Stateful-Failure Reactive Designs in Isabelle/UTP

Simon Foster James Baxter Ana Cavalcanti Jim Woodcock

October 28, 2020

Abstract

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

Contents

1 Introduction

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

2 Stateful-Failure Core Types

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

2.1 SFRD Alphabet

alphabet $('\sigma,\ '\varphi)$ *sfrd-vars* = $('\varphi$ *list*, $'\sigma)$ *rsp-vars* +
  ref :: $'\varphi$ *set*

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

type-synonym $('\sigma,'\varphi)$ *sfrd* = $('\sigma,\ '\varphi)$ *sfrd-vars*
type-synonym $('\sigma,'\varphi)$ *action* = $('\sigma,\ '\varphi)$ *sfrd hrel*
type-synonym $'\varphi$ *csp* = $(unit,'\varphi)$ *sfrd*
type-synonym $'\varphi$ *process* = $'\varphi$ *csp hrel*

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

translations
  (*type*) $('\sigma,'\varphi)$ *sfrd* <= (*type*) $('\sigma,\ '\varphi)$ *sfrd-vars*
  (*type*) $('\sigma,'\varphi)$ *action* <= (*type*) $('\sigma,\ '\varphi)$ *sfrd hrel*
  (*type*) $'\varphi$ *process* <= (*type*) $(unit,'\varphi)$ *action*

notation *sfrd-vars.more*$_L$ (Σ_C)

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

2.2 Basic laws

term $U(\$tr' = \$tr\ @\ [\lceil a\rceil_{S<}])$

lemma *R2c-tr-ext*: $R2c\ (U(\$tr' = \$tr\ @\ [\lceil a\rceil_{S<}])) = U(\$tr' = \$tr\ @\ [\lceil a\rceil_{S<}])$
  by (*rel-auto*)

lemma *circus-alpha-bij-lens*:
  $bij\text{-}lens\ (\{\$ok,\$ok',\$wait,\$wait',\$tr,\$tr',\$st,\$st',\$ref,\$ref'\}_\alpha :: \text{-} \Longrightarrow ('s,'e)\ sfrd \times ('s,'e)\ sfrd)$
  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*]: $\$ref\acute{} \sharp \lceil b \rceil_{S<}$
  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*]: $\$ref\acute{} \sharp P \Longrightarrow \$ref\acute{} \sharp R1(P)$
  by (*simp add*: *R1-def unrest*)

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

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

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

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 Stateful-Failure Reactive Relations

theory *utp-sfrd-rel*
imports *utp-sfrd-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 *Continuous-CRR* [*closure*]: *Continuous CRR*
by (*rel-blast*)

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*)

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

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

lemma *CRR-seqr-form*:
CRR(*P*) ;; *CRR*(*Q*) =
$(\exists\ tt_1 \cdot \exists\ tt_2 \cdot ((\exists\ \{\$ok, \$ok', \$wait, \$wait', \$ref\} \cdot P)[\![\ll[]\gg/\$tr]\!][\![\ll tt_1\gg/\$tr']\!]$;;
$(\exists\ \{\$ok, \$ok', \$wait, \$wait', \$ref\} \cdot Q)[\![\ll[]\gg/\$tr]\!][\![\ll tt_2\gg/\$tr']\!] \wedge \$tr' =_u \$tr\ \hat{}_u$
$\ll tt_1\gg\ \hat{}_u \ll tt_2\gg))$
by (*simp add*: *CRR-form*, *rel-auto*; *fastforce*)

CSP Reactive Finalisers

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

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

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

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

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

lemma *CRF-implies-CRR* [*closure*]:
assumes *P is CRF* **shows** *P is CRR*
proof –
have *CRR*(*CRF*(*P*)) = *CRF*(*P*)
by (*rel-auto*)
thus *?thesis*
by (*metis Healthy-def assms*)
qed

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

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

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

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

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

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

CSP Reactive Conditions

definition *CRC* :: ('s,'e) *action* ⇒ ('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 *CRC-intro'*:
assumes *P is CRR P is RC*
shows *P is CRC*
by (*metis CRC-def CRR-def Healthy-def RC-implies-RR assms*)

lemma *ref-unrest-RR* [*unrest*]: $\$ref \sharp P \Longrightarrow \$ref \sharp RR\ P$
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-intro′′:
  assumes P is CRR P is RC1
  shows P is CRC
  by (simp add: CRC-intro′ CRR-implies-RR RC-intro′ assms)

lemma CRC-implies-RR [closure]:
  assumes P is CRC
  shows P is RR
proof –
  have RR(CRC(P)) = CRC(P)
    by (rel-auto)
       (metis (no-types, lifting) Prefix-Order.prefixE Prefix-Order.prefixI append.assoc append-minus)+
  thus ?thesis
    by (metis Healthy-def assms)
qed

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

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

lemma CRF-unrest-ref′ [unrest]:
  assumes P is CRF
  shows $ref´ ♯ P
proof –
  have $ref´ ♯ CRF(P) by (simp add: CRF-def unrest)
  thus ?thesis by (simp add: assms Healthy-if)
qed

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

lemma *unrest-ref′-neg-RC* [*unrest*]:
  assumes *P is RR P is RC*
  shows $\$ref\,\acute{}\ \sharp\ P$
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 $\$ref\,\acute{}\ \sharp$ ...
    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-post-unrest-ref′* [*unrest*]: $\$ref\,\acute{}\ \sharp\ [b]_{S>}$
  by (*rel-auto*)

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

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

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

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

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

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

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 *conj-CRF-closed* [*closure*]: $[\![$ P *is CRF*; Q *is CRF* $]\!] \Longrightarrow (P \wedge Q)$ *is CRF*
  by (*rule CRF-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 *st-cond-left-impl-CRC-closed* [*closure*]:
  P *is CRC* $\Longrightarrow ([b]_{S<} \Rightarrow_r P)$ *is CRC*
  by (*rule CRC-intro*, *simp-all add*: *unrest closure*)

lemma *unrest-ref-map-st* [*unrest*]: $\$ref \sharp P \Longrightarrow \$ref \sharp P \oplus_r map\text{-}st_L[a]$
  by (*rel-auto*)

lemma *unrest-ref′-map-st* [*unrest*]: $\$ref' \sharp P \Longrightarrow \$ref' \sharp P \oplus_r map\text{-}st_L[a]$
  by (*rel-auto*)

lemma *unrest-ref-rdes-frame-ext* [*unrest*]:
  $\$ref \sharp P \Longrightarrow \$ref \sharp a{:}[P]_r{}^+$
  by (*rel-blast*)

lemma *unrest-ref′-rdes-frame-ext* [*unrest*]:
  $\$ref' \sharp P \Longrightarrow \$ref' \sharp a{:}[P]_r{}^+$
  by (*rel-blast*)

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

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

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

lemma *USUP-CRC-closed* [*closure*]: $[\![$ $A \neq \{\}$; $\bigwedge i.\ i \in A \Longrightarrow P\ i$ *is CRC* $]\!] \Longrightarrow (\bigsqcup\ i \in A \cdot P\ i)$ *is CRC*
  by (*rule CRC-intro*, *simp-all add*: *unrest closure*)

lemma *UINF-CRR-closed* [*closure*]: $[\![$ $\bigwedge i.\ i \in A \Longrightarrow P\ i$ *is CRR* $]\!] \Longrightarrow (\bigsqcap\ i \in A \cdot P\ i)$ *is CRR*

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

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

lemma *shEx-CRR-closed* [*closure*]:
assumes $\bigwedge x.\ P\ x$ *is* CRR
shows $(\exists\ x \cdot P(x))$ *is* CRR
proof –
have $CRR(\exists\ x \cdot CRR(P(x))) = (\exists\ x \cdot CRR(P(x)))$
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 *cond-st-CRR-closed* [*closure*]:
$[\![\ P$ *is* CRR; Q *is* $CRR\ ]\!] \Longrightarrow (P \triangleleft b \triangleright_R Q)$ *is* CRR
by (*simp-all add*: *CRR-intro closure unrest*)

lemma *seq-CRR-closed* [*closure*]:
assumes P *is* CRR Q *is* RR
shows $(P\ ;;\ Q)$ *is* CRR

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

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

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

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

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

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

lemma *tr-extend-seqr-lit* [*rdes*]:
fixes P :: (′*s*, ′*e*) *action*
assumes $\$ok \sharp P$ $\$wait \sharp P$ $\$ref \sharp P$
shows $U(\$tr' = \$tr\ @\ [\ll a \gg] \wedge \$st' = \$st) ;; P = P\llbracket U(\$tr\ @\ [\ll a \gg])/\$tr\rrbracket$
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 *R4-CRR-closed* [*closure*]: P *is CRR* $\Longrightarrow$ *R4*(P) *is CRR*
by (*simp add*: *R4-def conj-less-tr-RR-closed*)

lemma *R5-CRR-closed* [*closure*]:
assumes P *is CRR*
shows *R5*(P) *is CRR*
proof –
have *R5*(*CRR*(P)) *is CRR*
by (*rel-auto*; *blast*)
thus *?thesis*
by (*simp add*: *assms Healthy-if*)
qed

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

lemma *all-ref-CRC-closed* [*closure*]:
P *is CRC* $\Longrightarrow$ $(\forall\ \$ref \cdot P)$ *is CRC*
by (*simp add*: *CRC-implies-CRR CRR-unrest-ref all-unrest*)

lemma *ex-ref-CRR-closed* [*closure*]:
P *is CRR* $\Longrightarrow$ $(\exists\ \$ref \cdot P)$ *is CRR*
by (*simp add*: *CRR-unrest-ref ex-unrest*)

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

lemma *ex-ref′-CRR-closed* [*closure*]:
P *is CRR* $\Longrightarrow$ $(\exists\ \$ref' \cdot P)$ *is CRR*
using *CRR-implies-RR CRR-intro CRR-unrest-ref ex-ref′-RR-closed out-in-indep unrest-ex-diff* **by** *blast*

3.3 Introduction laws

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

lemma *CRR-refine-ext*:
assumes
P *is CRR* Q *is CRR*
$\bigwedge t\ s\ s'\ r'.\ P⟦«[]»,«t»,«s»,«s'»,«r'»/\$tr,\$tr',\$st,\$st',\$ref'⟧ \sqsubseteq Q⟦«[]»,«t»,«s»,«s'»,«r'»/\$tr,\$tr',\$st,\$st',\$ref'$
shows $P \sqsubseteq Q$
proof –
have $\bigwedge\ t\ s\ s'\ r'.\ (CRR\ P)⟦«[]»,«t»,«s»,«s'»,«r'»/\$tr,\$tr',\$st,\$st',\$ref'⟧$
$\sqsubseteq (CRR\ Q)⟦«[]»,«t»,«s»,«s'»,«r'»/\$tr,\$tr',\$st,\$st',\$ref'⟧$

using *assms* **by** (*simp add*: *Healthy-if*)
hence *CRR P* ⊑ *CRR Q*
by (*rel-auto*)
thus *?thesis*
by (*metis Healthy-if assms*(*1*) *assms*(*2*))
qed

lemma *CRR-eq-ext*:
assumes
P is CRR Q is CRR
⋀ *t s s′ r′*. *P*⟦«[]»,«*t*»,«*s*»,«*s′*»,«*r′*»/$*tr*,$*tr*´,$*st*,$*st*´,$*ref*´⟧ = *Q*⟦«[]»,«*t*»,«*s*»,«*s′*»,«*r′*»/$*tr*,$*tr*´,$*st*,$*st*´,$*ref*´
shows *P* = *Q*
proof −
have ⋀ *t s s′ r′*. (*CRR P*)⟦«[]»,«*t*»,«*s*»,«*s′*»,«*r′*»/$*tr*,$*tr*´,$*st*,$*st*´,$*ref*´⟧
= (*CRR Q*)⟦«[]»,«*t*»,«*s*»,«*s′*»,«*r′*»/$*tr*,$*tr*´,$*st*,$*st*´,$*ref*´⟧
using *assms* **by** (*simp add*: *Healthy-if*)
hence *CRR P* = *CRR Q*
by (*rel-auto*)
thus *?thesis*
by (*metis Healthy-if assms*(*1*) *assms*(*2*))
qed

lemma *CRR-refine-impl-prop*:
assumes *P is CRR Q is CRR*
⋀ *t s s′ r′*. ‘*Q*⟦«*r′*»,«*s*»,«*s′*»,«[]»,«*t*»/$*ref*´,$*st*,$*st*´,$*tr*,$*tr*´⟧‘ ⟹ ‘*P*⟦«*r′*»,«*s*»,«*s′*»,«[]»,«*t*»/$*ref*´,$*st*,$*st*´,$*tr*,
shows *P* ⊑ *Q*
by (*rule CRR-refine-ext*, *simp-all add*: *assms closure unrest usubst*)
(*rule refine-prop-intro*, *simp-all add*: *unrest unrest-all-circus-vars closure assms*)

3.4 UTP Theory

interpretation *crf-theory*: *utp-theory-kleene CRF* II_c
rewrites *P* ∈ *carrier crf-theory.thy-order* ⟷ *P is CRF*
and *le rrel-theory.thy-order* = (⊑)
and *eq rrel-theory.thy-order* = (=)
and *crf-top*: *crf-theory.utp-top* = *false*
and *crf-bottom*: *crf-theory.utp-bottom* = $true_r$
proof −
interpret *utp-theory-continuous CRF*
by (*unfold-locales*, *simp-all add*: *add*: *CRF-idem Continuous-CRF*)
show *top*:*utp-top* = *false*
by (*simp add*: *healthy-top*, *rel-auto*)
show *bot*:*utp-bottom* = $true_r$
by (*simp add*: *healthy-bottom*, *rel-auto*)
show *utp-theory-kleene CRF* II_c
by (*unfold-locales*, *simp-all add*: *closure rpred top*)
qed (*simp-all*)

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

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

lemma *crf-star-inductl*: *R is CRR* ⟹ *Q* ⊑ (*P* ;; *Q*) ⊓ *R* ⟹ *Q* ⊑ $P^{\star c}$;; *R*

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

3.5 Weakest Precondition

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

lemma *minus-nil* [*simp*]:
  $xs - \ll[]\gg = xs$ **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 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 ... = $(\forall\ (s_0,t_0) \cdot \ll t_0\gg \le_u \$tr' \wedge P[\![\ll s_0\gg,\ll t_0\gg/\$st',\$tr']\!] \Rightarrow_r (RR\ Q)[\![\ll s_0\gg,\ll t_0\gg/\$st,\$tr]\!])$
  by (*rel-auto*)
 also have ... = $(\forall\ (s_0,t_0) \cdot \ll t_0\gg \le_u \$tr' \wedge P[\![\ll s_0\gg,\ll t_0\gg/\$st',\$tr']\!] \Rightarrow_r Q[\![\ll s_0\gg,\ll t_0\gg/\$st,\$tr]\!])$
  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 $(P\ wp_r\ Q) = (\forall\ (s_0,t_0) \cdot \$tr\ \hat{}_u \ll t_0\gg \le_u \$tr' \wedge P[\![\ll s_0\gg,\ll[]\gg,\ll t_0\gg/\$st',\$tr,\$tr']\!]$
   $\Rightarrow_r R1(Q[\![\ll s_0\gg,\ll[]\gg,(\&tt-\ll t_0\gg)/\$st,\$tr,\$tr']\!]))$
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 ... = $R2(\forall\ (s_0,tr_0) \cdot \ll tr_0\gg \le_u \$tr' \wedge (RR\ P)[\![\ll s_0\gg,\ll tr_0\gg/\$st',\$tr']\!] \Rightarrow_r (RR\ Q)[\![\ll s_0\gg,\ll tr_0\gg/\$st,\$tr]\!])$
  by (*simp add*: *wp-rea-circus-form assms closure Healthy-if*)
 also have ... = $(\exists\ tt_0 \cdot (\forall\ (s_0,tr_0) \cdot \ll tr_0\gg \le_u \ll tt_0\gg \wedge (RR\ P)[\![\ll s_0\gg,\ll[]\gg,\ll tr_0\gg/\$st',\$tr,\$tr']\!]$
   $\Rightarrow_r (RR\ Q)[\![\ll s_0\gg,\ll tr_0\gg,\ll tt_0\gg/\$st,\$tr,\$tr']\!])$
   $\wedge \$tr' =_u \$tr\ \hat{}_u \ll tt_0\gg)$
  by (*simp add*: *R2-form*, *rel-auto*)
 also have ... = $(\exists\ tt_0 \cdot (\forall\ (s_0,tr_0) \cdot \ll tr_0\gg \le_u \ll tt_0\gg \wedge (RR\ P)[\![\ll s_0\gg,\ll[]\gg,\ll tr_0\gg/\$st',\$tr,\$tr']\!]$
   $\Rightarrow_r (RR\ Q)[\![\ll s_0\gg,\ll[]\gg,\ll tt_0-tr_0\gg/\$st,\$tr,\$tr']\!])$
   $\wedge \$tr' =_u \$tr\ \hat{}_u \ll tt_0\gg)$
  by (*rel-auto*)
 also have ... = $(\exists\ tt_0 \cdot (\forall\ (s_0,tr_0) \cdot \$tr\ \hat{}_u \ll tr_0\gg \le_u \$tr' \wedge (RR\ P)[\![\ll s_0\gg,\ll[]\gg,\ll tr_0\gg/\$st',\$tr,\$tr']\!]$

