What is intransitive noninterference?

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

M.H. Goldsmith
Formal Systems (Europe) Ltd
Keble Court, 26 Temple Street
Oxford OX4 1JS, UK *

Abstract

The term "intransitive noninterference" refers to the information flow properties required of systems like downgraders, in which it may be legitimate for information to flow indirectly between two users but not directly. We examine the usual definition of this property in terms of a modified purge function, and show that this is a distinctly weaker property than an alternative we derive from considerations of determinism.

1 Introduction

The term "noninterference" (closely analogous to, and often synonymous with, others such as "noninference" and "independence") refers to a situation in which a system has a number of users, and it is desired that the actions of one of these users have no effect on what another sees. The main motivating application is usually multi-level security (MLS) in which we are presented with a *security policy*: a relation between the users setting out which information flows are permissible (generally from low to high level) and thereby which are not (from high to low). In this domain, we end up trying to specify and prove that high-level users cannot "interfere" with low-level ones. In slightly different circumstances we might be concerned about potential disruption of a high-level user by low-level ones[1], so that the direction of undesirable interference gets reversed. For simplicity, in this paper we will, except when specifically stated otherwise, concentrate on the former, and assume we are attempting to prevent high-level information being revealed to low-level users.

A wide range of theories have been developed to support reasoning about noninterference, with a view to giving precise mathematical characterisations of the security and other properties that arise. Directly or indirectly these always involve studying how the range of low-level behaviours is affected by high-level actions. The semantic arenas in which these have been developed include the following:

- State machines [5] that make (possibly null) inputs and outputs with users on each cycle; the definitions of noninterference usually being based on *purge* functions that project behaviours into low-level actions only, or on the idea of *unwinding* in which each individual high-level action is specified to have no effect on the low-level view.
- Process algebras offer a more free-form model of interaction, often with *handshaken* communications, but noninterference properties are sometimes presented as complex-looking predicates on semantic models with no obvious route to automation.
- Trace theory [12], represents something of a compromise.

The authors have always preferred the process-algebra approach, since the essence of noninterference (or the lack of it) is in the interactions between a system and its users, and process algebras provide highly-developed theories specifically developed to handle the subtleties of interaction and communication. In [16] (and see also [14, 17]), the first author and others proposed a definition of noninterference based on the *determinism* of the low-level view: the actions of higher-level users are turned into nondeterministic choices by means of *abstraction* mechanisms, and if the low-level view is then deterministic we can conclude that it does not depend on anything that the high-level users do. This formulation (which has well-understood links with other process algebra versions, but avoids the difficulty that the refinement of a secure process may not be secure) has been the key to successful automated

*This paper is reproduced from the Proceedings of the 1999 IEEE Computer Security Foundations Workshop.

[1]These might be the actions of an untrusted user, or even events representing the random occurrence of system faults.

checking on FDR [3], as it is possible to automate efficient decision procedures for the CSP formulation of determinism. It is also possible, by varying the form of abstraction used on high-level behaviour, to make distinctions between the influences of high-level inputs and high-level outputs (a criticism that had previously been made of process algebra formulations). The most extensive study of the compositional and other properties of this type of noninterference can be found in [20].

Just about every initial approach to this subject begins by simplifying the problem to one where there are just two users – one high and one low – and analysing potential flows from one to the other. This is quite sufficient when the relation between users that defines permitted information flow – $A \rightsquigarrow B$ if information from A can flow to B – is *transitive* (i.e., $A \rightsquigarrow B \wedge B \rightsquigarrow C \Rightarrow A \rightsquigarrow C$). (We will always assume the relation is reflexive, namely that $A \rightsquigarrow A$ for all A.) Fixing A, we can then be confident that no information-flow is allowed from the set of users $\{B \mid B \not\rightsquigarrow A\}$ to any of the users in $\{C \mid C \rightsquigarrow A\}$ (or else such a C would provide a link between the given B and A in an application of the transitive rule). It follows that in order to check potential information flows to A we can divide up the alphabet of interaction of our system into two compound "users", one for each of these two sets.

Since the partition of the interface of a system for security purposes may well not exactly coincide with the division into users, we will usually refer to the members of these partitions as "domains". In the case where we are considering a CSP process P with alphabet A, these domains will just be subsets of A which partition it. A security policy $\mathcal{V}$ is then a specification of what these domains are plus the information flow relation $\rightsquigarrow$.

It seems intuitively obvious that the relation $\rightsquigarrow$ must be transitive: how can it make sense for A to have lower security level than B, and B to have lower level than C, without A having lower level than C? But this argument misses a crucial possibility, that some high-level users are trusted to *downgrade* material or otherwise influence low-level users. Indeed, it has been argued that no large-scale system for handling classified data would make sense without some mechanism for downgrading information after some review process, interval (e.g., the U.K. 30-year rule) or defined event (the execution of some classified mission plan, for example). Largely to handle this important problem, a variety of extended theories proposing definitions of "intransitive noninterference" have appeared, though we observe that this term is not really accurate, as it is in fact the *interference* rather than the *noninterference* relation which is not transitive. Perhaps the best way to read the term is as an abbreviation for "noninterference under an intransitive security policy".

The objective of this paper is to understand what is meant by intransitive noninterference, and to place it in the context (both theoretical and automation) of our existing determinism-based work.

The rest of this paper is constructed as follows: in the next section we summarise the CSP/determinism formulation of noninterference and put it context. We then survey previous work on intransitive noninterference, and build determinism-based analogues of some as well as a condition inspired directly by the intuition underlying the determinism formulation of noninterference. We show how these conditions can be checked automatically using techniques similar to those developed for earlier "transitive" properties.

