Some Laws of Non-interference

John Graham-Cumming*
Oxford University Computing Laboratory
Programming Research Group
8-11 Keble Road
Oxford, OX1 3QD
7 April 1992

Abstract

We define non-interference in the algebra of CSP; that definition leads to simple proof rules for non-interference concerning, amongst other things, composition of systems exhibiting non-interference.

We work through a case study of a multi-level secure system to illustrate those laws.

1 Introduction

The formal methods approach to system design is well documented (see, for example, [Ha85], [NC88] and [PST91]) and the need for formal methods in design of secure systems is well known (see, for example, [OBI]). The techniques used for development of secure systems need to take into account the special requirements of security, as current (functional) techniques (see, for example, [Jot81] and [Mo90, SS]) have been shown inadequate for the purposes of security (see [Ja88]).

The algebraic approach to producing systems simplifies the reasoning required and enables automated proofs. We use the algebraic approach when considering security. It also means that we are not restricted to considering systems in terms of a particular model such as traces.

This paper extends and improves the treatment of non-interference given in [GS91]. In that paper the authors give the definition of non-interference in terms of a traces model of computation. That model is derived from the traces of CSP (see [Ho85]). We extend the definition of non-interference in [GS91] by giving it algebraically. The algebraic definition permits improved proofs of non-interference and a small collection of laws for the construction of systems exhibiting non-interference is given. The proofs of those laws are given in the algebra of CSP.

Of some interest are laws for composition (see [Mc87]) and the following two are examined here: one for parallel composition and one for (external) choice composition. The move from traces to algebra introduces the possibility of non-deterministic systems and these are discussed in section 5 along with other approaches to non-interference and security.

In addition to composition of secure systems other laws to help in the construction of such systems are of interest. Those for prefixing and recursion are considered here. In addition composition may be more interesting if not considered in the form:
If systems $S$ and $T$ have property $P$ then the composition of $S$ and $T$ has property $P$.

That is because systems are rarely decomposed into parts having the same properties; a more common approach is to decompose into parts with properties that compose to provide the overall, desired property. One such law is given (law 6) and more may be found in [GC].

An example of an MLS (multi-level secure) system is given. Using the laws of section 3 that example is shown systematically to have the desired non-interference property without the need for an unwinding theorem (see [GM84]) or complex verification of security.

## 2 Non-interference expressed in CSP

Non-interference was first defined in [GM82] and later in [GM84]. Both those definitions are based on a state machine model of computation. In [GS91] (and elsewhere; see [Mc87] and [R90]) the definition of non-interference is given in terms of traces, or sequences of events. McCullough's definition distinguished between input and output, whereas that of [GS91] did not. The operations of a system are called events and the system and environment (in this case the users) have both to agree to engage in a particular event for it to occur.

The notation of CSP is used to express non-interference; the definition is algebraic and is valid in any semantic model of CSP which obeys that algebra.

### 2.1 Systems

We model a system by a CSP process $S$ which has alphabet $\alpha S$ (the set of events in which it may engage). Such a system can be considered to have users in two ways: a user may be specified by his activity (another CSP process) and act in parallel with $S$; or a user may be identified by a set of events from the alphabet of $S$ (in that case the user's activity is not specified and is assumed to be—at worst—any event at any time: the process $\text{RUN}$ on that user's alphabet).

We choose to identify the users of a system by the set of events in which they may engage. Then a user $u$ of system $S$ is a set of events $u \subseteq \alpha S$. We refer to that set as the user's interface as it expresses the set of events through which the system and a user may interact.

**Example 1:** The system $P$ has two users $u = \{uop\}$ and $v = \{vop\}$; each user has a single, named event. Initially the system can engage in either of the events. After $v$'s event has occurred the system stops whether or not $u$'s event has occurred.

$$P \triangleq (uop \rightarrow vop \rightarrow \text{STOP}_{\alpha P})$$

$$\left\{ vop \rightarrow \text{STOP}_{\alpha P} \right\}.$$ 

Its alphabet is $\alpha P = \{uop, vop\}$. The $|$ allows the environment (the users) to choose either branch by the event initially engaged in. $\triangle$

A user $u$'s activity is expressed by the process $\overline{uS}$ which represents all possible activity by $u$ at his interface, excluding the other users of the system.