   $\Rightarrow_r (RR\ Q)[\![\ll s_0\gg,\ll[]\gg,(\&tt-\ll tr_0\gg)/\$st,\$tr,\$tr']\!])$
   $\wedge \$tr' =_u \$tr\ \hat{}_u \ll tt_0\gg)$
  by (*rel-auto*, (*metis list-concat-minus-list-concat*)+)
 also have ... = $(\forall\ (s_0,tr_0) \cdot \$tr\ \hat{}_u \ll tr_0\gg \le_u \$tr' \wedge (RR\ P)[\![\ll s_0\gg,\ll[]\gg,\ll tr_0\gg/\$st',\$tr,\$tr']\!]$
   $\Rightarrow_r R1((RR\ Q)[\![\ll s_0\gg,\ll[]\gg,(\&tt-\ll tr_0\gg)/\$st,\$tr,\$tr']\!]))$
  by (*rel-auto*, *blast*+)
 also have ... = $(\forall\ (s_0,t_0) \cdot \$tr\ \hat{}_u \ll t_0\gg \le_u \$tr' \wedge P[\![\ll s_0\gg,\ll[]\gg,\ll t_0\gg/\$st',\$tr,\$tr']\!]$
   $\Rightarrow_r R1(Q[\![\ll s_0\gg,\ll[]\gg,(\&tt-\ll t_0\gg)/\$st,\$tr,\$tr']\!]))$
  by (*simp add*: *Healthy-if assms closure*)
 finally show *?thesis* **.**
qed

lemma *wp-rea-circus-form-alt*:
 assumes *P is CRR* $\$ref' \sharp P$ *Q is CRC*
 shows $(P\ wp_r\ Q) = (\forall\ (s_0,t_0) \cdot \$tr\ \hat{}_u \ll t_0\gg \le_u \$tr' \wedge P[\![\ll s_0\gg,\ll[]\gg,\ll t_0\gg/\$st',\$tr,\$tr']\!]$
   $\Rightarrow_r R1(Q[\![\ll s_0\gg,\ll[]\gg,(\&tt-\ll t_0\gg)/\$st,\$tr,\$tr']\!]))$
 oops

3.6 Trace Substitution

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

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 $P[\![\ll[]\gg]\!]_t = P$
proof −
  have $(CRR\ P)[\![\ll[]\gg]\!]_t = CRR\ P$
    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*)

3.7 Initial Interaction

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

lemma *usubst-rea-init′* [*usubst*]:
  $\sigma \dagger_S \mathcal{I}(s,t) = \mathcal{I}(\sigma\dagger s,\sigma\dagger t)$
  by (*rel-auto*)

lemma *unrest-rea-init* [*unrest*]:
  $[\![\ x \bowtie (\$tr)_v;\ x \bowtie (\$tr\acute{})_v;\ x \bowtie (\$st)_v\ ]\!] \Longrightarrow x \sharp \mathcal{I}(s,t)$

by (*simp add*: *rea-init-def unrest lens-indep-sym*)

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

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

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

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

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

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

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

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

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

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

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

3.8 Enabled Events

definition *csp-enable* :: *'s upred* ⇒ (*'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 *st-unrest-csp-enable* [*unrest*]: $[\![\ \&\mathbf{v} \sharp s;\ \&\mathbf{v} \sharp t;\ \&\mathbf{v} \sharp E\ ]\!] \Longrightarrow \$st \sharp \mathcal{E}(s,\ t,\ E)$
  by (*simp add*: *csp-enable-def unrest*)

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

lemma *csp-enable-st-pred* [*rpred*]:
  $([s_1]_{S<} \wedge \mathcal{E}(s_2,t,E)) = \mathcal{E}(s_1 \wedge s_2,t,E)$
  by (*rel-auto*)

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

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

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

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

lemma *csp-enable-nothing*: $\mathcal{E}(\mathit{true},\ll[]\gg,\ \{\}_u) = (\$tr' =_u \$tr)$
  by (*rel-auto*)

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

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

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

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

by (*rel-auto*)

lemma *refine-csp-enable*: $\mathcal{E}(b_1,\ t,\ E_1) \sqsubseteq \mathcal{E}(b_2,\ t,\ E_2) \longleftrightarrow (`b_2 \Rightarrow b_1 \wedge E_1 \subseteq_u E_2`)$
by (*rel-blast*)

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

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

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

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

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

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

3.9 Completed Trace Interaction

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

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

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

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

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

lemma *csp-do-R4-closed* [*closure*]:
$\Phi(b,\sigma,\mathit{bop}\ \mathit{Cons}\ x\ xs)$ *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*]:
  $\lceil\sigma\rceil_{S\sigma} \dagger \Phi(s,\varrho,t) = \Phi(\sigma \dagger s,\varrho \circ_s \sigma,\sigma \dagger t)$
  by (*rel-auto*)

lemma *csp-do-nothing*: $\Phi(true,id_s,\ll[]\gg) = II_c$
  by (*rel-auto*)

lemma *csp-do-nothing-0*: $\Phi(true,id_s,0) = II_c$
  by (*rel-auto*)

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

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 (*rel-auto*)

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

lemma *csp-do-cond* [*rpred*]:
  $\Phi(s_1,\sigma,t_1) \triangleleft b \triangleright_R \Phi(s_2,\varrho,t_2) = \Phi(s_1 \triangleleft b \triangleright s_2,\ \sigma \triangleleft b \triangleright \varrho,\ t_1 \triangleleft b \triangleright t_2)$
  by (*rel-auto*)

lemma *rea-assm-csp-do* [*rpred*]:
  $[b]^{\top}{}_r$;; $\Phi(s,\sigma,t) = \Phi(b\wedge s,\sigma,t)$
  by (*rel-auto*)

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

lemma *wp-rea-csp-do-lemma*:
  fixes P :: $('\sigma,\ '\varphi)$ *action*
  assumes $\$ok \sharp P$ $\$wait \sharp P$ $\$ref \sharp P$
  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*)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Iterated do relations

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

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

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

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

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

update-uexpr-rep-eq-thms

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

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

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

lemma *csp-do-csp-star* [*rpred*]:
  $\Phi(s,\ \sigma,\ t)^{\star c} = (\bigsqcap\ n \cdot \Phi(\bigsqcup\ i \in \{0..{<}n\} \cdot (\sigma\ \ \hat{}_s\ i) \dagger\ s{,}\sigma\ \ \hat{}_s\ n{,}titr\ n\ \sigma\ t))$
  (**is** $?lhs = (\bigsqcap\ n \cdot\ ?G(n))$)
proof −
  have $?lhs = II_c \sqcap (\bigsqcap\ n \cdot \Phi(\bigwedge\ i{\in}\{0..n\} \cdot (\sigma\ \hat{}_s i) \dagger\ s,\ \sigma\ \hat{}_s Suc\ n,\ titr\ (Suc\ n)\ \sigma\ t))$
    (**is** $- = II_c \sqcap (\bigsqcap\ n \cdot\ ?F(n))$)
    by (*simp add*: *crf-theory.Star-alt-def closure uplus-power-def rpred*)
  also have ... $= II_c \sqcap (\bigsqcap\ n{\in}\{1..\} \cdot\ ?F(n - 1))$
    by (*simp add*: *UINF-atLeast-Suc*)
  also have ... $= II_c \sqcap (\bigsqcap\ n \in \{1..\} \cdot \Phi(\bigsqcup\ i \in \{0..{<}n\} \cdot (\sigma\ \ \hat{}_s\ i) \dagger\ s{,}\sigma\ \ \hat{}_s\ n{,}titr\ n\ \sigma\ t))$
  proof −
    have $(\bigsqcap\ n{\in}\{1..\} \cdot\ ?F(n - 1)) = (\bigsqcap\ n \in \{1..\} \cdot\ ?G(n))$
      by (*rule UINF-cong*, *simp*, *metis* (*no-types*, *lifting*) *Suc-diff-le atLeastLessThanSuc-atLeastAtMost cancel-comm-monoid-add-class.diff-zero diff-Suc-Suc*)
    thus *?thesis* **by** *simp*
  qed
  also have ... $=\ ?G(0) \sqcap (\bigsqcap\ n \in \{1..\} \cdot\ ?G(n))$
    by (*simp add*: *usubst csp-do-nothing-0*)
  also have ... $= (\bigsqcap\ n \in insert\ 0\ \{1..\} \cdot\ ?G(n))$
    by (*simp*)
  also have ... $= (\bigsqcap\ n \cdot\ ?G(n))$
  proof −
    have $insert\ (0{::}nat)\ \{1..\} = \{0..\}$ **by** *auto*
    thus *?thesis*
      by *simp*
  qed
  finally show *?thesis* **.**
qed

3.10 Assumptions

abbreviation *crf-assume* :: *'s upred* ⇒ (*'s*, *'e*) *action* ($[\text{-}]_c$) **where**
$[b]_c \equiv \Phi(b,\ id_s,\ \ll[]\gg)$

lemma *crf-assume-true* [*rpred*]: $P\ is\ CRR \Longrightarrow [true]_c\ ;;\ P = P$

by (*simp add*: *crel-skip-left-unit csp-do-nothing*)

3.11 Downward closure of refusals

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

definition *CDC* :: $('s, 'e)$ *action* $\Rightarrow$ $('s, 'e)$ *action* **where**
[*upred-defs*]: $CDC(P) = (\exists\ ref_0 \cdot P[\![\ll ref_0\gg/\$ref']\!] \land \$ref' \subseteq_u \ll ref_0\gg)$

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

lemma *CDC-Continuous* [*closure*]: *Continuous CDC*
  by (*rel-auto*)

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

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

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

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

lemma *CDC-unrest* [*unrest*]: $[\![$ *vwb-lens x*; $(\$ref')_v \bowtie x$; $x \sharp P$ $]\!] \Longrightarrow x \sharp CDC(P)$
  by (*simp add*: *CDC-def unrest usubst lens-indep-sym*)

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

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

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

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

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

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

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

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

assumes P *is CDC* Q *is CDC*
shows $(P \vee Q)$ *is CDC*
proof −
have $CDC(P \vee Q) = (CDC(P) \vee CDC(Q))$
by (*rel-auto*)
thus *?thesis*
by (*metis Healthy-def assms*(*1*) *assms*(*2*))
qed

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

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

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

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

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

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

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

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

lemma *state-srea-CDC-closed* [*closure*]:

assumes *P is CDC*
shows *state 'a* · *P is CDC*
proof –
have *state 'a* · *CDC*(*P*) *is CDC*
by (*rel-blast*)
thus *?thesis*
by (*simp add*: *Healthy-if assms*)
qed

3.12 Renaming

abbreviation *pre-image f B* ≡ {*x*. *f*(*x*) ∈ *B*}

definition *csp-rename* :: (*'s*, *'e*) *action* ⇒ (*'e* ⇒ *'f*) ⇒ (*'s*, *'f*) *action* ((-)(|-|)$_c$ [*999*, *0*] *999*) **where**
[*upred-defs*]: *P*(|*f*|)$_c$ = *R2*((\$*tr*´ $=_u$ ≪[]≫ ∧ \$*st*´ $=_u$ \$*st*) ;; *P* ;; (\$*tr*´ $=_u$ *map*$_u$ ≪*f*≫ \$*tr* ∧ \$*st*´ $=_u$ \$*st* ∧ *uop* (*pre-image f*) \$*ref*´ $\subseteq_u$ \$*ref*))

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

lemma *csp-rename-disj* [*rpred*]: (*P* ∨ *Q*)(|*f*|)$_c$ = (*P*(|*f*|)$_c$ ∨ *Q*(|*f*|)$_c$)
by (*rel-blast*)

lemma *csp-rename-UINF-ind* [*rpred*]: (⊓ *i* · *P i*)(|*f*|)$_c$ = (⊓ *i* · (*P i*)(|*f*|)$_c$)
by (*rel-blast*)

lemma *csp-rename-UINF-mem* [*rpred*]: (⊓ *i* ∈ *A* · *P i*)(|*f*|)$_c$ = (⊓ *i* ∈ *A* · (*P i*)(|*f*|)$_c$)
by (*rel-blast*)

Renaming distributes through conjunction only when both sides are downward closed

lemma *csp-rename-conj* [*rpred*]:
assumes *inj f P is CRR Q is CRR P is CDC Q is CDC*
shows (*P* ∧ *Q*)(|*f*|)$_c$ = (*P*(|*f*|)$_c$ ∧ *Q*(|*f*|)$_c$)
proof –
from *assms*(*1*) **have** ((*CDC* (*CRR P*)) ∧ (*CDC* (*CRR Q*)))(|*f*|)$_c$ = ((*CDC* (*CRR P*))(|*f*|)$_c$ ∧ (*CDC* (*CRR Q*))(|*f*|)$_c$)
apply (*rel-auto*)
apply *blast*
apply *blast*
apply (*meson order-refl order-trans*)
done
thus *?thesis*
by (*simp add*: *assms Healthy-if*)
qed

lemma *csp-rename-seq* [*rpred*]:
assumes *P is CRR Q is CRR*
shows (*P* ;; *Q*)(|*f*|)$_c$ = *P*(|*f*|)$_c$;; *Q*(|*f*|)$_c$
oops

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

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

lemma *csp-rename-do* [*rpred*]: $\Phi(s,\sigma,t)(\!|f|\!)_c = \Phi(s,\sigma,map_u \ll f \gg t)$
  by (*rel-auto*)

lemma *csp-rename-enable* [*rpred*]: $\mathcal{E}(s,t,E)(\!|f|\!)_c = \mathcal{E}(s,map_u \ll f \gg t,\ uop\ (image\ f)\ E)$
  by (*rel-auto*)

lemma *st′-unrest-csp-rename* [*unrest*]: $\$st\acute{} \sharp P \Longrightarrow \$st\acute{} \sharp P(\!|f|\!)_c$
  by (*rel-blast*)

lemma *ref′-unrest-csp-rename* [*unrest*]: $\$ref\acute{} \sharp P \Longrightarrow \$ref\acute{} \sharp P(\!|f|\!)_c$
  by (*rel-blast*)

lemma *csp-rename-CDC-closed* [*closure*]:
  P *is CDC* $\Longrightarrow P(\!|f|\!)_c$ *is CDC*
  by (*rel-blast*)

