

Conservation of CSP Noninterference Security under Concurrent 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

February 6, 2026

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 latter operation is a process in which any event not shared by both operands can occur whenever the operand that admits the event can engage in it, whereas any event shared by both operands can occur just in case both can engage in it.

This paper formalizes Hoare’s definition of concurrent composition and proves, in the general case of a possibly intransitive policy, that CSP noninterference security is conserved under this operation. This result, along with the previous analogous one concerning sequential composition, enables the construction of more and more complex processes enforcing noninterference security by composing, sequentially or concurrently, simpler secure processes, whose security can in turn be proven using either the definition of security, or unwinding theorems.

Contents

1	Concurrent composition and noninterference security	2
1.1	Propaedeutic definitions and lemmas	2
1.2	Concurrent composition	5
1.3	Auxiliary intransitive purge functions	26
1.4	Conservation of noninterference security under concurrent composition	45
1.5	Conservation of noninterference security in the absence of fake events	64

1 Concurrent composition and noninterference security

theory *ConcurrentComposition*
imports *Noninterference-Sequential-Composition.Propaedeutics*
begin

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 latter operation is a process in which any event not shared by both operands can occur whenever the operand that admits the event can engage in it, whereas any event shared by both operands can occur just in case both can engage in it. In other words, shared events are those that synchronize the concurrent processes, which on the contrary can engage asynchronously in the respective non-shared events.

This paper formalizes Hoare's definition of concurrent composition and proves, in the general case of a possibly intransitive policy, that CSP noninterference security [6] is conserved under this operation, viz. the security of both of the input processes implies that of the output process. This result, along with the analogous one concerning sequential composition attained in [10], enables the construction of more and more complex processes enforcing noninterference security by composing, sequentially or concurrently, simpler secure processes, whose security can in turn be proven using either the definition of security formulated in [6], or the unwinding theorems demonstrated in [9], [7], and [8].

Throughout this paper, the salient points of definitions and proofs are commented; for additional information, cf. Isabelle documentation, particularly [5], [4], [3], and [2].

1.1 Propaedeutic definitions and lemmas

The starting point is comprised of some definitions and lemmas propaedeutic to the proof of the target security conservation theorem.

Particularly, the definition of operator *after* given in [1] is formalized, and it is proven that for any secure process P and any trace xs of P , P after xs is still a secure process. Then, this result is used to generalize the lemma stating the closure of the failures of a secure process P under intransitive purge, proven in [10], to the futures of P associated to any one of its traces. This is a generalization of the former result since $futures\ P\ xs = failures\ P$ for $xs = []$.

lemma *sinks-aux-elem* [rule-format]:

$u \in \text{sinks-aux } I D U \text{ } xs \longrightarrow u \in U \vee (\exists x \in \text{set } xs. u = D x)$

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

lemma *ipurge-ref-aux-cons*:

$\text{ipurge-ref-aux } I D U (x \# xs) X = \text{ipurge-ref-aux } I D (\text{sinks-aux } I D U [x]) xs X$

by (subgoal-tac $x \# xs = [x] @ xs$, simp only: ipurge-ref-aux-append, simp)

lemma *process-rule-1-futures*:

$xs \in \text{traces } P \Longrightarrow ([], \{\}) \in \text{futures } P xs$

by (simp add: futures-def, rule traces-failures)

lemma *process-rule-3-futures*:

$(ys, Y) \in \text{futures } P xs \Longrightarrow Y' \subseteq Y \Longrightarrow (ys, Y') \in \text{futures } P xs$

by (simp add: futures-def, rule process-rule-3)

lemma *process-rule-4-futures*:

$(ys, Y) \in \text{futures } P xs \Longrightarrow$

$(ys @ [x], \{\}) \in \text{futures } P xs \vee (ys, \text{insert } x Y) \in \text{futures } P xs$

by (simp add: futures-def, subst append-assoc [symmetric], rule process-rule-4)

lemma *process-rule-5-general* [rule-format]:

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

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

qed (subst append-assoc [symmetric], rule process-rule-5)

Here below is the definition of operator *after*, for which a symbolic notation similar to the one used in [1] is introduced. Then, it is proven that for any process *P* and any trace *xs* of *P*, the failures set and the divergences set of *P* after *xs* indeed enjoy their respective characteristic properties as defined in [6].

definition *future-divergences* :: 'a process \Rightarrow 'a list \Rightarrow 'a list set **where**

$\text{future-divergences } P xs \equiv \{ys. xs @ ys \in \text{divergences } P\}$

definition *after* :: 'a process \Rightarrow 'a list \Rightarrow 'a process (**infixl** $\langle \rangle$ 64) **where**

$P \setminus xs \equiv \text{Abs-process } (\text{futures } P xs, \text{future-divergences } P xs)$

lemma *process-rule-5-futures*:

$ys \in \text{future-divergences } P xs \Longrightarrow ys @ [x] \in \text{future-divergences } P xs$

by (simp add: future-divergences-def, subst append-assoc [symmetric], rule process-rule-5)

lemma *process-rule-6-futures*:

$ys \in \text{future-divergences } P xs \Longrightarrow (ys, Y) \in \text{futures } P xs$

by (simp add: futures-def future-divergences-def, rule process-rule-6)

lemma *after-rep*:
assumes $A: xs \in \text{traces } P$
shows $\text{Rep-process } (P \setminus xs) = (\text{futures } P \ xs, \text{future-divergences } P \ xs)$
(is - = ?X)
proof (*subst after-def*, *rule Abs-process-inverse*, *simp add: process-set-def*,
(subst conj-assoc [symmetric])+, *(rule conjI)+*)
show *process-prop-1 ?X*
proof (*simp add: process-prop-1-def*)
qed (*rule process-rule-1-futures [OF A]*)
next
show *process-prop-2 ?X*
proof (*simp add: process-prop-2-def del: all-simps, (rule allI)+, rule impI*)
qed (*rule process-rule-2-futures*)
next
show *process-prop-3 ?X*
proof (*simp add: process-prop-3-def del: all-simps, (rule allI)+, rule impI,*
erule conjE)
qed (*rule process-rule-3-futures*)
next
show *process-prop-4 ?X*
proof (*simp add: process-prop-4-def, (rule allI)+, rule impI*)
qed (*rule process-rule-4-futures*)
next
show *process-prop-5 ?X*
proof (*simp add: process-prop-5-def, rule allI, rule impI, rule allI*)
qed (*rule process-rule-5-futures*)
next
show *process-prop-6 ?X*
proof (*simp add: process-prop-6-def, rule allI, rule impI, rule allI*)
qed (*rule process-rule-6-futures*)
qed

lemma *after-failures*:
assumes $A: xs \in \text{traces } P$
shows $\text{failures } (P \setminus xs) = \text{futures } P \ xs$
by (*simp add: failures-def after-rep [OF A]*)

lemma *after-futures*:
assumes $A: xs \in \text{traces } P$
shows $\text{futures } (P \setminus xs) \ ys = \text{futures } P \ (xs \ @ \ ys)$
by (*simp add: futures-def after-failures [OF A]*)

Finally, the closure of the futures of a secure process under intransitive purge is proven.

lemma *after-secure*:
assumes $A: xs \in \text{traces } P$
shows $\text{secure } P \ I \ D \implies \text{secure } (P \setminus xs) \ I \ D$

by (*simp add: secure-def after-futures [OF A], blast*)

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

$\llbracket \text{secure } P \text{ I D}; (ys, Y) \in \text{futures } P \text{ xs} \rrbracket \implies$
 $(\text{ipurge-tr-aux } I \text{ D } U \text{ ys}, \text{ipurge-ref-aux } I \text{ D } U \text{ ys } Y) \in \text{futures } P \text{ xs}$

proof (*subgoal-tac xs \in traces P, simp add: after-futures [symmetric],*
rule ipurge-tr-ref-aux-futures, rule after-secure, assumption+)

qed (*simp add: futures-def, drule failures-traces, rule process-rule-2-traces*)

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

$\llbracket \text{secure } P \text{ I D}; (xs @ ys, Y) \in \text{failures } P \rrbracket \implies$
 $(xs @ \text{ipurge-tr-aux } I \text{ D } U \text{ ys}, \text{ipurge-ref-aux } I \text{ D } U \text{ ys } Y) \in \text{failures } P$

by (*drule ipurge-tr-ref-aux-futures, simp-all add: futures-def*)

1.2 Concurrent composition

In [1], the concurrent composition of two processes P , Q , expressed using notation $P \parallel Q$, is defined as a process whose alphabet is the union of the alphabets of P and Q , so that the shared events requiring the synchronous participation of both processes are those in the intersection of their alphabets.

In the formalization of Communicating Sequential Processes developed in [6], the alphabets of P and Q are the data types $'a$ and $'b$ nested in their respective types $'a \text{ process}$ and $'b \text{ process}$. Therefore, for any two maps p , q , the concurrent composition of P and Q with respect to p and q , expressed using notation $P \parallel Q <p, q>$, is defined in what follows as a process of type $'c \text{ process}$, where meaningful events are those in $\text{range } p \cup \text{range } q$ and shared events are those in $\text{range } p \cap \text{range } q$.

The case where $-(\text{range } p \cup \text{range } q) \neq \{\}$ constitutes a generalization of the definition given in [1], and the events in $-(\text{range } p \cup \text{range } q)$, not being mapped to any event in the alphabets of the input processes, shall be understood as fake events lacking any meaning. Consistently with this interpretation, such events are allowed to occur in divergent traces only – necessarily, since divergences are capable by definition of giving rise to any sort of event. As a result, while in [1] the refusals associated to non-divergent traces are the union of two sets, a refusal of P and a refusal of Q , in the following definition they are the union of three sets instead, where the third set is any subset of $-(\text{range } p \cup \text{range } q)$.

Since the definition given in [1] preserves the identity of the events of the input processes, a further generalization resulting from the following definition corresponds to the case where either map p , q is not injective. However, as shown below, these generalizations turn out to compromise neither the compliance of the output of concurrent composition with the characteristic properties of processes as defined in [6], nor even the validity of the target security conservation theorem.

Since divergences can contain fake events, whereas non-divergent traces cannot, it is necessary to add divergent failures to the failures set explicitly. The following definition of the divergences set restricts the definition given in [1], as it identifies a divergence with an arbitrary extension of an event sequence xs being a divergence of both P and Q , rather than a divergence of either process and a trace of the other one. This is a reasonable restriction, in that it requires the concurrent composition of P and Q to admit a shared event x in a divergent trace just in case both P and Q diverge and can then accept x , analogously to what is required for a non-divergent trace. Anyway, the definitions match if the input processes do not diverge, which is the case for any process of practical significance (cf. [1]).

definition *con-comp-divergences* ::

'a process \Rightarrow 'b process \Rightarrow ('a \Rightarrow 'c) \Rightarrow ('b \Rightarrow 'c) \Rightarrow 'c list set **where**
con-comp-divergences P Q p q \equiv
 $\{xs \ @ \ ys \mid xs \ ys.$
 $\text{set } xs \subseteq \text{range } p \cup \text{range } q \wedge$
 $\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p] \in \text{divergences } P \wedge$
 $\text{map } (\text{inv } q) [x \leftarrow xs. x \in \text{range } q] \in \text{divergences } Q\}$

definition *con-comp-failures* ::

'a process \Rightarrow 'b process \Rightarrow ('a \Rightarrow 'c) \Rightarrow ('b \Rightarrow 'c) \Rightarrow 'c failure set **where**
con-comp-failures P Q p q \equiv
 $\{(xs, X \cup Y \cup Z) \mid xs \ X \ Y \ Z.$
 $\text{set } xs \subseteq \text{range } p \cup \text{range } q \wedge$
 $X \subseteq \text{range } p \wedge Y \subseteq \text{range } q \wedge Z \subseteq -(\text{range } p \cup \text{range } q) \wedge$
 $(\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p], \text{inv } p \ ' X) \in \text{failures } P \wedge$
 $(\text{map } (\text{inv } q) [x \leftarrow xs. x \in \text{range } q], \text{inv } q \ ' Y) \in \text{failures } Q\} \cup$
 $\{(xs, X). xs \in \text{con-comp-divergences } P \ Q \ p \ q\}$

definition *con-comp* ::

'a process \Rightarrow 'b process \Rightarrow ('a \Rightarrow 'c) \Rightarrow ('b \Rightarrow 'c) \Rightarrow 'c process **where**
con-comp P Q p q \equiv
 $\text{Abs-process } (\text{con-comp-failures } P \ Q \ p \ q, \text{con-comp-divergences } P \ Q \ p \ q)$

abbreviation *con-comp-syntax* ::

'a process \Rightarrow 'b process \Rightarrow ('a \Rightarrow 'c) \Rightarrow ('b \Rightarrow 'c) \Rightarrow 'c process
 $\langle\langle (- \parallel - \langle -, - \rangle) \rangle 55$

where

$P \parallel Q \langle p, q \rangle \equiv \text{con-comp } P \ Q \ p \ q$

Here below is the proof that, for any two processes P , Q and any two maps p , q , sets *con-comp-failures* P Q p q and *con-comp-divergences* P Q p q enjoy the characteristic properties of the failures and the divergences sets of a process as defined in [6].

lemma *con-comp-prop-1*:
 $([], \{\}) \in \text{con-comp-failures } P \ Q \ p \ q$
proof (*simp add: con-comp-failures-def*)
qed (*rule disjI1, rule conjI, (rule process-rule-1)+*)

lemma *con-comp-prop-2*:
 $(xs \ @ \ [x], X) \in \text{con-comp-failures } P \ Q \ p \ q \implies$
 $(xs, \{\}) \in \text{con-comp-failures } P \ Q \ p \ q$
proof (*simp add: con-comp-failures-def del: filter-append,*
erule disjE, (erule exE)+, (erule conjE)+, rule disjI1)
fix $X \ Y$
assume
 $A: \text{set } xs \subseteq \text{range } p \cup \text{range } q \ \text{and}$
 $B: (\text{map } (\text{inv } p) [x \leftarrow xs \ @ \ [x]. x \in \text{range } p], \text{inv } p \ ' \ X) \in \text{failures } P \ \text{and}$
 $C: (\text{map } (\text{inv } q) [x \leftarrow xs \ @ \ [x]. x \in \text{range } q], \text{inv } q \ ' \ Y) \in \text{failures } Q$
show $\text{set } xs \subseteq \text{range } p \cup \text{range } q \ \wedge$
 $(\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p], \{\}) \in \text{failures } P \ \wedge$
 $(\text{map } (\text{inv } q) [x \leftarrow xs. x \in \text{range } q], \{\}) \in \text{failures } Q$
proof (*simp add: A, rule conjI, cases x \in range p,*
case-tac [3] x \in range q)
assume $x \in \text{range } p$
hence $(\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p] \ @ \ [\text{inv } p \ x], \text{inv } p \ ' \ X) \in \text{failures } P$
using B **by** *simp*
thus $(\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p], \{\}) \in \text{failures } P$
by (*rule process-rule-2*)
next
assume $x \notin \text{range } p$
hence $(\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p], \text{inv } p \ ' \ X) \in \text{failures } P$
using B **by** *simp*
moreover **have** $\{\} \subseteq \text{inv } p \ ' \ X \ ..$
ultimately **show** $(\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p], \{\}) \in \text{failures } P$
by (*rule process-rule-3*)
next
assume $x \in \text{range } q$
hence $(\text{map } (\text{inv } q) [x \leftarrow xs. x \in \text{range } q] \ @ \ [\text{inv } q \ x], \text{inv } q \ ' \ Y) \in \text{failures } Q$
using C **by** *simp*
thus $(\text{map } (\text{inv } q) [x \leftarrow xs. x \in \text{range } q], \{\}) \in \text{failures } Q$
by (*rule process-rule-2*)
next
assume $x \notin \text{range } q$
hence $(\text{map } (\text{inv } q) [x \leftarrow xs. x \in \text{range } q], \text{inv } q \ ' \ Y) \in \text{failures } Q$
using C **by** *simp*
moreover **have** $\{\} \subseteq \text{inv } q \ ' \ Y \ ..$
ultimately **show** $(\text{map } (\text{inv } q) [x \leftarrow xs. x \in \text{range } q], \{\}) \in \text{failures } Q$
by (*rule process-rule-3*)
qed
next
assume $A: xs \ @ \ [x] \in \text{con-comp-divergences } P \ Q \ p \ q$
show

$set\ xs \subseteq range\ p \cup range\ q \wedge$
 $(map\ (inv\ p)\ [x \leftarrow xs.\ x \in range\ p], \{\}) \in failures\ P \wedge$
 $(map\ (inv\ q)\ [x \leftarrow xs.\ x \in range\ q], \{\}) \in failures\ Q \vee$
 $xs \in con-comp-divergences\ P\ Q\ p\ q$
(is ?A \vee -)
proof (*insert A, simp add: con-comp-divergences-def,*
((erule exE)?, erule conjE)+)
fix $ws\ ys$
assume
 $B: xs @ [x] = ws @ ys$ **and**
 $C: set\ ws \subseteq range\ p \cup range\ q$ **and**
 $D: map\ (inv\ p)\ [x \leftarrow ws.\ x \in range\ p] \in divergences\ P$ **and**
 $E: map\ (inv\ q)\ [x \leftarrow ws.\ x \in range\ q] \in divergences\ Q$
show $?A \vee (\exists\ ws'. xs = ws' @ ys')$
 $(\exists\ ws'. xs = ws' @ ys') \wedge$
 $set\ ws' \subseteq range\ p \cup range\ q \wedge$
 $map\ (inv\ p)\ [x \leftarrow ws'.\ x \in range\ p] \in divergences\ P \wedge$
 $map\ (inv\ q)\ [x \leftarrow ws'.\ x \in range\ q] \in divergences\ Q$
(is - \vee ($\exists\ ws'. ?B\ ws')$)
proof (*cases ys, rule disjI1, rule-tac [2] disjI2*)
case Nil
hence $set\ (xs @ [x]) \subseteq range\ p \cup range\ q$
using B **and** C **by** *simp*
hence $insert\ x\ (set\ xs) \subseteq range\ p \cup range\ q$
by *simp*
moreover have $set\ xs \subseteq insert\ x\ (set\ xs)$
by (*rule subset-insertI*)
ultimately have $set\ xs \subseteq range\ p \cup range\ q$
by *simp*
moreover have $map\ (inv\ p)\ [x \leftarrow xs @ [x].\ x \in range\ p] \in divergences\ P$
using Nil **and** B **and** D **by** *simp*
hence $(map\ (inv\ p)\ [x \leftarrow xs.\ x \in range\ p], \{\}) \in failures\ P$
proof (*cases x \in range p, simp-all*)
assume $map\ (inv\ p)\ [x \leftarrow xs.\ x \in range\ p] @ [inv\ p\ x] \in divergences\ P$
hence $(map\ (inv\ p)\ [x \leftarrow xs.\ x \in range\ p] @ [inv\ p\ x], \{\}) \in failures\ P$
by (*rule process-rule-6*)
thus *?thesis*
by (*rule process-rule-2*)
next
assume $map\ (inv\ p)\ [x \leftarrow xs.\ x \in range\ p] \in divergences\ P$
thus *?thesis*
by (*rule process-rule-6*)
qed
moreover have $map\ (inv\ q)\ [x \leftarrow xs @ [x].\ x \in range\ q] \in divergences\ Q$
using Nil **and** B **and** E **by** *simp*
hence $(map\ (inv\ q)\ [x \leftarrow xs.\ x \in range\ q], \{\}) \in failures\ Q$
proof (*cases x \in range q, simp-all*)
assume $map\ (inv\ q)\ [x \leftarrow xs.\ x \in range\ q] @ [inv\ q\ x] \in divergences\ Q$
hence $(map\ (inv\ q)\ [x \leftarrow xs.\ x \in range\ q] @ [inv\ q\ x], \{\}) \in failures\ Q$

```

    by (rule process-rule-6)
  thus ?thesis
    by (rule process-rule-2)
next
  assume map (inv q) [x←xs. x ∈ range q] ∈ divergences Q
  thus ?thesis
    by (rule process-rule-6)
qed
ultimately show ?A
  by blast
next
  case Cons
  moreover have butlast (xs @ [x]) = butlast (ws @ ys)
    using B by simp
  ultimately have xs = ws @ butlast ys
    by (simp add: butlast-append)
  hence ∃ ys'. xs = ws @ ys' ..
  hence ?B ws
    using C and D and E by simp
  thus ∃ ws'. ?B ws' ..
qed
qed
qed

```

lemma *con-comp-prop-3*:

$\llbracket (xs, Y) \in \text{con-comp-failures } P \ Q \ p \ q; X \subseteq Y \rrbracket \implies$
 $(xs, X) \in \text{con-comp-failures } P \ Q \ p \ q$

proof (*simp add: con-comp-failures-def, erule disjE, simp-all,*
(erule exE)+, (erule conjE)+, rule disjI1, simp)

fix $X' \ Y' \ Z'$

assume

A: $X \subseteq X' \cup Y' \cup Z'$ **and**
B: $X' \subseteq \text{range } p$ **and**
C: $Y' \subseteq \text{range } q$ **and**
D: $Z' \subseteq - \text{range } p$ **and**
E: $Z' \subseteq - \text{range } q$ **and**
F: $(\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p], \text{inv } p \text{ ' } X') \in \text{failures } P$ **and**
G: $(\text{map } (\text{inv } q) [x \leftarrow xs. x \in \text{range } q], \text{inv } q \text{ ' } Y') \in \text{failures } Q$

show $\exists X' \ Y' \ Z'$.

$X = X' \cup Y' \cup Z' \wedge$
 $X' \subseteq \text{range } p \wedge$
 $Y' \subseteq \text{range } q \wedge$
 $Z' \subseteq - \text{range } p \wedge$
 $Z' \subseteq - \text{range } q \wedge$
 $(\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p], \text{inv } p \text{ ' } X') \in \text{failures } P \wedge$
 $(\text{map } (\text{inv } q) [x \leftarrow xs. x \in \text{range } q], \text{inv } q \text{ ' } Y') \in \text{failures } Q$

proof (*rule-tac x = X' ∩ X in exI, rule-tac x = Y' ∩ X in exI,*
rule-tac x = Z' ∩ X in exI, (subst conj-assoc [symmetric])+, (rule conjI)+)

show $X = X' \cap X \cup Y' \cap X \cup Z' \cap X$

```

    using A by blast
  next
    show  $X' \cap X \subseteq \text{range } p$ 
    using B by blast
  next
    show  $Y' \cap X \subseteq \text{range } q$ 
    using C by blast
  next
    show  $Z' \cap X \subseteq - \text{range } p$ 
    using D by blast
  next
    show  $Z' \cap X \subseteq - \text{range } q$ 
    using E by blast
  next
    have  $\text{inv } p \text{ ' } (X' \cap X) \subseteq \text{inv } p \text{ ' } X'$ 
    by blast
    with F show  $(\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p], \text{inv } p \text{ ' } (X' \cap X))$ 
       $\in \text{failures } P$ 
    by (rule process-rule-3)
  next
    have  $\text{inv } q \text{ ' } (Y' \cap X) \subseteq \text{inv } q \text{ ' } Y'$ 
    by blast
    with G show  $(\text{map } (\text{inv } q) [x \leftarrow xs. x \in \text{range } q], \text{inv } q \text{ ' } (Y' \cap X))$ 
       $\in \text{failures } Q$ 
    by (rule process-rule-3)
qed
qed

```

lemma *con-comp-prop-4*:

```

 $(xs, X) \in \text{con-comp-failures } P \ Q \ p \ q \implies$ 
 $(xs \ @ \ [x], \ \{\}) \in \text{con-comp-failures } P \ Q \ p \ q \ \vee$ 
 $(xs, \text{insert } x \ X) \in \text{con-comp-failures } P \ Q \ p \ q$ 
proof (simp add: con-comp-failures-def del: filter-append,
erule disjE, (erule exE)+, (erule conjE)+, simp-all del: filter-append)
fix  $X \ Y \ Z$ 
assume

```

$A: X \subseteq \text{range } p$ **and**

$B: Y \subseteq \text{range } q$ **and**

$C: Z \subseteq - \text{range } p$ **and**

$D: Z \subseteq - \text{range } q$ **and**

$E: (\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p], \text{inv } p \text{ ' } X) \in \text{failures } P$ **and**

$F: (\text{map } (\text{inv } q) [x \leftarrow xs. x \in \text{range } q], \text{inv } q \text{ ' } Y) \in \text{failures } Q$

show

$(x \in \text{range } p \ \vee \ x \in \text{range } q) \wedge$

$(\text{map } (\text{inv } p) [x \leftarrow xs \ @ \ [x]. x \in \text{range } p], \ \{\}) \in \text{failures } P \wedge$

$(\text{map } (\text{inv } q) [x \leftarrow xs \ @ \ [x]. x \in \text{range } q], \ \{\}) \in \text{failures } Q \ \vee$

$xs \ @ \ [x] \in \text{con-comp-divergences } P \ Q \ p \ q \ \vee$

$(\exists X' \ Y' \ Z'.$

$\text{insert } x \ (X \cup Y \cup Z) = X' \cup Y' \cup Z' \wedge$

$X' \subseteq \text{range } p \wedge$
 $Y' \subseteq \text{range } q \wedge$
 $Z' \subseteq - \text{range } p \wedge$
 $Z' \subseteq - \text{range } q \wedge$
 $(\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p], \text{inv } p \text{ ' } X') \in \text{failures } P \wedge$
 $(\text{map } (\text{inv } q) [x \leftarrow xs. x \in \text{range } q], \text{inv } q \text{ ' } Y') \in \text{failures } Q) \vee$
 $xs \in \text{con-comp-divergences } P \ Q \ p \ q$
(is - \vee - \vee ?A \vee -)

proof (cases $x \in \text{range } p$, case-tac [!] $x \in \text{range } q$, simp-all)

assume

$G: x \in \text{range } p$ **and**

$H: x \in \text{range } q$

show

$(\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p] @ [\text{inv } p \ x], \{\}) \in \text{failures } P \wedge$
 $(\text{map } (\text{inv } q) [x \leftarrow xs. x \in \text{range } q] @ [\text{inv } q \ x], \{\}) \in \text{failures } Q \vee$
 $xs @ [x] \in \text{con-comp-divergences } P \ Q \ p \ q \vee$
 $?A \vee$
 $xs \in \text{con-comp-divergences } P \ Q \ p \ q$
(is ?B \vee -)

proof (cases ?B, simp-all del: disj-not1, erule disjE)

assume

$I: (\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p] @ [\text{inv } p \ x], \{\}) \notin \text{failures } P$

have ?A

proof (rule-tac $x = \text{insert } x \ X$ **in** exI , rule-tac $x = Y$ **in** exI ,
rule-tac $x = Z$ **in** exI , (subst conj-assoc [symmetric])+, (rule conjI)+)

show $\text{insert } x \ (X \cup Y \cup Z) = \text{insert } x \ X \cup Y \cup Z$

by simp

next

show $\text{insert } x \ X \subseteq \text{range } p$

using A **and** G **by** simp

next

show $Y \subseteq \text{range } q$

using B .

next

show $Z \subseteq - \text{range } p$

using C .

next

show $Z \subseteq - \text{range } q$

using D .

