CSP and determinism in security modelling

A.W. Roscoe
Oxford University Computing Laboratory
Wolfson Building
Parks Road
Oxford OX1 3QD
United Kingdom

Abstract

We show how a variety of confidentiality properties can be expressed in terms of the abstraction mechanisms that CSP provides. We argue that determinism of the abstracted low-security viewpoint provides the best type of property. By changing the form of abstraction mechanism we are able to model different assumptions about how systems behave, including handling the distinction between input and output actions. A detailed analysis of the nature of nondeterminism shows why certain security properties have had the paradoxical property of not being preserved by refinement – a disadvantage not shared by the determinism-based conditions. Finally we give an efficient algorithm for testing the determinism properties on a model-checker.

1 Introduction

The type of problem that concerns us relates to *confidentiality* issues: how can we specify and verify that a system which interacts with more than one user will not allow information to leak from one to another. CSP is an excellent vehicle for this type of reasoning because it is a calculus of how systems interact, typically over series of communications.

We will argue that the most satisfactory definitions of security are based on the notion of *determinism*: a system's interface with a low-class user must not have any nondeterminism that can be resolved by a high-class user. CSP is therefore doubly well-suited to our purpose, since it treats the study of nondeterminism as fundamental.

The relevance of CSP ([7, 5], for example) to non-interference properties has been recognised before (a brief survey will be given in a later section), and perhaps a majority of authors defining notions of information flow and non-interference have produced definitions close in spirit to the previous paragraph. But I am not aware of any previous work which connects non-interference properties with the well-understood standard representation of determinism in CSP. We will see that not only does the combination give an intellectually satisfying theory, but also provides the key to efficient automated checking.

Let us suppose the process that is intended to be secure is P, and that we have identified (disjoint) subsets H and L which partition its alphabet, such that a user interacting with P in L must not gain any information about the interactions of P in H. The obvious model for this is where H represents the communications of a user with high security clearance, and L are those of a lower-grade user, but any partially-ordered hierarchy of non-interference could be established by using this property over a variety of decompositions of the alphabet. For example, if A, B, C and D represent the alphabets of users such that A's communications must be secure from the other three, B's from C and D, and C's from B and D, then we could use, successively, $(H, L) = (A, B \cup C \cup D)$, $(A \cup B, C \cup D)$, $(A \cup C, B \cup D)$ and $(A \cup B \cup C, D)$.

For clarity, we will henceforward assume the system has two users: U_L and U_H, interacting with P in L and H respectively.

The rest of this paper is structured as followed. We will consider first a number of properties expressed without explicit reference to determinism: these (in common with several other authors) give security specifications that are based directly on the underlying semantic models (traces, and failures/divergences).

We will then show how the notion of determinism provides an alternative characterisation of the security properties, and argue that it does itself provide a natural and nearly model-independent notion of security. Next we see how to allow, in one's choice of security property, for the nature of interactions between the system and its users; in particular the distinction between inputs and outputs. If one is prepared to use the more advanced model of CSP that includes infinite traces, it turns out to be possible to characterise the determinism properties as security with respect to appropriately chosen 'most nondeterministic' user in an interesting way.

After gaining an understanding through this work that determinism seems to capture what is required for security properties, there is next a section which examines why this is in some detail: looking closely at the nature of nondeterminism, refinement and semantic models of concurrency. We will in particular examine why other security properties have had the paradoxical property of not being closed under refinement (a property our determinism-based specifications *do* have).

There is also a section comparing our work with related work in the literature and, finally, there is a conclusions section discussing how it might be extended to encompass such things as details of timing and priority.

Several of the trace- and failure-based properties we introduce during this paper are equivalent to properties proposed by other authors. Hopefully these are signalled in the comparisons section, but I have deliberately not pointed out these similarities during the text since the main object there is to set up both a uniform way of describing the properties and a systematic notation.

The proofs of all theorems and several other detailed items have been omitted from this version of the paper because of space limitations. It is hoped that a fuller version will appear elsewhere.

2 Trace and failures/divergences specifications

All previous authors (see Section 7) specifying security properties via CSP have used specifications based directly on the semantic models used to represent CSP processes: either traces or failures/divergences. I will ultimately argue that it is better to address confidentiality via the determinism of an abstraction of the system; but in order to see why and develop concepts I will first introduce some properties of the first sort.

The *traces* of a process are the sequences of actions it can communicate. Usually, as in this section, these are restricted to *finite* traces, though later in this paper we will also use *infinite* traces which represent a communication history through all time.

At the this level, there is a simple assertion about a process which seems to capture much of what we require. In essence, it states that if two traces of P differ only in their high-class (H) actions, then the subsequent behaviour of P as seen in L is identical after these two traces. We will say that P is *eagerly trace-invariant* with respect to L, or $\mathcal{E}trINV_L(P)$ if and only if

$$\begin{aligned} & tr, tr' \in tracesP \wedge tr \upharpoonright L = tr' \upharpoonright L \\ & \quad \Rightarrow (P/tr) \setminus H =_T (P/tr') \setminus H \end{aligned}$$

(Here, $=_T$ denotes trace-equality. Details of this, and other CSP equivalences can be found in an appendix.)

The use of the term 'eagerly' here will be explained at length later on: it derives from the semantics of the CSP hiding operator, which is interpreted as though the internalised actions are done eagerly.

'Invariant' is used to describe this property because what it says is that the view in L does not change when an H action occurs.

The condition $\mathcal{E}trINV_L$ can be paraphrased as saying that the behaviour on any trace ignoring H-actions, is equivalent once the observation of subsequent H-actions is prevented. While hiding the actions of H is one way of preventing their observation, an interesting way of camouflaging them is provided by interleaving[1]: if P is any process then $P \interleave RUN_H$ (where $RUN_H = ?x : H \rightarrow RUN_H$) is one that has any behaviour of P with the arbitrary insertion of elements of H. There is no way of telling whether an element of H came from P or from RUN_H, but whatever happens this combination can never refuse an element of H. We will say that P is *lazily trace-invariant* with respect to H, or $\mathcal{L}trINV_L(P)$ if and only if

$$\begin{aligned} & tr, tr' \in tracesP \wedge tr \upharpoonright L = tr' \upharpoonright L \\ & \quad \Rightarrow (P/tr) \interleave RUN_H =_T (P/tr') \interleave RUN_H \end{aligned}$$

This condition only differs from the former one in the fact that H communications are made ambiguous (and thereby disguised, or camouflaged) rather than concealed. As the name of this condition suggests, the conceptual difference is that this condition's view is that concealed actions need not happen immediately: they can be delayed. In this case, though this will not be true as we develop our model past traces, the lazy condition is stronger than the eager one. The implication comes from the fact that, in general,

$$(P \interleave RUN_A) \setminus A =_T P \setminus A$$

so the interleaving style of equality implies the hiding one.

