The Generic Unwinding Theorem
for CSP Noninterference Security

Pasquale Noce
Security Certification Specialist at Arjo Systems - Gep S.p.A.
pasquale dot noce dot lavoro at gmail dot com
pasquale dot noce at arjowiggins-it dot com
June 11, 2019

Abstract

The classical definition of noninterference security for a deterministic state machine with outputs requires to consider the outputs produced by machine actions after any trace, i.e. any indefinitely long sequence of actions, of the machine. In order to render the verification of the security of such a machine more straightforward, there is a need of some sufficient condition for security such that just individual actions, rather than unbounded sequences of actions, have to be considered.

By extending previous results applying to transitive noninterference policies, Rushby has proven an unwinding theorem that provides a sufficient condition of this kind in the general case of a possibly intransitive policy. This condition has to be satisfied by a generic function mapping security domains into equivalence relations over machine states.

An analogous problem arises for CSP noninterference security, whose definition 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.

This paper provides a 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 one for classical noninterference security, and has to be satisfied by a generic function mapping security domains into equivalence relations over process traces; hence its name, Generic Unwinding Theorem. Variants of this theorem applying to deterministic processes and trace set processes are also proven. Finally, the sufficient condition for security expressed by the theorem is shown not to be a necessary condition as well, viz. there exists a secure process such that no domain-relation map satisfying the condition exists.
1 The Generic Unwinding Theorem

The classical definition of noninterference security for a deterministic state machine with outputs requires to consider the outputs produced by machine actions after any trace, i.e. any indefinitely long sequence of actions, of the machine. In order to render the verification of the security of such a machine more straightforward, there is a need of some sufficient condition for security such that just individual actions, rather than unbounded sequences of actions, have to be taken into consideration.

By extending previous results applying to transitive noninterference policies, Rushby [8] has proven an unwinding theorem that provides a sufficient condition of this kind in the general case of a possibly intransitive policy. This condition consists of a combination of predicates, which have to be satisfied by a generic function mapping security domains into equivalence relations over machine states.

An analogous problem arises for CSP noninterference security, whose definition 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. This paper provides a 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 one for classical noninterference security; in some detail, it consists of a combination of predicates, which are the translations of Rushby’s ones into Hoare’s Communicating Sequential Processes model of computation [1]. These predicates have to be satisfied by a generic function mapping security domains into equivalence relations over process traces; hence the name given to the condition, Generic Unwinding Theorem. Variants of this theorem applying to deterministic processes and trace set processes (cf. [7]) are also proven.

The sufficient condition for security expressed by the Generic Unwinding
Theorem would be even more valuable if it also provided a necessary condition, viz. if for any secure process, there existed some domain-relation map satisfying the condition. Particularly, a constructive proof of such proposition, showing that some specified domain-relation map satisfies the condition whatever secure process is given, would permit to determine whether a process is secure or not by verifying whether the condition is satisfied by that map or not. However, this paper proves by counterexample that the Generic Unwinding Theorem does not express a necessary condition for security as well, viz. a process and a noninterference policy for that process are constructed such that the process is secure with respect to the policy, but no domain-relation map satisfying the condition exists.

The contents of this paper are based on those of [6] and [7]. The salient points of definitions and proofs are commented; for additional information, cf. Isabelle documentation, particularly [5], [4], [3], and [2].

For the sake of brevity, given a function \( F \) of type \( 'a_1 \Rightarrow \ldots \Rightarrow 'a_m \Rightarrow 'a_{m+1} \Rightarrow \ldots \Rightarrow 'a_n \Rightarrow 'b \), the explanatory text may discuss of \( F \) using attributes that would more exactly apply to a term of type \( 'a_{m+1} \Rightarrow \ldots \Rightarrow 'a_n \Rightarrow 'b \).

In this case, it shall be understood that strictly speaking, such attributes apply to a term matching pattern \( F \ a_1 \ldots \ a_m \).

1.1 Propaedeutic definitions and lemmas

Here below are the translations of Rushby’s predicates *weakly step consistent* and *locally respects* [8], applying to deterministic state machines, into Hoare’s Communicating Sequential Processes model of computation [1].

The 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.
- While every machine action is accepted in a machine state, not every process event is generally accepted after a process trace. Thus, whenever an event is appended to an event list in the consequence of an implication, the antecedent of the implication constrains the event list to be a trace, and the event to be accepted after that trace. In this way, the predicates do not unnecessarily impose that the relations in the range of the domain-relation map hold between event lists not being process traces.
definition weakly-step-consistent ::
  'a process ⇒ ('a ⇒ 'd) dom-rel-map ⇒ bool where
weakly-step-consistent P D R ≡ ∀ u ∈ range D. ∀ xs ys x.
  (xs, ys) ∈ R u ∩ R (D x) ∧ x ∈ next-events P xs ∩ next-events P ys −→
  (xs @ [x], ys @ [x]) ∈ R u

definition locally-respects ::
  'a process ⇒ ('d × 'd) set ⇒ ('a ⇒ 'd) dom-rel-map ⇒ bool where
locally-respects P I D R ≡ ∀ u ∈ range D. ∀ xs x.
  (D x, u) /∈ I ∧ x ∈ next-events P xs −→
  (xs, xs @ [x]) ∈ R u

In what follows, some lemmas propaedeutic for the proof of the Generic
Unwinding Theorem are demonstrated.

lemma ipurge-tr-aux-single-event:
ipurge-tr-aux I D U [x] = (if ∃ v ∈ U. (v, D x) ∈ I
  then []
  else [x])
proof (cases ∃ v ∈ U. (v, D x) ∈ I)
case True
  have ipurge-tr-aux I D U [x] = ipurge-tr-aux I D U ([] @ [x]) by simp
also have . . . = [] using True by (simp only: ipurge-tr-aux.simps, simp)
finally show ?thesis using True by simp
next
case False
  have ipurge-tr-aux I D U [x] = ipurge-tr-aux I D U ([] @ [x]) by simp
also have . . . = [x] using False by (simp only: ipurge-tr-aux.simps, simp)
finally show ?thesis using False by simp
qed

lemma ipurge-tr-aux-cons:
ipurge-tr-aux I D U (x # xs) = (if ∃ v ∈ U. (v, D x) ∈ I
  then ipurge-tr-aux I D (insert (D x) U) xs
  else x # ipurge-tr-aux I D U xs)
proof (induction xs rule: rev-induct, case-tac [] ∃ v ∈ U. (v, D x) ∈ I,
simp-all add: ipurge-tr-aux-single-event del: ipurge-tr-aux.simps(2))
fix x' xs
assume A: ipurge-tr-aux I D U (x # xs) =
ipurge-tr-aux I D (insert (D x) U) xs
(is ?T = ?T')
assume ∃ v ∈ U. (v, D x) ∈ I
hence B: sinks-aux I D U (x # xs) = sinks-aux I D (insert (D x) U) xs
(is ?S = ?S')
  by (simp add: sinks-aux-cons)
show ipurge-tr-aux I D U (x # xs @ [x']) =
ipurge-tr-aux I D (insert (D x) U) (xs @ [x'])
proof (cases ∃ v ∈ ?S. (v, D x') ∈ I)
case True
  hence ipurge-tr-aux I D U ((x ≠ xs) @ [x']) = ?T
    by (simp only: ipurge-tr-aux.simps, simp)
  moreover have ∃ v ∈ ?S'. (v, D x') ∈ I using B and True by simp
  hence ipurge-tr-aux I D (insert (D x) U) (xs @ [x']) = ?T' by simp
  ultimately show ?thesis using A by simp
next
  case False
  hence ipurge-tr-aux I D U ((x ≠ xs) @ [x']) = ?T
    by (simp only: ipurge-tr-aux.simps, simp)
  moreover have ¬ (∃ v ∈ ?S'. (v, D x') ∈ I) using B and False by simp
  hence ipurge-tr-aux I D (insert (D x) U) (xs @ [x']) = ?T' @ [x'] by simp
  ultimately show ?thesis using A by simp
qed

next
fix x' xs
assume A: ipurge-tr-aux I D U (x ≠ xs) = x ≠ ipurge-tr-aux I D U xs
(is ?T = ?T')
assume B: sinks-aux I D U (x ≠ xs) = sinks-aux I D U xs
(is ?S = ?S')
by (simp add: sinks-aux-cons)
show ipurge-tr-aux I D U (x ≠ xs @ [x']) =
  x ≠ ipurge-tr-aux I D U (xs @ [x'])
proof (cases ∃ v ∈ ?S. (v, D x') ∈ I)
case True
  hence ipurge-tr-aux I D U ((x ≠ xs) @ [x']) = ?T
    by (simp only: ipurge-tr-aux.simps, simp)
  moreover have ∃ v ∈ ?S'. (v, D x') ∈ I using B and True by simp
  hence x ≠ ipurge-tr-aux I D U (xs @ [x']) = ?T' by simp
  ultimately show ?thesis using A by simp
next
case False
  hence ipurge-tr-aux I D U ((x ≠ xs) @ [x']) = ?T
    by (simp only: ipurge-tr-aux.simps, simp)
  moreover have ¬ (∃ v ∈ ?S'. (v, D x') ∈ I) using B and False by simp
  hence x ≠ ipurge-tr-aux I D U (xs @ [x']) = ?T' @ [x'] by simp
  ultimately show ?thesis using A by simp
qed

lemma unaffected-domains-subset:
assumes
  A: U ⊆ range D and
  B: U ≠ {}
shows unaffected-domains I D U xs ⊆ range D ∩ (−I) " range D
proof (subt unaffected-domains-def, rule subsetI, simp, erule conjE)
fix v
have U ⊆ sinks-aux I D U xs by (rule sinks-aux-subset)
moreover have $\exists u.\ u \in U$ using $B$ by (simp add: ex-in-conv) 
then obtain $u$ where $C$: $u \in U$ .. 
ultimately have $D$: $u \in sinks-aux I D U \ v$ .. 
assume $\forall u \in sinks-aux I D U \ v$. $(u, v) \notin I$ 
hence $(u, v) \notin I$ using $D$ .. 
hence $(u, v) \in -I$ by simp 
moreover have $u \in range D$ using $A$ and $C$ .. 
ultimately show $v \in (-I) \ " range D ..$
qed

