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 23, 2021

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

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*]:
$u \in \textit{sinks-aux}\ I\ D\ U\ xs \longrightarrow u \in U \vee (\exists\, x \in \textit{set}\ xs.\ u = D\ x)$
⟨*proof*⟩

lemma *ipurge-ref-aux-cons*:
$\textit{ipurge-ref-aux}\ I\ D\ U\ (x\ \#\ xs)\ X = \textit{ipurge-ref-aux}\ I\ D\ (\textit{sinks-aux}\ I\ D\ U\ [x])\ xs\ X$
⟨*proof*⟩

lemma *process-rule-1-futures*:
$xs \in \textit{traces}\ P \Longrightarrow ([], \{\}) \in \textit{futures}\ P\ xs$
⟨*proof*⟩

lemma *process-rule-3-futures*:
$(ys,\ Y) \in \textit{futures}\ P\ xs \Longrightarrow Y' \subseteq Y \Longrightarrow (ys,\ Y') \in \textit{futures}\ P\ xs$
⟨*proof*⟩

lemma *process-rule-4-futures*:
$(ys,\ Y) \in \textit{futures}\ P\ xs \Longrightarrow$
$(ys\ @\ [x], \{\}) \in \textit{futures}\ P\ xs \vee (ys,\ \textit{insert}\ x\ Y) \in \textit{futures}\ P\ xs$
⟨*proof*⟩

lemma *process-rule-5-general* [*rule-format*]:
$xs \in \textit{divergences}\ P \longrightarrow xs\ @\ ys \in \textit{divergences}\ P$
⟨*proof*⟩

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* :: $'a\ \textit{process} \Rightarrow\ 'a\ \textit{list} \Rightarrow\ 'a\ \textit{list set}$ **where**
$\textit{future-divergences}\ P\ xs \equiv \{ys.\ xs\ @\ ys \in \textit{divergences}\ P\}$

definition *after* :: $'a\ \textit{process} \Rightarrow\ 'a\ \textit{list} \Rightarrow\ 'a\ \textit{process}$ (**infixl** $\setminus$ *64*) **where**
$P \setminus xs \equiv \textit{Abs-process}\ (\textit{futures}\ P\ xs,\ \textit{future-divergences}\ P\ xs)$

lemma *process-rule-5-futures*:
$ys \in \textit{future-divergences}\ P\ xs \Longrightarrow ys\ @\ [x] \in \textit{future-divergences}\ P\ xs$
⟨*proof*⟩

lemma *process-rule-6-futures*:
$ys \in \textit{future-divergences}\ P\ xs \Longrightarrow (ys,\ Y) \in \textit{futures}\ P\ xs$
⟨*proof*⟩

lemma *after-rep*:
assumes *A*: $xs \in \textit{traces}\ P$

shows *Rep-process* $(P \setminus xs) = (futures\ P\ xs,\ future\text{-}divergences\ P\ xs)$
(**is** - = $?X$)
$\langle proof \rangle$

lemma *after-failures*:
assumes A: $xs \in traces\ P$
shows $failures\ (P \setminus xs) = futures\ P\ xs$
$\langle proof \rangle$

lemma *after-futures*:
assumes A: $xs \in traces\ P$
shows $futures\ (P \setminus xs)\ ys = futures\ P\ (xs\ @\ ys)$
$\langle proof \rangle$

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

lemma *after-secure*:
assumes A: $xs \in traces\ P$
shows $secure\ P\ I\ D \Longrightarrow secure\ (P \setminus xs)\ I\ D$
$\langle proof \rangle$

lemma *ipurge-tr-ref-aux-futures*:
$[\![secure\ P\ I\ D;\ (ys,\ Y) \in futures\ P\ xs]\!] \Longrightarrow$
$(ipurge\text{-}tr\text{-}aux\ I\ D\ U\ ys,\ ipurge\text{-}ref\text{-}aux\ I\ D\ U\ ys\ Y) \in futures\ P\ xs$
$\langle proof \rangle$

lemma *ipurge-tr-ref-aux-failures-general*:
$[\![secure\ P\ I\ D;\ (xs\ @\ ys,\ Y) \in failures\ P]\!] \Longrightarrow$
$(xs\ @\ ipurge\text{-}tr\text{-}aux\ I\ D\ U\ ys,\ ipurge\text{-}ref\text{-}aux\ I\ D\ U\ ys\ Y) \in failures\ P$
$\langle proof \rangle$

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 $'a\ process$ and $'b\ process$. 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 $'c\ process$, where meaningful events are those in $range\ p \cup range\ q$ and shared events are those in $range\ p \cap range\ q$.