The way this condition is stronger is that it, unlike the earlier one, does not permit the set of available L-

[1] I will argue later that the hiding $P \setminus H$ and interleaving $P \interleave RUN_H$ are alternative ways of *abstracting* a set H. They will correspond to a different view of how H-actions then occur.

actions (the ones that are possible initial communications) to change when an H-action is communicated.[2]

The traces model gives a very weak notion of equivalence for CSP processes, since it does not distinguish between a process that can always perform a trace and one that might also (nondeterministically) refuse to. The standard model for untimed CSP is therefore a richer one, the *failures/divergences* model (see Appendix). If we simply replace traces equivalence with failures/divergence equivalence in the security properties we get somewhat stronger security conditions $\mathcal{E}fdINV_L$ and $\mathcal{L}fdINV_L$. These are discussed in detail in the full version of this paper and in [16], but broadly speaking they are equivalent to the trace conditions in most cases where the latter are reasonable.

3 Examples

We now define some example processes that will be used to demonstrate and judge various properties we propose, including $\mathcal{E}trINV_L$ and $\mathcal{L}trINV_L$. In each of them we will take $H = \{a, b, c, d\}$ and $L = \{w, x, y, z\}$.

1. $P_1 = a \to x \to P_1 \,\square\, b \to y \to P_1$ is a process we would certainly not regard as secure: which element of L occurs depends directly on which element of H immediately preceded it.

2. $P_2 = a \to x \to P_2 \,\square\, b \to x \to P_2$ does not have this problem: the element of L which occurs is clearly independent of the H action. It does have the problem that the very fact that x is possible shows that some H action has occurred. Whether this constitutes a breach of security depends on what sort of communications a and b are, and how the high-class user interacts with the process.

3. $P_3 = a \to x \to P_3 \,\square\, b \to x \to x \to P_3$ shows this up more sharply, since the number of x's we can do depends on which of a and b has occurred. A similar, but more extreme example is

4. $P_4 = a \to P_4 \,\square\, b \to x \to P_4$, since the existence of an x implies that a b has occurred.

5. We would expect to allow communications in L to influence those in H, so the process

$$\begin{aligned} P_5 = \; & x \to (a \to P_5 \,\square\, x \to P_5 \,\square\, y \to P_5) \\ & \square\, y \to (b \to P_5 \,\square\, x \to P_5 \,\square\, y \to P_5) \end{aligned}$$

should be considered secure. So far as U_L is concerned, this process is one that is always prepared to communicate either x or y.

6. Finally we have a process modelling one which deals with either high-level or low-level enquiries, giving appropriate responses.

$$\begin{aligned} P_6 \;\; = \;\; & w \to y \to P_6 \\ & \square\, x \to z \to P_6 \\ & \square\, a \to c \to P_6 \\ & \square\, b \to d \to P_6 \end{aligned}$$

Of these processes, only P_1 fails $\mathcal{E}trINV_L$, this being because the traces $\langle a\rangle$ and $\langle b\rangle$ satisfy the preconditions of the condition, but $(P_1/\langle a\rangle) \setminus H =_T x \to RUN_{\{x,y\}}$, while $(P_1/\langle b\rangle) \setminus H =_T y \to RUN_{\{x,y\}}$.

Each of P_2, P_3 and P_4 has the property that $(P_i/tr) \setminus H = RUN_{\{x\}}$ for any trace tr at all. Thus each of them trivially has the property $\mathcal{E}trINV_L$ even though they can all be argued to breach our idea of security in some way. The breaches in P_2 and P_3 are somewhat subtle and depend somewhat on our view of communications. That in P_4 is not subtle at all, and should make us very careful of the weaker trace condition. The fundamental problem in P_4 is that, by communicating a's, U_H is able to postpone the availability of x indefinitely but never make it impossible.

These three processes all fail the lazy condition $\mathcal{L}trINV_L$, because in each case the set of available L actions can vary with an H-action.

P_5, on the other hand, satisfies this condition: for any trace tr, we have $P_5/tr \;|||\; RUN_H = RUN_{H\cup\{x,y\}}$.

P_6 satisfies $\mathcal{E}trINV_L$ and fails $\mathcal{L}trINV_L$. The reason for this failure is that, after the communication of one of the H-events $\{a, b\}$, the L-events are not possible until after the communication of the c or d as appropriate. Whether this really constitutes a breach of security depends on the nature of the communications $\{c, d\}$. If these are things that the high-security user can delay indefinitely, then U_L can detect that U_H has communicated something by his own inability to communicate with P. If, on the other hand, $\{c, d\}$ take the form of output signals to U_H that cannot be delayed (the output corresponding to the immediately preceding input, perhaps), then it is harsh to count this as a security breach.

One can argue that in this case the lazy trace condition is a little too strong and that we ought to be counting the output/signal H-events as ones to be dealt with by hiding in the manner of $\mathcal{E}trINV_L$, while the rest are dealt with as in $\mathcal{L}trINV_L$. We will ultimately propose something very similar to this, but in terms of pure traces this is not adequate because of the problems associated with P_4 above.

[2]That this is allowed for $\mathcal{E}trINV_L$ is illustrated by several examples below.

4 Determinism

The use of a specific semantic model (such as traces or failures/divergences) in the way we have defined properties so far makes strong assumptions about what observations an intruder can make. It is certainly possible to imagine a user who could take into account either of the following:

- The failures/divergences model does not distinguish between processes that could be observed distinct by a user who could record what events happen *after* given refusals, as well as before.
- Nor does it distinguish between processes which have the same ranges of nondeterministic behaviours, but with very different probability distributions.

