

The Ipurge Unwinding Theorem for CSP Noninterference Security

Pasquale Noce

Security Certification Specialist at Arjo Systems - Gep S.p.A.
pasquale dot noce dot lavoro at gmail dot com
pasquale dot noce at arjowiggins-it dot com

February 6, 2026

Abstract

The definition of noninterference security for Communicating Sequential Processes requires to consider any possible future, i.e. any indefinitely long sequence of subsequent events and any indefinitely large set of refused events associated to that sequence, for each process trace. In order to render the verification of the security of a process more straightforward, there is a need of some sufficient condition for security such that just individual accepted and refused events, rather than unbounded sequences and sets of events, have to be considered.

Of course, if such a sufficient condition were necessary as well, it would be even more valuable, since it would permit to prove not only that a process is secure by verifying that the condition holds, but also that a process is not secure by verifying that the condition fails to hold.

This paper provides a necessary and sufficient condition for CSP noninterference security, which indeed requires to just consider individual accepted and refused events and applies to the general case of a possibly intransitive policy. This condition follows Rushby's output consistency for deterministic state machines with outputs, and has to be satisfied by a specific function mapping security domains into equivalence relations over process traces. The definition of this function makes use of an intransitive purge function following Rushby's one; hence the name given to the condition, Ipurge Unwinding Theorem.

Furthermore, in accordance with Hoare's formal definition of deterministic processes, it is shown that a process is deterministic just in case it is a trace set process, i.e. it may be identified by means of a trace set alone, matching the set of its traces, in place of a failures-divergences pair. Then, variants of the Ipurge Unwinding Theorem are proven for deterministic processes and trace set processes.

Contents

1 The Ipurge Unwinding Theorem in its general form

2

1.1	Propaedeutic definitions and lemmas	3
1.2	Additional intransitive purge functions and their properties	10
1.3	A domain-relation map based on intransitive purge	25
1.4	The Ipurge Unwinding Theorem: proof of condition sufficiency	27
1.5	The Ipurge Unwinding Theorem: proof of condition necessity	40
2	The Ipurge Unwinding Theorem for deterministic and trace set processes	44
2.1	Deterministic processes	45
2.2	Trace set processes	50

1 The Ipurge Unwinding Theorem in its general form

```

theory IpurgeUnwinding
imports Noninterference-CSP.CSPNoninterference List-Interleaving.ListInterleaving
begin

```

The definition of noninterference security for Communicating Sequential Processes given in [6] requires to consider any possible future, i.e. any indefinitely long sequence of subsequent events and any indefinitely large set of refused events associated to that sequence, for each process trace. In order to render the verification of the security of a process more straightforward, there is a need of some sufficient condition for security such that just individual accepted and refused events, rather than unbounded sequences and sets of events, have to be considered.

Of course, if such a sufficient condition were necessary as well, it would be even more valuable, since it would permit to prove not only that a process is secure by verifying that the condition holds, but also that a process is not secure by verifying that the condition fails to hold.

This section provides a necessary and sufficient condition for CSP noninterference security, which indeed requires to just consider individual accepted and refused events and applies to the general case of a possibly intransitive policy. This condition follows Rushby’s output consistency for deterministic state machines with outputs [8], and has to be satisfied by a specific function mapping security domains into equivalence relations over process traces. The definition of this function makes use of an intransitive purge function following Rushby’s one; hence the name given to the condition, *Ipurge Unwinding Theorem*.

The contents of this paper are based on those of [6]. The salient points of definitions and proofs are commented; for additional information, cf. Isabelle documentation, particularly [5], [4], [3], and [2].

For the sake of brevity, given a function F of type $'a_1 \Rightarrow \dots \Rightarrow 'a_m \Rightarrow 'a_{m+1} \Rightarrow \dots \Rightarrow 'a_n \Rightarrow 'b$, the explanatory text may discuss of F using attributes that would more exactly apply to a term of type $'a_{m+1} \Rightarrow \dots \Rightarrow 'a_n \Rightarrow 'b$. In this case, it shall be understood that strictly speaking, such attributes apply to a term matching pattern $F a_1 \dots a_m$.

1.1 Propaedeutic definitions and lemmas

The definition of CSP noninterference security formulated in [6] requires that some sets of events be refusals, i.e. sets of refused events, for some traces. Therefore, a sufficient condition for security just involving individual refused events will require that some single events be refused, viz. form singleton refusals, after the occurrence of some traces. However, such a statement may actually be a sufficient condition for security just in the case of a process such that the union of any set of singleton refusals for a given trace is itself a refusal for that trace.

This turns out to be true if and only if the union of any set A of refusals, not necessarily singletons, is still a refusal. The direct implication is trivial. As regards the converse one, let A' be the set of the singletons included in some element of A . Then, each element of A' is a singleton refusal by virtue of rule $\llbracket (?xs, ?Y) \in failures ?P; ?X \subseteq ?Y \rrbracket \implies (?xs, ?X) \in failures ?P$, so that the union of the elements of A' , which is equal to the union of the elements of A , is a refusal by hypothesis.

This property, henceforth referred to as *refusals union closure* and formalized in what follows, clearly holds for any process admitting a meaningful interpretation, as it would be a nonsense, in the case of a process modeling a real system, to say that some sets of events are refused after the occurrence of a trace, but their union is not. Thus, taking the refusals union closure of the process as an assumption for the equivalence between process security and a given condition, as will be done in the Ipurge Unwinding Theorem, does not give rise to any actual limitation on the applicability of such a result.

As for predicates *view partition* and *future consistent*, defined here below as well, they translate Rushby's predicates *view-partitioned* and *output consistent* [8], applying to deterministic state machines with outputs, into Hoare's Communicating Sequential Processes model of computation [1]. The reason for the verbal difference between the active form of predicate *view partition* and the passive form of predicate *view-partitioned* is that the implied subject of the former is a domain-relation map rather than a process, whose homologous in [8], viz. a machine, is the implied subject of the latter predicate instead.

More remarkably, the formal differences with respect to Rushby's original predicates are the following ones:

- The relations in the range of the domain-relation map hold between event lists rather than machine states.
- The domains appearing as inputs of the domain-relation map do not unnecessarily encompass all the possible values of the data type of domains, but just the domains in the range of the event-domain map.
- The equality of the outputs in domain u produced by machine states equivalent for u , as required by output consistency, is replaced by the equality of the events in domain u accepted or refused after the occurrence of event lists equivalent for u ; hence the name of the property, *future consistency*.

An additional predicate, *weakly future consistent*, renders future consistency less strict by requiring the equality of subsequent accepted and refused events to hold only for event domains not allowed to be affected by some event domain.

type-synonym $('a, 'd)$ *dom-rel-map* = $'d \Rightarrow ('a \text{ list} \times 'a \text{ list}) \text{ set}$

type-synonym $('a, 'd)$ *domset-rel-map* = $'d \text{ set} \Rightarrow ('a \text{ list} \times 'a \text{ list}) \text{ set}$

definition *ref-union-closed* :: $'a \text{ process} \Rightarrow \text{bool}$ **where**

ref-union-closed $P \equiv$

$$\forall xs A. (\exists X. X \in A) \longrightarrow (\forall X \in A. (xs, X) \in \text{failures } P) \longrightarrow (xs, \bigcup X \in A. X) \in \text{failures } P$$

definition *view-partition* ::

$'a \text{ process} \Rightarrow ('a \Rightarrow 'd) \Rightarrow ('a, 'd) \text{ dom-rel-map} \Rightarrow \text{bool}$ **where**
view-partition $P D R \equiv \forall u \in \text{range } D. \text{equiv } (\text{traces } P) (R u)$

definition *next-dom-events* ::

$'a \text{ process} \Rightarrow ('a \Rightarrow 'd) \Rightarrow 'd \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ set}$ **where**
next-dom-events $P D u xs \equiv \{x. u = D x \wedge x \in \text{next-events } P xs\}$

definition *ref-dom-events* ::

$'a \text{ process} \Rightarrow ('a \Rightarrow 'd) \Rightarrow 'd \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ set}$ **where**
ref-dom-events $P D u xs \equiv \{x. u = D x \wedge \{x\} \in \text{refusals } P xs\}$

definition *future-consistent* ::

$'a \text{ process} \Rightarrow ('a \Rightarrow 'd) \Rightarrow ('a, 'd) \text{ dom-rel-map} \Rightarrow \text{bool}$ **where**
future-consistent $P D R \equiv$
 $\forall u \in \text{range } D. \forall xs ys. (xs, ys) \in R u \longrightarrow$
 $\text{next-dom-events } P D u xs = \text{next-dom-events } P D u ys \wedge$
 $\text{ref-dom-events } P D u xs = \text{ref-dom-events } P D u ys$

definition *weakly-future-consistent* ::

$'a \text{ process} \Rightarrow ('d \times 'd) \text{ set} \Rightarrow ('a \Rightarrow 'd) \Rightarrow ('a, 'd) \text{ dom-rel-map} \Rightarrow \text{bool}$ **where**

weakly-future-consistent $P I D R \equiv$
 $\forall u \in \text{range } D \cap (-I) \text{ `` range } D. \forall xs \ ys. (xs, ys) \in R \ u \longrightarrow$
 $\text{next-dom-events } P \ D \ u \ xs = \text{next-dom-events } P \ D \ u \ ys \wedge$
 $\text{ref-dom-events } P \ D \ u \ xs = \text{ref-dom-events } P \ D \ u \ ys$

Here below are some lemmas propaedeutic for the proof of the Ipurge Unwinding Theorem, just involving constants defined in [6].

lemma *process-rule-2-traces*:

$xs @ xs' \in \text{traces } P \implies xs \in \text{traces } P$

proof (*simp add: traces-def Domain-iff, erule exE, rule-tac x = {} in exI*)

qed (*rule process-rule-2-failures*)

lemma *process-rule-4 [rule-format]*:

$(xs, X) \in \text{failures } P \longrightarrow (xs @ [x], \{\}) \in \text{failures } P \vee (xs, \text{insert } x \ X) \in \text{failures } P$

proof (*simp add: failures-def*)

have *Rep-process* $P \in \text{process-set}$ (**is** $?P' \in -$) **by** (*rule Rep-process*)

hence $\forall xs \ x \ X. (xs, X) \in \text{fst } ?P' \longrightarrow$

$(xs @ [x], \{\}) \in \text{fst } ?P' \vee (xs, \text{insert } x \ X) \in \text{fst } ?P'$

by (*simp add: process-set-def process-prop-4-def*)

thus $(xs, X) \in \text{fst } ?P' \longrightarrow$

$(xs @ [x], \{\}) \in \text{fst } ?P' \vee (xs, \text{insert } x \ X) \in \text{fst } ?P'$

by blast

qed

lemma *failures-traces*:

$(xs, X) \in \text{failures } P \implies xs \in \text{traces } P$

by (*simp add: traces-def Domain-iff, rule exI*)

lemma *traces-failures*:

$xs \in \text{traces } P \implies (xs, \{\}) \in \text{failures } P$

proof (*simp add: traces-def Domain-iff, erule exE*)

qed (*erule process-rule-3, simp*)

lemma *sinks-interference [rule-format]*:

$D \ x \in \text{sinks } I \ D \ u \ xs \longrightarrow$

$(u, D \ x) \in I \vee (\exists v \in \text{sinks } I \ D \ u \ xs. (v, D \ x) \in I)$

proof (*induction xs rule: rev-induct, simp, rule impI*)

fix $x' \ xs$

assume

A: $D \ x \in \text{sinks } I \ D \ u \ xs \longrightarrow$

$(u, D \ x) \in I \vee (\exists v \in \text{sinks } I \ D \ u \ xs. (v, D \ x) \in I)$ **and**

B: $D \ x \in \text{sinks } I \ D \ u \ (xs @ [x'])$

show $(u, D \ x) \in I \vee (\exists v \in \text{sinks } I \ D \ u \ (xs @ [x']). (v, D \ x) \in I)$

proof (*cases (u, D x') \in I \vee (\exists v \in \text{sinks } I \ D \ u \ xs. (v, D x') \in I)*)

case True

hence $D \ x = D \ x' \vee D \ x \in \text{sinks } I \ D \ u \ xs$ **using B by simp**

```

moreover {
  assume  $C: D x = D x'$ 
  have ?thesis using True
  proof (rule disjE, erule-tac [2] bexE)
    assume  $(u, D x') \in I$ 
    hence  $(u, D x) \in I$  using  $C$  by simp
    thus ?thesis ..
  next
    fix  $v$ 
    assume  $(v, D x') \in I$ 
    hence  $(v, D x) \in I$  using  $C$  by simp
    moreover assume  $v \in \text{sinks } I D u xs$ 
    hence  $v \in \text{sinks } I D u (xs @ [x'])$  by simp
    ultimately have  $\exists v \in \text{sinks } I D u (xs @ [x']). (v, D x) \in I$  ..
    thus ?thesis ..
  qed
}
moreover {
  assume  $D x \in \text{sinks } I D u xs$ 
  with  $A$  have  $(u, D x) \in I \vee (\exists v \in \text{sinks } I D u xs. (v, D x) \in I)$  ..
  hence ?thesis
  proof (rule disjE, erule-tac [2] bexE)
    assume  $(u, D x) \in I$ 
    thus ?thesis ..
  next
    fix  $v$ 
    assume  $(v, D x) \in I$ 
    moreover assume  $v \in \text{sinks } I D u xs$ 
    hence  $v \in \text{sinks } I D u (xs @ [x'])$  by simp
    ultimately have  $\exists v \in \text{sinks } I D u (xs @ [x']). (v, D x) \in I$  ..
    thus ?thesis ..
  qed
}
ultimately show ?thesis ..
next
  case False
  hence  $C: \text{sinks } I D u (xs @ [x']) = \text{sinks } I D u xs$  by simp
  hence  $D x \in \text{sinks } I D u xs$  using  $B$  by simp
  with  $A$  have  $(u, D x) \in I \vee (\exists v \in \text{sinks } I D u xs. (v, D x) \in I)$  ..
  thus ?thesis using  $C$  by simp
qed
qed

```

lemma *sinks-interference-eq*:

$$((u, D x) \in I \vee (\exists v \in \text{sinks } I D u xs. (v, D x) \in I)) = (D x \in \text{sinks } I D u (xs @ [x]))$$

proof (*rule iffI, erule-tac [2] contrapos-pp, simp-all (no-asm-simp)*)

qed (*erule contrapos-nn, rule sinks-interference*)

In what follows, some lemmas concerning the constants defined above are proven.

In the definition of predicate *ref-union-closed*, the conclusion that the union of a set of refusals is itself a refusal for the same trace is subordinated to the condition that the set of refusals be nonempty. The first lemma shows that in the absence of this condition, the predicate could only be satisfied by a process admitting any event list as a trace, which proves that the condition must be present for the definition to be correct.

The subsequent lemmas prove that, for each domain u in the ranges respectively taken into consideration, the image of u under a future consistent or weakly future consistent domain-relation map may only correlate a pair of event lists such that either both are traces, or both are not traces. Finally, it is demonstrated that future consistency implies weak future consistency.

lemma

assumes $A: \forall xs A. (\forall X \in A. (xs, X) \in failures P) \longrightarrow$
 $(xs, \bigcup X \in A. X) \in failures P$
shows $\forall xs. xs \in traces P$

proof

fix xs

have $(\forall X \in \{\}. (xs, X) \in failures P) \longrightarrow (xs, \bigcup X \in \{\}. X) \in failures P$

using A **by** *blast*

moreover have $\forall X \in \{\}. (xs, X) \in failures P$ **by** *simp*

ultimately have $(xs, \bigcup X \in \{\}. X) \in failures P$ **..**

thus $xs \in traces P$ **by** (*rule failures-traces*)

qed

lemma *traces-dom-events*:

assumes $A: u \in range D$

shows $xs \in traces P =$

$(next-dom-events P D u xs \cup ref-dom-events P D u xs \neq \{\})$

$(is - = (?S \neq \{\}))$

proof

have $\exists x. u = D x$ **using** A **by** (*simp add: image-def*)

then obtain x **where** $B: u = D x$ **..**

assume $xs \in traces P$

hence $(xs, \{\}) \in failures P$ **by** (*rule traces-failures*)

hence $(xs @ [x], \{\}) \in failures P \vee (xs, \{x\}) \in failures P$ **by** (*rule process-rule-4*)

moreover {

assume $(xs @ [x], \{\}) \in failures P$

hence $xs @ [x] \in traces P$ **by** (*rule failures-traces*)

hence $x \in next-dom-events P D u xs$

using B **by** (*simp add: next-dom-events-def next-events-def*)

hence $x \in ?S$ **..**

}

moreover {

assume $(xs, \{x\}) \in failures P$

hence $x \in \text{ref-dom-events } P D u xs$
using B **by** (*simp add: ref-dom-events-def refusals-def*)
hence $x \in ?S ..$
}
ultimately have $x \in ?S ..$
hence $\exists x. x \in ?S ..$
thus $?S \neq \{\}$ **by** (*subst ex-in-conv [symmetric]*)
next
assume $?S \neq \{\}$
hence $\exists x. x \in ?S$ **by** (*subst ex-in-conv*)
then obtain x **where** $x \in ?S ..$
moreover **{**
assume $x \in \text{next-dom-events } P D u xs$
hence $xs @ [x] \in \text{traces } P$ **by** (*simp add: next-dom-events-def next-events-def*)
hence $xs \in \text{traces } P$ **by** (*rule process-rule-2-traces*)
}
moreover **{**
assume $x \in \text{ref-dom-events } P D u xs$
hence $(xs, \{x\}) \in \text{failures } P$ **by** (*simp add: ref-dom-events-def refusals-def*)
hence $xs \in \text{traces } P$ **by** (*rule failures-traces*)
}
ultimately show $xs \in \text{traces } P ..$
qed

lemma *fc-traces*:

assumes

A : *future-consistent* $P D R$ **and**

B : $u \in \text{range } D$ **and**

C : $(xs, ys) \in R u$

shows $(xs \in \text{traces } P) = (ys \in \text{traces } P)$

proof –

have $\forall u \in \text{range } D. \forall xs ys. (xs, ys) \in R u \longrightarrow$

$\text{next-dom-events } P D u xs = \text{next-dom-events } P D u ys \wedge$

$\text{ref-dom-events } P D u xs = \text{ref-dom-events } P D u ys$

using A **by** (*simp add: future-consistent-def*)

hence $\forall xs ys. (xs, ys) \in R u \longrightarrow$

$\text{next-dom-events } P D u xs = \text{next-dom-events } P D u ys \wedge$

$\text{ref-dom-events } P D u xs = \text{ref-dom-events } P D u ys$

using $B ..$

hence $(xs, ys) \in R u \longrightarrow$

$\text{next-dom-events } P D u xs = \text{next-dom-events } P D u ys \wedge$

$\text{ref-dom-events } P D u xs = \text{ref-dom-events } P D u ys$

by *blast*

hence $\text{next-dom-events } P D u xs = \text{next-dom-events } P D u ys \wedge$

$\text{ref-dom-events } P D u xs = \text{ref-dom-events } P D u ys$

using $C ..$

hence $\text{next-dom-events } P D u xs \cup \text{ref-dom-events } P D u xs \neq \{\} =$

$(\text{next-dom-events } P D u ys \cup \text{ref-dom-events } P D u ys \neq \{\})$

by *simp*

moreover have $xs \in \text{traces } P =$
 $(\text{next-dom-events } P D u xs \cup \text{ref-dom-events } P D u xs \neq \{\})$
using B **by** $(\text{rule traces-dom-events})$
moreover have $ys \in \text{traces } P =$
 $(\text{next-dom-events } P D u ys \cup \text{ref-dom-events } P D u ys \neq \{\})$
using B **by** $(\text{rule traces-dom-events})$
ultimately show $?thesis$ **by** simp
qed

lemma $wfc\text{-traces}$:

assumes

A : $\text{weakly-future-consistent } P I D R$ **and**

B : $u \in \text{range } D \cap (-I)$ “ $\text{range } D$ **and**

C : $(xs, ys) \in R u$

shows $(xs \in \text{traces } P) = (ys \in \text{traces } P)$

proof –

have $\forall u \in \text{range } D \cap (-I)$ “ $\text{range } D. \forall xs ys. (xs, ys) \in R u \longrightarrow$
 $\text{next-dom-events } P D u xs = \text{next-dom-events } P D u ys \wedge$

$\text{ref-dom-events } P D u xs = \text{ref-dom-events } P D u ys$

using A **by** $(\text{simp add: weakly-future-consistent-def})$

hence $\forall xs ys. (xs, ys) \in R u \longrightarrow$

$\text{next-dom-events } P D u xs = \text{next-dom-events } P D u ys \wedge$

$\text{ref-dom-events } P D u xs = \text{ref-dom-events } P D u ys$

using B **..**

hence $(xs, ys) \in R u \longrightarrow$

$\text{next-dom-events } P D u xs = \text{next-dom-events } P D u ys \wedge$

$\text{ref-dom-events } P D u xs = \text{ref-dom-events } P D u ys$

by blast

hence $\text{next-dom-events } P D u xs = \text{next-dom-events } P D u ys \wedge$

$\text{ref-dom-events } P D u xs = \text{ref-dom-events } P D u ys$

using C **..**

hence $\text{next-dom-events } P D u xs \cup \text{ref-dom-events } P D u xs \neq \{\} =$
 $(\text{next-dom-events } P D u ys \cup \text{ref-dom-events } P D u ys \neq \{\})$

by simp

moreover have B' : $u \in \text{range } D$ **using** B **..**

hence $xs \in \text{traces } P =$

$(\text{next-dom-events } P D u xs \cup \text{ref-dom-events } P D u xs \neq \{\})$

by $(\text{rule traces-dom-events})$

moreover have $ys \in \text{traces } P =$

$(\text{next-dom-events } P D u ys \cup \text{ref-dom-events } P D u ys \neq \{\})$

using B' **by** $(\text{rule traces-dom-events})$

ultimately show $?thesis$ **by** simp

qed

lemma $fc\text{-implies-wfc}$:

$\text{future-consistent } P D R \implies \text{weakly-future-consistent } P I D R$

by $(\text{simp only: future-consistent-def weakly-future-consistent-def, blast})$

Finally, the definition is given of an auxiliary function *singleton-set*, whose output is the set of the singleton subsets of a set taken as input, and then some basic properties of this function are proven.

definition *singleton-set* :: 'a set \Rightarrow 'a set set **where**
singleton-set $X \equiv \{Y. \exists x \in X. Y = \{x\}\}$

lemma *singleton-set-some*:

$(\exists Y. Y \in \text{singleton-set } X) = (\exists x. x \in X)$
proof (*rule iffI, simp-all add: singleton-set-def, erule-tac [!] exE, erule bexE*)
fix x
assume $x \in X$
thus $\exists x. x \in X$..
next
fix x
assume $A: x \in X$
have $\{x\} = \{x\}$..
hence $\exists x' \in X. \{x\} = \{x'\}$ **using** A ..
thus $\exists Y. \exists x' \in X. Y = \{x'\}$ **by** (*rule exI*)
qed

lemma *singleton-set-union*:

$(\bigcup Y \in \text{singleton-set } X. Y) = X$
proof (*subst singleton-set-def, rule equalityI, rule-tac [!] subsetI*)
fix x
assume $A: x \in (\bigcup Y \in \{Y'. \exists x' \in X. Y' = \{x'\}\}. Y)$
show $x \in X$
proof (*rule UN-E [OF A], simp*)
qed (*erule bexE, simp*)
next
fix x
assume $A: x \in X$
show $x \in (\bigcup Y \in \{Y'. \exists x' \in X. Y' = \{x'\}\}. Y)$
proof (*rule UN-I [of {x}]*)
qed (*simp-all add: A*)
qed

1.2 Additional intransitive purge functions and their properties

Functions *sinks-aux*, *ipurge-tr-aux*, and *ipurge-ref-aux*, defined here below, are auxiliary versions of functions *sinks*, *ipurge-tr*, and *ipurge-ref* taking as input a set of domains rather than a single domain. As shown below, these functions are useful for the study of single domain ones, involved in the definition of CSP noninterference security [6], since they distribute over list concatenation, while being susceptible to be expressed in terms of the corresponding single domain functions in case the input set of domains is a

singleton.