The case where $- (range\ p \cup range\ q) \neq \{\}$ constitutes a generalization of the definition given in [1], and the events in $- (range\ p \cup range\ q)$, 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 $- (range\ p \cup range\ q)$.

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 ⇒ 'b process ⇒ ('a ⇒ 'c) ⇒ ('b ⇒ 'c) ⇒ 'c list set **where**
con-comp-divergences P Q p q ≡
 {xs @ ys | 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}

definition *con-comp-failures* ::
 'a process ⇒ 'b process ⇒ ('a ⇒ 'c) ⇒ ('b ⇒ 'c) ⇒ 'c failure set **where**
con-comp-failures P Q p q ≡
 {(xs, X ∪ Y ∪ Z) | xs X Y Z.
  set xs ⊆ range p ∪ range q ∧
  X ⊆ range p ∧ Y ⊆ range q ∧ Z ⊆ − (range p ∪ range q) ∧
  (map (inv p) [x←xs. x ∈ range p], inv p ‘ X) ∈ failures P ∧
  (map (inv q) [x←xs. x ∈ range q], inv q ‘ Y) ∈ failures Q} ∪

$\{(xs,\ X).\ xs \in \textit{con-comp-divergences}\ P\ Q\ p\ q\}$

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

abbreviation *con-comp-syntax* ::
$'a\ process \Rightarrow\ 'b\ process \Rightarrow ('a \Rightarrow\ 'c) \Rightarrow ('b \Rightarrow\ 'c) \Rightarrow\ 'c\ process$
((- ‖ - <-, ->) *55*)
where
$P \parallel Q$ <p, q> $\equiv$ *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*:
$([],\ \{\}) \in \textit{con-comp-failures}\ P\ Q\ p\ q$
⟨*proof*⟩

lemma *con-comp-prop-2*:
$(xs\ @\ [x],\ X) \in \textit{con-comp-failures}\ P\ Q\ p\ q \Longrightarrow$
$(xs,\ \{\}) \in \textit{con-comp-failures}\ P\ Q\ p\ q$
⟨*proof*⟩

lemma *con-comp-prop-3*:
$[\![(xs,\ Y) \in \textit{con-comp-failures}\ P\ Q\ p\ q;\ X \subseteq Y]\!] \Longrightarrow$
$(xs,\ X) \in \textit{con-comp-failures}\ P\ Q\ p\ q$
⟨*proof*⟩

lemma *con-comp-prop-4*:
$(xs,\ X) \in \textit{con-comp-failures}\ P\ Q\ p\ q \Longrightarrow$
$(xs\ @\ [x],\ \{\}) \in \textit{con-comp-failures}\ P\ Q\ p\ q\ \vee$
$(xs,\ \textit{insert}\ x\ X) \in \textit{con-comp-failures}\ P\ Q\ p\ q$
⟨*proof*⟩

lemma *con-comp-prop-5*:
$xs \in \textit{con-comp-divergences}\ P\ Q\ p\ q \Longrightarrow$
$xs\ @\ [x] \in \textit{con-comp-divergences}\ P\ Q\ p\ q$
⟨*proof*⟩

lemma *con-comp-prop-6*:
$xs \in \textit{con-comp-divergences}\ P\ Q\ p\ q \Longrightarrow$
$(xs,\ X) \in \textit{con-comp-failures}\ P\ Q\ p\ q$
⟨*proof*⟩

lemma *con-comp-rep*:
$\mathit{Rep\text{-}process}\ (P \parallel Q <p,\ q>) =$
$\quad (\mathit{con\text{-}comp\text{-}failures}\ P\ Q\ p\ q,\ \mathit{con\text{-}comp\text{-}divergences}\ P\ Q\ p\ q)$
$\ (\mathbf{is}\ \text{-} = ?X)$
$\langle \mathit{proof} \rangle$

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*:
$\mathit{failures}\ (P \parallel Q <p,\ q>) = \mathit{con\text{-}comp\text{-}failures}\ P\ Q\ p\ q$
$\langle \mathit{proof} \rangle$

lemma *con-comp-divergences*:
$\mathit{divergences}\ (P \parallel Q <p,\ q>) = \mathit{con\text{-}comp\text{-}divergences}\ P\ Q\ p\ q$
$\langle \mathit{proof} \rangle$

lemma *con-comp-futures*:
$\mathit{futures}\ (P \parallel Q <p,\ q>)\ \mathit{xs} =$
$\quad \{(\mathit{ys},\ Y).\ (\mathit{xs}\ @\ \mathit{ys},\ Y) \in \mathit{con\text{-}comp\text{-}failures}\ P\ Q\ p\ q\}$
$\langle \mathit{proof} \rangle$