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

end

4 Stateful-Failure Healthiness Conditions

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

5 Definitions

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

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

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

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

definition *STOP* :: $'\varphi$ *process* **where**

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

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

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

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

definition *CSP3* :: ((*'σ*, *'φ*) *sfrd* × (*'σ*, *'φ*) *sfrd*) *health* **where**
[*upred-defs*]: $CSP3(P) = (Skip\ ;;\ P)$

definition *CSP4* :: ((*'σ*, *'φ*) *sfrd* × (*'σ*, *'φ*) *sfrd*) *health* **where**
[*upred-defs*]: $CSP4(P) = (P\ ;;\ Skip)$

definition *NCSP* :: ((*'σ*, *'φ*) *sfrd* × (*'σ*, *'φ*) *sfrd*) *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* [*simp*]: $SKIP = Skip$
  by (*rel-auto*)

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma *Skip-left-lemma*:
  assumes *P is CSP*
  shows $Skip\ ;;\ P = \mathbf{R}_s\ ((\forall\ \$ref \cdot pre_R\ P) \vdash (\exists\ \$ref \cdot cmt_R\ P))$
proof −
  have $Skip\ ;;\ P =$
        $\mathbf{R}_s\ ((\$tr' =_u \$tr \wedge \$st' =_u \$st)\ wp_r\ pre_R\ P \vdash$
            $(\$tr' =_u \$tr \wedge \$st' =_u \$st)\ ;;\ peri_R\ P \diamond$
            $(\$tr' =_u \$tr \wedge \$st' =_u \$st)\ ;;\ post_R\ P)$
    by (*simp add*: *SRD-composition-wp alpha rdes closure wp assms rpred C1*, *rel-auto*)
  also have ... $= \mathbf{R}_s\ ((\forall\ \$ref \cdot pre_R\ P) \vdash$
                    $(\$tr' =_u \$tr \wedge \neg\ \$wait' \wedge \$st' =_u \$st)\ ;;\ ((\exists\ \$st \cdot \lceil II \rceil_D) \triangleleft \$wait \triangleright cmt_R\ P))$
    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*:
  $[\![\ P\ is\ CSP;\ \$ref \sharp P[\![false/\$wait]\!]\ ]\!] \Longrightarrow P\ is\ CSP3$
  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*]: $\$ref \sharp Skip[\![false/\$wait]\!]$
by (*simp add*: *Skip-def RHS-def R1-def R2c-def R2s-def R3h-def design-def usubst unrest*)

lemma *Skip-unrest-ref′* [*unrest*]: $\$ref\acute{} \sharp Skip[\![false/\$wait]\!]$
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* $\longleftrightarrow$ ($\$ref \sharp P[\![false/\$wait]\!]$)
proof
assume *1*: P *is CSP3*
have $\$ref \sharp (Skip ;; P)[\![false/\$wait]\!]$
by (*simp add*: *usubst unrest*)
with *1* **show** $\$ref \sharp P[\![false/\$wait]\!]$
by (*metis CSP3-def Healthy-def*)
next
assume *1*:$\$ref \sharp P[\![false/\$wait]\!]$
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*:($\$ref \sharp P[\![false/\$wait]\!]$)
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 $(\neg_r\ pre_R(P))\ ;;\ R1(true) = (\neg_r\ pre_R(P))$ $\$st' \sharp peri_R(P)$ $\$ref' \sharp post_R(P)$
  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´* ♯ *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*) ⊢ *peri*$_R$(*X*) ⋄ *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 ⊆ ⟦*NCSP*⟧$_H$ $\Longrightarrow$ *A* ⊆ ⟦*CSP*⟧$_H$
using *NCSP-implies-CSP* **by** *blast*

lemma *NCSP-subset-implies-NSRD* [*closure*]:
A ⊆ ⟦*NCSP*⟧$_H$ $\Longrightarrow$ *A* ⊆ ⟦*NSRD*⟧$_H$
using *NCSP-implies-NSRD* **by** *blast*

lemma *CSP-Healthy-subset-member*: ⟦ *P* ∈ *A*; *A* ⊆ ⟦*CSP*⟧$_H$ ⟧ $\Longrightarrow$ *P is CSP*
by (*simp add*: *is-Healthy-subset-member*)

lemma *CSP3-Healthy-subset-member*: ⟦ *P* ∈ *A*; *A* ⊆ ⟦*CSP3*⟧$_H$ ⟧ $\Longrightarrow$ *P is CSP3*
by (*simp add*: *is-Healthy-subset-member*)

lemma *CSP4-Healthy-subset-member*: ⟦ *P* ∈ *A*; *A* ⊆ ⟦*CSP4*⟧$_H$ ⟧ $\Longrightarrow$ *P is CSP4*
by (*simp add*: *is-Healthy-subset-member*)

lemma *NCSP-Healthy-subset-member*: ⟦ *P* ∈ *A*; *A* ⊆ ⟦*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 ... $= \mathbf{R}_s\ ((\neg_r\ pre_R\ P)\ wp_r\ false \vdash ((\exists\ \$st' \cdot cmt_R\ P)[\![true/\$wait']\!] \triangleleft \$wait' \triangleright (\exists\ \$ref' \cdot$

$cmt_R\ P)\llbracket false/\$wait\acute{}\rrbracket))$
    by (*metis* (*no-types*, *hide-lams*) *subst-wait'-left-subst subst-wait'-right-subst wait'-cond-def*)
  also have ... = $\mathbf{R}_s((\neg_r\ pre_R\ P)\ wp_r\ false \vdash ((\exists\ \$st\acute{}\ \cdot\ peri_R(P)) \diamond (\exists\ \$ref\acute{}\ \cdot\ post_R(P))))$
    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 $\$ref\acute{} \sharp\ pre_R(P)$
proof –
  have $pre_R(P) = pre_R(\mathbf{R}_s((\neg_r\ pre_R\ P)\ wp_r\ false \vdash ((\exists\ \$st\acute{}\ \cdot\ peri_R(P)) \diamond (\exists\ \$ref\acute{}\ \cdot\ post_R(P)))))$
    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 $\$ref\acute{} \sharp$ ...
    by (*simp add*: *wp-rea-def unrest*)
  finally show *?thesis* **.**
qed

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

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

lemma *CSP4-ref'-unrest-post* [*unrest*]:
  assumes *P is CSP P is CSP4*
  shows $\$ref\acute{} \sharp\ post_R(P)$
proof –
  have $post_R(P) = post_R(\mathbf{R}_s((\neg_r\ pre_R\ P)\ wp_r\ false \vdash ((\exists\ \$st\acute{}\ \cdot\ peri_R(P)) \diamond (\exists\ \$ref\acute{}\ \cdot\ post_R(P)))))$
    using *CSP4-healthy-form assms*(*1*) *assms*(*2*) **by** *fastforce*
  also have ... = $R1\ (R2c\ ((\neg_r\ pre_R\ P)\ wp_r\ false \Rightarrow_r (\exists\ \$ref\acute{}\ \cdot\ post_R\ P)))$
    by (*simp add*: *rea-post-RHS-design usubst unrest wp-rea-def*)
  also have $\$ref\acute{} \sharp$ ...
    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 *rdes-frame-ext-NCSP-closed* [*closure*]:
assumes *vwb-lens a P is NCSP*
shows $a{:}[P]_R{}^{+}$ *is NCSP*
by (*metis* (*no-types*, *lifting*) *CSP3-def CSP4-def Healthy-intro NCSP-Skip NCSP-implies-NSRD NCSP-intro NSRD-is-SRD Skip-frame Skip-left-unit Skip-right-unit assms*(*1*) *assms*(*2*) *rdes-frame-ext-NSRD-closed seq-srea-frame*)

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

lemma *periR-Stop* [*rdes*]: $peri_R(Stop) = \mathcal{E}(true, \ll[]\gg, \{\}_u)$
  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* [*closure*]:
  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 *NCSP-postR-CRF* [*closure*]: P *is NCSP* $\Longrightarrow post_R$ P *is CRF*
  by (*rule CRF-intro*, *simp-all add*: *unrest closure*)

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 ⊆ ⟦CSP4⟧_H ⟹ (⨅ 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 ⊆ ⟦NCSP⟧_H; A ≠ {} ⟧ ⟹ (⨅ 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]: ⟦ ⋀ i. P(i) is NCSP; A ≠ {} ⟧ ⟹ (⨅ i∈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 ... = R_s ((∀ $ref · (¬_r pre_R(NCSP P)) wp_r false) ⊢
                    (∃ $ref · ∃ $st´ · peri_R(NCSP P)) ⋄
                    ((∃ $ref · ∃ $ref´ · post_R (NCSP P)) ∧ $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 (R_s ((pre_R X) ⊢ peri_R X ⋄ (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 (R_s ((∀ $ref · (¬_r pre_R (NCSP P)) wp_r false) ⊢
                              (∃ $ref · ∃ $st´ · peri_R (NCSP P)) ⋄
                              (∃ $ref · ∃ $ref´ · 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* $]\!]$ $\Longrightarrow$ *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

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

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

interpretation *csp-theory*: *utp-theory-kleene NCSP Skip*
rewrites $P \in$ *carrier csp-theory.thy-order* $\longleftrightarrow$ *P is NCSP*
and *carrier csp-theory.thy-order* $\rightarrow$ *carrier csp-theory.thy-order* $\equiv [\![NCSP]\!]_H \rightarrow [\![NCSP]\!]_H$
and *le csp-theory.thy-order* = $(\sqsubseteq)$
and *eq csp-theory.thy-order* = $(=)$
and *csp-top*: *csp-theory.utp-top* = *Miracle*
and *csp-bottom*: *csp-theory.utp-bottom* = *Chaos*
proof −
have *utp-theory-continuous NCSP*
by (*unfold-locales*, *simp-all add*: *Healthy-Idempotent Healthy-if NCSP-Idempotent NCSP-Continuous*)
then interpret *utp-theory-continuous NCSP*
by *simp*
show *t*: *utp-top* = *Miracle* **and** *b*:*utp-bottom* = *Chaos*
by (*simp-all add*: *healthy-top healthy-bottom NCSP-false NCSP-true*)
show *utp-theory-kleene NCSP Skip*
by (*unfold-locales*, *simp-all add*: *closure Skip-left-unit Skip-right-unit Miracle-left-zero t*)
qed (*simp-all*)

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

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

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

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

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

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

5.3 Algebraic laws

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

end

6 Stateful-Failure Reactive Contracts

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

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

syntax
  -ref-var :: *logic*
  -mk-CRD :: *logic* ⇒ *logic* ⇒ *logic* ⇒ *logic* ([-/ ⊢ -/ | -]$_C$)

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

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

Refinement introduction law for contracts

lemma *CRD-contract-refine*:
  assumes
    Q *is CSP* ‘$\lceil P_1 \rceil_{S<} \Rightarrow pre_R\ Q$‘
    ‘$\lceil P_1 \rceil_{S<} \wedge peri_R\ Q \Rightarrow \lceil P_2\ t\ r \rceil_{S<}⟦t \rightarrow \&tt⟧⟦r \rightarrow \$ref'⟧$‘
    ‘$\lceil P_1 \rceil_{S<} \wedge post_R\ Q \Rightarrow \lceil P_3\ x \rceil_{S}⟦x \rightarrow \&tt⟧$‘
  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*+)

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

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

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

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

end

7 External Choice

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

7.1 Definitions and syntax

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

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

syntax
  $\text{-}ExtChoice :: pttrn \Rightarrow 'a\ set \Rightarrow 'b \Rightarrow 'b\ \ ((3\Box\ \text{-}{\in}\text{-}\ {\cdot}/\ \text{-})\ [0,\ 0,\ 10]\ 10)$
  $\text{-}ExtChoice\text{-}simp :: pttrn \Rightarrow 'b \Rightarrow 'b\ \ ((3\Box\ \text{-}\ {\cdot}/\ \text{-})\ [0,\ 10]\ 10)$

translations
  $\Box P{\in}A \cdot B\ \ \rightleftharpoons CONST\ EXTCHOICE\ A\ (\lambda P.\ B)$
  $\Box P \cdot B\ \ \ \ \rightleftharpoons CONST\ EXTCHOICE\ (CONST\ UNIV)\ (\lambda P.\ B)$

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

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*)

7.2 Basic laws

7.3 Algebraic laws

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

lemma *ExtChoice-single*:
  $P\ is\ CSP \Longrightarrow ExtChoice\ \{P\} = P$
  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 $(\Box i{\in}A \cdot \mathbf{R}_s(P(i) \vdash Q(i))) =$
$\mathbf{R}_s\ ((\bigsqcup i{\in}A \cdot \mathit{pre}_R\ (\mathbf{R}_s\ (P\ i \vdash Q\ i))) \vdash$
$((\bigsqcup i{\in}A \cdot \mathit{cmt}_R\ (\mathbf{R}_s\ (P\ i \vdash Q\ i)))$
$\lhd \$tr' =_u \$tr \wedge \$wait' \rhd$
$(\bigsqcap i{\in}A \cdot \mathit{cmt}_R\ (\mathbf{R}_s\ (P\ i \vdash Q\ i)))))$

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

also have ... $=$
$\mathbf{R}_s\ ((\bigsqcup i{\in}A \cdot \mathit{R1}\ (\mathit{R2c}\ (\mathit{pre}_s \dagger P(i)))) \vdash$
$((\bigsqcup i{\in}A \cdot \mathit{R1}(\mathit{R2c}(\mathit{cmt}_s \dagger (P(i) \Rightarrow Q(i)))))$
$\lhd \$tr' =_u \$tr \wedge \$wait' \rhd$
$(\bigsqcap i{\in}A \cdot \mathit{R1}(\mathit{R2c}(\mathit{cmt}_s \dagger (P(i) \Rightarrow Q(i)))))))$

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

also have ... $=$
$\mathbf{R}_s\ ((\bigsqcup i{\in}A \cdot \mathit{R1}\ (\mathit{R2c}\ (\mathit{pre}_s \dagger P(i)))) \vdash$
$\mathit{R1}(\mathit{R2c}$
$((\bigsqcup i{\in}A \cdot \mathit{R1}(\mathit{R2c}(\mathit{cmt}_s \dagger (P(i) \Rightarrow Q(i)))))$
$\lhd \$tr' =_u \$tr \wedge \$wait' \rhd$
$(\bigsqcap i{\in}A \cdot \mathit{R1}(\mathit{R2c}(\mathit{cmt}_s \dagger (P(i) \Rightarrow Q(i)))))))))$

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

also have ... $=$
$\mathbf{R}_s\ ((\bigsqcup i{\in}A \cdot \mathit{R1}\ (\mathit{R2c}\ (\mathit{pre}_s \dagger P(i)))) \vdash$
$\mathit{R1}(\mathit{R2c}$
$((\bigsqcup i{\in}A \cdot (\mathit{cmt}_s \dagger (P(i) \Rightarrow Q(i))))$
$\lhd \$tr' =_u \$tr \wedge \$wait' \rhd$
$(\bigsqcap i{\in}A \cdot (\mathit{cmt}_s \dagger (P(i) \Rightarrow Q(i))))))))$

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 ... $=$
$\mathbf{R}_s\ ((\bigsqcup i{\in}A \cdot \mathit{R1}\ (\mathit{R2c}\ (\mathit{pre}_s \dagger P(i)))) \vdash$
$\mathit{cmt}_s \dagger$
$((\bigsqcup i{\in}A \cdot (\mathit{cmt}_s \dagger (P(i) \Rightarrow Q(i))))$
$\lhd \$tr' =_u \$tr \wedge \$wait' \rhd$
$(\bigsqcap i{\in}A \cdot (\mathit{cmt}_s \dagger (P(i) \Rightarrow Q(i))))))$

