

# Conservation of CSP Noninterference Security under Sequential Composition

Pasquale Noce

Security Certification Specialist at Arjo Systems, Italy  
pasquale dot noce dot lavoro at gmail dot com  
pasquale dot noce at arjosystems dot com

September 13, 2023

## Abstract

In his outstanding work on Communicating Sequential Processes, Hoare has defined two fundamental binary operations allowing to compose the input processes into another, typically more complex, process: sequential composition and concurrent composition. Particularly, the output of the former operation is a process that initially behaves like the first operand, and then like the second operand once the execution of the first one has terminated successfully, as long as it does.

This paper formalizes Hoare's definition of sequential composition and proves, in the general case of a possibly intransitive policy, that CSP noninterference security is conserved under this operation, provided that successful termination cannot be affected by confidential events and cannot occur as an alternative to other events in the traces of the first operand. Both of these assumptions are shown, by means of counterexamples, to be necessary for the theorem to hold.

## Contents

<b>1</b>	<b>Propaedeutic definitions and lemmas</b>	<b>2</b>
1.1	Preliminary propaedeutic lemmas . . . . .	4
1.2	Intransitive purge of event sets with trivial base case . . . . .	11
1.3	Closure of the failures of a secure process under intransitive purge . . . . .	16
1.3.1	Step 1 . . . . .	18
1.3.2	Step 2 . . . . .	20
1.3.3	Step 3 . . . . .	21
1.3.4	Step 4 . . . . .	21
1.3.5	Step 5 . . . . .	23
1.3.6	Step 6 . . . . .	24
1.3.7	Step 7 . . . . .	25

1.3.8	Step 8 . . . . .	25
1.3.9	Step 9 . . . . .	26
1.3.10	Step 10 . . . . .	28
1.4	Additional propaedeutic lemmas . . . . .	30
<b>2</b>	<b>Sequential composition and noninterference security</b>	<b>35</b>
2.1	Sequential processes . . . . .	36
2.2	Sequential composition . . . . .	39
2.3	Conservation of refusals union closure and sequentiality under sequential composition . . . . .	49
2.4	Conservation of noninterference security under sequential composition . . . . .	67
2.5	Generalization of the security conservation theorem to lists of processes . . . . .	105
<b>3</b>	<b>Necessity of nontrivial assumptions</b>	<b>108</b>
3.1	Preliminary definitions and lemmas . . . . .	109
3.2	Necessity of termination security . . . . .	109
3.3	Necessity of process sequentiality . . . . .	114

## 1 Propaedeutic definitions and lemmas

**theory** *Propaedeutics*

**imports** *Noninterference-Ipurge-Unwinding.DeterministicProcesses*

**begin**

*To our Lord Jesus Christ, my dear parents, and my "little" sister,  
for the immense love with which they surround me.*

In his outstanding work on Communicating Sequential Processes [1], Hoare has defined two fundamental binary operations allowing to compose the input processes into another, typically more complex, process: sequential composition and concurrent composition. Particularly, the output of the former operation is a process that initially behaves like the first operand, and then like the second operand once the execution of the first one has terminated successfully, as long as it does. In order to distinguish it from deadlock, successful termination is regarded as a special event in the process alphabet (required to be the same for both the input processes and the output one).

This paper formalizes Hoare's definition of sequential composition and proves, in the general case of a possibly intransitive policy, that CSP noninterference security [8] is conserved under this operation, viz. the security of both of the input processes implies that of the output process.

This property is conditional on two nontrivial assumptions. The first assumption is that the policy do not allow successful termination to be affected by confidential events, viz. by other events not allowed to affect some event in the process alphabet. The second assumption is that successful termination do not occur as an alternative to other events in the traces of the first operand, viz. that whenever the process can terminate successfully, it cannot engage in any other event. Both of these assumptions are shown, by means of counterexamples, to be necessary for the theorem to hold.

From the above sketch of the sequential composition of two processes  $P$  and  $Q$ , notwithstanding its informal character, it clearly follows that any failure of the output process is either a failure of  $P$  (case A), or a pair  $(xs @ ys, Y)$ , where  $xs$  is a trace of  $P$  and  $(ys, Y)$  is a failure of  $Q$  (case B). On the other hand, according to the definition of security given in [8], the output process is secure just in case, for each of its failures, any event  $x$  contained in the failure trace can be removed from the trace, or inserted into the trace of another failure after the same previous events as in the original trace, and the resulting pair is still a failure of the process, provided that the future of  $x$  is deprived of the events that may be affected by  $x$ .

In case A, this transformation is performed on a failure of process  $P$ ; being it secure, the result is still a failure of  $P$ , and then of the output process. In case B, the transformation may involve either  $ys$  alone, or both  $xs$  and  $ys$ , depending on the position at which  $x$  is removed or inserted. In the former subcase, being  $Q$  secure, the result has the form  $(xs @ ys', Y')$  where  $(ys', Y')$  is a failure of  $Q$ , thus it is still a failure of the output process. In the latter subcase,  $ys$  has to be deprived of the events that may be affected by  $x$ , as well as by any event affected by  $x$  in the involved portion of  $xs$ , and a similar transformation applies to  $Y$ . In order that the output process be secure, the resulting pair  $(ys'', Y'')$  must still be a failure of  $Q$ , so that the pair  $(xs' @ ys'', Y'')$ , where  $xs'$  results from the transformation of  $xs$ , be a failure of the output process.

The transformations bringing from  $ys$  and  $Y$  to  $ys''$  and  $Y''$  are implemented by the functions *ipurge-tr-aux* and *ipurge-ref-aux* defined in [9]. Therefore, the proof of the target security conservation theorem requires that of the following lemma: given a process  $P$ , a noninterference policy  $I$ , and an event-domain map  $D$ , if  $P$  is secure with respect to  $I$  and  $D$  and  $(xs, X)$  is a failure of  $P$ , then  $(ipurge-tr-aux I D U xs, ipurge-ref-aux I D U xs X)$  is still a failure of  $P$ . In other words, the lemma states that the failures of a secure process are closed under intransitive purge. This section contains a proof of such closure lemma, as well as further definitions and lemmas required for the proof of the target theorem.

Throughout this paper, the salient points of definitions and proofs are commented; for additional information, cf. Isabelle documentation, particularly [6], [4], [3], and [2].

## 1.1 Preliminary propaedeutic lemmas

In what follows, some lemmas required for the demonstration of the target closure lemma are proven.

Here below is the proof of some properties of functions *ipurge-tr* and *ipurge-ref*.

**lemma** *ipurge-tr-length*:

$length (ipurge-tr I D u xs) \leq length xs$   
**by** (*induction xs rule: rev-induct, simp-all*)

**lemma** *ipurge-ref-swap*:

$ipurge-ref I D u xs \{x \in X. P x\} =$   
 $\{x \in ipurge-ref I D u xs X. P x\}$

**proof** (*simp add: ipurge-ref-def*)

**qed** *blast*

**lemma** *ipurge-ref-last*:

$ipurge-ref I D u (xs @ [x]) X =$   
*if*  $(u, D x) \in I \vee (\exists v \in sinks I D u xs. (v, D x) \in I)$   
*then*  $ipurge-ref I D u xs \{x' \in X. (D x, D x') \notin I\}$   
*else*  $ipurge-ref I D u xs X$

**proof** (*cases (u, D x) \in I \vee (\exists v \in sinks I D u xs. (v, D x) \in I),*  
*simp-all add: ipurge-ref-def*)

**qed** *blast*

Here below is the proof of some properties of function *sinks-aux*.

**lemma** *sinks-aux-append*:

$sinks-aux I D U (xs @ ys) = sinks-aux I D (sinks-aux I D U xs) ys$

**proof** (*induction ys rule: rev-induct, simp, subst append-assoc [symmetric]*)

**qed** (*simp del: append-assoc*)

**lemma** *sinks-aux-union*:

$sinks-aux I D (U \cup V) xs =$   
 $sinks-aux I D U xs \cup sinks-aux I D V (ipurge-tr-aux I D U xs)$

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

**fix**  $x xs$

**assume**  $A: sinks-aux I D (U \cup V) xs =$   
 $sinks-aux I D U xs \cup sinks-aux I D V (ipurge-tr-aux I D U xs)$

**show**  $sinks-aux I D (U \cup V) (xs @ [x]) =$   
 $sinks-aux I D U (xs @ [x]) \cup sinks-aux I D V (ipurge-tr-aux I D U (xs @ [x]))$

**proof** (*cases \exists w \in sinks-aux I D (U \cup V) xs. (w, D x) \in I*)

**case** *True*

**hence**  $\exists w \in sinks-aux I D U xs \cup sinks-aux I D V (ipurge-tr-aux I D U xs).$   
 $(w, D x) \in I$

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

**hence**  $(\exists w \in sinks-aux I D U xs. (w, D x) \in I) \vee$

$(\exists w \in \text{sinks-aux } I D V (\text{ipurge-tr-aux } I D U xs). (w, D x) \in I)$   
**by** *blast*  
**thus** *?thesis*  
**using** *A* **and** *True* **by** (*cases*  $\exists w \in \text{sinks-aux } I D U xs. (w, D x) \in I, \text{simp-all}$ )  
**next**  
**case** *False*  
**hence**  $\neg (\exists w \in \text{sinks-aux } I D U xs \cup$   
 $\text{sinks-aux } I D V (\text{ipurge-tr-aux } I D U xs). (w, D x) \in I)$   
**using** *A* **by** *simp*  
**hence**  $\neg (\exists w \in \text{sinks-aux } I D U xs. (w, D x) \in I) \wedge$   
 $\neg (\exists w \in \text{sinks-aux } I D V (\text{ipurge-tr-aux } I D U xs). (w, D x) \in I)$   
**by** *blast*  
**thus** *?thesis*  
**using** *A* **and** *False* **by** *simp*  
**qed**  
**qed**

**lemma** *sinks-aux-subset-dom*:

**assumes** *A*:  $U \subseteq V$   
**shows**  $\text{sinks-aux } I D U xs \subseteq \text{sinks-aux } I D V xs$   
**proof** (*induction xs rule: rev-induct, simp add: A, rule subsetI*)  
**fix** *x xs w*  
**assume**  
 $B: \text{sinks-aux } I D U xs \subseteq \text{sinks-aux } I D V xs$  **and**  
 $C: w \in \text{sinks-aux } I D U (xs @ [x])$   
**show**  $w \in \text{sinks-aux } I D V (xs @ [x])$   
**proof** (*cases*  $\exists u \in \text{sinks-aux } I D U xs. (u, D x) \in I$ )  
**case** *True*  
**hence**  $w = D x \vee w \in \text{sinks-aux } I D U xs$   
**using** *C* **by** *simp*  
**moreover** {  
**assume** *D*:  $w = D x$   
**obtain** *u* **where** *E*:  $u \in \text{sinks-aux } I D U xs$  **and** *F*:  $(u, D x) \in I$   
**using** *True* ..  
**have**  $u \in \text{sinks-aux } I D V xs$  **using** *B* **and** *E* ..  
**with** *F* **have**  $\exists u \in \text{sinks-aux } I D V xs. (u, D x) \in I$  ..  
**hence** *?thesis* **using** *D* **by** *simp*  
**}**  
**moreover** {  
**assume**  $w \in \text{sinks-aux } I D U xs$   
**with** *B* **have**  $w \in \text{sinks-aux } I D V xs$  ..  
**hence** *?thesis* **by** *simp*  
**}**  
**ultimately show** *?thesis* ..  
**next**  
**case** *False*  
**hence**  $w \in \text{sinks-aux } I D U xs$   
**using** *C* **by** *simp*  
**with** *B* **have**  $w \in \text{sinks-aux } I D V xs$  ..

thus *?thesis* by *simp*  
 qed  
 qed

**lemma** *sinks-aux-subset-ipurge-tr-aux*:

*sinks-aux I D U (ipurge-tr-aux I' D' U' xs) ⊆ sinks-aux I D U xs*

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

**fix** *x xs w*

**assume**

*A*: *sinks-aux I D U (ipurge-tr-aux I' D' U' xs) ⊆ sinks-aux I D U xs* **and**

*B*: *w ∈ sinks-aux I D U (ipurge-tr-aux I' D' U' (xs @ [x]))*

**show** *w ∈ sinks-aux I D U (xs @ [x])*

**proof** (*cases*  $\exists u \in \text{sinks-aux } I D U \text{ xs. } (u, D x) \in I$ , *simp-all* (*no-asm-simp*))

**from** *B* **have** *w = D x ∨ w ∈ sinks-aux I D U (ipurge-tr-aux I' D' U' xs)*

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

**qed** (*cases*  $\exists u \in \text{sinks-aux } I D U \text{ (ipurge-tr-aux } I' D' U' \text{ xs). } (u, D x) \in I$ , *simp-all*)

**moreover** {

**assume** *w = D x*

**hence** *w = D x ∨ w ∈ sinks-aux I D U xs ..*

}

**moreover** {

**assume** *w ∈ sinks-aux I D U (ipurge-tr-aux I' D' U' xs)*

**with** *A* **have** *w ∈ sinks-aux I D U xs ..*

**hence** *w = D x ∨ w ∈ sinks-aux I D U xs ..*

}

**ultimately show** *w = D x ∨ w ∈ sinks-aux I D U xs ..*

**next**

**assume** *C*:  $\neg (\exists u \in \text{sinks-aux } I D U \text{ xs. } (u, D x) \in I)$

**have** *w ∈ sinks-aux I D U (ipurge-tr-aux I' D' U' xs)*

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

**case** *True*

**thus** *w ∈ sinks-aux I D U (ipurge-tr-aux I' D' U' xs)*

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

**next**

**case** *False*

**hence** *w ∈ sinks-aux I D U (ipurge-tr-aux I' D' U' xs @ [x])*

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

**moreover have**

$\neg (\exists u \in \text{sinks-aux } I D U \text{ (ipurge-tr-aux } I' D' U' \text{ xs). } (u, D x) \in I)$

(**is**  $\neg ?P$ )

**proof**

**assume** *?P*

**then obtain** *u* **where**

*D*: *u ∈ sinks-aux I D U (ipurge-tr-aux I' D' U' xs)* **and**

*E*: *(u, D x) ∈ I ..*

**have** *u ∈ sinks-aux I D U xs* **using** *A* **and** *D* ..

**with** *E* **have**  $\exists u \in \text{sinks-aux } I D U \text{ xs. } (u, D x) \in I$  ..

**thus** *False* **using** *C* **by** *contradiction*

**qed**  
**ultimately show**  $w \in \text{sinks-aux } I D U (\text{ipurge-tr-aux } I' D' U' xs)$   
**by** *simp*  
**qed**  
**with**  $A$  **show**  $w \in \text{sinks-aux } I D U xs \dots$   
**qed**  
**qed**

**lemma** *sinks-aux-subset-ipurge-tr*:  
 $\text{sinks-aux } I D U (\text{ipurge-tr } I' D' u' xs) \subseteq \text{sinks-aux } I D U xs$   
**by** (*simp add: ipurge-tr-aux-single-dom [symmetric] sinks-aux-subset-ipurge-tr-aux*)

**lemma** *sinks-aux-member-ipurge-tr-aux* [rule-format]:

$u \in \text{sinks-aux } I D (U \cup V) xs \longrightarrow$   
 $(u, w) \in I \longrightarrow$   
 $\neg (\exists v \in \text{sinks-aux } I D V xs. (v, w) \in I) \longrightarrow$   
 $u \in \text{sinks-aux } I D U (\text{ipurge-tr-aux } I D V xs)$   
**proof** (*induction xs arbitrary: u w rule: rev-induct, (rule-tac [!] impI)+, simp*)

**fix**  $u w$   
**assume**  
 $A: (u, w) \in I$  **and**  
 $B: \forall v \in V. (v, w) \notin I$   
**assume**  $u \in U \vee u \in V$   
**moreover** {  
**assume**  $u \in U$   
**}**  
**moreover** {  
**assume**  $u \in V$   
**with**  $B$  **have**  $(u, w) \notin I \dots$   
**hence**  $u \in U$  **using**  $A$  **by** *contradiction*  
**}**  
**ultimately show**  $u \in U \dots$

**next**

**fix**  $x xs u w$   
**assume**  
 $A: \bigwedge u w. u \in \text{sinks-aux } I D (U \cup V) xs \longrightarrow$   
 $(u, w) \in I \longrightarrow$   
 $\neg (\exists v \in \text{sinks-aux } I D V xs. (v, w) \in I) \longrightarrow$   
 $u \in \text{sinks-aux } I D U (\text{ipurge-tr-aux } I D V xs)$  **and**  
 $B: u \in \text{sinks-aux } I D (U \cup V) (xs @ [x])$  **and**  
 $C: (u, w) \in I$  **and**  
 $D: \neg (\exists v \in \text{sinks-aux } I D V (xs @ [x]). (v, w) \in I)$   
**show**  $u \in \text{sinks-aux } I D U (\text{ipurge-tr-aux } I D V (xs @ [x]))$   
**proof** (*cases  $\exists u' \in \text{sinks-aux } I D (U \cup V) xs. (u', D x) \in I$* )  
**case** *True*  
**hence**  $u = D x \vee u \in \text{sinks-aux } I D (U \cup V) xs$   
**using**  $B$  **by** *simp*  
**moreover** {  
**assume**  $E: u = D x$

**obtain**  $u'$  **where**  $u' \in \text{sinks-aux } I D (U \cup V) xs$  **and**  $F: (u', D x) \in I$   
**using** *True* ..  
**moreover have**  $u' \in \text{sinks-aux } I D (U \cup V) xs \longrightarrow$   
 $(u', D x) \in I \longrightarrow$   
 $\neg (\exists v \in \text{sinks-aux } I D V xs. (v, D x) \in I) \longrightarrow$   
 $u' \in \text{sinks-aux } I D U (\text{ipurge-tr-aux } I D V xs)$   
**(is**  $\longrightarrow - \longrightarrow \neg ?P \longrightarrow ?Q$ ) **using** *A* .  
**ultimately have**  $\neg ?P \longrightarrow ?Q$   
**by** *simp*  
**moreover have**  $\neg ?P$   
**proof**  
**have**  $(D x, w) \in I$  **using** *C* **and** *E* **by** *simp*  
**moreover assume**  $?P$   
**hence**  $D x \in \text{sinks-aux } I D V (xs @ [x])$  **by** *simp*  
**ultimately have**  $\exists v \in \text{sinks-aux } I D V (xs @ [x]). (v, w) \in I$  ..  
**moreover have**  $\neg (\exists v \in \text{sinks-aux } I D V (xs @ [x]). (v, w) \in I)$   
**using** *D* **by** *simp*  
**ultimately show** *False* **by** *contradiction*  
**qed**  
**ultimately have**  $?Q$  ..  
**with** *F* **have**  $\exists u' \in \text{sinks-aux } I D U (\text{ipurge-tr-aux } I D V xs)$ .  
 $(u', D x) \in I$  ..  
**hence**  $D x \in \text{sinks-aux } I D U (\text{ipurge-tr-aux } I D V xs @ [x])$   
**by** *simp*  
**moreover have**  $\text{ipurge-tr-aux } I D V xs @ [x] =$   
 $\text{ipurge-tr-aux } I D V (xs @ [x])$   
**using**  $\langle \neg ?P \rangle$  **by** *simp*  
**ultimately have**  $?thesis$  **using** *E* **by** *simp*  
**}**  
**moreover {**  
**assume**  $u \in \text{sinks-aux } I D (U \cup V) xs$   
**moreover have**  $u \in \text{sinks-aux } I D (U \cup V) xs \longrightarrow$   
 $(u, w) \in I \longrightarrow$   
 $\neg (\exists v \in \text{sinks-aux } I D V xs. (v, w) \in I) \longrightarrow$   
 $u \in \text{sinks-aux } I D U (\text{ipurge-tr-aux } I D V xs)$   
**(is**  $\longrightarrow - \longrightarrow \neg ?P \longrightarrow ?Q$ ) **using** *A* .  
**ultimately have**  $\neg ?P \longrightarrow ?Q$   
**using** *C* **by** *simp*  
**moreover have**  $\neg ?P$   
**proof**  
**assume**  $?P$   
**hence**  $\exists v \in \text{sinks-aux } I D V (xs @ [x]). (v, w) \in I$   
**by** *simp*  
**thus** *False* **using** *D* **by** *contradiction*  
**qed**  
**ultimately have**  $u \in \text{sinks-aux } I D U (\text{ipurge-tr-aux } I D V xs)$  ..  
**hence**  $?thesis$  **by** *simp*  
**}**  
**ultimately show**  $?thesis$  ..



**next**  
**case** *False*  
**hence**  $u \in \text{sinks-aux } I D (U \cup V) xs$   
**using** *B* **by** *simp*  
**moreover have**  $u \in \text{sinks-aux } I D (U \cup V) xs \longrightarrow$   
 $(u, w) \in I \longrightarrow$   
 $\neg (\exists v \in \text{sinks-aux } I D V xs. (v, w) \in I) \longrightarrow$   
 $u \in \text{sinks-aux } I D U (\text{ipurge-tr-aux } I D V xs)$   
**(is**  $\longrightarrow \neg \longrightarrow \neg ?P \longrightarrow ?Q$  **using** *A* **.**  
**ultimately have**  $\neg ?P \longrightarrow ?Q$   
**using** *C* **by** *simp*  
**moreover have**  $\neg ?P$   
**proof**  
**assume**  $?P$   
**hence**  $\exists v \in \text{sinks-aux } I D V (xs @ [x]). (v, w) \in I$   
**by** *simp*  
**thus** *False* **using** *D* **by** *contradiction*  
**qed**  
**ultimately have**  $u \in \text{sinks-aux } I D U (\text{ipurge-tr-aux } I D V xs) ..$   
**thus** *?thesis* **by** *simp*  
**qed**  
**qed**

**lemma** *sinks-aux-member-ipurge-tr*:  
**assumes**  
*A*:  $u \in \text{sinks-aux } I D (\text{insert } v U) xs$  **and**  
*B*:  $(u, w) \in I$  **and**  
*C*:  $\neg ((v, w) \in I \vee (\exists v' \in \text{sinks } I D v xs. (v', w) \in I))$   
**shows**  $u \in \text{sinks-aux } I D U (\text{ipurge-tr } I D v xs)$   
**proof** (*subst ipurge-tr-aux-single-dom [symmetric]*,  
*rule-tac w = w in sinks-aux-member-ipurge-tr-aux*)  
**show**  $u \in \text{sinks-aux } I D (U \cup \{v\}) xs$   
**using** *A* **by** *simp*  
**next**  
**show**  $(u, w) \in I$   
**using** *B* **.**  
**next**  
**show**  $\neg (\exists v' \in \text{sinks-aux } I D \{v\} xs. (v', w) \in I)$   
**using** *C* **by** (*simp add: sinks-aux-single-dom*)  
**qed**

Here below is the proof of some properties of functions *ipurge-tr-aux* and *ipurge-ref-aux*.

**lemma** *ipurge-tr-aux-append*:  
 $\text{ipurge-tr-aux } I D U (xs @ ys) =$   
 $\text{ipurge-tr-aux } I D U xs @ \text{ipurge-tr-aux } I D (\text{sinks-aux } I D U xs) ys$   
**proof** (*induction ys rule: rev-induct, simp, subst append-assoc [symmetric]*)

**qed** (*simp add: sinks-aux-append del: append-assoc*)

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

*ipurge-tr-aux I D U [x] = (if  $\exists v \in U. (v, D x) \in I$   
then  $\square$   
else [x])*

**by** (*subst (2) append-Nil [symmetric], simp del: append-Nil*)

**lemma** *ipurge-tr-aux-cons*:

*ipurge-tr-aux I D U (x # xs) = (if  $\exists u \in U. (u, D x) \in I$   
then *ipurge-tr-aux I D (insert (D x) U) xs*  
else  $x \# \text{ipurge-tr-aux I D U xs}$ )*

**proof** –

**have** *ipurge-tr-aux I D U (x # xs) = ipurge-tr-aux I D U ([x] @ xs)*

**by** *simp*

**also have**  $\dots =$

*ipurge-tr-aux I D U [x] @ ipurge-tr-aux I D (sinks-aux I D U [x]) xs*

**by** (*simp only: ipurge-tr-aux-append*)

**finally show** *?thesis*

**by** (*simp add: sinks-aux-single-event ipurge-tr-aux-single-event*)

**qed**

**lemma** *ipurge-tr-aux-union*:

*ipurge-tr-aux I D (U  $\cup$  V) xs =  
ipurge-tr-aux I D V (ipurge-tr-aux I D U xs)*

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

**fix**  $x xs$

**assume**  $A$ : *ipurge-tr-aux I D (U  $\cup$  V) xs =*

*ipurge-tr-aux I D V (ipurge-tr-aux I D U xs)*

**show** *ipurge-tr-aux I D (U  $\cup$  V) (xs @ [x]) =*

*ipurge-tr-aux I D V (ipurge-tr-aux I D U (xs @ [x]))*

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

**case** *True*

**hence**  $\exists w \in \text{sinks-aux I D U xs} \cup \text{sinks-aux I D V (ipurge-tr-aux I D U xs).$

$(w, D x) \in I$

**by** (*simp add: sinks-aux-union*)

**hence**  $(\exists w \in \text{sinks-aux I D U xs. (w, D x) \in I) \vee$

$(\exists w \in \text{sinks-aux I D V (ipurge-tr-aux I D U xs). (w, D x) \in I)$

**by** *blast*

**thus** *?thesis*

**using**  $A$  **and** *True* **by** (*cases  $\exists w \in \text{sinks-aux I D U xs. (w, D x) \in I, simp-all$* )

**next**

**case** *False*

**hence**  $\neg (\exists w \in \text{sinks-aux I D U xs} \cup$

$\text{sinks-aux I D V (ipurge-tr-aux I D U xs). (w, D x) \in I)$

**by** (*simp add: sinks-aux-union*)

**hence**  $\neg (\exists w \in \text{sinks-aux I D U xs. (w, D x) \in I) \wedge$

$\neg (\exists w \in \text{sinks-aux I D V (ipurge-tr-aux I D U xs). (w, D x) \in I)$

**by** *blast*

**thus** *?thesis*  
**using** *A and False by simp*  
**qed**  
**qed**

**lemma** *ipurge-tr-aux-insert:*  
*ipurge-tr-aux I D (insert v U) xs =*  
*ipurge-tr-aux I D U (ipurge-tr I D v xs)*  
**by** (*subst insert-is-Un, simp only: ipurge-tr-aux-union ipurge-tr-aux-single-dom*)

**lemma** *ipurge-ref-aux-subset:*  
*ipurge-ref-aux I D U xs X  $\subseteq$  X*  
**by** (*subst ipurge-ref-aux-def, rule subsetI, simp*)

## 1.2 Intransitive purge of event sets with trivial base case

Here below are the definitions of variants of functions *sinks-aux* and *ipurge-ref-aux*, respectively named *sinks-aux-less* and *ipurge-ref-aux-less*, such that their base cases in correspondence with an empty input list are trivial, viz. such that *sinks-aux-less I D U [] = {}* and *ipurge-ref-aux-less I D U [] X = X*. These functions will prove to be useful in what follows.

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

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

Here below is the proof of some properties of function *sinks-aux-less* used in what follows.

**lemma** *sinks-aux-sinks-aux-less:*  
*sinks-aux I D U xs = U  $\cup$  sinks-aux-less I D U xs*  
**by** (*induction xs rule: rev-induct, simp-all*)

**lemma** *sinks-aux-less-single-dom:*  
*sinks-aux-less I D {u} xs = sinks I D u xs*

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

**lemma** *sinks-aux-less-single-event*:

*sinks-aux-less I D U [x] = (if  $\exists u \in U. (u, D x) \in I$  then  $\{D x\}$  else  $\{\}$ )*  
**by** (*subst append-Nil [symmetric], simp del: append-Nil*)

**lemma** *sinks-aux-less-append*:

*sinks-aux-less I D U (xs @ ys) =*  
*sinks-aux-less I D U xs  $\cup$  sinks-aux-less I D (U  $\cup$  sinks-aux-less I D U xs) ys*  
**proof** (*induction ys rule: rev-induct, simp, subst append-assoc [symmetric]*)  
**qed** (*simp del: append-assoc*)

**lemma** *sinks-aux-less-cons*:

*sinks-aux-less I D U (x # xs) = (if  $\exists u \in U. (u, D x) \in I$*   
*then insert (D x) (sinks-aux-less I D (insert (D x) U) xs)*  
*else sinks-aux-less I D U xs)*

**proof** –

**have** *sinks-aux-less I D U (x # xs) = sinks-aux-less I D U ([x] @ xs)*

**by** *simp*

**also have**  $\dots =$

*sinks-aux-less I D U [x]  $\cup$  sinks-aux-less I D (U  $\cup$  sinks-aux-less I D U [x]) xs*

**by** (*simp only: sinks-aux-less-append*)

**finally show** *?thesis*

**by** (*cases  $\exists u \in U. (u, D x) \in I$ , simp-all add: sinks-aux-less-single-event*)

**qed**

Here below is the proof of some properties of function *ipurge-ref-aux-less* used in what follows.

**lemma** *ipurge-ref-aux-less-last*:

*ipurge-ref-aux-less I D U (xs @ [x]) X =*  
*(if  $\exists v \in U \cup$  sinks-aux-less I D U xs.  $(v, D x) \in I$*   
*then ipurge-ref-aux-less I D U xs  $\{x' \in X. (D x, D x') \notin I\}$*   
*else ipurge-ref-aux-less I D U xs X)*

**by** (*cases  $\exists v \in U \cup$  sinks-aux-less I D U xs.  $(v, D x) \in I$ ,*  
*simp-all add: ipurge-ref-aux-less-def*)

**lemma** *ipurge-ref-aux-less-nil*:

*ipurge-ref-aux-less I D U xs (ipurge-ref-aux I D U [] X) =*  
*ipurge-ref-aux I D U xs X*

**proof** (*simp add: ipurge-ref-aux-def ipurge-ref-aux-less-def sinks-aux-sinks-aux-less*)  
**qed** *blast*

**lemma** *ipurge-ref-aux-less-cons-1*:

**assumes** *A:  $\exists u \in U. (u, D x) \in I$*

**shows** *ipurge-ref-aux-less I D U (x # xs) X =*

*ipurge-ref-aux-less I D U (ipurge-tr I D (D x) xs) (ipurge-ref I D (D x) xs X)*

**proof** (*induction xs arbitrary: X rule: rev-induct,*

*simp add: ipurge-ref-def ipurge-ref-aux-less-def sinks-aux-less-single-event A*  
**fix**  $x' xs X$   
**assume**  $B: \bigwedge X.$   
 $ipurge-ref-aux-less I D U (x \# xs) X =$   
 $ipurge-ref-aux-less I D U (ipurge-tr I D (D x) xs)$   
 $(ipurge-ref I D (D x) xs X)$   
**show**  
 $ipurge-ref-aux-less I D U (x \# xs @ [x']) X =$   
 $ipurge-ref-aux-less I D U (ipurge-tr I D (D x) (xs @ [x']))$   
 $(ipurge-ref I D (D x) (xs @ [x']) X)$   
**proof** (*cases*  $\exists v \in U \cup sinks-aux-less I D U (x \# xs). (v, D x') \in I$ )  
**assume**  $C: \exists v \in U \cup sinks-aux-less I D U (x \# xs). (v, D x') \in I$   
**hence**  $ipurge-ref-aux-less I D U (x \# xs @ [x']) X =$   
 $ipurge-ref-aux-less I D U (x \# xs) \{y \in X. (D x', D y) \notin I\}$   
**by** (*subst append-Cons [symmetric]*),  
*simp add: ipurge-ref-aux-less-last del: append-Cons*  
**also have**  $\dots =$   
 $ipurge-ref-aux-less I D U (ipurge-tr I D (D x) xs)$   
 $(ipurge-ref I D (D x) xs \{y \in X. (D x', D y) \notin I\})$   
**using**  $B$  .  
**finally have**  $D: ipurge-ref-aux-less I D U (x \# xs @ [x']) X =$   
 $ipurge-ref-aux-less I D U (ipurge-tr I D (D x) xs)$   
 $(ipurge-ref I D (D x) xs \{y \in X. (D x', D y) \notin I\})$  .  
**show** *?thesis*  
**proof** (*cases*  $(D x, D x') \in I \vee (\exists v \in sinks I D (D x) xs. (v, D x') \in I)$ )  
**case** *True*  
**hence**  $ipurge-ref I D (D x) xs \{y \in X. (D x', D y) \notin I\} =$   
 $ipurge-ref I D (D x) (xs @ [x']) X$   
**by** (*simp add: ipurge-ref-last*)  
**moreover have**  $D x' \in sinks I D (D x) (xs @ [x'])$   
**using** *True* **by** (*simp only: sinks-interference-eq*)  
**hence**  $ipurge-tr I D (D x) xs = ipurge-tr I D (D x) (xs @ [x'])$   
**by** *simp*  
**ultimately show** *?thesis* **using**  $D$  **by** *simp*  
**next**  
**case** *False*  
**hence**  $ipurge-ref I D (D x) xs \{y \in X. (D x', D y) \notin I\} =$   
 $ipurge-ref I D (D x) (xs @ [x']) \{y \in X. (D x', D y) \notin I\}$   
**by** (*simp add: ipurge-ref-last*)  
**also have**  $\dots = \{y \in ipurge-ref I D (D x) (xs @ [x']) X. (D x', D y) \notin I\}$   
**by** (*simp add: ipurge-ref-swap*)  
**finally have**  $ipurge-ref-aux-less I D U (x \# xs @ [x']) X =$   
 $ipurge-ref-aux-less I D U (ipurge-tr I D (D x) xs)$   
 $\{y \in ipurge-ref I D (D x) (xs @ [x']) X. (D x', D y) \notin I\}$   
**using**  $D$  **by** *simp*  
**also have**  $\dots = ipurge-ref-aux-less I D U (ipurge-tr I D (D x) xs @ [x'])$   
 $(ipurge-ref I D (D x) (xs @ [x']) X)$   
**proof** –  
**have**  $\exists v \in U \cup sinks-aux-less I D U (ipurge-tr I D (D x) xs).$

$(v, D x') \in I$   
**proof** –  
**obtain**  $v$  **where**  
 $E: v \in U \cup \text{sinks-aux-less } I D U (x \# xs)$  **and**  
 $F: (v, D x') \in I$   
**using**  $C$  ..  
**have**  $v \in \text{sinks-aux } I D U (x \# xs)$   
**using**  $E$  **by** (*simp add: sinks-aux-sinks-aux-less*)  
**hence**  $v \in \text{sinks-aux } I D (insert (D x) U) xs$   
**using**  $A$  **by** (*simp add: sinks-aux-cons*)  
**hence**  $v \in \text{sinks-aux } I D U (ipurge-tr I D (D x) xs)$   
**using**  $F$  **and**  $False$  **by** (*rule sinks-aux-member-ipurge-tr*)  
**hence**  $v \in U \cup \text{sinks-aux-less } I D U (ipurge-tr I D (D x) xs)$   
**by** (*simp add: sinks-aux-sinks-aux-less*)  
**with**  $F$  **show** *?thesis* ..  
**qed**  
**thus** *?thesis* **by** (*simp add: ipurge-ref-aux-less-last*)  
**qed**  
**finally** **have** *ipurge-ref-aux-less*  $I D U (x \# xs @ [x']) X =$   
 $ipurge-ref-aux-less I D U (ipurge-tr I D (D x) xs @ [x'])$   
 $(ipurge-ref I D (D x) (xs @ [x']) X)$  .  
**moreover** **have**  $D x' \notin \text{sinks } I D (D x) (xs @ [x'])$   
**using**  $False$  **by** (*simp only: sinks-interference-eq, simp*)  
**hence**  $ipurge-tr I D (D x) xs @ [x'] = ipurge-tr I D (D x) (xs @ [x'])$   
**by** *simp*  
**ultimately** **show** *?thesis* **by** *simp*  
**qed**  
**next**  
**assume**  $C: \neg (\exists v \in U \cup \text{sinks-aux-less } I D U (x \# xs). (v, D x') \in I)$   
**hence** *ipurge-ref-aux-less*  $I D U (x \# xs @ [x']) X =$   
 $ipurge-ref-aux-less I D U (x \# xs) X$   
**by** (*subst append-Cons [symmetric]*),  
 $simp$  *add: ipurge-ref-aux-less-last del: append-Cons*)  
**also** **have** ... =  
 $ipurge-ref-aux-less I D U (ipurge-tr I D (D x) xs)$   
 $(ipurge-ref I D (D x) xs X)$   
**using**  $B$  .  
**also** **have** ... =  
 $ipurge-ref-aux-less I D U (ipurge-tr I D (D x) xs @ [x'])$   
 $(ipurge-ref I D (D x) xs X)$   
**proof** –  
**have**  $\neg (\exists v \in U \cup \text{sinks-aux-less } I D U (ipurge-tr I D (D x) xs).$   
 $(v, D x') \in I)$  (**is**  $\neg ?P$ )  
**proof**  
**assume**  $?P$   
**then** **obtain**  $v$  **where**  
 $D: v \in U \cup \text{sinks-aux-less } I D U (ipurge-tr I D (D x) xs)$  **and**  
 $E: (v, D x') \in I$  ..  
**have**  $\text{sinks-aux } I D U (ipurge-tr I D (D x) xs) \subseteq \text{sinks-aux } I D U xs$

by (rule sinks-aux-subset-ipurge-tr)  
 moreover have  $v \in \text{sinks-aux } I D U$  (ipurge-tr  $I D (D x) xs$ )  
 using  $D$  by (simp add: sinks-aux-sinks-aux-less)  
 ultimately have  $v \in \text{sinks-aux } I D U xs ..$   
 moreover have  $U \subseteq \text{insert } (D x) U$   
 by (rule subset-insertI)  
 hence  $\text{sinks-aux } I D U xs \subseteq \text{sinks-aux } I D (\text{insert } (D x) U) xs$   
 by (rule sinks-aux-subset-dom)  
 ultimately have  $v \in \text{sinks-aux } I D (\text{insert } (D x) U) xs ..$   
 hence  $v \in \text{sinks-aux } I D U (x \# xs)$   
 using  $A$  by (simp add: sinks-aux-cons)  
 hence  $v \in U \cup \text{sinks-aux-less } I D U (x \# xs)$   
 by (simp add: sinks-aux-sinks-aux-less)  
 with  $E$  have  $\exists v \in U \cup \text{sinks-aux-less } I D U (x \# xs). (v, D x') \in I ..$   
 thus *False* using  $C$  by contradiction  
 qed  
 thus ?thesis by (simp add: ipurge-ref-aux-less-last)  
 qed  
 also have ... =  
 ipurge-ref-aux-less  $I D U$  (ipurge-tr  $I D (D x) (xs @ [x'])$ )  
 (ipurge-ref  $I D (D x) (xs @ [x']) X$ )  
 proof -  
 have  $\neg ((D x, D x') \in I \vee (\exists v \in \text{sinks } I D (D x) xs. (v, D x') \in I))$   
 (is  $\neg ?P$ )  
 proof (rule notI, erule disjE)  
 assume  $D: (D x, D x') \in I$   
 have  $\text{insert } (D x) U \subseteq \text{sinks-aux } I D (\text{insert } (D x) U) xs$   
 by (rule sinks-aux-subset)  
 moreover have  $D x \in \text{insert } (D x) U$   
 by simp  
 ultimately have  $D x \in \text{sinks-aux } I D (\text{insert } (D x) U) xs ..$   
 hence  $D x \in \text{sinks-aux } I D U (x \# xs)$   
 using  $A$  by (simp add: sinks-aux-cons)  
 hence  $D x \in U \cup \text{sinks-aux-less } I D U (x \# xs)$   
 by (simp add: sinks-aux-sinks-aux-less)  
 with  $D$  have  $\exists v \in U \cup \text{sinks-aux-less } I D U (x \# xs). (v, D x') \in I ..$   
 thus *False* using  $C$  by contradiction  
 next  
 assume  $\exists v \in \text{sinks } I D (D x) xs. (v, D x') \in I$   
 then obtain  $v$  where  
 $D: v \in \text{sinks } I D (D x) xs$  and  
 $E: (v, D x') \in I ..$   
 have  $\{D x\} \subseteq \text{insert } (D x) U$   
 by simp  
 hence  $\text{sinks-aux } I D \{D x\} xs \subseteq \text{sinks-aux } I D (\text{insert } (D x) U) xs$   
 by (rule sinks-aux-subset-dom)  
 moreover have  $v \in \text{sinks-aux } I D \{D x\} xs$   
 using  $D$  by (simp add: sinks-aux-single-dom)  
 ultimately have  $v \in \text{sinks-aux } I D (\text{insert } (D x) U) xs ..$

**hence**  $v \in \text{sinks-aux } I D U (x \# xs)$   
**using**  $A$  **by** (*simp add: sinks-aux-cons*)  
**hence**  $v \in U \cup \text{sinks-aux-less } I D U (x \# xs)$   
**by** (*simp add: sinks-aux-sinks-aux-less*)  
**with**  $E$  **have**  $\exists v \in U \cup \text{sinks-aux-less } I D U (x \# xs). (v, D x') \in I ..$   
**thus**  $False$  **using**  $C$  **by** *contradiction*  
**qed**  
**hence**  $\text{ipurge-tr } I D (D x) xs @ [x'] = \text{ipurge-tr } I D (D x) (xs @ [x'])$   
**by** (*simp only: sinks-interference-eq, simp*)  
**moreover** **have**  $\text{ipurge-ref } I D (D x) xs X =$   
 $\text{ipurge-ref } I D (D x) (xs @ [x']) X$   
**using**  $\langle \neg ?P \rangle$  **by** (*simp add: ipurge-ref-last*)  
**ultimately** **show**  $?thesis$  **by** *simp*  
**qed**  
**finally** **show**  $?thesis .$   
**qed**  
**qed**

**lemma** *ipurge-ref-aux-less-cons-2*:  
 $\neg (\exists u \in U. (u, D x) \in I) \implies$   
 $\text{ipurge-ref-aux-less } I D U (x \# xs) X =$   
 $\text{ipurge-ref-aux-less } I D U xs X$   
**by** (*simp add: ipurge-ref-aux-less-def sinks-aux-less-cons*)

### 1.3 Closure of the failures of a secure process under intransitive purge

The intransitive purge of an event list  $xs$  with regard to a policy  $I$ , an event-domain map  $D$ , and a set of domains  $U$  can equivalently be computed as follows: for each item  $x$  of  $xs$ , if  $x$  may be affected by some domain in  $U$ , discard  $x$  and go on recursively using  $\text{ipurge-tr } I D (D x) xs'$  as input, where  $xs'$  is the sublist of  $xs$  following  $x$ ; otherwise, retain  $x$  and go on recursively using  $xs'$  as input.

In fact, in each recursive step, any item allowed to be indirectly affected by  $U$  through the effect of some item preceding  $x$  within  $xs$  has already been removed from the list. Hence, it is sufficient to check whether  $x$  may be directly affected by  $U$ , and remove  $x$ , as well as any residual item allowed to be affected by  $x$ , if this is the case.