1.2 The Generic Unwinding Theorem: proof of condition sufficiency

Rushby’s Unwinding Theorem for Intransitive Policies [8] states that a sufficient condition for a deterministic state machine with outputs to be secure is the existence of some domain-relation map $R$ such that:

1. $R$ is a view partition, i.e. the relations over machine states in its range are equivalence relations;

2. $R$ is output consistent, i.e. states equivalent with respect to the domain of an action produce the same output as a result of that action;

3. $R$ is weakly step consistent;

4. $R$ locally respects the policy.

The idea behind the theorem is that a machine is secure if its states can be partitioned, for each domain $u$, into equivalence classes (1), such that the states in any such class $C$ are indistinguishable with respect to the actions in $u$ (2), transition into the same equivalence class $C'$ as a result of an action (3), and transition remaining inside $C$ as a result of an action not allowed to affect $u$ (4).

This idea can simply be translated into the realm of Communicating Sequential Processes [1] by replacing the words ”machine”, ”state”, ”action” with ”process”, ”trace”, ”event”, respectively, as long as a clarification is provided of what it precisely means for a pair of traces to be ”indistinguishable” with respect to the events in a given domain. Intuitively, this happens just in case the events in that domain being accepted or refused after either trace are the same, thus the simplest choice would be to replace output consistency with future consistency as defined in [7]. However, indistinguishability between traces in the same equivalence class is not required in the case of a domain allowed to be affected by any domain, since the policy puts no restriction on the differences in process histories that may be detected by such a domain. Hence, it is sufficient to replace output consistency with weak future consistency [7].
Furthermore, indistinguishability with respect to individual refused events does not imply indistinguishability with respect to sets of refused events, i.e. refusals, unless for each trace, the corresponding refusals set is closed under set union. Therefore, for the condition to be sufficient for process security, the \textit{refusals union closure} of the process [7] is also required. As remarked in [7], this property holds for any process admitting a meaningful interpretation, so that taking it as an additional assumption does not give rise to any actual limitation on the applicability of the theorem.

As a result of these considerations, the Generic Unwinding Theorem, formalized in what follows as theorem \textit{generic-unwinding}, states that a sufficient condition for the CSP noninterference security [6] of a process being refusals union closed [7] is the existence of some domain-relation map $R$ such that:

1. $R$ is a view partition [7];
2. $R$ is weakly future consistent [7];
3. $R$ is weakly step consistent;
4. $R$ locally respects the policy.

\textbf{lemma} \textit{ruc-wfc-failures}:
\begin{itemize}
  \item \textbf{assumes}
  \begin{itemize}
    \item[$\text{RUC}$:] ref-union-closed $P$ and
    \item[$\text{WFC}$:] weakly-future-consistent $P$ I $D$ $R$ and
    \item[$A$:] $U \subseteq \text{range } D \cap (\sim I) \sim \text{range } D$ and
    \item[$B$:] $U \neq \emptyset$ and
    \item[$C$:] $\forall u \in U. \ (xs, xs') \in R u$ and
    \item[$D$:] $(xs, X) \in \text{failures } P$
  \end{itemize}
  \item \textbf{shows} $(xs', X \cap D - \sim U) \in \text{failures } P$
\end{itemize}
\textbf{proof} (cases $\exists x. \ x \in X \cap D - \sim U$)

\begin{itemize}
  \item let $\exists A = \text{singleton-set} \ (X \cap D - \sim U)$
  \item have $\forall xs. \ A. \ (\exists X. \ X \in A) \longrightarrow (\forall X. \ X \in A. \ (xs, X) \in \text{failures } P) \longrightarrow$
    \begin{itemize}
      \item $(xs, \bigcup X \in A. \ X) \in \text{failures } P$
    \end{itemize}
  \end{itemize}

\item using \textit{RUC} by (simp add: ref-union-closed-def)
\begin{itemize}
  \item hence $(xs', \bigcup X \in \exists A. \ X) \in \text{failures } P$
  \item by blast
\end{itemize}

moreover \textbf{case} True
\begin{itemize}
  \item hence $\exists X. \ X \in \exists A$ by (simp add: singleton-set-some)
  \item ultimately have $(\forall X. \ X \in \exists A. \ (xs', X) \in \text{failures } P) \longrightarrow$
    \begin{itemize}
      \item $(xs', \bigcup X \in \exists A. \ X) \in \text{failures } P$
    \end{itemize}
  \end{itemize}

moreover \textbf{have} $\forall X. \ X \in \exists A. \ (xs', X) \in \text{failures } P$
\textbf{proof} (simp add: singleton-set-def, rule allI, rule impI, erule bexE, erule IntE, simp)

fix $x$
have \( \forall u \in \text{range} \ D \cap (-I) \quad (\forall \, \text{range} \, D. \, \forall \, \text{ys} \, \exists \, u \in R \, u \rightarrow \text{next-dom-events} \, P \, D \, u \, \text{xs} = \text{next-dom-events} \, P \, D \, u \, \text{ys} \wedge \text{ref-dom-events} \, P \, D \, u \, \text{xs} = \text{ref-dom-events} \, P \, D \, u \, \text{ys} \) using \( \text{WFC} \) by (simp add: \( \text{weakly-future-consistent-def} \))

moreover assume \( E : D \, x \in U \) with \( A \) have \( D \, x \in \text{range} \ D \cap (-I) \quad (\forall \, \text{range} \, D \wedge \text{..} \)

ultimately have \( \forall \, \text{xs} \, \text{ys} \, (\exists \, u \in R \, (D \, x) \rightarrow \text{next-dom-events} \, P \, D \, (D \, x) \, \text{xs} = \text{next-dom-events} \, P \, D \, (D \, x) \, \text{ys} \wedge \text{ref-dom-events} \, P \, D \, (D \, x) \, \text{xs} = \text{ref-dom-events} \, P \, D \, (D \, x) \, \text{ys} \) ..

hence \( (xs, xs') \in R \, (D \, x) \rightarrow \text{next-dom-events} \, P \, D \, (D \, x) \, \text{xs} = \text{next-dom-events} \, P \, D \, (D \, x) \, \text{xs}' \wedge \text{ref-dom-events} \, P \, D \, (D \, x) \, \text{xs} = \text{ref-dom-events} \, P \, D \, (D \, x) \, \text{xs}' \)

by blast

moreover have \( (xs, xs') \in R \, (D \, x) \) using \( C \) and \( E \) ..

ultimately have \( \text{ref-dom-events} \, P \, D \, (D \, x) \, \text{xs} = \text{ref-dom-events} \, P \, D \, (D \, x) \, \text{xs}' \)

by simp

moreover assume \( x \in X \)

hence \( \{x\} \subseteq X \) by simp

with \( D \) have \( (xs, \{x\}) \in \text{failures} \, P \) by (rule process-rule-?)

hence \( x \in \text{ref-dom-events} \, P \, D \, (D \, x) \, \text{xs} \)

by (simp add: \( \text{ref-dom-events-def refusals-def} \))

ultimately have \( x \in \text{ref-dom-events} \, P \, D \, (D \, x) \, \text{xs}' \) by simp

thus \( (xs', \{x\}) \in \text{failures} \, P \) by (simp add: \( \text{ref-dom-events-def refusals-def} \))

qed

ultimately have \( (xs', \bigcup X \in ?A. \, X) \in \text{failures} \, P \) ..

thus \( (xs', X \cap D^{-U}) \in \text{failures} \, P \) by (simp only: \( \text{singleton-set-union} \))

next

have \( \exists \, u, \, u \in U \) using \( B \) by (simp add: \( \text{ex-in-cone} \))

then obtain \( u \) where \( E : u \in U \) ..

with \( A \) have \( u \in \text{range} \ D \cap (-I) \quad (\forall \, \text{range} \, D \wedge \text{..} \)

moreover have \( (xs, xs') \in R \, u \) using \( C \) and \( E \) ..