2 A review of noninterference conditions

Definitions of noninterference based on state machines are usually in terms of either or both of the ideas of *purging* and *unwinding*. Purging involves applying a function to the history of the system up to some point which removes all those parts (typically communications with high-level agents) that should not influence what a given agent A sees. Usually the definition of noninterference can be paraphrased by the statement that the purged history is still a valid history of the system, and leads to a state which looks identical to A. Unwinding, on the other hand, looks at individual steps and asserts that each high-level action leaves the aspects of the system which affect A unchanged. It is frequently claimed that purge-based definitions are more intuitive, and that unwinding-based definitions are more amenable to automated proof. Therefore many theorems have been proved showing that a given notion of unwinding implies a corresponding purge-based definition. These definitions are generally unproblematic in the context of *deterministic* machines (where the machine's behaviour is completely determined by its initial state and inputs to date) but can generate debating points when they are applied to nondeterministic systems. In particular it is then quite likely that, where a purge definition of security is formulated in terms of behaviours from a fixed initial state rather than from all states, it will pass more processes than the corresponding unwinding one thanks to essentially the same example as the $\mathit{Chaos} \sqcap \mathit{LEAK}$ one discussed below for CSP.

As discussed in the introduction, many researchers have chosen to formulate noninterference properties in the world of process algebra. The most frequently used process algebra for this purpose has been CSP (e.g., [9, 19, 7]), perhaps because it offers semantic models that apparently lend themselves well to definitions which mimic the above. For example we can specify conditions (essentially those of [19, 7]) on the set $\mathit{traces}(P)$ of a process's finite communication sequences which closely mimic the purge and and unwinding definitions for state machines quoted above (H and L being respectively the sets of high and low level actions):

- $\forall\, s \in \mathit{traces}(P)\,.\, s \setminus H \in \mathit{traces}(P) \wedge (P/s)^0 \cap L = (P/(s \setminus H))^0 \cap L$

- $\forall\, s \in \mathit{traces}(P)\,.\, \forall\, h \in H . s \frown \langle h \rangle \in \mathit{traces}(P) \Rightarrow \mathit{traces}(P/s) \cap L^* = \mathit{traces}(P/s \frown \langle h \rangle) \cap L^*$

where L^* is the set of finite sequences formed from L, $s \setminus H$ is the trace s with all members of H deleted, P/s represents the process that behaves like P after it has performed the trace s, and $(P/s)^0$ is the set of events P can perform immediately after completing the trace s. The first of these (purge) says that the set of events that P offers the low-level user are exactly the same as though the high-level user had never communicated anything. The second (unwinding) says that a high-level action never changes how the process looks to the low-level user (in this formulation on the assumption that no further high-level action occurs). These definitions, as well as several alternative formulations, are equivalent.

The study of nondeterminism is inherent in concurrency, as it arises naturally from the ways in which parallel combinations of processes behave. Nondeterminism can be said to arise in a communicating system when, after a given trace, the set of options that the process offers its environment might be different on different occasions. In particular, a potentially nondeterministic system is not adequately described by its set of traces. Therefore all process algebras employ semantic devices to make more subtle distinctions than those offered by traces, and in the case of CSP this is done using *failures* (combinations (s, X) of a trace s and a set of events X that the process can refuse to communicate after s) and *divergences* (traces s on which the process can engage in an infinite unbroken sequence of invisible (τ) actions).

It is straightforward to reformulate the definitions given above into these more sophisticated models, as was done for example in [19]. Doing so, however, creates problems both of a practical and intellectual nature. The practical problem (in fact shared by the trace definitions given above) is that we end up with conditions which do not appear to be easily addressable on the automated model checkers (like CSP's FDR) which have been so successful in recent years in making process algebras usable. The intellectual problem comes from the fact that we encounter the *refinement paradox*: it is possible to have a process P which is secure and a refinement Q of it which is not. Consider for a moment the most nondeterministic divergence-free CSP process $\mathit{Chaos}_{H \cup L}$, where H and L are respectively the high and low alphabets. Most theories would deem this secure, but since every divergence-free process refines it there are evidently many insecure processes that do. While we might not worry about the abstract process $\mathit{Chaos}_{H \cup L}$, we might well worry about the equivalent process

$$\mathit{Chaos}_{H \cup L} \sqcap \mathit{LEAK}$$

where $\sqcap$ is the nondeterministic choice operator and LEAK is any insecure process such as a perfect channel from high to low. Certainly the conventional understanding of what nondeterminism means in process algebras makes it far more natural to consider this insecure than secure. The trouble is that a theory which makes one process secure must also make all equivalent ones secure! Subtly different problems appear if we consider the process $\mathit{Chaos}_L \sqcap \mathit{LEAK}$, which never communicates anything with H at all unless it chooses the leaky alternative. For extensive discussion of these issues, see [14].

It is perhaps true to say that nondeterminism creates difficulties for formulations of noninterference in either a state-machine world or that of process algebra, but that the difficulties are less escapable in the latter mainly because it comes with more fixed notions of what things like refinement and process equivalence mean.

In [16], an alternative CSP formulation of security was proposed, based explicitly on process algebraic ideas rather than being a translation of ones from elsewhere. This was to create, using abstraction operators, a process $\mathcal{A}_H(P)$ that represents what a process P looks like to a user who cannot see the events H, and to define that P is secure if $\mathcal{A}_H(P)$ is deterministic. The intention is that the abstraction procedure should turn all the choices that the high-level user makes into nondeterministic choices internal to $\mathcal{A}_H(P)$. If any nondeterminism is visible at the outside this might well be as a result of these new choices and represent a way of transmitting information from high to low. On the other had, if $\mathcal{A}_H(P)$ is deterministic, then we can be sure that nothing high does will (at least in the terms described by the semantic model being used) affect what low sees.