Similarly, the intransitive purge of an event set  $X$  with regard to a policy  $I$ , an event-domain map  $D$ , a set of domains  $U$ , and an event list  $xs$  can be computed as follows. First of all, compute  $\text{ipurge-ref-aux } I D U [] X$  and use this set, along with  $xs$ , as the input for the subsequent step. Then, for each item  $x$  of  $xs$ , if  $x$  may be affected by some domain in  $U$ , go on recursively using  $\text{ipurge-tr } I D (D x) xs'$  and  $\text{ipurge-ref } I D (D x) xs' X'$  as input, where  $X'$  is the set input to the current recursive step; otherwise, go on recursively using  $xs'$  and  $X'$  as input.



In fact, in each recursive step, any item allowed to be affected by  $U$  either directly, or through the effect of some item preceding  $x$  within  $xs$ , has already been removed from the set (in the initial step and in subsequent steps, respectively). Thus, it is sufficient to check whether  $x$  may be directly affected by  $U$ , and remove any residual item allowed to be affected by  $x$  if this is the case.

Assume that the two computations be performed simultaneously by a single function, which will then take as input an event list-event set pair and return as output another such pair. Then, if the input pair is a failure of a secure process, the output pair is still a failure. In fact, for each item  $x$  of  $xs$  allowed to be affected by  $U$ , if  $ys$  is the partial output list for the sublist of  $xs$  preceding  $x$ , then  $(ys @ \text{ipurge-tr } I D (D x) xs', \text{ipurge-ref } I D (D x) xs' X')$  is a failure provided that such is  $(ys @ x \# xs', X')$ , by virtue of the definition of CSP noninterference security [8]. Hence, the property of being a failure is conserved upon each recursive call by the event list-event set pair such that the list matches the concatenation of the partial output list with the residual input list, and the set matches the residual input set. This holds until the residual input list is nil, which is the base case determining the end of the computation.

As shown by this argument, a proof by induction that the output event list-event set pair, under the aforesaid assumptions, is still a failure, requires that the partial output list be passed to the function as a further argument, in addition to the residual input list, in the recursive calls contained within the definition of the function. Therefore, the output list has to be accumulated into a parameter of the function, viz. the function needs to be tail-recursive. This suggests to prove the properties of interest of the function by applying the ten-step proof method for theorems on tail-recursive functions described in [7].

The starting point is to formulate a naive definition of the function, which will then be refined as specified by the proof method. A slight complication is due to the preliminary replacement of the input event set  $X$  with  $\text{ipurge-ref-aux } I D U [] X$ , to be performed before the items of the input event list start to be consumed recursively. A simple solution to this problem is to nest the accumulator of the output list within data type *option*. In this way, the initial state can be distinguished from the subsequent one, in which the input event list starts to be consumed, by assigning the distinct values *None* and *Some []*, respectively, to the accumulator.

Everything is now ready for giving a naive definition of the function under consideration:

**function** (*sequential*) *ipurge-fail-aux-t-naive* ::  
 $( 'd \times 'd ) \text{ set} \Rightarrow ( 'a \Rightarrow 'd ) \Rightarrow 'd \text{ set} \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ list option} \Rightarrow 'a \text{ set} \Rightarrow$   
 $'a \text{ failure}$

**where**

```

ipurge-fail-aux-t-naive I D U xs None X =
  ipurge-fail-aux-t-naive I D U xs (Some []) (ipurge-ref-aux I D U [] X) |
ipurge-fail-aux-t-naive I D U (x # xs) (Some ys) X =
  (if ∃ u ∈ U. (u, D x) ∈ I
   then ipurge-fail-aux-t-naive I D U
        (ipurge-tr I D (D x) xs) (Some ys) (ipurge-ref I D (D x) xs X)
   else ipurge-fail-aux-t-naive I D U
        xs (Some (ys @ [x])) X) |
ipurge-fail-aux-t-naive - - - (Some ys) X = (ys, X)
oops

```

The parameter into which the output list is accumulated is the last but one. As shown by the above informal argument, function *ipurge-fail-aux-t-naive* enjoys the following properties:

$$fst (ipurge-fail-aux-t-naive I D U xs None X) = ipurge-tr-aux I D U xs$$

$$snd (ipurge-fail-aux-t-naive I D U xs None X) = ipurge-ref-aux I D U xs X$$

$$\llbracket secure P I D; (xs, X) \in failures P \rrbracket \implies ipurge-fail-aux-t-naive I D U xs None X \in failures P$$

which altogether imply the target lemma, viz. the closure of the failures of a secure process under intransitive purge.

In what follows, the steps provided for by the aforesaid proof method will be dealt with one after the other, with the purpose of proving the target closure lemma in the final step. For more information on this proof method, cf. [7].

### 1.3.1 Step 1

In the definition of the auxiliary tail-recursive function *ipurge-fail-aux-t-aux*, the Cartesian product of the input parameter types of function *ipurge-fail-aux-t-naive* will be implemented as the following record type:

```

record ('a, 'd) ipurge-rec =
  Pol :: ('d × 'd) set
  Map :: 'a ⇒ 'd
  Doms :: 'd set
  List :: 'a list
  ListOp :: 'a list option
  Set :: 'a set

```

Here below is the resulting definition of function *ipurge-fail-aux-t-aux*:

**function** *ipurge-fail-aux-t-aux* :: ('a, 'd) *ipurge-rec* ⇒ ('a, 'd) *ipurge-rec*  
**where**

*ipurge-fail-aux-t-aux* (⟦Pol = I, Map = D, Doms = U, List = xs,  
 ListOp = None, Set = X⟧) =  
*ipurge-fail-aux-t-aux* (⟦Pol = I, Map = D, Doms = U, List = xs,  
 ListOp = Some [], Set = *ipurge-ref-aux* I D U [] X⟧) |

*ipurge-fail-aux-t-aux* (⟦Pol = I, Map = D, Doms = U, List = x # xs,  
 ListOp = Some ys, Set = X⟧) =  
 (if ∃ u ∈ U. (u, D x) ∈ I  
 then *ipurge-fail-aux-t-aux* (⟦Pol = I, Map = D, Doms = U,  
 List = *ipurge-tr* I D (D x) xs, ListOp = Some ys,  
 Set = *ipurge-ref* I D (D x) xs X⟧)  
 else *ipurge-fail-aux-t-aux* (⟦Pol = I, Map = D, Doms = U,  
 List = xs, ListOp = Some (ys @ [x]), Set = X⟧) |

*ipurge-fail-aux-t-aux*  
 (⟦Pol = I, Map = D, Doms = U, List = [], ListOp = Some ys, Set = X⟧) =  
 (⟦Pol = I, Map = D, Doms = U, List = [], ListOp = Some ys, Set = X⟧)

**proof** (*simp-all*, *atomize-elim*)

**fix** Y :: ('a, 'd) *ipurge-rec*

**show**

(∃ I D U xs X. Y = (⟦Pol = I, Map = D, Doms = U, List = xs,  
 ListOp = None, Set = X⟧) ∨  
 (∃ I D U x xs ys X. Y = (⟦Pol = I, Map = D, Doms = U, List = x # xs,  
 ListOp = Some ys, Set = X⟧) ∨  
 (∃ I D U ys X. Y = (⟦Pol = I, Map = D, Doms = U, List = [],  
 ListOp = Some ys, Set = X⟧))

**proof** (*cases* Y, *simp*)

**fix** xs :: 'a list **and** yso :: 'a list option

**show**

yso = None ∨  
 (∃ x' xs'. xs = x' # xs') ∧ (∃ ys. yso = Some ys) ∨  
 xs = [] ∧ (∃ ys. yso = Some ys)

**proof** (*cases* yso, *simp-all*)

**qed** (*subst disj-commute*, *rule spec* [OF *list.nchotomy*])

**qed**

**qed**

The length of the input event list of function *ipurge-fail-aux-t-aux* decreases in every recursive call except for the first one, where the input list is left unchanged while the nested output list passes from *None* to *Some* []. A

measure function decreasing in the first recursive call as well can then be obtained by increasing the length of the input list by one in case the nested output list matches *None*. Using such a measure function, the termination of function *ipurge-fail-aux-t-aux* is guaranteed by the fact that the event lists output by function *ipurge-tr* are not longer than the corresponding input ones.

**termination** *ipurge-fail-aux-t-aux*

**proof** (relation measure ( $\lambda Y. (if ListOp Y = None then Suc else id)$  (length (List Y))), *simp-all*)

**fix**  $D :: 'a \Rightarrow 'd$  and  $I x xs$

**have** length (*ipurge-tr*  $I D (D x) xs$ )  $\leq$  length  $xs$  **by** (rule *ipurge-tr-length*)

**thus** length (*ipurge-tr*  $I D (D x) xs$ )  $<$  Suc (length  $xs$ ) **by** *simp*

**qed**

### 1.3.2 Step 2

**definition** *ipurge-fail-aux-t-in* ::

$('d \times 'd) set \Rightarrow ('a \Rightarrow 'd) \Rightarrow 'd set \Rightarrow 'a list \Rightarrow 'a set \Rightarrow ('a, 'd) ipurge-rec$

**where**

*ipurge-fail-aux-t-in*  $I D U xs X \equiv$

$(\lfloor Pol = I, Map = D, Doms = U, List = xs, ListOp = None, Set = X \rfloor)$

**definition** *ipurge-fail-aux-t-out* ::  $('a, 'd) ipurge-rec \Rightarrow 'a failure$  **where**

*ipurge-fail-aux-t-out*  $Y \equiv (case ListOp Y of Some ys \Rightarrow ys, Set Y)$

**definition** *ipurge-fail-aux-t* ::

$('d \times 'd) set \Rightarrow ('a \Rightarrow 'd) \Rightarrow 'd set \Rightarrow 'a list \Rightarrow 'a set \Rightarrow 'a failure$

**where**

*ipurge-fail-aux-t*  $I D U xs X \equiv$

*ipurge-fail-aux-t-out* (*ipurge-fail-aux-t-aux* (*ipurge-fail-aux-t-in*  $I D U xs X$ ))

Since the significant inputs of function *ipurge-fail-aux-t-naive* match pattern  $\_, \_, \_, \_, None, \_$ , those of function *ipurge-fail-aux-t-aux*, as returned by function *ipurge-fail-aux-t-in*, match pattern  $(\lfloor Pol = \_, Map = \_, Doms = \_, List = \_, ListOp = None, Set = \_ \rfloor)$ .

Likewise, since the nested output lists returned by function *ipurge-fail-aux-t-aux* match pattern *Some*  $\_$ , function *ipurge-fail-aux-t-out* does not need to worry about dealing with nested output lists equal to *None*.

In terms of function *ipurge-fail-aux-t*, the statements to be proven in order to demonstrate the target closure lemma, previously expressed using function *ipurge-fail-aux-t-naive* and henceforth respectively named *ipurge-fail-aux-t-eq-tr*, *ipurge-fail-aux-t-eq-ref*, and *ipurge-fail-aux-t-failures*, take the following form:

$fst (ipurge-fail-aux-t I D U xs X) = ipurge-tr-aux I D U xs$

$snd (ipurge-fail-aux-t I D U xs X) = ipurge-ref-aux I D U xs X$

$\llbracket secure P I D; (xs, X) \in failures P \rrbracket \implies ipurge-fail-aux-t I D U xs X \in failures P$

### 1.3.3 Step 3

**inductive-set**  $ipurge-fail-aux-t-set$  ::  
 ( $'a, 'd$ )  $ipurge-rec \Rightarrow ('a, 'd) ipurge-rec set$   
**for**  $Y$  :: ( $'a, 'd$ )  $ipurge-rec$  **where**

$R0$ :  $Y \in ipurge-fail-aux-t-set Y$  |

$R1$ :  $(\langle Pol = I, Map = D, Doms = U, List = xs, ListOp = None, Set = X \rangle \in ipurge-fail-aux-t-set Y \implies$   
 $\langle Pol = I, Map = D, Doms = U, List = xs, ListOp = Some [], Set = ipurge-ref-aux I D U [] X \rangle \in ipurge-fail-aux-t-set Y$  |

$R2$ :  $\llbracket \langle Pol = I, Map = D, Doms = U, List = x \# xs, ListOp = Some ys, Set = X \rangle \in ipurge-fail-aux-t-set Y;$   
 $\exists u \in U. (u, D x) \in I \rrbracket \implies$   
 $\langle Pol = I, Map = D, Doms = U, List = ipurge-tr I D (D x) xs, ListOp = Some ys, Set = ipurge-ref I D (D x) xs X \rangle \in ipurge-fail-aux-t-set Y$  |

$R3$ :  $\llbracket \langle Pol = I, Map = D, Doms = U, List = x \# xs, ListOp = Some ys, Set = X \rangle \in ipurge-fail-aux-t-set Y;$   
 $\neg (\exists u \in U. (u, D x) \in I) \rrbracket \implies$   
 $\langle Pol = I, Map = D, Doms = U, List = xs, ListOp = Some (ys @ [x]), Set = X \rangle \in ipurge-fail-aux-t-set Y$

### 1.3.4 Step 4

**lemma**  $ipurge-fail-aux-t-subset$ :

**assumes**  $A$ :  $Z \in ipurge-fail-aux-t-set Y$

**shows**  $ipurge-fail-aux-t-set Z \subseteq ipurge-fail-aux-t-set Y$

**proof** (rule subsetI, erule  $ipurge-fail-aux-t-set.induct$ )

**show**  $Z \in ipurge-fail-aux-t-set Y$  **using**  $A$  .

**next**

**fix**  $I D U xs X$

**assume**  $(\langle Pol = I, Map = D, Doms = U, List = xs, ListOp = None, Set = X \rangle \in ipurge-fail-aux-t-set Y$

**thus**  $(\langle Pol = I, Map = D, Doms = U, List = xs,$

$ListOp = Some [], Set = ipurge-ref-aux I D U [] X \rangle \in ipurge-fail-aux-t-set Y$

**by** (rule  $R1$ )

**next**

**fix**  $I D U x xs ys X$

**assume**

$\langle Pol = I, Map = D, Doms = U, List = x \# xs, ListOp = Some\ ys, Set = X \rangle \in ipurge\text{-}fail\text{-}aux\text{-}t\text{-}set\ Y$  **and**  
 $\exists u \in U. (u, D\ x) \in I$   
**thus**  $\langle Pol = I, Map = D, Doms = U, List = ipurge\text{-}tr\ I\ D\ (D\ x)\ xs, ListOp = Some\ ys, Set = ipurge\text{-}ref\ I\ D\ (D\ x)\ xs\ X \rangle \in ipurge\text{-}fail\text{-}aux\text{-}t\text{-}set\ Y$   
**by** (rule R2)  
**next**  
**fix**  $I\ D\ U\ x\ xs\ ys\ X$   
**assume**  
 $\langle Pol = I, Map = D, Doms = U, List = x \# xs, ListOp = Some\ ys, Set = X \rangle \in ipurge\text{-}fail\text{-}aux\text{-}t\text{-}set\ Y$  **and**  
 $\neg (\exists u \in U. (u, D\ x) \in I)$   
**thus**  $\langle Pol = I, Map = D, Doms = U, List = xs, ListOp = Some\ (ys\ @\ [x]), Set = X \rangle \in ipurge\text{-}fail\text{-}aux\text{-}t\text{-}set\ Y$   
**by** (rule R3)  
**qed**

**lemma** *ipurge-fail-aux-t-aux-set*:

$ipurge\text{-}fail\text{-}aux\text{-}t\text{-}aux\ Y \in ipurge\text{-}fail\text{-}aux\text{-}t\text{-}set\ Y$   
**proof** (induction rule: *ipurge-fail-aux-t-aux.induct*,  
*simp-all add: R0 del: ipurge-fail-aux-t-aux.simps(2)*)  
**fix**  $I\ U\ xs\ X$  **and**  $D :: 'a \Rightarrow 'd$   
**let**  
 $?Y = \langle Pol = I, Map = D, Doms = U, List = xs, ListOp = None, Set = X \rangle$  **and**  
 $?Y' = \langle Pol = I, Map = D, Doms = U, List = xs, ListOp = Some\ [], Set = ipurge\text{-}ref\text{-}aux\ I\ D\ U\ []\ X \rangle$   
**have**  $?Y \in ipurge\text{-}fail\text{-}aux\text{-}t\text{-}set\ ?Y$   
**by** (rule R0)  
**moreover have**  $?Y \in ipurge\text{-}fail\text{-}aux\text{-}t\text{-}set\ ?Y \implies$   
 $?Y' \in ipurge\text{-}fail\text{-}aux\text{-}t\text{-}set\ ?Y$   
**by** (rule R1)  
**ultimately have**  $?Y' \in ipurge\text{-}fail\text{-}aux\text{-}t\text{-}set\ ?Y$   
**by** *simp*  
**hence**  $ipurge\text{-}fail\text{-}aux\text{-}t\text{-}set\ ?Y' \subseteq ipurge\text{-}fail\text{-}aux\text{-}t\text{-}set\ ?Y$   
**by** (rule *ipurge-fail-aux-t-subset*)  
**moreover assume**  $ipurge\text{-}fail\text{-}aux\text{-}t\text{-}aux\ ?Y' \in ipurge\text{-}fail\text{-}aux\text{-}t\text{-}set\ ?Y'$   
**ultimately show**  $ipurge\text{-}fail\text{-}aux\text{-}t\text{-}aux\ ?Y' \in ipurge\text{-}fail\text{-}aux\text{-}t\text{-}set\ ?Y$  ..  
**next**  
**fix**  $I\ U\ x\ xs\ ys\ X$  **and**  $D :: 'a \Rightarrow 'd$   
**let**  
 $?Y = \langle Pol = I, Map = D, Doms = U, List = x \# xs, ListOp = Some\ ys, Set = X \rangle$  **and**  
 $?Y' = \langle Pol = I, Map = D, Doms = U, List = ipurge\text{-}tr\ I\ D\ (D\ x)\ xs, ListOp = Some\ ys, Set = ipurge\text{-}ref\ I\ D\ (D\ x)\ xs\ X \rangle$  **and**  
 $?Y'' = \langle Pol = I, Map = D, Doms = U, List = xs, ListOp = Some\ (ys\ @\ [x]), Set = X \rangle$   
**assume**  
 $A: \exists u \in U. (u, D\ x) \in I \implies$

$ipurge\text{-}fail\text{-}aux\text{-}t\text{-}aux\ ?Y' \in ipurge\text{-}fail\text{-}aux\text{-}t\text{-}set\ ?Y'$  **and**  
 $B: \forall u \in U. (u, D\ x) \notin I \implies$   
 $ipurge\text{-}fail\text{-}aux\text{-}t\text{-}aux\ ?Y'' \in ipurge\text{-}fail\text{-}aux\text{-}t\text{-}set\ ?Y''$   
**show**  $ipurge\text{-}fail\text{-}aux\text{-}t\text{-}aux\ ?Y \in ipurge\text{-}fail\text{-}aux\text{-}t\text{-}set\ ?Y$   
**proof** (cases  $\exists u \in U. (u, D\ x) \in I$ , simp-all (no-asm-simp))  
**case** *True*  
**have**  $?Y \in ipurge\text{-}fail\text{-}aux\text{-}t\text{-}set\ ?Y$   
**by** (rule *R0*)  
**moreover have**  $?Y \in ipurge\text{-}fail\text{-}aux\text{-}t\text{-}set\ ?Y \implies \exists u \in U. (u, D\ x) \in I \implies$   
 $?Y' \in ipurge\text{-}fail\text{-}aux\text{-}t\text{-}set\ ?Y$   
**by** (rule *R2*)  
**ultimately have**  $?Y' \in ipurge\text{-}fail\text{-}aux\text{-}t\text{-}set\ ?Y$   
**using** *True* **by** *simp*  
**hence**  $ipurge\text{-}fail\text{-}aux\text{-}t\text{-}set\ ?Y' \subseteq ipurge\text{-}fail\text{-}aux\text{-}t\text{-}set\ ?Y$   
**by** (rule *ipurge-fail-aux-t-subset*)  
**moreover have**  $ipurge\text{-}fail\text{-}aux\text{-}t\text{-}aux\ ?Y' \in ipurge\text{-}fail\text{-}aux\text{-}t\text{-}set\ ?Y'$   
**using** *A* **and** *True* **by** *simp*  
**ultimately show**  $ipurge\text{-}fail\text{-}aux\text{-}t\text{-}aux\ ?Y' \in ipurge\text{-}fail\text{-}aux\text{-}t\text{-}set\ ?Y ..$   
**next**  
**case** *False*  
**have**  $?Y \in ipurge\text{-}fail\text{-}aux\text{-}t\text{-}set\ ?Y$   
**by** (rule *R0*)  
**moreover have**  $?Y \in ipurge\text{-}fail\text{-}aux\text{-}t\text{-}set\ ?Y \implies$   
 $\neg (\exists u \in U. (u, D\ x) \in I) \implies ?Y'' \in ipurge\text{-}fail\text{-}aux\text{-}t\text{-}set\ ?Y$   
**by** (rule *R3*)  
**ultimately have**  $?Y'' \in ipurge\text{-}fail\text{-}aux\text{-}t\text{-}set\ ?Y$   
**using** *False* **by** *simp*  
**hence**  $ipurge\text{-}fail\text{-}aux\text{-}t\text{-}set\ ?Y'' \subseteq ipurge\text{-}fail\text{-}aux\text{-}t\text{-}set\ ?Y$   
**by** (rule *ipurge-fail-aux-t-subset*)  
**moreover have**  $ipurge\text{-}fail\text{-}aux\text{-}t\text{-}aux\ ?Y'' \in ipurge\text{-}fail\text{-}aux\text{-}t\text{-}set\ ?Y''$   
**using** *B* **and** *False* **by** *simp*  
**ultimately show**  $ipurge\text{-}fail\text{-}aux\text{-}t\text{-}aux\ ?Y'' \in ipurge\text{-}fail\text{-}aux\text{-}t\text{-}set\ ?Y ..$   
**qed**  
**qed**

### 1.3.5 Step 5

**definition** *ipurge-fail-aux-t-inv-1* ::

$(\ 'd \times \ 'd) \text{ set} \Rightarrow (\ 'a \Rightarrow \ 'd) \Rightarrow \ 'd \text{ set} \Rightarrow \ 'a \text{ list} \Rightarrow (\ 'a, \ 'd) \text{ ipurge-rec} \Rightarrow \text{bool}$

**where**

$ipurge\text{-}fail\text{-}aux\text{-}t\text{-}inv\text{-}1\ I\ D\ U\ xs\ Y \equiv$

$(\text{case } ListOp\ Y\ of\ None \Rightarrow [] \mid Some\ ys \Rightarrow ys) @ ipurge\text{-}tr\text{-}aux\ I\ D\ U\ (List\ Y) = ipurge\text{-}tr\text{-}aux\ I\ D\ U\ xs$

**definition** *ipurge-fail-aux-t-inv-2* ::

$(\ 'd \times \ 'd) \text{ set} \Rightarrow (\ 'a \Rightarrow \ 'd) \Rightarrow \ 'd \text{ set} \Rightarrow \ 'a \text{ list} \Rightarrow \ 'a \text{ set} \Rightarrow$

$(\ 'a, \ 'd) \text{ ipurge-rec} \Rightarrow \text{bool}$

**where**

$ipurge\text{-}fail\text{-}aux\text{-}t\text{-}inv\text{-}2\ I\ D\ U\ xs\ X\ Y \equiv$

if  $ListOp\ Y = None$   
 then  $List\ Y = xs \wedge Set\ Y = X$   
 else  $ipurge-ref-aux-less\ I\ D\ U\ (List\ Y)\ (Set\ Y) = ipurge-ref-aux\ I\ D\ U\ xs\ X$

**definition**  $ipurge-fail-aux-t-inv-3 ::$   
 $'a\ process \Rightarrow ('d \times 'd)\ set \Rightarrow ('a \Rightarrow 'd) \Rightarrow 'a\ list \Rightarrow 'a\ set \Rightarrow$   
 $('a, 'd)\ ipurge-rec \Rightarrow bool$

**where**

$ipurge-fail-aux-t-inv-3\ P\ I\ D\ xs\ X\ Y \equiv$   
 $secure\ P\ I\ D \longrightarrow (xs, X) \in failures\ P \longrightarrow$   
 $((case\ ListOp\ Y\ of\ None \Rightarrow [] \mid Some\ ys \Rightarrow ys) @ List\ Y, Set\ Y) \in failures\ P$

Three invariants have been defined, one for each of lemmas  $ipurge-fail-aux-t-eq-tr$ ,  $ipurge-fail-aux-t-eq-ref$ , and  $ipurge-fail-aux-t-failures$ . More precisely, the invariants are  $ipurge-fail-aux-t-inv-1\ I\ D\ U\ xs$ ,  $ipurge-fail-aux-t-inv-2\ I\ D\ U\ xs\ X$ , and  $ipurge-fail-aux-t-inv-3\ P\ I\ D\ xs\ X$ , where the free variables are intended to match those appearing in the aforesaid lemmas.

Particularly:

- The first invariant expresses the fact that in each recursive step, any item of the residual input list  $List\ Y$  indirectly affected by  $U$  through the effect of previous, already consumed items has already been removed from the list, so that applying function  $ipurge-tr-aux\ I\ D\ U$  to the list is sufficient to obtain the intransitive purge of the whole original list.
- The second invariant expresses the fact that in each recursive step, any item of the residual input set  $Set\ Y$  affected by  $U$  either directly, or through the effect of previous, already consumed items, has already been removed from the set, so that applying function  $ipurge-ref-aux-less\ I\ D\ U\ (List\ Y)$  to the set is sufficient to obtain the intransitive purge of the whole original set.  
 The use of function  $ipurge-ref-aux-less$  ensures that the invariant implies the equality  $Set\ Y = ipurge-ref-aux\ I\ D\ U\ xs\ X$  for  $List\ Y = []$ , viz. for the output values of function  $ipurge-fail-aux-t-aux$ , which is the reason requiring the introduction of function  $ipurge-ref-aux-less$ .
- The third invariant expresses the fact that in each recursive step, the event list-event set pair such that the list matches the concatenation of the partial output list with  $List\ Y$ , and the set matches  $Set\ Y$ , is a failure provided that the original input pair is such as well.

### 1.3.6 Step 6

**lemma**  $ipurge-fail-aux-t-input-1:$



*ipurge-fail-aux-t-inv-1*  $I D U xs$   
 $\langle Pol = I, Map = D, Doms = U, List = xs, ListOp = None, Set = X \rangle$   
**by** (*simp add: ipurge-fail-aux-t-inv-1-def*)

**lemma** *ipurge-fail-aux-t-input-2*:  
*ipurge-fail-aux-t-inv-2*  $I D U xs X$   
 $\langle Pol = I, Map = D, Doms = U, List = xs, ListOp = None, Set = X \rangle$   
**by** (*simp add: ipurge-fail-aux-t-inv-2-def*)

**lemma** *ipurge-fail-aux-t-input-3*:  
*ipurge-fail-aux-t-inv-3*  $P I D xs X$   
 $\langle Pol = I, Map = D, Doms = U, List = xs, ListOp = None, Set = X \rangle$   
**by** (*simp add: ipurge-fail-aux-t-inv-3-def*)

### 1.3.7 Step 7

**definition** *ipurge-fail-aux-t-form* ::  $(\prime a, \prime d)$  *ipurge-rec*  $\Rightarrow$  *bool* **where**  
*ipurge-fail-aux-t-form*  $Y \equiv$   
*case* *ListOp*  $Y$  *of* *None*  $\Rightarrow$  *False* | *Some*  $ys \Rightarrow$  *List*  $Y = []$

**lemma** *ipurge-fail-aux-t-intro-1*:  
 $\llbracket ipurge-fail-aux-t-inv-1 I D U xs Y; ipurge-fail-aux-t-form Y \rrbracket \Longrightarrow$   
 $fst (ipurge-fail-aux-t-out Y) = ipurge-tr-aux I D U xs$   
**proof** (*simp add: ipurge-fail-aux-t-inv-1-def ipurge-fail-aux-t-form-def ipurge-fail-aux-t-out-def*)  
**qed** (*simp split: option.split-asm*)

**lemma** *ipurge-fail-aux-t-intro-2*:  
 $\llbracket ipurge-fail-aux-t-inv-2 I D U xs X Y; ipurge-fail-aux-t-form Y \rrbracket \Longrightarrow$   
 $snd (ipurge-fail-aux-t-out Y) = ipurge-ref-aux I D U xs X$   
**proof** (*simp add: ipurge-fail-aux-t-inv-2-def ipurge-fail-aux-t-form-def ipurge-fail-aux-t-out-def*)  
**qed** (*simp add: ipurge-ref-aux-less-def split: option.split-asm*)

**lemma** *ipurge-fail-aux-t-intro-3*:  
 $\llbracket ipurge-fail-aux-t-inv-3 P I D xs X Y; ipurge-fail-aux-t-form Y \rrbracket \Longrightarrow$   
 $secure P I D \longrightarrow (xs, X) \in failures P \longrightarrow$   
 $ipurge-fail-aux-t-out Y \in failures P$   
**proof** (*simp add: ipurge-fail-aux-t-inv-3-def ipurge-fail-aux-t-form-def ipurge-fail-aux-t-out-def*)  
**qed** (*simp split: option.split-asm*)

### 1.3.8 Step 8

**lemma** *ipurge-fail-aux-t-form-aux*:  
*ipurge-fail-aux-t-form* (*ipurge-fail-aux-t-aux*  $Y$ )  
**by** (*induction*  $Y$  *rule: ipurge-fail-aux-t-aux.induct,*  
*simp-all add: ipurge-fail-aux-t-form-def*)

### 1.3.9 Step 9

**lemma** *ipurge-fail-aux-t-invariance-aux*:

$Z \in \text{ipurge-fail-aux-t-set } Y \implies$

$\text{Pol } Z = \text{Pol } Y \wedge \text{Map } Z = \text{Map } Y \wedge \text{Doms } Z = \text{Doms } Y$

**by** (*erule ipurge-fail-aux-t-set.induct, simp-all*)

The lemma just proven, stating the invariance of the first three record fields over inductive set *ipurge-fail-aux-t-set*  $Y$ , is used in the following proofs of the invariance of predicates *ipurge-fail-aux-t-inv-1*  $I D U xs$ , *ipurge-fail-aux-t-inv-2*  $I D U xs X$ , and *ipurge-fail-aux-t-inv-3*  $P I D xs X$ .

The equality between the free variables appearing in the predicates and the corresponding fields of the record generating the set, which is required for such invariance properties to hold, is asserted in the enunciation of the properties by means of record updates. In the subsequent proofs of lemmas *ipurge-fail-aux-t-eq-tr*, *ipurge-fail-aux-t-eq-ref*, and *ipurge-fail-aux-t-failures*, the enforcement of this equality will be ensured by the identification of both predicate variables and record fields with the related free variables appearing in the lemmas.

**lemma** *ipurge-fail-aux-t-invariance-1*:

$\llbracket Z \in \text{ipurge-fail-aux-t-set } (Y \langle \text{Pol} := I, \text{Map} := D, \text{Doms} := U \rangle);$

$\text{ipurge-fail-aux-t-inv-1 } I D U xs \ (Y \langle \text{Pol} := I, \text{Map} := D, \text{Doms} := U \rangle) \rrbracket \implies$

$\text{ipurge-fail-aux-t-inv-1 } I D U xs Z$

**proof** (*erule ipurge-fail-aux-t-set.induct, assumption,*

*drule-tac* [!] *ipurge-fail-aux-t-invariance-aux,*

*simp-all add: ipurge-fail-aux-t-inv-1-def*)

**fix**  $x \ xs' \ ys$

**assume**  $ys @ \text{ipurge-tr-aux } I D U (x \# xs') = \text{ipurge-tr-aux } I D U xs$

(**is**  $?A = ?C$ )

**moreover assume**  $\exists u \in U. (u, D x) \in I$

**hence**  $?A = ys @ \text{ipurge-tr-aux } I D (\text{insert } (D x) U) xs'$

**by** (*simp add: ipurge-tr-aux-cons*)

**hence**  $?A = ys @ \text{ipurge-tr-aux } I D U (\text{ipurge-tr } I D (D x) xs')$

(**is**  $- = ?B$ ) **by** (*simp add: ipurge-tr-aux-insert*)

**ultimately show**  $?B = ?C$  **by** *simp*

**next**

**fix**  $x \ xs' \ ys$

**assume**  $ys @ \text{ipurge-tr-aux } I D U (x \# xs') = \text{ipurge-tr-aux } I D U xs$

(**is**  $?A = ?C$ )

**moreover assume**  $\forall u \in U. (u, D x) \notin I$

**hence**  $?A = ys @ x \# \text{ipurge-tr-aux } I D U xs'$

(**is**  $- = ?B$ ) **by** (*simp add: ipurge-tr-aux-cons*)

**ultimately show**  $?B = ?C$  **by** *simp*

**qed**

**lemma** *ipurge-fail-aux-t-invariance-2*:

$\llbracket Z \in \text{ipurge-fail-aux-t-set } (Y(\text{Pol} := I, \text{Map} := D, \text{Doms} := U));$   
 $\text{ipurge-fail-aux-t-inv-2 } I D U xs X (Y(\text{Pol} := I, \text{Map} := D, \text{Doms} := U)) \rrbracket \implies$   
 $\text{ipurge-fail-aux-t-inv-2 } I D U xs X Z$

**proof** (*erule ipurge-fail-aux-t-set.induct, assumption,*  
*drule-tac [!] ipurge-fail-aux-t-invariance-aux,*  
*simp-all add: ipurge-fail-aux-t-inv-2-def*)  
**show**  $\text{ipurge-ref-aux-less } I D U xs (\text{ipurge-ref-aux } I D U [] X) =$   
 $\text{ipurge-ref-aux } I D U xs X$   
**by** (*rule ipurge-ref-aux-less-nil*)

**next**  
**fix**  $x xs' X'$   
**assume**  $\text{ipurge-ref-aux-less } I D U (x \# xs') X' = \text{ipurge-ref-aux } I D U xs X$   
*(is ?A = ?C)*  
**moreover assume**  $\exists u \in U. (u, D x) \in I$   
**hence**  $?A = \text{ipurge-ref-aux-less } I D U (\text{ipurge-tr } I D (D x) xs')$   
 $(\text{ipurge-ref } I D (D x) xs' X')$   
*(is - = ?B)* **by** (*rule ipurge-ref-aux-less-cons-1*)  
**ultimately show**  $?B = ?C$  **by** *simp*

**next**  
**fix**  $x xs' X'$   
**assume**  $\text{ipurge-ref-aux-less } I D U (x \# xs') X' = \text{ipurge-ref-aux } I D U xs X$   
*(is ?A = ?C)*  
**moreover assume**  $\forall u \in U. (u, D x) \notin I$   
**hence**  $\neg (\exists u \in U. (u, D x) \in I)$  **by** *simp*  
**hence**  $?A = \text{ipurge-ref-aux-less } I D U xs' X'$   
*(is - = ?B)* **by** (*rule ipurge-ref-aux-less-cons-2*)  
**ultimately show**  $?B = ?C$  **by** *simp*

**qed**

**lemma ipurge-fail-aux-t-invariance-3:**  
 $\llbracket Z \in \text{ipurge-fail-aux-t-set } (Y(\text{Pol} := I, \text{Map} := D));$   
 $\text{ipurge-fail-aux-t-inv-3 } P I D xs X (Y(\text{Pol} := I, \text{Map} := D)) \rrbracket \implies$   
 $\text{ipurge-fail-aux-t-inv-3 } P I D xs X Z$

**proof** (*erule ipurge-fail-aux-t-set.induct, assumption,*  
*drule-tac [!] ipurge-fail-aux-t-invariance-aux,*  
*simp-all add: ipurge-fail-aux-t-inv-3-def, (rule-tac [!] impI)+*)  
**fix**  $xs' X'$   
**assume**  
*secure P I D and*  
*(xs, X) \in failures P and*  
*secure P I D \longrightarrow (xs, X) \in failures P \longrightarrow (xs', X') \in failures P*  
**hence**  $(xs', X') \in \text{failures } P$   
**by** *simp*  
**moreover have**  $\text{ipurge-ref-aux } I D (\text{Doms } Y) [] X' \subseteq X'$   
**by** (*rule ipurge-ref-aux-subset*)  
**ultimately show**  $(xs', \text{ipurge-ref-aux } I D (\text{Doms } Y) [] X') \in \text{failures } P$   
**by** (*rule process-rule-3*)

**next**  
**fix**  $x xs' ys X'$

**assume**  $S$ : *secure P I D* **and**  
 $(xs, X) \in \text{failures } P$  **and**  
*secure P I D*  $\longrightarrow (xs, X) \in \text{failures } P \longrightarrow (ys @ x \# xs', X') \in \text{failures } P$   
**hence**  $(ys @ x \# xs', X') \in \text{failures } P$   
**by** *simp*  
**hence**  $(x \# xs', X') \in \text{futures } P \text{ } ys$   
**by** (*simp add: futures-def*)  
**hence** (*ipurge-tr I D (D x) xs', ipurge-ref I D (D x) xs' X'*)  $\in \text{futures } P \text{ } ys$   
**using**  $S$  **by** (*simp add: secure-def*)  
**thus** (*ipurge-tr I D (D x) xs', ipurge-ref I D (D x) xs' X'*)  $\in \text{failures } P$   
**by** (*simp add: futures-def*)  
**qed**

### 1.3.10 Step 10

Here below are the proofs of lemmas *ipurge-fail-aux-t-eq-tr*, *ipurge-fail-aux-t-eq-ref*, and *ipurge-fail-aux-t-failures*, which are then applied to demonstrate the target closure lemma.

**lemma** *ipurge-fail-aux-t-eq-tr*:

*fst (ipurge-fail-aux-t I D U xs X) = ipurge-tr-aux I D U xs*

**proof** –

**let**  $?Y = (\text{Pol} = I, \text{Map} = D, \text{Doms} = U, \text{List} = xs, \text{ListOp} = \text{None}, \text{Set} = X)$

**have** *ipurge-fail-aux-t-aux ?Y*

$\in \text{ipurge-fail-aux-t-set } (?Y(\text{Pol} := I, \text{Map} := D, \text{Doms} := U))$

**by** (*simp add: ipurge-fail-aux-t-aux-set del: ipurge-fail-aux-t-aux.simps*)

**moreover have**

*ipurge-fail-aux-t-inv-1 I D U xs (?Y(\text{Pol} := I, \text{Map} := D, \text{Doms} := U))*

**by** (*simp add: ipurge-fail-aux-t-input-1*)

**ultimately have** *ipurge-fail-aux-t-inv-1 I D U xs (ipurge-fail-aux-t-aux ?Y)*

**by** (*rule ipurge-fail-aux-t-invariance-1*)

**moreover have** *ipurge-fail-aux-t-form (ipurge-fail-aux-t-aux ?Y)*

**by** (*rule ipurge-fail-aux-t-form-aux*)

**ultimately have** *fst (ipurge-fail-aux-t-out (ipurge-fail-aux-t-aux ?Y)) =*

*ipurge-tr-aux I D U xs*

**by** (*rule ipurge-fail-aux-t-intro-1*)

**moreover have**  $?Y = \text{ipurge-fail-aux-t-in } I D U xs X$

**by** (*simp add: ipurge-fail-aux-t-in-def*)

**ultimately show** *?thesis*

**by** (*simp add: ipurge-fail-aux-t-def*)

**qed**

**lemma** *ipurge-fail-aux-t-eq-ref*:

*snd (ipurge-fail-aux-t I D U xs X) = ipurge-ref-aux I D U xs X*

**proof** –

**let**  $?Y = (\text{Pol} = I, \text{Map} = D, \text{Doms} = U, \text{List} = xs, \text{ListOp} = \text{None}, \text{Set} = X)$

**have** *ipurge-fail-aux-t-aux* ?Y  
 ∈ *ipurge-fail-aux-t-set* (?Y( $\lfloor$ Pol := I, Map := D, Doms := U $\rfloor$ ))  
**by** (*simp add: ipurge-fail-aux-t-aux-set del: ipurge-fail-aux-t-aux.simps*)  
**moreover have**  
*ipurge-fail-aux-t-inv-2* I D U xs X (?Y( $\lfloor$ Pol := I, Map := D, Doms := U $\rfloor$ ))  
**by** (*simp add: ipurge-fail-aux-t-input-2*)  
**ultimately have** *ipurge-fail-aux-t-inv-2* I D U xs X (*ipurge-fail-aux-t-aux* ?Y)  
**by** (*rule ipurge-fail-aux-t-invariance-2*)  
**moreover have** *ipurge-fail-aux-t-form* (*ipurge-fail-aux-t-aux* ?Y)  
**by** (*rule ipurge-fail-aux-t-form-aux*)  
**ultimately have** *snd* (*ipurge-fail-aux-t-out* (*ipurge-fail-aux-t-aux* ?Y)) =  
*ipurge-ref-aux* I D U xs X  
**by** (*rule ipurge-fail-aux-t-intro-2*)  
**moreover have** ?Y = *ipurge-fail-aux-t-in* I D U xs X  
**by** (*simp add: ipurge-fail-aux-t-in-def*)  
**ultimately show** ?thesis  
**by** (*simp add: ipurge-fail-aux-t-def*)  
**qed**

**lemma** *ipurge-fail-aux-t-failures* [*rule-format*]:

*secure* P I D  $\longrightarrow$  (xs, X) ∈ *failures* P  $\longrightarrow$   
*ipurge-fail-aux-t* I D U xs X ∈ *failures* P

**proof** –

**let** ?Y = ( $\lfloor$ Pol = I, Map = D, Doms = U, List = xs, ListOp = None,  
 Set = X $\rfloor$ )  
**have** *ipurge-fail-aux-t-aux* ?Y  
 ∈ *ipurge-fail-aux-t-set* (?Y( $\lfloor$ Pol := I, Map := D $\rfloor$ ))  
**by** (*simp add: ipurge-fail-aux-t-aux-set del: ipurge-fail-aux-t-aux.simps*)  
**moreover have**  
*ipurge-fail-aux-t-inv-3* P I D xs X (?Y( $\lfloor$ Pol := I, Map := D $\rfloor$ ))  
**by** (*simp add: ipurge-fail-aux-t-input-3*)  
**ultimately have** *ipurge-fail-aux-t-inv-3* P I D xs X (*ipurge-fail-aux-t-aux* ?Y)  
**by** (*rule ipurge-fail-aux-t-invariance-3*)  
**moreover have** *ipurge-fail-aux-t-form* (*ipurge-fail-aux-t-aux* ?Y)  
**by** (*rule ipurge-fail-aux-t-form-aux*)  
**ultimately have** *secure* P I D  $\longrightarrow$  (xs, X) ∈ *failures* P  $\longrightarrow$   
*ipurge-fail-aux-t-out* (*ipurge-fail-aux-t-aux* ?Y) ∈ *failures* P  
**by** (*rule ipurge-fail-aux-t-intro-3*)  
**moreover have** ?Y = *ipurge-fail-aux-t-in* I D U xs X  
**by** (*simp add: ipurge-fail-aux-t-in-def*)  
**ultimately show** ?thesis  
**by** (*simp add: ipurge-fail-aux-t-def*)  
**qed**

**lemma** *ipurge-tr-ref-aux-failures*:

$\llbracket$ *secure* P I D; (xs, X) ∈ *failures* P $\rrbracket \implies$   
 (*ipurge-tr-aux* I D U xs, *ipurge-ref-aux* I D U xs X) ∈ *failures* P  
**proof** (*drule ipurge-fail-aux-t-failures* [**where** U = U], *assumption*,  
*cases ipurge-fail-aux-t* I D U xs X)

**qed** (*simp add: ipurge-fail-aux-t-eq-tr [where X = X, symmetric]*  
*ipurge-fail-aux-t-eq-ref [symmetric]*)

## 1.4 Additional propaedeutic lemmas

In what follows, additional lemmas required for the demonstration of the target security conservation theorem are proven.