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

also have ... $=$
$\mathbf{R}_s\ ((\bigsqcup i{\in}A \cdot \mathit{R1}\ (\mathit{R2c}\ (\mathit{pre}_s \dagger P(i)))) \vdash$
$\mathit{cmt}_s \dagger$
$((\bigsqcup i{\in}A \cdot (P(i) \Rightarrow Q(i)))$
$\lhd \$tr' =_u \$tr \wedge \$wait' \rhd$
$(\bigsqcap i{\in}A \cdot (P(i) \Rightarrow Q(i)))))$

by (*simp add*: *usubst*)

also have ... $=$
$\mathbf{R}_s\ ((\bigsqcup i{\in}A \cdot \mathit{R1}\ (\mathit{R2c}\ (\mathit{pre}_s \dagger P(i)))) \vdash$
$((\bigsqcup i{\in}A \cdot (P(i) \Rightarrow Q(i))) \lhd \$tr' =_u \$tr \wedge \$wait' \rhd (\bigsqcap i{\in}A \cdot (P(i) \Rightarrow Q(i)))))$

by (*simp add*: *rdes-export-cmt*)

also have ... $=$
$\mathbf{R}_s\ ((\mathit{R1}(\mathit{R2c}(\bigsqcup i{\in}A \cdot (\mathit{pre}_s \dagger P(i))))) \vdash$
$((\bigsqcup i{\in}A \cdot (P(i) \Rightarrow Q(i))) \lhd \$tr' =_u \$tr \wedge \$wait' \rhd (\bigsqcap i{\in}A \cdot (P(i) \Rightarrow Q(i)))))$

by (*simp add*: *not-UINF R1-UINF R2c-UINF assms*)

also have ... $=$
$\mathbf{R}_s\ ((\mathit{R2c}(\bigsqcup i{\in}A \cdot (\mathit{pre}_s \dagger P(i)))) \vdash$
$((\bigsqcup i{\in}A \cdot (P(i) \Rightarrow Q(i))) \lhd \$tr' =_u \$tr \wedge \$wait' \rhd (\bigsqcap i{\in}A \cdot (P(i) \Rightarrow Q(i)))))$

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 $(\Box\ i{\in}A \cdot \mathbf{R}_s(P_1(i) \vdash P_2(i) \diamond P_3(i))) =$
$\mathbf{R}_s$ $((\bigsqcup\ i{\in}A \cdot P_1(i)) \vdash (((\bigsqcup\ i{\in}A \cdot R5(P_2(i))) \vee (\bigsqcap\ i{\in}A \cdot R4(P_2(i)))) \diamond (\bigsqcap\ i{\in}A \cdot P_3(i))))$
by (*simp add*: *ExtChoice-tri-rdes assms*, *rel-auto*, *simp-all add*: *less-le assms*)

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

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

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

lemma *extChoice-rdes-def*:
  assumes P_1 *is RR* Q_1 *is RR*
  shows $\mathbf{R}_s(P_1 \vdash P_2 \diamond P_3) \Box \mathbf{R}_s(Q_1 \vdash Q_2 \diamond Q_3) =$
        $\mathbf{R}_s\ ((P_1 \wedge Q_1) \vdash (((P_2 \wedge Q_2) \triangleleft \$tr' =_u \$tr \triangleright (P_2 \vee Q_2)) \diamond (P_3 \vee Q_3)))$

by (*subst extChoice-tri-rdes*, *simp-all add*: *assms unrest*)

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

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

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

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

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

lemma *periR-ExtChoice* [*rdes*]:
assumes $A \neq \{\}$ $\bigwedge i.\ i{\in}A \Longrightarrow F\ i$ *is NCSP*
shows $peri_R(EXTCHOICE\ A\ F) = (((\bigsqcup\ P{\in}A \cdot pre_R\ (F\ P)) \Rightarrow_r (\bigsqcup\ P{\in}A \cdot peri_R\ (F\ P))) \triangleleft \boldsymbol{U}(\$tr' = \$tr) \triangleright (\bigsqcap\ P{\in}A \cdot peri_R\ (F\ P)))$
(**is** *?lhs* = *?rhs*)
proof −
have $?lhs = ((\bigsqcup\ P{\in}A \cdot pre_R\ (F\ P)) \Rightarrow_r (\bigsqcup\ P{\in}A \cdot peri_R\ (F\ P)) \triangleleft \boldsymbol{U}(\$tr' = \$tr) \triangleright (\bigsqcap\ P{\in}A \cdot peri_R\ (F\ P)))$
by (*simp add*: *ExtChoice-tri-rdes-def closure rdes assms*)
also have $... = ((\bigsqcup\ P{\in}A \cdot pre_R\ (F\ P)) \Rightarrow_r (\bigsqcup\ P{\in}A \cdot pre_R\ (F\ P) \Rightarrow_r peri_R\ (F\ P)) \triangleleft \boldsymbol{U}(\$tr' = \$tr) \triangleright (\bigsqcap\ P{\in}A \cdot pre_R\ (F\ P) \Rightarrow_r peri_R\ (F\ P)))$
by (*simp add*: *NSRD-peri-under-pre assms closure cong*: *UINF-cong USUP-cong*)
also have $... = ((\bigsqcup\ P{\in}A \cdot RR(pre_R\ (F\ P))) \Rightarrow_r (\bigsqcup\ P{\in}A \cdot RR(pre_R\ (F\ P)) \Rightarrow_r RR(peri_R\ (F\ P))) \triangleleft \boldsymbol{U}(\$tr' = \$tr) \triangleright (\bigsqcap\ P{\in}A \cdot RR(pre_R\ (F\ P)) \Rightarrow_r RR(peri_R\ (F\ P))))$
by (*simp add*: *Healthy-if assms closure cong*: *UINF-cong USUP-cong*)
also from *assms*(*1*) **have** $... = ((\bigsqcup\ P{\in}A \cdot RR(pre_R\ (F\ P))) \Rightarrow_r (\bigsqcup\ P{\in}A \cdot RR(pre_R\ (F\ P)) \Rightarrow_r RR(peri_R\ (F\ P)))) \triangleleft \boldsymbol{U}(\$tr' = \$tr) \triangleright ((\bigsqcap\ P{\in}A \cdot RR(pre_R\ (F\ P)) \Rightarrow_r RR(peri_R\ (F\ P))))$
by (*rel-auto*)
finally show *?thesis*
by (*simp add*: *Healthy-if NSRD-peri-under-pre assms closure cong*: *UINF-cong USUP-cong*)
qed

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

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

assumes $A \neq \{\}$ $\bigwedge$ *i. i*$\in$*A* $\Longrightarrow$ *F i is NCSP*
shows $post_R$(*EXTCHOICE A F*) = ($\bigsqcap$ *P*$\in$*A* $\cdot$ $post_R$ (*F P*))
(**is** *?lhs* = *?rhs*)
proof –
have *?lhs* = (($\bigsqcup$ *P*$\in$*A* $\cdot$ pre_R (*F P*)) $\Rightarrow_r$ ($\bigsqcap$ *P*$\in$*A* $\cdot$ $post_R$ (*F P*)))
by (*simp add*: *ExtChoice-tri-rdes-def closure rdes assms*)
also have ... = (($\bigsqcup$ *P*$\in$*A* $\cdot$ pre_R (*F P*)) $\Rightarrow_r$ ($\bigsqcap$ *P*$\in$*A* $\cdot$ pre_R (*F P*) $\Rightarrow_r$ $post_R$ (*F P*)))
by (*simp add*: *NSRD-post-under-pre assms closure cong*: *UINF-cong*)
also have ... = ($\bigsqcap$ *P*$\in$*A* $\cdot$ pre_R (*F P*) $\Rightarrow_r$ $post_R$ (*F P*))
by (*rel-auto*)
finally show *?thesis*
by (*simp add*: *NSRD-post-under-pre assms closure cong*: *UINF-cong*)
qed

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

lemma *periR-extChoice* [*rdes*]:
assumes *P is NCSP Q is NCSP*
shows $peri_R$(*P* $\Box$ *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*))
by (*simp add*: *ExtChoice-def assms cong*: *UINF-cong USUP-cong*)

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 $\bigwedge$ *i. i*$\in$*A* $\Longrightarrow$ *F i is NCSP*
shows *EXTCHOICE A F is CSP4*
proof (*cases A* = {})
case *True* **thus** *?thesis*
by (*simp add*: *ExtChoice-empty Healthy-def CSP4-def*, *simp add*: *Skip-is-CSP Stop-left-zero*)

```
next
  case False
  have 1:(¬_r (¬_r pre_R (EXTCHOICE A F)) ;;_h R1 true) = pre_R (EXTCHOICE A F)
  proof –
    have ⋀ P. P ∈ A ⟹ (¬_r pre_R(F P)) ;; R1 true = (¬_r pre_R(F P))
      by (simp add: NCSP-Healthy-subset-member NCSP-implies-NSRD NSRD-neg-pre-unit assms)
    thus ?thesis
       apply (simp add: False preR-EXTCHOICE closure NCSP-set-unrest-pre-wait' assms not-UINF
seq-UINF-distr not-USUP)
      apply (rule USUP-cong)
      apply (simp add: rpred assms closure)
      done
  qed
  have 2: $st´ ♯ peri_R (EXTCHOICE A F)
  proof –
    have a: ⋀ P. P ∈ A ⟹ $st´ ♯ pre_R(F P)
      by (simp add: NCSP-Healthy-subset-member NCSP-implies-NSRD NSRD-st'-unrest-pre assms)
    have b: ⋀ P. P ∈ A ⟹ $st´ ♯ peri_R(F P)
      by (simp add: NCSP-Healthy-subset-member NCSP-implies-NSRD NSRD-st'-unrest-peri assms)
    from a b show ?thesis
      apply (subst periR-ExtChoice)
         apply (simp-all add: assms closure unrest CSP4-set-unrest-pre-st' NCSP-set-unrest-pre-wait'
False)
      done
  qed
  have 3: $ref´ ♯ post_R (EXTCHOICE A F)
  proof –
    have a: ⋀ P. P ∈ A ⟹ $ref´ ♯ pre_R(F P)
      by (simp add: CSP4-ref'-unrest-pre assms closure)
    have b: ⋀ P. P ∈ A ⟹ $ref´ ♯ post_R(F P)
      by (simp add: CSP4-ref'-unrest-post assms closure)
    from a b show ?thesis
     by (subst postR-ExtChoice, simp-all add: assms CSP4-set-unrest-pre-st' NCSP-set-unrest-pre-wait'
unrest False)
  qed
  show ?thesis
    by (rule CSP4-tri-intro, simp-all add: 1 2 3 assms closure)
       (metis 1 R1-seqr-closure rea-not-R1 rea-not-not rea-true-R1)
qed

lemma CSP4-extChoice [closure]:
  assumes P is NCSP Q is NCSP
  shows P □ Q is CSP4
  by (simp add: extChoice-def, rule CSP4-ExtChoice, auto simp add: assms)

lemma NCSP-ExtChoice [closure]:
  assumes ⋀ i. i∈A ⟹ F i is NCSP
  shows EXTCHOICE A F is NCSP
proof (cases A = {})
  case True
  then show ?thesis by (simp add: ExtChoice-empty closure)
next
  case False
  show ?thesis
  proof (rule NCSP-intro)
```

show *1*:*EXTCHOICE A F is CSP*
by (*metis* (*mono-tags*) *CSP-ExtChoice*)
show *EXTCHOICE A F is CSP3*
by (*rule-tac CSP3-SRD-intro*, *simp-all add*: *CSP-Healthy-subset-member CSP3-Healthy-subset-member closure rdes unrest assms 1 False*)
show *EXTCHOICE A F is CSP4*
by (*simp add*: *CSP4-ExtChoice assms*)
qed
qed

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

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

7.5 Productivity and Guardedness

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

show *?thesis*
proof (*rule Productive-intro*, *simp-all add*: *assms closure rdes unrest 1 False*)
have $((\bigsqcup\ i{\in}I \cdot pre_R\ (P\ i)) \wedge (\bigsqcap\ i{\in}I \cdot post_R\ (P\ i))) =$
$((\bigsqcup\ i{\in}I \cdot pre_R\ (P\ i)) \wedge (\bigsqcap\ i{\in}I \cdot (pre_R\ (P\ i) \wedge post_R\ (P\ i))))$
by (*rel-auto*)
moreover have $(\bigsqcap\ i{\in}I \cdot (pre_R\ (P\ i) \wedge post_R\ (P\ i))) = (\bigsqcap\ i{\in}I \cdot ((pre_R\ (P\ i) \wedge post_R\ (P\ i)) \wedge \$tr <_u \$tr'))$
by (*rule UINF-cong*, *metis* (*no-types*, *lifting*) *1 NCSP-implies-CSP Productive-post-refines-tr-increase assms utp-pred-laws.inf.absorb1*)

