# The Ipurge Unwinding Theorem for CSP Noninterference Security 

Pasquale Noce<br>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

September 13, 2023


#### 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 failuresdivergences pair. Then, variants of the Ipurge Unwinding Theorem are proven for deterministic processes and trace set processes.


## Contents

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 ${ }^{\prime} a_{1} \Rightarrow \ldots \Rightarrow{ }^{\prime} a_{m} \Rightarrow{ }^{\prime} a_{m+1}$ $\Rightarrow \ldots \Rightarrow{ }^{\prime} a_{n} \Rightarrow{ }^{\prime} b$, the explanatory text may discuss of $F$ using attributes that would more exactly apply to a term of type ${ }^{\prime} a_{m+1} \Rightarrow \ldots \Rightarrow{ }^{\prime} a_{n} \Rightarrow{ }^{\prime} b$. In this case, it shall be understood that strictly speaking, such attributes apply to a term matching pattern $F a_{1} \ldots a_{m}$.

### 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 $A^{\prime}$ is a singleton refusal by virtue of rule $\llbracket(? x s, ? Y) \in$ failures ? $P ;$ ? $X \subseteq$ ? $Y \rrbracket \Longrightarrow(? x s, ? X) \in$ failures ? $P$, so that the union of the elements of $A^{\prime}$, 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 \(\left({ }^{\prime} a,{ }^{\prime} d\right)\) dom-rel-map \(={ }^{\prime} d \Rightarrow\left({ }^{\prime} a\right.\) list \(\times{ }^{\prime} a\) list \()\) set
type-synonym \(\left({ }^{\prime} a, ' d\right)\) domset-rel-map \(=' d\) set \(\Rightarrow\left({ }^{\prime} a\right.\) list \(\times\) 'a list \()\) set
definition ref-union-closed \(::\) 'a process \(\Rightarrow\) bool where
ref-union-closed \(P \equiv\)
    \(\forall x s A .(\exists X . X \in A) \longrightarrow(\forall X \in A .(x s, X) \in\) failures \(P) \longrightarrow\)
    \((x s, \bigcup X \in A . X) \in\) failures \(P\)
definition view-partition ::
    'a process \(\Rightarrow\left(' a \Rightarrow{ }^{\prime} d\right) \Rightarrow\left({ }^{\prime} a,^{\prime} d\right)\) dom-rel-map \(\Rightarrow\) bool where
view-partition \(P D R \equiv \forall u \in\) range \(D\). equiv (traces \(P)(R u)\)
definition next-dom-events ::
    'a process \(\Rightarrow\left({ }^{\prime} a \Rightarrow^{\prime} d\right) \Rightarrow{ }^{\prime} d \Rightarrow{ }^{\prime}\) a list \(\Rightarrow\) 'a set where
next-dom-events \(P D u x s \equiv\{x . u=D x \wedge x \in\) next-events \(P x s\}\)
definition ref-dom-events ::
    'a process \(\Rightarrow\left({ }^{\prime} a \Rightarrow{ }^{\prime} d\right) \Rightarrow{ }^{\prime} d \Rightarrow{ }^{\prime}\) 'a list \(\Rightarrow\) 'a set where
ref-dom-events \(P D u x s \equiv\{x . u=D x \wedge\{x\} \in\) refusals \(P x s\}\)
definition future-consistent ::
```

```
    \({ }^{\prime}\) a process \(\Rightarrow\left({ }^{\prime} a \Rightarrow{ }^{\prime} d\right) \Rightarrow\left({ }^{\prime} a,{ }^{\prime} d\right)\) dom-rel-map \(\Rightarrow\) bool where
```

    \({ }^{\prime}\) a process \(\Rightarrow\left({ }^{\prime} a \Rightarrow{ }^{\prime} d\right) \Rightarrow\left({ }^{\prime} a,{ }^{\prime} d\right)\) dom-rel-map \(\Rightarrow\) bool where
    future-consistent $P D R \equiv$
future-consistent $P D R \equiv$
$\forall u \in$ range $D . \forall x s$ ys. $(x s, y s) \in R u \longrightarrow$
$\forall u \in$ range $D . \forall x s$ ys. $(x s, y s) \in R u \longrightarrow$
next-dom-events $P D$ u xs $=$ next-dom-events $P D$ u ys $\wedge$
next-dom-events $P D$ u xs $=$ next-dom-events $P D$ u ys $\wedge$
ref-dom-events $P D$ u xs $=$ ref-dom-events $P D$ u ys

```
        ref-dom-events \(P D\) u xs \(=\) ref-dom-events \(P D\) u ys
```

definition weakly-future-consistent ::
'a process $\Rightarrow\left({ }^{\prime} d \times{ }^{\prime} d\right)$ set $\Rightarrow\left({ }^{\prime} a \Rightarrow{ }^{\prime} d\right) \Rightarrow\left({ }^{\prime} a,{ }^{\prime} d\right)$ dom-rel-map $\Rightarrow$ bool where
weakly-future-consistent PID $\equiv$
$\forall u \in$ range $D \cap(-I)$ " range $D . \forall x s$ ys. $(x s, y s) \in R u \longrightarrow$ next-dom-events $P D$ uxs $=$ next-dom-events $P D$ u ys $\wedge$ ref-dom-events $P D$ u xs $=$ 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\mathrm{ traces P > xs }\in\mathrm{ 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]:

```
    \((x s, X) \in\) failures \(P \longrightarrow(x s @[x],\{ \}) \in\) failures \(P \vee(x s\), insert \(x X) \in\) failures
\(P\)
proof (simp add: failures-def)
    have Rep-process \(P \in\) process-set (is ? \(P^{\prime} \in-\) ) by (rule Rep-process)
    hence \(\forall x s x X .(x s, X) \in f s t ? P^{\prime} \longrightarrow\)
        \((x s @[x],\{ \}) \in f s t ? P^{\prime} \vee(x s\), insert \(x X) \in f s t ? P^{\prime}\)
    by (simp add: process-set-def process-prop-4-def)
    thus \((x s, X) \in f s t ? P^{\prime} \longrightarrow\)
        \((x s @[x],\{ \}) \in f s t ? P^{\prime} \vee(x s\), insert \(x X) \in f s t ? P^{\prime}\)
    by blast
qed
lemma failures-traces:
    \((x s, X) \in\) failures \(P \Longrightarrow x s \in\) traces \(P\)
by (simp add: traces-def Domain-iff, rule exI)
lemma traces-failures:
    xs \(\in\) traces \(P \Longrightarrow(x s,\{ \}) \in\) failures \(P\)
proof (simp add: traces-def Domain-iff, erule exE)
qed (erule process-rule-3, simp)
```

lemma sinks-interference [rule-format]:
$D x \in$ sinks I $D u x s \longrightarrow$
$(u, D x) \in I \vee(\exists v \in \operatorname{sinks} I D u x s .(v, D x) \in I)$
proof (induction xs rule: rev-induct, simp, rule impI)
fix $x^{\prime} x s$
assume
A: $D x \in \operatorname{sinks} I D u x s \longrightarrow$
$(u, D x) \in I \vee(\exists v \in \operatorname{sinks} I D u x s .(v, D x) \in I)$ and
$B: D x \in \operatorname{sinks} I D u(x s @[x])$
show $(u, D x) \in I \vee(\exists v \in \operatorname{sinks} I D u(x s @[x]) .(v, D x) \in I)$
proof $\left(\right.$ cases $\left.\left(u, D x^{\prime}\right) \in I \vee\left(\exists v \in \operatorname{sinks} I D u x s .\left(v, D x^{\prime}\right) \in I\right)\right)$
case True
hence $D x=D x^{\prime} \vee D x \in \operatorname{sinks} I D u x s$ using $B$ by simp

```
    moreover \{
        assume \(C: D x=D x^{\prime}\)
        have ?thesis using True
        proof (rule disjE, erule-tac [2] bexE)
        assume \(\left(u, D x^{\prime}\right) \in I\)
        hence \((u, D x) \in I\) using \(C\) by simp
        thus ?thesis ..
    next
        fix \(v\)
        assume \(\left(v, D x^{\prime}\right) \in I\)
        hence \((v, D x) \in I\) using \(C\) by simp
        moreover assume \(v \in \operatorname{sinks} I D u x s\)
        hence \(v \in\) sinks \(I D u(x s @[x])\) by simp
        ultimately have \(\exists v \in \operatorname{sinks} I D u(x s @[x]) .(v, D x) \in I .\).
        thus ?thesis..
        qed
    \}
    moreover \{
        assume \(D x \in\) sinks \(I D u x s\)
    with \(A\) have \((u, D x) \in I \vee(\exists v \in \operatorname{sinks} I D u x s .(v, D x) \in I)\)..
    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 \(v \in \operatorname{sinks} I D u x s\)
        hence \(v \in \operatorname{sinks} I D u(x s @[x\rceil)\) by simp
        ultimately have \(\exists v \in \operatorname{sinks} I D u(x s @[x]) .(v, D x) \in I\)..
        thus ?thesis ..
        qed
    \}
    ultimately show ?thesis ..
next
        case False
        hence \(C\) : sinks I D u (xs @ \([x])=\) sinks I \(D u x s\) by simp
        hence \(D x \in\) sinks \(I D u\) xs using \(B\) by simp
        with \(A\) have \((u, D x) \in I \vee(\exists v \in \operatorname{sinks} I D u x s .(v, D x) \in I)\)..
        thus ?thesis using \(C\) by simp
    qed
qed
lemma sinks-interference-eq:
\(((u, D x) \in I \vee(\exists v \in \operatorname{sinks} I D u x s .(v, D x) \in I))=\)
\((D x \in \operatorname{sinks} I D u(x s @[x]))\)
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 \(A: \forall x s A .(\forall X \in A .(x s, X) \in\) failures \(P) \longrightarrow\)
        ( \(x s, \bigcup X \in A . X) \in\) failures \(P\)
    shows \(\forall x s\). xs \(\in\) traces \(P\)
proof
    fix \(x s\)
    have \((\forall X \in\} .(x s, X) \in\) failures \(P) \longrightarrow(x s, \bigcup X \in\{ \} . X) \in\) failures \(P\)
        using \(A\) by blast
    moreover have \(\forall X \in\}\). (xs, X) failures \(P\) by simp
    ultimately have \((x s, \bigcup X \in\{ \}\). \(X) \in\) failures \(P\)..
    thus \(x s \in\) traces \(P\) by (rule failures-traces)
qed
lemma traces-dom-events:
    assumes \(A: u \in\) range \(D\)
    shows \(x s \in\) traces \(P=\)
        (next-dom-events \(P D u x s \cup r e f-d o m-e v e n t s ~ P D u x s \neq\{ \})\)
        (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 \(x s \in\) traces \(P\)
    hence (xs, \(\}) \in\) failures \(P\) by (rule traces-failures)
    hence \((x s @[x],\{ \}) \in\) failures \(P \vee(x s,\{x\}) \in\) failures \(P\) by (rule process-rule-4)
    moreover \{
        assume (xs @ \([x],\{ \}) \in\) failures \(P\)
        hence \(x s @[x] \in\) traces \(P\) by (rule failures-traces)
        hence \(x \in\) next-dom-events \(P D\) uxs
        using \(B\) by (simp add: next-dom-events-def next-events-def)
        hence \(x \in\) ? \(S\)..
    \}
    moreover \{
        assume \((x s,\{x\}) \in\) failures \(P\)
```

```
    hence x fef-dom-events P D u xs
    using B by (simp add:ref-dom-events-def refusals-def)
    hence }x\in\mathrm{ ?S ..
    }
    ultimately have x < ?S ..
    hence }\existsx.x\in\mathrm{ ?S ..
    thus ?S }\not={}\mathrm{ by (subst ex-in-conv [symmetric])
next
    assume ?S }\not={
    hence \existsx. x\in?S by (subst ex-in-conv)
    then obtain }x\mathrm{ where }x\in\mathrm{ ?S ..
    moreover {
    assume x next-dom-events P Duxs
    hence xs @ [x]\in traces P by (simp add: next-dom-events-def next-events-def)
    hence xs \in traces P by (rule process-rule-2-traces)
    }
    moreover {
    assume x ref-dom-events P D uxs
    hence (xs, {x}) \in failures P by (simp add: ref-dom-events-def refusals-def)
    hence xs \in traces P by (rule failures-traces)
    }
    ultimately show xs \in traces P ..
qed
lemma fc-traces:
    assumes
    A: future-consistent P D R and
    B:u}\in\mathrm{ range D and
    C:(xs,ys)\inRu
    shows (xs\in traces P)}=(ys\in\operatorname{traces}P
proof -
    have }\forallu\in\mathrm{ range D. }\forallxs ys. (xs,ys) \inRu
    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
    using A by (simp add: future-consistent-def)
    hence }\forallxs\mathrm{ ys. (xs, ys) &Ru}
    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
    using B ..
    hence (xs,ys)\inRu\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
    by blast
    hence 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
    using C ..
    hence next-dom-events P D u xs \cup ref-dom-events P D uxs }\not={}
    (next-dom-events P D u ys \cup ref-dom-events P D u ys \not={})
    by simp
```

moreover have $x s \in$ traces $P=$
(next-dom-events $P D u x s \cup r e f-d o m-e v e n t s ~ P D u x s \neq\{ \})$
using $B$ by (rule traces-dom-events)
moreover have $y s \in$ traces $P=$
(next-dom-events $P D$ u ys $\cup$ ref-dom-events $P D u y s \neq\{ \}$ )
using $B$ by (rule traces-dom-events)
ultimately show ?thesis by simp
qed
lemma wfc-traces:
assumes
A: weakly-future-consistent PIDR and
$B: u \in$ range $D \cap(-I)$ " range $D$ and
$C:(x s, y s) \in R u$
shows $(x s \in$ traces $P)=(y s \in$ traces $P)$
proof -
have $\forall u \in$ range $D \cap(-I)$ " range $D . \forall x s$ ys. $(x s, y s) \in R u \longrightarrow$ next-dom-events $P D$ u xs $=$ next-dom-events $P D$ u ys $\wedge$
ref-dom-events $P D u x s=$ ref-dom-events $P D u$ ys
using $A$ by (simp add: weakly-future-consistent-def)
hence $\forall x s$ ys. $(x s, y s) \in R u \longrightarrow$
next-dom-events $P D$ uxs = next-dom-events $P D$ u ys $\wedge$
ref-dom-events $P D$ u xs $=$ ref-dom-events $P D$ u ys
using $B$..
hence $(x s, y s) \in R u \longrightarrow$
next-dom-events $P D$ u xs = next-dom-events $P D$ u ys $\wedge$
ref-dom-events $P D$ u xs $=$ ref-dom-events $P D$ u ys
by blast
hence next-dom-events $P D u x s=$ next-dom-events $P D u$ ys $\wedge$
ref-dom-events $P D u x s=$ ref-dom-events $P D u$ ys
using $C$..
hence next-dom-events $P D u x s \cup$ ref-dom-events $P D u x s \neq\{ \}=$ (next-dom-events $P D$ u ys $\cup$ ref-dom-events $P D u y s \neq\{ \}$ )
by $\operatorname{simp}$
moreover have $B^{\prime}: u \in$ range $D$ using $B$..
hence $x s \in$ traces $P=$
(next-dom-events $P D u x s \cup r e f-d o m-e v e n t s ~ P D u x s \neq\{ \})$
by (rule traces-dom-events)
moreover have ys $\in$ traces $P=$
(next-dom-events $P D$ u ys $\cup$ ref-dom-events $P D u$ ys $\neq\{ \}$ )
using $B^{\prime}$ by (rule traces-dom-events)
ultimately show?thesis by simp
qed
lemma $f$ c-implies-wfc:
future-consistent $P D R \Longrightarrow$ weakly-future-consistent PIDR
by (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:
    \((\exists Y . Y \in\) 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 \(\exists x^{\prime} \in X .\{x\}=\left\{x^{\prime}\right\}\) using \(A\)..
    thus \(\exists Y . \exists x^{\prime} \in X . Y=\left\{x^{\prime}\right\}\) by (rule exI)
qed
lemma singleton-set-union:
    \((\bigcup Y \in\) singleton-set \(X . Y)=X\)
proof (subst singleton-set-def, rule equalityI, rule-tac [!] subsetI)
    fix \(x\)
    assume \(A: x \in\left(\bigcup Y \in\left\{Y^{\prime} . \exists x^{\prime} \in X . Y^{\prime}=\left\{x^{\prime}\right\}\right\} . Y\right)\)
    show \(x \in X\)
    proof (rule UN-E [OF A], simp)
    qed (erule bexE, simp)
next
    fix \(x\)
    assume \(A: x \in X\)
    show \(x \in\left(\bigcup Y \in\left\{Y^{\prime} . \exists x^{\prime} \in X . Y^{\prime}=\left\{x^{\prime}\right\}\right\} . Y\right)\)
    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 $x s$, and outputs the set of the event domains not allowed to be affected by $U$ after the occurrence of $x s$.

## function sinks-aux ::