**Definition 1:** If system $S$ has user $u$ then $\overline{uS} \triangleq \text{STOP}_{\alpha S \setminus u}$. $\triangle$

That definition allows $u$ to perform any event at his interface and stops all other activity. This is the worst case of the user's activity and is used to give the definition of non-interference.

**Example 2:** In the system $P$ (of example 1)
user v’s possible activity is
\[ \overline{v}_P = \text{STOP}_a P_v = \text{STOP}_{\{\text{uop}\}} : \]
the process which allows uop to occur at any time and prevents uop from happening. △

We note two properties of \( \overline{v}_S \) that will be useful later; their proofs are simple manipulations in the algebra of CSP.

Lemma 1: If systems S and T have users u and v then \( \overline{v}_S || T = \overline{v}_S || \overline{v}_T = \overline{v}_T || \overline{v}_S \) and \( (u \cup v)_S = \overline{v}_S || (u \cup v)_S \). □

The set of valid sequences of events of S is called the set of traces of S and denoted \( \tau_S \) and consists of finite sequences of elements of \( \alpha_S \).

2.2 Non-interference Defined

Consider a system S with two users: u and v. To define when user u does not interfere with user v we define when the system appears in identical “states” from the viewpoint of user v. Then u does not interfere with v if u’s events do not cause v’s view of the system to change.

We define identical “states” by viewing the system S in terms of its traces and defining equivalence between traces. Note that as the system is not modelled by a state machine there is no explicit mention of state. Rather the states of a CSP process are equivalence classes of traces under some equivalence relation. Such a relation is defined below, for each user of a system, and if the states are of interest they can be constructed by finding the equivalence classes (see [GS91] for more details).

In [GS91] non-interference is defined using the equivalence \( \approx_v \). There two traces s and t are said to be equivalent, from the viewpoint of a user v, if for all sequences of events r from v
\[ s \circ r \in \tau_S \quad \text{iff} \quad t \circ r \in \tau_S. \]

That definition means that two traces are equivalent if the activity a user can engage in after one trace is exactly that which he can engage in after the other. Thus the two traces are considered indistinguishable by v and we say that they are \( v \)-equivalent.

That definition can easily be cast in the algebra of CSP using the after operator: /.

Example 3: Consider again system P from example 1; \( \{\text{uop}\} \) is a trace of P and so we can consider P after that trace.
\[ P / \{\text{uop}\} = \text{vop} \rightarrow \text{STOP}_a P. \]

In that case, after \( \{\text{uop}\} \) has occurred P behaves as the process \( \text{vop} \rightarrow \text{STOP}_a P \). △

Consider two traces s and t of S. To define \( \approx_S^v \) algebraically we need to determine when the processes S/s and S/t appear the same from v’s viewpoint. In the traces definition of \( \approx_S^v \), given above, that is done by examining all possible sequences of events from v. Similarly the process \( \overline{v}_S \) can be used to define \( \approx_S^v \) algebraically. \( \overline{v}_S \) is the process that allows any of v’s events to occur.

Definition 2: Traces s and t are \( v \)-equivalent in system S with user v if
\[ (S/s) || \overline{v}_S = (S/t) || \overline{v}_S. \]
and write that as \( s \approx_v^S t \). △
Definition 3: Systems $S$ and $T$ (both with user $v$) are $v$-equivalent when $\alpha S = \alpha T$ and

$$S||uS = T||vS.$$ 

That is written $S \equiv_v T$. \hfill \triangle

Those two definitions are connected by the equivalence:

$$s \sim^S v t \iff S/s \equiv_v S/t.$$ 

Note that both $\sim^S v$ and $\equiv_v$ are equivalence relations; the proofs are trivial.

