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

March 17, 2025

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.

Contents

1 The Generic Unwinding Theorem

theory *GenericUnwinding*
imports *Noninterference-Ipurge-Unwinding.DeterministicProcesses*
begin

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 consequent 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*) ⇒ (*'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*) ⇒ (*'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 @ [x']
     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 ∀ v ∈ U. (v, D x) ∉ I
  hence 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 @ [x']
     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
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 (subst 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 xs* **..**
assume $\forall\, u \in$ *sinks-aux I D U xs*. $(u, v) \notin I$
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 *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 *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.

lemma *ruc-wfc-failures*:
assumes
RUC: *ref-union-closed P* **and**
WFC: *weakly-future-consistent P I D R* **and**
A: $U \subseteq$ *range D* $\cap$ $(-I)$ `` *range D* **and**
B: $U \neq \{\}$ **and**
C: $\forall\, u \in U.\ (xs,\ xs') \in R\ u$ **and**
D: $(xs,\ X) \in$ *failures P*
shows $(xs',\ X \cap D -` U) \in$ *failures P*
proof (*cases* $\exists\, x.\ x \in X \cap D -` U$)
let *?A* = *singleton-set* $(X \cap D -` U)$
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 (*simp add*: *singleton-set-def*, *rule allI*, *rule impI*, *erule bexE*, *erule IntE*, *simp*)
fix *x*

have $\forall\, u \in \mathit{range}\ D \cap (-I)$ ‘‘ $\mathit{range}\ D.\ \forall\, xs\ ys.\ (xs,\ ys) \in R\ u \longrightarrow$
$\mathit{next\text{-}dom\text{-}events}\ P\ D\ u\ xs = \mathit{next\text{-}dom\text{-}events}\ P\ D\ u\ ys \wedge$
$\mathit{ref\text{-}dom\text{-}events}\ P\ D\ u\ xs = \mathit{ref\text{-}dom\text{-}events}\ P\ D\ u\ ys$
using *WFC* **by** (*simp add*: *weakly-future-consistent-def*)
moreover assume *E*: $D\ x \in U$
with *A* **have** $D\ x \in \mathit{range}\ D \cap (-\ I)$ ‘‘ $\mathit{range}\ D$ **..**
ultimately have $\forall\, xs\ ys.\ (xs,\ ys) \in R\ (D\ x) \longrightarrow$
$\mathit{next\text{-}dom\text{-}events}\ P\ D\ (D\ x)\ xs = \mathit{next\text{-}dom\text{-}events}\ P\ D\ (D\ x)\ ys \wedge$
$\mathit{ref\text{-}dom\text{-}events}\ P\ D\ (D\ x)\ xs = \mathit{ref\text{-}dom\text{-}events}\ P\ D\ (D\ x)\ ys$ **..**
hence $(xs,\ xs') \in R\ (D\ x) \longrightarrow$
$\mathit{next\text{-}dom\text{-}events}\ P\ D\ (D\ x)\ xs = \mathit{next\text{-}dom\text{-}events}\ P\ D\ (D\ x)\ xs' \wedge$
$\mathit{ref\text{-}dom\text{-}events}\ P\ D\ (D\ x)\ xs = \mathit{ref\text{-}dom\text{-}events}\ P\ D\ (D\ x)\ xs'$
by *blast*
moreover have $(xs,\ xs') \in R\ (D\ x)$ **using** *C* **and** *E* **..**
ultimately have $\mathit{ref\text{-}dom\text{-}events}\ P\ D\ (D\ x)\ xs =$
$\mathit{ref\text{-}dom\text{-}events}\ P\ D\ (D\ x)\ xs'$
by *simp*
moreover assume $x \in X$
hence $\{x\} \subseteq X$ **by** *simp*
with *D* **have** $(xs,\ \{x\}) \in \mathit{failures}\ P$ **by** (*rule process-rule-3*)
hence $x \in \mathit{ref\text{-}dom\text{-}events}\ P\ D\ (D\ x)\ xs$
by (*simp add*: *ref-dom-events-def refusals-def*)
ultimately have $x \in \mathit{ref\text{-}dom\text{-}events}\ P\ D\ (D\ x)\ xs'$ **by** *simp*
thus $(xs',\ \{x\}) \in \mathit{failures}\ P$ **by** (*simp add*: *ref-dom-events-def refusals-def*)
qed
ultimately have $(xs',\ \bigcup X \in ?A.\ X) \in \mathit{failures}\ P$ **..**
thus $(xs',\ X \cap D -‘ U) \in \mathit{failures}\ P$ **by** (*simp only*: *singleton-set-union*)
next
have $\exists\, u.\ u \in U$ **using** *B* **by** (*simp add*: *ex-in-conv*)
then obtain *u* **where** *E*: $u \in U$ **..**
with *A* **have** $u \in \mathit{range}\ D \cap (-\ I)$ ‘‘ $\mathit{range}\ D$ **..**
moreover have $(xs,\ xs') \in R\ u$ **using** *C* **and** *E* **..**
ultimately have $(xs \in \mathit{traces}\ P) = (xs' \in \mathit{traces}\ P)$
by (*rule wfc-traces* [*OF WFC*])
moreover have $xs \in \mathit{traces}\ P$ **using** *D* **by** (*rule failures-traces*)
ultimately have $xs' \in \mathit{traces}\ P$ **by** *simp*
hence $(xs',\ \{\}) \in \mathit{failures}\ P$ **by** (*rule traces-failures*)
moreover case *False*
hence $X \cap D -‘ U = \{\}$ **by** (*simp only*: *ex-in-conv*, *simp*)
ultimately show $(xs',\ X \cap D -‘ U) \in \mathit{failures}\ P$ **by** *simp*
qed

lemma *ruc-wfc-lr-failures-1*:
assumes
RUC: *ref-union-closed P* **and**
WFC: *weakly-future-consistent P I D R* **and**
LR: *locally-respects P I D R* **and**
A: $(xs\ @\ [y],\ Y) \in \mathit{failures}\ P$
shows $(xs,\ \{x \in Y.\ (D\ y,\ D\ x) \notin I\}) \in \mathit{failures}\ P$