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

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

```
lemma ExtChoice-Guarded [closure]:
  assumes ⋀ P. P ∈ A ⟹ Guarded P
  shows Guarded (λ X. □P∈A · P(X))
proof (rule GuardedI)
  fix X n
  have ⋀ Y. ((□P∈A · P Y) ∧ gvrt(n+1)) = ((□P∈A · (P Y ∧ gvrt(n+1))) ∧ gvrt(n+1))
  proof –
    fix Y
    let ?lhs = ((□P∈A · P Y) ∧ gvrt(n+1)) and ?rhs = ((□P∈A · (P Y ∧ gvrt(n+1))) ∧ 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 ((□P∈A · (P X ∧ gvrt(n+1))) ∧ gvrt(n+1)) = ((□P∈A · (P (X ∧ gvrt(n)) ∧
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-image: ExtChoice (P ‘ A) = EXTCHOICE A P
  by (rel-auto)

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 (subst (asm) ExtChoice-image[THEN sym], 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 $P \Box Q \Box R = P \Box (Q \Box R)$
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 ... =
$\mathbf{R}_s\ (((pre_R\ P \wedge pre_R\ Q) \wedge pre_R\ R) \vdash$
$(((cmt_R\ P \wedge cmt_R\ Q) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (cmt_R\ P \vee cmt_R\ Q) \wedge cmt_R\ R)$
$\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
$((cmt_R\ P \wedge cmt_R\ Q) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (cmt_R\ P \vee cmt_R\ Q) \vee cmt_R\ R)))$
by (*simp add*: *extChoice-rdes unrest*)
also have ... =
$\mathbf{R}_s\ (((pre_R\ P \wedge pre_R\ Q) \wedge pre_R\ R) \vdash$
$(((cmt_R\ P \wedge cmt_R\ Q) \wedge cmt_R\ R)$
$\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
$((cmt_R\ P \vee cmt_R\ Q) \vee cmt_R\ R)))$
by (*rule cong*[*of* $\mathbf{R}_s$ $\mathbf{R}_s$], *simp*, *rel-auto*)
also have ... =
$\mathbf{R}_s\ ((pre_R\ P \wedge pre_R\ Q \wedge pre_R\ R) \vdash$
$((cmt_R\ P \wedge (cmt_R\ Q \wedge cmt_R\ R)\ )$
$\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
$(cmt_R\ P \vee (cmt_R\ Q \vee cmt_R\ R))))$
by (*simp add*: *conj-assoc disj-assoc*)
also have ... =
$\mathbf{R}_s\ ((pre_R\ P \wedge pre_R\ Q \wedge pre_R\ R) \vdash$
$((cmt_R\ P \wedge (cmt_R\ Q \wedge cmt_R\ R) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (cmt_R\ Q \vee cmt_R\ R))$
$\triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright$
$(cmt_R\ P \vee (cmt_R\ Q \wedge cmt_R\ R) \triangleleft \$tr' =_u \$tr \wedge \$wait' \triangleright (cmt_R\ Q \vee cmt_R\ R))))$
by (*rule cong*[*of* $\mathbf{R}_s$ $\mathbf{R}_s$], *simp*, *rel-auto*)
also have ... = $\mathbf{R}_s(pre_R(P) \vdash cmt_R(P)) \Box (\mathbf{R}_s(pre_R(Q) \vdash cmt_R(Q)) \Box \mathbf{R}_s(pre_R(R) \vdash cmt_R(R)))$
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 $Chaos \,\square\, Q = Chaos$
proof −
have $Chaos \,\square\, Q = \mathbf{R}_s\ (false \vdash true) \,\square\, \mathbf{R}_s(pre_R(Q) \vdash cmt_R(Q))$
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 $P \,\square\, (\bigsqcap\ S) = (\bigsqcap\ Q{\in}S.\ P \,\square\, Q)$
proof −
let $?S1 = pre_R$ ‘ S **and** $?S2 = cmt_R$ ‘ S
have $P \,\square\, (\bigsqcap\ S) = P \,\square\, (\bigsqcap\ Q{\in}S \cdot \mathbf{R}_s(pre_R(Q) \vdash cmt_R(Q)))$
by (*simp add*: *SRD-as-reactive-design*[*THEN sym*] *Healthy-SUPREMUM UINF-as-Sup-collect assms*)
also have ... $= \mathbf{R}_s(pre_R(P) \vdash cmt_R(P)) \,\square\, \mathbf{R}_s((\bigsqcup\ Q \in S \cdot pre_R(Q)) \vdash (\bigsqcap\ Q \in S \cdot cmt_R(Q)))$
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 ... $= (\bigsqcap\ Q{\in}S \cdot \mathbf{R}_s(pre_R(P) \vdash cmt_R(P)) \,\square\, \mathbf{R}_s(pre_R(Q) \vdash cmt_R(Q)))$
by (*simp add*: *extChoice-rdes unrest*)
also have ... $= (\bigsqcap\ Q{\in}S.\ P \,\square\, CSP(Q))$
by (*simp add*: *UINF-as-Sup-collect*, *metis* (*no-types*, *lifting*) *Healthy-if SRD-as-reactive-design assms*(*1*))
also have ... $= (\bigsqcap\ Q{\in}S.\ P \,\square\, Q)$
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 $P \,\square\, (Q \sqcap R) = (P \,\square\, Q) \sqcap (P \,\square\, R)$
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 (□ i∈A · P i) ;; Q = (□ i∈A · P i ;; Q)
proof (cases A = {})
  case True
  then show ?thesis
    by (simp add: ExtChoice-empty NCSP-implies-CSP Stop-left-zero assms(2))
next
  case False
  show ?thesis
  proof −
    have 1:(□ i∈A · P i) = (□ i∈A · (R_s ((pre_R (P i)) ⊢ peri_R (P i) ⋄ (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:(□ i∈A · P i ;; Q) = (□ i∈A · (R_s ((pre_R (P i)) ⊢ peri_R (P i) ⋄ (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 □ Q) ;; R = (P ;; R □ Q ;; R)
  by (rdes-eq' cls: assms)

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

lemma extchoice-StateInvR-refine:
  assumes
    P is NCSP Q is NCSP
    sinv_R(b) ⊑ P sinv_R(b) ⊑ Q
  shows sinv_R(b) ⊑ P □ Q
proof −
  have P is R2 Q is R2 by (simp-all add: closure assms)
  hence 1:
    pre_R P ⊑ [b]_S< [b]_S> ⊑ ([b]_S< ∧ post_R P)
    pre_R Q ⊑ [b]_S< [b]_S> ⊑ ([b]_S< ∧ post_R Q)
   by (metis (no-types, lifting) CRR-implies-RR NCSP-implies-CSP RHS-tri-design-refine SRD-reactive-tri-design
StateInvR-def assms periR-RR postR-RR preR-CRR rea-st-cond-RR rea-true-RR refBy-order st-post-CRR)+
  show ?thesis
   by (rdes-refine-split cls: assms(1−2), simp-all add: 1 closure assms truer-bottom-rpred utp-pred-laws.inf-sup-distrib1)
qed

end
```

8 Stateful-Failure Programs

```
theory utp-sfrd-prog
  imports
    UTP.utp-full
```

utp-sfrd-extchoice
begin

8.1 Conditionals

lemma *NCSP-cond-srea* [*closure*]:
  assumes P *is NCSP* Q *is NCSP*
  shows $P \triangleleft b \triangleright_R Q$ *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 (*if*$_R$ $i{\in}A \cdot g(i) \rightarrow P(i)$ *else* Q *fi*) *is NCSP*
proof (*cases* $A = \{\}$)
  case *True*
  then show *?thesis*
    by (*simp add*: *assms*)
next
  case *False*
  then show *?thesis*
    by (*simp add*: *AlternateR-def closure assms*)
qed

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

8.4 Specification Statement

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

lemma *SpecC-is-NCSP* [*closure*]: *frm*:[*pre*,*post*]$_C$ *is NCSP*
  apply (*simp add*: *SpecC-def*)
  apply (*rule NCSP-rdes-intro*)
      apply (*simp-all add*: *closure unrest*)
   apply (*rel-auto*)+
  done

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

lemma *SpecC-false-pre*: a:[$false$,q]$_C$ = *Chaos*
  by (*rdes-eq*)

lemma *SpecC-false-post*: a:[$true$,$false$]$_C$ = *Miracle*
  by (*rdes-eq*)

lemma *SpecC-refine-seq*:
  vwb-lens a $\Longrightarrow$ a:[p,q]$_C$ $\sqsubseteq$ a:[p,r]$_C$;; a:[r,q]$_C$
  by ((*rdes-refine-split*; *rel-simp*), *metis vwb-lens.put-eq*)

8.5 Assumptions

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

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

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

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

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

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

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

8.6 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**
$while_C$ *b do P od* = *Skip* ;; $while_R$ *b do P od* ;; *Skip*

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

theorem *WhileC-iter-form*:
  assumes *P is NCSP P is Productive*
  shows $while_C$ *b do P od* = ([b]$_C$;; *P*)$^{\star C}$;; [$\neg$ b]$_C$
  by (*simp add*: *WhileC-def WhileR-iter-form assms closure*)

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

theorem *WhileC-rdes-def* [*rdes-def*]:
  assumes *P is CRC Q is CRR R is CRF* $\$st' \sharp$ *Q R is R4*
  shows $while_C\ b\ do\ \mathbf{R}_s(P \vdash Q \diamond R)\ od =$
    $\mathbf{R}_s\ (([b]_c\ ;;\ R)^{\star c}\ wp_r\ ([b]_{S<} \Rightarrow_r P) \vdash (([b]_c\ ;;\ R)^{\star c}\ ;;\ [b]_c\ ;;\ Q) \diamond (([b]_c\ ;;\ R)^{\star c}\ ;;\ [\neg\ b]_c))$
  (**is** *?lhs* = *?rhs*)
proof −
  have $?lhs = ([b]_C\ ;;\ \mathbf{R}_s\ (P \vdash Q \diamond R))^{\star C}\ ;;\ [\neg\ b]_C$
    by (*simp add*: *WhileC-iter-form assms closure unrest Productive-rdes-RR-intro*)
  also have ... = *?rhs*
    by (*simp add*: *rdes-def assms closure unrest rpred wp del*: *rea-star-wp*)
  finally show *?thesis* **.**
qed

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

lemma *WhileC-unfold*:
  assumes *P is NCSP P is Productive*
  shows $WhileC\ b\ P = (P\ ;;\ WhileC\ b\ P) \triangleleft b \triangleright_R Skip$
proof −
  have $WhileC\ b\ P = (Skip \vee [b]_C\ ;;\ P\ ;;\ ([b]_C\ ;;\ P)^{\star C})\ ;;\ [\neg\ b]_C$
    by (*simp add*: *WhileC-iter-form assms closure*)
     (*metis* (*no-types*, *lifting*) *AssumeCircus-NCSP RA1 StarC-unfold assms*(*1*) *csp-theory.Healthy-Sequence disj-upred-def*)
  also have $... = ([\neg\ b]_C \vee [b]_C\ ;;\ P\ ;;\ ([b]_C\ ;;\ P)^{\star C}\ ;;\ [\neg\ b]_C)$
    by (*metis* (*no-types*, *lifting*) *AssumeCircus-AssumeR*(*1*) *RA1 csp-theory.Unit-self seqr-or-distl*)
  also have $... = (P\ ;;\ WhileC\ b\ P) \triangleleft b \triangleright_R Skip$
   by (*metis* (*no-types*, *lifting*) *AssumeCircus-AssumeR*(*2*) *NCSP-implies-NSRD RA1 WhileC-NCSP-closed WhileC-iter-form assms*(*1*) *assms*(*2*) *cond-srea-AssumeR-form csp-theory.Healthy-Sequence csp-theory.Healthy-Unit csp-theory.Unit-Left uinf-or utp-pred-laws.sup-commute*)
  finally show *?thesis* **.**
qed

8.7 Iteration Construction

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

lemma *IterateC-IterateR-def*: *IterateC A g P* = *Skip* ;; *IterateR A g P* ;; *Skip*
  by (*simp add*: *IterateC-def IterateR-def WhileC-def*)

definition *IterateC-list* :: (*'s upred* × (*'s*, *'e*) *action*) *list* ⇒ (*'s*, *'e*) *action* **where**
[*upred-defs*, *ndes-simp*]:
  IterateC-list xs = *IterateC* {*0*..<*length xs*} (λ *i*. *map fst xs* ! *i*) (λ *i*. *map snd xs* ! *i*)

syntax
  -iter-C   :: *pttrn* ⇒ *logic* ⇒ *logic* ⇒ *logic* ⇒ *logic* (do_C -∈- · - → - *od*)
  -iter-gcommC :: *gcomms* ⇒ *logic* (do_C/ - /*od*)

translations
  -iter-C x A g P => *CONST IterateC A* (λ *x*. *g*) (λ *x*. *P*)

-iter-C x A g P <= *CONST IterateC A* (λ *x. g*) (λ *x'. P*)
-iter-gcommC cs $\rightharpoonup$ *CONST IterateC-list cs*
-iter-gcommC (*-gcomm-show cs*) $\leftharpoondown$ *CONST IterateC-list cs*

lemma *IterateC-NCSP-closed* [*closure*]:
assumes
$\bigwedge$ *i. i* $\in$ *I* $\Longrightarrow$ *P*(*i*) *is NCSP*
$\bigwedge$ *i. i* $\in$ *I* $\Longrightarrow$ *P*(*i*) *is Productive*
shows do_C *i*$\in$*I* $\cdot$ *g*(*i*) $\rightarrow$ *P*(*i*) *od is NCSP*
by (*simp add*: *IterateC-IterateR-def IterateR-NSRD-closed NCSP-implies-NSRD NSRD-coerce-NCSP assms*(*1*) *assms*(*2*))

lemma *IterateC-list-NCSP-closed* [*closure*]:
assumes
$\bigwedge$ *b P.* (*b*, *P*) $\in$ *set A* $\Longrightarrow$ *P is NCSP*
$\bigwedge$ *b P.* (*b*, *P*) $\in$ *set A* $\Longrightarrow$ *P is Productive*
shows *IterateC-list A is NCSP*
apply (*simp add*: *IterateC-list-def*, *rule IterateC-NCSP-closed*)
apply (*metis assms atLeastLessThan-iff nth-map nth-mem prod.collapse*)+
done

lemma *IterateC-list-alt-def*:
IterateC-list xs = $while_C$ ($\bigvee$ *b* $\in$ *set*(*map fst xs*) $\cdot$ *b*) *do AlternateR-list xs Chaos od*
proof −
have ($\bigvee$ *i* $\in$ {*0..<length*(*xs*)} $\cdot$ (*map fst xs*) ! *i*) = ($\bigvee$ *b* $\in$ *set*(*map fst xs*) $\cdot$ *b*)
by (*rel-auto*, *metis fst-conv in-set-conv-nth nth-map*)
thus *?thesis*
by (*simp add*: *IterateC-list-def IterateC-def AlternateR-list-def*)
qed

lemma *IterateC-empty*:
do_C *i*$\in${} $\cdot$ *g*(*i*) $\rightarrow$ *P*(*i*) *od* = *Skip*
by (*simp add*: *IterateC-IterateR-def IterateR-empty closure Skip-srdes-left-unit*)

lemma *IterateC-singleton*:
assumes *P k is NCSP P k is Productive*
shows do_C *i*$\in${*k*} $\cdot$ *g*(*i*) $\rightarrow$ *P*(*i*) *od* = $while_C$ *g*(*k*) *do P*(*k*) *od* (**is** *?lhs* = *?rhs*)
by (*simp add*: *IterateC-IterateR-def IterateR-singleton NCSP-implies-NSRD WhileC-def assms*)

lemma *IterateC-outer-refine-intro*:
assumes *I* $\neq$ {} $\bigwedge$ *i. i* $\in$ *I* $\Longrightarrow$ *P i is NCSP* $\bigwedge$ *i. i* $\in$ *I* $\Longrightarrow$ *P i is Productive*
$\bigwedge$ *i. i* $\in$ *I* $\Longrightarrow$ *S* $\sqsubseteq$ (*b i* $\rightarrow_R$ *P i* ;; *S*) *S is NCSP*
S $\sqsubseteq$ $[\neg (\bigsqcap\ i \in I \cdot b\ i)]^{\top}{}_R$
shows *S* $\sqsubseteq$ do_C *i*$\in$*I* $\cdot$ *b*(*i*) $\rightarrow$ *P*(*i*) *od*
proof −
have *S* $\sqsubseteq$ do_R *i*$\in$*I* $\cdot$ *b*(*i*) $\rightarrow$ *P*(*i*) *od*
by (*simp add*: *IterateR-outer-refine-intro NCSP-implies-NSRD assms*)
thus *?thesis*
unfolding *IterateC-IterateR-def*
by (*metis* (*full-types*) *Skip-left-unit Skip-right-unit assms*(*5*) *urel-dioid.mult-isol urel-dioid.mult-isor*)
qed

lemma *IterateC-outer-refine-init-intro*:
assumes
$\bigwedge$ *i. i* $\in$ *A* $\Longrightarrow$ *P i is NCSP*

```
    ⋀ i. i ∈ A ⟹ P i is Productive
    S is NCSP I is NCSP
    S ⊑ I ;; [¬ (∏ i ∈ A · b i)]⊤R
    ⋀ i. i ∈ A ⟹ S ⊑ S ;; b i →R P i
    ⋀ i. i ∈ A ⟹ S ⊑ I ;; b i →R P i
  shows S ⊑ I ;; doC i∈A · b(i) → P(i) od
proof (cases A = {})
  case True
  with assms(5) show ?thesis
    by (simp add: IterateC-empty assms closure Skip-right-unit AssumeR-true NSRD-right-unit)
next
  case False
  have S ⊑ I ;; doR i∈A · b(i) → P(i) od
    by (simp add: IterateR-outer-refine-init-intro NCSP-implies-NSRD assms False)
  thus ?thesis
    unfolding IterateC-IterateR-def
    by (metis (no-types, hide-lams) RA1 Skip-right-unit assms(3) assms(4) urel-dioid.mult-isor)
qed


lemma IterateC-list-outer-refine-intro:
  assumes
    A ≠ [] S is NCSP
    ⋀ b P. (b, P) ∈ set A ⟹ P is NCSP
    ⋀ b P. (b, P) ∈ set A ⟹ P is Productive
    ⋀ b P. (b, P) ∈ set A ⟹ S ⊑ (b →R P ;; S)
    S ⊑ [¬ (∏ (b, P) ∈ set A · b)]⊤R
  shows S ⊑ IterateC-list A
proof –
  have (∏ i ∈ {0..<length(A)} · (map fst A) ! i) = (∏ (b, P) ∈ set A · b)
    by (rel-auto, metis nth-mem prod.exhaust-sel, metis fst-conv in-set-conv-nth nth-map)
  thus ?thesis
    apply (simp add: IterateC-list-def)
    apply (rule IterateC-outer-refine-intro)
     apply (simp-all add: closure assms)
    apply (metis assms(3) nth-mem prod.collapse)
    apply (metis assms(4) nth-mem prod.collapse)
    done
qed

lemma IterateC-list-outer-refine-init-intro:
  assumes
    S is NCSP I is NCSP
    ⋀ b P. (b, P) ∈ set A ⟹ P is NCSP
    ⋀ b P. (b, P) ∈ set A ⟹ P is Productive
    S ⊑ I ;; [¬ (∏ (b, P) ∈ set A · b)]⊤R
    ⋀ b P. (b, P) ∈ set A ⟹ S ⊑ S ;; b →R P
    ⋀ b P. (b, P) ∈ set A ⟹ S ⊑ I ;; b →R P
  shows S ⊑ I ;; IterateC-list A
proof –
  have (∏ i ∈ {0..<length(A)} · (map fst A) ! i) = (∏ (b, P) ∈ set A · b)
    by (rel-auto, metis nth-mem prod.exhaust-sel, metis fst-conv in-set-conv-nth nth-map)
  thus ?thesis
    apply (simp add: IterateC-list-def)
    apply (rule IterateC-outer-refine-init-intro)
```

  apply (*simp-all add*: *closure assms*)
 apply (*metis assms*(*3*) *nth-mem prod.collapse*)
 apply (*metis assms*(*4*) *nth-mem prod.collapse*)
 done
qed