Lemma 2: $\sim^S v$ and $\equiv_v$ are equivalence relations. \hfill \square

Example 4: Consider the system $P$ of example 1. Then, as example 2 showed, $\nu P = \text{STOP}_{(\nu P)}$ and

$$(P/\langle \rangle)\nu P = \nu P \rightarrow \text{STOP}_{\nu P},$$

$$(P/\langle \nu P \rangle)\nu P = \nu P \rightarrow \text{STOP}_{\nu P},$$

$$(P/\langle \nu P, \nu P \rangle)\nu P = \text{STOP}_{\nu P},$$

$$(P/\langle \nu P \rangle)\nu P = \text{STOP}_{\nu P}.$$ 

Hence $\langle \rangle$ and $\langle \nu P \rangle$ are $v$-equivalent and $\langle \nu P, \nu P \rangle$ and $\langle \nu P, \nu P \rangle$ are $v$-equivalent; that is, the processes $P/\langle \rangle$ and $P/\langle \nu P \rangle$ are $v$-equivalent and $P/\langle \nu P, \nu P \rangle$ and $P/\langle \nu P \rangle$ are $v$-equivalent. \hfill \triangle

User $u$ does not interfere with user $v$ when the presence or absence of $u$'s events does not affect $v$'s view of the system. We use the function $\text{purge}_u$ to remove events belonging to $u$ from a trace. It is defined here for the singleton trace and it is strict and distributes through concatenation.

$$\langle e \rangle|_u \overset{\triangle}{=} \langle e \rangle, \quad \text{if } e \in u$$

$$\langle e \rangle|_u \overset{\triangle}{=} \langle e \rangle, \quad \text{otherwise.}$$

Definition 4: If system $S$ has users $u$ and $v$ then $u$ is non-interfering with $v$ iff

$$\forall t \in \tau S \bullet S/t \equiv_v S/(t|_u).$$

We write that as $u \not\sim v : S$. \hfill \triangle

That definition uses $\equiv_v$ to test equivalence from $v$'s viewpoint. $u$ does not interfere with $v$ when the system appears the same to $v$ after engaging in the trace $t$ or the trace $t|_u$, for all traces $t$ of $S$.

When we write $u \not\sim v : S$ we assume that both $u$ and $v$ are subsets of the alphabet of $S$ (we say they are users of $S$; they do not have to partition $\alpha S$) and that they are disjoint; i.e. $u \cap v = \emptyset$. That definition of non-interference, when viewed in terms of traces, is the definition given in [GS91].

Example 5: User $u$ does not interfere with user $v$ in the system $P$ of example 1. To see that consider each trace in turn (there are four traces) and verify that the system appears the same to $v$ whether or not $u$'s operations are present in that trace.

$\langle \rangle$ or $\langle \nu P \rangle$ Here $\langle \rangle|_u = \langle \rangle$ and $\langle \nu P \rangle|_u = \langle \nu P \rangle$ and this case is simple as $\equiv_v$ is reflexive (see lemma 2).

$\langle \nu P \rangle$ Here $\langle \nu P \rangle|_u = \langle \nu P \rangle$ and example 4 showed that $\langle \nu P \rangle$ and $\langle \rangle$ are $v$-equivalent.

$\langle \nu P, \nu P \rangle$ $\langle \nu P, \nu P \rangle|_u = \langle \nu P \rangle$ and example 4 showed that $\langle \nu P, \nu P \rangle$ and $\langle \nu P \rangle$ are $v$-equivalent.

Hence $u \not\sim v : P$. \hfill \triangle

In [GS91] the authors give an unwinding theorem using the equivalence $\sim_v^S$ with its definition given on traces. That unwinding theorem holds with the algebraic definition of $\sim_v^S$ and its proof is unchanged. We re-express the unwinding conditions of that theorem using $\equiv_v$.

Theorem 1: If system $S$ has users $u$ and $v$, and $\forall s, t \in \tau S; uop \in u; op \in \alpha S$

$$S/t \equiv_v S/(t \setminus \langle uop \rangle)$$

and
$S/s \equiv_s S/t \Rightarrow S/(s \sim (op)) \equiv_s S/(t \sim (op))$, then $u \not\rightarrow v : S$.

**Proof:** See [GS91, page 41]. □

Using the algebraic formulation of the unwinding conditions means that the laws of CSP algebra can be used (see [Ho85]) to verify those conditions. In the following section we simplify the proof of non-interference further by giving laws of non-interference.

### 3 Some Laws of Non-interference

CSP is rich with laws given in [Ho85] and has three principle composition operators: $||$ (parallel composition), $\triangledown$ (external choice composition) and $\sqcap$ (internal choice composition). Here the first two of those are examined and laws for them are given. Their semantic definitions are given in [Ho85], but we will concern ourselves with the laws those operators obey (references to relevant laws of CSP algebra from [Ho85] are given using the notation $Lx:pp$ to mean law $Lx$ on page $pp$).