Here below is the proof of some properties of functions *ipurge-tr-aux* and *ipurge-ref-aux*. Particularly, it is shown that in case an event list and its intransitive purge for some set of domains are both traces of a secure process, and the purged list has a future not affected by any purged event, then that future is also a future for the full event list.

**lemma** *ipurge-tr-aux-idem*:

*ipurge-tr-aux I D U (ipurge-tr-aux I D U xs) = ipurge-tr-aux I D U xs*  
**by** (*simp add: ipurge-tr-aux-union [symmetric]*)

**lemma** *ipurge-tr-aux-set*:

*set (ipurge-tr-aux I D U xs) ⊆ set xs*

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

**qed** *blast*

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

**assumes** *A: u ∈ U*

**shows**  $(\forall x \in \text{set } xs. (u, D x) \in I) \longrightarrow \text{ipurge-tr-aux } I D U xs = []$

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

**fix** *x xs*

**assume**  $(\forall x' \in \text{set } xs. (u, D x') \in I) \longrightarrow \text{ipurge-tr-aux } I D U xs = []$

**moreover assume** *B:  $\forall x' \in \text{set } (xs @ [x]). (u, D x') \in I$*

**ultimately have** *C: ipurge-tr-aux I D U xs = []*

**by** *simp*

**have**  $(u, D x) \in I$

**using** *B by simp*

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

**by** (*rule sinks-aux-subset*)

**hence**  $u \in \text{sinks-aux } I D U xs$

**using** *A ..*

**ultimately have**  $\exists u \in \text{sinks-aux } I D U xs. (u, D x) \in I ..$

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

**by** *simp*

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

**using** *C by simp*

**qed**

**lemma** *ipurge-tr-aux-del-failures [rule-format]*:

**assumes** *S: secure P I D*

**shows**  $(\forall u \in \text{sinks-aux-less } I D U ys. \forall z \in Z \cup \text{set } zs. (u, D z) \notin I) \longrightarrow$

$(xs @ \text{ipurge-tr-aux } I D U ys @ zs, Z) \in \text{failures } P \longrightarrow$

$xs @ ys \in \text{traces } P \longrightarrow$   
 $(xs @ ys @ zs, Z) \in \text{failures } P$

**proof** (induction  $ys$  arbitrary:  $zs$  rule: *rev-induct, simp, (rule impI)+*)  
**fix**  $y \ ys \ zs$

**assume**

$A: \bigwedge zs. (\forall u \in \text{sinks-aux-less } I D U \ ys. \forall z \in Z \cup \text{set } zs. (u, D z) \notin I) \longrightarrow$   
 $(xs @ \text{ipurge-tr-aux } I D U \ ys @ zs, Z) \in \text{failures } P \longrightarrow$

$xs @ ys \in \text{traces } P \longrightarrow$   
 $(xs @ ys @ zs, Z) \in \text{failures } P$  **and**

$B: \forall u \in \text{sinks-aux-less } I D U \ (ys @ [y]). \forall z \in Z \cup \text{set } zs. (u, D z) \notin I$  **and**

$C: (xs @ \text{ipurge-tr-aux } I D U \ (ys @ [y]) @ zs, Z) \in \text{failures } P$  **and**

$D: xs @ (ys @ [y]) \in \text{traces } P$

**show**  $(xs @ (ys @ [y]) @ zs, Z) \in \text{failures } P$

**proof** (cases  $\exists u \in \text{sinks-aux } I D U \ ys. (u, D y) \in I, \text{simp-all (no-asm)}$ )

**case** *True*

**have**

$(\forall u \in \text{sinks-aux-less } I D U \ ys. \forall z \in Z \cup \text{set } zs. (u, D z) \notin I) \longrightarrow$

$(xs @ \text{ipurge-tr-aux } I D U \ ys @ zs, Z) \in \text{failures } P \longrightarrow$

$xs @ ys \in \text{traces } P \longrightarrow$

$(xs @ ys @ zs, Z) \in \text{failures } P$

**using**  $A$  .

**moreover have**  $\exists u \in U \cup \text{sinks-aux-less } I D U \ ys. (u, D y) \in I$

**using** *True* **by** (*simp add: sinks-aux-sinks-aux-less*)

**hence**  $E: \forall u \in \text{insert } (D y) \ (\text{sinks-aux-less } I D U \ ys). \forall z \in Z \cup \text{set } zs.$

$(u, D z) \notin I$

**using**  $B$  **by** (*simp only: sinks-aux-less.simps if-True*)

**hence**  $\forall u \in \text{sinks-aux-less } I D U \ ys. \forall z \in Z \cup \text{set } zs. (u, D z) \notin I$

**by** *simp*

**moreover have**  $(xs @ \text{ipurge-tr-aux } I D U \ ys @ zs, Z) \in \text{failures } P$

**using**  $C$  **and** *True* **by** *simp*

**moreover have**  $(xs @ ys) @ [y] \in \text{traces } P$

**using**  $D$  **by** *simp*

**hence**  $xs @ ys \in \text{traces } P$

**by** (*rule process-rule-2-traces*)

**ultimately have**  $(xs @ ys @ zs, Z) \in \text{failures } P$

**by** *simp*

**hence**  $(zs, Z) \in \text{futures } P \ (xs @ ys)$

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

**moreover have**  $(xs @ ys @ [y], \{\}) \in \text{failures } P$

**using**  $D$  **by** (*rule traces-failures*)

**hence**  $([y], \{\}) \in \text{futures } P \ (xs @ ys)$

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

**ultimately have**  $(y \# \text{ipurge-tr } I D \ (D y) \ zs, \text{ipurge-ref } I D \ (D y) \ zs \ Z)$

$\in \text{futures } P \ (xs @ ys)$

**using**  $S$  **by** (*simp add: secure-def*)

**moreover have**  $\text{ipurge-tr } I D \ (D y) \ zs = zs$

**by** (*subst ipurge-tr-all, simp add: E*)

**moreover have**  $\text{ipurge-ref } I D \ (D y) \ zs \ Z = Z$

**by** (*rule ipurge-ref-all, simp add: E*)

**ultimately have**  $(y \# zs, Z) \in \text{futures } P (xs @ ys)$   
**by** *simp*  
**thus**  $(xs @ ys @ y \# zs, Z) \in \text{failures } P$   
**by** (*simp add: futures-def*)  
**next**  
**case** *False*  
**have** *E*:  
 $(\forall u \in \text{sinks-aux-less } I D U ys. \forall z \in Z \cup \text{set } (y \# zs). (u, D z) \notin I) \longrightarrow$   
 $(xs @ \text{ipurge-tr-aux } I D U ys @ (y \# zs), Z) \in \text{failures } P \longrightarrow$   
 $xs @ ys \in \text{traces } P \longrightarrow$   
 $(xs @ ys @ (y \# zs), Z) \in \text{failures } P$   
**using** *A* .  
**have** *F*:  $\neg (\exists u \in U \cup \text{sinks-aux-less } I D U ys. (u, D y) \in I)$   
**using** *False* **by** (*simp add: sinks-aux-sinks-aux-less*)  
**hence**  $\forall u \in \text{sinks-aux-less } I D U ys. \forall z \in Z \cup \text{set } zs. (u, D z) \notin I$   
**using** *B* **by** (*simp only: sinks-aux-less.simps if-False*)  
**moreover have**  $\forall u \in \text{sinks-aux-less } I D U ys. (u, D y) \notin I$   
**using** *F* **by** *simp*  
**ultimately have**  
 $\forall u \in \text{sinks-aux-less } I D U ys. \forall z \in Z \cup \text{set } (y \# zs). (u, D z) \notin I$   
**by** *simp*  
**with** *E* **have**  
 $(xs @ \text{ipurge-tr-aux } I D U ys @ (y \# zs), Z) \in \text{failures } P \longrightarrow$   
 $xs @ ys \in \text{traces } P \longrightarrow$   
 $(xs @ ys @ (y \# zs), Z) \in \text{failures } P ..$   
**moreover have**  $(xs @ \text{ipurge-tr-aux } I D U ys @ (y \# zs), Z) \in \text{failures } P$   
**using** *C* **and** *False* **by** *simp*  
**moreover have**  $(xs @ ys) @ [y] \in \text{traces } P$   
**using** *D* **by** *simp*  
**hence**  $xs @ ys \in \text{traces } P$   
**by** (*rule process-rule-2-traces*)  
**ultimately show**  $(xs @ ys @ (y \# zs), Z) \in \text{failures } P$   
**by** *simp*  
**qed**  
**qed**

**lemma** *ipurge-ref-aux-append*:  
 $\text{ipurge-ref-aux } I D U (xs @ ys) X = \text{ipurge-ref-aux } I D (\text{sinks-aux } I D U xs) ys X$   
**by** (*simp add: ipurge-ref-aux-def sinks-aux-append*)

**lemma** *ipurge-ref-aux-empty [rule-format]*:  
**assumes**  
 $A: u \in \text{sinks-aux } I D U xs$  **and**  
 $B: \forall x \in X. (u, D x) \in I$   
**shows**  $\text{ipurge-ref-aux } I D U xs X = \{\}$   
**proof** (*rule equals0I, simp add: ipurge-ref-aux-def, erule conjE*)  
**fix** *x*  
**assume**  $x \in X$   
**with** *B* **have**  $(u, D x) \in I ..$



**moreover assume**  $\forall u \in \text{sinks-aux } I D U xs. (u, D x) \notin I$   
**hence**  $(u, D x) \notin I$   
**using**  $A ..$   
**ultimately show**  $False$   
**by contradiction**  
**qed**

Here below is the proof of some properties of functions *sinks*, *ipurge-tr*, and *ipurge-ref*. Particularly, using the previous analogous result on function *ipurge-tr-aux*, it is shown that in case an event list and its intransitive purge for some domain are both traces of a secure process, and the purged list has a future not affected by any purged event, then that future is also a future for the full event list.

**lemma** *sinks-idem*:  
 $\text{sinks } I D u (\text{ipurge-tr } I D u xs) = \{\}$   
**by** (*induction xs rule: rev-induct, simp-all*)

**lemma** *sinks-elem* [*rule-format*]:  
 $v \in \text{sinks } I D u xs \longrightarrow (\exists x \in \text{set } xs. v = D x)$   
**by** (*induction xs rule: rev-induct, simp-all*)

**lemma** *ipurge-tr-append*:  
 $\text{ipurge-tr } I D u (xs @ ys) =$   
 $\text{ipurge-tr } I D u xs @ \text{ipurge-tr-aux } I D (\text{insert } u (\text{sinks } I D u xs)) ys$   
**proof** (*simp add: sinks-aux-single-dom [symmetric]*  
*ipurge-tr-aux-single-dom [symmetric]*)  
**qed** (*simp add: ipurge-tr-aux-append*)

**lemma** *ipurge-tr-idem*:  
 $\text{ipurge-tr } I D u (\text{ipurge-tr } I D u xs) = \text{ipurge-tr } I D u xs$   
**by** (*simp add: ipurge-tr-aux-single-dom [symmetric] ipurge-tr-aux-idem*)

**lemma** *ipurge-tr-set*:  
 $\text{set } (\text{ipurge-tr } I D u xs) \subseteq \text{set } xs$   
**by** (*simp add: ipurge-tr-aux-single-dom [symmetric] ipurge-tr-aux-set*)

**lemma** *ipurge-tr-del-failures* [*rule-format*]:  
**assumes**  
 $S: \text{secure } P I D$  **and**  
 $A: \forall v \in \text{sinks } I D u ys. \forall z \in Z \cup \text{set } zs. (v, D z) \notin I$  **and**  
 $B: (xs @ \text{ipurge-tr } I D u ys @ zs, Z) \in \text{failures } P$  **and**  
 $C: xs @ ys \in \text{traces } P$   
**shows**  $(xs @ ys @ zs, Z) \in \text{failures } P$   
**proof** (*rule ipurge-tr-aux-del-failures [OF S - - C, where U = {u}]*)  
**qed** (*simp add: A sinks-aux-less-single-dom, simp add: B ipurge-tr-aux-single-dom*)

**lemma** *ipurge-tr-del-traces* [rule-format]:

**assumes**

*S*: secure *P I D* **and**

*A*:  $\forall v \in \text{sinks } I D u \text{ ys. } \forall z \in \text{set } zs. (v, D z) \notin I$  **and**

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

*C*:  $xs @ ys \in \text{traces } P$

**shows**  $xs @ ys @ zs \in \text{traces } P$

**proof** (rule *failures-traces* [where  $X = \{\}$ ],

rule *ipurge-tr-del-failures* [OF  $S - - C$ , where  $u = u$ ])

**qed** (simp add: *A*, rule *traces-failures* [OF *B*])

**lemma** *ipurge-ref-append*:

*ipurge-ref* *I D u* ( $xs @ ys$ )  $X =$

*ipurge-ref-aux* *I D* (*insert* *u* (*sinks* *I D u* *xs*)) *ys*  $X$

**proof** (simp add: *sinks-aux-single-dom* [symmetric])

*ipurge-ref-aux-single-dom* [symmetric])

**qed** (simp add: *ipurge-ref-aux-append*)

**lemma** *ipurge-ref-distrib-inter*:

*ipurge-ref* *I D u* *xs* ( $X \cap Y$ ) = *ipurge-ref* *I D u* *xs*  $X \cap \text{ipurge-ref} *I D u* *xs*  $Y$$

**proof** (simp add: *ipurge-ref-def*)

**qed** blast

**lemma** *ipurge-ref-distrib-union*:

*ipurge-ref* *I D u* *xs* ( $X \cup Y$ ) = *ipurge-ref* *I D u* *xs*  $X \cup \text{ipurge-ref} *I D u* *xs*  $Y$$

**proof** (simp add: *ipurge-ref-def*)

**qed** blast

**lemma** *ipurge-ref-subset*:

*ipurge-ref* *I D u* *xs*  $X \subseteq X$

**by** (subst *ipurge-ref-def*, rule *subsetI*, simp)

**lemma** *ipurge-ref-subset-union*:

*ipurge-ref* *I D u* *xs* ( $X \cup Y$ )  $\subseteq X \cup \text{ipurge-ref} *I D u* *xs*  $Y$$

**proof** (simp add: *ipurge-ref-def*)

**qed** blast

**lemma** *ipurge-ref-subset-insert*:

*ipurge-ref* *I D u* *xs* (*insert* *x*  $X$ )  $\subseteq \text{insert}$  *x* (*ipurge-ref* *I D u* *xs*  $X$ )

**by** (simp only: *insert-def* *ipurge-ref-subset-union*)

**lemma** *ipurge-ref-empty* [rule-format]:

**assumes**

*A*:  $v = u \vee v \in \text{sinks } I D u \text{ xs}$  **and**

*B*:  $\forall x \in X. (v, D x) \in I$

**shows** *ipurge-ref* *I D u* *xs*  $X = \{\}$

**proof** (subst *ipurge-ref-aux-single-dom* [symmetric],

rule *ipurge-ref-aux-empty* [of *v*])

**show**  $v \in \text{sinks-aux } I D \{u\} \text{ xs}$

```

  using A by (simp add: sinks-aux-single-dom)
next
  fix x
  assume x ∈ X
  with B show (v, D x) ∈ I ..
qed

```

Finally, in what follows, properties *process-prop-1*, *process-prop-5*, and *process-prop-6* of processes (cf. [8]) are put into the form of introduction rules.

```

lemma process-rule-1:
  ([], {}) ∈ failures P
proof (simp add: failures-def)
  have Rep-process P ∈ process-set (is ?P' ∈ -)
  by (rule Rep-process)
  thus ( [], {} ) ∈ fst ?P'
  by (simp add: process-set-def process-prop-1-def)
qed

```

```

lemma process-rule-5 [rule-format]:
  xs ∈ divergences P → xs @ [x] ∈ divergences P
proof (simp add: divergences-def)
  have Rep-process P ∈ process-set (is ?P' ∈ -)
  by (rule Rep-process)
  hence ∀ xs x. xs ∈ snd ?P' → xs @ [x] ∈ snd ?P'
  by (simp add: process-set-def process-prop-5-def)
  thus xs ∈ snd ?P' → xs @ [x] ∈ snd ?P'
  by blast
qed

```

```

lemma process-rule-6 [rule-format]:
  xs ∈ divergences P → (xs, X) ∈ failures P
proof (simp add: failures-def divergences-def)
  have Rep-process P ∈ process-set (is ?P' ∈ -)
  by (rule Rep-process)
  hence ∀ xs X. xs ∈ snd ?P' → (xs, X) ∈ fst ?P'
  by (simp add: process-set-def process-prop-6-def)
  thus xs ∈ snd ?P' → (xs, X) ∈ fst ?P'
  by blast
qed

```

end

## 2 Sequential composition and noninterference security

theory *SequentialComposition*

```
imports Propaedeutics
begin
```

This section formalizes the definitions of sequential processes and sequential composition given in [1], and then proves that under the assumptions discussed above, noninterference security is conserved under sequential composition for any pair of processes sharing an alphabet that contains successful termination. Finally, this result is generalized to an arbitrary list of processes.

## 2.1 Sequential processes

In [1], a *sequential process* is defined as a process whose alphabet contains successful termination. Since sequential composition applies to sequential processes, the first problem put by the formalization of this operation is that of finding a suitable way to represent such a process.

A simple but effective strategy is to identify it with a process having alphabet *'a option*, where *'a* is the native type of its ordinary (i.e. distinct from termination) events. Then, ordinary events will be those matching pattern *Some -*, whereas successful termination will be denoted by the special event *None*. This means that the *sentences* of a sequential process, defined in [1] as the traces after which the process can terminate successfully, will be nothing but the event lists *xs* such that *xs @ [None]* is a trace (which implies that *xs* is a trace as well).

Once a suitable representation of successful termination has been found, the next step is to formalize the properties of sequential processes related to this event, expressing them in terms of the selected representation. The first of the resulting predicates, *weakly-sequential*, is the minimum required for allowing the identification of event *None* with successful termination, namely that *None* may occur in a trace as its last event only. The second predicate, *sequential*, following what Hoare does in [1], extends the first predicate with an additional requirement, namely that whenever the process can engage in event *None*, it cannot engage in any other event. A simple counterexample shows that this requirement does not imply the first one: a process whose traces are  $\{\[], [None], [None, None]\}$  satisfies the second requirement, but not the first one.

Moreover, here below is the definition of a further predicate, *secure-termination*, which applies to a security policy rather than to a process, and is satisfied just in case the policy does not allow event *None* to be affected by confidential events, viz. by ordinary events not allowed to affect some event in the alphabet. Interestingly, this property, which will prove to be necessary for the target theorem to hold, is nothing but the CSP counterpart of a condition required for a security type system to enforce termination-sensitive nonin-

interference security of programs, namely that program termination must not depend on confidential data (cf. [5], section 9.2.6).

**definition** *sentences* :: 'a option process  $\Rightarrow$  'a option list set **where**  
*sentences*  $P \equiv \{xs. xs @ [None] \in \text{traces } P\}$

**definition** *weakly-sequential* :: 'a option process  $\Rightarrow$  bool **where**  
*weakly-sequential*  $P \equiv$   
 $\forall xs \in \text{traces } P. None \notin \text{set } (\text{butlast } xs)$

**definition** *sequential* :: 'a option process  $\Rightarrow$  bool **where**  
*sequential*  $P \equiv$   
 $(\forall xs \in \text{traces } P. None \notin \text{set } (\text{butlast } xs)) \wedge$   
 $(\forall xs \in \text{sentences } P. \text{next-events } P \ xs = \{None\})$

**definition** *secure-termination* :: ('d  $\times$  'd) set  $\Rightarrow$  ('a option  $\Rightarrow$  'd)  $\Rightarrow$  bool **where**  
*secure-termination*  $I \ D \equiv$   
 $\forall x. (D \ x, D \ None) \in I \wedge x \neq None \longrightarrow (\forall u \in \text{range } D. (D \ x, u) \in I)$

Here below is the proof of some useful lemmas involving the constants just defined. Particularly, it is proven that process sequentiality is indeed stronger than weak sequentiality, and a sentence of a refusals union closed (cf. [9]), sequential process admits the set of all the ordinary events of the process as a refusal. The use of the latter lemma in the proof of the target security conservation theorem is the reason why the theorem requires to assume that the first of the processes to be composed be refusals union closed (cf. below).

**lemma** *seq-implies-weakly-seq*:  
*sequential*  $P \implies \text{weakly-sequential } P$   
**by** (*simp add: weakly-sequential-def sequential-def*)

**lemma** *weakly-seq-sentences-none*:

**assumes**

*WS*: *weakly-sequential*  $P$  **and**

*A*:  $xs \in \text{sentences } P$

**shows**  $None \notin \text{set } xs$

**proof** –

**have**  $\forall xs \in \text{traces } P. None \notin \text{set } (\text{butlast } xs)$

**using** *WS* **by** (*simp add: weakly-sequential-def*)

**moreover have**  $xs @ [None] \in \text{traces } P$

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

**ultimately have**  $None \notin \text{set } (\text{butlast } (xs @ [None]))$  ..

**thus** *?thesis*

**by** *simp*

**qed**

**lemma** *seq-sentences-none*:

**assumes**

*S*: *sequential P and*

*A*:  $xs \in \text{sentences } P$  **and**

*B*:  $xs @ y \# ys \in \text{traces } P$

**shows**  $y = \text{None}$

**proof** –

**have**  $\forall xs \in \text{sentences } P. \text{next-events } P \ xs = \{\text{None}\}$

**using** *S* **by** (*simp add: sequential-def*)

**hence**  $\text{next-events } P \ xs = \{\text{None}\}$

**using** *A* ..

**moreover have**  $(xs @ [y]) @ ys \in \text{traces } P$

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

**hence**  $xs @ [y] \in \text{traces } P$

**by** (*rule process-rule-2-traces*)

**hence**  $y \in \text{next-events } P \ xs$

**by** (*simp add: next-events-def*)

**ultimately show** *?thesis*

**by** *simp*

**qed**

**lemma** *seq-sentences-ref*:

**assumes**

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

*B*: *sequential P and*

*C*:  $xs \in \text{sentences } P$

**shows**  $(xs, \{x. x \neq \text{None}\}) \in \text{failures } P$

  (*is*  $(-, ?X) \in -$ )

**proof** –

**have**  $(\exists X. X \in \text{singleton-set } ?X) \longrightarrow$

$(\forall X \in \text{singleton-set } ?X. (xs, X) \in \text{failures } P) \longrightarrow$

$(xs, \bigcup X \in \text{singleton-set } ?X. X) \in \text{failures } P$

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

**moreover have**  $\exists x. x \in ?X$

**by** *blast*

**hence**  $\exists X. X \in \text{singleton-set } ?X$

**by** (*simp add: singleton-set-some*)

**ultimately have**  $(\forall X \in \text{singleton-set } ?X. (xs, X) \in \text{failures } P) \longrightarrow$

$(xs, \bigcup X \in \text{singleton-set } ?X. X) \in \text{failures } P$  ..

**moreover have**  $\forall X \in \text{singleton-set } ?X. (xs, X) \in \text{failures } P$

**proof** (*rule ballI, simp add: singleton-set-def del: not-None-eq, erule exE, erule conjE, simp (no-asm-simp)*)

**fix**  $x :: 'a \text{ option}$

**assume**  $D: x \neq \text{None}$

**have**  $xs @ [\text{None}] \in \text{traces } P$

**using** *C* **by** (*simp add: sentences-def*)

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

**by** (*rule process-rule-2-traces*)

**hence**  $(xs, \{\}) \in failures\ P$   
**by** (*rule traces-failures*)  
**hence**  $(xs @ [x], \{\}) \in failures\ P \vee (xs, \{x\}) \in failures\ P$   
**by** (*rule process-rule-4*)  
**thus**  $(xs, \{x\}) \in failures\ P$   
**proof** (*rule disjE, rule-tac ccontr, simp-all*)  
**assume**  $(xs @ [x], \{\}) \in failures\ P$   
**hence**  $xs @ [x] \in traces\ P$   
**by** (*rule failures-traces*)  
**with B and C have**  $x = None$   
**by** (*rule seq-sentences-none*)  
**thus** *False*  
**using D by** *contradiction*  
**qed**  
**qed**  
**ultimately have**  $(xs, \bigcup X \in singleton-set\ ?X. X) \in failures\ P ..$   
**thus** *?thesis*  
**by** (*simp only: singleton-set-union*)  
**qed**

## 2.2 Sequential composition

In what follows, the definition of the failures resulting from the sequential composition of two processes  $P, Q$  given in [1] is formalized as the inductive definition of set  $seq-comp-failures\ P\ Q$ . Then, the sequential composition of  $P$  and  $Q$ , denoted by means of notation  $P ; Q$  following [1], is defined as the process having  $seq-comp-failures\ P\ Q$  as failures set and the empty set as divergences set.

For the sake of generality, this definition is based on the mere implicit assumption that the input processes be weakly sequential, rather than sequential. This slightly complicates things, since the sentences of process  $P$  may number further events in addition to *None* in their future.

Therefore, the resulting refusals of a sentence  $xs$  of  $P$  will have the form *insert None*  $X \cap Y$ , where  $X$  is a refusal of  $xs$  in  $P$  and  $Y$  is an initial refusal of  $Q$  (cf. rule *SCF-R2*). In fact, after  $xs$ , process  $P ; Q$  must be able to refuse *None* if  $Q$  is, whereas it cannot refuse an ordinary event unless both  $P$  and  $Q$ , in their respective states, can.

Moreover, a trace  $xs$  of  $P ; Q$  may result from different combinations of a sentence of  $P$  with a trace of  $Q$ . Thus, in order that the refusals of  $P ; Q$  be closed under set union, the union of any two refusals of  $xs$  must still be a refusal (cf. rule *SCF-R4*). Indeed, this property will prove to be sufficient to ensure that for any two processes whose refusals are closed under set union, their sequential composition still be such, which is what is expected for any process of practical significance (cf. [9]).

According to the definition given in [1], a divergence of  $P ; Q$  is either a di-

vergence of  $P$ , or the concatenation of a sentence of  $P$  with a divergence of  $Q$ . Apparently, this definition does not match the formal one stated here below, which identifies the divergences set of  $P ; Q$  with the empty set. Nonetheless, as remarked above, sequential composition does not make sense unless the input processes are weakly sequential, since this is the minimum required to confer the meaning of successful termination on the corresponding alphabet symbol. But a weakly sequential process cannot have any divergence, so that the two definitions are actually equivalent. In fact, a divergence is a trace such that, however it is extended with arbitrary additional events, the resulting event list is still a trace (cf. process properties *process-prop-5* and *process-prop-6* in [8]). Therefore, if  $xs$  were a divergence, then  $xs @ [None, None]$  would be a trace, which is impossible in case the process satisfies predicate *weakly-sequential*.

**inductive-set** *seq-comp-failures* ::

'a option process  $\Rightarrow$  'a option process  $\Rightarrow$  'a option failure set  
**for**  $P$  :: 'a option process **and**  $Q$  :: 'a option process **where**

*SCF-R1*:  $\llbracket xs \notin \text{sentences } P; (xs, X) \in \text{failures } P; None \notin \text{set } xs \rrbracket \Longrightarrow$   
 $(xs, X) \in \text{seq-comp-failures } P \ Q \mid$

*SCF-R2*:  $\llbracket xs \in \text{sentences } P; (xs, X) \in \text{failures } P; ([], Y) \in \text{failures } Q \rrbracket \Longrightarrow$   
 $(xs, \text{insert } None \ X \cap Y) \in \text{seq-comp-failures } P \ Q \mid$

*SCF-R3*:  $\llbracket xs \in \text{sentences } P; (ys, Y) \in \text{failures } Q; ys \neq [] \rrbracket \Longrightarrow$   
 $(xs @ ys, Y) \in \text{seq-comp-failures } P \ Q \mid$

*SCF-R4*:  $\llbracket (xs, X) \in \text{seq-comp-failures } P \ Q; (xs, Y) \in \text{seq-comp-failures } P \ Q \rrbracket \Longrightarrow$   
 $(xs, X \cup Y) \in \text{seq-comp-failures } P \ Q$

**definition** *seq-comp* ::

'a option process  $\Rightarrow$  'a option process  $\Rightarrow$  'a option process (**infixl** ; 60)

**where**

$P ; Q \equiv \text{Abs-process } (\text{seq-comp-failures } P \ Q, \{\})$

Here below is the proof that, for any two processes  $P, Q$  defined over the same alphabet containing successful termination, set *seq-comp-failures*  $P \ Q$  indeed enjoys the characteristic properties of the failures set of a process as defined in [8] provided that  $P$  is weakly sequential, which is what happens in any meaningful case.

**lemma** *seq-comp-prop-1*:

$([], \{\}) \in \text{seq-comp-failures } P \ Q$

**proof** (cases  $[] \in \text{sentences } P$ )

**case** *False*



**moreover have**  $([], \{\}) \in \text{failures } P$   
**by** *(rule process-rule-1)*  
**moreover have**  $\text{None} \notin \text{set } []$   
**by** *simp*  
**ultimately show** *?thesis*  
**by** *(rule SCF-R1)*

**next**

**case** *True*  
**moreover have**  $([], \{\}) \in \text{failures } P$   
**by** *(rule process-rule-1)*  
**moreover have**  $([], \{\}) \in \text{failures } Q$   
**by** *(rule process-rule-1)*  
**ultimately have**  $([], \{\text{None}\} \cap \{\}) \in \text{seq-comp-failures } P Q$   
**by** *(rule SCF-R2)*  
**thus** *?thesis* **by** *simp*

**qed**

**lemma** *seq-comp-prop-2-aux* [*rule-format*]:

**assumes** *WS: weakly-sequential P*  
**shows**  $(ws, X) \in \text{seq-comp-failures } P Q \implies$   
 $ws = xs @ [x] \longrightarrow (xs, \{\}) \in \text{seq-comp-failures } P Q$

**proof** (*erule seq-comp-failures.induct, rule-tac [!] impI, simp-all, erule conjE*)

**fix**  $X'$

**assume**

$A: (xs @ [x], X') \in \text{failures } P$  **and**  
 $B: \text{None} \notin \text{set } xs$

**have**  $A': (xs, \{\}) \in \text{failures } P$   
**using**  $A$  **by** *(rule process-rule-2)*

**show**  $(xs, \{\}) \in \text{seq-comp-failures } P Q$   
**proof** (*cases xs ∈ sentences P*)

**case** *False*  
**thus** *?thesis*  
**using**  $A'$  **and**  $B$  **by** *(rule SCF-R1)*

**next**

**case** *True*  
**have**  $([], \{\}) \in \text{failures } Q$   
**by** *(rule process-rule-1)*  
**with** *True* **and**  $A'$  **have**  $(xs, \{\text{None}\} \cap \{\}) \in \text{seq-comp-failures } P Q$   
**by** *(rule SCF-R2)*  
**thus** *?thesis* **by** *simp*

**qed**

**next**

**fix**  $X'$

**assume**  $A: (xs @ [x], X') \in \text{failures } P$   
**hence**  $A': (xs, \{\}) \in \text{failures } P$   
**by** *(rule process-rule-2)*

**show**  $(xs, \{\}) \in \text{seq-comp-failures } P Q$   
**proof** (*cases xs ∈ sentences P*)

**case** *False*

**have**  $\forall xs \in \text{traces } P. \text{None} \notin \text{set} (\text{butlast } xs)$   
**using** *WS* **by** (*simp add: weakly-sequential-def*)  
**moreover have**  $xs @ [x] \in \text{traces } P$   
**using** *A* **by** (*rule failures-traces*)  
**ultimately have**  $\text{None} \notin \text{set} (\text{butlast } (xs @ [x])) \dots$   
**hence**  $\text{None} \notin \text{set } xs$  **by** *simp*  
**with** *False* **and** *A'* **show** *?thesis*  
**by** (*rule SCF-R1*)  
**next**  
**case** *True*  
**have**  $([], \{\}) \in \text{failures } Q$   
**by** (*rule process-rule-1*)  
**with** *True* **and** *A'* **have**  $(xs, \{\text{None}\} \cap \{\}) \in \text{seq-comp-failures } P Q$   
**by** (*rule SCF-R2*)  
**thus** *?thesis* **by** *simp*  
**qed**  
**next**  
**fix**  $xs' \ ys \ Y$   
**assume**  
*A*:  $xs' @ ys = xs @ [x]$  **and**  
*B*:  $xs' \in \text{sentences } P$  **and**  
*C*:  $(ys, Y) \in \text{failures } Q$  **and**  
*D*:  $ys \neq []$   
**have**  $\exists y \ ys'. \ ys = ys' @ [y]$   
**using** *D* **by** (*rule-tac xs = ys in rev-cases, simp-all*)  
**then obtain**  $y$  **and**  $ys'$  **where**  $D': \ ys = ys' @ [y]$   
**by** *blast*  
**hence**  $xs = xs' @ ys'$   
**using** *A* **by** *simp*  
**thus**  $(xs, \{\}) \in \text{seq-comp-failures } P Q$   
**proof** (*cases ys' = [], simp-all*)  
**case** *True*  
**have**  $xs' @ [\text{None}] \in \text{traces } P$   
**using** *B* **by** (*simp add: sentences-def*)  
**hence**  $xs' \in \text{traces } P$   
**by** (*rule process-rule-2-traces*)  
**hence**  $(xs', \{\}) \in \text{failures } P$   
**by** (*rule traces-failures*)  
**moreover have**  $([], \{\}) \in \text{failures } Q$   
**by** (*rule process-rule-1*)  
**ultimately have**  $(xs', \{\text{None}\} \cap \{\}) \in \text{seq-comp-failures } P Q$   
**by** (*rule SCF-R2 [OF B]*)  
**thus**  $(xs', \{\}) \in \text{seq-comp-failures } P Q$   
**by** *simp*  
**next**  
**case** *False*  
**have**  $(ys' @ [y], Y) \in \text{failures } Q$   
**using** *C* **and** *D'* **by** *simp*  
**hence**  $C': (ys', \{\}) \in \text{failures } Q$

```

    by (rule process-rule-2)
    with B show  $(xs' @ ys', \{\}) \in seq\text{-comp-failures } P Q$ 
    using False by (rule SCF-R3)
qed
qed

lemma seq-comp-prop-2:
  assumes WS: weakly-sequential P
  shows  $(xs @ [x], X) \in seq\text{-comp-failures } P Q \implies$ 
     $(xs, \{\}) \in seq\text{-comp-failures } P Q$ 
  by (erule seq-comp-prop-2-aux [OF WS], simp)

lemma seq-comp-prop-3 [rule-format]:
   $(xs, Y) \in seq\text{-comp-failures } P Q \implies X \subseteq Y \longrightarrow$ 
   $(xs, X) \in seq\text{-comp-failures } P Q$ 
proof (induction arbitrary: X rule: seq-comp-failures.induct, rule-tac [!] impI)
  fix xs X Y
  assume
    A:  $xs \notin sentences P$  and
    B:  $(xs, X) \in failures P$  and
    C:  $None \notin set xs$  and
    D:  $Y \subseteq X$ 
  have  $(xs, Y) \in failures P$ 
  using B and D by (rule process-rule-3)
  with A show  $(xs, Y) \in seq\text{-comp-failures } P Q$ 
  using C by (rule SCF-R1)
next
  fix xs X Y Z
  assume
    A:  $xs \in sentences P$  and
    B:  $(xs, X) \in failures P$  and
    C:  $([], Y) \in failures Q$  and
    D:  $Z \subseteq insert None X \cap Y$ 
  have  $Z - \{None\} \subseteq X$ 
  using D by blast
  with B have  $(xs, Z - \{None\}) \in failures P$ 
  by (rule process-rule-3)
  moreover have  $Z \subseteq Y$ 
  using D by simp
  with C have  $([], Z) \in failures Q$ 
  by (rule process-rule-3)
  ultimately have  $(xs, insert None (Z - \{None\}) \cap Z) \in seq\text{-comp-failures } P Q$ 
  by (rule SCF-R2 [OF A])
  moreover have  $insert None (Z - \{None\}) \cap Z = Z$ 
  by blast
  ultimately show  $(xs, Z) \in seq\text{-comp-failures } P Q$ 
  by simp
next
  fix xs ys X Y

```

**assume**  
 $A: xs \in \text{sentences } P$  **and**  
 $B: (ys, Y) \in \text{failures } Q$  **and**  
 $C: ys \neq []$  **and**  
 $D: X \subseteq Y$   
**have**  $(ys, X) \in \text{failures } Q$   
**using**  $B$  **and**  $D$  **by** (rule process-rule-3)  
**with**  $A$  **show**  $(xs @ ys, X) \in \text{seq-comp-failures } P Q$   
**using**  $C$  **by** (rule SCF-R3)  
**next**  
**fix**  $xs X Y Z$   
**assume**  
 $A: \bigwedge W. W \subseteq X \longrightarrow (xs, W) \in \text{seq-comp-failures } P Q$  **and**  
 $B: \bigwedge W. W \subseteq Y \longrightarrow (xs, W) \in \text{seq-comp-failures } P Q$  **and**  
 $C: Z \subseteq X \cup Y$   
**have**  $Z \cap X \subseteq X \longrightarrow (xs, Z \cap X) \in \text{seq-comp-failures } P Q$   
**using**  $A$  .  
**hence**  $(xs, Z \cap X) \in \text{seq-comp-failures } P Q$   
**by** *simp*  
**moreover have**  $Z \cap Y \subseteq Y \longrightarrow (xs, Z \cap Y) \in \text{seq-comp-failures } P Q$   
**using**  $B$  .  
**hence**  $(xs, Z \cap Y) \in \text{seq-comp-failures } P Q$   
**by** *simp*  
**ultimately have**  $(xs, Z \cap X \cup Z \cap Y) \in \text{seq-comp-failures } P Q$   
**by** (rule SCF-R4)  
**hence**  $(xs, Z \cap (X \cup Y)) \in \text{seq-comp-failures } P Q$   
**by** (*simp add: Int-Un-distrib*)  
**moreover have**  $Z \cap (X \cup Y) = Z$   
**using**  $C$  **by** (rule Int-absorb2)  
**ultimately show**  $(xs, Z) \in \text{seq-comp-failures } P Q$   
**by** *simp*  
**qed**

**lemma** *seq-comp-prop-4*:

**assumes**  $WS: \text{weakly-sequential } P$   
**shows**  $(xs, X) \in \text{seq-comp-failures } P Q \implies$   
 $(xs @ [x], \{\}) \in \text{seq-comp-failures } P Q \vee$   
 $(xs, \text{insert } x X) \in \text{seq-comp-failures } P Q$   
**proof** (erule *seq-comp-failures.induct, simp-all*)  
**fix**  $xs X$   
**assume**  
 $A: xs \notin \text{sentences } P$  **and**  
 $B: (xs, X) \in \text{failures } P$  **and**  
 $C: \text{None} \notin \text{set } xs$   
**have**  $(xs @ [x], \{\}) \in \text{failures } P \vee$   
 $(xs, \text{insert } x X) \in \text{failures } P$   
**using**  $B$  **by** (rule process-rule-4)  
**thus**  $(xs @ [x], \{\}) \in \text{seq-comp-failures } P Q \vee$   
 $(xs, \text{insert } x X) \in \text{seq-comp-failures } P Q$

```

proof
  assume  $D: (xs @ [x], \{\}) \in failures P$ 
  show ?thesis
  proof (cases  $xs @ [x] \in sentences P$ )
    case False
      have  $None \notin set (xs @ [x])$ 
      proof (simp add: C, rule notI)
        assume  $None = x$ 
        hence  $(xs @ [None], \{\}) \in failures P$ 
        using  $D$  by simp
        hence  $xs @ [None] \in traces P$ 
        by (rule failures-traces)
        hence  $xs \in sentences P$ 
        by (simp add: sentences-def)
        thus False
        using  $A$  by contradiction
      qed
    with False and  $D$  have  $(xs @ [x], \{\}) \in seq-comp-failures P Q$ 
    by (rule SCF-R1)
    thus ?thesis ..
  next
    case True
      have  $([], \{\}) \in failures Q$ 
      by (rule process-rule-1)
      with True and  $D$  have  $(xs @ [x], \{None\} \cap \{\}) \in seq-comp-failures P Q$ 
      by (rule SCF-R2)
      thus ?thesis by simp
    qed
  next
    assume  $(xs, insert x X) \in failures P$ 
    with  $A$  have  $(xs, insert x X) \in seq-comp-failures P Q$ 
    using  $C$  by (rule SCF-R1)
    thus ?thesis ..
  qed
next
  fix  $xs X Y$ 
  assume
     $A: xs \in sentences P$  and
     $B: (xs, X) \in failures P$  and
     $C: ([], Y) \in failures Q$ 
  show  $(xs @ [x], \{\}) \in seq-comp-failures P Q \vee$ 
     $(xs, insert x (insert None X \cap Y)) \in seq-comp-failures P Q$ 
  proof (cases  $x = None, simp$ )
    case True
      have  $([] @ [None], \{\}) \in failures Q \vee ( [], insert None Y) \in failures Q$ 
      using  $C$  by (rule process-rule-4)
      thus  $(xs @ [None], \{\}) \in seq-comp-failures P Q \vee$ 
         $(xs, insert None (insert None X \cap Y)) \in seq-comp-failures P Q$ 
      proof (rule disjE, simp)

```

**assume**  $([None], \{\}) \in failures\ Q$   
**moreover have**  $[None] \neq []$   
**by** *simp*  
**ultimately have**  $(xs @ [None], \{\}) \in seq-comp-failures\ P\ Q$   
**by** (rule *SCF-R3* [*OF A*])  
**thus** ?thesis ..  
**next**  
**assume**  $([], insert\ None\ Y) \in failures\ Q$   
**with** *A* **and** *B* **have**  $(xs, insert\ None\ X \cap insert\ None\ Y)$   
 $\in seq-comp-failures\ P\ Q$   
**by** (rule *SCF-R2*)  
**moreover have**  $insert\ None\ X \cap insert\ None\ Y =$   
 $insert\ None\ (insert\ None\ X \cap Y)$   
**by** *blast*  
**ultimately have**  $(xs, insert\ None\ (insert\ None\ X \cap Y))$   
 $\in seq-comp-failures\ P\ Q$   
**by** *simp*  
**thus** ?thesis ..  
**qed**  
**next**  
**case** *False*  
**have**  $(xs @ [x], \{\}) \in failures\ P \vee (xs, insert\ x\ X) \in failures\ P$   
**using** *B* **by** (rule *process-rule-4*)  
**thus** ?thesis  
**proof** (rule *disjE*, cases  $xs @ [x] \in sentences\ P$ )  
**assume**  
 $D: xs @ [x] \notin sentences\ P$  **and**  
 $E: (xs @ [x], \{\}) \in failures\ P$   
**have**  $None \notin set\ xs$   
**using** *WS* **and** *A* **by** (rule *weakly-seq-sentences-none*)  
**hence**  $None \notin set\ (xs @ [x])$   
**using** *False* **by** (*simp del: not-None-eq*)  
**with** *D* **and** *E* **have**  $(xs @ [x], \{\}) \in seq-comp-failures\ P\ Q$   
**by** (rule *SCF-R1*)  
**thus** ?thesis ..  
**next**  
**assume**  
 $xs @ [x] \in sentences\ P$  **and**  
 $(xs @ [x], \{\}) \in failures\ P$   
**moreover have**  $([], \{\}) \in failures\ Q$   
**by** (rule *process-rule-1*)  
**ultimately have**  $(xs @ [x], \{None\} \cap \{\}) \in seq-comp-failures\ P\ Q$   
**by** (rule *SCF-R2*)  
**thus** ?thesis **by** *simp*  
**next**  
**assume** *D*:  $(xs, insert\ x\ X) \in failures\ P$   
**have**  $([] @ [x], \{\}) \in failures\ Q \vee ([], insert\ x\ Y) \in failures\ Q$   
**using** *C* **by** (rule *process-rule-4*)  
**thus** ?thesis

```

proof (rule disjE, simp)
  assume ( $[x], \{\}$ )  $\in$  failures  $Q$ 
  moreover have  $[x] \neq []$ 
  by simp
  ultimately have ( $xs @ [x], \{\}$ )  $\in$  seq-comp-failures  $P Q$ 
  by (rule SCF-R3 [OF A])
  thus ?thesis ..
next
  assume ( $[], insert\ x\ Y$ )  $\in$  failures  $Q$ 
  with  $A$  and  $D$  have ( $xs, insert\ None\ (insert\ x\ X) \cap insert\ x\ Y$ )
     $\in$  seq-comp-failures  $P Q$ 
  by (rule SCF-R2)
  moreover have  $insert\ None\ (insert\ x\ X) \cap insert\ x\ Y =$ 
     $insert\ x\ (insert\ None\ X \cap Y)$ 
  by blast
  ultimately have ( $xs, insert\ x\ (insert\ None\ X \cap Y)$ )
     $\in$  seq-comp-failures  $P Q$ 
  by simp
  thus ?thesis ..
qed
qed
qed
next
fix  $xs\ ys\ Y$ 
assume
   $A: xs \in$  sentences  $P$  and
   $B: (ys, Y) \in$  failures  $Q$  and
   $C: ys \neq []$ 
have ( $ys @ [x], \{\}$ )  $\in$  failures  $Q \vee (ys, insert\ x\ Y) \in$  failures  $Q$ 
using  $B$  by (rule process-rule-4)
thus ( $xs @ ys @ [x], \{\}$ )  $\in$  seq-comp-failures  $P Q \vee$ 
  ( $xs @ ys, insert\ x\ Y$ )  $\in$  seq-comp-failures  $P Q$ 
proof
  assume ( $ys @ [x], \{\}$ )  $\in$  failures  $Q$ 
  moreover have  $ys @ [x] \neq []$ 
  by simp
  ultimately have ( $xs @ ys @ [x], \{\}$ )  $\in$  seq-comp-failures  $P Q$ 
  by (rule SCF-R3 [OF A])
  thus ?thesis ..
next
  assume ( $ys, insert\ x\ Y$ )  $\in$  failures  $Q$ 
  with  $A$  have ( $xs @ ys, insert\ x\ Y$ )  $\in$  seq-comp-failures  $P Q$ 
  using  $C$  by (rule SCF-R3)
  thus ?thesis ..
qed
next
fix  $xs\ X\ Y$ 
assume
  ( $xs @ [x], \{\}$ )  $\in$  seq-comp-failures  $P Q \vee$ 

```

```

      (xs, insert x X) ∈ seq-comp-failures P Q and
      (xs @ [x], {}) ∈ seq-comp-failures P Q ∨
      (xs, insert x Y) ∈ seq-comp-failures P Q
thus (xs @ [x], {}) ∈ seq-comp-failures P Q ∨
      (xs, insert x (X ∪ Y)) ∈ seq-comp-failures P Q
proof (cases (xs @ [x], {}) ∈ seq-comp-failures P Q, simp-all)
  assume
    (xs, insert x X) ∈ seq-comp-failures P Q and
    (xs, insert x Y) ∈ seq-comp-failures P Q
  hence (xs, insert x X ∪ insert x Y) ∈ seq-comp-failures P Q
  by (rule SCF-R4)
  thus (xs, insert x (X ∪ Y)) ∈ seq-comp-failures P Q
  by simp
qed
qed

lemma seq-comp-rep:
  assumes WS: weakly-sequential P
  shows Rep-process (P ; Q) = (seq-comp-failures P Q, {})
proof (subst seq-comp-def, rule Abs-process-inverse, simp add: process-set-def,
  (subst conj-assoc [symmetric])+, (rule conjI)+)
  show process-prop-1 (seq-comp-failures P Q, {})
  proof (simp add: process-prop-1-def)
  qed (rule seq-comp-prop-1)
next
  show process-prop-2 (seq-comp-failures P Q, {})
  proof (simp add: process-prop-2-def del: all-simps, (rule allI)+, rule impI)
  qed (rule seq-comp-prop-2 [OF WS])
next
  show process-prop-3 (seq-comp-failures P Q, {})
  proof (simp add: process-prop-3-def del: all-simps, (rule allI)+, rule impI,
  erule conjE)
  qed (rule seq-comp-prop-3)
next
  show process-prop-4 (seq-comp-failures P Q, {})
  proof (simp add: process-prop-4-def, (rule allI)+, rule impI)
  qed (rule seq-comp-prop-4 [OF WS])
next
  show process-prop-5 (seq-comp-failures P Q, {})
  by (simp add: process-prop-5-def)
next
  show process-prop-6 (seq-comp-failures P Q, {})
  by (simp add: process-prop-6-def)
qed

```

Here below, the previous result is applied to derive useful expressions for the outputs of the functions returning the elements of a process, as defined in [8] and [9], when acting on the sequential composition of a pair of processes.



