -csp-inputu-var :: *logic* ⇒ *id* ⇒ *logic* ⇒ *logic* (-**?$**- ⟶ - [*81*, *0*, *80*] *80*)

translations
  c?$*x*:*A* ⟶ *P* ⇌ *CONST InputVarCSP c x A P*
  c?$*x* ⟶ *P* ⇀ *CONST InputVarCSP c x* (*CONST UNIV*) *P*
  c?$*x* ⟶ *P* <= *c*?$*x*:*true* ⟶ *P*

lemma *outp-constraint-prod*:
  (*outp-constraint* ≪*a*≫ *x* ∧ *outp-constraint* ≪*b*≫ *y*) =
    outp-constraint ≪(*a*, *b*)≫ (*x*, *y*)
  by (*simp add*: *outp-constraint-def*, *pred-auto*)

lemma *subst-outp-constraint* [*usubst*]:
  $\sigma \dagger (v =_o x) = (\sigma \dagger v =_o x)$
  by (*rel-auto*)

lemma *UINF-one-point-simp* [*rpred*]:
  $[\![\bigwedge i.\ P\ i\ is\ R1]\!] \Longrightarrow (\bigsqcap\ x \cdot [\ll i \gg =_o x]_{S<} \wedge P(x)) = P(i)$
  by (*rel-blast*)

lemma *USUP-one-point-simp* [*rpred*]:
  $[\![\bigwedge i.\ P\ i\ is\ R1]\!] \Longrightarrow (\bigsqcup\ x \cdot [\ll i\gg =_o x]_{S<} \Rightarrow_r P(x)) = P(i)$
  by (*rel-blast*)

lemma *USUP-eq-event-eq* [*rpred*]:
  assumes $\bigwedge y.\ P(y)\ is\ RR$
  shows $(\bigsqcup\ y \cdot [v =_o y]_{S<} \Rightarrow_r P(y)) = P(y)[\![y \to \lceil v\rceil_{S\leftarrow}]\!]$
proof –
  have $(\bigsqcup\ y \cdot [v =_o y]_{S<} \Rightarrow_r RR(P(y))) = RR(P(y))[\![y \to \lceil v\rceil_{S\leftarrow}]\!]$
    apply (*rel-simp*, *safe*)
    apply *metis*
    apply *blast*
    apply *simp*
    done
  thus *?thesis*
    by (*simp add*: *Healthy-if assms*)
qed

lemma *UINF-eq-event-eq* [*rpred*]:
  assumes $\bigwedge y.\ P(y)\ is\ RR$
  shows $(\bigsqcap\ y \cdot [v =_o y]_{S<} \wedge P(y)) = P(y)[\![y \to \lceil v\rceil_{S\leftarrow}]\!]$
proof –
  have $(\bigsqcap\ y \cdot [v =_o y]_{S<} \wedge RR(P(y))) = RR(P(y))[\![y \to \lceil v\rceil_{S\leftarrow}]\!]$
    by (*rel-simp*, *safe*, *metis*)
  thus *?thesis*
    by (*simp add*: *Healthy-if assms*)
qed

Proofs that the input constrained parser versions of output is the same as the regular definition.

lemma *output-prefix-is-OutputCSP* [*simp*]:
  assumes *A is NCSP*
  shows $x!(P) \to A = OutputCSP\ x\ P\ A$ (**is** *?lhs* = *?rhs*)
  by (*rule SRD-eq-intro*, *simp-all add*: *assms closure rdes*, *rel-auto*+)

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

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

end

10 Recursion in Circus

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

10.1 Fixed-points

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

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

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

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

lemma *mu-CSP-equiv*:
assumes *Monotonic F* $F \in [\![CSP]\!]_H \to [\![CSP]\!]_H$
shows $(\mu_R\ F) = (\mu_C\ F)$
by (*simp add*: *srd-mu-equiv assms comp-def*)

lemma *mu-CSP-unfold*:
P *is 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*]: $(\mu_C\ (op\ ;;\ Q)) = (\mu\ X \cdot Q\ ;;\ CSP\ X)$
by (*simp add*: *comp-def*)

lemma *mu-csp-basic-refine*:
assumes
P *is CSP* Q *is NCSP* Q *is Productive* $pre_R(P) = true_r$ $pre_R(Q) = true_r$
$peri_R\ P \sqsubseteq peri_R\ Q$
$peri_R\ P \sqsubseteq post_R\ Q\ ;;\ peri_R\ P$
shows $P \sqsubseteq (\mu_C\ X \cdot Q\ ;;\ X)$
proof (*rule SRD-refine-intro'*, *simp-all add*: *closure usubst alpha rpred rdes unrest wp seq-UINF-distr assms*)
show $peri_R\ P \sqsubseteq (\bigsqcap\ i \cdot post_R\ Q\ \hat{}\ i\ ;;\ peri_R\ Q)$
proof (*rule UINF-refines'*)
fix i
show $peri_R\ P \sqsubseteq post_R\ Q\ \hat{}\ i\ ;;\ peri_R\ Q$
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* $pre_R(Q) = true_r$
$[P\ t\ r]_{S<}[\![(t, r) \to (\&tt, \$ref')_u]\!] \sqsubseteq peri_R\ Q$

$[P\ t\ r]_{S<}[\![(t,\ r) \rightarrow (\&tt,\ \$ref\,')_u]\!] \sqsubseteq post_R\ Q\ ;;_h\ [P\ t\ r]_{S<}[\![(t,\ r) \rightarrow (\&tt,\ \$ref\,')_u]\!]$
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 $[P\ trace\ refs]_{S<}[\![trace \rightarrow \&tt]\!][\![refs \rightarrow \$ref\,']\!] \sqsubseteq peri_R\ Q$
using *assms* **by** (*simp add: msubst-pair*)
show $[P\ trace\ refs]_{S<}[\![trace \rightarrow \&tt]\!][\![refs \rightarrow \$ref\,']\!] \sqsubseteq post_R\ Q\ ;;\ [P\ trace\ refs]_{S<}[\![trace \rightarrow \&tt]\!][\![refs \rightarrow \$ref\,']\!]$
using *assms* **by** (*simp add: msubst-pair*)
qed

10.2 Example action expansion

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

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

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

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

end

11 Circus Trace Merge

theory *utp-circus-traces*
imports *utp-circus-core*
begin

11.1 Function Definition

fun *tr-par* ::
$'\vartheta\ set \Rightarrow\ '\vartheta\ list \Rightarrow\ '\vartheta\ list \Rightarrow\ '\vartheta\ list\ set$ **where**
tr-par cs [] [] = {[]} |
tr-par cs $(e\ \#\ t)$ [] = (*if* $e \in cs$ *then* {[]} *else* {[e]} $\frown$ (*tr-par cs t* [])) |
tr-par cs [] $(e\ \#\ t)$ = (*if* $e \in cs$ *then* {[]} *else* {[e]} $\frown$ (*tr-par cs* [] *t*)) |
tr-par cs $(e_1\ \#\ t_1)\ (e_2\ \#\ t_2)$ =
(*if* $e_1 = e_2$
then
if $e_1 \in cs$ (* $\wedge\ e_2 \in cs$ *)
then $\{[e_1]\} \frown$ (*tr-par cs* $t_1\ t_2$)
else
$(\{[e_1]\} \frown$ (*tr-par cs* $t_1\ (e_2\ \#\ t_2))) \cup$
$(\{[e_2]\} \frown$ (*tr-par cs* $(e_1\ \#\ t_1)\ t_2$))
else
if $e_1 \in cs$ *then*
if $e_2 \in cs$ *then* {[]}
else
$\{[e_2]\} \frown$ (*tr-par cs* $(e_1\ \#\ t_1)\ t_2$)
else

if $e_2 \in cs$ *then*
$\{[e_1]\} \frown (tr\text{-}par\ cs\ t_1\ (e_2\ \#\ t_2))$
else
$\{[e_1]\} \frown (tr\text{-}par\ cs\ t_1\ (e_2\ \#\ t_2)) \cup$
$\{[e_2]\} \frown (tr\text{-}par\ cs\ (e_1\ \#\ t_1)\ t_2))$

abbreviation *tr-inter* :: $'\vartheta$ *list* $\Rightarrow$ $'\vartheta$ *list* $\Rightarrow$ $'\vartheta$ *list set* (**infixr** $|||_t$ *100*) **where**
$x\ |||_t\ y \equiv tr\text{-}par\ \{\}\ x\ y$

11.2 Lifted Trace Merge

syntax *-utr-par* ::
logic $\Rightarrow$ *logic* $\Rightarrow$ *logic* $\Rightarrow$ *logic* ((- $\star_{-}$/ -) [*100*, *0*, *101*] *100*)

The function *trop* is used to lift ternary operators.

translations
t1 $\star_{cs}$ *t2* == (*CONST trop*) (*CONST tr-par*) *cs t1 t2*

11.3 Trace Merge Lemmas

lemma *tr-par-empty*:
tr-par cs t1 [] = {*takeWhile* ($\lambda x.\ x \notin cs$) *t1*}
tr-par cs [] *t2* = {*takeWhile* ($\lambda x.\ x \notin cs$) *t2*}
— Subgoal 1
apply (*induct t1*; *simp*)
— Subgoal 2
apply (*induct t2*; *simp*)
done

lemma *tr-par-sym*:
tr-par cs t1 t2 = *tr-par cs t2 t1*
apply (*induct t1 arbitrary*: *t2*)
— Subgoal 1
apply (*simp add*: *tr-par-empty*)
— Subgoal 2
apply (*induct-tac t2*)
— Subgoal 2.1
apply (*clarsimp*)
— Subgoal 2.2
apply (*clarsimp*)
apply (*blast*)
done

lemma *tr-inter-sym*: $x\ |||_t\ y = y\ |||_t\ x$
by (*simp add*: *tr-par-sym*)

lemma *trace-merge-nil* [*simp*]: $x \star_{\{\}_u} \langle\rangle = \{x\}_u$
by (*pred-auto*, *simp-all add*: *tr-par-empty*, *metis takeWhile-eq-all-conv*)

lemma *trace-merge-empty* [*simp*]:
$(\langle\rangle \star_{cs} \langle\rangle) = \{\langle\rangle\}_u$
by (*rel-auto*)

lemma *trace-merge-single-empty* [*simp*]:
$a \in cs \Longrightarrow \langle «a» \rangle \star_{«cs»} \langle\rangle = \{\langle\rangle\}_u$
by (*rel-auto*)

lemma *trace-merge-empty-single* [*simp*]:
  $a \in cs \Longrightarrow \langle\rangle \star_{\ll cs \gg} \langle\ll a \gg\rangle = \{\langle\rangle\}_u$
  by (*rel-auto*)

lemma *trace-merge-commute*: $t_1 \star_{cs} t_2 = t_2 \star_{cs} t_1$
  by (*rel-simp*, *simp add*: *tr-par-sym*)

lemma *csp-trace-simps* [*simp*]:
  $v \ \hat{}_u \ \langle\rangle = v \ \langle\rangle \ \hat{}_u \ v = v$
  $v + \langle\rangle = v \ \langle\rangle + v = v$
  $bop \ (op \ \#) \ x \ xs \ \hat{}_u \ ys = bop \ (op \ \#) \ x \ (xs \ \hat{}_u \ ys)$
  by (*rel-auto*)+

end

12 Circus Parallel Composition

theory *utp-circus-parallel*
  imports
    utp-circus-prefix
    utp-circus-traces
    utp-circus-recursion
begin

12.1 Merge predicates

definition *CSPInnerMerge* :: $('\alpha \Longrightarrow '\sigma) \Rightarrow '\psi \ set \Rightarrow ('\beta \Longrightarrow '\sigma) \Rightarrow (('\sigma,'\psi) \ st\text{-}csp) \ merge \ (N_C)$ **where**
  [*upred-defs*]:
  CSPInnerMerge ns1 cs ns2 = (
    $\$ref' \subseteq_u ((\$0{-}ref \cup_u \$1{-}ref) \cap_u \ll cs \gg) \cup_u ((\$0{-}ref \cap_u \$1{-}ref) - \ll cs \gg) \wedge$
    $\$tr_< \leq_u \$tr' \wedge$
    $(\$tr' - \$tr_<) \in_u (\$0{-}tr - \$tr_<) \star_{\ll cs \gg} (\$1{-}tr - \$tr_<) \wedge$
    $(\$0{-}tr - \$tr_<) \restriction_u \ll cs \gg =_u (\$1{-}tr - \$tr_<) \restriction_u \ll cs \gg \wedge$
    $\$st' =_u (\$st_< \oplus \$0{-}st \ on \ \&ns1) \oplus \$1{-}st \ on \ \&ns2)$

definition *CSPInnerInterleave* :: $('\alpha \Longrightarrow '\sigma) \Rightarrow ('\beta \Longrightarrow '\sigma) \Rightarrow (('\sigma,'\psi) \ st\text{-}csp) \ merge \ (N_I)$ **where**
  [*upred-defs*]:
  N_I *ns1 ns2* = (
    $\$ref' \subseteq_u (\$0{-}ref \cap_u \$1{-}ref) \wedge$
    $\$tr_< \leq_u \$tr' \wedge$
    $(\$tr' - \$tr_<) \in_u (\$0{-}tr - \$tr_<) \star_{\{\}_u} (\$1{-}tr - \$tr_<) \wedge$
    $\$st' =_u (\$st_< \oplus \$0{-}st \ on \ \&ns1) \oplus \$1{-}st \ on \ \&ns2)$

An intermediate merge hides the state, whilst a final merge hides the refusals.

definition *CSPInterMerge* **where**
[*upred-defs*]: *CSPInterMerge P ns1 cs ns2 Q* = $(P \parallel_{(\exists \ \$st' \ \cdot \ N_C \ ns1 \ cs \ ns2)} Q)$

definition *CSPFinalMerge* **where**
[*upred-defs*]: *CSPFinalMerge P ns1 cs ns2 Q* = $(P \parallel_{(\exists \ \$ref' \ \cdot \ N_C \ ns1 \ cs \ ns2)} Q)$

syntax
  -cinter-merge :: *logic* ⇒ *salpha* ⇒ *logic* ⇒ *salpha* ⇒ *logic* ⇒ *logic* (- $[\![$-|-|-$]\!]^I$ - [*85*,*0*,*0*,*0*,*86*] *86*)
  -cfinal-merge :: *logic* ⇒ *salpha* ⇒ *logic* ⇒ *salpha* ⇒ *logic* ⇒ *logic* (- $[\![$-|-|-$]\!]^F$ - [*85*,*0*,*0*,*0*,*86*] *86*)
  -wrC :: *logic* ⇒ *salpha* ⇒ *logic* ⇒ *salpha* ⇒ *logic* ⇒ *logic* (- *wr*[-|-|-]$_C$ - [*85*,*0*,*0*,*0*,*86*] *86*)

translations
  -cinter-merge P ns1 cs ns2 Q == CONST CSPInterMerge P ns1 cs ns2 Q
  -cfinal-merge P ns1 cs ns2 Q == CONST CSPFinalMerge P ns1 cs ns2 Q
  -wrC P ns1 cs ns2 Q == $P\ wr_R(N_C\ ns1\ cs\ ns2)\ Q$

lemma *CSPInnerMerge-R2m* [*closure*]: N_C *ns1 cs ns2 is R2m*
  by (*rel-auto*)

lemma *CSPInnerMerge-RDM* [*closure*]: N_C *ns1 cs ns2 is RDM*
  by (*rule RDM-intro*, *simp add*: *closure*, *simp-all add*: *CSPInnerMerge-def unrest*)

lemma *ex-ref'-R2m-closed* [*closure*]:
  assumes *P is R2m*
  shows $(\exists\ \$ref\acute{}\ \cdot\ P)$ *is R2m*
proof −
  have $R2m(\exists\ \$ref\acute{}\ \cdot\ R2m\ P) = (\exists\ \$ref\acute{}\ \cdot\ R2m\ P)$
    by (*rel-auto*)
  thus *?thesis*
    by (*metis Healthy-def' assms*)
qed

lemma *CSPInnerMerge-unrests* [*unrest*]:
  $\$ok_< \sharp\ N_C\ ns1\ cs\ ns2$
  $\$wait_< \sharp\ N_C\ ns1\ cs\ ns2$
  by (*rel-auto*)+

lemma *CSPInterMerge-RR-closed* [*closure*]:
  assumes *P is RR Q is RR*
  shows $P\ [\![ns1|cs|ns2]\!]^I\ Q$ *is RR*
  by (*simp add*: *CSPInterMerge-def parallel-RR-closed assms closure unrest*)