The difficulty of modelling these things, especially the probabilistic one, leads me to the conclusion that it is better, wherever possible, to restrict our sort of security analysis to deterministic processes P, or at least to processes whose appearance to U_L is deterministic. There is a detailed examination of both of these problems (choice of semantic model, and the nature of nondeterminism) in Section 8.

The appealing feature of this characterisation via determinism is that, intuitively, the only way information can leak from U_H to U_L is via the process behaving differently towards U_L depending on what U_H has done. This will appear exactly as though U_H is resolving some nondeterminism in how the process behaves towards U_L. The most convincing way of ensuring that this nondeterminism never gets resolved is to show that there is none there to resolve: in other words to make the process appear completely deterministic to U_L.

This view is re-inforced when we observe that, thanks to the following result, determinism always implies the trace- or failure- based conditions and, in the case of a deterministic P, is usually equivalent.

Theorem 1

1. If $P \setminus H$ is deterministic, then P satisfies $\mathcal{E}trINV_L$ and $\mathcal{E}fdINV_L$.
2. If $P \mid\mid\mid RUN_H$ is deterministic, then P satisfies $\mathcal{L}trINV_L$ and $\mathcal{L}fdINV_L$
3. If P is deterministic, $P \setminus H$ is divergence-free and P satisfies $\mathcal{E}trINV_L$, then $P \setminus H$ is deterministic.
4. If P is deterministic and P satisfies $\mathcal{L}trINV_L$, then $P \mid\mid\mid RUN_H$ is deterministic. ∎

This tells us that, for deterministic P, the trace-based conditions are sufficient. Furthermore, it shows that what these conditions really tell us is that, under alternative interpretations of how to abstract away H communications, the L interface is deterministic and therefore cannot have been influenced by H. Last but not least, it gives us efficient ways of deciding the conditions. (We will see in an appendix that checking determinism is a mechanically tractable operation.)

Following the conventions of earlier definitions, we give names to these conditions as follows:

- $\mathcal{E}IND_L(P)$ holds if $P \setminus H$ is deterministic. We say that P is *eagerly independent* with respect to L.
- $\mathcal{L}IND_L(P)$ holds if $P \mid\mid\mid RUN_H$ is deterministic. In this case P is *lazily independent* with respect to L.

As we have indicated before, and will discuss in detail in the next section, both of these conditions are based on ways of abstracting away the behaviour of U_H. A third one[3], which is interesting in the context of this view of security through determinism, is to build a model of the most nondeterministic possible U_H, and to show that the view of U_L is deterministic even when the process is given this extreme U_H. The standard properties of CSP refinement then show this view will remain deterministic for any U_H. This condition, which we will term $\mathcal{S}IND_L$, or *strongly independent* is defined:

- $(P \,[\![H]\!]\, CHAOS_H) \setminus H$ is deterministic.

(Here, $CHAOS_A = STOP \sqcap ?x : A \rightarrow CHAOS_A$ is the most nondeterministic divergence-free process which communicates in A.)

This condition has been called the *strong* independence condition because it is equivalent to the conjunction of the other two.

Theorem 2 A process satisfies $\mathcal{S}IND_L$ if, and only if, it satisfies each of $\mathcal{E}IND_L$ and $\mathcal{L}IND_L$. ∎

To some extent we can say this condition is not lazy like the interleaving condition, because it does care about the possibility of infinite sequences of H-actions, and is not eager because the user may at any time refuse the whole of H (which, semantically, has the effect of showing up the refusals of P when it can still do H-actions).

While we could have defined a condition for general failures/divergences using this form of abstraction, emulating $\mathcal{E}fdINV_L$ and $\mathcal{L}fdINV_L$, it would not

[3] This was suggested by Paul Gardiner.

have had the same elegant relationship with the other two. To some extent this is because of the limitations on reasoning about processes which appear nondeterministic to U_L that were discussed at the start of this section (specifically, the one relating to what may be observed after refusal).

We will later develop this theme that a security condition contains an implicit model of the most nondeterministic conceivable U_H.

5 Inputs and outputs

In the context of the usual CSP interpretation of what communication means, it is clear to me that the interleaving-based lazy security property is what we want. That interpretation is that all events are things which both the process and environment must agree on; they cannot happen until this agreement is reached.

We are seeking to specify that no information about what the high-security user does, or does not do, can leak to the low-security one. In the context of a high security user whose co-operation is required for an event to take place, we should not make the assumption that this co-operation will be forthcoming. This implies that the set of actions accepted (and subsequent behaviours) should be the same before any H-action as before any of whatever choice of H-action is eventually made, if such a choice is made before any L-action is communicated. This is very much what the interleaving conditions say. In summary, the interleaving properties do not assume that high-security actions take place so quickly that no refusals can be observed by U_L between them.

But that is exactly what the hiding conditions do. They only record refusals when all high-security actions are exhausted. The usual CSP argument for this set of refusals when *abstracting* away a set A of events is that a user interfacing in the complementary set cannot be sure that what he or she sees refused will continue to be refused for ever until all actions in A have been exhausted (for subsequent A-actions might change the offered set). This makes the crucial assumption that abstracted (i.e., hidden) actions happen as soon as they become available, or at least within some bound. We can thus describe CSP hiding as a sort of *eager* abstraction, while our interleaving construct $P \mathbin{|||} RUN_A$ effectively abstracts from the same set A *lazily*.

In fact, of course, a user might interact with a system in more ways than synchronisation requiring both parties' agreement. In particular there may be many communications which are effectively unresistable by the user: for example the appearance of information on a VDU screen. We will call these "signal" communications (under which name they have been considered in the real-time concurrency world), but in most cases they will take the form of output communications from a system to the external environment.

Where an output communication takes place the instant it becomes available, clearly no refusal is observable before it occurs. In this case there is a strong argument for taking the view provided by the hiding condition: so far as the low security user is concerned, the (guaranteed) transient occurrence of a high-security output event is indistinguishable from the ordinary internal actions of the process which occur as the result of previous events. It may therefore be thought unreasonable to insist that the system must be able to perform L-events before this action (or perhaps as extensive a range of them as after it). For example, if the events $\{c, d\}$ in example P_6 are signals it is unreasonable to call this process insecure: at all times when refusal is observable, what is refused is entirely uninfluenced by the history of H-actions, as is the set of available L-actions. (Notice that if z is not a signal then the H-refusals are affected by the history of L-actions, but that does not concern us as we are not worried about information-flow in this direction.)