**lemma** *seq-comp-failures*:  
*weakly-sequential*  $P \implies$   
 $\text{failures } (P ; Q) = \text{seq-comp-failures } P Q$   
**by** (*drule* *seq-comp-rep* [**where**  $Q = Q$ ], *simp* *add*: *failures-def*)

**lemma** *seq-comp-divergences*:  
*weakly-sequential*  $P \implies$   
 $\text{divergences } (P ; Q) = \{\}$   
**by** (*drule* *seq-comp-rep* [**where**  $Q = Q$ ], *simp* *add*: *divergences-def*)

**lemma** *seq-comp-futures*:  
*weakly-sequential*  $P \implies$   
 $\text{futures } (P ; Q) \text{ } xs = \{(ys, Y). (xs @ ys, Y) \in \text{seq-comp-failures } P Q\}$   
**by** (*simp* *add*: *futures-def* *seq-comp-failures*)

**lemma** *seq-comp-traces*:  
*weakly-sequential*  $P \implies$   
 $\text{traces } (P ; Q) = \text{Domain } (\text{seq-comp-failures } P Q)$   
**by** (*simp* *add*: *traces-def* *seq-comp-failures*)

**lemma** *seq-comp-refusals*:  
*weakly-sequential*  $P \implies$   
 $\text{refusals } (P ; Q) \text{ } xs \equiv \text{seq-comp-failures } P Q \text{ “ } \{xs\}$   
**by** (*simp* *add*: *refusals-def* *seq-comp-failures*)

**lemma** *seq-comp-next-events*:  
*weakly-sequential*  $P \implies$   
 $\text{next-events } (P ; Q) \text{ } xs = \{x. xs @ [x] \in \text{Domain } (\text{seq-comp-failures } P Q)\}$   
**by** (*simp* *add*: *next-events-def* *seq-comp-traces*)

### 2.3 Conservation of refusals union closure and sequentiality under sequential composition

Here below is the proof that, for any two processes  $P, Q$  and any failure  $(xs, X)$  of  $P ; Q$ , the refusal  $X$  is the union of a set of refusals where, for any such refusal  $W$ ,  $(xs, W)$  is a failure of  $P ; Q$  by virtue of one of rules *SCF-R1*, *SCF-R2*, or *SCF-R3*.

The converse is also proven, under the assumption that the refusals of both  $P$  and  $Q$  be closed under union: namely, for any trace  $xs$  of  $P ; Q$  and any set of refusals where, for any such refusal  $W$ ,  $(xs, W)$  is a failure of the aforesaid kind, the union of these refusals is still a refusal of  $xs$ .

The proof of the latter lemma makes use of the axiom of choice.

**lemma** *seq-comp-refusals-1*:  
 $(xs, X) \in \text{seq-comp-failures } P Q \implies \exists R.$   
 $X = (\bigcup n \in \{..length\ xs\}. \bigcup W \in R \ n. W) \wedge$

$(\forall W \in R \ 0.$   
 $\quad xs \notin \text{sentences } P \wedge \text{None} \notin \text{set } xs \wedge (xs, W) \in \text{failures } P \vee$   
 $\quad xs \in \text{sentences } P \wedge (\exists U \ V. (xs, U) \in \text{failures } P \wedge ([], V) \in \text{failures } Q \wedge$   
 $\quad \quad W = \text{insert None } U \cap V)) \wedge$   
 $(\forall n \in \{0 < .. \text{length } xs\}. \forall W \in R \ n.$   
 $\quad \text{take } (\text{length } xs - n) \ xs \in \text{sentences } P \wedge$   
 $\quad (\text{drop } (\text{length } xs - n) \ xs, W) \in \text{failures } Q) \wedge$   
 $(\exists n \in \{.. \text{length } xs\}. \exists W. W \in R \ n)$   
**(is -  $\implies \exists R. ?T R \ xs \ X$ )**

**proof** (*erule seq-comp-failures.induct, (erule-tac [4] exE)+*)  
**fix**  $xs \ X$   
**assume**  
 $A: xs \notin \text{sentences } P$  **and**  
 $B: (xs, X) \in \text{failures } P$  **and**  
 $C: \text{None} \notin \text{set } xs$   
**show**  $\exists R. ?T R \ xs \ X$   
**proof** (*rule-tac  $x = \lambda n. \text{if } n = 0 \text{ then } \{X\} \text{ else } \{\}$  in exI,*  
*simp add: A B C, rule equalityI, rule-tac [!] subsetI, simp-all*)  
**fix**  $x$   
**assume**  $\exists n \in \{.. \text{length } xs\}.$   
 $\quad \exists W \in \text{if } n = 0 \text{ then } \{X\} \text{ else } \{\}. x \in W$   
**thus**  $x \in X$   
**by** (*simp split: if-split-asm*)  
**qed**

**next**  
**fix**  $xs \ X \ Y$   
**assume**  
 $A: xs \in \text{sentences } P$  **and**  
 $B: (xs, X) \in \text{failures } P$  **and**  
 $C: ([], Y) \in \text{failures } Q$   
**show**  $\exists R. ?T R \ xs \ (\text{insert None } X \cap Y)$   
**proof** (*rule-tac  $x = \lambda n. \text{if } n = 0 \text{ then } \{\text{insert None } X \cap Y\} \text{ else } \{\}$  in exI,*  
*simp add: A, rule conjI, rule equalityI, rule-tac [1-2] subsetI, simp-all*)  
**fix**  $x$   
**assume**  $\exists n \in \{.. \text{length } xs\}.$   
 $\quad \exists W \in \text{if } n = 0 \text{ then } \{\text{insert None } X \cap Y\} \text{ else } \{\}. x \in W$   
**thus**  $(x = \text{None} \vee x \in X) \wedge x \in Y$   
**by** (*simp split: if-split-asm*)  
**next**  
**show**  $\exists U. (xs, U) \in \text{failures } P \wedge (\exists V. ([], V) \in \text{failures } Q \wedge$   
 $\quad \text{insert None } X \cap Y = \text{insert None } U \cap V)$   
**proof** (*rule-tac  $x = X$  in exI, rule conjI, simp add: B*)  
**qed** (*rule-tac  $x = Y$  in exI, rule conjI, simp-all add: C*)  
**qed**

**next**  
**fix**  $xs \ ys \ Y$   
**assume**  
 $A: xs \in \text{sentences } P$  **and**  
 $B: (ys, Y) \in \text{failures } Q$  **and**

```

  C: ys ≠ []
show ∃ R. ?T R (xs @ ys) Y
proof (rule-tac x = λn. if n = length ys then {Y} else {} in exI,
simp add: A B C, rule equalityI, rule-tac [!] subsetI, simp-all)
  fix x
  assume ∃ n ∈ {..length xs + length ys}.
    ∃ W ∈ if n = length ys then {Y} else {}. x ∈ W
  thus x ∈ Y
  by (simp split: if-split-asm)
qed
next
fix xs X Y Rx Ry
assume
  A: ?T Rx xs X and
  B: ?T Ry xs Y
show ∃ R. ?T R xs (X ∪ Y)
proof (rule-tac x = λn. Rx n ∪ Ry n in exI, rule conjI, rule-tac [2] conjI,
rule-tac [3] conjI, rule-tac [2] ballI, (rule-tac [3] ballI)+)
  have X ∪ Y = (∪ n ≤ length xs. ∪ W ∈ Rx n. W) ∪
    (∪ n ≤ length xs. ∪ W ∈ Ry n. W)
  using A and B by simp
  also have ... = (∪ n ≤ length xs. (∪ W ∈ Rx n. W) ∪ (∪ W ∈ Ry n. W))
  by blast
  also have ... = (∪ n ≤ length xs. ∪ W ∈ Rx n ∪ Ry n. W)
  by simp
  finally show X ∪ Y = (∪ n ≤ length xs. ∪ W ∈ Rx n ∪ Ry n. W) .
next
fix W
assume W ∈ Rx 0 ∪ Ry 0
thus
  xs ∉ sentences P ∧ None ∉ set xs ∧ (xs, W) ∈ failures P ∨
  xs ∈ sentences P ∧ (∃ U V. (xs, U) ∈ failures P ∧ ([], V) ∈ failures Q ∧
    W = insert None U ∩ V)
  (is ?T' W)
proof
  have ∀ W ∈ Rx 0. ?T' W
  using A by simp
  moreover assume W ∈ Rx 0
  ultimately show ?thesis ..
next
  have ∀ W ∈ Ry 0. ?T' W
  using B by simp
  moreover assume W ∈ Ry 0
  ultimately show ?thesis ..
qed
next
fix n W
assume C: n ∈ {0 < ..length xs}
assume W ∈ Rx n ∪ Ry n

```

**thus**  
*take*  $(\text{length } xs - n) \text{ } xs \in \text{sentences } P \wedge$   
 $(\text{drop } (\text{length } xs - n) \text{ } xs, W) \in \text{failures } Q$   
*(is*  $?T' n W)$

**proof**  
**have**  $\forall n \in \{0 < .. \text{length } xs\}. \forall W \in Rx n. ?T' n W$   
*using*  $A$  **by** *simp*  
**hence**  $\forall W \in Rx n. ?T' n W$   
*using*  $C$  ..  
**moreover assume**  $W \in Rx n$   
**ultimately show** *?thesis* ..

**next**  
**have**  $\forall n \in \{0 < .. \text{length } xs\}. \forall W \in Ry n. ?T' n W$   
*using*  $B$  **by** *simp*  
**hence**  $\forall W \in Ry n. ?T' n W$   
*using*  $C$  ..  
**moreover assume**  $W \in Ry n$   
**ultimately show** *?thesis* ..

**qed**

**next**  
**have**  $\exists n \in \{.. \text{length } xs\}. \exists W. W \in Rx n$   
*using*  $A$  **by** *simp*  
**then obtain**  $n$  **where**  $C: n \in \{.. \text{length } xs\}$  **and**  $D: \exists W. W \in Rx n$  ..  
**obtain**  $W$  **where**  $W \in Rx n$   
*using*  $D$  ..  
**hence**  $W \in Rx n \cup Ry n$  ..  
**hence**  $\exists W. W \in Rx n \cup Ry n$  ..  
**thus**  $\exists n \in \{.. \text{length } xs\}. \exists W. W \in Rx n \cup Ry n$   
*using*  $C$  ..

**qed**

**qed**

**lemma** *seq-comp-refusals-finite* [*rule-format*]:  
**assumes**  $A: xs \in \text{Domain } (\text{seq-comp-failures } P Q)$   
**shows**  $\text{finite } A \implies (\forall x \in A. (xs, F x) \in \text{seq-comp-failures } P Q) \longrightarrow$   
 $(xs, \bigcup x \in A. F x) \in \text{seq-comp-failures } P Q$

**proof** (*erule finite-induct, simp, rule-tac* [2] *impI*)  
**have**  $\exists X. (xs, X) \in \text{seq-comp-failures } P Q$   
*using*  $A$  **by** (*simp add: Domain-iff*)  
**then obtain**  $X$  **where**  $(xs, X) \in \text{seq-comp-failures } P Q$  ..  
**moreover have**  $\{\} \subseteq X$  ..  
**ultimately show**  $(xs, \{\}) \in \text{seq-comp-failures } P Q$   
*by* (*rule seq-comp-prop-3*)

**next**  
**fix**  $x' A$   
**assume**  $B: \forall x \in \text{insert } x' A. (xs, F x) \in \text{seq-comp-failures } P Q$   
**hence**  $(xs, F x') \in \text{seq-comp-failures } P Q$   
*by* *simp*  
**moreover assume**  $(\forall x \in A. (xs, F x) \in \text{seq-comp-failures } P Q) \longrightarrow$

$(xs, \bigcup x \in A. F x) \in \text{seq-comp-failures } P Q$   
**hence**  $(xs, \bigcup x \in A. F x) \in \text{seq-comp-failures } P Q$   
**using**  $B$  **by** *simp*  
**ultimately have**  $(xs, F x' \cup (\bigcup x \in A. F x)) \in \text{seq-comp-failures } P Q$   
**by** (*rule SCF-R4*)  
**thus**  $(xs, \bigcup x \in \text{insert } x' A. F x) \in \text{seq-comp-failures } P Q$   
**by** *simp*  
**qed**

**lemma** *seq-comp-refusals-2*:

**assumes**

$A$ : *ref-union-closed*  $P$  **and**

$B$ : *ref-union-closed*  $Q$  **and**

$C$ :  $xs \in \text{Domain } (\text{seq-comp-failures } P Q)$  **and**

$D$ :  $X = (\bigcup n \in \{..length\ xs\}. \bigcup W \in R n. W) \wedge$

$(\forall W \in R 0.$

$xs \notin \text{sentences } P \wedge \text{None} \notin \text{set } xs \wedge (xs, W) \in \text{failures } P \vee$

$xs \in \text{sentences } P \wedge (\exists U V. (xs, U) \in \text{failures } P \wedge ([], V) \in \text{failures } Q \wedge$

$W = \text{insert None } U \cap V)) \wedge$

$(\forall n \in \{0 < ..length\ xs\}. \forall W \in R n.$

$\text{take } (length\ xs - n)\ xs \in \text{sentences } P \wedge$

$(\text{drop } (length\ xs - n)\ xs, W) \in \text{failures } Q)$

**shows**  $(xs, X) \in \text{seq-comp-failures } P Q$

**proof** –

**have**  $\exists Y. (xs, Y) \in \text{seq-comp-failures } P Q$

**using**  $C$  **by** (*simp add: Domain-iff*)

**then obtain**  $Y$  **where**  $(xs, Y) \in \text{seq-comp-failures } P Q$  ..

**moreover have**  $\{ \} \subseteq Y$  ..

**ultimately have**  $E: (xs, \{ \}) \in \text{seq-comp-failures } P Q$

**by** (*rule seq-comp-prop-3*)

**have**  $(xs, \bigcup W \in R 0. W) \in \text{seq-comp-failures } P Q$

**proof** (*cases*  $\exists W. W \in R 0$ , *cases*  $xs \in \text{sentences } P$ )

**assume**  $\neg (\exists W. W \in R 0)$

**thus** *?thesis*

**using**  $E$  **by** *simp*

**next**

**assume**

$F: \exists W. W \in R 0$  **and**

$G: xs \notin \text{sentences } P$

**have**  $H: \forall W \in R 0. \text{None} \notin \text{set } xs \wedge (xs, W) \in \text{failures } P$

**using**  $D$  **and**  $G$  **by** *simp*

**show** *?thesis*

**proof** (*rule SCF-R1 [OF G]*)

**have**  $\forall xs A. (\exists W. W \in A) \longrightarrow (\forall W \in A. (xs, W) \in \text{failures } P) \longrightarrow$

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

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

**hence**  $(\exists W. W \in R 0) \longrightarrow (\forall W \in R 0. (xs, W) \in \text{failures } P) \longrightarrow$

$(xs, \bigcup W \in R 0. W) \in \text{failures } P$

**by** *blast*

**thus**  $(xs, \bigcup W \in R \ 0. \ W) \in \text{failures } P$   
**using**  $F$  **and**  $H$  **by** *simp*  
**next**  
**obtain**  $W$  **where**  $W \in R \ 0$  **using**  $F$  ..  
**thus**  $\text{None} \notin \text{set } xs$   
**using**  $H$  **by** *simp*  
**qed**  
**next**  
**assume**  
 $F: \exists W. W \in R \ 0$  **and**  
 $G: xs \in \text{sentences } P$   
**have**  $\forall W \in R \ 0. \exists U \ V. (xs, U) \in \text{failures } P \wedge ([], V) \in \text{failures } Q \wedge$   
 $W = \text{insert None } U \cap V$   
**using**  $D$  **and**  $G$  **by** *simp*  
**hence**  $\exists F. \forall W \in R \ 0. \exists V. (xs, F \ W) \in \text{failures } P \wedge ([], V) \in \text{failures } Q \wedge$   
 $W = \text{insert None } (F \ W) \cap V$   
**by** (*rule bchoice*)  
**then obtain**  $F$  **where**  $\forall W \in R \ 0.$   
 $\exists V. (xs, F \ W) \in \text{failures } P \wedge ([], V) \in \text{failures } Q \wedge$   
 $W = \text{insert None } (F \ W) \cap V$  ..  
**hence**  $\exists G. \forall W \in R \ 0. (xs, F \ W) \in \text{failures } P \wedge ([], G \ W) \in \text{failures } Q \wedge$   
 $W = \text{insert None } (F \ W) \cap G \ W$   
**by** (*rule bchoice*)  
**then obtain**  $G$  **where**  $H: \forall W \in R \ 0.$   
 $(xs, F \ W) \in \text{failures } P \wedge ([], G \ W) \in \text{failures } Q \wedge$   
 $W = \text{insert None } (F \ W) \cap G \ W$  ..  
**have**  $(xs, \text{insert None } (\bigcup W \in R \ 0. \ F \ W) \cap (\bigcup W \in R \ 0. \ G \ W))$   
 $\in \text{seq-comp-failures } P \ Q$   
**(is**  $(-, ?S) \in -$   
**proof** (*rule SCF-R2 [OF G]*)  
**have**  $\forall xs \ A. (\exists X. X \in A) \longrightarrow (\forall X \in A. (xs, X) \in \text{failures } P) \longrightarrow$   
 $(xs, \bigcup X \in A. \ X) \in \text{failures } P$   
**using**  $A$  **by** (*simp add: ref-union-closed-def*)  
**hence**  $(\exists X. X \in F \ ' \ R \ 0) \longrightarrow (\forall X \in F \ ' \ R \ 0. (xs, X) \in \text{failures } P) \longrightarrow$   
 $(xs, \bigcup X \in F \ ' \ R \ 0. \ X) \in \text{failures } P$   
**(is**  $?A \longrightarrow ?B \longrightarrow ?C$ )  
**by** (*erule-tac x = xs in allE, erule-tac x = F \ ' \ R \ 0 in allE*)  
**moreover obtain**  $W$  **where**  $W \in R \ 0$  **using**  $F$  ..  
**hence**  $?A$   
**proof** (*simp add: image-def, rule-tac x = F \ W in exI*)  
**qed** (*rule bexI, simp*)  
**ultimately have**  $?B \longrightarrow ?C$  ..  
**moreover have**  $?B$   
**proof** (*rule ballI, simp add: image-def, erule bexE*)  
**fix**  $W \ X$   
**assume**  $W \in R \ 0$   
**hence**  $(xs, F \ W) \in \text{failures } P$   
**using**  $H$  **by** *simp*  
**moreover assume**  $X = F \ W$

**ultimately show**  $(xs, X) \in failures\ P$   
**by** *simp*  
**qed**  
**ultimately have**  $?C$  ..  
**thus**  $(xs, \bigcup W \in R\ 0. F\ W) \in failures\ P$   
**by** *simp*  
**next**  
**have**  $\forall xs\ A. (\exists Y. Y \in A) \longrightarrow (\forall Y \in A. (xs, Y) \in failures\ Q) \longrightarrow$   
 $(xs, \bigcup Y \in A. Y) \in failures\ Q$   
**using**  $B$  **by** (*simp add: ref-union-closed-def*)  
**hence**  $(\exists Y. Y \in G\ 'R\ 0) \longrightarrow (\forall Y \in G\ 'R\ 0. ([], Y) \in failures\ Q) \longrightarrow$   
 $([], \bigcup Y \in G\ 'R\ 0. Y) \in failures\ Q$   
**(is**  $?A \longrightarrow ?B \longrightarrow ?C)$   
**by** (*erule-tac x = [] in allE, erule-tac x = G\ 'R\ 0 in allE*)  
**moreover obtain**  $W$  **where**  $W \in R\ 0$  **using**  $F$  ..  
**hence**  $?A$   
**proof** (*simp add: image-def, rule-tac x = G\ W in exI*)  
**qed** (*rule bexI, simp*)  
**ultimately have**  $?B \longrightarrow ?C$  ..  
**moreover have**  $?B$   
**proof** (*rule ballI, simp add: image-def, erule bexE*)  
**fix**  $W\ Y$   
**assume**  $W \in R\ 0$   
**hence**  $([], G\ W) \in failures\ Q$   
**using**  $H$  **by** *simp*  
**moreover assume**  $Y = G\ W$   
**ultimately show**  $([], Y) \in failures\ Q$   
**by** *simp*  
**qed**  
**ultimately have**  $?C$  ..  
**thus**  $([], \bigcup W \in R\ 0. G\ W) \in failures\ Q$   
**by** *simp*  
**qed**  
**moreover have**  $(\bigcup W \in R\ 0. W) \subseteq ?S$   
**proof** (*rule subsetI, simp, erule bexE*)  
**fix**  $x\ W$   
**assume**  $I: W \in R\ 0$   
**hence**  $W = insert\ None\ (F\ W) \cap G\ W$   
**using**  $H$  **by** *simp*  
**moreover assume**  $x \in W$   
**ultimately have**  $x \in insert\ None\ (F\ W) \cap G\ W$   
**by** *simp*  
**thus**  $(x = None \vee (\exists W \in R\ 0. x \in F\ W)) \wedge (\exists W \in R\ 0. x \in G\ W)$   
**(is**  $?A \wedge ?B)$   
**proof** (*rule IntE, simp*)  
**assume**  $x = None \vee x \in F\ W$   
**moreover** {  
**assume**  $x = None$   
**hence**  $?A$  ..

```

}
moreover {
  assume  $x \in F W$ 
  hence  $\exists W \in R \ 0. x \in F W$  using  $I ..$ 
  hence  $?A ..$ 
}
ultimately have  $?A ..$ 
moreover assume  $x \in G W$ 
hence  $?B$  using  $I ..$ 
ultimately show  $?thesis ..$ 
qed
qed
ultimately show  $?thesis$ 
by (rule seq-comp-prop-3)
qed
moreover have  $\forall n \in \{0 <..length\ xs\}$ .
   $(xs, \bigcup W \in R \ n. W) \in seq\text{-comp-failures } P \ Q$ 
proof
  fix  $n$ 
  assume  $F: n \in \{0 <..length\ xs\}$ 
  hence  $G: \forall W \in R \ n.$ 
    take  $(length\ xs - n) \ xs \in sentences\ P \wedge$ 
     $(drop\ (length\ xs - n) \ xs, W) \in failures\ Q$ 
    using  $D$  by simp
  show  $(xs, \bigcup W \in R \ n. W) \in seq\text{-comp-failures } P \ Q$ 
  proof (cases  $\exists W. W \in R \ n$ )
    case False
    thus  $?thesis$ 
    using  $E$  by simp
  next
  case True
  have  $(take\ (length\ xs - n) \ xs @ drop\ (length\ xs - n) \ xs, \bigcup W \in R \ n. W)$ 
     $\in seq\text{-comp-failures } P \ Q$ 
  proof (rule SCF-R3)
    obtain  $W$  where  $W \in R \ n$  using True ..
    thus  $take\ (length\ xs - n) \ xs \in sentences\ P$ 
    using  $G$  by simp
  next
  have  $\forall xs \ A. (\exists W. W \in A) \longrightarrow (\forall W \in A. (xs, W) \in failures\ Q) \longrightarrow$ 
     $(xs, \bigcup W \in A. W) \in failures\ Q$ 
    using  $B$  by (simp add: ref-union-closed-def)
  hence  $(\exists W. W \in R \ n) \longrightarrow$ 
     $(\forall W \in R \ n. (drop\ (length\ xs - n) \ xs, W) \in failures\ Q) \longrightarrow$ 
     $(drop\ (length\ xs - n) \ xs, \bigcup W \in R \ n. W) \in failures\ Q$ 
    by blast
  thus  $(drop\ (length\ xs - n) \ xs, \bigcup W \in R \ n. W) \in failures\ Q$ 
  using  $G$  and True by simp
  next
  show  $drop\ (length\ xs - n) \ xs \neq []$ 

```



```

    using F by (simp, arith)
  qed
  thus ?thesis
  by simp
  qed
  qed
  ultimately have F:  $\forall n \in \{..length\ xs\}.$ 
     $(xs, \bigcup W \in R\ n.\ W) \in seq-comp-failures\ P\ Q$ 
  by auto
  have  $(xs, \bigcup n \in \{..length\ xs\}.\ \bigcup W \in R\ n.\ W) \in seq-comp-failures\ P\ Q$ 
  proof (rule seq-comp-refusals-finite [OF C], simp)
    fix n
    assume  $n \in \{..length\ xs\}$ 
    with F show  $(xs, \bigcup W \in R\ n.\ W) \in seq-comp-failures\ P\ Q ..$ 
  qed
  moreover have  $X = (\bigcup n \in \{..length\ xs\}.\ \bigcup W \in R\ n.\ W)$ 
  using D by simp
  ultimately show ?thesis
  by simp
  qed

```

In what follows, the previous results are used to prove that refusals union closure, weak sequentiality, and sequentiality are conserved under sequential composition. The proof of the first of these lemmas makes use of the axiom of choice.

Since the target security conservation theorem, in addition to the security of both of the processes to be composed, also requires to assume that the first process be refusals union closed and sequential (cf. below), these further conservation lemmas will permit to generalize the theorem to the sequential composition of an arbitrary list of processes.

**lemma** *seq-comp-ref-union-closed*:

**assumes**

*WS*: weakly-sequential *P* **and**

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

*B*: ref-union-closed *Q*

**shows** ref-union-closed (*P* ; *Q*)

**proof** (*simp only*: ref-union-closed-def seq-comp-failures [OF *WS*],

(rule *allI*)+, (rule *impI*)+, erule *exE*)

**fix** *xs A X'*

**assume**

*C*:  $\forall X \in A.\ (xs, X) \in seq-comp-failures\ P\ Q$  **and**

*D*:  $X' \in A$

**have**  $\forall X \in A.\ \exists R.$

$X = (\bigcup n \in \{..length\ xs\}.\ \bigcup W \in R\ n.\ W) \wedge$

$(\forall W \in R.\ 0.$

$xs \notin sentences\ P \wedge None \notin set\ xs \wedge (xs, W) \in failures\ P \vee$

$xs \in \text{sentences } P \wedge (\exists U V. (xs, U) \in \text{failures } P \wedge ([], V) \in \text{failures } Q \wedge$   
 $W = \text{insert None } U \cap V)) \wedge$   
 $(\forall n \in \{0 < .. \text{length } xs\}. \forall W \in R n.$   
 $\text{take } (\text{length } xs - n) \text{ } xs \in \text{sentences } P \wedge$   
 $(\text{drop } (\text{length } xs - n) \text{ } xs, W) \in \text{failures } Q)$   
 $(\text{is } \forall X \in A. \exists R. ?T R X)$

**proof**

**fix**  $X$

**assume**  $X \in A$

**with**  $C$  **have**  $(xs, X) \in \text{seq-comp-failures } P Q ..$

**hence**  $\exists R. X = (\bigcup n \in \{.. \text{length } xs\}. \bigcup W \in R n. W) \wedge$   
 $(\forall W \in R 0.$   
 $xs \notin \text{sentences } P \wedge \text{None} \notin \text{set } xs \wedge (xs, W) \in \text{failures } P \vee$   
 $xs \in \text{sentences } P \wedge (\exists U V. (xs, U) \in \text{failures } P \wedge ([], V) \in \text{failures } Q \wedge$   
 $W = \text{insert None } U \cap V)) \wedge$   
 $(\forall n \in \{0 < .. \text{length } xs\}. \forall W \in R n.$   
 $\text{take } (\text{length } xs - n) \text{ } xs \in \text{sentences } P \wedge$   
 $(\text{drop } (\text{length } xs - n) \text{ } xs, W) \in \text{failures } Q) \wedge$   
 $(\exists n \in \{.. \text{length } xs\}. \exists W. W \in R n)$

**by** (rule seq-comp-refusals-1)

**thus**  $\exists R. ?T R X$

**by** blast

**qed**

**hence**  $\exists R. \forall X \in A. ?T (R X) X$

**by** (rule bchoice)

**then obtain**  $R$  **where**  $E: \forall X \in A. ?T (R X) X ..$

**have**  $xs \in \text{Domain } (\text{seq-comp-failures } P Q)$

**proof** (simp add: Domain-iff)

**have**  $(xs, X) \in \text{seq-comp-failures } P Q$

**using**  $C$  **and**  $D ..$

**thus**  $\exists X. (xs, X) \in \text{seq-comp-failures } P Q ..$

**qed**

**moreover have**  $?T (\lambda n. \bigcup X \in A. R X n) (\bigcup X \in A. X)$

**proof** (rule conjI, rule-tac [2] conjI)

**show**  $(\bigcup X \in A. X) = (\bigcup n \in \{.. \text{length } xs\}. \bigcup W \in \bigcup X \in A. R X n. W)$

**proof** (rule equalityI, rule-tac [!] subsetI, simp-all,  
erule bexE, (erule-tac [2] bexE)+)

**fix**  $x X$

**assume**  $F: X \in A$

**hence**  $X = (\bigcup n \in \{.. \text{length } xs\}. \bigcup W \in R X n. W)$

**using**  $E$  **by** simp

**moreover assume**  $x \in X$

**ultimately have**  $x \in (\bigcup n \in \{.. \text{length } xs\}. \bigcup W \in R X n. W)$

**by** simp

**hence**  $\exists n \in \{.. \text{length } xs\}. \exists W \in R X n. x \in W$

**by** simp

**hence**  $\exists X \in A. \exists n \in \{.. \text{length } xs\}. \exists W \in R X n. x \in W$

**using**  $F ..$

**thus**  $\exists n \in \{.. \text{length } xs\}. \exists X \in A. \exists W \in R X n. x \in W$

by *blast*  
**next**  
 fix  $x\ n\ X\ W$   
 assume  $F: X \in A$   
 hence  $G: X = (\bigcup n \in \{..length\ xs\}. \bigcup W \in R\ X\ n. W)$   
 using  $E$  by *simp*  
 assume  $x \in W$  and  $W \in R\ X\ n$   
 hence  $\exists W \in R\ X\ n. x \in W ..$   
 moreover assume  $n \in \{..length\ xs\}$   
 ultimately have  $\exists n \in \{..length\ xs\}. \exists W \in R\ X\ n. x \in W ..$   
 hence  $x \in (\bigcup n \in \{..length\ xs\}. \bigcup W \in R\ X\ n. W)$   
 by *simp*  
 hence  $x \in X$   
 by (*subst*  $G$ )  
 thus  $\exists X \in A. x \in X$   
 using  $F ..$   
**qed**  
**next**  
**show**  $\forall W \in \bigcup X \in A. R\ X\ 0.$   
 $xs \notin sentences\ P \wedge None \notin set\ xs \wedge (xs, W) \in failures\ P \vee$   
 $xs \in sentences\ P \wedge (\exists U\ V. (xs, U) \in failures\ P \wedge ([], V) \in failures\ Q \wedge$   
 $W = insert\ None\ U \cap V)$   
 (is  $\forall W \in -. ?T\ W$ )  
**proof** (*rule ballI, simp only: UN-iff, erule bexE*)  
 fix  $W\ X$   
 assume  $X \in A$   
 hence  $\forall W \in R\ X\ 0. ?T\ W$   
 using  $E$  by *simp*  
 moreover assume  $W \in R\ X\ 0$   
 ultimately show  $?T\ W ..$   
**qed**  
**next**  
**show**  $\forall n \in \{0 < ..length\ xs\}. \forall W \in \bigcup X \in A. R\ X\ n.$   
 $take\ (length\ xs - n)\ xs \in sentences\ P \wedge$   
 $(drop\ (length\ xs - n)\ xs, W) \in failures\ Q$   
 (is  $\forall n \in -. \forall W \in -. ?T\ n\ W$ )  
**proof** (*(rule ballI)+, simp only: UN-iff, erule bexE*)  
 fix  $n\ W\ X$   
 assume  $X \in A$   
 hence  $\forall n \in \{0 < ..length\ xs\}. \forall W \in R\ X\ n. ?T\ n\ W$   
 using  $E$  by *simp*  
 moreover assume  $n \in \{0 < ..length\ xs\}$   
 ultimately have  $\forall W \in R\ X\ n. ?T\ n\ W ..$   
 moreover assume  $W \in R\ X\ n$   
 ultimately show  $?T\ n\ W ..$   
**qed**  
**qed**  
**ultimately show**  $(xs, \bigcup X \in A. X) \in seq-comp-failures\ P\ Q$   
 by (*rule seq-comp-refusals-2 [OF A B]*)

qed

**lemma** *seq-comp-weakly-sequential*:

**assumes**

*A*: weakly-sequential *P* **and**

*B*: weakly-sequential *Q*

**shows** weakly-sequential (*P* ; *Q*)

**proof** (*subst weakly-sequential-def*, *rule ballI*, *drule traces-failures*,

*subst (asm) seq-comp-failures [OF A]*, *erule seq-comp-failures.induct*, *simp-all*)

**fix** *xs* :: 'a option list

**assume** *C*: *None*  $\notin$  set *xs*

**show** *None*  $\notin$  set (butlast *xs*)

**proof**

**assume** *None*  $\in$  set (butlast *xs*)

**hence** *None*  $\in$  set *xs*

**by** (*rule in-set-butlastD*)

**thus** *False*

**using** *C* **by** *contradiction*

qed

next

**fix** *xs*

**assume** *xs*  $\in$  sentences *P*

**with** *A* **have** *C*: *None*  $\notin$  set *xs*

**by** (*rule weakly-seq-sentences-none*)

**show** *None*  $\notin$  set (butlast *xs*)

**proof**

**assume** *None*  $\in$  set (butlast *xs*)

**hence** *None*  $\in$  set *xs*

**by** (*rule in-set-butlastD*)

**thus** *False*

**using** *C* **by** *contradiction*

qed

next

**fix** *xs ys Y*

**assume** *xs*  $\in$  sentences *P*

**with** *A* **have** *C*: *None*  $\notin$  set *xs*

**by** (*rule weakly-seq-sentences-none*)

**have**  $\forall xs \in$  traces *Q*. *None*  $\notin$  set (butlast *xs*)

**using** *B* **by** (*simp add: weakly-sequential-def*)

**moreover assume** (*ys, Y*)  $\in$  failures *Q*

**hence** *ys*  $\in$  traces *Q*

**by** (*rule failures-traces*)

**ultimately have** *None*  $\notin$  set (butlast *ys*) ..

**hence** *None*  $\notin$  set (*xs* @ butlast *ys*)

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

**moreover assume** *ys*  $\neq$  []

**hence** butlast (*xs* @ *ys*) = *xs* @ butlast *ys*

**by** (*simp add: butlast-append*)

**ultimately show** *None*  $\notin$  set (butlast (*xs* @ *ys*))

by *simp*  
qed

**lemma** *seq-comp-sequential*:

**assumes**

*A*: *sequential P and*

*B*: *sequential Q*

**shows** *sequential (P ; Q)*

**proof** (*subst sequential-def, rule conjI*)

**have** *weakly-sequential P*

**using** *A* **by** (*rule seq-implies-weakly-seq*)

**moreover have** *weakly-sequential Q*

**using** *B* **by** (*rule seq-implies-weakly-seq*)

**ultimately have** *weakly-sequential (P ; Q)*

**by** (*rule seq-comp-weakly-sequential*)

**thus**  $\forall xs \in \text{traces } (P ; Q). \text{None} \notin \text{set } (\text{butlast } xs)$

**by** (*simp add: weakly-sequential-def*)

**next**

**have** *C*: *weakly-sequential P*

**using** *A* **by** (*rule seq-implies-weakly-seq*)

**show**  $\forall xs \in \text{sentences } (P ; Q). \text{next-events } (P ; Q) \text{ } xs = \{\text{None}\}$

**proof** (*rule ballI, simp add: sentences-def next-events-def, rule equalityI,*

*rule-tac [!] subsetI, simp-all, (drule traces-failures)+,*

*simp add: seq-comp-failures [OF C]*)

**fix** *xs x*

**assume**

*D*:  $(xs @ [\text{None}], \{\}) \in \text{seq-comp-failures } P \text{ } Q$  **and**

*E*:  $(xs @ [x], \{\}) \in \text{seq-comp-failures } P \text{ } Q$

**have**  $\exists R. \{\} = (\bigcup n \in \{\dots \text{length } (xs @ [\text{None}])\}. \bigcup W \in R \text{ } n. W) \wedge$   
 $(\forall W \in R \text{ } 0.$

$xs @ [\text{None}] \notin \text{sentences } P \wedge$

$\text{None} \notin \text{set } (xs @ [\text{None}]) \wedge (xs @ [\text{None}], W) \in \text{failures } P \vee$

$xs @ [\text{None}] \in \text{sentences } P \wedge$

$(\exists U \text{ } V. (xs @ [\text{None}], U) \in \text{failures } P \wedge ([], V) \in \text{failures } Q \wedge$

$W = \text{insert } \text{None } U \cap V)) \wedge$

$(\forall n \in \{0 < \dots \text{length } (xs @ [\text{None}])\}. \forall W \in R \text{ } n.$