lemma *CSPInterMerge-unrest-st'* [*unrest*]:
  $\$st\acute{}\ \sharp\ P\ [\![ns1|cs|ns2]\!]^I\ Q$
  by (*rel-auto*)

lemma *CSPFinalMerge-RR-closed* [*closure*]:
  assumes *P is RR Q is RR*
  shows $P\ [\![ns1|cs|ns2]\!]^F\ Q$ *is RR*
  by (*simp add*: *CSPFinalMerge-def parallel-RR-closed assms closure unrest*)

lemma *CSPInnerMerge-empty-Interleave*:
  $N_C\ ns1\ \{\}\ ns2 = N_I\ ns1\ ns2$
  by (*rel-auto*)

definition *CSPMerge* :: $('\alpha \Longrightarrow '\sigma) \Rightarrow '\psi\ set \Rightarrow ('\beta \Longrightarrow '\sigma) \Rightarrow (('\sigma,'\psi)\ st\text{-}csp)\ merge\ (M_C)$ **where**
[*upred-defs*]: $M_C\ ns1\ cs\ ns2 = M_R(N_C\ ns1\ cs\ ns2)$;; *Skip*

definition *CSPInterleave* :: $('\alpha \Longrightarrow '\sigma) \Rightarrow ('\beta \Longrightarrow '\sigma) \Rightarrow (('\sigma,'\psi)\ st\text{-}csp)\ merge\ (M_I)$ **where**
[*upred-defs*]: $M_I\ ns1\ ns2 = M_R(N_I\ ns1\ ns2)$;; *Skip*

lemma *swap-CSPInnerMerge*:
  $ns1 \bowtie ns2 \Longrightarrow swap_m$;; $(N_C\ ns1\ cs\ ns2) = (N_C\ ns2\ cs\ ns1)$
  apply (*rel-auto*)
  using *tr-par-sym* **apply** *blast*

apply (*simp add*: *lens-indep-comm*)
using *tr-par-sym* **apply** *blast*
apply (*simp add*: *lens-indep-comm*)
done

lemma *SymMerge-CSPInnerMerge-NS* [*closure*]: $N_C\ 0_L\ cs\ 0_L$ *is SymMerge*
by (*simp add*: *Healthy-def swap-CSPInnerMerge*)

lemma *SymMerge-CSPInnerInterleave* [*closure*]:
$N_I\ 0_L\ 0_L$ *is SymMerge*
by (*metis CSPInnerMerge-empty-Interleave SymMerge-CSPInnerMerge-NS*)

lemma *SymMerge-CSPInnerInterleave* [*closure*]:
AssocMerge $(N_I\ 0_L\ 0_L)$
apply (*rel-auto*)
apply (*rename-tac tr* tr_2' ref_0 tr_0' ref_0' tr_1' ref_1' tr' ref_2' tr_i' ref_3')
oops

lemma *CSPInterMerge-false* [*rpred*]: $P\ [\![ns1|cs|ns2]\!]^I\ false = false$
by (*simp add*: *CSPInterMerge-def*)

lemma *CSPFinalMerge-false* [*rpred*]: $P\ [\![ns1|cs|ns2]\!]^F\ false = false$
by (*simp add*: *CSPFinalMerge-def*)

lemma *CSPInterMerge-commute*:
assumes $ns1 \bowtie ns2$
shows $P\ [\![ns1|cs|ns2]\!]^I\ Q = Q\ [\![ns2|cs|ns1]\!]^I\ P$
proof −
have $P\ [\![ns1|cs|ns2]\!]^I\ Q = P \parallel_{\exists\ \$st\acute{}\ \cdot\ N_C\ ns1\ cs\ ns2}\ Q$
by (*simp add*: *CSPInterMerge-def*)
also have ... $= P \parallel_{\exists\ \$st\acute{}\ \cdot\ (swap_m\ ;;\ N_C\ ns2\ cs\ ns1)}\ Q$
by (*simp add*: *swap-CSPInnerMerge lens-indep-sym assms*)
also have ... $= P \parallel_{swap_m\ ;;\ (\exists\ \$st\acute{}\ \cdot\ N_C\ ns2\ cs\ ns1)}\ Q$
by (*simp add*: *seqr-exists-right*)
also have ... $= Q \parallel_{(\exists\ \$st\acute{}\ \cdot\ N_C\ ns2\ cs\ ns1)}\ P$
by (*simp add*: *par-by-merge-commute-swap*[*THEN sym*])
also have ... $= Q\ [\![ns2|cs|ns1]\!]^I\ P$
by (*simp add*: *CSPInterMerge-def*)
finally show *?thesis* **.**
qed

lemma *CSPFinalMerge-commute*:
assumes $ns1 \bowtie ns2$
shows $P\ [\![ns1|cs|ns2]\!]^F\ Q = Q\ [\![ns2|cs|ns1]\!]^F\ P$
proof −
have $P\ [\![ns1|cs|ns2]\!]^F\ Q = P \parallel_{\exists\ \$ref\acute{}\ \cdot\ N_C\ ns1\ cs\ ns2}\ Q$
by (*simp add*: *CSPFinalMerge-def*)
also have ... $= P \parallel_{\exists\ \$ref\acute{}\ \cdot\ (swap_m\ ;;\ N_C\ ns2\ cs\ ns1)}\ Q$
by (*simp add*: *swap-CSPInnerMerge lens-indep-sym assms*)
also have ... $= P \parallel_{swap_m\ ;;\ (\exists\ \$ref\acute{}\ \cdot\ N_C\ ns2\ cs\ ns1)}\ Q$
by (*simp add*: *seqr-exists-right*)
also have ... $= Q \parallel_{(\exists\ \$ref\acute{}\ \cdot\ N_C\ ns2\ cs\ ns1)}\ P$
by (*simp add*: *par-by-merge-commute-swap*[*THEN sym*])

also have ... = Q $[\![ns2|cs|ns1]\!]^F$ P
  by (*simp add*: *CSPFinalMerge-def*)
finally show *?thesis* **.**
qed

Important theorem that shows the form of a parallel process

lemma *CSPInnerMerge-form*:
  fixes P Q :: ($'\sigma$,$'\varphi$) *action*
  assumes *vwb-lens ns1 vwb-lens ns2 P is RR Q is RR*
  shows
  $P \parallel_{N_C\ ns1\ cs\ ns2} Q =$
    $(\exists\ (ref_0,\ ref_1,\ st_0,\ st_1,\ tt_0,\ tt_1) \cdot$
      $P[\![\ll ref_0\gg,\ll st_0\gg,\langle\rangle,\ll tt_0\gg/\$ref\acute{},\$st\acute{},\$tr,\$tr\acute{}]\!] \wedge Q[\![\ll ref_1\gg,\ll st_1\gg,\langle\rangle,\ll tt_1\gg/\$ref\acute{},\$st\acute{},\$tr,\$tr\acute{}]\!]$
      $\wedge\ \$ref\acute{} \subseteq_u ((\ll ref_0\gg \cup_u \ll ref_1\gg) \cap_u \ll cs\gg) \cup_u ((\ll ref_0\gg \cap_u \ll ref_1\gg) - \ll cs\gg)$
      $\wedge\ \$tr \le_u \$tr\acute{}$
      $\wedge\ \&tt \in_u \ll tt_0\gg \star_{\ll cs\gg} \ll tt_1\gg$
      $\wedge \ll tt_0\gg \restriction_u \ll cs\gg =_u \ll tt_1\gg \restriction_u \ll cs\gg$
      $\wedge\ \$st\acute{} =_u (\$st \oplus \ll st_0\gg\ on\ \&ns1) \oplus \ll st_1\gg\ on\ \&ns2)$
  (**is** *?lhs* = *?rhs*)
proof −
  have P:($\exists$ {\$$ok\acute{}$,\$$wait\acute{}$} $\cdot$ $R2(P)$) = P (**is** $?P'$ = -)
    by (*simp add*: *ex-unrest ex-plus Healthy-if assms RR-implies-R2 unrest closure*)
  have Q:($\exists$ {\$$ok\acute{}$,\$$wait\acute{}$} $\cdot$ $R2(Q)$) = Q (**is** $?Q'$ = -)
    by (*simp add*: *ex-unrest ex-plus Healthy-if assms RR-implies-R2 unrest closure*)
  from *assms*(*1*,*2*)
  have $?P' \parallel_{N_C\ ns1\ cs\ ns2} ?Q' =$
    $(\exists\ (ref_0,\ ref_1,\ st_0,\ st_1,\ tt_0,\ tt_1) \cdot$
      $?P'[\![\ll ref_0\gg,\ll st_0\gg,\langle\rangle,\ll tt_0\gg/\$ref\acute{},\$st\acute{},\$tr,\$tr\acute{}]\!] \wedge ?Q'[\![\ll ref_1\gg,\ll st_1\gg,\langle\rangle,\ll tt_1\gg/\$ref\acute{},\$st\acute{},\$tr,\$tr\acute{}]\!]$
      $\wedge\ \$ref\acute{} \subseteq_u ((\ll ref_0\gg \cup_u \ll ref_1\gg) \cap_u \ll cs\gg) \cup_u ((\ll ref_0\gg \cap_u \ll ref_1\gg) - \ll cs\gg)$
      $\wedge\ \$tr \le_u \$tr\acute{}$
      $\wedge\ \&tt \in_u \ll tt_0\gg \star_{\ll cs\gg} \ll tt_1\gg$
      $\wedge \ll tt_0\gg \restriction_u \ll cs\gg =_u \ll tt_1\gg \restriction_u \ll cs\gg$
      $\wedge\ \$st\acute{} =_u (\$st \oplus \ll st_0\gg\ on\ \&ns1) \oplus \ll st_1\gg\ on\ \&ns2)$
    apply (*simp add*: *par-by-merge-alt-def*, *rel-auto*, *blast*)
    apply (*rename-tac ok wait tr st ref tr′ ref′* ref_0 ref_1 st_0 st_1 tr_0 ok_0 tr_1 $wait_0$ ok_1 $wait_1$)
    apply (*rule-tac x=ok* **in** *exI*)
    apply (*rule-tac x=wait* **in** *exI*)
    apply (*rule-tac x=tr* **in** *exI*)
    apply (*rule-tac x=st* **in** *exI*)
    apply (*rule-tac x=ref* **in** *exI*)
    apply (*rule-tac x=tr* @ tr_0 **in** *exI*)
    apply (*rule-tac x=*st_0 **in** *exI*)
    apply (*rule-tac x=*ref_0 **in** *exI*)
    apply (*auto*)
    apply (*metis Prefix-Order.prefixI append-minus*)
  done
  thus *?thesis*
    by (*simp add*: *P Q*)
qed

lemma *CSPInterMerge-form*:
  fixes P Q :: ($'\sigma$,$'\varphi$) *action*
  assumes *vwb-lens ns1 vwb-lens ns2 P is RR Q is RR*
  shows
  P $[\![ns1|cs|ns2]\!]^I$ Q =

$(\exists\ (ref_0,\ ref_1,\ st_0,\ st_1,\ tt_0,\ tt_1) \cdot$
$P[\![\ll ref_0\gg,\ll st_0\gg,\langle\rangle,\ll tt_0\gg/\$ref\acute{},\$st\acute{},\$tr,\$tr\acute{}]\!] \wedge Q[\![\ll ref_1\gg,\ll st_1\gg,\langle\rangle,\ll tt_1\gg/\$ref\acute{},\$st\acute{},\$tr,\$tr\acute{}]\!]$
$\wedge\ \$ref\acute{} \subseteq_u ((\ll ref_0\gg \cup_u \ll ref_1\gg) \cap_u \ll cs\gg) \cup_u ((\ll ref_0\gg \cap_u \ll ref_1\gg) - \ll cs\gg)$
$\wedge\ \$tr \le_u \$tr\acute{}$
$\wedge\ \&tt \in_u \ll tt_0\gg \star_{\ll cs\gg} \ll tt_1\gg$
$\wedge \ll tt_0\gg \restriction_u \ll cs\gg =_u \ll tt_1\gg \restriction_u \ll cs\gg)$
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = $(\exists\ \$st\acute{} \cdot P \parallel_{N_C}\ ns1\ cs\ ns2\ Q)$
by (*simp add*: *CSPInterMerge-def par-by-merge-def seqr-exists-right*)
also have ... =
$(\exists\ \$st\acute{} \cdot$
$(\exists\ (ref_0,\ ref_1,\ st_0,\ st_1,\ tt_0,\ tt_1) \cdot$
$P[\![\ll ref_0\gg,\ll st_0\gg,\langle\rangle,\ll tt_0\gg/\$ref\acute{},\$st\acute{},\$tr,\$tr\acute{}]\!] \wedge Q[\![\ll ref_1\gg,\ll st_1\gg,\langle\rangle,\ll tt_1\gg/\$ref\acute{},\$st\acute{},\$tr,\$tr\acute{}]\!]$
$\wedge\ \$ref\acute{} \subseteq_u ((\ll ref_0\gg \cup_u \ll ref_1\gg) \cap_u \ll cs\gg) \cup_u ((\ll ref_0\gg \cap_u \ll ref_1\gg) - \ll cs\gg)$
$\wedge\ \$tr \le_u \$tr\acute{}$
$\wedge\ \&tt \in_u \ll tt_0\gg \star_{\ll cs\gg} \ll tt_1\gg$
$\wedge \ll tt_0\gg \restriction_u \ll cs\gg =_u \ll tt_1\gg \restriction_u \ll cs\gg$
$\wedge\ \$st\acute{} =_u (\$st \oplus \ll st_0\gg\ on\ \&ns1) \oplus \ll st_1\gg\ on\ \&ns2))$
by (*simp add*: *CSPInnerMerge-form assms*)
also have ... = *?rhs*
by (*rel-blast*)
finally show *?thesis* **.**
qed

lemma *CSPFinalMerge-form*:
fixes *P Q* :: $('\sigma,'\varphi)$ *action*
assumes *vwb-lens ns1 vwb-lens ns2 P is RR Q is RR* $\$ref\acute{} \sharp P$ $\$ref\acute{} \sharp Q$
shows
$(P\ [\![ns1|cs|ns2]\!]^F\ Q) =$
$(\exists\ (st_0,\ st_1,\ tt_0,\ tt_1) \cdot$
$P[\![\ll st_0\gg,\langle\rangle,\ll tt_0\gg/\$st\acute{},\$tr,\$tr\acute{}]\!] \wedge Q[\![\ll st_1\gg,\langle\rangle,\ll tt_1\gg/\$st\acute{},\$tr,\$tr\acute{}]\!]$
$\wedge\ \$tr \le_u \$tr\acute{}$
$\wedge\ \&tt \in_u \ll tt_0\gg \star_{\ll cs\gg} \ll tt_1\gg$
$\wedge \ll tt_0\gg \restriction_u \ll cs\gg =_u \ll tt_1\gg \restriction_u \ll cs\gg$
$\wedge\ \$st\acute{} =_u (\$st \oplus \ll st_0\gg\ on\ \&ns1) \oplus \ll st_1\gg\ on\ \&ns2)$
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = $(\exists\ \$ref\acute{} \cdot P \parallel_{N_C}\ ns1\ cs\ ns2\ Q)$
by (*simp add*: *CSPFinalMerge-def par-by-merge-def seqr-exists-right*)
also have ... =
$(\exists\ \$ref\acute{} \cdot$
$(\exists\ (ref_0,\ ref_1,\ st_0,\ st_1,\ tt_0,\ tt_1) \cdot$
$P[\![\ll ref_0\gg,\ll st_0\gg,\langle\rangle,\ll tt_0\gg/\$ref\acute{},\$st\acute{},\$tr,\$tr\acute{}]\!] \wedge Q[\![\ll ref_1\gg,\ll st_1\gg,\langle\rangle,\ll tt_1\gg/\$ref\acute{},\$st\acute{},\$tr,\$tr\acute{}]\!]$
$\wedge\ \$ref\acute{} \subseteq_u ((\ll ref_0\gg \cup_u \ll ref_1\gg) \cap_u \ll cs\gg) \cup_u ((\ll ref_0\gg \cap_u \ll ref_1\gg) - \ll cs\gg)$
$\wedge\ \$tr \le_u \$tr\acute{}$
$\wedge\ \&tt \in_u \ll tt_0\gg \star_{\ll cs\gg} \ll tt_1\gg$
$\wedge \ll tt_0\gg \restriction_u \ll cs\gg =_u \ll tt_1\gg \restriction_u \ll cs\gg$
$\wedge\ \$st\acute{} =_u (\$st \oplus \ll st_0\gg\ on\ \&ns1) \oplus \ll st_1\gg\ on\ \&ns2))$
by (*simp add*: *CSPInnerMerge-form assms*)
also have ... =
$(\exists\ \$ref\acute{} \cdot$
$(\exists\ (ref_0,\ ref_1,\ st_0,\ st_1,\ tt_0,\ tt_1) \cdot$
$(\exists\ \$ref\acute{} \cdot P)[\![\ll ref_0\gg,\ll st_0\gg,\langle\rangle,\ll tt_0\gg/\$ref\acute{},\$st\acute{},\$tr,\$tr\acute{}]\!] \wedge (\exists\ \$ref\acute{} \cdot Q)[\![\ll ref_1\gg,\ll st_1\gg,\langle\rangle,\ll tt_1\gg/\$ref\acute{},\$st\acute{},\$tr$
$\wedge\ \$ref\acute{} \subseteq_u ((\ll ref_0\gg \cup_u \ll ref_1\gg) \cap_u \ll cs\gg) \cup_u ((\ll ref_0\gg \cap_u \ll ref_1\gg) - \ll cs\gg)$