lemma *con-comp-traces*:
$\mathit{traces}\ (P \parallel Q <p,\ q>) = \mathit{Domain}\ (\mathit{con\text{-}comp\text{-}failures}\ P\ Q\ p\ q)$
$\langle \mathit{proof} \rangle$

lemma *con-comp-refusals*:
$\mathit{refusals}\ (P \parallel Q <p,\ q>)\ \mathit{xs} \equiv \mathit{con\text{-}comp\text{-}failures}\ P\ Q\ p\ q\ ``\ \{\mathit{xs}\}$
$\langle \mathit{proof} \rangle$

lemma *con-comp-next-events*:
$\mathit{next\text{-}events}\ (P \parallel Q <p,\ q>)\ \mathit{xs} =$
$\quad \{x.\ \mathit{xs}\ @\ [x] \in \mathit{Domain}\ (\mathit{con\text{-}comp\text{-}failures}\ P\ Q\ p\ q)\}$
$\langle \mathit{proof} \rangle$

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* $(P \parallel Q <p,\ q>)$
$\langle proof \rangle$

lemma *con-comp-failures-traces*:
$(xs,\ X) \in \mathit{con\text{-}comp\text{-}failures}\ P\ Q\ p\ q \Longrightarrow$
$\mathit{map}\ (\mathit{inv}\ p)\ [x{\leftarrow}xs.\ x \in \mathit{range}\ p] \in \mathit{traces}\ P\ \wedge$
$\mathit{map}\ (\mathit{inv}\ q)\ [x{\leftarrow}xs.\ x \in \mathit{range}\ q] \in \mathit{traces}\ Q$
$\langle proof \rangle$

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

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 *range* $p \cap$ *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 *range* $p \cap$ *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 $-$ (*range* $p \cup$ *range* q) $\neq \{\}$, the event-domain map D' for the output process must assign a security domain to the fake events in $-$ (*range* $p \cup$ *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 *range* $p \cup$ *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 $D'\ y =$ *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, *Some* e for the output process, the above considerations about policy I' entail that (*Some* d, *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* ::
$('d \times 'd)$ *set* $\Rightarrow$ $('d$ *option* $\times$ $'d$ *option*$)$ *set* **where**
con-comp-pol $I \equiv$
$\{(\textit{Some } d,\ \textit{Some } e) \mid d\ e.\ (d,\ e) \in I\} \cup \{(u,\ v).\ v = \textit{None}\}$

function *con-comp-map* ::
$('a \Rightarrow 'd) \Rightarrow ('b \Rightarrow 'd) \Rightarrow ('a \Rightarrow 'c) \Rightarrow ('b \Rightarrow 'c) \Rightarrow 'c \Rightarrow 'd$ *option* **where**
$x \in \textit{range } p \Longrightarrow$
$\textit{con-comp-map } D\ E\ p\ q\ x = \textit{Some } (D\ (\textit{inv } p\ x))\ |$
$x \notin \textit{range } p \Longrightarrow x \in \textit{range } q \Longrightarrow$
$\textit{con-comp-map } D\ E\ p\ q\ x = \textit{Some } (E\ (\textit{inv } q\ x))\ |$
$x \notin \textit{range } p \Longrightarrow x \notin \textit{range } q \Longrightarrow$
$\textit{con-comp-map } D\ E\ p\ q\ x = \textit{None}$
$\langle \textit{proof} \rangle$
termination $\langle \textit{proof} \rangle$

definition *consistent-maps* ::
$('a \Rightarrow 'd) \Rightarrow ('b \Rightarrow 'd) \Rightarrow ('a \Rightarrow 'c) \Rightarrow ('b \Rightarrow 'c) \Rightarrow$ *bool* **where**
consistent-maps $D\ E\ p\ q \equiv$
$\forall x \in \textit{range } p \cap \textit{range } q.\ D\ (\textit{inv } p\ x) = E\ (\textit{inv } q\ x)$

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* ::
 $('d \times 'd)\ set \Rightarrow ('a \Rightarrow 'd) \Rightarrow ('a \Rightarrow bool) \Rightarrow 'd\ set \Rightarrow 'a\ list \Rightarrow 'a\ list$
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-tr-aux-foldr* $I\ D\ P$ (*sinks-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-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-tr-aux-foldr*, it follows that the presence of list *ipurge-tr-aux-foldr* $I\ D\ P$ (*sinks-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-ref-aux-foldr*, which at first glance just seems a uselessly complicate and inefficient way to compute the intransitive purge of an event set.