next

have

$(\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p] @ [\text{inv } p \ x], \{\})$
 $\in \text{failures } P \vee$
 $(\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p], \text{insert } (\text{inv } p \ x) \ (\text{inv } p \text{ ' } X))$
 $\in \text{failures } P$
using E **by** (rule process-rule-4)

thus $(\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p], \text{inv } p \text{ ' } \text{insert } x \ X) \in \text{failures } P$

using I **by** simp

next

```

    show (map (inv q) [x←xs. x ∈ range q], inv q ‘ Y) ∈ failures Q
      using F .
  qed
  thus ?thesis
    by simp
next
assume
  I: (map (inv q) [x←xs. x ∈ range q] @ [inv q x], {}) ∉ failures Q
have ?A
proof (rule-tac x = X in exI, rule-tac x = insert x Y in exI,
  rule-tac x = Z in exI, (subst conj-assoc [symmetric])+, (rule conjI)+)
  show insert x (X ∪ Y ∪ Z) = X ∪ insert x Y ∪ Z
    by simp
next
show X ⊆ range p
  using A .
next
show insert x Y ⊆ range q
  using B and H by simp
next
show Z ⊆ - range p
  using C .
next
show Z ⊆ - range q
  using D .
next
show (map (inv p) [x←xs. x ∈ range p], inv p ‘ X) ∈ failures P
  using E .
next
have
  (map (inv q) [x←xs. x ∈ range q] @ [inv q x], {})
    ∈ failures Q ∨
  (map (inv q) [x←xs. x ∈ range q], insert (inv q x) (inv q ‘ Y))
    ∈ failures Q
  using F by (rule process-rule-4)
  thus (map (inv q) [x←xs. x ∈ range q], inv q ‘ insert x Y) ∈ failures Q
    using I by simp
qed
thus ?thesis
  by simp
qed
next
assume G: x ∈ range p
show
  (map (inv p) [x←xs. x ∈ range p] @ [inv p x], {}) ∈ failures P ∧
  (map (inv q) [x←xs. x ∈ range q], {}) ∈ failures Q ∨
  xs @ [x] ∈ con-comp-divergences P Q p q ∨
  ?A ∨
  xs ∈ con-comp-divergences P Q p q

```

```

proof (cases (map (inv p) [x←xs. x ∈ range p] @ [inv p x], {}))
  ∈ failures P)
  case True
  moreover have {} ⊆ inv q ‘ Y ..
  with F have (map (inv q) [x←xs. x ∈ range q], {}) ∈ failures Q
  by (rule process-rule-3)
  ultimately show ?thesis
  by simp
next
  case False
  have ?A
  proof (rule-tac x = insert x X in exI, rule-tac x = Y in exI,
    rule-tac x = Z in exI, (subst conj-assoc [symmetric])+, (rule conjI)+)
  show insert x (X ∪ Y ∪ Z) = insert x X ∪ Y ∪ Z
  by simp
  next
  show insert x X ⊆ range p
  using A and G by simp
  next
  show Y ⊆ range q
  using B .
  next
  show Z ⊆ – range p
  using C .
  next
  show Z ⊆ – range q
  using D .
  next
  have
    (map (inv p) [x←xs. x ∈ range p] @ [inv p x], {})
      ∈ failures P ∨
    (map (inv p) [x←xs. x ∈ range p], insert (inv p x) (inv p ‘ X))
      ∈ failures P
  using E by (rule process-rule-4)
  thus (map (inv p) [x←xs. x ∈ range p], inv p ‘ insert x X) ∈ failures P
  using False by simp
  next
  show (map (inv q) [x←xs. x ∈ range q], inv q ‘ Y) ∈ failures Q
  using F .
  qed
  thus ?thesis
  by simp
qed
next
assume G: x ∈ range q
show
  (map (inv p) [x←xs. x ∈ range p], {}) ∈ failures P ∧
  (map (inv q) [x←xs. x ∈ range q] @ [inv q x], {}) ∈ failures Q ∨
  xs @ [x] ∈ con-comp-divergences P Q p q ∨

```

```

?A ∨
xs ∈ con-comp-divergences P Q p q
proof (cases (map (inv q) [x←xs. x ∈ range q] @ [inv q x], {}))
∈ failures Q)
case True
moreover have {} ⊆ inv p ‘ X ..
with E have (map (inv p) [x←xs. x ∈ range p], {}) ∈ failures P
by (rule process-rule-3)
ultimately show ?thesis
by simp
next
case False
have ?A
proof (rule-tac x = X in exI, rule-tac x = insert x Y in exI,
rule-tac x = Z in exI, (subst conj-assoc [symmetric])+, (rule conjI)+)
show insert x (X ∪ Y ∪ Z) = X ∪ insert x Y ∪ Z
by simp
next
show X ⊆ range p
using A .
next
show insert x Y ⊆ range q
using B and G by simp
next
show Z ⊆ - range p
using C .
next
show Z ⊆ - range q
using D .
next
show (map (inv p) [x←xs. x ∈ range p], inv p ‘ X) ∈ failures P
using E .
next
have
(map (inv q) [x←xs. x ∈ range q] @ [inv q x], {})
∈ failures Q ∨
(map (inv q) [x←xs. x ∈ range q], insert (inv q x) (inv q ‘ Y))
∈ failures Q
using F by (rule process-rule-4)
thus (map (inv q) [x←xs. x ∈ range q], inv q ‘ insert x Y) ∈ failures Q
using False by simp
qed
thus ?thesis
by simp
qed
next
assume
G: x ∉ range p and
H: x ∉ range q

```

have ?A
proof (rule-tac x = X in exI, rule-tac x = Y in exI,
rule-tac x = insert x Z in exI, (subst conj-assoc [symmetric])+,
(rule conjI)+)
show insert x (X ∪ Y ∪ Z) = X ∪ Y ∪ insert x Z
by simp
next
show X ⊆ range p
using A .
next
show Y ⊆ range q
using B .
next
show insert x Z ⊆ - range p
using C and G **by** simp
next
show insert x Z ⊆ - range q
using D and H **by** simp
next
show (map (inv p) [x←xs. x ∈ range p], inv p ‘ X) ∈ failures P
using E .
next
show (map (inv q) [x←xs. x ∈ range q], inv q ‘ Y) ∈ failures Q
using F .
qed
thus
xs @ [x] ∈ con-comp-divergences P Q p q ∨
?A ∨
xs ∈ con-comp-divergences P Q p q
by simp
qed
qed

lemma con-comp-prop-5:

xs ∈ con-comp-divergences P Q p q ⇒
xs @ [x] ∈ con-comp-divergences P Q p q
proof (simp add: con-comp-divergences-def, erule exE, (erule conjE)+, erule exE)
fix xs' ys'
assume
A: set xs' ⊆ range p ∪ range q **and**
B: map (inv p) [x←xs'. x ∈ range p] ∈ divergences P **and**
C: map (inv q) [x←xs'. x ∈ range q] ∈ divergences Q **and**
D: xs = xs' @ ys'
show ∃ xs'.
(∃ ys'. xs @ [x] = xs' @ ys') ∧
set xs' ⊆ range p ∪ range q ∧
map (inv p) [x←xs'. x ∈ range p] ∈ divergences P ∧
map (inv q) [x←xs'. x ∈ range q] ∈ divergences Q
proof (rule-tac x = xs' in exI, simp-all add: A B C)

```

qed (rule-tac  $x = ys' @ [x]$  in  $exI$ , simp add:  $D$ )
qed

lemma con-comp-prop-6:
 $xs \in \text{con-comp-divergences } P \ Q \ p \ q \implies$ 
 $(xs, X) \in \text{con-comp-failures } P \ Q \ p \ q$ 
by (simp add: con-comp-failures-def)

lemma con-comp-rep:
 $\text{Rep-process } (P \parallel Q \langle p, q \rangle) =$ 
 $(\text{con-comp-failures } P \ Q \ p \ q, \text{con-comp-divergences } P \ Q \ p \ q)$ 
(is - = ?X)
proof (subst con-comp-def, rule Abs-process-inverse, simp add: process-set-def,
(subst conj-assoc [symmetric])+, (rule conjI)+)
show process-prop-1 ?X
proof (simp add: process-prop-1-def)
qed (rule con-comp-prop-1)
next
show process-prop-2 ?X
proof (simp add: process-prop-2-def del: all-simps, (rule allI)+, rule impI)
qed (rule con-comp-prop-2)
next
show process-prop-3 ?X
proof (simp add: process-prop-3-def del: all-simps, (rule allI)+, rule impI,
erule conjE)
qed (rule con-comp-prop-3)
next
show process-prop-4 ?X
proof (simp add: process-prop-4-def, (rule allI)+, rule impI)
qed (rule con-comp-prop-4)
next
show process-prop-5 ?X
proof (simp add: process-prop-5-def, rule allI, rule impI, rule allI)
qed (rule con-comp-prop-5)
next
show process-prop-6 ?X
proof (simp add: process-prop-6-def, rule allI, rule impI, rule allI)
qed (rule con-comp-prop-6)
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 [6] and [9], when acting on the concurrent composition of a pair of processes.

```

lemma con-comp-failures:
 $\text{failures } (P \parallel Q \langle p, q \rangle) = \text{con-comp-failures } P \ Q \ p \ q$ 
by (simp add: failures-def con-comp-rep)

```

lemma *con-comp-divergences*:
 $divergences (P \parallel Q <p, q>) = con-comp-divergences P Q p q$
by (*simp add: divergences-def con-comp-rep*)

lemma *con-comp-futures*:
 $futures (P \parallel Q <p, q>) xs =$
 $\{(ys, Y). (xs @ ys, Y) \in con-comp-failures P Q p q\}$
by (*simp add: futures-def con-comp-failures*)

lemma *con-comp-traces*:
 $traces (P \parallel Q <p, q>) = Domain (con-comp-failures P Q p q)$
by (*simp add: traces-def con-comp-failures*)

lemma *con-comp-refusals*:
 $refusals (P \parallel Q <p, q>) xs \equiv con-comp-failures P Q p q \text{ “ } \{xs\}$
by (*simp add: refusals-def con-comp-failures*)

lemma *con-comp-next-events*:
 $next-events (P \parallel Q <p, q>) xs =$
 $\{x. xs @ [x] \in Domain (con-comp-failures P Q p q)\}$
by (*simp add: next-events-def con-comp-traces*)

In what follows, three lemmas are proven. The first one, whose proof makes use of the axiom of choice, establishes an additional property required for the above definition of concurrent composition to be correct, namely that for any two processes whose refusals are closed under set union, their concurrent composition still be such, which is what is expected for any process of practical significance (cf. [9]). The other two lemmas are auxiliary properties of concurrent composition used in the proof of the target security conservation theorem.

lemma *con-comp-ref-union-closed*:
assumes
 $A: ref-union-closed P$ **and**
 $B: ref-union-closed Q$
shows $ref-union-closed (P \parallel Q <p, q>)$
proof (*simp add: ref-union-closed-def con-comp-failures con-comp-failures-def con-comp-divergences-def del: SUP-identity-eq cong: SUP-cong-simp, (rule allI)+, (rule impI)+, erule exE, rule disjI1*)
fix $xs A X$
assume $\forall X \in A. \exists R S T.$
 $X = R \cup S \cup T \wedge$
 $set xs \subseteq range p \cup range q \wedge$
 $R \subseteq range p \wedge$
 $S \subseteq range q \wedge$
 $T \subseteq - range p \wedge$

$T \subseteq - \text{range } q \wedge$
 $(\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p], \text{inv } p \text{ ' } R) \in \text{failures } P \wedge$
 $(\text{map } (\text{inv } q) [x \leftarrow xs. x \in \text{range } q], \text{inv } q \text{ ' } S) \in \text{failures } Q$
 $(\text{is } \forall X \in A. \exists R S T. ?F X R S T)$
hence $\exists r. \forall X \in A. \exists S T. ?F X (r X) S T$
by *(rule bchoice)*
then obtain r **where** $\forall X \in A. \exists S T. ?F X (r X) S T ..$
hence $\exists s. \forall X \in A. \exists T. ?F X (r X) (s X) T$
by *(rule bchoice)*
then obtain s **where** $\forall X \in A. \exists T. ?F X (r X) (s X) T ..$
hence $\exists t. \forall X \in A. ?F X (r X) (s X) (t X)$
by *(rule bchoice)*
then obtain t **where** $C: \forall X \in A. ?F X (r X) (s X) (t X) ..$
assume $D: X \in A$
show $\exists R S T. ?F (\bigcup X \in A. X) R S T$
proof *(rule-tac* $x = \bigcup X \in A. r X$ **in** exI , *rule-tac* $x = \bigcup X \in A. s X$ **in** exI ,
rule-tac $x = \bigcup X \in A. t X$ **in** exI , *(subst conj-assoc [symmetric])+*,
(rule conjI)+)
show $(\bigcup X \in A. X) = (\bigcup X \in A. r X) \cup (\bigcup X \in A. s X) \cup (\bigcup X \in A. t X)$
proof *(simp add: set-eq-iff, rule allI, rule iffI, erule-tac [2] disjE,*
erule-tac [3] disjE, erule-tac [!] bexE)
fix $x X$
have $\forall X \in A. X = r X \cup s X \cup t X$
using C **by** *simp*
moreover assume $E: X \in A$
ultimately have $X = r X \cup s X \cup t X ..$
moreover assume $x \in X$
ultimately have $x \in r X \vee x \in s X \vee x \in t X$
by *blast*
hence $\exists X \in A. x \in r X \vee x \in s X \vee x \in t X$
using $E ..$
thus $(\exists X \in A. x \in r X) \vee (\exists X \in A. x \in s X) \vee (\exists X \in A. x \in t X)$
by *blast*
next
fix $x X$
have $\forall X \in A. X = r X \cup s X \cup t X$
using C **by** *simp*
moreover assume $E: X \in A$
ultimately have $X = r X \cup s X \cup t X ..$
moreover assume $x \in r X$
ultimately have $x \in X$
by *blast*
thus $\exists X \in A. x \in X$
using $E ..$
next
fix $x X$
have $\forall X \in A. X = r X \cup s X \cup t X$
using C **by** *simp*
moreover assume $E: X \in A$

```

ultimately have  $X = r X \cup s X \cup t X$  ..
moreover assume  $x \in s X$ 
ultimately have  $x \in X$ 
  by blast
thus  $\exists X \in A. x \in X$ 
  using E ..
next
fix  $x X$ 
have  $\forall X \in A. X = r X \cup s X \cup t X$ 
  using C by simp
moreover assume E:  $X \in A$ 
ultimately have  $X = r X \cup s X \cup t X$  ..
moreover assume  $x \in t X$ 
ultimately have  $x \in X$ 
  by blast
thus  $\exists X \in A. x \in X$ 
  using E ..
qed
next
have  $\forall X \in A. \text{set } xs \subseteq \text{range } p \cup \text{range } q$ 
  using C by simp
thus  $\text{set } xs \subseteq \text{range } p \cup \text{range } q$ 
  using D ..
next
show  $(\bigcup X \in A. r X) \subseteq \text{range } p$ 
proof (rule subsetI, erule UN-E)
  fix  $x X$ 
  have  $\forall X \in A. r X \subseteq \text{range } p$ 
    using C by simp
  moreover assume  $X \in A$ 
  ultimately have  $r X \subseteq \text{range } p$  ..
  moreover assume  $x \in r X$ 
  ultimately show  $x \in \text{range } p$  ..
qed
next
show  $(\bigcup X \in A. s X) \subseteq \text{range } q$ 
proof (rule subsetI, erule UN-E)
  fix  $x X$ 
  have  $\forall X \in A. s X \subseteq \text{range } q$ 
    using C by simp
  moreover assume  $X \in A$ 
  ultimately have  $s X \subseteq \text{range } q$  ..
  moreover assume  $x \in s X$ 
  ultimately show  $x \in \text{range } q$  ..
qed
next
show  $(\bigcup X \in A. t X) \subseteq \text{range } p$ 
proof (rule subsetI, erule UN-E)
  fix  $x X$ 

```