ultimately have \( (xs \in \text{traces} \, P) = (xs' \in \text{traces} \, P) \)

by (rule \( \text{wfc-traces [OF \, \text{WFC]}] \))

moreover have \( xs \in \text{traces} \, P \) using \( D \) by (rule \( \text{failures-traces} \))

ultimately have \( xs' \in \text{traces} \, P \) by simp

hence \( (xs', \{\}) \in \text{failures} \, P \) by (rule \( \text{traces-failures} \))

moreover case \( \text{False} \)

hence \( X \cap D^{-U} = \{\} \) by (simp only: \( \text{ex-in-cone, simp} \))

ultimately show \( (xs', X \cap D^{-U}) \in \text{failures} \, P \) by simp

qed

lemma \( \text{ruc-wfc-lr-failures-I} : \)

assumes

\( \text{RUC: ref-union-closed} \, P \) and
\( \text{WFC: weakly-future-consistent} \, P \, I \, D \, R \) and
\( \text{LR: locally-respects} \, P \, I \, D \, R \) and
\( A : (xs \otimes [y], Y) \in \text{failures} \, P \)

shows \( (xs, \{x \in Y. \, (D \, y, D \, x) \notin I\}) \in \text{failures} \, P \)
proof (cases \( \exists x. \ x \in \{ x \in Y. \ (D \ y, \ D \ x) \notin I \} \))

let \(?A = \) singleton-set \( \{ x \in Y. \ (D \ y, \ D \ x) \notin I \} \)

have \( \forall xs \ A. \ (\exists X. \ X \in A) \longrightarrow (\forall X \in A. \ (xs, \ X) \in \) failures \( P \) \longrightarrow \\
\( (xs, \ \bigcup X \in ?A. \ X) \in \) failures \( P \)

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

hence \( (\exists X. \ X \in ?A) \longrightarrow (\forall X \in ?A. \ (xs, \ X) \in \) failures \( P \) \longrightarrow \\
\( (xs, \ \bigcup X \in ?A. \ X) \in \) failures \( P \)

by blast

moreover case True

hence \( \exists X. \ X \in ?A \) by (simp add: singleton-set-some)

ultimately have \( (\forall x \in \) ?A. \ (xs, \ X) \in \) failures \( P \) \longrightarrow \\
\( (xs, \ \bigcup X \in ?A. \ X) \in \) failures \( P \) ..

moreover have \( \forall X \in ?A. \ (xs, \ X) \in \) failures \( P \)

proof (rule ballI, simp add: singleton-set-def, erule exE, (erule conjE)+, simp)

fix \( x \)

have \( \forall u \in \) range \( D \cap (-I) \) " range \( D. \ \forall xs \ ys. \ (xs, \ ys) \in R \ u \longrightarrow \\
next-dom-events \( P \ D \ u \ xs = \) next-dom-events \( P \ D \ u \ ys \land \\
ref-dom-events \( P \ D \ u \ xs = \) ref-dom-events \( P \ D \ u \ ys \\
using WFC by (simp add: weakly-future-consistent-def)

moreover assume \( B: \ (D \ y, \ D \ x) \notin I \)

hence \( D \ x \in \) range \( D \cap (-I) \) " range \( D \) by (simp add: Image-iff, rule exI)

ultimately have \( \forall xs \ ys. \ (xs, \ ys) \in R \ (D \ x) \longrightarrow \\
next-dom-events \( P \ D \ (D \ x) \ xs = \) next-dom-events \( P \ D \ (D \ x) \ ys \land \\
ref-dom-events \( P \ D \ (D \ x) \ xs = \) ref-dom-events \( P \ D \ (D \ x) \ ys \) ..

hence \( C: \ (xs, \ xs \ @ [y]) \in R \ (D \ x) \longrightarrow \\
ref-dom-events \( P \ D \ (D \ x) \ xs = \) ref-dom-events \( P \ D \ (D \ x) \ (xs \ @ [\ y]) \\
by simp

have \( \forall xs \ y. \ (D \ y, \ D \ x) \notin I \land y \in \) next-events \( P \ xs \longrightarrow \\
\( (xs, \ xs \ @ [\ y]) \in R \ (D \ x) \\
using LR by (simp add: locally-respects-def)

hence \( (D \ y, \ D \ x) \notin I \land y \in \) next-events \( P \ xs \longrightarrow (xs, \ xs \ @ [\ y]) \in R \ (D \ x) \\
by blast

moreover have \( xs \ @ [\ y] \in \) traces \( P \) using \( A \) by (rule failures-traces)

hence \( y \in \) next-events \( P \ xs \) by (simp add: next-events-def)

ultimately have \( (xs, \ xs \ @ [y]) \in R \ (D \ x) \) using \( B \) by simp

with \( C \) have \( \) ref-dom-events \( P \ D \ (D \ x) \ xs = \)

ref-dom-events \( P \ D \ (D \ x) \ (xs \ @ [\ y]) \) ..

moreover assume \( D: \ x \in Y \)

have \( x \in \) ref-dom-events \( P \ D \ (D \ x) \ (xs \ @ [\ y]) \)

proof (simp add: ref-dom-events-def refusals-def)

have \( \{ x \} \subseteq Y \) using \( D \) by (simp add: i purge-ref-def)

with \( A \) show \( (xs \ @ [\ y], \ \{ x \}) \in \) failures \( P \) by (rule process-rule-3)

qed

ultimately have \( x \in \) ref-dom-events \( P \ D \ (D \ x) \ xs \) by simp

thus \( (xs, \ \{ x \}) \in \) failures \( P \) by (simp add: ref-dom-events-def refusals-def)

qed

ultimately have \( (xs, \ \bigcup X \in ?A. \ X) \in \) failures \( P \) ..

thus \( ?\)thesis by (simp only: singleton-set-union)
\[ \text{case False} \]

\[ \text{hence } \{ x \in Y . (D y, D x) \notin I \} = \{ \} \text{ by simp} \]

\[ \text{moreover have } (xs, \{ \}) \in \text{failures } P \text{ using } A \text{ by (rule process-rule-2)} \]

\[ \text{ultimately show } ?\text{thesis by (simp (no-asm-simp))} \]

\[ \text{qed} \]

\text{lemma ruc-wfc-lr-failures-2:} 

\begin{itemize}
  \item \text{assumes} \[ \text{RUC: ref-union-closed } P \text{ and} \]
  \item \text{WFC: weakly-future-consistent } P I D R \text{ and} \]
  \item \text{LR: locally-respects } P I D R \text{ and} \]
  \item \text{A: } (xs, Z) \in \text{failures } P \text{ and} \]
  \item \text{Y: } xs @ [y] \in \text{traces } P \]
\end{itemize}

\[ \text{shows } (xs @ [y], \{ x \in Z . (D y, D x) \notin I \}) \in \text{failures } P \]

\text{proof (cases } \exists x. x \in \{ x \in Z . (D y, D x) \notin I \})

\[ \text{let } ?A = \text{singleton-set } \{ x \in Z . (D y, D x) \notin I \} \]

\[ \text{have } \forall xs A. (\exists X. X \in A) \rightarrow (\forall X \in A. (xs, X) \in \text{failures } P) \rightarrow \]

\[ (xs, \bigcup X \in A. X) \in \text{failures } P \]

\[ \text{using RUC by (simp add: ref-union-closed-def)} \]

\[ \text{hence } (\exists X. X \in ?A) \rightarrow (\forall X \in ?A. (xs @ [y], X) \in \text{failures } P) \rightarrow \]

\[ (xs @ [y], \bigcup X \in ?A. X) \in \text{failures } P \]

\[ \text{by blast} \]

\text{moreover case True} 

\[ \text{hence } \exists X. X \in ?A \text{ by (simp add: singleton-set-some)} \]

\[ \text{ultimately have } (\forall X \in ?A. (xs @ [y], X) \in \text{failures } P) \rightarrow \]

\[ (xs @ [y], \bigcup X \in ?A. X) \in \text{failures } P \]

\[ \text{moreover have } \forall X \in ?A. (xs @ [y], X) \in \text{failures } P \]

\text{proof (rule ballI, simp add: singleton-set-def, erule exE, (erule conjE)+, simp)}

\[ \text{fix } x \]

\[ \text{have } \forall u \in \text{range } D \cap (-1) \leftarrow \text{range } D . \forall xs ys. (xs, ys) \in R u \rightarrow \]

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

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

\[ \text{using WFC by (simp add: weakly-future-consistent-def)} \]

\text{moreover assume B: } (D y, D x) \notin I 

\[ \text{hence } D x \in \text{range } D \cap (-1) \leftarrow \text{range } D \text{ by (simp add: Image-iff, rule exI)} \]

\[ \text{ultimately have } \forall xs ys. (xs, ys) \in R (D x) \rightarrow \]

\[ \text{next-dom-events } P D (D x) xs = \text{next-dom-events } P D (D x) ys \wedge \]

\[ \text{ref-dom-events } P D (D x) xs = \text{ref-dom-events } P D (D x) ys \]

\[ \text{hence } C: (xs, xs @ [y]) \in R (D x) \rightarrow \]

\[ \text{ref-dom-events } P D (D x) xs = \text{ref-dom-events } P D (D x) (xs @ [y]) \]

\[ \text{by simp} \]

\[ \text{have } \forall xs y. (D y, D x) \notin I \wedge y \in \text{next-events } P xs \rightarrow \]

\[ (xs, xs @ [y]) \in R (D x) \]

\[ \text{using LR by (simp add: locally-respects-def)} \]

\[ \text{hence } (D y, D x) \notin I \wedge y \in \text{next-events } P xs \rightarrow (xs, xs @ [y]) \in R (D x) \]

\[ \text{by blast} \]

\[ \text{moreover have } y \in \text{next-events } P xs \text{ using Y by (simp add: next-events-def)} \]

\[ \text{ultimately have } (xs, xs @ [y]) \in R (D x) \text{ using } B \text{ by simp} \]

\[ \text{with } C \text{ have ref-dom-events } P D (D x) xs = \]