We start with the trivial processes $STOP_A$ and $RUN_A$ for some alphabet (set of events) $A$. $STOP_A$ is the process which does nothing; $RUN_A$ is the process which is willing to engage in any event from $A$ at any time. They both have the same alphabet $A = \alpha STOP_A = \alpha RUN_A$.

Suppose that two users $u$ and $v$ are subsets of the alphabet $A$ and hence are users of both $STOP_A$ and $RUN_A$. Clearly in both these systems $u$ does not interfere with $v$. In the case of $STOP_A$ no activity is possible so $u$ cannot communicate with $v$ (via the process); in $RUN_A$ any event may occur at any time and so $u$ cannot communicate with $v$ through the process. Those facts are expressed by the following two laws.

- **Law 1:** $u \not\rightarrow v : STOP_A$ if $u$ and $v$ are users of $STOP_A$. □
- **Law 2:** $u \not\rightarrow v : RUN_A$ if $u$ and $v$ are users of $RUN_A$. □

The proofs of both those laws are trivial.

To build processes CSP uses prefixing (see [Ho85, page 25]); a process $S$ is prefixed by an event $e$ when that event is first engaged in and then the process behaves like $S$. Prefixing by $e$ is written $e \rightarrow S$

and the resulting process has alphabet $\alpha S$ ($e$ is assumed to be in $\alpha S$).

Suppose that $u$ and $v$ are users of $S$, that $u$ does not interfere with $v$ in $S$ ($u \not\rightarrow v : S$) and that $op$ is an event in which $u$ cannot engage ($op \in \alpha S \setminus u$). Then $u$ is non-interfering with $v$ in

$op \rightarrow S$,

because letting the system perform an event not belonging to $u$ initially does not affect whether $u$'s events interfere with $v$. The related law is

- **Law 3:** If $u \not\rightarrow v : S$ and $op \in \alpha S \setminus u$ then $u \not\rightarrow v : (op \rightarrow S)$.

**Proof:** Put $T = (op \rightarrow S)$; we verify definition 4 for $T$.

Consider a trace $t$ of $T$. The case $t = ()$ is trivial; if $t \neq ()$ then for some $s \in T S$ we have $t = (op) \sim s$, $t|_u = (op) \sim s|_u$, and

$T/t = T/((op) \sim s) \quad \text{[definition of $t$]}
= S/s \quad \text{[L2:53, L3A:53]}
\equiv_v S/s|_u \quad \text{[u \not\rightarrow v : S]}
= T/((op) \sim s|_u) \quad \text{[L2:53, L3A:53]}
= T/t|_u \quad \text{[op \in \alpha S \setminus u]}$
The result follows.

Example 5 showed that \( u = \{\text{uop}\} \) was non-interfering with \( v = \{\text{vop}\} \) in the process

\[
P = (\text{uop} \rightarrow \text{vop} \rightarrow \text{STOP}_\alpha P)
\]

with \( \alpha P = \{\text{uop}, \text{vop}\} \). That example was a particular case of a more general law:

**Law 4:** If \( u \not\leadsto v : S, \ uop \in u, \ op \in \alpha S \setminus u \) and

\[
T \triangleq (\text{uop} \rightarrow \text{op} \rightarrow S)
\]

\[
(\text{op} \rightarrow S)
\]

then \( u \not\leadsto v : T \).

**Proof:** Following the same pattern as the proof of law 3 we examine the three possible kinds of non-empty trace of \( T \). If \( t \in \tau T \) (non-empty) then \( t = (\text{uop}) \) or for any \( s \in \tau S: \ t = (\text{uop}, \text{op})^s \) or \( t = (\text{op})^s \). As an example of the form of the proof we examine the case \( t = (\text{uop}, \text{op})^s \) and verify definition 4 for \( T \):

\[
(U \setminus 0) \triangleq (\text{op})^s
\]

The result follows.

The proof of non-interference in example 5 simplifies to the application of law 1 followed by law 4; there is no need to perform complicated calculations of traces or of CSP algebra.