A further function, *unaffected-domains*, takes as inputs a set of domains U and an event list xs , and outputs the set of the event domains not allowed to be affected by U after the occurrence of xs .

function *sinks-aux* ::
 $('d \times 'd) \text{ set} \Rightarrow ('a \Rightarrow 'd) \Rightarrow 'd \text{ set} \Rightarrow 'a \text{ list} \Rightarrow 'd \text{ set}$ **where**
sinks-aux - - $U \ [] = U \ |$
sinks-aux $I D U (xs \ @ \ [x]) = (if \ \exists v \in \text{sinks-aux } I D U xs. (v, D x) \in I$
 then insert $(D x) (\text{sinks-aux } I D U xs)$
 else sinks-aux $I D U xs)$
proof (*atomize-elim, simp-all add: split-paired-all*)
qed (*rule rev-cases, rule disjI1, assumption, simp*)
termination by *lexicographic-order*

function *ipurge-tr-aux* ::
 $('d \times 'd) \text{ set} \Rightarrow ('a \Rightarrow 'd) \Rightarrow 'd \text{ set} \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ list}$ **where**
ipurge-tr-aux - - - $\ [] = \ [] \ |$
ipurge-tr-aux $I D U (xs \ @ \ [x]) = (if \ \exists v \in \text{sinks-aux } I D U xs. (v, D x) \in I$
 then ipurge-tr-aux $I D U xs$
 else ipurge-tr-aux $I D U xs \ @ \ [x])$
proof (*atomize-elim, simp-all add: split-paired-all*)
qed (*rule rev-cases, rule disjI1, assumption, simp*)
termination by *lexicographic-order*

definition *ipurge-ref-aux* ::
 $('d \times 'd) \text{ set} \Rightarrow ('a \Rightarrow 'd) \Rightarrow 'd \text{ set} \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ set}$ **where**
ipurge-ref-aux $I D U xs X \equiv$
 $\{x \in X. \forall v \in \text{sinks-aux } I D U xs. (v, D x) \notin I\}$

definition *unaffected-domains* ::
 $('d \times 'd) \text{ set} \Rightarrow ('a \Rightarrow 'd) \Rightarrow 'd \text{ set} \Rightarrow 'a \text{ list} \Rightarrow 'd \text{ set}$ **where**
unaffected-domains $I D U xs \equiv$
 $\{u \in \text{range } D. \forall v \in \text{sinks-aux } I D U xs. (v, u) \notin I\}$

Function *ipurge-tr-rev*, defined here below in terms of function *sources*, is the reverse of function *ipurge-tr* with regard to both the order in which events are considered, and the criterion by which they are purged.

In some detail, both functions *sources* and *ipurge-tr-rev* take as inputs a domain u and an event list xs , whose recursive decomposition is performed by item prepending rather than appending. Then:

- *sources* outputs the set of the domains of the events in xs allowed to affect u ;
- *ipurge-tr-rev* outputs the sublist of xs obtained by recursively deleting the events not allowed to affect u , as detected via function *sources*.

In other words, these functions follow Rushby's ones *sources* and *ipurge* [8], formalized in [6] as *c-sources* and *c-ipurge*. The only difference consists of dropping the implicit supposition that the noninterference policy be reflexive, as done in the definition of CPS noninterference security [6]. This goal is achieved by defining the output of function *sources*, when it is applied to the empty list, as being the empty set rather than the singleton comprised of the input domain.

As for functions *sources-aux* and *ipurge-tr-rev-aux*, they are auxiliary versions of functions *sources* and *ipurge-tr-rev* taking as input a set of domains rather than a single domain. As shown below, these functions distribute over list concatenation, while being susceptible to be expressed in terms of the corresponding single domain functions in case the input set of domains is a singleton.

```
primrec sources :: ('d × 'd) set ⇒ ('a ⇒ 'd) ⇒ 'd ⇒ 'a list ⇒ 'd set where
sources - - - [] = {} |
sources I D u (x # xs) =
  (if (D x, u) ∈ I ∨ (∃ v ∈ sources I D u xs. (D x, v) ∈ I)
   then insert (D x) (sources I D u xs)
   else sources I D u xs)
```

```
primrec ipurge-tr-rev :: ('d × 'd) set ⇒ ('a ⇒ 'd) ⇒ 'd ⇒ 'a list ⇒ 'a list where
ipurge-tr-rev - - - [] = [] |
ipurge-tr-rev I D u (x # xs) = (if D x ∈ sources I D u (x # xs)
  then x # ipurge-tr-rev I D u xs
  else ipurge-tr-rev I D u xs)
```

```
primrec sources-aux ::
('d × 'd) set ⇒ ('a ⇒ 'd) ⇒ 'd set ⇒ 'a list ⇒ 'd set where
sources-aux - - U [] = U |
sources-aux I D U (x # xs) = (if ∃ v ∈ sources-aux I D U xs. (D x, v) ∈ I
  then insert (D x) (sources-aux I D U xs)
  else sources-aux I D U xs)
```

```
primrec ipurge-tr-rev-aux ::
('d × 'd) set ⇒ ('a ⇒ 'd) ⇒ 'd set ⇒ 'a list ⇒ 'a list where
ipurge-tr-rev-aux - - [] = [] |
ipurge-tr-rev-aux I D U (x # xs) = (if ∃ v ∈ sources-aux I D U xs. (D x, v) ∈ I
  then x # ipurge-tr-rev-aux I D U xs
  else ipurge-tr-rev-aux I D U xs)
```

Here below are some lemmas on functions *sinks-aux*, *ipurge-tr-aux*, *ipurge-ref-aux*, and *unaffected-domains*. As anticipated above, these lemmas essentially concern distributivity over list concatenation and expressions in terms of single domain functions in the degenerate case of a singleton set of domains.

lemma *sinks-aux-subset*:

$U \subseteq \text{sinks-aux } I D U xs$

proof (*induction xs rule: rev-induct, simp-all, rule impI*)

qed (*rule subset-insertI2*)

lemma *sinks-aux-single-dom*:

$\text{sinks-aux } I D \{u\} xs = \text{insert } u (\text{sinks } I D u xs)$

by (*induction xs rule: rev-induct, simp-all add: insert-commute*)

lemma *sinks-aux-single-event*:

$\text{sinks-aux } I D U [x] = (\text{if } \exists v \in U. (v, D x) \in I$

$\text{then insert } (D x) U$

$\text{else } U)$

proof –

have $\text{sinks-aux } I D U [x] = \text{sinks-aux } I D U ([] @ [x])$ **by** *simp*

thus *?thesis* **by** (*simp only: sinks-aux.simps*)

qed

lemma *sinks-aux-cons*:

$\text{sinks-aux } I D U (x \# xs) = (\text{if } \exists v \in U. (v, D x) \in I$

$\text{then sinks-aux } I D (\text{insert } (D x) U) xs$

$\text{else sinks-aux } I D U xs)$

proof (*induction xs rule: rev-induct, case-tac [!] $\exists v \in U. (v, D x) \in I$,*

simp-all add: sinks-aux-single-event del: sinks-aux.simps(2))

fix $x' xs$

assume $A: \text{sinks-aux } I D U (x \# xs) = \text{sinks-aux } I D (\text{insert } (D x) U) xs$

(**is** $?S = ?S'$)

show $\text{sinks-aux } I D U (x \# xs @ [x']) =$

$\text{sinks-aux } I D (\text{insert } (D x) U) (xs @ [x'])$

proof (*cases $\exists v \in ?S. (v, D x') \in I$*)

case *True*

hence $\text{sinks-aux } I D U ((x \# xs) @ [x']) = \text{insert } (D x') ?S$

by (*simp only: sinks-aux.simps, simp*)

moreover have $\exists v \in ?S'. (v, D x') \in I$ **using** A **and** *True* **by** *simp*

hence $\text{sinks-aux } I D (\text{insert } (D x) U) (xs @ [x']) = \text{insert } (D x') ?S'$

by *simp*

ultimately show *?thesis* **using** A **by** *simp*

next

case *False*

hence $\text{sinks-aux } I D U ((x \# xs) @ [x']) = ?S$

by (*simp only: sinks-aux.simps, simp*)

moreover have $\neg (\exists v \in ?S'. (v, D x') \in I)$ **using** A **and** *False* **by** *simp*

hence $\text{sinks-aux } I D (\text{insert } (D x) U) (xs @ [x']) = ?S'$ **by** *simp*

ultimately show *?thesis* **using** A **by** *simp*

qed

next

fix $x' xs$

assume $A: \text{sinks-aux } I D U (x \# xs) = \text{sinks-aux } I D U xs$

(**is** $?S = ?S'$)

show $\text{sinks-aux } I D U (x \# xs @ [x']) = \text{sinks-aux } I D U (xs @ [x'])$
proof (*cases* $\exists v \in ?S. (v, D x') \in I$)
 case *True*
 hence $\text{sinks-aux } I D U ((x \# xs) @ [x']) = \text{insert } (D x') ?S$
 by (*simp only: sinks-aux.simps, simp*)
 moreover have $\exists v \in ?S'. (v, D x') \in I$ **using** *A* **and** *True* **by** *simp*
 hence $\text{sinks-aux } I D U (xs @ [x']) = \text{insert } (D x') ?S'$ **by** *simp*
 ultimately show *?thesis* **using** *A* **by** *simp*
 next
 case *False*
 hence $\text{sinks-aux } I D U ((x \# xs) @ [x']) = ?S$
 by (*simp only: sinks-aux.simps, simp*)
 moreover have $\neg (\exists v \in ?S'. (v, D x') \in I)$ **using** *A* **and** *False* **by** *simp*
 hence $\text{sinks-aux } I D U (xs @ [x']) = ?S'$ **by** *simp*
 ultimately show *?thesis* **using** *A* **by** *simp*
 qed
qed

lemma *ipurge-tr-aux-single-dom*:

$\text{ipurge-tr-aux } I D \{u\} xs = \text{ipurge-tr } I D u xs$

proof (*induction xs rule: rev-induct, simp*)

fix $x xs$

assume *A*: $\text{ipurge-tr-aux } I D \{u\} xs = \text{ipurge-tr } I D u xs$

show $\text{ipurge-tr-aux } I D \{u\} (xs @ [x]) = \text{ipurge-tr } I D u (xs @ [x])$

proof (*cases* $\exists v \in \text{sinks-aux } I D \{u\} xs. (v, D x) \in I$,

simp-all only: ipurge-tr-aux.simps if-True if-False)

case *True*

hence $(u, D x) \in I \vee (\exists v \in \text{sinks } I D u xs. (v, D x) \in I)$

by (*simp add: sinks-aux-single-dom*)

hence $\text{ipurge-tr } I D u (xs @ [x]) = \text{ipurge-tr } I D u xs$ **by** *simp*

thus $\text{ipurge-tr-aux } I D \{u\} xs = \text{ipurge-tr } I D u (xs @ [x])$

using *A* **by** *simp*

next

case *False*

hence $\neg ((u, D x) \in I \vee (\exists v \in \text{sinks } I D u xs. (v, D x) \in I))$

by (*simp add: sinks-aux-single-dom*)

hence $D x \notin \text{sinks } I D u (xs @ [x])$

by (*simp only: sinks-interference-eq, simp*)

hence $\text{ipurge-tr } I D u (xs @ [x]) = \text{ipurge-tr } I D u xs @ [x]$ **by** *simp*

thus $\text{ipurge-tr-aux } I D \{u\} xs @ [x] = \text{ipurge-tr } I D u (xs @ [x])$

using *A* **by** *simp*

qed

qed

lemma *ipurge-ref-aux-single-dom*:

$\text{ipurge-ref-aux } I D \{u\} xs X = \text{ipurge-ref } I D u xs X$

by (*simp add: ipurge-ref-aux-def ipurge-ref-def sinks-aux-single-dom*)

lemma *ipurge-ref-aux-all* [*rule-format*]:

$(\forall u \in U. \neg (\exists v \in D \text{ ' } (X \cup \text{set } xs). (u, v) \in I)) \longrightarrow$
 $\text{ipurge-ref-aux } I D U xs X = X$
proof (*induction xs, simp-all add: ipurge-ref-aux-def sinks-aux-cons*)
qed (*rule impI, rule equalityI, rule-tac [!] subsetI, simp-all*)

lemma ipurge-ref-all:
 $\neg (\exists v \in D \text{ ' } (X \cup \text{set } xs). (u, v) \in I) \implies \text{ipurge-ref } I D u xs X = X$
by (*subst ipurge-ref-aux-single-dom [symmetric], rule ipurge-ref-aux-all, simp*)

lemma unaffected-domains-single-dom:
 $\{x \in X. D x \in \text{unaffected-domains } I D \{u\} xs\} = \text{ipurge-ref } I D u xs X$
by (*simp add: ipurge-ref-def unaffected-domains-def sinks-aux-single-dom*)

Here below are some lemmas on functions *sources*, *ipurge-tr-rev*, *sources-aux*, and *ipurge-tr-rev-aux*. As anticipated above, the lemmas on the last two functions basically concern distributivity over list concatenation and expressions in terms of single domain functions in the degenerate case of a singleton set of domains.

lemma sources-sinks:
 $\text{sources } I D u xs = \text{sinks } (I^{-1}) D u (\text{rev } xs)$
by (*induction xs, simp-all*)

lemma sources-sinks-aux:
 $\text{sources-aux } I D U xs = \text{sinks-aux } (I^{-1}) D U (\text{rev } xs)$
by (*induction xs, simp-all*)

lemma sources-aux-subset:
 $U \subseteq \text{sources-aux } I D U xs$
by (*subst sources-sinks-aux, rule sinks-aux-subset*)

lemma sources-aux-append:
 $\text{sources-aux } I D U (xs @ ys) = \text{sources-aux } I D (\text{sources-aux } I D U ys) xs$
by (*induction xs, simp-all*)

lemma sources-aux-append-nil [rule-format]:
 $\text{sources-aux } I D U ys = U \longrightarrow$
 $\text{sources-aux } I D U (xs @ ys) = \text{sources-aux } I D U xs$
by (*induction xs, simp-all*)

lemma ipurge-tr-rev-aux-append:
 $\text{ipurge-tr-rev-aux } I D U (xs @ ys) =$
 $\text{ipurge-tr-rev-aux } I D (\text{sources-aux } I D U ys) xs @ \text{ipurge-tr-rev-aux } I D U ys$
by (*induction xs, simp-all add: sources-aux-append*)

lemma ipurge-tr-rev-aux-nil-1 [rule-format]:
 $\text{ipurge-tr-rev-aux } I D U xs = [] \longrightarrow (\forall u \in U. \neg (\exists v \in D \text{ ' } \text{set } xs. (v, u) \in I))$

by (induction xs rule: rev-induct, simp-all add: ipurge-tr-rev-aux-append)

lemma ipurge-tr-rev-aux-nil-2 [rule-format]:

$(\forall u \in U. \neg (\exists v \in D \text{ ' set } xs. (v, u) \in I)) \longrightarrow \text{ipurge-tr-rev-aux } I D U xs = []$

by (induction xs rule: rev-induct, simp-all add: ipurge-tr-rev-aux-append)

lemma ipurge-tr-rev-aux-nil:

$(\text{ipurge-tr-rev-aux } I D U xs = []) = (\forall u \in U. \neg (\exists v \in D \text{ ' set } xs. (v, u) \in I))$

proof (rule iffI, rule ballI, erule ipurge-tr-rev-aux-nil-1, assumption)

qed (rule ipurge-tr-rev-aux-nil-2, erule bspec)

lemma ipurge-tr-rev-aux-nil-sources [rule-format]:

$\text{ipurge-tr-rev-aux } I D U xs = [] \longrightarrow \text{sources-aux } I D U xs = U$

by (induction xs , simp-all)

lemma ipurge-tr-rev-aux-append-nil-1 [rule-format]:

$\text{ipurge-tr-rev-aux } I D U ys = [] \longrightarrow$

$\text{ipurge-tr-rev-aux } I D U (xs @ ys) = \text{ipurge-tr-rev-aux } I D U xs$

by (induction xs , simp-all add: ipurge-tr-rev-aux-nil-sources sources-aux-append-nil)

lemma ipurge-tr-rev-aux-first [rule-format]:

$\text{ipurge-tr-rev-aux } I D U xs = x \# ws \longrightarrow$

$(\exists ys zs. xs = ys @ x \# zs \wedge$

$\text{ipurge-tr-rev-aux } I D (\text{sources-aux } I D U (x \# zs)) ys = [] \wedge$

$(\exists v \in \text{sources-aux } I D U zs. (D x, v) \in I))$

proof (induction xs , simp, rule impI)

fix $x' xs$

assume

$A: \text{ipurge-tr-rev-aux } I D U xs = x \# ws \longrightarrow$

$(\exists ys zs. xs = ys @ x \# zs \wedge$

$\text{ipurge-tr-rev-aux } I D (\text{sources-aux } I D U (x \# zs)) ys = [] \wedge$

$(\exists v \in \text{sources-aux } I D U zs. (D x, v) \in I))$ **and**

$B: \text{ipurge-tr-rev-aux } I D U (x' \# xs) = x \# ws$

show $\exists ys zs. x' \# xs = ys @ x \# zs \wedge$

$\text{ipurge-tr-rev-aux } I D (\text{sources-aux } I D U (x \# zs)) ys = [] \wedge$

$(\exists v \in \text{sources-aux } I D U zs. (D x, v) \in I)$

proof (cases $\exists v \in \text{sources-aux } I D U xs. (D x', v) \in I$)

case True

then have $x' = x$ **using** B **by** simp

with True **have** $x' \# xs = x \# xs \wedge$

$\text{ipurge-tr-rev-aux } I D (\text{sources-aux } I D U (x \# xs)) [] = [] \wedge$

$(\exists v \in \text{sources-aux } I D U xs. (D x, v) \in I)$

by simp

thus ?thesis **by** blast

next

case False

hence ipurge-tr-rev-aux $I D U xs = x \# ws$ **using** B **by** simp

with A **have** $\exists ys zs. xs = ys @ x \# zs \wedge$

$\text{ipurge-tr-rev-aux } I D (\text{sources-aux } I D U (x \# zs)) ys = [] \wedge$