primrec *ipurge-ref-aux-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

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

The reason for the introduction of such intransitive purge functions is that the recursive equations contained in their definitions, along with lemma *ipurge-tr-ref-aux-failures-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 $map\ (inv\ p)\ (filter\ (\lambda x.\ x \in range\ p)\ xs)$, moving functions *ipurge-tr-aux* and *ipurge-ref-aux* into the arguments of $map\ (inv\ p)$ and $(`)\ (inv\ p)$, 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*:
$U \subseteq V \Longrightarrow$
$ipurge\text{-}tr\text{-}aux\ I\ D\ U\ (ipurge\text{-}tr\text{-}aux\text{-}foldr\ I\ D\ P\ V\ xs) =$
$ipurge\text{-}tr\text{-}aux\text{-}foldr\ I\ D\ P\ V\ xs$
⟨*proof*⟩

lemma *ipurge-tr-aux-foldr-eq*:
$[x \leftarrow ipurge\text{-}tr\text{-}aux\ I\ D\ U\ xs.\ P\ x] = ipurge\text{-}tr\text{-}aux\text{-}foldr\ I\ D\ P\ U\ xs$
⟨*proof*⟩

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*]:
$U \subseteq V \longrightarrow$ *sinks-aux I D U* (*ipurge-tr-aux-foldr I D P V xs*) = *U*
⟨*proof*⟩

lemma *ipurge-tr-aux-foldr-ref-aux*:
assumes *A*: $U \subseteq V$
shows *ipurge-ref-aux I D U* (*ipurge-tr-aux-foldr I D P V xs*) *X* =
ipurge-ref-aux I D U [] *X*
⟨*proof*⟩

lemma *ipurge-ref-aux-foldr-subset* [*rule-format*]:
sinks-aux I D U ys $\subseteq V \longrightarrow$
ipurge-ref-aux I D U ys (*ipurge-ref-aux-foldr I D P V xs X*) =
ipurge-ref-aux-foldr I D P V xs X
⟨*proof*⟩

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*⟩

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: $U \subseteq$ *range Some* **and**
B: *set xs* $\subseteq$ *range p* $\cup$ *range q*
shows *sinks-aux* (*con-comp-pol I*) (*con-comp-map D E p q*) *U xs* $\subseteq$ *range Some*
(**is** *sinks-aux* - *?D′* - - $\subseteq$ -)
⟨*proof*⟩

lemma *con-comp-sinks-aux* [*rule-format*]:
assumes *A*: $U \subseteq$ *range Some*
shows *set xs* $\subseteq$ *range p* $\longrightarrow$
sinks-aux I D (*the ‘ U*) (*map* (*inv p*) *xs*) =
the ‘ sinks-aux (*con-comp-pol I*) (*con-comp-map D E p q*) *U xs*
(**is** - $\longrightarrow$ - = *the ‘ sinks-aux ?I′ ?D′* - -)
⟨*proof*⟩

lemma *con-comp-ipurge-tr-aux* [*rule-format*]:
assumes *A*: $U \subseteq$ *range Some*
shows *set xs* $\subseteq$ *range p* $\longrightarrow$
ipurge-tr-aux I D (*the ‘ U*) (*map* (*inv p*) *xs*) =

$map\ (inv\ p)\ (ipurge\text{-}tr\text{-}aux\ (con\text{-}comp\text{-}pol\ I)\ (con\text{-}comp\text{-}map\ D\ E\ p\ q)\ U\ xs)$
(**is** - $\longrightarrow$ - = $map\ (inv\ p)\ (ipurge\text{-}tr\text{-}aux\ ?I'\ ?D'$ - -))
⟨*proof*⟩

lemma *con-comp-ipurge-ref-aux*:
assumes
A: $U \subseteq range\ Some$ **and**
B: $set\ xs \subseteq range\ p$ **and**
C: $X \subseteq range\ p$
shows $ipurge\text{-}ref\text{-}aux\ I\ D\ (the\ `\ U)\ (map\ (inv\ p)\ xs)\ (inv\ p\ `\ X) =$
$inv\ p\ `\ ipurge\text{-}ref\text{-}aux\ (con\text{-}comp\text{-}pol\ I)\ (con\text{-}comp\text{-}map\ D\ E\ p\ q)\ U\ xs\ X$
(**is** - = $inv\ p\ `\ ipurge\text{-}ref\text{-}aux\ ?I'\ ?D'$ - - -)
⟨*proof*⟩