**Example 6:** We show that the process \( P \) (see example 1) has the property \( u \not\leadsto v : P \) (cf. example 5).

\[
\begin{align*}
\text{u} & \not\leadsto \text{v} : \text{STOP}_\alpha P & \text{[Law 1]} \\
\implies \text{u} & \not\leadsto \text{v} : (\text{uop} \rightarrow \text{vop} \rightarrow \text{STOP}_\alpha P) \\
& \quad (\text{vop} \rightarrow \text{STOP}_\alpha P) & \text{[Law 4]}
\end{align*}
\]

Hence \( u \not\leadsto v : P \).

The CSP operator \( || \) (see [Ho85, page 68]) is used to compose two processes by insisting that they agree to engage in events common to their alphabets. Events outside the common alphabet are not restricted.

**Example 7:** Consider the two processes \( P \) (as in example 1) and

\[
Q \triangleq (\text{uop} \rightarrow \text{bleep} \rightarrow \text{STOP}_\alpha Q)
\]

with \( \alpha Q = \{\text{uop}, \text{vop}, \text{bleep}\} \). The parallel combination of these processes is

\[
P || Q = (\text{uop} \rightarrow \text{bleep} \rightarrow \text{STOP}_\alpha P || \alpha Q)
\]

as the common alphabet is \( \{\text{uop}, \text{vop}\} \) and \( Q \) never agrees to a \( \text{vop} \) happening so no \( \text{vop} \) can occur. Both \( P \) and \( Q \) agree on the first event \( \text{uop} \) and hence it occurs followed by \( \text{bleep} \) as \( \text{bleep} \) is not in the common alphabet.

The law for parallel combination of processes that exhibit non-interference is very simple; non-interference is preserved by parallel combination. The operator \( \downarrow \) (see [Ho85, 1.6.2 (page 44)]) restricts a trace to the elements drawn from a particular set; in terms of a purge it is defined (when the trace is from \( \tau S \))

\[
\downarrow A = t|A = t|A S|A.
\]

This is the simplest law for parallel composition given here. It represents the case when both systems have the same security property (non-interference) which are then composed to form a system with the same property. As noted in the introduction more useful laws are given in [GC].
Law 5: If \( u \not \succ v : S \) and \( u \not \succ v : T \) then \( u \not \succ v : (S || T) \).

Proof: If \( R = S || T \) and \( t \in \tau R \) then

\[
\begin{align*}
(R/t)[\overline{v}]_R &= (S/t)_{\alpha S} \parallel \overline{t} (t/t)_{\alpha T} \parallel \overline{v}_R [L2:72] \\
&= (S/t)_{\alpha S} \parallel \overline{v}_R \parallel \overline{(T/t)_{\alpha T}} \parallel \overline{v}_R [L2:70] \\
&= (S/t)_{\alpha S} \parallel \overline{v}_R \parallel \overline{(T/t)_{\alpha T}} \parallel \overline{(T/t)_{\alpha T}} \parallel \overline{v}_R [\text{lemma 1}] \\
&= (S/t)_{\alpha S} \parallel \overline{v}_R \parallel \overline{(T/t)_{\alpha T}} \parallel \overline{(S/t)_{\alpha T}} \parallel \overline{v}_R [L2:70] \\
&= (S/t)_{\alpha S} \parallel \overline{v}_R \parallel \overline{(S/t)_{\alpha T}} \parallel \overline{(S/t)_{\alpha T}} \parallel \overline{v}_R [L2:72] \\
&= (R/t)[\overline{v}]_R \parallel \overline{v}_R [L2:72]
\end{align*}
\]

The result follows.

Law 5 shows that processes that exhibit non-interference are still non-interfering when cooperating. Sometimes it may be useful to compose processes which do not co-operate: they have disjoint alphabets and do not synchronize.

Composing such processes is aided by law 6:

Law 6: If \( u \not \succ v : S, w \not \succ x : T \) and \( \alpha S \cap \alpha T = \emptyset \) then \( (u \cup w) \not \succ (v \cup x) : (S || T) \).