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* $P\ D\ u\ ys \wedge$
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* $P\ D\ (D\ x)\ ys \wedge$
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 $\forall\, xs\ y.\ (D\ y,\ D\ x) \notin I \wedge y \in$ *next-events* $P\ xs \longrightarrow$
$(xs,\ xs\ @\ [y]) \in R\ (D\ x)$
using *LR* **by** (*simp add*: *locally-respects-def*)
hence $(D\ y,\ D\ x) \notin I \wedge y \in$ *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*: *ipurge-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*)
next

case *False*
hence $\{x \in Y.\ (D\ y,\ D\ x) \notin I\} = \{\}$ **by** *simp*
moreover have $(xs,\ \{\}) \in$ *failures* P **using** A **by** (*rule process-rule-2*)
ultimately show *?thesis* **by** (*simp* (*no-asm-simp*))
qed

lemma *ruc-wfc-lr-failures-2*:
assumes
RUC: *ref-union-closed* P **and**
WFC: *weakly-future-consistent* $P\ I\ D\ R$ **and**
LR: *locally-respects* $P\ I\ D\ R$ **and**
A: $(xs,\ Z) \in$ *failures* P **and**
Y: $xs\ @\ [y] \in$ *traces* P
shows $(xs\ @\ [y],\ \{x \in Z.\ (D\ y,\ D\ x) \notin I\}) \in$ *failures* P
proof (*cases* $\exists\, x.\ x \in \{x \in Z.\ (D\ y,\ D\ x) \notin I\}$)
let *?A* = *singleton-set* $\{x \in Z.\ (D\ y,\ D\ x) \notin I\}$
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*$.\ (xs\ @\ [y],\ X) \in$ *failures* $P) \longrightarrow$
$(xs\ @\ [y],\ \bigcup X \in$ *?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*$.\ (xs\ @\ [y],\ X) \in$ *failures* $P) \longrightarrow$
$(xs\ @\ [y],\ \bigcup X \in$ *?A*$.\ X) \in$ *failures* P **..**
moreover have $\forall\, X \in$ *?A*$.\ (xs\ @\ [y],\ X) \in$ *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* $P\ D\ u\ ys \wedge$
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* $P\ D\ (D\ x)\ ys \wedge$
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 $\forall\, xs\ y.\ (D\ y,\ D\ x) \notin I \wedge y \in$ *next-events* $P\ xs \longrightarrow$
$(xs,\ xs\ @\ [y]) \in R\ (D\ x)$
using LR **by** (*simp add*: *locally-respects-def*)
hence $(D\ y,\ D\ x) \notin I \wedge y \in$ *next-events* $P\ xs \longrightarrow (xs,\ xs\ @\ [y]) \in R\ (D\ x)$
by *blast*
moreover have $y \in$ *next-events* $P\ xs$ **using** Y **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 ∈ Z
   have x ∈ ref-dom-events P D (D x) xs
   proof (simp add: ref-dom-events-def refusals-def)
    have {x} ⊆ Z using D by (simp add: ipurge-ref-def)
    with A show (xs, {x}) ∈ failures P by (rule process-rule-3)
   qed
   ultimately have x ∈ ref-dom-events P D (D x) (xs @ [y]) by simp
   thus (xs @ [y], {x}) ∈ failures P
    by (simp add: ref-dom-events-def refusals-def)
  qed
  ultimately have (xs @ [y], ⋃X ∈ ?A. X) ∈ failures P ..
  thus ?thesis by (simp only: singleton-set-union)
next
  case False
  hence {x ∈ Z. (D y, D x) ∉ I} = {} by simp
  moreover have (xs @ [y], {}) ∈ failures P using Y by (rule traces-failures)
  ultimately show ?thesis by (simp (no-asm-simp))
qed