$\text{take } (\text{length } (xs @ [\text{None}]) - n) (xs @ [\text{None}]) \in \text{sentences } P \wedge$

$(\text{drop } (\text{length } (xs @ [\text{None}]) - n) (xs @ [\text{None}]), W) \in \text{failures } Q) \wedge$

$(\exists n \in \{\dots \text{length } (xs @ [\text{None}])\}. \exists W. W \in R \text{ } n)$

$(\text{is } \exists R. ?T R)$

**using** *D* **by** (*rule seq-comp-refusals-1*)

**then obtain** *R* **where** *F*:  $?T R \dots$

**hence**  $\exists n \in \{\dots \text{Suc } (\text{length } xs)\}. \exists W. W \in R \text{ } n$

**by** *simp*

**moreover have**  $R \text{ } 0 = \{\}$

**proof** (*rule equals0I*)

**fix** *W*

**assume**  $W \in R \text{ } 0$

**hence**  $xs @ [\text{None}] \in \text{sentences } P$

using  $F$  by *simp*  
 with  $C$  have  $\text{None} \notin \text{set } (xs @ [\text{None}])$   
 by (*rule weakly-seq-sentences-none*)  
 thus  $\text{False}$   
 by *simp*  
 qed  
 ultimately have  $\exists n \in \{0 < .. \text{Suc } (\text{length } xs)\}. \exists W. W \in R n$   
 proof –  
 assume  $\exists n \in \{.. \text{Suc } (\text{length } xs)\}. \exists W. W \in R n$   
 then obtain  $n$  where  $G: n \in \{.. \text{Suc } (\text{length } xs)\}$  and  $H: \exists W. W \in R n ..$   
 assume  $I: R 0 = \{\}$   
 show  $\exists n \in \{0 < .. \text{Suc } (\text{length } xs)\}. \exists W. W \in R n$   
 proof (*cases n*)  
 case 0  
 hence  $\exists W. W \in R 0$   
 using  $H$  by *simp*  
 then obtain  $W$  where  $W \in R 0 ..$   
 moreover have  $W \notin R 0$   
 using  $I$  by (*rule equals0D*)  
 ultimately show *?thesis*  
 by *contradiction*  
 next  
 case ( $\text{Suc } m$ )  
 hence  $n \in \{0 < .. \text{Suc } (\text{length } xs)\}$   
 using  $G$  by *simp*  
 with  $H$  show *?thesis ..*  
 qed  
 qed  
 then obtain  $n$  and  $W$  where  $G: n \in \{0 < .. \text{Suc } (\text{length } xs)\}$  and  $W \in R n$   
 by *blast*  
 hence  
 take  $(\text{Suc } (\text{length } xs) - n) (xs @ [\text{None}]) \in \text{sentences } P \wedge$   
 ( $\text{drop } (\text{Suc } (\text{length } xs) - n) (xs @ [\text{None}]), W) \in \text{failures } Q$   
 using  $F$  by *simp*  
 moreover have  $H: \text{Suc } (\text{length } xs) - n \leq \text{length } xs$   
 using  $G$  by (*simp, arith*)  
 ultimately have  $I$ :  
 take  $(\text{Suc } (\text{length } xs) - n) xs \in \text{sentences } P \wedge$   
 ( $\text{drop } (\text{Suc } (\text{length } xs) - n) xs @ [\text{None}], W) \in \text{failures } Q$   
 by *simp*  
 have  $\exists R. \{\} = (\bigcup n \in \{.. \text{length } (xs @ [x])\}. \bigcup W \in R n. W) \wedge$   
 ( $\forall W \in R 0.$   
    $xs @ [x] \notin \text{sentences } P \wedge$   
    $\text{None} \notin \text{set } (xs @ [x]) \wedge (xs @ [x], W) \in \text{failures } P \vee$   
    $xs @ [x] \in \text{sentences } P \wedge$   
    $(\exists U V. (xs @ [x], U) \in \text{failures } P \wedge ([], V) \in \text{failures } Q \wedge$   
    $W = \text{insert None } U \cap V)) \wedge$   
 ( $\forall n \in \{0 < .. \text{length } (xs @ [x])\}. \forall W \in R n.$   
   take  $(\text{length } (xs @ [x]) - n) (xs @ [x]) \in \text{sentences } P \wedge$

$(\text{drop } (\text{length } (xs @ [x]) - n) (xs @ [x]), W) \in \text{failures } Q) \wedge$   
 $(\exists n \in \{..\text{length } (xs @ [x])\}. \exists W. W \in R \ n)$   
 $(\text{is } \exists R. ?T \ R)$   
**using**  $E$  **by**  $(\text{rule seq-comp-refusals-1})$   
**then obtain**  $R'$  **where**  $J: ?T \ R' \ ..$   
**hence**  $\exists n \in \{..\text{Suc } (\text{length } xs)\}. \exists W. W \in R' \ n$   
**by**  $\text{simp}$   
**then obtain**  $n'$  **where**  $K: n' \in \{..\text{Suc } (\text{length } xs)\}$  **and**  $L: \exists W. W \in R' \ n' \ ..$   
**have**  $n' = 0 \vee n' \in \{0 < ..\text{Suc } (\text{length } xs)\}$   
**using**  $K$  **by**  $\text{auto}$   
**thus**  $x = \text{None}$   
**proof**  
**assume**  $n' = 0$   
**hence**  $\exists W. W \in R' \ 0$   
**using**  $L$  **by**  $\text{simp}$   
**then obtain**  $W'$  **where**  $W' \in R' \ 0 \ ..$   
**hence**  
 $xs @ [x] \notin \text{sentences } P \wedge$   
 $\text{None} \notin \text{set } (xs @ [x]) \wedge (xs @ [x], W') \in \text{failures } P \vee$   
 $xs @ [x] \in \text{sentences } P \wedge$   
 $(\exists U \ V. (xs @ [x], U) \in \text{failures } P \wedge ([], V) \in \text{failures } Q \wedge$   
 $W' = \text{insert } \text{None } U \cap V)$   
**using**  $J$  **by**  $\text{simp}$   
**hence**  $M: xs @ [x] \in \text{traces } P \wedge \text{None} \notin \text{set } (xs @ [x])$   
**proof**  $(\text{cases } xs @ [x] \in \text{sentences } P, \text{simp-all}, (\text{erule-tac } [2] \ \text{conjE})+,$   
 $\text{simp-all})$   
**case**  $\text{False}$   
**assume**  $(xs @ [x], W') \in \text{failures } P$   
**thus**  $xs @ [x] \in \text{traces } P$   
**by**  $(\text{rule failures-traces})$   
**next**  
**case**  $\text{True}$   
**hence**  $(xs @ [x]) @ [\text{None}] \in \text{traces } P$   
**by**  $(\text{simp add: sentences-def})$   
**hence**  $xs @ [x] \in \text{traces } P$   
**by**  $(\text{rule process-rule-2-traces})$   
**moreover have**  $\text{None} \notin \text{set } (xs @ [x])$   
**using**  $C$  **and**  $\text{True}$  **by**  $(\text{rule weakly-seq-sentences-none})$   
**ultimately show**  $xs @ [x] \in \text{traces } P \wedge \text{None} \neq x \wedge \text{None} \notin \text{set } xs$   
**by**  $\text{simp}$   
**qed**  
**have**  $xs @ [x] = \text{take } (\text{Suc } (\text{length } xs) - n) (xs @ [x]) @$   
 $\text{drop } (\text{Suc } (\text{length } xs) - n) (xs @ [x])$   
**by**  $(\text{simp only: append-take-drop-id})$   
**hence**  $xs @ [x] = \text{take } (\text{Suc } (\text{length } xs) - n) xs @$   
 $\text{drop } (\text{Suc } (\text{length } xs) - n) xs @ [x]$   
**using**  $H$  **by**  $\text{simp}$   
**moreover have**  $\exists y \ ys. \text{drop } (\text{Suc } (\text{length } xs) - n) xs @ [x] = y \ \# \ ys$   
**by**  $(\text{cases } \text{drop } (\text{Suc } (\text{length } xs) - n) xs @ [x], \text{simp-all})$

**then obtain  $y$  and  $ys$  where**  $drop (Suc (length xs) - n) xs @ [x] = y \# ys$   
**by** *blast*  
**ultimately have**  $N: xs @ [x] = take (Suc (length xs) - n) xs @ y \# ys$   
**by** *simp*  
**have**  $take (Suc (length xs) - n) xs \in sentences P$   
**using**  $I ..$   
**moreover have**  $take (Suc (length xs) - n) xs @ y \# ys \in traces P$   
**using**  $M$  **and**  $N$  **by** *simp*  
**ultimately have**  $y = None$   
**by** (*rule seq-sentences-none [OF A]*)  
**moreover have**  $y \neq None$   
**using**  $M$  **and**  $N$  **by** (*rule-tac not-sym, simp*)  
**ultimately show** *?thesis*  
**by** *contradiction*  
**next**  
**assume**  $M: n' \in \{0 < .. Suc (length xs)\}$   
**moreover obtain**  $W'$  **where**  $W' \in R' n'$   
**using**  $L ..$   
**ultimately have**  
 $take (Suc (length xs) - n') (xs @ [x]) \in sentences P \wedge$   
 $(drop (Suc (length xs) - n') (xs @ [x]), W') \in failures Q$   
**using**  $J$  **by** *simp*  
**moreover have**  $N: Suc (length xs) - n' \leq length xs$   
**using**  $M$  **by** (*simp, arith*)  
**ultimately have**  $O:$   
 $take (Suc (length xs) - n') xs \in sentences P \wedge$   
 $(drop (Suc (length xs) - n') xs @ [x], W') \in failures Q$   
**by** *simp*  
**moreover have**  $n = n'$   
**proof** (*rule ccontr, simp add: neq-iff, erule disjE*)  
**assume**  $P: n < n'$   
**have**  $take (Suc (length xs) - n) xs =$   
 $take (Suc (length xs) - n') (take (Suc (length xs) - n) xs) @$   
 $drop (Suc (length xs) - n') (take (Suc (length xs) - n) xs)$   
**by** (*simp only: append-take-drop-id*)  
**also have**  $... =$   
 $take (Suc (length xs) - n') xs @$   
 $drop (Suc (length xs) - n') (take (Suc (length xs) - n) xs)$   
**using**  $P$  **by** (*simp add: min-def*)  
**also have**  $... =$   
 $take (Suc (length xs) - n') xs @$   
 $take (n' - n) (drop (Suc (length xs) - n') xs)$   
**using**  $M$  **by** (*subst drop-take, simp*)  
**finally have**  $take (Suc (length xs) - n) xs =$   
 $take (Suc (length xs) - n') xs @$   
 $take (n' - n) (drop (Suc (length xs) - n') xs) .$   
**moreover have**  $take (n' - n) (drop (Suc (length xs) - n') xs) \neq []$   
**proof** (*rule-tac notI, simp, erule disjE*)  
**assume**  $n' \leq n$



**thus** *False*  
**using** *P* **by** *simp*  
**next**  
**assume**  $\text{length } xs \leq \text{Suc } (\text{length } xs) - n'$   
**moreover have**  $\text{Suc } (\text{length } xs) - n' < \text{Suc } (\text{length } xs) - n$   
**using** *M* **and** *P* **by** *simp*  
**hence**  $\text{Suc } (\text{length } xs) - n' < \text{length } xs$   
**using** *H* **by** *simp*  
**ultimately show** *False*  
**by** *simp*  
**qed**  
**hence**  $\exists y \text{ ys. take } (n' - n) (\text{drop } (\text{Suc } (\text{length } xs) - n') xs) = y \# \text{ ys}$   
**by** (*cases take (n' - n) (drop (Suc (length xs) - n') xs), simp-all*)  
**then obtain** *y* **and** *ys* **where**  
 $\text{take } (n' - n) (\text{drop } (\text{Suc } (\text{length } xs) - n') xs) = y \# \text{ ys}$   
**by** *blast*  
**ultimately have** *Q*:  $\text{take } (\text{Suc } (\text{length } xs) - n) xs =$   
 $\text{take } (\text{Suc } (\text{length } xs) - n') xs @ y \# \text{ ys}$   
**by** *simp*  
**have**  $\text{take } (\text{Suc } (\text{length } xs) - n') xs \in \text{sentences } P$   
**using** *O* ..  
**moreover have** *R*:  $\text{take } (\text{Suc } (\text{length } xs) - n) xs \in \text{sentences } P$   
**using** *I* ..  
**hence**  $\text{take } (\text{Suc } (\text{length } xs) - n) xs @ [\text{None}] \in \text{traces } P$   
**by** (*simp add: sentences-def*)  
**hence**  $\text{take } (\text{Suc } (\text{length } xs) - n) xs \in \text{traces } P$   
**by** (*rule process-rule-2-traces*)  
**hence**  $\text{take } (\text{Suc } (\text{length } xs) - n') xs @ y \# \text{ ys} \in \text{traces } P$   
**using** *Q* **by** *simp*  
**ultimately have**  $y = \text{None}$   
**by** (*rule seq-sentences-none [OF A]*)  
**moreover have**  $\text{None} \notin \text{set } (\text{take } (\text{Suc } (\text{length } xs) - n) xs)$   
**using** *C* **and** *R* **by** (*rule weakly-seq-sentences-none*)  
**hence**  $y \neq \text{None}$   
**using** *Q* **by** (*rule-tac not-sym, simp*)  
**ultimately show** *False*  
**by** *contradiction*  
**next**  
**assume**  $P: n' < n$   
**have**  $\text{take } (\text{Suc } (\text{length } xs) - n') xs =$   
 $\text{take } (\text{Suc } (\text{length } xs) - n) (\text{take } (\text{Suc } (\text{length } xs) - n') xs) @$   
 $\text{drop } (\text{Suc } (\text{length } xs) - n) (\text{take } (\text{Suc } (\text{length } xs) - n') xs)$   
**by** (*simp only: append-take-drop-id*)  
**also have**  $\dots =$   
 $\text{take } (\text{Suc } (\text{length } xs) - n) xs @$   
 $\text{drop } (\text{Suc } (\text{length } xs) - n) (\text{take } (\text{Suc } (\text{length } xs) - n') xs)$   
**using** *P* **by** (*simp add: min-def*)  
**also have**  $\dots =$   
 $\text{take } (\text{Suc } (\text{length } xs) - n) xs @$

*take*  $(n - n')$  (*drop* (*Suc* (*length xs*) -  $n$ ) *xs*)  
**using** *G* **by** (*subst drop-take*, *simp*)  
**finally have** *take* (*Suc* (*length xs*) -  $n'$ ) *xs* =  
*take* (*Suc* (*length xs*) -  $n$ ) *xs* @  
*take*  $(n - n')$  (*drop* (*Suc* (*length xs*) -  $n$ ) *xs*) .  
**moreover have** *take*  $(n - n')$  (*drop* (*Suc* (*length xs*) -  $n$ ) *xs*)  $\neq$  []  
**proof** (*rule-tac notI*, *simp*, *erule disjE*)  
**assume**  $n \leq n'$   
**thus** *False*  
**using** *P* **by** *simp*  
**next**  
**assume** *length xs*  $\leq$  *Suc* (*length xs*) -  $n$   
**moreover have** *Suc* (*length xs*) -  $n <$  *Suc* (*length xs*) -  $n'$   
**using** *G* **and** *P* **by** *simp*  
**hence** *Suc* (*length xs*) -  $n <$  *length xs*  
**using** *N* **by** *simp*  
**ultimately show** *False*  
**by** *simp*  
**qed**  
**hence**  $\exists y$  *ys*. *take*  $(n - n')$  (*drop* (*Suc* (*length xs*) -  $n$ ) *xs*) =  $y \#$  *ys*  
**by** (*cases take*  $(n - n')$  (*drop* (*Suc* (*length xs*) -  $n$ ) *xs*), *simp-all*)  
**then obtain** *y* **and** *ys* **where**  
*take*  $(n - n')$  (*drop* (*Suc* (*length xs*) -  $n$ ) *xs*) =  $y \#$  *ys*  
**by** *blast*  
**ultimately have** *Q*: *take* (*Suc* (*length xs*) -  $n'$ ) *xs* =  
*take* (*Suc* (*length xs*) -  $n$ ) *xs* @  $y \#$  *ys*  
**by** *simp*  
**have** *take* (*Suc* (*length xs*) -  $n$ ) *xs*  $\in$  *sentences P*  
**using** *I* ..  
**moreover have** *R*: *take* (*Suc* (*length xs*) -  $n'$ ) *xs*  $\in$  *sentences P*  
**using** *O* ..  
**hence** *take* (*Suc* (*length xs*) -  $n'$ ) *xs* @ [*None*]  $\in$  *traces P*  
**by** (*simp add: sentences-def*)  
**hence** *take* (*Suc* (*length xs*) -  $n'$ ) *xs*  $\in$  *traces P*  
**by** (*rule process-rule-2-traces*)  
**hence** *take* (*Suc* (*length xs*) -  $n$ ) *xs* @  $y \#$  *ys*  $\in$  *traces P*  
**using** *Q* **by** *simp*  
**ultimately have**  $y =$  *None*  
**by** (*rule seq-sentences-none [OF A]*)  
**moreover have** *None*  $\notin$  *set* (*take* (*Suc* (*length xs*) -  $n'$ ) *xs*)  
**using** *C* **and** *R* **by** (*rule weakly-seq-sentences-none*)  
**hence**  $y \neq$  *None*  
**using** *Q* **by** (*rule-tac not-sym*, *simp*)  
**ultimately show** *False*  
**by** *contradiction*  
**qed**  
**ultimately have** (*drop* (*Suc* (*length xs*) -  $n$ ) *xs* @ [*x*], *W'*)  $\in$  *failures Q*  
**by** *simp*  
**hence** *P*: *drop* (*Suc* (*length xs*) -  $n$ ) *xs* @ [*x*]  $\in$  *traces Q*

```

    by (rule failures-traces)
  have (drop (Suc (length xs) - n) xs @ [None], W) ∈ failures Q
  using I ..
  hence drop (Suc (length xs) - n) xs @ [None] ∈ traces Q
  by (rule failures-traces)
  hence drop (Suc (length xs) - n) xs ∈ sentences Q
  by (simp add: sentences-def)
  with B show ?thesis
  using P by (rule seq-sentences-none)
qed
qed
qed

```

## 2.4 Conservation of noninterference security under sequential composition

Everything is now ready for proving the target security conservation theorem. The two closure properties that the definition of noninterference security requires process futures to satisfy, one for the addition of events into traces and the other for the deletion of events from traces (cf. [8]), will be faced separately; here below is the proof of the former property. Unsurprisingly, rule induction on set *seq-comp-failures* is applied, and the closure of the failures of a secure process under intransitive purge (proven in the previous section) is used to meet the proof obligations arising from rule *SCF-R3*.

**lemma** *seq-comp-secure-aux-1-case-1*:

**assumes**

*A*: *secure-termination I D and*

*B*: *sequential P and*

*C*: *secure P I D and*

*D*:  $xs @ y \# ys \notin \text{sentences } P$  **and**

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

*F*:  $\text{None} \neq y$  **and**

*G*:  $\text{None} \notin \text{set } xs$  **and**

*H*:  $\text{None} \notin \text{set } ys$

**shows**  $(xs @ \text{ipurge-tr } I D (D y) ys, \text{ipurge-ref } I D (D y) ys X) \in \text{seq-comp-failures } P Q$

**proof** –

**have**  $(y \# ys, X) \in \text{futures } P xs$

**using** *E* **by** (*simp add: futures-def*)

**hence**  $(\text{ipurge-tr } I D (D y) ys, \text{ipurge-ref } I D (D y) ys X) \in \text{futures } P xs$

**using** *C* **by** (*simp add: secure-def*)

**hence** *I*:  $(xs @ \text{ipurge-tr } I D (D y) ys, \text{ipurge-ref } I D (D y) ys X) \in \text{failures } P$

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

**show** ?thesis  
**proof** (cases  $xs @ ipurge-tr I D (D y) ys \in sentences P$ ,  
cases  $(D y, D None) \in I \vee (\exists u \in sinks I D (D y) ys. (u, D None) \in I)$ ,  
simp-all)  
**assume**  $xs @ ipurge-tr I D (D y) ys \notin sentences P$   
**thus** ?thesis **using**  $I$   
**proof** (rule SCF-R1, simp add:  $F G$ )  
**have**  $set (ipurge-tr I D (D y) ys) \subseteq set ys$   
**by** (rule ipurge-tr-set)  
**thus**  $None \notin set (ipurge-tr I D (D y) ys)$   
**using**  $H$  **by** (rule contra-subsetD)  
**qed**  
**next**  
**assume**  
 $J: xs @ ipurge-tr I D (D y) ys \in sentences P$  **and**  
 $K: (D y, D None) \in I \vee (\exists u \in sinks I D (D y) ys. (u, D None) \in I)$   
**have**  $ipurge-ref I D (D y) ys X = \{\}$   
**proof** (rule disjE [OF  $K$ ], erule-tac [2] bexE)  
**assume**  $L: (D y, D None) \in I$   
**show** ?thesis  
**proof** (rule ipurge-ref-empty [of  $D y$ ], simp)  
**fix**  $x$   
**have**  $(D y, D None) \in I \wedge y \neq None \longrightarrow (\forall u \in range D. (D y, u) \in I)$   
**using**  $A$  **by** (simp add: secure-termination-def)  
**hence**  $\forall u \in range D. (D y, u) \in I$   
**using**  $F$  **and**  $L$  **by** simp  
**thus**  $(D y, D x) \in I$   
**by** simp  
**qed**  
**next**  
**fix**  $u$   
**assume**  
 $L: u \in sinks I D (D y) ys$  **and**  
 $M: (u, D None) \in I$   
**have**  $\exists y' \in set ys. u = D y'$   
**using**  $L$  **by** (rule sinks-elem)  
**then obtain**  $y'$  **where**  $N: y' \in set ys$  **and**  $O: u = D y' ..$   
**have**  $P: y' \neq None$   
**proof**  
**assume**  $y' = None$   
**hence**  $None \in set ys$   
**using**  $N$  **by** simp  
**thus**  $False$   
**using**  $H$  **by** contradiction  
**qed**  
**show** ?thesis  
**proof** (rule ipurge-ref-empty [of  $u$ ], simp add:  $L$ )  
**fix**  $x$   
**have**  $(D y', D None) \in I \wedge y' \neq None \longrightarrow (\forall v \in range D. (D y', v) \in I)$

using  $A$  by (*simp add: secure-termination-def*)  
 moreover have  $(D y', D None) \in I$   
 using  $M$  and  $O$  by *simp*  
 ultimately have  $\forall v \in \text{range } D. (D y', v) \in I$   
 using  $P$  by *simp*  
 thus  $(u, D x) \in I$   
 using  $O$  by *simp*  
 qed  
 qed  
 thus ?thesis  
 proof *simp*  
 have  $([], \{\}) \in \text{failures } Q$   
 by (*rule process-rule-1*)  
 with  $J$  and  $I$  have  $(xs @ \text{ipurge-tr } I D (D y) ys,$   
    $\text{insert None } (\text{ipurge-ref } I D (D y) ys X) \cap \{\}) \in \text{seq-comp-failures } P Q$   
 by (*rule SCF-R2*)  
 thus  $(xs @ \text{ipurge-tr } I D (D y) ys, \{\}) \in \text{seq-comp-failures } P Q$   
 by *simp*  
 qed  
 next  
 assume  
    $J: xs @ \text{ipurge-tr } I D (D y) ys \in \text{sentences } P$  and  
    $K: (D y, D None) \notin I \wedge (\forall u \in \text{sinks } I D (D y) ys. (u, D None) \notin I)$   
 have  $(xs @ [y]) @ ys \in \text{sentences } P$   
 proof (*simp add: sentences-def del: append-assoc, subst (2) append-assoc,*  
   *rule ipurge-tr-del-traces [OF C, where u = D y], simp-all add: K*)  
 have  $(y \# ys, X) \in \text{futures } P xs$   
 using  $E$  by (*simp add: futures-def*)  
 moreover have  $xs @ \text{ipurge-tr } I D (D y) ys @ [None] \in \text{traces } P$   
 using  $J$  by (*simp add: sentences-def*)  
 hence  $(xs @ \text{ipurge-tr } I D (D y) ys @ [None], \{\}) \in \text{failures } P$   
 by (*rule traces-failures*)  
 hence  $(\text{ipurge-tr } I D (D y) ys @ [None], \{\}) \in \text{futures } P xs$   
 by (*simp add: futures-def*)  
 ultimately have  $(y \# \text{ipurge-tr } I D (D y) (\text{ipurge-tr } I D (D y) ys @ [None]),$   
    $\text{ipurge-ref } I D (D y) (\text{ipurge-tr } I D (D y) ys @ [None]) \{\}) \in \text{futures } P xs$   
 using  $C$  by (*simp add: secure-def del: ipurge-tr.simps*)  
 hence  $(xs @ y \# \text{ipurge-tr } I D (D y) (\text{ipurge-tr } I D (D y) ys @ [None]), \{\})$   
    $\in \text{failures } P$   
 by (*simp add: futures-def ipurge-ref-def*)  
 moreover have  $\text{sinks } I D (D y) (\text{ipurge-tr } I D (D y) ys) = \{\}$   
 by (*rule sinks-idem*)  
 hence  $\neg ((D y, D None) \in I \vee$   
    $(\exists u \in \text{sinks } I D (D y) (\text{ipurge-tr } I D (D y) ys). (u, D None) \in I))$   
 using  $K$  by *simp*  
 hence  $D None \notin \text{sinks } I D (D y) (\text{ipurge-tr } I D (D y) ys @ [None])$   
 by (*simp only: sinks-interference-eq, simp*)  
 ultimately have  $(xs @ y \# \text{ipurge-tr } I D (D y) (\text{ipurge-tr } I D (D y) ys)$   
    $@ [None], \{\}) \in \text{failures } P$

by *simp*  
 hence  $(xs @ y \# \text{ipurge-tr } I D (D y) ys @ [None], \{\}) \in \text{failures } P$   
 by (*simp add: ipurge-tr-idem*)  
 thus  $xs @ y \# \text{ipurge-tr } I D (D y) ys @ [None] \in \text{traces } P$   
 by (*rule failures-traces*)  
 next  
 show  $xs @ y \# ys \in \text{traces } P$   
 using *E* by (*rule failures-traces*)  
 qed  
 hence  $xs @ y \# ys \in \text{sentences } P$   
 by *simp*  
 thus ?thesis  
 using *D* by *contradiction*  
 qed  
 qed

**lemma** *seq-comp-secure-aux-1-case-2*:

**assumes**

*A*: *secure-termination* *I D* **and**

*B*: *sequential* *P* **and**

*C*: *secure* *P I D* **and**

*D*: *secure* *Q I D* **and**

*E*:  $xs @ y \# ys \in \text{sentences } P$  **and**

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

*G*:  $([], Y) \in \text{failures } Q$

**shows**  $(xs @ \text{ipurge-tr } I D (D y) ys,$

*ipurge-ref* *I D (D y) ys (insert None X  $\cap$  Y))  $\in \text{seq-comp-failures } P Q$*

**proof** –

**have**  $(y \# ys, X) \in \text{futures } P xs$

using *F* by (*simp add: futures-def*)

**hence**  $(\text{ipurge-tr } I D (D y) ys, \text{ipurge-ref } I D (D y) ys X)$

$\in \text{futures } P xs$

using *C* by (*simp add: secure-def*)

**hence** *H*:  $(xs @ \text{ipurge-tr } I D (D y) ys, \text{ipurge-ref } I D (D y) ys X)$

$\in \text{failures } P$

by (*simp add: futures-def*)

**have** *weakly-sequential* *P*

using *B* by (*rule seq-implies-weakly-seq*)

**hence** *I*:  $None \notin \text{set } (xs @ y \# ys)$

using *E* by (*rule weakly-seq-sentences-none*)

**show** ?thesis

**proof** (*cases*  $xs @ \text{ipurge-tr } I D (D y) ys \in \text{sentences } P,$

*case-tac* [2]  $(D y, D None) \in I \vee (\exists u \in \text{sinks } I D (D y) ys. (u, D None) \in I),$

*simp-all*)

**assume** *J*:  $xs @ \text{ipurge-tr } I D (D y) ys \in \text{sentences } P$

**have** *ipurge-ref* *I D (D y) ys*  $Y \subseteq Y$

by (*rule ipurge-ref-subset*)

**with** *G* **have**  $([], \text{ipurge-ref } I D (D y) ys Y) \in \text{failures } Q$

by (*rule process-rule-3*)

**with  $J$  and  $H$  have**  $(xs @ \text{ipurge-tr } I D (D y) ys,$   
 $\text{insert None } (\text{ipurge-ref } I D (D y) ys X) \cap \text{ipurge-ref } I D (D y) ys Y)$   
 $\in \text{seq-comp-failures } P Q$   
**by**  $(\text{rule } \text{SCF-R2})$   
**moreover have**  
 $\text{ipurge-ref } I D (D y) ys (\text{insert None } X) \cap \text{ipurge-ref } I D (D y) ys Y \subseteq$   
 $\text{insert None } (\text{ipurge-ref } I D (D y) ys X) \cap \text{ipurge-ref } I D (D y) ys Y$   
**proof**  $(\text{rule } \text{subsetI}, \text{simp del: insert-iff}, \text{erule conjE})$   
**fix**  $x$   
**have**  $\text{ipurge-ref } I D (D y) ys (\text{insert None } X) \subseteq$   
 $\text{insert None } (\text{ipurge-ref } I D (D y) ys X)$   
**by**  $(\text{rule } \text{ipurge-ref-subset-insert})$   
**moreover assume**  $x \in \text{ipurge-ref } I D (D y) ys (\text{insert None } X)$   
**ultimately show**  $x \in \text{insert None } (\text{ipurge-ref } I D (D y) ys X) ..$   
**qed**  
**ultimately have**  $(xs @ \text{ipurge-tr } I D (D y) ys,$   
 $\text{ipurge-ref } I D (D y) ys (\text{insert None } X) \cap \text{ipurge-ref } I D (D y) ys Y)$   
 $\in \text{seq-comp-failures } P Q$   
**by**  $(\text{rule } \text{seq-comp-prop-3})$   
**thus**  $?thesis$   
**by**  $(\text{simp add: ipurge-ref-distrib-inter})$   
**next**  
**assume**  
 $J: xs @ \text{ipurge-tr } I D (D y) ys \notin \text{sentences } P$  **and**  
 $K: (D y, D None) \in I \vee (\exists u \in \text{sinks } I D (D y) ys. (u, D None) \in I)$   
**have**  $\text{ipurge-ref } I D (D y) ys (\text{insert None } X \cap Y) = \{\}$   
**proof**  $(\text{rule } \text{disjE } [OF K], \text{erule-tac } [2] \text{bexE})$   
**assume**  $L: (D y, D None) \in I$   
**show**  $?thesis$   
**proof**  $(\text{rule } \text{ipurge-ref-empty } [of D y], \text{simp})$   
**fix**  $x$   
**have**  $(D y, D None) \in I \wedge y \neq None \longrightarrow (\forall u \in \text{range } D. (D y, u) \in I)$   
**using**  $A$  **by**  $(\text{simp add: secure-termination-def})$   
**moreover have**  $y \neq None$   
**using**  $I$  **by**  $(\text{rule-tac not-sym}, \text{simp})$   
**ultimately have**  $\forall u \in \text{range } D. (D y, u) \in I$   
**using**  $L$  **by**  $\text{simp}$   
**thus**  $(D y, D x) \in I$   
**by**  $\text{simp}$   
**qed**  
**next**  
**fix**  $u$   
**assume**  
 $L: u \in \text{sinks } I D (D y) ys$  **and**  
 $M: (u, D None) \in I$   
**have**  $\exists y' \in \text{set } ys. u = D y'$   
**using**  $L$  **by**  $(\text{rule sinks-elem})$   
**then obtain**  $y'$  **where**  $N: y' \in \text{set } ys$  **and**  $O: u = D y' ..$   
**have**  $P: y' \neq None$

**proof**  
 assume  $y' = \text{None}$   
 hence  $\text{None} \in \text{set } ys$   
 using  $N$  by *simp*  
 moreover have  $\text{None} \notin \text{set } ys$   
 using  $I$  by *simp*  
 ultimately show *False*  
 by *contradiction*  
**qed**  
 show *?thesis*  
**proof** (rule *ipurge-ref-empty* [of  $u$ ], *simp add: L*)  
 fix  $x$   
 have  $(D y', D \text{None}) \in I \wedge y' \neq \text{None} \longrightarrow (\forall v \in \text{range } D. (D y', v) \in I)$   
 using  $A$  by (*simp add: secure-termination-def*)  
 moreover have  $(D y', D \text{None}) \in I$   
 using  $M$  and  $O$  by *simp*  
 ultimately have  $\forall v \in \text{range } D. (D y', v) \in I$   
 using  $P$  by *simp*  
 thus  $(u, D x) \in I$   
 using  $O$  by *simp*  
**qed**  
**qed**  
 thus *?thesis*  
**proof** *simp*  
 have  $\{\} \subseteq \text{ipurge-ref } I D (D y) ys X ..$   
 with  $H$  have  $(xs @ \text{ipurge-tr } I D (D y) ys, \{\}) \in \text{failures } P$   
 by (rule *process-rule-3*)  
 with  $J$  show  $(xs @ \text{ipurge-tr } I D (D y) ys, \{\}) \in \text{seq-comp-failures } P Q$   
**proof** (rule *SCF-R1*)  
 show  $\text{None} \notin \text{set } (xs @ \text{ipurge-tr } I D (D y) ys)$   
 using  $I$   
**proof** (*simp*, (*erule-tac conjE*)+)  
 have  $\text{set } (\text{ipurge-tr } I D (D y) ys) \subseteq \text{set } ys$   
 by (rule *ipurge-tr-set*)  
 moreover assume  $\text{None} \notin \text{set } ys$   
 ultimately show  $\text{None} \notin \text{set } (\text{ipurge-tr } I D (D y) ys)$   
 by (rule *contra-subsetD*)  
**qed**  
**qed**  
**qed**  
**next**  
**assume**  
 $J: xs @ \text{ipurge-tr } I D (D y) ys \notin \text{sentences } P$  **and**  
 $K: (D y, D \text{None}) \notin I \wedge (\forall u \in \text{sinks } I D (D y) ys. (u, D \text{None}) \notin I)$   
 have  $xs @ y \# ys @ [\text{None}] \in \text{traces } P$   
 using  $E$  by (*simp add: sentences-def*)  
 hence  $(xs @ y \# ys @ [\text{None}], \{\}) \in \text{failures } P$   
 by (rule *traces-failures*)  
 hence  $(y \# ys @ [\text{None}], \{\}) \in \text{futures } P xs$



by (*simp add: futures-def*)  
 hence (*ipurge-tr I D (D y) (ys @ [None]),*  
*ipurge-ref I D (D y) (ys @ [None]) {}*)  $\in$  *futures P xs*  
 (*is (-, ?Y)  $\in$  -*)  
 using *C* by (*simp add: secure-def del: ipurge-tr.simps*)  
 hence (*xs @ ipurge-tr I D (D y) (ys @ [None]), ?Y*)  $\in$  *failures P*  
 by (*simp add: futures-def*)  
 hence *xs @ ipurge-tr I D (D y) (ys @ [None])*  $\in$  *traces P*  
 by (*rule failures-traces*)  
 moreover have  $\neg ((D y, D None) \in I \vee$   
 $(\exists u \in \text{sinks } I D (D y) \text{ ys. } (u, D None) \in I))$   
 using *K* by *simp*  
 hence *D None*  $\notin$  *sinks I D (D y) (ys @ [None])*  
 by (*simp only: sinks-interference-eq, simp*)  
 ultimately have *xs @ ipurge-tr I D (D y) ys @ [None]*  $\in$  *traces P*  
 by *simp*  
 hence *xs @ ipurge-tr I D (D y) ys*  $\in$  *sentences P*  
 by (*simp add: sentences-def*)  
 thus *?thesis*  
 using *J* by *contradiction*  
 qed  
 qed

**lemma** *seq-comp-secure-aux-1-case-3:*

**assumes**  
*A: secure-termination I D and*  
*B: ref-union-closed Q and*  
*C: sequential Q and*  
*D: secure Q I D and*  
*E: secure R I D and*  
*F: ws  $\in$  sentences Q and*  
*G: (ys', Y)  $\in$  failures R and*  
*H: ws @ ys' = xs @ y # ys*  
**shows** (*xs @ ipurge-tr I D (D y) ys, ipurge-ref I D (D y) ys Y*)  
 $\in$  *seq-comp-failures Q R*  
**proof** (*cases length xs < length ws*)  
**case** *True*  
**have** *drop (Suc (length xs)) (xs @ y # ys) = drop (Suc (length xs)) (ws @ ys')*  
**using** *H* by *simp*  
**hence** *I: ys = drop (Suc (length xs)) ws @ ys'*  
 (*is - = ?ws' @ -*)  
**using** *True* by *simp*  
**let** *?U = insert (D y) (sinks I D (D y) ?ws')*  
**have** *ipurge-tr I D (D y) ys =*  
*ipurge-tr I D (D y) ?ws' @ ipurge-tr-aux I D ?U ys'*  
**using** *I* by (*simp add: ipurge-tr-append*)  
**moreover** have *ipurge-ref I D (D y) ys Y = ipurge-ref-aux I D ?U ys' Y*  
**using** *I* by (*simp add: ipurge-ref-append*)  
**ultimately show** *?thesis*

**proof** (cases  $xs @ ipurge-tr I D (D y) ?ws' \in sentences Q$ , *simp-all*)  
**assume**  $J: xs @ ipurge-tr I D (D y) ?ws' \in sentences Q$   
**have**  $K: (ipurge-tr-aux I D ?U ys', ipurge-ref-aux I D ?U ys' Y) \in failures R$   
**using**  $E$  and  $G$  **by** (rule *ipurge-tr-ref-aux-failures*)  
**show**  $(xs @ ipurge-tr I D (D y) ?ws' @ ipurge-tr-aux I D ?U ys',$   
 $ipurge-ref-aux I D ?U ys' Y) \in seq-comp-failures Q R$   
**proof** (cases *ipurge-tr-aux I D ?U ys'*)  
**case** *Nil*  
**have**  $(xs @ ipurge-tr I D (D y) ?ws', \{x. x \neq None\}) \in failures Q$   
**using**  $B$  and  $C$  and  $J$  **by** (rule *seq-sentences-ref*)  
**moreover have**  $([], ipurge-ref-aux I D ?U ys' Y) \in failures R$   
**using**  $K$  and *Nil* **by** *simp*  
**ultimately have**  $(xs @ ipurge-tr I D (D y) ?ws',$   
 $insert None \{x. x \neq None\} \cap ipurge-ref-aux I D ?U ys' Y)$   
 $\in seq-comp-failures Q R$   
**by** (rule *SCF-R2 [OF J]*)  
**moreover have**  $insert None \{x. x \neq None\} \cap$   
 $ipurge-ref-aux I D ?U ys' Y = ipurge-ref-aux I D ?U ys' Y$   
**by** *blast*  
**ultimately show** *?thesis*  
**using** *Nil* **by** *simp*  
**next**  
**case** *Cons*  
**hence**  $ipurge-tr-aux I D ?U ys' \neq []$   
**by** *simp*  
**with**  $J$  and  $K$  **have**  
 $((xs @ ipurge-tr I D (D y) ?ws') @ ipurge-tr-aux I D ?U ys',$   
 $ipurge-ref-aux I D ?U ys' Y) \in seq-comp-failures Q R$   
**by** (rule *SCF-R3*)  
**thus** *?thesis*  
**by** *simp*  
**qed**  
**next**  
**assume**  $J: xs @ ipurge-tr I D (D y) ?ws' \notin sentences Q$   
**have**  $ws = take (Suc (length xs)) ws @ ?ws'$   
**by** *simp*  
**moreover have**  $take (Suc (length xs)) (ws @ ys') =$   
 $take (Suc (length xs)) (xs @ y \# ys)$   
**using**  $H$  **by** *simp*  
**hence**  $take (Suc (length xs)) ws = xs @ [y]$   
**using** *True* **by** *simp*  
**ultimately have**  $K: xs @ y \# ?ws' \in sentences Q$   
**using**  $F$  **by** *simp*  
**hence**  $xs @ y \# ?ws' @ [None] \in traces Q$   
**by** (*simp add: sentences-def*)  
**hence**  $(xs @ y \# ?ws' @ [None], \{\}) \in failures Q$   
**by** (rule *traces-failures*)  
**hence**  $(y \# ?ws' @ [None], \{\}) \in futures Q xs$   
**by** (*simp add: futures-def*)

**hence** (*ipurge-tr*  $I D (D y) (?ws' @ [None])$ ,  
*ipurge-ref*  $I D (D y) (?ws' @ [None]) \{\}$ )  $\in$  *futures*  $Q xs$   
**using**  $D$  **by** (*simp add: secure-def del: ipurge-tr.simps*)  
**hence**  $L: (xs @ ipurge-tr I D (D y) (?ws' @ [None]), \{\}) \in$  *failures*  $Q$   
**by** (*simp add: futures-def ipurge-ref-def*)  
**have** *weakly-sequential*  $Q$   
**using**  $C$  **by** (*rule seq-implies-weakly-seq*)  
**hence**  $N: None \notin set (xs @ y \# ?ws')$   
**using**  $K$  **by** (*rule weakly-seq-sentences-none*)  
**show** ( $xs @ ipurge-tr I D (D y) ?ws' @ ipurge-tr-aux I D ?U ys'$ ,  
*ipurge-ref-aux*  $I D ?U ys' Y$ )  $\in$  *seq-comp-failures*  $Q R$   
**proof** (*cases*  $(D y, D None) \in I \vee$   
 $(\exists u \in sinks I D (D y) ?ws'. (u, D None) \in I)$ )  
**assume**  $M: (D y, D None) \in I \vee$   
 $(\exists u \in sinks I D (D y) ?ws'. (u, D None) \in I)$   
**have** *ipurge-tr-aux*  $I D ?U ys' = []$   
**proof** (*rule disjE [OF M], erule-tac [2] bexE*)  
**assume**  $O: (D y, D None) \in I$   
**show** *thesis*  
**proof** (*rule ipurge-tr-aux-nil [of D y], simp*)  
**fix**  $x$   
**have**  $(D y, D None) \in I \wedge y \neq None \longrightarrow (\forall u \in range D. (D y, u) \in I)$   
**using**  $A$  **by** (*simp add: secure-termination-def*)  
**moreover** **have**  $y \neq None$   
**using**  $N$  **by** (*rule-tac not-sym, simp*)  
**ultimately** **have**  $\forall u \in range D. (D y, u) \in I$   
**using**  $O$  **by** *simp*  
**thus**  $(D y, D x) \in I$   
**by** *simp*  
**qed**  
**next**  
**fix**  $u$   
**assume**  
 $O: u \in sinks I D (D y) ?ws'$  **and**  
 $P: (u, D None) \in I$   
**have**  $\exists w \in set ?ws'. u = D w$   
**using**  $O$  **by** (*rule sinks-elim*)  
**then obtain**  $w$  **where**  $Q: w \in set ?ws'$  **and**  $R: u = D w ..$   
**have**  $S: w \neq None$   
**proof**  
**assume**  $w = None$   
**hence**  $None \in set ?ws'$   
**using**  $Q$  **by** *simp*  
**moreover** **have**  $None \notin set ?ws'$   
**using**  $N$  **by** *simp*  
**ultimately** **show** *False*  
**by** *contradiction*  
**qed**  
**show** *thesis*

```

proof (rule ipurge-tr-aux-nil [of u], simp add: O)
  fix x
  have (D w, D None) ∈ I ∧ w ≠ None → (∀ v ∈ range D. (D w, v) ∈ I)
    using A by (simp add: secure-termination-def)
  moreover have (D w, D None) ∈ I
    using P and R by simp
  ultimately have ∀ v ∈ range D. (D w, v) ∈ I
    using S by simp
  thus (u, D x) ∈ I
    using R by simp
qed
qed
moreover have ipurge-ref-aux I D ?U ys' Y = {}
proof (rule disjE [OF M], erule-tac [2] bexE)
  assume O: (D y, D None) ∈ I
  show ?thesis
  proof (rule ipurge-ref-aux-empty [of D y])
    have ?U ⊆ sinks-aux I D ?U ys'
      by (rule sinks-aux-subset)
    moreover have D y ∈ ?U
      by simp
    ultimately show D y ∈ sinks-aux I D ?U ys' ..
  next
  fix x
  have (D y, D None) ∈ I ∧ y ≠ None → (∀ u ∈ range D. (D y, u) ∈ I)
    using A by (simp add: secure-termination-def)
  moreover have y ≠ None
    using N by (rule-tac not-sym, simp)
  ultimately have ∀ u ∈ range D. (D y, u) ∈ I
    using O by simp
  thus (D y, D x) ∈ I
    by simp
  qed
next
fix u
assume
  O: u ∈ sinks I D (D y) ?ws' and
  P: (u, D None) ∈ I
have ∃ w ∈ set ?ws'. u = D w
  using O by (rule sinks-elim)
then obtain w where Q: w ∈ set ?ws' and R: u = D w ..
have S: w ≠ None
proof
  assume w = None
  hence None ∈ set ?ws'
    using Q by simp
  moreover have None ∉ set ?ws'
    using N by simp
  ultimately show False

