Composing and Decomposing Systems under Security Properties

A. W. Roscoe and L. Wulf
Oxford University Computing Laboratory
Parks Road, Wolfson Building, Oxford OX1 3QD, UK

3 February 1995

Abstract

We investigate the formal relationship between separability of processes and the types of non-interference properties they enjoy. Though intuitively appealing, separability – the ability to define a process as a parallel composition of disjoint components – alone cannot adequately prove the absence of information flow. We present a number of laws for the composition of secure systems, and an example to show how such laws can be applied.

Separability is an idea which has origin in the design of secure operating systems [Rus81]. Informally, a system is separable if its users (or user processes) can be isolated from each other. The purpose of this isolation is to achieve mutual non-interference between users.

The separability condition has been formalised elsewhere (e.g. [Bur89, Jac90]), and we adopt the definition that a process is separable if equivalent to a parallel composition of sub-processes with disjoint alphabets. This condition is succinctly expressed in the process algebraic notation of CSP [Hoa85] which we will employ in the following.

We formally relate separability to a number of non-interference conditions including "lazy non-interference" of [RWW94]. An interesting by-result of this is that separability alone should generally not be regarded as proving the absence of information flow, as long as there is the possibility of non-determinism. We present laws which preserve security under parallel composition, even in the presence of non-determinism.

In the following section we summarise two related but subtly different types of non-interference, which we call independence and invariance, respectively. These will be related to the separability condition in Section 2. The subsequent section shows how to compose secure parts to yield a secure whole; the power of this idea is illustrated in Section 4. The final section presents our conclusions.

1 Notions of Non-interference

The notation employed in the rest of this paper is summarised in the Appendix. We describe processes within the context of the failures-divergences model of CSP, although *unless specifically allowed otherwise* we assume all processes are free of divergence. Given a process P whose alphabet can be partitioned into sets A and B, we reserve the notation

$$\mathcal{L}_B(P)$$

for the process $RUN_A \mid\mid\mid P$; the "$\mathcal{L}$" with reference to the "lazy" abstraction of [RWW94]. We do not address "eager" abstraction here since it supposes that abstracted events happen instantaneously, which the lazy form does not. Lazy abstraction thus takes a more standard view of how actions are modelled, and appears likely to be more relevant in practice. A detailed motivation for the definition and its uses in the specification of non-interference properties can be found in [RWW94, Ros95]. DEFINITION Let P be a process and $\{A, B\}$ be a partition of αP. Then B is said to be independent (of A) in P, written $\mathcal{L}\mathrm{IND}_B(P)$, if and only if the process $\mathcal{L}_B(P)$ is deterministic.[1] A number of weaker definitions of non-interference have been proposed. Not all of these involve an explicit process abstraction, but typically they demand that two processes, after some activity in one part of the alphabet, are equivalent modulo lazy abstraction. All conditions stated in the following definition are equivalent, and we refer to them with the generic term "invariance".

THEOREM 1.1 *Let $\{A, B\}$ be a partition of the alphabet of process P. The*

[1] The notation used in this paper is different from that in our previous paper [RWW94] since our developing understanding of the variety and relationships of different security properties has required a revised and more systematic nomenclature. In the earlier paper the condition $\mathcal{L}\mathrm{IND}_B(P)$ was written $\mathbf{L\text{-}SEC}_A(P)$.

following statements are equivalent.

1. $\forall\, a \in A \bullet$
   $s\hat{\ }\langle a\rangle \in \textsc{traces}(P) \;\Rightarrow\; \mathcal{L}_B(P/s) =_{FD} \mathcal{L}_B(P/s\hat{\ }\langle a\rangle)$
2. $\forall\, s, t \in \textsc{traces}(P) \bullet$
   $s \mid\!\backslash\, B = t \mid\!\backslash\, B \;\Rightarrow\; \mathcal{L}_B(P/s) =_{FD} \mathcal{L}_B(P/t)$
3. $\forall\, s \in \textsc{traces}(P) \bullet$
   $s \mid\!\backslash\, B \in \textsc{traces}(P) \;\wedge\; \mathcal{L}_B(P/s) =_{FD} \mathcal{L}_B(P/(s \mid\!\backslash\, B))$
4. $\forall\, s \in \textsc{traces}(P) \bullet$
   $s \mid\!\backslash\, B \in \textsc{traces}(P) \;\wedge\; (P/s)^0 \cap B = (P/(s \mid\!\backslash\, B))^0 \cap B$
   $\wedge\; \textsc{refs}(P/s) \cap \mathbb{P}\, B = \textsc{refs}(P/(s \mid\!\backslash\, B)) \cap \mathbb{P}\, B$
5. $\forall\, s, t \in \textsc{traces}(P) \bullet$
   $s \mid\!\backslash\, B = t \mid\!\backslash\, B \;\Rightarrow\; (P/s)^0 \cap B = (P/t)^0 \cap B \;\wedge$
   $\textsc{refs}(P/s) \cap \mathbb{P}\, B = \textsc{refs}(P/t) \cap \mathbb{P}\, B$

where the notation $\mathbb{P}\, B$ denotes the power set of B.