```
\(\left({ }^{\prime} d \times{ }^{\prime} d\right)\) set \(\Rightarrow\left({ }^{\prime} a \Rightarrow{ }^{\prime} d\right) \Rightarrow{ }^{\prime} d\) set \(\Rightarrow\) 'a list \(\Rightarrow{ }^{\prime} d\) set where
sinks-aux - \(U[]=U \mid\)
sinks-aux I D U (xs @ [x]) \(=(\) if \(\exists v \in\) sinks-aux I D Uxs. \((v, D x) \in I\)
    then insert ( \(D x\) ) (sinks-aux I D Uxs)
    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 ::
$(' d \times ' d)$ set $\Rightarrow\left({ }^{\prime} a \Rightarrow{ }^{\prime} d\right) \Rightarrow{ }^{\prime} d$ set $\Rightarrow{ }^{\prime} a$ list $\Rightarrow{ }^{\prime} a$ list where
ipurge-tr-aux -- [] = [] |
ipurge-tr-aux IDU(xs@ [x]) $=($ if $\exists v \in$ sinks-aux ID $U$ 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 ::
$\left(' d \times{ }^{\prime} d\right)$ set $\Rightarrow\left({ }^{\prime} a \Rightarrow{ }^{\prime} d\right) \Rightarrow{ }^{\prime} d$ set $\Rightarrow{ }^{\prime} a$ list $\Rightarrow{ }^{\prime} a$ set $\Rightarrow{ }^{\prime} a$ set where
ipurge-ref-aux I D U xs $X \equiv$
$\{x \in X . \forall v \in$ sinks-aux I D Uxs. $(v, D x) \notin I\}$
definition unaffected-domains ::
$\left({ }^{\prime} d \times{ }^{\prime} d\right)$ set $\Rightarrow\left({ }^{\prime} a \Rightarrow{ }^{\prime} d\right) \Rightarrow{ }^{\prime} d$ set $\Rightarrow{ }^{\prime}$ a list $\Rightarrow{ }^{\prime} d$ set where
unaffected-domains I D U xs $\equiv$
$\{u \in$ range $D . \forall v \in \operatorname{sinks-aux~I~D~} U x s .(v, u) \notin I\}$

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 $x s$, whose recursive decomposition is performed by item prepending rather than appending. Then:

- sources outputs the set of the domains of the events in $x s$ allowed to affect $u$;
- ipurge-tr-rev outputs the sublist of $x s$ 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 $\left.\times{ }^{\prime} d\right)$ set $\Rightarrow\left({ }^{\prime} a \Rightarrow{ }^{\prime} d\right) \Rightarrow{ }^{\prime} d \Rightarrow{ }^{\prime} a$ list $\Rightarrow^{\prime} d$ set where
sources - - [] = \{\}|
sources I D u $(x \# x s)=$
(if $(D x, u) \in I \vee(\exists v \in$ sources $I D u x s .(D x, v) \in I)$
then insert ( $D x$ ) (sources I $D u x s$ )
else sources I Duxs)
primrec ipurge-tr-rev :: ('d $\times$ 'd) set $\Rightarrow\left({ }^{\prime} a \Rightarrow{ }^{\prime} d\right) \Rightarrow{ }^{\prime} d \Rightarrow{ }^{\prime} a$ list $\Rightarrow{ }^{\prime} a$ list where ipurge-tr-rev - - [] = [] |
ipurge-tr-rev I Du(x\#xs) $=($ if $D x \in$ sources $I D u(x \# x s)$
then $x$ \# ipurge-tr-rev I $D$ u xs
else ipurge-tr-rev I Duxs)
primrec sources-aux ::

```
\(\left(' d \times{ }^{\prime} d\right)\) set \(\Rightarrow\left(' a \Rightarrow{ }^{\prime} d\right) \Rightarrow{ }^{\prime} d\) set \(\Rightarrow{ }^{\prime}\) a list \(\Rightarrow{ }^{\prime} d\) set where
sources-aux - \(U[]=U \mid\)
sources-aux I D \(U(x \# x s)=(\) if \(\exists v \in\) sources-aux I D \(U\) xs. \((D x, v) \in I\)
    then insert \((D x)\) (sources-aux I D Uxs)
    else sources-aux I D U xs)
primrec ipurge-tr-rev-aux ::
    \(\left({ }^{\prime} d \times{ }^{\prime} d\right)\) set \(\Rightarrow\left({ }^{\prime} a \Rightarrow{ }^{\prime} d\right) \Rightarrow{ }^{\prime} d\) set \(\Rightarrow{ }^{\prime} a\) list \(\Rightarrow{ }^{\prime} a\) list where
ipurge-tr-rev-aux -- [] = [] |
ipurge-tr-rev-aux I D \(U(x \# x s)=(\) if \(\exists v \in\) sources-aux I D \(U\) xs. \((D x, v) \in I\)
    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\subseteqsinks-aux I D U xs
proof (induction xs rule: rev-induct, simp-all, rule impI)
qed (rule subset-insertI2)
lemma sinks-aux-single-dom:
    sinks-aux I D {u}xs= insert u(sinks I Duxs)
by (induction xs rule: rev-induct, simp-all add: insert-commute)
lemma sinks-aux-single-event:
    sinks-aux I D U [x] = (if \existsv\inU. (v,D x)\inI
    then insert (D x) U
    else U)
proof -
    have sinks-aux I D U [x] = sinks-aux I D U ([] @ [x]) by simp
    thus ?thesis by (simp only: sinks-aux.simps)
qed
lemma sinks-aux-cons:
    sinks-aux I D U (x# xs) = (if \existsv\inU. (v,D x) \inI
    then sinks-aux I D (insert (D x) U) xs
    else sinks-aux I D U xs)
proof (induction xs rule: rev-induct, case-tac [!] \existsv\inU. (v,D x) \inI,
    simp-all add: sinks-aux-single-event del: sinks-aux.simps(2))
    fix }\mp@subsup{x}{}{\prime}x
    assume A: sinks-aux I D U (x # xs) = sinks-aux I D (insert (D x) U)xs
        (is ?S = ?S')
    show sinks-aux I D U (x# xs @ [x]) =
        sinks-aux I D (insert (D x)U) (xs@ @ [x])
    proof (cases }\existsv\in?S.(v,D\mp@subsup{x}{}{\prime})\inI
        case True
        hence sinks-aux I D U ((x # xs)@ [x]) = insert (D x') ?S
        by (simp only: sinks-aux.simps, simp)
        moreover have }\existsv\in?\mp@subsup{S}{}{\prime}.(v,D\mp@subsup{x}{}{\prime})\inI\mathrm{ using A and True by simp
        hence sinks-aux I D (insert (D x) U) (xs@ @ [x]) = insert (D x') ?S'
            by simp
        ultimately show ?thesis using A by simp
    next
        case False
        hence sinks-aux ID U ((x# xs)@ [x]) =?S
        by (simp only: sinks-aux.simps, simp)
        moreover have }\neg(\existsv\in?\mp@subsup{S}{}{\prime}.(v,D\mp@subsup{x}{}{\prime})\inI)\mathrm{ using A and False by simp
        hence sinks-aux I D (insert (D x) U) (xs@ @ | ] ) =? S' by simp
        ultimately show ?thesis using A by simp
    qed
next
    fix }\mp@subsup{x}{}{\prime}x
    assume A: sinks-aux I D U (x# xs) = sinks-aux I D U xs
        (is ?S = ? S')
```

```
    show sinks-aux I D U (x# xs @ [x])= sinks-aux I D U (xs @ [x])
    proof (cases }\existsv\in?S.(v,D\mp@subsup{x}{}{\prime})\inI
    case True
    hence sinks-aux I D U ((x# xs)@ [x\) = insert (D x') ?S
        by (simp only: sinks-aux.simps, simp)
    moreover have }\existsv\in??\mp@subsup{S}{}{\prime}.(v,D\mp@subsup{x}{}{\prime})\inI\mathrm{ using A and True by simp
    hence sinks-aux I D U (xs @ [x]) = insert (D 㘯)?S' by simp
    ultimately show ?thesis using A by simp
    next
    case False
    hence sinks-aux I D U ((x# xs)@ [x]) = ?S
        by (simp only: sinks-aux.simps, simp)
    moreover have}\neg(\existsv\in?\mp@subsup{S}{}{\prime}.(v,D\mp@subsup{x}{}{\prime})\inI)\mathrm{ using A and False by simp
    hence 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:
    ipurge-tr-aux I D {u} xs = ipurge-tr I D u xs
proof (induction xs rule: rev-induct, simp)
    fix x xs
    assume A: ipurge-tr-aux I D {u} xs = ipurge-tr I D uxs
    show ipurge-tr-aux I D {u}(xs@ [x])=ipurge-tr I Du(xs@ @x])
    proof (cases }\existsv\in\mathrm{ sinks-aux I D {u} xs. (v,D x) }\inI\mathrm{ ,
        simp-all only: ipurge-tr-aux.simps if-True if-False)
        case True
        hence (u,Dx) \inI\vee (\existsv\in\operatorname{sinks}IDuxs. (v,Dx)\inI)
        by (simp add: sinks-aux-single-dom)
    hence ipurge-tr IDu(xs @ [x])= ipurge-tr I Duxs by simp
    thus ipurge-tr-aux I D {u} xs = ipurge-tr I Du(xs @ [x])
        using A by simp
    next
        case False
        hence }\neg((u,Dx)\inI\vee (\existsv\in\operatorname{sinks}IDuxs. (v,Dx)\inI)
        by (simp add: sinks-aux-single-dom)
    hence D x\not\in sinks I D u (xs @ [x])
        by (simp only: sinks-interference-eq, simp)
    hence ipurge-tr I Du(xs@ [x])= ipurge-tr I D u xs @ [x] by simp
    thus ipurge-tr-aux I D {u} xs @ [x] = ipurge-tr I D u (xs @ [x])
        using A by simp
    qed
qed
lemma ipurge-ref-aux-single-dom:
    ipurge-ref-aux I D {u} xs X = 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]:
```

```
\(\left(\forall u \in U . \neg\left(\exists v \in D^{\prime}(X \cup\right.\right.\) set \(\left.\left.x s) .(u, v) \in I\right)\right) \longrightarrow\)
```