```

have  $\forall X \in A. t X \subseteq - \text{range } p$ 
  using  $C$  by simp
moreover assume  $X \in A$ 
ultimately have  $t X \subseteq - \text{range } p ..$ 
moreover assume  $x \in t X$ 
ultimately show  $x \in - \text{range } p ..$ 
qed
next
show  $(\bigcup X \in A. t X) \subseteq - \text{range } q$ 
proof (rule subsetI, erule UN-E)
  fix  $x X$ 
  have  $\forall X \in A. t X \subseteq - \text{range } q$ 
    using  $C$  by simp
  moreover assume  $X \in A$ 
  ultimately have  $t X \subseteq - \text{range } q ..$ 
  moreover assume  $x \in t X$ 
  ultimately show  $x \in - \text{range } q ..$ 
qed
next
let  $?A' = \{ \text{inv } p \text{ ' } X \mid X. X \in r \text{ ' } A \}$ 
have
   $(\exists X. X \in ?A') \longrightarrow$ 
   $(\forall X \in ?A'. (\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p], X) \in \text{failures } P) \longrightarrow$ 
   $(\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p], \bigcup X \in ?A'. X) \in \text{failures } P$ 
  using  $A$  by (simp add: ref-union-closed-def)
moreover have  $\exists X. X \in ?A'$ 
using  $D$  by blast
ultimately have
   $(\forall X \in ?A'. (\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p], X) \in \text{failures } P) \longrightarrow$ 
   $(\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p], \bigcup X \in ?A'. X) \in \text{failures } P ..$ 
moreover have
   $\forall X \in ?A'. (\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p], X) \in \text{failures } P$ 
proof (rule ballI, simp, erule exE, erule conjE)
  fix  $R R'$ 
  assume  $R \in r \text{ ' } A$ 
  hence  $\exists X \in A. R = r X$ 
  by (simp add: image-iff)
  then obtain  $X$  where  $E: X \in A$  and  $F: R = r X ..$ 
  have  $\forall X \in A. (\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p], \text{inv } p \text{ ' } r X) \in \text{failures } P$ 
  using  $C$  by simp
  hence  $(\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p], \text{inv } p \text{ ' } r X) \in \text{failures } P$ 
  using  $E ..$ 
  moreover assume  $R' = \text{inv } p \text{ ' } R$ 
  ultimately show  $(\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p], R') \in \text{failures } P$ 
  using  $F$  by simp
qed
ultimately have  $(\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p], \bigcup X \in ?A'. X)$ 
   $\in \text{failures } P ..$ 
moreover have  $(\bigcup X \in ?A'. X) = \text{inv } p \text{ ' } (\bigcup X \in A. r X)$ 

```

proof (*subst set-eq-iff, simp, rule allI, rule iffI, (erule exE, erule conjE)+*)
fix $a R R'$
assume $R \in r \text{ ' } A$
hence $\exists X \in A. R = r X$
by (*simp add: image-iff*)
then obtain X **where** $E: X \in A$ **and** $F: R = r X ..$
assume $a \in R'$ **and** $R' = inv p \text{ ' } R$
hence $a \in inv p \text{ ' } r X$
using F **by** *simp*
hence $\exists x \in r X. a = inv p x$
by (*simp add: image-iff*)
then obtain x **where** $G: x \in r X$ **and** $H: a = inv p x ..$
have $x \in (\bigcup X \in A. r X)$
using E **and** G **by** (*rule UN-I*)
with H **have** $\exists x \in (\bigcup X \in A. r X). a = inv p x ..$
thus $a \in inv p \text{ ' } (\bigcup X \in A. r X)$
by (*simp add: image-iff*)
next
fix a
assume $a \in inv p \text{ ' } (\bigcup X \in A. r X)$
hence $\exists x \in (\bigcup X \in A. r X). a = inv p x$
by (*simp add: image-iff*)
then obtain x **where** $E: x \in (\bigcup X \in A. r X)$ **and** $F: a = inv p x ..$
obtain X **where** $G: X \in A$ **and** $H: x \in r X$ **using** $E ..$
show $\exists R'. (\exists R. R' = inv p \text{ ' } R \wedge R \in r \text{ ' } A) \wedge a \in R'$
proof (*rule-tac x = inv p \text{ ' } r X in exI, rule conjI,*
rule-tac x = r X in exI)
qed (*rule-tac [2] image-eqI, simp add: G, simp add: F, simp add: H*)
qed
ultimately show (*map (inv p) [x←xs. x ∈ range p], inv p \text{ ' } (\bigcup X \in A. r X)*)
 $\in failures P$
by *simp*
next
let $?A' = \{inv q \text{ ' } X \mid X. X \in s \text{ ' } A\}$
have
 $(\exists X. X \in ?A') \longrightarrow$
 $(\forall X \in ?A'. (map (inv q) [x←xs. x \in range q], X) \in failures Q) \longrightarrow$
 $(map (inv q) [x←xs. x \in range q], \bigcup X \in ?A'. X) \in failures Q$
using B **by** (*simp add: ref-union-closed-def*)
moreover have $\exists X. X \in ?A'$
using D **by** *blast*
ultimately have
 $(\forall X \in ?A'. (map (inv q) [x←xs. x \in range q], X) \in failures Q) \longrightarrow$
 $(map (inv q) [x←xs. x \in range q], \bigcup X \in ?A'. X) \in failures Q ..$
moreover have
 $\forall X \in ?A'. (map (inv q) [x←xs. x \in range q], X) \in failures Q$
proof (*rule ballI, simp, erule exE, erule conjE*)
fix $S S'$
assume $S \in s \text{ ' } A$

hence $\exists X \in A. S = s X$
by (*simp add: image-iff*)
then obtain X **where** $E: X \in A$ **and** $F: S = s X$..
have $\forall X \in A. (\text{map } (\text{inv } q) [x \leftarrow xs. x \in \text{range } q], \text{inv } q \text{ ' } s X) \in \text{failures } Q$
using C **by** *simp*
hence $(\text{map } (\text{inv } q) [x \leftarrow xs. x \in \text{range } q], \text{inv } q \text{ ' } s X) \in \text{failures } Q$
using E ..
moreover assume $S' = \text{inv } q \text{ ' } S$
ultimately show $(\text{map } (\text{inv } q) [x \leftarrow xs. x \in \text{range } q], S') \in \text{failures } Q$
using F **by** *simp*
qed
ultimately have $(\text{map } (\text{inv } q) [x \leftarrow xs. x \in \text{range } q], \bigcup X \in ?A'. X) \in \text{failures } Q$..
moreover have $(\bigcup X \in ?A'. X) = \text{inv } q \text{ ' } (\bigcup X \in A. s X)$
proof (*subst set-eq-iff, simp, rule allI, rule iffI, (erule exE, erule conjE)+*)
fix $b S S'$
assume $S \in s \text{ ' } A$
hence $\exists X \in A. S = s X$
by (*simp add: image-iff*)
then obtain X **where** $E: X \in A$ **and** $F: S = s X$..
assume $b \in S'$ **and** $S' = \text{inv } q \text{ ' } S$
hence $b \in \text{inv } q \text{ ' } s X$
using F **by** *simp*
hence $\exists x \in s X. b = \text{inv } q x$
by (*simp add: image-iff*)
then obtain x **where** $G: x \in s X$ **and** $H: b = \text{inv } q x$..
have $x \in (\bigcup X \in A. s X)$
using E **and** G **by** (*rule UN-I*)
with H **have** $\exists x \in (\bigcup X \in A. s X). b = \text{inv } q x$..
thus $b \in \text{inv } q \text{ ' } (\bigcup X \in A. s X)$
by (*simp add: image-iff*)
next
fix b
assume $b \in \text{inv } q \text{ ' } (\bigcup X \in A. s X)$
hence $\exists x \in (\bigcup X \in A. s X). b = \text{inv } q x$
by (*simp add: image-iff*)
then obtain x **where** $E: x \in (\bigcup X \in A. s X)$ **and** $F: b = \text{inv } q x$..
obtain X **where** $G: X \in A$ **and** $H: x \in s X$ **using** E ..
show $\exists S'. (\exists S. S' = \text{inv } q \text{ ' } S \wedge S \in s \text{ ' } A) \wedge b \in S'$
proof (*rule-tac x = inv q ' s X in exI, rule conjI, rule-tac x = s X in exI*)
qed (*rule-tac [2] image-eqI, simp add: G, simp add: F, simp add: H*)
qed
ultimately show $(\text{map } (\text{inv } q) [x \leftarrow xs. x \in \text{range } q], \text{inv } q \text{ ' } (\bigcup X \in A. s X)) \in \text{failures } Q$
by *simp*
qed
qed

lemma *con-comp-failures-traces*:

$(xs, X) \in \text{con-comp-failures } P \ Q \ p \ q \implies$
 $\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p] \in \text{traces } P \wedge$
 $\text{map } (\text{inv } q) [x \leftarrow xs. x \in \text{range } q] \in \text{traces } Q$
proof (*simp add: con-comp-failures-def con-comp-divergences-def, erule disjE,*
(erule exE)+, (erule conjE)+, erule-tac [2] exE, (erule-tac [2] conjE)+,
erule-tac [2] exE)
fix $X \ Y$
assume $(\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p], \text{inv } p \text{ ' } X) \in \text{failures } P$
hence $\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p] \in \text{traces } P$
by (*rule failures-traces*)
moreover assume $(\text{map } (\text{inv } q) [x \leftarrow xs. x \in \text{range } q], \text{inv } q \text{ ' } Y) \in \text{failures } Q$
hence $\text{map } (\text{inv } q) [x \leftarrow xs. x \in \text{range } q] \in \text{traces } Q$
by (*rule failures-traces*)
ultimately show *?thesis ..*

next

fix $vs \ ws$
assume $A: xs = vs \ @ \ ws$
assume $\text{map } (\text{inv } p) [x \leftarrow vs. x \in \text{range } p] \in \text{divergences } P$
hence $\text{map } (\text{inv } p) [x \leftarrow vs. x \in \text{range } p] \ @ \ \text{map } (\text{inv } p) [x \leftarrow ws. x \in \text{range } p]$
 $\in \text{divergences } P$
by (*rule process-rule-5-general*)
hence $\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p] \in \text{divergences } P$
using A **by** *simp*
hence $(\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p], \{\}) \in \text{failures } P$
by (*rule process-rule-6*)
hence $\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p] \in \text{traces } P$
by (*rule failures-traces*)
moreover assume $\text{map } (\text{inv } q) [x \leftarrow vs. x \in \text{range } q] \in \text{divergences } Q$
hence $\text{map } (\text{inv } q) [x \leftarrow vs. x \in \text{range } q] \ @ \ \text{map } (\text{inv } q) [x \leftarrow ws. x \in \text{range } q]$
 $\in \text{divergences } Q$
by (*rule process-rule-5-general*)
hence $\text{map } (\text{inv } q) [x \leftarrow xs. x \in \text{range } q] \in \text{divergences } Q$
using A **by** *simp*
hence $(\text{map } (\text{inv } q) [x \leftarrow xs. x \in \text{range } q], \{\}) \in \text{failures } Q$
by (*rule process-rule-6*)
hence $\text{map } (\text{inv } q) [x \leftarrow xs. x \in \text{range } q] \in \text{traces } Q$
by (*rule failures-traces*)
ultimately show *?thesis ..*

qed

lemma *con-comp-failures-divergences*:

$(xs \ @ \ y \ \# \ ys, Y) \in \text{con-comp-failures } P \ Q \ p \ q \implies$
 $y \notin \text{range } p \implies$
 $y \notin \text{range } q \implies$
 $\exists xs'.$
 $(\exists ys'. xs \ @ \ zs = xs' \ @ \ ys') \wedge$
 $\text{set } xs' \subseteq \text{range } p \cup \text{range } q \wedge$
 $\text{map } (\text{inv } p) [x \leftarrow xs'. x \in \text{range } p] \in \text{divergences } P \wedge$

$\text{map } (\text{inv } q) [x \leftarrow xs'. x \in \text{range } q] \in \text{divergences } Q$
proof (*simp add: con-comp-failures-def con-comp-divergences-def,*
erule exE, (erule conjE)+, erule exE)
fix $xs' ys'$
assume
A: $y \notin \text{range } p$ and
B: $y \notin \text{range } q$ and
C: $\text{set } xs' \subseteq \text{range } p \cup \text{range } q$ and
D: $\text{map } (\text{inv } p) [x \leftarrow xs'. x \in \text{range } p] \in \text{divergences } P$ and
E: $\text{map } (\text{inv } q) [x \leftarrow xs'. x \in \text{range } q] \in \text{divergences } Q$ and
F: $xs @ y \# ys = xs' @ ys'$
have $\text{length } xs' \leq \text{length } xs$
proof (*rule ccontr*)
assume $\neg \text{length } xs' \leq \text{length } xs$
moreover have $\text{take } (\text{length } xs') (xs @ [y] @ ys) =$
 $\text{take } (\text{length } xs') (xs @ [y]) @ \text{take } (\text{length } xs' - \text{Suc } (\text{length } xs)) ys$
(is - = - @ ?vs)
by (*simp only: take-append, simp*)
ultimately have $\text{take } (\text{length } xs') (xs @ y \# ys) = xs @ y \# ?vs$
by *simp*
moreover have $\text{take } (\text{length } xs') (xs @ y \# ys) =$
 $\text{take } (\text{length } xs') (xs' @ ys')$
using *F by simp*
ultimately have $xs' = xs @ y \# ?vs$
by *simp*
hence $\text{set } (xs @ y \# ?vs) \subseteq \text{range } p \cup \text{range } q$
using *C by simp*
hence $y \in \text{range } p \cup \text{range } q$
by *simp*
thus *False*
using *A and B by simp*
qed
moreover have $xs @ zs =$
 $\text{take } (\text{length } xs') (xs @ zs) @ \text{drop } (\text{length } xs') (xs @ zs)$
(is - = - @ ?vs)
by (*simp only: append-take-drop-id*)
ultimately have $xs @ zs = \text{take } (\text{length } xs') (xs @ y \# ys) @ ?vs$
by *simp*
moreover have $\text{take } (\text{length } xs') (xs @ y \# ys) =$
 $\text{take } (\text{length } xs') (xs' @ ys')$
using *F by simp*
ultimately have *G: $xs @ zs = xs' @ ?vs$*
by (*simp del: take-append, simp*)
show *?thesis*
proof (*rule-tac x = xs' in exI, rule conjI, rule-tac x = ?vs in exI*)
qed (*subst G, simp-all add: C D E*)
qed

In order to prove that CSP noninterference security is conserved under concurrent composition, the first issue to be solved is to identify the noninterference policy I' and the event-domain map D' with respect to which the output process is secure.

If the events of the input processes corresponding to those of the output process contained in $\text{range } p \cap \text{range } q$ were mapped by the respective event-domain maps D, E into distinct security domains, there would be no criterion for determining the domains of the aforesaid events of the output process, due to the equivalence of the input processes ensuing from the commutative property of concurrent composition. Therefore, D and E must map the events of the input processes into security domains of the same type $'d$, and for each x in $\text{range } p \cap \text{range } q$, D and E must map the events of the input processes corresponding to x into the same domain. This requirement is formalized here below by means of predicate *consistent-maps*.

Similarly, if distinct noninterference policies applied to the input processes, there would exist some ordered pair of security domains included in one of the policies, but not in the other one. Thus, again, there would be no criterion for determining the inclusion of such a pair of domains in the policy I' applying to the output process. As a result, the input processes are required to enforce the same noninterference policy I , so that for any two domains d, e of type $'d$, the ordered pair comprised of the corresponding security domains for the output process will be included in I' just in case $(d, e) \in I$.

However, in case $-(\text{range } p \cup \text{range } q) \neq \{\}$, the event-domain map D' for the output process must assign a security domain to the fake events in $-(\text{range } p \cup \text{range } q)$ as well. Since such events lack any meaning, they may all be mapped to the same security domain, distinct from the domains of the meaningful events in $\text{range } p \cup \text{range } q$. A simple way to do this is to identify the type of the security domains for the output process with $'d$ option. Then, for any meaningful event x , D' will assign x to domain *Some* d , where d is the domain of the events of the input processes mapped to x , whereas $D' y = \text{None}$ for any fake event y . Such an event-domain map, denoted using notation *con-comp-map* $D E p q$, is defined here below.

Therefore, for any two security domains *Some* $d, \text{Some } e$ for the output process, the above considerations about policy I' entail that $(\text{Some } d, \text{Some } e) \in I'$ just in case $(d, e) \in I$. Furthermore, since fake events may only occur in divergent traces, which are extensions of divergences of the input processes comprised of meaningful events, I' must allow the security domain *None* of fake events to be affected by any meaningful domain matching pattern *Some* -. Such a noninterference policy, denoted using notation *con-comp-pol* I , is defined here below. Observe that *con-comp-pol* I keeps being reflexive or transitive if I is.

definition *con-comp-pol* ::
 ('d × 'd) set ⇒ ('d option × 'd option) set **where**
con-comp-pol I ≡
 {(Some d, Some e) | d e. (d, e) ∈ I} ∪ {(u, v). v = None}

function *con-comp-map* ::
 ('a ⇒ 'd) ⇒ ('b ⇒ 'd) ⇒ ('a ⇒ 'c) ⇒ ('b ⇒ 'c) ⇒ 'c ⇒ 'd option **where**
 x ∈ range p ⇒
 con-comp-map D E p q x = Some (D (inv p x)) |
 x ∉ range p ⇒ x ∈ range q ⇒
 con-comp-map D E p q x = Some (E (inv q x)) |
 x ∉ range p ⇒ x ∉ range q ⇒
 con-comp-map D E p q x = None
by (atomize-elim, simp-all add: split-paired-all, blast)
termination by lexicographic-order

definition *consistent-maps* ::
 ('a ⇒ 'd) ⇒ ('b ⇒ 'd) ⇒ ('a ⇒ 'c) ⇒ ('b ⇒ 'c) ⇒ bool **where**
consistent-maps D E p q ≡
 ∀x ∈ range p ∩ range q. D (inv p x) = E (inv q x)

1.3 Auxiliary intransitive purge functions

Let I be a noninterference policy, D an event-domain map, U a domain set, and $xs = x \# xs'$ an event list. Suppose to take event x just in case it satisfies predicate P , to append xs' to the resulting list (matching either $[x]$ or $[]$), and then to compute the intransitive purge of the resulting list with domain set U . If recursion with respect to the input list is added, replacing xs' with the list produced by the same algorithm using xs' as input list and *sinks-aux* $I D U [x]$ as domain set, the final result matches that obtained by applying filter P to the intransitive purge of xs with domain set U . In fact, in each recursive step, the processed item of the input list is retained in the output list just in case it passes filter P and may be affected neither by the domains in U , nor by the domains of the previous items affected by some domain in U .

Here below is the formal definition of such purge function, named *ipurge-tr-aux-foldr* as its action resembles that of function *foldr*.

primrec *ipurge-tr-aux-foldr* ::
 ('d × 'd) set ⇒ ('a ⇒ 'd) ⇒ ('a ⇒ bool) ⇒ 'd set ⇒ 'a list ⇒ 'a list
where
ipurge-tr-aux-foldr I D P U [] = [] |
ipurge-tr-aux-foldr I D P U (x # xs) = *ipurge-tr-aux* I D U
 ((if P x then [x] else []) @
ipurge-tr-aux-foldr I D P (sinks-aux I D U [x]) xs)

Likewise, given $I, D, U, xs = x \# xs'$, and an event set X , suppose to take x just in case it satisfies predicate P , to append $ipurge\text{-}tr\text{-}aux\text{-}foldr\ I\ D\ P\ (sinks\text{-}aux\ I\ D\ U\ [x])\ xs'$ to the resulting list (matching either $[x]$ or $[]$), and then to compute the intransitive purge of X using the resulting list as input list and U as domain set. If recursion with respect to the input list is added, replacing X with the set produced by the same algorithm using xs' as input list, X as input set, and $sinks\text{-}aux\ I\ D\ U\ [x]$ as domain set, the final result matches the intransitive purge of X with input list xs and domain set U . In fact, each recursive step is such as to remove from X any event that may be affected either by the domains in U , or by the domains of the items of xs preceding the processed one which are affected by some domain in U .

From the above considerations on function $ipurge\text{-}tr\text{-}aux\text{-}foldr$, it follows that the presence of list $ipurge\text{-}tr\text{-}aux\text{-}foldr\ I\ D\ P\ (sinks\text{-}aux\ I\ D\ U\ [x])\ xs'$ has no impact on the final result, because none of its items may be affected by the domains in U .

Here below is the formal definition of such purge function, named $ipurge\text{-}ref\text{-}aux\text{-}foldr$, which at first glance just seems a uselessly complicate and inefficient way to compute the intransitive purge of an event set.

primrec $ipurge\text{-}ref\text{-}aux\text{-}foldr ::$

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

where

$ipurge\text{-}ref\text{-}aux\text{-}foldr\ I\ D\ P\ U\ []\ X = ipurge\text{-}ref\text{-}aux\ I\ D\ U\ []\ X \mid$
 $ipurge\text{-}ref\text{-}aux\text{-}foldr\ I\ D\ P\ U\ (x \# xs)\ X = ipurge\text{-}ref\text{-}aux\ I\ D\ U$
 $((if\ P\ x\ then\ [x]\ else\ [])\ @$
 $ipurge\text{-}tr\text{-}aux\text{-}foldr\ I\ D\ P\ (sinks\text{-}aux\ I\ D\ U\ [x])\ xs)$
 $(ipurge\text{-}ref\text{-}aux\text{-}foldr\ I\ D\ P\ (sinks\text{-}aux\ I\ D\ U\ [x])\ xs\ X)$

The reason for the introduction of such intransitive purge functions is that the recursive equations contained in their definitions, along with lemma $ipurge\text{-}tr\text{-}ref\text{-}aux\text{-}failures\text{-}general$, enable to prove by induction on list ys , assuming that process P be secure in addition to further, minor premises, the following implication:

$(map\ (inv\ p)\ (filter\ (\lambda x. x \in range\ p)\ (xs\ @\ ys)),\ inv\ p\ 'Y) \in failures\ P \longrightarrow$
 $(map\ (inv\ p)\ (filter\ (\lambda x. x \in range\ p)\ xs)\ @\ map\ (inv\ p)\ (ipurge\text{-}tr\text{-}aux\text{-}foldr$
 $(con\text{-}comp\text{-}pol\ I)\ (con\text{-}comp\text{-}map\ D\ E\ p\ q)\ (\lambda x. x \in range\ p)\ U\ ys),\ inv\ p$
 $'\ ipurge\text{-}ref\text{-}aux\text{-}foldr\ (con\text{-}comp\text{-}pol\ I)\ (con\text{-}comp\text{-}map\ D\ E\ p\ q)\ (\lambda x. x \in$
 $range\ p)\ U\ ys\ Y) \in failures\ P$

In fact, for $ys = y \# ys'$, the induction hypothesis entails that the consequent holds if xs, ys , and U are replaced with $xs\ @\ [y], ys'$, and $sinks\text{-}aux\ (con\text{-}comp\text{-}pol\ I)\ (con\text{-}comp\text{-}map\ D\ E\ p\ q)\ U\ [y]$, respectively. The proof can

then be accomplished by applying lemma *ipurge-tr-ref-aux-failures-general* to the resulting future of trace $\text{map } (\text{inv } p) (\text{filter } (\lambda x. x \in \text{range } p) xs)$, moving functions *ipurge-tr-aux* and *ipurge-ref-aux* into the arguments of $\text{map } (\text{inv } p)$ and $(\cdot) (\text{inv } p)$, and using the recursive equations contained in the definitions of functions *ipurge-tr-aux-foldr* and *ipurge-ref-aux-foldr*.

This property, along with the match of the outputs of functions *ipurge-tr-aux-foldr* and *ipurge-ref-aux-foldr* with the filtered intransitive purge of the input event list and the intransitive purge of the input event set, respectively, permits to solve the main proof obligations arising from the demonstration of the target security conservation theorem.

Here below is the proof of the equivalence between function *ipurge-tr-aux-foldr* and the filtered intransitive purge of an event list.

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

$U \subseteq V \implies$

$\text{ipurge-tr-aux } I D U (\text{ipurge-tr-aux-foldr } I D P V xs) =$
 $\text{ipurge-tr-aux-foldr } I D P V xs$

proof (*induction xs, simp-all add: ipurge-tr-aux-union [symmetric]*)

qed (*drule Un-absorb2, simp*)

lemma *ipurge-tr-aux-foldr-eq*:

$[x \leftarrow \text{ipurge-tr-aux } I D U xs. P x] = \text{ipurge-tr-aux-foldr } I D P U xs$

proof (*induction xs arbitrary: U, simp*)

fix $x xs U$

assume

$A: \bigwedge U. [x \leftarrow \text{ipurge-tr-aux } I D U xs. P x] = \text{ipurge-tr-aux-foldr } I D P U xs$

show $[x \leftarrow \text{ipurge-tr-aux } I D U (x \# xs). P x] =$

$\text{ipurge-tr-aux-foldr } I D P U (x \# xs)$

proof (*cases $\exists u \in U. (u, D x) \in I,$*

simp-all only: ipurge-tr-aux-foldr.simps ipurge-tr-aux-cons

sinks-aux-single-event if-True if-False)

case *True*

have $B: [x \leftarrow \text{ipurge-tr-aux } I D (\text{insert } (D x) U) xs. P x] =$

$\text{ipurge-tr-aux-foldr } I D P (\text{insert } (D x) U) xs$

using A .

show $[x \leftarrow \text{ipurge-tr-aux } I D (\text{insert } (D x) U) xs. P x] = \text{ipurge-tr-aux } I D U$

$((\text{if } P x \text{ then } [x] \text{ else } []) @ \text{ipurge-tr-aux-foldr } I D P (\text{insert } (D x) U) xs)$

proof (*cases P x, simp-all add: ipurge-tr-aux-cons True*

del: con-comp-map.simps)

have $\text{insert } (D x) U \subseteq \text{insert } (D x) U ..$

hence $\text{ipurge-tr-aux } I D (\text{insert } (D x) U)$

$(\text{ipurge-tr-aux-foldr } I D P (\text{insert } (D x) U) xs) =$

$\text{ipurge-tr-aux-foldr } I D P (\text{insert } (D x) U) xs$

by (*rule ipurge-tr-aux-foldr-subset*)

thus $[x \leftarrow \text{ipurge-tr-aux } I D (\text{insert } (D x) U) xs. P x] =$

$\text{ipurge-tr-aux } I D (\text{insert } (D x) U)$

$(\text{ipurge-tr-aux-foldr } I D P (\text{insert } (D x) U) xs)$

```

    using B by simp
  next
  have  $U \subseteq \text{insert } (D x) U$ 
  by (rule subset-insertI)
  hence  $\text{ipurge-tr-aux } I D U$ 
    ( $\text{ipurge-tr-aux-foldr } I D P (\text{insert } (D x) U) xs =$ 
      $\text{ipurge-tr-aux-foldr } I D P (\text{insert } (D x) U) xs$ )
  by (rule ipurge-tr-aux-foldr-subset)
  thus  $[x \leftarrow \text{ipurge-tr-aux } I D (\text{insert } (D x) U) xs. P x] =$ 
     $\text{ipurge-tr-aux } I D U$ 
    ( $\text{ipurge-tr-aux-foldr } I D P (\text{insert } (D x) U) xs$ )
  using B by simp
qed
next
case False
have  $B: [x \leftarrow \text{ipurge-tr-aux } I D U xs. P x] = \text{ipurge-tr-aux-foldr } I D P U xs$ 
  using A .
show  $[x \leftarrow x \# \text{ipurge-tr-aux } I D U xs. P x] = \text{ipurge-tr-aux } I D U$ 
  ((if  $P x$  then  $[x]$  else  $[]$ ) @  $\text{ipurge-tr-aux-foldr } I D P U xs$ )
proof (cases  $P x$ , simp-all add: ipurge-tr-aux-cons False
  del: con-comp-map.simps)
  have  $U \subseteq U ..$ 
  hence  $\text{ipurge-tr-aux } I D U (\text{ipurge-tr-aux-foldr } I D P U xs) =$ 
     $\text{ipurge-tr-aux-foldr } I D P U xs$ 
  by (rule ipurge-tr-aux-foldr-subset)
  thus  $[x \leftarrow \text{ipurge-tr-aux } I D U xs. P x] =$ 
     $\text{ipurge-tr-aux } I D U (\text{ipurge-tr-aux-foldr } I D P U xs)$ 
  using B by simp
next
  have  $U \subseteq U ..$ 
  hence  $\text{ipurge-tr-aux } I D U (\text{ipurge-tr-aux-foldr } I D P U xs) =$ 
     $\text{ipurge-tr-aux-foldr } I D P U xs$ 
  by (rule ipurge-tr-aux-foldr-subset)
  thus  $[x \leftarrow \text{ipurge-tr-aux } I D U xs. P x] =$ 
     $\text{ipurge-tr-aux } I D U (\text{ipurge-tr-aux-foldr } I D P U xs)$ 
  using B by simp
qed
qed
qed

```

Here below is the proof of the equivalence between function *ipurge-ref-aux-foldr* and the intransitive purge of an event set.

```

lemma ipurge-tr-aux-foldr-sinks-aux [rule-format]:
   $U \subseteq V \longrightarrow \text{sinks-aux } I D U (\text{ipurge-tr-aux-foldr } I D P V xs) = U$ 
proof (induction  $xs$  arbitrary:  $V$ , simp, rule impI)
  fix  $x xs V$ 
  assume

```

A: $\bigwedge V. U \subseteq V \longrightarrow \text{sinks-aux } I D U (\text{ipurge-tr-aux-foldr } I D P V xs) = U$ **and**
B: $U \subseteq V$
show $\text{sinks-aux } I D U (\text{ipurge-tr-aux-foldr } I D P V (x \# xs)) = U$
proof (*cases* $P x$, *case-tac* $[\!]$ $\exists v \in V. (v, D x) \in I$,
simp-all (*no-asm-simp*) *add:* $\text{sinks-aux-cons ipurge-tr-aux-cons}$)
have $U \subseteq \text{insert } (D x) V \longrightarrow$
 $\text{sinks-aux } I D U (\text{ipurge-tr-aux-foldr } I D P (\text{insert } (D x) V) xs) = U$
(is - $\longrightarrow \text{sinks-aux } I D U ?ys = U$)
using A .
moreover have $U \subseteq \text{insert } (D x) V$
using B **by** (*rule subset-insertI2*)
ultimately have $\text{sinks-aux } I D U ?ys = U$..
moreover have $\text{insert } (D x) V \subseteq \text{insert } (D x) V$..
hence $\text{ipurge-tr-aux } I D (\text{insert } (D x) V)$
 $(\text{ipurge-tr-aux-foldr } I D P (\text{insert } (D x) V) xs) = ?ys$
(is ?zs = -)
by (*rule ipurge-tr-aux-foldr-subset*)
ultimately show $\text{sinks-aux } I D U ?zs = U$
by *simp*
next
assume $C: \neg (\exists v \in V. (v, D x) \in I)$
have $\neg (\exists u \in U. (u, D x) \in I)$
proof
assume $\exists u \in U. (u, D x) \in I$
then obtain u **where** $D: u \in U$ **and** $E: (u, D x) \in I$..
have $u \in V$
using B **and** D ..
with E **have** $\exists v \in V. (v, D x) \in I$..
thus *False*
using C **by** *contradiction*
qed
thus
 $((\exists v \in U. (v, D x) \in I) \longrightarrow \text{sinks-aux } I D (\text{insert } (D x) U)$
 $(\text{ipurge-tr-aux } I D V (\text{ipurge-tr-aux-foldr } I D P V xs) = U) \wedge$
 $(\forall v \in U. (v, D x) \notin I) \longrightarrow \text{sinks-aux } I D U$
 $(\text{ipurge-tr-aux } I D V (\text{ipurge-tr-aux-foldr } I D P V xs) = U)$
proof *simp*
have $U \subseteq V \longrightarrow \text{sinks-aux } I D U (\text{ipurge-tr-aux-foldr } I D P V xs) = U$
(is - $\longrightarrow \text{sinks-aux } I D U ?ys = U$)
using A .
hence $\text{sinks-aux } I D U ?ys = U$ **using** B ..
moreover have $V \subseteq V$..
hence $\text{ipurge-tr-aux } I D V (\text{ipurge-tr-aux-foldr } I D P V xs) = ?ys$
(is ?zs = -)
by (*rule ipurge-tr-aux-foldr-subset*)
ultimately show $\text{sinks-aux } I D U ?zs = U$
by *simp*
qed
next

have $U \subseteq \text{insert } (D \ x) \ V \longrightarrow$
 $\text{sinks-aux } I \ D \ U \ (\text{ipurge-tr-aux-foldr } I \ D \ P \ (\text{insert } (D \ x) \ V) \ xs) = U$
 $(\text{is } - \longrightarrow \text{sinks-aux } I \ D \ U \ ?ys = U)$
using A .
moreover have $U \subseteq \text{insert } (D \ x) \ V$
using B **by** $(\text{rule subset-insertI2})$
ultimately have $\text{sinks-aux } I \ D \ U \ ?ys = U$..
moreover have $V \subseteq \text{insert } (D \ x) \ V$
by $(\text{rule subset-insertI})$
hence $\text{ipurge-tr-aux } I \ D \ V$
 $(\text{ipurge-tr-aux-foldr } I \ D \ P \ (\text{insert } (D \ x) \ V) \ xs) = ?ys$
 $(\text{is } ?zs = -)$
by $(\text{rule ipurge-tr-aux-foldr-subset})$
ultimately show $\text{sinks-aux } I \ D \ U \ ?zs = U$
by simp
next
have $U \subseteq V \longrightarrow \text{sinks-aux } I \ D \ U \ (\text{ipurge-tr-aux-foldr } I \ D \ P \ V \ xs) = U$
 $(\text{is } - \longrightarrow \text{sinks-aux } I \ D \ U \ ?ys = U)$
using A .
hence $\text{sinks-aux } I \ D \ U \ ?ys = U$ **using** B ..
moreover have $V \subseteq V$..
hence $\text{ipurge-tr-aux } I \ D \ V \ (\text{ipurge-tr-aux-foldr } I \ D \ P \ V \ xs) = ?ys$
 $(\text{is } ?zs = -)$
by $(\text{rule ipurge-tr-aux-foldr-subset})$
ultimately show $\text{sinks-aux } I \ D \ U \ ?zs = U$
by simp
qed
qed

lemma $\text{ipurge-tr-aux-foldr-ref-aux}$:
assumes $A: U \subseteq V$
shows $\text{ipurge-ref-aux } I \ D \ U \ (\text{ipurge-tr-aux-foldr } I \ D \ P \ V \ xs) \ X =$
 $\text{ipurge-ref-aux } I \ D \ U \ [] \ X$
by $(\text{simp add: ipurge-ref-aux-def ipurge-tr-aux-foldr-sinks-aux } [OF \ A])$

lemma $\text{ipurge-ref-aux-foldr-subset } [rule-format]$:
 $\text{sinks-aux } I \ D \ U \ ys \subseteq V \longrightarrow$
 $\text{ipurge-ref-aux } I \ D \ U \ ys \ (\text{ipurge-ref-aux-foldr } I \ D \ P \ V \ xs \ X) =$
 $\text{ipurge-ref-aux-foldr } I \ D \ P \ V \ xs \ X$
proof $(\text{induction } xs \ \text{arbitrary: } ys \ U \ V, \ \text{rule-tac } [!]) \ \text{impI}$,
 $\text{simp add: ipurge-ref-aux-def, blast}$
fix $x \ xs \ ys \ U \ V$
assume
 $A: \bigwedge ys \ U \ V.$
 $\text{sinks-aux } I \ D \ U \ ys \subseteq V \longrightarrow$
 $\text{ipurge-ref-aux } I \ D \ U \ ys \ (\text{ipurge-ref-aux-foldr } I \ D \ P \ V \ xs \ X) =$
 $\text{ipurge-ref-aux-foldr } I \ D \ P \ V \ xs \ X$ **and**
 $B: \text{sinks-aux } I \ D \ U \ ys \subseteq V$
show $\text{ipurge-ref-aux } I \ D \ U \ ys \ (\text{ipurge-ref-aux-foldr } I \ D \ P \ V \ (x \ \# \ xs) \ X) =$

$ipurge-ref-aux-foldr\ I\ D\ P\ V\ (x\ \# \ xs)\ X$
proof (*cases* $P\ x$, *simp-all* *add*: $ipurge-ref-aux-cons$)
have C : $sinks-aux\ I\ D\ V\ [x] \subseteq sinks-aux\ I\ D\ V\ [x] ..$
show
 $ipurge-ref-aux\ I\ D\ U\ ys\ (ipurge-ref-aux\ I\ D\ (sinks-aux\ I\ D\ V\ [x])$
 $(ipurge-tr-aux-foldr\ I\ D\ P\ (sinks-aux\ I\ D\ V\ [x])\ xs)$
 $(ipurge-ref-aux-foldr\ I\ D\ P\ (sinks-aux\ I\ D\ V\ [x])\ xs\ X)) =$
 $ipurge-ref-aux\ I\ D\ (sinks-aux\ I\ D\ V\ [x])$
 $(ipurge-tr-aux-foldr\ I\ D\ P\ (sinks-aux\ I\ D\ V\ [x])\ xs)$
 $(ipurge-ref-aux-foldr\ I\ D\ P\ (sinks-aux\ I\ D\ V\ [x])\ xs\ X)$
proof (*simp* *add*: $ipurge-tr-aux-foldr-ref-aux\ [OF\ C]$)
have $sinks-aux\ I\ D\ (sinks-aux\ I\ D\ V\ [x])\ [] \subseteq sinks-aux\ I\ D\ V\ [x] \longrightarrow$
 $ipurge-ref-aux\ I\ D\ (sinks-aux\ I\ D\ V\ [x])\ []$
 $(ipurge-ref-aux-foldr\ I\ D\ P\ (sinks-aux\ I\ D\ V\ [x])\ xs\ X) =$
 $ipurge-ref-aux-foldr\ I\ D\ P\ (sinks-aux\ I\ D\ V\ [x])\ xs\ X$
 $(is\ ?A \longrightarrow ?us = ?vs)$
using A .
moreover **have** $?A$
by *simp*
ultimately **have** $?us = ?vs ..$
thus $ipurge-ref-aux\ I\ D\ U\ ys\ ?us = ?us$
proof *simp*
have $sinks-aux\ I\ D\ U\ ys \subseteq sinks-aux\ I\ D\ V\ [x] \longrightarrow$
 $ipurge-ref-aux\ I\ D\ U\ ys$
 $(ipurge-ref-aux-foldr\ I\ D\ P\ (sinks-aux\ I\ D\ V\ [x])\ xs\ X) =$
 $ipurge-ref-aux-foldr\ I\ D\ P\ (sinks-aux\ I\ D\ V\ [x])\ xs\ X$
 $(is\ - \longrightarrow ?T)$
using A .
moreover **have** $V \subseteq sinks-aux\ I\ D\ V\ [x]$
by (*rule* $sinks-aux-subset$)
hence $sinks-aux\ I\ D\ U\ ys \subseteq sinks-aux\ I\ D\ V\ [x]$
using B **by** *simp*
ultimately **show** $?T ..$
qed
qed
next
have C : $V \subseteq sinks-aux\ I\ D\ V\ [x]$
by (*rule* $sinks-aux-subset$)
show
 $ipurge-ref-aux\ I\ D\ U\ ys\ (ipurge-ref-aux\ I\ D\ V$
 $(ipurge-tr-aux-foldr\ I\ D\ P\ (sinks-aux\ I\ D\ V\ [x])\ xs)$
 $(ipurge-ref-aux-foldr\ I\ D\ P\ (sinks-aux\ I\ D\ V\ [x])\ xs\ X)) =$
 $ipurge-ref-aux\ I\ D\ V$
 $(ipurge-tr-aux-foldr\ I\ D\ P\ (sinks-aux\ I\ D\ V\ [x])\ xs)$
 $(ipurge-ref-aux-foldr\ I\ D\ P\ (sinks-aux\ I\ D\ V\ [x])\ xs\ X)$
proof (*simp* *add*: $ipurge-tr-aux-foldr-ref-aux\ [OF\ C]$)
have $sinks-aux\ I\ D\ V\ [] \subseteq sinks-aux\ I\ D\ V\ [x] \longrightarrow$
 $ipurge-ref-aux\ I\ D\ V\ []$
 $(ipurge-ref-aux-foldr\ I\ D\ P\ (sinks-aux\ I\ D\ V\ [x])\ xs\ X) =$

ipurge-ref-aux-foldr $I D P$ (*sinks-aux* $I D V [x]$) $xs X$
 (is $?A \longrightarrow ?us = ?vs$)
 using A .
 moreover have $?A$
 using C by *simp*
 ultimately have $?us = ?vs$..
 thus *ipurge-ref-aux* $I D U ys$ $?us = ?us$
 proof *simp*
 have *sinks-aux* $I D U ys \subseteq$ *sinks-aux* $I D V [x] \longrightarrow$
ipurge-ref-aux $I D U ys$
 (*ipurge-ref-aux-foldr* $I D P$ (*sinks-aux* $I D V [x]$) $xs X$) =
ipurge-ref-aux-foldr $I D P$ (*sinks-aux* $I D V [x]$) $xs X$
 (is $- \longrightarrow ?T$)
 using A .
 moreover have *sinks-aux* $I D U ys \subseteq$ *sinks-aux* $I D V [x]$
 using B and C by *simp*
 ultimately show $?T$..
 qed
 qed
 qed
 qed

lemma *ipurge-ref-aux-foldr-eq*:

ipurge-ref-aux $I D U xs X =$ *ipurge-ref-aux-foldr* $I D P U xs X$

proof (*induction* xs arbitrary: U , *simp*)

fix $x xs U$

assume A : $\bigwedge U$. *ipurge-ref-aux* $I D U xs X =$ *ipurge-ref-aux-foldr* $I D P U xs X$

show *ipurge-ref-aux* $I D U (x \# xs) X =$

ipurge-ref-aux-foldr $I D P U (x \# xs) X$

proof (*cases* $P x$, *simp-all* add: *ipurge-ref-aux-cons*)

have *sinks-aux* $I D U [x] \subseteq$ *sinks-aux* $I D U [x]$..

hence

ipurge-ref-aux $I D$ (*sinks-aux* $I D U [x]$)

(*ipurge-tr-aux-foldr* $I D P$ (*sinks-aux* $I D U [x]$) xs)

(*ipurge-ref-aux-foldr* $I D P$ (*sinks-aux* $I D U [x]$) $xs X$) =

ipurge-ref-aux $I D$ (*sinks-aux* $I D U [x]$) []

(*ipurge-ref-aux-foldr* $I D P$ (*sinks-aux* $I D U [x]$) $xs X$)

(is *ipurge-ref-aux* $- - - ?xs' ?X' = -$)

by (rule *ipurge-tr-aux-foldr-ref-aux*)

also have *sinks-aux* $I D$ (*sinks-aux* $I D U [x]$) [] \subseteq *sinks-aux* $I D U [x]$

by *simp*

hence *ipurge-ref-aux* $I D$ (*sinks-aux* $I D U [x]$) [] $?X' = ?X'$

by (rule *ipurge-ref-aux-foldr-subset*)

finally have *ipurge-ref-aux* $I D$ (*sinks-aux* $I D U [x]$) $?xs' ?X' = ?X'$.

thus *ipurge-ref-aux* $I D$ (*sinks-aux* $I D U [x]$) $xs X =$

ipurge-ref-aux $I D$ (*sinks-aux* $I D U [x]$) $?xs' ?X'$

proof *simp*

show *ipurge-ref-aux* $I D$ (*sinks-aux* $I D U [x]$) $xs X =$

ipurge-ref-aux-foldr $I D P$ (*sinks-aux* $I D U [x]$) $xs X$