```

by contradiction  
 qed  
 show *?thesis*  
 proof (rule *ipurge-ref-aux-empty* [of *u*])  
   have  $?U \subseteq \text{sinks-aux } I D ?U \text{ys}'$   
   by (rule *sinks-aux-subset*)  
   moreover have  $u \in ?U$   
   using *O* by *simp*  
   ultimately show  $u \in \text{sinks-aux } I D ?U \text{ys}' ..$   
 next  
 fix *x*  
 have  $(D w, D None) \in I \wedge w \neq None \longrightarrow (\forall v \in \text{range } D. (D w, v) \in I)$   
   using *A* by (*simp add: secure-termination-def*)  
   moreover have  $(D w, D None) \in I$   
   using *P* and *R* by *simp*  
   ultimately have  $\forall v \in \text{range } D. (D w, v) \in I$   
   using *S* by *simp*  
   thus  $(u, D x) \in I$   
   using *R* by *simp*  
 qed  
 qed  
 ultimately show *?thesis*  
 proof *simp*  
   have  $D None \in \text{sinks } I D (D y) (?ws' @ [None])$   
   using *M* by (*simp only: sinks-interference-eq*)  
   hence  $(xs @ \text{ipurge-tr } I D (D y) ?ws', \{\}) \in \text{failures } Q$   
   using *L* by *simp*  
   moreover have  $None \notin \text{set } (xs @ \text{ipurge-tr } I D (D y) ?ws')$   
   proof –  
     show *?thesis*  
     using *N*  
     proof (*simp, (erule-tac conjE)+*)  
       have  $\text{set } (\text{ipurge-tr } I D (D y) ?ws') \subseteq \text{set } ?ws'$   
       by (rule *ipurge-tr-set*)  
       moreover assume  $None \notin \text{set } ?ws'$   
       ultimately show  $None \notin \text{set } (\text{ipurge-tr } I D (D y) ?ws')$   
       by (rule *contra-subsetD*)  
     qed  
   qed  
   ultimately show  $(xs @ \text{ipurge-tr } I D (D y) ?ws', \{\})$   
    $\in \text{seq-comp-failures } Q R$   
   by (rule *SCF-R1* [OF *J*])  
 qed  
 next  
 assume  $\neg ((D y, D None) \in I \vee$   
    $(\exists u \in \text{sinks } I D (D y) ?ws'. (u, D None) \in I))$   
 hence  $D None \notin \text{sinks } I D (D y) (?ws' @ [None])$   
   by (*simp only: sinks-interference-eq, simp*)  
 hence  $(xs @ \text{ipurge-tr } I D (D y) ?ws' @ [None], \{\}) \in \text{failures } Q$

```

    using L by simp
    hence xs @ ipurge-tr I D (D y) ?ws' @ [None] ∈ traces Q
    by (rule failures-traces)
    hence xs @ ipurge-tr I D (D y) ?ws' ∈ sentences Q
    by (simp add: sentences-def)
    thus ?thesis
    using J by contradiction
  qed
qed
next
case False
have drop (length ws) (ws @ ys') = drop (length ws) (xs @ y # ys)
  using H by simp
hence ys' = drop (length ws) xs @ y # ys
  (is - = ?xs' @ -)
  using False by simp
hence (?xs' @ y # ys, Y) ∈ failures R
  using G by simp
hence (y # ys, Y) ∈ futures R ?xs'
  by (simp add: futures-def)
hence (ipurge-tr I D (D y) ys, ipurge-ref I D (D y) ys Y)
  ∈ futures R ?xs'
  using E by (simp add: secure-def)
hence I: (?xs' @ ipurge-tr I D (D y) ys, ipurge-ref I D (D y) ys Y)
  ∈ failures R
  by (simp add: futures-def)
have xs = take (length ws) xs @ ?xs'
  by simp
hence xs = take (length ws) (xs @ y # ys) @ ?xs'
  using False by simp
hence xs = take (length ws) (ws @ ys') @ ?xs'
  using H by simp
hence J: xs = ws @ ?xs'
  by simp
show ?thesis
proof (cases ?xs' @ ipurge-tr I D (D y) ys = [], insert I, subst J, simp)
  have (ws, {x. x ≠ None}) ∈ failures Q
    using B and C and F by (rule seq-sentences-ref)
  moreover assume ([], ipurge-ref I D (D y) ys Y) ∈ failures R
  ultimately have (ws, insert None {x. x ≠ None}) ∩
    ipurge-ref I D (D y) ys Y ∈ seq-comp-failures Q R
    by (rule SCF-R2 [OF F])
  moreover have insert None {x. x ≠ None} ∩ ipurge-ref I D (D y) ys Y =
    ipurge-ref I D (D y) ys Y
    by blast
  ultimately show (ws, ipurge-ref I D (D y) ys Y) ∈ seq-comp-failures Q R
    by simp
next
assume ?xs' @ ipurge-tr I D (D y) ys ≠ []

```

**with  $F$  and  $I$  have**  
 $(ws @ ?xs' @ \text{ipurge-tr } I D (D y) ys, \text{ipurge-ref } I D (D y) ys Y)$   
 $\in \text{seq-comp-failures } Q R$   
**by** (*rule SCF-R3*)  
**hence**  $((ws @ ?xs') @ \text{ipurge-tr } I D (D y) ys, \text{ipurge-ref } I D (D y) ys Y)$   
 $\in \text{seq-comp-failures } Q R$   
**by** *simp*  
**thus** *?thesis*  
**using**  $J$  **by** *simp*  
**qed**  
**qed**

**lemma** *seq-comp-secure-aux-1* [*rule-format*]:  
**assumes**  
 $A$ : *secure-termination*  $I D$  **and**  
 $B$ : *ref-union-closed*  $P$  **and**  
 $C$ : *sequential*  $P$  **and**  
 $D$ : *secure*  $P I D$  **and**  
 $E$ : *secure*  $Q I D$   
**shows**  $(ws, Y) \in \text{seq-comp-failures } P Q \implies$   
 $ws = xs @ y \# ys \longrightarrow$   
 $(xs @ \text{ipurge-tr } I D (D y) ys, \text{ipurge-ref } I D (D y) ys Y)$   
 $\in \text{seq-comp-failures } P Q$

**proof** (*erule seq-comp-failures.induct, rule-tac [!] impI, simp-all, (erule conjE)+*)  
**fix**  $X$   
**assume**  
 $xs @ y \# ys \notin \text{sentences } P$  **and**  
 $(xs @ y \# ys, X) \in \text{failures } P$  **and**  
 $\text{None} \neq y$  **and**  
 $\text{None} \notin \text{set } xs$  **and**  
 $\text{None} \notin \text{set } ys$   
**thus**  $(xs @ \text{ipurge-tr } I D (D y) ys, \text{ipurge-ref } I D (D y) ys X)$   
 $\in \text{seq-comp-failures } P Q$   
**by** (*rule seq-comp-secure-aux-1-case-1 [OF A C D]*)

**next**  
**fix**  $X Y$   
**assume**  
 $xs @ y \# ys \in \text{sentences } P$  **and**  
 $(xs @ y \# ys, X) \in \text{failures } P$  **and**  
 $([], Y) \in \text{failures } Q$   
**thus**  $(xs @ \text{ipurge-tr } I D (D y) ys,$   
 $\text{ipurge-ref } I D (D y) ys (\text{insert None } X \cap Y)) \in \text{seq-comp-failures } P Q$   
**by** (*rule seq-comp-secure-aux-1-case-2 [OF A C D E]*)

**next**  
**fix**  $ws ys' Y$   
**assume**  
 $ws \in \text{sentences } P$  **and**  
 $(ys', Y) \in \text{failures } Q$  **and**  
 $ws @ ys' = xs @ y \# ys$

**thus** ( $xs @ ipurge\text{-}tr\ I\ D\ (D\ y)\ ys, ipurge\text{-}ref\ I\ D\ (D\ y)\ ys\ Y$ )  
 $\in seq\text{-}comp\text{-}failures\ P\ Q$   
**by** (rule *seq-comp-secure-aux-1-case-3* [*OF A B C D E*])  
**next**  
**fix**  $X\ Y$   
**assume**  
 $(xs @ ipurge\text{-}tr\ I\ D\ (D\ y)\ ys, ipurge\text{-}ref\ I\ D\ (D\ y)\ ys\ X)$   
 $\in seq\text{-}comp\text{-}failures\ P\ Q$  **and**  
 $(xs @ ipurge\text{-}tr\ I\ D\ (D\ y)\ ys, ipurge\text{-}ref\ I\ D\ (D\ y)\ ys\ Y)$   
 $\in seq\text{-}comp\text{-}failures\ P\ Q$   
**hence**  $(xs @ ipurge\text{-}tr\ I\ D\ (D\ y)\ ys,$   
 $ipurge\text{-}ref\ I\ D\ (D\ y)\ ys\ X \cup ipurge\text{-}ref\ I\ D\ (D\ y)\ ys\ Y)$   
 $\in seq\text{-}comp\text{-}failures\ P\ Q$   
**by** (rule *SCF-R4*)  
**thus** ( $xs @ ipurge\text{-}tr\ I\ D\ (D\ y)\ ys, ipurge\text{-}ref\ I\ D\ (D\ y)\ ys\ (X \cup Y)$ )  
 $\in seq\text{-}comp\text{-}failures\ P\ Q$   
**by** (*simp add: ipurge-ref-distrib-union*)  
**qed**

**lemma** *seq-comp-secure-1*:

**assumes**

*A*: *secure-termination I D* **and**

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

*C*: *sequential P* **and**

*D*: *secure P I D* **and**

*E*: *secure Q I D*

**shows**  $(xs @ y \# ys, Y) \in seq\text{-}comp\text{-}failures\ P\ Q \implies$

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

$\in seq\text{-}comp\text{-}failures\ P\ Q$

**by** (rule *seq-comp-secure-aux-1* [*OF A B C D E*, **where**  $ws = xs @ y \# ys$ ],  
*simp-all*)

This completes the proof that the former requirement for noninterference security is satisfied, so it is the turn of the latter one. Again, rule induction on set *seq-comp-failures* is applied, and the closure of the failures of a secure process under intransitive purge is used to meet the proof obligations arising from rule *SCF-R3*. In more detail, rule induction is applied to the trace into which the event is inserted, and then a case distinction is performed on the trace from which the event is extracted, using the expression of its refusal as union of a set of refusals derived previously.

**lemma** *seq-comp-secure-aux-2-case-1*:

**assumes**

*A*: *secure-termination I D* **and**

*B*: *sequential P* **and**

*C*: *secure P I D* **and**

*D*:  $xs @ zs \notin sentences\ P$  **and**



*E*:  $(xs @ zs, X) \in failures P$  **and**  
*F*:  $None \notin set\ xs$  **and**  
*G*:  $None \notin set\ zs$  **and**  
*H*:  $(xs @ [y], \{\}) \in seq-comp-failures P Q$   
**shows**  $(xs @ y \# ipurge-tr\ I\ D\ (D\ y)\ zs, ipurge-ref\ I\ D\ (D\ y)\ zs\ X)$   
 $\in seq-comp-failures P Q$

**proof** –

**have**  $\exists R. \{\} = (\bigcup n \in \{..length\ (xs @ [y])\}. \bigcup W \in R\ n. W) \wedge$   
 $(\forall W \in R\ 0.$   
 $xs @ [y] \notin sentences\ P \wedge None \notin set\ (xs @ [y]) \wedge$   
 $(xs @ [y], W) \in failures\ P \vee$   
 $xs @ [y] \in sentences\ P \wedge (\exists U\ V. (xs @ [y], U) \in failures\ P \wedge$   
 $([], V) \in failures\ Q \wedge W = insert\ None\ U \cap V)) \wedge$   
 $(\forall n \in \{0 < ..length\ (xs @ [y])\}. \forall W \in R\ n.$   
 $take\ (length\ (xs @ [y]) - n)\ (xs @ [y]) \in sentences\ P \wedge$   
 $(drop\ (length\ (xs @ [y]) - n)\ (xs @ [y]), W) \in failures\ Q) \wedge$   
 $(\exists n \in \{..length\ (xs @ [y])\}. \exists W. W \in R\ n)$   
 $(is\ \exists R. ?T\ R)$   
**using** *H* **by** (*rule seq-comp-refusals-1*)  
**then obtain** *R* **where** *I*:  $?T\ R\ ..$   
**hence**  $\exists n \in \{..length\ (xs @ [y])\}. \exists W. W \in R\ n$   
**by** *simp*  
**moreover have**  $\forall n \in \{0 < ..length\ (xs @ [y])\}. R\ n = \{\}$   
**proof** (*rule ballI, rule equalsOI*)  
**fix** *n W*  
**assume** *J*:  $n \in \{0 < ..length\ (xs @ [y])\}$   
**hence**  $\forall W \in R\ n. take\ (length\ (xs @ [y]) - n)\ (xs @ [y]) \in sentences\ P$   
**using** *I* **by** *simp*  
**moreover assume**  $W \in R\ n$   
**ultimately have**  $take\ (length\ (xs @ [y]) - n)\ (xs @ [y]) \in sentences\ P ..$   
**moreover have**  $take\ (length\ (xs @ [y]) - n)\ (xs @ [y]) =$   
 $take\ (length\ (xs @ [y]) - n)\ (xs @ zs)$   
**using** *J* **by** *simp*  
**ultimately have** *K*:  $take\ (length\ (xs @ [y]) - n)\ (xs @ zs) \in sentences\ P$   
**by** *simp*  
**show** *False*  
**proof** (*cases drop (length (xs @ [y]) - n) (xs @ zs)*)  
**case** *Nil*  
**hence**  $xs @ zs \in sentences\ P$   
**using** *K* **by** *simp*  
**thus** *False*  
**using** *D* **by** *contradiction*  
**next**  
**case** (*Cons v vs*)  
**moreover have**  $xs @ zs = take\ (length\ (xs @ [y]) - n)\ (xs @ zs) @$   
 $drop\ (length\ (xs @ [y]) - n)\ (xs @ zs)$   
**by** (*simp only: append-take-drop-id*)  
**ultimately have** *L*:  $xs @ zs =$   
 $take\ (length\ (xs @ [y]) - n)\ (xs @ zs) @ v \# vs$

by (*simp del: take-append*)  
**hence** ( $\text{take } (\text{length } (xs @ [y]) - n) (xs @ zs) @ v \# vs, X$ )  
 $\in \text{failures } P$   
**using**  $E$  by (*simp del: take-append*)  
**hence**  $\text{take } (\text{length } (xs @ [y]) - n) (xs @ zs) @ v \# vs \in \text{traces } P$   
**by** (*rule failures-traces*)  
**with**  $B$  and  $K$  **have**  $v = \text{None}$   
**by** (*rule seq-sentences-none*)  
**moreover** **have**  $\text{None} \notin \text{set } (xs @ zs)$   
**using**  $F$  and  $G$  **by** *simp*  
**hence**  $\text{None} \notin \text{set } (\text{take } (\text{length } (xs @ [y]) - n) (xs @ zs) @ v \# vs)$   
**by** (*subst (asm) L*)  
**hence**  $v \neq \text{None}$   
**by** (*rule-tac not-sym, simp*)  
**ultimately show** *False*  
**by** *contradiction*  
**qed**  
**qed**  
**ultimately have**  $\exists W. W \in R \ 0$   
**proof** *simp*  
**assume**  $\exists n \in \{.. \text{Suc } (\text{length } xs)\}. \exists W. W \in R \ n$   
**then obtain**  $n$  **where**  $J: n \in \{.. \text{Suc } (\text{length } xs)\}$  **and**  $K: \exists W. W \in R \ n ..$   
**assume**  $L: \forall n \in \{0 < .. \text{Suc } (\text{length } xs)\}. R \ n = \{\}$   
**show** *?thesis*  
**proof** (*cases n*)  
**case**  $0$   
**thus** *?thesis*  
**using**  $K$  **by** *simp*  
**next**  
**case** (*Suc m*)  
**obtain**  $W$  **where**  $W \in R \ n$   
**using**  $K ..$   
**moreover** **have**  $0 < n$   
**using** *Suc* **by** *simp*  
**hence**  $n \in \{0 < .. \text{Suc } (\text{length } xs)\}$   
**using**  $J$  **by** *simp*  
**with**  $L$  **have**  $R \ n = \{\} ..$   
**hence**  $W \notin R \ n$   
**by** (*rule equals0D*)  
**ultimately show** *?thesis*  
**by** *contradiction*  
**qed**  
**qed**  
**then obtain**  $W$  **where**  $J: W \in R \ 0 ..$   
**have**  $\forall W \in R \ 0.$   
 $xs @ [y] \notin \text{sentences } P \wedge$   
 $\text{None} \notin \text{set } (xs @ [y]) \wedge (xs @ [y], W) \in \text{failures } P \vee$   
 $xs @ [y] \in \text{sentences } P \wedge$   
 $(\exists U \ V. (xs @ [y], U) \in \text{failures } P \wedge ([], V) \in \text{failures } Q \wedge$

$W = \text{insert None } U \cap V$   
(is  $\forall W \in R \ 0. \ ?T \ W$ )  
**using**  $I$  **by** *simp*  
**hence**  $?T \ W$  **using**  $J \ ..$   
**hence**  $K: (xs \ @ \ [y], \ \{\}) \in \text{failures } P \wedge \text{None} \neq y$   
**proof** (*cases*  $xs \ @ \ [y] \in \text{sentences } P$ , *simp-all del: ex-simps*,  
*(erule-tac exE)+*, *(erule-tac [] conjE)+*, *simp-all*)  
**case** *False*  
**assume**  $(xs \ @ \ [y], \ W) \in \text{failures } P$   
**moreover have**  $\{\} \subseteq W \ ..$   
**ultimately show**  $(xs \ @ \ [y], \ \{\}) \in \text{failures } P$   
**by** (*rule process-rule-3*)  
**next**  
**fix**  $U$   
**case** *True*  
**assume**  $(xs \ @ \ [y], \ U) \in \text{failures } P$   
**moreover have**  $\{\} \subseteq U \ ..$   
**ultimately have**  $(xs \ @ \ [y], \ \{\}) \in \text{failures } P$   
**by** (*rule process-rule-3*)  
**moreover have** *weakly-sequential*  $P$   
**using**  $B$  **by** (*rule seq-implies-weakly-seq*)  
**hence**  $\text{None} \notin \text{set } (xs \ @ \ [y])$   
**using** *True* **by** (*rule weakly-seq-sentences-none*)  
**hence**  $\text{None} \neq y$   
**by** *simp*  
**ultimately show** *?thesis*  $..$   
**qed**  
**have**  $(zs, \ X) \in \text{futures } P \ xs$   
**using**  $E$  **by** (*simp add: futures-def*)  
**moreover have**  $([y], \ \{\}) \in \text{futures } P \ xs$   
**using**  $K$  **by** (*simp add: futures-def*)  
**ultimately have**  $(y \ \# \ \text{ipurge-tr } I \ D \ (D \ y) \ zs, \ \text{ipurge-ref } I \ D \ (D \ y) \ zs \ X) \in$   
*futures } P \ xs*  
**using**  $C$  **by** (*simp add: secure-def*)  
**hence**  $L: (xs \ @ \ y \ \# \ \text{ipurge-tr } I \ D \ (D \ y) \ zs, \ \text{ipurge-ref } I \ D \ (D \ y) \ zs \ X) \in$   
*failures } P*  
**by** (*simp add: futures-def*)  
**show** *?thesis*  
**proof** (*cases*  $xs \ @ \ y \ \# \ \text{ipurge-tr } I \ D \ (D \ y) \ zs \in \text{sentences } P$ ,  
*cases*  $(D \ y, \ D \ \text{None}) \in I \vee (\exists u \in \text{sinks } I \ D \ (D \ y) \ zs. (u, \ D \ \text{None}) \in I)$ ,  
*simp-all*)  
**assume**  $xs \ @ \ y \ \# \ \text{ipurge-tr } I \ D \ (D \ y) \ zs \notin \text{sentences } P$   
**thus** *?thesis* **using**  $L$   
**proof** (*rule SCF-R1*, *simp add: F K*)  
**have**  $\text{set } (\text{ipurge-tr } I \ D \ (D \ y) \ zs) \subseteq \text{set } zs$   
**by** (*rule ipurge-tr-set*)  
**thus**  $\text{None} \notin \text{set } (\text{ipurge-tr } I \ D \ (D \ y) \ zs)$   
**using**  $G$  **by** (*rule contra-subsetD*)  
**qed**

```

next
  assume
    M: xs @ y # ipurge-tr I D (D y) zs ∈ sentences P and
    N: (D y, D None) ∈ I ∨ (∃ u ∈ sinks I D (D y) zs. (u, D None) ∈ I)
  have ipurge-ref I D (D y) zs X = {}
  proof (rule disjE [OF N], erule-tac [2] bexE)
    assume O: (D y, D None) ∈ I
    show ?thesis
    proof (rule ipurge-ref-empty [of D y], simp)
      fix x
      have (D y, D None) ∈ I ∧ y ≠ None → (∀ u ∈ range D. (D y, u) ∈ I)
        using A by (simp add: secure-termination-def)
      moreover have y ≠ None
        using K by (rule-tac not-sym, simp)
      ultimately have ∀ u ∈ range D. (D y, u) ∈ I
        using O by simp
      thus (D y, D x) ∈ I
        by simp
    qed
  next
  fix u
  assume
    O: u ∈ sinks I D (D y) zs and
    P: (u, D None) ∈ I
  have ∃ z ∈ set zs. u = D z
    using O by (rule sinks-elem)
  then obtain z where Q: z ∈ set zs and R: u = D z ..
  have S: z ≠ None
  proof
    assume z = None
    hence None ∈ set zs
      using Q by simp
    thus False
      using G by contradiction
  qed
  show ?thesis
  proof (rule ipurge-ref-empty [of u], simp add: O)
    fix x
    have (D z, D None) ∈ I ∧ z ≠ None → (∀ v ∈ range D. (D z, v) ∈ I)
      using A by (simp add: secure-termination-def)
    moreover have (D z, D None) ∈ I
      using P and R by simp
    ultimately have ∀ v ∈ range D. (D z, v) ∈ I
      using S by simp
    thus (u, D x) ∈ I
      using R by simp
  qed
  qed
  thus ?thesis

```

**proof** *simp*  
**have**  $([], \{\}) \in \text{failures } Q$   
**by** (*rule process-rule-1*)  
**with**  $M$  **and**  $L$  **have**  $(xs @ y \# \text{ipurge-tr } I D (D y) zs,$   
 $\text{insert None (ipurge-ref } I D (D y) zs X) \cap \{\}) \in \text{seq-comp-failures } P Q$   
**by** (*rule SCF-R2*)  
**thus**  $(xs @ y \# \text{ipurge-tr } I D (D y) zs, \{\}) \in \text{seq-comp-failures } P Q$   
**by** *simp*  
**qed**  
**next**  
**assume**  
 $M: xs @ y \# \text{ipurge-tr } I D (D y) zs \in \text{sentences } P$  **and**  
 $N: (D y, D \text{None}) \notin I \wedge (\forall u \in \text{sinks } I D (D y) zs. (u, D \text{None}) \notin I)$   
**have**  $xs @ zs \in \text{sentences } P$   
**proof** (*simp add: sentences-def,*  
*rule ipurge-tr-del-traces [OF C, where u = D y], simp add: N)*  
**have**  $xs @ y \# \text{ipurge-tr } I D (D y) zs @ [\text{None}] \in \text{traces } P$   
**using**  $M$  **by** (*simp add: sentences-def*)  
**hence**  $(xs @ y \# \text{ipurge-tr } I D (D y) zs @ [\text{None}], \{\}) \in \text{failures } P$   
**by** (*rule traces-failures*)  
**hence**  $(y \# \text{ipurge-tr } I D (D y) zs @ [\text{None}], \{\}) \in \text{futures } P xs$   
**by** (*simp add: futures-def*)  
**hence**  $(\text{ipurge-tr } I D (D y) (\text{ipurge-tr } I D (D y) zs @ [\text{None}]},$   
 $\text{ipurge-ref } I D (D y) (\text{ipurge-tr } I D (D y) zs @ [\text{None}]) \{\}) \in \text{futures } P xs$   
**using**  $C$  **by** (*simp add: secure-def del: ipurge-tr.simps*)  
**hence**  $(xs @ \text{ipurge-tr } I D (D y) (\text{ipurge-tr } I D (D y) zs @ [\text{None}]), \{\})$   
 $\in \text{failures } P$   
**by** (*simp add: futures-def ipurge-ref-def*)  
**moreover** **have**  $\text{sinks } I D (D y) (\text{ipurge-tr } I D (D y) zs) = \{\}$   
**by** (*rule sinks-idem*)  
**hence**  $\neg ((D y, D \text{None}) \in I \vee$   
 $(\exists u \in \text{sinks } I D (D y) (\text{ipurge-tr } I D (D y) zs). (u, D \text{None}) \in I))$   
**using**  $N$  **by** *simp*  
**hence**  $D \text{None} \notin \text{sinks } I D (D y) (\text{ipurge-tr } I D (D y) zs @ [\text{None}])$   
**by** (*simp only: sinks-interference-eq, simp*)  
**ultimately** **have**  $(xs @ \text{ipurge-tr } I D (D y) (\text{ipurge-tr } I D (D y) zs)$   
 $@ [\text{None}], \{\}) \in \text{failures } P$   
**by** *simp*  
**hence**  $(xs @ \text{ipurge-tr } I D (D y) zs @ [\text{None}], \{\}) \in \text{failures } P$   
**by** (*simp add: ipurge-tr-idem*)  
**thus**  $xs @ \text{ipurge-tr } I D (D y) zs @ [\text{None}] \in \text{traces } P$   
**by** (*rule failures-traces*)  
**next**  
**show**  $xs @ zs \in \text{traces } P$   
**using**  $E$  **by** (*rule failures-traces*)  
**qed**  
**thus** *?thesis*  
**using**  $D$  **by** *contradiction*  
**qed**

qed

**lemma** *seq-comp-secure-aux-2-case-2*:

**assumes**

*A*: *secure-termination I D* **and**

*B*: *sequential P* **and**

*C*: *secure P I D* **and**

*D*: *secure Q I D* **and**

*E*:  $xs @ zs \in \text{sentences } P$  **and**

*F*:  $(xs @ zs, X) \in \text{failures } P$  **and**

*G*:  $([], Y) \in \text{failures } Q$  **and**

*H*:  $(xs @ [y], \{\}) \in \text{seq-comp-failures } P Q$

**shows**  $(xs @ y \# \text{ipurge-tr } I D (D y) zs,$

$\text{ipurge-ref } I D (D y) zs (\text{insert None } X \cap Y)) \in \text{seq-comp-failures } P Q$

**proof** –

**have**  $\exists R. \{\} = (\bigcup n \in \{..length (xs @ [y])\}. \bigcup W \in R n. W) \wedge$   
 $(\forall W \in R 0.$

$xs @ [y] \notin \text{sentences } P \wedge \text{None} \notin \text{set } (xs @ [y]) \wedge$

$(xs @ [y], W) \in \text{failures } P \vee$

$xs @ [y] \in \text{sentences } P \wedge (\exists U V. (xs @ [y], U) \in \text{failures } P \wedge$

$([], V) \in \text{failures } Q \wedge W = \text{insert None } U \cap V)) \wedge$

$(\forall n \in \{0 < ..length (xs @ [y])\}. \forall W \in R n.$

$\text{take } (length (xs @ [y]) - n) (xs @ [y]) \in \text{sentences } P \wedge$

$(\text{drop } (length (xs @ [y]) - n) (xs @ [y]), W) \in \text{failures } Q) \wedge$

$(\exists n \in \{..length (xs @ [y])\}. \exists W. W \in R n)$

$(\text{is } \exists R. ?T R)$

**using** *H* **by** (*rule seq-comp-refusals-1*)

**then obtain** *R* **where** *I*:  $?T R ..$

**hence**  $\exists n \in \{..length (xs @ [y])\}. \exists W. W \in R n$

**by** *simp*

**then obtain** *n* **where** *J*:  $n \in \{..length (xs @ [y])\}$  **and** *K*:  $\exists W. W \in R n ..$

**have** *weakly-sequential P*

**using** *B* **by** (*rule seq-implies-weakly-seq*)

**hence** *L*:  $\text{None} \notin \text{set } (xs @ zs)$

**using** *E* **by** (*rule weakly-seq-sentences-none*)

**have**  $n = 0 \vee n \in \{0 < ..length (xs @ [y])\}$

**using** *J* **by** *auto*

**thus** *?thesis*

**proof**

**assume**  $n = 0$

**hence**  $\exists W. W \in R 0$

**using** *K* **by** *simp*

**then obtain** *W* **where** *M*:  $W \in R 0 ..$

**have**  $\forall W \in R 0.$

$xs @ [y] \notin \text{sentences } P \wedge$

$\text{None} \notin \text{set } (xs @ [y]) \wedge (xs @ [y], W) \in \text{failures } P \vee$

$xs @ [y] \in \text{sentences } P \wedge$

$(\exists U V. (xs @ [y], U) \in \text{failures } P \wedge ( [], V) \in \text{failures } Q \wedge$

$W = \text{insert None } U \cap V)$

(is  $\forall W \in R \ 0. \ ?T \ W$ )  
 using  $I$  by *simp*  
 hence  $?T \ W$  using  $M \ ..$   
 hence  $N: (xs \ @ \ [y], \ \{\}) \in \text{failures } P \wedge \text{None} \notin \text{set } xs \wedge \text{None} \neq y$   
 proof (cases  $xs \ @ \ [y] \in \text{sentences } P$ , *simp-all del: ex-simps*,  
 (*erule-tac exE*)+, (*erule-tac [!] conjE*)+, *simp-all*)  
   case *False*  
     assume  $(xs \ @ \ [y], \ W) \in \text{failures } P$   
     moreover have  $\{\} \subseteq W \ ..$   
     ultimately show  $(xs \ @ \ [y], \ \{\}) \in \text{failures } P$   
     by (*rule process-rule-3*)  
 next  
 fix  $U$   
 case *True*  
 assume  $(xs \ @ \ [y], \ U) \in \text{failures } P$   
 moreover have  $\{\} \subseteq U \ ..$   
 ultimately have  $(xs \ @ \ [y], \ \{\}) \in \text{failures } P$   
 by (*rule process-rule-3*)  
 moreover have *weakly-sequential*  $P$   
 using  $B$  by (*rule seq-implies-weakly-seq*)  
 hence  $\text{None} \notin \text{set } (xs \ @ \ [y])$   
 using *True* by (*rule weakly-seq-sentences-none*)  
 hence  $\text{None} \notin \text{set } xs \wedge \text{None} \neq y$   
 by *simp*  
 ultimately show *?thesis* ..  
 qed  
 have  $(zs, \ X) \in \text{futures } P \ xs$   
 using  $F$  by (*simp add: futures-def*)  
 moreover have  $([y], \ \{\}) \in \text{futures } P \ xs$   
 using  $N$  by (*simp add: futures-def*)  
 ultimately have  $(y \ \# \ \text{ipurge-tr } I \ D \ (D \ y) \ zs, \ \text{ipurge-ref } I \ D \ (D \ y) \ zs \ X)$   
    $\in \text{futures } P \ xs$   
 using  $C$  by (*simp add: secure-def*)  
 hence  $O: (xs \ @ \ y \ \# \ \text{ipurge-tr } I \ D \ (D \ y) \ zs, \ \text{ipurge-ref } I \ D \ (D \ y) \ zs \ X)$   
    $\in \text{failures } P$   
 by (*simp add: futures-def*)  
 show *?thesis*  
 proof (cases  $xs \ @ \ y \ \# \ \text{ipurge-tr } I \ D \ (D \ y) \ zs \in \text{sentences } P$ ,  
*case-tac [2] (D y, D None) \in I \vee*  
 ( $\exists u \in \text{sinks } I \ D \ (D \ y) \ zs. (u, \ D \ \text{None}) \in I$ ),  
*simp-all*)  
   assume  $P: xs \ @ \ y \ \# \ \text{ipurge-tr } I \ D \ (D \ y) \ zs \in \text{sentences } P$   
   have  $\text{ipurge-ref } I \ D \ (D \ y) \ zs \ Y \subseteq Y$   
   by (*rule ipurge-ref-subset*)  
   with  $G$  have  $([], \ \text{ipurge-ref } I \ D \ (D \ y) \ zs \ Y) \in \text{failures } Q$   
   by (*rule process-rule-3*)  
   with  $P$  and  $O$  have  $(xs \ @ \ y \ \# \ \text{ipurge-tr } I \ D \ (D \ y) \ zs,$   
      $\text{insert } \text{None} \ (\text{ipurge-ref } I \ D \ (D \ y) \ zs \ X) \cap \text{ipurge-ref } I \ D \ (D \ y) \ zs \ Y)$   
      $\in \text{seq-comp-failures } P \ Q$

**by** (*rule SCF-R2*)  
**moreover have**  
 $ipurge-ref\ I\ D\ (D\ y)\ zs\ (insert\ None\ X) \cap ipurge-ref\ I\ D\ (D\ y)\ zs\ Y \subseteq$   
 $insert\ None\ (ipurge-ref\ I\ D\ (D\ y)\ zs\ X) \cap ipurge-ref\ I\ D\ (D\ y)\ zs\ Y$   
**proof** (*rule subsetI, simp del: insert-iff, erule conjE*)  
**fix**  $x$   
**have**  $ipurge-ref\ I\ D\ (D\ y)\ zs\ (insert\ None\ X) \subseteq$   
 $insert\ None\ (ipurge-ref\ I\ D\ (D\ y)\ zs\ X)$   
**by** (*rule ipurge-ref-subset-insert*)  
**moreover assume**  $x \in ipurge-ref\ I\ D\ (D\ y)\ zs\ (insert\ None\ X)$   
**ultimately show**  $x \in insert\ None\ (ipurge-ref\ I\ D\ (D\ y)\ zs\ X) ..$   
**qed**  
**ultimately have**  $(xs\ @\ y\ \# ipurge-tr\ I\ D\ (D\ y)\ zs,$   
 $ipurge-ref\ I\ D\ (D\ y)\ zs\ (insert\ None\ X) \cap ipurge-ref\ I\ D\ (D\ y)\ zs\ Y)$   
 $\in seq-comp-failures\ P\ Q$   
**by** (*rule seq-comp-prop-3*)  
**thus** *?thesis*  
**by** (*simp add: ipurge-ref-distrib-inter*)  
**next**  
**assume**  
 $P: xs\ @\ y\ \# ipurge-tr\ I\ D\ (D\ y)\ zs \notin sentences\ P$  **and**  
 $Q: (D\ y, D\ None) \in I \vee (\exists u \in sinks\ I\ D\ (D\ y)\ zs. (u, D\ None) \in I)$   
**have**  $ipurge-ref\ I\ D\ (D\ y)\ zs\ (insert\ None\ X \cap Y) = \{\}$   
**proof** (*rule disjE [OF Q], erule-tac [2] bexE*)  
**assume**  $R: (D\ y, D\ None) \in I$   
**show** *?thesis*  
**proof** (*rule ipurge-ref-empty [of D y], simp*)  
**fix**  $x$   
**have**  $(D\ y, D\ None) \in I \wedge y \neq None \longrightarrow (\forall u \in range\ D. (D\ y, u) \in I)$   
**using**  $A$  **by** (*simp add: secure-termination-def*)  
**moreover have**  $y \neq None$   
**using**  $N$  **by** (*rule-tac not-sym, simp*)  
**ultimately have**  $\forall u \in range\ D. (D\ y, u) \in I$   
**using**  $R$  **by** *simp*  
**thus**  $(D\ y, D\ None) \in I$   
**by** *simp*  
**qed**  
**next**  
**fix**  $u$   
**assume**  
 $R: u \in sinks\ I\ D\ (D\ y)\ zs$  **and**  
 $S: (u, D\ None) \in I$   
**have**  $\exists z \in set\ zs. u = D\ z$   
**using**  $R$  **by** (*rule sinks-elem*)  
**then obtain**  $z$  **where**  $T: z \in set\ zs$  **and**  $U: u = D\ z ..$   
**have**  $V: z \neq None$   
**proof**  
**assume**  $z = None$   
**hence**  $None \in set\ zs$



```

    using  $T$  by simp
  moreover have  $\text{None} \notin \text{set } zs$ 
    using  $L$  by simp
  ultimately show False
    by contradiction
qed
show ?thesis
proof (rule ipurge-ref-empty [of  $u$ ], simp add: R)
  fix  $x$ 
  have  $(D\ z, D\ \text{None}) \in I \wedge z \neq \text{None} \longrightarrow (\forall v \in \text{range } D. (D\ z, v) \in I)$ 
    using  $A$  by (simp add: secure-termination-def)
  moreover have  $(D\ z, D\ \text{None}) \in I$ 
    using  $S$  and  $U$  by simp
  ultimately have  $\forall v \in \text{range } D. (D\ z, v) \in I$ 
    using  $V$  by simp
  thus  $(u, D\ x) \in I$ 
    using  $U$  by simp
qed
qed
thus ?thesis
proof simp
  have  $\{\} \subseteq \text{ipurge-ref } I\ D\ (D\ y)\ zs\ X\ ..$ 
  with  $O$  have  $(xs\ @\ y\ \# \text{ipurge-tr } I\ D\ (D\ y)\ zs, \{\}) \in \text{failures } P$ 
    by (rule process-rule-3)
  with  $P$  show  $(xs\ @\ y\ \# \text{ipurge-tr } I\ D\ (D\ y)\ zs, \{\})$ 
     $\in \text{seq-comp-failures } P\ Q$ 
  proof (rule SCF-R1, simp add: N)
    have  $\text{set } (\text{ipurge-tr } I\ D\ (D\ y)\ zs) \subseteq \text{set } zs$ 
      by (rule ipurge-tr-set)
    moreover have  $\text{None} \notin \text{set } zs$ 
      using  $L$  by simp
    ultimately show  $\text{None} \notin \text{set } (\text{ipurge-tr } I\ D\ (D\ y)\ zs)$ 
      by (rule contra-subsetD)
  qed
qed
next
assume
   $P: xs\ @\ y\ \# \text{ipurge-tr } I\ D\ (D\ y)\ zs \notin \text{sentences } P$  and
   $Q: (D\ y, D\ \text{None}) \notin I \wedge (\forall u \in \text{sinks } I\ D\ (D\ y)\ zs. (u, D\ \text{None}) \notin I)$ 
have  $xs\ @\ zs\ @\ [\text{None}] \in \text{traces } P$ 
  using  $E$  by (simp add: sentences-def)
hence  $(xs\ @\ zs\ @\ [\text{None}], \{\}) \in \text{failures } P$ 
  by (rule traces-failures)
hence  $(zs\ @\ [\text{None}], \{\}) \in \text{futures } P\ xs$ 
  by (simp add: futures-def)
moreover have  $([y], \{\}) \in \text{futures } P\ xs$ 
  using  $N$  by (simp add: futures-def)
ultimately have  $(y\ \# \text{ipurge-tr } I\ D\ (D\ y)\ (zs\ @\ [\text{None}]),$ 
  ipurge-ref } I\ D\ (D\ y)\ (zs\ @\ [\text{None}])\ \{\}) \in \text{futures } P\ xs

```

(is  $(-, ?Z) \in -$ )  
 using  $C$  by (simp add: secure-def del: ipurge-tr.simps)  
 hence  $(xs @ y \# ipurge-tr I D (D y) (zs @ [None]), ?Z) \in failures P$   
 by (simp add: futures-def)  
 hence  $xs @ y \# ipurge-tr I D (D y) (zs @ [None]) \in traces P$   
 by (rule failures-traces)  
 moreover have  $\neg ((D y, D None) \in I \vee$   
 $(\exists u \in sinks I D (D y) zs. (u, D None) \in I))$   
 using  $Q$  by simp  
 hence  $D None \notin sinks I D (D y) (zs @ [None])$   
 by (simp only: sinks-interference-eq, simp)  
 ultimately have  $xs @ y \# ipurge-tr I D (D y) zs @ [None] \in traces P$   
 by simp  
 hence  $xs @ y \# ipurge-tr I D (D y) zs \in sentences P$   
 by (simp add: sentences-def)  
 thus ?thesis  
 using  $P$  by contradiction  
 qed  
 next  
 assume  $M: n \in \{0 < .. length (xs @ [y])\}$   
 have  $\forall n \in \{0 < .. length (xs @ [y])\}. \forall W \in R n.$   
 $take (length (xs @ [y]) - n) (xs @ [y]) \in sentences P \wedge$   
 $(drop (length (xs @ [y]) - n) (xs @ [y]), W) \in failures Q$   
 $(is \forall n \in -. \forall W \in -. ?T n W)$   
 using  $I$  by simp  
 hence  $\forall W \in R n. ?T n W$   
 using  $M$  ..  
 moreover obtain  $W$  where  $W \in R n$   
 using  $K$  ..  
 ultimately have  $N: ?T n W$  ..  
 moreover have  $O: take (length (xs @ [y]) - n) (xs @ [y]) =$   
 $take (length (xs @ [y]) - n) (xs @ zs)$   
 using  $M$  by simp  
 ultimately have  $P: take (length (xs @ [y]) - n) (xs @ zs) \in sentences P$   
 by simp  
 have  $Q: drop (length (xs @ [y]) - n) (xs @ zs) = []$   
 proof (cases drop (length (xs @ [y]) - n) (xs @ zs), simp)  
 case (Cons v vs)  
 moreover have  $xs @ zs = take (length (xs @ [y]) - n) (xs @ zs) @$   
 $drop (length (xs @ [y]) - n) (xs @ zs)$   
 by (simp only: append-take-drop-id)  
 ultimately have  $R: xs @ zs =$   
 $take (length (xs @ [y]) - n) (xs @ zs) @ v \# vs$   
 by (simp del: take-append)  
 hence  $(take (length (xs @ [y]) - n) (xs @ zs) @ v \# vs, X)$   
 $\in failures P$   
 using  $F$  by (simp del: take-append)  
 hence  $take (length (xs @ [y]) - n) (xs @ zs) @ v \# vs \in traces P$   
 by (rule failures-traces)