    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:
$\neg\left(\exists v \in D^{\prime}(X \cup\right.$ set xs $\left.) .(u, v) \in I\right) \Longrightarrow$ ipurge-ref $I D$ uss $X=X$
by (subst ipurge-ref-aux-single-dom [symmetric], rule ipurge-ref-aux-all, simp)
lemma unaffected-domains-single-dom:
$\{x \in X . D x \in$ unaffected-domains I $D\{u\} x s\}=$ ipurge-ref I $D u x s X$
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:
sources I Duxs sinks $\left(I^{-1}\right) D u($ rev xs $)$
by (induction xs, simp-all)
lemma sources-sinks-aux:
sources-aux I D U xs = sinks-aux $\left(I^{-1}\right) D U($ rev xs $)$
by (induction xs, simp-all)
lemma sources-aux-subset:
$U \subseteq$ sources-aux I D U xs
by (subst sources-sinks-aux, rule sinks-aux-subset)
lemma sources-aux-append:
sources-aux I D U (xs@ys)=sources-aux I D (sources-aux I D U ys) xs by (induction xs, simp-all)
lemma sources-aux-append-nil [rule-format]:
sources-aux I D U ys $=U \longrightarrow$
sources-aux I D U (xs @ ys) = sources-aux I D Uxs
by (induction xs, simp-all)
lemma ipurge-tr-rev-aux-append:
ipurge-tr-rev-aux IDU(xs @ ys) =
ipurge-tr-rev-aux I D (sources-aux I D U ys) xs @ 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]:
ipurge-tr-rev-aux I D $U x s=[] \longrightarrow\left(\forall u \in U . \neg\left(\exists v \in D^{\prime}\right.\right.$ set $\left.\left.x s .(v, u) \in I\right)\right)$
by (induction xs rule: rev-induct, simp-all add: ipurge-tr-rev-aux-append)
lemma ipurge-tr-rev-aux-nil-2 [rule-format]:
$\left(\forall u \in U . \neg\left(\exists v \in D^{\prime}\right.\right.$ set $\left.\left.x s .(v, u) \in I\right)\right) \longrightarrow$ 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:
(ipurge-tr-rev-aux I D U xs $=[])=(\forall u \in U . \neg(\exists v \in D$ 'set $x s .(v, u) \in I))$
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]:
ipurge-tr-rev-aux I D $U$ xs $=[] \longrightarrow$ sources-aux I D $U x s=U$
by (induction xs, simp-all)
lemma ipurge-tr-rev-aux-append-nil-1 [rule-format]:
ipurge-tr-rev-aux I D U ys = [] $\longrightarrow$
ipurge-tr-rev-aux I D $U$ (xs @ ys) = 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]:
ipurge-tr-rev-aux I D U xs =x \# ws $\longrightarrow$
( $\exists$ ys zs. $x s=y s @ x \# z s \wedge$
ipurge-tr-rev-aux I D (sources-aux I D $U(x \# z s))$ ys $=[] \wedge$
$(\exists v \in$ sources-aux I D U zs. $(D x, v) \in I))$
proof (induction xs, simp, rule impI)
fix $x^{\prime} x s$
assume
A: ipurge-tr-rev-aux I D U xs $=x \#$ ws $\longrightarrow$ ( $\exists$ ys zs. xs $=y s$ @ $x \# z s \wedge$ ipurge-tr-rev-aux I D (sources-aux I D $U(x \# z s))$ ys $=[] \wedge$ $(\exists v \in$ sources-aux I D $U$ zs. $(D x, v) \in I))$ and B: ipurge-tr-rev-aux I $D U\left(x^{\prime} \# x s\right)=x \#$ ws
show $\exists y s$ zs. $x^{\prime} \# x s=y s @ x \# z s \wedge$
ipurge-tr-rev-aux I D (sources-aux I D $U(x \# z s))$ ys $=[] \wedge$
$(\exists v \in$ sources-aux I D U zs. $(D x, v) \in I)$
proof (cases $\exists v \in$ sources-aux I D Uxs. $\left.\left(D x^{\prime}, v\right) \in I\right)$
case True
then have $x^{\prime}=x$ using $B$ by simp
with True have $x^{\prime} \# x s=x \# x s \wedge$
ipurge-tr-rev-aux I D (sources-aux I D U (x\#xs)) [] = [] ^
$(\exists v \in$ sources-aux I D Uxs. $(D x, v) \in I)$
by $\operatorname{simp}$
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 $\exists y s z s . x s=y s$ @ $x \# z s \wedge$
ipurge-tr-rev-aux I D (sources-aux I D $U(x \# z s))$ ys $=[] \wedge$

```
    (\existsv\in sources-aux I D U zs. (D x,v) \inI) ..
    then obtain ys and zs where xs: xs = ys @ x # zs ^
        ipurge-tr-rev-aux I D (sources-aux I D U (x # zs)) ys = []^
        (\existsv\in sources-aux I D U zs. (D x,v) \inI)
        by blast
    then have
        \neg ( \exists v \in \text { sources-aux I D (sources-aux I D U (x\#zs)) ys. (D x', v) fI)}
        using False by (simp add: sources-aux-append)
    hence ipurge-tr-rev-aux I D (sources-aux I D U (x # zs)) (x' # ys)=
        ipurge-tr-rev-aux I D (sources-aux I D U (x# zs)) ys
        by simp
    with xs have }\mp@subsup{x}{}{\prime}#xs=(\mp@subsup{x}{}{\prime}#ys)@x#zs
        ipurge-tr-rev-aux I D (sources-aux I D U (x##zs)) (x'# ys)=[]^
        (\existsv\in sources-aux I D U zs. (D x,v) \inI)
    by (simp del: sources-aux.simps)
    thus ?thesis by blast
    qed
qed
lemma ipurge-tr-rev-aux-last-1 [rule-format]:
    ipurge-tr-rev-aux I D U xs =ws @ [x] \longrightarrow(\existsv\inU. (D x,v)\inI)
proof (induction xs rule: rev-induct, simp, rule impI)
    fix xs x'
    assume
    A: ipurge-tr-rev-aux I D U xs =ws @ [x]\longrightarrow(\existsv\inU. (D x,v) \inI) and
    B: ipurge-tr-rev-aux I D U (xs @ [x ]) = ws @ [x]
    show \existsv\inU.(D x,v)\inI
    proof (cases \existsv\inU.(D\mp@subsup{x}{}{\prime},v)\inI)
        case True
        hence ipurge-tr-rev-aux I D U (xs @ [x])=
            ipurge-tr-rev-aux I D (insert (D x') U) xs @ [x]
            by (simp add: ipurge-tr-rev-aux-append)
    hence }\mp@subsup{x}{}{\prime}=x\mathrm{ using B by simp
    thus ?thesis using True by simp
    next
        case False
        hence ipurge-tr-rev-aux I D U (xs @ [x | ) = ipurge-tr-rev-aux I D U xs
            by (simp add: ipurge-tr-rev-aux-append)
    hence 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]:
ipurge-tr-rev-aux I D U xs=ws @ [x] \longrightarrow
    (\existsys zs.xs=ys @ x # zs ^ ipurge-tr-rev-aux I D U zs=[])
proof (induction xs rule: rev-induct, simp, rule impI)
    fix xs x'
    assume
```

```
    A: ipurge-tr-rev-aux I D U xs =ws @ [x] \longrightarrow
            (\existsys zs.xs=ys @ x # zs ^ ipurge-tr-rev-aux I D U zs=[]) and
    B: ipurge-tr-rev-aux I D U (xs @ [x])=ws @ [x]
    show \existsys zs. xs @ [x]=ys@ x # zs ^ ipurge-tr-rev-aux I D U zs = []
    proof (cases \existsv\inU.(D\mp@subsup{x}{}{\prime},v)\inI)
    case True
    hence ipurge-tr-rev-aux I D U (xs @ [x`])=
        ipurge-tr-rev-aux I D (insert (D x') U) xs @ [x]
    by (simp add: ipurge-tr-rev-aux-append)
    hence xs @ [x]=xs @ x # [] ^ ipurge-tr-rev-aux I D U [] = []
    using B by simp
    thus ?thesis by blast
next
    case False
    hence ipurge-tr-rev-aux I D U (xs @ [x ]) = ipurge-tr-rev-aux I D U xs
        by (simp add: ipurge-tr-rev-aux-append)
    hence ipurge-tr-rev-aux I D U xs = ws @ [x] using B by simp
    with A have \existsys zs.xs=ys @ x # zs ^ ipurge-tr-rev-aux I D U zs=[]..
    then obtain ys and zs where
        C:xs=ys @ x # zs ^ ipurge-tr-rev-aux I D U zs=[]
        by blast
    hence xs @ [x]=ys @ x# zs @ [x] by simp
    moreover have
        ipurge-tr-rev-aux I D U (zs @ [x])=ipurge-tr-rev-aux I D U zs
        using False by (simp add: ipurge-tr-rev-aux-append)
    hence ipurge-tr-rev-aux I DU(zs @ [x |) = [] using C by simp
    ultimately have xs @ [x]=ys @ x # zs @ [x]^^
        ipurge-tr-rev-aux I D U (zs @ [x]) = [] ..
    thus ?thesis by blast
    qed
qed
lemma ipurge-tr-rev-aux-all [rule-format]:
    (\forallv\inD'set xs. \existsu\inU. (v,u)\inI)\longrightarrow \ipurge-tr-rev-aux I D U xs = xs
proof (induction xs, simp, rule impI, simp, erule conjE)
    fix }x\mathrm{ xs
    assume }\existsu\inU.(Dx,u)\in
    then obtain u where A:u\inU and B:(Dx,u)\inI ..
    have U\subseteq sources-aux I D U xs by (rule sources-aux-subset)
    hence }u\in\mathrm{ sources-aux I D U xs using A ..
    with B show }\existsu\in\mathrm{ sources-aux I D U xs. (D x,u) fI ..
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 \(\Rightarrow{ }^{\prime} a \Rightarrow{ }^{\prime} a\) list \(\Rightarrow\) nat option where
offset - - [] = None
offset \(n x(y \# y s)=(\) if \(y=x\) then Some \(n\) else offset \((\) Suc \(n) x y s)\)
lemma offset-not-none-1 [rule-format]:
    offset \(k x x=\) None \(\longrightarrow(\exists y s z s . x s=y s @ x \# z s)\)
proof (induction xs arbitrary: \(k\), simp, rule impI)
    fix \(w\) xs \(k\)
    assume
        \(A: \bigwedge k\). offset \(k x x s \neq\) None \(\longrightarrow(\exists y s z s . x s=y s @ x \# z s)\) and
        B: offset \(k x(w \# x s) \neq\) None
    show \(\exists y s z s . w \# x s=y s @ x \# z s\)
    proof (cases \(w=x\), simp)
        case True
        hence \(x \# x s=[] @ x \# x s\) by \(\operatorname{simp}\)
        thus \(\exists y s z s . x \# x s=y s @ x \# z s\) by blast
    next
        case False
        hence offset \(k x(w \# x s)=o f f s e t(S u c k) x\) xs by simp
        hence offset (Suc \(k\) ) \(x\) xs \(\neq\) None using \(B\) by simp
        moreover have offset (Suc k) x \(x s \neq\) None \(\longrightarrow(\exists y s z s . x s=y s @ x \# z s)\)
        using \(A\).
        ultimately have \(\exists y s z s . x s=y s\) @ \(x \# z s\) by simp
        then obtain ys and \(z s\) where \(x s=y s @ x \# z s\) by blast
        hence \(w \# x s=(w \# y s) @ x \# z s\) by \(\operatorname{simp}\)
        thus \(\exists y s z s . w \# x s=y s @ x \# z s\) by blast
    qed
qed
lemma offset-not-none-2 [rule-format]:
    \(x s=y s @ x \# z s \longrightarrow\) offset \(k x x s \neq\) None
proof (induction xs arbitrary: ys \(k\), simp-all del: not-None-eq, rule impI)
    fix \(w\) xs ys \(k\)
    assume
        \(A: \bigwedge y s^{\prime} k^{\prime} . x s=y s^{\prime} @ x \# z s \longrightarrow\) offset \(k^{\prime} x\left(y s^{\prime} @ x \# z s\right) \neq\) None and
        \(B: w \# x s=y s @ x \# z s\)
    show offset \(k x(y s @ x \# z s) \neq\) None
    proof (cases ys, simp-all del: not-None-eq, rule impI)
        fix \(y^{\prime} y s^{\prime}\)
        have \(x s=y s^{\prime} @ x \# z s \longrightarrow\) offset \((\) Suc \(k) x\left(y s^{\prime} @ x \# z s\right) \neq\) None
            using \(A\).
        moreover assume \(y s=y^{\prime} \#\) ys \({ }^{\prime}\)
        hence \(x s=y s^{\prime} @ x \# z s\) using \(B\) by simp
        ultimately show offset \((\) Suc \(k) x\left(y s^{\prime} @ x \# z s\right) \neq\) None ..
    qed
qed
lemma offset-not-none:
    \((\) offset \(k x x s \neq\) None \()=(\exists y s z s . x s=y s @ x \# z s)\)
```

by (rule iffI, erule offset-not-none-1, (erule exE)+, rule offset-not-none-2)
lemma offset-addition [rule-format]:
offset $k x x s \neq$ None $\longrightarrow$ offset $(n+m) x$ xs $=$ Some $($ the $(o f f s e t n x a s)+m)$
proof (induction xs arbitrary: $k n$, simp, rule impI)
fix $w$ xs $k n$

## assume

A: $\wedge k n$. offset $k x$ xs $\neq$ None $\longrightarrow$
offset $(n+m) x x s=$ Some (the (offset $n x x s)+m)$ and $B$ : offset $k x(w \# x s) \neq$ None
show offset $(n+m) x(w \# x s)=$ Some (the $($ offset $n x(w \# x s))+m)$
proof (cases $w=x$, simp-all)

## case False

hence offset $k x(w \# x s)=o f f s e t(S u c k) x$ xs by simp
hence offset (Suc k) x xs $\neq$ None using $B$ by simp
moreover have offset (Suc k) x xs $\neq$ None $\longrightarrow$
offset $($ Suc $n+m) x$ ss $=$ Some (the (offset (Suc n) xxs) $+m$ )
using $A$.
ultimately show offset $(\operatorname{Suc}(n+m)) x$ xs $=$ Some (the (offset (Suc n) xxs) $+m$ )
by $\operatorname{simp}$
qed
qed
lemma offset-suc:
assumes $A$ : offset $k x x s \neq$ None
shows offset (Suc n) x xs $=$ Some (Suc (the (offset nxxs)) )
proof -
have offset (Suc n) x xs $=$ offset $(n+$ Suc 0) $x$ xs by simp
also have $\ldots=$ Some (the (offset $n x x s$ ) + Suc 0) using $A$ by (rule off-set-addition)
also have $\ldots=$ Some (Suc (the (offset $n x x s)$ )) by simp
finally show ?thesis.
qed
lemma ipurge-tr-rev-aux-first-offset [rule-format]:
$x s=y s @ x \# z s \wedge$ ipurge-tr-rev-aux I D (sources-aux I D $U(x \# z s)) y s=[] \wedge$
$(\exists v \in$ sources-aux I D Uzs. $(D x, v) \in I) \longrightarrow$
$y s=$ take $($ the $($ offset $0 x x s)) x s$
proof (induction xs arbitrary: ys, simp, rule impI, (erule conjE)+)
fix $x^{\prime} x s$ ys
assume
$A: \bigwedge y s . x s=y s @ x \# z s \wedge$
ipurge-tr-rev-aux I D (sources-aux I D $U(x \# z s))$ ys $=[] \wedge$
$(\exists v \in$ sources-aux I D $U z s .(D x, v) \in I) \longrightarrow$
$y s=$ take (the (offset $0 x x s)$ ) xs and
$B: x^{\prime} \# x s=y s @ x \# z s$ and
$C$ : ipurge-tr-rev-aux I D (sources-aux I D $U(x \# z s))$ ys $=[]$ and
$D: \exists v \in$ sources-aux I D Uzs. $(D x, v) \in I$

```
show ys = take (the (offset 0x (x'# xs))) (x'# xs)
proof (cases ys)
    case Nil
    then have }\mp@subsup{x}{}{\prime}=x\mathrm{ using B by simp
    with Nil show ?thesis by simp
next
    case (Cons y ys')
    hence E: xs=ys'@ @ #zs using B by simp
    moreover have
        F: ipurge-tr-rev-aux I D (sources-aux I D U (x # zs)) (y # ys') = []
    using Cons and C by simp
    hence
        G:\neg(\existsv\in sources-aux I D (sources-aux I D U (x#zs)) ys'. (D y,v)\inI)
    by (rule-tac notI, simp)
    hence ipurge-tr-rev-aux I D (sources-aux I D U (x # zs)) ys' = []
    using F by simp
    ultimately have xs=ys'@ @ #zs ^
        ipurge-tr-rev-aux I D (sources-aux I D U (x # zs)) ys' = [] ^
        (\existsv\in sources-aux I D U zs. (D x,v) \inI)
    using D by blast
    with A have H:ys' = take (the (offset 0xxs)) xs ..
    have I: 秋= y using Cons and B by simp
    hence
        J:\neg (\existsv\in sources-aux I D (sources-aux I D Uzs) (ys'@ @x]). (D x',v) \inI)
    using G by (simp add: sources-aux-append)
    have }\mp@subsup{x}{}{\prime}\not=
    proof
    assume }\mp@subsup{x}{}{\prime}=
    hence }\existsv\in\mathrm{ sources-aux I D U zs. (D x',v) GI using D by simp
    then obtain v}\mathrm{ where K:v 的urces-aux I D U zs and L: (D x',v) & I ..
    have sources-aux I D U zs\subseteq
        sources-aux I D (sources-aux I D U zs) (ys' @ [x])
        by (rule sources-aux-subset)
    hence v sources-aux I D (sources-aux I D U zs) (ys'` @ [x]) using K ..
    with L have
        \existsv\in sources-aux I D (sources-aux I D U zs) (ys' @ [x]). (D x', v) \inI ..
        thus False using J by contradiction
    qed
    hence offset 0x ( }\mp@subsup{x}{}{\prime}##xs)=offset (Suc 0) x xs by simp
    also have \ldots.. Some (Suc (the (offset 0 x xs)))
    proof -
    have \existsys zs.xs=ys @ x# zs using E by blast
    hence offset 0 x xs \not=None by (simp only: offset-not-none)
    thus ?thesis by (rule offset-suc)
qed
finally have take (the (offset 0x (x'# xs))) ( }\mp@subsup{x}{}{\prime}##xs)
    x' # take (the (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 Vxs $\longrightarrow$ 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 $U(x s$ @ ys) = ipurge-tr-rev-aux I D Vxs $\longrightarrow$ 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 $D V(x \# x s)$
show ipurge-tr-rev-aux I D U ys = []
proof (cases $\exists v \in$ sources-aux I $D V$ xs. $(D x, v) \in I)$
case True
hence $C$ : ipurge-tr-rev-aux ID $U(x \# x s @ y s)=$
x \# ipurge-tr-rev-aux I D V xs
using $B$ by simp
hence $\exists v s$ ws. $x \# x s$ @ $y s=v s$ @ $x \# w s \wedge$
ipurge-tr-rev-aux I D (sources-aux I D $U(x \#$ ws $)$ ) vs $=[] \wedge$
$(\exists v \in$ sources-aux I D U ws. $(D x, v) \in I)$
by (rule ipurge-tr-rev-aux-first)
then obtain vs and ws where $*: x \# x s$ @ $y s=v s$ @ $x \# w s \wedge$
ipurge-tr-rev-aux I D (sources-aux I D U (x\#ws)) vs = [] ^
$(\exists v \in$ sources-aux I D $U$ ws. $(D x, v) \in I)$
by blast
then have $v s=$ take (the (offset $0 x(x \# x s$ @ ys))) (x\#xs @ ys)
by (rule ipurge-tr-rev-aux-first-offset)
hence $v s=[]$ by $\operatorname{simp}$
with $*$ have $\exists v \in$ sources-aux I $D U(x s @ y s) .(D x, v) \in I$ 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 $\neg(\exists v \in$ sources-aux I $D U(x s @ y s) .(D x, v) \in I)$
proof
assume $\exists v \in$ sources-aux I D $U$ (xs @ys). $(D x, v) \in I$
hence ipurge-tr-rev-aux I D V $(x \# x s)=$
x \# ipurge-tr-rev-aux I D U (xs @ ys)
using $B$ by simp
hence $\exists$ vs ws. $x \# x s=v s$ @ $x \#$ ws $\wedge$
ipurge-tr-rev-aux I D (sources-aux I D V (x \# ws)) vs = [] ^
$(\exists v \in$ sources-aux I D V ws. $(D x, v) \in I)$
by (rule ipurge-tr-rev-aux-first)
then obtain vs and ws where $*: x \# x s=v s @ x \# w s \wedge$
ipurge-tr-rev-aux I D (sources-aux I D V $(x \#$ ws $)$ ) vs $=[] \wedge$ $(\exists v \in$ sources-aux I D V ws. $(D x, v) \in I)$
by blast