$\wedge\ \$tr \leq_u \$tr'$
$\wedge\ \&tt \in_u \ll tt_0 \gg \star_{\ll cs \gg} \ll tt_1 \gg$
$\wedge \ll tt_0 \gg \restriction_u \ll cs \gg =_u \ll tt_1 \gg \restriction_u \ll cs \gg$
$\wedge\ \$st' =_u (\$st \oplus \ll st_0 \gg\ \mathit{on}\ \&ns1) \oplus \ll st_1 \gg\ \mathit{on}\ \&ns2))$
by (*simp add*: *ex-unrest assms*)
also have ... =
$(\exists\ (st_0,\ st_1,\ tt_0,\ tt_1) \cdot$
$(\exists\ \$ref' \cdot P)[\![\ll st_0 \gg, \langle\rangle, \ll tt_0 \gg / \$st', \$tr, \$tr']\!] \wedge (\exists\ \$ref' \cdot Q)[\![\ll st_1 \gg, \langle\rangle, \ll tt_1 \gg / \$st', \$tr, \$tr']\!]$
$\wedge\ \$tr \leq_u \$tr'$
$\wedge\ \&tt \in_u \ll tt_0 \gg \star_{\ll cs \gg} \ll tt_1 \gg$
$\wedge \ll tt_0 \gg \restriction_u \ll cs \gg =_u \ll tt_1 \gg \restriction_u \ll cs \gg$
$\wedge\ \$st' =_u (\$st \oplus \ll st_0 \gg\ \mathit{on}\ \&ns1) \oplus \ll st_1 \gg\ \mathit{on}\ \&ns2)$
by (*rel-blast*)
also have ... = *?rhs*
by (*simp add*: *ex-unrest assms*)
finally show *?thesis* **.**
qed

lemma *merge-csp-do-left*:
assumes *vwb-lens ns1 vwb-lens ns2 ns1* $\bowtie$ *ns2 P is RR*
shows $\Phi(s_0,\sigma_0,t_0) \parallel_{N_C\ ns1\ cs\ ns2} P =$
$(\exists\ (ref_1,\ st_1,\ tt_1) \cdot$
$[s_0]_{S<} \wedge$
$[\$ref' \mapsto_s \ll ref_1 \gg, \$st' \mapsto_s \ll st_1 \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll tt_1 \gg] \dagger P \wedge$
$\$ref' \subseteq_u \ll cs \gg \cup_u (\ll ref_1 \gg - \ll cs \gg) \wedge$
$[\ll trace \gg \in_u t_0 \star_{\ll cs \gg} \ll tt_1 \gg \wedge t_0 \restriction_u \ll cs \gg =_u \ll tt_1 \gg \restriction_u \ll cs \gg]_t \wedge$
$\$st' =_u \$st \oplus \ll \sigma_0 \gg (\$st)_a\ \mathit{on}\ \&ns1 \oplus \ll st_1 \gg\ \mathit{on}\ \&ns2)$
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* =
$(\exists\ (ref_0,\ ref_1,\ st_0,\ st_1,\ tt_0,\ tt_1) \cdot$
$[\$ref' \mapsto_s \ll ref_0 \gg, \$st' \mapsto_s \ll st_0 \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll tt_0 \gg] \dagger \Phi(s_0,\sigma_0,t_0) \wedge$
$[\$ref' \mapsto_s \ll ref_1 \gg, \$st' \mapsto_s \ll st_1 \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll tt_1 \gg] \dagger P \wedge$
$\$ref' \subseteq_u (\ll ref_0 \gg \cup_u \ll ref_1 \gg) \cap_u \ll cs \gg \cup_u (\ll ref_0 \gg \cap_u \ll ref_1 \gg - \ll cs \gg) \wedge$
$\$tr \leq_u \$tr' \wedge$
$\&tt \in_u \ll tt_0 \gg \star_{\ll cs \gg} \ll tt_1 \gg \wedge \ll tt_0 \gg \restriction_u \ll cs \gg =_u \ll tt_1 \gg \restriction_u \ll cs \gg \wedge \$st' =_u \$st \oplus \ll st_0 \gg\ \mathit{on}\ \&ns1$
$\oplus \ll st_1 \gg\ \mathit{on}\ \&ns2)$
by (*simp add*: *CSPInnerMerge-form assms closure*)
also have ... =
$(\exists\ (ref_1,\ st_1,\ tt_1) \cdot$
$[s_0]_{S<} \wedge$
$[\$ref' \mapsto_s \ll ref_1 \gg, \$st' \mapsto_s \ll st_1 \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll tt_1 \gg] \dagger P \wedge$
$\$ref' \subseteq_u \ll cs \gg \cup_u (\ll ref_1 \gg - \ll cs \gg) \wedge$
$[\ll trace \gg \in_u t_0 \star_{\ll cs \gg} \ll tt_1 \gg \wedge t_0 \restriction_u \ll cs \gg =_u \ll tt_1 \gg \restriction_u \ll cs \gg]_t \wedge$
$\$st' =_u \$st \oplus \ll \sigma_0 \gg (\$st)_a\ \mathit{on}\ \&ns1 \oplus \ll st_1 \gg\ \mathit{on}\ \&ns2)$
by (*rel-blast*)
finally show *?thesis* **.**
qed

lemma *merge-csp-do-right*:
assumes *vwb-lens ns1 vwb-lens ns2 ns1* $\bowtie$ *ns2 P is RR*
shows $P \parallel_{N_C\ ns1\ cs\ ns2} \Phi(s_1,\sigma_1,t_1) =$
$(\exists\ (ref_0,\ st_0,\ tt_0) \cdot$
$[\$ref' \mapsto_s \ll ref_0 \gg, \$st' \mapsto_s \ll st_0 \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll tt_0 \gg] \dagger P \wedge$
$[s_1]_{S<} \wedge$

$\$ref' \subseteq_u \ll cs \gg \cup_u (\ll ref_0 \gg - \ll cs \gg) \land$
$[\ll trace \gg \in_u \ll tt_0 \gg \star_{\ll cs \gg} t_1 \land \ll tt_0 \gg \upharpoonright_u \ll cs \gg =_u t_1 \upharpoonright_u \ll cs \gg]_t \land$
$\$st' =_u \$st \oplus \ll st_0 \gg$ *on* $\&ns1 \oplus \ll \sigma_1 \gg(\$st)_a$ *on* $\&ns2$)
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* =
$(\exists\ (ref_0, ref_1, st_0, st_1, tt_0, tt_1) \cdot$
$[\$ref' \mapsto_s \ll ref_0 \gg, \$st' \mapsto_s \ll st_0 \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll tt_0 \gg] \dagger P \land$
$[\$ref' \mapsto_s \ll ref_1 \gg, \$st' \mapsto_s \ll st_1 \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll tt_1 \gg] \dagger \Phi(s_1,\sigma_1,t_1) \land$
$\$ref' \subseteq_u (\ll ref_0 \gg \cup_u \ll ref_1 \gg) \cap_u \ll cs \gg \cup_u (\ll ref_0 \gg \cap_u \ll ref_1 \gg - \ll cs \gg) \land$
$\$tr \leq_u \$tr' \land$
$\&tt \in_u \ll tt_0 \gg \star_{\ll cs \gg} \ll tt_1 \gg \land \ll tt_0 \gg \upharpoonright_u \ll cs \gg =_u \ll tt_1 \gg \upharpoonright_u \ll cs \gg \land \$st' =_u \$st \oplus \ll st_0 \gg$ *on* $\&ns1 \oplus \ll st_1 \gg$ *on* $\&ns2)$
by (*simp add*: *CSPInnerMerge-form assms closure*)
also have ... = *?rhs*
by (*rel-blast*)
finally show *?thesis* **.**
qed

The result of merge two terminated stateful traces is to (1) require both state preconditions hold, (2) merge the traces using, and (3) merge the state using a parallel assignment.

lemma *FinalMerge-csp-do-left*:
assumes *vwb-lens ns1 vwb-lens ns2 ns1* $\bowtie$ *ns2 P is RR* $\$ref' \sharp P$
shows $\Phi(s_0,\sigma_0,t_0)\ [\![ns1|cs|ns2]\!]^F\ P =$
$(\exists\ (st_1, t_1) \cdot$
$[s_0]_{S<} \land$
$[\$st' \mapsto_s \ll st_1 \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_1 \gg] \dagger P \land$
$[\ll trace \gg \in_u t_0 \star_{\ll cs \gg} \ll t_1 \gg \land t_0 \upharpoonright_u \ll cs \gg =_u \ll t_1 \gg \upharpoonright_u \ll cs \gg]_t \land$
$\$st' =_u \$st \oplus \ll \sigma_0 \gg(\$st)_a$ *on* $\&ns1 \oplus \ll st_1 \gg$ *on* $\&ns2)$
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* =
$(\exists\ (st_0, st_1, tt_0, tt_1) \cdot$
$[\$st' \mapsto_s \ll st_0 \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll tt_0 \gg] \dagger \Phi(s_0,\sigma_0,t_0) \land$
$[\$st' \mapsto_s \ll st_1 \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll tt_1 \gg] \dagger RR(\exists\ \$ref' \cdot P) \land$
$\$tr \leq_u \$tr' \land \&tt \in_u \ll tt_0 \gg \star_{\ll cs \gg} \ll tt_1 \gg \land \ll tt_0 \gg \upharpoonright_u \ll cs \gg =_u \ll tt_1 \gg \upharpoonright_u \ll cs \gg \land$
$\$st' =_u \$st \oplus \ll st_0 \gg$ *on* $\&ns1 \oplus \ll st_1 \gg$ *on* $\&ns2)$
by (*simp add*: *CSPFinalMerge-form ex-unrest Healthy-if unrest closure assms*)
also have ... =
$(\exists\ (st_1, tt_1) \cdot$
$[s_0]_{S<} \land$
$[\$st' \mapsto_s \ll st_1 \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll tt_1 \gg] \dagger RR(\exists\ \$ref' \cdot P) \land$
$[\ll trace \gg \in_u t_0 \star_{\ll cs \gg} \ll tt_1 \gg \land t_0 \upharpoonright_u \ll cs \gg =_u \ll tt_1 \gg \upharpoonright_u \ll cs \gg]_t \land$
$\$st' =_u \$st \oplus \ll \sigma_0 \gg(\$st)_a$ *on* $\&ns1 \oplus \ll st_1 \gg$ *on* $\&ns2)$
by (*rel-blast*)
also have ... =
$(\exists\ (st_1, t_1) \cdot$
$[s_0]_{S<} \land$
$[\$st' \mapsto_s \ll st_1 \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_1 \gg] \dagger P \land$
$[\ll trace \gg \in_u t_0 \star_{\ll cs \gg} \ll t_1 \gg \land t_0 \upharpoonright_u \ll cs \gg =_u \ll t_1 \gg \upharpoonright_u \ll cs \gg]_t \land$
$\$st' =_u \$st \oplus \ll \sigma_0 \gg(\$st)_a$ *on* $\&ns1 \oplus \ll st_1 \gg$ *on* $\&ns2)$
by (*simp add*: *ex-unrest Healthy-if unrest closure assms*)
finally show *?thesis* **.**
qed

lemma *FinalMerge-csp-do-right*:
assumes *vwb-lens ns1 vwb-lens ns2 ns1* $\bowtie$ *ns2 P is RR* $\$ref' \sharp P$
shows P $[\![ns1|cs|ns2]\!]^F$ $\Phi(s_1,\sigma_1,t_1)$ =
$(\exists\ (st_0,\ t_0) \cdot$
$[\$st' \mapsto_s \ll st_0 \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_0 \gg] \dagger P \wedge$
$[s_1]_{S<} \wedge$
$[\ll trace \gg \in_u \ll t_0 \gg \star_{\ll cs \gg} t_1 \wedge \ll t_0 \gg \upharpoonright_u \ll cs \gg =_u t_1 \upharpoonright_u \ll cs \gg]_t \wedge$
$\$st' =_u \$st \oplus \ll st_0 \gg$ *on* $\&ns1 \oplus \ll \sigma_1 \gg (\$st)_a$ *on* $\&ns2)$
(**is** *?lhs* = *?rhs*)
proof –
have P $[\![ns1|cs|ns2]\!]^F$ $\Phi(s_1,\sigma_1,t_1) = \Phi(s_1,\sigma_1,t_1)$ $[\![ns2|cs|ns1]\!]^F$ P
by (*simp add*: *assms CSPFinalMerge-commute*)
also have ... = *?rhs*
apply (*simp add*: *FinalMerge-csp-do-left assms lens-indep-sym conj-comm*)
apply (*rel-auto*)
using *assms*(*3*) *lens-indep.lens-put-comm tr-par-sym* **apply** *fastforce+*
done
finally show *?thesis* **.**
qed

lemma *FinalMerge-csp-do*:
assumes *vwb-lens ns1 vwb-lens ns2 ns1* $\bowtie$ *ns2*
shows $\Phi(s_1,\sigma_1,t_1)$ $[\![ns1|cs|ns2]\!]^F$ $\Phi(s_2,\sigma_2,t_2)$ =
$([s_1 \wedge s_2]_{S<} \wedge [\ll trace \gg \in_u t_1 \star_{\ll cs \gg} t_2 \wedge t_1 \upharpoonright_u \ll cs \gg =_u t_2 \upharpoonright_u \ll cs \gg]_t \wedge [\langle \sigma_1\ [\&ns1|\&ns2]_s$
$\sigma_2 \rangle_a]_{S'})$
(**is** *?lhs* = *?rhs*)
proof –
have *?lhs* =
$(\exists\ (st_0,\ st_1,\ tt_0,\ tt_1) \cdot$
$[\$st' \mapsto_s \ll st_0 \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll tt_0 \gg] \dagger \Phi(s_1,\sigma_1,t_1) \wedge$
$[\$st' \mapsto_s \ll st_1 \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll tt_1 \gg] \dagger \Phi(s_2,\sigma_2,t_2) \wedge$
$\$tr \leq_u \$tr' \wedge \&tt \in_u \ll tt_0 \gg \star_{\ll cs \gg} \ll tt_1 \gg \wedge \ll tt_0 \gg \upharpoonright_u \ll cs \gg =_u \ll tt_1 \gg \upharpoonright_u \ll cs \gg \wedge$
$\$st' =_u \$st \oplus \ll st_0 \gg$ *on* $\&ns1 \oplus \ll st_1 \gg$ *on* $\&ns2)$
by (*simp add*: *CSPFinalMerge-form unrest closure assms*)
also have ... =
$([s_1 \wedge s_2]_{S<} \wedge [\ll trace \gg \in_u t_1 \star_{\ll cs \gg} t_2 \wedge t_1 \upharpoonright_u \ll cs \gg =_u t_2 \upharpoonright_u \ll cs \gg]_t \wedge [\langle \sigma_1\ [\&ns1|\&ns2]_s$
$\sigma_2 \rangle_a]_{S'})$
by (*rel-auto*)
finally show *?thesis* **.**
qed

lemma *FinalMerge-csp-do'* [*rpred*]:
assumes *vwb-lens ns1 vwb-lens ns2 ns1* $\bowtie$ *ns2*
shows $\Phi(s_1,\sigma_1,t_1)$ $[\![ns1|cs|ns2]\!]^F$ $\Phi(s_2,\sigma_2,t_2)$ =
$(\bigsqcap\ trace \mid \ll trace \gg \in_u \lceil t_1 \star_{\ll cs \gg} t_2 \rceil_{S<} \cdot$
$\Phi(s_1 \wedge s_2 \wedge t_1 \upharpoonright_u \ll cs \gg =_u t_2 \upharpoonright_u \ll cs \gg, \sigma_1\ [\&ns1|\&ns2]_s\ \sigma_2, \ll trace \gg))$
by (*simp add*: *FinalMerge-csp-do assms*, *rel-auto*)