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

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

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

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

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 $P \parallel Q <p, q>$ is secure with respect to the noninterference policy *con-comp-pol* I and the event-domain map *con-comp-map* $D\ E\ p\ q$, provided that condition *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 *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: $y \in$ *range* $p \lor y \in$ *range* q **and**
C: *set* $ys \subseteq$ *range* $p \cup$ *range* q **and**
D: $Y \subseteq$ *range* p **and**
E: (*map* (*inv* p) [$x \leftarrow xs$ @ y # ys. $x \in$ *range* p], *inv* p ‘ Y) $\in$ *failures* P
shows
(*map* (*inv* p) [$x \leftarrow xs$ @ *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* p],
inv p ‘ *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* P
(**is** (*map* (*inv* p) [$x \leftarrow xs$ @ *ipurge-tr* $?I'$ $?D'$ - -. -], -) $\in$ -)
⟨*proof*⟩

lemma *con-comp-secure-add-aux-1*:
assumes
A: *secure* $P\ I\ D$ **and**
B: $y \in$ *range* $p \lor y \in$ *range* q **and**
C: *set* $zs \subseteq$ *range* $p \cup$ *range* q **and**
D: $Z \subseteq$ *range* p **and**
E: (*map* (*inv* p) [$x \leftarrow xs$ @ zs. $x \in$ *range* p], *inv* p ‘ Z) $\in$ *failures* P **and**
F: *map* (*inv* p) [$x \leftarrow xs$ @ [y]. $x \in$ *range* p] $\in$ *traces* P
shows
(*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 ‘ *ipurge-ref* (*con-comp-pol* I) (*con-comp-map* $D\ E\ p\ q$)
(*con-comp-map* $D\ E\ p\ q\ y$) $zs\ Z$) $\in$ *failures* P
(**is** (*map* (*inv* p) [$x \leftarrow xs$ @ y # *ipurge-tr* $?I'$ $?D'$ - -. -], -) $\in$ -)
⟨*proof*⟩

lemma *con-comp-consistent-maps*:
$\mathit{consistent\text{-}maps}\ D\ E\ p\ q \Longrightarrow \mathit{con\text{-}comp\text{-}map}\ D\ E\ p\ q = \mathit{con\text{-}comp\text{-}map}\ E\ D\ q\ p$
$\langle \mathit{proof} \rangle$

lemma *con-comp-secure-del-aux-2*:
assumes A: $\mathit{consistent\text{-}maps}\ D\ E\ p\ q$
shows
$\mathit{secure}\ Q\ I\ E \Longrightarrow$
$y \in \mathit{range}\ p \lor y \in \mathit{range}\ q \Longrightarrow$
$\mathit{set}\ ys \subseteq \mathit{range}\ p \cup \mathit{range}\ q \Longrightarrow$
$Y \subseteq \mathit{range}\ q \Longrightarrow$
$(\mathit{map}\ (\mathit{inv}\ q)\ [x \leftarrow xs\ @\ y\ \#\ ys.\ x \in \mathit{range}\ q],\ \mathit{inv}\ q\ `\ Y) \in \mathit{failures}\ Q \Longrightarrow$
$(\mathit{map}\ (\mathit{inv}\ q)\ [x \leftarrow xs\ @\ \mathit{ipurge\text{-}tr}\ (\mathit{con\text{-}comp\text{-}pol}\ I)\ (\mathit{con\text{-}comp\text{-}map}\ D\ E\ p\ q)$
$(\mathit{con\text{-}comp\text{-}map}\ D\ E\ p\ q\ y)\ ys.\ x \in \mathit{range}\ q],$
$\mathit{inv}\ q\ `\ \mathit{ipurge\text{-}ref}\ (\mathit{con\text{-}comp\text{-}pol}\ I)\ (\mathit{con\text{-}comp\text{-}map}\ D\ E\ p\ q)$
$(\mathit{con\text{-}comp\text{-}map}\ D\ E\ p\ q\ y)\ ys\ Y) \in \mathit{failures}\ Q$
$\langle \mathit{proof} \rangle$