lemma gu-condition-imply-secure-aux [rule-format]:
  assumes
    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 U ⊆ range D ⟶ U ≠ {} ⟶ xs @ ys ∈ traces P ⟶
    (∀ u ∈ unaffected-domains I D U []. (xs, xs') ∈ R u) ⟶
    (∀ u ∈ unaffected-domains I D U ys.
      (xs @ ys, xs' @ ipurge-tr-aux I D U ys) ∈ 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: ⋀xs xs' U. U ⊆ range D ⟶ U ≠ {} ⟶ xs @ ys ∈ traces P ⟶
      (∀ u. u ∈ range D ∧ (∀ v ∈ U. (v, u) ∉ I) ⟶
        (xs, xs') ∈ R u) ⟶
      (∀ u. u ∈ range D ∧ (∀ v ∈ sinks-aux I D U ys. (v, u) ∉ I) ⟶
        (xs @ ys, xs' @ ipurge-tr-aux I D U ys) ∈ R u) and
    B: U ⊆ range D and
    C: U ≠ {} and
    D: xs @ y # ys ∈ traces P and
    E: ∀ u. u ∈ range D ∧ (∀ v ∈ U. (v, u) ∉ I) ⟶ (xs, xs') ∈ R u and
    F: u ∈ range D and
    G: ∀ v ∈ sinks-aux I D U (y # ys). (v, u) ∉ I
  show (xs @ y # ys, xs' @ ipurge-tr-aux I D U (y # ys)) ∈ R u
  proof (cases ∃ v ∈ U. (v, D y) ∈ I,
   simp-all (no-asm-simp) add: ipurge-tr-aux-cons)
    case True
```

let *?U′ = insert (D y) U*
have *?U′ ⊆ range D ⟶ ?U′ ≠ {} ⟶ (xs @ [y]) @ ys ∈ traces P ⟶*
(∀ u. u ∈ range D ∧ (∀ v ∈ ?U′. (v, u) ∉ I) ⟶
(xs @ [y], xs′) ∈ R u) ⟶
(∀ u. u ∈ range D ∧ (∀ v ∈ sinks-aux I D ?U′ ys. (v, u) ∉ I) ⟶
((xs @ [y]) @ ys, xs′ @ ipurge-tr-aux I D ?U′ ys) ∈ R u)
using *A* **.**
hence
(∀ u. u ∈ range D ∧ (∀ v ∈ ?U′. (v, u) ∉ I) ⟶
(xs @ [y], xs′) ∈ R u) ⟶
(∀ u. u ∈ range D ∧ (∀ v ∈ sinks-aux I D ?U′ ys. (v, u) ∉ I) ⟶
(xs @ y # ys, xs′ @ ipurge-tr-aux I D ?U′ ys) ∈ R u)
using *B* **and** *D* **by** *simp*
moreover have
∀ u. u ∈ range D ∧ (∀ v ∈ ?U′. (v, u) ∉ I) ⟶
(xs @ [y], xs′) ∈ R u
proof (*rule allI*, *rule impI*, *erule conjE*)
fix *u*
assume *F*: *u ∈ range D* **and** *G*: *∀ v ∈ ?U′. (v, u) ∉ I*
moreover have *u ∈ range D ∧ (∀ v ∈ U. (v, u) ∉ I) ⟶ (xs, xs′) ∈ R u*
using *E* **..**
ultimately have *H*: *(xs, xs′) ∈ R u* **by** *simp*
have *∀ u ∈ range D. (D y, u) ∉ I ∧ y ∈ next-events P xs ⟶*
(xs, xs @ [y]) ∈ R u
using *LR* **by** (*simp add*: *locally-respects-def*)
hence *(D y, u) ∉ I ∧ y ∈ next-events P xs ⟶ (xs, xs @ [y]) ∈ R u*
using *F* **..**
moreover have *D y ∈ ?U′* **by** *simp*
with *G* **have** *(D y, u) ∉ I* **..**
moreover have *(xs @ [y]) @ ys ∈ traces P* **using** *D* **by** *simp*
hence *y ∈ next-events P xs*
by (*simp* (*no-asm-simp*) *add*: *next-events-def*, *rule process-rule-2-traces*)
ultimately have *I*: *(xs, xs @ [y]) ∈ R u* **by** *simp*
have *∀ u ∈ range D. equiv (traces P) (R u)*
using *VP* **by** (*simp add*: *view-partition-def*)
hence *J*: *equiv (traces P) (R u)* **using** *F* **..**
hence *trans (R u)* **by** (*simp add*: *equiv-def*)
moreover have *sym (R u)* **using** *J* **by** (*simp add*: *equiv-def*)
hence *(xs @ [y], xs) ∈ R u* **using** *I* **by** (*rule symE*)
ultimately show *(xs @ [y], xs′) ∈ R u* **using** *H* **by** (*rule transE*)
qed
ultimately have
∀ u. u ∈ range D ∧ (∀ v ∈ sinks-aux I D ?U′ ys. (v, u) ∉ I) ⟶
(xs @ y # ys, xs′ @ 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′ @ ipurge-tr-aux I D ?U′ ys) ∈ R u **..**
moreover have *sinks-aux I D U (y # ys) = sinks-aux I D ?U′ ys*
using *Cons* **and** *True* **by** (*simp add*: *sinks-aux-cons*)

hence $\forall\, v \in \mathit{sinks\text{-}aux}\ I\ D\ ?U'\ \mathit{ys}.\ (v,\ u) \notin I$ **using** G **by** *simp*
with F **have** $u \in \mathit{range}\ D \wedge (\forall\, v \in \mathit{sinks\text{-}aux}\ I\ D\ ?U'\ \mathit{ys}.\ (v,\ u) \notin I)$ **..**
ultimately show $(\mathit{xs} \mathbin{@} y \# \mathit{ys},\ \mathit{xs}' \mathbin{@} \mathit{ipurge\text{-}tr\text{-}aux}\ I\ D\ ?U'\ \mathit{ys}) \in R\ u$ **..**
next
case *False*
have
$U \subseteq \mathit{range}\ D \longrightarrow U \neq \{\} \longrightarrow (\mathit{xs} \mathbin{@} [y]) \mathbin{@} \mathit{ys} \in \mathit{traces}\ P \longrightarrow$
$(\forall\, u.\ u \in \mathit{range}\ D \wedge (\forall\, v \in U.\ (v,\ u) \notin I) \longrightarrow$
$(\mathit{xs} \mathbin{@} [y],\ \mathit{xs}' \mathbin{@} [y]) \in R\ u) \longrightarrow$
$(\forall\, u.\ u \in \mathit{range}\ D \wedge (\forall\, v \in \mathit{sinks\text{-}aux}\ I\ D\ U\ \mathit{ys}.\ (v,\ u) \notin I) \longrightarrow$
$((\mathit{xs} \mathbin{@} [y]) \mathbin{@} \mathit{ys},\ (\mathit{xs}' \mathbin{@} [y]) \mathbin{@} \mathit{ipurge\text{-}tr\text{-}aux}\ I\ D\ U\ \mathit{ys}) \in R\ u)$
using A **.**
hence
$(\forall\, u.\ u \in \mathit{range}\ D \wedge (\forall\, v \in U.\ (v,\ u) \notin I) \longrightarrow$
$(\mathit{xs} \mathbin{@} [y],\ \mathit{xs}' \mathbin{@} [y]) \in R\ u) \longrightarrow$
$(\forall\, u.\ u \in \mathit{range}\ D \wedge (\forall\, v \in \mathit{sinks\text{-}aux}\ I\ D\ U\ \mathit{ys}.\ (v,\ u) \notin I) \longrightarrow$
$(\mathit{xs} \mathbin{@} y \# \mathit{ys},\ \mathit{xs}' \mathbin{@} y \# \mathit{ipurge\text{-}tr\text{-}aux}\ I\ D\ U\ \mathit{ys}) \in R\ u)$
using B **and** C **and** D **by** *simp*
moreover have
$\forall\, u.\ u \in \mathit{range}\ D \wedge (\forall\, v \in U.\ (v,\ u) \notin I) \longrightarrow$
$(\mathit{xs} \mathbin{@} [y],\ \mathit{xs}' \mathbin{@} [y]) \in R\ u$
proof (*rule allI*, *rule impI*, *erule conjE*)
fix u
assume F: $u \in \mathit{range}\ D$ **and** G: $\forall\, v \in U.\ (v,\ u) \notin I$
moreover have $u \in \mathit{range}\ D \wedge (\forall\, v \in U.\ (v,\ u) \notin I) \longrightarrow (\mathit{xs},\ \mathit{xs}') \in R\ u$
using E **..**
ultimately have $(\mathit{xs},\ \mathit{xs}') \in R\ u$ **by** *simp*
moreover have $D\ y \in \mathit{range}\ D \wedge$
$(\forall\, v \in U.\ (v,\ D\ y) \notin I) \longrightarrow (\mathit{xs},\ \mathit{xs}') \in R\ (D\ y)$
using E **..**
hence $(\mathit{xs},\ \mathit{xs}') \in R\ (D\ y)$ **using** *False* **by** *simp*
ultimately have H: $(\mathit{xs},\ \mathit{xs}') \in R\ u \cap R\ (D\ y)$ **..**
have $\exists\, v.\ v \in U$ **using** C **by** (*simp add*: *ex-in-conv*)
then obtain v **where** I: $v \in U$ **..**
hence $(v,\ D\ y) \in -I$ **using** *False* **by** *simp*
moreover have $v \in \mathit{range}\ D$ **using** B **and** I **..**
ultimately have $D\ y \in (-I)\ ``\ \mathit{range}\ D$ **..**
hence J: $D\ y \in \mathit{range}\ D \cap (-I)\ ``\ \mathit{range}\ D$ **by** *simp*