My assertion, then, is that we should divide the set H into two parts: D, the events on which U_H can delay the system (those which are like usual CSP actions) and S, the signals.[4] In many cases these will be, respectively, the inputs and outputs. We should then deal with the D-events using the interleaving model of abstraction, and the S-events using hiding.

This generates the following hybrid, or *mixed* conditions:

- $\mathcal{M}trINV_L^{(D,S)}$ holds of P if

$$\begin{array}{l} tr, tr' \in tracesP \wedge tr \restriction L = tr' \restriction L \Rightarrow \\ ((P/tr) \setminus S) \mathbin{|||} RUN_D =_T ((P/tr') \setminus S) \mathbin{|||} RUN_D \end{array}$$

- $\mathcal{M}IND_L^{(D,S)}(P)$ holds if $(P \setminus S) \mathbin{|||} RUN_D$ is deterministic.

The reason why there is now a superscript alphabet component as well as the usual subscript is because the abstraction is treating elements of H (the complement of the usual subscript L) in different ways.

The order of the two abstractions is immaterial since if D and S are disjoint we have

$$(P \setminus S) \mathbin{|||} RUN_D = (P \mathbin{|||} RUN_D) \setminus S$$

[4] Wherever we use D and S in this way it will be assumed that they partition H, just as H and L partition Σ.

at all levels of CSP equivalence. P_6 satisfies all of these with $D = \{a, b\}$ and $S = \{c, d\}$.

These conditions are related to each other and those previously described as follows:

Theorem 3

- If P satisfies $\mathcal{M}IND_L^{(D,S)}$ then it satisfies $\mathcal{M}trINV_L^{(D,S)}$.
- If P is deterministic and $P \setminus S$ is divergence-free then it satisfies $\mathcal{M}IND_L^{(D,S)}$ if it satisfies $\mathcal{M}trINV_L^{(D,S)}$. ∎

I believe that the determinism based *independence* condition is, for essentially the reasons set out in the last section, the most persuasive and satisfying of the mixed conditions. This result shows that in the context of a deterministic P and lack of infinite sequences of signal events, the mixed conditions are, as was the case for the other two levels, equivalent.

It is clear that the possibility of introducing divergence when we hide a set of events in CSP has a considerable impact on our conditions whenever they involve hiding. Basically, except for pure trace conditions, we are regarding as dangerous any situation where there is an infinite unbroken sequence of hidden (by the condition) high-security events. The reasons for this are discussed in the full version of this paper.

6 Abstract models of U_H

When we introduced the condition $\mathcal{S}IND_L$ it was motivated with the idea that the $CHAOS_H$ process represented the most nondeterministic conceivable U_H. Since it represents the widest range of behaviours the real U_H might display, the determinism of the system interacting with $CHAOS_H$ implies its appearance to U_L will be independent of whatever the real U_H might choose.

All three of our other determinism properties can be expressed in the same form: "$(P \,[\![\, H \,]\!]\, U) \setminus H$ is deterministic" for suitably chosen processes U. In the case of the hiding condition it is trivial: $U = RUN_H$ because $P \,[\![\, H \,]\!]\, RUN_H = P$ in CSP for all P and H. The others are a good deal more subtle: this is clear from the fact that the lazy condition does not forbid infinite sequences of H-actions, even though they are to be hidden. The solution is to be found in a model of CSP we have not mentioned hitherto: the infinite traces model where each process is identified with a triple (F, D, I) where F and D are the failures and divergences as before, and I is the set of all infinite traces the process can perform. This model, introduced in [12], is required when we want to reason about infinitely nondeterministic CSP processes. It is, for example, required to model properly the hiding of infinite sets.

What is important for us about the infinite traces model is that when divergence is introduced by hiding this is now inferred from an actual infinite sequence of hidden actions rather than because all finite approximations to the infinite sequence are possible. There are now processes that have all finite prefixes of an infinite trace u but not u itself: all that is required that however far we get down u it is possible for the process to refuse some future element of u as well as accept it. The most extreme example of this is the process $FINITE_A$ which has the same failures (all with traces from A^*) and divergences (none) as $CHAOS_A$, but while $CHAOS_A$ has every infinite trace of A (i.e., A^ω), $FINITE_A$ has none. Another way of describing $FINITE_A$ is $\sqcap\{Q_n \mid n \in \mathbf{N}\}$ where

$$Q_0 = STOP \qquad \text{and} \qquad Q_{n+1} = a : A \to Q_n$$

(this uses the infinite nondeterminism operator $\sqcap$ which itself requires the more refined model).

$FINITE_A$ is the 'user' process U we need for the lazy condition: it does not permit the occurrence of the infinite sequences of H-events that allows divergence in $\mathcal{E}IND_L$ and $\mathcal{S}IND_L$. The following result has a proof almost identical to that of Theorem 2: the only difference is that divergence is excluded by different means.

Theorem 4 P satisfies $\mathcal{L}IND_L$ if, and only if, $(P \,[\![\, H \,]\!]\, FINITE_H) \setminus H$ is deterministic. ∎

We should perhaps remark here that the definition of determinism is not changed by the introduction of infinite traces: it is still decided by finite traces and failures. The infinite traces of a deterministic process are completely determined by its failures: they are the obvious closure of the process' finite traces. Thus it is not possible to remove any infinite trace from RUN_H, for example (which is a deterministic process).

The corresponding process for the mixed condition $\mathcal{M}IND_L^{(D,S)}$ is, as one might expect, $RUN_S \,|||\, FINITE_D$. The result establishing this is the following.

Theorem 5 P satisfies $\mathcal{M}IND_L^{(D,S)}$ if, and only if, $(P \,[\![\, D \cup S \,]\!]\, (RUN_S \,|||\, FINITE_D)) \setminus (D \cup S)$ is deterministic. ∎

These various 'users' suggest a more general approach to security specification: for a particular context, choose a process U which characterises all possible behaviours of U_H under which it is expected that confidentiality will be maintained. Usually this will be all its behaviours, but it is possible to imagine other circumstances, for example if the system P represents a mail system where it is allowable for a high-security user to send a message to a low-security one, we might expect to maintain confidentiality so long as no such messages are sent. (This type of property is known as *conditional* non-interference.) The general approach is as follows:

- The finite traces of U will be the elements of H^* which, provided U_H has communicated within this set, no information must leak to U_L.
- The refusals after a given trace will be the set of those events that U_H can delay. (Usually these will not vary with the trace.)
- The infinite traces will be those which U_H might allow where there is a danger of thereby preempting events available to U_L (as discussed in the previous section).
- U is always divergence-free.