$(\exists v \in \text{sources-aux } I D U \text{ } zs. (D x, v) \in I) ..$
then obtain ys **and** zs **where** $xs: xs = ys @ x \# zs \wedge$
 $\text{ipurge-tr-rev-aux } I D (\text{sources-aux } I D U (x \# zs)) ys = [] \wedge$
 $(\exists v \in \text{sources-aux } I D U \text{ } zs. (D x, v) \in I)$
by *blast*
then have
 $\neg (\exists v \in \text{sources-aux } I D (\text{sources-aux } I D U (x \# zs)) ys. (D x', v) \in I)$
using *False* **by** (*simp add: sources-aux-append*)
hence $\text{ipurge-tr-rev-aux } I D (\text{sources-aux } I D U (x \# zs)) (x' \# ys) =$
 $\text{ipurge-tr-rev-aux } I D (\text{sources-aux } I D U (x \# zs)) ys$
by *simp*
with xs **have** $x' \# xs = (x' \# ys) @ x \# zs \wedge$
 $\text{ipurge-tr-rev-aux } I D (\text{sources-aux } I D U (x \# zs)) (x' \# ys) = [] \wedge$
 $(\exists v \in \text{sources-aux } I D U \text{ } zs. (D x, v) \in I)$
by (*simp del: sources-aux.simps*)
thus *?thesis* **by** *blast*
qed
qed

lemma *ipurge-tr-rev-aux-last-1* [*rule-format*]:
 $\text{ipurge-tr-rev-aux } I D U xs = ws @ [x] \longrightarrow (\exists v \in U. (D x, v) \in I)$
proof (*induction xs rule: rev-induct, simp, rule impI*)
fix $xs x'$
assume
 $A: \text{ipurge-tr-rev-aux } I D U xs = ws @ [x] \longrightarrow (\exists v \in U. (D x, v) \in I)$ **and**
 $B: \text{ipurge-tr-rev-aux } I D U (xs @ [x']) = ws @ [x]$
show $\exists v \in U. (D x, v) \in I$
proof (*cases* $\exists v \in U. (D x', v) \in I$)
case *True*
hence $\text{ipurge-tr-rev-aux } I D U (xs @ [x']) =$
 $\text{ipurge-tr-rev-aux } I D (\text{insert } (D x') U) xs @ [x']$
by (*simp add: ipurge-tr-rev-aux-append*)
hence $x' = x$ **using** B **by** *simp*
thus *?thesis* **using** *True* **by** *simp*
next
case *False*
hence $\text{ipurge-tr-rev-aux } I D U (xs @ [x']) = \text{ipurge-tr-rev-aux } I D U xs$
by (*simp add: ipurge-tr-rev-aux-append*)
hence $\text{ipurge-tr-rev-aux } I D U xs = ws @ [x]$ **using** B **by** *simp*
with A **show** *?thesis* ..
qed
qed

lemma *ipurge-tr-rev-aux-last-2* [*rule-format*]:
 $\text{ipurge-tr-rev-aux } I D U xs = ws @ [x] \longrightarrow$
 $(\exists ys zs. xs = ys @ x \# zs \wedge \text{ipurge-tr-rev-aux } I D U zs = [])$
proof (*induction xs rule: rev-induct, simp, rule impI*)
fix $xs x'$
assume

A: $ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ xs = ws @ [x] \longrightarrow$
 $(\exists\ ys\ zs.\ xs = ys @ x \# zs \wedge ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ zs = [])$ **and**
B: $ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ (xs @ [x']) = ws @ [x]$
show $\exists\ ys\ zs.\ xs @ [x'] = ys @ x \# zs \wedge ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ zs = []$
proof (*cases* $\exists\ v \in U.\ (D\ x', v) \in I$)
case *True*
hence $ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ (xs @ [x']) =$
 $ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ (insert\ (D\ x')\ U)\ xs @ [x']$
by (*simp add: ipurge-tr-rev-aux-append*)
hence $xs @ [x'] = xs @ x \# [] \wedge ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ [] = []$
using *B* **by** *simp*
thus *?thesis* **by** *blast*
next
case *False*
hence $ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ (xs @ [x']) = ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ xs$
by (*simp add: ipurge-tr-rev-aux-append*)
hence $ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ xs = ws @ [x]$ **using** *B* **by** *simp*
with *A* **have** $\exists\ ys\ zs.\ xs = ys @ x \# zs \wedge ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ zs = []$..
then obtain *ys* **and** *zs* **where**
C: $xs = ys @ x \# zs \wedge ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ zs = []$
by *blast*
hence $xs @ [x'] = ys @ x \# zs @ [x']$ **by** *simp*
moreover have
 $ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ (zs @ [x']) = ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ zs$
using *False* **by** (*simp add: ipurge-tr-rev-aux-append*)
hence $ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ (zs @ [x']) = []$ **using** *C* **by** *simp*
ultimately have $xs @ [x'] = ys @ x \# zs @ [x'] \wedge$
 $ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ (zs @ [x']) = []$..
thus *?thesis* **by** *blast*
qed
qed

lemma *ipurge-tr-rev-aux-all* [*rule-format*]:
 $(\forall\ v \in D.\ \text{set}\ xs.\ \exists\ u \in U.\ (v, u) \in I) \longrightarrow ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ xs = xs$
proof (*induction xs, simp, rule impI, simp, erule conjE*)
fix *x xs*
assume $\exists\ u \in U.\ (D\ x, u) \in I$
then obtain *u* **where** *A:* $u \in U$ **and** *B:* $(D\ x, u) \in I$..
have $U \subseteq sources\text{-}aux\ I\ D\ U\ xs$ **by** (*rule sources-aux-subset*)
hence $u \in sources\text{-}aux\ I\ D\ U\ xs$ **using** *A* ..
with *B* **show** $\exists\ u \in sources\text{-}aux\ I\ D\ U\ xs.\ (D\ x, u) \in I$..
qed

Here below, further properties of the functions defined above are investigated thanks to the introduction of function *offset*, which searches a list for a given item and returns the offset of its first occurrence, if any, from the first item of the list.

primrec *offset* :: nat \Rightarrow 'a \Rightarrow 'a list \Rightarrow nat option **where**
offset - - [] = None |
offset n x (y # ys) = (if y = x then Some n else *offset* (Suc n) x ys)

lemma *offset-not-none-1* [rule-format]:

offset k x xs \neq None \longrightarrow (\exists ys zs. xs = ys @ x # zs)

proof (induction xs arbitrary: k, simp, rule impI)

fix w xs k

assume

A: $\bigwedge k$. *offset* k x xs \neq None \longrightarrow (\exists ys zs. xs = ys @ x # zs) **and**

B: *offset* k x (w # xs) \neq None

show \exists ys zs. w # xs = ys @ x # zs

proof (cases w = x, simp)

case True

hence x # xs = [] @ x # xs **by** simp

thus \exists ys zs. x # xs = ys @ x # zs **by** blast

next

case False

hence *offset* k x (w # xs) = *offset* (Suc k) x xs **by** simp

hence *offset* (Suc k) x xs \neq None **using** B **by** simp

moreover have *offset* (Suc k) x xs \neq None \longrightarrow (\exists ys zs. xs = ys @ x # zs)

using A .

ultimately have \exists ys zs. xs = ys @ x # zs **by** simp

then obtain ys and zs **where** xs = ys @ x # zs **by** blast

hence w # xs = (w # ys) @ x # zs **by** simp

thus \exists ys zs. w # xs = ys @ x # zs **by** blast

qed

qed

lemma *offset-not-none-2* [rule-format]:

xs = ys @ x # zs \longrightarrow *offset* k x xs \neq None

proof (induction xs arbitrary: ys k, simp-all del: not-None-eq, rule impI)

fix w xs ys k

assume

A: $\bigwedge ys' k'$. xs = ys' @ x # zs \longrightarrow *offset* k' x (ys' @ x # zs) \neq None **and**

B: w # xs = ys @ x # zs

show *offset* k x (ys @ x # zs) \neq None

proof (cases ys, simp-all del: not-None-eq, rule impI)

fix y' ys'

have xs = ys' @ x # zs \longrightarrow *offset* (Suc k) x (ys' @ x # zs) \neq None

using A .

moreover assume ys = y' # ys'

hence xs = ys' @ x # zs **using** B **by** simp

ultimately show *offset* (Suc k) x (ys' @ x # zs) \neq None ..

qed

qed

lemma *offset-not-none*:

(*offset* k x xs \neq None) = (\exists ys zs. xs = ys @ x # zs)

by (rule iffI, erule offset-not-none-1, (erule exE)+, rule offset-not-none-2)

lemma *offset-addition* [rule-format]:

$\text{offset } k \ x \ xs \neq \text{None} \longrightarrow \text{offset } (n + m) \ x \ xs = \text{Some } (\text{the } (\text{offset } n \ x \ xs) + m)$

proof (induction xs arbitrary: k n, simp, rule impI)

fix w xs k n

assume

A: $\bigwedge k \ n. \text{offset } k \ x \ xs \neq \text{None} \longrightarrow$

$\text{offset } (n + m) \ x \ xs = \text{Some } (\text{the } (\text{offset } n \ x \ xs) + m)$ and

B: $\text{offset } k \ x \ (w \ \# \ xs) \neq \text{None}$

show $\text{offset } (n + m) \ x \ (w \ \# \ xs) = \text{Some } (\text{the } (\text{offset } n \ x \ (w \ \# \ xs)) + m)$

proof (cases w = x, simp-all)

case False

hence $\text{offset } k \ x \ (w \ \# \ xs) = \text{offset } (\text{Suc } k) \ x \ xs$ by simp

hence $\text{offset } (\text{Suc } k) \ x \ xs \neq \text{None}$ using B by simp

moreover have $\text{offset } (\text{Suc } k) \ x \ xs \neq \text{None} \longrightarrow$

$\text{offset } (\text{Suc } n + m) \ x \ xs = \text{Some } (\text{the } (\text{offset } (\text{Suc } n) \ x \ xs) + m)$

using A .

ultimately show $\text{offset } (\text{Suc } (n + m)) \ x \ xs =$

$\text{Some } (\text{the } (\text{offset } (\text{Suc } n) \ x \ xs) + m)$

by simp

qed

qed

lemma *offset-suc*:

assumes A: $\text{offset } k \ x \ xs \neq \text{None}$

shows $\text{offset } (\text{Suc } n) \ x \ xs = \text{Some } (\text{Suc } (\text{the } (\text{offset } n \ x \ xs)))$

proof –

have $\text{offset } (\text{Suc } n) \ x \ xs = \text{offset } (n + \text{Suc } 0) \ x \ xs$ by simp

also have $\dots = \text{Some } (\text{the } (\text{offset } n \ x \ xs) + \text{Suc } 0)$ using A by (rule offset-addition)

also have $\dots = \text{Some } (\text{Suc } (\text{the } (\text{offset } n \ x \ xs)))$ by simp

finally show ?thesis .

qed

lemma *ipurge-tr-rev-aux-first-offset* [rule-format]:

$xs = ys \ @ \ x \ \# \ zs \ \wedge \ \text{ipurge-tr-rev-aux } I \ D \ (\text{sources-aux } I \ D \ U \ (x \ \# \ zs)) \ ys = [] \ \wedge$

$(\exists v \in \text{sources-aux } I \ D \ U \ zs. (D \ x, v) \in I) \longrightarrow$

$ys = \text{take } (\text{the } (\text{offset } 0 \ x \ xs)) \ xs$

proof (induction xs arbitrary: ys, simp, rule impI, (erule conjE)+)

fix x' xs ys

assume

A: $\bigwedge ys. xs = ys \ @ \ x \ \# \ zs \ \wedge$

$\text{ipurge-tr-rev-aux } I \ D \ (\text{sources-aux } I \ D \ U \ (x \ \# \ zs)) \ ys = [] \ \wedge$

$(\exists v \in \text{sources-aux } I \ D \ U \ zs. (D \ x, v) \in I) \longrightarrow$

$ys = \text{take } (\text{the } (\text{offset } 0 \ x \ xs)) \ xs$ and

B: $x' \ \# \ xs = ys \ @ \ x \ \# \ zs$ and

C: $\text{ipurge-tr-rev-aux } I \ D \ (\text{sources-aux } I \ D \ U \ (x \ \# \ zs)) \ ys = []$ and

D: $\exists v \in \text{sources-aux } I \ D \ U \ zs. (D \ x, v) \in I$

show $ys = \text{take } (\text{the } (\text{offset } 0 \ x \ (x' \# \ xs))) \ (x' \# \ xs)$
proof $(\text{cases } ys)$
 case Nil
 then have $x' = x$ **using** B **by** simp
 with Nil **show** $?thesis$ **by** simp
next
 case $(Cons \ y \ ys')$
 hence $E: xs = ys' @ x \# \ zs$ **using** B **by** simp
 moreover have
 $F: \text{ipurge-tr-rev-aux } I \ D \ (\text{sources-aux } I \ D \ U \ (x \# \ zs)) \ (y \# \ ys') = []$
 using $Cons$ **and** C **by** simp
 hence
 $G: \neg (\exists v \in \text{sources-aux } I \ D \ (\text{sources-aux } I \ D \ U \ (x \# \ zs)) \ ys'. \ (D \ y, \ v) \in I)$
 by $(\text{rule-tac } \text{notI}, \ \text{simp})$
 hence $\text{ipurge-tr-rev-aux } I \ D \ (\text{sources-aux } I \ D \ U \ (x \# \ zs)) \ ys' = []$
 using F **by** simp
 ultimately have $xs = ys' @ x \# \ zs \wedge$
 $\text{ipurge-tr-rev-aux } I \ D \ (\text{sources-aux } I \ D \ U \ (x \# \ zs)) \ ys' = [] \wedge$
 $(\exists v \in \text{sources-aux } I \ D \ U \ zs. \ (D \ x, \ v) \in I)$
 using D **by** blast
 with A **have** $H: ys' = \text{take } (\text{the } (\text{offset } 0 \ x \ xs)) \ xs \ ..$
 have $I: x' = y$ **using** $Cons$ **and** B **by** simp
 hence
 $J: \neg (\exists v \in \text{sources-aux } I \ D \ (\text{sources-aux } I \ D \ U \ zs) \ (ys' @ [x]). \ (D \ x', \ v) \in I)$
 using G **by** $(\text{simp } \text{add: } \text{sources-aux-append})$
 have $x' \neq x$
 proof
 assume $x' = x$
 hence $\exists v \in \text{sources-aux } I \ D \ U \ zs. \ (D \ x', \ v) \in I$ **using** D **by** simp
 then obtain v **where** $K: v \in \text{sources-aux } I \ D \ U \ zs$ **and** $L: (D \ x', \ v) \in I \ ..$
 have $\text{sources-aux } I \ D \ U \ zs \subseteq$
 $\text{sources-aux } I \ D \ (\text{sources-aux } I \ D \ U \ zs) \ (ys' @ [x])$
 by $(\text{rule } \text{sources-aux-subset})$
 hence $v \in \text{sources-aux } I \ D \ (\text{sources-aux } I \ D \ U \ zs) \ (ys' @ [x])$ **using** $K \ ..$
 with L **have**
 $\exists v \in \text{sources-aux } I \ D \ (\text{sources-aux } I \ D \ U \ zs) \ (ys' @ [x]). \ (D \ x', \ v) \in I \ ..$
 thus $False$ **using** J **by** contradiction
 qed
 hence $\text{offset } 0 \ x \ (x' \# \ xs) = \text{offset } (\text{Suc } 0) \ x \ xs$ **by** simp
 also have $\dots = \text{Some } (\text{Suc } (\text{the } (\text{offset } 0 \ x \ xs)))$
 proof $-$
 have $\exists ys \ zs. \ xs = ys @ x \# \ zs$ **using** E **by** blast
 hence $\text{offset } 0 \ x \ xs \neq None$ **by** $(\text{simp } \text{only: } \text{offset-not-none})$
 thus $?thesis$ **by** $(\text{rule } \text{offset-suc})$
 qed
 finally have $\text{take } (\text{the } (\text{offset } 0 \ x \ (x' \# \ xs))) \ (x' \# \ xs) =$
 $x' \# \ \text{take } (\text{the } (\text{offset } 0 \ x \ xs)) \ xs$
 by simp
 thus $?thesis$ **using** $Cons$ **and** H **and** I **by** simp

qed
qed

lemma *ipurge-tr-rev-aux-append-nil-2* [rule-format]:

ipurge-tr-rev-aux I D U (xs @ ys) = ipurge-tr-rev-aux I D V xs \longrightarrow
ipurge-tr-rev-aux I D U ys = []

proof (*induction xs, simp, simp only: append-Cons, rule impI*)

fix *x xs*

assume

A: ipurge-tr-rev-aux I D U (xs @ ys) = ipurge-tr-rev-aux I D V xs \longrightarrow
ipurge-tr-rev-aux I D U ys = [] **and**

B: ipurge-tr-rev-aux I D U (x # xs @ ys) = ipurge-tr-rev-aux I D V (x # xs)

show *ipurge-tr-rev-aux I D U ys = []*

proof (*cases $\exists v \in \text{sources-aux I D V xs. (D x, v) \in I$*)

case *True*

hence *C: ipurge-tr-rev-aux I D U (x # xs @ ys) =*
x # ipurge-tr-rev-aux I D V xs

using *B* **by** *simp*

hence $\exists vs ws. x \# xs @ ys = vs @ x \# ws \wedge$
ipurge-tr-rev-aux I D (sources-aux I D U (x # ws)) vs = [] \wedge
($\exists v \in \text{sources-aux I D U ws. (D x, v) \in I$)

by (*rule ipurge-tr-rev-aux-first*)

then obtain *vs and ws where* $*$: $x \# xs @ ys = vs @ x \# ws \wedge$

ipurge-tr-rev-aux I D (sources-aux I D U (x # ws)) vs = [] \wedge
($\exists v \in \text{sources-aux I D U ws. (D x, v) \in I$)

by *blast*

then have *vs = take (the (offset 0 x (x # xs @ ys))) (x # xs @ ys)*

by (*rule ipurge-tr-rev-aux-first-offset*)

hence *vs = []* **by** *simp*

with $*$ **have** $\exists v \in \text{sources-aux I D U (xs @ ys). (D x, v) \in I$ **by** *simp*

hence *ipurge-tr-rev-aux I D U (xs @ ys) = ipurge-tr-rev-aux I D V xs*

using *C* **by** *simp*

with *A* **show** *?thesis ..*

next

case *False*

moreover have $\neg (\exists v \in \text{sources-aux I D U (xs @ ys). (D x, v) \in I)$

proof

assume $\exists v \in \text{sources-aux I D U (xs @ ys). (D x, v) \in I$

hence *ipurge-tr-rev-aux I D V (x # xs) =*

x # ipurge-tr-rev-aux I D U (xs @ ys)

using *B* **by** *simp*

hence $\exists vs ws. x \# xs = vs @ x \# ws \wedge$

ipurge-tr-rev-aux I D (sources-aux I D V (x # ws)) vs = [] \wedge

($\exists v \in \text{sources-aux I D V ws. (D x, v) \in I$)

by (*rule ipurge-tr-rev-aux-first*)

then obtain *vs and ws where* $*$: $x \# xs = vs @ x \# ws \wedge$

ipurge-tr-rev-aux I D (sources-aux I D V (x # ws)) vs = [] \wedge

($\exists v \in \text{sources-aux I D V ws. (D x, v) \in I$)

by *blast*

then have $vs = take\ (the\ (offset\ 0\ x\ (x\ \# \ xs)))\ (x\ \# \ xs)$
by $(rule\ ipurge-tr-rev-aux-first-offset)$
hence $vs = []$ **by** $simp$
with $*$ **have** $\exists v \in sources-aux\ I\ D\ V\ xs.\ (D\ x,\ v) \in I$ **by** $simp$
thus $False$ **using** $False$ **by** $contradiction$
qed
ultimately have $ipurge-tr-rev-aux\ I\ D\ U\ (xs\ @\ ys) =$
 $ipurge-tr-rev-aux\ I\ D\ V\ xs$
using B **by** $simp$
with A **show** $?thesis\ ..$
qed
qed

lemma $ipurge-tr-rev-aux-append-nil$:
 $(ipurge-tr-rev-aux\ I\ D\ U\ (xs\ @\ ys) = ipurge-tr-rev-aux\ I\ D\ U\ xs) =$
 $(ipurge-tr-rev-aux\ I\ D\ U\ ys = [])$
by $(rule\ iffI,\ erule\ ipurge-tr-rev-aux-append-nil-2,\ rule\ ipurge-tr-rev-aux-append-nil-1)$

In what follows, it is proven by induction that the lists output by functions $ipurge-tr$ and $ipurge-tr-rev$, as well as those output by $ipurge-tr-aux$ and $ipurge-tr-rev-aux$, satisfy predicate *Interleaves* (cf. [7]), in correspondence with suitable input predicates expressed in terms of functions $sinks$ and $sinks-aux$, respectively. Then, some lemmas on the aforesaid functions are demonstrated without induction, using previous lemmas along with the properties of predicate *Interleaves*.

lemma $Interleaves-ipurge-tr$:
 $xs \cong \{ipurge-tr-rev\ I\ D\ u\ xs,\ rev\ (ipurge-tr\ (I^{-1})\ D\ u\ (rev\ xs)),$
 $\lambda y\ ys.\ D\ y \in sinks\ (I^{-1})\ D\ u\ (rev\ (y\ \# \ ys))\}$
proof $(induction\ xs,\ simp,\ simp\ only:\ rev.simps)$
fix $x\ xs$
assume $A:\ xs \cong \{ipurge-tr-rev\ I\ D\ u\ xs,\ rev\ (ipurge-tr\ (I^{-1})\ D\ u\ (rev\ xs)),$
 $\lambda y\ ys.\ D\ y \in sinks\ (I^{-1})\ D\ u\ (rev\ ys\ @\ [y])\}$
 $(is\ - \cong \{?ys,\ ?zs,\ ?P\})$
show $x\ \# \ xs \cong$
 $\{ipurge-tr-rev\ I\ D\ u\ (x\ \# \ xs),\ rev\ (ipurge-tr\ (I^{-1})\ D\ u\ (rev\ xs\ @\ [x])),\ ?P\}$
proof $(cases\ ?P\ x\ xs,\ simp-all\ add:\ sources-sinks\ del:\ sinks.simps)$
case $True$
thus $x\ \# \ xs \cong \{x\ \# \ ?ys,\ ?zs,\ ?P\}$ **using** A **by** $(cases\ ?zs,\ simp-all)$
next
case $False$
thus $x\ \# \ xs \cong \{?ys,\ x\ \# \ ?zs,\ ?P\}$ **using** A **by** $(cases\ ?ys,\ simp-all)$
qed
qed

lemma $Interleaves-ipurge-tr-aux$:
 $xs \cong \{ipurge-tr-rev-aux\ I\ D\ U\ xs,\ rev\ (ipurge-tr-aux\ (I^{-1})\ D\ U\ (rev\ xs)),$