10
ref-dom-events \( P D (D x) \) (\( zs \oplus [y] \)) ..

moreover assume \( D : x \in Z \)
have \( x \in \text{ref-dom-events} \ P D (D x) \) \( zs \)
proof (simp add: ref-dom-events-def refusals-def)
  have \( \{x\} \subseteq Z \) using \( D \) by (simp add: ipurge-ref-def)
  with \( A \) show \( (xs, \{x\}) \in \text{failures} \ P \) by (rule process-rule-3)
qed
ultimately have \( x \in \text{ref-dom-events} \ P D (D x) \) (\( zs \oplus [y] \)) by simp
thus \( (zs \oplus [y], \{x\}) \in \text{failures} \ P \) by (simp add: ref-dom-events-def refusals-def)
qed
ultimately have \( (xs \oplus [y], \bigcup X \in ?A. X) \in \text{failures} \ P \) ..
thus \(?thesis \) by (simp only: singleton-set-union)

next
case False
hence \( \{x \in Z. (D y, D x) \notin I\} = \{\} \) by simp
moreover have \( (xs \oplus [y], \{\}) \in \text{failures} \ P \) using \( Y \) by (rule traces-failures)
ultimately show \(?thesis \) by (simp (no-asmp-simp))
qed

lemma gu-condition-imply-secure-aux [rule-format]:
assumes
  VP: \( \text{view-partition} \ P D R \) and
  WFC: \( \text{weakly-future-consistent} \ P I D R \) and
  WSC: \( \text{weakly-step-consistent} \ P D R \) and
  LR: \( \text{locally-respects} \ P I D R \)
shows \( U \subseteq \text{range} \ D \longrightarrow U \neq \{\} \longrightarrow xs \oplus ys \in \text{traces} \ P \longrightarrow \)
  \( (\forall u \in \text{unaffected-domains} \ I D U []). (zs, xs') \in R u \longrightarrow \)
  \( (\forall u \in \text{unaffected-domains} \ I D U ys. \) \( (xs \oplus ys, xs' \oplus \text{ipurge-tr-aux} I D U ys) \in R u) \)
proof (induction \( ys \) arbitrary: \( xs \) \( xs' \) \( U \), simp-all add: unaffected-domains-def, ((rule impI)+, (rule allI)+), erule conjE)
fix \( y \) \( ys \) \( xs \) \( xs' \) \( U \) \( u \)
assume
A: \( \forall xs xs' U. U \subseteq \text{range} \ D \longrightarrow U \neq \{\} \longrightarrow xs \oplus ys \in \text{traces} \ P \longrightarrow \)
  \( (\forall u. u \in \text{range} \ D \land (\forall v \in U. (v, u) \notin I) \longrightarrow \)
  \( (xs, xs') \in R u) \longrightarrow \)
  \( (\forall u. u \in \text{range} \ D \land (\forall v \in \text{sinks-aux} I D U ys. (v, u) \notin I) \longrightarrow \)
  \( (xs \oplus ys, xs' \oplus \text{ipurge-tr-aux} I D U ys) \in R u) \) \( \text{and} \)
B: \( U \subseteq \text{range} \ D \) \( \text{and} \)
C: \( U \neq \{\} \) \( \text{and} \)
D: \( xs \oplus y \neq ys \in \text{traces} \ P \) \( \text{and} \)
E: \( \forall u. u \in \text{range} \ D \land (\forall v \in U. (v, u) \notin I) \longrightarrow (xs, xs') \in R u \) \( \text{and} \)
F: \( u \in \text{range} \ D \) \( \text{and} \)
G: \( \forall v \in \text{sinks-aux} I D U (y \neq ys). (v, u) \notin I \)
show \( (xs \oplus y \neq ys, xs' \oplus \text{ipurge-tr-aux} I D U (y \neq ys)) \in R u \)
proof (cases \( \exists v \in U. (v, D y) \in I, \)
  simp-all (no-asmp-simp) add: ipurge-tr-aux-cons)
case True

11
\[
\text{let } \forall U' = \text{insert } (D \ y) U \\
\text{have } \forall U' \subseteq D \rightarrow \forall U' \neq \{\} \rightarrow (xs @ [y]) @ ys \in \text{traces } P \rightarrow \\
(\forall u. \ u \in \text{range } D \land (\forall v \in \forall U'. \ (v, u) \notin I)) \rightarrow \\
(xs @ [y], xs') \in R u \rightarrow \\
(\forall u. \ u \in \text{range } D \land (\forall v \in \text{sinks-\text{-}aux } I \ D \ ?U'. \ ys. \ (v, u) \notin I)) \rightarrow \\
(xs @ [y]) @ ys, xs' @ \text{ipurge-\text{-}aux } I \ D \ ?U'. \ ys) \in R u \\
\text{using } A . \\
hence \\
(\forall u. \ u \in \text{range } D \land (\forall v \in \forall U'. \ (v, u) \notin I)) \rightarrow \\
(xs @ [y], xs') \in R u \\
\text{using } B \text{ and } D \text{ by } \text{simp} \\
\text{moreover have } \\
\forall u. \ u \in \text{range } D \land (\forall v \in \forall U'. \ (v, u) \notin I) \rightarrow \\
(xs @ [y], xs') \in R u \\
\text{proof } (\text{rule allI, rule impI, erule conjE}) \\
\text{fix } u \\
\text{assume } F: \ u \in \text{range } D \text{ and } G: \ (\forall v \in \forall U'. \ (v, u) \notin I) \\
\text{moreover have } u \in \text{range } D \land (\forall v \in U. \ (v, u) \notin I) \rightarrow (xs, xs') \in R u \text{ using } E .. \\
\text{ultimately have } H: \ (xs, xs') \in R u \text{ by } \text{simp} \\
\text{have } \forall u \in \text{range } D. \ (D y, u) \notin I \land y \in \text{next\text{-}events } P xs \rightarrow \\
(xs, xs @ [y]) \in R u \text{ using } \text{LR by } (\text{simp add: locally\text{-}respects-def}) \\
\text{hence } (D y, u) \notin I \land y \in \text{next\text{-}events } P xs \rightarrow (xs, xs @ [y]) \in R u \text{ using } F .. \\
\text{moreover have } D y \in \forall U' \text{ by } \text{simp} \\
\text{with } G \text{ have } (D y, u) \notin I .. \\
\text{moreover have } (xs @ [y]) @ ys \in \text{traces } P \text{ using } D \text{ by } \text{simp} \\
\text{hence } y \in \text{next\text{-}events } P xs \text{ by } (\text{simp (no-asn-simp) add: next\text{-}events-def, rule process-rule-2-traces}) \\
\text{ultimately have } I: \ (xs, xs @ [y]) \in R u \text{ by } \text{simp} \\
\text{have } \forall u \in \text{range } D. \ \text{equiv (traces } P \text{) (R u) using } \text{VP by } (\text{simp add: view\text{-}partition-def}) \\
\text{hence } J: \ \text{equiv (traces } P \text{) (R u) using } F .. \\
\text{hence } \text{trans (R u) by } (\text{simp add: equiv-def}) \\
\text{moreover have } \text{sym (R u) using } J \text{ by } (\text{simp add: equiv-def}) \\
\text{hence } (xs @ [y], xs) \in R u \text{ using } I \text{ by } (\text{rule symE}) \\
\text{ultimately show } (xs @ [y], xs') \in R u \text{ using } H \text{ by } (\text{rule transE}) \\
\text{qed} \\
\text{ultimately have } \\
\forall u. \ u \in \text{range } D \land (\forall v \in \text{sinks-\text{-}aux } I \ D \ ?U'. \ ys. \ (v, u) \notin I) \rightarrow \\
(xs @ y \neq ys, xs' @ \text{ipurge-\text{-}aux } I \ D \ ?U'. \ ys) \in R u .. \\
hence \\
u \in \text{range } D \land (\forall v \in \text{sinks-\text{-}aux } I \ D \ ?U'. \ ys. \ (v, u) \notin I) \rightarrow \\
(xs @ y \neq ys, xs' @ \text{ipurge-\text{-}aux } I \ D \ ?U'. \ ys) \in R u .. \\
\text{moreover have } \text{sinks-\text{-}aux } I \ D \ U (y \neq ys) = \text{sinks-\text{-}aux } I \ D \ ?U'. \ ys \\
\text{using } \text{Cons and True by } (\text{simp add: sinks\text{-}aux\text{-}cons})
\]

12
hence $\forall v \in \text{sinks-aux } I \ D \ ?U' \ ys. \ (v, u) \notin I$ using $G$ by simp

with $F$ have $u \in \text{range } D \land (\forall v \in \text{sinks-aux } I \ D \ ?U' \ ys. \ (v, u) \notin I)$ ..

ultimately show $(xs \ @ y \ # \ ys, xs' \ @ \text{ipurge-tr-aux } I \ D \ ?U' \ ys) \in R \ u$ ..

next

case False

have

$U \subseteq \text{range } D \rightarrow U \neq \{\} \rightarrow (xs \ @ [y]) \in \text{traces } P \rightarrow$