have $\forall\, u \in \mathit{range}\ D \cap (-I)\ ``\ \mathit{range}\ D.\ \forall\, \mathit{xs}\ \mathit{ys}.\ (\mathit{xs},\ \mathit{ys}) \in R\ u \longrightarrow$
$\mathit{next\text{-}dom\text{-}events}\ P\ D\ u\ \mathit{xs} = \mathit{next\text{-}dom\text{-}events}\ P\ D\ u\ \mathit{ys} \wedge$
$\mathit{ref\text{-}dom\text{-}events}\ P\ D\ u\ \mathit{xs} = \mathit{ref\text{-}dom\text{-}events}\ P\ D\ u\ \mathit{ys}$
using *WFC* **by** (*simp add*: *weakly-future-consistent-def*)
hence $\forall\, \mathit{xs}\ \mathit{ys}.\ (\mathit{xs},\ \mathit{ys}) \in R\ (D\ y) \longrightarrow$
$\mathit{next\text{-}dom\text{-}events}\ P\ D\ (D\ y)\ \mathit{xs} = \mathit{next\text{-}dom\text{-}events}\ P\ D\ (D\ y)\ \mathit{ys} \wedge$
$\mathit{ref\text{-}dom\text{-}events}\ P\ D\ (D\ y)\ \mathit{xs} = \mathit{ref\text{-}dom\text{-}events}\ P\ D\ (D\ y)\ \mathit{ys}$
using J **..**
hence $(\mathit{xs},\ \mathit{xs}') \in R\ (D\ y) \longrightarrow$
$\mathit{next\text{-}dom\text{-}events}\ P\ D\ (D\ y)\ \mathit{xs} = \mathit{next\text{-}dom\text{-}events}\ P\ D\ (D\ y)\ \mathit{xs}' \wedge$
$\mathit{ref\text{-}dom\text{-}events}\ P\ D\ (D\ y)\ \mathit{xs} = \mathit{ref\text{-}dom\text{-}events}\ P\ D\ (D\ y)\ \mathit{xs}'$

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 **..**
moreover 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*

assume $(xs \mathbin{@} [y],\ Y) \in failures\ P$
with *RUC* **and** *WFC* **and** *LR* **show**
$(xs, \{x \in Y.\ (D\ y, D\ x) \notin I\}) \in failures\ P$
by (*rule ruc-wfc-lr-failures-1*)
next
fix $y'\ ys\ Y$
assume
A: $\bigwedge Y'.\ (xs \mathbin{@} y \# ys,\ Y') \in failures\ P \longrightarrow$
$(xs \mathbin{@} ipurge\text{-}tr\ I\ D\ (D\ y)\ ys, ipurge\text{-}ref\ I\ D\ (D\ y)\ ys\ Y') \in failures\ P$ **and**
B: $(xs \mathbin{@} y \# ys \mathbin{@} [y'],\ Y) \in failures\ P$
show $(xs \mathbin{@} ipurge\text{-}tr\ I\ D\ (D\ y)\ (ys \mathbin{@} [y']), ipurge\text{-}ref\ I\ D\ (D\ y)\ (ys \mathbin{@} [y'])\ Y)$
$\in failures\ P$
proof (*cases* $D\ y' \in sinks\ I\ D\ (D\ y)\ (ys \mathbin{@} [y'])$, *simp del*: *sinks.simps*)
let $?Y' = \{x \in Y.\ (D\ y', D\ x) \notin I\}$
have $(xs \mathbin{@} y \# ys, ?Y') \in failures\ P \longrightarrow$
$(xs \mathbin{@} ipurge\text{-}tr\ I\ D\ (D\ y)\ ys, ipurge\text{-}ref\ I\ D\ (D\ y)\ ys\ ?Y') \in failures\ P$
using *A* **.**
moreover have $((xs \mathbin{@} y \# ys) \mathbin{@} [y'],\ Y) \in failures\ P$ **using** *B* **by** *simp*
with *RUC* **and** *WFC* **and** *LR* **have** $(xs \mathbin{@} y \# ys, ?Y') \in failures\ P$
by (*rule ruc-wfc-lr-failures-1*)
ultimately have
$(xs \mathbin{@} ipurge\text{-}tr\ I\ D\ (D\ y)\ ys, ipurge\text{-}ref\ I\ D\ (D\ y)\ ys\ ?Y') \in failures\ P$ **..**
moreover case *True*
hence $ipurge\text{-}ref\ I\ D\ (D\ y)\ (ys \mathbin{@} [y'])\ Y = ipurge\text{-}ref\ I\ D\ (D\ y)\ ys\ ?Y'$
by (*rule ipurge-ref-eq*)
ultimately show
$(xs \mathbin{@} ipurge\text{-}tr\ I\ D\ (D\ y)\ ys, ipurge\text{-}ref\ I\ D\ (D\ y)\ (ys \mathbin{@} [y'])\ Y) \in failures\ P$
by *simp*
next
case *False*
have $unaffected\text{-}domains\ I\ D\ \{D\ y\}\ (ys \mathbin{@} [y']) \subseteq range\ D \cap (-I)\ ``\ range\ D$
(**is** $?U \subseteq$ -)
by (*rule unaffected-domains-subset*, *simp-all*)
moreover have $?U \neq \{\}$
proof −
have $(D\ y, D\ y') \notin I$ **using** *False* **by** (*rule-tac notI*, *simp*)
moreover
have $\neg\ ((D\ y, D\ y') \in I \vee (\exists v \in sinks\ I\ D\ (D\ y)\ ys.\ (v, D\ y') \in I))$
using *False* **by** (*simp only*: *sinks-interference-eq*, *simp*)
then have $\forall v \in sinks\ I\ D\ (D\ y)\ (ys \mathbin{@} [y']).\ (v, D\ y') \notin I$ **by** *simp*
ultimately show $?U \neq \{\}$
apply (*simp* (*no-asm-simp*) *add*: *unaffected-domains-def sinks-aux-single-dom set-eq-iff*)
using ‹$(D\ y, D\ y') \notin I$› **by** *auto*
qed
moreover have *C*: $xs \mathbin{@} y \# ys \mathbin{@} [y'] \in traces\ P$
using *B* **by** (*rule failures-traces*)
have $\forall u \in ?U.\ ((xs \mathbin{@} [y]) \mathbin{@} ys \mathbin{@} [y'],$
$xs \mathbin{@} ipurge\text{-}tr\text{-}aux\ I\ D\ \{D\ y\}\ (ys \mathbin{@} [y'])) \in R\ u$

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 $\forall\, u \in$ *range D*. $\forall$ *xs x*.
($D\ x,\ u) \notin I \wedge x \in$ *next-events P xs* $\longrightarrow$ ($xs,\ xs$ @ $[x]) \in R\ u$
using *LR* **by** (*simp add*: *locally-respects-def*)
moreover assume *D*: $u \in$ *range D*
ultimately have $\forall$ *xs x*.
($D\ x,\ u) \notin I \wedge x \in$ *next-events P xs* $\longrightarrow$ ($xs,\ xs$ @ $[x]) \in R\ u$ **..**
hence ($D\ y,\ u) \notin I \wedge y \in$ *next-events P xs* $\longrightarrow$
($xs,\ xs$ @ $[y]) \in R\ u$
by *blast*
moreover assume ($D\ y,\ u) \notin I$
moreover have (xs @ $[y]$) @ *ys* @ $[y'] \in$ *traces P* **using** *C* **by** *simp*
hence *xs* @ $[y] \in$ *traces P* **by** (*rule process-rule-2-traces*)
hence $y \in$ *next-events P xs* **by** (*simp add*: *next-events-def*)
ultimately have *E*: ($xs,\ xs$ @ $[y]) \in R\ u$ **by** *simp*
have $\forall\, u \in$ *range D*. *equiv* (*traces P*) (*R u*)
using *VP* **by** (*simp add*: *view-partition-def*)
hence *equiv* (*traces P*) (*R u*) **using** *D* **..**
hence *sym* (*R u*) **by** (*simp add*: *equiv-def*)
thus (xs @ $[y],\ xs) \in R\ u$ **using** *E* **by** (*rule symE*)
qed
hence $\forall\, u \in$ *?U*. (xs @ y # *ys* @ $[y']$,
xs @ *ipurge-tr I D* (*D y*) (*ys* @ $[y']$)) $\in R\ u$
by (*simp only*: *ipurge-tr-aux-single-dom*, *simp*)
ultimately have (xs @ *ipurge-tr I D* (*D y*) (*ys* @ $[y']$), $Y \cap D -$ ‘ *?U*)
$\in$ *failures P*
using *B* **by** (*rule ruc-wfc-failures* [*OF RUC WFC*])
moreover have
$Y \cap D -$ ‘ *?U* = $\{x \in Y.\ D\ x \in$ *unaffected-domains I D* $\{D\ y\}$ (*ys* @ $[y'])\}$
by (*simp add*: *set-eq-iff*)
ultimately show *?thesis* **by** (*simp only*: *unaffected-domains-single-dom*)
qed
qed

lemma *gu-condition-imply-secure-2* [*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* **and**
Y: xs @ $[y] \in$ *traces P*
shows (xs @ $zs,\ Z) \in$ *failures P* $\longrightarrow$
(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*,
simp add: *ipurge-ref-def*)
fix *Z*