$\lambda y ys. \exists v \in \text{sinks-aux } (I^{-1}) D U (\text{rev } ys). (D y, v) \in I$
proof (*induction xs, simp, simp only: rev.simps*)
fix $x xs$
assume $A: xs \cong \{\text{ipurge-tr-rev-aux } I D U xs,$
 $\text{rev } (\text{ipurge-tr-aux } (I^{-1}) D U (\text{rev } xs)),$
 $\lambda y ys. \exists v \in \text{sinks-aux } (I^{-1}) D U (\text{rev } ys). (D y, v) \in I\}$
 $(\text{is } - \cong \{?ys, ?zs, ?P\})$
show $x \# xs \cong$
 $\{\text{ipurge-tr-rev-aux } I D U (x \# xs),$
 $\text{rev } (\text{ipurge-tr-aux } (I^{-1}) D U (\text{rev } xs @ [x])), ?P\}$
proof (*cases ?P x xs, simp-all (no-asm-simp) add: sources-sinks-aux*)
case *True*
thus $x \# xs \cong \{x \# ?ys, ?zs, ?P\}$ **using** A **by** (*cases ?zs, simp-all*)
next
case *False*
thus $x \# xs \cong \{?ys, x \# ?zs, ?P\}$ **using** A **by** (*cases ?ys, simp-all*)
qed
qed

lemma *ipurge-tr-aux-all:*

$(\text{ipurge-tr-aux } I D U xs = xs) = (\forall u \in U. \neg (\exists v \in D \text{ ' set } xs. (u, v) \in I))$

proof –

have $A: \text{rev } xs \cong \{\text{ipurge-tr-rev-aux } (I^{-1}) D U (\text{rev } xs),$
 $\text{rev } (\text{ipurge-tr-aux } ((I^{-1})^{-1}) D U (\text{rev } (\text{rev } xs))),$
 $\lambda y ys. \exists v \in \text{sinks-aux } ((I^{-1})^{-1}) D U (\text{rev } ys). (D y, v) \in (I^{-1})\}$
 $(\text{is } - \cong \{-, -, ?P\})$

by (*rule Interleaves-ipurge-tr-aux*)

show *?thesis*

proof

assume $\text{ipurge-tr-aux } I D U xs = xs$

hence $\text{rev } xs \cong \{\text{ipurge-tr-rev-aux } (I^{-1}) D U (\text{rev } xs), \text{rev } xs, ?P\}$

using A **by** *simp*

hence $\text{rev } xs \simeq \{\text{ipurge-tr-rev-aux } (I^{-1}) D U (\text{rev } xs), \text{rev } xs, ?P\}$

by (*rule Interleaves-interleaves*)

moreover have $\text{rev } xs \simeq \{\[], \text{rev } xs, ?P\}$ **by** (*rule interleaves-nil-all*)

ultimately have $\text{ipurge-tr-rev-aux } (I^{-1}) D U (\text{rev } xs) = \[]$

by (*rule interleaves-equal-fst*)

thus $\forall u \in U. \neg (\exists v \in D \text{ ' set } xs. (u, v) \in I)$

by (*simp add: ipurge-tr-rev-aux-nil*)

next

assume $\forall u \in U. \neg (\exists v \in D \text{ ' set } xs. (u, v) \in I)$

hence $\text{ipurge-tr-rev-aux } (I^{-1}) D U (\text{rev } xs) = \[]$

by (*simp add: ipurge-tr-rev-aux-nil*)

hence $\text{rev } xs \cong \{\[], \text{rev } (\text{ipurge-tr-aux } I D U xs), ?P\}$ **using** A **by** *simp*

hence $\text{rev } xs \simeq \{\[], \text{rev } (\text{ipurge-tr-aux } I D U xs), ?P\}$

by (*rule Interleaves-interleaves*)

hence $\text{rev } xs \simeq \{\text{rev } (\text{ipurge-tr-aux } I D U xs), \[], \lambda w ws. \neg ?P w ws\}$

by (*subst (asm) interleaves-swap*)

moreover have $\text{rev } xs \simeq \{\text{rev } xs, \[], \lambda w ws. \neg ?P w ws\}$

by (rule interleaves-all-nil)
 ultimately have $\text{rev } (\text{ipurge-tr-aux } I D U xs) = \text{rev } xs$
 by (rule interleaves-equal-fst)
 thus $\text{ipurge-tr-aux } I D U xs = xs$ by simp
 qed
 qed

lemma *ipurge-tr-rev-aux-single-dom*:

$\text{ipurge-tr-rev-aux } I D \{u\} xs = \text{ipurge-tr-rev } I D u xs$ (is $?ys = ?ys'$)

proof –

have $xs \cong \{?ys, \text{rev } (\text{ipurge-tr-aux } (I^{-1}) D \{u\} (\text{rev } xs)),$
 $\lambda y ys. \exists v \in \text{sinks-aux } (I^{-1}) D \{u\} (\text{rev } ys). (D y, v) \in I\}$
 by (rule Interleaves-ipurge-tr-aux)
 hence $xs \cong \{?ys, \text{rev } (\text{ipurge-tr } (I^{-1}) D u (\text{rev } xs)),$
 $\lambda y ys. (u, D y) \in I^{-1} \vee (\exists v \in \text{sinks } (I^{-1}) D u (\text{rev } ys). (v, D y) \in I^{-1})\}$
 by (simp add: ipurge-tr-aux-single-dom sinks-aux-single-dom)
 hence $xs \cong \{?ys, \text{rev } (\text{ipurge-tr } (I^{-1}) D u (\text{rev } xs)),$
 $\lambda y ys. D y \in \text{sinks } (I^{-1}) D u (\text{rev } (y \# ys))\}$
 (is $- \cong \{-, ?zs, ?P\}$)
 by (simp only: sinks-interference-eq, simp)
 moreover have $xs \cong \{?ys', ?zs, ?P\}$ by (rule Interleaves-ipurge-tr)
 ultimately show *?thesis* by (rule Interleaves-equal-fst)
 qed

lemma *ipurge-tr-all*:

$(\text{ipurge-tr } I D u xs = xs) = (\neg (\exists v \in D \text{ ' set } xs. (u, v) \in I))$
 by (subst ipurge-tr-aux-single-dom [symmetric], simp add: ipurge-tr-aux-all)

lemma *ipurge-tr-rev-all*:

$\forall v \in D \text{ ' set } xs. (v, u) \in I \implies \text{ipurge-tr-rev } I D u xs = xs$
proof (subst ipurge-tr-rev-aux-single-dom [symmetric], rule ipurge-tr-rev-aux-all)
 qed (simp (no-asm-simp))

1.3 A domain-relation map based on intransitive purge

In what follows, constant *rel-ipurge* is defined as the domain-relation map that associates each domain u to the relation comprised of the pairs of traces whose images under function $\text{ipurge-tr-rev } I D u$ are equal, viz. whose events affecting u are the same.

An auxiliary domain set-relation map, *rel-ipurge-aux*, is also defined by replacing ipurge-tr-rev with ipurge-tr-rev-aux , so as to exploit the distributivity of the latter function over list concatenation. Unsurprisingly, since ipurge-tr-rev-aux degenerates into ipurge-tr-rev for a singleton set of domains, the same happens for *rel-ipurge-aux* and *rel-ipurge*.

Subsequently, some basic properties of domain-relation map *rel-ipurge* are proven, namely that it is a view partition, and is future consistent if and only if it is weakly future consistent. The nontrivial implication, viz. the direct

one, derives from the fact that for each domain u allowed to be affected by any event domain, function $ipurge\text{-}tr\text{-}rev\ I\ D\ u$ matches the identity function, so that two traces are correlated by the image of $rel\text{-}ipurge$ under u just in case they are equal.

definition $rel\text{-}ipurge$::

$'a\ process \Rightarrow ('d \times 'd)\ set \Rightarrow ('a \Rightarrow 'd) \Rightarrow ('a, 'd)\ dom\text{-}rel\text{-}map$ **where**
 $rel\text{-}ipurge\ P\ I\ D\ u \equiv \{(xs, ys). xs \in traces\ P \wedge ys \in traces\ P \wedge$
 $ipurge\text{-}tr\text{-}rev\ I\ D\ u\ xs = ipurge\text{-}tr\text{-}rev\ I\ D\ u\ ys\}$

definition $rel\text{-}ipurge\text{-}aux$::

$'a\ process \Rightarrow ('d \times 'd)\ set \Rightarrow ('a \Rightarrow 'd) \Rightarrow ('a, 'd)\ domset\text{-}rel\text{-}map$ **where**
 $rel\text{-}ipurge\text{-}aux\ P\ I\ D\ U \equiv \{(xs, ys). xs \in traces\ P \wedge ys \in traces\ P \wedge$
 $ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ xs = ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ ys\}$

lemma $rel\text{-}ipurge\text{-}aux\text{-}single\text{-}dom$:

$rel\text{-}ipurge\text{-}aux\ P\ I\ D\ \{u\} = rel\text{-}ipurge\ P\ I\ D\ u$
by ($simp\ add$: $rel\text{-}ipurge\text{-}def\ rel\text{-}ipurge\text{-}aux\text{-}def\ ipurge\text{-}tr\text{-}rev\text{-}aux\text{-}single\text{-}dom$)

lemma $view\text{-}partition\text{-}rel\text{-}ipurge$:

$view\text{-}partition\ P\ D\ (rel\text{-}ipurge\ P\ I\ D)$
proof ($subst\ view\text{-}partition\text{-}def$, $rule\ ballI$, $rule\ equivI$)

fix u
show $rel\text{-}ipurge\ P\ I\ D\ u \subseteq traces\ P \times traces\ P$
by ($rule\ subsetI$) ($simp\ add$: $rel\text{-}ipurge\text{-}def\ split\text{-}paired\text{-}all$)
next
fix u
show $refl\text{-}on\ (traces\ P)\ (rel\text{-}ipurge\ P\ I\ D\ u)$
by ($rule\ refl\text{-}onI$) ($simp\ add$: $rel\text{-}ipurge\text{-}def$)
next
fix u
show $sym\ (rel\text{-}ipurge\ P\ I\ D\ u)$
by ($rule\ symI$, $simp\ add$: $rel\text{-}ipurge\text{-}def$)
next
fix u
show $trans\ (rel\text{-}ipurge\ P\ I\ D\ u)$
by ($rule\ transI$, $simp\ add$: $rel\text{-}ipurge\text{-}def$)
qed

lemma $fc\text{-}equals\text{-}wfc\text{-}rel\text{-}ipurge$:

$future\text{-}consistent\ P\ D\ (rel\text{-}ipurge\ P\ I\ D) =$
 $weakly\text{-}future\text{-}consistent\ P\ I\ D\ (rel\text{-}ipurge\ P\ I\ D)$

proof ($rule\ iffI$, $erule\ fc\text{-}implies\text{-}wfc$,
 $simp\ only$: $future\text{-}consistent\text{-}def\ weakly\text{-}future\text{-}consistent\text{-}def$,
 $rule\ ballI$, ($rule\ allI$) $+$, $rule\ impI$)

fix $u\ xs\ ys$

assume

$A: \forall u \in range\ D \cap (-I) \text{ “ } range\ D. \forall xs\ ys. (xs, ys) \in rel\text{-}ipurge\ P\ I\ D\ u \longrightarrow$

```

    next-dom-events P D u xs = next-dom-events P D u ys ∧
    ref-dom-events P D u xs = ref-dom-events P D u ys and
    B: u ∈ range D and
    C: (xs, ys) ∈ rel-ipurge P I D u
show next-dom-events P D u xs = next-dom-events P D u ys ∧
    ref-dom-events P D u xs = ref-dom-events P D u ys
proof (cases u ∈ range D ∩ (-I) “ range D)
case True
with A have ∀ xs ys. (xs, ys) ∈ rel-ipurge P I D u →
    next-dom-events P D u xs = next-dom-events P D u ys ∧
    ref-dom-events P D u xs = ref-dom-events P D u ys ..
hence (xs, ys) ∈ rel-ipurge P I D u →
    next-dom-events P D u xs = next-dom-events P D u ys ∧
    ref-dom-events P D u xs = ref-dom-events P D u ys
by blast
thus ?thesis using C ..
next
case False
hence D: u ∉ (-I) “ range D using B by simp
have ipurge-tr-rev I D u xs = ipurge-tr-rev I D u ys
using C by (simp add: rel-ipurge-def)
moreover have ∀ zs. ipurge-tr-rev I D u zs = zs
proof (rule allI, rule ipurge-tr-rev-all, rule ballI, erule imageE, rule ccontr)
    fix v x
    assume (v, u) ∉ I
    hence (v, u) ∈ -I by simp
    moreover assume v = D x
    hence v ∈ range D by simp
    ultimately have u ∈ (-I) “ range D ..
    thus False using D by contradiction
qed
ultimately show ?thesis by simp
qed
qed

```

1.4 The Ipurge Unwinding Theorem: proof of condition sufficiency

The Ipurge Unwinding Theorem, formalized in what follows as theorem *ipurge-unwinding*, states that a necessary and sufficient condition for the CSP noninterference security [6] of a process being refusals union closed is that domain-relation map *rel-ipurge* be weakly future consistent. Notwithstanding the equivalence of future consistency and weak future consistency for *rel-ipurge* (cf. above), expressing the theorem in terms of the latter reduces the range of the domains to be considered in order to prove or disprove the security of a process, and then is more convenient.

According to the definition of CSP noninterference security formulated in [6], a process is regarded as being secure just in case the occurrence of an

event e may only affect future events allowed to be affected by e . Identifying security with the weak future consistency of *rel-ipurge* means reversing the view of the problem with respect to the direction of time. In fact, from this view, a process is secure just in case the occurrence of an event e may only be affected by past events allowed to affect e . Therefore, what the Ipurge Unwinding Theorem proves is that ultimately, opposite perspectives with regard to the direction of time give rise to equivalent definitions of the noninterference security of a process.

Here below, it is proven that the condition expressed by the Ipurge Unwinding Theorem is sufficient for security.

lemma *ipurge-tr-rev-ipurge-tr-aux-1* [rule-format]:
 $U \subseteq \text{unaffected-domains } I D (D \text{ ' set } ys) zs \longrightarrow$
 $\text{ipurge-tr-rev-aux } I D U (xs @ ys @ zs) =$
 $\text{ipurge-tr-rev-aux } I D U (xs @ \text{ipurge-tr-aux } I D (D \text{ ' set } ys) zs)$
proof (induction zs arbitrary: U rule: rev-induct, rule-tac [!] impI, simp)
fix U
assume $A: U \subseteq \text{unaffected-domains } I D (D \text{ ' set } ys) []$
have $\forall u \in U. \forall v \in D \text{ ' set } ys. (v, u) \notin I$
proof
fix u
assume $u \in U$
with A **have** $u \in \text{unaffected-domains } I D (D \text{ ' set } ys) []$..
thus $\forall v \in D \text{ ' set } ys. (v, u) \notin I$ **by** (simp add: unaffected-domains-def)
qed
hence $\text{ipurge-tr-rev-aux } I D U ys = []$ **by** (simp add: ipurge-tr-rev-aux-nil)
thus $\text{ipurge-tr-rev-aux } I D U (xs @ ys) = \text{ipurge-tr-rev-aux } I D U xs$
by (simp add: ipurge-tr-rev-aux-append-nil)
next
fix $z zs U$
let $?U' = \text{insert } (D z) U$
assume
 $A: \bigwedge U. U \subseteq \text{unaffected-domains } I D (D \text{ ' set } ys) zs \longrightarrow$
 $\text{ipurge-tr-rev-aux } I D U (xs @ ys @ zs) =$
 $\text{ipurge-tr-rev-aux } I D U (xs @ \text{ipurge-tr-aux } I D (D \text{ ' set } ys) zs)$ **and**
 $B: U \subseteq \text{unaffected-domains } I D (D \text{ ' set } ys) (zs @ [z])$
have $C: U \subseteq \text{unaffected-domains } I D (D \text{ ' set } ys) zs$
proof
fix u
assume $u \in U$
with B **have** $u \in \text{unaffected-domains } I D (D \text{ ' set } ys) (zs @ [z])$..
thus $u \in \text{unaffected-domains } I D (D \text{ ' set } ys) zs$
by (simp add: unaffected-domains-def)
qed
have $D: \text{ipurge-tr-rev-aux } I D U (xs @ ys @ zs) =$
 $\text{ipurge-tr-rev-aux } I D U (xs @ \text{ipurge-tr-aux } I D (D \text{ ' set } ys) zs)$
proof –

have $U \subseteq \text{unaffected-domains } I D (D \text{ ' set } ys) zs \longrightarrow$
 $\text{ipurge-tr-rev-aux } I D U (xs @ ys @ zs) =$
 $\text{ipurge-tr-rev-aux } I D U (xs @ \text{ipurge-tr-aux } I D (D \text{ ' set } ys) zs)$
using A .
thus $?thesis$ **using** C ..

qed
have $E: \neg (\exists v \in \text{sinks-aux } I D (D \text{ ' set } ys) zs. (v, D z) \in I) \longrightarrow$
 $\text{ipurge-tr-rev-aux } I D ?U' (xs @ ys @ zs) =$
 $\text{ipurge-tr-rev-aux } I D ?U' (xs @ \text{ipurge-tr-aux } I D (D \text{ ' set } ys) zs)$
(is $?P \longrightarrow ?Q)$

proof
assume $?P$
have $?U' \subseteq \text{unaffected-domains } I D (D \text{ ' set } ys) zs \longrightarrow$
 $\text{ipurge-tr-rev-aux } I D ?U' (xs @ ys @ zs) =$
 $\text{ipurge-tr-rev-aux } I D ?U' (xs @ \text{ipurge-tr-aux } I D (D \text{ ' set } ys) zs)$
using A .
moreover have $?U' \subseteq \text{unaffected-domains } I D (D \text{ ' set } ys) zs$
by ($\text{simp add: } C, \text{ simp add: unaffected-domains-def } \langle ?P \rangle$ $[simplified]$)
ultimately show $?Q$..