$(\forall u. \ u \in \text{range } D \land (\forall v \in U. \ (v, u) \notin I) \rightarrow$

$(xs \ @ [y], xs' \ @ [y]) \in R \ u) \rightarrow$

$(\forall u. \ u \in \text{range } D \land (\forall v \in \text{sinks-aux } I \ D \ U \ ys. \ (v, u) \notin I) \rightarrow$

$((xs \ @ [y]) \in \text{ipurge-tr-aux } I \ D \ U \ ys) \in R \ u)$

using $\text{A}$ ..

hence

$(\forall u. \ u \in \text{range } D \land (\forall v \in U. \ (v, u) \notin I) \rightarrow$

$(xs \ @ [y], xs' \ @ [y]) \in R \ u) \rightarrow$

$(\forall u. \ u \in \text{range } D \land (\forall v \in \text{sinks-aux } I \ D \ U \ ys. \ (v, u) \notin I) \rightarrow$

$(xs \ @ y \ # \ ys, xs' \ @ y \ # \text{ipurge-tr-aux } I \ D \ U \ ys) \in R \ u)$

using $\text{B and C and D by simp}$

moreover have

$\forall u. \ u \in \text{range } D \land (\forall v \in U. \ (v, u) \notin I) \rightarrow$

$(xs \ @ [y], xs' \ @ [y]) \in R \ u$

proof (rule allI, rule impI, erule conjE)

fix $u$

assume $F$: $u \in \text{range } D$ and $G$: $\forall v \in U. \ (v, u) \notin I$

moreover have $u \in \text{range } D \land (\forall v \in U. \ (v, u) \notin I) \rightarrow (xs, xs') \in R \ u$

using $E$ ..

ultimately have $(xs, xs') \in R \ u$ by simp

moreover have $D y \in \text{range } D \land$

$(\forall v \in U. \ (v, D y) \notin I) \rightarrow (xs, xs') \in R \ (D y)$

using $E$ ..

hence $(xs, xs') \in R \ (D y)$ using False by simp

ultimately have $H$: $(xs, xs') \in R \ u \cap R \ (D y)$ ..

have $\exists v. \ v \in U$ using $C$ by (simp add: ex-in-conv)

then obtain $v$ where $I$: $v \in U$ ..

hence $(v, D y) \in \{\} \ using \ False \ by \ simp$

moreover have $v \in \text{range } D$ using $B$ and $I$ ..

ultimately have $D y \in \{\} \ using \ False \ by \ simp$

hence $\forall u \in \text{range } D \cap \{\} \ using \ False \ by \ simp$

have $\forall u \in \text{range } D \cap \{\} \ using \ False \ by \ simp$

ultimately have $J$: $D y \in \text{range } D \cap \{\} \ using \ False \ by \ simp$

hence $\forall xs \ys. \ (xs, ys) \in R \ (D y)$ ..

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

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

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

hence $\forall xs \ys. \ (xs, ys) \in R \ (D y)$ ..

next-dom-events $P \ D \ (D y) \ ys = \text{next-dom-events } P \ D \ (D y) \ ys \land$

ref-dom-events $P \ D \ (D y) \ ys = \text{ref-dom-events } P \ D \ (D y) \ ys$

using $J$ ..

hence $(xs, xs') \in R \ (D y)$ ..

next-dom-events $P \ D \ (D y) \ ys = \text{next-dom-events } P \ D \ (D y) \ xs' \land$

ref-dom-events $P \ D \ (D y) \ ys = \text{ref-dom-events } P \ D \ (D y) \ xs'${}
by blast
hence next-dom-events P D (D y) xs = next-dom-events P D (D y) xs'
using H by simp
moreover have (xs @ [y]) @ ys ∈ traces P using D by simp
hence K: y ∈ next-events P xs
by (simp (no-asm-simp) add: next-events-def, rule process-rule-2-traces)
hence y ∈ next-dom-events P D (D y) xs
by (simp add: next-dom-events-def)
ultimately have y ∈ next-events P xs' by (simp add: next-dom-events-def)
with K have L: y ∈ next-events P xs ∩ next-events P xs' ..
have ∀ u ∈ range D. ∀ xs ys x.
  (xs, ys) ∈ R u ∩ R (D x) ∧ x ∈ next-events P xs ∩ next-events P ys →
  (xs @ [x], ys @ [x]) ∈ R u
using WSC by (simp add: weakly-step-consistent-def)
hence ∀ xs ys x.
  (xs, ys) ∈ R u ∩ R (D x) ∧ x ∈ next-events P xs ∩ next-events P ys →
  (xs @ [x], ys @ [x]) ∈ R u
using F ..
hence
  (xs, xs') ∈ R u ∩ R (D y) ∧ y ∈ next-events P xs ∩ next-events P xs' →
  (xs @ [y], xs' @ [y]) ∈ R u
by blast
thus (xs @ [y], xs' @ [y]) ∈ R u using H and L by simp
qed
ultimately have
  ∀ u. u ∈ range D ∧ (∀ v ∈ sinks-aux I D U ys. (v, u) ∈ I) →
  (xs @ y # ys, xs' @ y # ipurge-tr-aux I D U ys) ∈ R u ..
hence u ∈ range D ∧ (∀ v ∈ sinks-aux I D U ys. (v, u) ∈ I) →
  (xs @ y # ys, xs' @ y # ipurge-tr-aux I D U ys) ∈ R u ..
morerover have sinks-aux I D U (y # ys) = sinks-aux I D U ys
using Cons and False by (simp add: sinks-aux-cons)
hence ∀ v ∈ sinks-aux I D U ys. (v, u) ∈ I using G by simp
with F have u ∈ range D ∧ (∀ v ∈ sinks-aux I D U ys. (v, u) ∈ I) ..
ultimately show (xs @ y # ys, xs' @ y # ipurge-tr-aux I D U ys) ∈ R u ..
qed
qed

lemma gu-condition-imply-secure-1 [rule-format]:
assumes
  RUC: ref-union-closed P and
  VP: view-partition P D R and
  WFC: weakly-future-consistent P I D R and
  WSC: weakly-step-consistent P D R and
  LR: locally-respects P I D R
shows (xs @ y # ys, Y) ∈ failures P →
  (xs @ ipurge-tr I D (D y) ys, ipurge-ref I D (D y) ys Y) ∈ failures P
proof (induction ys arbitrary: Y rule: rev-induct, rule-tac [!] impI,
  simp add: ipurge-ref-def)
fix Y
proof (cases D y' ∈ sinks I D (D y) (ys @ [y'])), simp del: sinks.simps)
let ?Y' = {x ∈ Y. (D y', D x) ∉ I}

have (xs @ y ≠ ys, ?Y') ∈ failures P →
(x @ ipurge-tr I D (D y) ys, ipurge-ref I D (D y) ys ?Y') ∈ failures P

moreover assume (x @ y ≠ ys, ?Y') ∈ failures P

moreover have (xs @ y ≠ ys, ?Y') ∈ failures P
using (rule unaffected-domains-subset, simp-all)

moreover have ?U ≠ {}

moreover have (D y, D y') ∉ I using False by (rule_tac notI, simp)

ultimately show ?U ≠ {}

apply (simp (no-asm-simp) add: unaffected-domains-def sinks-aux-single-dom)

set-eq-iff
qed

moreover have C: xs @ y ≠ ys @ [y'] ∈ traces P
using B by (rule failures-traces)

have ∀ u ∈ ?U. (xs @ [y]) @ ys @ [y'], xs @ ipurge-tr-aux I D (D y) (ys @ [y']) ∈ R u
proof (rule ballI, rule gu-condition-imply-secure-aux [OF VP WFC WSC LR], simp-all add: unaffected-domains-def C, (erule conjE)+)

fix u
have \( \forall u \in \text{range } D. \forall xs x. \) 
\((D x, u) \notin I \wedge x \in \text{next-events } P \) 
\(xs \rightarrow (xs, xs \oplus [x]) \in R u\)

using LR by (simp add: locally-respects-def)

moreover assume \(D: u \in \text{range } D\)

ultimately have \(\forall xs x.\)
\((D x, u) \notin I \wedge x \in \text{next-events } P \) 
\(xs \rightarrow (xs, xs \oplus [y]) \in R u\)

by blast

moreover assume \((D y, u) \notin I\)

moreover have \(xs \oplus [y] \oplus ys \oplus [y'] \in \text{traces } P \) using C by simp

hence \(xs \oplus [y] \in \text{traces } P\) by (rule process-rule-2-traces)

hence \(y \in \text{next-events } P \) by (simp add: next-events-def)

ultimately have \(E: (xs, xs \oplus [y]) \in R u\) by simp

have \(\forall u \in \text{range } D. \equiv (\text{traces } P) (R u)\)

using VP by (simp add: view-partition-def)

hence \(\equiv (\text{traces } P) (R u)\) using D ..