Proof: If \( R = S || T, t \in \tau R, u' = u \cup w \) and \( v' = v \cup x \) then

\[
R/t[\overline{v}]_R = (S/t)_{\alpha S} \parallel \overline{(T/t)_{\alpha T}} \parallel \overline{v}_R [L2:72] = (S/t)_{\alpha S} \parallel \overline{v}_R \parallel \overline{(T/t)_{\alpha T}} \parallel \overline{(T/t)_{\alpha T}} \parallel \overline{v}_R [\text{lemma 1}] = (S/t)_{\alpha S} \parallel \overline{v}_R \parallel \overline{(S/t)_{\alpha T}} \parallel \overline{(S/t)_{\alpha T}} \parallel \overline{v}_R [L2:70] = (S/t)_{\alpha S} \parallel \overline{v}_R \parallel \overline{(S/t)_{\alpha T}} \parallel \overline{(S/t)_{\alpha T}} \parallel \overline{v}_R [L2:72] = (R/t)[\overline{v}]_R \parallel \overline{v}_R [L2:72]
\]

The result follows.

The external (or general) choice operator \( \parallel \) of CSP (used in example 1; see [Ho85, page 106]) can also be used to compose systems. When \( S \) and \( T \) are composed with \( \parallel \) the resulting system behaves like one of \( S \) or \( T \) after the first event has been engaged in; that first event selects either \( S \) or \( T \).

The choice composition of two processes that exhibit non-interference also exhibits non-interference if there are no events that both can perform on the first step. The requirement about the “first step” is formalised by the definition of the set of initials of a process:

\[
\iota S = \{ e | (e) \in \tau S \}
\]

(see [Ho85, page 109]); the initials of \( S \) is the set of events that \( S \) can engage in first.

Law 7: If \( u \not \succ v : S, w \not \succ x : T \) and \( \alpha S \cap \alpha T = \emptyset \) and \( u \cap \iota S = u \cap \iota T = \emptyset \) then \( u \not \succ v : (S || T) \).

Proof: If \( t \in \tau (S || T) \) then the case \( t = () \) is trivial; so suppose \( t \neq () \). In that case \( t \) is
trace of $S$ or $T$ (not both, since $s \cap t \neq \emptyset$) and we consider the case $t \in \tau S$:

\[
\frac{(S \parallel T)/t}{t} = \frac{S/t}{t} \quad [L2:108]
\]

\[
= \frac{S/t}{t} \quad [L2:108]
\]

The same proof is true of the case $t \in \tau T$ and the result follows.

Rather than treat recursion (see [Ho85, page 27]) in its most general form we give two laws of recursion from [GC] which are useful in the case study. The first law says that if $u \not\prec v : S$, $B \subseteq \alpha S \setminus \iota S$, and $u \cap \iota S = \emptyset$, then $u$ does not interfere with $v$ in

\[
\mu X \cdot (\langle op : B \rightarrow X \rangle \parallel S)
\]

with a restriction on $S$ to ensure that that recursion has a unique solution; that restriction concerns the guardedness of $S$ (see [Ho85, page 28]). (Note the restriction on the initials of $S$ and compare with law 7.) That non-interference comes about because the presence or absence of the $op$'s is irrelevant in that recursion. The related law is

\begin{align*}
\text{Law 8: If } u & \not\prec v : X \Rightarrow u \not\prec v : S, B \subseteq \alpha S \setminus \iota S, u \cap \iota S = \emptyset, S \text{ is guarded in } X \\
& \text{and } \\
& T = \mu X \cdot (\langle op : B \rightarrow X \rangle \parallel S)
\end{align*}

then $u \not\prec v : T$.

A similar law holds for mutual recursion, we give the case of two mutually recursive processes here (mutual recursion is defined in [Ho83, page 13]).

\begin{align*}
\text{Law 9: If } (u \not\prec v : X_0 \land u \not\prec v : X_1) & \Rightarrow (u \not\prec v : S_0 \land u \not\prec v : S_1), B_0 \subseteq \alpha S_0 \setminus \iota S_0, B_1 \subseteq \\
& \alpha S_1 \setminus \iota S_1, u \cap \iota S_0 = u \cap \iota S_1 = \emptyset, X = (X_0, X_1), \\
& S_0 \text{ guarded in } X, S_1 \text{ guarded in } X \text{ and }
\end{align*}

\[T = \mu X \cdot (\langle op : B_0 \rightarrow X_0 \rangle \parallel S_0, \\
\langle op : B_1 \rightarrow X_1 \rangle \parallel S_1 )_m
\]

(for $m \in \{0, 1\}$) then $u \not\prec v : T$.