Condition (2.) is the *failures-divergence* invariance discussed in [Ros95]. Condition (4.) is a straightforward extension of Allen's [All91] that also takes refusal behaviour into account rather than initials only. Condition (5.) is Ryan's [Rya91] definition of security. DEFINITION Let P be a process and $\{A, B\}$ be a partition of αP. Then B is said to be *invariant* (from A) in P, written $\mathcal{L}\mathrm{INV}_B(P)$, if and only if any of the conditions above is satisfied. Invariance and independence are related as follows.

LEMMA 1.2 *Given that $\{A, B\}$ partitions the alphabet of process P, then*

$$\mathcal{L}\mathrm{IND}_B(P) \;\Rightarrow\; \mathcal{L}\mathrm{INV}_B(P).$$

THEOREM 1.3 *Given that $\{A, B\}$ partitions the alphabet of process P, and P is deterministic, then*

$$\mathcal{L}\mathrm{IND}_B(P) \;\Leftrightarrow\; \mathcal{L}\mathrm{INV}_B(P).$$

PROOF Can be found in [Ros95]. ∎

2 Characterising Separability

Separability as a security criterion has intuitive appeal. Processes with disjoint alphabets do not synchronise on events when combined in parallel;

there is thus no scope for direct interference between them. We adopt the following standard definition of a process that can be separated into two components. DEFINITION (Jacob [Jac90]) Let P be a process and $\{A, B\}$ partition αP. P is said to be *separable* with respect to $\{A, B\}$ if there exist processes P_A and P_B with $\alpha P_A = A$ and $\alpha P_B = B$ such that

$$P \;=\; P_A \underset{\emptyset}{\|} P_B.$$

An important strengthening of this condition is the requirement for the component processes to be deterministic. This case is referred to as strong separability. DEFINITION Process P is said to be *strongly separable* with respect to partition $\{A, B\}$ if and only if there exist *deterministic* processes P_B and P_A with $\alpha P_A = A$ and $\alpha P_B = B$ such that

$$P \;=\; P_A \underset{\emptyset}{\|} P_B.$$

Let us start to investigate the relationship between separability and the non-interference conditions of the previous section. It is tempting to conjecture (and indeed the authors did) that separability is equivalent to mutual invariance, i.e. that a process P can be split as $P_A \underset{\emptyset}{\|} P_B$ precisely if both $\mathcal{L}\mathrm{INV}_B(P)$ and $\mathcal{L}\mathrm{INV}_A(P)$ hold. For many processes this will indeed be the case.

On closer inspection, however, this conjecture turns out to be untrue. The simplest counterexample is given by

$$P \;=\; (a \rightarrow P) \sqcap (b \rightarrow P).$$

After any trace of P, event a may be accepted or refused, irrespective of any b events occurring, and the same is true for the reverse. Therefore both $\{a\}$ and $\{b\}$ are invariant in P. The process, however, is not separable. In particular, P is *not* equivalent to

$$P' \;=\; (STOP \sqcap (a \rightarrow P')) \underset{\emptyset}{\|} (STOP \sqcap (b \rightarrow P'))$$

since P' may refuse the set $\{a, b\}$ which P cannot. This means a user with interface $\{a\}$ who sees this event refused can deduce that the other event b is not refused; this is a flow of information we probably want to prohibit. This example demonstrates that (i) it is essential to consider refusals in addition to traces when analysing information flows, and (ii) the flows resulting from refusals can be more subtle than anticipated.

In order to establish an equivalence between separability and mutual invariance, it is necessary to strengthen the definition of invariance. This can be achieved by considering the behaviour of the process, not after a particular trace s is observed, but after a particular failure (s, X) is observed. We therefore re-define the standard 'after' operator to include refusal information as follows. DEFINITION Let $P/(s, X)$ (for $(s, X) \in \text{FAILS}(P)$ and $s \notin \text{DIVS}(P)$) be defined

$$\begin{aligned}
\text{FAILS}(P/(s, X)) &= \{ (\langle\rangle, Y) \mid (s, X \cup Y) \in \text{FAILS}(P) \} \cup \\
&\quad \{ (\langle a\rangle\hat{\ }t, Y) \mid (s\hat{\ }\langle a\rangle\hat{\ }t, Y) \in \text{FAILS}(P) \wedge a \notin X \} \\
\text{DIVS}(P/(s, X)) &= \{ \langle a\rangle\hat{\ }t \mid s\hat{\ }\langle a\rangle\hat{\ }t \in \text{DIVS}(P) \wedge a \notin X \}
\end{aligned}$$