lemma *con-comp-secure-add-aux-2*:
assumes A: $\mathit{consistent\text{-}maps}\ D\ E\ p\ q$
shows
$\mathit{secure}\ Q\ I\ E \Longrightarrow$
$y \in \mathit{range}\ p \lor y \in \mathit{range}\ q \Longrightarrow$
$\mathit{set}\ zs \subseteq \mathit{range}\ p \cup \mathit{range}\ q \Longrightarrow$
$Z \subseteq \mathit{range}\ q \Longrightarrow$
$(\mathit{map}\ (\mathit{inv}\ q)\ [x \leftarrow xs\ @\ zs.\ x \in \mathit{range}\ q],\ \mathit{inv}\ q\ `\ Z) \in \mathit{failures}\ Q \Longrightarrow$
$\mathit{map}\ (\mathit{inv}\ q)\ [x \leftarrow xs\ @\ [y].\ x \in \mathit{range}\ q] \in \mathit{traces}\ Q \Longrightarrow$
$(\mathit{map}\ (\mathit{inv}\ q)\ [x \leftarrow xs\ @\ y\ \#\ \mathit{ipurge\text{-}tr}\ (\mathit{con\text{-}comp\text{-}pol}\ I)$
$(\mathit{con\text{-}comp\text{-}map}\ D\ E\ p\ q)\ (\mathit{con\text{-}comp\text{-}map}\ D\ E\ p\ q\ y)\ zs.\ x \in \mathit{range}\ q],$
$\mathit{inv}\ q\ `\ \mathit{ipurge\text{-}ref}\ (\mathit{con\text{-}comp\text{-}pol}\ I)$
$(\mathit{con\text{-}comp\text{-}map}\ D\ E\ p\ q)\ (\mathit{con\text{-}comp\text{-}map}\ D\ E\ p\ q\ y)\ zs\ Z) \in \mathit{failures}\ Q$
$\langle \mathit{proof} \rangle$

lemma *con-comp-secure-del-case-1*:
assumes
A: $\mathit{consistent\text{-}maps}\ D\ E\ p\ q$ **and**
B: $\mathit{secure}\ P\ I\ D$ **and**
C: $\mathit{secure}\ Q\ I\ E$
shows
$\exists R\ S\ T.$
$Y = R \cup S \cup T\ \land$
$(y \in \mathit{range}\ p \lor y \in \mathit{range}\ q)\ \land$
$\mathit{set}\ xs \subseteq \mathit{range}\ p \cup \mathit{range}\ q\ \land$
$\mathit{set}\ ys \subseteq \mathit{range}\ p \cup \mathit{range}\ q\ \land$
$R \subseteq \mathit{range}\ p\ \land$
$S \subseteq \mathit{range}\ q\ \land$
$T \subseteq -\ \mathit{range}\ p\ \land$
$T \subseteq -\ \mathit{range}\ q\ \land$
$(\mathit{map}\ (\mathit{inv}\ p)\ [x \leftarrow xs\ @\ y\ \#\ ys.\ x \in \mathit{range}\ p],\ \mathit{inv}\ p\ `\ R) \in \mathit{failures}\ P\ \land$

$(map\ (inv\ q)\ [x \leftarrow xs\ @\ y\ \#\ ys.\ x \in range\ q],\ inv\ q\ `\ S) \in failures\ Q \Longrightarrow$
$\exists R\ S\ T.$
$ipurge\text{-}ref\ (con\text{-}comp\text{-}pol\ I)\ (con\text{-}comp\text{-}map\ D\ E\ p\ q)$
$(con\text{-}comp\text{-}map\ D\ E\ p\ q\ y)\ ys\ Y = R \cup S \cup T\ \wedge$
$set\ xs \subseteq range\ p \cup range\ q\ \wedge$
$set\ (ipurge\text{-}tr\ (con\text{-}comp\text{-}pol\ I)\ (con\text{-}comp\text{-}map\ D\ E\ p\ q)$
$(con\text{-}comp\text{-}map\ D\ E\ p\ q\ y)\ ys) \subseteq range\ p \cup range\ q\ \wedge$
$R \subseteq range\ p\ \wedge$
$S \subseteq range\ q\ \wedge$
$T \subseteq -\ range\ p\ \wedge$
$T \subseteq -\ range\ q\ \wedge$
$(map\ (inv\ p)\ [x \leftarrow xs\ @\ ipurge\text{-}tr\ (con\text{-}comp\text{-}pol\ I)\ (con\text{-}comp\text{-}map\ D\ E\ p\ q)$
$(con\text{-}comp\text{-}map\ D\ E\ p\ q\ y)\ ys.\ x \in range\ p],\ inv\ p\ `\ R) \in failures\ P\ \wedge$
$(map\ (inv\ q)\ [x \leftarrow xs\ @\ ipurge\text{-}tr\ (con\text{-}comp\text{-}pol\ I)\ (con\text{-}comp\text{-}map\ D\ E\ p\ q)$
$(con\text{-}comp\text{-}map\ D\ E\ p\ q\ y)\ ys.\ x \in range\ q],\ inv\ q\ `\ S) \in failures\ Q$
(**is** - $\Longrightarrow \exists$ - - -. *ipurge-ref ?I′ ?D′* - - - = - $\wedge$ -)
⟨*proof*⟩