hence \(\equiv (\text{traces } P) (R u)\) by (simp add: equiv-def)

thus \((xs \oplus [y], xs) \in R u\) using E by (rule symE)

qed

lemma gu-condition-imply-secure-2 [rule-format]:

assumes
\(\text{RUC: ref-union-closed } P \) and
\(\text{VP: view-partition } P D R \) and
\(\text{WFC: weakly-future-consistent } P I D R \) and
\(\text{WSC: weakly-step-consistent } P D R \) and
\(\text{LR: locally-respects } P I D R \) and
\(Y: \text{xs} \oplus [y] \in \text{traces } P\)

shows \((xs @ zs, Z) \in \text{failures } P \rightarrow \)
\((xs @ y \# \text{ipurge-tr } I D (D y) \) zs, \text{ipurge-ref } I D (D y) \) zs Z \in \text{failures } P\)

proof (induction zs arbitrary: Z rule: rev-induct, rule-tac !! impI, simp add: ipurge-ref-def)

fix Z
assume \((zs, Z) \in \text{failures } P\)

with \(\text{RUC}\) and \(\text{WFC}\) and \(\text{LR}\) show
\((xs @ [y], \{x \in Z. (D y, D x) \notin I\}) \in \text{failures } P\)

using \(Y\) by (rule \(\text{ruc-wfc-lr-failures-2}\))

next

fix \(z\) zs \(Z\)

assume
\(A: \bigwedge \ Z'. (zs @ zs, Z') \in \text{failures } P \implies (xs @ y \neq \text{ipurge-tr } I D (D y) zs,\) \ipurge-ref I D (D y) zs Z' \in \text{failures } P\) and
\(B: (xs @ zs @ [z], Z) \in \text{failures } P\)

show \((xs @ y \neq \text{ipurge-tr } I D (D y) (zs @ [z]),\)
\ipurge-ref I D (D y) (zs @ [z]) Z) \in \text{failures } P\)

proof
(cases \(D z \in \text{sinks } I D (D y) (zs @ [z]), \) simp del: sinks.simps)

let \(\overline{?Z'} = \{x \in Z. (D z, D x) \notin I\}\)

have \((xs @ zs, \overline{?Z'}) \in \text{failures } P \implies (xs @ y \neq \text{ipurge-tr } I D (D y) zs, \ipurge-ref I D (D y) zs \overline{?Z'}) \in \text{failures } P\)

using \(A\).

moreover have \((xs @ zs) @ [z], Z) \in \text{failures } P\) using \(B\) by simp

with \(\text{RUC}\) and \(\text{WFC}\) and \(\text{LR}\) have \((xs @ zs, \overline{?Z'}) \in \text{failures } P\)

by (rule \(\text{ruc-wfc-lr-failures-1}\))

ultimately have
\((xs @ y \neq \text{ipurge-tr } I D (D y) zs, \ipurge-ref I D (D y) zs \overline{?Z'}) \in \text{failures } P\) ..

moreover case \(\text{True}\)

hence \(\ipurge-ref I D (D y) (zs @ [z]) Z = \ipurge-ref I D (D y) zs \overline{?Z'}\)

by (rule \(\text{ipurge-ref-eq}\))

ultimately show
\((xs @ y \neq \text{ipurge-tr } I D (D y) zs, \ipurge-ref I D (D y) (zs @ [z]) Z) \in \text{failures } P\)

by simp

next

case \(\text{False}\)

have unaffected-domains \(I D \{D y\} (zs @ [z]) \subseteq \text{range } D \cap (-I) \) \(\cap \text{range } D\)

(is \(\overline{?U} \subseteq \cdot\))

by (rule unaffected-domains-subset, simp-all)

moreover have \(\overline{?U} \neq \{\}\)

proof —

have \((D y, D z) \notin I\) using \(\text{False}\) by (rule-tac notI, simp)

moreover

have \(\neg ((D y, D z) \in I \lor (\exists v \in \text{sinks } I D (D y) zs. (v, D z) \in I))\)

using \(\text{False}\) by (simp only: sinks-interference-eq, simp)

then have \(\forall v \in \text{sinks } I D (D y) (zs @ [z]). (v, D z) \notin I\) by simp

ultimately show \(\overline{?U} \neq \{\}\)

apply (simp (no-asmp-simp) add: unaffected-domains-def sinks-aux-single-dom)

set-eq-iff

using \((D y, D z) \notin I\) by auto

qed

moreover have \(C: xs @ zs @ [z] \in \text{traces } P\) using \(B\) by (rule failures-traces)

have \(\forall u \in \overline{?U}. (xs @ zs @ [z]),\)
\[(xs @ [y]) @ \text{ipurge-tr-aux} I D \{ D y \} (zs @ [z])) \in R u\]

**Proof**: (rule \text{ballI}, rule \text{gu-condition-imply-secure-aux} \[OF \text{VP WFC WSC LR},\]
\text{simp-all add: unaffected-domains-def C, (erule conjE)+)

fix \(u\)

have \(\forall u \in \text{range } D. \forall xs\).

\((D x, u) \notin I \land x \in \text{next-events } P xs \rightarrow (xs, xs @ [x]) \in R u\)

using \(LR\) by (simp add: \text{locally-respects-def})

moreover assume \(D: u \in \text{range } D\)

ultimately have \(\forall xs x.

(D x, u) \notin I \land x \in \text{next-events } P xs \rightarrow (xs, xs @ [y]) \in R u\)

by \(\text{blast}\)

moreover have \(y \in \text{next-events } P xs\) using \(Y\) by (simp add: \text{next-events-def})

ultimately show \(\exists U. (xs, xs @ [y]) \in R u\)

by (simp only: unaffected-domains-single-dom)

qed

**Theorem**: \text{generic-unwinding:}

assumes
\(RUC: \text{ref-union-closed } P \text{ and}\)
\(VP: \text{view-partition } P D R \text{ and}\)
\(WFC: \text{weakly-future-consistent } P I D R \text{ and}\)
\(WSC: \text{weakly-step-consistent } P D R \text{ and}\)
\(LR: \text{locally-respects } P I D R\)

shows \(\text{secure } P I D\)

**Proof**: (simp add: \text{secure-def futures-def}, (rule allI)+, rule \text{impI}, erule conjE)

fix \(xs \ y \ ys \ Z\)

assume
\(A: (xs @ y \neq ys, Y) \in \text{failures } P \text{ and}\)
\(B: (xs @ zs, Z) \in \text{failures } P\)

show \((xs @ \text{ipurge-tr } I D \{ D y \} ys, \text{ipurge-ref } I D \{ D y \} ys Y) \in \text{failures } P \land\)
\((xs @ y \neq \text{ipurge-tr } I D \{ D y \} zs, \text{ipurge-ref } I D \{ D y \} zs Z) \in \text{failures } P\)

(is \(?P \land ?Q\))

**Proof**: show \(?P\) using \(RUC\) and \(VP\) and \(WFC\) and \(WSC\) and \(LR\) and \(A\)

by (rule \text{gu-condition-imply-secure-I})

next

have \(((xs @ [y]) @ ys, Y) \in \text{failures } P\) using \(A\) by \(\text{simp}\)
hence \((xs @ [y], \{\}) \in failures P\) by (rule process-rule-2-failures)

hence \(xs @ [y] \in traces P\) by (rule failures-traces)

with \(\text{RUC and VP and WFC and WSC and LR}\) show \(?Q\) using \(B\)

by (rule gu-condition-imply-secure-2)

\[\text{qed}\]

\[\text{qed}\]

It is interesting to observe that unlike symmetry and transitivity, the assumed reflexivity of the relations in the range of the domain-relation map is never used in the proof of the Generic Unwinding Theorem. Nonetheless, by assuming that such relations be equivalence relations over process traces rather than just symmetric and transitive ones, reflexivity has been kept among assumptions for both historical reasons – Rushby’s Unwinding Theorem for deterministic state machines deals with equivalence relations – and practical reasons – predicate \(\text{refl-on (traces P)}\) may only be verified by a relation included in \(\text{traces P} \times \text{traces P}\), thus ensuring that traces be not correlated with non-trace event lists, which is a necessary condition for weak future consistency (cf. [7]).

Here below are convenient variants of the Generic Unwinding Theorem applying to deterministic processes and trace set processes (cf. [7]).

**Theorem d-generic-unwinding:**

\[
\begin{align*}
\text{deterministic P} & \implies \text{view-partition P D R} \\
\text{d-weakly-future-consistent P I D R} & \implies \\
\text{weakly-step-consistent P D R} & \implies \\
\text{locally-respects P I D R} & \implies \\
\text{secure P I D}
\end{align*}
\]

**Proof** (rule generic-unwinding, rule d-implies-ruc, simp-all)

\[\text{qed}\] (drule d-wfc-equals-dwfc [of P I D R], simp)

**Theorem ts-generic-unwinding:**

\[
\begin{align*}
\text{trace-set T} & \implies \text{view-partition (ts-process T) D R} \\
\text{d-weakly-future-consistent (ts-process T) I D R} & \implies \\
\text{weakly-step-consistent (ts-process T) D R} & \implies \\
\text{locally-respects (ts-process T) I D R} & \implies \\
\text{secure (ts-process T) I D}
\end{align*}
\]

**Proof** (rule d-generic-unwinding, simp-all)

\[\text{qed}\] (rule ts-process-d)

1.3 The Generic Unwinding Theorem: counterexample to condition necessity

At a first glance, it seems reasonable to hypothesize that the Generic Unwinding Theorem expresses a necessary, as well as sufficient, condition for
security, viz. that whenever a process is secure with respect to a policy, there should exist a set of "views" of process traces, one per domain, satisfying the apparently simple assumptions of the theorem.