**with  $B$  and  $P$  have  $v = \text{None}$**   
**by** (*rule seq-sentences-none*)  
**moreover have**  
 $\text{None} \notin \text{set } (\text{take } (\text{length } (xs @ [y]) - n) (xs @ zs) @ v \# vs)$   
**using  $L$  by** (*subst (asm) R*)  
**hence  $v \neq \text{None}$**   
**by** (*rule-tac not-sym, simp*)  
**ultimately show** *?thesis*  
**by** *contradiction*  
**qed**  
**hence  $R$ :  $zs = []$**   
**using  $M$  by** *simp*  
**moreover have  $xs @ zs = \text{take } (\text{length } (xs @ [y]) - n) (xs @ zs) @$**   
 $\text{drop } (\text{length } (xs @ [y]) - n) (xs @ zs)$   
**by** (*simp only: append-take-drop-id*)  
**ultimately have  $\text{take } (\text{length } (xs @ [y]) - n) (xs @ zs) = xs$**   
**using  $Q$  by** *simp*  
**hence  $\text{take } (\text{length } (xs @ [y]) - n) (xs @ [y]) = xs$**   
**using  $O$  by** *simp*  
**moreover have  $xs @ [y] = \text{take } (\text{length } (xs @ [y]) - n) (xs @ [y]) @$**   
 $\text{drop } (\text{length } (xs @ [y]) - n) (xs @ [y])$   
**by** (*simp only: append-take-drop-id*)  
**ultimately have  $\text{drop } (\text{length } (xs @ [y]) - n) (xs @ [y]) = [y]$**   
**by** *simp*  
**hence  $S$ :  $([y], W) \in \text{failures } Q$**   
**using  $N$  by** *simp*  
**show** *?thesis using E and R*  
**proof** (*rule-tac SCF-R3, simp-all*)  
**have  $\forall xs\ y\ ys\ Y\ zs\ Z.$**   
 $(y \# ys, Y) \in \text{futures } Q\ xs \wedge (zs, Z) \in \text{futures } Q\ xs \longrightarrow$   
 $(\text{ipurge-tr } I\ D\ (D\ y)\ ys, \text{ipurge-ref } I\ D\ (D\ y)\ ys\ Y) \in \text{futures } Q\ xs \wedge$   
 $(y \# \text{ipurge-tr } I\ D\ (D\ y)\ zs, \text{ipurge-ref } I\ D\ (D\ y)\ zs\ Z) \in \text{futures } Q\ xs$   
**using  $D$  by** (*simp add: secure-def*)  
**hence  $([y], W) \in \text{futures } Q\ [] \wedge ([], Y) \in \text{futures } Q\ [] \longrightarrow$**   
 $(\text{ipurge-tr } I\ D\ (D\ y)\ [], \text{ipurge-ref } I\ D\ (D\ y)\ []\ W) \in \text{futures } Q\ [] \wedge$   
 $(y \# \text{ipurge-tr } I\ D\ (D\ y)\ [], \text{ipurge-ref } I\ D\ (D\ y)\ []\ Y) \in \text{futures } Q\ []$   
**by** *blast*  
**moreover have  $([y], W) \in \text{futures } Q\ []$**   
**using  $S$  by** (*simp add: futures-def*)  
**moreover have  $([], Y) \in \text{futures } Q\ []$**   
**using  $G$  by** (*simp add: futures-def*)  
**ultimately have  $([y], \text{ipurge-ref } I\ D\ (D\ y)\ []\ Y) \in \text{failures } Q$**   
 $(\text{is } (-, ?Y') \in -)$   
**by** (*simp add: futures-def*)  
**moreover have  $\text{ipurge-ref } I\ D\ (D\ y)\ []\ (\text{insert } \text{None } X) \cap ?Y' \subseteq ?Y'$**   
**by** *simp*  
**ultimately have  $([y], \text{ipurge-ref } I\ D\ (D\ y)\ []\ (\text{insert } \text{None } X) \cap ?Y')$**   
 $\in \text{failures } Q$   
**by** (*rule process-rule-3*)

**thus**  $([y], \text{ipurge-ref } I D (D y) [] (\text{insert None } X \cap Y)) \in \text{failures } Q$   
**by**  $(\text{simp add: ipurge-ref-distrib-inter})$   
**qed**  
**qed**  
**qed**

**lemma** *seq-comp-secure-aux-2-case-3*:

**assumes**

*A: secure-termination I D and*

*B: ref-union-closed P and*

*C: sequential P and*

*D: secure P I D and*

*E: secure Q I D and*

*F: ws ∈ sentences P and*

*G: (ys, Y) ∈ failures Q and*

*H: ys ≠ [] and*

*I: ws @ ys = xs @ zs and*

*J: (xs @ [y], {}) ∈ seq-comp-failures P Q*

**shows**  $(xs @ y \# \text{ipurge-tr } I D (D y) zs, \text{ipurge-ref } I D (D y) zs Y)$   
 $\in \text{seq-comp-failures } P Q$

**proof** –

**have**  $\exists R. \{\} = (\bigcup n \in \{..length (xs @ [y])\}. \bigcup W \in R n. W) \wedge$   
 $(\forall W \in R 0.$

$xs @ [y] \notin \text{sentences } P \wedge \text{None} \notin \text{set } (xs @ [y]) \wedge$

$(xs @ [y], W) \in \text{failures } P \vee$

$xs @ [y] \in \text{sentences } P \wedge (\exists U V. (xs @ [y], U) \in \text{failures } P \wedge$

$([], V) \in \text{failures } Q \wedge W = \text{insert None } U \cap V)) \wedge$

$(\forall n \in \{0 < ..length (xs @ [y])\}. \forall W \in R n.$

$\text{take } (length (xs @ [y]) - n) (xs @ [y]) \in \text{sentences } P \wedge$

$(\text{drop } (length (xs @ [y]) - n) (xs @ [y]), W) \in \text{failures } Q) \wedge$

$(\exists n \in \{..length (xs @ [y])\}. \exists W. W \in R n)$

$(\text{is } \exists R. ?T R)$

**using** *J* **by**  $(\text{rule seq-comp-refusals-1})$

**then obtain** *R* **where** *J*:  $?T R ..$

**hence**  $\exists n \in \{..length (xs @ [y])\}. \exists W. W \in R n$

**by** *simp*

**then obtain** *n* **where** *K*:  $n \in \{..length (xs @ [y])\}$  **and** *L*:  $\exists W. W \in R n ..$

**have** *M*:  $n = 0 \vee n \in \{0 < ..length (xs @ [y])\}$

**using** *K* **by** *auto*

**show** *?thesis*

**proof**  $(\text{cases } length xs < length ws)$

**case** *True*

**have**  $\forall W \in R 0.$

$xs @ [y] \notin \text{sentences } P \wedge$

$\text{None} \notin \text{set } (xs @ [y]) \wedge (xs @ [y], W) \in \text{failures } P \vee$

$xs @ [y] \in \text{sentences } P \wedge$

$(\exists U V. (xs @ [y], U) \in \text{failures } P \wedge ( [], V) \in \text{failures } Q \wedge$

$W = \text{insert None } U \cap V)$

$(\text{is } \forall W \in -. ?T W)$

using  $J$  by *simp*  
 moreover have  $n \notin \{0 <.. \text{length } (xs @ [y])\}$   
 proof  
 assume  $N: n \in \{0 <.. \text{length } (xs @ [y])\}$   
 hence  $\forall W \in R$   $n$ . take  $(\text{length } (xs @ [y]) - n) (xs @ [y]) \in \text{sentences } P$   
 using  $J$  by *simp*  
 moreover obtain  $W$  where  $W \in R$   $n$   
 using  $L$  ..  
 ultimately have take  $(\text{length } (xs @ [y]) - n) (xs @ [y]) \in \text{sentences } P$  ..  
 moreover have take  $(\text{length } (xs @ [y]) - n) (xs @ [y]) =$   
   take  $(\text{length } (xs @ [y]) - n) (xs @ zs)$   
 using  $N$  by *simp*  
 ultimately have take  $(\text{length } (xs @ [y]) - n) (xs @ zs) \in \text{sentences } P$   
 by *simp*  
 hence take  $(\text{length } (xs @ [y]) - n) (ws @ ys) \in \text{sentences } P$   
 using  $I$  by *simp*  
 moreover have  $\text{length } (xs @ [y]) - n \leq \text{length } xs$   
 using  $N$  by (*simp, arith*)  
 hence  $O: \text{length } (xs @ [y]) - n < \text{length } ws$   
 using *True* by *simp*  
 ultimately have  $P: \text{take } (\text{length } (xs @ [y]) - n) ws \in \text{sentences } P$   
 by *simp*  
 show *False*  
 proof (*cases drop (length (xs @ [y]) - n) ws*)  
   case *Nil*  
   thus *False*  
   using  $O$  by *simp*  
 next  
   case (*Cons v ws*)  
   moreover have  $ws = \text{take } (\text{length } (xs @ [y]) - n) ws @$   
      $\text{drop } (\text{length } (xs @ [y]) - n) ws$   
   by *simp*  
   ultimately have  $Q: ws = \text{take } (\text{length } (xs @ [y]) - n) ws @ v \# ws$   
   by *simp*  
   hence take  $(\text{length } (xs @ [y]) - n) ws @ v \# ws \in \text{sentences } P$   
   using  $F$  by *simp*  
   hence  $(\text{take } (\text{length } (xs @ [y]) - n) ws @ v \# ws) @ [None] \in \text{traces } P$   
   by (*simp add: sentences-def*)  
   hence take  $(\text{length } (xs @ [y]) - n) ws @ v \# ws \in \text{traces } P$   
   by (*rule process-rule-2-traces*)  
   with  $C$  and  $P$  have  $v = None$   
   by (*rule seq-sentences-none*)  
   moreover have *weakly-sequential*  $P$   
   using  $C$  by (*rule seq-implies-weakly-seq*)  
   hence  $None \notin \text{set } ws$   
   using  $F$  by (*rule weakly-seq-sentences-none*)  
   hence  $None \notin \text{set } (\text{take } (\text{length } (xs @ [y]) - n) ws @ v \# ws)$   
   by (*subst (asm) Q*)  
   hence  $v \neq None$

```

    by (rule-tac not-sym, simp)
  ultimately show False
    by contradiction
qed
hence  $n = 0$ 
  using  $M$  by blast
hence  $\exists W. W \in R \ 0$ 
  using  $L$  by simp
then obtain  $W$  where  $W \in R \ 0 \ ..$ 
ultimately have  $?T \ W \ ..$ 
hence  $N: (xs \ @ \ [y], \ \{\}) \in failures \ P \ \wedge \ None \notin set \ xs \ \wedge \ None \neq y$ 
proof (cases  $xs \ @ \ [y] \in sentences \ P$ , simp-all del: ex-simps,
  (erule-tac exE)+, (erule-tac [!] conjE)+, simp-all)
  case False
    assume  $(xs \ @ \ [y], W) \in failures \ P$ 
    moreover have  $\{\} \subseteq W \ ..$ 
    ultimately show  $(xs \ @ \ [y], \ \{\}) \in failures \ P$ 
      by (rule process-rule-3)
  next
    fix  $U$ 
    case True
    assume  $(xs \ @ \ [y], U) \in failures \ P$ 
    moreover have  $\{\} \subseteq U \ ..$ 
    ultimately have  $(xs \ @ \ [y], \ \{\}) \in failures \ P$ 
      by (rule process-rule-3)
    moreover have weakly-sequential  $P$ 
      using  $C$  by (rule seq-implies-weakly-seq)
    hence  $None \notin set \ (xs \ @ \ [y])$ 
      using True by (rule weakly-seq-sentences-none)
    hence  $None \notin set \ xs \ \wedge \ None \neq y$ 
      by simp
    ultimately show  $?thesis \ ..$ 
qed
have  $drop \ (length \ xs) \ (xs \ @ \ zs) = drop \ (length \ xs) \ (ws \ @ \ ys)$ 
  using  $I$  by simp
hence  $O: zs = drop \ (length \ xs) \ ws \ @ \ ys$ 
  (is  $- = ?ws' \ @ \ -$ )
  using True by simp
let  $?U = insert \ (D \ y) \ (sinks \ I \ D \ (D \ y) \ ?ws')$ 
have  $ipurge-tr \ I \ D \ (D \ y) \ zs =$ 
   $ipurge-tr \ I \ D \ (D \ y) \ ?ws' \ @ \ ipurge-tr-aux \ I \ D \ ?U \ ys$ 
  using  $O$  by (simp add: ipurge-tr-append)
moreover have  $ipurge-ref \ I \ D \ (D \ y) \ zs \ Y = ipurge-ref-aux \ I \ D \ ?U \ ys \ Y$ 
  using  $O$  by (simp add: ipurge-ref-append)
ultimately show  $?thesis$ 
proof (cases  $xs \ @ \ y \ \# \ ipurge-tr \ I \ D \ (D \ y) \ ?ws' \in sentences \ P$ , simp-all)
  assume  $P: xs \ @ \ y \ \# \ ipurge-tr \ I \ D \ (D \ y) \ ?ws' \in sentences \ P$ 
  have  $Q: (ipurge-tr-aux \ I \ D \ ?U \ ys, ipurge-ref-aux \ I \ D \ ?U \ ys \ Y) \in failures \ Q$ 

```

**using**  $E$  **and**  $G$  **by** (rule *ipurge-tr-ref-aux-failures*)  
**show**  $(xs @ y \# \text{ipurge-tr } I D (D y) ?ws' @ \text{ipurge-tr-aux } I D ?U ys,$   
 $\text{ipurge-ref-aux } I D ?U ys Y) \in \text{seq-comp-failures } P Q$   
**proof** (cases *ipurge-tr-aux*  $I D ?U ys$ )  
**case**  $Nil$   
**have**  $(xs @ y \# \text{ipurge-tr } I D (D y) ?ws', \{x. x \neq None\}) \in \text{failures } P$   
**using**  $B$  **and**  $C$  **and**  $P$  **by** (rule *seq-sentences-ref*)  
**moreover have**  $([], \text{ipurge-ref-aux } I D ?U ys Y) \in \text{failures } Q$   
**using**  $Q$  **and**  $Nil$  **by** *simp*  
**ultimately have**  $(xs @ y \# \text{ipurge-tr } I D (D y) ?ws',$   
 $\text{insert } None \{x. x \neq None\} \cap \text{ipurge-ref-aux } I D ?U ys Y)$   
 $\in \text{seq-comp-failures } P Q$   
**by** (rule *SCF-R2* [*OF P*])  
**moreover have**  $\text{insert } None \{x. x \neq None\} \cap$   
 $\text{ipurge-ref-aux } I D ?U ys Y = \text{ipurge-ref-aux } I D ?U ys Y$   
**by** *blast*  
**ultimately show** *?thesis*  
**using**  $Nil$  **by** *simp*  
**next**  
**case**  $Cons$   
**hence**  $\text{ipurge-tr-aux } I D ?U ys \neq []$   
**by** *simp*  
**with**  $P$  **and**  $Q$  **have**  
 $((xs @ y \# \text{ipurge-tr } I D (D y) ?ws') @ \text{ipurge-tr-aux } I D ?U ys,$   
 $\text{ipurge-ref-aux } I D ?U ys Y) \in \text{seq-comp-failures } P Q$   
**by** (rule *SCF-R3*)  
**thus** *?thesis*  
**by** *simp*  
**qed**  
**next**  
**assume**  $P: xs @ y \# \text{ipurge-tr } I D (D y) ?ws' \notin \text{sentences } P$   
**have**  $ws = \text{take } (\text{length } xs) ws @ ?ws'$   
**by** *simp*  
**moreover have**  $\text{take } (\text{length } xs) (ws @ ys) = \text{take } (\text{length } xs) (xs @ zs)$   
**using**  $I$  **by** *simp*  
**hence**  $\text{take } (\text{length } xs) ws = xs$   
**using**  $True$  **by** *simp*  
**ultimately have**  $xs @ ?ws' \in \text{sentences } P$   
**using**  $F$  **by** *simp*  
**hence**  $xs @ ?ws' @ [None] \in \text{traces } P$   
**by** (*simp add: sentences-def*)  
**hence**  $(xs @ ?ws' @ [None], \{\}) \in \text{failures } P$   
**by** (rule *traces-failures*)  
**hence**  $(?ws' @ [None], \{\}) \in \text{futures } P xs$   
**by** (*simp add: futures-def*)  
**moreover have**  $([y], \{\}) \in \text{futures } P xs$   
**using**  $N$  **by** (*simp add: futures-def*)  
**ultimately have**  $(y \# \text{ipurge-tr } I D (D y) (?ws' @ [None]),$   
 $\text{ipurge-ref } I D (D y) (?ws' @ [None]) \{\}) \in \text{futures } P xs$

using  $D$  by (simp add: secure-def del: ipurge-tr.simps)  
 hence  $Q: (xs @ y \# ipurge\text{-}tr\ I\ D\ (D\ y)\ (?ws' @ [None]), \{\}) \in failures\ P$   
 by (simp add: futures-def ipurge-ref-def)  
 have  $set\ ?ws' \subseteq set\ ws$   
 by (rule set-drop-subset)  
 moreover have weakly-sequential  $P$   
 using  $C$  by (rule seq-implies-weakly-seq)  
 hence  $None \notin set\ ws$   
 using  $F$  by (rule weakly-seq-sentences-none)  
 ultimately have  $R: None \notin set\ ?ws'$   
 by (rule contra-subsetD)  
 show  $(xs @ y \# ipurge\text{-}tr\ I\ D\ (D\ y)\ ?ws' @ ipurge\text{-}tr\ aux\ I\ D\ ?U\ ys,$   
 $ipurge\text{-}ref\ aux\ I\ D\ ?U\ ys\ Y) \in seq\text{-}comp\text{-}failures\ P\ Q$   
 proof (cases  $(D\ y, D\ None) \in I \vee$   
 $(\exists u \in sinks\ I\ D\ (D\ y)\ ?ws'. (u, D\ None) \in I)$ )  
 assume  $S: (D\ y, D\ None) \in I \vee$   
 $(\exists u \in sinks\ I\ D\ (D\ y)\ ?ws'. (u, D\ None) \in I)$   
 have  $ipurge\text{-}tr\ aux\ I\ D\ ?U\ ys = []$   
 proof (rule disjE [OF  $S$ ], erule-tac [2] bexE)  
 assume  $T: (D\ y, D\ None) \in I$   
 show ?thesis  
 proof (rule ipurge-tr-aux-nil [of  $D\ y$ ], simp)  
 fix  $x$   
 have  $(D\ y, D\ None) \in I \wedge y \neq None \longrightarrow (\forall u \in range\ D. (D\ y, u) \in I)$   
 using  $A$  by (simp add: secure-termination-def)  
 moreover have  $y \neq None$   
 using  $N$  by (rule-tac not-sym, simp)  
 ultimately have  $\forall u \in range\ D. (D\ y, u) \in I$   
 using  $T$  by simp  
 thus  $(D\ y, D\ x) \in I$   
 by simp  
 qed  
 next  
 fix  $u$   
 assume  
 $T: u \in sinks\ I\ D\ (D\ y)\ ?ws'$  and  
 $U: (u, D\ None) \in I$   
 have  $\exists w \in set\ ?ws'. u = D\ w$   
 using  $T$  by (rule sinks-elem)  
 then obtain  $w$  where  $V: w \in set\ ?ws'$  and  $W: u = D\ w ..$   
 have  $X: w \neq None$   
 proof  
 assume  $w = None$   
 hence  $None \in set\ ?ws'$   
 using  $V$  by simp  
 moreover have  $None \notin set\ ?ws'$   
 using  $R$  by simp  
 ultimately show  $False$   
 by contradiction



```

qed
show ?thesis
proof (rule ipurge-tr-aux-nil [of u], simp add: T)
  fix x
  have  $(D\ w, D\ None) \in I \wedge w \neq None \longrightarrow$ 
     $(\forall v \in \text{range } D. (D\ w, v) \in I)$ 
  using A by (simp add: secure-termination-def)
  moreover have  $(D\ w, D\ None) \in I$ 
  using U and W by simp
  ultimately have  $\forall v \in \text{range } D. (D\ w, v) \in I$ 
  using X by simp
  thus  $(u, D\ x) \in I$ 
  using W by simp
qed
qed
moreover have ipurge-ref-aux I D ?U ys Y = {}
proof (rule disjE [OF S], erule-tac [2] bexE)
  assume T:  $(D\ y, D\ None) \in I$ 
  show ?thesis
  proof (rule ipurge-ref-aux-empty [of D y])
    have  $?U \subseteq \text{sinks-aux } I\ D\ ?U\ ys$ 
    by (rule sinks-aux-subset)
    moreover have  $D\ y \in ?U$ 
    by simp
    ultimately show  $D\ y \in \text{sinks-aux } I\ D\ ?U\ ys ..$ 
  next
  fix x
  have  $(D\ y, D\ None) \in I \wedge y \neq None \longrightarrow (\forall u \in \text{range } D. (D\ y, u) \in I)$ 
  using A by (simp add: secure-termination-def)
  moreover have  $y \neq None$ 
  using N by (rule-tac not-sym, simp)
  ultimately have  $\forall u \in \text{range } D. (D\ y, u) \in I$ 
  using T by simp
  thus  $(D\ y, D\ x) \in I$ 
  by simp
qed
next
fix u
assume
  T:  $u \in \text{sinks } I\ D\ (D\ y)\ ?ws'$  and
  U:  $(u, D\ None) \in I$ 
have  $\exists w \in \text{set } ?ws'. u = D\ w$ 
using T by (rule sinks-elem)
then obtain w where V:  $w \in \text{set } ?ws'$  and W:  $u = D\ w ..$ 
have X:  $w \neq None$ 
proof
  assume  $w = None$ 
  hence  $None \in \text{set } ?ws'$ 
  using V by simp

```

```

    moreover have  $None \notin \text{set } ?ws'$ 
    using  $R$  by simp
    ultimately show False
    by contradiction
  qed
  show ?thesis
  proof (rule ipurge-ref-aux-empty [of  $u$ ])
    have  $?U \subseteq \text{sinks-aux } I D ?U ys$ 
    by (rule sinks-aux-subset)
    moreover have  $u \in ?U$ 
    using  $T$  by simp
    ultimately show  $u \in \text{sinks-aux } I D ?U ys ..$ 
  next
  fix  $x$ 
  have  $(D w, D None) \in I \wedge w \neq None \longrightarrow$ 
     $(\forall v \in \text{range } D. (D w, v) \in I)$ 
  using  $A$  by (simp add: secure-termination-def)
  moreover have  $(D w, D None) \in I$ 
  using  $U$  and  $W$  by simp
  ultimately have  $\forall v \in \text{range } D. (D w, v) \in I$ 
  using  $X$  by simp
  thus  $(u, D x) \in I$ 
  using  $W$  by simp
  qed
  qed
  ultimately show ?thesis
  proof simp
    have  $D None \in \text{sinks } I D (D y) (?ws' @ [None])$ 
    using  $S$  by (simp only: sinks-interference-eq)
    hence  $(xs @ y \# \text{ipurge-tr } I D (D y) ?ws', \{\}) \in \text{failures } P$ 
    using  $Q$  by simp
    moreover have  $None \notin \text{set } (xs @ y \# \text{ipurge-tr } I D (D y) ?ws')$ 
    proof (simp add: N)
      have  $\text{set } (\text{ipurge-tr } I D (D y) ?ws') \subseteq \text{set } ?ws'$ 
      by (rule ipurge-tr-set)
      thus  $None \notin \text{set } (\text{ipurge-tr } I D (D y) ?ws')$ 
      using  $R$  by (rule contra-subsetD)
    qed
    ultimately show  $(xs @ y \# \text{ipurge-tr } I D (D y) ?ws', \{\})$ 
       $\in \text{seq-comp-failures } P Q$ 
    by (rule SCF-R1 [OF  $P$ ])
  qed
  next
  assume  $\neg ((D y, D None) \in I \vee$ 
     $(\exists u \in \text{sinks } I D (D y) ?ws'. (u, D None) \in I))$ 
  hence  $D None \notin \text{sinks } I D (D y) (?ws' @ [None])$ 
  by (simp only: sinks-interference-eq, simp)
  hence  $(xs @ y \# \text{ipurge-tr } I D (D y) ?ws' @ [None], \{\}) \in \text{failures } P$ 
  using  $Q$  by simp

```

```

hence  $xs @ y \# \text{ipurge-tr } I D (D y) ?ws' @ [None] \in \text{traces } P$ 
  by (rule failures-traces)
hence  $xs @ y \# \text{ipurge-tr } I D (D y) ?ws' \in \text{sentences } P$ 
  by (simp add: sentences-def)
thus ?thesis
  using  $P$  by contradiction
qed
qed
next
case False
have  $\forall n \in \{0 < .. \text{length } (xs @ [y])\}. \forall W \in R n.$ 
  take  $(\text{length } (xs @ [y]) - n) (xs @ [y]) \in \text{sentences } P \wedge$ 
   $(\text{drop } (\text{length } (xs @ [y]) - n) (xs @ [y]), W) \in \text{failures } Q$ 
  (is  $\forall n \in -. \forall W \in -. ?T n W)$ 
  using  $J$  by simp
moreover have  $n \neq 0$ 
proof
  have  $\forall W \in R 0.$ 
     $xs @ [y] \notin \text{sentences } P \wedge$ 
     $None \notin \text{set } (xs @ [y]) \wedge (xs @ [y], W) \in \text{failures } P \vee$ 
     $xs @ [y] \in \text{sentences } P \wedge$ 
     $(\exists U V. (xs @ [y], U) \in \text{failures } P \wedge ([], V) \in \text{failures } Q \wedge$ 
     $W = \text{insert } None U \cap V)$ 
    (is  $\forall W \in -. ?T' W)$ 
    using  $J$  by blast
  moreover assume  $n = 0$ 
  hence  $\exists W. W \in R 0$ 
  using  $L$  by simp
  then obtain  $W$  where  $W \in R 0 ..$ 
  ultimately have  $?T' W ..$ 
  hence  $N: xs @ [y] \in \text{traces } P \wedge None \notin \text{set } (xs @ [y])$ 
  proof (cases  $xs @ [y] \in \text{sentences } P$ , simp-all del: ex-simps,
  (erule-tac exE)+, (erule-tac  $[\!]$  conjE)+, simp-all)
  case False
  assume  $(xs @ [y], W) \in \text{failures } P$ 
  moreover have  $\{\} \subseteq W ..$ 
  ultimately have  $(xs @ [y], \{\}) \in \text{failures } P$ 
  by (rule process-rule-3)
  thus  $xs @ [y] \in \text{traces } P$ 
  by (rule failures-traces)
next
fix  $U$ 
case True
assume  $(xs @ [y], U) \in \text{failures } P$ 
moreover have  $\{\} \subseteq U ..$ 
ultimately have  $(xs @ [y], \{\}) \in \text{failures } P$ 
by (rule process-rule-3)
hence  $xs @ [y] \in \text{traces } P$ 
by (rule failures-traces)

```

**moreover have** *weakly-sequential P*  
**using** *C* **by** (*rule seq-implies-weakly-seq*)  
**hence**  $\text{None} \notin \text{set } (xs @ [y])$   
**using** *True* **by** (*rule weakly-seq-sentences-none*)  
**hence**  $\text{None} \neq y \wedge \text{None} \notin \text{set } xs$   
**by** *simp*  
**ultimately show**  $xs @ [y] \in \text{traces } P \wedge \text{None} \neq y \wedge \text{None} \notin \text{set } xs ..$   
**qed**  
**have**  $\text{take } (\text{length } xs) (xs @ zs) @ [y] = \text{take } (\text{length } xs) (ws @ ys) @ [y]$   
**using** *I* **by** *simp*  
**hence**  $xs @ [y] = ws @ \text{take } (\text{length } xs - \text{length } ws) ys @ [y]$   
**using** *False* **by** *simp*  
**moreover have**  $\exists v \text{ vs. } \text{take } (\text{length } xs - \text{length } ws) ys @ [y] = v \# vs$   
**by** (*cases take (length xs - length ws) ys @ [y], simp-all*)  
**then obtain** *v* **and** *vs* **where**  
 $\text{take } (\text{length } xs - \text{length } ws) ys @ [y] = v \# vs$   
**by** *blast*  
**ultimately have**  $O: xs @ [y] = ws @ v \# vs$   
**by** *simp*  
**hence**  $ws @ v \# vs \in \text{traces } P$   
**using** *N* **by** *simp*  
**with** *C* **and** *F* **have**  $v = \text{None}$   
**by** (*rule seq-sentences-none*)  
**moreover have**  $v \neq \text{None}$   
**using** *N* **and** *O* **by** (*rule-tac not-sym, simp*)  
**ultimately show** *False*  
**by** *contradiction*  
**qed**  
**hence**  $N: n \in \{0 < .. \text{length } (xs @ [y])\}$   
**using** *M* **by** *blast*  
**ultimately have**  $\forall W \in R \text{ n. } ?T \text{ n } W ..$   
**moreover obtain** *W* **where**  $W \in R \text{ n}$   
**using** *L* ..  
**ultimately have**  $O: ?T \text{ n } W ..$   
**have**  $P: \text{length } (xs @ [y]) - n \leq \text{length } xs$   
**using** *N* **by** (*simp, arith*)  
**have**  $\text{length } (xs @ [y]) - n = \text{length } ws$   
**proof** (*rule ccontr, simp only: neq-iff, erule disjE*)  
**assume**  $Q: \text{length } (xs @ [y]) - n < \text{length } ws$   
**moreover have**  $ws = \text{take } (\text{length } (xs @ [y]) - n) ws @$   
 $\text{drop } (\text{length } (xs @ [y]) - n) ws$   
 $(\text{is } - = - @ ?ws')$   
**by** *simp*  
**ultimately have**  $ws = \text{take } (\text{length } (xs @ [y]) - n) (ws @ ys) @ ?ws'$   
**by** *simp*  
**hence**  $ws = \text{take } (\text{length } (xs @ [y]) - n) (xs @ zs) @ ?ws'$   
**using** *I* **by** *simp*  
**hence**  $ws = \text{take } (\text{length } (xs @ [y]) - n) (xs @ [y]) @ ?ws'$   
**using** *P* **by** *simp*

**moreover have**  $?ws' \neq []$   
**using**  $Q$  **by** *simp*  
**hence**  $\exists v \text{ vs. } ?ws' = v \# \text{ vs}$   
**by** (*cases ?ws', simp-all*)  
**then obtain**  $v$  **and**  $\text{vs}$  **where**  $?ws' = v \# \text{ vs}$   
**by** *blast*  
**ultimately have**  $S: ws = \text{take } (\text{length } (xs @ [y]) - n) (xs @ [y]) @ v \# \text{ vs}$   
**by** *simp*  
**hence**  $(\text{take } (\text{length } (xs @ [y]) - n) (xs @ [y]) @ v \# \text{ vs}) @ [None]$   
 $\in \text{traces } P$   
**using**  $F$  **by** (*simp add: sentences-def*)  
**hence**  $T: \text{take } (\text{length } (xs @ [y]) - n) (xs @ [y]) @ v \# \text{ vs} \in \text{traces } P$   
**by** (*rule process-rule-2-traces*)  
**have**  $\text{take } (\text{length } (xs @ [y]) - n) (xs @ [y]) \in \text{sentences } P$   
**using**  $O ..$   
**with**  $C$  **have**  $v = None$   
**using**  $T$  **by** (*rule seq-sentences-none*)  
**moreover have** *weakly-sequential*  $P$   
**using**  $C$  **by** (*rule seq-implies-weakly-seq*)  
**hence**  $None \notin \text{set } ws$   
**using**  $F$  **by** (*rule weakly-seq-sentences-none*)  
**hence**  $v \neq None$   
**using**  $S$  **by** (*rule-tac not-sym, simp*)  
**ultimately show** *False*  
**by** *contradiction*  
**next**  
**assume**  $Q: \text{length } ws < \text{length } (xs @ [y]) - n$   
**have**  $\text{take } (\text{length } (xs @ [y]) - n) (xs @ [y]) =$   
 $\text{take } (\text{length } (xs @ [y]) - n) (xs @ zs)$   
**using**  $P$  **by** *simp*  
**also have**  $\dots = \text{take } (\text{length } (xs @ [y]) - n) (ws @ ys)$   
**using**  $I$  **by** *simp*  
**also have**  $\dots = \text{take } (\text{length } (xs @ [y]) - n) ws @$   
 $\text{take } (\text{length } (xs @ [y]) - n - \text{length } ws) ys$   
 $(\text{is } - = - @ ?ys')$   
**by** *simp*  
**also have**  $\dots = ws @ ?ys'$   
**using**  $Q$  **by** *simp*  
**finally have**  $\text{take } (\text{length } (xs @ [y]) - n) (xs @ [y]) = ws @ ?ys' .$   
**moreover have**  $?ys' \neq []$   
**using**  $Q$  **and**  $H$  **by** *simp*  
**hence**  $\exists v \text{ vs. } ?ys' = v \# \text{ vs}$   
**by** (*cases ?ys', simp-all*)  
**then obtain**  $v$  **and**  $\text{vs}$  **where**  $?ys' = v \# \text{ vs}$   
**by** *blast*  
**ultimately have**  $S: \text{take } (\text{length } (xs @ [y]) - n) (xs @ [y]) = ws @ v \# \text{ vs}$   
**by** *simp*  
**hence**  $(ws @ v \# \text{ vs}) @ [None] \in \text{traces } P$   
**using**  $O$  **by** (*simp add: sentences-def*)

**hence**  $ws @ v \# vs \in \text{traces } P$   
**by** (rule process-rule-2-traces)  
**with**  $C$  and  $F$  **have**  $T: v = \text{None}$   
**by** (rule seq-sentences-none)  
**have** weakly-sequential  $P$   
**using**  $C$  **by** (rule seq-implies-weakly-seq)  
**moreover** **have**  $\text{take } (\text{length } (xs @ [y]) - n) (xs @ [y]) \in \text{sentences } P$   
**using**  $O ..$   
**ultimately** **have**  $\text{None} \notin \text{set } (\text{take } (\text{length } (xs @ [y]) - n) (xs @ [y]))$   
**by** (rule weakly-seq-sentences-none)  
**hence**  $v \neq \text{None}$   
**using**  $S$  **by** (rule-tac not-sym, simp)  
**thus**  $\text{False}$   
**using**  $T$  **by** contradiction  
**qed**  
**hence**  $(\text{drop } (\text{length } ws) (xs @ [y]), W) \in \text{failures } Q$   
**using**  $O$  **by** simp  
**hence**  $(\text{drop } (\text{length } ws) xs @ [y], W) \in \text{failures } Q$   
(is  $(?xs' @ -, -) \in -$ )  
**using**  $\text{False}$  **by** simp  
**hence**  $([y], W) \in \text{futures } Q ?xs'$   
**by** (simp add: futures-def)  
**moreover** **have**  $\text{drop } (\text{length } ws) (ws @ ys) = \text{drop } (\text{length } ws) (xs @ zs)$   
**using**  $I$  **by** simp  
**hence**  $ys = ?xs' @ zs$   
**using**  $\text{False}$  **by** simp  
**hence**  $(?xs' @ zs, Y) \in \text{failures } Q$   
**using**  $G$  **by** simp  
**hence**  $(zs, Y) \in \text{futures } Q ?xs'$   
**by** (simp add: futures-def)  
**ultimately** **have**  $(y \# \text{ipurge-tr } I D (D y) zs, \text{ipurge-ref } I D (D y) zs Y)$   
 $\in \text{futures } Q ?xs'$   
**using**  $E$  **by** (simp add: secure-def)  
**hence**  $(?xs' @ y \# \text{ipurge-tr } I D (D y) zs, \text{ipurge-ref } I D (D y) zs Y)$   
 $\in \text{failures } Q$   
**by** (simp add: futures-def)  
**moreover** **have**  $?xs' @ y \# \text{ipurge-tr } I D (D y) zs \neq []$   
**by** simp  
**ultimately** **have**  $(ws @ ?xs' @ y \# \text{ipurge-tr } I D (D y) zs,$   
 $\text{ipurge-ref } I D (D y) zs Y) \in \text{seq-comp-failures } P Q$   
**by** (rule SCF-R3 [OF F])  
**hence**  $((ws @ ?xs') @ y \# \text{ipurge-tr } I D (D y) zs,$   
 $\text{ipurge-ref } I D (D y) zs Y) \in \text{seq-comp-failures } P Q$   
**by** simp  
**moreover** **have**  $xs = \text{take } (\text{length } ws) xs @ ?xs'$   
**by** simp  
**hence**  $xs = \text{take } (\text{length } ws) (xs @ zs) @ ?xs'$   
**using**  $\text{False}$  **by** simp  
**hence**  $xs = \text{take } (\text{length } ws) (ws @ ys) @ ?xs'$

**using**  $I$  **by** *simp*  
**hence**  $xs = ws @ ?xs'$   
**by** *simp*  
**ultimately show** *?thesis*  
**by** *simp*  
**qed**  
**qed**

**lemma** *seq-comp-secure-aux-2* [*rule-format*]:  
**assumes**  
 $A$ : *secure-termination*  $I D$  **and**  
 $B$ : *ref-union-closed*  $P$  **and**  
 $C$ : *sequential*  $P$  **and**  
 $D$ : *secure*  $P I D$  **and**  
 $E$ : *secure*  $Q I D$   
**shows**  $(ws, Z) \in \text{seq-comp-failures } P Q \implies$   
 $ws = xs @ zs \longrightarrow$   
 $(xs @ [y], \{\}) \in \text{seq-comp-failures } P Q \longrightarrow$   
 $(xs @ y \# \text{ipurge-tr } I D (D y) zs, \text{ipurge-ref } I D (D y) zs Z)$   
 $\in \text{seq-comp-failures } P Q$   
**proof** (*erule seq-comp-failures.induct*, (*rule-tac* [*!*] *impI*) $+$ , *simp-all*, (*erule conjE*) $+$ )  
**fix**  $X$   
**assume**  
 $xs @ zs \notin \text{sentences } P$  **and**  
 $(xs @ zs, X) \in \text{failures } P$  **and**  
 $\text{None} \notin \text{set } xs$  **and**  
 $\text{None} \notin \text{set } zs$  **and**  
 $(xs @ [y], \{\}) \in \text{seq-comp-failures } P Q$   
**thus**  $(xs @ y \# \text{ipurge-tr } I D (D y) zs, \text{ipurge-ref } I D (D y) zs X)$   
 $\in \text{seq-comp-failures } P Q$   
**by** (*rule seq-comp-secure-aux-2-case-1* [*OF A C D*])  
**next**  
**fix**  $X Y$   
**assume**  
 $xs @ zs \in \text{sentences } P$  **and**  
 $(xs @ zs, X) \in \text{failures } P$  **and**  
 $([], Y) \in \text{failures } Q$  **and**  
 $(xs @ [y], \{\}) \in \text{seq-comp-failures } P Q$   
**thus**  $(xs @ y \# \text{ipurge-tr } I D (D y) zs,$   
 $\text{ipurge-ref } I D (D y) zs (\text{insert None } X \cap Y)) \in \text{seq-comp-failures } P Q$   
**by** (*rule seq-comp-secure-aux-2-case-2* [*OF A C D E*])  
**next**  
**fix**  $ws ys Y$   
**assume**  
 $ws \in \text{sentences } P$  **and**  
 $(ys, Y) \in \text{failures } Q$  **and**  
 $ys \neq []$  **and**  
 $ws @ ys = xs @ zs$  **and**  
 $(xs @ [y], \{\}) \in \text{seq-comp-failures } P Q$

**thus**  $(xs @ y \# \text{ipurge-tr } I D (D y) zs, \text{ipurge-ref } I D (D y) zs Y)$   
 $\in \text{seq-comp-failures } P Q$   
**by** (rule *seq-comp-secure-aux-2-case-3* [*OF A B C D E*])  
**next**  
**fix**  $X Y$   
**assume**  
 $(xs @ y \# \text{ipurge-tr } I D (D y) zs, \text{ipurge-ref } I D (D y) zs X)$   
 $\in \text{seq-comp-failures } P Q$  **and**  
 $(xs @ y \# \text{ipurge-tr } I D (D y) zs, \text{ipurge-ref } I D (D y) zs Y)$   
 $\in \text{seq-comp-failures } P Q$   
**hence**  $(xs @ y \# \text{ipurge-tr } I D (D y) zs,$   
 $\text{ipurge-ref } I D (D y) zs X \cup \text{ipurge-ref } I D (D y) zs Y)$   
 $\in \text{seq-comp-failures } P Q$   
**by** (rule *SCF-R4*)  
**thus**  $(xs @ y \# \text{ipurge-tr } I D (D y) zs, \text{ipurge-ref } I D (D y) zs (X \cup Y))$   
 $\in \text{seq-comp-failures } P Q$   
**by** (*simp add: ipurge-ref-distrib-union*)  
**qed**

**lemma** *seq-comp-secure-2*:

**assumes**

*A*: *secure-termination*  $I D$  **and**

*B*: *ref-union-closed*  $P$  **and**

*C*: *sequential*  $P$  **and**

*D*: *secure*  $P I D$  **and**

*E*: *secure*  $Q I D$

**shows**  $(xs @ zs, Z) \in \text{seq-comp-failures } P Q \implies$

$(xs @ [y], \{\}) \in \text{seq-comp-failures } P Q \implies$

$(xs @ y \# \text{ipurge-tr } I D (D y) zs, \text{ipurge-ref } I D (D y) zs Z)$

$\in \text{seq-comp-failures } P Q$

**by** (rule *seq-comp-secure-aux-2* [*OF A B C D E*, **where**  $ws = xs @ zs$ ], *simp-all*)

Finally, the target security conservation theorem can be enunciated and proven, which is done here below. The theorem states that for any two processes  $P, Q$  defined over the same alphabet containing successful termination, to which the noninterference policy  $I$  and the event-domain map  $D$  apply, if:

- $I$  and  $D$  enforce termination security,
- $P$  is refusals union closed and sequential, and
- both  $P$  and  $Q$  are secure with respect to  $I$  and  $D$ ,

then  $P ; Q$  is secure as well.