(noting that $P/(s, \emptyset) = P/s$). Equipped with this operator, Definition 1 is strengthened as follows. DEFINITION Let A and B partition αP. Then B is strongly invariant in P, written $\mathcal{L}\mathbf{S}\text{INV}_B(P)$, if P is divergence-free and, whenever $(s, Y) \in \text{FAILS}(P)$ then $(s \mid\backslash\, B, Y \cap B) \in \text{FAILS}(P)$ and

$$\mathcal{L}_B(P/(s, Y)) = \mathcal{L}_B(P/(s \mid\backslash\, B, Y \cap B)).$$

Strong invariance lies half-way between invariance and independence, and consequently (Theorems 1.3 and 2.1) these three conditions are equivalent for deterministic processes.

THEOREM 2.1 *Let A and B partition αP. Then*

$$\mathcal{L}\text{IND}_B(P) \Rightarrow \mathcal{L}\mathbf{S}\text{INV}_B(P) \Rightarrow \mathcal{L}\text{INV}_B(P).$$

We can now turn attention to the main theorem of this section.

THEOREM 2.2 *If A and B partition αP, and process P does not diverge, then P is separable relative to this partition if, and only if,*

$$\mathcal{L}\mathbf{S}\text{INV}_A(P) \wedge \mathcal{L}\mathbf{S}\text{INV}_B(P).$$

PROOF

The proof in the "only if" direction is easy. If $P = Q \underset{\emptyset}{\|} R$ with $\alpha Q = A$ and $\alpha R = B$ then by symmetry it is sufficient to prove one of the two invariance properties. If $(s, Y \cup Z) \in \text{FAILS}(Q \underset{\emptyset}{\|} R)$ (where, as will be our convention from now on, $Y \subseteq A$ and $Z \subseteq B$) then, by definition of the parallel operator, $(s \restriction A, Y) \in \text{FAILS}(Q)$ and (combining this with $(\langle\rangle, \emptyset) \in \text{FAILS}(R)$), $(s \restriction A, Y) \in \text{FAILS}(Q \underset{\emptyset}{\|} R)$. Now,

$$\begin{aligned}
\mathcal{L}_A(P/(s, Y \cup Z)) &= (Q/(s \restriction A, Y) \underset{\emptyset}{\|} R/(s \restriction B, Z)) \mathbin{|||} RUN_B \\
&= (Q/(s \restriction A, Y) \mathbin{|||} R/(s \restriction B, Z)) \mathbin{|||} RUN_B \\
&= Q/(s \restriction A, Y) \mathbin{|||} RUN_B \\
&= Q/(s \restriction A, Y) \mathbin{|||} R \mathbin{|||} RUN_B \\
&= \mathcal{L}_A(P/(s \restriction A, Y))
\end{aligned}$$

The first line here is a simple property of 'after', the second because the alphabets of Q and R are disjoint, the next two both because RUN_B is a zero of $|||$ for divergence-free processes with alphabet B, and the last one by definition.

The "if" proof requires us to show that if both invariance properties hold then we can separate P. In fact, we will show that

$$P = (P \underset{B}{\|} STOP) \underset{\emptyset}{\|} (P \underset{A}{\|} STOP).$$

Call the left- and right-hand processes in the above parallel P_A and P_B respectively. The main part of the proof is contained in the following lemma:

LEMMA 2.3 *(Under assumptions of the theorem we are trying to prove) If* $s \in \text{TRACES}(P) \cap \text{TRACES}(P_A \underset{\emptyset}{\|} P_B)$ *then*

(a) $(P/s)^0 = ((P_A \underset{\emptyset}{\|} P_B)/s)^0$

(b) $\text{REFS}(P/s) = \text{REFS}((P_A \underset{\emptyset}{\|} P_B)/s)$

PROOF Assume first that $a \in (P/s)^0$, and wlog that $a \in A$. Then we know $s\hat{}\langle a\rangle \restriction A \in \text{TRACES}(P)$, and $s \restriction B \in \text{TRACES}(P)$ by the invariance properties. It follows that $s\hat{}\langle a\rangle \restriction A \in \text{TRACES}(P_A)$ and $s \restriction B \in \text{TRACES}(P_B)$ by definition of these processes. It easily follows that $s\hat{}\langle a\rangle \in \text{TRACES}(P_A \underset{\emptyset}{\|} P_B)$ which is what we want.

Conversely, assume that $a \in ((P_A \underset{\emptyset}{\|} P_B)/s)^0$ and wlog $a \in A$. Then we have $s\hat{}\langle a\rangle \restriction A \in \text{TRACES}(P_A) \subseteq \text{TRACES}(P)$, and hence

$$a \in (P/s \restriction A)^0 \subseteq (\mathcal{L}_A(P/s \restriction A))^0 = (\mathcal{L}_A(P/s))^0$$

and it follows that $a \in (P/s)^0$ as required.