qed
show $\text{ipurge-tr-rev-aux } I D U (xs @ ys @ zs @ [z]) =$
 $\text{ipurge-tr-rev-aux } I D U (xs @ \text{ipurge-tr-aux } I D (D \text{ ' set } ys) (zs @ [z]))$

proof ($\text{cases } \exists v \in \text{sinks-aux } I D (D \text{ ' set } ys) zs. (v, D z) \in I,$
 $\text{simp-all (no-asm-simp)})$
case $True$
have $\neg (\exists u \in U. (D z, u) \in I)$
proof
assume $\exists u \in U. (D z, u) \in I$
then obtain u **where** $F: u \in U$ **and** $G: (D z, u) \in I$..
have $D z \in \text{sinks-aux } I D (D \text{ ' set } ys) (zs @ [z])$ **using** $True$ **by** simp
with G **have** $\exists v \in \text{sinks-aux } I D (D \text{ ' set } ys) (zs @ [z]). (v, u) \in I$..
moreover have $u \in \text{unaffected-domains } I D (D \text{ ' set } ys) (zs @ [z])$
using B **and** F ..
hence $\neg (\exists v \in \text{sinks-aux } I D (D \text{ ' set } ys) (zs @ [z]). (v, u) \in I)$
by ($\text{simp add: unaffected-domains-def}$)
ultimately show $False$ **by** contradiction

qed
hence $\text{ipurge-tr-rev-aux } I D U ((xs @ ys @ zs) @ [z]) =$
 $\text{ipurge-tr-rev-aux } I D U (xs @ ys @ zs)$
by ($\text{subst ipurge-tr-rev-aux-append, simp}$)
also have $\dots = \text{ipurge-tr-rev-aux } I D U$
 $(xs @ \text{ipurge-tr-aux } I D (D \text{ ' set } ys) zs)$
using D .
finally show $\text{ipurge-tr-rev-aux } I D U (xs @ ys @ zs @ [z]) =$
 $\text{ipurge-tr-rev-aux } I D U (xs @ \text{ipurge-tr-aux } I D (D \text{ ' set } ys) zs)$
by simp

next
case $False$
note $F = \text{this}$

show $ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ (xs\ @\ ys\ @\ zs\ @\ [z]) =$
 $ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ (xs\ @\ ipurge\text{-}tr\text{-}aux\ I\ D\ (D\ 'set\ ys)\ zs\ @\ [z])$
proof (*cases* $\exists u \in U. (D\ z, u) \in I$)
case *True*
hence $ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ ((xs\ @\ ys\ @\ zs)\ @\ [z]) =$
 $ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ ?U'\ (xs\ @\ ys\ @\ zs)\ @\ [z]$
by (*subst ipurge-tr-rev-aux-append, simp*)
also have ... =
 $ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ ?U'\ (xs\ @\ ipurge\text{-}tr\text{-}aux\ I\ D\ (D\ 'set\ ys)\ zs)\ @\ [z]$
using *E and F by simp*
also have ... =
 $ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ ((xs\ @\ ipurge\text{-}tr\text{-}aux\ I\ D\ (D\ 'set\ ys)\ zs)\ @\ [z])$
using *True by (subst ipurge-tr-rev-aux-append, simp)*
finally show *?thesis by simp*
next
case *False*
hence $ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ ((xs\ @\ ys\ @\ zs)\ @\ [z]) =$
 $ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ (xs\ @\ ys\ @\ zs)$
by (*subst ipurge-tr-rev-aux-append, simp*)
also have ... =
 $ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ (xs\ @\ ipurge\text{-}tr\text{-}aux\ I\ D\ (D\ 'set\ ys)\ zs)$
using *D .*
also have ... =
 $ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ ((xs\ @\ ipurge\text{-}tr\text{-}aux\ I\ D\ (D\ 'set\ ys)\ zs)\ @\ [z])$
using *False by (subst ipurge-tr-rev-aux-append, simp)*
finally show *?thesis by simp*
qed
qed
qed

lemma *ipurge-tr-rev-ipurge-tr-aux-2 [rule-format]:*
 $U \subseteq unaffected\text{-}domains\ I\ D\ (D\ 'set\ ys)\ zs \longrightarrow$
 $ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ (xs\ @\ zs) =$
 $ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ (xs\ @\ ys\ @\ ipurge\text{-}tr\text{-}aux\ I\ D\ (D\ 'set\ ys)\ zs)$
proof (*induction zs arbitrary: U rule: rev-induct, rule-tac [!] impI, simp*)
fix *U*
assume *A: U ⊆ unaffected-domains I D (D 'set ys) []*
have $\forall u \in U. \forall v \in D\ 'set\ ys. (v, u) \notin I$
proof
fix *u*
assume $u \in U$
with *A have* $u \in unaffected\text{-}domains\ I\ D\ (D\ 'set\ ys)\ []$ **..**
thus $\forall v \in D\ 'set\ ys. (v, u) \notin I$ **by** (*simp add: unaffected-domains-def*)
qed
hence $ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ ys = []$ **by** (*simp add: ipurge-tr-rev-aux-nil*)
hence $ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ (xs\ @\ ys) = ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ xs$
by (*simp add: ipurge-tr-rev-aux-append-nil*)
thus $ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ xs = ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ U\ (xs\ @\ ys)$ **..**
next

```

fix z zs U
let ?U' = insert (D z) U
assume
  A:  $\bigwedge U. U \subseteq \text{unaffected-domains } I D (D \text{ ' set } ys) zs \longrightarrow$ 
    ipurge-tr-rev-aux I D U (xs @ zs) =
    ipurge-tr-rev-aux I D U (xs @ ys @ ipurge-tr-aux I D (D \text{ ' set } ys) zs) and
  B:  $U \subseteq \text{unaffected-domains } I D (D \text{ ' set } ys) (zs @ [z])$ 
have C:  $U \subseteq \text{unaffected-domains } I D (D \text{ ' set } ys) zs$ 
proof
  fix u
  assume  $u \in U$ 
  with B have  $u \in \text{unaffected-domains } I D (D \text{ ' set } ys) (zs @ [z]) ..$ 
  thus  $u \in \text{unaffected-domains } I D (D \text{ ' set } ys) zs$ 
  by (simp add: unaffected-domains-def)
qed
have D: ipurge-tr-rev-aux I D U (xs @ zs) =
  ipurge-tr-rev-aux I D U (xs @ ys @ ipurge-tr-aux I D (D \text{ ' set } ys) zs)
proof -
  have  $U \subseteq \text{unaffected-domains } I D (D \text{ ' set } ys) zs \longrightarrow$ 
    ipurge-tr-rev-aux I D U (xs @ zs) =
    ipurge-tr-rev-aux I D U (xs @ ys @ ipurge-tr-aux I D (D \text{ ' set } ys) zs)
  using A .
  thus ?thesis using C ..
qed
have E:  $\neg (\exists v \in \text{sinks-aux } I D (D \text{ ' set } ys) zs. (v, D z) \in I) \longrightarrow$ 
  ipurge-tr-rev-aux I D ?U' (xs @ zs) =
  ipurge-tr-rev-aux I D ?U' (xs @ ys @ ipurge-tr-aux I D (D \text{ ' set } ys) zs)
  (is ?P  $\longrightarrow$  ?Q)
proof
  assume ?P
  have ?U'  $\subseteq \text{unaffected-domains } I D (D \text{ ' set } ys) zs \longrightarrow$ 
    ipurge-tr-rev-aux I D ?U' (xs @ zs) =
    ipurge-tr-rev-aux I D ?U' (xs @ ys @ ipurge-tr-aux I D (D \text{ ' set } ys) zs)
  using A .
  moreover have ?U'  $\subseteq \text{unaffected-domains } I D (D \text{ ' set } ys) zs$ 
  by (simp add: C, simp add: unaffected-domains-def <?P> [simplified])
  ultimately show ?Q ..
qed
show ipurge-tr-rev-aux I D U (xs @ zs @ [z]) =
  ipurge-tr-rev-aux I D U (xs @ ys @ ipurge-tr-aux I D (D \text{ ' set } ys) (zs @ [z]))
proof (cases  $\exists v \in \text{sinks-aux } I D (D \text{ ' set } ys) zs. (v, D z) \in I,$ 
  simp-all (no-asm-simp))
  case True
  have  $\neg (\exists u \in U. (D z, u) \in I)$ 
  proof
    assume  $\exists u \in U. (D z, u) \in I$ 
    then obtain u where F:  $u \in U$  and G:  $(D z, u) \in I ..$ 
    have  $D z \in \text{sinks-aux } I D (D \text{ ' set } ys) (zs @ [z])$  using True by simp
    with G have  $\exists v \in \text{sinks-aux } I D (D \text{ ' set } ys) (zs @ [z]). (v, u) \in I ..$ 

```

moreover have $u \in \text{unaffected-domains } I D (D \text{ ' set } ys) (zs @ [z])$
using B and F ..
hence $\neg (\exists v \in \text{sinks-aux } I D (D \text{ ' set } ys) (zs @ [z]). (v, u) \in I)$
by (*simp add: unaffected-domains-def*)
ultimately show $False$ **by** *contradiction*
qed
hence $\text{ipurge-tr-rev-aux } I D U ((xs @ zs) @ [z]) =$
 $\text{ipurge-tr-rev-aux } I D U (xs @ zs)$
by (*subst ipurge-tr-rev-aux-append, simp*)
also have
 $\dots = \text{ipurge-tr-rev-aux } I D U (xs @ ys @ \text{ipurge-tr-aux } I D (D \text{ ' set } ys) zs)$
using D .
finally show $\text{ipurge-tr-rev-aux } I D U (xs @ zs @ [z]) =$
 $\text{ipurge-tr-rev-aux } I D U (xs @ ys @ \text{ipurge-tr-aux } I D (D \text{ ' set } ys) zs)$
by *simp*
next
case $False$
note $F = \text{this}$
show $\text{ipurge-tr-rev-aux } I D U (xs @ zs @ [z]) =$
 $\text{ipurge-tr-rev-aux } I D U (xs @ ys @ \text{ipurge-tr-aux } I D (D \text{ ' set } ys) zs @ [z])$
proof (*cases* $\exists u \in U. (D z, u) \in I$)
case $True$
hence $\text{ipurge-tr-rev-aux } I D U ((xs @ zs) @ [z]) =$
 $\text{ipurge-tr-rev-aux } I D ?U' (xs @ zs) @ [z]$
by (*subst ipurge-tr-rev-aux-append, simp*)
also have $\dots =$
 $\text{ipurge-tr-rev-aux } I D ?U'$
 $(xs @ ys @ \text{ipurge-tr-aux } I D (D \text{ ' set } ys) zs) @ [z]$
using E and F **by** *simp*
also have $\dots =$
 $\text{ipurge-tr-rev-aux } I D U$
 $((xs @ ys @ \text{ipurge-tr-aux } I D (D \text{ ' set } ys) zs) @ [z])$
using $True$ **by** (*subst ipurge-tr-rev-aux-append, simp*)
finally show *?thesis* **by** *simp*
next
case $False$
hence $\text{ipurge-tr-rev-aux } I D U ((xs @ zs) @ [z]) =$
 $\text{ipurge-tr-rev-aux } I D U (xs @ zs)$
by (*subst ipurge-tr-rev-aux-append, simp*)
also have $\dots =$
 $\text{ipurge-tr-rev-aux } I D U (xs @ ys @ \text{ipurge-tr-aux } I D (D \text{ ' set } ys) zs)$
using D .
also have $\dots =$
 $\text{ipurge-tr-rev-aux } I D U$
 $((xs @ ys @ \text{ipurge-tr-aux } I D (D \text{ ' set } ys) zs) @ [z])$
using $False$ **by** (*subst ipurge-tr-rev-aux-append, simp*)
finally show *?thesis* **by** *simp*
qed
qed

qed

lemma *ipurge-tr-rev-ipurge-tr-1*:

assumes $A: u \in \text{unaffected-domains } I D \{D y\} zs$

shows $\text{ipurge-tr-rev } I D u (xs @ y \# zs) =$

$\text{ipurge-tr-rev } I D u (xs @ \text{ipurge-tr } I D (D y) zs)$

proof –

have $\text{ipurge-tr-rev } I D u (xs @ y \# zs) =$

$\text{ipurge-tr-rev-aux } I D \{u\} (xs @ [y] @ zs)$

by (*simp add: ipurge-tr-rev-aux-single-dom*)

also have $\dots = \text{ipurge-tr-rev-aux } I D \{u\}$

$(xs @ \text{ipurge-tr-aux } I D (D ' \text{set } [y]) zs)$

by (*rule ipurge-tr-rev-ipurge-tr-aux-1, simp add: A*)

also have $\dots = \text{ipurge-tr-rev } I D u (xs @ \text{ipurge-tr } I D (D y) zs)$

by (*simp add: ipurge-tr-aux-single-dom ipurge-tr-rev-aux-single-dom*)

finally show *?thesis* .

qed

lemma *ipurge-tr-rev-ipurge-tr-2*:

assumes $A: u \in \text{unaffected-domains } I D \{D y\} zs$

shows $\text{ipurge-tr-rev } I D u (xs @ zs) =$

$\text{ipurge-tr-rev } I D u (xs @ y \# \text{ipurge-tr } I D (D y) zs)$

proof –

have $\text{ipurge-tr-rev } I D u (xs @ zs) = \text{ipurge-tr-rev-aux } I D \{u\} (xs @ zs)$

by (*simp add: ipurge-tr-rev-aux-single-dom*)

also have

$\dots = \text{ipurge-tr-rev-aux } I D \{u\} (xs @ [y] @ \text{ipurge-tr-aux } I D (D ' \text{set } [y]) zs)$

by (*rule ipurge-tr-rev-ipurge-tr-aux-2, simp add: A*)

also have $\dots = \text{ipurge-tr-rev } I D u (xs @ y \# \text{ipurge-tr } I D (D y) zs)$

by (*simp add: ipurge-tr-aux-single-dom ipurge-tr-rev-aux-single-dom*)

finally show *?thesis* .

qed

lemma *iu-condition-imply-secure-aux-1*:

assumes

$RUC: \text{ref-union-closed } P$ **and**

$IU: \text{weakly-future-consistent } P I D (\text{rel-ipurge } P I D)$ **and**

$A: (xs @ y \# ys, Y) \in \text{failures } P$ **and**

$B: xs @ \text{ipurge-tr } I D (D y) ys \in \text{traces } P$ **and**

$C: \exists y'. y' \in \text{ipurge-ref } I D (D y) ys Y$

shows $(xs @ \text{ipurge-tr } I D (D y) ys, \text{ipurge-ref } I D (D y) ys Y) \in \text{failures } P$

proof –

let $?A = \text{singleton-set } (\text{ipurge-ref } I D (D y) ys Y)$

have $(\exists X. X \in ?A) \longrightarrow$

$(\forall X \in ?A. (xs @ \text{ipurge-tr } I D (D y) ys, X) \in \text{failures } P) \longrightarrow$

$(xs @ \text{ipurge-tr } I D (D y) ys, \bigcup X \in ?A. X) \in \text{failures } P$

using RUC **by** (*simp add: ref-union-closed-def*)

moreover obtain y' **where** $D: y' \in \text{ipurge-ref } I D (D y) ys Y$ **using** C ..

hence $\exists X. X \in ?A$ **by** (*simp add: singleton-set-some, rule exI*)

ultimately have $(\forall X \in ?A. (xs @ \text{ipurge-tr } I D (D y) ys, X) \in \text{failures } P) \longrightarrow$
 $(xs @ \text{ipurge-tr } I D (D y) ys, \bigcup X \in ?A. X) \in \text{failures } P ..$
moreover have $\forall X \in ?A. (xs @ \text{ipurge-tr } I D (D y) ys, X) \in \text{failures } P$
proof (*rule ballI, simp add: singleton-set-def, erule bexE, simp*)
fix y'
have $\forall u \in \text{range } D \cap (-I) \text{ “ range } D.$
 $\forall xs \ ys. (xs, ys) \in \text{rel-ipurge } P I D u \longrightarrow$
 $\text{ref-dom-events } P D u xs = \text{ref-dom-events } P D u ys$
using IU **by** (*simp add: weakly-future-consistent-def*)
moreover assume $E: y' \in \text{ipurge-ref } I D (D y) ys Y$
hence $(D y, D y') \notin I$ **by** (*simp add: ipurge-ref-def*)
hence $D y' \in \text{range } D \cap (-I) \text{ “ range } D$ **by** (*simp add: Image-iff, rule exI*)
ultimately have $\forall xs \ ys. (xs, ys) \in \text{rel-ipurge } P I D (D y') \longrightarrow$
 $\text{ref-dom-events } P D (D y') xs = \text{ref-dom-events } P D (D y') ys ..$
hence
 $F: (xs @ y \# ys, xs @ \text{ipurge-tr } I D (D y) ys) \in \text{rel-ipurge } P I D (D y') \longrightarrow$
 $\text{ref-dom-events } P D (D y') (xs @ y \# ys) =$
 $\text{ref-dom-events } P D (D y') (xs @ \text{ipurge-tr } I D (D y) ys)$
by *blast*
have $y' \in \{x \in Y. D x \in \text{unaffected-domains } I D \{D y\} ys\}$
using E **by** (*simp add: unaffected-domains-single-dom*)
hence $D y' \in \text{unaffected-domains } I D \{D y\} ys$ **by** *simp*
hence $\text{ipurge-tr-rev } I D (D y') (xs @ y \# ys) =$
 $\text{ipurge-tr-rev } I D (D y') (xs @ \text{ipurge-tr } I D (D y) ys)$
by (*rule ipurge-tr-rev-ipurge-tr-1*)
moreover have $xs @ y \# ys \in \text{traces } P$ **using** A **by** (*rule failures-traces*)
ultimately have
 $(xs @ y \# ys, xs @ \text{ipurge-tr } I D (D y) ys) \in \text{rel-ipurge } P I D (D y')$
using B **by** (*simp add: rel-ipurge-def*)
with F **have** $\text{ref-dom-events } P D (D y') (xs @ y \# ys) =$
 $\text{ref-dom-events } P D (D y') (xs @ \text{ipurge-tr } I D (D y) ys) ..$
moreover have $y' \in \text{ref-dom-events } P D (D y') (xs @ y \# ys)$
proof (*simp add: ref-dom-events-def refusals-def*)
have $\{y'\} \subseteq Y$ **using** E **by** (*simp add: ipurge-ref-def*)
with A **show** $(xs @ y \# ys, \{y'\}) \in \text{failures } P$ **by** (*rule process-rule-3*)
qed
ultimately have $y' \in \text{ref-dom-events } P D (D y')$
 $(xs @ \text{ipurge-tr } I D (D y) ys)$
by *simp*
thus $(xs @ \text{ipurge-tr } I D (D y) ys, \{y'\}) \in \text{failures } P$
by (*simp add: ref-dom-events-def refusals-def*)
qed
ultimately have $(xs @ \text{ipurge-tr } I D (D y) ys, \bigcup X \in ?A. X) \in \text{failures } P ..$
thus *?thesis* **by** (*simp only: singleton-set-union*)
qed

lemma *iu-condition-imply-secure-aux-2:*

assumes

RUC: ref-union-closed **P** **and**

IU: weakly-future-consistent $P I D$ (rel-ipurge $P I D$) **and**
A: $(xs @ zs, Z) \in failures P$ **and**
B: $xs @ y \# ipurge-tr I D (D y) zs \in traces P$ **and**
C: $\exists z'. z' \in ipurge-ref I D (D y) zs Z$
shows $(xs @ y \# ipurge-tr I D (D y) zs, ipurge-ref I D (D y) zs Z) \in failures P$
proof –
let $?A = singleton-set (ipurge-ref I D (D y) zs Z)$
have $(\exists X. X \in ?A) \longrightarrow$
 $(\forall X \in ?A. (xs @ y \# ipurge-tr I D (D y) zs, X) \in failures P) \longrightarrow$
 $(xs @ y \# ipurge-tr I D (D y) zs, \bigcup X \in ?A. X) \in failures P$
using *RUC* **by** (simp add: ref-union-closed-def)
moreover obtain z' **where** $D: z' \in ipurge-ref I D (D y) zs Z$ **using** *C* ..
hence $\exists X. X \in ?A$ **by** (simp add: singleton-set-some, rule exI)
ultimately have
 $(\forall X \in ?A. (xs @ y \# ipurge-tr I D (D y) zs, X) \in failures P) \longrightarrow$
 $(xs @ y \# ipurge-tr I D (D y) zs, \bigcup X \in ?A. X) \in failures P$..
moreover have $\forall X \in ?A. (xs @ y \# ipurge-tr I D (D y) zs, X) \in failures P$
proof (rule ballI, simp add: singleton-set-def, erule bexE, simp)
fix z'
have $\forall u \in range D \cap (-I)$ “range D .
 $\forall xs ys. (xs, ys) \in rel-ipurge P I D u \longrightarrow$
 $ref-dom-events P D u xs = ref-dom-events P D u ys$
using *IU* **by** (simp add: weakly-future-consistent-def)
moreover assume $E: z' \in ipurge-ref I D (D y) zs Z$
hence $(D y, D z') \notin I$ **by** (simp add: ipurge-ref-def)
hence $D z' \in range D \cap (-I)$ “range D **by** (simp add: Image-iff, rule exI)
ultimately have $\forall xs ys. (xs, ys) \in rel-ipurge P I D (D z') \longrightarrow$
 $ref-dom-events P D (D z') xs = ref-dom-events P D (D z') ys$..
hence
 $F: (xs @ zs, xs @ y \# ipurge-tr I D (D y) zs) \in rel-ipurge P I D (D z') \longrightarrow$
 $ref-dom-events P D (D z') (xs @ zs) =$
 $ref-dom-events P D (D z') (xs @ y \# ipurge-tr I D (D y) zs)$
by *blast*
have $z' \in \{x \in Z. D x \in unaffected-domains I D \{D y\} zs\}$
using *E* **by** (simp add: unaffected-domains-single-dom)
hence $D z' \in unaffected-domains I D \{D y\} zs$ **by** *simp*
hence $ipurge-tr-rev I D (D z') (xs @ zs) =$
 $ipurge-tr-rev I D (D z') (xs @ y \# ipurge-tr I D (D y) zs)$
by (rule ipurge-tr-rev-ipurge-tr-2)
moreover have $xs @ zs \in traces P$ **using** *A* **by** (rule failures-traces)
ultimately have
 $(xs @ zs, xs @ y \# ipurge-tr I D (D y) zs) \in rel-ipurge P I D (D z')$
using *B* **by** (simp add: rel-ipurge-def)
with *F* **have** $ref-dom-events P D (D z') (xs @ zs) =$
 $ref-dom-events P D (D z') (xs @ y \# ipurge-tr I D (D y) zs)$..
moreover have $z' \in ref-dom-events P D (D z') (xs @ zs)$
proof (simp add: ref-dom-events-def refusals-def)
have $\{z'\} \subseteq Z$ **using** *E* **by** (simp add: ipurge-ref-def)
with *A* **show** $(xs @ zs, \{z'\}) \in failures P$ **by** (rule process-rule-3)

qed
ultimately have $z' \in \text{ref-dom-events } P \ D \ (D \ z')$
 $(xs \ @ \ y \ \# \ \text{ipurge-tr } I \ D \ (D \ y) \ zs)$
by simp
thus $(xs \ @ \ y \ \# \ \text{ipurge-tr } I \ D \ (D \ y) \ zs, \ \{z'\}) \in \text{failures } P$
by $(\text{simp add: ref-dom-events-def refusals-def})$
qed
ultimately have
 $(xs \ @ \ y \ \# \ \text{ipurge-tr } I \ D \ (D \ y) \ zs, \ \bigcup X \in ?A. \ X) \in \text{failures } P \ ..$
thus $?thesis$ **by** $(\text{simp only: singleton-set-union})$
qed