There are as many possibilities here as there are processes. One example is given by the following, which defines the process specifying that there are as in the case of $\mathcal{M}IND_L^{(D,S)}$ a set of delayable events D and a set of signals S, but which assumes that U_H will not attempt an event from D until at least one S signal has appeared since any previous member of D.

$$
\begin{aligned}
U &= R \,[\![\, D \,]\!]\, FINITE_D, \quad \text{where} \\
R &= x : D \to y : S \to R \\
&\quad \Box\, y : S \to R
\end{aligned}
$$

If a process P has $(P[\![D \cup S]\!]U) \setminus (D \cup S)$ deterministic, then it guarantees not to transmit information to U_L provided U_H keeps to the regime set out above.[5]

The properties of CSP refinement easily imply that the more nondeterministic the process U is, the stronger a security condition it yields. In particular, $CHAOS_H$ gives the strongest property since this is the most nondeterministic divergence-free process.

It is interesting how we have characterised security properties using a combination of unbounded nondeterminism and determinism: a process is considered secure if it is capable of factoring out all the nondeterminism contained in a process like $FINITE_H$, and leaving a deterministic result.

[5] One can imagine that this and similar approaches can begin to bring in explicit timing information, with the events from S denoting the ticks of a clock. This particular condition would then say that no information leaks provided U_H does not exceed a particular speed of interaction.

7 Comparisons with other work

A number of authors have proposed methods of specifying this type of security property, both in CSP and otherwise. We will deal with these separately.

7.1 CSP

Authors using CSP to analyse non-interference have tended to use either intuitive rationale or traces only, rather than using the failures/divergences model and its ability to analyse determinism. An excellent survey can be found in Graham-Cumming's thesis [5].

Some of the proposed conditions have been essentially the same as $\mathcal{L}trINV$, others have been equivalent to $\mathcal{E}trINV$, and others different to both.

The definitions of non-interference given by Allen [1], Graham-Cumming [5] and Ryan [17] are all essentially equivalent to $\mathcal{L}trINV$, the latter also encompassing $\mathcal{L}fdINV$. Graham-Cumming demonstrates that these three are essentially equivalent, and it is not at all hard to show that Allen's definition (here altered to reflect our notation and conventions):

$$
\begin{aligned}
&t \in tracesP \Rightarrow \\
&t \setminus L \in tracesP \wedge (P/t)^0 \cap L = (P/(t \setminus L))^0 \cap L
\end{aligned}
$$

(where P^0 denotes the initial events of P) is equivalent to ours. Paraphrased, it says that the events available to U_L at any time are exactly the same as they would have been if U_H had never communicated. This corresponds closely to the intuition of interleaving $|||$ used in the definition of $\mathcal{L}trINV$, but perhaps the easiest way to see that the definitions are the same is to observe that $\mathcal{L}trINV_L$ holds of P if and only if $\mathcal{L}IND_L$ holds of the (unique) deterministic process P_{det} with the same traces. Thanks to the characterisation of $\mathcal{L}IND_L$ given in Section 6, this holds if and only if $(P_{det} \,[\![\, H \,]\!]\, FINITE_H) \setminus H$ is deterministic. This, in turn, implies that $(P_{det} \,[\![\, H \,]\!]\, Q) \setminus H = (P_{det} [\![H]\!] FINITE_H) \setminus H$ whenever $Q \sqsupseteq FINITE_H$. If t is any trace of P, two allowable Q's are $STOP$ and the process $Q(t \setminus H)$ which communicates its trace argument and then stops. The fact that

$$(P_{det} \,[\![\, H \,]\!]\, STOP)) \setminus H = (P_{det} \,[\![\, H \,]\!]\, Q(t \setminus H)) \setminus H$$

which we can derive from $\mathcal{L}trINV$ simply says that the behaviour of P_{det} visible to U_L when U_H communicates any trace is the same as though U_H did nothing at all. The reverse argument (that the Allen condition implies ours) is an inductive proof going through many of the same arguments as those we made for earlier results.

McCullough [9] gives a definition that is essentially equivalent to our definition $\mathcal{L}trINV$:

$$t \in traces P \Rightarrow$$
$$\forall h \in H.(P/t) \setminus H =_T (P/t\hat{}\langle h\rangle) \setminus H$$

Jacob [7] gives a definition of non-interference which, though cast in different notation, specifies that

$$P \, [\![H]\!] \, STOP =_T P \setminus H$$

In other words, any trace that U_L can observe is possible when U_H does nothing at all. To some extent these conditions can be thought of as even lazier than our lazy conditions. They are problematic even in the context of traces. Suppose H and L each consist of one (obviously named) channel, and

$$\begin{array}{rcl} LEAKY & = & low?x \rightarrow LEAKY \\ & & \Box \; high?x \rightarrow low!x \rightarrow LEAK \\ \text{where} & & \\ LEAK & = & high?x \rightarrow low!x \rightarrow LEAK \end{array}$$

This satisfies the above condition, even though we can guarantee that whatever U_H communicates will be transmitted direct to U_L! One might argue that U_L will not know that what he is seeing is U_H's behaviour rather than random noise produced by the initial state (though entropy tests, etc., seem to provide a rebuttal to this). But as soon as we start to consider refusals, the process is obviously insecure: as soon as U_L sees any of his own communications refused, he knows that U_H has communicated and that everything he sees subsequently will be a copy of U_H's actions!

CSP-based conditions have sometimes been thought to be impractical to verify. The development of the model-checker FDR[6], and the determinism algorithm presented in the second appendix of this paper, show emphatically that this not true any more.

7.2 Other approaches

Once again a fuller survey can be found in [5].