```
            then have vs \(=\) take (the (offset \(0 x(x \# x s)))(x \# x s)\)
            by (rule ipurge-tr-rev-aux-first-offset)
            hence \(v s=[]\) by \(\operatorname{simp}\)
            with \(*\) have \(\exists v \in\) sources-aux I \(D V x s .(D x, v) \in I\) by simp
            thus False using False by contradiction
                    qed
                            ultimately have ipurge-tr-rev-aux ID \(U(x s\) @ 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 $U$ (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:
```

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

```
lemma Interleaves-ipurge-tr-aux:
    \(x s \cong\left\{\right.\) ipurge-tr-rev-aux I D \(U\) xs, rev (ipurge-tr-aux \(\left(I^{-1}\right) D U(\) rev xs \()\) ),
\(\lambda y\) ys. \(\exists v \in \operatorname{sinks}\)-aux \(\left(I^{-1}\right) D U(\) rev ys \(\left.) .(D y, v) \in I\right\}\) proof (induction xs, simp, simp only: rev.simps)
fix \(x\) xs
assume \(A: x s \cong\{\) ipurge-tr-rev-aux \(I D U x s\), rev (ipurge-tr-aux \(\left(I^{-1}\right) D U(\) rev \(\left.x s)\right)\), \(\lambda y\) ys. \(\exists v \in \operatorname{sinks}\)-aux \(\left(I^{-1}\right) D U(\) rev ys \(\left.) .(D y, v) \in I\right\}\) (is \(-\cong\{? y s, ? z s, ? P\}\) )
show \(x \# x s \cong\)
\{ipurge-tr-rev-aux I D U (x \# xs),
rev (ipurge-tr-aux ( \(I^{-1}\) ) D U (rev xs @ \(\left.[x]\right)\) ), ?P\}
proof (cases ?P \(x\) xs, simp-all (no-asm-simp) add: sources-sinks-aux) case True
thus \(x \# x s \cong\{x \#\) ? ys, ?zs, ? \(P\}\) using \(A\) by (cases ?zs, simp-all)
next
case False
thus \(x \# x s \cong\{? y s, x \#\) ?zs, ?P\} using \(A\) by (cases ?ys, simp-all)
qed
qed
lemma ipurge-tr-aux-all:
(ipurge-tr-aux ID Uxs \(=x s)=\left(\forall u \in U . \neg\left(\exists v \in D^{\prime}\right.\right.\) set \(\left.\left.x s .(u, v) \in I\right)\right)\)
proof -
have \(A\) : rev \(x s \cong\left\{\right.\) ipurge-tr-rev-aux \(\left(I^{-1}\right) D U\) (rev xs), rev (ipurge-tr-aux \(\left(\left(I^{-1}\right)^{-1}\right) D U(\) rev (rev xs \(\left.)\right)\) ),
\(\lambda y\) ys. \(\exists v \in\) sinks-aux \(\left(\left(I^{-1}\right)^{-1}\right) D U(\) rev ys \(\left.) .(D y, v) \in\left(I^{-1}\right)\right\}\)
(is \(-\cong\{-,-, ? P\}\) )
by (rule Interleaves-ipurge-tr-aux)
show ?thesis
proof
assume ipurge-tr-aux I D \(U\) xs \(=x s\)
hence rev \(x s \cong\left\{\right.\) ipurge-tr-rev-aux \(\left(I^{-1}\right) D U(\) rev xs), rev xs, ?P \(\}\)
using \(A\) by simp
hence rev \(x s \simeq\left\{\right.\) ipurge-tr-rev-aux \(\left(I^{-1}\right) D U(\) rev xs), rev xs, ?P \(\}\)
by (rule Interleaves-interleaves)
moreover have rev \(x s \simeq\{[]\), rev \(x s, ? P\}\) by (rule interleaves-nil-all)
ultimately have ipurge-tr-rev-aux \(\left(I^{-1}\right) D U(\) rev xs \()=[]\)
by (rule interleaves-equal-fst)
thus \(\forall u \in U . \neg\left(\exists v \in D^{\prime}\right.\) set \(\left.x s .(u, v) \in I\right)\)
by (simp add: ipurge-tr-rev-aux-nil)
next
assume \(\forall u \in U . \neg(\exists v \in D\) 'set \(x s .(u, v) \in I)\)
hence ipurge-tr-rev-aux ( \(I^{-1}\) ) \(D U(\) rev xs \()=[]\)
by (simp add: ipurge-tr-rev-aux-nil)
hence rev \(x s \cong\{[]\), rev (ipurge-tr-aux I D U xs), ?P\} using \(A\) by simp
hence rev \(x s \simeq\{[]\), rev (ipurge-tr-aux I D U xs), ?P \(\}\)
by (rule Interleaves-interleaves)
hence rev xs \(\simeq\{\) rev (ipurge-tr-aux I D U xs), [], \(\lambda w\) ws. \(\neg ? P\) wws \(\}\) by (subst (asm) interleaves-swap)
moreover have rev \(x s \simeq\{\) rev \(x s,[], \lambda w w s . \neg ? P\) wws \(\}\)
```

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

```
lemma ipurge-tr-rev-aux-single-dom:
    ipurge-tr-rev-aux I D \(\{u\}\) xs \(=\) ipurge-tr-rev I \(D u x s\) (is ?ys \(=\) ? \(\left.y s s^{\prime}\right)\)
proof -
    have \(x s \cong\left\{\right.\) ? \(y s\), rev (ipurge-tr-aux \(\left(I^{-1}\right) D\{u\}\) (rev xs)),
        \(\lambda y\) ys. \(\exists v \in \operatorname{sinks-aux}\left(I^{-1}\right) D\{u\}\) (rev ys). \(\left.(D y, v) \in I\right\}\)
    by (rule Interleaves-ipurge-tr-aux)
    hence \(x s \cong\left\{\right.\) ? ys, rev (ipurge-tr \(\left(I^{-1}\right) D u(\) rev xs) \()\),
        \(\lambda y\) ys. \(\left.(u, D y) \in I^{-1} \vee\left(\exists v \in \operatorname{sinks}\left(I^{-1}\right) D u(r e v y s) .(v, D y) \in I^{-1}\right)\right\}\)
    by (simp add: ipurge-tr-aux-single-dom sinks-aux-single-dom)
    hence \(x s \cong\left\{\right.\) ? ys, rev (ipurge-tr \(\left(I^{-1}\right) D u(\) rev \(x s)\) ),
        \(\lambda y\) ys. \(\left.D y \in \operatorname{sinks}\left(I^{-1}\right) D u(\operatorname{rev}(y \# y s))\right\}\)
        (is \(-\cong\{-, ? z s, ? P\}\) )
    by (simp only: sinks-interference-eq, simp)
    moreover have \(x s \cong\left\{\right.\) ? \(\mathrm{ys}^{\prime}\), ? ? \(\left.\mathrm{zs}, ? \mathrm{P}\right\}\) by (rule Interleaves-ipurge-tr)
    ultimately show ?thesis by (rule Interleaves-equal-fst)
qed
lemma ipurge-tr-all:
(ipurge-tr I D uxs \(=x s)=(\neg(\exists v \in D\) 'set \(x s .(u, v) \in I))\)
by (subst ipurge-tr-aux-single-dom [symmetric], simp add: ipurge-tr-aux-all)
lemma ipurge-tr-rev-all:
\(\forall v \in D '\) set \(x s .(v, u) \in I \Longrightarrow\) ipurge-tr-rev \(I D u x s=x s\)
proof (subst ipurge-tr-rev-aux-single-dom [symmetric], rule ipurge-tr-rev-aux-all)
qed (simp (no-asm-simp))

\subsection*{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 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-tr-rev I \(D u\) matches the identity function, so that two traces are correlated by the image of rel-ipurge under \(u\) just in case they are equal.
definition rel-ipurge ::
```

    'a process \(\Rightarrow\left({ }^{\prime} d \times{ }^{\prime} d\right)\) set \(\Rightarrow\left({ }^{\prime} a \Rightarrow{ }^{\prime} d\right) \Rightarrow\left({ }^{\prime} a,{ }^{\prime} d\right)\) dom-rel-map where
    rel-ipurge P I D $u \equiv\{(x s, y s)$. xs $\in$ traces $P \wedge y s \in$ traces $P \wedge$
ipurge-tr-rev I D uxs = ipurge-tr-rev IDuys\}
definition rel-ipurge-aux ::
'a process $\Rightarrow(' d \times ' d)$ set $\Rightarrow\left({ }^{\prime} a \Rightarrow{ }^{\prime} d\right) \Rightarrow\left({ }^{\prime} a,{ }^{\prime} d\right)$ domset-rel-map where
rel-ipurge-aux $P I D U \equiv\{(x s, y s)$. $x s \in$ traces $P \wedge y s \in$ traces $P \wedge$
ipurge-tr-rev-aux I D U xs = ipurge-tr-rev-aux I D U ys $\}$
lemma rel-ipurge-aux-single-dom:
rel-ipurge-aux P I D $\{u\}=$ rel-ipurge $P$ I D u
by (simp add: rel-ipurge-def rel-ipurge-aux-def ipurge-tr-rev-aux-single-dom)
lemma view-partition-rel-ipurge:
view-partition P D (rel-ipurge P I D)
proof (subst view-partition-def, rule ballI, rule equivI)
fix $u$
show refl-on (traces $P$ ) (rel-ipurge P I Du)
proof (rule refl-onI, simp-all add: rel-ipurge-def)
qed (rule subsetI, simp add: split-paired-all)
next
fix $u$
show sym (rel-ipurge P I D u)
by (rule symI, simp add: rel-ipurge-def)
next
fix $u$
show trans (rel-ipurge P I Du)
by (rule transI, simp add: rel-ipurge-def)
qed

```
lemma fc-equals-wfc-rel-ipurge:
    future-consistent \(P D(\) rel-ipurge \(P I D)=\)
    weakly-future-consistent PID (rel-ipurge P I D)
proof (rule iffI, erule fc-implies-wfc,
    simp only: future-consistent-def weakly-future-consistent-def,
    rule ballI, (rule allI)+, rule impI)
    fix \(u\) xs ys
    assume
        \(A: \forall u \in\) range \(D \cap(-I)\) " range \(D . \forall x s\) ys. \((x s, y s) \in\) rel-ipurge \(P I D u \longrightarrow\)
            next-dom-events \(P D u x s=\) next-dom-events \(P D u\) ys \(\wedge\)
            ref-dom-events \(P D u x s=\) ref-dom-events \(P D u y s\) and
        \(B: u \in\) range \(D\) and
```

    \(C:(x s, y s) \in\) rel-ipurge P I D u
    show next-dom-events $P D$ uss $=$ next-dom-events $P D$ uss $\wedge$
ref-dom-events $P D u x s=$ ref-dom-events $P D$ u ys
proof (cases $u \in$ range $D \cap(-I)$ " range $D$ )
case True
with $A$ have $\forall x s$ ys. $(x s, y s) \in$ rel-ipurge $P I D u \longrightarrow$
next-dom-events $P D u x s=$ next-dom-events $P D u$ ys $\wedge$
ref-dom-events $P D u x s=r e f-d o m-e v e n t s ~ P D u y s .$.
hence $(x s, y s) \in$ rel-ipurge P I D $u \longrightarrow$
next-dom-events $P D$ uxs = next-dom-events $P D$ u ys $\wedge$
ref-dom-events $P D u x s=$ ref-dom-events $P D u$ ys
by blast
thus ?thesis using $C$..
next
case False
hence $D: u \notin(-I)$ " 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 $\forall z s$. ipurge-tr-rev $I D u z s=z s$
proof (rule allI, rule ipurge-tr-rev-all, rule ballI, erule image $E$, rule ccontr)
fix $v x$
assume $(v, u) \notin I$
hence $(v, u) \in-I$ by $\operatorname{simp}$
moreover assume $v=D x$
hence $v \in$ range $D$ by simp
ultimately have $u \in(-I)$ " range $D$..
thus False using $D$ by contradiction
qed
ultimately show?thesis by simp
qed
qed