```

    using A .
  qed
next
  have  $U \subseteq \text{sinks-aux } I D U [x]$ 
  by (rule sinks-aux-subset)
  hence
    ipurge-ref-aux  $I D U$ 
      (ipurge-tr-aux-foldr  $I D P$  (sinks-aux  $I D U [x]$ )  $xs$ )
      (ipurge-ref-aux-foldr  $I D P$  (sinks-aux  $I D U [x]$ )  $xs X$ ) =
    ipurge-ref-aux  $I D U []$ 
      (ipurge-ref-aux-foldr  $I D P$  (sinks-aux  $I D U [x]$ )  $xs X$ )
    (is ipurge-ref-aux - - - ? $xs'$  ? $X' = -$ )
  by (rule ipurge-tr-aux-foldr-ref-aux)
  also have  $\text{sinks-aux } I D U [] \subseteq \text{sinks-aux } I D U [x]$ 
  by (simp, rule sinks-aux-subset)
  hence ipurge-ref-aux  $I D U []$  ? $X' = ?X'$ 
  by (rule ipurge-ref-aux-foldr-subset)
  finally have ipurge-ref-aux  $I D U$  ? $xs'$  ? $X' = ?X'$  .
  thus ipurge-ref-aux  $I D$  (sinks-aux  $I D U [x]$ )  $xs X =$ 
    ipurge-ref-aux  $I D U$  ? $xs'$  ? $X'$ 
  proof simp
    show ipurge-ref-aux  $I D$  (sinks-aux  $I D U [x]$ )  $xs X =$ 
      ipurge-ref-aux-foldr  $I D P$  (sinks-aux  $I D U [x]$ )  $xs X$ 
    using A .
  qed
qed
qed
qed

```

Finally, here below is the proof of the implication involving functions *ipurge-tr-aux-foldr* and *ipurge-ref-aux-foldr* discussed above.

lemma *con-comp-sinks-aux-range*:

assumes

A: $U \subseteq \text{range } \text{Some}$ **and**

B: $\text{set } xs \subseteq \text{range } p \cup \text{range } q$

shows $\text{sinks-aux } (\text{con-comp-pol } I) (\text{con-comp-map } D E p q) U xs \subseteq \text{range } \text{Some}$

(is $\text{sinks-aux } - ?D' - - \subseteq -$)

proof (rule subsetI, drule sinks-aux-elem, erule disjE, erule-tac [2] bexE)

fix u

assume $u \in U$

with *A* **show** $u \in \text{range } \text{Some} ..$

next

fix $u x$

assume $x \in \text{set } xs$

with *B* **have** $x \in \text{range } p \cup \text{range } q ..$

hence ? $D' x \in \text{range } \text{Some}$

by (cases $x \in \text{range } p$, simp-all)

moreover **assume** $u = ?D' x$

ultimately show $u \in \text{range } \text{Some}$
 by *simp*
 qed

lemma *con-comp-sinks-aux* [rule-format]:

assumes $A: U \subseteq \text{range } \text{Some}$

shows $\text{set } xs \subseteq \text{range } p \longrightarrow$

$\text{sinks-aux } I D (\text{the } ' U) (\text{map } (\text{inv } p) xs) =$
 $\text{the } ' \text{sinks-aux } (\text{con-comp-pol } I) (\text{con-comp-map } D E p q) U xs$
 (is $- \longrightarrow - = \text{the } ' \text{sinks-aux } ?I' ?D' - -$)

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

fix $x xs$

assume $\text{set } xs \subseteq \text{range } p \longrightarrow$

$\text{sinks-aux } I D (\text{the } ' U) (\text{map } (\text{inv } p) xs) =$
 $\text{the } ' \text{sinks-aux } ?I' ?D' U xs$

moreover assume $B: \text{set } (xs @ [x]) \subseteq \text{range } p$

ultimately have $C: \text{sinks-aux } I D (\text{the } ' U) (\text{map } (\text{inv } p) xs) =$
 $\text{the } ' \text{sinks-aux } ?I' ?D' U xs$

by *simp*

show $\text{sinks-aux } I D (\text{the } ' U) (\text{map } (\text{inv } p) (xs @ [x])) =$
 $\text{the } ' \text{sinks-aux } ?I' ?D' U (xs @ [x])$

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

case *True*

then obtain u **where**

$D: u \in \text{sinks-aux } ?I' ?D' U xs$ **and** $E: (u, ?D' x) \in ?I' ..$

have $(\text{the } u, D (\text{inv } p x)) \in I$

using B **and** E **by** (*simp add: con-comp-pol-def, erule-tac exE, simp*)

moreover have $u \in \text{the } ' \text{sinks-aux } ?I' ?D' U xs$

using D **by** *simp*

hence $u \in \text{sinks-aux } I D (\text{the } ' U) (\text{map } (\text{inv } p) xs)$

using C **by** *simp*

ultimately have $\exists d \in \text{sinks-aux } I D (\text{the } ' U) (\text{map } (\text{inv } p) xs).$
 $(d, D (\text{inv } p x)) \in I ..$

hence $\text{sinks-aux } I D (\text{the } ' U) (\text{map } (\text{inv } p) (xs @ [x])) =$
 $\text{insert } (D (\text{inv } p x)) (\text{sinks-aux } I D (\text{the } ' U) (\text{map } (\text{inv } p) xs))$

by *simp*

thus $\text{sinks-aux } I D (\text{the } ' U) (\text{map } (\text{inv } p) (xs @ [x])) =$
 $\text{insert } (\text{the } (?D' x)) (\text{the } ' \text{sinks-aux } ?I' ?D' U xs)$

using B **and** C **by** *simp*

next

case *False*

have $\neg (\exists d \in \text{sinks-aux } I D (\text{the } ' U) (\text{map } (\text{inv } p) xs).$
 $(d, D (\text{inv } p x)) \in I)$

proof (*rule notI, erule bexE*)

fix d

assume $d \in \text{sinks-aux } I D (\text{the } ' U) (\text{map } (\text{inv } p) xs)$

hence $d \in \text{the } ' \text{sinks-aux } ?I' ?D' U xs$

using C **by** *simp*

hence $\exists u \in \text{sinks-aux } ?I' ?D' U xs$. $d = \text{the } u$
by (*simp add: image-iff*)
then obtain u **where**
 $D: u \in \text{sinks-aux } ?I' ?D' U xs$ **and** $E: d = \text{the } u$..
have $set\ xs \subseteq \text{range } p \cup \text{range } q$
using B **by** (*simp, blast*)
with A **have** $\text{sinks-aux } ?I' ?D' U xs \subseteq \text{range } Some$
by (*rule con-comp-sinks-aux-range*)
hence $u \in \text{range } Some$
using D ..
hence $u = Some\ d$
using E **by** (*simp add: image-iff*)
moreover assume $(d, D\ (inv\ p\ x)) \in I$
hence $(Some\ d, Some\ (D\ (inv\ p\ x))) \in ?I'$
by (*simp add: con-comp-pol-def*)
ultimately have $(u, ?D' x) \in ?I'$
using B **by** *simp*
hence $\exists u \in \text{sinks-aux } ?I' ?D' U xs$. $(u, ?D' x) \in ?I'$
using D ..
thus *False*
using *False* **by** *contradiction*
qed
thus $\text{sinks-aux } I\ D\ (\text{the } 'U)\ (\text{map } (inv\ p)\ (xs\ @\ [x])) =$
 $\text{the } ' \text{sinks-aux } ?I' ?D' U xs$
using C **by** *simp*
qed
qed

lemma *con-comp-ipurge-tr-aux* [*rule-format*]:

assumes $A: U \subseteq \text{range } Some$
shows $set\ xs \subseteq \text{range } p \longrightarrow$
 $ipurge-tr-aux\ I\ D\ (\text{the } 'U)\ (\text{map } (inv\ p)\ xs) =$
 $\text{map } (inv\ p)\ (ipurge-tr-aux\ (\text{con-comp-pol } I)\ (\text{con-comp-map } D\ E\ p\ q)\ U\ xs)$
 $(\text{is } - \longrightarrow - = \text{map } (inv\ p)\ (ipurge-tr-aux\ ?I' ?D' - -))$
proof (*induction xs rule: rev-induct, simp, rule impI*)
fix $x\ xs$
assume $set\ xs \subseteq \text{range } p \longrightarrow$
 $ipurge-tr-aux\ I\ D\ (\text{the } 'U)\ (\text{map } (inv\ p)\ xs) =$
 $\text{map } (inv\ p)\ (ipurge-tr-aux\ ?I' ?D' U\ xs)$
moreover assume $B: set\ (xs\ @\ [x]) \subseteq \text{range } p$
ultimately have $C: ipurge-tr-aux\ I\ D\ (\text{the } 'U)\ (\text{map } (inv\ p)\ xs) =$
 $\text{map } (inv\ p)\ (ipurge-tr-aux\ ?I' ?D' U\ xs)$
by *simp*
show $ipurge-tr-aux\ I\ D\ (\text{the } 'U)\ (\text{map } (inv\ p)\ (xs\ @\ [x])) =$
 $\text{map } (inv\ p)\ (ipurge-tr-aux\ ?I' ?D' U\ (xs\ @\ [x]))$
proof (*cases* $\exists u \in \text{sinks-aux } ?I' ?D' U xs$. $(u, ?D' x) \in ?I'$)
case *True*
then obtain u **where**
 $D: u \in \text{sinks-aux } ?I' ?D' U xs$ **and** $E: (u, ?D' x) \in ?I'$..

have $(\text{the } u, D (\text{inv } p \ x)) \in I$
using B **and** E **by** $(\text{simp add: con-comp-pol-def, erule-tac exE, simp})$
moreover have F : $\text{the } u \in \text{the ' sinks-aux ?I' ?D' U } xs$
using D **by** simp
have $\text{set } xs \subseteq \text{range } p$
using B **by** simp
with A **have** $\text{sinks-aux I D (the ' U) (map (inv } p) \ xs) =$
 $\text{the ' sinks-aux ?I' ?D' U } xs$
by $(\text{rule con-comp-sinks-aux})$
hence $\text{the } u \in \text{sinks-aux I D (the ' U) (map (inv } p) \ xs)$
using F **by** simp
ultimately have $\exists d \in \text{sinks-aux I D (the ' U) (map (inv } p) \ xs).$
 $(d, D (\text{inv } p \ x)) \in I ..$
hence $\text{ipurge-tr-aux I D (the ' U) (map (inv } p) \ (xs \ @ \ [x])) =$
 $\text{ipurge-tr-aux I D (the ' U) (map (inv } p) \ xs)$
by simp
moreover have $\text{map (inv } p) (\text{ipurge-tr-aux ?I' ?D' U } (xs \ @ \ [x])) =$
 $\text{map (inv } p) (\text{ipurge-tr-aux ?I' ?D' U } xs)$
using True **by** simp
ultimately show $?thesis$
using C **by** simp

next

case False

have $\neg (\exists d \in \text{sinks-aux I D (the ' U) (map (inv } p) \ xs).$
 $(d, D (\text{inv } p \ x)) \in I)$

proof $(\text{rule notI, erule bexE})$

fix d

assume $d \in \text{sinks-aux I D (the ' U) (map (inv } p) \ xs)$

moreover have $\text{set } xs \subseteq \text{range } p$

using B **by** simp

with A **have** $\text{sinks-aux I D (the ' U) (map (inv } p) \ xs) =$
 $\text{the ' sinks-aux ?I' ?D' U } xs$
by $(\text{rule con-comp-sinks-aux})$

ultimately have $d \in \text{the ' sinks-aux ?I' ?D' U } xs$

by simp

hence $\exists u \in \text{sinks-aux ?I' ?D' U } xs. d = \text{the } u$

by $(\text{simp add: image-iff})$

then obtain u **where**

$D: u \in \text{sinks-aux ?I' ?D' U } xs$ **and** $E: d = \text{the } u ..$

have $\text{set } xs \subseteq \text{range } p \cup \text{range } q$

using B **by** (simp, blast)

with A **have** $\text{sinks-aux ?I' ?D' U } xs \subseteq \text{range } \text{Some}$

by $(\text{rule con-comp-sinks-aux-range})$

hence $u \in \text{range } \text{Some}$

using $D ..$

hence $u = \text{Some } d$

using E **by** $(\text{simp add: image-iff})$

moreover assume $(d, D (\text{inv } p \ x)) \in I$

hence $(\text{Some } d, \text{Some } (D (\text{inv } p \ x))) \in ?I'$

by (*simp add: con-comp-pol-def*)
 ultimately have $(u, ?D' x) \in ?I'$
 using *B* by *simp*
 hence $\exists u \in \text{sinks-aux } ?I' ?D' U \text{ xs. } (u, ?D' x) \in ?I'$
 using *D* ..
 thus *False*
 using *False* by *contradiction*
 qed
 hence *ipurge-tr-aux* *I D* (*the* ' *U*) (*map* (*inv p*) (*xs* @ [*x*])) =
ipurge-tr-aux *I D* (*the* ' *U*) (*map* (*inv p*) *xs*) @ [*inv p x*]
 by *simp*
 moreover have *map* (*inv p*) (*ipurge-tr-aux* *?I' ?D' U* (*xs* @ [*x*])) =
map (*inv p*) (*ipurge-tr-aux* *?I' ?D' U* *xs*) @ [*inv p x*]
 using *False* by *simp*
 ultimately show *?thesis*
 using *C* by *simp*
 qed
 qed

lemma *con-comp-ipurge-ref-aux*:
 assumes
 A: $U \subseteq \text{range } \text{Some}$ and
 B: $\text{set } \text{xs} \subseteq \text{range } p$ and
 C: $X \subseteq \text{range } p$
 shows *ipurge-ref-aux* *I D* (*the* ' *U*) (*map* (*inv p*) *xs*) (*inv p* ' *X*) =
inv p ' *ipurge-ref-aux* (*con-comp-pol* *I*) (*con-comp-map* *D E p q*) *U* *xs* *X*
 (*is* - = *inv p* ' *ipurge-ref-aux* *?I' ?D' - - -*)
proof (*simp add: ipurge-ref-aux-def set-eq-iff image-iff, rule allI, rule iffI,*
erule conjE, erule bexE, erule-tac [2] exE, (erule-tac [2] conjE)+)
 fix *a x*
 assume
 D: $x \in X$ and
 E: $a = \text{inv } p \ x$ and
 F: $\forall d \in \text{sinks-aux } I D (\text{the } ' U) (\text{map } (\text{inv } p) \text{xs}). (d, D a) \notin I$
 show $\exists x. x \in X \wedge (\forall u \in \text{sinks-aux } ?I' ?D' U \text{ xs. } (u, ?D' x) \notin ?I') \wedge$
 $a = \text{inv } p \ x$
proof (*rule-tac* $x = x$ **in** *exI, simp add: D E, rule ballI*)
 fix *u*
 assume *G*: $u \in \text{sinks-aux } ?I' ?D' U \text{ xs}$
 moreover have *sinks-aux* *I D* (*the* ' *U*) (*map* (*inv p*) *xs*) =
the ' *sinks-aux* *?I' ?D' U* *xs*
 using *A* and *B* by (*rule con-comp-sinks-aux*)
 ultimately have *the* $u \in \text{sinks-aux } I D (\text{the } ' U) (\text{map } (\text{inv } p) \text{xs})$
 by *simp*
 with *F* have (*the* $u, D a$) $\notin I$..
 moreover have $\text{set } \text{xs} \subseteq \text{range } p \cup \text{range } q$
 using *B* by *blast*
 with *A* have *sinks-aux* *?I' ?D' U* *xs* $\subseteq \text{range } \text{Some}$
 by (*rule con-comp-sinks-aux-range*)