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

(**is** - $\Longrightarrow$ ($\exists$ *R S T*. *?F R S T ys*) $\vee$ *?G*)
⟨*proof*⟩

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*) $\in$ *con-comp-failures P Q p q* **and**
E: *y* $\in$ *range p* $\vee$ *y* $\in$ *range q*
shows
$\exists$ *R S T*.
Z = *R* $\cup$ *S* $\cup$ *T* $\wedge$
set xs $\subseteq$ *range p* $\cup$ *range q* $\wedge$
set 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*←*xs* @ *zs*. *x* $\in$ *range p*], *inv p* ‘ *R*) $\in$ *failures P* $\wedge$
(*map* (*inv q*) [*x*←*xs* @ *zs*. *x* $\in$ *range q*], *inv q* ‘ *S*) $\in$ *failures Q* $\Longrightarrow$
$\exists$ *R S T*.
ipurge-ref (*con-comp-pol I*) (*con-comp-map D E p q*)
(*con-comp-map D E p q y*) *zs Z* = *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*) *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*←*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*←*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*
(**is** - $\Longrightarrow$ $\exists$ - - -. *ipurge-ref ?I′ ?D′* - - - = - $\wedge$ -)
⟨*proof*⟩

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*) $\in$ *con-comp-failures P Q p q* **and**
E: *y* $\in$ *range p* $\vee$ *y* $\in$ *range q*
shows
$\exists$ *xs′*.

$(\exists\, ys'.\ xs\ @\ zs = xs'\ @\ ys') \wedge$
$set\ xs' \subseteq range\ p \cup range\ q \wedge$
$map\ (inv\ p)\ [x{\leftarrow}xs'.\ x \in range\ p] \in divergences\ P \wedge$
$map\ (inv\ q)\ [x{\leftarrow}xs'.\ x \in range\ q] \in divergences\ Q \Longrightarrow$
$(\exists\, R\ S\ T.$
$ipurge\text{-}ref\ (con\text{-}comp\text{-}pol\ I)\ (con\text{-}comp\text{-}map\ D\ E\ p\ q)$
$(con\text{-}comp\text{-}map\ D\ E\ p\ q\ y)\ zs\ Z = R \cup S \cup T \wedge$
$set\ xs \subseteq range\ p \cup range\ q \wedge$
$set\ (ipurge\text{-}tr\ (con\text{-}comp\text{-}pol\ I)\ (con\text{-}comp\text{-}map\ D\ E\ p\ q)$
$(con\text{-}comp\text{-}map\ D\ E\ p\ q\ y)\ zs) \subseteq range\ p \cup range\ q \wedge$
$R \subseteq range\ p \wedge$
$S \subseteq range\ q \wedge$
$T \subseteq -\ range\ p \wedge$
$T \subseteq -\ range\ q \wedge$
$(map\ (inv\ p)\ [x{\leftarrow}xs\ @\ y\ \#\ ipurge\text{-}tr\ (con\text{-}comp\text{-}pol\ I)$
$(con\text{-}comp\text{-}map\ D\ E\ p\ q)\ (con\text{-}comp\text{-}map\ D\ E\ p\ q\ y)\ zs.\ x \in range\ p],$
$inv\ p\ `\ R) \in failures\ P \wedge$
$(map\ (inv\ q)\ [x{\leftarrow}xs\ @\ y\ \#\ ipurge\text{-}tr\ (con\text{-}comp\text{-}pol\ I)$
$(con\text{-}comp\text{-}map\ D\ E\ p\ q)\ (con\text{-}comp\text{-}map\ D\ E\ p\ q\ y)\ zs.\ x \in range\ q],$
$inv\ q\ `\ S) \in failures\ Q) \vee$
$(\exists\, xs'.$
$(\exists\, ys'.\ xs\ @\ y\ \#\ ipurge\text{-}tr\ (con\text{-}comp\text{-}pol\ I)\ (con\text{-}comp\text{-}map\ D\ E\ p\ q)$
$(con\text{-}comp\text{-}map\ D\ E\ p\ q\ y)\ zs = xs'\ @\ ys') \wedge$
$set\ xs' \subseteq range\ p \cup range\ q \wedge$
$map\ (inv\ p)\ [x{\leftarrow}xs'.\ x \in range\ p] \in divergences\ P \wedge$
$map\ (inv\ q)\ [x{\leftarrow}xs'.\ x \in range\ q] \in divergences\ Q)$
(**is** - $\Longrightarrow (\exists\, R\ S\ T.\ ?F\ R\ S\ T\ zs) \vee\ ?G$)
⟨*proof*⟩

theorem *con-comp-secure*:
assumes
A: *consistent-maps D E p q* **and**
B: *secure P I D* **and**
C: *secure Q I E*
shows $secure\ (P \parallel Q <p,\ q>)\ (con\text{-}comp\text{-}pol\ I)\ (con\text{-}comp\text{-}map\ D\ E\ p\ q)$
⟨*proof*⟩