This completes the proof that $(P/s)^0 = ((P_A \underset{\emptyset}{\|} P_B)/s)^0$.

One half of the proof for refusals is very like the above. The other is more interesting since it uses the full power of the enhanced 'after' operator contained in the definition of $\mathcal{L}\mathbf{S}$INV. This is to show that any refusal of $(P_A \underset{\emptyset}{\|} P_B)/s$ is a refusal of P/s. If there were a counter-example, $Y \cup Z$, to this assertion, then we have $(s \upharpoonright A, Y) \in \textsc{fails}(P_A)$ and $(s \upharpoonright B, Z) \in \textsc{fails}(P_B)$ since $(s, Y \cup Z) \in \textsc{fails}(P_A \underset{\emptyset}{\|} P_B)$. The definitions of P_A and P_B then imply that $(s \upharpoonright A, Y)$ and $(s \upharpoonright B, Z)$ are both members of $\textsc{fails}(P)$. Hence, by $\mathcal{L}\mathbf{S}\mathrm{INV}_A(P)$ applied to the failure $(s, \emptyset)$, $(s, Y) \in \textsc{fails}(P)$. By assumption, $Z \notin \textsc{refs}(P/(s, Y))$ and hence

$$\begin{aligned}
Z \notin\ & \textsc{refs}(\mathcal{L}_B(P/(s, Y))) \\
& = \textsc{refs}(\mathcal{L}_B(P/(s \upharpoonright B, Y \cap B))) \qquad \text{by } \mathcal{L}\mathbf{S}\mathrm{INV}_B(P) \\
& = \textsc{refs}(\mathcal{L}_B(P/s))
\end{aligned}$$

as $Y \cap B = \emptyset$. But this contradicts what we already know. ∎

The above lemma proves the remainder of the theorem, since given part (a) it is an easy induction that the two sets $\textsc{traces}(P_A \underset{\emptyset}{\|} P_B)$ and $\textsc{traces}(P)$ are identical. ∎

Suppose that process P is separable and that all its components are deterministic. Since a parallel composition of deterministic processes is always deterministic [Hoa85], P itself will be deterministic. From Theorems 1.3 and 2.2 we therefore obtain:

Corollary 2.4 *Process P is strongly separable with respect to partition $\{A, B\}$ if, and only if,*

$$\mathcal{L}\mathrm{IND}_B(P) \wedge \mathcal{L}\mathrm{IND}_A(P).$$

3 Secure Composition

The previous section has shown the close link between separability of a process into disjoint components and security conditions. Separability when provable, is evidence of the lack of information flow. However we should point out that separability is, surprisingly, *not* sufficient evidence to exclude information flow. For while clearly any process that is actually *constructed* as the disjoint parallel composition of two processes will be secure no matter what the internal structure of the processes, mere semantic equivalence to such a system (which is what separability states) turns out not to be good enough. The reason is that the resolution of nondeterminism within the system may create insecurities. This is discussed at length in [Ros95]. Here, we merely give a simple example.

$$CHAOS_{A \cup B} \; = \; STOP \sqcap x : A \cup B \to CHAOS_{A \cup B}$$

is the most nondeterministic divergence-free process with alphabet $A \cup B$. And it is separable (it equals $CHAOS_A \underset{\emptyset}{\|} CHAOS_B$). Constructed as above we would have few doubts as to the security of this process, but unfortunately it is semantically equal to the process

$$CHAOS_{A \cup B} \sqcap P$$

for any divergence-free P at all, however insecure. A process like this, with an internal mechanism that allows it to behave securely or insecurely (perhaps being certain to choose the latter because of considerations below the level of modelling) should not be regarded as secure. But it is still separable. It is only *strong* separability that should be regarded as establishing the absence of information flow (though noting it still ignores timing issues).

From a more practical point of view, however, separability of processes is far from a universal security condition. The main reason for this is that non-interference is generally regarded as an asymmetric property in the sense that if A must not interfere with B, we need not always exclude the reverse. Fortunately, Corollary 2.4 shows how we can "skew" the condition of separability to give an asymmetric property: we might want to demand $\mathcal{L}\mathrm{IND}_B(P)$ but do not necessarily require A to be independent as well.

Secondly, even if the final result of a development is separable, it is highly unlikely that the actual system is built as the disjoint parallel composition of two processes. (If it were, there would not be much need of a detailed formal analysis!) If we are to seek rules which allow us to build up secure systems, we will need ones that allow us to deal with cases of composing processes whose alphabets do intersect.