```

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

```
using \(A\).
thus ?thesis using \(C\)..

\section*{qed}
have \(E: \neg(\exists v \in \operatorname{sinks}\)-aux \(I D(D\) 'set ys) zs. \((v, D z) \in I) \longrightarrow\) ipurge-tr-rev-aux I D? \(U^{\prime}(x s\) @ ys @ zs) =
ipurge-tr-rev-aux I \(D\) ? \(U^{\prime}(x s\) @ ipurge-tr-aux I \(D(D\) 'set ys) zs)
\((\) is ? \(P \longrightarrow ? Q)\)
proof
assume ?P
have ? \(U^{\prime} \subseteq\) unaffected-domains \(I D(D\) 'set ys) zs \(\longrightarrow\) ipurge-tr-rev-aux I D? \(U^{\prime}(x s\) @ ys @ zs) = ipurge-tr-rev-aux I \(D\) ? \(U^{\prime}(x s @ i p u r g e-t r-a u x I D(D\) 'set ys) zs) using \(A\).
moreover have ? \(U^{\prime} \subseteq\) unaffected-domains \(I D(D\) ' set ys) zs by (simp add: C, simp add: unaffected-domains-def 〈?P〉[simplified])
ultimately show ?Q ..
qed
show ipurge-tr-rev-aux IDU(xs@ys @ zs @ [z])=
ipurge-tr-rev-aux I D \(U\) (xs @ ipurge-tr-aux I \(D(D\) 'set ys) \((z s @[z]))\)
proof (cases \(\exists v \in\) sinks-aux \(I D(D\) '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 \(D z \in\) sinks-aux \(I D(D\) 'set ys) (zs @ [z]) using True by simp with \(G\) have \(\exists v \in \operatorname{sinks}\)-aux I \(D(D\) 'set ys) \((z s @[z]) .(v, u) \in I .\).
moreover have \(u \in\) unaffected-domains \(I D(D\) 'set ys) (zs @ \([z])\)
using \(B\) and \(F\)..
hence \(\neg(\exists v \in\) sinks-aux \(I D(D\) 'set ys) \((z s @[z]) .(v, u) \in I)\)
by (simp add: unaffected-domains-def)
ultimately show False by contradiction
qed
hence ipurge-tr-rev-aux I D U ((xs @ys @zs) @ \([z])=\) ipurge-tr-rev-aux IDU(xs @ ys @ zs)
by (subst ipurge-tr-rev-aux-append, simp)
also have \(\ldots=\) ipurge-tr-rev-aux I D U
(xs @ ipurge-tr-aux I D (D'set ys) zs)
using \(D\).
finally show ipurge-tr-rev-aux IDU(xs@ys @zs @ \([z])=\) ipurge-tr-rev-aux I D \(U\) (xs @ ipurge-tr-aux I D (D'set ys)zs)
by \(\operatorname{simp}\)
next
case False
note \(F=\) this
show ipurge-tr-rev-aux ID \(U\) (xs @ ys @ zs @ \([z])=\)
ipurge-tr-rev-aux I D U (xs @ ipurge-tr-aux I D (D'set ys) zs @ [z])
proof (cases \(\exists u \in U .(D z, u) \in I)\)
```

        case True
        hence ipurge-tr-rev-aux ID U ((xs @ ys @ zs) @ [z])=
        ipurge-tr-rev-aux I D? ? U'(xs @ ys @ zs) @ [z]
        by (subst ipurge-tr-rev-aux-append, simp)
        also have ... =
            ipurge-tr-rev-aux I D? 'U'(xs@ ipurge-tr-aux I D(D'set ys)zs)@ [z]
            using E and F by simp
            also have ... =
            ipurge-tr-rev-aux I D U ((xs @ ipurge-tr-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-tr-rev-aux I D U ((xs @ ys @ zs)@ [z])=
            ipurge-tr-rev-aux I D U (xs @ ys @ zs)
            by (subst ipurge-tr-rev-aux-append, simp)
            also have ...=
                ipurge-tr-rev-aux I D U(xs @ ipurge-tr-aux ID(D'set ys)zs)
            using D .
            also have ...=
                ipurge-tr-rev-aux I D U ((xs @ ipurge-tr-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-domains I D(D'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'set ys) zs)
proof (induction zs arbitrary: U rule: rev-induct, rule-tac [!] impI, simp)
fix }
assume A:U\subsetequnaffected-domains I D (D'set ys) []
have }\forallu\inU.\forallv\inD' set ys. (v,u)\not\in
proof
fix u
assume }u\in
with A have u\in unaffected-domains I D (D'set ys) [] ..
thus }\forallv\inD'set ys. (v,u)\not\inI by (simp add: unaffected-domains-def
qed
hence ipurge-tr-rev-aux I D U ys = [] by (simp add: ipurge-tr-rev-aux-nil)
hence ipurge-tr-rev-aux I D U (xs @ ys)= ipurge-tr-rev-aux I D U xs
by (simp add: ipurge-tr-rev-aux-append-nil)
thus ipurge-tr-rev-aux I D U xs = ipurge-tr-rev-aux I D U (xs @ ys)..
next
fix z zs U
let ? U' U' insert (Dz)U
assume

```
```

    A: \(\wedge U . U \subseteq\) unaffected-domains I \(D(D\) 'set ys) zs \(\longrightarrow\)
        ipurge-tr-rev-aux I D U (xs @ zs) =
        ipurge-tr-rev-aux ID U (xs @ ys @ ipurge-tr-aux I D ( \(D\) 'set ys) zs) and
    \(B: U \subseteq\) unaffected-domains \(I D(D\) 'set ys) \((z s @[z])\)
    have $C: U \subseteq$ unaffected-domains $I D(D$ 'set ys) zs
proof
fix $u$
assume $u \in U$
with $B$ have $u \in$ unaffected-domains $I D(D$ 'set ys) (zs @ [z])..
thus $u \in$ unaffected-domains $I D(D$ ' set ys) zs
by (simp add: unaffected-domains-def)
qed
have $D$ : ipurge-tr-rev-aux I D $U(x s$ @ zs)=
ipurge-tr-rev-aux I D $U$ (xs @ ys @ ipurge-tr-aux ID (D'set ys) zs)
proof -
have $U \subseteq$ unaffected-domains I $D(D$ '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 ID (D'set ys) zs)
using $A$.
thus ?thesis using $C$..
qed
have $E: \neg(\exists v \in$ sinks-aux $I D(D$ 'set ys) zs. $(v, D z) \in I) \longrightarrow$
ipurge-tr-rev-aux I D? $U^{\prime}(x s$ @ zs) =
ipurge-tr-rev-aux I D? $U^{\prime}(x s$ @ ys @ ipurge-tr-aux I D (D'set ys) zs)
$($ is ? $P \longrightarrow$ ? $Q$ )
proof
assume ?P
have ? $U^{\prime} \subseteq$ unaffected-domains I $D(D$ 'set $y s) z s \longrightarrow$
ipurge-tr-rev-aux I D? $U^{\prime}(x s @ z s)=$
ipurge-tr-rev-aux I D? $U^{\prime}(x s @ y s$ @ ipurge-tr-aux ID (D'set ys) zs)
using $A$.
moreover have ? $U^{\prime} \subseteq$ unaffected-domains $I D(D$ ' set ys) zs
by (simp add: C, simp add: unaffected-domains-def 〈?P〉[simplified])
ultimately show? Q ..
qed
show ipurge-tr-rev-aux ID $\mathrm{U}(x s @ z s @[z])=$
ipurge-tr-rev-aux I D $U$ (xs @ ys @ ipurge-tr-aux I D (D'set ys) (zs @ [z]))
proof (cases $\exists v \in$ sinks-aux I $D(D$ '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 $D z \in$ sinks-aux $I D(D$ 'set ys) (zs @ [z]) using True by simp
with $G$ have $\exists v \in \operatorname{sinks-aux~I~} D(D$ 'set $y s)(z s @[z]) .(v, u) \in I .$.
moreover have $u \in$ unaffected-domains $I D(D$ 'set ys) (zs @ $[z])$
using $B$ and $F$..
hence $\neg(\exists v \in$ sinks-aux I $D(D$ ‘set $y s)(z s @[z]) .(v, u) \in I)$

```
```

        by (simp add: unaffected-domains-def)
        ultimately show False by contradiction
    qed
    hence ipurge-tr-rev-aux I D U ((xs @ zs) @ [z])=
        ipurge-tr-rev-aux I D U (xs @ zs)
    by (subst ipurge-tr-rev-aux-append, simp)
    also have
        ...= ipurge-tr-rev-aux IDU(xs @ ys @ ipurge-tr-aux I D(D`set ys)zs)
    using D.
    finally show ipurge-tr-rev-aux I D U (xs @ zs @ [z])=
        ipurge-tr-rev-aux I D U (xs @ ys @ ipurge-tr-aux ID (D'set ys)zs)
    by simp
    next
case False
note F= this
show ipurge-tr-rev-aux IDU(xs @ zs @ [z])=
ipurge-tr-rev-aux I D U (xs@ ys @ ipurge-tr-aux I D (D` set ys)zs @ [z])     proof (cases \existsu\inU.(Dz,u)\inI)         case True         hence ipurge-tr-rev-aux I D U ((xs @ zs) @ [z])=             ipurge-tr-rev-aux I D ? U' (xs @ zs) @ [z]             by (subst ipurge-tr-rev-aux-append, simp)             also have ...=                 ipurge-tr-rev-aux I D ? U'                 (xs @ ys @ ipurge-tr-aux I D (D'set ys) zs)@ [z]             using E and F by simp             also have ...=                 ipurge-tr-rev-aux I D U                 ((xs@ ys @ ipurge-tr-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-tr-rev-aux I D U ((xs @ zs) @ [z])=
ipurge-tr-rev-aux I D U (xs @ zs)
by (subst ipurge-tr-rev-aux-append, simp)
also have ... =
ipurge-tr-rev-aux I D U (xs @ ys @ ipurge-tr-aux I D (D'set ys)zs)
using D.
also have ... =
ipurge-tr-rev-aux I D U
((xs@ ys @ ipurge-tr-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-1:

```
assumes \(A: u \in\) unaffected-domains \(I D\{D y\} z s\)
shows ipurge-tr-rev I Du(xs @ y \#zs)=
ipurge-tr-rev I \(D u(x s @ i p u r g e-t r I D(D y) z s)\)
proof -
have ipurge-tr-rev I Du(xs @y\#zs)= ipurge-tr-rev-aux I D \(\{u\}\) (xs @ \([y]\) @ zs)
by (simp add: ipurge-tr-rev-aux-single-dom)
also have \(\ldots=\) ipurge-tr-rev-aux I \(D\{u\}\)
(xs @ ipurge-tr-aux I D (D'set [y]) zs)
by (rule ipurge-tr-rev-ipurge-tr-aux-1, simp add: A)
also have \(\ldots=\) ipurge-tr-rev I \(D u(x s @ i p u r g e-t r I D(D y) z s)\)
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 \(A: u \in\) unaffected-domains \(I D\{D y\} z s\)
shows ipurge-tr-rev IDu(xs@zs)=
ipurge-tr-rev I Du(xs @ \(y\) \# ipurge-tr I D (D y) zs)
proof -
have ipurge-tr-rev I Du(xs @ zs)=ipurge-tr-rev-aux ID\{u\}(xs@zs)
by (simp add: ipurge-tr-rev-aux-single-dom)
also have
\(\ldots=\) ipurge-tr-rev-aux I \(D\{u\}(x s @[y] @ i p u r g e-t r-a u x I D(D ‘ s e t[y]) z s)\)
by (rule ipurge-tr-rev-ipurge-tr-aux-2, simp add: A)
also have \(\ldots=\) ipurge-tr-rev I \(D u(x s @ y \#\) ipurge-tr I \(D(D y) z s)\)
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: ref-union-closed \(P\) and
IU: weakly-future-consistent P I D (rel-ipurge P I D) and
\(A:(x s @ y \# y s, Y) \in\) failures \(P\) and
\(B\) : xs @ ipurge-tr I \(D(D y)\) ys \(\in\) traces \(P\) and
\(C: \exists y^{\prime} \cdot y^{\prime} \in\) ipurge-ref \(I D(D y)\) ys \(Y\)
shows (xs @ ipurge-tr I \(D(D y)\) ys, ipurge-ref \(I D(D y)\) ys \(Y) \in\) failures \(P\)
proof -
let ? \(A=\) singleton-set (ipurge-ref I \(D(D y)\) ys \(Y)\)
have \((\exists X . X \in ? A) \longrightarrow\)
\((\forall X \in\) ? \(A\). (xs @ ipurge-tr I \(D(D y) y s, X) \in\) failures \(P) \longrightarrow\)
(xs @ ipurge-tr I D (D y) ys, \(\bigcup X \in ? A . X) \in\) failures \(P\)
using \(R U C\) by (simp add: ref-union-closed-def)
moreover obtain \(y^{\prime}\) where \(D: y^{\prime} \in\) ipurge-ref \(I D(D y)\) ys \(Y\) using \(C\)..
hence \(\exists X . X \in ?\) ? by (simp add: singleton-set-some, rule exI)
ultimately have \((\forall X \in\) ? A. (xs @ ipurge-tr \(I D(D y)\) ys, \(X) \in\) failures \(P) \longrightarrow\) (xs @ ipurge-tr I \(D(D y) y s, \bigcup X \in ? A . X) \in\) failures \(P .\).
moreover have \(\forall X \in\) ?A. (xs @ ipurge-tr \(I D(D y) y s, X) \in\) failures \(P\)
proof (rule ballI, simp add: singleton-set-def, erule bexE, simp)
fix \(y^{\prime}\)
have \(\forall u \in\) range \(D \cap(-I)\) " range \(D\).
\(\forall x s\) ys. \((x s, y s) \in\) rel-ipurge P I Du \(\longrightarrow\) ref-dom-events \(P D u x s=r e f-d o m-e v e n t s ~ P ~ d u s\)
using \(I U\) by (simp add: weakly-future-consistent-def)
moreover assume \(E: y^{\prime} \in\) ipurge-ref \(I D(D y)\) ys \(Y\)
hence \(\left(D y, D y^{\prime}\right) \notin I\) by (simp add: ipurge-ref-def)
hence \(D y^{\prime} \in\) range \(D \cap(-I)\) " range \(D\) by (simp add: Image-iff, rule exI)
ultimately have \(\forall x s\) ys. \((x s, y s) \in\) rel-ipurge \(P I D\left(D y^{\prime}\right) \longrightarrow\)
ref-dom-events \(P D\left(D y^{\prime}\right) x s=r e f-d o m-e v e n t s ~ P D\left(D y^{\prime}\right)\) ys ..
hence
\(F:(x s\) @ \(y \# y s, x s @ i p u r g e-t r I D(D y) y s) \in\) rel-ipurge P I D \(\left(D y^{\prime}\right) \longrightarrow\)
ref-dom-events \(P D\left(D y^{\prime}\right)(x s @ y \# y s)=\)
ref-dom-events P D (D y') (xs @ ipurge-tr I D (D y) ys)
by blast
have \(y^{\prime} \in\{x \in Y . D x \in\) unaffected-domains \(I D\{D y\} y s\}\)
using \(E\) by (simp add: unaffected-domains-single-dom)
hence \(D y^{\prime} \in\) unaffected-domains \(I D\{D y\}\) ys by simp
hence ipurge-tr-rev I D ( \(D y^{\prime}\) ) (xs @ y \# ys \()=\)
ipurge-tr-rev I \(D\left(D y^{\prime}\right)(x s @ i p u r g e-t r I D(D y) y s)\)
by (rule ipurge-tr-rev-ipurge-tr-1)
moreover have \(x s\) @ \(y \# y s \in\) traces \(P\) using \(A\) by (rule failures-traces)
ultimately have
\((x s @ y \# y s, x s\) @ ipurge-tr I D \((D y) y s) \in\) rel-ipurge P I D \(\left(D y^{\prime}\right)\)
using \(B\) by (simp add: rel-ipurge-def)
with \(F\) have ref-dom-events \(P D\left(D y^{\prime}\right)(x s @ y \# y s)=\) ref-dom-events \(P D\left(D y^{\prime}\right)(x s\) @ ipurge-tr \(I D(D y) y s)\)..
moreover have \(y^{\prime} \in\) ref-dom-events \(P D\left(D y^{\prime}\right)(x s @ y \# y s)\)
proof (simp add: ref-dom-events-def refusals-def)
have \(\left\{y^{\prime}\right\} \subseteq Y\) using \(E\) by (simp add: ipurge-ref-def)
with \(A\) show \(\left(x s @ y \# y s,\left\{y^{\prime}\right\}\right) \in\) failures \(P\) by (rule process-rule-3)
qed
ultimately have \(y^{\prime} \in\) ref-dom-events \(P D\left(D y^{\prime}\right)\)
(xs @ ipurge-tr I D (D y) ys)
by simp
thus (xs @ ipurge-tr I D (Dy) ys, \(\left.\left\{y^{\prime}\right\}\right) \in\) failures \(P\)
by (simp add: ref-dom-events-def refusals-def)
qed
ultimately have (xs @ ipurge-tr I \(D(D y) y s, \bigcup X \in ? A . X) \in\) 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
\(I U\) : weakly-future-consistent P I D (rel-ipurge P I D) and
A: \((x s @ z s, Z) \in\) failures \(P\) and
B: xs @y \# ipurge-tr I D (Dy) zs \(\in\) traces \(P\) and
\(C: \exists z^{\prime} . z^{\prime} \in\) ipurge-ref \(I D(D y)\) zs \(Z\)
shows \((x s\) @ \(y \#\) ipurge-tr \(I D(D y) z s\), ipurge-ref \(I D(D y) z s Z) \in\) failures \(P\) proof -
let ? \(A=\) singleton-set (ipurge-ref \(I D(D y)\) zs \(Z)\)
have \((\exists X . X \in ? A) \longrightarrow\)
\((\forall X \in\) ? \(A\). ( \(x s\) @ \(y\) \# ipurge-tr I \(D(D y) z s, X) \in\) failures \(P) \longrightarrow\)
(xs @y \# ipurge-tr \(I D(D y) z s, \bigcup X \in ? A . X) \in\) failures \(P\)
using \(R U C\) by (simp add: ref-union-closed-def)
moreover obtain \(z^{\prime}\) where \(D: z^{\prime} \in\) ipurge-ref \(I D(D y) z s Z\) using \(C\)..
hence \(\exists X . X \in\) ? \(A\) by (simp add: singleton-set-some, rule exI)
ultimately have
\((\forall X \in\) ? \(A\). ( \(x s\) @ \(y\) \# ipurge-tr \(I D(D y) z s, X) \in\) failures \(P) \longrightarrow\) ( \(x s\) @ \(y\) \# ipurge-tr \(I D(D y) z s, \bigcup X \in ? A . X) \in\) failures \(P\)..
moreover have \(\forall X \in\) ?A. (xs @ \(y \#\) ipurge-tr \(I D(D y) z s, X) \in\) failures \(P\)
proof (rule ballI, simp add: singleton-set-def, erule bexE, simp)
fix \(z^{\prime}\)
have \(\forall u \in\) range \(D \cap(-I)\) " range \(D\).
\(\forall x s\) ys. \((x s, y s) \in\) rel-ipurge P I Du \(\longrightarrow\)
ref-dom-events \(P D u x s=\) ref-dom-events \(P D u\) ys
using \(I U\) by (simp add: weakly-future-consistent-def)
moreover assume \(E: z^{\prime} \in\) ipurge-ref \(I D(D y)\) zs \(Z\)
hence \(\left(D y, D z^{\prime}\right) \notin I\) by (simp add: ipurge-ref-def)
hence \(D z^{\prime} \in\) range \(D \cap(-I)\) " range \(D\) by (simp add: Image-iff, rule exI)
ultimately have \(\forall x s\) ys. \((x s, y s) \in\) rel-ipurge \(P I D\left(D z^{\prime}\right) \longrightarrow\)
ref-dom-events \(P D\left(D z^{\prime}\right)\) xs \(=\) ref-dom-events \(P D\left(D z^{\prime}\right)\) ys ..

\section*{hence}
\(F:(x s @ z s, x s @ y \#\) ipurge-tr \(I D(D y) z s) \in\) rel-ipurge P I D \(\left(D z^{\prime}\right) \longrightarrow\)
ref-dom-events \(P D\left(D z^{\prime}\right)(x s @ z s)=\)
ref-dom-events \(P D\left(D z^{\prime}\right)(x s @ y \#\) ipurge-tr \(I D(D y) z s)\)
by blast
have \(z^{\prime} \in\{x \in Z . D x \in\) unaffected-domains \(I D\{D y\} z s\}\)
using \(E\) by (simp add: unaffected-domains-single-dom)
hence \(D z^{\prime} \in\) unaffected-domains \(I D\{D y\} z s\) by simp
hence ipurge-tr-rev I \(D\left(D z^{\prime}\right)(x s @ z s)=\)
ipurge-tr-rev I \(D\left(D z^{\prime}\right)\left(x s @ y \#\right.\) ipurge-tr \(I D\left(\begin{array}{l}D \\ \text { ) }\end{array}\right.\) zs)
by (rule ipurge-tr-rev-ipurge-tr-2)
moreover have xs @ zs \(\in\) traces \(P\) using \(A\) by (rule failures-traces)
ultimately have
\((x s @ z s, x s @ y \#\) ipurge-tr I \(D(D y) z s) \in\) rel-ipurge PID \(\left(D z^{\prime}\right)\)
using \(B\) by (simp add: rel-ipurge-def)
with \(F\) have ref-dom-events \(P D\left(D z^{\prime}\right)(x s @ z s)=\) ref-dom-events \(P D\left(D z^{\prime}\right)(x s\) @ \(y\) \# ipurge-tr I D (D y) zs) ..
moreover have \(z^{\prime} \in\) ref-dom-events \(P D\left(D z^{\prime}\right)(x s @ z s)\)
proof (simp add: ref-dom-events-def refusals-def)
have \(\left\{z^{\prime}\right\} \subseteq Z\) using \(E\) by (simp add: ipurge-ref-def)
with \(A\) show \(\left(x s @ z s,\left\{z^{\prime}\right\}\right) \in\) failures \(P\) by (rule process-rule-3)
qed
ultimately have \(z^{\prime} \in\) ref-dom-events \(P D\left(D z^{\prime}\right)\)
(xs @ y \# ipurge-tr I D (D y) zs)
```