Non-interference properties have always been expressed in terms of some sort of labelled transition system (LTS), whether the deterministic automata of Goguen and Meseguer [4], the formulations in CSP (which can be viewed as a specially designed notation for building and reasoning about LTS's) or LTS's directly as in Johnson and Thayer [8].

I should perhaps remark that, since any LTS can be interpreted in the same semantic models as CSP, all of the conditions developed in this paper can be adapted to these other worlds: give us an LTS and the conditions can be defined and tested directly.

[6]FDR is a product of Formal Systems (Europe) Ltd.

Many formal notations, such as Z, have been used which do not express communicating behaviour as naturally as CSP. The definitions of systems have therefore tended to be direct specifications of the transition functions of the overall state machines. This has resulted in very heavy use being made of so-called *unwinding theorems* which allow non-interference to be inducted from behaviour over single actions of an LTS.

Johnson and Thayer [8] give a non-interference definition over LTS's which is very close to our definition $\mathcal{L}fdINV$: it is constructed in terms of *testing* equivalences, a process-algebraic theory very closely related to failures equivalence.

8 The arguments for determinism

I have argued that determinism of a low-security viewpoint provides the natural specification of non-interference properties. In this section I will try to provide more insight into this by showing that this restriction effectively circumvents two specific imprecisions that are introduced into many models of concurrency. These relate to the precise nature of nondeterminism, and to the way nondeterminism is modelled. They are discussed in the following subsections. Both were alluded to earlier: in this section we provide more detailed analysis.

8.1 The nature of nondeterminism

I believe that all of the CSP security properties I have quoted (both mine and due to others) that were not based on determinism suffer from the *refinement paradox*: it is possible to have a process P that is considered secure, but with refinements that are insecure. That the determinism-based conditions *are* closed under refinement is an elementary consequence of the fact that deterministic processes are maximal under the refinement order, and each condition takes the form:

- $C[P]$ is deterministic

for a context $C[\cdot]$ that is monotone (like all other CSP constructs) under the refinement order. It would appear to me to require a high order of sophistry to argue that a process which is allowed to behave as though it were insecure can be secure.

To back up this argument, consider the case where there are two channels: *high* and *low*, and that the process $LEAK$ is defined as in the last section. If our system is constructed $LEAK \sqcap CHAOS_\Sigma$ where the internal choice is viewed operationally as something that gets resolved immediately (perhaps with a probability biased towards $LEAK$), then even though the system is equivalent to $CHAOS_\Sigma$ and would be declared secure by most conditions other than the determinism ones[7], it does not seem sensible to accept a proc ess built operationally like this one as secure. Clearly $LEAK$ refines this process and is anything but secure.

The first, subtler, and I think more important of the two imprecisions, since it will apply to more models of computation than just models of concurrency, relates to the nature of nondeterminism. Most models, both of CSP and more generally, identify two different types of nondeterminism which can be summarised as *probabilistic* and *under-specification*. I think that the refinement paradox is one manifestation of this confusion.

Probabilistic nondeterminism is perhaps the easier sort to comprehend: it arises when a process is genuinely random in its behaviour, and exhibits patterns of behaviour according to the laws of probability. A process exhibiting probabilistically nondeterministic behaviour can in some sense be relied on to behave in particular ways, for example controlled by the (strong and weak) laws of large numbers in probability theory. For a full model of such a system, we need to know the distributions of behaviour, though only a few (and very complex) models of concurrency do this successfully. Most models (including all of the usually-seen models of CSP) simply take the set of all possible outcomes without giving the distributions, thereby preventing us from making any inference, in the models, about the relative likelihood of different outcomes.

Under-specification, or *don't-care* nondeterminism is most often how people building models of concurrency understand by nondeterminism. Saying that a process is nondeterministic in this sense does not imply that it genuinely must exhibit all the behaviours shown in its model, merely that every behaviour it does exhibit is taken from the set. It is entirely permissible to implement the process $P \sqcap Q$ by Q. It is valid to implement this sort of process by any implementation that always keeps its patterns of behaviour within those that the process allows. Such a process might be

- Deterministic: always selecting the same answer to the same stimulus.
- Probabilistically nondeterministic, provided that the range of choice is within the overall allowed.
- Having nondeterminism which is resolved by agents, or at a modelling level, that we do not model. For example, modelling a system at the level of detail provided by real-time can often exclude behaviours which cannot be excluded in untimed semantics, or show a deterministic dependence of the outcome on the precise time when some earlier action happened. Similarly, it is often only possible to get a satisfactory probabilistic model in a real-time context, since the distribution of outcomes might be heavily influenced by the precise times when earlier actions have occurred. Thus, if modelling these systems in an untimed context, we are forced to present a simple range of behaviours without being able to give a reliable distribution of their likelihoods. We are not given sufficient information to deduce a precise distribution.

In identifying refinement with the reduction of nondeterminism, it is clear that we are thinking of nondeterminism of this type. After all, if we were to take a coin that gives heads and tails with equal probability, we would be unlikely to accept a 'refinement' that always gives a head. Indeed, a process which only exhibits probabilistic nondeterminism should be thought of as fully refined. There are no satisfactory treatments (as far as I am aware) of the combination of these forms of nondeterminism: this is a fascinating research problem.

Let us try to understand how the two sorts of nondeterminism fit into the security picture. Information can leak from high to low security only if the appearance of the process in alphabet L differs depending on what has, or has not, been communicated in H. If the process viewed from L appears deterministic, we have seen that no information can be transmitted in how H-behaviour resolves nondeterminism. In a reasonable model, the nondeterminism that is introduced by abstracting away the H behaviour is not probabilistic, for it would be against the spirit of what we are doing to assume a distribution of how the high security user will behave. We should (as was done, for example in the failures-based properties) consider the different appearance in L depending on an arbitrary pair of H behaviours. Let B_1 and B_2 be the appearance in L given h_1 and h_2 respectively. If we are modelling nondeterministism without probability (i.e., recording only the range of possible behaviour, rather than their distribution) then there are four possibilities:

- B_1 and B_2 are identical and deterministic. In this case L is certain to see the same behaviour given h_1 or h_2, so L cannot distinguish them
- B_1 and B_2 are identical and nondeterministic. In

[7] Those that fail are those where hiding H in $CHAOS_\Sigma$ causes divergence.