lemma *CSPFinalMerge-UINF-ind-left* [*rpred*]:
$(\bigsqcap\ i \cdot P(i))$ $[\![ns1|cs|ns2]\!]^F$ $Q = (\bigsqcap\ i \cdot P(i)$ $[\![ns1|cs|ns2]\!]^F$ $Q)$
by (*simp add*: *CSPFinalMerge-def par-by-merge-USUP-ind-left*)

lemma *CSPFinalMerge-UINF-ind-right* [*rpred*]:
P $[\![ns1|cs|ns2]\!]^F$ $(\bigsqcap\ i \cdot Q(i)) = (\bigsqcap\ i \cdot P$ $[\![ns1|cs|ns2]\!]^F$ $Q(i))$
by (*simp add*: *CSPFinalMerge-def par-by-merge-USUP-ind-right*)

lemma *InterMerge-csp-enable*:
assumes *vwb-lens ns1 vwb-lens ns2 ns1* $\bowtie$ *ns2*
shows $\mathcal{E}(s_1,t_1,E_1)$ $[\![ns1|cs|ns2]\!]^I$ $\mathcal{E}(s_2,t_2,E_2)$ =
$([s_1 \wedge s_2]_{S<} \wedge$
$(\forall\ e{\in}\lceil(E_1 \cap_u E_2 \cap_u \ll cs\gg) \cup_u ((E_1 \cup_u E_2) - \ll cs\gg)\rceil_{S<} \cdot \ll e\gg \notin_u \$ref') \wedge$
$[\ll trace\gg \in_u t_1 \star_{\ll cs\gg} t_2 \wedge t_1 \upharpoonright_u \ll cs\gg =_u t_2 \upharpoonright_u \ll cs\gg]_t)$
(**is** *?lhs* = *?rhs*)
proof –
have *?lhs* =
$(\exists\ (ref_0, ref_1, st_0, st_1, tt_0, tt_1) \cdot$
$[\$ref' \mapsto_s \ll ref_0\gg, \$st' \mapsto_s \ll st_0\gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll tt_0\gg] \dagger \mathcal{E}(s_1,t_1,E_1) \wedge$
$[\$ref' \mapsto_s \ll ref_1\gg, \$st' \mapsto_s \ll st_1\gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll tt_1\gg] \dagger \mathcal{E}(s_2,t_2,E_2) \wedge$
$\$ref' \subseteq_u (\ll ref_0\gg \cup_u \ll ref_1\gg) \cap_u \ll cs\gg \cup_u (\ll ref_0\gg \cap_u \ll ref_1\gg - \ll cs\gg) \wedge$
$\$tr \le_u \$tr' \wedge \&tt \in_u \ll tt_0\gg \star_{\ll cs\gg} \ll tt_1\gg \wedge \ll tt_0\gg \upharpoonright_u \ll cs\gg =_u \ll tt_1\gg \upharpoonright_u \ll cs\gg)$
by (*simp add*: *CSPInterMerge-form unrest closure assms*)
also have ... =
$(\exists\ (ref_0, ref_1, tt_0, tt_1) \cdot$
$[\$ref' \mapsto_s \ll ref_0\gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll tt_0\gg] \dagger \mathcal{E}(s_1,t_1,E_1) \wedge$
$[\$ref' \mapsto_s \ll ref_1\gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll tt_1\gg] \dagger \mathcal{E}(s_2,t_2,E_2) \wedge$
$\$ref' \subseteq_u (\ll ref_0\gg \cup_u \ll ref_1\gg) \cap_u \ll cs\gg \cup_u (\ll ref_0\gg \cap_u \ll ref_1\gg - \ll cs\gg) \wedge$
$\$tr \le_u \$tr' \wedge \&tt \in_u \ll tt_0\gg \star_{\ll cs\gg} \ll tt_1\gg \wedge \ll tt_0\gg \upharpoonright_u \ll cs\gg =_u \ll tt_1\gg \upharpoonright_u \ll cs\gg)$
by (*rel-auto*)
also have ... =
$(\ [s_1 \wedge s_2]_{S<} \wedge$
$(\forall\ e{\in}\lceil(E_1 \cap_u E_2 \cap_u \ll cs\gg) \cup_u ((E_1 \cup_u E_2) - \ll cs\gg)\rceil_{S<} \cdot \ll e\gg \notin_u \$ref') \wedge$
$[\ll trace\gg \in_u t_1 \star_{\ll cs\gg} t_2 \wedge t_1 \upharpoonright_u \ll cs\gg =_u t_2 \upharpoonright_u \ll cs\gg]_t$
)
apply (*rel-auto*)
apply (*rename-tac tr st tr′ ref′*)
apply (*rule-tac x*=– $[\![E_1]\!]_e$ *st* **in** *exI*)
apply (*simp*)
apply (*rule-tac x*=– $[\![E_2]\!]_e$ *st* **in** *exI*)
apply (*auto*)
done
finally show *?thesis* **.**
qed

lemma *InterMerge-csp-enable′* [*rpred*]:
assumes *vwb-lens ns1 vwb-lens ns2 ns1* $\bowtie$ *ns2*
shows $\mathcal{E}(s_1,t_1,E_1)$ $[\![ns1|cs|ns2]\!]^I$ $\mathcal{E}(s_2,t_2,E_2)$ =
$(\bigsqcap\ trace \mid \ll trace\gg \in_u \lceil t_1 \star_{\ll cs\gg} t_2\rceil_{S<} \cdot$
$\mathcal{E}(\ s_1 \wedge s_2 \wedge t_1 \upharpoonright_u \ll cs\gg =_u t_2 \upharpoonright_u \ll cs\gg$
$, \ll trace\gg$
$, (E_1 \cap_u E_2 \cap_u \ll cs\gg) \cup_u ((E_1 \cup_u E_2) - \ll cs\gg)))$
by (*simp add*: *InterMerge-csp-enable assms*, *rel-auto*)

lemma *InterMerge-csp-enable-csp-do*:
assumes *vwb-lens ns1 vwb-lens ns2 ns1* $\bowtie$ *ns2*
shows $\mathcal{E}(s_1,t_1,E_1)$ $[\![ns1|cs|ns2]\!]^I$ $\Phi(s_2,\sigma_2,t_2)$ =
$([s_1 \wedge s_2]_{S<} \wedge (\forall\ e{\in}\lceil(E_1 - \ll cs\gg)\rceil_{S<} \cdot \ll e\gg \notin_u \$ref') \wedge$
$[\ll trace\gg \in_u t_1 \star_{\ll cs\gg} t_2 \wedge t_1 \upharpoonright_u \ll cs\gg =_u t_2 \upharpoonright_u \ll cs\gg]_t)$
(**is** *?lhs* = *?rhs*)
proof –
have *?lhs* =

$(\exists\ (ref_0,\ ref_1,\ st_0,\ st_1,\ tt_0,\ tt_1) \cdot$
$[\$ref\,' \mapsto_s \ll ref_0 \gg,\ \$st\,' \mapsto_s \ll st_0 \gg,\ \$tr \mapsto_s \langle\rangle,\ \$tr\,' \mapsto_s \ll tt_0 \gg] \dagger \mathcal{E}(s_1,t_1,E_1) \wedge$
$[\$ref\,' \mapsto_s \ll ref_1 \gg,\ \$st\,' \mapsto_s \ll st_1 \gg,\ \$tr \mapsto_s \langle\rangle,\ \$tr\,' \mapsto_s \ll tt_1 \gg] \dagger \Phi(s_2,\sigma_2,t_2) \wedge$
$\$ref\,' \subseteq_u (\ll ref_0 \gg \cup_u \ll ref_1 \gg) \cap_u \ll cs \gg \cup_u (\ll ref_0 \gg \cap_u \ll ref_1 \gg - \ll cs \gg) \wedge$
$\$tr \le_u \$tr\,' \wedge \&tt \in_u \ll tt_0 \gg \star_{\ll cs \gg} \ll tt_1 \gg \wedge \ll tt_0 \gg \upharpoonright_u \ll cs \gg =_u \ll tt_1 \gg \upharpoonright_u \ll cs \gg)$
by (*simp add*: *CSPInterMerge-form unrest closure assms*)
also have ... =
$(\exists\ (ref_0,\ ref_1,\ tt_0) \cdot$
$[\$ref\,' \mapsto_s \ll ref_0 \gg,\ \$tr \mapsto_s \langle\rangle,\ \$tr\,' \mapsto_s \ll tt_0 \gg] \dagger \mathcal{E}(s_1,t_1,E_1) \wedge$
$[s_2]_{S<} \wedge$
$\$ref\,' \subseteq_u (\ll ref_0 \gg \cup_u \ll ref_1 \gg) \cap_u \ll cs \gg \cup_u (\ll ref_0 \gg \cap_u \ll ref_1 \gg - \ll cs \gg) \wedge$
$[\ll trace \gg \in_u t_1 \star_{\ll cs \gg} t_2 \wedge t_1 \upharpoonright_u \ll cs \gg =_u t_2 \upharpoonright_u \ll cs \gg]_t)$
by (*rel-auto*)
also have ... = $([s_1 \wedge s_2]_{S<} \wedge (\forall\ e \in \lceil (E_1 - \ll cs \gg) \rceil_{S<} \cdot \ll e \gg \notin_u \$ref\,') \wedge$
$[\ll trace \gg \in_u t_1 \star_{\ll cs \gg} t_2 \wedge t_1 \upharpoonright_u \ll cs \gg =_u t_2 \upharpoonright_u \ll cs \gg]_t)$
by (*rel-auto*)
finally show *?thesis* **.**
qed

lemma *InterMerge-csp-enable-csp-do′* [*rpred*]:
assumes *vwb-lens ns1 vwb-lens ns2 ns1* $\bowtie$ *ns2*
shows $\mathcal{E}(s_1,t_1,E_1)\ [\![ns1|cs|ns2]\!]^I\ \Phi(s_2,\sigma_2,t_2) =$
$(\bigsqcap\ trace \mid \ll trace \gg \in_u \lceil t_1 \star_{\ll cs \gg} t_2 \rceil_{S<} \cdot$
$\mathcal{E}(s_1 \wedge s_2 \wedge t_1 \upharpoonright_u \ll cs \gg =_u t_2 \upharpoonright_u \ll cs \gg,\ \ll trace \gg,\ E_1 - \ll cs \gg))$
by (*simp add*: *InterMerge-csp-enable-csp-do assms*, *rel-auto*)

lemma *InterMerge-csp-do-csp-enable*:
assumes *vwb-lens ns1 vwb-lens ns2 ns1* $\bowtie$ *ns2*
shows $\Phi(s_1,\sigma_1,t_1)\ [\![ns1|cs|ns2]\!]^I\ \mathcal{E}(s_2,t_2,E_2) =$
$([s_1 \wedge s_2]_{S<} \wedge (\forall\ e \in \lceil (E_2 - \ll cs \gg) \rceil_{S<} \cdot \ll e \gg \notin_u \$ref\,') \wedge$
$[\ll trace \gg \in_u t_1 \star_{\ll cs \gg} t_2 \wedge t_1 \upharpoonright_u \ll cs \gg =_u t_2 \upharpoonright_u \ll cs \gg]_t)$
(**is** *?lhs* = *?rhs*)
proof −
have $\Phi(s_1,\sigma_1,t_1)\ [\![ns1|cs|ns2]\!]^I\ \mathcal{E}(s_2,t_2,E_2) = \mathcal{E}(s_2,t_2,E_2)\ [\![ns2|cs|ns1]\!]^I\ \Phi(s_1,\sigma_1,t_1)$
by (*simp add*: *CSPInterMerge-commute assms*)
also have ... = *?rhs*
by (*simp add*: *InterMerge-csp-enable-csp-do assms lens-indep-sym trace-merge-commute conj-comm eq-upred-sym*)
finally show *?thesis* **.**
qed

lemma *InterMerge-csp-do-csp-enable′* [*rpred*]:
assumes *vwb-lens ns1 vwb-lens ns2 ns1* $\bowtie$ *ns2*
shows $\Phi(s_1,\sigma_1,t_1)\ [\![ns1|cs|ns2]\!]^I\ \mathcal{E}(s_2,t_2,E_2) =$
$(\bigsqcap\ trace \mid \ll trace \gg \in_u \lceil t_1 \star_{\ll cs \gg} t_2 \rceil_{S<} \cdot$
$\mathcal{E}(s_1 \wedge s_2 \wedge t_1 \upharpoonright_u \ll cs \gg =_u t_2 \upharpoonright_u \ll cs \gg,\ \ll trace \gg,\ E_2 - \ll cs \gg))$
by (*simp add*: *InterMerge-csp-do-csp-enable assms*, *rel-auto*)

lemma *CSPInterMerge-or-left* [*rpred*]:
$(P \vee Q)\ [\![ns1|cs|ns2]\!]^I\ R = (P\ [\![ns1|cs|ns2]\!]^I\ R \vee Q\ [\![ns1|cs|ns2]\!]^I\ R)$
by (*simp add*: *CSPInterMerge-def par-by-merge-or-left*)

lemma *CSPInterMerge-or-right* [*rpred*]:
$P\ [\![ns1|cs|ns2]\!]^I\ (Q \vee R) = (P\ [\![ns1|cs|ns2]\!]^I\ Q \vee P\ [\![ns1|cs|ns2]\!]^I\ R)$
by (*simp add*: *CSPInterMerge-def par-by-merge-or-right*)

lemma *CSPInterMerge-UINF-ind-left* [*rpred*]:
$(\bigsqcap\ i \cdot P(i))\ [\![ns1|cs|ns2]\!]^I\ Q = (\bigsqcap\ i \cdot P(i)\ [\![ns1|cs|ns2]\!]^I\ Q)$
by (*simp add*: *CSPInterMerge-def par-by-merge-USUP-ind-left*)

lemma *CSPInterMerge-UINF-ind-right* [*rpred*]:
$P\ [\![ns1|cs|ns2]\!]^I\ (\bigsqcap\ i \cdot Q(i)) = (\bigsqcap\ i \cdot P\ [\![ns1|cs|ns2]\!]^I\ Q(i))$
by (*simp add*: *CSPInterMerge-def par-by-merge-USUP-ind-right*)

lemma *par-by-merge-seq-remove*: $(P \parallel_{M\ ;;\ R}\ Q) = (P \parallel_M Q)\ ;;\ R$
by (*simp add*: *par-by-merge-seq-add*[*THEN sym*])

lemma *merge-csp-do-right*:
assumes *vwb-lens ns1 vwb-lens ns2 ns1* $\bowtie$ *ns2 P is RC*
shows $\Phi(s_1,\sigma_1,t_1)\ wr[ns1|cs|ns2]_C\ P = undefined$
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* =
$(\neg_r\ (\exists\ (ref_0,\ st_0,\ tt_0) \cdot$
$[\$ref' \mapsto_s \ll ref_0\gg, \$st' \mapsto_s \ll st_0\gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll tt_0\gg] \dagger (\neg_r\ RC(P)) \wedge$
$[s_1]_{S<} \wedge$
$\$ref' \subseteq_u \ll cs\gg \cup_u (\ll ref_0\gg - \ll cs\gg) \wedge$
$[\ll trace\gg \in_u \ll tt_0\gg \star_{\ll cs\gg}\ t_1 \wedge \ll tt_0\gg \upharpoonright_u \ll cs\gg =_u t_1 \upharpoonright_u \ll cs\gg]_t \wedge$
$\$st' =_u \$st \oplus \ll st_0\gg\ on\ \&ns1 \oplus \ll\sigma_1\gg(\$st)_a\ on\ \&ns2)\ ;;\ R1\ true)$
by (*simp add*: *wrR-def par-by-merge-seq-remove merge-csp-do-right closure assms Healthy-if rpred*)
also have ... =
$(\neg_r\ (\exists\ (ref_0,\ st_0,\ tt_0) \cdot$
$[\$ref' \mapsto_s \ll ref_0\gg, \$st' \mapsto_s \ll st_0\gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll tt_0\gg] \dagger (\neg_r\ RC(P)) \wedge$
$[s_1]_{S<} \wedge$
$\$ref' \subseteq_u \ll cs\gg \cup_u (\ll ref_0\gg - \ll cs\gg) \wedge$
$[\ll trace\gg \in_u \ll tt_0\gg \star_{\ll cs\gg}\ t_1 \wedge \ll tt_0\gg \upharpoonright_u \ll cs\gg =_u t_1 \upharpoonright_u \ll cs\gg]_t\ ;;\ true_r \wedge$
$\$st' =_u \$st \oplus \ll st_0\gg\ on\ \&ns1 \oplus \ll\sigma_1\gg(\$st)_a\ on\ \&ns2))$
apply (*rel-auto*)