Two different abstraction operators are used, the choice depending on whether it is assumed that the handshaken model of communication applies to all high actions (specifically, whether these are all delayable by the user) or whether some *signal* or output actions S (such as indicator lights or information being printed on a screen) are not delayable. These are respectively *lazy abstraction* $\mathcal{L}_H(P)$ and *mixed abstraction* $\mathcal{M}_H^S(P)$. The definitions of these operators given in [16, 14], in terms of masking H actions by interleaving, evolved in [15] to the following, equivalent for all the security definitions but superior for other purposes. For a divergence free, finitely nondeterministic and nonterminating (i.e., without the event ✓) process P:

- $\mathcal{L}_H(P) = (P \underset{H}{\parallel} Chaos_H) \setminus H$
- $\mathcal{M}_H^S(P) = (P \underset{H \setminus S}{\parallel} Chaos_{H \setminus S}) \setminus H$

The effect of the hiding operator $\setminus H$ is to conceal the high actions that occur from the environment, as we are forming the view of what the process looks like without them. The role of the parallel compositions with *Chaos* is to reflect the fact that the high-level user need not offer any of the actions he or she can delay. If the high action h can be delayed, then the process $h \to l \to STOP$ might look to low either like $STOP$ or at least the availability of l might be much delayed, in either case conveying information to low about high behaviour. The above definitions are actually those from the stable failures model $\mathcal{F}$ for CSP, though the lazy abstraction of allowed P is always divergence-free and mixed abstraction is only allowed when $P \setminus (S \cap H)$ is divergence-free. We give these because they are the most practical for automation purposes; for alternatives and a discussion of the relationship between abstraction and choice of model, see [15].

These two definitions immediately give rise to definitions of noninterference: P is respectively lazily ($\mathcal{L}IND_H(P)$) or mixed independent ($\mathcal{M}IND_H^S(P)$) if the processes $\mathcal{L}_H(P)$ and $\mathcal{M}_H^S(P)$ are deterministic. Lazy independence is precisely equivalent to the purge and unwinding definitions given above when P is deterministic, but much more severe when P is nondeterministic. Our new definitions are closed under refinement, and thereby avoid the refinement paradox. What this means is that they only pass processes which *must* behave securely, and therefore fail some secure processes because there are semantically equivalent processes whose security, like that of $Chaos_{H \cup L} \sqcap LEAK$, is questionable. In other words, our conditions err on the side of caution.

It is certainly the case that in order to assess the security of nondeterministic CSP processes accurately, it is necessary to use semantic models that are significantly finer than those traditionally used for CSP. The best-known attempts at using finer models are those of Focardi and Gorrieri [1, 2] (in a modified CCS) which again have the property of coinciding with all the other definitions on deterministic processes, and use the transition system models of CCS with weak and strong bisimulation equivalences as their methods of determining process equivalence (e.g., BNDC, BSNNI and SBSNNI). Unfortunately even these conditions pass some questionable processes related to the $Chaos_L \sqcap LEAK$ example, as shown by Forster [4] who goes on to offer some stronger conditions. But the latter can again be accused of over-caution, since the standard transition system model does not contain sufficient information to tell us the source of nondeterminism. It seems certain that any model that is capable of giving a proper treatment of this subject must retain many more details about nondeterministic choices (for example the form they take and what mechanism resolves each) than we have been used to recording.

Because of this continuing uncertainty about the right formulation for nondeterministic P, and because the $\mathcal{L}IND_H(P)$ and $\mathcal{M}IND_H^S(P)$ conditions are efficiently decidable on FDR, we decided to address intransitive noninterference properties by looking for analogues of these. Since these conditions are based on a binary partition of the alphabet, a simplification which, as discussed in the introduction, fails in the case of intransitive security policies, we certainly do have some work to do.

From here on we will concentrate on formulations of conditions within the language of CSP, to make direct comparisons easier. We are, however, confident that the basic issues we raise are independent of this choice.

3 A critique of *ipurge*

The main motivating examples for intransitive noninterference are *downgrading*, in which a trusted individual is permitted to move files from high to low classification, and other similar cases in which agent A can only influence B through a highly defined route (an *assured pipeline*) such as an encrypter[2] or a certification service (one which is allowed to determine whether or not B can see the information).

In order to understand how different theories relate to this, let us examine a pair of simple example processes of this type. First, a file system with a downgrader: let's suppose we have a set of users U, and a function *clearance* from this set to a partial order of security levels $(L, \leq)$. Similarly, there is a set F of

[2] The motivation here is that a high-level agent who encrypts data effectively downgrades it, even though in most mathematical senses the information content remains the same.

filenames, and a mapping (which we are going to allow to vary) from F to L. We will assume that users may only *write* to files at their own level, and may *read* from files that are at their own level or below. A realistic operation which will be interesting later on is the *copy* operation which allows a user to copy data between files at his or her own level. In addition, there is a special interface for a trusted downgrader agent (though, in general, one could imagine there being a family of downgrading agents with different privileges) which can inspect files and decide to reclassify them to lower security levels. A simple CSP description of this process is given below:

$$
\begin{array}{l}
FS(cl, fv) \ = \\
readreq?(u,f) : \{(u,f) \mid clearance(u) \geq cl(f)\}!fv(u) \\
\quad \rightarrow FS(cl, fv) \\
\square\ write?(u,f) : \{((u,f) \mid clearance(u) = cl(f)\}?v \\
\quad \rightarrow FS(cl, fv[v/f]) \\
\square\ copy?(u, f1, f2) : \{((u, f1, f2) \mid clearance(u) = cl(f1) \\
\qquad\qquad\qquad\qquad\qquad\qquad = cl(f2)\} \\
\quad \rightarrow FS(cl, fv[fv(f1)/f2]) \\
\square\ dgread?f!fv(f) \rightarrow FS(cl, fv) \\
\square\ downgrade?f?c : \{c \mid c < cl(f)\} \rightarrow FS(cl[c/f], fv)
\end{array}
$$

Our second example is of a process that guards a channel ensuring that only suitable data gets through: it is assumed that it has some internal mechanism by which it decides, but we have implemented this by a nondeterministic choice. Data is input on channel *in*, then either sent on via channel *out* or returned to its sender if deemed unsuitable

$$
\begin{array}{ll}
GUARD = in?x \rightarrow & (out!x \rightarrow GUARD \\
 & \sqcap\ return!x \rightarrow GUARD)
\end{array}
$$

This process really only represents the intermediary; questions of noninterference will arise best once it is connected to a producer, and a consumer, of the information.