8.8 Assignment

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma *AssignsCSP-as-AssignsR*: $\langle\sigma\rangle_R$;; *Skip* = $\langle\sigma\rangle_C$

by (*rdes-eq*)

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

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

8.9 Assignment with update

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

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

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

syntax
-csp-assign-upd :: *svid* ⇒ *logic* ⇒ *logic* ⇒ *logic* (-[-] $:=_C$ - [*61*,*0*,*62*] *62*)

translations
-csp-assign-upd x k v == *CONST AssignCSP-update* (*CONST udom*) (*CONST uupd*) *x k v*

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

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

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

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

8.10 State abstraction

lemma *ref-unrest-abs-st* [*unrest*]:
$\$ref \sharp P \Longrightarrow \$ref \sharp \langle P\rangle_S$
$\$ref' \sharp P \Longrightarrow \$ref' \sharp \langle P\rangle_S$
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

8.11 Guards

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

syntax
-GuardCSP :: *logic* ⇒ *logic* ⇒ *logic* (**infixr** $\&_C$ *60*)

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

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

lemma *csp-do-cond-conj*:
assumes P *is CRR*
shows $(\lceil b\rceil_{S<} \wedge P) = \Phi(b, id_s, \ll[]\gg) ;; P$
proof −

have $(\lceil b \rceil_{S<} \wedge \mathit{CRR}(P)) = \Phi(b,\ id_s,\ \ll[]\gg)\ ;;\ \mathit{CRR}(P)$
by (*rel-auto*)
thus *?thesis*
by (*simp add*: *Healthy-if assms*)
qed

lemma *Guard-rdes-def* [*rdes-def*]:
assumes *P is RR Q is CRR R is CRR*
shows $g\ \&_C\ \mathbf{R}_s(P \vdash Q \diamond R) = \mathbf{R}_s\ (([g]_{S<} \Rightarrow_r P) \vdash ((\Phi(g,\ id_s,\ \ll[]\gg)\ ;;\ Q) \vee \mathcal{E}(\neg g, \ll[]\gg, \{\}_u)) \diamond (\Phi(g,\ id_s,\ \ll[]\gg)\ ;;\ R))$
(**is** *?lhs* = *?rhs*)
proof −
have $?lhs = \mathbf{R}_s\ ((\lceil g \rceil_{S<} \Rightarrow_r P) \vdash ((P \Rightarrow_r Q) \triangleleft \lceil g \rceil_{S<} \triangleright (\$tr' =_u \$tr)) \diamond (\lceil g \rceil_{S<} \wedge (P \Rightarrow_r R)))$
by (*simp add*: *Guard-tri-design rdes assms closure*)
also have ... $= \mathbf{R}_s\ (([g]_{S<} \Rightarrow_r P) \vdash ((\lceil g \rceil_{S<} \wedge Q) \vee \mathcal{E}(\neg g, \ll[]\gg, \{\}_u)) \diamond (\lceil g \rceil_{S<} \wedge R))$
by (*rel-auto*)
also have ... $= \mathbf{R}_s\ (([g]_{S<} \Rightarrow_r P) \vdash ((\Phi(g,\ id_s,\ \ll[]\gg)\ ;;\ Q) \vee \mathcal{E}(\neg g, \ll[]\gg, \{\}_u)) \diamond (\Phi(g,\ id_s,\ \ll[]\gg)\ ;;\ R))$
by (*simp add*: *assms*(*2*) *assms*(*3*) *csp-do-cond-conj*)
finally show *?thesis* **.**
qed

lemma *Guard-rdes-def'*:
assumes $\$ok' \sharp P$
shows $g\ \&_C\ (\mathbf{R}_s(P \vdash Q)) = \mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r P) \vdash (\lceil g \rceil_{S<} \wedge Q \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
proof −
have $g\ \&_C\ (\mathbf{R}_s(P \vdash Q)) = \mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r pre_R\ (\mathbf{R}_s\ (P \vdash Q))) \vdash (\lceil g \rceil_{S<} \wedge cmt_R\ (\mathbf{R}_s\ (P \vdash Q)) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$
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 ... $= \mathbf{R}_s((\lceil g \rceil_{S<} \Rightarrow_r pre_s \dagger P)[\![true,false/\$ok,\$wait]\!] \vdash (\lceil g \rceil_{S<} \wedge (P \Rightarrow Q) \vee \lceil \neg g \rceil_{S<} \wedge \$tr' =_u \$tr \wedge \$wait'))$

by $(\textit{simp add}: \textit{rdes-export-pre})$
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 $(\textit{rel-auto})$
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 $(\textit{simp add}: \textit{rdes-export-pre})$
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 $(\textit{rule cong}[\textit{of}\ \mathbf{R}_s\ \mathbf{R}_s], \textit{simp}, \textit{rel-auto})$
finally show *?thesis* **.**
qed

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

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

lemma *postR-Guard* [*rdes*]:
assumes P *is NCSP*
shows $post_R(b\ \&_C\ P) = ([b]_{S<} \wedge post_R\ P)$
proof −
have $post_R(b\ \&_C\ P) = ((\lceil b\rceil_{S<} \Rightarrow_r pre_R\ P) \Rightarrow_r (\lceil b\rceil_{S<} \wedge post_R\ P))$
by (*simp add*: *Guard-tri-design rea-post-RHS-design usubst unrest R2c-rea-not R2c-and R2c-rea-impl R2c-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 $b\ \&_C\ P$ *is CSP3*

proof –
  from *assms* **have** *1*:$\$ref \sharp P\llbracket false/\$wait \rrbracket$
    by (*simp add*: *CSP-Guard CSP3-iff*)
  hence $\$ref \sharp pre_R (P\llbracket 0/\$tr \rrbracket)$ $\$ref \sharp pre_R P$ $\$ref \sharp cmt_R P$
    by (*pred-blast*)+
  hence $\$ref \sharp (b \&_C P)\llbracket false/\$wait \rrbracket$
    by (*simp add*: *CSP3-iff GuardCSP-def RHS-def R1-def R2c-def R2s-def R3h-def design-def unrest usubst*)
  thus *?thesis*
    by (*metis CSP3-intro CSP-Guard*)
qed

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

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

lemma *Productive-Guard* [*closure*]:
  assumes *P is CSP P is Productive* $\$wait' \sharp pre_R(P)$
  shows $b \&_C P$ *is Productive*
proof –
  have $b \&_C P = b \&_C \mathbf{R}_s(pre_R(P) \vdash peri_R(P) \diamond (post_R(P) \wedge \$tr <_u \$tr'))$

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

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

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

8.12 Basic events

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

definition *DoCSP* :: ('φ, 'σ) *uexpr* $\Rightarrow$ ('σ, 'φ) *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*]:
  $do_C(a) = \mathbf{R}_s(true_r \vdash (\mathcal{E}(true,\ll[]\gg,\{a\}_u) \diamond \Phi(true,id_s,U([a]))))$
  by (*simp add*: *DoCSP-def do$_u$-def wait′-cond-def*, *rel-auto*)

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

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

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

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

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

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

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

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

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

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

lemma *wp-rea-DoCSP*:

assumes *P is NCSP*
shows $U(\$tr' = \$tr \mathbin{@} [\lceil a \rceil_{S<}] \wedge \$st' = \$st)$ wp_r pre_R P =
$(\neg_r (\neg_r pre_R P)[\![U(\$tr \mathbin{@} [\lceil a \rceil_{S<}])/\$tr]\!])$
by (*simp add*: *wp-rea-def wp-rea-DoCSP-lemma unrest usubst ex-unrest assms closure*)

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

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

8.13 Event prefix

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

abbreviation *OutputCSP c v P* $\equiv$ *PrefixCSP* $(c \cdot v)_u$ *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* $\in$ $[\![H]\!]_H \rightarrow [\![CSP]\!]_H$
  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*:
  $R1\ (R2s\ (U(\$tr\' = \$tr\ @\ [\lceil a\rceil_{S<}]) \wedge \lceil II\rceil_R)) = (U(\$tr\' = \$tr\ @\ [\lceil a\rceil_{S<}]) \wedge \lceil II\rceil_R)$
  by (*rel-auto*)

lemma *PrefixCSP-RHS-tri-lemma2*:
  fixes $P :: ('\sigma,\ '\varphi)\ action$
  assumes $\$ok \sharp P$ $\$wait \sharp P$
  shows $(U(\$tr\' = \$tr\ @\ [\lceil a\rceil_{S<}] \wedge \$st\' = \$st) \wedge \neg\ \$wait\')\ ;;\ P = (\exists\ \$ref \cdot P[\![U(\$tr\ @\ [\lceil a\rceil_{S<}])/\$tr]\!])$
  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 $U(\$tr\' = \$tr\ @\ [\lceil a\rceil_{S<}] \wedge \$st\' = \$st)\ ;;\ P = P[\![U(\$tr\ @\ [\lceil a\rceil_{S<}])/\$tr]\!]$
  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 $pre_R(PrefixCSP\ a\ P) = (\Phi(true,id_s,U([a]))\ wp_r\ pre_R\ P)$
  by (*simp add*: *PrefixCSP-def assms closure rdes rpred Healthy-if wp usubst unrest*)

lemma *PrefixCSP-RHS-tri*:
  assumes *P is NCSP*
  shows $PrefixCSP\ a\ P = \mathbf{R}_s\ (\Phi(true,id_s,U([a]))\ wp_r\ pre_R\ P \vdash (\mathcal{E}(true,\ll[]\gg,\ \{a\}_u) \vee \Phi(true,id_s,U([a]))\ ;;\ peri_R\ P) \diamond \Phi(true,id_s,U([a]))\ ;;\ post_R\ P)$
  by (*simp add*: *PrefixCSP-def Healthy-if unrest assms closure NSRD-composition-wp rdes rpred usubst wp*)

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

lemma *PrefixCSP-rdes-def-1* [*rdes-def*]:
  assumes *P is CRC Q is CRR R is CRR*
    $\$st\' \sharp Q$ $\$ref\' \sharp R$
  shows $PrefixCSP\ a\ (\mathbf{R}_s(P \vdash Q \diamond R)) =$
    $\mathbf{R}_s\ (\Phi(true,id_s,U([a]))\ wp_r\ P \vdash (\mathcal{E}(true,\ll[]\gg,\ \{a\}_u) \vee \Phi(true,id_s,U([a]))\ ;;\ Q) \diamond \Phi(true,id_s,U([a]))\ ;;\ R)$
  by (*simp add*: *PrefixCSP-def Healthy-if assms closure*, *rdes-simp cls*: *assms*)

8.14 Guarded external choice

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

syntax

-GuardedChoiceCSP :: *logic* ⇒ *logic* ⇒ *logic* ⇒ *logic* (□ - ∈ - → - [*0*,*0*,*85*] *86*)

translations
  □ *x*∈*A* → *P* == *CONST GuardedChoiceCSP A* (λ *x*. *P*)

lemma *GuardedChoiceCSP* [*rdes-def*]:
  assumes ⋀ *x*. *P*(*x*) *is NCSP A* ≠ {}
  shows (□ *x*∈*A* → *P*(*x*)) =
    $\mathbf{R}_s$ (($\bigsqcup$ *x* ∈ *A* · Φ(*true*,id_s,≪[*x*]≫) wp_r pre_R (*P x*)) ⊢
      (($\bigsqcup$ *x* ∈ *A* · $\mathcal{E}$(*true*,≪[]≫, {≪*x*≫}$_u$)) ◁ \$*tr*´ $=_u$ \$*tr* ▷ ($\bigsqcap$ *x* ∈ *A* · Φ(*true*,id_s,≪[*x*]≫) ;; $peri_R$ (*P x*))) ⋄
      ($\bigsqcap$ *x* ∈ *A* · Φ(*true*,id_s,≪[*x*]≫) ;; $post_R$ (*P x*)))
  by (*simp add*: *PrefixCSP-RHS-tri assms ExtChoice-tri-rdes closure unrest*, *rel-auto*)

8.15 Input prefix

definition *InputCSP* ::
  ('*a*, 'ϑ) *chan* ⇒ ('*a* ⇒ 'σ *upred*) ⇒ ('*a* ⇒ ('σ, 'ϑ) *action*) ⇒ ('σ, 'ϑ) *action* **where**
[*upred-defs*]: *InputCSP c A P* = (□ *x*∈*UNIV* · *A*(*x*) &$_C$ *PrefixCSP* (*c*·≪*x*≫)$_u$ (*P x*))

definition *InputVarCSP* :: ('*a*, 'ϑ) *chan* ⇒ ('*a* ⟹ 'σ) ⇒ ('*a* ⇒ 'σ *upred*) ⇒ ('σ, 'ϑ) *action* **where**
[*upred-defs*, *rdes-def*]: *InputVarCSP c x A* = *InputCSP c A* (λ *v*. ⟨[*x* $\mapsto_s$ ≪*v*≫]⟩$_C$)

definition do_I ::
  ('*a*, 'ϑ) *chan* ⇒
  ('*a* ⟹ ('σ, 'ϑ) *sfrd*) ⇒
  ('*a* ⇒ ('σ, 'ϑ) *action*) ⇒
  ('σ, 'ϑ) *action* **where**
do_I *c x P* = (
  (\$*tr*´ $=_u$ \$*tr* ∧ {*e* | *P*(*e*) · (*c*·≪*e*≫)$_u$} $\cap_u$ \$*ref*´ $=_u$ {}$_u$)
    ◁ \$*wait*´ ▷
  ((\$*tr*´ − \$*tr*) $\in_u$ {*e* | *P*(*e*) · *U*([(*c*·≪*e*≫)$_u$])} ∧ (*c*·\$*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*]: ⟦ ⋀ *v*. *P*(*v*) *is NCSP* ⟧ ⟹ *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 *InputVarCSP-NCSP* [*closure*]: *InputVarCSP c x A is NCSP*
  by (*simp add*: *AssignsCSP-NCSP InputCSP-NCSP InputVarCSP-def*)

lemma *Productive-InputCSP* [*closure*]:
  ⟦ ⋀ *v*. *P*(*v*) *is NCSP* ⟧ ⟹ *InputCSP x A P is Productive*
  by (*auto simp add*: *InputCSP-def unrest closure intro*: *Productive-ExtChoice*)

lemma *Productive-InputVarCSP* [*closure*]: *InputVarCSP c x A is Productive*
  by (*simp add*: *InputVarCSP-def closure*)

lemma *R4-st-pred-conj-do* [*rpred*]:
  ((*R4* [s_1]$_{S<}$) ∧ Φ(s_2,σ,*t*) ;; *P*) = *R4*(Φ(s_1 ∧ s_2,σ,*t*) ;; *P*)

by (*rel-auto*)