        by simp
        thus (xs@ @ # ipurge-tr I D (D y) zs, {z'}) \in failures P
        by (simp add: ref-dom-events-def refusals-def)
    qed
    ultimately have
    (xs@ @ # ipurge-tr I D (D y) zs, \bigcupX \in?A. X) \in failures P ..
    thus ?thesis by (simp only: singleton-set-union)
    qed
lemma iu-condition-imply-secure-1 [rule-format]:
assumes
RUC: ref-union-closed P and
IU:weakly-future-consistent P I D (rel-ipurge P I D)
shows (xs@y\#ys,Y)\in failures P\longrightarrow
(xs@ ipurge-tr I D (D y) ys, ipurge-ref I D (D y) ys Y) f failures P
proof (induction ys arbitrary: Y rule: rev-induct, rule-tac [!] impI)
fix Y
assume A: (xs@ [y],Y)\in failures P
show (xs @ ipurge-tr I D (D y) [], ipurge-ref I D (D y) [] Y) f failures P
proof (cases \exists}\mp@subsup{y}{}{\prime}.\mp@subsup{y}{}{\prime}\in\mathrm{ ipurge-ref I D (D y) [] Y)
case True
have xs @ [y]\in traces P using A by (rule failures-traces)
hence xs \in traces P by (rule process-rule-2-traces)
hence xs @ ipurge-tr I D (D y) [] \in traces P by simp
with RUC and IU and A show ?thesis
using True by (rule iu-condition-imply-secure-aux-1)
next
case False
moreover have (xs, {}) \in failures P using A by (rule process-rule-2)
ultimately show ?thesis by simp
qed
next
fix }\mp@subsup{y}{}{\prime}\mathrm{ ys Y
assume
A: \bigwedge}\mp@subsup{Y}{}{\prime}.(xs@y\#ys,\mp@subsup{Y}{}{\prime})\in\mathrm{ failures }P
(xs@ ipurge-tr I D (D y) ys, ipurge-ref I D (D y) ys Y') f failures P and
B:(xs@y \# ys @ [y],Y) \in failures P
have (xs @ y\# ys,{}) \in failures P\longrightarrow
(xs@ ipurge-tr I D (D y) ys, ipurge-ref I D (D y) ys {}) f failures P
(is - \longrightarrow(-, ? Y') \in-)
using A .
moreover have ((xs@y\#ys)@ @ y ],Y)\in failures P using B by simp
hence C: (xs @ y \# ys,{})\in failures P by (rule process-rule-2)
ultimately have (xs @ ipurge-tr I D (D y) ys,? 'Y') f failures P ..
moreover have {}\subseteq ? Y' ..
ultimately have D:(xs @ ipurge-tr I D (D y) ys, {}) f failures P
by (rule process-rule-3)
have E:xs@ ipurge-tr I D (D y) (ys @ [y]) \in traces P
proof (cases D y' \in sinks I D (D y) (ys @ [y]))

```
```

    case True
    hence (xs @ ipurge-tr I D (D y) (ys @ [y|), {}) \in failures P using D by simp
    thus ?thesis by (rule failures-traces)
    next
case False
have }\forallu\in\mathrm{ range D }\cap(-I) " range D
\foralls ys. (xs, ys) \in rel-ipurge P I Du\longrightarrow
next-dom-events P D u xs = next-dom-events P D u ys
using IU by (simp add: weakly-future-consistent-def)
moreover have (D y,D y')\not\inI
using False by (simp add: sinks-interference-eq [symmetric] del: sinks.simps)
hence D y' \in range D \cap (-I) " range D by (simp add: Image-iff, rule exI)
ultimately have }\forallxs ys. (xs,ys)\in rel-ipurge PID (D y')
next-dom-events P D (D y') xs = next-dom-events P D (D y') ys ..
hence
F:(xs @ y \# ys,xs @ ipurge-tr I D (D y) ys) \in rel-ipurge P I D (D y')}
next-dom-events P D (D y') (xs @ y \# ys)=
next-dom-events P D (D y')(xs @ ipurge-tr I D (D y) ys)
by blast
have }\forallv\in\operatorname{insert ( D y) (sinks I D (D y) ys). (v,D y')\not\inI
using False by (simp add: sinks-interference-eq [symmetric] del: sinks.simps)
hence }\forallv\in\mathrm{ sinks-aux I D{D y} ys. (v,D y')}\not\in
by (simp add: sinks-aux-single-dom)
hence D y'}\in\mathrm{ unaffected-domains I D{D y} ys
by (simp add: unaffected-domains-def)
hence ipurge-tr-rev I D (D y') (xs @ y \# ys)=
ipurge-tr-rev I D (D y')(xs @ ipurge-tr I D (D y)ys)
by (rule ipurge-tr-rev-ipurge-tr-1)
moreover have xs @ y \# ys \in traces P using C by (rule failures-traces)
moreover have xs @ ipurge-tr I D (D y) ys \in traces P
using D by (rule failures-traces)
ultimately have
(xs @ y \# ys, xs @ ipurge-tr I D (D y) ys) \inrel-ipurge P I D (D y')
by (simp add: rel-ipurge-def)
with F have next-dom-events P D (D y')(xs @ y \# ys)=
next-dom-events P D (D y') (xs @ ipurge-tr I D (D y) ys) ..
moreover have }\mp@subsup{y}{}{\prime}\in\mathrm{ 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 }\mp@subsup{y}{}{\prime}\in\mathrm{ next-dom-events P D (D y')
(xs @ ipurge-tr I D (D y) ys)
by simp
hence xs @ ipurge-tr I D (D y) ys @ [y] t traces P
by (simp add: next-dom-events-def next-events-def)
thus ?thesis using False by simp
qed
show (xs @ ipurge-tr I D (D y) (ys @ [y]), ipurge-ref I D (D y) (ys @ [y])Y)
failures P
proof (cases \existsx.x ipurge-ref I D (D y)(ys @ [y])Y)

```
```

    case True
    with RUC and IU and B and E show ?thesis by (rule iu-condition-imply-secure-aux-1)
    next
    case False
    moreover have (xs @ ipurge-tr I D (D y) (ys @ [y ]), {}) \in failures P
        using E by (rule traces-failures)
    ultimately show ?thesis by simp
    qed
    qed
lemma iu-condition-imply-secure-2 [rule-format]:
assumes
RUC: ref-union-closed P and
IU:weakly-future-consistent P I D (rel-ipurge P I D) and
Y:xs@ @ [y]\in traces P
shows (xs @ zs,Z) failures P}
(xs @ y \# ipurge-tr I D (D y) zs, ipurge-ref I D (D y) zs Z) \in failures P
proof (induction zs arbitrary: Z rule: rev-induct, rule-tac [!] impI)
fix }
assume A: (xs @ [], Z) \in failures P
show (xs @ y \# ipurge-tr I D (D y) [], ipurge-ref I D (D y) [] Z) f failures P
proof (cases \exists\mp@subsup{z}{}{\prime}.\mp@subsup{z}{}{\prime}\in\mathrm{ ipurge-ref I D (D y) [] Z)}
case True
have xs@y \# ipurge-tr I D (D y) [] \in traces P using Y by simp
with RUC and IU and A show ?thesis
using True by (rule iu-condition-imply-secure-aux-2)
next
case False
moreover have (xs @ [y],{}) \in failures P using Y by (rule traces-failures)
ultimately show ?thesis by simp
qed
next
fix zzs Z
assume
A:\Z.(xs@zs,Z)\in failures P\longrightarrow
(xs@y \# ipurge-tr I D (D y) zs, ipurge-ref I D (D y) zs Z) f failures P and
B:(xs@ zs @ [z], Z) \in failures P
have (xs @ zs, {}) \in failures P\longrightarrow
(xs@y \# ipurge-tr I D (D y) zs, ipurge-ref I D (D y) zs {}) f failures P
(is -\longrightarrow(-, ?Z')\in-)
using }A\mathrm{ .
moreover have ((xs @ zs) @ [z],Z) \in failures P using B by simp
hence C:(xs @zs,{})\in failures P by (rule process-rule-2)
ultimately have (xs @ y \# ipurge-tr I D (D y) zs,? 'Z') \in failures P ..
moreover have {} \subseteq? 'Z' ..
ultimately have D:(xs @ y \# ipurge-tr I D (D y) zs, {}) \in failures P
by (rule process-rule-3)
have E:xs@ y \# ipurge-tr I D (D y) (zs@ [z]) \in traces P
proof (cases D z f sinks I D (D y)(zs @ [z]))

```
case True
hence (xs @ y \# ipurge-tr ID \((D y)(z s\) @ \([z]),\{ \}) \in\) failures \(P\)
using \(D\) by simp
thus ?thesis by (rule failures-traces)
next
case False
have \(\forall u \in\) range \(D \cap(-I)\) " range \(D\).
\(\forall x s\) ys. \((x s, y s) \in\) rel-ipurge \(P I D u \longrightarrow\)
next-dom-events \(P D u x s=\) next-dom-events \(P D\) u ys
using \(I U\) 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 \(D z \in\) range \(D \cap(-I)\) " range \(D\) by (simp add: Image-iff, rule exI)
ultimately have \(\forall x s\) ys. \((x s, y s) \in\) rel-ipurge \(P I D(D z) \longrightarrow\)
next-dom-events \(P\) D \(\left(\begin{array}{l}z\end{array}\right)\) xs \(=\) next-dom-events \(P D(D z)\) ys ..
hence
\(F:(x s @ z s, x s @ y \#\) ipurge-tr \(I D(D y) z s) \in\) rel-ipurge P I D \((D z) \longrightarrow\) next-dom-events \(P D(D z)(x s @ z s)=\) next-dom-events \(P D(D z)(x s @ y \#\) ipurge-tr \(I D(D y) z s)\)
by blast
have \(\forall v \in \operatorname{insert}(D y)(\operatorname{sinks} I D(D y) z s) .(v, D z) \notin I\)
using False by (simp add: sinks-interference-eq [symmetric] del: sinks.simps)
hence \(\forall v \in\) sinks-aux I \(D\{D y\} z s .(v, D z) \notin I\)
by (simp add: sinks-aux-single-dom)
hence \(D z \in\) unaffected-domains \(I D\{D y\} z s\)
by (simp add: unaffected-domains-def)
hence ipurge-tr-rev \(I D(D z)(x s @ z s)=\)
ipurge-tr-rev I \(D\left(\begin{array}{l}D \\ )\end{array}\right.\) (xs @ y \# ipurge-tr I \(\left.D\binom{D}{y} z s\right)\)
by (rule ipurge-tr-rev-ipurge-tr-2)
moreover have \(x s @ z s \in\) traces \(P\) using \(C\) by (rule failures-traces)
moreover have xs @y \# ipurge-tr I D (D y) zs \(\in\) traces \(P\)
using \(D\) by (rule failures-traces)
ultimately have
\((x s @ z s, x s\) @ \(y \#\) ipurge-tr I D \((D y) z s) \in\) rel-ipurge P I D (Dz)
by (simp add: rel-ipurge-def)
with \(F\) have next-dom-events \(P D(D z)(x s @ z s)=\)
next-dom-events \(P D(D z)(x s @ y \#\) ipurge-tr \(I D(D y) z s)\)..
moreover have \(z \in\) next-dom-events \(P D\left(\begin{array}{l}D \\ )\end{array}(x s\right.\) @ \(z s)\)
proof (simp add: next-dom-events-def next-events-def)
qed (rule failures-traces \([\) OF B])
ultimately have \(z \in\) next-dom-events \(P D(D z)\)
(xs @ y \# ipurge-tr I D (D y) zs)
by \(\operatorname{simp}\)
hence \(x s\) @ \(y\) \# ipurge-tr I \(D(D y) z s @[z] \in \operatorname{traces} P\)
by (simp add: next-dom-events-def next-events-def)
thus ?thesis using False by simp
qed
show (xs @ y \# ipurge-tr I D (Dy) (zs @ [z]),
ipurge-ref I D (Dy) (zs @ [z]) Z)
```