The formal definitions (e.g,[13, 21]) of intransitive noninterference in the literature that we have discovered all have their roots in that of [18] (which was itself influenced by [6, 8]) are all based on the state-machine model, and are very similar to each other. They are all based on *purge* functions in which all actions are retained that have a transitive causal link through the trace under consideration to the agent whose view is being considered. In other words, if $A \rightsquigarrow B$ and $B \rightsquigarrow C$ but $A \not\rightsquigarrow C$, then the purge (frequently termed *ipurge*) from C's perspective of the trace $\langle a, b, a \rangle$ would, with the obvious assignments of agents to actions, be $\langle a, b \rangle$ since there is a legitimate route for the first a but not the second to influence C. This, *prima facie*, seems very reasonable, as plainly the a before b may have influenced it (and hence C), and the one after has no legitimate route for influencing C. We believe that the motivation for this definition may well have been the fact that it is reasonable to specify that the resulting purge function does actually generate a trace of any reasonable process meeting the noninterference requirements. For example, consider the following trace of our downgrader process in which there are just two security levels and two users, and a file f with initial data 0 and initial security level hi.

$$
\begin{array}{l}
write.(hugh, f).1 \\
downgrade.f.lo \\
read.(lois, f).1
\end{array}
$$

This is a perfectly legitimate trace, and *ipurge*-ing it with respect to level *lo* does not change it. If, however, the purge function had removed the initial event things would have been different, as

$$
\begin{array}{l}
downgrade.f.lo \\
read.(lois, f).1
\end{array}
$$

is not a trace of the system. In other words, this *ipurge* function retains enough information fully to explain the events seen at low level in terms of the overall behaviours of the system.

But this definition also has its problems. Let us suppose that our implementation of the file system were erroneous, and the event intended to downgrade the level *hi* file f_1 to level *lo* has the effect of downgrading another f_2 as well.

Then our system would have the trace

$$
\begin{array}{l}
write.(hugh, f_2).1 \\
downgrade.f_1.lo \\
read.(lois, f_2).1
\end{array}
$$

which we would regard as generating an unwanted information flow. However, the *ipurge* function would, on the assumption that the only security domains were the security classifications (plus the downgrader) leave this trace alone and the definition of noninterference based on *ipurge* would fail to catch the leak. The problem here is that the interference that this trace causes to Lois is certainly accountable for by things Hugh did before the downgrader acted, but unfortunately not the things that were intended by the downgrader to influence him. In other words, anything the downgrader does is permitting all information which can influence it to pass on to Lois, irrespective of what the downgrader actually intended. Thus we have failed to capture the essence of what downgrading is all about.

This particular problem can be repaired by dividing the security domains into many pieces. It is simplest if we ignore two of the channels of our file system: *copy* and *dgread*. It is then necessary to have two for each file/classification pairing, one $U_{c,f}$ for the users at that level's operations on the file, and one $D_{c,f}$ for downgradings of the file *to* that level. The appropriate security policy is then that no information flow is allowed between any pair of domains associated with different files, or directly from $U_{c1,f}$ to $U_{c2,f}$ unless $c2 \geq c1$. $U_{c1,f} \rightsquigarrow D_{c2,f}$ for all $c1$ and $c2$, and $D_{c1,f} \rightsquigarrow U_{c2,f}$ just when $c2 \geq c1$. The misbehaviour discussed above then violates the *ipurge*-based security condition because it contains information flow between files.

To include the *dgread* actions (which plainly should not, in themselves, influence any ordinary user) we can introduce a further, domain (or perhaps one for each file) which can be influenced by all others (of that file) but influence no others. The *copy* action clearly generates an information flow between files: it is possible to deal with this by allowing these to provide the same sort of intransitive bridge between files as the downgrading actions do between levels. Thus we have apparently been forced to divide up the security levels into subdomains, and to have a part of the security policy that is intransitive quite apart from the pieces directly concerned with the downgrader.

Unfortunately, even the complicated security policy described above is not adequate for ensuring that a system with the given alphabet does not generate undesired information flows. Suppose that some bug in the system means that downgrading a file reverts the value of the file to the most recent back-up (and that a back-up occurs at every write).[3] Then we would have the trace

$$\begin{array}{l} write.(hugh, f).secret \\ write.(hugh, f).public \\ downgrade.f.lo \\ read.(lois, f).secret \end{array}$$

which seemingly carries undesired information flow: presumably the downgrader wanted to downgrade the contents of the file at the point of the *downgrade* action. However, because there is no information flow between files, this trace would be left alone by *ipurge* in even the refined policy, and this behaviour would be passed. This illustrates the fact that this definition of security permits *all* information contained in the trace of high-level actions prior to downgrading to be passed to low level, rather than the much more controlled information flow desired. We cannot see how the security domains could reasonably be re-worked to catch this type of flow.

[3]A more complex, but perhaps more plausible, instance of the same type of behaviour would be in a distributed file system in which files can temporarily have multiple values during an update. One can imagine a file might be downgraded during such a time, perhaps temporarily making an out-of-date and still secret value available at some points in the network.