The proofs are omitted for reasons of simplicity and may be found in [GC].

Now we proceed to a large example, with the set of laws given here to help in the proof of non-interference.

4 A Case Study

In order to illustrate the laws of the previous section we consider the construction of a multi-level secure system.

The system $M$ consists of a collection of binary stores (variables that store the digit 0 or 1), one for each user. Each binary store can be read (without destroying its contents) or the value stored can be inverted.

We suppose that the system has $n$ users and that each user is assigned a level. Each user (for $0 \leq i < n$) is called $U_i$ and consists of the set of events that user may engage in; user $U_i$ is assigned level $i$, with the usual ordering on natural numbers.

The policy is MLS expressed using non-interference (as in [GM84]):

\[0 \leq p, q < n \cdot q > p \Rightarrow u_q \not\prec u_p : M. \quad (1)\]

That policy says that there must be no interference from high users to low users. Each user $u_i$ will be able to change the state of the binary store belonging to $u_j$ ($i \leq j$) using event $j.i.flip$ and read the store belonging to $u_k$ ($i \leq k$) using channel $k.i.read$. The events user $u_i$ may engage in are given by the set

\[
u_i \triangleq \{k.i.flip \mid i \leq k < n\} \\
\cup \{k.i.read.x \mid x \in B ; 0 \leq k \leq i\}
\]

29
where $B = \{0, 1\}$. That set and the definition of $M_k$ include the restrictions “no read-up” and “no write-down” by disallowing events that would breach those restrictions.

First consider the construction of the binary store for one of the users $u_k$. That store is a variable (much like $\text{VAR}$ in [Ho85, page 137]) which, in this case, stores only the numbers 0 or 1. We call user $u_k$'s store $M_k$ and define it:

$$
M_k = \llbracket k.\text{flip} \rightarrow M_k^0 \rrbracket
$$

$$
M_k^x = \llbracket k.\text{flip} \rightarrow M_k^x \rrbracket
\llbracket k.\text{read!}x \rightarrow M_k^x \rrbracket
$$

On the first flip that store is initialised to 0 and then accepts further flip's (changing state as appropriate) or outputs the value it is holding when requested; note how the restrictions on which users can input and output values are governed by the $\llbracket \rrbracket$'s.

We define the set of events

$$
u^k_i \triangleq \{k.\text{flip} \mid i \leq k\} \\
\cup \{k.\text{read!}x \mid x \in B; i \geq k\}.
$$

$u^k_i$ is the set of events user $u_i$ uses to interact with $u_k$'s store. Using that definition we prove that the following non-interference is true in store $M_k$.

$$
0 \leq p, q, k < n \cdot j > i \Rightarrow u^k_q \not\vdash u^k_p : M_k. \quad (2)
$$

That is user $u_q$ does not interfere with user $u_p$ when interacting with $M_k$. That fact is verified by lemma 3.

**Lemma 3:** Equation 2 holds.

**Proof:** We write $M_k^x$ (for $x \in B$) as the mutual recursion $\mu X \cdot F(X)_x$ where $X = (X_0, X_1)$ and

$$
F_k(X) = \llbracket k.\text{flip} \rightarrow X_{-x} \rrbracket
\llbracket \llbracket k.\text{read!}x \rightarrow X_x \rrbracket \rrbracket
$$

Then $M_k = \llbracket k.\text{flip} \rightarrow \mu X \cdot F(X)_0 \rrbracket$. We start by assuming that $u^k_q \not\vdash u^k_p : X_0$ and $u^k_q \not\vdash u^k_p : X_1$ and prove that $u^k_q \not\vdash u^k_p : M_k$ when $q > p > k$ (other cases are similar).

$$
\Rightarrow u^k_q \not\vdash u^k_p : X_0 \land u^k_q \not\vdash u^k_p : X_1
$$

$$
\Rightarrow u^k_q \not\vdash u^k_p : (k.\text{flip} \rightarrow X_{-x})
$$

$$
\Rightarrow u^k_q \not\vdash u^k_p : k.\text{flip} \rightarrow X_{-x}
$$

$$
\Rightarrow u^k_q \not\vdash u^k_p : \llbracket k.\text{flip} \rightarrow M_k^0 \rrbracket
$$

$$
\Rightarrow u^k_q \not\vdash u^k_p : \llbracket \llbracket k.\text{flip} \rightarrow M_k^0 \rrbracket \rrbracket
$$