    failures P
    proof (cases \existsx.x\in ipurge-ref I D(Dy)(zs@ [z])Z)
    case True
    with RUC and IU and B and E show ?thesis by (rule iu-condition-imply-secure-aux-2)
    next
        case False
        moreover have (xs @ y # ipurge-tr I D (D y) (zs @ [z]), {}) \in failures P
        using E by (rule 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 (rel-ipurge P I D)
shows secure P I D
proof (simp add: secure-def futures-def,(rule allI)+, rule impI, erule conjE)
fix xs y ys Yzs Z
assume
A:(xs@ y \# ys,Y) failures P and
B:(xs@ zs,Z) f failures P
show (xs @ ipurge-tr I D (D y) ys, ipurge-ref I D (D y) ys Y) f failures P ^
(xs@y\# ipurge-tr I D (Dy)zs,ipurge-ref I D (D y) zs Z) f failures P
(is ?P}\wedge?Q
proof
show ?P using RUC and IU and A by (rule iu-condition-imply-secure-1)
next
have ((xs @ [y])@ ys,Y)\in failures P using A by simp
hence (xs @ [y], {}) \in failures P by (rule process-rule-2-failures)
hence xs @ [y]\in traces P by (rule failures-traces)
with RUC and IU show ?Q using B by (rule iu-condition-imply-secure-2)
qed
qed

```

\subsection*{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 [rule-format]:
assumes \(S\) : secure P I D
shows (xs @ ys @ zs, X) \(\in\) failures \(P \longrightarrow\) ipurge-tr-rev-aux \(I D\left(D^{\prime}(X \cup\right.\) set zs \(\left.)\right)\) ys \(=[] \longrightarrow(x s @ z s, X) \in\) failures \(P\)
proof (induction ys rule: rev-induct, simp-all, (rule impI)+)
fix \(y\) ys
```

    assume \(*\) : ipurge-tr-rev-aux \(I D\left(D^{\prime}(X \cup\right.\) set zs \(\left.)\right)(y s @[y])=[]\)
    then have \(A: \neg\left(\exists v \in D^{\prime}(X \cup\right.\) set \(\left.z s) .(D y, v) \in I\right)\)
    by (cases \(\exists v \in D^{\prime}(X \cup\) set \(z s) .(D y, v) \in I\),
    simp-all add: ipurge-tr-rev-aux-append)
    with $*$ have $B$ : ipurge-tr-rev-aux I $D\left(D^{\prime}(X \cup\right.$ set zs $\left.)\right)$ ys $=[]$
by (simp add: ipurge-tr-rev-aux-append)
assume (xs @ys @y $\# z s, X) \in$ failures $P$
hence $(y \# z s, X) \in$ futures $P(x s$ @ ys) by (simp add: futures-def)
hence (ipurge-tr I $D(D y)$ zs, ipurge-ref I $D(D y)$ zs $X$ )
$\in$ futures $P$ (xs @ ys)
using $S$ by (simp add: secure-def)
moreover have ipurge-tr $I D(D y) z s=z s$ using $A$ by (simp add: ipurge-tr-all)
moreover have ipurge-ref $I D(D y) z s X=X$ using $A$ by (rule ipurge-ref-all)
ultimately have $(z s, X) \in$ futures $P(x s$ @ ys) by simp
hence $C:(x s @ y s @ z s, X) \in$ failures $P$ by (simp add: futures-def)
assume (xs @ ys @zs, X) $\in$ failures $P \longrightarrow$
ipurge-tr-rev-aux I D $\left(D^{\prime}(X \cup\right.$ set zs $\left.)\right)$ ys $=[] \longrightarrow$
$(x s @ z s, X) \in$ failures $P$
hence ipurge-tr-rev-aux $\operatorname{ID}\left(D^{\prime}(X \cup\right.$ set zs)) ys $=[] \longrightarrow$
(xs@zs, X) $\in$ failures $P$
using $C$..
thus $(x s$ @ $z s, X) \in$ failures $P$ using $B$..
qed
lemma secure-implies-failure-consistency [rule-format]:
assumes $S$ : secure P I D
shows $(x s, y s) \in$ rel-ipurge-aux P I D $\left(D^{\prime}(X \cup\right.$ set $\left.z s)\right) \longrightarrow$
$(x s @ z s, X) \in$ failures $P \longrightarrow(y s @ z s, X) \in$ 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 $x s$ zs
assume (xs @ zs, X) $\in$ failures $P$
hence ([] @ xs @ zs, X) $\in$ failures $P$ by simp
moreover assume ipurge-tr-rev-aux I $D\left(D^{\prime}(X \cup\right.$ set zs $\left.)\right)$ xs $=[]$
ultimately have ([]@zs, X) $\in$ failures $P$
using $S$ by (rule-tac secure-implies-failure-consistency-aux)
thus $(z s, X) \in$ failures $P$ by simp
next
fix $y$ ys $x s z s$
assume
A: $\bigwedge x s^{\prime} z s^{\prime} . x s^{\prime} \in$ traces $P \wedge y s \in$ traces $P \wedge$
ipurge-tr-rev-aux I $D\left(D^{\prime}\left(X \cup\right.\right.$ set $\left.\left.z s^{\prime}\right)\right) x s^{\prime}=$
ipurge-tr-rev-aux I $D\left(D^{\prime}(X \cup\right.$ set zs')) ys $\longrightarrow$
$\left(x s^{\prime} @ z s^{\prime}, X\right) \in$ failures $P \longrightarrow\left(y s @ z s^{\prime}, X\right) \in$ failures $P$ and
$B:(x s @ z s, X) \in$ failures $P$ and
$C: x s \in$ traces $P$ and
$D: y s @[y] \in$ traces $P$ and
E: ipurge-tr-rev-aux I $D\left(D^{\prime}(X \cup\right.$ set zs $\left.)\right)$ xs $=$
ipurge-tr-rev-aux I D $\left(D^{\prime}(X \cup\right.$ set zs $\left.)\right)(y s$ @ $[y])$

```
```

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

```
```

        ipurge-tr-rev-aux \(I D(D\) ‘ \((X \cup\) set \(z s)) x s=\)
        ipurge-tr-rev-aux I D \(\left(D^{\prime}(X \cup\right.\) set zs \(\left.)\right)\) ys \(\longrightarrow\)
        \((x s @ z s, X) \in\) failures \(P \longrightarrow(y s\) @ zs, X) \(\in\) failures \(P\)
        using \(A\).
        ultimately have (ys @ zs, \(X\) ) \(\in\) failures \(P\) using \(B\) and \(C\) by simp
        hence \((z s, X) \in\) futures \(P\) ys by (simp add: futures-def)
        moreover have \(\exists Y\). ([y], Y) f futures \(P\) ys
        using \(D\) by (simp add: traces-def Domain-iff futures-def)
        then obtain \(Y\) where \(([y], Y) \in\) futures \(P\) ys ..
        ultimately have
            (y \# ipurge-tr I D (D y) zs, ipurge-ref I D (D y) zs X) futures \(P\) ys
        using \(S\) by (simp add: secure-def)
        moreover have ipurge-tr \(I D(D y) z s=z s\)
        using False by (simp add: ipurge-tr-all)
        moreover have 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:
secure $P$ I $D \Longrightarrow(x s, y s) \in$ rel-ipurge-aux P I D $(D$ 'set zs $) \Longrightarrow$
xs @ zs $\in$ traces $P \Longrightarrow y s @ z s \in$ 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:
secure $P$ I $D \Longrightarrow(x s, y s) \in$ rel-ipurge $P I D(D x) \Longrightarrow$ $x \in$ next-events $P x s \Longrightarrow x \in$ 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:
secure $P$ I $D \Longrightarrow(x s, y s) \in$ rel-ipurge-aux P I D $\left(D^{\prime} X\right) \Longrightarrow$ $X \in$ refusals $P$ xs $\Longrightarrow X \in$ 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:
secure $P$ I $D \Longrightarrow(x s, y s) \in$ rel-ipurge $P I D(D x) \Longrightarrow$
$\{x\} \in$ refusals $P$ xs $\Longrightarrow\{x\} \in$ refusals $P$ ys
by (rule secure-implies-refusal-consistency, simp-all add: rel-ipurge-aux-single-dom)
theorem secure-implies-iu-condition:
assumes $S$ : secure P I D
shows future-consistent $P D$ (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) \inrel-ipurge P ID (Dx) and x\in next-events P xs
    with S show }x\in\mathrm{ next-events P ys by (rule secure-implies-next-event-consistency)
    next
fix xs ys x
have }\forallu\in\mathrm{ range D. equiv (traces P) (rel-ipurge P I Du)
using view-partition-rel-ipurge by (simp add: view-partition-def)
hence sym (rel-ipurge PID(D x)) by (simp add: equiv-def)
moreover assume (xs,ys)\in rel-ipurge PID (Dx)
ultimately have (ys,xs)\in rel-ipurge P I D (D x) by (rule symE)
moreover assume x next-events P ys
ultimately show }x\in\mathrm{ next-events }P\mathrm{ xs
using S by (rule-tac secure-implies-next-event-consistency)
next
fix xs ys x
assume (xs,ys)\in rel-ipurge P I D (D x) and {x}\in refusals P xs
with S show {x}\in refusals P ys by (rule secure-implies-ref-event-consistency)
next
fix xs ys x
have }\forallu\in\mathrm{ 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)\in rel-ipurge P I D (D x)
ultimately have (ys,xs)\in rel-ipurge P I D (D x) by (rule symE)
moreover assume {x}\in refusals P ys
ultimately show {x}\in refusals P xs
using S by (rule-tac secure-implies-ref-event-consistency)
qed
theorem ipurge-unwinding:
ref-union-closed P\Longrightarrow
secure PI 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-imply-secure)
end

```

\section*{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.

\subsection*{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 $\Rightarrow\left({ }^{\prime} a \Rightarrow{ }^{\prime} d\right) \Rightarrow\left({ }^{\prime} a,{ }^{\prime} d\right)$ dom-rel-map $\Rightarrow$ bool where
d-future-consistent $P D R \equiv$
$\forall u \in$ range $D . \forall x s$ ys. $(x s, y s) \in R u \longrightarrow$
$(x s \in$ traces $P)=(y s \in$ traces $P) \wedge$
next-dom-events $P D u$ xs $=$ next-dom-events $P D u$ ys
definition $d$-weakly-future-consistent ::
'a process $\Rightarrow(' d \times ' d)$ set $\Rightarrow\left({ }^{\prime} a \Rightarrow{ }^{\prime} d\right) \Rightarrow\left({ }^{\prime} a,{ }^{\prime} d\right)$ dom-rel-map $\Rightarrow$ bool where
d-weakly-future-consistent P I D $R \equiv$
$\forall u \in$ range $D \cap(-I)$ " range $D . \forall x s$ ys. $(x s, y s) \in R u \longrightarrow$
$(x s \in$ traces $P)=(y s \in$ traces $P) \wedge$
next-dom-events $P D$ uxs $=$ next-dom-events $P D$ u ys
lemma dfc-implies-dwfc:
$d$-future-consistent $P D R \Longrightarrow d$-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 $y s$
assume
$A: \forall u \in$ range $D \cap(-I)$ " range $D . \forall x s$ ys. $(x s, y s) \in$ rel-ipurge $P I D u \longrightarrow$

```
```

        (xs \in traces P) = (ys\in traces P)^
        next-dom-events P D u xs = next-dom-events P Du ys and
    B:u\in range D and
    C:(xs, ys) \in rel-ipurge P I D u
    show (xs\in traces P)}=(ys\in\mathrm{ traces P)^
    next-dom-events P D u xs = next-dom-events P D u ys
    proof (cases u f range D \cap (-I) " range D)
case True
with A have }\forallxs\mathrm{ ys. (xs, ys) f rel-ipurge P I D u}
(xs traces P) = (ys \in traces P) ^
next-dom-events P D u xs = next-dom-events P D u ys ..
hence (xs,ys) \in rel-ipurge PIDu\longrightarrow
(xs\in traces P)=(ys\intraces P)^
next-dom-events P D u xs = next-dom-events P D u ys
by blast
thus ?thesis using C ..
next
case False
hence }D:u\not\in(-I) " range D using B by sim
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 }\forallzs.ipurge-tr-rev I Duzs=z
proof (rule allI, rule ipurge-tr-rev-all, rule ballI, erule imageE, rule ccontr)
fix vx
assume (v,u)\not\inI
hence (v,u)\in-I by simp
moreover assume v=Dx
hence }v\in\mathrm{ range D by simp
ultimately have }u\in(-I) " range D ..
thus False using D by contradiction
qed
ultimately show ?thesis by simp
qed
qed
lemma d-fc-equals-dfc:
assumes A: deterministic P
shows future-consistent P D R =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: \forallu\in range D. \forallxs ys. (xs,ys) \inRu\longrightarrow
(xs\in traces P) = (ys \in traces P)^
next-dom-events P D u xs = next-dom-events P D u ys
show }\forallu\in\mathrm{ range D. }\forall\mathrm{ xs ys. (xs, ys) }\inR|u
ref-dom-events P D u xs=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 }