lemma *iu-condition-imply-secure-1* [rule-format]:
assumes
 $RUC: \text{ref-union-closed } P$ **and**
 $IU: \text{weakly-future-consistent } P \ I \ D \ (\text{rel-ipurge } P \ I \ D)$
shows $(xs \ @ \ y \ \# \ ys, \ Y) \in \text{failures } P \ \longrightarrow$
 $(xs \ @ \ \text{ipurge-tr } I \ D \ (D \ y) \ ys, \ \text{ipurge-ref } I \ D \ (D \ y) \ ys \ Y) \in \text{failures } P$
proof $(\text{induction } ys \ \text{arbitrary: } Y \ \text{rule: rev-induct, rule-tac } [!]) \ \text{impI}$
fix Y
assume $A: (xs \ @ \ [y], \ Y) \in \text{failures } P$
show $(xs \ @ \ \text{ipurge-tr } I \ D \ (D \ y) \ [], \ \text{ipurge-ref } I \ D \ (D \ y) \ [] \ Y) \in \text{failures } P$
proof $(\text{cases } \exists y'. \ y' \in \text{ipurge-ref } I \ D \ (D \ y) \ [] \ Y)$
case True
have $xs \ @ \ [y] \in \text{traces } P$ **using** A **by** $(\text{rule failures-traces})$
hence $xs \in \text{traces } P$ **by** $(\text{rule process-rule-2-traces})$
hence $xs \ @ \ \text{ipurge-tr } I \ D \ (D \ y) \ [] \in \text{traces } P$ **by simp**
with RUC **and** IU **and** A **show** $?thesis$
using $True$ **by** $(\text{rule iu-condition-imply-secure-aux-1})$
next
case False
moreover have $(xs, \ \{\}) \in \text{failures } P$ **using** A **by** $(\text{rule process-rule-2})$
ultimately show $?thesis$ **by simp**
qed

next
fix $y' \ ys \ Y$
assume
 $A: \bigwedge Y'. (xs \ @ \ y \ \# \ ys, \ Y') \in \text{failures } P \ \longrightarrow$
 $(xs \ @ \ \text{ipurge-tr } I \ D \ (D \ y) \ ys, \ \text{ipurge-ref } I \ D \ (D \ y) \ ys \ Y') \in \text{failures } P$ **and**
 $B: (xs \ @ \ y \ \# \ ys \ @ \ [y'], \ Y) \in \text{failures } P$
have $(xs \ @ \ y \ \# \ ys, \ \{\}) \in \text{failures } P \ \longrightarrow$
 $(xs \ @ \ \text{ipurge-tr } I \ D \ (D \ y) \ ys, \ \text{ipurge-ref } I \ D \ (D \ y) \ ys \ \{\}) \in \text{failures } P$
 $(\text{is } - \longrightarrow (-, \ ?Y') \in -)$
using A .
moreover have $((xs \ @ \ y \ \# \ ys) \ @ \ [y'], \ Y) \in \text{failures } P$ **using** B **by simp**
hence $C: (xs \ @ \ y \ \# \ ys, \ \{\}) \in \text{failures } P$ **by** $(\text{rule process-rule-2})$
ultimately have $(xs \ @ \ \text{ipurge-tr } I \ D \ (D \ y) \ ys, \ ?Y') \in \text{failures } P \ ..$
moreover have $\{\} \subseteq ?Y' \ ..$
ultimately have $D: (xs \ @ \ \text{ipurge-tr } I \ D \ (D \ y) \ ys, \ \{\}) \in \text{failures } P$

by (rule process-rule-3)
have $E: xs @ \text{ipurge-tr } I D (D y) (ys @ [y']) \in \text{traces } P$
proof (cases $D y' \in \text{sinks } I D (D y) (ys @ [y'])$)
 case *True*
 hence $(xs @ \text{ipurge-tr } I D (D y) (ys @ [y']), \{\}) \in \text{failures } P$ **using** D **by** *simp*
 thus ?thesis **by** (rule failures-traces)
next
 case *False*
 have $\forall u \in \text{range } D \cap (-I) \text{ “ range } D.$
 $\forall xs \ ys. (xs, ys) \in \text{rel-ipurge } P I D u \longrightarrow$
 $\text{next-dom-events } P D u xs = \text{next-dom-events } P D u ys$
 using *IU* **by** (*simp add: weakly-future-consistent-def*)
 moreover **have** $(D y, D y') \notin I$
 using *False* **by** (*simp add: sinks-interference-eq [symmetric] del: sinks.simps*)
 hence $D y' \in \text{range } D \cap (-I) \text{ “ range } D$ **by** (*simp add: Image-iff, rule exI*)
 ultimately **have** $\forall xs \ ys. (xs, ys) \in \text{rel-ipurge } P I D (D y') \longrightarrow$
 $\text{next-dom-events } P D (D y') xs = \text{next-dom-events } P D (D y') ys \dots$
 hence
 $F: (xs @ y \# ys, xs @ \text{ipurge-tr } I D (D y) ys) \in \text{rel-ipurge } P I D (D y') \longrightarrow$
 $\text{next-dom-events } P D (D y') (xs @ y \# ys) =$
 $\text{next-dom-events } P D (D y') (xs @ \text{ipurge-tr } I D (D y) ys)$
 by *blast*
 have $\forall v \in \text{insert } (D y) (\text{sinks } I D (D y) ys). (v, D y') \notin I$
 using *False* **by** (*simp add: sinks-interference-eq [symmetric] del: sinks.simps*)
 hence $\forall v \in \text{sinks-aux } I D \{D y\} ys. (v, D y') \notin I$
 by (*simp add: sinks-aux-single-dom*)
 hence $D y' \in \text{unaffected-domains } I D \{D y\} ys$
 by (*simp add: unaffected-domains-def*)
 hence $\text{ipurge-tr-rev } I D (D y') (xs @ y \# ys) =$
 $\text{ipurge-tr-rev } I D (D y') (xs @ \text{ipurge-tr } I D (D y) ys)$
 by (rule *ipurge-tr-rev-ipurge-tr-1*)
 moreover **have** $xs @ y \# ys \in \text{traces } P$ **using** C **by** (rule failures-traces)
 moreover **have** $xs @ \text{ipurge-tr } I D (D y) ys \in \text{traces } P$
 using D **by** (rule failures-traces)
 ultimately **have**
 $(xs @ y \# ys, xs @ \text{ipurge-tr } I D (D y) ys) \in \text{rel-ipurge } P I D (D y')$
 by (*simp add: rel-ipurge-def*)
 with F **have** $\text{next-dom-events } P D (D y') (xs @ y \# ys) =$
 $\text{next-dom-events } P D (D y') (xs @ \text{ipurge-tr } I D (D y) ys) \dots$
 moreover **have** $y' \in \text{next-dom-events } P D (D y') (xs @ y \# ys)$
 proof (*simp add: next-dom-events-def next-events-def*)
 qed (rule failures-traces [OF B])
 ultimately **have** $y' \in \text{next-dom-events } P D (D y')$
 $(xs @ \text{ipurge-tr } I D (D y) ys)$
 by *simp*
 hence $xs @ \text{ipurge-tr } I D (D y) ys @ [y'] \in \text{traces } P$
 by (*simp add: next-dom-events-def next-events-def*)
 thus ?thesis **using** *False* **by** *simp*
qed

show $(xs @ \text{ipurge-tr } I D (D y) (ys @ [y']), \text{ipurge-ref } I D (D y) (ys @ [y']) Y) \in \text{failures } P$
proof $(\text{cases } \exists x. x \in \text{ipurge-ref } I D (D y) (ys @ [y']) Y)$
case *True*
with *RUC* **and** *IU* **and** *B* **and** *E* **show** *?thesis* **by** $(\text{rule } \text{iu-condition-imply-secure-aux-1})$
next
case *False*
moreover **have** $(xs @ \text{ipurge-tr } I D (D y) (ys @ [y']), \{\}) \in \text{failures } P$
using *E* **by** $(\text{rule } \text{traces-failures})$
ultimately **show** *?thesis* **by** *simp*
qed
qed

lemma *iu-condition-imply-secure-2* $[\text{rule-format}]$:

assumes
RUC: *ref-union-closed* *P* **and**
IU: *weakly-future-consistent* *P* *I D* $(\text{rel-}\text{ipurge } P \text{ } I D)$ **and**
Y: $xs @ [y] \in \text{traces } P$
shows $(xs @ zs, Z) \in \text{failures } P \longrightarrow$
 $(xs @ y \# \text{ipurge-tr } I D (D y) zs, \text{ipurge-ref } I D (D y) zs Z) \in \text{failures } P$
proof $(\text{induction } zs \text{ arbitrary: } Z \text{ rule: } \text{rev-induct, rule-tac } [!] \text{ impI})$
fix *Z*
assume *A*: $(xs @ [], Z) \in \text{failures } P$
show $(xs @ y \# \text{ipurge-tr } I D (D y) [], \text{ipurge-ref } I D (D y) [] Z) \in \text{failures } P$
proof $(\text{cases } \exists z'. z' \in \text{ipurge-ref } I D (D y) [] Z)$
case *True*
have $xs @ y \# \text{ipurge-tr } I D (D y) [] \in \text{traces } P$ **using** *Y* **by** *simp*
with *RUC* **and** *IU* **and** *A* **show** *?thesis*
using *True* **by** $(\text{rule } \text{iu-condition-imply-secure-aux-2})$
next
case *False*
moreover **have** $(xs @ [y], \{\}) \in \text{failures } P$ **using** *Y* **by** $(\text{rule } \text{traces-failures})$
ultimately **show** *?thesis* **by** *simp*
qed
next
fix *z* *zs* *Z*
assume
A: $\bigwedge Z. (xs @ zs, Z) \in \text{failures } P \longrightarrow$
 $(xs @ y \# \text{ipurge-tr } I D (D y) zs, \text{ipurge-ref } I D (D y) zs Z) \in \text{failures } P$ **and**
B: $(xs @ zs @ [z], Z) \in \text{failures } P$
have $(xs @ zs, \{\}) \in \text{failures } P \longrightarrow$
 $(xs @ y \# \text{ipurge-tr } I D (D y) zs, \text{ipurge-ref } I D (D y) zs \{\}) \in \text{failures } P$
(is $\longrightarrow (-, ?Z') \in -)$
using *A* .
moreover **have** $((xs @ zs) @ [z], Z) \in \text{failures } P$ **using** *B* **by** *simp*
hence *C*: $(xs @ zs, \{\}) \in \text{failures } P$ **by** $(\text{rule } \text{process-rule-2})$
ultimately **have** $(xs @ y \# \text{ipurge-tr } I D (D y) zs, ?Z') \in \text{failures } P$..
moreover **have** $\{\} \subseteq ?Z'$..
ultimately **have** *D*: $(xs @ y \# \text{ipurge-tr } I D (D y) zs, \{\}) \in \text{failures } P$

by (*rule process-rule-3*)
have $E: xs @ y \# \text{ipurge-tr } I D (D y) (zs @ [z]) \in \text{traces } P$
proof (*cases* $D z \in \text{sinks } I D (D y) (zs @ [z])$)
 case *True*
 hence $(xs @ y \# \text{ipurge-tr } I D (D y) (zs @ [z]), \{\}) \in \text{failures } P$
 using D **by** *simp*
 thus *?thesis* **by** (*rule failures-traces*)
next
 case *False*
 have $\forall u \in \text{range } D \cap (-I) \text{ “ range } D.$
 $\forall xs \ ys. (xs, ys) \in \text{rel-ipurge } P I D u \longrightarrow$
 $\text{next-dom-events } P D u xs = \text{next-dom-events } P D u ys$
 using IU **by** (*simp add: weakly-future-consistent-def*)
 moreover **have** $(D y, D z) \notin I$
 using *False* **by** (*simp add: sinks-interference-eq [symmetric] del: sinks.simps*)
 hence $D z \in \text{range } D \cap (-I) \text{ “ range } D$ **by** (*simp add: Image-iff, rule exI*)
 ultimately **have** $\forall xs \ ys. (xs, ys) \in \text{rel-ipurge } P I D (D z) \longrightarrow$
 $\text{next-dom-events } P D (D z) xs = \text{next-dom-events } P D (D z) ys ..$
 hence
 $F: (xs @ zs, xs @ y \# \text{ipurge-tr } I D (D y) zs) \in \text{rel-ipurge } P I D (D z) \longrightarrow$
 $\text{next-dom-events } P D (D z) (xs @ zs) =$
 $\text{next-dom-events } P D (D z) (xs @ y \# \text{ipurge-tr } I D (D y) zs)$
 by *blast*
 have $\forall v \in \text{insert } (D y) (\text{sinks } I D (D y) zs). (v, D z) \notin I$
 using *False* **by** (*simp add: sinks-interference-eq [symmetric] del: sinks.simps*)
 hence $\forall v \in \text{sinks-aux } I D \{D y\} zs. (v, D z) \notin I$
 by (*simp add: sinks-aux-single-dom*)
 hence $D z \in \text{unaffected-domains } I D \{D y\} zs$
 by (*simp add: unaffected-domains-def*)
 hence $\text{ipurge-tr-rev } I D (D z) (xs @ zs) =$
 $\text{ipurge-tr-rev } I D (D z) (xs @ y \# \text{ipurge-tr } I D (D y) zs)$
 by (*rule ipurge-tr-rev-ipurge-tr-2*)
 moreover **have** $xs @ zs \in \text{traces } P$ **using** C **by** (*rule failures-traces*)
 moreover **have** $xs @ y \# \text{ipurge-tr } I D (D y) zs \in \text{traces } P$
 using D **by** (*rule failures-traces*)
 ultimately **have**
 $(xs @ zs, xs @ y \# \text{ipurge-tr } I D (D y) zs) \in \text{rel-ipurge } P I D (D z)$
 by (*simp add: rel-ipurge-def*)
 with F **have** $\text{next-dom-events } P D (D z) (xs @ zs) =$
 $\text{next-dom-events } P D (D z) (xs @ y \# \text{ipurge-tr } I D (D y) zs) ..$
 moreover **have** $z \in \text{next-dom-events } P D (D z) (xs @ zs)$
 proof (*simp add: next-dom-events-def next-events-def*)
 qed (*rule failures-traces [OF B]*)
 ultimately **have** $z \in \text{next-dom-events } P D (D z)$
 $(xs @ y \# \text{ipurge-tr } I D (D y) zs)$
 by *simp*
 hence $xs @ y \# \text{ipurge-tr } I D (D y) zs @ [z] \in \text{traces } P$
 by (*simp add: next-dom-events-def next-events-def*)
 thus *?thesis* **using** *False* **by** *simp*

```

qed
show  $(xs @ y \# \text{ipurge-tr } I D (D y) (zs @ [z]),$ 
   $\text{ipurge-ref } I D (D y) (zs @ [z]) Z$ 
   $\in \text{failures } P$ 
proof  $(\text{cases } \exists x. x \in \text{ipurge-ref } I D (D y) (zs @ [z]) Z)$ 
  case True
  with RUC and IU and B and E show ?thesis by  $(\text{rule } \text{iu-condition-imply-secure-aux-2})$ 
next
  case False
  moreover have  $(xs @ y \# \text{ipurge-tr } I D (D y) (zs @ [z]), \{\}) \in \text{failures } P$ 
  using E by  $(\text{rule } \text{traces-failures})$ 
  ultimately show ?thesis by simp
qed
qed

```

theorem *iu-condition-imply-secure*:

```

assumes
  RUC: ref-union-closed P and
  IU: weakly-future-consistent P I D  $(\text{rel-ipurge } P I D)$ 
shows secure P I D
proof  $(\text{simp add: } \text{secure-def } \text{futures-def}, (\text{rule } \text{allI})+, \text{rule } \text{impI}, \text{erule } \text{conjE})$ 
fix xs y ys Y zs Z
assume
  A:  $(xs @ y \# ys, Y) \in \text{failures } P$  and
  B:  $(xs @ zs, Z) \in \text{failures } P$ 
show  $(xs @ \text{ipurge-tr } I D (D y) ys, \text{ipurge-ref } I D (D y) ys Y) \in \text{failures } P \wedge$ 
   $(xs @ y \# \text{ipurge-tr } I D (D y) zs, \text{ipurge-ref } I D (D y) zs Z) \in \text{failures } P$ 
  (is ?P  $\wedge$  ?Q)
proof
  show ?P using RUC and IU and A by  $(\text{rule } \text{iu-condition-imply-secure-1})$ 
next
  have  $((xs @ [y]) @ ys, Y) \in \text{failures } P$  using A by simp
  hence  $(xs @ [y], \{\}) \in \text{failures } P$  by  $(\text{rule } \text{process-rule-2-failures})$ 
  hence  $xs @ [y] \in \text{traces } P$  by  $(\text{rule } \text{failures-traces})$ 
  with RUC and IU show ?Q using B by  $(\text{rule } \text{iu-condition-imply-secure-2})$ 
qed
qed

```

1.5 The Ipurge Unwinding Theorem: proof of condition necessity

Here below, it is proven that the condition expressed by the Ipurge Unwinding Theorem is necessary for security. Finally, the lemmas concerning condition sufficiency and necessity are gathered in the main theorem.

lemma *secure-implies-failure-consistency-aux* $[\text{rule-format}]$:

```

assumes S: secure P I D
shows  $(xs @ ys @ zs, X) \in \text{failures } P \longrightarrow$ 

```

$ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ (D\ ' (X \cup set\ zs))\ ys = [] \longrightarrow (xs @ zs, X) \in failures\ P$
proof (*induction ys rule: rev-induct, simp-all, (rule impI)+*)
fix $y\ ys$
assume $*$: $ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ (D\ ' (X \cup set\ zs))\ (ys @ [y]) = []$
then have A : $\neg (\exists v \in D\ ' (X \cup set\ zs). (D\ y, v) \in I)$
by (*cases $\exists v \in D\ ' (X \cup set\ zs). (D\ y, v) \in I$,
simp-all add: ipurge-tr-rev-aux-append*)
with $*$ **have** B : $ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ (D\ ' (X \cup set\ zs))\ ys = []$
by (*simp add: ipurge-tr-rev-aux-append*)
assume $(xs @ ys @ y \# zs, X) \in failures\ P$
hence $(y \# zs, X) \in futures\ P\ (xs @ ys)$ **by** (*simp add: futures-def*)
hence ($ipurge\text{-}tr\ I\ D\ (D\ y)\ zs, ipurge\text{-}ref\ I\ D\ (D\ y)\ zs\ X$)
 $\in futures\ P\ (xs @ ys)$
using S **by** (*simp add: secure-def*)
moreover have $ipurge\text{-}tr\ I\ D\ (D\ y)\ zs = zs$ **using** A **by** (*simp add: ipurge-tr-all*)
moreover have $ipurge\text{-}ref\ I\ D\ (D\ y)\ zs\ X = X$ **using** A **by** (*rule ipurge-ref-all*)
ultimately have $(zs, X) \in futures\ P\ (xs @ ys)$ **by** *simp*
hence C : $(xs @ ys @ zs, X) \in failures\ P$ **by** (*simp add: futures-def*)
assume $(xs @ ys @ zs, X) \in failures\ P \longrightarrow$
 $ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ (D\ ' (X \cup set\ zs))\ ys = [] \longrightarrow$
 $(xs @ zs, X) \in failures\ P$
hence $ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ (D\ ' (X \cup set\ zs))\ ys = [] \longrightarrow$
 $(xs @ zs, X) \in failures\ P$
using C **..**
thus $(xs @ zs, X) \in failures\ P$ **using** B **..**
qed

lemma *secure-implies-failure-consistency* [*rule-format*]:

assumes S : *secure* $P\ I\ D$
shows $(xs, ys) \in rel\text{-}ipurge\text{-}aux\ P\ I\ D\ (D\ ' (X \cup set\ zs)) \longrightarrow$
 $(xs @ zs, X) \in failures\ P \longrightarrow (ys @ zs, X) \in failures\ P$
proof (*induction ys arbitrary: xs zs rule: rev-induct,
simp-all add: rel-ipurge-aux-def, (rule-tac [!] impI)+, (erule-tac [!] conjE)+*)
fix $xs\ zs$
assume $(xs @ zs, X) \in failures\ P$
hence $([] @ xs @ zs, X) \in failures\ P$ **by** *simp*
moreover assume $ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ (D\ ' (X \cup set\ zs))\ xs = []$
ultimately have $([] @ zs, X) \in failures\ P$
using S **by** (*rule-tac secure-implies-failure-consistency-aux*)
thus $(zs, X) \in failures\ P$ **by** *simp*

next

fix $y\ ys\ xs\ zs$

assume

A : $\bigwedge xs'\ zs'. xs' \in traces\ P \wedge ys \in traces\ P \wedge$
 $ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ (D\ ' (X \cup set\ zs'))\ xs' =$
 $ipurge\text{-}tr\text{-}rev\text{-}aux\ I\ D\ (D\ ' (X \cup set\ zs'))\ ys \longrightarrow$
 $(xs' @ zs', X) \in failures\ P \longrightarrow (ys @ zs', X) \in failures\ P$ **and**
 B : $(xs @ zs, X) \in failures\ P$ **and**
 C : $xs \in traces\ P$ **and**