```
  assume (xs, Z) ∈ failures P
  with RUC and WFC and LR show
   (xs @ [y], {x ∈ Z. (D y, D x) ∉ I}) ∈ failures P
   using Y by (rule ruc-wfc-lr-failures-2)
next
  fix z zs Z
  assume
    A: ⋀Z'. (xs @ zs, Z') ∈ failures P ⟶
      (xs @ y # ipurge-tr I D (D y) zs,
      ipurge-ref I D (D y) zs Z') ∈ failures P and
    B: (xs @ zs @ [z], Z) ∈ failures P
  show (xs @ y # ipurge-tr I D (D y) (zs @ [z]),
    ipurge-ref I D (D y) (zs @ [z]) Z) ∈ failures P
  proof (cases D z ∈ sinks I D (D y) (zs @ [z]), simp del: sinks.simps)
    let ?Z' = {x ∈ Z. (D z, D x) ∉ I}
    have (xs @ zs, ?Z') ∈ failures P ⟶
      (xs @ y # ipurge-tr I D (D y) zs, ipurge-ref I D (D y) zs ?Z') ∈ failures P
     using A .
    moreover have ((xs @ zs) @ [z], Z) ∈ failures P using B by simp
    with RUC and WFC and LR have (xs @ zs, ?Z') ∈ failures P
     by (rule ruc-wfc-lr-failures-1)
    ultimately have
     (xs @ y # ipurge-tr I D (D y) zs, ipurge-ref I D (D y) zs ?Z') ∈ failures P ..
    moreover case True
    hence ipurge-ref I D (D y) (zs @ [z]) Z = ipurge-ref I D (D y) zs ?Z'
     by (rule ipurge-ref-eq)
    ultimately show
     (xs @ y # ipurge-tr I D (D y) zs, ipurge-ref I D (D y) (zs @ [z]) Z)
       ∈ failures P
     by simp
 next
   case False
   have unaffected-domains I D {D y} (zs @ [z]) ⊆ range D ∩ (−I) `` range D
     (is ?U ⊆ -)
     by (rule unaffected-domains-subset, simp-all)
   moreover have ?U ≠ {}
   proof −
     have (D y, D z) ∉ I using False by (rule-tac notI, simp)
     moreover
     have ¬ ((D y, D z) ∈ I ∨ (∃ v ∈ sinks I D (D y) zs. (v, D z) ∈ I))
       using False by (simp only: sinks-interference-eq, simp)
     then have ∀ v ∈ sinks I D (D y) (zs @ [z]). (v, D z) ∉ I by simp
     ultimately show ?U ≠ {}
      apply (simp (no-asm-simp) add: unaffected-domains-def sinks-aux-single-dom
set-eq-iff)
       using ‹(D y, D z) ∉ I› by auto
   qed
   moreover have C: xs @ zs @ [z] ∈ traces P using B by (rule failures-traces)
   have ∀ u ∈ ?U. (xs @ zs @ [z],
```