What is required is a development method that allows functional designs *and* verification of their security properties. Such a method would permit the definition of components, not necessarily with disjoint alphabets, and their composition to yield systems secure as a whole. The compositional laws we present below show under which conditions we can combine subsystems such that their composition is guaranteed to be secure. We concentrate here on the determinism-based independence properties, since we believe they give the most satisfactory definition of absence of information flow.

The context of the following two laws is a system in which events A are required not to interfere with events B. They simply allow us to add a process at one end of a secure system without violating security.

THEOREM 3.1 *If $\mathcal{L}\mathrm{IND}_B(P)$ holds, $\alpha Q \subseteq A$, and process Q does not diverge, then*

$$\mathcal{L}\mathrm{IND}_B(P \underset{\alpha P \cap \alpha Q}{\|} Q).$$

THEOREM 3.2 *If $\mathcal{L}\mathrm{IND}_B(P)$ holds, $\alpha Q \subseteq B$, and process Q is deterministic, then*

$$\mathcal{L}\mathrm{IND}_B(P \underset{\alpha P \cap \alpha Q}{\|} Q).$$

The final law states that we safely combine two secure components, confident in the knowledge that the result will be secure as well.

THEOREM 3.3 *If both $\mathcal{L}\mathrm{IND}_B(P)$ and $\mathcal{L}\mathrm{IND}_B(Q)$ hold, then*

$$\mathcal{L}\mathrm{IND}_B(P \underset{\alpha P \cap \alpha Q}{\|} Q).$$

PROOF

We base the proof on the following lemma, which holds by virtue of properties of the $|||$ operator.

LEMMA 3.4 *Suppose S is divergence-free. Then $\mathcal{L}_B(S)$ is non-deterministic if, and only if, there exist $t_1, t_2 \in \textsc{traces}(S)$ and $b \in B$ such that*

$$t_1 \mathbin{|\backslash} B = t_2 \mathbin{|\backslash} B \ \wedge$$
$$t_1 \hat{}\langle b \rangle \in \textsc{traces}(S) \ \wedge \ (t_2, \{b\}) \in \textsc{fails}(S).$$

If $RUN_A \mathbin{|||} (P \underset{\alpha P \cap \alpha Q}{\|} Q)$ were non-deterministic (but free of divergence), we would therefore know that there must be traces t_1, t_2 and an event $b \in B$

such that

$$t_1 \upharpoonright B = t_2 \upharpoonright B \;\wedge$$
$$t_1 {}^\frown \langle b \rangle \in \text{TRACES}(P \underset{\alpha P \cap \alpha Q}{\parallel} Q) \;\wedge$$
$$(t_2, \{b\}) \in \text{FAILS}(P \underset{\alpha P \cap \alpha Q}{\parallel} Q).$$

We distinguish two cases: either b is in both alphabets of P and Q, or b is only in one of the alphabets of the processes, in which case we assume wlog $b \in \alpha P$.

Case $b \in \alpha P - \alpha Q$. Since P does not need Q's cooperation for b we know it must be possible for P to refuse this event after t_2. Also, it is P that contributes b after t_1, and so we conclude

$$(t_2 \upharpoonright \alpha P, \{b\}) \in \text{FAILS}(P) \;\wedge\; (t_1 \upharpoonright \alpha P){}^\frown \langle b \rangle \in \text{TRACES}(P)$$

by the properties of parallel composition, which contradicts the assumption of $\mathcal{L}\text{IND}_B(P)$.

Case $b \in \alpha P \cap \alpha Q$. The refusal of b after t_2 may have been caused by either P or Q; we assume wlog that P refuses the event after t_2. Since both processes have to synchronise for b to occur after t_1, we must come to precisely the same conclusion as in the first case.

In either case we derive a contradiction with the assumptions, and so the theorem holds. ∎

4 Example

The following example specification is intended to illustrate how the ideas described in the previous section allow the functional design of a system that is secure by construction.

4.1 Informal Requirements

A system is required in which four users are to share access to common resources. These resources are binary variables on which the following operations can be carried out:

- reading the current value of the variable,
- toggling (i.e. complementing) the current value, and

- setting the value to 1, which is only possible if the variable is unset (value 0).

There are three variables X, Y, and Z, which are to provide the following functionality to their four users A, B, C, and D.

1. Variable X can be read by A when Y is set, and toggled by any of A, C, and D when Z is set.

2. Variable Y can be read by A and B when Z is set, toggled by B and D any time, and set only by B.

3. Variable Z can be read by any user any time, but toggled only by D.

The security requirements we need to take into account are:

- A must not interfere with any other user.

- Neither B nor C must interfere with D, and B and C must not interfere with each other.

- D may interfere with any user.