oops

12.2 Parallel operator

syntax
-par-circus :: *logic* ⇒ *salpha* ⇒ *logic* ⇒ *salpha* ⇒ *logic* ⇒ *logic* (- ⟦-‖-‖-⟧ - [*75*,*0*,*0*,*0*,*76*] *76*)
-par-csp :: *logic* ⇒ *logic* ⇒ *logic* ⇒ *logic* (- ⟦-⟧$_C$ - [*75*,*0*,*76*] *76*)
-inter-circus :: *logic* ⇒ *salpha* ⇒ *salpha* ⇒ *logic* ⇒ *logic* (- ⟦-‖-⟧ - [*75*,*0*,*0*,*76*] *76*)
-inter-csp :: *logic* ⇒ *logic* ⇒ *logic* (**infixr** ||| *75*)

translations
-par-circus P ns1 cs ns2 Q == $P \parallel_{M_C}$ *ns1 cs ns2 Q*
-par-csp P cs Q == *-par-circus P* 0_L *cs* 0_L *Q*
-inter-circus P ns1 ns2 Q == *-par-circus P ns1* {} *ns2 Q*
-inter-csp P Q == *-par-csp P* {} *Q*

definition *CSP5* :: ('σ, 'φ) *action* ⇒ ('σ, 'φ) *action* **where**
[*upred-defs*]: *CSP5*(*P*) = (*P* ||| *Skip*)

definition *C2* :: ('σ, 'φ) *action* ⇒ ('σ, 'φ) *action* **where**
[*upred-defs*]: $C2(P) = (P\ [\![\Sigma\|\{\}\|\emptyset]\!]\ Skip)$

lemma *Skip-right-form*:
assumes P_1 *is RC* P_2 *is RR* P_3 *is RR* $\$st' \sharp P_2$
shows $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) \;;;\; Skip = \mathbf{R}_s(P_1 \vdash P_2 \diamond (\exists\ \$ref' \cdot P_3))$
proof −
have $1{:}RR(P_3) \;;;\; \Phi(true,id,\langle\rangle) = (\exists\ \$ref' \cdot RR(P_3))$
by (*rel-auto*)
show *?thesis*
by (*rdes-simp cls*: *assms*, *metis 1 Healthy-if assms*(*3*))
qed

lemma *ParCSP-rdes-def* [*rdes-def*]:
fixes $P_1 :: ('s,'e)$ *action*
assumes P_1 *is CRC* Q_1 *is CRC* P_2 *is CRR* Q_2 *is CRR* P_3 *is CRR* Q_3 *is CRR*
$\$st' \sharp P_2$ $\$st' \sharp Q_2$
$ns1 \bowtie ns2$
shows $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3)\ [\![ns1\|cs\|ns2]\!]\ \mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3) =$
$\mathbf{R}_s\ (((Q_1 \Rightarrow_r Q_2)\ wr[ns1|cs|ns2]_C\ P_1\ \wedge$
$(Q_1 \Rightarrow_r Q_3)\ wr[ns1|cs|ns2]_C\ P_1\ \wedge$
$(P_1 \Rightarrow_r P_2)\ wr[ns2|cs|ns1]_C\ Q_1\ \wedge$
$(P_1 \Rightarrow_r P_3)\ wr[ns2|cs|ns1]_C\ Q_1) \vdash$
$((P_1 \Rightarrow_r P_2)\ [\![ns1|cs|ns2]\!]^I\ (Q_1 \Rightarrow_r Q_2)\ \vee$
$(P_1 \Rightarrow_r P_3)\ [\![ns1|cs|ns2]\!]^I\ (Q_1 \Rightarrow_r Q_2)\ \vee$
$(P_1 \Rightarrow_r P_2)\ [\![ns1|cs|ns2]\!]^I\ (Q_1 \Rightarrow_r Q_3)) \diamond$
$((P_1 \Rightarrow_r P_3)\ [\![ns1|cs|ns2]\!]^F\ (Q_1 \Rightarrow_r Q_3)))$
(**is** $?P\ [\![ns1\|cs\|ns2]\!]\ ?Q = ?rhs$)
proof −
have $?P\ [\![ns1\|cs\|ns2]\!]\ ?Q = (?P \parallel_{M_R(N_C\ ns1\ cs\ ns2)}\ ?Q) \;;;_h\ Skip$
by (*simp add*: *CSPMerge-def par-by-merge-seq-add*)
also
have ... = $\mathbf{R}_s\ (((Q_1 \Rightarrow_r Q_2)\ wr[ns1|cs|ns2]_C\ P_1\ \wedge$
$(Q_1 \Rightarrow_r Q_3)\ wr[ns1|cs|ns2]_C\ P_1\ \wedge$
$(P_1 \Rightarrow_r P_2)\ wr[ns2|cs|ns1]_C\ Q_1\ \wedge$
$(P_1 \Rightarrow_r P_3)\ wr[ns2|cs|ns1]_C\ Q_1) \vdash$
$((P_1 \Rightarrow_r P_2)\ [\![ns1|cs|ns2]\!]^I\ (Q_1 \Rightarrow_r Q_2)\ \vee$
$(P_1 \Rightarrow_r P_3)\ [\![ns1|cs|ns2]\!]^I\ (Q_1 \Rightarrow_r Q_2)\ \vee$
$(P_1 \Rightarrow_r P_2)\ [\![ns1|cs|ns2]\!]^I\ (Q_1 \Rightarrow_r Q_3)) \diamond$
$(P_1 \Rightarrow_r P_3) \parallel_{N_C\ ns1\ cs\ ns2} (Q_1 \Rightarrow_r Q_3)) \;;;_h\ Skip$
by (*simp add*: *parallel-rdes-def swap-CSPInnerMerge CSPInterMerge-def closure assms*)
also
have ... = $\mathbf{R}_s\ (((Q_1 \Rightarrow_r Q_2)\ wr[ns1|cs|ns2]_C\ P_1\ \wedge$
$(Q_1 \Rightarrow_r Q_3)\ wr[ns1|cs|ns2]_C\ P_1\ \wedge$
$(P_1 \Rightarrow_r P_2)\ wr[ns2|cs|ns1]_C\ Q_1\ \wedge$
$(P_1 \Rightarrow_r P_3)\ wr[ns2|cs|ns1]_C\ Q_1) \vdash$
$((P_1 \Rightarrow_r P_2)\ [\![ns1|cs|ns2]\!]^I\ (Q_1 \Rightarrow_r Q_2)\ \vee$
$(P_1 \Rightarrow_r P_3)\ [\![ns1|cs|ns2]\!]^I\ (Q_1 \Rightarrow_r Q_2)\ \vee$
$(P_1 \Rightarrow_r P_2)\ [\![ns1|cs|ns2]\!]^I\ (Q_1 \Rightarrow_r Q_3)) \diamond$
$(\exists\ \$ref' \cdot ((P_1 \Rightarrow_r P_3) \parallel_{N_C\ ns1\ cs\ ns2} (Q_1 \Rightarrow_r Q_3))))$
by (*simp add*: *Skip-right-form closure parallel-RR-closed assms unrest*)
also
have ... = $\mathbf{R}_s\ (((Q_1 \Rightarrow_r Q_2)\ wr[ns1|cs|ns2]_C\ P_1\ \wedge$
$(Q_1 \Rightarrow_r Q_3)\ wr[ns1|cs|ns2]_C\ P_1\ \wedge$
$(P_1 \Rightarrow_r P_2)\ wr[ns2|cs|ns1]_C\ Q_1\ \wedge$
$(P_1 \Rightarrow_r P_3)\ wr[ns2|cs|ns1]_C\ Q_1) \vdash$
$((P_1 \Rightarrow_r P_2)\ [\![ns1|cs|ns2]\!]^I\ (Q_1 \Rightarrow_r Q_2)\ \vee$

$(P_1 \Rightarrow_r P_3)$ $[\![ns1|cs|ns2]\!]^I$ $(Q_1 \Rightarrow_r Q_2) \vee$
$(P_1 \Rightarrow_r P_2)$ $[\![ns1|cs|ns2]\!]^I$ $(Q_1 \Rightarrow_r Q_3)) \diamond$
$((P_1 \Rightarrow_r P_3)$ $[\![ns1|cs|ns2]\!]^F$ $(Q_1 \Rightarrow_r Q_3)))$
proof −
have $(\exists\ \$ref' \cdot ((P_1 \Rightarrow_r P_3) \parallel_{N_C\ ns1\ cs\ ns2} (Q_1 \Rightarrow_r Q_3))) = ((P_1 \Rightarrow_r P_3)\ [\![ns1|cs|ns2]\!]^F\ (Q_1 \Rightarrow_r Q_3))$
by (*rel-blast*)
thus *?thesis* **by** *simp*
qed
finally show *?thesis* **.**
qed

12.3 Parallel Laws

lemma *ParCSP-expand*:
$P\ [\![ns1\|cs\|ns2]\!]\ Q = (P \parallel_{RN_C\ ns1\ cs\ ns2} Q)$;; *Skip*
by (*simp add*: *CSPMerge-def par-by-merge-seq-add*)

lemma *parallel-is-CSP* [*closure*]:
assumes *P is CSP Q is CSP*
shows $(P\ [\![ns1\|cs\|ns2]\!]\ Q)$ *is CSP*
proof −
have $(P \parallel_{M_R(N_C\ ns1\ cs\ ns2)} Q)$ *is CSP*
by (*simp add*: *closure assms*)
hence $(P \parallel_{M_R(N_C\ ns1\ cs\ ns2)} Q)$;; *Skip is CSP*
by (*simp add*: *closure*)
thus *?thesis*
by (*simp add*: *CSPMerge-def par-by-merge-seq-add*)
qed

lemma *parallel-is-CSP3* [*closure*]:
assumes *P is CSP P is CSP3 Q is CSP Q is CSP3*
shows $(P\ [\![ns1\|cs\|ns2]\!]\ Q)$ *is CSP3*
proof −
have $(P \parallel_{M_R(N_C\ ns1\ cs\ ns2)} Q)$ *is CSP*
by (*simp add*: *closure assms*)
hence $(P \parallel_{M_R(N_C\ ns1\ cs\ ns2)} Q)$;; *Skip is CSP*
by (*simp add*: *closure*)
thus *?thesis*
oops

theorem *parallel-commutative*:
assumes $ns1 \bowtie ns2$
shows $(P\ [\![ns1\|cs\|ns2]\!]\ Q) = (Q\ [\![ns2\|cs\|ns1]\!]\ P)$
proof −
have $(P\ [\![ns1\|cs\|ns2]\!]\ Q) = P \parallel_{swap_m\ ;;\ (M_C\ ns2\ cs\ ns1)} Q$
by (*simp add*: *CSPMerge-def seqr-assoc*[*THEN sym*] *swap-merge-rd swap-CSPInnerMerge lens-indep-sym assms*)
also have ... $= Q\ [\![ns2\|cs\|ns1]\!]\ P$
by (*metis par-by-merge-commute-swap*)
finally show *?thesis* **.**
qed

lemma *interleave-commute*:
$P\ |||\ Q = Q\ |||\ P$

using *parallel-commutative zero-lens-indep* **by** *blast*

The form of C2 tells us that a normal CSP process has a downward closed set of refusals

lemma *C2-form*:
assumes P *is NCSP*
shows $C2(P) = \mathbf{R}_s\ (pre_R\ P \vdash (\exists\ ref_0 \cdot peri_R\ P[\![\ll ref_0 \gg/\$ref']\!] \wedge \$ref' \subseteq_u \ll ref_0 \gg) \diamond post_R\ P)$
proof −
have $1{:}\Phi(true,id,\langle\rangle)\ wr[\Sigma|\{\}|\emptyset]_C\ pre_R\ P = pre_R\ P$ (**is** *?lhs* = *?rhs*)
proof −
have $?lhs = (\neg_r\ (\exists\ (ref_0,\ st_0,\ tt_0) \cdot$
$[\$ref' \mapsto_s \ll ref_0 \gg,\ \$st' \mapsto_s \ll st_0 \gg,\ \$tr \mapsto_s \langle\rangle,\ \$tr' \mapsto_s \ll tt_0 \gg] \dagger (\exists\ \$ref';\$st' \cdot RR(\neg_r\ pre_R\ P)) \wedge$
$\$ref' \subseteq_u \ll ref_0 \gg \wedge [\ll trace \gg =_u \ll tt_0 \gg]_t \wedge$
$\$st' =_u \$st \oplus \ll st_0 \gg\ on\ \Sigma \oplus \ll id \gg(\$st)_a\ on\ \emptyset)\ ;;\ R1\ true)$
by (*simp add*: *wrR-def par-by-merge-seq-remove rpred merge-csp-do-right ex-unrest Healthy-if pr-var-def closure assms unrest usubst*)
also have ... $= (\neg_r\ (\exists\ \$ref';\$st' \cdot RR(\neg_r\ pre_R\ P))\ ;;\ R1\ true)$
by (*rel-auto*)
also have ... $= (\neg_r\ (\neg_r\ pre_R\ P)\ ;;\ R1\ true)$
by (*simp add*: *Healthy-if closure ex-unrest unrest assms*)
also have ... $= pre_R\ P$
by (*simp add*: *NCSP-implies-NSRD NSRD-neg-pre-unit R1-preR assms rea-not-not*)
finally show *?thesis* **.**
qed
have 2: $(pre_R\ P \Rightarrow_r peri_R\ P)\ [\![\Sigma|\{\}|\emptyset]\!]^I\ \Phi(true,id,\langle\rangle) =$
$(\exists\ ref_0 \cdot (peri_R\ P)[\![\ll ref_0 \gg/\$ref']\!] \wedge \$ref' \subseteq_u \ll ref_0 \gg)$ (**is** *?lhs* = *?rhs*)
proof −
have $?lhs = peri_R\ P\ [\![\Sigma|\{\}|\emptyset]\!]^I\ \Phi(true,id,\langle\rangle)$
by (*simp add*: *SRD-peri-under-pre closure assms unrest*)
also have ... $= (\exists\ \$st' \cdot (peri_R\ P \parallel_{N_C\ 1_L\ \{\}\ 0_L} \Phi(true,id,\langle\rangle)))$
by (*simp add*: *CSPInterMerge-def par-by-merge-def seqr-exists-right*)
also have ... $=$
$(\exists\ \$st' \cdot \exists\ (ref_0,\ st_0,\ tt_0) \cdot$
$[\$ref' \mapsto_s \ll ref_0 \gg,\ \$st' \mapsto_s \ll st_0 \gg,\ \$tr \mapsto_s \langle\rangle,\ \$tr' \mapsto_s \ll tt_0 \gg] \dagger (\exists\ \$st' \cdot RR(peri_R\ P)) \wedge$
$\$ref' \subseteq_u \ll ref_0 \gg \wedge [\ll trace \gg =_u \ll tt_0 \gg]_t \wedge \$st' =_u \$st \oplus \ll st_0 \gg\ on\ \Sigma \oplus \ll id \gg(\$st)_a\ on\ \emptyset)$
by (*simp add*: *merge-csp-do-right pr-var-def assms Healthy-if assms closure rpred unrest ex-unrest*)
also have ... $=$
$(\exists\ ref_0 \cdot (\exists\ \$st' \cdot RR(peri_R\ P))[\![\ll ref_0 \gg/\$ref']\!] \wedge \$ref' \subseteq_u \ll ref_0 \gg)$
by (*rel-auto*)
also have ... = *?rhs*
by (*simp add*: *closure ex-unrest Healthy-if unrest assms*)
finally show *?thesis* **.**
qed
have 3: $(pre_R\ P \Rightarrow_r post_R\ P)\ [\![\Sigma|\{\}|\emptyset]\!]^F\ \Phi(true,id,\langle\rangle) = post_R(P)$ (**is** *?lhs* = *?rhs*)
proof −
have $?lhs = post_R\ P\ [\![\Sigma|\{\}|\emptyset]\!]^F\ \Phi(true,id,\langle\rangle)$
by (*simp add*: *SRD-post-under-pre closure assms unrest*)
also have ... $= (\exists\ (st_0,\ t_0) \cdot$
$[\$st' \mapsto_s \ll st_0 \gg,\ \$tr \mapsto_s \langle\rangle,\ \$tr' \mapsto_s \ll t_0 \gg] \dagger RR(post_R\ P) \wedge$
$[\ll trace \gg =_u \ll t_0 \gg]_t \wedge \$st' =_u \$st \oplus \ll st_0 \gg\ on\ \Sigma \oplus \ll id \gg(\$st)_a\ on\ \emptyset)$
by (*simp add*: *FinalMerge-csp-do-right pr-var-def assms closure unrest rpred Healthy-if*)
also have ... $= RR(post_R(P))$
by (*rel-auto*)
finally show *?thesis*
by (*simp add*: *Healthy-if assms closure*)