```
    (xs @ [y]) @ ipurge-tr-aux I D {D y} (zs @ [z])) ∈ 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 ∀ u ∈ range D. ∀ xs x.
      (D x, u) ∉ I ∧ x ∈ next-events P xs ⟶ (xs, xs @ [x]) ∈ R u
      using LR by (simp add: locally-respects-def)
    moreover assume D: u ∈ range D
    ultimately have ∀ xs x.
      (D x, u) ∉ I ∧ x ∈ next-events P xs ⟶ (xs, xs @ [x]) ∈ R u ..
    hence (D y, u) ∉ I ∧ y ∈ next-events P xs ⟶ (xs, xs @ [y]) ∈ R u by blast
    moreover assume (D y, u) ∉ I
    moreover have y ∈ next-events P xs using Y by (simp add: next-events-def)
    ultimately show (xs, xs @ [y]) ∈ R u by simp
  qed
  hence ∀ u ∈ ?U. (xs @ zs @ [z],
    xs @ y # ipurge-tr I D (D y) (zs @ [z])) ∈ R u
    by (simp only: ipurge-tr-aux-single-dom, simp)
  ultimately have (xs @ y # ipurge-tr I D (D y) (zs @ [z]), Z ∩ D -` ?U)
    ∈ failures P
    using B by (rule ruc-wfc-failures [OF RUC WFC])
  moreover have
    Z ∩ D -` ?U = {x ∈ Z. D x ∈ unaffected-domains I D {D y} (zs @ [z])}
    by (simp add: set-eq-iff)
  ultimately show ?thesis by (simp only: unaffected-domains-single-dom)
 qed
qed