4.2 Design

Using USER to denote the set $\{A, B, C, D\}$ and BIT to denote the range of the binary variables, we specify the access operations as

$$\begin{aligned}
\mathit{Read} &= \{\, \mathit{readX}.u.b, \mathit{readY}.u.b, \mathit{readZ}.u.b \mid u \in \mathit{USER}, b \in \mathit{BIT} \,\} \\
\mathit{Set} &= \{\, \mathit{setY}.u \mid u \in \mathit{USER} \,\} \\
\mathit{Toggle} &= \{\, \mathit{toggleX}.u, \mathit{toggleY}.u, \mathit{toggleZ}.u \mid u \in \mathit{USER} \,\}
\end{aligned}$$

The alphabets of the processes implementing the variables are given by

$$\begin{aligned}
\alpha \mathit{VARX} &= \{\, \mathit{readX}.u.b, \mathit{toggleX}.u \mid u \in \mathit{USER}, b \in \mathit{BIT} \,\} \\
\alpha \mathit{VARY} &= \{\, \mathit{readX}.u.b, \mathit{readY}.u.b, \mathit{setY}.u, \mathit{toggleY}.u \\
&\qquad \mid u \in \mathit{USER}, b \in \mathit{BIT} \,\} \\
\alpha \mathit{VARZ} &= \{\, \mathit{readY}.u.b, \mathit{readZ}.u.b, \mathit{toggleX}.u, \mathit{toggleZ}.u \\
&\qquad \mid u \in \mathit{USER}, b \in \mathit{BIT} \,\}
\end{aligned}$$

It is straightforward to implement the desired functionality with a single process for each variable.

$$
\begin{aligned}
\mathit{VARX}(x) \;=\;& (\mathit{readX}.A!x \rightarrow \mathit{VARX}(x))\\
& \Box\, (\mathit{toggleX}.A \rightarrow \mathit{VARX}((x+1)\mathrm{mod}2))\\
& \Box\, (\mathit{toggleX}.C \rightarrow \mathit{VARX}((x+1)\mathrm{mod}2))\\
& \Box\, (\mathit{toggleX}.D \rightarrow \mathit{VARX}((x+1)\mathrm{mod}2))\\
\mathit{VARY}(y) \;=\;& (\mathit{readY}.A!y \rightarrow \mathit{VARY}(y))\\
& \Box\, (\mathit{readY}.B!y \rightarrow \mathit{VARY}(y))\\
& \Box\, (\mathit{toggleY}.B \rightarrow \mathit{VARY}((y+1)\mathrm{mod}2))\\
& \Box\, (\mathit{toggleY}.D \rightarrow \mathit{VARY}((y+1)\mathrm{mod}2))\\
& \Box\, (\text{if}\,(y=0)\\
& \qquad \text{then}\,(\mathit{setY}.B \rightarrow \mathit{VARY}(1))\\
& \qquad \text{else}\,(\mathit{readX}.A?x \rightarrow \mathit{VARY}(y)))\\
\mathit{VARZ}(z) \;=\;& (\mathit{readZ}?\mathit{user}!z \rightarrow \mathit{VARZ}(z))\\
& \Box\, (\mathit{toggleZ}.D \rightarrow \mathit{VARZ}((z+1)\mathrm{mod}2))\\
& \Box\, (\text{if}\,(z=1)\\
& \qquad \text{then}\,(\mathit{toggleX}?\mathit{user} \rightarrow \mathit{VARZ}(z)\\
& \qquad\qquad \Box\, \mathit{readY}.A?y \rightarrow \mathit{VARZ}(z)\\
& \qquad\qquad \Box\, \mathit{readY}.B?y \rightarrow \mathit{VARZ}(z))\\
& \qquad \text{else}\,\mathit{STOP})
\end{aligned}
$$

Let us compose the system in two steps

$$
\begin{aligned}
\mathit{VARYZ} \;=\;& \mathit{VARY}(0) \underset{\alpha \mathit{VARY} \cap \alpha \mathit{VARZ}}{\parallel} \mathit{VARZ}(0),\\
\mathit{SYSTEM} \;=\;& \mathit{VARYZ} \underset{\alpha \mathit{VARYZ} \cap \alpha \mathit{VARX}}{\parallel} \mathit{VARX}(0)
\end{aligned}
$$

and check at the same time whether the compositions preserve our security requirements. This can be done by applying the compositional law described in Theorem 3.3 four times, for each step. Before doing this, we need to define the alphabets of the users of the system.

$$
\begin{aligned}
\mathit{userA} \;=\;& \{\, \mathit{readX}.A.b, \mathit{readY}.A.b, \mathit{readZ}.A.b, \mathit{toggleX}.A \mid b \in \mathit{BIT}\,\},\\
\mathit{userB} \;=\;& \{\, \mathit{readY}.B.b, \mathit{readZ}.B.b, \mathit{setY}.B, \mathit{toggleY}.B \mid b \in \mathit{BIT}\,\},\\
\mathit{userC} \;=\;& \{\, \mathit{readZ}.C.b, \mathit{toggleX}.C \mid b \in \mathit{BIT}\,\},\\
\mathit{userD} \;=\;& \{\, \mathit{readZ}.D.b, \mathit{toggleX}.D, \mathit{toggleY}.D, \mathit{toggleZ}.D \mid b \in \mathit{BIT}\,\}
\end{aligned}
$$

and define $\mathit{userBD} = \mathit{userB} \cup \mathit{userD}$, $\mathit{userCD} = \mathit{userC} \cup \mathit{userD}$, and $\mathit{userBCD} = \mathit{userB} \cup \mathit{userCD}$.