this case there is certainly no concrete evidence of insecurity, since we cannot tell that the behaviour is different depending on h_1 and h_2, but nor can we tell that it is not, since the actual behaviour of the process might be (i) resolved to specific different behaviours by h_1 and h_2 by some mechanism at a finer level than we are modelling (as might happen in the abstraction between timed and untimed CSP, for example). Even if all of the behaviours represented in $B_1 = B_2$ remain possible given h_1 and h_2, (ii) probability distributions may exist and be different (even though we are not modelling them), thereby allowing discrimination between h_1 and h_2 up to some (and in some cases any) confidence level. As an example, suppose B_1 and B_2 were respectively $C_{a,b}(p)$ and $C_{a,b}(q)$, where $0 \leq p < q \leq 1$ and

$$C_{a,b}(r) = a \to C_{a,b}(r) \sqcap_r b \to C_{a,b}(r)$$

and $\sqcap_r$ is a probabilistic choice operator that chooses its left- and right-hand argument with probabilities $1 - r$ and r respectively. However close p and q are, watching the ratio of as and bs for long enough gives an arbitrarily (below certainty) discrimination between B_1 and B_2 and thereby between h_1 and h_2.

In this case we therefore have no evidence from our modelling either of information leakage or the lack of it.

- B_1 and B_2 are different but have some common refinement. Unless we have some way of knowing that all of the behaviours represented within B_1 and B_2 are actually possible within the implementation, this case is actually essentially the same as the last one: it might be the case that the way(s) B_1 and B_2 actually appear are indistinguishable, or they may be different. The only difference with the previous case is that one or other process might exhibit a behaviour which we can tell at this level of modelling is not possible for the other.
- B_1 and B_2 have no common refinement (as will, for example, be the case where they are deterministic and different). In this case the behaviour after h_1 and h_2 is certainly different and (depending perhaps on what is communicated in L), it is possible to distinguish them.

If, on the other hand, nondeterminism is being modelled purely probabilistically, in the sense that B_1 and B_2 are both precise distributions of behaviour, then it is clear that no distinction can be made between h_1 and h_2 if these distributions are the same. If they are different then some (perhaps Bayesian or probabilistic) inference can be made.

The element of uncertainty introduced by non-probabilistic nondeterminism has thus meant that there are three possible conclusions about the security or otherwise of a system: secure, insecure or don't know. I think that people have devised properties that imputed security, but were not closed under refinement, by assuming that a process' behaviour was accurately described by the range of nondeterministic behaviour it could take (implicitly assuming that all such behaviours would arise with no usable measure of probability, or perhaps equal probability). As we have already stated, a completely determined probabilistic process cannot be refined, so the refinement paradox, where a refinement of a secure process fails to be secure, arises out of the confusion that non-probabilistic, underspecified, nondeterminism that can be refined with the other sort.

The only sense in which such conditions could reasonably be asserted to establish security is on the assumption that an inquisitive agent uses exactly the same mathematical model to analyse it as we do. In other words, the most complete picture that agent can have of the process is the description of the system in our mathematical model, and draws no further inference by, for example, watching how nondeterminism is resolved in practice[8]. Taking the view, as we do, that the agent might well have, or be able to collect, knowledge of the system at a more detailed level than our model, the most we can say about the security of a system which looks nondeterministic to the low security user is 'don't know'; refinement could resolve this don't know either to 'secure' or 'insecure'.

Assuming for a moment that we are accepting the possibility of an agent who only models the system at our level, then it is not the refinement of a non-probabilistically nondeterministic *system* which can create an insecurity (for the system was certainly allowed to behave the same way before it was refined as after) but the refinement of the agent's *knowledge* of the system that comes from our rather particular view of how he is gaining information from it, that creates the refinement paradox. In other words, we can refine

[8]This latter view must be almost untenable, given that in many cases the *only* knowledge that will be drawn of a system is what can be deduced from observing it. Faced with a process which happens to resolve the nondeterminism in *CHAOS* as *LEAK* (as defined earlier), it seems hard to believe an agent could not make the inference he was seeing something interesting, even if he knew nothing *a priori* about the system.

the system as long as we can keep this refinement as secret as we have to keep the inner workings of the system in order to keep this view of security even remotely sustainable.

Of course the virtue of specifying that the interface to a low-security user should be deterministic is that all these difficulties, philosophical and otherwise, are entirely removed. Assuming the low-security user is only making the sort of observations we are modelling, then no matter what he knows about how the system is built, and therefore how its choices affect what he sees, he can make no inference about the high-security interactions.

And because a deterministic process is already maximal under the refinement order, security properties of this sort will invariably be closed under refinement.

8.2 How is the system observed?

Semantic models always make implicit assumptions about what sort of things are interesting about a process' behaviour. There have been a remarkable number of models of concurrent behaviour, making subtly different assumptions about how processes are observed and thereby compared. This creates a further problem when we come to consider the problems of computer security, since by using a specific model we are implicitly assuming that our inquisitive agent is using only the type of observations of the system that our particular model contains. In the last section we were essentially worrying that he might have sufficient knowledge of how our system works that he can infer things from the resolution of nondeterminism which is visible at this level of observation. We should equally worry that the agent might be using a different sort of observation and gaining information that way.

Here, the idea of using determinism comes to our rescue yet again. For, with one major exception (the divide between timed and untimed models) all the different models of concurrency that treat nondeterminism at all (essentially all those more sophisticated than the traces model) collapse at the deterministic processes. In other words, a process will be considered deterministic in one if and only if it is considered deterministic in them all, and a pair of deterministic processes will be identified in one of these models if and only if they are identified in them all. This says that if our criterion for security is that the low-security interface is deterministic, then (apart from the timing question) judging determinism in one model (for example failures/divergences) implies that an observer who uses the observations of another semantic model (e.g., refusal testing or bisimulation) will be no more able to gain information.