**theorem** *seq-comp-secure*:



```

assumes
  A: secure-termination I D and
  B: ref-union-closed P and
  C: sequential P and
  D: secure P I D and
  E: secure Q I D
shows secure (P ; Q) I D
proof (simp add: secure-def seq-comp-futures seq-implies-weakly-seq [OF C],
  (rule allI)+, rule impI, erule conjE)
fix xs y ys Y zs Z
assume
  F:  $(xs @ y \# ys, Y) \in \text{seq-comp-failures } P \ Q$  and
  G:  $(xs @ zs, Z) \in \text{seq-comp-failures } P \ Q$ 
show
   $(xs @ \text{ipurge-tr } I \ D \ (D \ y) \ ys, \text{ipurge-ref } I \ D \ (D \ y) \ ys \ Y)$ 
   $\in \text{seq-comp-failures } P \ Q \wedge$ 
   $(xs @ y \# \text{ipurge-tr } I \ D \ (D \ y) \ zs, \text{ipurge-ref } I \ D \ (D \ y) \ zs \ Z)$ 
   $\in \text{seq-comp-failures } P \ Q$ 
  (is  $?A \wedge ?B$ )
proof
  show  $?A$ 
  by (rule seq-comp-secure-1 [OF A B C D E F])
next
  have H: weakly-sequential P
  using C by (rule seq-implies-weakly-seq)
  hence  $((xs @ [y]) @ ys, Y) \in \text{failures } (P ; Q)$ 
  using F by (simp add: seq-comp-failures)
  hence  $(xs @ [y], \{\}) \in \text{failures } (P ; Q)$ 
  by (rule process-rule-2-failures)
  hence  $(xs @ [y], \{\}) \in \text{seq-comp-failures } P \ Q$ 
  using H by (simp add: seq-comp-failures)
  thus  $?B$ 
  by (rule seq-comp-secure-2 [OF A B C D E G])
qed
qed

```

## 2.5 Generalization of the security conservation theorem to lists of processes

The target security conservation theorem, in the basic version just proven, applies to the sequential composition of a pair of processes. However, given an arbitrary list of processes where each process satisfies its assumptions, the theorem could be orderly applied to the composition of the first two processes in the list, then to the composition of the resulting process with the third process in the list, and so on, until the last process is reached. The final outcome would be that the sequential composition of all the processes in the list is secure.

Of course, this argument works provided that the assumptions of the theo-

rem keep being satisfied by the composed processes produced in each step of the recursion. But this is what indeed happens, by virtue of the conservation of refusals union closure and sequentiality under sequential composition, proven previously, and of the conservation of security under sequential composition, ensured by the target theorem itself.

Therefore, the target security conservation theorem can be generalized to an arbitrary list of processes, which is done here below. The resulting theorem states that for any nonempty list of processes defined over the same alphabet containing successful termination, to which the noninterference policy  $I$  and the event-domain map  $D$  apply, if:

- $I$  and  $D$  enforce termination security,
- each process in the list, with the possible exception of the last one, is refusals union closed and sequential, and
- each process in the list is secure with respect to  $I$  and  $D$ ,

then the sequential composition of all the processes in the list is secure as well.

As a precondition, the above conservation lemmas for weak sequentiality, refusals union closure, and sequentiality are generalized, too.

**lemma** *seq-comp-list-weakly-sequential* [rule-format]:

$(\forall X \in \text{set } (P \# PS). \text{weakly-sequential } X) \longrightarrow$   
 $\text{weakly-sequential } (\text{foldl } (;) P PS)$

**proof** (*induction PS rule: rev-induct, simp, rule impI, simp, (erule conjE)+*)

**qed** (*rule seq-comp-weakly-sequential*)

**lemma** *seq-comp-list-ref-union-closed* [rule-format]:

$(\forall X \in \text{set } (\text{butlast } (P \# PS)). \text{weakly-sequential } X) \longrightarrow$   
 $(\forall X \in \text{set } (P \# PS). \text{ref-union-closed } X) \longrightarrow$   
 $\text{ref-union-closed } (\text{foldl } (;) P PS)$

**proof** (*induction PS rule: rev-induct, simp, (rule impI)+, simp, split if-split-asm, simp, rule seq-comp-ref-union-closed, assumption+*)

**fix**  $PS$  **and**  $Q :: 'a \text{ option process}$

**assume**

$A: \text{weakly-sequential } P$  **and**

$B: \forall X \in \text{set } PS. \text{weakly-sequential } X$  **and**

$C: \text{ref-union-closed } Q$  **and**

$D: (\forall X \in \text{set } (P \# \text{butlast } PS). \text{weakly-sequential } X) \longrightarrow$   
 $\text{ref-union-closed } (\text{foldl } (;) P PS)$

**have**  $\text{weakly-sequential } (\text{foldl } (;) P PS)$

**proof** (*rule seq-comp-list-weakly-sequential, simp, erule disjE, simp add: A*)

**fix**  $X$

**assume**  $X \in \text{set } PS$

**with**  $B$  **show**  $\text{weakly-sequential } X ..$

**qed**  
**moreover have**  $\forall X \in \text{set } (P \# \text{butlast } PS)$ . *weakly-sequential X*  
**proof** (*rule ballI, simp, erule disjE, simp add: A*)  
**fix**  $X$   
**assume**  $X \in \text{set } (\text{butlast } PS)$   
**hence**  $X \in \text{set } PS$   
**by** (*rule in-set-butlastD*)  
**with B show** *weakly-sequential X ..*  
**qed**  
**with D have** *ref-union-closed (foldl (;) P PS) ..*  
**ultimately show** *ref-union-closed (foldl (;) P PS ; Q)*  
**using C by** (*rule seq-comp-ref-union-closed*)  
**qed**

**lemma** *seq-comp-list-sequential* [*rule-format*]:  
 $(\forall X \in \text{set } (P \# PS)$ . *sequential X*)  $\longrightarrow$   
*sequential (foldl (;) P PS)*  
**proof** (*induction PS rule: rev-induct, simp, rule impI, simp, (erule conjE)+*)  
**qed** (*rule seq-comp-sequential*)

**theorem** *seq-comp-list-secure* [*rule-format*]:  
**assumes**  $A$ : *secure-termination I D*  
**shows**  
 $(\forall X \in \text{set } (\text{butlast } (P \# PS)))$ . *ref-union-closed X  $\wedge$  sequential X*  $\longrightarrow$   
 $(\forall X \in \text{set } (P \# PS)$ . *secure X I D*)  $\longrightarrow$   
*secure (foldl (;) P PS) I D*  
**proof** (*induction PS rule: rev-induct, simp, (rule impI)+, simp, split if-split-asm,*  
*simp, rule seq-comp-secure [OF A], assumption+*)  
**fix**  $PS Q$   
**assume**  
 $B$ :  $PS \neq []$  **and**  
 $C$ : *ref-union-closed P* **and**  
 $D$ : *sequential P* **and**  
 $E$ :  $\forall X \in \text{set } PS$ . *ref-union-closed X  $\wedge$  sequential X* **and**  
 $F$ : *secure Q I D* **and**  
 $G$ :  $(\forall X \in \text{set } (P \# \text{butlast } PS))$ . *ref-union-closed X  $\wedge$  sequential X*  $\longrightarrow$   
*secure (foldl (;) P PS) I D*  
**have** *ref-union-closed (foldl (;) P PS)*  
**proof** (*rule seq-comp-list-ref-union-closed, simp-all add: B, erule-tac [!] disjE,*  
*simp-all add: C*)  
**show** *weakly-sequential P*  
**using D by** (*rule seq-implies-weakly-seq*)  
**next**  
**fix**  $X$   
**assume**  $X \in \text{set } (\text{butlast } PS)$   
**hence**  $X \in \text{set } PS$   
**by** (*rule in-set-butlastD*)  
**with E have** *ref-union-closed X  $\wedge$  sequential X ..*  
**hence** *sequential X ..*

```

    thus weakly-sequential X
      by (rule seq-implies-weakly-seq)
next
  fix X
  assume X ∈ set PS
  with E have ref-union-closed X ∧ sequential X ..
  thus ref-union-closed X ..
qed
moreover have sequential (foldl (;) P PS)
proof (rule seq-comp-list-sequential, simp, erule disjE, simp add: D)
  fix X
  assume X ∈ set PS
  with E have ref-union-closed X ∧ sequential X ..
  thus sequential X ..
qed
moreover have ∀ X ∈ set (P # butlast PS). ref-union-closed X ∧ sequential X
proof (rule ballI, simp, erule disjE, simp add: C D)
  fix X
  assume X ∈ set (butlast PS)
  hence X ∈ set PS
  by (rule in-set-butlastD)
  with E show ref-union-closed X ∧ sequential X ..
qed
with G have secure (foldl (;) P PS) I D ..
ultimately show secure (foldl (;) P PS ; Q) I D
  using F by (rule seq-comp-secure [OF A])
qed
end

```

### 3 Necessity of nontrivial assumptions

```

theory Counterexamples
imports SequentialComposition
begin

```

The security conservation theorem proven in this paper contains two non-trivial assumptions; namely, the security policy must satisfy predicate *secure-termination*, and the first input process must satisfy predicate *sequential* instead of *weakly-sequential* alone. This section shows, by means of counterexamples, that both of these assumptions are necessary for the theorem to hold.

In more detail, two counterexamples will be constructed: the former drops the termination security assumption, whereas the latter drops the process sequentiality assumption, replacing it with weak sequentiality alone. In both cases, all the other assumptions of the theorem keep being satisfied.

Both counterexamples make use of reflexive security policies, which is the case for any policy of practical significance, and are based on trace set processes as defined in [9]. The security of the processes input to sequential composition, as well as the insecurity of the resulting process, are demonstrated by means of the Ipurge Unwinding Theorem proven in [9].

### 3.1 Preliminary definitions and lemmas

Both counterexamples will use the same type *event* as native type of ordinary events, as well as the same process  $Q$  as second input to sequential composition. Here below are the definitions of these constants, followed by few useful lemmas on process  $Q$ .

**datatype** *event* =  $a \mid b$

**definition**  $Q :: \text{event option process}$  **where**  
 $Q \equiv \text{ts-process } \{\ [], [Some\ b] \}$

**lemma** *trace-set-snd*:  
 $\text{trace-set } \{\ [], [Some\ b] \}$   
**by** (*simp add: trace-set-def*)

**lemmas** *failures-snd* =  $\text{ts-process-failures } [OF\ \text{trace-set-snd}]$

**lemmas** *traces-snd* =  $\text{ts-process-traces } [OF\ \text{trace-set-snd}]$

**lemmas** *next-events-snd* =  $\text{ts-process-next-events } [OF\ \text{trace-set-snd}]$

**lemmas** *unwinding-snd* =  $\text{ts-ipurge-unwinding } [OF\ \text{trace-set-snd}]$

### 3.2 Necessity of termination security

The reason why the conservation of noninterference security under sequential composition requires the security policy to satisfy predicate *secure-termination* is that the second input process cannot engage in its events unless the first process has terminated successfully. Thus, the ordinary events of the first process can indirectly affect the events of the second process by affecting the successful termination of the first process. Therefore, if an ordinary event is allowed to affect successful termination, then the policy must allow it to affect any other event as well, which is exactly what predicate *secure-termination* states.

A counterexample showing the necessity of this assumption can then be constructed by defining a reflexive policy  $I_1$  that allows event *Some a* to affect *None*, but not *Some b*, and a deterministic process  $P_1$  that can engage in *None* only after engaging in *Some a*. The resulting process  $P_1 ; Q$  will

number  $[Some\ a, Some\ b]$ , but not  $[Some\ b]$ , among its traces, so that event *Some a* affects the occurrence of event *Some b* in contrast with policy  $I_1$ , viz.  $P_1 ; Q$  is not secure with respect to  $I_1$ .

Here below are the definitions of constants  $I_1$  and  $P_1$ , followed by few useful lemmas on process  $P_1$ .

**definition**  $I_1 :: (event\ option \times event\ option)\ set\ \mathbf{where}$   
 $I_1 \equiv \{(Some\ a, None)\}^=$

**definition**  $P_1 :: event\ option\ process\ \mathbf{where}$   
 $P_1 \equiv ts\text{-}process\ \{\[], [Some\ a], [Some\ a, None]\}$

**lemma** *trace-set-fst-1*:  
 $trace\text{-}set\ \{\[], [Some\ a], [Some\ a, None]\}$   
**by** (*simp add: trace-set-def*)

**lemmas** *failures-fst-1* = *ts-process-failures* [*OF trace-set-fst-1*]

**lemmas** *traces-fst-1* = *ts-process-traces* [*OF trace-set-fst-1*]

**lemmas** *next-events-fst-1* = *ts-process-next-events* [*OF trace-set-fst-1*]

**lemmas** *unwinding-fst-1* = *ts-ipurge-unwinding* [*OF trace-set-fst-1*]

Here below is the proof that policy  $I_1$  does not satisfy predicate *secure-termination*, whereas the remaining assumptions of the security conservation theorem keep being satisfied. For the sake of simplicity, the identity function is used as event-domain map.

**lemma** *not-secure-termination-1*:  
 $\neg\ secure\text{-}termination\ I_1\ id$   
**proof** (*simp add: secure-termination-def I1-def, rule exI [where x = Some a], simp*)  
**qed** (*rule exI [where x = Some b], simp*)

**lemma** *ref-union-closed-fst-1*:  
 $ref\text{-}union\text{-}closed\ P_1$   
**by** (*rule d-implies-ruc, subst P1-def, rule ts-process-d, rule trace-set-fst-1*)

**lemma** *sequential-fst-1*:  
 $sequential\ P_1$   
**proof** (*simp add: sequential-def sentences-def P1-def traces-fst-1*)  
**qed** (*simp add: set-eq-iff next-events-fst-1*)

**lemma** *secure-fst-1*:  
 $secure\ P_1\ I_1\ id$

**proof** (*simp add: P<sub>1</sub>-def unwinding-fst-1 dfc-equals-dwfc-rel-ipurge [symmetric]*  
*d-future-consistent-def rel-ipurge-def traces-fst-1, (rule allI)+*)

**fix**  $u\ xs\ ys$

**show**

$(xs = [] \vee xs = [Some\ a] \vee xs = [Some\ a,\ None]) \wedge$   
 $(ys = [] \vee ys = [Some\ a] \vee ys = [Some\ a,\ None]) \wedge$   
 $ipurge\text{-}tr\text{-}rev\ I_1\ id\ u\ xs = ipurge\text{-}tr\text{-}rev\ I_1\ id\ u\ ys \longrightarrow$   
 $next\text{-}dom\text{-}events\ (ts\text{-}process\ \{\ [],\ [Some\ a],\ [Some\ a,\ None]\})\ id\ u\ xs =$   
 $next\text{-}dom\text{-}events\ (ts\text{-}process\ \{\ [],\ [Some\ a],\ [Some\ a,\ None]\})\ id\ u\ ys$

**proof** (*simp add: next-dom-events-def next-events-fst-1, cases u*)

**case** *None*

**show**

$(xs = [] \vee xs = [Some\ a] \vee xs = [Some\ a,\ None]) \wedge$   
 $(ys = [] \vee ys = [Some\ a] \vee ys = [Some\ a,\ None]) \wedge$   
 $ipurge\text{-}tr\text{-}rev\ I_1\ id\ u\ xs = ipurge\text{-}tr\text{-}rev\ I_1\ id\ u\ ys \longrightarrow$   
 $\{x.\ u = x \wedge (xs = [] \wedge x = Some\ a \vee xs = [Some\ a] \wedge x = None)\} =$   
 $\{x.\ u = x \wedge (ys = [] \wedge x = Some\ a \vee ys = [Some\ a] \wedge x = None)\}$

**by** (*simp add: I<sub>1</sub>-def None, rule impI, (erule conjE)+,*  
*((erule disjE)+)?, simp)+*)

**next**

**case** (*Some v*)

**show**

$(xs = [] \vee xs = [Some\ a] \vee xs = [Some\ a,\ None]) \wedge$   
 $(ys = [] \vee ys = [Some\ a] \vee ys = [Some\ a,\ None]) \wedge$   
 $ipurge\text{-}tr\text{-}rev\ I_1\ id\ u\ xs = ipurge\text{-}tr\text{-}rev\ I_1\ id\ u\ ys \longrightarrow$   
 $\{x.\ u = x \wedge (xs = [] \wedge x = Some\ a \vee xs = [Some\ a] \wedge x = None)\} =$   
 $\{x.\ u = x \wedge (ys = [] \wedge x = Some\ a \vee ys = [Some\ a] \wedge x = None)\}$

**by** (*simp add: I<sub>1</sub>-def Some, rule impI, (erule conjE)+, cases v,*  
*((erule disjE)+)?, simp, blast?)+*)

**qed**

**qed**

**lemma** *secure-snd-1:*

*secure Q I<sub>1</sub> id*

**proof** (*simp add: Q-def unwinding-snd dfc-equals-dwfc-rel-ipurge [symmetric]*  
*d-future-consistent-def rel-ipurge-def traces-snd, (rule allI)+*)

**fix**  $u\ xs\ ys$

**show**

$(xs = [] \vee xs = [Some\ b]) \wedge$   
 $(ys = [] \vee ys = [Some\ b]) \wedge$   
 $ipurge\text{-}tr\text{-}rev\ I_1\ id\ u\ xs = ipurge\text{-}tr\text{-}rev\ I_1\ id\ u\ ys \longrightarrow$   
 $next\text{-}dom\text{-}events\ (ts\text{-}process\ \{\ [],\ [Some\ b]\})\ id\ u\ xs =$   
 $next\text{-}dom\text{-}events\ (ts\text{-}process\ \{\ [],\ [Some\ b]\})\ id\ u\ ys$

**proof** (*simp add: next-dom-events-def next-events-snd, cases u*)

**case** *None*

**show**

$(xs = [] \vee xs = [Some\ b]) \wedge$   
 $(ys = [] \vee ys = [Some\ b]) \wedge$   
 $ipurge\text{-}tr\text{-}rev\ I_1\ id\ u\ xs = ipurge\text{-}tr\text{-}rev\ I_1\ id\ u\ ys \longrightarrow$

$\{x. u = x \wedge xs = [] \wedge x = \text{Some } b\} = \{x. u = x \wedge ys = [] \wedge x = \text{Some } b\}$   
**by** (*simp add: None, rule impI, (erule conjE)+,*  
*((erule disjE)+)?, simp)+*)  
**next**  
**case** (*Some v*)  
**show**  
 $(xs = [] \vee xs = [\text{Some } b]) \wedge$   
 $(ys = [] \vee ys = [\text{Some } b]) \wedge$   
 $\text{ipurge-tr-rev } I_1 \text{ id } u \text{ xs} = \text{ipurge-tr-rev } I_1 \text{ id } u \text{ ys} \longrightarrow$   
 $\{x. u = x \wedge xs = [] \wedge x = \text{Some } b\} = \{x. u = x \wedge ys = [] \wedge x = \text{Some } b\}$   
**by** (*simp add: I<sub>1</sub>-def Some, rule impI, (erule conjE)+, cases v,*  
*((erule disjE)+)?, simp)+*)  
**qed**  
**qed**

In what follows, the insecurity of process  $P_1 ; Q$  is demonstrated by proving that event list  $[\text{Some } a, \text{Some } b]$  is a trace of the process, whereas  $[\text{Some } b]$  is not.

**lemma** *traces-comp-1:*

*traces* ( $P_1 ; Q$ ) = *Domain* (*seq-comp-failures*  $P_1 Q$ )  
**by** (*subst seq-comp-traces, rule seq-implies-weakly-seq, rule sequential-fst-1, simp*)

**lemma** *ref-union-closed-comp-1:*

*ref-union-closed* ( $P_1 ; Q$ )  
**proof** (*rule seq-comp-ref-union-closed, rule seq-implies-weakly-seq,*  
*rule sequential-fst-1, rule ref-union-closed-fst-1*)  
**qed** (*rule d-implies-ruc, subst Q-def, rule ts-process-d, rule trace-set-snd*)

**lemma** *not-secure-comp-1-aux-aux-1:*

$(xs, X) \in \text{seq-comp-failures } P_1 Q \implies xs \neq [\text{Some } b]$   
**proof** (*rule notI, erule rev-mp, erule seq-comp-failures.induct, (rule-tac [!] impI)+,*  
*simp-all add: P<sub>1</sub>-def Q-def sentences-def*)  
**qed** (*simp-all add: failures-fst-1 traces-fst-1*)

**lemma** *not-secure-comp-1-aux-1:*

$[\text{Some } b] \notin \text{traces } (P_1 ; Q)$   
**proof** (*simp add: traces-comp-1 Domain-iff, rule allI, rule notI*)  
**qed** (*drule not-secure-comp-1-aux-aux-1, simp*)

**lemma** *not-secure-comp-1-aux-2:*

$[\text{Some } a, \text{Some } b] \in \text{traces } (P_1 ; Q)$   
**proof** (*simp add: traces-comp-1 Domain-iff, rule exI [where x = {}]*)  
**have**  $[\text{Some } a] \in \text{sentences } P_1$   
**by** (*simp add: P<sub>1</sub>-def sentences-def traces-fst-1*)  
**moreover have**  $([\text{Some } b], \{\}) \in \text{failures } Q$   
**by** (*simp add: Q-def failures-snd*)  
**moreover have**  $[\text{Some } b] \neq []$



**by** *simp*  
**ultimately have**  $([Some\ a] @ [Some\ b], \{\}) \in seq\text{-}comp\text{-}failures\ P_1\ Q$   
**by** (*rule SCF-R3*)  
**thus**  $([Some\ a, Some\ b], \{\}) \in seq\text{-}comp\text{-}failures\ P_1\ Q$   
**by** *simp*  
**qed**

**lemma** *not-secure-comp-1*:  
 $\neg secure\ (P_1 ; Q)\ I_1\ id$

**proof** (*subst ipurge-unwinding, rule ref-union-closed-comp-1, simp*  
*add: fc-equals-wfc-rel-ipurge [symmetric] future-consistent-def rel-ipurge-def*  
*del: disj-not1, rule exI [where x = Some b], rule exI [where x = []], rule conjI*)  
**show**  $[] \in traces\ (P_1 ; Q)$   
**by** (*rule failures-traces [where X = {}], rule process-rule-1*)  
**next**  
**show**  $\exists ys. ys \in traces\ (P_1 ; Q) \wedge$   
 $ipurge\text{-}tr\text{-}rev\ I_1\ id\ (Some\ b)\ [] = ipurge\text{-}tr\text{-}rev\ I_1\ id\ (Some\ b)\ ys \wedge$   
 $(next\text{-}dom\text{-}events\ (P_1 ; Q)\ id\ (Some\ b)\ [] \neq$   
 $next\text{-}dom\text{-}events\ (P_1 ; Q)\ id\ (Some\ b)\ ys \vee$   
 $ref\text{-}dom\text{-}events\ (P_1 ; Q)\ id\ (Some\ b)\ [] \neq$   
 $ref\text{-}dom\text{-}events\ (P_1 ; Q)\ id\ (Some\ b)\ ys)$   
**proof** (*rule exI [where x = [Some a]], rule conjI, rule-tac [2] conjI,*  
*rule-tac [3] disjI1*)  
**have**  $[Some\ a] @ [Some\ b] \in traces\ (P_1 ; Q)$   
**by** (*simp add: not-secure-comp-1-aux-2*)  
**thus**  $[Some\ a] \in traces\ (P_1 ; Q)$   
**by** (*rule process-rule-2-traces*)  
**next**  
**show**  $ipurge\text{-}tr\text{-}rev\ I_1\ id\ (Some\ b)\ [] = ipurge\text{-}tr\text{-}rev\ I_1\ id\ (Some\ b)\ [Some\ a]$   
**by** (*simp add: I1-def*)  
**next**  
**show**  
 $next\text{-}dom\text{-}events\ (P_1 ; Q)\ id\ (Some\ b)\ [] \neq$   
 $next\text{-}dom\text{-}events\ (P_1 ; Q)\ id\ (Some\ b)\ [Some\ a]$   
**proof** (*simp add: next-dom-events-def next-events-def set-eq-iff,*  
*rule exI [where x = Some b], simp*)  
**qed** (*simp add: not-secure-comp-1-aux-1 not-secure-comp-1-aux-2*)  
**qed**  
**qed**

Here below, the previous results are used to show that constants  $I_1$ ,  $P_1$ ,  $Q$ , and  $id$  indeed constitute a counterexample to the statement obtained by removing termination security from the assumptions of the security conservation theorem.

**lemma** *counterexample-1*:  
 $\neg (ref\text{-}union\text{-}closed\ P_1 \wedge$   
 $sequential\ P_1 \wedge$

```

    secure P1 I1 id ∧
    secure Q I1 id →
    secure (P1 ; Q) I1 id
proof (simp, simp only: conj-assoc [symmetric], (rule conjI)+)
  show ref-union-closed P1
  by (rule ref-union-closed-fst-1)
next
  show sequential P1
  by (rule sequential-fst-1)
next
  show secure P1 I1 id
  by (rule secure-fst-1)
next
  show secure Q I1 id
  by (rule secure-snd-1)
next
  show ¬ secure (P1 ; Q) I1 id
  by (rule not-secure-comp-1)
qed

```

### 3.3 Necessity of process sequentiality

The reason why the conservation of noninterference security under sequential composition requires the first input process to satisfy predicate *sequential*, instead of the more permissive predicate *weakly-sequential*, is that the possibility for the first process to engage in events alternative to successful termination entails the possibility for the resulting process to engage in events alternative to the initial ones of the second process. Namely, the resulting process would admit some state in which events of the first process can occur in alternative to events of the second process. But neither process, though being secure on its own, will in general be prepared to handle securely the alternative events added by the other process. Therefore, the first process must not admit alternatives to successful termination, which is exactly what predicate *sequential* states in addition to *weakly-sequential*.

A counterexample showing the necessity of this assumption can then be constructed by defining a reflexive policy  $I_2$  that does not allow event *Some b* to affect *Some a*, and a deterministic process  $P_2$  that can engage in *Some a* in alternative to *None*. The resulting process  $P_2 ; Q$  will number both  $[Some\ b]$  and  $[Some\ a]$ , but not  $[Some\ b, Some\ a]$ , among its traces, so that event *Some b* affects the occurrence of event *Some a* in contrast with policy  $I_2$ , viz.  $P_2 ; Q$  is not secure with respect to  $I_2$ .

Here below are the definitions of constants  $I_2$  and  $P_2$ , followed by few useful lemmas on process  $P_2$ .

**definition**  $I_2 :: (\text{event option} \times \text{event option}) \text{ set}$  **where**  
 $I_2 \equiv \{(None, Some\ a)\}^=$

**definition**  $P_2$  :: event option process **where**  
 $P_2 \equiv ts\text{-process } \{\ [], [None], [Some\ a], [Some\ a,\ None] \}$

**lemma** *trace-set-fst-2*:  
 $trace\text{-set } \{\ [], [None], [Some\ a], [Some\ a,\ None] \}$   
**by** (*simp add: trace-set-def*)

**lemmas** *failures-fst-2 = ts-process-failures [OF trace-set-fst-2]*

**lemmas** *traces-fst-2 = ts-process-traces [OF trace-set-fst-2]*

**lemmas** *next-events-fst-2 = ts-process-next-events [OF trace-set-fst-2]*

**lemmas** *unwinding-fst-2 = ts-ipurge-unwinding [OF trace-set-fst-2]*

Here below is the proof that process  $P_2$  does not satisfy predicate *sequential*, but rather predicate *weakly-sequential* only, whereas the remaining assumptions of the security conservation theorem keep being satisfied. For the sake of simplicity, the identity function is used as event-domain map.

**lemma** *secure-termination-2*:  
 $secure\text{-termination } I_2\ id$   
**by** (*simp add: secure-termination-def I\_2-def*)

**lemma** *ref-union-closed-fst-2*:  
 $ref\text{-union-closed } P_2$   
**by** (*rule d-implies-ruc, subst P\_2-def, rule ts-process-d, rule trace-set-fst-2*)

**lemma** *weakly-sequential-fst-2*:  
 $weakly\text{-sequential } P_2$   
**by** (*simp add: weakly-sequential-def P\_2-def traces-fst-2*)

**lemma** *not-sequential-fst-2*:  
 $\neg sequential\ P_2$   
**proof** (*simp add: sequential-def, rule disjI2, rule bexI [where x = []]*)  
**show**  $next\text{-events } P_2\ [] \neq \{None\}$   
**proof** (*rule notI, drule eqset-imp-iff [where x = Some a], simp*)  
**qed** (*simp add: P\_2-def next-events-fst-2*)  
**next**  
**show**  $[] \in sentences\ P_2$   
**by** (*simp add: sentences-def P\_2-def traces-fst-2*)  
**qed**

**lemma** *secure-fst-2*:  
 $secure\ P_2\ I_2\ id$   
**proof** (*simp add: P\_2-def unwinding-fst-2 dfc-equals-dwfc-rel-ipurge [symmetric]*  
 $d\text{-future-consistent-def rel-ipurge-def traces-fst-2, (rule allI)+$ )

**fix**  $u\ xs\ ys$   
**show**  
 $(xs = [] \vee xs = [None] \vee xs = [Some\ a] \vee xs = [Some\ a,\ None]) \wedge$   
 $(ys = [] \vee ys = [None] \vee ys = [Some\ a] \vee ys = [Some\ a,\ None]) \wedge$   
 $ipurge\text{-}tr\text{-}rev\ I_2\ id\ u\ xs = ipurge\text{-}tr\text{-}rev\ I_2\ id\ u\ ys \longrightarrow$   
 $next\text{-}dom\text{-}events\ (ts\text{-}process\ \{\ [],\ [None],\ [Some\ a],\ [Some\ a,\ None]\})\ id\ u\ xs =$   
 $next\text{-}dom\text{-}events\ (ts\text{-}process\ \{\ [],\ [None],\ [Some\ a],\ [Some\ a,\ None]\})\ id\ u\ ys$   
**proof** (*simp add: next-dom-events-def next-events-fst-2, cases u*)  
**case**  $None$   
**show**  
 $(xs = [] \vee xs = [None] \vee xs = [Some\ a] \vee xs = [Some\ a,\ None]) \wedge$   
 $(ys = [] \vee ys = [None] \vee ys = [Some\ a] \vee ys = [Some\ a,\ None]) \wedge$   
 $ipurge\text{-}tr\text{-}rev\ I_2\ id\ u\ xs = ipurge\text{-}tr\text{-}rev\ I_2\ id\ u\ ys \longrightarrow$   
 $\{x.\ u = x \wedge (xs = [] \wedge x = None \vee xs = [] \wedge x = Some\ a \vee$   
 $xs = [Some\ a] \wedge x = None)\} =$   
 $\{x.\ u = x \wedge (ys = [] \wedge x = None \vee ys = [] \wedge x = Some\ a \vee$   
 $ys = [Some\ a] \wedge x = None)\}$   
**by** (*simp add: I<sub>2</sub>-def None, rule impI, (erule conjE)+,*  
*((erule disjE)+)?, simp, blast?)+*)  
**next**  
**case**  $(Some\ v)$   
**show**  
 $(xs = [] \vee xs = [None] \vee xs = [Some\ a] \vee xs = [Some\ a,\ None]) \wedge$   
 $(ys = [] \vee ys = [None] \vee ys = [Some\ a] \vee ys = [Some\ a,\ None]) \wedge$   
 $ipurge\text{-}tr\text{-}rev\ I_2\ id\ u\ xs = ipurge\text{-}tr\text{-}rev\ I_2\ id\ u\ ys \longrightarrow$   
 $\{x.\ u = x \wedge (xs = [] \wedge x = None \vee xs = [] \wedge x = Some\ a \vee$   
 $xs = [Some\ a] \wedge x = None)\} =$   
 $\{x.\ u = x \wedge (ys = [] \wedge x = None \vee ys = [] \wedge x = Some\ a \vee$   
 $ys = [Some\ a] \wedge x = None)\}$   
**by** (*simp add: I<sub>2</sub>-def Some, rule impI, (erule conjE)+, cases v,*  
*((erule disjE)+)?, simp, blast?)+*)  
**qed**  
**qed**

**lemma** *secure-snd-2:*  
 $secure\ Q\ I_2\ id$   
**proof** (*simp add: Q-def unwinding-snd dfc-equals-dwfc-rel-ipurge [symmetric]*  
*d-future-consistent-def rel-ipurge-def traces-snd, (rule allI)+*)  
**fix**  $u\ xs\ ys$   
**show**  
 $(xs = [] \vee xs = [Some\ b]) \wedge$   
 $(ys = [] \vee ys = [Some\ b]) \wedge$   
 $ipurge\text{-}tr\text{-}rev\ I_2\ id\ u\ xs = ipurge\text{-}tr\text{-}rev\ I_2\ id\ u\ ys \longrightarrow$   
 $next\text{-}dom\text{-}events\ (ts\text{-}process\ \{\ [],\ [Some\ b]\})\ id\ u\ xs =$   
 $next\text{-}dom\text{-}events\ (ts\text{-}process\ \{\ [],\ [Some\ b]\})\ id\ u\ ys$   
**proof** (*simp add: next-dom-events-def next-events-snd, cases u*)  
**case**  $None$   
**show**  
 $(xs = [] \vee xs = [Some\ b]) \wedge$

$(ys = [] \vee ys = [Some\ b]) \wedge$   
 $ipurge\text{-}tr\text{-}rev\ I_2\ id\ u\ xs = ipurge\text{-}tr\text{-}rev\ I_2\ id\ u\ ys \longrightarrow$   
 $\{x. u = x \wedge xs = [] \wedge x = Some\ b\} = \{x. u = x \wedge ys = [] \wedge x = Some\ b\}$   
**by** (*simp add: None, rule impI, (erule conjE)+,*  
*((erule disjE)+)?, simp)+*)  
**next**  
**case** (*Some v*)  
**show**  
 $(xs = [] \vee xs = [Some\ b]) \wedge$   
 $(ys = [] \vee ys = [Some\ b]) \wedge$   
 $ipurge\text{-}tr\text{-}rev\ I_2\ id\ u\ xs = ipurge\text{-}tr\text{-}rev\ I_2\ id\ u\ ys \longrightarrow$   
 $\{x. u = x \wedge xs = [] \wedge x = Some\ b\} = \{x. u = x \wedge ys = [] \wedge x = Some\ b\}$   
**by** (*simp add: I<sub>2</sub>-def Some, rule impI, (erule conjE)+, cases v,*  
*((erule disjE)+)?, simp)+*)  
**qed**  
**qed**

In what follows, the insecurity of process  $P_2 ; Q$  is demonstrated by proving that event lists  $[Some\ b]$  and  $[Some\ a]$  are traces of the process, whereas  $[Some\ b, Some\ a]$  is not.

**lemma** *traces-comp-2:*

*traces (P<sub>2</sub> ; Q) = Domain (seq-comp-failures P<sub>2</sub> Q)*  
**by** (*subst seq-comp-traces, rule weakly-sequential-fst-2, simp*)

**lemma** *ref-union-closed-comp-2:*

*ref-union-closed (P<sub>2</sub> ; Q)*  
**proof** (*rule seq-comp-ref-union-closed, rule weakly-sequential-fst-2,*  
*rule ref-union-closed-fst-2*)  
**qed** (*rule d-implies-ruc, subst Q-def, rule ts-process-d, rule trace-set-snd*)

**lemma** *not-secure-comp-2-aux-aux-1:*

*(xs, X) ∈ seq-comp-failures P<sub>2</sub> Q ⇒ xs ≠ [Some b, Some a]*  
**proof** (*rule notI, erule rev-mp, erule seq-comp-failures.induct, (rule-tac [!] impI)+,*  
*simp-all add: P<sub>2</sub>-def Q-def sentences-def*)  
**qed** (*simp-all add: failures-fst-2 traces-fst-2 failures-snd*)

**lemma** *not-secure-comp-2-aux-1:*

*[Some b, Some a] ∉ traces (P<sub>2</sub> ; Q)*  
**proof** (*simp add: traces-comp-2 Domain-iff, rule allI, rule notI*)  
**qed** (*drule not-secure-comp-2-aux-aux-1, simp*)

**lemma** *not-secure-comp-2-aux-2:*

*[Some a] ∈ traces (P<sub>2</sub> ; Q)*  
**proof** (*simp add: traces-comp-2 Domain-iff, rule exI [where x = {}]*)  
**have** *[Some a] ∈ sentences P<sub>2</sub>*  
**by** (*simp add: P<sub>2</sub>-def sentences-def traces-fst-2*)  
**moreover have** (*[Some a], {}*) *∈ failures P<sub>2</sub>*

**by** (*simp add: P<sub>2</sub>-def failures-fst-2*)  
**moreover have** ( $\square, \{\}$ )  $\in$  *failures Q*  
**by** (*simp add: Q-def failures-snd*)  
**ultimately have** ( $[Some\ a], insert\ None\ \{\} \cap \{\}$ )  $\in$  *seq-comp-failures P<sub>2</sub> Q*  
**by** (*rule SCF-R2*)  
**thus** ( $[Some\ a], \{\}$ )  $\in$  *seq-comp-failures P<sub>2</sub> Q*  
**by** *simp*  
**qed**

**lemma not-secure-comp-2-aux-3:**  
 $[Some\ b] \in traces\ (P_2 ; Q)$   
**proof** (*simp add: traces-comp-2 Domain-iff, rule exI [where x = {}]*)  
**have**  $\square \in sentences\ P_2$   
**by** (*simp add: P<sub>2</sub>-def sentences-def traces-fst-2*)  
**moreover have** ( $[Some\ b], \{\}$ )  $\in$  *failures Q*  
**by** (*simp add: Q-def failures-snd*)  
**moreover have**  $[Some\ b] \neq \square$   
**by** *simp*  
**ultimately have** ( $\square @ [Some\ b], \{\}$ )  $\in$  *seq-comp-failures P<sub>2</sub> Q*  
**by** (*rule SCF-R3*)  
**thus** ( $[Some\ b], \{\}$ )  $\in$  *seq-comp-failures P<sub>2</sub> Q*  
**by** *simp*  
**qed**

**lemma not-secure-comp-2:**  
 $\neg secure\ (P_2 ; Q)\ I_2\ id$   
**proof** (*subst ipurge-unwinding, rule ref-union-closed-comp-2, simp add: fc-equals-wfc-rel-ipurge [symmetric] future-consistent-def rel-ipurge-def del: disj-not1, rule exI [where x = Some a], rule exI [where x = []], rule conjI*)  
**show**  $\square \in traces\ (P_2 ; Q)$   
**by** (*rule failures-traces [where X = {}], rule process-rule-1*)  
**next**  
**show**  $\exists ys. ys \in traces\ (P_2 ; Q) \wedge$   
 $ipurge-tr-rev\ I_2\ id\ (Some\ a)\ \square = ipurge-tr-rev\ I_2\ id\ (Some\ a)\ ys \wedge$   
 $(next-dom-events\ (P_2 ; Q)\ id\ (Some\ a)\ \square \neq$   
 $next-dom-events\ (P_2 ; Q)\ id\ (Some\ a)\ ys \vee$   
 $ref-dom-events\ (P_2 ; Q)\ id\ (Some\ a)\ \square \neq$   
 $ref-dom-events\ (P_2 ; Q)\ id\ (Some\ a)\ ys)$   
**proof** (*rule exI [where x = [Some b]], rule conjI, rule-tac [2] conjI, rule-tac [3] disjI1*)  
**show**  $[Some\ b] \in traces\ (P_2 ; Q)$   
**by** (*rule not-secure-comp-2-aux-3*)  
**next**  
**show**  $ipurge-tr-rev\ I_2\ id\ (Some\ a)\ \square = ipurge-tr-rev\ I_2\ id\ (Some\ a)\ [Some\ b]$   
**by** (*simp add: I<sub>2</sub>-def*)  
**next**  
**show**  
 $next-dom-events\ (P_2 ; Q)\ id\ (Some\ a)\ \square \neq$   
 $next-dom-events\ (P_2 ; Q)\ id\ (Some\ a)\ [Some\ b]$

```

proof (simp add: next-dom-events-def next-events-def set-eq-iff,
  rule exI [where x = Some a], simp)
qed (simp add: not-secure-comp-2-aux-1 not-secure-comp-2-aux-2)
qed
qed

```

Here below, the previous results are used to show that constants  $I_2$ ,  $P_2$ ,  $Q$ , and  $id$  indeed constitute a counterexample to the statement obtained by replacing process sequentiality with weak sequentiality in the assumptions of the security conservation theorem.

```

lemma counterexample-2:
  ¬ (secure-termination  $I_2$   $id$  ∧
    ref-union-closed  $P_2$  ∧
    weakly-sequential  $P_2$  ∧
    secure  $P_2$   $I_2$   $id$  ∧
    secure  $Q$   $I_2$   $id$  →
    secure ( $P_2$  ;  $Q$ )  $I_2$   $id$ )
proof (simp, simp only: conj-assoc [symmetric], (rule conjI)+)
  show secure-termination  $I_2$   $id$ 
    by (rule secure-termination-2)
  next
  show ref-union-closed  $P_2$ 
    by (rule ref-union-closed-fst-2)
  next
  show weakly-sequential  $P_2$ 
    by (rule weakly-sequential-fst-2)
  next
  show secure  $P_2$   $I_2$   $id$ 
    by (rule secure-fst-2)
  next
  show secure  $Q$   $I_2$   $id$ 
    by (rule secure-snd-2)
  next
  show ¬ secure ( $P_2$  ;  $Q$ )  $I_2$   $id$ 
    by (rule not-secure-comp-2)
qed
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-Isabelle2016/dist/Isabelle2016/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*, Feb. 2016. <http://isabelle.in.tum.de/website-Isabelle2016/dist/Isabelle2016/doc/prog-prove.pdf>.
- [5] T. Nipkow and G. Klein. *Concrete Semantics with Isabelle/HOL*. Springer, 2014. <http://www.concrete-semantics.org/concrete-semantics.pdf>.
- [6] T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, Feb. 2016. <http://isabelle.in.tum.de/website-Isabelle2016/dist/Isabelle2016/doc/tutorial.pdf>.
- [7] P. Noce. A general method for the proof of theorems on tail-recursive functions. *Archive of Formal Proofs*, Dec. 2013. [http://isa-afp.org/entries/Tail\\_Recursive\\_Functions.shtml](http://isa-afp.org/entries/Tail_Recursive_Functions.shtml), Formal proof development.
- [8] P. Noce. Noninterference security in communicating sequential processes. *Archive of Formal Proofs*, May 2014. [http://isa-afp.org/entries/Noninterference\\_CSP.shtml](http://isa-afp.org/entries/Noninterference_CSP.shtml), Formal proof development.
- [9] 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](http://isa-afp.org/entries/Noninterference_Ipurge_Unwinding.shtml), Formal proof development.