It can thus be surprising to discover that this hypothesis is false, as proven in what follows by constructing a counterexample. The key observation for attaining this result is that symmetry, transitivity, weak step consistency, and local policy respect permit to infer the correlation of pairs of traces, and can then be given the form of introduction rules in the inductive definition of a set. In this way, a "minimum" domain-relation map rel-induct is obtained, viz. a map such that, for each domain u, the image of u under this map is included in the image of u under any map which has the aforesaid properties – particularly, which satisfies the assumptions of the Generic Unwinding Theorem.

Although reflexivity can be given the form of an introduction rule, too, it has been omitted from the inductive definition. This has been done in order to ensure that the "minimum" domain-relation map, and consequently the counterexample as well, still remain such even if reflexivity, being unnecessary (cf. above), were removed from the assumptions of the Generic Unwinding Theorem.

\[
\text{inductive-set rel-induct-aux ::} \\
\quad \text{'}a \text{ process } \Rightarrow \text{('}d \times \text{'}d\text{) set } \Rightarrow \text{('}a \Rightarrow \text{'}d\text{) } \Rightarrow \text{('}d \times \text{'}a \text{ list } \times \text{'}a \text{ list) set}
\]

\text{for P :: 'a process and I :: ('}d \times \text{'}d\text{) set and D :: 'a } \Rightarrow \text{'}d \text{ where}

\quad \text{rule-sym: } \quad (u, xs, ys) \in \text{rel-induct-aux P I D } \Rightarrow \\
\quad \quad (u, ys, xs) \in \text{rel-induct-aux P I D} \\
\quad \text{rule-trans: } \quad [(u, xs, ys) \in \text{rel-induct-aux P I D}; \\
\quad (u, ys, zs) \in \text{rel-induct-aux P I D}] \Rightarrow \\
\quad \quad (u, xs, zs) \in \text{rel-induct-aux P I D} \\
\quad \text{rule-WSC: } \quad [(u, xs, ys) \in \text{rel-induct-aux P I D}; \\
\quad (D x, xs, ys) \in \text{rel-induct-aux P I D}; \\
\quad x \in \text{next-events P xs } \cap \text{next-events P ys}] \Rightarrow \\
\quad \quad (u, xs \oplus [x], ys \oplus [x]) \in \text{rel-induct-aux P I D} \\
\quad \text{rule-LR: } \quad [(u \in \text{range D}; (D x, u) \notin I; x \in \text{next-events P xs}] \Rightarrow \\
\quad \quad (u, xs, xs \oplus [x]) \in \text{rel-induct-aux P I D}
\]

\text{definition rel-induct ::}

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

\quad \text{rel-induct P I D u } \equiv \text{rel-induct-aux P I D } \uplus \{u\}

\text{lemma rel-induct-subset:}

\quad \text{assumes}

\quad \quad VP: \text{ view-partition P D R and}

\quad \quad \text{WSC: weakly-step-consistent P D R and}

\quad \quad \text{LR: locally-respects P I D R}

\quad \text{shows rel-induct P I D u } \subseteq \text{ R u}

\text{proof (rule subsetI, simp add: rel-induct-def split-paired-all),}
erule rel-induct-aux.induct)

fix u xs ys
have \( \forall u \in \text{range } D. \text{ equiv (traces } P) \ (R u) \)
  using VP by (simp add: view-partition-def)
moreover assume \((u, xs, ys) \in \text{rel-induct-aux } P \text{ I } D\)

hence \( u \in \text{range } D \) by (rule rel-induct-aux.induct)
ultimately have equiv (traces \( P \)) \( (R u) .. \)

hence sym \((R u) \) by (simp add: equiv-def)
moreover assume \((xs, ys) \in R u \)
ultimately show \((ys, xs) \in R u \) by (rule symE)

next
fix u xs ys z
have \( \forall u \in \text{range } D. \text{ equiv (traces } P) \ (R u) \)
  using VP by (simp add: view-partition-def)
moreover assume \((u, xs, ys) \in \text{rel-induct-aux } P \text{ I } D\)

hence \( u \in \text{range } D \) by (rule rel-induct-aux.induct)
ultimately have equiv (traces \( P \)) \( (R u) .. \)

hence trans \((R u) \) by (simp add: equiv-def)
moreover assume \((xs, ys) \in R u \) and \((ys, zs) \in R u \)
ultimately show \((xs, zs) \in R u \) by (rule transE)

next
fix u xs ys x
have \( \forall u \in \text{range } D, \forall xs ys x. \)
  \((xs, ys) \in R u \cap R (D x) \land x \in \text{next-events } P \text{ xs } \cap \text{next-events } P \text{ ys } \rightarrow \)
  \((xs \oplus [x], ys \oplus [x]) \in R u \)
  using WSC by (simp add: weakly-step-consistent-def)
moreover assume \((u, xs, ys) \in \text{rel-induct-aux } P \text{ I } D\)

hence \( u \in \text{range } D \) by (rule rel-induct-aux.induct)
ultimately have \( \forall xs ys x. \)
  \((xs, ys) \in R u \cap R (D x) \land x \in \text{next-events } P \text{ xs } \cap \text{next-events } P \text{ ys } \rightarrow \)
  \((xs \oplus [x], ys \oplus [x]) \in R u \ .. \)

hence \((xs, ys) \in R u \cap R (D x) \land x \in \text{next-events } P \text{ xs } \cap \text{next-events } P \text{ ys } \rightarrow \)
  \((xs \oplus [x], ys \oplus [x]) \in R u \)
  by blast
moreover assume \((xs, ys) \in R u \) and \((xs, ys) \in R (D x) \) and
  \( x \in \text{next-events } P \text{ xs } \cap \text{next-events } P \text{ ys} \)
ultimately show \((xs \oplus [x], ys \oplus [x]) \in R u \) by simp

next
fix u xs x
have \( \forall u \in \text{range } D, \forall xs x. \)
  \((D x, u) \notin I \land x \in \text{next-events } P \text{ xs } \rightarrow (xs, xs \oplus [x]) \in R u \)
  using LR by (simp add: locally-respects-def)
moreover assume \( u \in \text{range } D \)
ultimately have \( \forall xs x. \)
  \((D x, u) \notin I \land x \in \text{next-events } P \text{ xs } \rightarrow (xs, xs \oplus [x]) \in R u \ .. \)

hence \((D x, u) \notin I \land x \in \text{next-events } P \text{ xs } \rightarrow (xs, xs \oplus [x]) \in R u \) by blast
moreover assume \((D x, u) \notin I \) and \( x \in \text{next-events } P \text{ xs} \)
ultimately show \((xs, xs @ [x]) \in R u\) by simp
qed

The next step consists of the definition of a trace set \(T_c\), the corresponding trace set process \(P_c\) (cf. [7]), and a reflexive, intransitive noninterference policy \(I_c\) for this process, where subscript "c" stands for "counterexample". As event-domain map, the identity function is used, which explains why the policy is defined over events themselves.

datatype \(event_c = a_c | b_c | c_c\)

definition \(T_c :: event_c list set\) where \(T_c \equiv \{\}\)
\([a_c], [a_c, b_c], [a_c, b_c, c_c], [a_c, b_c, c_c, a_c], [b_c], [b_c, a_c], [b_c, c_c], [b_c, a_c, c_c]\)\

definition \(P_c :: event_c process\) where \(P_c \equiv ts-process T_c\)

definition \(I_c :: (event_c \times event_c) set\) where \(I_c \equiv \{(a_c, a_c), (b_c, b_c), (b_c, c_c), (c_c, c_c), (c_c, a_c)\}\)

Process \(P_c\) can be shown to be secure with respect to policy \(I_c\). This result can be obtained by applying the Ipurge Unwinding Theorem, in the version for trace set processes [7], and then performing an exhaustive case distinction over all traces and domains, which obviously is possible by virtue of their finiteness.

Nevertheless, \(P_c\) and \(I_c\) are such that there exists no domain-relation map satisfying the assumptions of the Generic Unwinding Theorem. A proof \textit{ad absurdum} is given, based on the fact that the pair of traces \([a_c, b_c, c_c], [b_c, a_c, c_c]\) can be shown to be contained in the image of \(a_c\) under the "minimum" domain-relation map \(\text{rel-induct}\). Therefore, it would also be contained in the image of \(a_c\) under a map satisfying the assumptions of the Generic Unwinding Theorem, so that according to weak future consistency, \(a_c\) should be a possible subsequent event for trace \([a_c, b_c, c_c]\) just in case it were such for trace \([b_c, a_c, c_c]\). However, this conclusion contradicts the fact that \(a_c\) is a possible subsequent event for the former trace only.