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 $range\ p \cup range\ q = UNIV$, 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 $P \parallel Q <p, q>$ is secure with respect to the same noninterference policy I and the event-domain map $\mathit{the} \circ \mathit{con\text{-}comp\text{-}map}\ D\ E\ p\ q$, provided that conditions $\mathit{range}\ p \cup \mathit{range}\ q = \mathit{UNIV}$ and $\mathit{consistent\text{-}maps}\ D\ E\ p\ q$ are satisfied.

lemma *con-comp-sinks-range*:
$u \in \mathit{range\ Some} \Longrightarrow$
$\mathit{set}\ xs \subseteq \mathit{range}\ p \cup \mathit{range}\ q \Longrightarrow$
$\mathit{sinks}\ (\mathit{con\text{-}comp\text{-}pol}\ I)\ (\mathit{con\text{-}comp\text{-}map}\ D\ E\ p\ q)\ u\ xs \subseteq \mathit{range\ Some}$
$\langle \mathit{proof} \rangle$

lemma *con-comp-sinks-no-fake*:
assumes
A: $\mathit{range}\ p \cup \mathit{range}\ q = \mathit{UNIV}$ **and**
B: $u \in \mathit{range\ Some}$
shows $\mathit{sinks}\ I\ (\mathit{the} \circ \mathit{con\text{-}comp\text{-}map}\ D\ E\ p\ q)\ (\mathit{the}\ u)\ xs =$
$\mathit{the}\ `\ \mathit{sinks}\ (\mathit{con\text{-}comp\text{-}pol}\ I)\ (\mathit{con\text{-}comp\text{-}map}\ D\ E\ p\ q)\ u\ xs$
(**is** $- = \mathit{the}\ `\ \mathit{sinks}\ ?I'\ ?D'\ -\ -$)
$\langle \mathit{proof} \rangle$

lemma *con-comp-ipurge-tr-no-fake*:
assumes
A: $\mathit{range}\ p \cup \mathit{range}\ q = \mathit{UNIV}$ **and**
B: $u \in \mathit{range\ Some}$
shows $\mathit{ipurge\text{-}tr}\ (\mathit{con\text{-}comp\text{-}pol}\ I)\ (\mathit{con\text{-}comp\text{-}map}\ D\ E\ p\ q)\ u\ xs =$
$\mathit{ipurge\text{-}tr}\ I\ (\mathit{the} \circ \mathit{con\text{-}comp\text{-}map}\ D\ E\ p\ q)\ (\mathit{the}\ u)\ xs$
(**is** $\mathit{ipurge\text{-}tr}\ ?I'\ ?D'\ -\ - = -$)
$\langle \mathit{proof} \rangle$

lemma *con-comp-ipurge-ref-no-fake*:
assumes
A: $\mathit{range}\ p \cup \mathit{range}\ q = \mathit{UNIV}$ **and**
B: $u \in \mathit{range\ Some}$
shows $\mathit{ipurge\text{-}ref}\ (\mathit{con\text{-}comp\text{-}pol}\ I)\ (\mathit{con\text{-}comp\text{-}map}\ D\ E\ p\ q)\ u\ xs\ X =$
$\mathit{ipurge\text{-}ref}\ I\ (\mathit{the} \circ \mathit{con\text{-}comp\text{-}map}\ D\ E\ p\ q)\ (\mathit{the}\ u)\ xs\ X$
(**is** $\mathit{ipurge\text{-}ref}\ ?I'\ ?D'\ -\ -\ - = -$)
$\langle \mathit{proof} \rangle$

theorem *con-comp-secure-no-fake*:
assumes
A: $\mathit{range}\ p \cup \mathit{range}\ q = \mathit{UNIV}$ **and**
B: $\mathit{consistent\text{-}maps}\ D\ E\ p\ q$ **and**
C: $\mathit{secure}\ P\ I\ D$ **and**
D: $\mathit{secure}\ Q\ I\ E$
shows $\mathit{secure}\ (P \parallel Q <p, q>)\ I\ (\mathit{the} \circ \mathit{con\text{-}comp\text{-}map}\ D\ E\ p\ q)$
$\langle \mathit{proof} \rangle$

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.