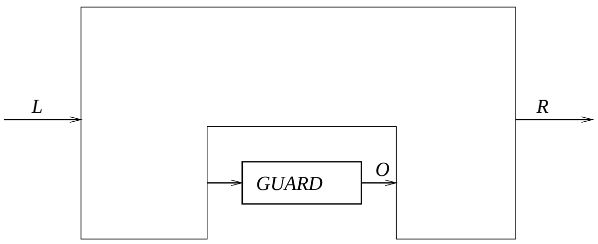

Figure 1. Network with intended assured pipeline

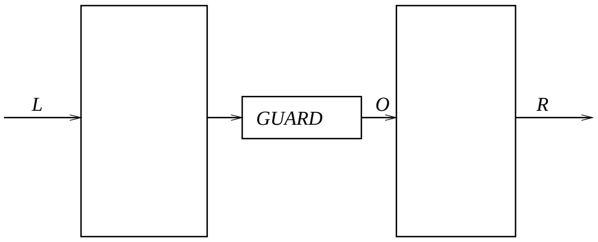

Figure 2. Assured pipeline by construction

We would argue that the *ipurge* definition makes sense as an explanation of the behaviour of a downgrading system which we already fully understand, in the sense that it represents a natural generalisation of the definition of transitive noninterference that it will satisfy. However, it has deficiencies when it comes to deciding whether the information flows contained in a process purported to be a downgrading system are limited to what we might expect.

The situation is rather similar with the other example. If it is part of a network as shown in Figure 1, we might reasonably require that no information passes from the users on the left to those on the right other than via $GUARD$. The natural thing to do is to study the relationship between the views of the system in the set L and R of events used by these two classes respectively, and the outputs O of $GUARD$. Plainly it is legitimate for L to influence O, and O to influence R, but not for L to influence R directly.

Now if the network were actually built as in Figure 2, we would have nothing to worry about: $GUARD$ has the above property by construction. So the information flow analysis is only an issue when the more complex picture of Figure 1 is at hand, and physically, at least, there are other potential channels that we want to ensure either are not there or convey only information in any case allowed by $GUARD$. The problem with the *ipurge* definition here is that again each time $GUARD$ does anything it permits all information contained in all preceding L interactions to be passed (via some other route) to R, even if only a small proportion of this is contained in the O history.

In both cases the problem is simply that the *ipurge* definition does not allow for the possibility that the intermediary may want to be selective about which of the information that has influenced it is passed on.

4 A determinism-based formulation

One of the advantages of the determinism-based formulation of information flow is that it gives an entirely different perspective on the problem. Thus, when we were thinking about how to formulate intransitive noninterference in terms of determinism we attempted to create the most natural specification from this different viewpoint though initially we expected that (as had been the case with transitive noninterference) that we would end up with a definition that would coincide for many cases. As we shall see, we end up with a rather different notion.

For simplicity, let us initially consider the case of a process P with three users A, B and C, with information allowed to flow between any pair of them except $A \not\rightsquigarrow C$. In trying to detect the potential for information flow from A to C through P there are two obvious views of the process we could consider.[4] The first is $\mathcal{L}_{A\cup B}(P)$ which is the view of C alone, and the second is $\mathcal{L}_A(P)$ in which just the actions of A, whose actions are not meant to create any nondeterminism in C's view are abstracted. The first is not really much use to us, because the act of creating the abstraction identifies nondeterminism created by A (which is forbidden) with that created by B (which is not). The second process may well be nondeterministic since we are allowing information flow from A to B, which would manifest itself in nondeterminism visible to B.

The specification that suggested itself to us, therefore, was that when we abstract away the events in A, the process has no nondeterminism visible to C:

$$\forall\, s \in \mathit{traces}(\mathcal{L}_A(P))\,.\,\forall\, c \in C\,.$$
$$\neg(s \frown \langle c\rangle \in \mathit{traces}(\mathcal{L}_A(P)) \wedge (s,\{c\}) \in \mathit{failures}(\mathcal{L}_A(P))$$

This is a straightforward generalisation of the usual definition of determinism in CSP (we might phrase it "$\mathcal{L}_A(P)$ is *locally deterministic* in C") and says that after no trace s of the abstraction might C see either the acceptance or refusal of an event c in its alphabet.

The natural generalisation of this to an arbitrary (reflexive) security policy $\mathcal{V}$ is that, for each domain C, we abstract the union of the alphabets of the processes that are not allowed to influence C, which we can define:

$$\mathit{noflow}(C) = \bigcup\{A \mid A \not\rightsquigarrow C\}$$

and insist that, in just the same sense as above, the process

$$\mathcal{L}_{\mathit{noflow}(C)}(P) \quad \text{is locally deterministic in } C.$$

If this is true (for all the users C) we will say that P *obeys the (lazy) policy* $\mathcal{V}$. Given a subset S of events of the alphabet of P that are signals, it is clear how to define a corresponding notion using mixed abstraction. However, for the rest of the present paper we will concentrate solely on lazy abstraction and omit the bracketed *lazy* from the phrase above.

The following lemmas set out some basic desirable properties of this definition.

Lemma 1 *If P obeys policy $\mathcal{V}$ and $P' \sqsupseteq P$, then P' obeys $\mathcal{V}$ also.*

This is because the abstraction operators used are monotone and if $Q' \sqsupseteq Q$ has nondeterminism visible in a subset X of its alphabet, so does Q.