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

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

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

8.16 Renaming

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

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

lemma *RenameCSP-pre-CRC-closed*:
assumes *P is CRR*
shows $\neg_r\ (\neg_r\ P)(|f|)_c\ ;;\ R1\ true \textit{ is } CRC$
apply (*rule CRC-intro′′*)
apply (*simp add*: *unrest closure assms*)
apply (*simp add*: *Healthy-def*, *simp add*: *RC1-def rpred closure CRC-idem assms seqr-assoc*)
done

lemma *RenameCSP-NCSP-closed* [*closure*]:
assumes *P is NCSP*
shows $P(|f|)_C$ *is NCSP*
by (*simp add*: *RenameCSP-def RenameCSP-pre-CRC-closed closure assms unrest*)

lemma *csp-rename-false* [*rpred*]:
$false(|f|)_c = false$
by (*rel-auto*)

lemma *umap-nil* [*simp*]: $map_u\ f \ll[]\gg = \ll[]\gg$
  by (*rel-auto*)

lemma *rename-Skip*: $Skip(\!|f|\!)_C = Skip$
  by (*rdes-eq*)

lemma *rename-Chaos*: $Chaos(\!|f|\!)_C = Chaos$
  by (*rdes-eq-split*; *rel-simp*; *force*)

lemma *rename-Miracle*: $Miracle(\!|f|\!)_C = Miracle$
  by (*rdes-eq*)

lemma *rename-DoCSP*: $(do_C(a))(\!|f|\!)_C = do_C(\ll f\gg(a)_a)$
  by (*rdes-eq*)

8.17 Algebraic laws

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

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

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

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

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

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

lemma *Guard-expansion*:
  assumes *P is NCSP*
  shows $(g_1 \vee g_2) \ \&_C\ P = (g_1 \ \&_C\ P) \ \Box \ (g_2 \ \&_C\ P)$
  apply (*rdes-eq-split cls*: *assms*)
    apply (*rel-simp'*, *fastforce simp add*: *dual-order.order-iff-strict*)
   apply (*rel-simp'*, *simp add*: *dual-order.order-iff-strict*, *fastforce*)
  apply (*rel-simp'*, *simp add*: *dual-order.order-iff-strict*, *fastforce*)
  done

lemma *Conditional-as-Guard*:
  assumes *P is NCSP Q is NCSP*
  shows $P \triangleleft b \triangleright_R Q = b \ \&_C\ P \ \Box \ (\neg\ b) \ \&_C\ Q$
  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 $((a \rightarrow_C P) \Box (b \rightarrow_C Q))$;; $R = (a \rightarrow_C P$;; $R) \Box (b \rightarrow_C Q$;; $R)$
  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 $\Box\ x \in A \rightarrow P(x)$;; $Q = \Box\ x \in A \rightarrow (P(x)$;; $Q)$
  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*) =
        $(\Box\ i \in A \cdot g(i)\ \&_C\ P(i))\ \Box\ (\bigwedge\ i \in A \cdot \neg\ g(i))\ \&_C\ Q$
proof (*cases* $A = \{\}$)
  case *True*
  then show *?thesis* **by** (*simp add*: *ExtChoice-empty extChoice-Stop closure assms*)
next
  case *False*
  show *?thesis*

  proof −
    have *1*:$(\bigsqcap\ i \in A \cdot g\ i \rightarrow_R P\ i) = (\bigsqcap\ i \in A \cdot g\ i \rightarrow_R \mathbf{R}_s(pre_R(P\ i) \vdash peri_R(P\ i) \diamond post_R(P\ i)))$
      by (*rule UINF-cong*, *simp add*: *NCSP-implies-CSP SRD-reactive-tri-design assms*(*1*))
    have *2*:$(\Box\ i \in A \cdot g(i)\ \&_C\ P(i)) = (\Box\ i \in A \cdot g(i)\ \&_C\ \mathbf{R}_s(pre_R(P\ i) \vdash peri_R(P\ i) \diamond post_R(P\ i)))$
      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 USUP-cong ExtChoice-cong*)
  qed
qed

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

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

find-theorems *R4*

end

9 Recursion in Stateful-Failures

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

9.1 Fixed-points

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

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

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

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

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*]: $(\mu_C\ ((;;)\ Q)) = (\mu\ X \cdot Q\ ;;\ CSP\ X)$
  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 $\mathit{peri}_R\ P \sqsubseteq \mathit{post}_R\ Q\ \hat{}\ i\ ;;\ \mathit{peri}_R\ Q$
    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 $P :: {'e}\ \mathit{list} \Rightarrow {'e}\ \mathit{set} \Rightarrow {'s}\ \mathit{upred}$
  assumes
    Q *is NCSP* Q *is Productive* $\mathit{pre}_R(Q) = \mathit{true}_r$
    $[P\ t\ r]_{S<}[\![(t,\ r) \rightarrow (\&tt,\ \$ref')_u]\!] \sqsubseteq \mathit{peri}_R\ Q$
    $[P\ t\ r]_{S<}[\![(t,\ r) \rightarrow (\&tt,\ \$ref')_u]\!] \sqsubseteq \mathit{post}_R\ Q\ ;;_h\ [P\ t\ r]_{S<}[\![(t,\ r) \rightarrow (\&tt,\ \$ref')_u]\!]$
  shows $[\mathit{true} \vdash P\ \mathit{trace}\ \mathit{refs} \mid R]_C \sqsubseteq (\mu_C\ X \cdot Q\ ;;\ X)$
proof (*rule mu-csp-basic-refine*, *simp-all add*: *msubst-pair assms closure alpha rdes rpred Healthy-if R1-false*)
  show $[P\ \mathit{trace}\ \mathit{refs}]_{S<}[\![\mathit{trace} \rightarrow \&tt]\!][\![\mathit{refs} \rightarrow \$ref']\!] \sqsubseteq \mathit{peri}_R\ Q$
    using *assms* **by** (*simp add*: *msubst-pair*)
  show $[P\ \mathit{trace}\ \mathit{refs}]_{S<}[\![\mathit{trace} \rightarrow \&tt]\!][\![\mathit{refs} \rightarrow \$ref']\!] \sqsubseteq \mathit{post}_R\ Q\ ;;\ [P\ \mathit{trace}\ \mathit{refs}]_{S<}[\![\mathit{trace} \rightarrow \&tt]\!][\![\mathit{refs} \rightarrow \$ref']\!]$
    using *assms* **by** (*simp add*: *msubst-pair*)
qed

9.2 Example action expansion

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

lemma *preR-mu-example1* [*rdes*]: $\mathit{pre}_R(\mu\ X \cdot \ll a \gg \rightarrow_C X) = \mathit{true}_r$
  by (*simp add*: *mu-example1 rdes closure unrest wp*)

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

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

end

10 Linking to the Failures-Divergences Model

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

10.1 Failures-Divergences Semantics

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

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

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

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

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*: $`[\$st \mapsto_s \ll s\gg,\ \$tr \mapsto_s \ll[]\gg,\ \$tr' \mapsto_s \ll t\gg] \dagger pre_R\ Q`$
  with *a* **show** $`[\$st \mapsto_s \ll s\gg,\ \$tr \mapsto_s \ll[]\gg,\ \$tr' \mapsto_s \ll t\gg] \dagger pre_R\ P`$
  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 $\neg\ `[\$st \mapsto_s \ll s\gg,\ \$tr \mapsto_s \ll[]\gg,\ \$tr' \mapsto_s \ll t\gg] \dagger pre_R\ P`$
    hence $\neg\ `[\$st \mapsto_s \ll s\gg,\ \$tr \mapsto_s \ll[]\gg,\ \$tr' \mapsto_s \ll t\gg] \dagger CRC(pre_R\ P)`$
      by (*simp add*: *assms closure Healthy-if*)
    hence $`[\$st \mapsto_s \ll s\gg,\ \$tr \mapsto_s \ll[]\gg,\ \$tr' \mapsto_s \ll t\gg] \dagger (\neg_r\ CRC(pre_R\ P))`$
      by (*rel-auto*)
    hence $`[\$st \mapsto_s \ll s\gg,\ \$tr \mapsto_s \ll[]\gg,\ \$tr' \mapsto_s \ll t\gg] \dagger (\neg_r\ pre_R\ P)`$
      by (*simp add*: *assms closure Healthy-if*)
    with *a b* **show** *False*
      by (*rel-auto*)
  qed
qed

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

lemma *periR-refine-failures*:
  assumes *P is NCSP Q is NCSP* $\bigwedge s.\ fl[\![Q]\!]s \subseteq fl[\![P]\!]s$
  shows $(pre_R(P) \wedge peri_R(P)) \sqsubseteq (pre_R(Q) \wedge peri_R(Q))$
proof (*rule CRR-refine-impl-prop*, *simp-all add*: *assms closure unrest subst-unrest-3*)
  fix $t\ s\ r'$
  assume *a*: $`[\$ref' \mapsto_s \ll r'\gg,\ \$st \mapsto_s \ll s\gg,\ \$tr \mapsto_s \ll[]\gg,\ \$tr' \mapsto_s \ll t\gg] \dagger (pre_R\ Q \wedge peri_R\ Q)`$
  from *assms(3)*[*of s*] **have** *b*: $(t,\ r') \in fl[\![Q]\!]s \Longrightarrow (t,\ r') \in fl[\![P]\!]s$
    by (*auto*)

```
  with a show ‘[$ref´ ↦_s «r'», $st ↦_s «s», $tr ↦_s «[]», $tr´ ↦_s «t»] † (pre_R P ∧ peri_R P)‘
    by (simp add: failures-def)
qed

lemma periR-eq-failures:
  assumes P is NCSP Q is NCSP ⋀ s. fl⟦P⟧s = fl⟦Q⟧s
  shows (pre_R(P) ∧ peri_R(P)) = (pre_R(Q) ∧ 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 ⋀ s. tr⟦Q⟧s ⊆ tr⟦P⟧s
  shows (pre_R(P) ∧ post_R(P)) ⊑ (pre_R(Q) ∧ 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 ↦_s «s», $st´ ↦_s «s'», $tr ↦_s «[]», $tr´ ↦_s «t»] † (pre_R Q ∧ post_R Q)‘
  from assms(3)[of s] have b: (t, s') ∈ tr⟦Q⟧s ⟹ (t, s') ∈ tr⟦P⟧s
    by (auto)
  with a show ‘[$st ↦_s «s», $st´ ↦_s «s'», $tr ↦_s «[]», $tr´ ↦_s «t»] † (pre_R P ∧ post_R P)‘
    by (simp add: traces-def)
qed

lemma postR-eq-traces:
  assumes P is NCSP Q is NCSP ⋀ s. tr⟦P⟧s = tr⟦Q⟧s
  shows (pre_R(P) ∧ post_R(P)) = (pre_R(Q) ∧ 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 ⋀ s. dv⟦Q⟧s ⊆ dv⟦P⟧s ⋀ s. fl⟦Q⟧s ⊆ fl⟦P⟧s ⋀ s. tr⟦Q⟧s ⊆ tr⟦P⟧s
  shows P ⊑ Q
proof (rule SRD-refine-intro', simp-all add: closure assms)
  show a: ‘pre_R P ⇒ pre_R Q‘
    using assms(1) assms(2) assms(3) preR-refine-divergences refBy-order by blast
  show peri_R P ⊑ (pre_R P ∧ peri_R Q)
  proof –
    have peri_R P ⊑ (pre_R Q ∧ 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 ⊑ (pre_R P ∧ post_R Q)
  proof –
    have post_R P ⊑ (pre_R Q ∧ post_R Q)
      by (meson assms(1) assms(2) assms(5) postR-refine-traces utp-pred-laws.le-inf-iff)
    then show ?thesis
      by (metis a refBy-order utp-pred-laws.inf.absorb-iff1 utp-pred-laws.inf-assoc)
  qed
qed
```

10.2 Circus Operators

```
lemma traces-Skip:
  tr⟦Skip⟧s = {([], s)}
  by (simp add: traces-def rdes alpha closure, rel-simp)

lemma failures-Skip:
  fl⟦Skip⟧s = {}
```

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

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

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

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

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

lemma *traces-AssignsCSP*:
$tr[\![\langle\sigma\rangle_C]\!]s = \{([], [\![\sigma]\!]_e\ s)\}$
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 $dv[\![g\ \&_C\ P]\!]s = (if\ ([\![g]\!]_e s)\ then\ dv[\![g\ \&_C\ P]\!]s\ else\ \{\})$
  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 *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 $tr[\![P\ ;;\ Q]\!]s =$
$$\{(t_1\ @\ t_2,\ s') \mid t_1\ t_2\ s_0\ s'.\ (t_1,\ s_0) \in tr[\![P]\!]s \wedge (t_2,\ s') \in tr[\![Q]\!]s_0$$
$$\wedge\ (t_1 @ t_2) \notin dv[\![P]\!]s$$
$$\wedge\ (\forall\ (t,\ s_1) \in tr[\![P]\!]s.\ t \le t_1 @ t_2 \longrightarrow (t_1 @ t_2) - t \notin dv[\![Q]\!]s_1)\ \}$$
  (**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 $?\sigma = [\$st \mapsto_s \ll s\gg,\ \$st' \mapsto_s \ll s'\gg,\ \$tr \mapsto_s \ll[]\gg,\ \$tr' \mapsto_s \ll t\gg]$
    assume
      a1: ‘$?\sigma \dagger (post_R\ P\ ;;\ post_R\ Q)$‘ **and**
      a2: ‘$[\$st \mapsto_s \ll s\gg,\ \$tr \mapsto_s \ll[]\gg,\ \$tr' \mapsto_s \ll t\gg] \dagger pre_R\ P$‘ **and**
      a3: ‘$[\$st \mapsto_s \ll s\gg,\ \$tr \mapsto_s \ll[]\gg,\ \$tr' \mapsto_s \ll t\gg] \dagger (post_R\ P\ wp_r\ pre_R\ Q)$‘
    from *a1* **have** ‘$?\sigma \dagger (\exists\ tr_0 \cdot ((post_R\ P)[\![\ll tr_0\gg/\$tr']\!]\ ;;\ (post_R\ Q)[\![\ll tr_0\gg/\$tr]\!]) \wedge \ll tr_0\gg \le_u \$tr')$‘
      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*: $tr_0 \le t$
      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 `?σ † ((post_R P)⟦«s_0»,«tr_0»/$st´,$tr´⟧ ;; (post_R Q)⟦«s_0»,«tr_0»/$st,$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 ↦_s «s», $st´ ↦_s «s_0», $tr ↦_s «[]», $tr´ ↦_s «tr_0»] † post_R P) ;;
          ([$st ↦_s «s_0», $st´ ↦_s «s'», $tr ↦_s «tr_0», $tr´ ↦_s «t»] † post_R Q))`
    by (rel-auto)
  hence `((([$st ↦_s «s», $st´ ↦_s «s_0», $tr ↦_s «[]», $tr´ ↦_s «tr_0»] † post_R P) ∧
          ([$st ↦_s «s_0», $st´ ↦_s «s'», $tr ↦_s «tr_0», $tr´ ↦_s «t»] † post_R Q))`
    by (simp add: seqr-to-conj unrest-any-circus-var assms closure unrest)
  hence postP: `([$st ↦_s «s», $st´ ↦_s «s_0», $tr ↦_s «[]», $tr´ ↦_s «tr_0»] † post_R P)` and
        postQ': `([$st ↦_s «s_0», $st´ ↦_s «s'», $tr ↦_s «tr_0», $tr´ ↦_s «t»] † post_R Q)`
    by (rel-auto)+
   from postQ' have `[$st ↦_s «s_0», $st´ ↦_s «s'»] † [$tr ↦_s «tr_0», $tr´ ↦_s «tr_0» + («t» −
«tr_0»)] † post_R Q`
    using tr0 by (rel-auto)
  hence `[$st ↦_s «s_0», $st´ ↦_s «s'»] † [$tr ↦_s 0, $tr´ ↦_s «t» − «tr_0»] † post_R Q`
    by (simp add: R2-subst-tr closure assms)
  hence postQ: `[$st ↦_s «s_0», $st´ ↦_s «s'», $tr ↦_s «[]», $tr´ ↦_s «t − tr_0»] † post_R Q`
    by (rel-auto)
  have preP: `[$st ↦_s «s», $tr ↦_s «[]», $tr´ ↦_s «tr_0»] † pre_R P`
  proof −
    have (pre_R P)⟦0,«tr_0»/$tr,$tr´⟧ ⊑ (pre_R P)⟦0,«t»/$tr,$tr´⟧
      by (simp add: RC-prefix-refine closure assms tr0)
     hence [$st ↦_s «s», $tr ↦_s «[]», $tr´ ↦_s «tr_0»] † pre_R P ⊑ [$st ↦_s «s», $tr ↦_s «[]», $tr´
↦_s «t»] † pre_R P
      by (rel-auto)
    thus ?thesis
      by (simp add: taut-refine-impl a2)
  qed