qed
show *?thesis*
proof −
have $C2(P) = \mathbf{R}_s\ (\Phi(true,id,\langle\rangle)\ wr[\Sigma|\{\}|\emptyset]_C\ pre_R\ P \vdash$
$(pre_R\ P \Rightarrow_r peri_R\ P)\ [\![\Sigma|\{\}|\emptyset]\!]^I\ \Phi(true,id,\langle\rangle) \diamond (pre_R\ P \Rightarrow_r post_R\ P)\ [\![\Sigma|\{\}|\emptyset]\!]^F\ \Phi(true,id,\langle\rangle))$
by (*simp add*: *C2-def*, *rdes-simp cls*: *assms*, *simp add*: *id-def pr-var-def*)
also have ... $= \mathbf{R}_s\ (pre_R\ P \vdash (\exists\ ref_0 \cdot peri_R\ P[\![\ll ref_0\gg/\$ref']\!] \wedge \$ref' \subseteq_u \ll ref_0\gg) \diamond post_R\ P)$
by (*simp add*: *1 2 3*)
finally show *?thesis* **.**
qed
qed

lemma *Skip-C2-closed* [*closure*]:
Skip is C2
apply (*simp add*: *Healthy-def C2-form*)
apply (*simp add*: *C2-form closure rdes usubst*)
apply (*simp add*: *rdes-def*)
done

lemma *ref-down-CRR* [*closure*]:
assumes *P is NCSP*
shows $(\exists\ ref_0 \cdot peri_R\ P[\![\ll ref_0\gg/\$ref']\!] \wedge \$ref' \subseteq_u \ll ref_0\gg)$ *is CRR*
proof −
have $(\exists\ ref_0 \cdot peri_R\ P[\![\ll ref_0\gg/\$ref']\!] \wedge \$ref' \subseteq_u \ll ref_0\gg) =$
$(\exists\ ref_0 \cdot (CRR(peri_R\ P))[\![\ll ref_0\gg/\$ref']\!] \wedge \$ref' \subseteq_u \ll ref_0\gg)$
by (*simp add*: *Healthy-if assms closure*)
also have ... $= CRR(\exists\ ref_0 \cdot peri_R\ P[\![\ll ref_0\gg/\$ref']\!] \wedge \$ref' \subseteq_u \ll ref_0\gg)$
by (*rel-auto*)
finally show *?thesis*
by (*simp add*: *Healthy-def'*)
qed

lemma *C2-idem*:
assumes *P is NCSP*
shows $C2(C2(P)) = C2(P)$ (**is** *?lhs = ?rhs*)
proof −
have $?lhs = \mathbf{R}_s\ (pre_R\ P \vdash (\exists\ ref_0 \cdot (pre_R\ P \Rightarrow_r (\exists\ ref_0' \cdot peri_R\ P[\![\ll ref_0'\gg/\$ref']\!] \wedge \ll ref_0\gg \subseteq_u \ll ref_0'\gg)) \wedge \$ref' \subseteq_u \ll ref_0\gg) \diamond post_R\ P)$
by (*simp add*: *C2-form closure unrest rdes SRD-post-under-pre SRD-peri-under-pre usubst NCSP-rdes-intro assms*)
also have
... $= \mathbf{R}_s\ (pre_R\ P \vdash (\exists\ ref_0 \cdot (\exists\ ref_0' \cdot peri_R\ P[\![\ll ref_0'\gg/\$ref']\!] \wedge \ll ref_0\gg \subseteq_u \ll ref_0'\gg) \wedge \$ref' \subseteq_u \ll ref_0\gg) \diamond post_R\ P)$
by (*rel-auto*)
also have
... $= \mathbf{R}_s\ (pre_R\ P \vdash (\exists\ ref_0 \cdot peri_R\ P[\![\ll ref_0\gg/\$ref']\!] \wedge \$ref' \subseteq_u \ll ref_0\gg) \diamond post_R\ P)$
by (*rel-auto*)
also have ... $= C2(P)$
by (*simp add*: *C2-form closure unrest assms*)
finally show *?thesis* **.**
qed

lemma *Stop-C2-closed* [*closure*]:
Stop is C2
apply (*simp add*: *Healthy-def C2-form*)

apply (*simp add*: *C2-form closure rdes usubst*)
apply (*rel-auto*)
done

lemma *Miracle-C2-closed* [*closure*]:
Miracle is C2
apply (*simp add*: *Healthy-def C2-form*)
apply (*simp add*: *C2-form closure rdes usubst*)
apply (*simp add*: *rdes-def*)
done

lemma *Chaos-C2-closed* [*closure*]:
Chaos is C2
apply (*simp add*: *Healthy-def C2-form*)
apply (*simp add*: *C2-form closure rdes usubst unrest*)
apply (*simp add*: *rdes-def*)
apply (*rel-auto*)
done

lemma
assumes *vwb-lens ns1 vwb-lens ns2 ns1* $\bowtie$ *ns2 P is RR*
shows $P\ wr[ns1|cs|ns2]_C\ false = undefined$ (**is** *?lhs* = *?rhs*)
proof −
have $?lhs = (\neg_r\ (\exists\ (ref_0, ref_1, st_0, st_1, tt_0, tt_1) \cdot$
$[\$ref\acute{}\ \mapsto_s \ll ref_0\gg, \$st\acute{}\ \mapsto_s \ll st_0\gg, \$tr \mapsto_s \langle\rangle, \$tr\acute{}\ \mapsto_s \ll tt_0\gg] \dagger R1\ true\ \wedge$
$[\$ref\acute{}\ \mapsto_s \ll ref_1\gg, \$st\acute{}\ \mapsto_s \ll st_1\gg, \$tr \mapsto_s \langle\rangle, \$tr\acute{}\ \mapsto_s \ll tt_1\gg] \dagger P\ \wedge$
$\$ref\acute{}\ \subseteq_u (\ll ref_0\gg \cup_u \ll ref_1\gg) \cap_u \ll cs\gg \cup_u (\ll ref_0\gg \cap_u \ll ref_1\gg - \ll cs\gg)\ \wedge$
$\$tr \leq_u \$tr\acute{}\ \wedge$
$\&tt \in_u \ll tt_0\gg \star_{\ll cs\gg} \ll tt_1\gg \wedge \ll tt_0\gg \restriction_u \ll cs\gg =_u \ll tt_1\gg \restriction_u \ll cs\gg\ \wedge$
$\$st\acute{}\ =_u \$st \oplus \ll st_0\gg\ on\ \&ns1 \oplus \ll st_1\gg\ on\ \&ns2)\ ;;$
$R1\ true)$
by (*simp add*: *wrR-def par-by-merge-seq-remove CSPInnerMerge-form assms closure usubst unrest*)
also have ... $= (\neg_r\ (\exists\ (tt_0, tt_1) \cdot$
$[\$tr \mapsto_s \langle\rangle, \$tr\acute{}\ \mapsto_s \ll tt_1\gg] \dagger P\ \wedge$
$\$tr \leq_u \$tr\acute{}\ \wedge$
$\&tt \in_u \ll tt_0\gg \star_{\ll cs\gg} \ll tt_1\gg \wedge \ll tt_0\gg \restriction_u \ll cs\gg =_u \ll tt_1\gg \restriction_u \ll cs\gg)\ ;;$
$R1\ true)$
by (*rel-blast*)
also have ... $= (\neg_r\ (\exists\ (tt_0, tt_1) \cdot$
$[\$tr \mapsto_s \langle\rangle, \$tr\acute{}\ \mapsto_s \ll tt_1\gg] \dagger RR(P)\ \wedge$
$\$tr \leq_u \$tr\acute{}\ \wedge$
$\&tt \in_u \ll tt_0\gg \star_{\ll cs\gg} \ll tt_1\gg \wedge \ll tt_0\gg \restriction_u \ll cs\gg =_u \ll tt_1\gg \restriction_u \ll cs\gg)\ ;;$
$R1\ true)$
by (*simp add*: *Healthy-if assms*)
oops

end

13 Linking to the Failures-Divergences Model

theory *utp-circus-fdsem*
imports *utp-circus-parallel utp-circus-recursion*
begin

13.1 Failures-Divergences Semantics

The following functions play a similar role to those in Roscoe's CSP semantics, and are calculated from the Circus reactive design semantics. A major difference is that these three functions account for state. Each divergence, trace, and failure is subject to an initial state. Moreover, the traces are terminating traces, and therefore also provide a final state following the given interaction. A more subtle difference from the Roscoe semantics is that the set of traces do not include the divergences. The same semantic information is present, but we construct a direct analogy with the pre-, peri- and postconditions of our reactive designs.

definition *divergences* :: $('\sigma,'\varphi)$ *action* $\Rightarrow$ $'\sigma \Rightarrow$ $'\varphi$ *list set* $(dv[\![-]\!]$- $[0,100]$ $100)$ **where**
[*upred-defs*]: *divergences* P s = $\{t \mid t.$ ‘$(\neg_r$ $pre_R(P))[\![\ll s\gg,\langle\rangle,\ll t\gg/\$st,\$tr,\$tr']\!]$‘$\}$

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

definition *failures* :: $('\sigma,'\varphi)$ *action* $\Rightarrow$ $'\sigma \Rightarrow$ $('\varphi$ *list* $\times$ $'\varphi$ *set*$)$ *set* $(fl[\![-]\!]$- $[0,100]$ $100)$ **where**
[*upred-defs*]: *failures* P s = $\{(t,r) \mid t\ r.$ ‘$(pre_R(P) \wedge peri_R(P))[\![\ll r\gg,\ll s\gg,\langle\rangle,\ll t\gg/\$ref',\$st,\$tr,\$tr']\!]$‘$\}$

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

lemma *preR-refine-divergences*:
  assumes P *is NCSP* Q *is NCSP* $\bigwedge s.\ dv[\![P]\!]s \subseteq dv[\![Q]\!]s$
  shows $pre_R(P) \sqsubseteq pre_R(Q)$
proof (*rule CRR-refine-impl-prop*, *simp-all add*: *assms closure usubst unrest*)
  fix t s
  assume a: ‘$[\$st \mapsto_s \ll s\gg,\ \$tr \mapsto_s \langle\rangle,\ \$tr' \mapsto_s \ll t\gg] \dagger pre_R\ Q$‘
  with a **show** ‘$[\$st \mapsto_s \ll s\gg,\ \$tr \mapsto_s \langle\rangle,\ \$tr' \mapsto_s \ll t\gg] \dagger pre_R\ P$‘
  proof (*rule-tac ccontr*)
    from *assms*(3)[*of s*] **have** b: $t \in dv[\![P]\!]s \Longrightarrow t \in dv[\![Q]\!]s$
      by (*auto*)
    assume $\neg$ ‘$[\$st \mapsto_s \ll s\gg,\ \$tr \mapsto_s \langle\rangle,\ \$tr' \mapsto_s \ll t\gg] \dagger pre_R\ P$‘
    hence $\neg$ ‘$[\$st \mapsto_s \ll s\gg,\ \$tr \mapsto_s \langle\rangle,\ \$tr' \mapsto_s \ll t\gg] \dagger CRC(pre_R\ P)$‘
      by (*simp add*: *assms closure Healthy-if*)
    hence ‘$[\$st \mapsto_s \ll s\gg,\ \$tr \mapsto_s \langle\rangle,\ \$tr' \mapsto_s \ll t\gg] \dagger (\neg_r\ CRC(pre_R\ P))$‘
      by (*rel-auto*)
    hence ‘$[\$st \mapsto_s \ll s\gg,\ \$tr \mapsto_s \langle\rangle,\ \$tr' \mapsto_s \ll t\gg] \dagger (\neg_r\ pre_R\ P)$‘
      by (*simp add*: *assms closure Healthy-if*)
    with a b **show** *False*
      by (*rel-auto*)
  qed
qed

lemma *preR-eq-divergences*:
  assumes P *is NCSP* Q *is NCSP* $\bigwedge s.\ dv[\![P]\!]s = dv[\![Q]\!]s$
  shows $pre_R(P) = pre_R(Q)$
  by (*metis assms dual-order.antisym order-refl preR-refine-divergences*)

lemma *periR-refine-failures*:
  assumes P *is NCSP* Q *is NCSP* $\bigwedge s.\ fl[\![Q]\!]s \subseteq fl[\![P]\!]s$
  shows $(pre_R(P) \wedge peri_R(P)) \sqsubseteq (pre_R(Q) \wedge peri_R(Q))$

proof (*rule CRR-refine-impl-prop*, *simp-all add*: *assms closure unrest subst-unrest-3*)
fix $t\ s\ r'$
assume a: ‘$[\$ref\,\acute{}\ \mapsto_s \ll r'\gg, \$st \mapsto_s \ll s\gg, \$tr \mapsto_s \langle\rangle, \$tr\,\acute{}\ \mapsto_s \ll t\gg] \dagger (pre_R\ Q \wedge peri_R\ Q)$‘
from $assms(3)[of\ s]$ **have** b: $(t, r') \in fl[\![Q]\!]s \Longrightarrow (t, r') \in fl[\![P]\!]s$
by (*auto*)
with a **show** ‘$[\$ref\,\acute{}\ \mapsto_s \ll r'\gg, \$st \mapsto_s \ll s\gg, \$tr \mapsto_s \langle\rangle, \$tr\,\acute{}\ \mapsto_s \ll t\gg] \dagger (pre_R\ P \wedge peri_R\ P)$‘
by (*simp add*: *failures-def*)
qed

lemma *periR-eq-failures*:
assumes P *is NCSP* Q *is NCSP* $\bigwedge s.\ fl[\![P]\!]s = fl[\![Q]\!]s$
shows $(pre_R(P) \wedge peri_R(P)) = (pre_R(Q) \wedge peri_R(Q))$
by (*metis* (*full-types*) *assms dual-order.antisym order-refl periR-refine-failures*)

lemma *postR-refine-traces*:
assumes P *is NCSP* Q *is NCSP* $\bigwedge s.\ tr[\![Q]\!]s \subseteq tr[\![P]\!]s$
shows $(pre_R(P) \wedge post_R(P)) \sqsubseteq (pre_R(Q) \wedge post_R(Q))$
proof (*rule CRR-refine-impl-prop*, *simp-all add*: *assms closure unrest subst-unrest-5*)
fix $t\ s\ s'$
assume a: ‘$[\$st \mapsto_s \ll s\gg, \$st\,\acute{}\ \mapsto_s \ll s'\gg, \$tr \mapsto_s \langle\rangle, \$tr\,\acute{}\ \mapsto_s \ll t\gg] \dagger (pre_R\ Q \wedge post_R\ Q)$‘
from $assms(3)[of\ s]$ **have** b: $(t, s') \in tr[\![Q]\!]s \Longrightarrow (t, s') \in tr[\![P]\!]s$
by (*auto*)
with a **show** ‘$[\$st \mapsto_s \ll s\gg, \$st\,\acute{}\ \mapsto_s \ll s'\gg, \$tr \mapsto_s \langle\rangle, \$tr\,\acute{}\ \mapsto_s \ll t\gg] \dagger (pre_R\ P \wedge post_R\ P)$‘
by (*simp add*: *traces-def*)
qed

lemma *postR-eq-traces*:
assumes P *is NCSP* Q *is NCSP* $\bigwedge s.\ tr[\![P]\!]s = tr[\![Q]\!]s$
shows $(pre_R(P) \wedge post_R(P)) = (pre_R(Q) \wedge post_R(Q))$
by (*metis assms dual-order.antisym order-refl postR-refine-traces*)

lemma *circus-fd-refine-intro*:
assumes P *is NCSP* Q *is NCSP* $\bigwedge s.\ dv[\![Q]\!]s \subseteq dv[\![P]\!]s$ $\bigwedge s.\ fl[\![Q]\!]s \subseteq fl[\![P]\!]s$ $\bigwedge s.\ tr[\![Q]\!]s \subseteq tr[\![P]\!]s$
shows $P \sqsubseteq Q$
proof (*rule SRD-refine-intro'*, *simp-all add*: *closure assms*)
show a: ‘$pre_R\ P \Rightarrow pre_R\ Q$‘
using *assms(1) assms(2) assms(3) preR-refine-divergences refBy-order* **by** *blast*
show $peri_R\ P \sqsubseteq (pre_R\ P \wedge peri_R\ Q)$
proof −
have $peri_R\ P \sqsubseteq (pre_R\ Q \wedge peri_R\ Q)$
by (*metis* (*no-types*) *assms(1) assms(2) assms(4) periR-refine-failures utp-pred-laws.le-inf-iff*)
then show *?thesis*
by (*metis a refBy-order utp-pred-laws.inf.order-iff utp-pred-laws.inf-assoc*)
qed
show $post_R\ P \sqsubseteq (pre_R\ P \wedge post_R\ Q)$
proof −
have $post_R\ P \sqsubseteq (pre_R\ Q \wedge post_R\ Q)$
by (*meson assms(1) assms(2) assms(5) postR-refine-traces utp-pred-laws.le-inf-iff*)
then show *?thesis*
by (*metis a refBy-order utp-pred-laws.inf.absorb-iff1 utp-pred-laws.inf-assoc*)
qed
qed