theorem generic-unwinding:
  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 secure P I D
proof (simp add: secure-def futures-def, (rule allI)+, rule impI, erule conjE)
  fix xs y ys Y zs Z
  assume
    A: (xs @ y # ys, Y) ∈ failures P and
    B: (xs @ zs, Z) ∈ failures P
  show (xs @ ipurge-tr I D (D y) ys, ipurge-ref I D (D y) ys Y) ∈ failures P ∧
    (xs @ y # ipurge-tr I D (D y) zs, ipurge-ref I D (D y) zs Z) ∈ failures P
    (is ?P ∧ ?Q)
  proof
    show ?P using RUC and VP and WFC and WSC and LR and A
     by (rule gu-condition-imply-secure-1)
  next
    have ((xs @ [y]) @ ys, Y) ∈ failures P using A by simp
```

hence (*xs* @ [*y*], {}) ∈ *failures P* **by** (*rule process-rule-2-failures*)
hence *xs* @ [*y*] ∈ *traces P* **by** (*rule failures-traces*)
with *RUC* **and** *VP* **and** *WFC* **and** *WSC* **and** *LR* **show** *?Q* **using** *B*
by (*rule gu-condition-imply-secure-2*)
qed
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 *refl-on* (*traces P*) may only be verified by a relation included in *traces P* × *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*:
deterministic P ⟹
view-partition P D R ⟹
d-weakly-future-consistent P I D R ⟹
weakly-step-consistent P D R ⟹
locally-respects P I D R ⟹
secure P I D
proof (*rule generic-unwinding*, *rule d-implies-ruc*, *simp-all*)
qed (*drule d-wfc-equals-dwfc* [*of P I D R*], *simp*)

theorem *ts-generic-unwinding*:
trace-set T ⟹
view-partition (*ts-process T*) *D R* ⟹
d-weakly-future-consistent (*ts-process T*) *I D R* ⟹
weakly-step-consistent (*ts-process T*) *D R* ⟹
locally-respects (*ts-process T*) *I D R* ⟹
secure (*ts-process T*) *I D*
proof (*rule d-generic-unwinding*, *simp-all*)
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.

inductive-set *rel-induct-aux* ::
$'a$ *process* $\Rightarrow$ $('d \times 'd)$ *set* $\Rightarrow$ $('a \Rightarrow 'd) \Rightarrow ('d \times 'a$ *list* $\times 'a$ *list*$)$ *set*
for $P :: 'a$ *process* **and** $I :: ('d \times 'd)$ *set* **and** $D :: 'a \Rightarrow 'd$ **where**
rule-sym: $(u, xs, ys) \in$ *rel-induct-aux* $P\ I\ D \Longrightarrow$
$(u, ys, xs) \in$ *rel-induct-aux* $P\ I\ D$ |
rule-trans: $[\![(u, xs, ys) \in$ *rel-induct-aux* $P\ I\ D;$
$(u, ys, zs) \in$ *rel-induct-aux* $P\ I\ D]\!] \Longrightarrow$
$(u, xs, zs) \in$ *rel-induct-aux* $P\ I\ D$ |
rule-WSC: $[\![(u, xs, ys) \in$ *rel-induct-aux* $P\ I\ D;$
$(D\ x, xs, ys) \in$ *rel-induct-aux* $P\ I\ D;$
$x \in$ *next-events* $P\ xs \cap$ *next-events* $P\ ys]\!] \Longrightarrow$
$(u, xs\ @\ [x], ys\ @\ [x]) \in$ *rel-induct-aux* $P\ I\ D$ |
rule-LR: $[\![u \in$ *range* $D;\ (D\ x, u) \notin I;\ x \in$ *next-events* $P\ xs]\!] \Longrightarrow$
$(u, xs, xs\ @\ [x]) \in$ *rel-induct-aux* $P\ I\ D$

definition *rel-induct* ::
$'a$ *process* $\Rightarrow$ $('d \times 'd)$ *set* $\Rightarrow$ $('a \Rightarrow 'd) \Rightarrow ('a, 'd)$ *dom-rel-map* **where**
rel-induct $P\ I\ D\ u \equiv$ *rel-induct-aux* $P\ I\ D$ `` $\{u\}$

lemma *rel-induct-subset*:
assumes
VP: *view-partition* $P\ D\ R$ **and**
WSC: *weakly-step-consistent* $P\ D\ R$ **and**
LR: *locally-respects* $P\ I\ D\ R$
shows *rel-induct* $P\ I\ D\ u \subseteq R\ u$
proof (*rule subsetI*, *simp add*: *rel-induct-def split-paired-all*,

erule rel-induct-aux.induct)
fix *u xs ys*
have $\forall\, u \in$ *range D*. *equiv* (*traces P*) (*R u*)
using *VP* **by** (*simp add*: *view-partition-def*)
moreover assume (*u*, *xs*, *ys*) $\in$ *rel-induct-aux P I D*
hence *u* $\in$ *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 zs*
have $\forall\, u \in$ *range D*. *equiv* (*traces P*) (*R u*)
using *VP* **by** (*simp add*: *view-partition-def*)
moreover assume (*u*, *xs*, *ys*) $\in$ *rel-induct-aux P I D*
hence *u* $\in$ *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 $\forall\, u \in$ *range D*. $\forall$ *xs ys x*.
(*xs*, *ys*) $\in$ *R u* $\cap$ *R* (*D x*) $\wedge$ *x* $\in$ *next-events P xs* $\cap$ *next-events P ys* $\longrightarrow$
(*xs* @ [*x*], *ys* @ [*x*]) $\in$ *R u*
using *WSC* **by** (*simp add*: *weakly-step-consistent-def*)
moreover assume (*u*, *xs*, *ys*) $\in$ *rel-induct-aux P I D*
hence *u* $\in$ *range D* **by** (*rule rel-induct-aux.induct*)
ultimately have $\forall$ *xs ys x*.
(*xs*, *ys*) $\in$ *R u* $\cap$ *R* (*D x*) $\wedge$ *x* $\in$ *next-events P xs* $\cap$ *next-events P ys* $\longrightarrow$
(*xs* @ [*x*], *ys* @ [*x*]) $\in$ *R u* **..**
hence (*xs*, *ys*) $\in$ *R u* $\cap$ *R* (*D x*) $\wedge$ *x* $\in$ *next-events P xs* $\cap$ *next-events P ys* $\longrightarrow$
(*xs* @ [*x*], *ys* @ [*x*]) $\in$ *R u*
by *blast*
moreover assume
(*xs*, *ys*) $\in$ *R u* **and**
(*xs*, *ys*) $\in$ *R* (*D x*) **and**
x $\in$ *next-events P xs* $\cap$ *next-events P ys*
ultimately show (*xs* @ [*x*], *ys* @ [*x*]) $\in$ *R u* **by** *simp*
next
fix *u xs x*
have $\forall\, u \in$ *range D*. $\forall$ *xs x*.
(*D x*, *u*) $\notin$ *I* $\wedge$ *x* $\in$ *next-events P xs* $\longrightarrow$ (*xs*, *xs* @ [*x*]) $\in$ *R u*
using *LR* **by** (*simp add*: *locally-respects-def*)
moreover assume *u* $\in$ *range D*
ultimately have $\forall$ *xs x*.
(*D x*, *u*) $\notin$ *I* $\wedge$ *x* $\in$ *next-events P xs* $\longrightarrow$ (*xs*, *xs* @ [*x*]) $\in$ *R u* **..**
hence (*D x*, *u*) $\notin$ *I* $\wedge$ *x* $\in$ *next-events P xs* $\longrightarrow$ (*xs*, *xs* @ [*x*]) $\in$ *R u* **by** *blast*
moreover assume (*D x*, *u*) $\notin$ *I* **and** *x* $\in$ *next-events P 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 $event_c = a_c \mid b_c \mid c_c$

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**
$P_c \equiv ts\text{-}process\ T_c$

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

lemma *counterexample-next-events-1*:
$(x \in next\text{-}events\ (ts\text{-}process\ T_c)\ xs) = (xs\ @\ [x] \in T_c)$

by (*rule ts-process-next-events*, *rule counterexample-trace-set*)

lemma *counterexample-next-events-2*:
$(x \in \textit{next-events}\ P_c\ xs) = (xs\ @\ [x] \in T_c)$
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*) ∈ *rel-ipurge* (*ts-process* T_c) I_c *id u* ⟶
(*xs* ∈ *traces* (*ts-process* T_c)) = (*ys* ∈ *traces* (*ts-process* T_c)) ∧
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* ∈ T_c ∧ *ys* ∈ T_c ∧
ipurge-tr-rev I_c *id u xs* = *ipurge-tr-rev* I_c *id u ys* ⟶
{*x*. *u* = *x* ∧ *xs* @ [*x*] ∈ T_c} = {*x*. *u* = *x* ∧ *ys* @ [*x*] ∈ T_c}