```

hence  $u \in \text{range } \text{Some}$ 
using  $G$  ..
hence  $\exists d. u = \text{Some } d$ 
by (simp add: image-iff)
then obtain  $d$  where  $H: u = \text{Some } d$  ..
ultimately have  $(d, D (\text{inv } p \ x)) \notin I$ 
using  $E$  by simp
hence  $(u, \text{Some } (D (\text{inv } p \ x))) \notin ?I'$ 
using  $H$  by (simp add: con-comp-pol-def)
moreover have  $x \in \text{range } p$ 
using  $C$  and  $D$  ..
ultimately show  $(u, ?D' \ x) \notin ?I'$ 
by simp
qed
next
fix  $a \ x$ 
assume
   $D: x \in X$  and
   $E: a = \text{inv } p \ x$  and
   $F: \forall u \in \text{sinks-aux } ?I' \ ?D' \ U \ xs. (u, ?D' \ x) \notin ?I'$ 
show  $(\exists x \in X. a = \text{inv } p \ x) \wedge$ 
   $(\forall u \in \text{sinks-aux } I \ D \ (\text{the } 'U) \ (\text{map } (\text{inv } p) \ xs). (u, D \ a) \notin I)$ 
proof (rule conjI, rule-tac [2] ballI)
  show  $\exists x \in X. a = \text{inv } p \ x$ 
  using  $E$  and  $D$  ..
next
fix  $d$ 
assume  $d \in \text{sinks-aux } I \ D \ (\text{the } 'U) \ (\text{map } (\text{inv } p) \ xs)$ 
moreover have  $\text{sinks-aux } I \ D \ (\text{the } 'U) \ (\text{map } (\text{inv } p) \ xs) =$ 
   $\text{the } ' \text{sinks-aux } ?I' \ ?D' \ U \ xs$ 
using  $A$  and  $B$  by (rule con-comp-sinks-aux)
ultimately have  $d \in \text{the } ' \text{sinks-aux } ?I' \ ?D' \ U \ xs$ 
by simp
hence  $\exists u \in \text{sinks-aux } ?I' \ ?D' \ U \ xs. d = \text{the } u$ 
by (simp add: image-iff)
then obtain  $u$  where  $G: u \in \text{sinks-aux } ?I' \ ?D' \ U \ xs$  and  $H: d = \text{the } u$  ..
have  $(u, ?D' \ x) \notin ?I'$ 
using  $F$  and  $G$  ..
moreover have  $\text{set } xs \subseteq \text{range } p \cup \text{range } q$ 
using  $B$  by blast
with  $A$  have  $\text{sinks-aux } ?I' \ ?D' \ U \ xs \subseteq \text{range } \text{Some}$ 
by (rule con-comp-sinks-aux-range)
hence  $u \in \text{range } \text{Some}$ 
using  $G$  ..
hence  $u = \text{Some } d$ 
using  $H$  by (simp add: image-iff)
moreover have  $x \in \text{range } p$ 
using  $C$  and  $D$  ..
ultimately have  $(d, D (\text{inv } p \ x)) \notin I$ 

```

by (*simp add: con-comp-pol-def*)
 thus ($d, D a \notin I$)
 using E by *simp*
 qed
 qed

lemma *con-comp-sinks-filter*:
 $\text{sinks } (\text{con-comp-pol } I) (\text{con-comp-map } D E p q) u$
 $[x \leftarrow xs. x \in \text{range } p \cup \text{range } q] =$
 $\text{sinks } (\text{con-comp-pol } I) (\text{con-comp-map } D E p q) u xs \cap \text{range } \text{Some}$
 $(\text{is sinks } ?I' ?D' - - = -)$

proof (*induction xs rule: rev-induct, simp*)
 fix $x xs$
 assume $A: \text{sinks } ?I' ?D' u [x \leftarrow xs. x \in \text{range } p \cup \text{range } q] =$
 $\text{sinks } ?I' ?D' u xs \cap \text{range } \text{Some}$
 $(\text{is sinks } - - - ?xs' = -)$
 show $\text{sinks } ?I' ?D' u [x \leftarrow xs @ [x]. x \in \text{range } p \cup \text{range } q] =$
 $\text{sinks } ?I' ?D' u (xs @ [x]) \cap \text{range } \text{Some}$
proof (*cases $x \in \text{range } p \cup \text{range } q$, simp-all del: Un-iff sinks.simps,*
cases $(u, ?D' x) \in ?I' \vee (\exists v \in \text{sinks } ?I' ?D' u ?xs'. (v, ?D' x) \in ?I')$)
 assume
 $B: x \in \text{range } p \cup \text{range } q$ and
 $C: (u, ?D' x) \in ?I' \vee (\exists v \in \text{sinks } ?I' ?D' u ?xs'. (v, ?D' x) \in ?I')$
 have $\text{sinks } ?I' ?D' u (?xs' @ [x]) =$
 $\text{insert } (?D' x) (\text{sinks } ?I' ?D' u ?xs')$
 using C by *simp*
 also have ... =
 $\text{insert } (?D' x) (\text{sinks } ?I' ?D' u xs \cap \text{range } \text{Some})$
 using A by *simp*
 also have ... =
 $\text{insert } (?D' x) (\text{sinks } ?I' ?D' u xs) \cap \text{insert } (?D' x) (\text{range } \text{Some})$
 by *simp*
 finally have $\text{sinks } ?I' ?D' u (?xs' @ [x]) =$
 $\text{insert } (?D' x) (\text{sinks } ?I' ?D' u xs) \cap \text{insert } (?D' x) (\text{range } \text{Some})$.
 moreover have $\text{insert } (?D' x) (\text{range } \text{Some}) = \text{range } \text{Some}$
 using B by (*rule-tac insert-absorb, cases $x \in \text{range } p$, simp-all*)
 ultimately have $\text{sinks } ?I' ?D' u (?xs' @ [x]) =$
 $\text{insert } (?D' x) (\text{sinks } ?I' ?D' u xs) \cap \text{range } \text{Some}$
 by *simp*
 moreover have $(u, ?D' x) \in ?I' \vee$
 $(\exists v \in \text{sinks } ?I' ?D' u xs. (v, ?D' x) \in ?I')$
 using A and C by (*simp, blast*)
 ultimately show $\text{sinks } ?I' ?D' u (?xs' @ [x]) =$
 $\text{sinks } ?I' ?D' u (xs @ [x]) \cap \text{range } \text{Some}$
 by *simp*
 next
 assume
 $B: x \in \text{range } p \cup \text{range } q$ and
 $C: \neg ((u, ?D' x) \in ?I' \vee (\exists v \in \text{sinks } ?I' ?D' u ?xs'. (v, ?D' x) \in ?I'))$

have $\text{sinks } ?I' ?D' u (?xs' @ [x]) = \text{sinks } ?I' ?D' u ?xs'$
using C **by** *simp*
hence $\text{sinks } ?I' ?D' u (?xs' @ [x]) = \text{sinks } ?I' ?D' u xs \cap \text{range } \text{Some}$
using A **by** *simp*
moreover from C **have**
 $\neg ((u, ?D' x) \in ?I' \vee (\exists v \in \text{sinks } ?I' ?D' u xs. (v, ?D' x) \in ?I'))$
proof (*rule-tac notI, simp del: bex-simps*)
assume $\exists v \in \text{sinks } ?I' ?D' u xs. (v, ?D' x) \in ?I'$
then obtain v **where** $E: v \in \text{sinks } ?I' ?D' u xs$ **and** $F: (v, ?D' x) \in ?I' ..$
have $\exists d. ?D' x = \text{Some } d$
using B **by** (*cases x \in range p, simp-all*)
then obtain d **where** $?D' x = \text{Some } d ..$
hence $(v, \text{Some } d) \in ?I'$
using F **by** *simp*
hence $v \in \text{range } \text{Some}$
by (*cases v, simp-all add: con-comp-pol-def*)
with E **have** $v \in \text{sinks } ?I' ?D' u xs \cap \text{range } \text{Some} ..$
hence $v \in \text{sinks } ?I' ?D' u ?xs'$
using A **by** *simp*
with F **have** $\exists v \in \text{sinks } ?I' ?D' u ?xs'. (v, ?D' x) \in ?I' ..$
thus *False*
using C **by** *simp*

qed

ultimately show $\text{sinks } ?I' ?D' u (?xs' @ [x]) =$
 $\text{sinks } ?I' ?D' u (xs @ [x]) \cap \text{range } \text{Some}$
by *simp*

next

assume $B: x \notin \text{range } p \cup \text{range } q$
hence $(u, ?D' x) \in ?I'$
by (*simp add: con-comp-pol-def*)
hence $\text{sinks } ?I' ?D' u (xs @ [x]) = \text{insert } (?D' x) (\text{sinks } ?I' ?D' u xs)$
by *simp*
moreover have $\text{insert } (?D' x) (\text{sinks } ?I' ?D' u xs) \cap \text{range } \text{Some} =$
 $\text{sinks } ?I' ?D' u xs \cap \text{range } \text{Some}$
using B **by** *simp*
ultimately have $\text{sinks } ?I' ?D' u (xs @ [x]) \cap \text{range } \text{Some} =$
 $\text{sinks } ?I' ?D' u xs \cap \text{range } \text{Some}$
by *simp*
thus $\text{sinks } ?I' ?D' u ?xs' = \text{sinks } ?I' ?D' u (xs @ [x]) \cap \text{range } \text{Some}$
using A **by** *simp*

qed

qed

lemma *con-comp-ipurge-tr-filter:*

$\text{ipurge-tr } (\text{con-comp-pol } I) (\text{con-comp-map } D E p q) u$
 $[\text{x} \leftarrow xs. x \in \text{range } p \cup \text{range } q] =$
 $\text{ipurge-tr } (\text{con-comp-pol } I) (\text{con-comp-map } D E p q) u xs$
(is ipurge-tr ?I' ?D' - - = -)

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

fix $x\ xs$
assume $A: \text{ipurge-tr } ?I' ?D' u [x \leftarrow xs. x \in \text{range } p \cup \text{range } q] =$
 $\text{ipurge-tr } ?I' ?D' u\ xs$
(is ipurge-tr - - - ?xs' = -)
show $\text{ipurge-tr } ?I' ?D' u [x \leftarrow xs @ [x]. x \in \text{range } p \cup \text{range } q] =$
 $\text{ipurge-tr } ?I' ?D' u (xs @ [x])$
proof *(cases $x \in \text{range } p \cup \text{range } q$, simp-all del: Un-iff ipurge-tr.simps,*
cases $?D' x \in \text{sinks } ?I' ?D' u (?xs' @ [x])$)
assume
 $B: x \in \text{range } p \cup \text{range } q$ **and**
 $C: ?D' x \in \text{sinks } ?I' ?D' u (?xs' @ [x])$
have $\text{ipurge-tr } ?I' ?D' u (?xs' @ [x]) = \text{ipurge-tr } ?I' ?D' u\ ?xs'$
using C **by** *simp*
hence $\text{ipurge-tr } ?I' ?D' u (?xs' @ [x]) = \text{ipurge-tr } ?I' ?D' u\ xs$
using A **by** *simp*
moreover have $?D' x \in \text{sinks } ?I' ?D' u [x \leftarrow xs @ [x]. x \in \text{range } p \cup \text{range } q]$
using B **and** C **by** *simp*
hence $?D' x \in \text{sinks } ?I' ?D' u (xs @ [x])$
by *(simp only: con-comp-sinks-filter, blast)*
ultimately show $\text{ipurge-tr } ?I' ?D' u (?xs' @ [x]) =$
 $\text{ipurge-tr } ?I' ?D' u (xs @ [x])$
by *simp*
next
assume
 $B: x \in \text{range } p \cup \text{range } q$ **and**
 $C: \neg (?D' x \in \text{sinks } ?I' ?D' u (?xs' @ [x]))$
have $\text{ipurge-tr } ?I' ?D' u (?xs' @ [x]) = \text{ipurge-tr } ?I' ?D' u\ ?xs' @ [x]$
using C **by** *simp*
hence $\text{ipurge-tr } ?I' ?D' u (?xs' @ [x]) = \text{ipurge-tr } ?I' ?D' u\ xs @ [x]$
using A **by** *simp*
moreover have $?D' x \notin \text{sinks } ?I' ?D' u [x \leftarrow xs @ [x]. x \in \text{range } p \cup \text{range } q]$
using B **and** C **by** *simp*
hence $?D' x \notin \text{sinks } ?I' ?D' u (xs @ [x]) \cap \text{range } \text{Some}$
by *(simp only: con-comp-sinks-filter, simp)*
hence $?D' x \notin \text{sinks } ?I' ?D' u (xs @ [x])$
using B **by** *(cases $x \in \text{range } p$, simp-all)*
ultimately show $\text{ipurge-tr } ?I' ?D' u (?xs' @ [x]) =$
 $\text{ipurge-tr } ?I' ?D' u (xs @ [x])$
by *simp*
next
assume $x \notin \text{range } p \cup \text{range } q$
hence $(u, ?D' x) \in ?I'$
by *(simp add: con-comp-pol-def)*
hence $?D' x \in \text{sinks } ?I' ?D' u (xs @ [x])$
by *simp*
thus $\text{ipurge-tr } ?I' ?D' u\ ?xs' = \text{ipurge-tr } ?I' ?D' u (xs @ [x])$
using A **by** *simp*
qed
qed

lemma *con-comp-ipurge-ref-filter*:

ipurge-ref (*con-comp-pol* *I*) (*con-comp-map* *D E p q*) *u*
 $[x \leftarrow xs. x \in \text{range } p \cup \text{range } q] X =$
ipurge-ref (*con-comp-pol* *I*) (*con-comp-map* *D E p q*) *u xs X*
(is *ipurge-ref* ?*I'* ?*D'* - - - = -)

proof (*simp* add: *ipurge-ref-def con-comp-sinks-filter set-eq-iff del: Un-iff*,
rule allI, rule iffI, simp-all, (erule conjE)+, rule ballI)
fix *x v*
assume
A: (u, ?D' x) \notin ?I' and
B: $\forall v \in \text{sinks } ?I' ?D' u xs \cap \text{range } \text{Some}. (v, ?D' x) \notin ?I' \text{ and}$
C: $v \in \text{sinks } ?I' ?D' u xs$
show $(v, ?D' x) \notin ?I'$
proof (*cases v, simp*)
have $?D' x \in \text{range } \text{Some}$
using *A* **by** (*cases ?D' x, simp-all add: con-comp-pol-def*)
thus $(\text{None}, ?D' x) \notin ?I'$
by (*simp add: image-iff con-comp-pol-def*)
next
fix *d*
assume $v = \text{Some } d$
hence $v \in \text{range } \text{Some}$
by *simp*
with *C* **have** $v \in \text{sinks } ?I' ?D' u xs \cap \text{range } \text{Some} ..$
with *B* **show** $(v, ?D' x) \notin ?I' ..$
qed
qed

lemma *con-comp-secure-aux* [*rule-format*]:
assumes
A: secure P I D and
B: $Y \subseteq \text{range } p$
shows $\text{set } ys \subseteq \text{range } p \cup \text{range } q \longrightarrow U \subseteq \text{range } \text{Some} \longrightarrow$
 $(\text{map } (\text{inv } p) [x \leftarrow xs @ ys. x \in \text{range } p], \text{inv } p \text{ ' } Y) \in \text{failures } P \longrightarrow$
 $(\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p] @$
 $\text{map } (\text{inv } p) (\text{ipurge-tr-aux-foldr } (\text{con-comp-pol } I) (\text{con-comp-map } D E p q)$
 $(\lambda x. x \in \text{range } p) U ys),$
 $\text{inv } p \text{ ' ipurge-ref-aux-foldr } (\text{con-comp-pol } I) (\text{con-comp-map } D E p q)$
 $(\lambda x. x \in \text{range } p) U ys Y) \in \text{failures } P$

proof (*induction ys arbitrary: xs U, (rule-tac [!] impI)+, simp*)
fix *xs U*
assume $(\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p], \text{inv } p \text{ ' } Y) \in \text{failures } P$
moreover have
ipurge-ref-aux (*con-comp-pol* *I*) (*con-comp-map* *D E p q*) *U [] Y* $\subseteq Y$
(is ?*Y'* \subseteq -)
by (*rule ipurge-ref-aux-subset*)
hence $\text{inv } p \text{ ' } ?Y' \subseteq \text{inv } p \text{ ' } Y$
by (*rule image-mono*)

ultimately show $(\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p], \text{inv } p \text{ ' } ?Y) \in \text{failures } P$
by (rule process-rule-3)

next

fix $y \text{ } ys \text{ } xs \text{ } U$

assume $\bigwedge xs \text{ } U. \text{set } ys \subseteq \text{range } p \cup \text{range } q \longrightarrow U \subseteq \text{range } \text{Some} \longrightarrow$

$(\text{map } (\text{inv } p) [x \leftarrow xs \text{ } @ \text{ } ys. x \in \text{range } p], \text{inv } p \text{ ' } Y) \in \text{failures } P \longrightarrow$

$(\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p] \text{ } @$

$\text{map } (\text{inv } p) (\text{ipurge-tr-aux-foldr } (\text{con-comp-pol } I) (\text{con-comp-map } D \text{ } E \text{ } p \text{ } q)$

$(\lambda x. x \in \text{range } p) \text{ } U \text{ } ys),$

$\text{inv } p \text{ ' } \text{ipurge-ref-aux-foldr } (\text{con-comp-pol } I) (\text{con-comp-map } D \text{ } E \text{ } p \text{ } q)$

$(\lambda x. x \in \text{range } p) \text{ } U \text{ } ys \text{ } Y) \in \text{failures } P$

hence $\text{set } ys \subseteq \text{range } p \cup \text{range } q \longrightarrow$

$\text{sinks-aux } (\text{con-comp-pol } I) (\text{con-comp-map } D \text{ } E \text{ } p \text{ } q) \text{ } U \text{ } [y] \subseteq \text{range } \text{Some} \longrightarrow$

$(\text{map } (\text{inv } p) [x \leftarrow (xs \text{ } @ \text{ } [y]) \text{ } @ \text{ } ys. x \in \text{range } p], \text{inv } p \text{ ' } Y) \in \text{failures } P \longrightarrow$

$(\text{map } (\text{inv } p) [x \leftarrow xs \text{ } @ \text{ } [y]. x \in \text{range } p] \text{ } @$

$\text{map } (\text{inv } p) (\text{ipurge-tr-aux-foldr } (\text{con-comp-pol } I) (\text{con-comp-map } D \text{ } E \text{ } p \text{ } q)$

$(\lambda x. x \in \text{range } p) (\text{sinks-aux } (\text{con-comp-pol } I) (\text{con-comp-map } D \text{ } E \text{ } p \text{ } q)$

$U \text{ } [y]) \text{ } ys),$

$\text{inv } p \text{ ' } \text{ipurge-ref-aux-foldr } (\text{con-comp-pol } I) (\text{con-comp-map } D \text{ } E \text{ } p \text{ } q)$

$(\lambda x. x \in \text{range } p) (\text{sinks-aux } (\text{con-comp-pol } I) (\text{con-comp-map } D \text{ } E \text{ } p \text{ } q)$

$U \text{ } [y]) \text{ } ys \text{ } Y) \in \text{failures } P$

(is $- \longrightarrow - \longrightarrow - \longrightarrow$

$(- \text{ } @ \text{ } \text{map } (\text{inv } p) (\text{ipurge-tr-aux-foldr } ?I' \text{ } ?D' \text{ } ?F \text{ } ?U' \text{ } -), -) \in -)$.

moreover assume $C: \text{set } (y \# ys) \subseteq \text{range } p \cup \text{range } q$

hence $\text{set } ys \subseteq \text{range } p \cup \text{range } q$

by *simp*

ultimately have $?U' \subseteq \text{range } \text{Some} \longrightarrow$

$(\text{map } (\text{inv } p) [x \leftarrow (xs \text{ } @ \text{ } [y]) \text{ } @ \text{ } ys. x \in \text{range } p], \text{inv } p \text{ ' } Y) \in \text{failures } P \longrightarrow$

$(\text{map } (\text{inv } p) [x \leftarrow xs \text{ } @ \text{ } [y]. x \in \text{range } p] \text{ } @$

$\text{map } (\text{inv } p) (\text{ipurge-tr-aux-foldr } ?I' \text{ } ?D' \text{ } ?F \text{ } ?U' \text{ } ys),$

$\text{inv } p \text{ ' } \text{ipurge-ref-aux-foldr } ?I' \text{ } ?D' \text{ } ?F \text{ } ?U' \text{ } ys \text{ } Y) \in \text{failures } P \text{ } ..$

moreover assume $D: U \subseteq \text{range } \text{Some}$

hence $?U' \subseteq \text{range } \text{Some}$

proof (cases $\exists u \in U. (u, ?D' \text{ } y) \in ?I'$, *simp-all add: sinks-aux-single-event*)

have $y \in \text{range } p \cup \text{range } q$

using C **by** *simp*

thus $?D' \text{ } y \in \text{range } \text{Some}$

by (cases $y \in \text{range } p$, *simp-all*)

qed

ultimately have

$(\text{map } (\text{inv } p) [x \leftarrow (xs \text{ } @ \text{ } [y]) \text{ } @ \text{ } ys. x \in \text{range } p], \text{inv } p \text{ ' } Y) \in \text{failures } P \longrightarrow$

$(\text{map } (\text{inv } p) [x \leftarrow xs \text{ } @ \text{ } [y]. x \in \text{range } p] \text{ } @$

$\text{map } (\text{inv } p) (\text{ipurge-tr-aux-foldr } ?I' \text{ } ?D' \text{ } ?F \text{ } ?U' \text{ } ys),$

$\text{inv } p \text{ ' } \text{ipurge-ref-aux-foldr } ?I' \text{ } ?D' \text{ } ?F \text{ } ?U' \text{ } ys \text{ } Y) \in \text{failures } P \text{ } ..$

moreover assume

$(\text{map } (\text{inv } p) [x \leftarrow xs \text{ } @ \text{ } y \# ys. x \in \text{range } p], \text{inv } p \text{ ' } Y) \in \text{failures } P$

ultimately have

$(\text{map } (\text{inv } p) [x \leftarrow xs \text{ } @ \text{ } [y]. x \in \text{range } p] \text{ } @$

$\text{map } (\text{inv } p) (\text{ipurge-tr-aux-foldr } ?I' \text{ } ?D' \text{ } ?F \text{ } ?U' \text{ } ys),$

$inv\ p \text{ ' } ipurge\text{-ref-}\mathit{aux}\text{-foldr}\ ?I' ?D' ?F ?U' ys\ Y) \in failures\ P$
by *simp*
hence
 $(map\ (inv\ p)\ [x \leftarrow xs.\ x \in range\ p])\ @$
 $map\ (inv\ p)\ ((if\ y \in range\ p\ then\ [y]\ else\ []))\ @$
 $ipurge\text{-tr-}\mathit{aux}\text{-foldr}\ ?I' ?D' ?F ?U' ys),$
 $inv\ p \text{ ' } ipurge\text{-ref-}\mathit{aux}\text{-foldr}\ ?I' ?D' ?F ?U' ys\ Y) \in failures\ P$
by (*cases* $y \in range\ p$, *simp-all*)
with *A* **have**
 $(map\ (inv\ p)\ [x \leftarrow xs.\ x \in range\ p])\ @$
 $ipurge\text{-tr-}\mathit{aux}\ I\ D\ (the\ \text{' } U)\ (map\ (inv\ p)\ ((if\ y \in range\ p\ then\ [y]\ else\ []))\ @$
 $ipurge\text{-tr-}\mathit{aux}\text{-foldr}\ ?I' ?D' ?F ?U' ys),$
 $ipurge\text{-ref-}\mathit{aux}\ I\ D\ (the\ \text{' } U)\ (map\ (inv\ p)\ ((if\ y \in range\ p\ then\ [y]\ else\ []))\ @$
 $ipurge\text{-tr-}\mathit{aux}\text{-foldr}\ ?I' ?D' ?F ?U' ys))$
 $(inv\ p \text{ ' } ipurge\text{-ref-}\mathit{aux}\text{-foldr}\ ?I' ?D' ?F ?U' ys\ Y) \in failures\ P$
by (*rule* *ipurge-tr-ref-aux-failures-general*)
moreover **have**
 $ipurge\text{-tr-}\mathit{aux}\ I\ D\ (the\ \text{' } U)\ (map\ (inv\ p)\ ((if\ y \in range\ p\ then\ [y]\ else\ []))\ @$
 $ipurge\text{-tr-}\mathit{aux}\text{-foldr}\ ?I' ?D' ?F ?U' ys) =$
 $map\ (inv\ p)\ (ipurge\text{-tr-}\mathit{aux}\ ?I' ?D' U\ ((if\ y \in range\ p\ then\ [y]\ else\ []))\ @$
 $ipurge\text{-tr-}\mathit{aux}\text{-foldr}\ ?I' ?D' ?F ?U' ys))$
by (*rule* *con-comp-ipurge-tr-aux*, *simp-all add*):
 $D\ ipurge\text{-tr-}\mathit{aux}\text{-foldr}\text{-eq}\ [symmetric],\ blast)$
moreover **have**
 $ipurge\text{-ref-}\mathit{aux}\ I\ D\ (the\ \text{' } U)\ (map\ (inv\ p)\ ((if\ y \in range\ p\ then\ [y]\ else\ []))\ @$
 $ipurge\text{-tr-}\mathit{aux}\text{-foldr}\ ?I' ?D' ?F ?U' ys)$
 $(inv\ p \text{ ' } ipurge\text{-ref-}\mathit{aux}\text{-foldr}\ ?I' ?D' ?F ?U' ys\ Y) =$
 $inv\ p \text{ ' } ipurge\text{-ref-}\mathit{aux}\ ?I' ?D' U\ ((if\ y \in range\ p\ then\ [y]\ else\ []))\ @$
 $ipurge\text{-tr-}\mathit{aux}\text{-foldr}\ ?I' ?D' ?F ?U' ys)$
 $(ipurge\text{-ref-}\mathit{aux}\text{-foldr}\ ?I' ?D' ?F ?U' ys\ Y)$
proof (*rule* *con-comp-ipurge-ref-aux*, *simp-all add*):
 $D\ ipurge\text{-tr-}\mathit{aux}\text{-foldr}\text{-eq}\ [symmetric]\ ipurge\text{-ref-}\mathit{aux}\text{-foldr}\text{-eq}\ [symmetric],\ blast)$
have $ipurge\text{-ref-}\mathit{aux}\ ?I' ?D' ?U' ys\ Y \subseteq Y$
by (*rule* *ipurge-ref-aux-subset*)
thus $ipurge\text{-ref-}\mathit{aux}\ ?I' ?D' ?U' ys\ Y \subseteq range\ p$
using *B* **by** *simp*
qed
ultimately **show**
 $(map\ (inv\ p)\ [x \leftarrow xs.\ x \in range\ p])\ @$
 $map\ (inv\ p)\ (ipurge\text{-tr-}\mathit{aux}\text{-foldr}\ ?I' ?D' ?F\ U\ (y\ \# ys)),$
 $inv\ p \text{ ' } ipurge\text{-ref-}\mathit{aux}\text{-foldr}\ ?I' ?D' ?F\ U\ (y\ \# ys)\ Y) \in failures\ P$
by *simp*
qed

1.4 Conservation of noninterference security under concurrent composition

Everything is now ready for proving the target security conservation theorem. It states that for any two processes P, Q being secure with respect to

the noninterference policy I and the event-domain maps D, E , their concurrent composition $P \parallel Q \langle p, q \rangle$ is secure with respect to the noninterference policy $\text{con-comp-pol } I$ and the event-domain map $\text{con-comp-map } D E p q$, provided that condition $\text{consistent-maps } D E p q$ is satisfied.

The only assumption, in addition to the security of the input processes, is the consistency of the respective event-domain maps. Particularly, this assumption permits to solve the proof obligations concerning the latter input process by just swapping D for E and p for q in the term $\text{con-comp-map } D E p q$ and then applying the corresponding lemmas proven for the former input process.

lemma *con-comp-secure-del-aux-1*:

assumes

A : *secure* $P I D$ **and**

B : $y \in \text{range } p \vee y \in \text{range } q$ **and**

C : $\text{set } ys \subseteq \text{range } p \cup \text{range } q$ **and**

D : $Y \subseteq \text{range } p$ **and**

E : $(\text{map } (\text{inv } p) [x \leftarrow xs \ @ \ y \ # \ ys. \ x \in \text{range } p], \text{inv } p \ ' \ Y) \in \text{failures } P$

shows

$(\text{map } (\text{inv } p) [x \leftarrow xs \ @ \ \text{ipurge-tr } (\text{con-comp-pol } I) (\text{con-comp-map } D E p q)$

$(\text{con-comp-map } D E p q \ y) \ ys. \ x \in \text{range } p],$

$\text{inv } p \ ' \ \text{ipurge-ref } (\text{con-comp-pol } I) (\text{con-comp-map } D E p q)$

$(\text{con-comp-map } D E p q \ y) \ ys \ Y) \in \text{failures } P$

(is $(\text{map } (\text{inv } p) [x \leftarrow xs \ @ \ \text{ipurge-tr } ?I' \ ?D' \ - \ - \ -], \ -) \in \ -)$

proof (*simp add*:

ipurge-tr-aux-single-dom [symmetric] ipurge-ref-aux-single-dom [symmetric]

ipurge-tr-aux-foldr-eq ipurge-ref-aux-foldr-eq [where $P = \lambda x. x \in \text{range } p$])

have $(\text{map } (\text{inv } p) [x \leftarrow xs \ @ \ [y]. \ x \in \text{range } p] \ @$

$\text{map } (\text{inv } p) (\text{ipurge-tr-aux-foldr } ?I' \ ?D' (\lambda x. x \in \text{range } p) \ \{?D' \ y\} \ ys),$

$\text{inv } p \ ' \ \text{ipurge-ref-aux-foldr } ?I' \ ?D' (\lambda x. x \in \text{range } p) \ \{?D' \ y\} \ ys \ Y)$

$\in \text{failures } P$

(is $(\ - \ @ \ \text{map } (\text{inv } p) (\text{ipurge-tr-aux-foldr } \ - \ ?F \ - \ -), \ -) \in \ -)$

proof (*rule con-comp-secure-aux [OF A D C]*)

show $\{?D' \ y\} \subseteq \text{range } \text{Some}$

using B **by** (*cases $y \in \text{range } p$, simp-all*)

next

show $(\text{map } (\text{inv } p) [x \leftarrow (xs \ @ \ [y]) \ @ \ ys. \ x \in \text{range } p], \text{inv } p \ ' \ Y) \in \text{failures } P$

using E **by** *simp*

qed

thus $(\text{map } (\text{inv } p) [x \leftarrow xs. \ x \in \text{range } p] \ @$

$\text{map } (\text{inv } p) (\text{ipurge-tr-aux-foldr } ?I' \ ?D' \ ?F \ \{?D' \ y\} \ ys),$

$\text{inv } p \ ' \ \text{ipurge-ref-aux-foldr } ?I' \ ?D' \ ?F \ \{?D' \ y\} \ ys \ Y)$

$\in \text{failures } P$

proof (*cases $y \in \text{range } p$, simp-all*)

assume $(\text{map } (\text{inv } p) [x \leftarrow xs. \ x \in \text{range } p] \ @ \ \text{inv } p \ y \ #$

$\text{map } (\text{inv } p) (\text{ipurge-tr-aux-foldr } ?I' \ ?D' \ ?F \ \{\text{Some } (D (\text{inv } p \ y))\} \ ys),$

$\text{inv } p \ ' \ \text{ipurge-ref-aux-foldr } ?I' \ ?D' \ ?F \ \{\text{Some } (D (\text{inv } p \ y))\} \ ys \ Y)$

$\in \text{failures } P$
hence $(\text{inv } p \ y \ \#$
 $\text{map } (\text{inv } p) (\text{ipurge-tr-aux-foldr } ?I' \ ?D' \ ?F \ \{\text{Some } (D \ (\text{inv } p \ y))\} \ ys),$
 $\text{inv } p \ ' \ \text{ipurge-ref-aux-foldr } ?I' \ ?D' \ ?F \ \{\text{Some } (D \ (\text{inv } p \ y))\} \ ys \ Y)$
 $\in \text{futures } P \ (\text{map } (\text{inv } p) \ [x \leftarrow xs. \ x \in \text{range } p])$
by $(\text{simp add: futures-def})$
hence
 $(\text{ipurge-tr } I \ D \ (D \ (\text{inv } p \ y)))$
 $(\text{map } (\text{inv } p) (\text{ipurge-tr-aux-foldr } ?I' \ ?D' \ ?F \ \{\text{Some } (D \ (\text{inv } p \ y))\} \ ys)),$
 $\text{ipurge-ref } I \ D \ (D \ (\text{inv } p \ y))$
 $(\text{map } (\text{inv } p) (\text{ipurge-tr-aux-foldr } ?I' \ ?D' \ ?F \ \{\text{Some } (D \ (\text{inv } p \ y))\} \ ys))$
 $(\text{inv } p \ ' \ \text{ipurge-ref-aux-foldr } ?I' \ ?D' \ ?F \ \{\text{Some } (D \ (\text{inv } p \ y))\} \ ys \ Y)$
 $\in \text{futures } P \ (\text{map } (\text{inv } p) \ [x \leftarrow xs. \ x \in \text{range } p])$
using A **by** $(\text{simp add: secure-def})$
hence
 $(\text{ipurge-tr-aux } I \ D \ (\text{the } ' \ \{\text{Some } (D \ (\text{inv } p \ y))\}))$
 $(\text{map } (\text{inv } p) (\text{ipurge-tr-aux-foldr } ?I' \ ?D' \ ?F \ \{\text{Some } (D \ (\text{inv } p \ y))\} \ ys)),$
 $\text{ipurge-ref-aux } I \ D \ (\text{the } ' \ \{\text{Some } (D \ (\text{inv } p \ y))\})$
 $(\text{map } (\text{inv } p) (\text{ipurge-tr-aux-foldr } ?I' \ ?D' \ ?F \ \{\text{Some } (D \ (\text{inv } p \ y))\} \ ys))$
 $(\text{inv } p \ ' \ \text{ipurge-ref-aux-foldr } ?I' \ ?D' \ ?F \ \{\text{Some } (D \ (\text{inv } p \ y))\} \ ys \ Y)$
 $\in \text{futures } P \ (\text{map } (\text{inv } p) \ [x \leftarrow xs. \ x \in \text{range } p])$
by $(\text{simp add: ipurge-tr-aux-single-dom ipurge-ref-aux-single-dom})$
moreover have
 $\text{ipurge-tr-aux } I \ D \ (\text{the } ' \ \{\text{Some } (D \ (\text{inv } p \ y))\})$
 $(\text{map } (\text{inv } p) (\text{ipurge-tr-aux-foldr } ?I' \ ?D' \ ?F \ \{\text{Some } (D \ (\text{inv } p \ y))\} \ ys)) =$
 $\text{map } (\text{inv } p) (\text{ipurge-tr-aux } ?I' \ ?D' \ \{\text{Some } (D \ (\text{inv } p \ y))\})$
 $(\text{ipurge-tr-aux-foldr } ?I' \ ?D' \ ?F \ \{\text{Some } (D \ (\text{inv } p \ y))\} \ ys)$
by $(\text{rule con-comp-ipurge-tr-aux, simp-all add:}$
 $\text{ipurge-tr-aux-foldr-eq } [\text{symmetric}], \text{blast})$
moreover have
 $\text{ipurge-ref-aux } I \ D \ (\text{the } ' \ \{\text{Some } (D \ (\text{inv } p \ y))\})$
 $(\text{map } (\text{inv } p) (\text{ipurge-tr-aux-foldr } ?I' \ ?D' \ ?F \ \{\text{Some } (D \ (\text{inv } p \ y))\} \ ys))$
 $(\text{inv } p \ ' \ \text{ipurge-ref-aux-foldr } ?I' \ ?D' \ ?F \ \{\text{Some } (D \ (\text{inv } p \ y))\} \ ys \ Y) =$
 $\text{inv } p \ ' \ \text{ipurge-ref-aux } ?I' \ ?D' \ \{\text{Some } (D \ (\text{inv } p \ y))\}$
 $(\text{ipurge-tr-aux-foldr } ?I' \ ?D' \ ?F \ \{\text{Some } (D \ (\text{inv } p \ y))\} \ ys)$
 $(\text{ipurge-ref-aux-foldr } ?I' \ ?D' \ ?F \ \{\text{Some } (D \ (\text{inv } p \ y))\} \ ys \ Y)$
proof $(\text{rule con-comp-ipurge-ref-aux, simp-all add:}$
 $\text{ipurge-tr-aux-foldr-eq } [\text{symmetric}] \ \text{ipurge-ref-aux-foldr-eq } [\text{symmetric}], \text{blast})$
have $\text{ipurge-ref-aux } ?I' \ ?D' \ \{\text{Some } (D \ (\text{inv } p \ y))\} \ ys \ Y \subseteq Y$
by $(\text{rule ipurge-ref-aux-subset})$
thus $\text{ipurge-ref-aux } ?I' \ ?D' \ \{\text{Some } (D \ (\text{inv } p \ y))\} \ ys \ Y \subseteq \text{range } p$
using D **by** simp
qed
ultimately have
 $(\text{map } (\text{inv } p) (\text{ipurge-tr-aux } ?I' \ ?D' \ \{\text{Some } (D \ (\text{inv } p \ y))\})$
 $(\text{ipurge-tr-aux-foldr } ?I' \ ?D' \ ?F \ \{\text{Some } (D \ (\text{inv } p \ y))\} \ ys)),$
 $\text{inv } p \ ' \ \text{ipurge-ref-aux } ?I' \ ?D' \ \{\text{Some } (D \ (\text{inv } p \ y))\}$
 $(\text{ipurge-tr-aux-foldr } ?I' \ ?D' \ ?F \ \{\text{Some } (D \ (\text{inv } p \ y))\} \ ys)$
 $(\text{ipurge-ref-aux-foldr } ?I' \ ?D' \ ?F \ \{\text{Some } (D \ (\text{inv } p \ y))\} \ ys \ Y))$

$\in \text{futures } P \text{ (map (inv p) [x←xs. x ∈ range p])}$
by simp
moreover have
 $\text{ipurge-tr-aux } ?I' ?D' \{ \text{Some } (D \text{ (inv p y)}) \}$
 $(\text{ipurge-tr-aux-foldr } ?I' ?D' ?F \{ \text{Some } (D \text{ (inv p y)}) \} \text{ ys}) =$
 $\text{ipurge-tr-aux-foldr } ?I' ?D' ?F \{ \text{Some } (D \text{ (inv p y)}) \} \text{ ys}$
by (rule ipurge-tr-aux-foldr-subset, simp)
moreover have
 $\text{ipurge-ref-aux } ?I' ?D' \{ \text{Some } (D \text{ (inv p y)}) \}$
 $(\text{ipurge-tr-aux-foldr } ?I' ?D' ?F \{ \text{Some } (D \text{ (inv p y)}) \} \text{ ys})$
 $(\text{ipurge-ref-aux-foldr } ?I' ?D' ?F \{ \text{Some } (D \text{ (inv p y)}) \} \text{ ys } Y) =$
 $\text{ipurge-ref-aux-foldr } ?I' ?D' ?F \{ \text{Some } (D \text{ (inv p y)}) \} \text{ ys } Y$
by (rule ipurge-ref-aux-foldr-subset, subst ipurge-tr-aux-foldr-sinks-aux, simp)
ultimately have
 $(\text{map (inv p) (ipurge-tr-aux-foldr } ?I' ?D' ?F \{ \text{Some } (D \text{ (inv p y)}) \} \text{ ys}),$
 $\text{inv p } ' \text{ ipurge-ref-aux-foldr } ?I' ?D' ?F \{ \text{Some } (D \text{ (inv p y)}) \} \text{ ys } Y)$
 $\in \text{futures } P \text{ (map (inv p) [x←xs. x ∈ range p])}$
by simp
thus (map (inv p) [x←xs. x ∈ range p] @
 $\text{map (inv p) (ipurge-tr-aux-foldr } ?I' ?D' ?F \{ \text{Some } (D \text{ (inv p y)}) \} \text{ ys}),$
 $\text{inv p } ' \text{ ipurge-ref-aux-foldr } ?I' ?D' ?F \{ \text{Some } (D \text{ (inv p y)}) \} \text{ ys } Y)$
 $\in \text{failures } P$
by (simp add: futures-def)
qed
qed

lemma con-comp-secure-add-aux-1:

assumes

A: secure P I D and

B: y ∈ range p ∨ y ∈ range q and

C: set zs ⊆ range p ∪ range q and

D: Z ⊆ range p and

E: (map (inv p) [x←xs @ zs. x ∈ range p], inv p ' Z) ∈ failures P and

F: map (inv p) [x←xs @ [y]. x ∈ range p] ∈ traces P

shows

$(\text{map (inv p) [x←xs @ y \# ipurge-tr (con-comp-pol I) (con-comp-map D E p q)}$

$(\text{con-comp-map D E p q y) zs. x ∈ range p],$

$\text{inv p } ' \text{ ipurge-ref (con-comp-pol I) (con-comp-map D E p q)}$

$(\text{con-comp-map D E p q y) zs } Z) \in \text{failures } P$

(is (map (inv p) [x←xs @ y \# ipurge-tr ?I' ?D' - -. -], -) ∈ -)

proof –

have

$(\text{map (inv p) [x←(xs @ [y]) @ ipurge-tr } ?I' ?D' (?D' y) \text{ zs. } x \in \text{range p}],$

$\text{inv p } ' \text{ ipurge-ref } ?I' ?D' (?D' y) \text{ zs } Z) \in \text{failures } P$

proof (subst filter-append, simp del: filter-append add:

$\text{ipurge-tr-aux-single-dom [symmetric] ipurge-ref-aux-single-dom [symmetric]}$

$\text{ipurge-tr-aux-foldr-eq ipurge-ref-aux-foldr-eq [where P = } \lambda x. x \in \text{range p]})$

have (map (inv p) [x←xs. x ∈ range p] @

$\text{map (inv p) (ipurge-tr-aux-foldr } ?I' ?D' (\lambda x. x \in \text{range p}) \{ ?D' y \} \text{ zs}),$

$inv\ p \text{ ' } ipurge\text{-ref-aux-foldr } ?I' \ ?D' (\lambda x. x \in range\ p) \{?D' \ y\} \ zs\ Z)$
 $\in failures\ P$
 $(is\ (-\ @\ map\ (inv\ p)\ (ipurge\text{-tr-aux-foldr}\ -\ -\ ?F\ -\ -),\ -) \in -)$
proof (*rule con-comp-secure-aux [OF A D C]*)
show $\{?D' \ y\} \subseteq range\ Some$
using B **by** (*cases* $y \in range\ p$, *simp-all*)
next
show ($map\ (inv\ p)\ [x \leftarrow xs\ @\ zs.\ x \in range\ p]$, $inv\ p \text{ ' } Z) \in failures\ P$
using E .
qed
thus ($map\ (inv\ p)\ [x \leftarrow xs\ @\ [y].\ x \in range\ p] \ @$
 $map\ (inv\ p)\ (ipurge\text{-tr-aux-foldr } ?I' \ ?D' \ ?F \ \{?D' \ y\} \ zs)$,
 $inv\ p \text{ ' } ipurge\text{-ref-aux-foldr } ?I' \ ?D' \ ?F \ \{?D' \ y\} \ zs\ Z)$
 $\in failures\ P$
proof (*cases* $y \in range\ p$, *simp-all*)
case $True$
assume ($map\ (inv\ p)\ [x \leftarrow xs.\ x \in range\ p] \ @$
 $map\ (inv\ p)\ (ipurge\text{-tr-aux-foldr } ?I' \ ?D' \ ?F \ \{Some\ (D\ (inv\ p\ y))\} \ zs)$,
 $inv\ p \text{ ' } ipurge\text{-ref-aux-foldr } ?I' \ ?D' \ ?F \ \{Some\ (D\ (inv\ p\ y))\} \ zs\ Z)$
 $\in failures\ P$
hence
 $(map\ (inv\ p)\ (ipurge\text{-tr-aux-foldr } ?I' \ ?D' \ ?F \ \{Some\ (D\ (inv\ p\ y))\} \ zs)$,
 $inv\ p \text{ ' } ipurge\text{-ref-aux-foldr } ?I' \ ?D' \ ?F \ \{Some\ (D\ (inv\ p\ y))\} \ zs\ Z)$
 $\in futures\ P\ (map\ (inv\ p)\ [x \leftarrow xs.\ x \in range\ p])$
by (*simp add: futures-def*)
moreover have ($map\ (inv\ p)\ [x \leftarrow xs\ @\ [y].\ x \in range\ p]$, $\{\}$) $\in failures\ P$
using F **by** (*rule traces-failures*)
hence ($[inv\ p\ y]$, $\{\}$) $\in futures\ P\ (map\ (inv\ p)\ [x \leftarrow xs.\ x \in range\ p])$
using $True$ **by** (*simp add: futures-def*)
ultimately have
 $(inv\ p\ y \# ipurge\text{-tr } I\ D\ (D\ (inv\ p\ y)))$
 $(map\ (inv\ p)\ (ipurge\text{-tr-aux-foldr } ?I' \ ?D' \ ?F \ \{Some\ (D\ (inv\ p\ y))\} \ zs))$,
 $ipurge\text{-ref } I\ D\ (D\ (inv\ p\ y))$
 $(map\ (inv\ p)\ (ipurge\text{-tr-aux-foldr } ?I' \ ?D' \ ?F \ \{Some\ (D\ (inv\ p\ y))\} \ zs))$
 $(inv\ p \text{ ' } ipurge\text{-ref-aux-foldr } ?I' \ ?D' \ ?F \ \{Some\ (D\ (inv\ p\ y))\} \ zs\ Z)$
 $\in futures\ P\ (map\ (inv\ p)\ [x \leftarrow xs.\ x \in range\ p])$
using A **by** (*simp add: secure-def*)
hence
 $(inv\ p\ y \# ipurge\text{-tr-aux } I\ D\ (the \text{ ' } \{Some\ (D\ (inv\ p\ y))\}))$
 $(map\ (inv\ p)\ (ipurge\text{-tr-aux-foldr } ?I' \ ?D' \ ?F \ \{Some\ (D\ (inv\ p\ y))\} \ zs))$,
 $ipurge\text{-ref-aux } I\ D\ (the \text{ ' } \{Some\ (D\ (inv\ p\ y))\})$
 $(map\ (inv\ p)\ (ipurge\text{-tr-aux-foldr } ?I' \ ?D' \ ?F \ \{Some\ (D\ (inv\ p\ y))\} \ zs))$
 $(inv\ p \text{ ' } ipurge\text{-ref-aux-foldr } ?I' \ ?D' \ ?F \ \{Some\ (D\ (inv\ p\ y))\} \ zs\ Z)$
 $\in futures\ P\ (map\ (inv\ p)\ [x \leftarrow xs.\ x \in range\ p])$
by (*simp add: ipurge-tr-aux-single-dom ipurge-ref-aux-single-dom*)
moreover have
 $ipurge\text{-tr-aux } I\ D\ (the \text{ ' } \{Some\ (D\ (inv\ p\ y))\})$
 $(map\ (inv\ p)\ (ipurge\text{-tr-aux-foldr } ?I' \ ?D' \ ?F \ \{Some\ (D\ (inv\ p\ y))\} \ zs)) =$
 $map\ (inv\ p)\ (ipurge\text{-tr-aux } ?I' \ ?D' \ \{Some\ (D\ (inv\ p\ y))\})$

$(\text{ipurge-tr-aux-foldr } ?I' ?D' ?F \{ \text{Some } (D \text{ (inv } p \ y)) \} \text{ } zs)$
by (rule con-comp-ipurge-tr-aux, simp-all add:
 $\text{ipurge-tr-aux-foldr-eq [symmetric], blast}$)
moreover have
 $\text{ipurge-ref-aux } I \ D \ (\text{the } \{ \text{Some } (D \text{ (inv } p \ y)) \})$
 $(\text{map } (\text{inv } p) (\text{ipurge-tr-aux-foldr } ?I' ?D' ?F \{ \text{Some } (D \text{ (inv } p \ y)) \} \text{ } zs))$
 $(\text{inv } p \ \{ \text{ipurge-ref-aux-foldr } ?I' ?D' ?F \{ \text{Some } (D \text{ (inv } p \ y)) \} \text{ } zs \ Z \} =$
 $\text{inv } p \ \{ \text{ipurge-ref-aux } ?I' ?D' \{ \text{Some } (D \text{ (inv } p \ y)) \}$
 $(\text{ipurge-tr-aux-foldr } ?I' ?D' ?F \{ \text{Some } (D \text{ (inv } p \ y)) \} \text{ } zs)$
 $(\text{ipurge-ref-aux-foldr } ?I' ?D' ?F \{ \text{Some } (D \text{ (inv } p \ y)) \} \text{ } zs \ Z)$
proof (rule con-comp-ipurge-ref-aux, simp-all add:
 $\text{ipurge-tr-aux-foldr-eq [symmetric] ipurge-ref-aux-foldr-eq [symmetric], blast}$)
have $\text{ipurge-ref-aux } ?I' ?D' \{ \text{Some } (D \text{ (inv } p \ y)) \} \text{ } zs \ Z \subseteq Z$
by (rule ipurge-ref-aux-subset)
thus $\text{ipurge-ref-aux } ?I' ?D' \{ \text{Some } (D \text{ (inv } p \ y)) \} \text{ } zs \ Z \subseteq \text{range } p$
using D **by** simp
qed
ultimately have
 $(\text{inv } p \ y \ \# \ \text{map } (\text{inv } p) (\text{ipurge-tr-aux } ?I' ?D' \{ \text{Some } (D \text{ (inv } p \ y)) \})$
 $(\text{ipurge-tr-aux-foldr } ?I' ?D' ?F \{ \text{Some } (D \text{ (inv } p \ y)) \} \text{ } zs),$
 $\text{inv } p \ \{ \text{ipurge-ref-aux } ?I' ?D' \{ \text{Some } (D \text{ (inv } p \ y)) \}$
 $(\text{ipurge-tr-aux-foldr } ?I' ?D' ?F \{ \text{Some } (D \text{ (inv } p \ y)) \} \text{ } zs)$
 $(\text{ipurge-ref-aux-foldr } ?I' ?D' ?F \{ \text{Some } (D \text{ (inv } p \ y)) \} \text{ } zs \ Z)$
 $\in \text{futures } P \ (\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p])$
by simp
moreover have
 $\text{ipurge-tr-aux } ?I' ?D' \{ \text{Some } (D \text{ (inv } p \ y)) \}$
 $(\text{ipurge-tr-aux-foldr } ?I' ?D' ?F \{ \text{Some } (D \text{ (inv } p \ y)) \} \text{ } zs) =$
 $\text{ipurge-tr-aux-foldr } ?I' ?D' ?F \{ \text{Some } (D \text{ (inv } p \ y)) \} \text{ } zs$
by (rule ipurge-tr-aux-foldr-subset, simp)
moreover have
 $\text{ipurge-ref-aux } ?I' ?D' \{ \text{Some } (D \text{ (inv } p \ y)) \}$
 $(\text{ipurge-tr-aux-foldr } ?I' ?D' ?F \{ \text{Some } (D \text{ (inv } p \ y)) \} \text{ } zs)$
 $(\text{ipurge-ref-aux-foldr } ?I' ?D' ?F \{ \text{Some } (D \text{ (inv } p \ y)) \} \text{ } zs \ Z) =$
 $\text{ipurge-ref-aux-foldr } ?I' ?D' ?F \{ \text{Some } (D \text{ (inv } p \ y)) \} \text{ } zs \ Z$
by (rule ipurge-ref-aux-foldr-subset, subst ipurge-tr-aux-foldr-sinks-aux, simp)
ultimately have
 $(\text{inv } p \ y \ \# \ \text{map } (\text{inv } p) (\text{ipurge-tr-aux-foldr } ?I' ?D' ?F$
 $\{ \text{Some } (D \text{ (inv } p \ y)) \} \text{ } zs),$
 $\text{inv } p \ \{ \text{ipurge-ref-aux-foldr } ?I' ?D' ?F$
 $\{ \text{Some } (D \text{ (inv } p \ y)) \} \text{ } zs \ Z)$
 $\in \text{futures } P \ (\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p])$
by simp
thus $(\text{map } (\text{inv } p) [x \leftarrow xs. x \in \text{range } p] \ @ \ \text{inv } p \ y \ \#$
 $\text{map } (\text{inv } p) (\text{ipurge-tr-aux-foldr } ?I' ?D' ?F \{ \text{Some } (D \text{ (inv } p \ y)) \} \text{ } zs),$
 $\text{inv } p \ \{ \text{ipurge-ref-aux-foldr } ?I' ?D' ?F \{ \text{Some } (D \text{ (inv } p \ y)) \} \text{ } zs \ Z)$
 $\in \text{failures } P$
by (simp add: futures-def)
qed