Lemma 2 *If $\mathcal{V}$ is transitive, then P obeys $\mathcal{V}$ if and only if it does so in the alternative definition we can infer in the way described in the introduction (in terms of multiple binary partitions of the alphabet) using concept of lazy independence discussed in Section 2.*

Lemma 3 *If the policies $\mathcal{V}$ and $\mathcal{V}'$ are based on the same partition of P's alphabet into domains, and $A \rightsquigarrow B$ in $\mathcal{V}$ implies $A \rightsquigarrow B$ in $\mathcal{V}'$, then if P obeys $\mathcal{V}$ it obeys $\mathcal{V}'$ as well.*

Evidently much of the underlying intuition in our definition is similar to that discussed for independence in Section 2. In particular we must expect to get sometimes over-pessimistic assessments in the case where the original process P has some nondeterminism visible to a low-level user.

An obvious question that arises is: what does our definition mean in terms of trace-sets when applied to a deterministic process P? Returning to the three user case, nondeterminism of $\mathcal{L}_A(P)$ can occur precisely

[4] The particular abstraction operator used is not the issue: we have used lazy abstraction here because it is, as discussed earlier, the most appropriate in a pure CSP context, but if the ways in which the abstracted user(s) interact with P included signal/output events, it would again be right to use mixed abstraction.

when there are (necessarily different) traces s and s' of P, and $c \in C$, such that

$$\begin{array}{ll} s \setminus A = s' \setminus A & \text{and} \\ s \frown \langle c \rangle \in \mathit{traces}(P) & \text{and} \\ (s', \{c\}) \in \mathit{failures}(P) & \end{array}$$

This easily leads to the following alternative characterisation of our property.

Theorem 1 *Suppose P is deterministic and $\mathcal{V}$ is a security policy. Then P obeys $\mathcal{V}$ if and only if, for all domains C and $s, s' \in$ traces(P),*

$$\begin{array}{l} s \setminus noflow(C) = s' \setminus noflow(C) \\ \Rightarrow (P/s)^0 \cap C = (P/s')^0 \cap C \end{array}$$

This new characterisation has obvious similarities with the purge style discussed earlier. As in the purge-based trace definition in Section 2, we are stating that the choices C is offered by P are independent of any events that have occurred in the part of the alphabet that is not meant to pass information to C. Unlike the earlier definition, we are not demanding that the 'purged trace' $s \setminus noflow(C)$ is a trace of P. This is, of course, closely related to the discussion at the beginning of Section 3 where we showed that a simple definition of purge would not, in the case of an intransitive policy, produce a trace. What has happened is that the intuition of determinism has led us back to this simple purge function and produce ways of specifying noninterference that do not rely on $purge(s)$ being a trace.

For the simple reason that the equivalence relation on traces induced by *ipurge* is (often strictly) finer than than produced by $s \setminus noflow(C)$, the above theorem allows us to deduce the following result.

Theorem 2 *Suppose P is deterministic and $\mathcal{V}$ is a security policy. Then if P obeys $\mathcal{V}$ in our new sense, it satisfies the ipurge definition of noninterference with respect to $\mathcal{V}$.*

(The assumption of P being deterministic is not in fact necessary here.) In other words, our new definition is less tolerant than the old one. As we shall soon see, sometimes it is strictly less tolerant.

It is interesting to consider how the examples discussed in Section 3 behave under this new definition. Let us first consider the case of the downgrader with just two users Hugh and Lois, and again we start by considering the situation where there are three domains for the security policy, the two users plus the downgrader, and the events *copy* and *dgread* are removed from the definition. The result is the opposite of last time: instead of a situation where leaky version of the system satisfies the property, we now find that even the apparently reasonable implementation given at the start of Section 3 does not. For we have the two traces

$$\begin{array}{l} write.(hugh, f).0 \\ downgrade.f.lo \end{array}$$

and

$$\begin{array}{l} write.(hugh, f).1 \\ downgrade.f.lo \end{array}$$

which are the same once the high-level actions ($noflow(Lois)$) are deleted, but after which Lois obviously gets different events offered by the system when she attempts to read f. What has happened is that the downgrading action does not contain enough information to explain the value that Lois can now see: we actually seem to need some of Hugh's history of interaction (as retained by *ipurge*) to explain this. But as we saw in the last section, this can allow the passing of far too much information, so what is the solution?

The most satisfactory answer we have found comes by looking more carefully at the definition of the downgrader. As presently defined, the agent controlling the *downgrade* actions can perform them without having seen what he or she is downgrading, and even if we assume that the file has always been read first via *dgread*, there is nothing in our definition to stop Hugh from writing a secret into the file between these two actions. From the point of view of responsibility and attribution, we argue that the definition of the downgrader would be greatly improved by adding a further field to the *downgrade* channel, namely the contents of the file at the point of downgrading. By doing so we would have created a "log" which identifies precisely what information has been downgraded. And by doing so we immediately remove the above problem, since the pair of traces above now become ones which are still different after Hugh's actions are removed. We have created a system in which it makes sense to claim that it is only the actions of the downgrader that affect Lois, not some actions of Hugh predicated upon intermediate actions of the downgrader. We believe that being forced to consider the tighter definition of noninterference produced a clearer and better description of the downgrader.

If there are more than two security levels (or, more precisely, more than one level to which it is possible to downgrade a file), it is still necessary to divide the *downgrade* events into multiple domains This is because we need to be able to determine from the security policy alone what users are to be influenced by a given $downgrade.f.c.v$ (noting the extra field v); unless c is the lowest classification this will not be them all. There is, however, no need to divide either the user or downgrader domains by filename as well (as we experimented with earlier). The security domains for our downgrader are then two for each level c: U_c containing all the events of the users with that classification, and D_c containing all event of the form $downgrade.f.c.v$ for the given c. The security policy is that $U_c \rightsquigarrow U_d$ if and only if $c \leq d$, $U_c \rightsquigarrow D_d$ for all c and d, and $D_d \rightsquigarrow U_c$ if and only if $d \leq c$. The modified definition of the downgrader obeys this policy, and if we introduced any of the potential errors into it discussed earlier that contain security flaws then the result would not obey it.