13.2 Circus Operators

lemma *traces-Skip*:

$tr[\![Skip]\!]s = \{([], s)\}$
by (*simp add*: *traces-def rdes alpha closure*, *rel-simp*)

lemma *failures-Skip*:
$fl[\![Skip]\!]s = \{\}$
by (*simp add*: *failures-def*, *rdes-calc*)

lemma *divergences-Skip*:
$dv[\![Skip]\!]s = \{\}$
by (*simp add*: *divergences-def*, *rdes-calc*)

lemma *traces-Stop*:
$tr[\![Stop]\!]s = \{\}$
by (*simp add*: *traces-def*, *rdes-calc*)

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

lemma *divergences-Stop*:
$dv[\![Stop]\!]s = \{\}$
by (*simp add*: *divergences-def*, *rdes-calc*)

lemma *traces-AssignsCSP*:
$tr[\![\langle\sigma\rangle_C]\!]s = \{([], \sigma(s))\}$
by (*simp add*: *traces-def rdes closure usubst alpha*, *rel-auto*)

lemma *failures-AssignsCSP*:
$fl[\![\langle\sigma\rangle_C]\!]s = \{\}$
by (*simp add*: *failures-def*, *rdes-calc*)

lemma *divergences-AssignsCSP*:
$dv[\![\langle\sigma\rangle_C]\!]s = \{\}$
by (*simp add*: *divergences-def*, *rdes-calc*)

lemma *failures-Miracle*: $fl[\![Miracle]\!]s = \{\}$
by (*simp add*: *failures-def rdes closure usubst*)

lemma *divergences-Miracle*: $dv[\![Miracle]\!]s = \{\}$
by (*simp add*: *divergences-def rdes closure usubst*)

lemma *failures-Chaos*: $fl[\![Chaos]\!]s = \{\}$
by (*simp add*: *failures-def rdes*, *rel-auto*)

lemma *divergences-Chaos*: $dv[\![Chaos]\!]s = UNIV$
by (*simp add*: *divergences-def rdes*, *rel-auto*)

lemma *traces-Chaos*: $tr[\![Chaos]\!]s = \{\}$
by (*simp add*: *traces-def rdes closure usubst*)

lemma *divergences-cond*:
assumes *P is NCSP Q is NCSP*
shows $dv[\![P \triangleleft b \triangleright_R Q]\!]s = (if\ ([\![b]\!]_e s)\ then\ dv[\![P]\!]s\ else\ dv[\![Q]\!]s)$
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 $tr[\![P \triangleleft b \triangleright_R Q]\!]s = (if\ ([\![b]\!]_e s)\ then\ tr[\![P]\!]s\ else\ tr[\![Q]\!]s)$
  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 $fl[\![P \triangleleft b \triangleright_R Q]\!]s = (if\ ([\![b]\!]_e s)\ then\ fl[\![P]\!]s\ else\ fl[\![Q]\!]s)$
  by (*rdes-simp cls*: *assms*, *simp add*: *divergences-def failures-def rdes closure rpred assms*, *rel-auto*)

lemma *divergences-guard*:
  assumes P *is NCSP*
  shows $dv[\![g\ \&_u\ P]\!]s = (if\ ([\![g]\!]_e s)\ then\ dv[\![g\ \&_u\ P]\!]s\ else\ \{\})$
  by (*rdes-simp cls*: *assms*, *simp add*: *divergences-def traces-def rdes closure rpred assms*, *rel-auto*)

lemma *traces-do*: $tr[\![do_C(e)]\!]s = \{([[\![e]\!]_e s],\ s)\}$
  by (*rdes-simp*, *simp add*: *traces-def rdes closure rpred*, *rel-auto*)

lemma *failures-do*: $fl[\![do_C(e)]\!]s = \{([],\ E) \mid E.\ [\![e]\!]_e s \notin E\}$
  by (*rdes-simp*, *simp add*: *failures-def rdes closure rpred usubst*, *rel-auto*)

lemma *divergences-do*: $dv[\![do_C(e)]\!]s = \{\}$
  by (*rel-auto*)

lemma *nil-least* [*simp*]:
  $\langle\rangle \leq_u x = true$ **by** *rel-auto*

lemma *minus-nil* [*simp*]:
  $xs - \langle\rangle = xs$ **by** *rel-auto*

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

lemma *wp-rea-circus-lemma-2*:
  assumes P *is CRR*
  shows $in\alpha \sharp P[\![\ll s_0 \gg, \ll t_0 \gg / \$st, \$tr]\!]$
proof –
  have $in\alpha \sharp (CRR\ P)[\![\ll s_0 \gg, \ll t_0 \gg / \$st, \$tr]\!]$
    by (*rel-auto*)
  thus *?thesis*
    by (*simp add*: *Healthy-if assms ex-unrest*)
qed

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

lemma *wp-rea-circus-form*:

assumes P *is CRR* $\$ref' \sharp P\ Q$ *is CRC*
shows $(P\ wp_r\ Q) = (\forall\ (s_0,t_0) \cdot \ll t_0 \gg \leq_u \$tr' \wedge P[\![\ll s_0 \gg, \ll t_0 \gg / \$st', \$tr']\!] \Rightarrow_r Q[\![\ll s_0 \gg, \ll t_0 \gg / \$st, \$tr]\!])$
proof $-$
have $(P\ wp_r\ Q) = (\neg_r (\exists\ t_0 \cdot P[\![\ll t_0 \gg / \$tr']\!] ;; (\neg_r\ Q)[\![\ll t_0 \gg / \$tr]\!] \wedge \ll t_0 \gg \leq_u \$tr'))$
by (*simp-all add*: *wp-rea-def R2-tr-middle closure RR-implies-R2 assms*)
also have ... $= (\neg_r (\exists\ (s_0,t_0) \cdot P[\![\ll s_0 \gg, \ll t_0 \gg / \$st', \$tr']\!] ;; (\neg_r\ Q)[\![\ll s_0 \gg, \ll t_0 \gg / \$st, \$tr]\!] \wedge \ll t_0 \gg \leq_u \$tr'))$
by (*rel-blast*)
also have ... $= (\neg_r (\exists\ (s_0,t_0) \cdot P[\![\ll s_0 \gg, \ll t_0 \gg / \$st', \$tr']\!] \wedge (\neg_r\ Q)[\![\ll s_0 \gg, \ll t_0 \gg / \$st, \$tr]\!] \wedge \ll t_0 \gg \leq_u \$tr'))$
by (*simp add*: *seqr-to-conj add*: *wp-rea-circus-lemma-1 wp-rea-circus-lemma-2 assms closure conj-assoc*)
also have ... $= (\forall\ (s_0,t_0) \cdot \neg_r\ P[\![\ll s_0 \gg, \ll t_0 \gg / \$st', \$tr']\!] \vee \neg_r (\neg_r\ Q)[\![\ll s_0 \gg, \ll t_0 \gg / \$st, \$tr]\!] \vee \neg_r \ll t_0 \gg \leq_u \$tr')$
by (*rel-auto*)
also have ... $= (\forall\ (s_0,t_0) \cdot \neg_r\ P[\![\ll s_0 \gg, \ll t_0 \gg / \$st', \$tr']\!] \vee \neg_r (\neg_r\ RR\ Q)[\![\ll s_0 \gg, \ll t_0 \gg / \$st, \$tr]\!] \vee \neg_r \ll t_0 \gg \leq_u \$tr')$
by (*simp add*: *Healthy-if assms closure*)
also have ... $= (\forall\ (s_0,t_0) \cdot \neg_r\ P[\![\ll s_0 \gg, \ll t_0 \gg / \$st', \$tr']\!] \vee (RR\ Q)[\![\ll s_0 \gg, \ll t_0 \gg / \$st, \$tr]\!] \vee \neg_r \ll t_0 \gg \leq_u \$tr')$
by (*rel-auto*)
also have ... $= (\forall\ (s_0,t_0) \cdot \ll t_0 \gg \leq_u \$tr' \wedge P[\![\ll s_0 \gg, \ll t_0 \gg / \$st', \$tr']\!] \Rightarrow_r (RR\ Q)[\![\ll s_0 \gg, \ll t_0 \gg / \$st, \$tr]\!])$
by (*rel-auto*)
also have ... $= (\forall\ (s_0,t_0) \cdot \ll t_0 \gg \leq_u \$tr' \wedge P[\![\ll s_0 \gg, \ll t_0 \gg / \$st', \$tr']\!] \Rightarrow_r Q[\![\ll s_0 \gg, \ll t_0 \gg / \$st, \$tr]\!])$
by (*simp add*: *Healthy-if assms closure*)
finally show *?thesis* **.**
qed

lemma *wp-rea-circus-form-alt*:
assumes P *is CRR* $\$ref' \sharp P\ Q$ *is CRC*
shows $(P\ wp_r\ Q) = (\forall\ (s_0,t_0) \cdot \$tr\ \hat{}_u \ll t_0 \gg \leq_u \$tr' \wedge P[\![\ll s_0 \gg, \langle\rangle, \ll t_0 \gg / \$st', \$tr, \$tr']\!]$
$\Rightarrow_r R1(Q[\![\ll s_0 \gg, \langle\rangle, \&tt - \ll t_0 \gg / \$st, \$tr, \$tr']\!]))$
proof $-$
have $(P\ wp_r\ Q) = R2(P\ wp_r\ Q)$
by (*simp add*: *CRC-implies-RR CRR-implies-RR Healthy-if RR-implies-R2 assms wp-rea-R2-closed*)
also have ... $= R2(\forall\ (s_0,tr_0) \cdot \ll tr_0 \gg \leq_u \$tr' \wedge (RR\ P)[\![\ll s_0 \gg, \ll tr_0 \gg / \$st', \$tr']\!] \Rightarrow_r (RR\ Q)[\![\ll s_0 \gg, \ll tr_0 \gg / \$st, \$tr]\!])$
by (*simp add*: *wp-rea-circus-form assms closure Healthy-if*)
also have ... $= (\exists\ tt_0 \cdot (\forall\ (s_0,tr_0) \cdot \ll tr_0 \gg \leq_u \ll tt_0 \gg \wedge (RR\ P)[\![\ll s_0 \gg, \langle\rangle, \ll tr_0 \gg / \$st', \$tr, \$tr']\!]$
$\Rightarrow_r (RR\ Q)[\![\ll s_0 \gg, \ll tr_0 \gg, \ll tt_0 \gg / \$st, \$tr, \$tr']\!])$
$\wedge \$tr' =_u \$tr\ \hat{}_u \ll tt_0 \gg)$
by (*simp add*: *R2-form*, *rel-auto*)
also have ... $= (\exists\ tt_0 \cdot (\forall\ (s_0,tr_0) \cdot \ll tr_0 \gg \leq_u \ll tt_0 \gg \wedge (RR\ P)[\![\ll s_0 \gg, \langle\rangle, \ll tr_0 \gg / \$st', \$tr, \$tr']\!]$
$\Rightarrow_r (RR\ Q)[\![\ll s_0 \gg, \langle\rangle, \ll tt_0 - tr_0 \gg / \$st, \$tr, \$tr']\!])$
$\wedge \$tr' =_u \$tr\ \hat{}_u \ll tt_0 \gg)$
by (*rel-auto*)
also have ... $= (\exists\ tt_0 \cdot (\forall\ (s_0,tr_0) \cdot \$tr\ \hat{}_u \ll tr_0 \gg \leq_u \$tr' \wedge (RR\ P)[\![\ll s_0 \gg, \langle\rangle, \ll tr_0 \gg / \$st', \$tr, \$tr']\!]$
$\Rightarrow_r (RR\ Q)[\![\ll s_0 \gg, \langle\rangle, \&tt - \ll tr_0 \gg / \$st, \$tr, \$tr']\!])$
$\wedge \$tr' =_u \$tr\ \hat{}_u \ll tt_0 \gg)$
by (*rel-auto*, (*metis list-concat-minus-list-concat*)+)
also have ... $= (\forall\ (s_0,tr_0) \cdot \$tr\ \hat{}_u \ll tr_0 \gg \leq_u \$tr' \wedge (RR\ P)[\![\ll s_0 \gg, \langle\rangle, \ll tr_0 \gg / \$st', \$tr, \$tr']\!]$
$\Rightarrow_r R1((RR\ Q)[\![\ll s_0 \gg, \langle\rangle, \&tt - \ll tr_0 \gg / \$st, \$tr, \$tr']\!]))$
by (*rel-auto*, *blast*+)
also have ... $= (\forall\ (s_0,t_0) \cdot \$tr\ \hat{}_u \ll t_0 \gg \leq_u \$tr' \wedge P[\![\ll s_0 \gg, \langle\rangle, \ll t_0 \gg / \$st', \$tr, \$tr']\!]$
$\Rightarrow_r R1(Q[\![\ll s_0 \gg, \langle\rangle, \&tt - \ll t_0 \gg / \$st, \$tr, \$tr']\!]))$
by (*simp add*: *Healthy-if assms closure*)
finally show *?thesis* **.**

qed

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

lemma *traces-seq*:
fixes P :: $('s,\ 'e)$ *action*
assumes P *is NCSP* Q *is NCSP*
shows $tr[\![P\ ;;\ Q]\!]s =$
$\{(t_1\ @\ t_2,\ s') \mid t_1\ t_2\ s_0\ s'.\ (t_1,\ s_0) \in tr[\![P]\!]s \wedge (t_2,\ s') \in tr[\![Q]\!]s_0$
$\wedge\ (t_1 @ t_2) \notin dv[\![P]\!]s$
$\wedge\ (\forall\ (t,\ s_1) \in tr[\![P]\!]s.\ t \leq t_1 @ t_2 \longrightarrow (t_1 @ t_2) - t \notin dv[\![Q]\!]s_1)\ \}$
(**is** *?lhs* = *?rhs*)
proof
show *?lhs* $\subseteq$ *?rhs*
proof (*rdes-expand cls*: *assms*, *simp add*: *traces-def divergences-def rdes closure assms rdes-def unrest rpred usubst*, *auto*)
fix t :: *'e list* **and** s' :: *'s*
let $?\sigma = [\$st \mapsto_s \ll s \gg,\ \$st' \mapsto_s \ll s' \gg,\ \$tr \mapsto_s \langle\rangle,\ \$tr' \mapsto_s \ll t \gg]$
assume
a1: ‘$?\sigma \dagger (post_R\ P\ ;;\ post_R\ Q)$‘ **and**
a2: ‘$[\$st \mapsto_s \ll s \gg,\ \$tr \mapsto_s \langle\rangle,\ \$tr' \mapsto_s \ll t \gg] \dagger pre_R\ P$‘ **and**
a3: ‘$[\$st \mapsto_s \ll s \gg,\ \$tr \mapsto_s \langle\rangle,\ \$tr' \mapsto_s \ll t \gg] \dagger (post_R\ P\ wp_r\ pre_R\ Q)$‘
from *a1* **have** ‘$?\sigma \dagger (\exists\ tr_0 \cdot ((post_R\ P)[\![\ll tr_0 \gg/\$tr']\!]\ ;;\ (post_R\ Q)[\![\ll tr_0 \gg/\$tr]\!]) \wedge \ll tr_0 \gg \leq_u \$tr')$‘
by (*simp add*: *R2-tr-middle assms closure*)
then obtain tr_0 **where** *p1*:‘$?\sigma \dagger ((post_R\ P)[\![\ll tr_0 \gg/\$tr']\!]\ ;;\ (post_R\ Q)[\![\ll tr_0 \gg/\$tr]\!])$‘ **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** ‘$?\sigma \dagger (\exists\ st_0 \cdot (post_R\ P)[\![\ll tr_0 \gg/\$tr']\!][\![\ll st_0 \gg/\$st']\!]\ ;;\ (post_R\ Q)[\![\ll tr_0 \gg/\$tr]\!][\![\ll st_0 \gg/\$st]\!])$‘
by (*simp add*: *seqr-middle*[*of st*, *THEN sym*])
then obtain s_0 **where** ‘$?\sigma \dagger ((post_R\ P)[\![\ll s_0 \gg,\ll tr_0 \gg/\$st',\$tr']\!]\ ;;\ (post_R\ Q)[\![\ll s_0 \gg,\ll tr_0 \gg/\$st,\$tr]\!])$‘
apply (*simp add*: *usubst*)
apply (*erule taut-shEx-elim*)
apply (*simp add*: *unrest-all-circus-vars-st-st'* *closure unrest assms*)
apply (*rel-auto*)
done
hence ‘$((([\$st \mapsto_s \ll s \gg,\ \$st' \mapsto_s \ll s_0 \gg,\ \$tr \mapsto_s \langle\rangle,\ \$tr' \mapsto_s \ll tr_0 \gg] \dagger post_R\ P)\ ;;$
$([\$st \mapsto_s \ll s_0 \gg,\ \$st' \mapsto_s \ll s' \gg,\ \$tr \mapsto_s \ll tr_0 \gg,\ \$tr' \mapsto_s \ll t \gg] \dagger post_R\ Q))$‘
by (*rel-auto*)
hence ‘$((([\$st \mapsto_s \ll s \gg,\ \$st' \mapsto_s \ll s_0 \gg,\ \$tr \mapsto_s \langle\rangle,\ \$tr' \mapsto_s \ll tr_0 \gg] \dagger post_R\ P) \wedge$
$([\$st \mapsto_s \ll s_0 \gg,\ \$st' \mapsto_s \ll s' \gg,\ \$tr \mapsto_s \ll tr_0 \gg,\ \$tr' \mapsto_s \ll t \gg] \dagger post_R\ Q))$‘
by (*simp add*: *seqr-to-conj unrest-any-circus-var assms closure unrest*)
hence *postP*: ‘$([\$st \mapsto_s \ll s \gg,\ \$st' \mapsto_s \ll s_0 \gg,\ \$tr \mapsto_s \langle\rangle,\ \$tr' \mapsto_s \ll tr_0 \gg] \dagger post_R\ P)$‘ **and**
postQ': ‘$([\$st \mapsto_s \ll s_0 \gg,\ \$st' \mapsto_s \ll s' \gg,\ \$tr \mapsto_s \ll tr_0 \gg,\ \$tr' \mapsto_s \ll t \gg] \dagger post_R\ Q)$‘
by (*rel-auto*)+
from *postQ'* **have** ‘$[\$st \mapsto_s \ll s_0 \gg,\ \$st' \mapsto_s \ll s' \gg] \dagger [\$tr \mapsto_s \ll tr_0 \gg,\ \$tr' \mapsto_s \ll tr_0 \gg + (\ll t \gg - \ll tr_0 \gg)] \dagger post_R\ Q$‘