```

qed
thus ?thesis
  by simp
qed

```

lemma *con-comp-consistent-maps*:

```

consistent-maps D E p q  $\implies$  con-comp-map D E p q = con-comp-map E D q p
using [[simproc del: defined-all]] proof (simp add: consistent-maps-def, rule ext)
fix x
assume A:  $\forall x \in \text{range } p \cap \text{range } q. D (\text{inv } p \ x) = E (\text{inv } q \ x)$ 
show con-comp-map D E p q x = con-comp-map E D q p x
proof (rule con-comp-map.cases [of (D, E, p, q, x)], simp-all, (erule conjE)+)
  fix p' q' D' E' x'
  assume
    B: p = p' and
    C: q = q' and
    D: D = D' and
    E: E = E' and
    F: x'  $\in$  range p'
  show Some (D' (inv p' x')) = con-comp-map E' D' q' p' x'
  proof (cases x'  $\in$  range q', simp-all add: F)
    case True
      with F have x'  $\in$  range p'  $\cap$  range q' ..
      hence x'  $\in$  range p  $\cap$  range q
      using B and C by simp
      with A have D (inv p x') = E (inv q x') ..
      thus D' (inv p' x') = E' (inv q' x')
      using B and C and D and E by simp
    qed
  qed
qed

```

lemma *con-comp-secure-del-aux-2*:

```

assumes A: consistent-maps D E p q
shows
  secure Q I E  $\implies$ 
  y  $\in$  range p  $\vee$  y  $\in$  range q  $\implies$ 
  set ys  $\subseteq$  range p  $\cup$  range q  $\implies$ 
  Y  $\subseteq$  range q  $\implies$ 
  (map (inv q) [x $\leftarrow$ xs @ y # ys. x  $\in$  range q], inv q ' Y)  $\in$  failures Q  $\implies$ 
  (map (inv q) [x $\leftarrow$ xs @ ipurge-tr (con-comp-pol I) (con-comp-map D E p q)
    (con-comp-map D E p q y) ys. x  $\in$  range q],
  inv q ' ipurge-ref (con-comp-pol I) (con-comp-map D E p q)
  (con-comp-map D E p q y) ys Y)  $\in$  failures Q
proof (simp only: con-comp-consistent-maps [OF A], rule con-comp-secure-del-aux-1)
qed (simp-all, blast+)

```

lemma *con-comp-secure-add-aux-2*:

```

assumes A: consistent-maps D E p q

```

shows

$secure\ Q\ I\ E \implies$

$y \in range\ p \vee y \in range\ q \implies$

$set\ zs \subseteq range\ p \cup range\ q \implies$

$Z \subseteq range\ q \implies$

$(map\ (inv\ q)\ [x \leftarrow xs\ @\ zs.\ x \in range\ q],\ inv\ q\ 'Z) \in failures\ Q \implies$

$map\ (inv\ q)\ [x \leftarrow xs\ @\ [y].\ x \in range\ q] \in traces\ Q \implies$

$(map\ (inv\ q)\ [x \leftarrow xs\ @\ y\ \# \ ipurge\text{-}tr\ (con\text{-}comp\text{-}pol\ I)$

$(con\text{-}comp\text{-}map\ D\ E\ p\ q)\ (con\text{-}comp\text{-}map\ D\ E\ p\ q\ y)\ zs.\ x \in range\ q],$

$inv\ q\ ' \ ipurge\text{-}ref\ (con\text{-}comp\text{-}pol\ I)$

$(con\text{-}comp\text{-}map\ D\ E\ p\ q)\ (con\text{-}comp\text{-}map\ D\ E\ p\ q\ y)\ zs\ Z) \in failures\ Q$

proof (*simp only: con-comp-consistent-maps [OF A], rule con-comp-secure-add-aux-1*)

qed (*simp-all, blast+*)

lemma *con-comp-secure-del-case-1:*

assumes

A: consistent-maps D E p q and

B: secure P I D and

C: secure Q I E

shows

$\exists R\ S\ T.$

$Y = R \cup S \cup T \wedge$

$(y \in range\ p \vee y \in range\ q) \wedge$

$set\ xs \subseteq range\ p \cup range\ q \wedge$

$set\ ys \subseteq range\ p \cup range\ q \wedge$

$R \subseteq range\ p \wedge$

$S \subseteq range\ q \wedge$

$T \subseteq -\ range\ p \wedge$

$T \subseteq -\ range\ q \wedge$

$(map\ (inv\ p)\ [x \leftarrow xs\ @\ y\ \# \ ys.\ x \in range\ p],\ inv\ p\ 'R) \in failures\ P \wedge$

$(map\ (inv\ q)\ [x \leftarrow xs\ @\ y\ \# \ ys.\ x \in range\ q],\ inv\ q\ 'S) \in failures\ Q \implies$

$\exists R\ S\ T.$

$ipurge\text{-}ref\ (con\text{-}comp\text{-}pol\ I)\ (con\text{-}comp\text{-}map\ D\ E\ p\ q)$

$(con\text{-}comp\text{-}map\ D\ E\ p\ q\ y)\ ys\ Y = R \cup S \cup T \wedge$

$set\ xs \subseteq range\ p \cup range\ q \wedge$

$set\ (ipurge\text{-}tr\ (con\text{-}comp\text{-}pol\ I)\ (con\text{-}comp\text{-}map\ D\ E\ p\ q)$

$(con\text{-}comp\text{-}map\ D\ E\ p\ q\ y)\ ys) \subseteq range\ p \cup range\ q \wedge$

$R \subseteq range\ p \wedge$

$S \subseteq range\ q \wedge$

$T \subseteq -\ range\ p \wedge$

$T \subseteq -\ range\ q \wedge$

$(map\ (inv\ p)\ [x \leftarrow xs\ @\ ipurge\text{-}tr\ (con\text{-}comp\text{-}pol\ I)\ (con\text{-}comp\text{-}map\ D\ E\ p\ q)$

$(con\text{-}comp\text{-}map\ D\ E\ p\ q\ y)\ ys.\ x \in range\ p],\ inv\ p\ 'R) \in failures\ P \wedge$

$(map\ (inv\ q)\ [x \leftarrow xs\ @\ ipurge\text{-}tr\ (con\text{-}comp\text{-}pol\ I)\ (con\text{-}comp\text{-}map\ D\ E\ p\ q)$

$(con\text{-}comp\text{-}map\ D\ E\ p\ q\ y)\ ys.\ x \in range\ q],\ inv\ q\ 'S) \in failures\ Q$

(is $\implies \exists - - -. ipurge\text{-}ref\ ?I'\ ?D' - - - = - \wedge -)$

proof (*(erule exE)+, (erule conjE)+*)

fix *R S T*

assume

D: $Y = R \cup S \cup T$ **and**
E: $y \in \text{range } p \vee y \in \text{range } q$ **and**
F: $\text{set } xs \subseteq \text{range } p \cup \text{range } q$ **and**
G: $\text{set } ys \subseteq \text{range } p \cup \text{range } q$ **and**
H: $R \subseteq \text{range } p$ **and**
I: $S \subseteq \text{range } q$ **and**
J: $T \subseteq - \text{range } p$ **and**
K: $T \subseteq - \text{range } q$ **and**
L: $(\text{map } (\text{inv } p) [x \leftarrow xs @ y \# ys. x \in \text{range } p], \text{inv } p ' R) \in \text{failures } P$ **and**
M: $(\text{map } (\text{inv } q) [x \leftarrow xs @ y \# ys. x \in \text{range } q], \text{inv } q ' S) \in \text{failures } Q$
show *?thesis*
proof (*rule-tac* $x = \text{ipurge-ref } ?I' ?D' (?D' y) ys R$ **in** *exI*,
rule-tac $x = \text{ipurge-ref } ?I' ?D' (?D' y) ys S$ **in** *exI*, *rule-tac* $x = \{\}$ **in** *exI*,
(subst conj-assoc [symmetric])+, *(rule conjI)+*, *simp-all del: filter-append*)
have *ipurge-ref* $?I' ?D' (?D' y) ys Y =$
ipurge-ref $?I' ?D' (?D' y) ys (R \cup S \cup T)$
using *D* **by** *simp*
hence *ipurge-ref* $?I' ?D' (?D' y) ys Y =$
ipurge-ref $?I' ?D' (?D' y) ys R \cup$
ipurge-ref $?I' ?D' (?D' y) ys S \cup$
ipurge-ref $?I' ?D' (?D' y) ys T$
by (*simp add: ipurge-ref-distrib-union*)
moreover have *ipurge-ref* $?I' ?D' (?D' y) ys T = \{\}$
proof (*rule ipurge-ref-empty* [*of ?D' y*], *simp*, *insert E*,
cases y \in \text{range } p, *simp-all*)
fix *x*
assume $N: x \in T$
with *J* **have** $x \in - \text{range } p$ **..**
moreover have $x \in - \text{range } q$
using *K* **and** *N* **..**
ultimately have $?D' x = \text{None}$
by *simp*
thus (*Some* (*D* (*inv p y*)), $?D' x) \in ?I'$
by (*simp add: con-comp-pol-def*)
next
fix *x*
assume $N: x \in T$
with *J* **have** $x \in - \text{range } p$ **..**
moreover have $x \in - \text{range } q$
using *K* **and** *N* **..**
ultimately have $?D' x = \text{None}$
by *simp*
thus (*Some* (*E* (*inv q y*)), $?D' x) \in ?I'$
by (*simp add: con-comp-pol-def*)
qed
ultimately show *ipurge-ref* $?I' ?D' (?D' y) ys Y =$
ipurge-ref $?I' ?D' (?D' y) ys R \cup$
ipurge-ref $?I' ?D' (?D' y) ys S$
by *simp*