For the first composition, there are four proof obligations for security of process VARYZ.

1. show independence of *userD* in $VARY(0)$ and in $VARZ(0)$,
2. show independence of *userBD* in $VARY(0)$ and in $VARZ(0)$,
3. show independence of *userCD* in $VARY(0)$ and in $VARZ(0)$,
4. show independence of *userBCD* in $VARY(0)$ and in $VARZ(0)$.

This is sufficient, by Theorem 3.3, to prove that

$$\mathcal{L}\mathrm{IND}_{userD}(VARYZ) \wedge \mathcal{L}\mathrm{IND}_{userBD}(VARYZ) \wedge$$
$$\mathcal{L}\mathrm{IND}_{userCD}(VARYZ) \wedge \mathcal{L}\mathrm{IND}_{userBCD}(VARYZ).$$

Given these, there are again four proof obligations for the second step.

1. show independence of *userD* in $VARX(0)$,
2. show independence of *userBD* in $VARX(0)$,
3. show independence of *userCD* in $VARX(0)$,
4. show independence of *userBCD* in $VARX(0)$.

Verifying these will guarantee

$$\mathcal{L}\mathrm{IND}_{userD}(SYSTEM) \wedge \mathcal{L}\mathrm{IND}_{userBD}(SYSTEM) \wedge$$
$$\mathcal{L}\mathrm{IND}_{userCD}(SYSTEM) \wedge \mathcal{L}\mathrm{IND}_{userBCD}(SYSTEM)$$

and thus the security in the overall system. Of course, we can check these conditions directly, but this will yield the expected result.

We point out that all conditions above can be verified using the CSP model checker FDR[2]. This tool allows direct verification of whether a process is deterministic, as described in [RWW94].

4.3 An Alternative Design

Let us now enhance the functionality of the system. Variables Y and Z should remain unchanged, while X should additionally allow user B to set

[2] FDR (Failures-Divergence Refinement) is a product of Formal Systems (Europe) Ltd., 3 Alfred St., Oxford OX1 3EH, UK.

its value. With this change, we have to re-define the sets $\alpha\,VARX$ and $userB$ as

$$
\begin{aligned}
\alpha\,VARX &= \{\, readX.u.b, setX.u, toggleX.u \mid u \in USER, b \in BIT \,\} \\
userB &= \{\, readY.B.b, readZ.B.b, setX.B, setY.B, toggleY.B \\
&\qquad \mid b \in BIT \,\}
\end{aligned}
$$

Our system implementation has to be modified in one process:

$$
\begin{aligned}
VARX(x) = {} & (readX.A!x \to VARX(x)) \\
& \Box\, (toggleX.A \to VARX((x+1)\mathrm{mod}2)) \\
& \Box\, (toggleX.C \to VARX((x+1)\mathrm{mod}2)) \\
& \Box\, (toggleX.D \to VARX((x+1)\mathrm{mod}2)) \\
& \Box\, (\text{if } (x=0) \\
& \qquad \text{then } setX.B \to VARX(1) \\
& \qquad \text{else } STOP)
\end{aligned}
$$

The complete system is composed in the same two steps as described in the initial design, and the proof obligations are precisely the same. For the first step, all conditions do indeed hold (since we have not changed $VARY$ or $VARZ$), which again establishes that $VARYZ$ is secure.

The final composition, however, does *not* preserve security. It is not possible to meet any of the obligations for the second step. This is the case because user B may find the event $setX.B$ refused after either A or C has toggled X, and thus either can interfere with B.

The security breach is caught by our conditions as follows. For condition (2.) for instance, we require the process

$$RUN_{userAC} \mathbin{|||} VARX(0)$$

to be deterministic. It is not, however, since after (for example) trace $\langle toggleX.A\rangle$ the event $setX.B$ may be accepted or refused in a non-deterministic fashion. Thus neither $\mathcal{L}\mathrm{IND}_{userBD}(VARX(0))$ nor $\mathcal{L}\mathrm{IND}_{userBD}(SYSTEM)$ holds, the latter because process

$$RUN_{userAC} \mathbin{|||} SYSTEM$$

may accept or refuse $setX.B$ after (e.g.) trace $\langle toggleZ.D, toggleX.C\rangle$.