```
```

    assume u\in range D
    with B have }\forallxs ys. (xs, ys)\inRu
    (xs\intraces P) = (ys \in traces P)^
    next-dom-events P D u xs = next-dom-events P D u ys ..
    hence (xs,ys)\inRu\longrightarrow
    (xs traces P) = (ys \in traces P) ^
    next-dom-events P D u xs = next-dom-events P D u ys
    by blast
    moreover assume (xs,ys)\inRu
    ultimately have C:(xs\in traces P)=(ys\in traces P)^
    next-dom-events P D u xs = next-dom-events P D u ys ..
    show (u=D x ^ {x}\in refusals P xs) = (u=D x ^{x}\in refusals P ys)
    proof (cases u=D x, simp-all, cases xs }\in\mathrm{ traces P)
    assume D:u=Dx and E:xs\in traces P
    have
        A':}\forallxs\in\mathrm{ traces P. }\forallX.X\in\mathrm{ refusals P xs = (X П next-events P xs = {})
        using A by (simp add: deterministic-def)
    hence }\forallX.X\in\mathrm{ refusals P xs = (X П next-events P xs = {}) using E ..
    hence {x}\in refusals P xs=({x}\cap next-events P xs = {})..
    moreover have ys }\in\mathrm{ traces P using C and E by simp
    with }\mp@subsup{A}{}{\prime}\mathrm{ have }\forallX.X\in\mathrm{ refusals }P\mathrm{ ys = (X П next-events P ys={})..
    hence {x}\in refusals P ys=({x}\cap next-events P ys = {})..
    moreover have {x}\cap next-events P xs ={x}\cap next-events P ys
    proof (simp add: set-eq-iff, rule allI, rule iffI, erule-tac [!] conjE, simp-all)
        assume x\in next-events P xs
    hence x\in next-dom-events P D u xs using D by (simp add: next-dom-events-def)
        hence }x\in\mathrm{ next-dom-events P D u ys using C by simp
        thus }x\in\mathrm{ next-events P ys by (simp add: next-dom-events-def)
    next
        assume x\in next-events P ys
    hence x next-dom-events P D u ys using D by (simp add: next-dom-events-def)
        hence x\in next-dom-events P D u xs using C by simp
        thus }x\in\mathrm{ next-events P xs by (simp add: next-dom-events-def)
        qed
        ultimately show ({x}\in refusals P xs) =({x}\in refusals P ys) by simp
    next
    assume D: xs \not\intraces P
    hence }\forallX.(xs,X)\not\in failures P by (simp add: traces-def Domain-iff
    hence refusals P xs = {} by (rule-tac equals0I, simp add: refusals-def)
    moreover have ys }\not\in\mathrm{ traces P using C and D by simp
    hence }\forallX\mathrm{ . (ys, X) # failures P by (simp add: traces-def Domain-iff)
    hence refusals P ys = {} by (rule-tac equals0I, simp add: refusals-def)
    ultimately show ({x}\in refusals P xs) =({x}\in 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 \(B: \forall u \in\) range \(D \cap(-I)\) " range \(D . \forall\) xs ys. \((x s, y s) \in R u \longrightarrow\)
\((x s \in\) traces \(P)=(y s \in\) traces \(P) \wedge\)
next-dom-events \(P D u x s=\) next-dom-events \(P D u\) ys
show \(\forall u \in\) range \(D \cap(-I)\) " range \(D . \forall x s\) ys. \((x s, y s) \in R u \longrightarrow\)
ref-dom-events \(P D u x s=\) ref-dom-events \(P D\) u ys
proof (rule ballI, (rule allI)+, rule impI,
\(\operatorname{simp}\) (no-asm-simp) add: ref-dom-events-def set-eq-iff, rule allI)
fix \(u\) xs ys \(x\)
assume \(u \in\) range \(D \cap(-I)\) " range \(D\)
with \(B\) have \(\forall x s\) ys. \((x s, y s) \in R u \longrightarrow\)
\((\) xs \(\in\) traces \(P)=(y s \in\) traces \(P) \wedge\)
next-dom-events \(P D\) u xs = next-dom-events \(P D\) u ys ..
hence \((x s, y s) \in R u \longrightarrow\)
\((x s \in\) traces \(P)=(y s \in\) traces \(P) \wedge\)
next-dom-events \(P D u x s=\) next-dom-events \(P D u\) ys
by blast
moreover assume \((x s, y s) \in R u\)
ultimately have \(C:(x s \in\) traces \(P)=(y s \in\) traces \(P) \wedge\)
next-dom-events \(P D\) u xs = next-dom-events \(P D\) u ys ..
show \((u=D x \wedge\{x\} \in\) refusals \(P x s)=(u=D x \wedge\{x\} \in\) refusals \(P\) ys \()\)
proof (cases \(u=D x\), simp-all, cases xs \(\in\) traces \(P\) )
assume \(D: u=D x\) and \(E: x s \in \operatorname{traces} P\)
have \(A^{\prime}: \forall x s \in\) traces \(P . \forall X\).
\(X \in\) refusals \(P x s=(X \cap\) next-events \(P x s=\{ \})\)
using \(A\) by (simp add: deterministic-def)
hence \(\forall X . X \in\) refusals \(P\) xs \(=(X \cap\) next-events \(P x s=\{ \})\) using \(E .\).
hence \(\{x\} \in\) refusals \(P x s=(\{x\} \cap\) next-events \(P\) xs \(=\{ \})\)..
moreover have ys \(\in\) traces \(P\) using \(C\) and \(E\) by simp
with \(A^{\prime}\) have \(\forall X . X \in\) refusals \(P\) ys \(=(X \cap\) next-events \(P\) ys \(=\{ \}) .\).
hence \(\{x\} \in\) refusals \(P\) ys \(=(\{x\} \cap\) next-events \(P\) ys \(=\{ \}) .\).
moreover have \(\{x\} \cap\) next-events \(P x s=\{x\} \cap\) next-events \(P\) ys
proof (simp add: set-eq-iff, rule allI, rule iffI, erule-tac [!] conjE, simp-all)
assume \(x \in\) next-events \(P\) xs
hence \(x \in\) next-dom-events \(P D u x s\) using \(D\) by (simp add: next-dom-events-def)
hence \(x \in\) next-dom-events \(P D\) u ys using \(C\) by simp
thus \(x \in\) next-events \(P\) ys by (simp add: next-dom-events-def)
next
assume \(x \in\) next-events \(P\) ys
hence \(x \in\) next-dom-events \(P D\) us using \(D\) by (simp add: next-dom-events-def)
hence \(x \in\) next-dom-events \(P D u\) xs using \(C\) by simp
thus \(x \in\) next-events \(P\) xs by (simp add: next-dom-events-def)
qed
ultimately show \((\{x\} \in\) refusals \(P x s)=(\{x\} \in\) refusals \(P\) ys) by simp
next
assume \(D\) : xs \(\notin\) traces \(P\)
```

        hence }\forallX.(xs,X)\not\in failures P by (simp add: traces-def Domain-iff
        hence refusals P xs = {} by (rule-tac equalsOI, simp add:refusals-def)
        moreover have ys }\not\in\mathrm{ traces P using C and D by simp
        hence }\forallX.(ys,X)\not\in\mathrm{ failures P by (simp add: traces-def Domain-iff)
        hence refusals P ys ={} by (rule-tac equals0I, simp add: refusals-def)
        ultimately show ({x}\in refusals P xs) = ({x}\in refusals P 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 $A$ : deterministic $P$
shows ref-union-closed $P$
proof (subst ref-union-closed-def, (rule allI)+, (rule impI)+, erule exE)
fix $x s A X$
have $\forall x s \in$ traces $P . \forall X . X \in$ refusals $P x s=(X \cap$ next-events $P x s=\{ \})$
using $A$ by (simp add: deterministic-def)
moreover assume $B: \forall X \in A .(x s, X) \in$ failures $P$ and $X \in A$
hence $(x s, X) \in$ failures $P$..
hence $x s \in$ traces $P$ by (rule failures-traces)
ultimately have $C: \forall X . X \in$ refusals $P x s=(X \cap$ next-events $P x s=\{ \}) .$.
have $D: \forall X \in A . X \cap$ next-events $P x s=\{ \}$
proof
fix $X$
assume $X \in A$
with $B$ have $(x s, X) \in$ failures $P$..
hence $X \in$ refusals $P$ xs by (simp add: refusals-def)
thus $X \cap$ next-events $P$ xs $=\{ \}$ using $C$ by simp
qed
have $(\bigcup X \in A . X) \in$ refusals $P$ xs $=((\bigcup X \in A . X) \cap$ next-events $P$ xs $=\{ \})$
using $C$..
hence $E:(x s, \bigcup X \in A . X) \in$ failures $P=$
$((\bigcup X \in A . X) \cap$ next-events $P x s=\{ \})$
by (simp add: refusals-def)
show $(x s, \bigcup X \in A . X) \in$ failures $P$
proof (rule ssubst [OF E], rule equals0I, erule IntE, erule $U N-E$ )
fix $x X$
assume $X \in A$
with $D$ have $X \cap$ next-events $P$ xs $=\{ \} .$.
moreover assume $x \in X$ and $x \in$ next-events $P$ xs

```
```

    hence }x\inX\cap\mathrm{ next-events }P\mathrm{ xs ..
    hence }\existsx.x\inX\cap\mathrm{ next-events P xs ..
    hence }X\cap\mathrm{ next-events P xs }\not={}\mathrm{ 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)

```

\subsection*{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
\(t s\)-process \(T \equiv\) Abs-process \((\{(x s, X) . x s \in T \wedge(\forall x \in X . x s @[x] \notin T)\},\{ \})\)
definition trace-set :: 'a list set \(\Rightarrow\) bool where
trace-set \(T \equiv[] \in T \wedge(\forall x s x . x s @[x] \in T \longrightarrow x s \in T)\)
lemma \(t s\)-process-rep:
assumes \(A\) : trace-set \(T\)
shows Rep-process (ts-process \(T\) ) \(=\)
\((\{(x s, X) . x s \in T \wedge(\forall x \in X . x s @[x] \notin T)\},\{ \})\)
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 []\(\in T\) using \(A\) by (simp add: trace-set-def)
```

next
show }\forallxs.(\existsx.xs@[x]\inT\wedge(\existsX.\forall\mp@subsup{x}{}{\prime}\inX.xs@[x,x]\not\inT))\longrightarrowxs\in
proof (rule allI, rule impI, erule exE, erule conjE)
fix xs x
have \forallxs x. xs @ [x] \inT\longrightarrow <s \inT using A by (simp add: trace-set-def)
hence xs@ [x]\inT\longrightarrowxs\inT by blast
moreover assume xs @ [x] \inT
ultimately show xs \inT ..
qed
next
show \forallxs X. xs }\inT\wedge(\existsY.(\forallx\inY.xs@[x]\not\inT)\wedgeX\subseteqY)
(\forallx\inX.xs @ [x]\not\inT)
proof ((rule allI)+, rule impI, (erule conjE,(erule exE)?)+, rule ballI)
fix xs x X Y
assume }\forallx\inY.xs@[x]\not\in
moreover assume X\subseteqY and x\inX
hence }x\inY\mathrm{ ..
ultimately show xs @ [x]\not\inT ..
qed
qed
lemma ts-process-failures:
trace-set T\Longrightarrow
failures (ts-process T)={(xs,X).xs\inT^(\forallx\inX. xs @ [x]\not\inT)}
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\inT^(\forally\inY.xs @ ys @ [y]\not\inT)}
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={}\mathrm{ in exI, simp)
lemma ts-process-refusals:
trace-set T\Longrightarrowxs}\inT
refusals (ts-process T) xs ={X.\forallx\inX.xs@ @x]\not\inT}
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] \inT)
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 $t$ s-process- $d$ :
trace-set $T \Longrightarrow$ 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 $x s X$
assume $\forall x \in X . x s @[x] \notin T$
thus $X \cap\{x . x s @[x] \in T\}=\{ \}$
by (rule-tac equals0I, erule-tac IntE, simp)
next
fix $x s X$
assume $A: X \cap\{x . x s @[x] \in T\}=\{ \}$
show $\forall x \in X$. xs @ $[x] \notin T$
proof (rule ballI, rule notI)
fix $x$
assume $x \in X$ and $x s @[x] \in T$
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 $x s$
have $B$ : Rep-process $P \in$ process-set (is ? $P^{\prime} \in-$ ) by (rule Rep-process)
hence $\forall x s . \exists x . x s \in$ snd ? $P^{\prime} \longrightarrow x s @[x] \in$ snd ? $P^{\prime}$
by (simp add: process-set-def process-prop-5-def)
hence $\exists x . x s \in$ snd ? $P^{\prime} \longrightarrow x s @[x] \in$ snd ? $P^{\prime}$..
then obtain $x$ where $x s \in$ snd ? $P^{\prime} \longrightarrow x s @[x] \in$ snd ? $P^{\prime} .$.
moreover assume $C: x s \in$ snd ? $P^{\prime}$
ultimately have $D: x s @[x] \in$ snd ? $P^{\prime} .$.
have $E: \forall x s X . x s \in$ snd $? P^{\prime} \longrightarrow(x s, X) \in f s t ? P^{\prime}$
using $B$ by (simp add: process-set-def process-prop- 6 -def)
hence $x s \in$ snd ? $P^{\prime} \longrightarrow(x s,\{x\}) \in f s t ? P^{\prime}$ by blast
hence $\{x\} \in$ refusals $P$ xs
using $C$ by (drule-tac mp, simp-all add: failures-def refusals-def)
moreover have $x s @[x] \in$ snd ? $P^{\prime} \longrightarrow(x s @[x],\{ \}) \in f s t ? P^{\prime}$
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 \not={} by (simp add: next-events-def)
    ultimately have G: ({x}\in refusals P xs )\not=({x}\cap next-events Pxs={})
    by simp
    have }\forallxs\in\mathrm{ traces P.}\forallX.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 }\forallX.X\in\mathrm{ refusals P xs = (X @ 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 的ocess-set (is ?P' }\mp@subsup{P}{}{\prime}\in\mathrm{ -) by (rule Rep-process)
hence ([],{})\infst ?P' by (simp add: process-set-def process-prop-1-def)
thus \existsX.([], X) \infst ?P'..
next
fix xs x X
have Rep-process P 的ocess-set (is ?P' }\in\mathrm{ -) by (rule Rep-process)
hence \forallxs x X. (xs @ [x],X) \infst?P' \longrightarrow (xs,{}) \in fst?P'
by (simp add: process-set-def process-prop-2-def)
hence (xs@ @x],X)\infst?? 'P}\longrightarrow(xs,{})\infst?P' by blas
moreover assume (xs @ [x],X) \infst?P'
ultimately have (xs, {}) \infst ?P'..
thus }\existsX.(xs,X)\infst?\mp@subsup{P}{}{\prime}
qed
lemma d-implies-ts-process-traces:
deterministic P\Longrightarrowts-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: \forallxs \in traces P.\forallX.
(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.\forallx\inX. xs @ [x]\not\in traces P}
by (rule ts-process-refusals)
moreover have X Gefusals (ts-process (traces P)) xs

```
```

        using D by (simp add: refusals-def)
        ultimately have }\forallx\inX.xs@[x]\not\intraces P by sim
        hence }X\cap\mathrm{ next-events Pxs={}
        by (rule-tac equals0I, erule-tac IntE, simp add: next-events-def)
        moreover have }\forallX.(X\in\mathrm{ refusals P xs )}=(X\cap\mathrm{ next-events P xs = {})
        using }A\mathrm{ and E ..
    hence ( }X\in\mathrm{ refusals P xs)=(X \ next-events P xs = {})..
    ultimately have }X\in\mathrm{ 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
    ```

```

    with A have \forallX.(X 隹usals P xs )=(X\cap next-events P xs = {})..
    ```

```

    moreover have }X\in\mathrm{ refusals P xs using D by (simp add: refusals-def)
    ultimately have F:X\cap{x.xs@ @ [x]\in\operatorname{traces}P}={}
    by (simp add: next-events-def)
    have }\forallx\inX.xs@[x]\not\in\mathrm{ traces P
    proof (rule ballI, rule notI)
        fix }
        assume }x\inX\mathrm{ and xs@ @ [x] E traces P
        hence }x\inX\cap{x.xs@[x]\in\mathrm{ traces P} by simp
        moreover have x\not\inX\cap{x. xs @ [x]\in traces P} using F by (rule equalsOD)
            ultimately show False by contradiction
    qed
    moreover have
        refusals (ts-process (traces P)) xs ={X.\forallx\inX.xs@ [x]\not\intraces P}
        using B and E by (rule ts-process-refusals)
    ultimately have X (refusals (ts-process (traces P)) xs by simp
    thus (xs, X) \infailures (ts-process (traces P)) by (simp add: refusals-def)
    qed
    qed
lemma ts-process-traces-implies-d:
ts-process (traces P) =P\Longrightarrow 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\Longrightarrow
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

```

\section*{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.```