  have preQ: `[$st ↦_s «s_0», $tr ↦_s «[]», $tr´ ↦_s «t − tr_0»] † pre_R Q`
  proof −
    from postP a3 have `[$st ↦_s «s_0», $tr ↦_s «tr_0», $tr´ ↦_s «t»] † pre_R Q`
      apply (simp add: wp-rea-def)
      apply (rel-auto)
      using tr0 apply blast+
      done
    hence `[$st ↦_s «s_0»] † [$tr ↦_s «tr_0», $tr´ ↦_s «tr_0» + («t» − «tr_0»)] † pre_R Q`
      by (rel-auto)

    hence `[$st ↦_s «s_0»] † [$tr ↦_s 0, $tr´ ↦_s «t» − «tr_0»] † pre_R Q`
      by (simp add: R2-subst-tr closure assms)
    thus ?thesis
      by (rel-auto)
  qed

  from a2 have ndiv: ¬ `[$st ↦_s «s», $tr ↦_s «[]», $tr´ ↦_s «t»] † (¬_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: ⋀t0 s1.
         ‘[$st ↦s ≪s≫, $tr ↦s ≪[]≫, $tr´ ↦s ≪t0≫] † preR P‘ ⟹
         ‘[$st ↦s ≪s≫, $st´ ↦s ≪s1≫, $tr ↦s ≪[]≫, $tr´ ↦s ≪t0≫] † postR P‘ ⟹
          t0 ≤ t ⟹ ‘[$st ↦s ≪s1≫, $tr ↦s ≪[]≫, $tr´ ↦s ≪t − t0≫] † (¬r preR Q)‘ ⟹ False
  proof −
    fix t0 s1
    assume b:
      ‘[$st ↦s ≪s≫, $tr ↦s ≪[]≫, $tr´ ↦s ≪t0≫] † preR P‘
      ‘[$st ↦s ≪s≫, $st´ ↦s ≪s1≫, $tr ↦s ≪[]≫, $tr´ ↦s ≪t0≫] † postR P‘
      t0 ≤ t
      ‘[$st ↦s ≪s1≫, $tr ↦s ≪[]≫, $tr´ ↦s ≪t − t0≫] † (¬r preR Q)‘

    from a3 have c: ∀ (s0, t0) · ≪t0≫ ≤u ≪t≫
                           ∧ [$st ↦s ≪s≫, $st´ ↦s ≪s0≫, $tr ↦s ≪[]≫, $tr´ ↦s ≪t0≫] † postR P
                           ⇒ [$st ↦s ≪s0≫, $tr ↦s ≪[]≫, $tr´ ↦s ≪t≫ − ≪t0≫] † preR Q‘
      by (simp add: wp-rea-circus-form-alt[of postR P preR Q] closure assms unrest usubst)
        (rel-simp)

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

  show ∃ t1 t2.
          t = t1 @ t2 ∧
          (∃ s0. ‘[$st ↦s ≪s≫, $tr ↦s ≪[]≫, $tr´ ↦s ≪t1≫] † preR P ∧
                  [$st ↦s ≪s≫, $st´ ↦s ≪s0≫, $tr ↦s ≪[]≫, $tr´ ↦s ≪t1≫] † postR P‘ ∧
                 ‘[$st ↦s ≪s0≫, $tr ↦s ≪[]≫, $tr´ ↦s ≪t2≫] † preR Q ∧
                  [$st ↦s ≪s0≫, $st´ ↦s ≪s′≫, $tr ↦s ≪[]≫, $tr´ ↦s ≪t2≫] † postR Q‘ ∧
                  ¬ ‘[$st ↦s ≪s≫, $tr ↦s ≪[]≫, $tr´ ↦s ≪t1 @ t2≫] † (¬r preR P)‘ ∧
                  (∀ t0 s1. ‘[$st ↦s ≪s≫, $tr ↦s ≪[]≫, $tr´ ↦s ≪t0≫] † preR P ∧
                           [$st ↦s ≪s≫, $st´ ↦s ≪s1≫, $tr ↦s ≪[]≫, $tr´ ↦s ≪t0≫] † postR P‘ ⟶
                           t0 ≤ t1 @ t2 ⟶ ¬ ‘[$st ↦s ≪s1≫, $tr ↦s ≪[]≫, $tr´ ↦s ≪(t1 @ t2) − t0≫] †
(¬r preR Q)‘))
    apply (rule-tac x=tr0 in exI)
    apply (rule-tac x=(t − tr0) in exI)
    apply (auto)
    using tr0 apply auto[1]
    apply (rule-tac x=s0 in exI)
    apply (auto intro:wpr simp add: taut-conj preP preQ postP postQ ndiv wpr t-minus-tr0)
    done
qed

show ?rhs ⊆ ?lhs
proof (rdes-expand cls: assms, simp add: traces-def divergences-def rdes closure assms rdes-def unrest
rpred usubst, auto)
  fix t1 t2 :: 'e list and s0 s′ :: 's
  assume
    a1: ¬ ‘[$st ↦s ≪s≫, $tr ↦s ≪[]≫, $tr´ ↦s ≪t1 @ t2≫] † (¬r preR P)‘ and
    a2: ‘[$st ↦s ≪s≫, $tr ↦s ≪[]≫, $tr´ ↦s ≪t1≫] † preR P‘ and
    a3: ‘[$st ↦s ≪s≫, $st´ ↦s ≪s0≫, $tr ↦s ≪[]≫, $tr´ ↦s ≪t1≫] † postR P‘ and
    a4: ‘[$st ↦s ≪s0≫, $tr ↦s ≪[]≫, $tr´ ↦s ≪t2≫] † preR Q‘ and
    a5: ‘[$st ↦s ≪s0≫, $st´ ↦s ≪s′≫, $tr ↦s ≪[]≫, $tr´ ↦s ≪t2≫] † postR Q‘ and
    a6: ∀ t s1. ‘[$st ↦s ≪s≫, $tr ↦s ≪[]≫, $tr´ ↦s ≪t≫] † preR P ∧
                [$st ↦s ≪s≫, $st´ ↦s ≪s1≫, $tr ↦s ≪[]≫, $tr´ ↦s ≪t≫] † postR P‘ ⟶
```

$t \leq t_1$ @ $t_2 \longrightarrow \neg$ ‘[\$*st* $\mapsto_s$ «s_1», \$*tr* $\mapsto_s$ «[]», \$*tr*´ $\mapsto_s$ «(t_1 @ t_2) − t»] † ($\neg_r$ *pre*$_R$ *Q*)‘

from *a1* **have** *preP*: ‘[\$*st* $\mapsto_s$ «*s*», \$*tr* $\mapsto_s$ «[]», \$*tr*´ $\mapsto_s$ «t_1 @ t_2»] † (*pre*$_R$ *P*)‘
  by (*simp add*: *taut-not unrest-all-circus-vars-st assms closure unrest*, *rel-auto*)

have ‘[\$*st* $\mapsto_s$ «s_0», \$*st*´ $\mapsto_s$ «*s'*», \$*tr* $\mapsto_s$ «t_1», \$*tr*´ $\mapsto_s$ «t_1»+«t_2»] † *post*$_R$ *Q*‘
proof −
  have [\$*st* $\mapsto_s$ «s_0», \$*st*´ $\mapsto_s$ «*s'*», \$*tr* $\mapsto_s$ «[]», \$*tr*´ $\mapsto_s$ «t_2»] † *post*$_R$ *Q* =
    [\$*st* $\mapsto_s$ «s_0», \$*st*´ $\mapsto_s$ «*s'*»] † [\$*tr* $\mapsto_s$ «[]», \$*tr*´ $\mapsto_s$ «t_2»] † *post*$_R$ *Q*
    by *rel-auto*
  also have ... = [\$*st* $\mapsto_s$ «s_0», \$*st*´ $\mapsto_s$ «*s'*»] † [\$*tr* $\mapsto_s$ «t_1», \$*tr*´ $\mapsto_s$ «t_1»+«t_2»] † *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* $\mapsto_s$ «*s*», \$*st*´ $\mapsto_s$ «*s'*», \$*tr* $\mapsto_s$ «[]», \$*tr*´ $\mapsto_s$ «t_1 @ t_2»] † (*post*$_R$ *P* ;; *post*$_R$ *Q*)‘
  by (*rel-auto*, *meson Prefix-Order.prefixI*)

have ‘[\$*st* $\mapsto_s$ «s_0», \$*tr* $\mapsto_s$ «t_1», \$*tr*´ $\mapsto_s$ «t_1»+«t_2»] † *pre*$_R$ *Q*‘
proof −
  have [\$*st* $\mapsto_s$ «s_0», \$*tr* $\mapsto_s$ «t_1», \$*tr*´ $\mapsto_s$ «t_1»+«t_2»] † *pre*$_R$ *Q* =
    [\$*st* $\mapsto_s$ «s_0»] † [\$*tr* $\mapsto_s$ «t_1», \$*tr*´ $\mapsto_s$ «t_1»+«t_2»] † *pre*$_R$ *Q*
    by *rel-auto*
  also have ... = [\$*st* $\mapsto_s$ «s_0»] † [\$*tr* $\mapsto_s$ *0*, \$*tr*´ $\mapsto_s$ «t_2»] † *pre*$_R$ *Q*
    by (*simp add*: *R2-subst-tr assms closure*)
  finally show *?thesis* **using** *a4*
    by (*rel-auto*)
qed

from *a6*
have *a6'*: $\bigwedge$ *t* s_1. ⟦ $t \leq t_1$ @ t_2; ‘[\$*st* $\mapsto_s$ «*s*», \$*tr* $\mapsto_s$ «[]», \$*tr*´ $\mapsto_s$ «*t*»] † *pre*$_R$ *P*‘; ‘[\$*st* $\mapsto_s$ «*s*», \$*st*´ $\mapsto_s$ «s_1», \$*tr* $\mapsto_s$ «[]», \$*tr*´ $\mapsto_s$ «*t*»] † *post*$_R$ *P*‘ ⟧ $\Longrightarrow$
    ‘[\$*st* $\mapsto_s$ «s_1», \$*tr* $\mapsto_s$ «[]», \$*tr*´ $\mapsto_s$ «(t_1 @ t_2) − *t*»] † *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* $\mapsto_s$ «*s*», \$*tr* $\mapsto_s$ «[]», \$*tr*´ $\mapsto_s$ «t_1 @ t_2»] † (*post*$_R$ *P* *wp*$_r$ *pre*$_R$ *Q*)‘
proof −
  have $\bigwedge$ s_1 t_0. ⟦ $t_0 \leq t_1$ @ t_2; ‘[\$*st* $\mapsto_s$ «*s*», \$*st*´ $\mapsto_s$ «s_1», \$*tr* $\mapsto_s$ «[]», \$*tr*´ $\mapsto_s$ «t_0»] † *post*$_R$ *P*‘ ⟧
    $\Longrightarrow$ ‘[\$*st* $\mapsto_s$ «s_1», \$*tr* $\mapsto_s$ «[]», \$*tr*´ $\mapsto_s$ «(t_1 @ t_2) − t_0»] † *pre*$_R$ *Q*‘
  proof −
    fix s_1 t_0
    assume *c*:$t_0 \leq t_1$ @ t_2 ‘[\$*st* $\mapsto_s$ «*s*», \$*st*´ $\mapsto_s$ «s_1», \$*tr* $\mapsto_s$ «[]», \$*tr*´ $\mapsto_s$ «t_0»] † *post*$_R$ *P*‘

    have *preP'*: ‘[\$*st* $\mapsto_s$ «*s*», \$*tr* $\mapsto_s$ «[]», \$*tr*´ $\mapsto_s$ «t_0»] † *pre*$_R$ *P*‘
    proof −
      have (*pre*$_R$ *P*)⟦*0*,«t_0»/\$*tr*,\$*tr*´⟧ $\sqsubseteq$ (*pre*$_R$ *P*)⟦*0*,«t_1 @ t_2»/\$*tr*,\$*tr*´⟧
        by (*simp add*: *RC-prefix-refine closure assms c*)
      hence [\$*st* $\mapsto_s$ «*s*», \$*tr* $\mapsto_s$ «[]», \$*tr*´ $\mapsto_s$ «t_0»] † *pre*$_R$ *P* $\sqsubseteq$ [\$*st* $\mapsto_s$ «*s*», \$*tr* $\mapsto_s$ «[]», \$*tr*´ $\mapsto_s$ «t_1 @ t_2»] † *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* $\mapsto_s$ ≪s_1≫, \$*tr* $\mapsto_s$ ≪[]≫, \$*tr´* $\mapsto_s$ ≪(t_1 @ t_2) − t_0≫] † 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* $\mapsto_s$ ≪*s*≫, \$*tr* $\mapsto_s$ ≪[]≫, \$*tr´* $\mapsto_s$ ≪t_1 @ t_2≫] † pre_R *P* ∧
[\$*st* $\mapsto_s$ ≪*s*≫, \$*tr* $\mapsto_s$ ≪[]≫, \$*tr´* $\mapsto_s$ ≪t_1 @ t_2≫] † ($post_R$ *P* wp_r pre_R *Q*)) ∧
[\$*st* $\mapsto_s$ ≪*s*≫, \$*st´* $\mapsto_s$ ≪*s′*≫, \$*tr* $\mapsto_s$ ≪[]≫, \$*tr´* $\mapsto_s$ ≪t_1 @ t_2≫] † ($post_R$ *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*≫ $\rightarrow_C$ *P*⟧*s* = {(*a* # *t*, *s′*) | *t s′*. (*t*, *s′*) ∈ *tr*⟦*P*⟧*s*}
apply (*auto simp add*: *PrefixCSP-def traces-seq traces-do divergences-do lit.rep-eq assms closure Healthy-if trace-divergence-disj*)
apply (*meson assms trace-divergence-disj*)
done

10.3 Deadlock Freedom

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

definition *CDF* :: (*′s*, *′e*) *action* **where**
[*rdes-def*]: *CDF* = $\mathbf{R}_s$($true_r$ ⊢ (⨅ (*s*, *t*, *E*, *e*) · $\mathcal{E}$(≪*s*≫, ≪*t*≫, ≪*insert e E*≫)) ⋄ $true_r$)

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

lemma *CDF-seq-idem*: *CDF* ;; *CDF* = *CDF*
by (*rdes-eq*)

lemma *CDF-refine-intro*: *CDF* ⊑ *P* ⟹ *CDF* ⊑ (*CDF* ;; *P*)

by (*metis CDF-seq-idem urel-dioid.mult-isol*)

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

lemma *CDF-ext-st* [*alpha*]: $CDF \oplus_p abs\text{-}st_L = CDF$
by (*rdes-eq*)

end

11 Meta-theory for Stateful-Failure Reactive Designs

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

References

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

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