lemma \(\text{counterexample-trace-set:}\)
\(\text{trace-set } T_c\)
by (simp add: \text{trace-set-def } T_c-def)

lemma \(\text{counterexample-next-events-1:}\)
\((x \in \text{next-events } (ts-process } T_c) \; xs) = (xs @ [x] \in T_c)\)
by (rule ts-process-next-events, rule counterexample-trace-set)

lemma counterexample-next-events-2:
\( (x \in \text{next-events } P_c \, xs) = (xs \oplus [x] \in T_c) \)
by (subst \( P_c \)-def, rule counterexample-next-events-1)

lemma counterexample-secure:
secure \( P_c \, I_c \id \)
proof (simp add: \( P_c \)-def ts-ipurge-unwinding [OF counterexample-trace-set]
dfc-equals-dwfc-rel-ipurge [symmetric] d-future-consistent-def, (rule allI)+)
fix \( u \, xs \, ys \)
show \( (xs, ys) \in \text{rel-ipurge } (\text{ts-process } T_c) \, I_c \id u \longrightarrow\)
\((xs \in \text{traces } (\text{ts-process } T_c)) = (ys \in \text{traces } (\text{ts-process } T_c)) \wedge \)
next-dom-events (ts-process T_c) id u xs =
next-dom-events (ts-process T_c) id u ys
proof (simp add: rel-ipurge-def ts-process-traces [OF counterexample-trace-set]
next-dom-events-def counterexample-next-events-1)
show \( xs \in T_c \wedge ys \in T_c \wedge \)
ipurge-tr-rev I_c id u xs = ipurge-tr-rev I_c id u ys \longrightarrow\)
\( \{x. \, u = x \wedge xs @ [x] \in T_c\} = \{x. \, u = x \wedge ys @ [x] \in T_c\} \)
apply (simp add: T_c-def I_c-def)
apply clarify
apply (cases u; elim disjE; simp; blast)
done
qed

lemma counterexample-not-gu-condition-aux:
\( ([a_c, b_c, c_c], [b_c, a_c, c_c]) \in \text{rel-induct } P_c \, I_c \id a_c \)
proof (simp add: rel-induct-def)
have \( (a_c, [a_c], [b_c, a_c]) \in \text{rel-induct-aux } P_c \, I_c \id \) a_c
proof –
  have \( A: \, a_c \in \text{range id } by \ simp \)
  moreover have \( B: \, (id b_c, a_c) \notin I_c \id \) by (simp add: I_c-def)
  moreover have \( b_c \in \text{next-events } P_c \, [\] \]
    by (simp add: counterexample-next-events-2 T_c-def)
  ultimately have \( (a_c, [\], [\oplus [b_c]]) \in \text{rel-induct-aux } P_c \, I_c \id \) by (rule rule-LR)
  hence \( C: \, (a_c, [], [b_c]) \in \text{rel-induct-aux } P_c \, I_c \id \) by simp
  moreover from \( C \) have \( (id a_c, [], [b_c]) \in \text{rel-induct-aux } P_c \, I_c \id \) by simp
  moreover have \( a_c \in \text{next-events } P_c \, [\] \cap \text{next-events } P_c \, [b_c] \)
    by (simp add: counterexample-next-events-2 T_c-def)
  ultimately have \( (a_c, [\oplus [a_c], [b_c]]) \in \text{rel-induct-aux } P_c \, I_c \id \)
    by (rule rule-WSC)
  hence \( D: \, (a_c, [a_c], [b_c, a_c]) \in \text{rel-induct-aux } P_c \, I_c \id \) by simp
  have \( b_c \in \text{next-events } P_c \, [a_c] \)
    by (simp add: counterexample-next-events-2 T_c-def)
  with \( A \) and \( B \) have \( (a_c, [a_c], [a_c] \oplus [b_c]) \in \text{rel-induct-aux } P_c \, I_c \id \)
    by (rule rule-LR)

23
hence $(a_e, [a_e], [a_e], b_e)) ∈ rel-induct-aux P_e I_e id by simp
hence $(a_e, [a_e], [a_e], [b_e, a_e]) ∈ rel-induct-aux P_e I_e id by (rule rule-sym)
thus ?thesis using D by (rule rule-trans)

qed

moreover have $(id c_e, [a_e, b_e], [b_e, a_e]) ∈ rel-induct-aux P_e I_e id

proof simp
  have $A$: $c_e ∈ range id by simp
  moreover have $B$: $(id a_e, c_e) ⬇ I_e by (simp add: I_e-def)
  moreover have $C$: $a_e ∈ next-events P_e []
    by (simp add: counterexample-next-events-2 T_e-def)
  ultimately have $(c_e, [], [] @ [a_e]) ∈ rel-induct-aux P_e I_e id by (rule rule-LR)
  hence $D$: $(c_e, [], [a_e]) ∈ rel-induct-aux P_e I_e id by simp
  have $b_e ∈ range id by simp
  moreover have $(id a_e, b_e) ⬇ I_e by (simp add: I_e-def)
  ultimately have $(b_e, [], [] @ [a_e]) ∈ rel-induct-aux P_e I_e id
    using $C$ by (rule rule-LR)
  hence $(id b_e, [], [a_e]) ∈ rel-induct-aux P_e I_e id by simp
  moreover have $b_e ∈ next-events P_e [] ∩ next-events P_e [a_e]
    by (simp add: counterexample-next-events-2 T_e-def)
  ultimately have $(c_e, [], [] @ [b_e], [a_e] @ [b_e]) ∈ rel-induct-aux P_e I_e id
    by (rule rule-WSC [OF $D$])
  hence $(c_e, [b_e], [a_e], b_e)) ∈ rel-induct-aux P_e I_e id by simp
  hence $(c_e, [a_e, b_e], [b_e]) ∈ rel-induct-aux P_e I_e id by (rule rule-sym)
  moreover have $a_e ∈ next-events P_e [b_e]
    by (simp add: counterexample-next-events-2 T_e-def)
  with $A$ and $B$ have $(c_e, [b_e], [b_e] @ [a_e]) ∈ rel-induct-aux P_e I_e id
    by (rule rule-LR)
  hence $(c_e, [b_e], [b_e, a_e]) ∈ rel-induct-aux P_e I_e id by simp
  ultimately show $(c_e, [a_e, b_e], [b_e, a_e]) ∈ rel-induct-aux P_e I_e id
    by (rule rule-trans)

qed

moreover have $c_e ∈ next-events P_e [a_e, b_e] ∩ next-events P_e [b_e, a_e]
  by (simp add: counterexample-next-events-2 T_e-def)
  ultimately have $(a_e, [a_e, b_e] @ [c_e], [b_e, a_e] @ [c_e]) ∈ rel-induct-aux P_e I_e id
    by (rule rule-WSC)
  thus $(a_e, [a_e, b_e, c_e], [b_e, a_e, c_e]) ∈ rel-induct-aux P_e I_e id by simp

qed

lemma counterexample-not-qu-condition:
~ $(∃ R. view-partition P_e id R ∧
  weakly-future-consistent P_e I_e id R ∧
  weakly-step-consistent P_e id R ∧
  locally-respects P_e I_e id R)

proof (rule notI, erule exE, (erule conjE)+

fix $R$
assume weakly-future-consistent $P_e I_e id R$

hence $∀ u ∈ range id ∩ (¬I_e) “range id. ∀ xs ys. (xs, ys) ∈ R u → next-dom-events P_e id u xs = next-dom-events P_e id u ys
by (simp add: weakly-future-consistent-def)
moreover have \( a_c \in \text{range id} \cap (\neg I_c) \) \( \subseteq \) \( \text{range id} \)
by (simp add: \( I_c \)-def, rule ImageI \( \text{of } b_c \), simp-all)
ultimately have \( \forall \) \( xs \) \( ys \). \( (xs, ys) \) \( \in \) \( R a_c \) \( \longrightarrow \)
next-dom-events \( P_c \) \( \text{id } a_c \) \( xs \) = next-dom-events \( P_c \) \( \text{id } a_c \) \( ys \) ..
hence \( ([a_c, b_c, c_c], [b_c, a_c, c_c]) \) \( \in \) \( R a_c \) \( \longrightarrow \)
next-dom-events \( P_c \) \( \text{id } a_c \) \( [a_c, b_c, c_c] \) = next-dom-events \( P_c \) \( \text{id } a_c \) \( [b_c, a_c, c_c] \)
by blast
moreover assume
view-partition \( P_c \) \( \text{id } R \) and
weakly-step-consistent \( P_c \) \( \text{id } R \) and
locally-respects \( P_c \) \( I_c \) \( \text{id } R \)
hence rel-induct \( P_c \) \( I_c \) \( \text{id } a_c \) \( \subseteq \) \( R a_c \) by (rule rel-induct-subset)
hence \( ([a_c, b_c, c_c], [b_c, a_c, c_c]) \) \( \in \) \( R a_c \)
using counterexample-not-gu-condition-aux ..
ultimately have
next-dom-events \( P_c \) \( \text{id } a_c \) \( [a_c, b_c, c_c] \) = next-dom-events \( P_c \) \( \text{id } a_c \) \( [b_c, a_c, c_c] \) ..
thus False
by (simp add: next-dom-events-def counterexample-next-events-2 \( T_c \)-def)
qed

\textbf{theorem} not-secure-implies-gu-condition:
\(~ (\text{secure } P_c \ I_c \ \text{id } \longrightarrow \)
\( (\exists R. \ \text{view-partition } P_c \ \text{id } R \land \)
weakly-future-consistent \( P_c \ I_c \ \text{id } R \land \)
weakly-step-consistent \( P_c \ I_c \ \text{id } R \land \)
locally-respects \( P_c \ I_c \ \text{id } R)\))
\textbf{proof} (simp del: not-ex, rule conjI, rule counterexample-secure)
\textbf{qed} (rule counterexample-not-gu-condition)
end

\textbf{References}