apply (*simp add*: T_c*-def* I_c*-def*)
apply *clarify*
apply (*cases u*; *elim disjE*; *simp*; *blast*)
done
qed
qed

lemma *counterexample-not-gu-condition-aux*:
([a_c, b_c, c_c], [b_c, a_c, c_c]) ∈ *rel-induct* $P_c\ I_c\ id\ a_c$
proof (*simp add*: *rel-induct-def*)
have (a_c, [a_c, b_c], [b_c, a_c]) ∈ *rel-induct-aux* $P_c\ I_c\ id$
proof −
have *A*: a_c ∈ *range id* **by** *simp*
moreover have *B*: (*id* b_c, a_c) ∉ I_c **by** (*simp add*: I_c*-def*)
moreover have b_c ∈ *next-events* P_c []
by (*simp add*: *counterexample-next-events-2* T_c*-def*)
ultimately have (a_c, [], [] @ [b_c]) ∈ *rel-induct-aux* $P_c\ I_c\ id$ **by** (*rule rule-LR*)
hence *C*: (a_c, [], [b_c]) ∈ *rel-induct-aux* $P_c\ I_c\ id$ **by** *simp*
moreover from *C* **have** (*id* a_c, [], [b_c]) ∈ *rel-induct-aux* $P_c\ I_c\ id$ **by** *simp*
moreover have a_c ∈ *next-events* P_c [] ∩ *next-events* P_c [b_c]
by (*simp add*: *counterexample-next-events-2* T_c*-def*)
ultimately have (a_c, [] @ [a_c], [b_c] @ [a_c]) ∈ *rel-induct-aux* $P_c\ I_c\ id$
by (*rule rule-WSC*)
hence *D*: (a_c, [a_c], [b_c, a_c]) ∈ *rel-induct-aux* $P_c\ I_c\ id$ **by** *simp*
have b_c ∈ *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] @ [b_c]) ∈ *rel-induct-aux* $P_c\ I_c\ id$
by (*rule rule-LR*)

hence $(a_c, [a_c], [a_c, b_c]) \in$ *rel-induct-aux* P_c I_c *id* **by** *simp*
hence $(a_c, [a_c, b_c], [a_c]) \in$ *rel-induct-aux* P_c I_c *id* **by** (*rule rule-sym*)
thus *?thesis* **using** *D* **by** (*rule rule-trans*)
qed
moreover have $(id\ c_c, [a_c, b_c], [b_c, a_c]) \in$ *rel-induct-aux* P_c I_c *id*
proof *simp*
have *A*: $c_c \in$ *range id* **by** *simp*
moreover have *B*: $(id\ a_c, c_c) \notin I_c$ **by** (*simp add*: I_c*-def*)
moreover have *C*: $a_c \in$ *next-events* P_c []
by (*simp add*: *counterexample-next-events-2* T_c*-def*)
ultimately have $(c_c, [], [] @ [a_c]) \in$ *rel-induct-aux* P_c I_c *id* **by** (*rule rule-LR*)
hence *D*: $(c_c, [], [a_c]) \in$ *rel-induct-aux* P_c I_c *id* **by** *simp*
have $b_c \in$ *range id* **by** *simp*
moreover have $(id\ a_c, b_c) \notin I_c$ **by** (*simp add*: I_c*-def*)
ultimately have $(b_c, [], [] @ [a_c]) \in$ *rel-induct-aux* P_c I_c *id*
using *C* **by** (*rule rule-LR*)
hence $(id\ b_c, [], [a_c]) \in$ *rel-induct-aux* P_c I_c *id* **by** *simp*
moreover have $b_c \in$ *next-events* P_c [] $\cap$ *next-events* P_c $[a_c]$
by (*simp add*: *counterexample-next-events-2* T_c*-def*)
ultimately have $(c_c, [] @ [b_c], [a_c] @ [b_c]) \in$ *rel-induct-aux* P_c I_c *id*
by (*rule rule-WSC* [*OF D*])
hence $(c_c, [b_c], [a_c, b_c]) \in$ *rel-induct-aux* P_c I_c *id* **by** *simp*
hence $(c_c, [a_c, b_c], [b_c]) \in$ *rel-induct-aux* P_c I_c *id* **by** (*rule rule-sym*)
moreover have $a_c \in$ *next-events* P_c $[b_c]$
by (*simp add*: *counterexample-next-events-2* T_c*-def*)
with *A* **and** *B* **have** $(c_c, [b_c], [b_c] @ [a_c]) \in$ *rel-induct-aux* P_c I_c *id*
by (*rule rule-LR*)
hence $(c_c, [b_c], [b_c, a_c]) \in$ *rel-induct-aux* P_c I_c *id* **by** *simp*
ultimately show $(c_c, [a_c, b_c], [b_c, a_c]) \in$ *rel-induct-aux* P_c I_c *id*
by (*rule rule-trans*)
qed
moreover have $c_c \in$ *next-events* P_c $[a_c, b_c] \cap$ *next-events* P_c $[b_c, a_c]$
by (*simp add*: *counterexample-next-events-2* T_c*-def*)
ultimately have $(a_c, [a_c, b_c] @ [c_c], [b_c, a_c] @ [c_c]) \in$ *rel-induct-aux* P_c I_c *id*
by (*rule rule-WSC*)
thus $(a_c, [a_c, b_c, c_c], [b_c, a_c, c_c]) \in$ *rel-induct-aux* P_c I_c *id* **by** *simp*
qed

lemma *counterexample-not-gu-condition*:
$\neg$ ($\exists\, R.$ *view-partition* P_c *id* $R \wedge$
weakly-future-consistent P_c I_c *id* $R \wedge$
weakly-step-consistent P_c *id* $R \wedge$
locally-respects P_c I_c *id* R)
proof (*rule notI*, *erule exE*, (*erule conjE*)+)
fix *R*
assume *weakly-future-consistent* P_c I_c *id R*
hence $\forall\, u \in$ *range id* $\cap$ $(-I_c)$ `` *range id*. $\forall\, xs\ ys.\ (xs, ys) \in R\ u \longrightarrow$
next-dom-events P_c *id u xs* = *next-dom-events* P_c *id u ys*
by (*simp add*: *weakly-future-consistent-def*)

moreover have $a_c \in range\ id \cap (-I_c)$ `` $range\ id$
by (*simp add*: I_c-*def*, *rule ImageI* [*of* b_c], *simp-all*)
ultimately have $\forall xs\ ys.\ (xs,\ ys) \in R\ a_c \longrightarrow$
$next\text{-}dom\text{-}events\ P_c\ id\ a_c\ xs = next\text{-}dom\text{-}events\ P_c\ id\ a_c\ ys$ **..**
hence $([a_c,\ b_c,\ c_c],\ [b_c,\ a_c,\ c_c]) \in R\ a_c \longrightarrow$
$next\text{-}dom\text{-}events\ P_c\ id\ a_c\ [a_c,\ b_c,\ c_c] = next\text{-}dom\text{-}events\ P_c\ id\ a_c\ [b_c,\ a_c,\ c_c]$
by *blast*
moreover assume
$view\text{-}partition\ P_c\ id\ R$ **and**
$weakly\text{-}step\text{-}consistent\ P_c\ id\ R$ **and**
$locally\text{-}respects\ P_c\ I_c\ id\ R$
hence $rel\text{-}induct\ P_c\ I_c\ id\ a_c \subseteq R\ a_c$ **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
$next\text{-}dom\text{-}events\ P_c\ id\ a_c\ [a_c,\ b_c,\ c_c] = next\text{-}dom\text{-}events\ P_c\ id\ a_c\ [b_c,\ a_c,\ c_c]$ **..**
thus *False*
by (*simp add*: *next-dom-events-def counterexample-next-events-2* T_c-*def*)
qed

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

end

References

[1] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, Inc., 1985.

[2] A. Krauss. *Defining Recursive Functions in Isabelle/HOL.* http://isabelle.in.tum.de/website-Isabelle2015/dist/Isabelle2015/doc/functions.pdf.

[3] T. Nipkow. *A Tutorial Introduction to Structured Isar Proofs.* http://isabelle.in.tum.de/website-Isabelle2011/dist/Isabelle2011/doc/isar-overview.pdf.

[4] T. Nipkow. *Programming and Proving in Isabelle/HOL*, May 2015. http://isabelle.in.tum.de/website-Isabelle2015/dist/Isabelle2015/doc/prog-prove.pdf.

[5] T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, May 2015. http://isabelle.in.tum.de/website-Isabelle2015/dist/Isabelle2015/doc/tutorial.pdf.

[6] P. Noce. Noninterference security in communicating sequential processes. *Archive of Formal Proofs*, May 2014. http://isa-afp.org/entries/Noninterference_CSP.shtml, Formal proof development.

[7] P. Noce. The ipurge unwinding theorem for csp noninterference security. *Archive of Formal Proofs*, June 2015. http://isa-afp.org/entries/Noninterference_Ipurge_Unwinding.shtml, Formal proof development.

[8] J. Rushby. Noninterference, transitivity, and channel-control security policies. Technical report, SRI International, 1992.