There is no problem in incorporating the *copy* events into this scheme, as they no longer cross the borderlines of domains. The *dgread* channel, if we wish to include it in the security policy, should be handled as before: putting it into a separate top domain which is influenced by all others but influences none other than itself. It is arguably better however, for a channel like this along which we are happy for any information to flow and which has a peripheral role in the analysis anyway, simply to abstract it away before we form the security policy. If, by doing so, nondeterminism were introduced at lower domains (which does not occur with the present example), it would provide the same warning of potential information flow resulting from use of such a channel that including it in the policy generates.

With our other example, the *GUARD* process, it would normally be appropriate to make the outputs of this process play the intermediary (downgrader-like) role in the security policy, probably abstracting away its inputs if these are not overall inputs to the system. Our new definition of noninterference then says unequivocally that the information flow to the downstream side must be accountable for purely in terms of what *GUARD* chooses to output. Again we regard this as more satisfactory than the definition that allows anything which has occurred prior to a *GUARD* output to be transmitted in a way that bypasses this component process.

5 Automation

The key to making the determinism characterisation of (transitive) noninterference practically useful has been the existence of efficient decision procedures for the determinism of a finite-state CSP process. Aside from normalising (see, for example, [15]) the complete process value, and then inspecting the result, which in many cases is likely to be relatively slow, we are aware of two different algorithms for this. The more elegant (also described in [15]) exploits the fact that the deterministic processes are the maximal elements of the failures/divergences model of CSP under the refinement order. The algorithm involves extracting an arbitrary deterministic refinement P' of our target process P: P is then deterministic if and only if P refines P'. The alternative method, developed by Lazić [10] for the purpose of extending ideas of data independence to determinism checking, involves running two copies of P in parallel with each other in such a way that the first is always allowed to choose any communication it likes, and only then is the second offered it (and only it). The process is deterministic just when this second offer is never refused, and this is something that can easily be checked by a refinement check.

The unconventional parallel composition used in the last of these methods is achieved by renaming each event a of one (say the second) copy of P to a distinct "shadow" event a' (which is different both from all events of P and all other b'), and linking the two by the process

$$TEST = ?a : A \rightarrow a' \rightarrow TEST$$

where A is the alphabet of P. Because we are combining two copies of P in parallel, this method is, in the worst case, quadratic in the state-space size of P, though it is unlikely to be this bad in practice. In the type of example we are considering, where P is produced by applying abstraction operators to a (usually) deterministic process, there are often substantial advantages to be had in applying state-compression operators to P prior to this type of check.[5]

For the generalised concept of noninterference introduced in this paper we need to be able to check for *local* determinism in a set of events C, so it is natural to ask how the above methods extend to it. The one which involves checking over the normal form is little different: the main effort is in the normalisation (unchanged from the previous case), but the check that has to be done on each normal form node is slightly different. We have to discover if any node has both c as an initial event and $\{c\}$ as a refusal set for any $c \in C$. The method based on the maximality of deterministic processes does not work, as locally deterministic processes are not necessarily maximal, and we cannot see a reasonable fix[6]

[5]Compressing an abstracted process can often itself be more efficient by abstracting all events as early as possible as the process is built up and using the compression operators hierarchically.

[6]In fact, the formulation of lazy abstraction given earlier also causes problems with this algorithm in checking ordinary determinism thanks to the way it handles divergence. In that case, however, there is a convenient fix, but not one which is valid

This leaves the method in which P is run in parallel with itself. This is easy to amend: all we have to do is change the renaming and $TEST$ process a little. Shadow events are only created for events in C, so the second copy P' of P has all its other events left unchanged by the renaming. The $TEST$ process allows P and P' to synchronise as they please on these other events, except that after P has performed an event in C it insists that P' does its shadow and nothing else: if B are the events of P less the set C,

$$\begin{aligned} TEST &= ?x : B \rightarrow TEST \\ &\quad \Box ?c : C \rightarrow c' \rightarrow TEST \end{aligned}$$

The resultant combination (in which all three processes must synchronise on events in B, P and $TEST$ on C, and $TEST$ and P' on C') can deadlock immediately after an event in C if and only if P is not locally deterministic in C.

We conclude that the generalised noninterference condition is decidable in the same spirit as the original determinism-based one, but what was, in some ways, the most attractive option is not now available to us. We have performed a number of experiments with the parallel method described above for the type of example process discussed in this paper, but it is obviously desirable that larger case-studies, where the results are less easily predictable, are carried out.

It is also possible to decide the *ipurge* definition of noninterference in the context of CSP and FDR, by carefully combining a renamed version of the target process with a monitor process whose role, essentially, is to decide which events need to be abstracted. This, in fact, uses only ordinary, as opposed to local, determinism checking, but is complex to formulate for security policies with many failures of transitivity.

6 Conclusions

It was a great surprise to us to discover how the previously published definitions of intransitive noninterference behaved when we considered them in the context of our examples. We assume that these definitions were driven somewhat by the desire to have traditional-looking definitions based on the purge function and on unwinding, but our conclusion is that they are worryingly liberal in the sense that they fail to catch potential security leaks. (The *ipurge* definition does, as several of the references cited at the start of Section 3 demonstrate, yield useful definitions of unwinding.)

for local determinism. That is to use a specific implementation of $\mathcal{L}_A(P)$, namely the one in which H behaves as $STOP$, as a reference.