We note that none of the other pre-conditions for the composition holds, since the processes

$$
\begin{aligned}
& RUN_{userABC} \mathbin{|||} VARX(0) \\
& RUN_{userAB} \mathbin{|||} VARX(0) \\
& RUN_{userA} \mathbin{|||} VARX(0)
\end{aligned}
$$

are non-deterministic for the same reasons.

5 Conclusions

We have seen both how security properties can be used to decompose processes, and how security properties are preserved under parallel composition.

It is satisfying to see that the historically important property of separability can be characterised using our abstraction mechanisms, but the reader should also note the limitations we pointed out on the soundness of separability as a security specification. Only in the context of deterministic systems do independence and invariance collapse down to the same predicate and thus either condition (or separability) may be used to prove lack of information-flow.

Since the determinism-based independence conditions are the most satisfactory definitions of absence of flow, we have concentrated on their compositional properties, though similar studies (and doubtless similar results) could be obtained for others. We have shown a number of laws which preserve event independence under parallel composition. These compositional properties have a two-fold relevance in computer security. Firstly, they allow to place emphasis – if so desired – on the functional properties of system components without sacrificing security concerns. Secondly, they provide the key to formal or automatic verification since the security of the whole system follows from that of its parts.

It is worth noting that, as discussed in [RWW94] and [Ros95], conditions based on determinism map well onto existing model-checking technology. The examples presented in the previous section have all been verified (or shown insecure) using the tool FDR, which has the capability of handling much larger examples than this one. Of course the existence of composition laws like ours should extend the range of systems which come within the reach of tools like FDR.

References

[All91] P. G. Allen. "A Comparison of Non-interference and Non-deducibility using CSP", *Proc. 1991 IEEE Computer Security Workshop*, pp 43-54. IEEE Computer Society Press 1991.

[Bur89] R. Burnham. *The Specification of Security in Distributed Computing Systems*, Oxford University MSc Thesis, 1989.

[Gra92] J. Graham-Cumming. *The Formal Development of Secure Systems*, Oxford University DPhil Thesis, 1992.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*, Prentice Hall 1985.

[Jac90] J. Jacob. “Separability and the Detection of Hidden Channels”, *Information Processing Letters* 34(1990) 27–29.

[Ros95] A. W. Roscoe. “CSP and Determinism in Security Modelling”, to appear in *Proc. 1995 IEEE Symposium on Security and Privacy.*

[RWW94] A. W. Roscoe, J. C. P. Woodcock, L. Wulf. “Non-interference through Determinism”, Proc. Third European Symposium on Research in Computer Security (ESORICS-94), Springer LNCS 875.

[Rus81] J. M. Rushby. “The Design and Verification of Secure Systems”, *ACM Operating Systems Rev.* **15***(5).*

[Rya91] P. Y. A. Ryan. “A CSP Formulation of Non-interference”, *Cipher*, pp 19-27. IEEE Computer Society Press, 1991.

A CSP Summary

The following provides a brief summary of the CSP notation as used in this paper; further details may be found in [Hoa85].

A.1 Trace Notation

$\langle e \rangle$	trace containing (only) event e
$s\hat{\ }t$	concatenation of traces s and t
$s \restriction X$	s restricted to events in set X

A.2 Processes

αP	alphabet (possible events) of process P
$(P)^0$	possible initial events of P
P/t	process P after it has engaged in trace t
$P \underset{A}{\parallel} Q$	parallel composition with synchronisation (only) on A
$P \mid\mid\mid Q$	parallel interleaving (without synchronisation)
$P \sqcap Q$	internal (non-deterministic) choice between P and Q
$P \,\Box\, Q$	external (deterministic) choice between P and Q

Two special processes are $STOP$ and RUN. Process $STOP$ never engages in any event. RUN_A is always willing to contribute an event from set A:

$$RUN_A \;=\; a : A \rightarrow RUN_A$$

A.3 Semantic Model

In the failures-divergences model, each process P is determined by its failure set, $\textsc{fails}(P)$, and its divergence set, $\textsc{divs}(P)$.

- each failure is a pair (s, X) where s is a finite trace of the process and X is a set of events which it may refuse after s, and
- each divergence is a finite trace on which the process can perform an infinite sequence of internal actions.

The events initially refused by P are denoted $\textsc{refs}(P)$. Two processes are regarded as equal in the model if they agree in their failures and divergences.

A process P is *deterministic* if and only if (1) it is free of divergence; and (2) it satisfies

$$(tr, \{a\}) \in \textsc{fails}(P) \;\Rightarrow\; tr\hat{\ }\langle a \rangle \notin \textsc{traces}(P)$$

for all traces tr, that is, there never is a choice between accepting and refusing an event.