```

next
  show set xs ⊆ range p ∪ range q
  using F .
next
  have set (ipurge-tr ?I' ?D' (?D' y) ys) ⊆ set ys
  by (rule ipurge-tr-set)
  thus set (ipurge-tr ?I' ?D' (?D' y) ys) ⊆ range p ∪ range q
  using G by simp
next
  have ipurge-ref ?I' ?D' (?D' y) ys R ⊆ R
  by (rule ipurge-ref-subset)
  thus ipurge-ref ?I' ?D' (?D' y) ys R ⊆ range p
  using H by simp
next
  have ipurge-ref ?I' ?D' (?D' y) ys S ⊆ S
  by (rule ipurge-ref-subset)
  thus ipurge-ref ?I' ?D' (?D' y) ys S ⊆ range q
  using I by simp
next
  show (map (inv p) [x←xs @ ipurge-tr ?I' ?D' (?D' y) ys. x ∈ range p],
        inv p ' ipurge-ref ?I' ?D' (?D' y) ys R) ∈ failures P
  by (rule con-comp-secure-del-aux-1 [OF B E G H L])
next
  show (map (inv q) [x←xs @ ipurge-tr ?I' ?D' (?D' y) ys. x ∈ range q],
        inv q ' ipurge-ref ?I' ?D' (?D' y) ys S) ∈ failures Q
  by (rule con-comp-secure-del-aux-2 [OF A C E G I M])
qed
qed

```

lemma *con-comp-secure-del-case-2*:

assumes

A: consistent-maps *D E p q* **and**

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

C: secure *Q I E*

shows

$\exists xs'$.
 $(\exists ys'. xs @ y \# ys = xs' @ ys') \wedge$
 $set\ xs' \subseteq range\ p \cup range\ q \wedge$
 $map\ (inv\ p)\ [x \leftarrow xs'. x \in range\ p] \in divergences\ P \wedge$
 $map\ (inv\ q)\ [x \leftarrow xs'. x \in range\ q] \in divergences\ Q \implies$
 $(\exists R\ S\ T.$
 $ipurge-ref\ (con-comp-pol\ I)\ (con-comp-map\ D\ E\ p\ q)$
 $(con-comp-map\ D\ E\ p\ q\ y)\ ys\ Y = R \cup S \cup T \wedge$
 $set\ xs \subseteq range\ p \cup range\ q \wedge$
 $set\ (ipurge-tr\ (con-comp-pol\ I)\ (con-comp-map\ D\ E\ p\ q)$
 $(con-comp-map\ D\ E\ p\ q\ y)\ ys) \subseteq range\ p \cup range\ q \wedge$
 $R \subseteq range\ p \wedge$
 $S \subseteq range\ q \wedge$
 $T \subseteq -\ range\ p \wedge$

$T \subseteq - \text{range } q \wedge$
 $(\text{map } (\text{inv } p) [x \leftarrow xs \text{ @ } \text{ipurge-tr } (\text{con-comp-pol } I) (\text{con-comp-map } D E p q)$
 $(\text{con-comp-map } D E p q y) \text{ ys. } x \in \text{range } p], \text{inv } p \text{ ' } R) \in \text{failures } P \wedge$
 $(\text{map } (\text{inv } q) [x \leftarrow xs \text{ @ } \text{ipurge-tr } (\text{con-comp-pol } I) (\text{con-comp-map } D E p q)$
 $(\text{con-comp-map } D E p q y) \text{ ys. } x \in \text{range } q], \text{inv } q \text{ ' } S) \in \text{failures } Q) \vee$
 $(\exists xs'.$
 $(\exists \text{ys}'. xs \text{ @ } \text{ipurge-tr } (\text{con-comp-pol } I) (\text{con-comp-map } D E p q)$
 $(\text{con-comp-map } D E p q y) \text{ ys} = xs' \text{ @ } \text{ys}') \wedge$
 $\text{set } xs' \subseteq \text{range } p \cup \text{range } q \wedge$
 $\text{map } (\text{inv } p) [x \leftarrow xs'. x \in \text{range } p] \in \text{divergences } P \wedge$
 $\text{map } (\text{inv } q) [x \leftarrow xs'. x \in \text{range } q] \in \text{divergences } Q)$
 $(\text{is } - \implies (\exists R S T. ?F R S T \text{ ys}) \vee ?G)$
proof (*erule exE*, (*erule conjE*)+, *erule exE*)
fix $xs' \text{ ys}'$
assume
 $D: \text{set } xs' \subseteq \text{range } p \cup \text{range } q$ **and**
 $E: \text{map } (\text{inv } p) [x \leftarrow xs'. x \in \text{range } p] \in \text{divergences } P$ **and**
 $F: \text{map } (\text{inv } q) [x \leftarrow xs'. x \in \text{range } q] \in \text{divergences } Q$ **and**
 $G: xs \text{ @ } y \# \text{ys} = xs' \text{ @ } \text{ys}'$
show *?thesis*
proof (*cases length xs < length xs'*, *rule disjI1*, *rule-tac [2] disjI2*)
case *True*
moreover have $\text{take } (\text{length } xs') (xs \text{ @ } [y] \text{ @ } \text{ys}) =$
 $\text{take } (\text{length } xs') (xs \text{ @ } [y]) \text{ @ } \text{take } (\text{length } xs' - \text{Suc } (\text{length } xs)) \text{ ys}$
by (*simp only: take-append, simp*)
ultimately have $\text{take } (\text{length } xs') (xs \text{ @ } [y] \text{ @ } \text{ys}) =$
 $xs \text{ @ } y \# \text{take } (\text{length } xs' - \text{Suc } (\text{length } xs)) \text{ ys}$
 $(\text{is } - = - \text{ @ } - \# ?vs)$
by *simp*
moreover have $\text{take } (\text{length } xs') (xs \text{ @ } [y] \text{ @ } \text{ys}) =$
 $\text{take } (\text{length } xs') (xs' \text{ @ } \text{ys}')$
using G **by** *simp*
ultimately have $H: xs \text{ @ } y \# ?vs = xs'$
by *simp*
moreover have $y \in \text{set } (xs \text{ @ } y \# ?vs)$
by *simp*
ultimately have $y \in \text{set } xs'$
by *simp*
with D **have** $I: y \in \text{range } p \cup \text{range } q$..
have $\text{set } xs \subseteq \text{set } (xs \text{ @ } y \# ?vs)$
by *auto*
hence $\text{set } xs \subseteq \text{set } xs'$
using H **by** *simp*
hence $J: \text{set } xs \subseteq \text{range } p \cup \text{range } q$
using D **by** *simp*
have $\exists R S T. ?F R S T [x \leftarrow \text{ys. } x \in \text{range } p \cup \text{range } q]$
 $(\text{is } \exists - - -. \text{ipurge-ref } ?I' ?D' - - - = - \wedge -)$
proof (*rule con-comp-secure-del-case-1 [OF A B C]*,
rule-tac x = range p ∩ Y in exI, *rule-tac x = range q ∩ Y in exI*,

```

rule-tac x = - range p ∩ - range q ∩ Y in exI,
(subst conj-assoc [symmetric])+, (rule conjI)+, simp-all del: filter-append)
show Y = range p ∩ Y ∪ range q ∩ Y ∪ - range p ∩ - range q ∩ Y
  by blast
next
show y ∈ range p ∨ y ∈ range q
  using I by simp
next
show set xs ⊆ range p ∪ range q
  using J .
next
show {x ∈ set ys. x ∈ range p ∨ x ∈ range q} ⊆ range p ∪ range q
  by blast
next
show - range p ∩ - range q ∩ Y ⊆ - range p
  by blast
next
show - range p ∩ - range q ∩ Y ⊆ - range q
  by blast
next
have map (inv p) [x←xs @ y # ?vs. x ∈ range p] ∈ divergences P
  using E and H by simp
hence map (inv p) [x←xs @ y # ?vs. x ∈ range p] @
  map (inv p) [x←drop (length xs' - Suc (length xs)) ys. x ∈ range p]
  ∈ divergences P
  (is - @ map (inv p) [x←?ws. -] ∈ -)
  by (rule process-rule-5-general)
hence map (inv p) [x←(xs @ y # ?vs) @ ?ws. x ∈ range p] ∈ divergences P
  by (subst filter-append, simp)
hence map (inv p) [x←(xs @ [y]) @ ys. x ∈ range p] ∈ divergences P
  by simp
hence map (inv p) [x←(xs @ [y]) @ [x←ys. x ∈ range p ∨ x ∈ range q].
  x ∈ range p] ∈ divergences P
proof (subst (asm) filter-append, subst filter-append, subst filter-filter)
qed (subgoal-tac (λx. (x ∈ range p ∨ x ∈ range q) ∧ x ∈ range p) =
  (λx. x ∈ range p), simp, blast)
hence map (inv p) [x←xs @ y # [x←ys. x ∈ range p ∨ x ∈ range q].
  x ∈ range p] ∈ divergences P
  by simp
thus (map (inv p) [x←xs @ y # [x←ys. x ∈ range p ∨ x ∈ range q].
  x ∈ range p], inv p '(range p ∩ Y)) ∈ failures P
  by (rule process-rule-6)
next
have map (inv q) [x←xs @ y # ?vs. x ∈ range q] ∈ divergences Q
  using F and H by simp
hence map (inv q) [x←xs @ y # ?vs. x ∈ range q] @
  map (inv q) [x←drop (length xs' - Suc (length xs)) ys. x ∈ range q]
  ∈ divergences Q
  (is - @ map (inv q) [x←?ws. -] ∈ -)

```

by (rule process-rule-5-general)
 hence map (inv q) [x←(xs @ y # ?vs) @ ?ws. x ∈ range q] ∈ divergences Q
 by (subst filter-append, simp)
 hence map (inv q) [x←(xs @ [y]) @ ys. x ∈ range q] ∈ divergences Q
 by simp
 hence map (inv q) [x←(xs @ [y]) @ [x←ys. x ∈ range p ∨ x ∈ range q].
 x ∈ range q] ∈ divergences Q
 proof (subst (asm) filter-append, subst filter-append, subst filter-filter)
 qed (subgoal-tac (λx. (x ∈ range p ∨ x ∈ range q) ∧ x ∈ range q) =
 (λx. x ∈ range q), simp, blast)
 hence map (inv q) [x←xs @ y # [x←ys. x ∈ range p ∨ x ∈ range q].
 x ∈ range q] ∈ divergences Q
 by simp
 thus (map (inv q) [x←xs @ y # [x←ys. x ∈ range p ∨ x ∈ range q].
 x ∈ range q], inv q ' (range q ∩ Y)) ∈ failures Q
 by (rule process-rule-6)
 qed
 then obtain R and S and T where
 ?F R S T [x←ys. x ∈ range p ∪ range q]
 by blast
 thus ∃ R S T. ?F R S T ys
 proof (rule-tac x = R in exI, rule-tac x = S in exI, rule-tac x = T in exI)
 qed (simp only: con-comp-ipurge-tr-filter con-comp-ipurge-ref-filter)
 next
 let
 ?I' = con-comp-pol I and
 ?D' = con-comp-map D E p q
 case False
 moreover have xs @ ipurge-tr ?I' ?D' (?D' y) ys =
 take (length xs') (xs @ ipurge-tr ?I' ?D' (?D' y) ys) @
 drop (length xs') (xs @ ipurge-tr ?I' ?D' (?D' y) ys)
 (is - = - @ ?vs)
 by (simp only: append-take-drop-id)
 ultimately have xs @ ipurge-tr ?I' ?D' (?D' y) ys =
 take (length xs') (xs @ y # ys) @ ?vs
 by simp
 hence H: xs @ ipurge-tr ?I' ?D' (?D' y) ys = xs' @ ?vs
 using G by simp
 show ?G
 proof (rule-tac x = xs' in exI, rule conjI, rule-tac x = ?vs in exI)
 qed (subst H, simp-all add: D E F)
 qed
 qed
 lemma con-comp-secure-add-case-1:
 assumes
 A: consistent-maps D E p q and
 B: secure P I D and
 C: secure Q I E and

$D: (xs @ y \# ys, Y) \in \text{con-comp-failures } P \ Q \ p \ q$ **and**
 $E: y \in \text{range } p \vee y \in \text{range } q$
shows
 $\exists R \ S \ T.$
 $Z = R \cup S \cup T \wedge$
 $\text{set } xs \subseteq \text{range } p \cup \text{range } q \wedge$
 $\text{set } zs \subseteq \text{range } p \cup \text{range } q \wedge$
 $R \subseteq \text{range } p \wedge$
 $S \subseteq \text{range } q \wedge$
 $T \subseteq - \text{range } p \wedge$
 $T \subseteq - \text{range } q \wedge$
 $(\text{map } (\text{inv } p) [x \leftarrow xs @ zs. x \in \text{range } p], \text{inv } p \text{ ' } R) \in \text{failures } P \wedge$
 $(\text{map } (\text{inv } q) [x \leftarrow xs @ zs. x \in \text{range } q], \text{inv } q \text{ ' } S) \in \text{failures } Q \implies$
 $\exists R \ S \ T.$
 $\text{ipurge-ref } (\text{con-comp-pol } I) (\text{con-comp-map } D \ E \ p \ q)$
 $(\text{con-comp-map } D \ E \ p \ q \ y) \text{ zs } Z = R \cup S \cup T \wedge$
 $\text{set } xs \subseteq \text{range } p \cup \text{range } q \wedge$
 $\text{set } (\text{ipurge-tr } (\text{con-comp-pol } I) (\text{con-comp-map } D \ E \ p \ q)$
 $(\text{con-comp-map } D \ E \ p \ q \ y) \text{ zs}) \subseteq \text{range } p \cup \text{range } q \wedge$
 $R \subseteq \text{range } p \wedge$
 $S \subseteq \text{range } q \wedge$
 $T \subseteq - \text{range } p \wedge$
 $T \subseteq - \text{range } q \wedge$
 $(\text{map } (\text{inv } p) [x \leftarrow xs @ y \# \text{ipurge-tr } (\text{con-comp-pol } I)$
 $(\text{con-comp-map } D \ E \ p \ q) (\text{con-comp-map } D \ E \ p \ q \ y) \text{ zs. } x \in \text{range } p],$
 $\text{inv } p \text{ ' } R) \in \text{failures } P \wedge$
 $(\text{map } (\text{inv } q) [x \leftarrow xs @ y \# \text{ipurge-tr } (\text{con-comp-pol } I)$
 $(\text{con-comp-map } D \ E \ p \ q) (\text{con-comp-map } D \ E \ p \ q \ y) \text{ zs. } x \in \text{range } q],$
 $\text{inv } q \text{ ' } S) \in \text{failures } Q$
 $(\text{is } - \implies \exists - - -. \text{ipurge-ref } ?I' \ ?D' - - - = - \wedge -)$
proof $((\text{erule } \text{exE})+, (\text{erule } \text{conjE})+)$
fix $R \ S \ T$
assume
 $F: Z = R \cup S \cup T$ **and**
 $G: \text{set } xs \subseteq \text{range } p \cup \text{range } q$ **and**
 $H: \text{set } zs \subseteq \text{range } p \cup \text{range } q$ **and**
 $I: R \subseteq \text{range } p$ **and**
 $J: S \subseteq \text{range } q$ **and**
 $K: T \subseteq - \text{range } p$ **and**
 $L: T \subseteq - \text{range } q$ **and**
 $M: (\text{map } (\text{inv } p) [x \leftarrow xs @ zs. x \in \text{range } p], \text{inv } p \text{ ' } R) \in \text{failures } P$ **and**
 $N: (\text{map } (\text{inv } q) [x \leftarrow xs @ zs. x \in \text{range } q], \text{inv } q \text{ ' } S) \in \text{failures } Q$
show $?thesis$
proof $(\text{rule-tac } x = \text{ipurge-ref } ?I' \ ?D' \ (?D' \ y) \text{ zs } R \text{ in } \text{exI},$
 $\text{rule-tac } x = \text{ipurge-ref } ?I' \ ?D' \ (?D' \ y) \text{ zs } S \text{ in } \text{exI}, \text{rule-tac } x = \{\} \text{ in } \text{exI},$
 $(\text{subst } \text{conj-assoc } [\text{symmetric}])+, (\text{rule } \text{conjI})+, \text{simp-all del: filter-append})$
have $\text{ipurge-ref } ?I' \ ?D' \ (?D' \ y) \text{ zs } Z =$
 $\text{ipurge-ref } ?I' \ ?D' \ (?D' \ y) \text{ zs } (R \cup S \cup T)$
using F **by** simp

hence $\text{ipurge-ref } ?I' ?D' (?D' y) \text{ zs } Z =$
 $\text{ipurge-ref } ?I' ?D' (?D' y) \text{ zs } R \cup$
 $\text{ipurge-ref } ?I' ?D' (?D' y) \text{ zs } S \cup$
 $\text{ipurge-ref } ?I' ?D' (?D' y) \text{ zs } T$
by (*simp add: ipurge-ref-distrib-union*)
moreover have $\text{ipurge-ref } ?I' ?D' (?D' y) \text{ zs } T = \{\}$
proof (*rule ipurge-ref-empty [of ?D' y], simp, insert E,*
cases y ∈ range p, simp-all)
fix x
assume $O: x \in T$
with K **have** $x \in - \text{range } p ..$
moreover have $x \in - \text{range } q$
using L **and** $O ..$
ultimately have $?D' x = \text{None}$
by *simp*
thus $(\text{Some } (D (\text{inv } p y)), ?D' x) \in ?I'$
by (*simp add: con-comp-pol-def*)
next
fix x
assume $O: x \in T$
with K **have** $x \in - \text{range } p ..$
moreover have $x \in - \text{range } q$
using L **and** $O ..$
ultimately have $?D' x = \text{None}$
by *simp*
thus $(\text{Some } (E (\text{inv } q y)), ?D' x) \in ?I'$
by (*simp add: con-comp-pol-def*)
qed
ultimately show $\text{ipurge-ref } ?I' ?D' (?D' y) \text{ zs } Z =$
 $\text{ipurge-ref } ?I' ?D' (?D' y) \text{ zs } R \cup$
 $\text{ipurge-ref } ?I' ?D' (?D' y) \text{ zs } S$
by *simp*
next
show $\text{set } xs \subseteq \text{range } p \cup \text{range } q$
using $G .$
next
have $\text{set } (\text{ipurge-tr } ?I' ?D' (?D' y) \text{ zs}) \subseteq \text{set } xs$
by (*rule ipurge-tr-set*)
thus $\text{set } (\text{ipurge-tr } ?I' ?D' (?D' y) \text{ zs}) \subseteq \text{range } p \cup \text{range } q$
using H **by** *simp*
next
have $\text{ipurge-ref } ?I' ?D' (?D' y) \text{ zs } R \subseteq R$
by (*rule ipurge-ref-subset*)
thus $\text{ipurge-ref } ?I' ?D' (?D' y) \text{ zs } R \subseteq \text{range } p$
using I **by** *simp*
next
have $\text{ipurge-ref } ?I' ?D' (?D' y) \text{ zs } S \subseteq S$
by (*rule ipurge-ref-subset*)
thus $\text{ipurge-ref } ?I' ?D' (?D' y) \text{ zs } S \subseteq \text{range } q$

using J **by** *simp*
next
have $\text{map } (\text{inv } p) [x \leftarrow xs \ @ \ y \ # \ ys. \ x \in \text{range } p] \in \text{traces } P \wedge$
 $\text{map } (\text{inv } q) [x \leftarrow xs \ @ \ y \ # \ ys. \ x \in \text{range } q] \in \text{traces } Q$
using D **by** (*rule con-comp-failures-traces*)
hence $\text{map } (\text{inv } p) [x \leftarrow (xs \ @ \ [y]) \ @ \ ys. \ x \in \text{range } p] \in \text{traces } P$
by *simp*
hence $\text{map } (\text{inv } p) [x \leftarrow xs \ @ \ [y]. \ x \in \text{range } p] \ @$
 $\text{map } (\text{inv } p) [x \leftarrow ys. \ x \in \text{range } p] \in \text{traces } P$
by (*subst (asm) filter-append, simp*)
hence $\text{map } (\text{inv } p) [x \leftarrow xs \ @ \ [y]. \ x \in \text{range } p] \in \text{traces } P$
by (*rule process-rule-2-traces*)
thus ($\text{map } (\text{inv } p) [x \leftarrow xs \ @ \ y \ # \ \text{ipurge-tr } ?I' \ ?D' \ (?D' \ y) \ zs. \ x \in \text{range } p],$
 $\text{inv } p \ ' \ \text{ipurge-ref } ?I' \ ?D' \ (?D' \ y) \ zs \ R) \in \text{failures } P$
by (*rule con-comp-secure-add-aux-1 [OF B E H I M]*)
next
have $\text{map } (\text{inv } p) [x \leftarrow xs \ @ \ y \ # \ ys. \ x \in \text{range } p] \in \text{traces } P \wedge$
 $\text{map } (\text{inv } q) [x \leftarrow xs \ @ \ y \ # \ ys. \ x \in \text{range } q] \in \text{traces } Q$
using D **by** (*rule con-comp-failures-traces*)
hence $\text{map } (\text{inv } q) [x \leftarrow (xs \ @ \ [y]) \ @ \ ys. \ x \in \text{range } q] \in \text{traces } Q$
by *simp*
hence $\text{map } (\text{inv } q) [x \leftarrow xs \ @ \ [y]. \ x \in \text{range } q] \ @$
 $\text{map } (\text{inv } q) [x \leftarrow ys. \ x \in \text{range } q] \in \text{traces } Q$
by (*subst (asm) filter-append, simp*)
hence $\text{map } (\text{inv } q) [x \leftarrow xs \ @ \ [y]. \ x \in \text{range } q] \in \text{traces } Q$
by (*rule process-rule-2-traces*)
thus ($\text{map } (\text{inv } q) [x \leftarrow xs \ @ \ y \ # \ \text{ipurge-tr } ?I' \ ?D' \ (?D' \ y) \ zs. \ x \in \text{range } q],$
 $\text{inv } q \ ' \ \text{ipurge-ref } ?I' \ ?D' \ (?D' \ y) \ zs \ S) \in \text{failures } Q$
by (*rule con-comp-secure-add-aux-2 [OF A C E H J N]*)
qed
qed

lemma *con-comp-secure-add-case-2*:

assumes

A : *consistent-maps* $D \ E \ p \ q$ **and**

B : *secure* $P \ I \ D$ **and**

C : *secure* $Q \ I \ E$ **and**

D : $(xs \ @ \ y \ # \ ys, \ Y) \in \text{con-comp-failures } P \ Q \ p \ q$ **and**

E : $y \in \text{range } p \vee y \in \text{range } q$

shows

$\exists xs'$.

$(\exists ys'. \ xs \ @ \ zs = xs' \ @ \ ys') \wedge$

$\text{set } xs' \subseteq \text{range } p \cup \text{range } q \wedge$

$\text{map } (\text{inv } p) [x \leftarrow xs'. \ x \in \text{range } p] \in \text{divergences } P \wedge$

$\text{map } (\text{inv } q) [x \leftarrow xs'. \ x \in \text{range } q] \in \text{divergences } Q \implies$

$(\exists R \ S \ T.$

$\text{ipurge-ref } (\text{con-comp-pol } I) (\text{con-comp-map } D \ E \ p \ q)$

$(\text{con-comp-map } D \ E \ p \ q \ y) \ zs \ Z = R \cup S \cup T \wedge$

$\text{set } xs \subseteq \text{range } p \cup \text{range } q \wedge$

$set (ipurge-tr (con-comp-pol I) (con-comp-map D E p q)$
 $(con-comp-map D E p q y) zs) \subseteq range p \cup range q \wedge$
 $R \subseteq range p \wedge$
 $S \subseteq range q \wedge$
 $T \subseteq - range p \wedge$
 $T \subseteq - range q \wedge$
 $(map (inv p) [x \leftarrow xs @ y \# ipurge-tr (con-comp-pol I)$
 $(con-comp-map D E p q) (con-comp-map D E p q y) zs. x \in range p],$
 $inv p ' R) \in failures P \wedge$
 $(map (inv q) [x \leftarrow xs @ y \# ipurge-tr (con-comp-pol I)$
 $(con-comp-map D E p q) (con-comp-map D E p q y) zs. x \in range q],$
 $inv q ' S) \in failures Q) \vee$
 $(\exists xs'.$
 $(\exists ys'. xs @ y \# ipurge-tr (con-comp-pol I) (con-comp-map D E p q)$
 $(con-comp-map D E p q y) zs = xs' @ ys') \wedge$
 $set xs' \subseteq range p \cup range q \wedge$
 $map (inv p) [x \leftarrow xs'. x \in range p] \in divergences P \wedge$
 $map (inv q) [x \leftarrow xs'. x \in range q] \in divergences Q)$
 $(is - \implies (\exists R S T. ?F R S T zs) \vee ?G)$
proof (erule exE, (erule conjE)+, erule exE)
fix xs' ys'
assume
 $F: set xs' \subseteq range p \cup range q$ **and**
 $G: map (inv p) [x \leftarrow xs'. x \in range p] \in divergences P$ **and**
 $H: map (inv q) [x \leftarrow xs'. x \in range q] \in divergences Q$ **and**
 $I: xs @ zs = xs' @ ys'$
show ?thesis
proof (cases length xs < length xs', rule disjI1, rule-tac [2] disjI2)
case True
moreover have take (length xs') (xs @ zs) =
take (length xs') xs @ take (length xs' - length xs) zs
by simp
ultimately have take (length xs') (xs @ zs) =
xs @ take (length xs' - length xs) zs
(is - = - @ ?vs)
by simp
moreover have take (length xs') (xs @ zs) =
take (length xs') (xs' @ ys')
using I by simp
ultimately have J: xs @ ?vs = xs'
by simp
moreover have set xs \subseteq set (xs @ ?vs)
by simp
ultimately have set xs \subseteq set xs'
by simp
hence K: set xs \subseteq range p \cup range q
using F by simp
have $\exists R S T. ?F R S T [x \leftarrow zs. x \in range p \cup range q]$
(is $\exists - - . ipurge-ref ?I' ?D' - - - = - \wedge -$)

proof (*rule con-comp-secure-add-case-1* [*OF A B C D E*],
rule-tac $x = \text{range } p \cap Z$ **in** exI , *rule-tac* $x = \text{range } q \cap Z$ **in** exI ,
rule-tac $x = - \text{range } p \cap - \text{range } q \cap Z$ **in** exI ,
(*subst conj-assoc* [*symmetric*])+, (*rule conjI*)+, *simp-all del: filter-append*)
show $Z = \text{range } p \cap Z \cup \text{range } q \cap Z \cup - \text{range } p \cap - \text{range } q \cap Z$
by *blast*
next
show $\text{set } xs \subseteq \text{range } p \cup \text{range } q$
using K .
next
show $\{x \in \text{set } zs. x \in \text{range } p \vee x \in \text{range } q\} \subseteq \text{range } p \cup \text{range } q$
by *blast*
next
show $- \text{range } p \cap - \text{range } q \cap Z \subseteq - \text{range } p$
by *blast*
next
show $- \text{range } p \cap - \text{range } q \cap Z \subseteq - \text{range } q$
by *blast*
next
have $\text{map } (\text{inv } p) [x \leftarrow xs \text{ @ } ?vs. x \in \text{range } p] \in \text{divergences } P$
using G **and** J **by** *simp*
hence $\text{map } (\text{inv } p) [x \leftarrow xs \text{ @ } ?vs. x \in \text{range } p] \text{ @}$
 $\text{map } (\text{inv } p) [x \leftarrow \text{drop } (\text{length } xs' - \text{length } xs) \text{ } zs. x \in \text{range } p]$
 $\in \text{divergences } P$
(is $- \text{ @ } \text{map } (\text{inv } p) [x \leftarrow ?ws. -] \in -)$
by (*rule process-rule-5-general*)
hence $\text{map } (\text{inv } p) [x \leftarrow (xs \text{ @ } ?vs) \text{ @ } ?ws. x \in \text{range } p] \in \text{divergences } P$
by (*subst filter-append, simp*)
hence $\text{map } (\text{inv } p) [x \leftarrow xs \text{ @ } zs. x \in \text{range } p] \in \text{divergences } P$
by *simp*
hence $\text{map } (\text{inv } p) [x \leftarrow xs \text{ @ } [x \leftarrow zs. x \in \text{range } p \vee x \in \text{range } q].$
 $x \in \text{range } p] \in \text{divergences } P$
proof (*subst (asm) filter-append, subst filter-append, subst filter-filter*)
qed (*subgoal-tac* $(\lambda x. (x \in \text{range } p \vee x \in \text{range } q) \wedge x \in \text{range } p) =$
 $(\lambda x. x \in \text{range } p), \text{simp}, \text{blast}$)
thus ($\text{map } (\text{inv } p) [x \leftarrow xs \text{ @ } [x \leftarrow zs. x \in \text{range } p \vee x \in \text{range } q].$
 $x \in \text{range } p], \text{inv } p \text{ ' } (\text{range } p \cap Z)) \in \text{failures } P$
by (*rule process-rule-6*)
next
have $\text{map } (\text{inv } q) [x \leftarrow xs \text{ @ } ?vs. x \in \text{range } q] \in \text{divergences } Q$
using H **and** J **by** *simp*
hence $\text{map } (\text{inv } q) [x \leftarrow xs \text{ @ } ?vs. x \in \text{range } q] \text{ @}$
 $\text{map } (\text{inv } q) [x \leftarrow \text{drop } (\text{length } xs' - \text{length } xs) \text{ } zs. x \in \text{range } q]$
 $\in \text{divergences } Q$
(is $- \text{ @ } \text{map } (\text{inv } q) [x \leftarrow ?ws. -] \in -)$
by (*rule process-rule-5-general*)
hence $\text{map } (\text{inv } q) [x \leftarrow (xs \text{ @ } ?vs) \text{ @ } ?ws. x \in \text{range } q] \in \text{divergences } Q$
by (*subst filter-append, simp*)
hence $\text{map } (\text{inv } q) [x \leftarrow xs \text{ @ } zs. x \in \text{range } q] \in \text{divergences } Q$

by *simp*
hence $\text{map } (\text{inv } q) [x \leftarrow xs \ @ \ [x \leftarrow zs. x \in \text{range } p \vee x \in \text{range } q].$
 $x \in \text{range } q] \in \text{divergences } Q$
proof (*subst (asm) filter-append, subst filter-append, subst filter-filter*)
qed (*subgoal-tac* ($\lambda x. (x \in \text{range } p \vee x \in \text{range } q) \wedge x \in \text{range } q$) =
 $(\lambda x. x \in \text{range } q), \text{simp}, \text{blast}$)
thus ($\text{map } (\text{inv } q) [x \leftarrow xs \ @ \ [x \leftarrow zs. x \in \text{range } p \vee x \in \text{range } q].$
 $x \in \text{range } q], \text{inv } q \text{ ' } (\text{range } q \cap Z)) \in \text{failures } Q$
by (*rule process-rule-6*)
qed
then obtain R and S and T where
 $?F R S T [x \leftarrow zs. x \in \text{range } p \cup \text{range } q]$
by *blast*
thus $\exists R S T. ?F R S T zs$
proof (*rule-tac* $x = R$ **in** exI , *rule-tac* $x = S$ **in** exI , *rule-tac* $x = T$ **in** exI)
qed (*simp only: con-comp-ipurge-tr-filter con-comp-ipurge-ref-filter*)
next
let
 $?I' = \text{con-comp-pol } I$ **and**
 $?D' = \text{con-comp-map } D E p q$
case *False*
moreover have $xs \ @ \ y \ # \ \text{ipurge-tr } ?I' ?D' (?D' y) zs =$
 $\text{take } (\text{length } xs') (xs \ @ \ y \ # \ \text{ipurge-tr } ?I' ?D' (?D' y) zs) \ @$
 $\text{drop } (\text{length } xs') (xs \ @ \ y \ # \ \text{ipurge-tr } ?I' ?D' (?D' y) zs)$
 $(\text{is } - = - \ @ \ ?vs)$
by (*simp only: append-take-drop-id*)
ultimately have $xs \ @ \ y \ # \ \text{ipurge-tr } ?I' ?D' (?D' y) zs =$
 $\text{take } (\text{length } xs') (xs \ @ \ zs) \ @ \ ?vs$
by *simp*
hence $J: xs \ @ \ y \ # \ \text{ipurge-tr } ?I' ?D' (?D' y) zs = xs' \ @ \ ?vs$
using I **by** *simp*
show $?G$
proof (*rule-tac* $x = xs'$ **in** exI , *rule conjI*, *rule-tac* $x = ?vs$ **in** exI)
qed (*subst J, simp-all add: F G H*)
qed
qed

theorem *con-comp-secure*:
assumes
 $A: \text{consistent-maps } D E p q$ **and**
 $B: \text{secure } P I D$ **and**
 $C: \text{secure } Q I E$
shows $\text{secure } (P \parallel Q \langle p, q \rangle) (\text{con-comp-pol } I) (\text{con-comp-map } D E p q)$
proof (*simp add: secure-def con-comp-futures, (rule allI)+, rule impI,*
erule conjE, rule conjI, (erule rev-mp)+, rotate-tac [2], erule-tac [2] rev-mp)
fix $xs y ys Y zs Z$
show
 $(xs \ @ \ zs, Z) \in \text{con-comp-failures } P Q p q \longrightarrow$
 $(xs \ @ \ y \ # \ ys, Y) \in \text{con-comp-failures } P Q p q \longrightarrow$