D: $ys @ [y] \in \text{traces } P$ **and**
E: $\text{ipurge-tr-rev-aux } I D (D' (X \cup \text{set } zs)) xs =$
 $\text{ipurge-tr-rev-aux } I D (D' (X \cup \text{set } zs)) (ys @ [y])$
show $(ys @ y \# zs, X) \in \text{failures } P$
proof (cases $\exists v \in D' (X \cup \text{set } zs). (D y, v) \in I$)
case *True*
hence **F:** $\text{ipurge-tr-rev-aux } I D (D' (X \cup \text{set } zs)) xs =$
 $\text{ipurge-tr-rev-aux } I D (D' (X \cup \text{set } (y \# zs))) ys @ [y]$
using **E** **by** (*simp add: ipurge-tr-rev-aux-append*)
hence
 $\exists vs ws. xs = vs @ y \# ws \wedge \text{ipurge-tr-rev-aux } I D (D' (X \cup \text{set } zs)) ws = []$
by (*rule ipurge-tr-rev-aux-last-2*)
then obtain **vs** **and** **ws** **where**
G: $xs = vs @ y \# ws \wedge \text{ipurge-tr-rev-aux } I D (D' (X \cup \text{set } zs)) ws = []$
by *blast*
hence $\text{ipurge-tr-rev-aux } I D (D' (X \cup \text{set } zs)) xs =$
 $\text{ipurge-tr-rev-aux } I D (D' (X \cup \text{set } zs)) ((vs @ [y]) @ ws)$
by *simp*
hence $\text{ipurge-tr-rev-aux } I D (D' (X \cup \text{set } zs)) xs =$
 $\text{ipurge-tr-rev-aux } I D (D' (X \cup \text{set } zs)) (vs @ [y])$
using **G** **by** (*simp only: ipurge-tr-rev-aux-append-nil*)
moreover have $\exists v \in D' (X \cup \text{set } zs). (D y, v) \in I$
using **F** **by** (*rule ipurge-tr-rev-aux-last-1*)
ultimately have $\text{ipurge-tr-rev-aux } I D (D' (X \cup \text{set } zs)) xs =$
 $\text{ipurge-tr-rev-aux } I D (D' (X \cup \text{set } (y \# zs))) vs @ [y]$
by (*simp add: ipurge-tr-rev-aux-append*)
hence $\text{ipurge-tr-rev-aux } I D (D' (X \cup \text{set } (y \# zs))) vs =$
 $\text{ipurge-tr-rev-aux } I D (D' (X \cup \text{set } (y \# zs))) ys$
using **F** **by** *simp*
moreover have $vs @ y \# ws \in \text{traces } P$ **using** **C** **and** **G** **by** *simp*
hence $vs \in \text{traces } P$ **by** (*rule process-rule-2-traces*)
moreover have $ys \in \text{traces } P$ **using** **D** **by** (*rule process-rule-2-traces*)
moreover have $vs \in \text{traces } P \wedge ys \in \text{traces } P \wedge$
 $\text{ipurge-tr-rev-aux } I D (D' (X \cup \text{set } (y \# zs))) vs =$
 $\text{ipurge-tr-rev-aux } I D (D' (X \cup \text{set } (y \# zs))) ys \longrightarrow$
 $(vs @ y \# zs, X) \in \text{failures } P \longrightarrow (ys @ y \# zs, X) \in \text{failures } P$
using **A** .
ultimately have **H:** $(vs @ y \# zs, X) \in \text{failures } P \longrightarrow$
 $(ys @ y \# zs, X) \in \text{failures } P$
by *simp*
have $((vs @ [y]) @ ws @ zs, X) \in \text{failures } P$ **using** **B** **and** **G** **by** *simp*
moreover have $\text{ipurge-tr-rev-aux } I D (D' (X \cup \text{set } zs)) ws = []$ **using** **G** ..
ultimately have $((vs @ [y]) @ zs, X) \in \text{failures } P$
using **S** **by** (*rule-tac secure-implies-failure-consistency-aux*)
thus *?thesis* **using** **H** **by** *simp*
next
case *False*
hence $\text{ipurge-tr-rev-aux } I D (D' (X \cup \text{set } zs)) xs =$
 $\text{ipurge-tr-rev-aux } I D (D' (X \cup \text{set } zs)) ys$

using E **by** (*simp add: ipurge-tr-rev-aux-append*)
moreover have $ys \in \text{traces } P$ **using** D **by** (*rule process-rule-2-traces*)
moreover have $xs \in \text{traces } P \wedge ys \in \text{traces } P \wedge$
 $\text{ipurge-tr-rev-aux } I D (D \text{ ' } (X \cup \text{set } zs)) xs =$
 $\text{ipurge-tr-rev-aux } I D (D \text{ ' } (X \cup \text{set } zs)) ys \longrightarrow$
 $(xs @ zs, X) \in \text{failures } P \longrightarrow (ys @ zs, X) \in \text{failures } P$
using A .
ultimately have $(ys @ zs, X) \in \text{failures } P$ **using** B **and** C **by** *simp*
hence $(zs, X) \in \text{futures } P$ **ys by** (*simp add: futures-def*)
moreover have $\exists Y. ([y], Y) \in \text{futures } P$ **ys**
using D **by** (*simp add: traces-def Domain-iff futures-def*)
then obtain Y **where** $([y], Y) \in \text{futures } P$ **ys ..**
ultimately have
 $(y \# \text{ipurge-tr } I D (D y) zs, \text{ipurge-ref } I D (D y) zs X) \in \text{futures } P$ **ys**
using S **by** (*simp add: secure-def*)
moreover have $\text{ipurge-tr } I D (D y) zs = zs$
using *False* **by** (*simp add: ipurge-tr-all*)
moreover have $\text{ipurge-ref } I D (D y) zs X = X$
using *False* **by** (*rule ipurge-ref-all*)
ultimately show *?thesis* **by** (*simp add: futures-def*)
qed
qed

lemma *secure-implies-trace-consistency*:
 $\text{secure } P I D \Longrightarrow (xs, ys) \in \text{rel-ipurge-aux } P I D (D \text{ ' } \text{set } zs) \Longrightarrow$
 $xs @ zs \in \text{traces } P \Longrightarrow ys @ zs \in \text{traces } P$
proof (*simp add: traces-def Domain-iff, rule-tac x = {} in exI,*
rule secure-implies-failure-consistency, simp-all)
qed (*erule exE, erule process-rule-3, simp*)

lemma *secure-implies-next-event-consistency*:
 $\text{secure } P I D \Longrightarrow (xs, ys) \in \text{rel-ipurge } P I D (D x) \Longrightarrow$
 $x \in \text{next-events } P xs \Longrightarrow x \in \text{next-events } P ys$
by (*auto simp add: next-events-def rel-ipurge-aux-single-dom intro: secure-implies-trace-consistency*)

lemma *secure-implies-refusal-consistency*:
 $\text{secure } P I D \Longrightarrow (xs, ys) \in \text{rel-ipurge-aux } P I D (D \text{ ' } X) \Longrightarrow$
 $X \in \text{refusals } P xs \Longrightarrow X \in \text{refusals } P ys$
by (*simp add: refusals-def, subst append-Nil2 [symmetric],*
rule secure-implies-failure-consistency, simp-all)

lemma *secure-implies-ref-event-consistency*:
 $\text{secure } P I D \Longrightarrow (xs, ys) \in \text{rel-ipurge } P I D (D x) \Longrightarrow$
 $\{x\} \in \text{refusals } P xs \Longrightarrow \{x\} \in \text{refusals } P ys$
by (*rule secure-implies-refusal-consistency, simp-all add: rel-ipurge-aux-single-dom*)

theorem *secure-implies-iu-condition*:
assumes S : $\text{secure } P I D$
shows $\text{future-consistent } P D$ ($\text{rel-ipurge } P I D$)

```

proof (simp add: future-consistent-def next-dom-events-def ref-dom-events-def,
(rule allI)+, rule impI, rule conjI, rule-tac [!] equalityI, rule-tac [!] subsetI,
simp-all, erule-tac [!] conjE)
  fix xs ys x
  assume (xs, ys) ∈ rel-ipurge P I D (D x) and x ∈ next-events P xs
  with S show x ∈ next-events P ys by (rule secure-implies-next-event-consistency)
next
  fix xs ys x
  have ∀ u ∈ range D. equiv (traces P) (rel-ipurge P I D u)
    using view-partition-rel-ipurge by (simp add: view-partition-def)
  hence sym (rel-ipurge P I D (D x)) by (simp add: equiv-def)
  moreover assume (xs, ys) ∈ rel-ipurge P I D (D x)
  ultimately have (ys, xs) ∈ rel-ipurge P I D (D x) by (rule symE)
  moreover assume x ∈ next-events P ys
  ultimately show x ∈ next-events P xs
    using S by (rule-tac secure-implies-next-event-consistency)
next
  fix xs ys x
  assume (xs, ys) ∈ rel-ipurge P I D (D x) and {x} ∈ refusals P xs
  with S show {x} ∈ refusals P ys by (rule secure-implies-ref-event-consistency)
next
  fix xs ys x
  have ∀ u ∈ range D. equiv (traces P) (rel-ipurge P I D u)
    using view-partition-rel-ipurge by (simp add: view-partition-def)
  hence sym (rel-ipurge P I D (D x)) by (simp add: equiv-def)
  moreover assume (xs, ys) ∈ rel-ipurge P I D (D x)
  ultimately have (ys, xs) ∈ rel-ipurge P I D (D x) by (rule symE)
  moreover assume {x} ∈ refusals P ys
  ultimately show {x} ∈ refusals P xs
    using S by (rule-tac secure-implies-ref-event-consistency)
qed

theorem ipurge-unwinding:
  ref-union-closed P ⇒
  secure P I D = weakly-future-consistent P I D (rel-ipurge P I D)
proof (rule iffI, subst fc-equals-wfc-rel-ipurge [symmetric])
qed (erule secure-implies-iu-condition, rule iu-condition-implies-secure)

end

```

2 The Ipurge Unwinding Theorem for deterministic and trace set processes

```

theory DeterministicProcesses
imports IpurgeUnwinding
begin

```

In accordance with Hoare’s formal definition of deterministic processes [1], this section shows that a process is deterministic just in case it is a *trace set process*, i.e. it may be identified by means of a trace set alone, matching the set of its traces, in place of a failures-divergences pair. Then, variants of the Ipurge Unwinding Theorem are proven for deterministic processes and trace set processes.

2.1 Deterministic processes

Here below are the definitions of predicates *d-future-consistent* and *d-weakly-future-consistent*, which are variants of predicates *future-consistent* and *weakly-future-consistent* meant for applying to deterministic processes. In some detail, being deterministic processes such that refused events are completely specified by accepted events (cf. [1], [6]), the new predicates are such that their truth values can be determined by just considering the accepted events of the process taken as input.

Then, it is proven that these predicates are characterized by the same connection as that of their general-purpose counterparts, viz. *d-future-consistent* implies *d-weakly-future-consistent*, and they are equivalent for domain-relation map *rel-ipurge*. Finally, the predicates are shown to be equivalent to their general-purpose counterparts in the case of a deterministic process.

definition *d-future-consistent* ::

'a process \Rightarrow (*'a* \Rightarrow *'d*) \Rightarrow (*'a*, *'d*) *dom-rel-map* \Rightarrow *bool* **where**
d-future-consistent *P D R* \equiv
 $\forall u \in \text{range } D. \forall xs \ ys. (xs, ys) \in R \ u \longrightarrow$
 $(xs \in \text{traces } P) = (ys \in \text{traces } P) \wedge$
 $\text{next-dom-events } P \ D \ u \ xs = \text{next-dom-events } P \ D \ u \ ys$

definition *d-weakly-future-consistent* ::

'a process \Rightarrow (*'d* \times *'d*) *set* \Rightarrow (*'a* \Rightarrow *'d*) \Rightarrow (*'a*, *'d*) *dom-rel-map* \Rightarrow *bool* **where**
d-weakly-future-consistent *P I D R* \equiv
 $\forall u \in \text{range } D \cap (-I) \text{ “ range } D. \forall xs \ ys. (xs, ys) \in R \ u \longrightarrow$
 $(xs \in \text{traces } P) = (ys \in \text{traces } P) \wedge$
 $\text{next-dom-events } P \ D \ u \ xs = \text{next-dom-events } P \ D \ u \ ys$

lemma *dfc-implies-dwfc*:

d-future-consistent *P D R* \implies *d-weakly-future-consistent* *P I D R*
by (*simp only*: *d-future-consistent-def* *d-weakly-future-consistent-def*, *blast*)

lemma *dfc-equals-dwfc-rel-ipurge*:

d-future-consistent *P D* (*rel-ipurge* *P I D*) =
d-weakly-future-consistent *P I D* (*rel-ipurge* *P I D*)

proof (*rule iffI*, *erule dfc-implies-dwfc*,

simp only: *d-future-consistent-def* *d-weakly-future-consistent-def*,
rule ballI, (*rule allI*) $+$, *rule impI*)

fix $u\ xs\ ys$
assume
 $A: \forall u \in \text{range } D \cap (-I) \text{ “ range } D. \forall xs\ ys. (xs, ys) \in \text{rel-ipurge } P\ I\ D\ u \longrightarrow$
 $(xs \in \text{traces } P) = (ys \in \text{traces } P) \wedge$
 $\text{next-dom-events } P\ D\ u\ xs = \text{next-dom-events } P\ D\ u\ ys$ **and**
 $B: u \in \text{range } D$ **and**
 $C: (xs, ys) \in \text{rel-ipurge } P\ I\ D\ u$
show $(xs \in \text{traces } P) = (ys \in \text{traces } P) \wedge$
 $\text{next-dom-events } P\ D\ u\ xs = \text{next-dom-events } P\ D\ u\ ys$
proof (*cases* $u \in \text{range } D \cap (-I) \text{ “ range } D$)
case *True*
with A **have** $\forall xs\ ys. (xs, ys) \in \text{rel-ipurge } P\ I\ D\ u \longrightarrow$
 $(xs \in \text{traces } P) = (ys \in \text{traces } P) \wedge$
 $\text{next-dom-events } P\ D\ u\ xs = \text{next-dom-events } P\ D\ u\ ys$ **..**
hence $(xs, ys) \in \text{rel-ipurge } P\ I\ D\ u \longrightarrow$
 $(xs \in \text{traces } P) = (ys \in \text{traces } P) \wedge$
 $\text{next-dom-events } P\ D\ u\ xs = \text{next-dom-events } P\ D\ u\ ys$
by *blast*
thus *?thesis using C ..*
next
case *False*
hence $D: u \notin (-I) \text{ “ range } D$ **using** B **by** *simp*
have $\text{ipurge-tr-rev } I\ D\ u\ xs = \text{ipurge-tr-rev } I\ D\ u\ ys$
using C **by** (*simp add: rel-ipurge-def*)
moreover have $\forall zs. \text{ipurge-tr-rev } I\ D\ u\ zs = zs$
proof (*rule allI, rule ipurge-tr-rev-all, rule ballI, erule imageE, rule ccontr*)
fix $v\ x$
assume $(v, u) \notin I$
hence $(v, u) \in -I$ **by** *simp*
moreover assume $v = D\ x$
hence $v \in \text{range } D$ **by** *simp*
ultimately have $u \in (-I) \text{ “ range } D$ **..**
thus *False using D by contradiction*
qed
ultimately show *?thesis by simp*
qed
qed

lemma *d-fc-equals-dfc:*

assumes $A: \text{deterministic } P$
shows $\text{future-consistent } P\ D\ R = \text{d-future-consistent } P\ D\ R$
proof (*rule iffI, simp-all only: d-future-consistent-def,*
rule ballI, (rule allI)+, rule impI, rule conjI, rule fc-traces, assumption+,
simp-all add: future-consistent-def del: ball-simps)
assume $B: \forall u \in \text{range } D. \forall xs\ ys. (xs, ys) \in R\ u \longrightarrow$
 $(xs \in \text{traces } P) = (ys \in \text{traces } P) \wedge$
 $\text{next-dom-events } P\ D\ u\ xs = \text{next-dom-events } P\ D\ u\ ys$
show $\forall u \in \text{range } D. \forall xs\ ys. (xs, ys) \in R\ u \longrightarrow$
 $\text{ref-dom-events } P\ D\ u\ xs = \text{ref-dom-events } P\ D\ u\ ys$

proof (*rule ballI*, (*rule allI*)₊, *rule impI*,
simp add: ref-dom-events-def set-eq-iff, *rule allI*)
fix $u\ xs\ ys\ x$
assume $u \in \text{range } D$
with B **have** $\forall xs\ ys. (xs, ys) \in R\ u \longrightarrow$
 $(xs \in \text{traces } P) = (ys \in \text{traces } P) \wedge$
 $\text{next-dom-events } P\ D\ u\ xs = \text{next-dom-events } P\ D\ u\ ys \ ..$
hence $(xs, ys) \in R\ u \longrightarrow$
 $(xs \in \text{traces } P) = (ys \in \text{traces } P) \wedge$
 $\text{next-dom-events } P\ D\ u\ xs = \text{next-dom-events } P\ D\ u\ ys$
by *blast*
moreover assume $(xs, ys) \in R\ u$
ultimately have $C: (xs \in \text{traces } P) = (ys \in \text{traces } P) \wedge$
 $\text{next-dom-events } P\ D\ u\ xs = \text{next-dom-events } P\ D\ u\ ys \ ..$
show $(u = D\ x \wedge \{x\} \in \text{refusals } P\ xs) = (u = D\ x \wedge \{x\} \in \text{refusals } P\ ys)$
proof (*cases* $u = D\ x$, *simp-all*, *cases* $xs \in \text{traces } P$)
assume $D: u = D\ x$ **and** $E: xs \in \text{traces } P$
have
 $A': \forall xs \in \text{traces } P. \forall X. X \in \text{refusals } P\ xs = (X \cap \text{next-events } P\ xs = \{\})$
using A **by** (*simp add: deterministic-def*)
hence $\forall X. X \in \text{refusals } P\ xs = (X \cap \text{next-events } P\ xs = \{\})$ **using** $E \ ..$
hence $\{x\} \in \text{refusals } P\ xs = (\{x\} \cap \text{next-events } P\ xs = \{\}) \ ..$
moreover have $ys \in \text{traces } P$ **using** C **and** E **by** *simp*
with A' **have** $\forall X. X \in \text{refusals } P\ ys = (X \cap \text{next-events } P\ ys = \{\}) \ ..$
hence $\{x\} \in \text{refusals } P\ ys = (\{x\} \cap \text{next-events } P\ ys = \{\}) \ ..$
moreover have $\{x\} \cap \text{next-events } P\ xs = \{x\} \cap \text{next-events } P\ ys$
proof (*simp add: set-eq-iff*, *rule allI*, *rule iffI*, *erule-tac* [!], *conjE*, *simp-all*)
assume $x \in \text{next-events } P\ xs$
hence $x \in \text{next-dom-events } P\ D\ u\ xs$ **using** D **by** (*simp add: next-dom-events-def*)
hence $x \in \text{next-dom-events } P\ D\ u\ ys$ **using** C **by** *simp*
thus $x \in \text{next-events } P\ ys$ **by** (*simp add: next-dom-events-def*)
next
assume $x \in \text{next-events } P\ ys$
hence $x \in \text{next-dom-events } P\ D\ u\ ys$ **using** D **by** (*simp add: next-dom-events-def*)
hence $x \in \text{next-dom-events } P\ D\ u\ xs$ **using** C **by** *simp*
thus $x \in \text{next-events } P\ xs$ **by** (*simp add: next-dom-events-def*)
qed
ultimately show $(\{x\} \in \text{refusals } P\ xs) = (\{x\} \in \text{refusals } P\ ys)$ **by** *simp*
next
assume $D: xs \notin \text{traces } P$
hence $\forall X. (xs, X) \notin \text{failures } P$ **by** (*simp add: traces-def Domain-iff*)
hence $\text{refusals } P\ xs = \{\}$ **by** (*rule-tac equals0I*, *simp add: refusals-def*)
moreover have $ys \notin \text{traces } P$ **using** C **and** D **by** *simp*
hence $\forall X. (ys, X) \notin \text{failures } P$ **by** (*simp add: traces-def Domain-iff*)
hence $\text{refusals } P\ ys = \{\}$ **by** (*rule-tac equals0I*, *simp add: refusals-def*)
ultimately show $(\{x\} \in \text{refusals } P\ xs) = (\{x\} \in \text{refusals } P\ ys)$ **by** *simp*
qed
qed
qed

lemma *d-wfc-equals-dwfc*:

assumes *A*: *deterministic P*

shows *weakly-future-consistent P I D R = d-weakly-future-consistent P I D R*

proof (*rule iffI, simp-all only: d-weakly-future-consistent-def, rule ballI, (rule allI)+, rule impI, rule conjI, rule wfc-traces, assumption+, simp-all add: weakly-future-consistent-def del: ball-simps*)

assume *B*: $\forall u \in \text{range } D \cap (- I) \text{ “ range } D. \forall xs\ ys. (xs, ys) \in R\ u \longrightarrow (xs \in \text{traces } P) = (ys \in \text{traces } P) \wedge \text{next-dom-events } P\ D\ u\ xs = \text{next-dom-events } P\ D\ u\ ys$

show $\forall u \in \text{range } D \cap (- I) \text{ “ range } D. \forall xs\ ys. (xs, ys) \in R\ u \longrightarrow \text{ref-dom-events } P\ D\ u\ xs = \text{ref-dom-events } P\ D\ u\ ys$

proof (*rule ballI, (rule allI)+, rule impI, simp (no-asm-simp) add: ref-dom-events-def set-eq-iff, rule allI*)

fix *u xs ys x*

assume $u \in \text{range } D \cap (- I) \text{ “ range } D$

with *B* **have** $\forall xs\ ys. (xs, ys) \in R\ u \longrightarrow (xs \in \text{traces } P) = (ys \in \text{traces } P) \wedge \text{next-dom-events } P\ D\ u\ xs = \text{next-dom-events } P\ D\ u\ ys \dots$

hence $(xs, ys) \in R\ u \longrightarrow (xs \in \text{traces } P) = (ys \in \text{traces } P) \wedge \text{next-dom-events } P\ D\ u\ xs = \text{next-dom-events } P\ D\ u\ ys$