using *tr0* **by** (*rel-auto*)
hence ‘$[\$st \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg] \dagger [\$tr \mapsto_s 0, \$tr' \mapsto_s \ll t \gg - \ll tr_0 \gg] \dagger post_R\ Q$‘
by (*simp add*: *R2-subst-tr closure assms*)
hence *postQ*: ‘$[\$st \mapsto_s \ll s_0 \gg, \$st' \mapsto_s \ll s' \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t - tr_0 \gg] \dagger post_R\ Q$‘
by (*rel-auto*)
have *preP*: ‘$[\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll tr_0 \gg] \dagger pre_R\ P$‘
proof −
have $(pre_R\ P)[\![0,\ll tr_0 \gg/\$tr,\$tr']\!] \sqsubseteq (pre_R\ P)[\![0,\ll t \gg/\$tr,\$tr']\!]$
by (*simp add*: *RC-prefix-refine closure assms tr0*)
hence $[\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll tr_0 \gg] \dagger pre_R\ P \sqsubseteq [\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t \gg] \dagger pre_R\ P$
by (*rel-auto*)
thus *?thesis*
by (*simp add*: *taut-refine-impl a2*)
qed

have *preQ*: ‘$[\$st \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t - tr_0 \gg] \dagger pre_R\ Q$‘
proof −
from *postP a3* **have** ‘$[\$st \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \ll tr_0 \gg, \$tr' \mapsto_s \ll t \gg] \dagger pre_R\ Q$‘
apply (*simp add*: *wp-rea-def*)
apply (*rel-auto*)
using *tr0* **apply** *blast+*
done
hence ‘$[\$st \mapsto_s \ll s_0 \gg] \dagger [\$tr \mapsto_s \ll tr_0 \gg, \$tr' \mapsto_s \ll tr_0 \gg + (\ll t \gg - \ll tr_0 \gg)] \dagger pre_R\ Q$‘
by (*rel-auto*)

hence ‘$[\$st \mapsto_s \ll s_0 \gg] \dagger [\$tr \mapsto_s 0, \$tr' \mapsto_s \ll t \gg - \ll tr_0 \gg] \dagger pre_R\ Q$‘
by (*simp add*: *R2-subst-tr closure assms*)
thus *?thesis*
by (*rel-auto*)
qed

from *a2* **have** *ndiv*: ¬ ‘$[\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t \gg] \dagger (\neg_r\ pre_R\ P)$‘
by (*rel-auto*)

have *t-minus-tr0*: $tr_0 @ (t - tr_0) = t$
using *append-minus tr0* **by** *blast*

from *a3*
have *wpr*: $\bigwedge t_0\ s_1.$
‘$[\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_0 \gg] \dagger pre_R\ P$‘ $\Longrightarrow$
‘$[\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_0 \gg] \dagger post_R\ P$‘ $\Longrightarrow$
$t_0 \le t \Longrightarrow$ ‘$[\$st \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t - t_0 \gg] \dagger (\neg_r\ pre_R\ Q)$‘ $\Longrightarrow$ *False*
proof −
fix $t_0\ s_1$
assume *b*:
‘$[\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_0 \gg] \dagger pre_R\ P$‘
‘$[\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_0 \gg] \dagger post_R\ P$‘
$t_0 \le t$
‘$[\$st \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t - t_0 \gg] \dagger (\neg_r\ pre_R\ Q)$‘

from *a3* **have** *c*: ‘$\forall\ (s_0,\ t_0) \cdot \ll t_0 \gg \le_u \ll t \gg$
$\wedge [\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_0 \gg] \dagger post_R\ P$
$\Rightarrow [\$st \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t \gg - \ll t_0 \gg] \dagger pre_R\ Q$‘
by (*simp add*: *wp-rea-circus-form-alt*[*of* $post_R\ P\ pre_R\ Q$] *closure assms unrest usubst*)

(*rel-simp*)

from *c b*(*2–4*) **show** *False*
by (*rel-auto*)
qed

show $\exists\, t_1\ t_2.$
$t = t_1 @ t_2 \wedge$
$(\exists\, s_0.$ ‘$[\$st \mapsto_s \ll s\gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_1\gg] \dagger pre_R\ P \wedge$
$[\$st \mapsto_s \ll s\gg, \$st' \mapsto_s \ll s_0\gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_1\gg] \dagger post_R\ P$‘ $\wedge$
‘$[\$st \mapsto_s \ll s_0\gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_2\gg] \dagger pre_R\ Q \wedge$
$[\$st \mapsto_s \ll s_0\gg, \$st' \mapsto_s \ll s'\gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_2\gg] \dagger post_R\ Q$‘ $\wedge$
$\neg$ ‘$[\$st \mapsto_s \ll s\gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_1 @ t_2\gg] \dagger (\neg_r\ pre_R\ P)$‘ $\wedge$
$(\forall\, t_0\ s_1.$ ‘$[\$st \mapsto_s \ll s\gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_0\gg] \dagger pre_R\ P \wedge$
$[\$st \mapsto_s \ll s\gg, \$st' \mapsto_s \ll s_1\gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_0\gg] \dagger post_R\ P$‘ $\longrightarrow$
$t_0 \le t_1 @ t_2 \longrightarrow \neg$ ‘$[\$st \mapsto_s \ll s_1\gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll(t_1 @ t_2) - t_0\gg] \dagger (\neg_r\ pre_R\ Q)$‘))
apply (*rule-tac* $x=tr_0$ **in** *exI*)
apply (*rule-tac* $x=(t - tr_0)$ **in** *exI*)
apply (*auto*)
using *tr0* **apply** *auto*[*1*]
apply (*rule-tac* $x=s_0$ **in** *exI*)
apply (*auto intro*:*wpr simp add*: *taut-conj preP preQ postP postQ ndiv wpr t-minus-tr0*)
done
qed

show *?rhs* $\subseteq$ *?lhs*
proof (*rdes-expand cls*: *assms*, *simp add*: *traces-def divergences-def rdes closure assms rdes-def unrest rpred usubst*, *auto*)
fix $t_1\ t_2$ :: *'e list* **and** $s_0\ s'$ :: *'s*
assume
a1: $\neg$ ‘$[\$st \mapsto_s \ll s\gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_1 @ t_2\gg] \dagger (\neg_r\ pre_R\ P)$‘ **and**
a2: ‘$[\$st \mapsto_s \ll s\gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_1\gg] \dagger pre_R\ P$‘ **and**
a3: ‘$[\$st \mapsto_s \ll s\gg, \$st' \mapsto_s \ll s_0\gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_1\gg] \dagger post_R\ P$‘ **and**
a4: ‘$[\$st \mapsto_s \ll s_0\gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_2\gg] \dagger pre_R\ Q$‘ **and**
a5: ‘$[\$st \mapsto_s \ll s_0\gg, \$st' \mapsto_s \ll s'\gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_2\gg] \dagger post_R\ Q$‘ **and**
a6: $\forall\, t\ s_1.$ ‘$[\$st \mapsto_s \ll s\gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t\gg] \dagger pre_R\ P \wedge$
$[\$st \mapsto_s \ll s\gg, \$st' \mapsto_s \ll s_1\gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t\gg] \dagger post_R\ P$‘ $\longrightarrow$
$t \le t_1 @ t_2 \longrightarrow \neg$ ‘$[\$st \mapsto_s \ll s_1\gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll(t_1 @ t_2) - t\gg] \dagger (\neg_r\ pre_R\ Q)$‘

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

have ‘$[\$st \mapsto_s \ll s_0\gg, \$st' \mapsto_s \ll s'\gg, \$tr \mapsto_s \ll t_1\gg, \$tr' \mapsto_s \ll t_1\gg+\ll t_2\gg] \dagger post_R\ Q$‘
proof –
have $[\$st \mapsto_s \ll s_0\gg, \$st' \mapsto_s \ll s'\gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_2\gg] \dagger post_R\ Q =$
$[\$st \mapsto_s \ll s_0\gg, \$st' \mapsto_s \ll s'\gg] \dagger [\$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_2\gg] \dagger post_R\ Q$
by *rel-auto*
also have ... $= [\$st \mapsto_s \ll s_0\gg, \$st' \mapsto_s \ll s'\gg] \dagger [\$tr \mapsto_s \ll t_1\gg, \$tr' \mapsto_s \ll t_1\gg+\ll t_2\gg] \dagger post_R\ Q$
by (*simp add*: *R2-subst-tr assms closure*, *rel-auto*)
finally show *?thesis* **using** *a5*
by (*rel-auto*)
qed
with *a3*
have *postPQ*: ‘$[\$st \mapsto_s \ll s\gg, \$st' \mapsto_s \ll s'\gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_1 @ t_2\gg] \dagger (post_R\ P\ ;;\ post_R$

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

have ‘$[\$st \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \ll t_1 \gg, \$tr' \mapsto_s \ll t_1 \gg + \ll t_2 \gg] \dagger pre_R\ Q$‘
proof −
have $[\$st \mapsto_s \ll s_0 \gg, \$tr \mapsto_s \ll t_1 \gg, \$tr' \mapsto_s \ll t_1 \gg + \ll t_2 \gg] \dagger pre_R\ Q =$
$[\$st \mapsto_s \ll s_0 \gg] \dagger [\$tr \mapsto_s \ll t_1 \gg, \$tr' \mapsto_s \ll t_1 \gg + \ll t_2 \gg] \dagger pre_R\ Q$
by *rel-auto*
also have ... $= [\$st \mapsto_s \ll s_0 \gg] \dagger [\$tr \mapsto_s 0, \$tr' \mapsto_s \ll t_2 \gg] \dagger pre_R\ Q$
by (*simp add*: *R2-subst-tr assms closure*)
finally show *?thesis* **using** *a4*
by (*rel-auto*)
qed

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

have *wpR*: ‘$[\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_1\ @\ t_2 \gg] \dagger (post_R\ P\ wp_r\ pre_R\ Q)$‘
proof −
have $\bigwedge s_1\ t_0.\ [\![\ t_0 \le t_1\ @\ t_2;$ ‘$[\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_0 \gg] \dagger post_R\ P$‘ $]\!]$
$\Longrightarrow$ ‘$[\$st \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll (t_1\ @\ t_2) - t_0 \gg] \dagger pre_R\ Q$‘
proof −
fix $s_1\ t_0$
assume $c{:}t_0 \le t_1\ @\ t_2$ ‘$[\$st \mapsto_s \ll s \gg, \$st' \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_0 \gg] \dagger post_R\ P$‘

have *preP'*: ‘$[\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_0 \gg] \dagger pre_R\ P$‘
proof −
have $(pre_R\ P)[\![0, \ll t_0 \gg / \$tr, \$tr']\!] \sqsubseteq (pre_R\ P)[\![0, \ll t_1\ @\ t_2 \gg / \$tr, \$tr']\!]$
by (*simp add*: *RC-prefix-refine closure assms c*)
hence $[\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_0 \gg] \dagger pre_R\ P \sqsubseteq [\$st \mapsto_s \ll s \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_1\ @\ t_2 \gg] \dagger pre_R\ P$
by (*rel-auto*)
thus *?thesis*
by (*simp add*: *taut-refine-impl preP*)
qed

with *c a3 preP a6'*[*of* $t_0\ s_1$] **show** ‘$[\$st \mapsto_s \ll s_1 \gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll (t_1\ @\ t_2) - t_0 \gg] \dagger pre_R\ Q$‘
by (*simp*)
qed

thus *?thesis*
apply (*simp-all add*: *wp-rea-circus-form-alt assms closure unrest usubst rea-impl-alt-def*)
apply (*simp add*: *R1-def usubst tcontr-alt-def*)
apply (*auto intro*!: *taut-shAll-intro-2*)
apply (*rule taut-impl-intro*)
apply (*simp add*: *unrest-all-circus-vars-st-st' unrest closure assms*)

apply (*rel-simp*)
done
qed
show ‘($[\$st \mapsto_s \ll s\gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_1 @ t_2\gg] \dagger pre_R\ P \wedge$
$[\$st \mapsto_s \ll s\gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_1 @ t_2\gg] \dagger (post_R\ P\ wp_r\ pre_R\ Q)) \wedge$
$[\$st \mapsto_s \ll s\gg, \$st' \mapsto_s \ll s'\gg, \$tr \mapsto_s \langle\rangle, \$tr' \mapsto_s \ll t_1 @ t_2\gg] \dagger (post_R\ P\ ;;\ post_R\ Q)$‘
by (*auto simp add*: *taut-conj preP postPQ wpR*)
qed
qed

lemma *Cons-minus* [*simp*]: $(a \# t) - [a] = t$
by (*metis append-Cons append-Nil append-minus*)

lemma *traces-prefix*:
assumes *P is NCSP*
shows $tr[\![a \to P]\!]s = \{(a \# t, s') \mid t\ s'.\ (t, s') \in tr[\![P]\!]s\}$
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

13.3 Deadlock Freedom

definition $DF :: {'e}\ set \Rightarrow ({'s}, {'e})\ action$ **where**
$DF(A) = (\mu_C\ X \cdot (\bigsqcap\ a{\in}A \cdot a \to Skip)\ ;;\ X)$

lemma *DF-CSP* [*closure*]: $A \neq \{\} \Longrightarrow DF(A)$ *is CSP*
by (*simp add*: *DF-def closure unrest*)

end

14 Meta theory for Circus

theory *utp-circus*
imports
utp-circus-core
utp-circus-rel
utp-circus-healths
utp-circus-contracts
utp-circus-extchoice
utp-circus-actions
utp-circus-prefix
utp-circus-recursion
utp-circus-traces
utp-circus-parallel
utp-circus-fdsem
begin end

References

[1] S. Foster, F. Zeyda, and J. Woodcock. Unifying heterogeneous state-spaces with lenses. In *ICTAC*, LNCS 9965. Springer, 2016.

[2] M. V. M. Oliveira. *Formal Derivation of State-Rich Reactive Programs using Circus*. PhD thesis, Department of Computer Science - University of York, UK, 2006. YCST-2006-02.