Hence $u^k_q \not\vdash u^k_p : M_k$. \quad \square

The total system consists of the $M_k$ acting in parallel:

$$
M = \llbracket k<n M_k \rrbracket
$$

and the system has the multi-level security property:

**Theorem 2:** Equation 1 holds.

**Proof:** By lemma 3 we know $u^k_q \not\vdash u^k_p : M_k$.

the alphabets of the $M_k$ are disjoint and $u_q =$
The case study shows how useful laws of non-interference are in developing a system and how a complex proof of security or unwinding is unnecessary.

\[ U_{k<n} u^k_p : M \]

\[ \Rightarrow (U_{k<n} u^k_p) \not\rightarrow (U_{k<n} u^k_p) : (||^{<n}_k M_k) \]

\[ \Rightarrow u_q \not\rightarrow u_p : M \]

Hence \( u_q \not\rightarrow u_p : M \) and equation 1 holds. \( \square \)

The case study shows how useful laws of non-interference are in developing a system and how a complex proof of security or unwinding is unnecessary.

5 Discussion

Using the CSP model of computation, and in particular the traces semantics, has caused some concern because of the synchronous nature of those systems. As there is no distinction between input and output in CSP (both are simply events) McLean ([Mcgl]) was concerned that a non-interfering system in CSP could not contain auditing information. He was unsure whether auditing events would introduce interference when a high level user reads auditing information about a low level user. It does not; the events which maintain an audit file are not events that must be engaged in with the permission of any user and so the low level user has no control over them happening; high level events reading the audit log are engaged in by the high level user, but need not introduce interference as they can be examined along with all other high level events in the way described above.

The composition given in McCullough’s paper [Mc87] was shown not to preserve non-interference. That was due to the way in which McCullough defined the composition. We have avoided giving a special composition operator and have used those of CSP; in this case (as noted by O’Halloran in [O90] and Allen in [A91]) non-interference is preserved by parallel composition \( || \).

Allen, in [A91] (and Ryan in [R90]), defined non-interference using CSP. That definition used the after operator but was restricted to the traces of a process. Our approach is more general, but, in the traces semantics, the definitions of non-interference given here and in [GS91] and [A91] are equivalent.

The Bell & LaPadula model (see [BL76]) introduced some of the common terms used in computer security. It is the view of the author that that model should be considered as describing some security techniques for use in building secure systems and that, when specifying security, information flow policies (such as non-interference or those in [Ja91]) be used. In the case study we used the restrictions “no read–up” and “no write–down” to help achieve the desired non-interference when we specified \( M_k \).

We have not yet mentioned how non-interference is affected by non-determinism (all our examples and laws have concerned deterministic systems). Introducing non-determinism does not cause problems with non-interference (as defined in CSP) because the algebraic definition already takes into account non-deterministic systems. Clearly further laws are required to aid the development of systems with non-determinism, but the treatment of non-interference given here includes non-deterministic systems.

6 Conclusion and Future Work

The small collection of laws, for the construction of systems, presented here are drawn from a much larger set in [GC]. That collection of laws includes proof rules for most CSP operators (with a number of variants for differ-
ent forms of composition) and considers security policies that are more general than non-interference.

A generalisation of non-interference is to allow non-interference between users when they perform sequences of commands; the definition given here means that each individual command must not cause interference with another user. In database systems (and elsewhere, remote procedure calls for example) sequences of operations, sometimes called transactions, occur and these sequences are considered atomic. That generalisation can be expressed algebraically, as non-interference has been, and laws proved for it. It proves useful in handling systems which typically engage in sequences of events for each "system call."

We hope to report on those pieces of work at a later date.

Acknowledgements

I would particularly like to thank: Jeff Sanders and Jeremy Jacob for their contribution to the ideas and approach to security and non-interference presented here; Jaisook Landauer for an interesting discussion of non-interference and the two anonymous referees.

I am supported by the Science and Engineering Research Council.

References


[BL76] Bell, D. E. and LaPadula, L. J., Secure Computer System: Unified exposition and Multics Implementa-