by *blast*

moreover **assume** $(xs, ys) \in R\ u$

ultimately **have** *C*: $(xs \in \text{traces } P) = (ys \in \text{traces } P) \wedge \text{next-dom-events } P\ D\ u\ xs = \text{next-dom-events } P\ D\ u\ ys \dots$

show $(u = D\ x \wedge \{x\} \in \text{refusals } P\ xs) = (u = D\ x \wedge \{x\} \in \text{refusals } P\ ys)$

proof (*cases u = D x, simp-all, cases xs \in traces P*)

assume *D*: $u = D\ x$ **and** *E*: $xs \in \text{traces } P$

have *A'*: $\forall xs \in \text{traces } P. \forall X. X \in \text{refusals } P\ xs = (X \cap \text{next-events } P\ xs = \{\})$

using *A* **by** (*simp add: deterministic-def*)

hence $\forall X. X \in \text{refusals } P\ xs = (X \cap \text{next-events } P\ xs = \{\})$ **using** *E* **..**

hence $\{x\} \in \text{refusals } P\ xs = (\{x\} \cap \text{next-events } P\ xs = \{\}) \dots$

moreover **have** $ys \in \text{traces } P$ **using** *C* **and** *E* **by** *simp*

with *A'* **have** $\forall X. X \in \text{refusals } P\ ys = (X \cap \text{next-events } P\ ys = \{\}) \dots$

hence $\{x\} \in \text{refusals } P\ ys = (\{x\} \cap \text{next-events } P\ ys = \{\}) \dots$

moreover **have** $\{x\} \cap \text{next-events } P\ xs = \{x\} \cap \text{next-events } P\ ys$

proof (*simp add: set-eq-iff, rule allI, rule iffI, erule-tac [!] conjE, simp-all*)

assume $x \in \text{next-events } P\ xs$

hence $x \in \text{next-dom-events } P\ D\ u\ xs$ **using** *D* **by** (*simp add: next-dom-events-def*)

hence $x \in \text{next-dom-events } P\ D\ u\ ys$ **using** *C* **by** *simp*

thus $x \in \text{next-events } P\ ys$ **by** (*simp add: next-dom-events-def*)

next

assume $x \in \text{next-events } P\ ys$

hence $x \in \text{next-dom-events } P\ D\ u\ ys$ **using** *D* **by** (*simp add: next-dom-events-def*)

hence $x \in \text{next-dom-events } P\ D\ u\ xs$ **using** *C* **by** *simp*

thus $x \in \text{next-events } P\ xs$ **by** (*simp add: next-dom-events-def*)

qed

ultimately show $(\{x\} \in \text{refusals } P \text{ } xs) = (\{x\} \in \text{refusals } P \text{ } ys)$ **by simp**
next
assume $D: xs \notin \text{traces } P$
hence $\forall X. (xs, X) \notin \text{failures } P$ **by** (*simp add: traces-def Domain-iff*)
hence $\text{refusals } P \text{ } xs = \{\}$ **by** (*rule-tac equals0I, simp add: refusals-def*)
moreover have $ys \notin \text{traces } P$ **using** C **and** D **by simp**
hence $\forall X. (ys, X) \notin \text{failures } P$ **by** (*simp add: traces-def Domain-iff*)
hence $\text{refusals } P \text{ } ys = \{\}$ **by** (*rule-tac equals0I, simp add: refusals-def*)
ultimately show $(\{x\} \in \text{refusals } P \text{ } xs) = (\{x\} \in \text{refusals } P \text{ } ys)$ **by simp**
qed
qed
qed

Here below is the proof of a variant of the Ipurge Unwinding Theorem applying to deterministic processes. Unsurprisingly, its enunciation contains predicate *d-weakly-future-consistent* in place of *weakly-future-consistent*. Furthermore, the assumption that the process be refusals union closed is replaced by the assumption that it be deterministic, since the former property is shown to be entailed by the latter.

lemma *d-implies-ruc*:

assumes $A: \text{deterministic } P$
shows *ref-union-closed* P

proof (*subst ref-union-closed-def, (rule allI)+, (rule impI)+, erule exE*)

fix $xs \ A \ X$

have $\forall xs \in \text{traces } P. \forall X. X \in \text{refusals } P \text{ } xs = (X \cap \text{next-events } P \text{ } xs = \{\})$

using A **by** (*simp add: deterministic-def*)

moreover assume $B: \forall X \in A. (xs, X) \in \text{failures } P$ **and** $X \in A$

hence $(xs, X) \in \text{failures } P$ **..**

hence $xs \in \text{traces } P$ **by** (*rule failures-traces*)

ultimately have $C: \forall X. X \in \text{refusals } P \text{ } xs = (X \cap \text{next-events } P \text{ } xs = \{\})$ **..**

have $D: \forall X \in A. X \cap \text{next-events } P \text{ } xs = \{\}$

proof

fix X

assume $X \in A$

with B **have** $(xs, X) \in \text{failures } P$ **..**

hence $X \in \text{refusals } P \text{ } xs$ **by** (*simp add: refusals-def*)

thus $X \cap \text{next-events } P \text{ } xs = \{\}$ **using** C **by simp**

qed

have $(\bigcup X \in A. X) \in \text{refusals } P \text{ } xs = ((\bigcup X \in A. X) \cap \text{next-events } P \text{ } xs = \{\})$

using C **..**

hence $E: (xs, \bigcup X \in A. X) \in \text{failures } P =$

$((\bigcup X \in A. X) \cap \text{next-events } P \text{ } xs = \{\})$

by (*simp add: refusals-def*)

show $(xs, \bigcup X \in A. X) \in \text{failures } P$

proof (*rule ssubst [OF E], rule equals0I, erule IntE, erule UN-E*)

fix $x \ X$

assume $X \in A$
with D **have** $X \cap \text{next-events } P \text{ xs} = \{\}$..
moreover assume $x \in X$ **and** $x \in \text{next-events } P \text{ xs}$
hence $x \in X \cap \text{next-events } P \text{ xs}$..
hence $\exists x. x \in X \cap \text{next-events } P \text{ xs}$..
hence $X \cap \text{next-events } P \text{ xs} \neq \{\}$ **by** (*subst ex-in-conv [symmetric]*)
ultimately show *False* **by contradiction**
qed
qed

theorem *d-ipurge-unwinding*:
assumes A : *deterministic P*
shows *secure P I D = d-weakly-future-consistent P I D (rel-ipurge P I D)*
proof (*insert d-wfc-equals-dwfc [of P I D rel-ipurge P I D, OF A], erule subst*)
qed (*insert d-implies-ruc [OF A], rule ipurge-unwinding*)

2.2 Trace set processes

In [1], section 2.8, Hoare formulates a simplified definition of a deterministic process, identified with a *trace set*, i.e. a set of event lists containing the empty list and any prefix of each of its elements. Of course, this is consistent with the definition of determinism applying to processes identified with failures-divergences pairs, which implies that their refusals are completely specified by their traces (cf. [1], [6]).

Here below are the definitions of a function *ts-process*, converting the input set of lists into a process, and a predicate *trace-set*, returning *True* just in case the input set of lists has the aforesaid properties. An analysis is then conducted about the output of the functions defined in [6], section 1.1, when acting on a *trace set process*, i.e. a process that may be expressed as *ts-process T* where *trace-set T* matches *True*.

definition *ts-process* :: 'a list set \Rightarrow 'a process **where**
ts-process T \equiv *Abs-process* ($\{(xs, X). xs \in T \wedge (\forall x \in X. xs @ [x] \notin T)\}, \{\}$)

definition *trace-set* :: 'a list set \Rightarrow bool **where**
trace-set T \equiv $\[] \in T \wedge (\forall xs x. xs @ [x] \in T \longrightarrow xs \in T)$

lemma *ts-process-rep*:
assumes A : *trace-set T*
shows *Rep-process (ts-process T) =*
 $\{(xs, X). xs \in T \wedge (\forall x \in X. xs @ [x] \notin T)\}, \{\}$
proof (*subst ts-process-def, rule Abs-process-inverse, simp add: process-set-def,*
(subst conj-assoc [symmetric])+, (rule conjI)+, simp-all add:
process-prop-1-def
process-prop-2-def
process-prop-3-def
process-prop-4-def

process-prop-5-def
process-prop-6-def
show $\square \in T$ **using** A **by** (*simp add: trace-set-def*)
next
show $\forall xs. (\exists x. xs @ [x] \in T \wedge (\exists X. \forall x' \in X. xs @ [x, x'] \notin T)) \longrightarrow xs \in T$
proof (*rule allI, rule impI, erule exE, erule conjE*)
fix $xs\ x$
have $\forall xs\ x. xs @ [x] \in T \longrightarrow xs \in T$ **using** A **by** (*simp add: trace-set-def*)
hence $xs @ [x] \in T \longrightarrow xs \in T$ **by** *blast*
moreover assume $xs @ [x] \in T$
ultimately show $xs \in T$..
qed
next
show $\forall xs\ X. xs \in T \wedge (\exists Y. (\forall x \in Y. xs @ [x] \notin T) \wedge X \subseteq Y) \longrightarrow$
 $(\forall x \in X. xs @ [x] \notin T)$
proof (*((rule allI)+, rule impI, (erule conjE, (erule exE)?)+, rule ballI)*)
fix $xs\ x\ X\ Y$
assume $\forall x \in Y. xs @ [x] \notin T$
moreover assume $X \subseteq Y$ **and** $x \in X$
hence $x \in Y$..
ultimately show $xs @ [x] \notin T$..
qed
qed

lemma *ts-process-failures*:
trace-set $T \Longrightarrow$
failures (*ts-process* T) = $\{(xs, X). xs \in T \wedge (\forall x \in X. xs @ [x] \notin T)\}$
by (*drule ts-process-rep, simp add: failures-def*)

lemma *ts-process-futures*:
trace-set $T \Longrightarrow$
futures (*ts-process* T) $xs =$
 $\{(ys, Y). xs @ ys \in T \wedge (\forall y \in Y. xs @ ys @ [y] \notin T)\}$
by (*simp add: futures-def ts-process-failures*)

lemma *ts-process-traces*:
trace-set $T \Longrightarrow$ *traces* (*ts-process* T) = T
proof (*drule ts-process-failures, simp add: traces-def, rule set-eqI, rule iffI, simp-all*)
qed (*rule-tac x = {} in exI, simp*)

lemma *ts-process-refusals*:
trace-set $T \Longrightarrow xs \in T \Longrightarrow$
refusals (*ts-process* T) $xs = \{X. \forall x \in X. xs @ [x] \notin T\}$
by (*drule ts-process-failures, simp add: refusals-def*)

lemma *ts-process-next-events*:
trace-set $T \Longrightarrow (x \in$ *next-events* (*ts-process* T) $xs) = (xs @ [x] \in T)$
by (*drule ts-process-traces, simp add: next-events-def*)

In what follows, the proof is given of two results which provide a connection between the notions of deterministic and trace set processes: any trace set process is deterministic, and any process is deterministic just in case it is equal to the trace set process corresponding to the set of its traces.

lemma *ts-process-d*:

trace-set $T \implies$ *deterministic* (*ts-process* T)
proof (*frule* *ts-process-traces*, *simp* *add*: *deterministic-def*, *rule* *ballI*,
drule *ts-process-refusals*, *assumption*, *simp* *add*: *next-events-def*,
rule *allI*, *rule* *iffI*)
fix $xs\ X$
assume $\forall x \in X. xs @ [x] \notin T$
thus $X \cap \{x. xs @ [x] \in T\} = \{\}$
by (*rule-tac* *equals0I*, *erule-tac* *IntE*, *simp*)
next
fix $xs\ X$
assume $A: X \cap \{x. xs @ [x] \in T\} = \{\}$
show $\forall x \in X. xs @ [x] \notin T$
proof (*rule* *ballI*, *rule* *notI*)
fix x
assume $x \in X$ **and** $xs @ [x] \in T$
hence $x \in X \cap \{x. xs @ [x] \in T\}$ **by** *simp*
moreover **have** $x \notin X \cap \{x. xs @ [x] \in T\}$ **using** A **by** (*rule* *equals0D*)
ultimately **show** *False* **by** *contradiction*
qed
qed

definition *divergences* :: 'a process \Rightarrow 'a list set **where**
divergences $P \equiv$ *snd* (*Rep-process* P)

lemma *d-divergences*:

assumes A : *deterministic* P
shows *divergences* $P = \{\}$
proof (*subst* *divergences-def*, *rule* *equals0I*)
fix xs
have B : *Rep-process* $P \in$ *process-set* (**is** $?P' \in -$) **by** (*rule* *Rep-process*)
hence $\forall xs. \exists x. xs \in$ *snd* $?P' \longrightarrow xs @ [x] \in$ *snd* $?P'$
by (*simp* *add*: *process-set-def* *process-prop-5-def*)
hence $\exists x. xs \in$ *snd* $?P' \longrightarrow xs @ [x] \in$ *snd* $?P' ..$
then **obtain** x **where** $xs \in$ *snd* $?P' \longrightarrow xs @ [x] \in$ *snd* $?P' ..$
moreover **assume** C : $xs \in$ *snd* $?P'$
ultimately **have** D : $xs @ [x] \in$ *snd* $?P' ..$
have E : $\forall xs\ X. xs \in$ *snd* $?P' \longrightarrow (xs, X) \in$ *fst* $?P'$
using B **by** (*simp* *add*: *process-set-def* *process-prop-6-def*)
hence $xs \in$ *snd* $?P' \longrightarrow (xs, \{x\}) \in$ *fst* $?P'$ **by** *blast*
hence $\{x\} \in$ *refusals* $P\ xs$
using C **by** (*drule-tac* *mp*, *simp-all* *add*: *failures-def* *refusals-def*)
moreover **have** $xs @ [x] \in$ *snd* $?P' \longrightarrow (xs @ [x], \{\}) \in$ *fst* $?P'$

using E **by** *blast*
hence $(xs @ [x], \{\}) \in failures\ P$
using D **by** (*drule-tac mp, simp-all add: failures-def*)
hence $F: xs @ [x] \in traces\ P$ **by** (*rule failures-traces*)
hence $\{x\} \cap next-events\ P\ xs \neq \{\}$ **by** (*simp add: next-events-def*)
ultimately have $G: (\{x\} \in refusals\ P\ xs) \neq (\{x\} \cap next-events\ P\ xs = \{\})$
by *simp*
have $\forall xs \in traces\ P. \forall X. X \in refusals\ P\ xs = (X \cap next-events\ P\ xs = \{\})$
using A **by** (*simp add: deterministic-def*)
moreover have $xs \in traces\ P$ **using** F **by** (*rule process-rule-2-traces*)
ultimately have $\forall X. X \in refusals\ P\ xs = (X \cap next-events\ P\ xs = \{\})$..
hence $\{x\} \in refusals\ P\ xs = (\{x\} \cap next-events\ P\ xs = \{\})$..
thus *False* **using** G **by** *contradiction*
qed

lemma *trace-set-traces:*

trace-set (traces P)
proof (*simp only: trace-set-def traces-def failures-def Domain-iff,*
rule conjI, (rule-tac [2] allI)+, rule-tac [2] impI, erule-tac [2] exE)
have $Rep-process\ P \in process-set$ (**is** $?P' \in -$) **by** (*rule Rep-process*)
hence $([], \{\}) \in fst\ ?P'$ **by** (*simp add: process-set-def process-prop-1-def*)
thus $\exists X. ([], X) \in fst\ ?P'$..

next

fix $xs\ x\ X$
have $Rep-process\ P \in process-set$ (**is** $?P' \in -$) **by** (*rule Rep-process*)
hence $\forall xs\ x\ X. (xs @ [x], X) \in fst\ ?P' \longrightarrow (xs, \{\}) \in fst\ ?P'$
by (*simp add: process-set-def process-prop-2-def*)
hence $(xs @ [x], X) \in fst\ ?P' \longrightarrow (xs, \{\}) \in fst\ ?P'$ **by** *blast*
moreover assume $(xs @ [x], X) \in fst\ ?P'$
ultimately have $(xs, \{\}) \in fst\ ?P'$..
thus $\exists X. (xs, X) \in fst\ ?P'$..

qed

lemma *d-implies-ts-process-traces:*

deterministic P \implies ts-process (traces P) = P
proof (*simp add: Rep-process-inject [symmetric] prod-eq-iff failures-def [symmetric],*
insert trace-set-traces [of P], frule ts-process-rep, frule d-divergences,
simp add: divergences-def deterministic-def)
assume $A: \forall xs \in traces\ P. \forall X.$
 $(X \in refusals\ P\ xs) = (X \cap next-events\ P\ xs = \{\})$
assume $B: trace-set (traces\ P)$
hence $C: traces (ts-process (traces\ P)) = traces\ P$ **by** (*rule ts-process-traces*)
show $failures (ts-process (traces\ P)) = failures\ P$
proof (*rule equalityI, rule-tac [!] subsetI, simp-all only: split-paired-all*)
fix $xs\ X$
assume $D: (xs, X) \in failures (ts-process (traces\ P))$
hence $xs \in traces (ts-process (traces\ P))$ **by** (*rule failures-traces*)
hence $E: xs \in traces\ P$ **using** C **by** *simp*
with B **have**

refusals (ts-process (traces P)) xs = {X. $\forall x \in X. xs @ [x] \notin traces P$ }
by (*rule ts-process-refusals*)
moreover have $X \in refusals (ts-process (traces P)) xs$
using *D* **by** (*simp add: refusals-def*)
ultimately have $\forall x \in X. xs @ [x] \notin traces P$ **by** *simp*
hence $X \cap next-events P xs = \{\}$
by (*rule-tac equals0I, erule-tac IntE, simp add: next-events-def*)
moreover have $\forall X. (X \in refusals P xs) = (X \cap next-events P xs = \{\})$
using *A* **and** *E* **..**
hence $(X \in refusals P xs) = (X \cap next-events P xs = \{\})$ **..**
ultimately have $X \in refusals P xs$ **by** *simp*
thus $(xs, X) \in failures P$ **by** (*simp add: refusals-def*)
next
fix $xs X$
assume $D: (xs, X) \in failures P$
hence $E: xs \in traces P$ **by** (*rule failures-traces*)
with *A* **have** $\forall X. (X \in refusals P xs) = (X \cap next-events P xs = \{\})$ **..**
hence $(X \in refusals P xs) = (X \cap next-events P xs = \{\})$ **..**
moreover have $X \in refusals P xs$ **using** *D* **by** (*simp add: refusals-def*)
ultimately have $F: X \cap \{x. xs @ [x] \in traces P\} = \{\}$
by (*simp add: next-events-def*)
have $\forall x \in X. xs @ [x] \notin traces P$
proof (*rule ballI, rule notI*)
fix x
assume $x \in X$ **and** $xs @ [x] \in traces P$
hence $x \in X \cap \{x. xs @ [x] \in traces P\}$ **by** *simp*
moreover have $x \notin X \cap \{x. xs @ [x] \in traces P\}$ **using** *F* **by** (*rule equals0D*)
ultimately show *False* **by** *contradiction*
qed
moreover have
refusals (ts-process (traces P)) xs = {X. $\forall x \in X. xs @ [x] \notin traces P$ }
using *B* **and** *E* **by** (*rule ts-process-refusals*)
ultimately have $X \in refusals (ts-process (traces P)) xs$ **by** *simp*
thus $(xs, X) \in failures (ts-process (traces P))$ **by** (*simp add: refusals-def*)
qed
qed

lemma *ts-process-traces-implies-d*:
 $ts-process (traces P) = P \implies deterministic P$
by (*insert trace-set-traces [of P], drule ts-process-d, simp*)

lemma *d-equals-ts-process-traces*:
 $deterministic P = (ts-process (traces P) = P)$
by (*rule iffI, erule d-implies-ts-process-traces, rule ts-process-traces-implies-d*)

Finally, a variant of the Ipurge Unwinding Theorem applying to trace set processes is derived from the variant for deterministic processes. Particularly, the assumption that the process be deterministic is replaced by the

assumption that it be a trace set process, since the former property is entailed by the latter (cf. above).

theorem *ts-ipurge-unwinding*:

trace-set $T \implies$
secure (*ts-process* T) $I D =$
d-weakly-future-consistent (*ts-process* T) $I D$ (*rel-ipurge* (*ts-process* T) $I D$)
by (*rule d-ipurge-unwinding*, *rule ts-process-d*)

end

References

- [1] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Inc., 1985.
- [2] A. Krauss. *Defining Recursive Functions in Isabelle/HOL*. <http://isabelle.in.tum.de/website-Isabelle2015/dist/Isabelle2015/doc/functions.pdf>.
- [3] T. Nipkow. *A Tutorial Introduction to Structured Isar Proofs*. <http://isabelle.in.tum.de/website-Isabelle2011/dist/Isabelle2011/doc/isar-overview.pdf>.
- [4] T. Nipkow. *Programming and Proving in Isabelle/HOL*, May 2015. <http://isabelle.in.tum.de/website-Isabelle2015/dist/Isabelle2015/doc/prog-prove.pdf>.
- [5] T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, May 2015. <http://isabelle.in.tum.de/website-Isabelle2015/dist/Isabelle2015/doc/tutorial.pdf>.
- [6] P. Noce. Noninterference security in communicating sequential processes. *Archive of Formal Proofs*, May 2014. http://isa-afp.org/entries/Noninterference_CSP.shtml, Formal proof development.
- [7] P. Noce. Reasoning about lists via list interleaving. *Archive of Formal Proofs*, June 2015. http://isa-afp.org/entries/List_Interleaving.shtml, Formal proof development.
- [8] J. Rushby. Noninterference, transitivity, and channel-control security policies. Technical report, SRI International, 1992.