The advantage of our new definition, which as can be seen from the trace reformulation given in Theorem 1 is in some way closer to those of ordinary noninterference, is that it clearly establishes the principle that the only permitted influences on the low-level process are by the intermediary (even though the latter may well have been influenced in what it said by the high-level process). We do not have to find some way of formulating which high-level information the intermediary intended through or make the bold assumption (seemingly made by the various authors who have used *ipurge*) that it was all to be allowed. As we saw, the new definition forced us to create what was really a better downgrading system in which the blame for any (managerially) undesired information flow is directly attributable to something the downgrader does. We expect that further case study work will help to determine whether the rigours of satisfying our stronger formulation of noninterference are generally helpful or turn out to be unreasonably onerous. Faced with a choice of two conditions, one of which is demonstrably too weak and the other arguably too strong, it is surely better to aim to satisfy the latter rather than being happy with the former.

As stated earlier, in this work we have concentrated on the deterministic P case of systems described in CSP. Theorem 1 easily points to how analogous definitions could be formulated in state-machine models or other paradigms with similar notations of traces, but it is not quite so easy to see how the decision procedures we have developed for CSP would carry over to the usual frameworks these use. We imagine that work currently under way to help understand ordinary noninterference in the context of nondeterministic CSP should extend to the intransitive context we have set out. However, as discussed earlier, the right answer for how to handle this issue is still uncertain.

The CSP characterisations given in [15] of (transitive) noninterference and fault tolerance are closely related to each other. This is because a system is obviously fault tolerant if the occurrence of faults (which we can code as being triggered by specific events) has no effect on what the user sees. The definition, in fact, is that the system with fault events lazily abstracted refines the one in which the faults are prevented (by parallel composition with $STOP$ synchronising on the fault events) from occurring. While this definition is intuitively appealing, its use is limited to circumstances under which the system is sufficiently robust that the user never sees any negative effect from faults. One can imagine wanting to analyse systems in which faults can affect the user's view, but hopefully only in a controlled way. We believe that the same ideas used here to reason about intransitive noninterference may provide a means by which such systems can be analysed. The fault actions are again analogous to the high-level actions of MLS, and the user's view is again analogous to the low-level ones. But this time we might have a fault recovery system or operating system playing a downgrader-like role, and want to show that the only effects the user can see resulting from faults are those s/he is warned about by this intermediary. This will be a topic for future research.

Acknowledgements

We would like to thank Sylvan Pinsky and Peter Ryan for advice and discussions about intransitive noninterference.

The work reported in this paper was funded by DERA Malvern and the US Office of Naval Research.

References

[1] R. Focardi, Comparing two information flow security policies, Proceedings of CSFW IX, IEEE Computer Society Press, 1996.

[2] R. Focardi and R. Gorrieri, A classification of security policies for process algebras, Journal of Computer Security, **3**, 1, pp5-33, 1994.

[3] Formal Systems (Europe) Ltd. *FDR2 manual*, 1998. `http://www.formal.demon.co.uk/fdr2manual/`.

[4] R. Forster, Non-interference properties for nondeterministic processes, Dissertation for transfer to D.Phil status, Oxford University Computing Laboratory 1997.

[5] J.A. Goguen and J. Meseguer. *Security policies and security models*, in Proceedings of the 1982 IEEE Symposium on Security and Privacy, pp 1 1-20. IEEE Computer Society Press, 1982.

[6] J.A. Goguen and J. Meseguer. *Inference control and unwinding*, in Proceedings of the 1984 IEEE Symposium on Security and Privacy, pp 75-86. IEEE Computer Society Press, 1984.

[7] J. Graham-Cumming, *The formal development of secure systems*, Oxford University D.Phil Thesis, 1992.

[8] J.Haigh and W. Young, Extending the non-inference model of MLS for SAT, in Proceedings of the 1986 IEEE Symposium on Security and Privacy, pp 232-239, IEEE Computer Society Press, 1986.

[9] J.L. Jacob, Specifying Security Properties, in *Developments in Concurrency and Communication*, C.A.R. Hoare (ed.) Addison-Wesley 1990.

[10] R.S. Lazić and A.W. Roscoe. Verifying determinism of concurrent systems which use unbounded arrays. Technical report, Oxford University Computing Laboratory, April 1998. Full version of [11].

[11] R.S. Lazić and A.W. Roscoe. Verifying Determinism of Data Independent Systems with Labellings, Arrays and Constants. In *Proceedings of Infinity'98*, 1998. To appear; full version issued as [10].

[12] J. MacLean Proving noninterference and functional correctness using traces, Journal of Computer Security **1**: 37-57 (1992).

[13] S. Pinsky, Absorbing covers and intransitive noninterference, Proceedings of 1995 IEEE Symposium on Security and Privacy.

[14] A.W. Roscoe, CSP and determinism in security modelling, Proceedings of 1995 IEEE Symposium on Security and Privacy.

[15] A.W. Roscoe. *The Theory and Practice of Concurrency.* Prentice Hall, 1998. ISBN 0-13-6774409-5, pp. xv+565.

[16] A.W. Roscoe, J.C.P. Woodcock, and L. Wulf. Non-interference through determinism. *Journal of Computer Security*, 4(1), 1996. Revised from Proceedings of the European Symposium on Research in Computer Security (ESORICS) 1994, LNCS 875.

[17] A.W. Roscoe and L. Wulf, Composing and decomposing processes under security properties, Proceedings of CSFW VIII, IEEE Computer Society Press, 1995.

[18] John Rushby. Noninterference, transitivity, and channel-control security policies. Technical Report csl-92-2, SRI, 1992.

[19] P.Y.A. Ryan, A CSP formulation of non-interference, Cipher, pp 19-27. IEEE Computer Society Press, 1991.

[20] Lars Wulf. *Interaction and Security in Distributed Computing.* DPhil, Wolfson College, University of Oxford, Hilary 1997.

[21] W. Young and W. Bevier, A state-based approach to non-interference, Proceedings of CSFW VII, IEEE Computer Society Press, 1994.