```

      (xs @ ipurge-tr (con-comp-pol I) (con-comp-map D E p q)
       (con-comp-map D E p q y) ys,
      ipurge-ref (con-comp-pol I) (con-comp-map D E p q)
       (con-comp-map D E p q y) ys Y)
    ∈ con-comp-failures P Q p q
  (is - → - → (- @ ipurge-tr ?I' ?D' - -, -) ∈ -)
proof ((rule impI)+, thin-tac (xs @ zs, Z) ∈ con-comp-failures P Q p q,
  simp-all add: con-comp-failures-def con-comp-divergences-def
  del: filter-append, erule disjE, rule disjI1)
qed (erule con-comp-secure-del-case-1 [OF A B C],
  rule con-comp-secure-del-case-2 [OF A B C])
next
fix xs y ys Y zs Z
assume D: (xs @ y # ys, Y) ∈ con-comp-failures P Q p q
show
  (xs @ zs, Z) ∈ con-comp-failures P Q p q →
  (xs @ y # ipurge-tr (con-comp-pol I) (con-comp-map D E p q)
   (con-comp-map D E p q y) zs,
  ipurge-ref (con-comp-pol I) (con-comp-map D E p q)
   (con-comp-map D E p q y) zs Z)
  ∈ con-comp-failures P Q p q
  (is - → (- @ - # ipurge-tr ?I' ?D' - -, -) ∈ -)
proof (rule impI, simp-all add: con-comp-failures-def con-comp-divergences-def
  del: filter-append, cases y ∈ range p ∨ y ∈ range q, simp del: filter-append,
  erule disjE, rule disjI1, rule-tac [3] disjI2)
qed (erule con-comp-secure-add-case-1 [OF A B C D], assumption,
  erule con-comp-secure-add-case-2 [OF A B C D], assumption,
  rule con-comp-failures-divergences [OF D], simp-all)
qed

```

1.5 Conservation of noninterference security in the absence of fake events

In what follows, it is proven that in the absence of fake events, namely if $\text{range } p \cup \text{range } q = \text{UNIV}$, the output of the concurrent composition of two secure processes is secure with respect to the same noninterference policy enforced by the input processes, and to the event-domain map that simply associates each event to the same security domain as the corresponding events of the input processes.

More formally, for any two processes P, Q being secure with respect to the noninterference policy I and the event-domain maps D, E , their concurrent composition $P \parallel Q \langle p, q \rangle$ is secure with respect to the same noninterference policy I and the event-domain map $\text{the} \circ \text{con-comp-map } D E p q$, provided that conditions $\text{range } p \cup \text{range } q = \text{UNIV}$ and $\text{consistent-maps } D E p q$ are satisfied.

lemma *con-comp-sinks-range*:

$u \in \text{range } \text{Some} \implies$
 $\text{set } xs \subseteq \text{range } p \cup \text{range } q \implies$
 $\text{sinks } (\text{con-comp-pol } I) (\text{con-comp-map } D E p q) u xs \subseteq \text{range } \text{Some}$
by ($\text{insert con-comp-sinks-aux-range [of \{u\} xs p q I D E]$,
 $\text{simp add: sinks-aux-single-dom}$)

lemma *con-comp-sinks-no-fake*:

assumes

A : $\text{range } p \cup \text{range } q = \text{UNIV}$ **and**

B : $u \in \text{range } \text{Some}$

shows $\text{sinks } I (\text{the } \circ \text{con-comp-map } D E p q) (\text{the } u) xs =$
 $\text{the ' sinks } (\text{con-comp-pol } I) (\text{con-comp-map } D E p q) u xs$
(is - = the ' sinks ?I' ?D' - -)

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

fix $x xs$

assume C : $\text{sinks } I (\text{the } \circ ?D') (\text{the } u) xs = \text{the ' sinks ?I' ?D' } u xs$

have $x \in \text{range } p \cup \text{range } q$

using A **by** *simp*

hence D : $?D' x = \text{Some } (\text{the } (?D' x))$

by (*cases x \in \text{range } p, simp-all*)

have E : $u = \text{Some } (\text{the } u)$

using B **by** (*simp add: image-iff*)

show $\text{sinks } I (\text{the } \circ ?D') (\text{the } u) (xs @ [x]) = \text{the ' sinks ?I' ?D' } u (xs @ [x])$

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

case *True*

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

by *simp*

moreover have $(\text{the } u, \text{the } (?D' x)) \in I \vee$

$(\exists d \in \text{sinks } I (\text{the } \circ ?D') (\text{the } u) xs. (d, \text{the } (?D' x)) \in I)$

proof (*rule disjE [OF True], rule disjI1, rule-tac [2] disjI2*)

assume $(u, ?D' x) \in ?I'$

hence $(\text{Some } (\text{the } u), \text{Some } (\text{the } (?D' x))) \in ?I'$

using D **and** E **by** *simp*

thus $(\text{the } u, \text{the } (?D' x)) \in I$

by (*simp add: con-comp-pol-def*)

next

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

then obtain v **where** F : $v \in \text{sinks } ?I' ?D' u xs$ **and** G : $(v, ?D' x) \in ?I' ..$

have $\text{sinks } ?I' ?D' u xs \subseteq \text{range } \text{Some}$

by (*rule con-comp-sinks-range, simp-all add: A B*)

hence $v \in \text{range } \text{Some}$

using $F ..$

hence $v = \text{Some } (\text{the } v)$

by (*simp add: image-iff*)

hence $(\text{Some } (\text{the } v), \text{Some } (\text{the } (?D' x))) \in ?I'$

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

hence $(\text{the } v, \text{the } (?D' x)) \in I$

by (*simp add: con-comp-pol-def*)

moreover have $\text{the } v \in \text{sinks } I (\text{the } \circ ?D') (\text{the } u) xs$

using C and F by *simp*
 ultimately show $\exists d \in \text{sinks } I \text{ (the } \circ \text{ ?}D'\text{) (the } u\text{) } xs.$
 $(d, \text{the } (?D' x)) \in I ..$
 qed
 hence $\text{sinks } I \text{ (the } \circ \text{ ?}D'\text{) (the } u\text{) } (xs @ [x]) =$
 $\text{insert (the } (?D' x)) (\text{sinks } I \text{ (the } \circ \text{ ?}D'\text{) (the } u\text{) } xs)$
 by *simp*
 ultimately show *?thesis*
 using C by *simp*
 next
 case *False*
 hence $\text{sinks } ?I' \text{ ?}D' u \text{ (xs @ [x])} = \text{sinks } ?I' \text{ ?}D' u \text{ xs}$
 by *simp*
 moreover have $\neg ((\text{the } u, \text{the } (?D' x)) \in I \vee$
 $(\exists v \in \text{sinks } I \text{ (the } \circ \text{ ?}D'\text{) (the } u\text{) } xs. (v, \text{the } (?D' x)) \in I))$
 proof (*insert False, simp, erule conjE, rule conjI, rule-tac [2] ballI*)
 assume $(u, ?D' x) \notin ?I'$
 hence $(\text{Some } (\text{the } u), \text{Some } (\text{the } (?D' x))) \notin ?I'$
 using D and E by *simp*
 thus $(\text{the } u, \text{the } (?D' x)) \notin I$
 by (*simp add: con-comp-pol-def*)
 next
 fix d
 assume $d \in \text{sinks } I \text{ (the } \circ \text{ ?}D'\text{) (the } u\text{) } xs$
 hence $d \in \text{the ' sinks } ?I' \text{ ?}D' u \text{ xs}$
 using C by *simp*
 hence $\exists v \in \text{sinks } ?I' \text{ ?}D' u \text{ xs. } d = \text{the } v$
 by (*simp add: image-iff*)
 then obtain v where $F: v \in \text{sinks } ?I' \text{ ?}D' u \text{ xs}$ and $G: d = \text{the } v ..$
 have $\text{sinks } ?I' \text{ ?}D' u \text{ xs} \subseteq \text{range } \text{Some}$
 by (*rule con-comp-sinks-range, simp-all add: A B*)
 hence $v \in \text{range } \text{Some}$
 using $F ..$
 hence $H: v = \text{Some } d$
 using G by (*simp add: image-iff*)
 assume $\forall v \in \text{sinks } ?I' \text{ ?}D' u \text{ xs. } (v, ?D' x) \notin ?I'$
 hence $(v, ?D' x) \notin ?I'$
 using $F ..$
 hence $(\text{Some } d, \text{Some } (\text{the } (?D' x))) \notin ?I'$
 using D and H by *simp*
 thus $(d, \text{the } (?D' x)) \notin I$
 by (*simp add: con-comp-pol-def*)
 qed
 hence $\text{sinks } I \text{ (the } \circ \text{ ?}D'\text{) (the } u\text{) } (xs @ [x]) = \text{sinks } I \text{ (the } \circ \text{ ?}D'\text{) (the } u\text{) } xs$
 by *simp*
 ultimately show *?thesis*
 using C by *simp*
 qed
 qed

lemma *con-comp-ipurge-tr-no-fake*:

assumes

A: $\text{range } p \cup \text{range } q = \text{UNIV}$ **and**

B: $u \in \text{range } \text{Some}$

shows $\text{ipurge-tr } (\text{con-comp-pol } I) (\text{con-comp-map } D E p q) u xs =$

$\text{ipurge-tr } I (\text{the } \circ \text{con-comp-map } D E p q) (\text{the } u) xs$

(**is** $\text{ipurge-tr } ?I' ?D' - - = -$)

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

fix $x xs$

assume *C*: $\text{ipurge-tr } ?I' ?D' u xs = \text{ipurge-tr } I (\text{the } \circ ?D') (\text{the } u) xs$

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

proof (*cases ?D' x ∈ sinks ?I' ?D' u (xs @ [x])*)

case *True*

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

by *simp*

moreover have $\text{the } (?D' x) \in \text{the 'sinks } ?I' ?D' u (xs @ [x])$

using *True by simp*

hence $\text{the } (?D' x) \in \text{sinks } I (\text{the } \circ ?D') (\text{the } u) (xs @ [x])$

by (*subst con-comp-sinks-no-fake [OF A B]*)

hence $\text{ipurge-tr } I (\text{the } \circ ?D') (\text{the } u) (xs @ [x]) =$

$\text{ipurge-tr } I (\text{the } \circ ?D') (\text{the } u) xs$

by *simp*

ultimately show *?thesis*

using *C by simp*

next

case *False*

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

by *simp*

moreover have $\text{the } (?D' x) \notin \text{the 'sinks } ?I' ?D' u (xs @ [x])$

proof

assume $\text{the } (?D' x) \in \text{the 'sinks } ?I' ?D' u (xs @ [x])$

hence $\exists v \in \text{sinks } ?I' ?D' u (xs @ [x]). \text{the } (?D' x) = \text{the } v$

by (*simp add: image-iff*)

then obtain v **where**

D: $v \in \text{sinks } ?I' ?D' u (xs @ [x])$ **and** *E*: $\text{the } (?D' x) = \text{the } v ..$

have $x \in \text{range } p \cup \text{range } q$

using *A by simp*

hence $\exists d. ?D' x = \text{Some } d$

by (*cases x ∈ range p, simp-all*)

then obtain d **where** $?D' x = \text{Some } d ..$

moreover have $\text{sinks } ?I' ?D' u (xs @ [x]) \subseteq \text{range } \text{Some}$

by (*rule con-comp-sinks-range, simp-all add: A B*)

hence $v \in \text{range } \text{Some}$

using *D ..*

hence $\exists d'. v = \text{Some } d'$

by (*simp add: image-iff*)

then obtain d' **where** $v = \text{Some } d' ..$

ultimately have $?D' x = v$

```

    using E by simp
    hence ?D' x ∈ sinks ?I' ?D' u (xs @ [x])
    using D by simp
    thus False
    using False by contradiction
  qed
  hence the (?D' x) ∉ sinks I (the ∘ ?D') (the u) (xs @ [x])
  by (subst con-comp-sinks-no-fake [OF A B])
  hence ipurge-tr I (the ∘ ?D') (the u) (xs @ [x]) =
    ipurge-tr I (the ∘ ?D') (the u) xs @ [x]
  by simp
  ultimately show ?thesis
  using C by simp
qed
qed

lemma con-comp-ipurge-ref-no-fake:
  assumes
    A: range p ∪ range q = UNIV and
    B: u ∈ range Some
  shows ipurge-ref (con-comp-pol I) (con-comp-map D E p q) u xs X =
    ipurge-ref I (the ∘ con-comp-map D E p q) (the u) xs X
    (is ipurge-ref ?I' ?D' - - - = -)
proof (simp add: ipurge-ref-def set-eq-iff, rule allI,
  simp-all add: con-comp-sinks-no-fake [OF A B])
  fix x
  have x ∈ range p ∪ range q
  using A by simp
  hence C: ?D' x = Some (the (?D' x))
  by (cases x ∈ range p, simp-all)
  have D: u = Some (the u)
  using B by (simp add: image-iff)
  show
    (x ∈ X ∧ (u, ?D' x) ∉ con-comp-pol I ∧
      (∀ v ∈ sinks ?I' ?D' u xs. (v, ?D' x) ∉ con-comp-pol I)) =
    (x ∈ X ∧ (the u, the (?D' x)) ∉ I ∧
      (∀ v ∈ sinks ?I' ?D' u xs. (the v, the (?D' x)) ∉ I))
  proof (rule iffI, (erule-tac [!] conjE)+, simp-all, rule-tac [!] conjI,
    rule-tac [2] ballI, rule-tac [4] ballI)
    assume (u, ?D' x) ∉ ?I'
    hence (Some (the u), Some (the (?D' x))) ∉ ?I'
    using C and D by simp
    thus (the u, the (?D' x)) ∉ I
    by (simp add: con-comp-pol-def)
  next
  fix v
  assume ∀ v ∈ sinks ?I' ?D' u xs. (v, ?D' x) ∉ ?I' and
    E: v ∈ sinks ?I' ?D' u xs
  hence (v, ?D' x) ∉ ?I' ..

```

moreover have $\text{sinks } ?I' \ ?D' \ u \ xs \subseteq \text{range } \text{Some}$
by (*rule con-comp-sinks-range, simp-all add: A B*)
hence $v \in \text{range } \text{Some}$
using $E \ ..$
hence $v = \text{Some } (\text{the } v)$
by (*simp add: image-iff*)
ultimately have $(\text{Some } (\text{the } v), \text{Some } (\text{the } (?D' \ x))) \notin ?I'$
using C **by** *simp*
thus $(\text{the } v, \text{the } (?D' \ x)) \notin I$
by (*simp add: con-comp-pol-def*)
next
assume $(\text{the } u, \text{the } (?D' \ x)) \notin I$
hence $(\text{Some } (\text{the } u), \text{Some } (\text{the } (?D' \ x))) \notin ?I'$
by (*simp add: con-comp-pol-def*)
thus $(u, ?D' \ x) \notin ?I'$
using C **and** D **by** *simp*
next
fix v
assume $\forall v \in \text{sinks } ?I' \ ?D' \ u \ xs. (\text{the } v, \text{the } (?D' \ x)) \notin I$ **and**
 $E: v \in \text{sinks } ?I' \ ?D' \ u \ xs$
hence $(\text{the } v, \text{the } (?D' \ x)) \notin I \ ..$
hence $(\text{Some } (\text{the } v), \text{Some } (\text{the } (?D' \ x))) \notin ?I'$
by (*simp add: con-comp-pol-def*)
moreover have $\text{sinks } ?I' \ ?D' \ u \ xs \subseteq \text{range } \text{Some}$
by (*rule con-comp-sinks-range, simp-all add: A B*)
hence $v \in \text{range } \text{Some}$
using $E \ ..$
hence $v = \text{Some } (\text{the } v)$
by (*simp add: image-iff*)
ultimately show $(v, ?D' \ x) \notin ?I'$
using C **by** *simp*
qed
qed

theorem *con-comp-secure-no-fake:*
assumes
 $A: \text{range } p \cup \text{range } q = \text{UNIV}$ **and**
 $B: \text{consistent-maps } D \ E \ p \ q$ **and**
 $C: \text{secure } P \ I \ D$ **and**
 $D: \text{secure } Q \ I \ E$
shows $\text{secure } (P \parallel Q \langle p, q \rangle) \ I$ (*the* \circ *con-comp-map* $D \ E \ p \ q$)
proof (*insert con-comp-secure [OF B C D], simp add: secure-def,*
(rule allI)+, rule impI)
fix $xs \ y \ ys \ Y \ zs \ Z$
let
 $?I' = \text{con-comp-pol } I$ **and**
 $?D' = \text{con-comp-map } D \ E \ p \ q$
have $y \in \text{range } p \cup \text{range } q$
using A **by** *simp*

hence $E: ?D' y \in \text{range } \text{Some}$
by (cases $y \in \text{range } p$, simp-all)
assume $\forall xs\ y\ ys\ Y\ zs\ Z.$
 $(y \# ys, Y) \in \text{futures } (P \parallel Q <p, q>) xs \wedge$
 $(zs, Z) \in \text{futures } (P \parallel Q <p, q>) xs \longrightarrow$
 $(\text{ipurge-tr } ?I' ?D' (?D' y) ys, \text{ipurge-ref } ?I' ?D' (?D' y) ys Y)$
 $\in \text{futures } (P \parallel Q <p, q>) xs \wedge$
 $(y \# \text{ipurge-tr } ?I' ?D' (?D' y) zs, \text{ipurge-ref } ?I' ?D' (?D' y) zs Z)$
 $\in \text{futures } (P \parallel Q <p, q>) xs$ **and**
 $(y \# ys, Y) \in \text{futures } (P \parallel Q <p, q>) xs \wedge$
 $(zs, Z) \in \text{futures } (P \parallel Q <p, q>) xs$
hence
 $(\text{ipurge-tr } ?I' ?D' (?D' y) ys, \text{ipurge-ref } ?I' ?D' (?D' y) ys Y)$
 $\in \text{futures } (P \parallel Q <p, q>) xs \wedge$
 $(y \# \text{ipurge-tr } ?I' ?D' (?D' y) zs, \text{ipurge-ref } ?I' ?D' (?D' y) zs Z)$
 $\in \text{futures } (P \parallel Q <p, q>) xs$
by blast
thus
 $(\text{ipurge-tr } I (\text{the } \circ ?D') (\text{the } (?D' y)) ys,$
 $\text{ipurge-ref } I (\text{the } \circ ?D') (\text{the } (?D' y)) ys Y) \in \text{futures } (P \parallel Q <p, q>) xs \wedge$
 $(y \# \text{ipurge-tr } I (\text{the } \circ ?D') (\text{the } (?D' y)) zs,$
 $\text{ipurge-ref } I (\text{the } \circ ?D') (\text{the } (?D' y)) zs Z) \in \text{futures } (P \parallel Q <p, q>) xs$
by (simp add: con-comp-ipurge-tr-no-fake [OF A E]
con-comp-ipurge-ref-no-fake [OF A E])
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, 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>.

- [6] P. Noce. Noninterference security in communicating sequential processes. *Archive of Formal Proofs*, May 2014. http://isa-afp.org/entries/Noninterference_CSP.shtml, Formal proof development.
- [7] P. Noce. The generic unwinding theorem for csp noninterference security. *Archive of Formal Proofs*, June 2015. http://isa-afp.org/entries/Noninterference_Generic_Unwinding.shtml, Formal proof development.
- [8] P. Noce. The inductive unwinding theorem for csp noninterference security. *Archive of Formal Proofs*, Aug. 2015. http://isa-afp.org/entries/Noninterference_Inductive_Unwinding.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, Formal proof development.
- [10] P. Noce. Conservation of csp noninterference security under sequential composition. *Archive of Formal Proofs*, Apr. 2016. http://isa-afp.org/entries/Noninterference_Sequential_Composition.shtml, Formal proof development.