To see what might happen outside this area of agreement, consider the following pairs of processes (which might be the behaviour visible to the low-security user after high security behaviours h_1 and h_2).

$$\begin{array}{rcl} B_1 & = & a \rightarrow (b \rightarrow STOP \sqcap c \rightarrow STOP) \\ B_2 & = & (a \rightarrow b \rightarrow STOP) \sqcap (a \rightarrow c \rightarrow STOP) \\ \\ RT_1 & = & STOP \sqcap (a \rightarrow STOP \,\square\, b \rightarrow STOP) \\ RT_2 & = & STOP \sqcap (a \rightarrow STOP \sqcap b \rightarrow STOP) \end{array}$$

All of these processes are nondeterministic. Neither the pair $\{B_1, B_2\}$ nor $\{RT_1, RT_2\}$ are distinguishable in the failures/divergences model, so any security condition which expected the pairs to be the same in that model would be satisfied. If an observer, or indeed anything else, has the ability to make copies of the system in mid-flow (perhaps by generating a core dump) then one should believe B_1 and B_2 inequivalent (this corresponds to *bisimulation* equivalence). For if the two systems are copied after a is communicated, the two versions of B_2 are guaranteed to agree on the next communications (for the choice between b and c must already have been made) whereas the two versions of B_1 may not agree (for the choice may not have been made when the copies are created). For RT_1 and RT_2, suppose (much less speculatively!) that the observer chooses to record what he sees after a set of actions is refused by the process (note that failures do no do this). When observing RT_1, once the refusal of the set $\{a\}$ has occurred, it is not possible for the process to communicate b. This is not the case with RT_2, making these processes observably different. This latter model is known as *refusal testing*.

9 Conclusions and prospects

It seems that both from the point of view of what is intuitively right and from that of automation, confidentiality conditions based on determinism are the most satisfactory.

The determinism conditions, unlike the others, have the important property that they are preserved under refinement: if P is secure and $P \sqsubseteq P'$ then P' is secure. This is a much more satisfactory position than the reverse, since the usual interpretation of refinement (in CSP, at least) is that if P' refines P then one can never tell, when looking at P', that one is not looking at P. We saw in Section 8 why other conditions confuse types of nondeterminism and give rise to the refinement paradox.

The examples seen in this paper have all been extremely simple, for straightforward examples are easy to define and bring out the various properties clearly.

Given the automatability of the decision process, our conditions are capable of being applied to a wide variety of systems of a much larger size. Indeed we have already applied a prototype determinism checker to check security properties of communications protocols and a file system[15]. The fact that the algorithms are so closely related to those of FDR means that the security condition checking will be able to take advantage of features such as state-machine compression when these are implemented for FDR.

What our work really captures is how to characterise notions of independence in CSP. Other potential applications are in fault tolerance: showing that error actions do not affect ordinary behaviour, and feature interaction: showing that a pair of services added to (for example) a telephone exchange do not interact.

Since the initial version of this paper was written, the methods developed in it have been applied to a variety of applications by the author and others. One, [15], studies how our techniques may be applied to state-based systems specified in Z, using a file-system as a case-study. Others include encryption protocols, ATM security, and railway signalling systems. The relationships between our conditions and those of others (especially *separability* – the factorisation of a process into two independent parallel parts) have been further investigated in [16], which also proves various results about the compositional behaviour of the independence properties set out in this paper.

Acknowledgements

Jim Woodcock instigated this work by pointing out the relationship between non-interference and determinism: he suggested the property I have called $\mathcal{E}IND_L$ to me and all my work has derived from that. I am grateful to Lars Wulf, Paul Gardiner, John Graham-Cumming, Gavin Lowe, Peter Ryan and Michael Goldsmith for their comments. Michael also produced for me an impressive prototype implementation of my determinism-checking algorithm, something which has brought these methods excitingly to life.

10 Appendix: CSP reference

We give here a summary of the main features of CSP used in this paper, in particular the definitions of the operators which we made significant use of. A general reference to CSP is [6], and further theoretical background can be found in [2, 13, 12], for example.

Three models for untimed CSP have been mentioned in this paper: Traces, Failures/Divergences and Infinite Traces. What they have in common is that they each provide an abstract way of representing a process as the set of behaviours it can have in one or more categories. The models are all *compositional*, in the sense that the value in a model of any compound process can be computed from the values of its parts, and thereby give rise to *denotational semantics* for the language. In each case it is possible to prove that the value computed by the semantics is the same as we would expect from making direct observations of the operational semantics of CSP.

10.1 Traces Model

All the models are based on the set Σ of all communications a process might make. In general, Σ may be finite or infinite.

A *trace* is a communication history of a process: the sequence of all events it has communicated by some point. In the Traces model we will only consider finite traces, but later we will see that infinite traces also perform a useful role.

The Traces model was the first abstract model of CSP to be defined, and represents each process as a subset P of Σ^*, the set of finite sequences from Σ. P must satisfy the following pair of simple axioms to be in $\mathcal{T}_\Sigma$, the Traces model over Σ.

($T1$) P is non-empty.

($T2$) P is prefix-closed: i.e., if $s\hat{\ }t$ (the concatenation of s and t) is in P, then so is s. We say that s is a *prefix* of w if there is t such that $s\hat{\ }t = w$; this is abbreviated $s \leq w$.

Each CSP operator other than recursion reduces to a pointwise operator over the constituent traces of a process, so for example:

- $P \setminus A = \{s \setminus A \mid s \in P\}$, where $s \setminus A = s \upharpoonright (\Sigma - A)$.
- $P \mathbin{[\![A]\!]} Q = \bigcup\{s \mathbin{[\![A]\!]} t \mid s \in P \wedge t \in Q \wedge s \upharpoonright A = t \upharpoonright A\}$, where

$$\begin{array}{rcl}
s \mathbin{[\![A]\!]} t & = & t \mathbin{[\![A]\!]} s \\
s \mathbin{[\![A]\!]} \langle\rangle & = & \{s\} \\
 & & \text{provided } s \upharpoonright A = \langle\rangle \\
\langle a\rangle\hat{\ }s \mathbin{[\![A]\!]} \langle a\rangle\hat{\ }t & = & \{\langle a\rangle\hat{\ }w \mid w \in \{s \mathbin{[\![A]\!]} t\} \\
 & & \text{if } a \in A \\
\langle a\rangle\hat{\ }s \mathbin{[\![A]\!]} \langle b\rangle\hat{\ }t & = & \{\langle b\rangle\hat{\ }w \mid w \in \{\langle a\rangle\hat{\ }s \mathbin{[\![A]\!]} t\} \\
 & & \text{if } a \in A \text{ and } b \notin A \\
\langle b\rangle\hat{\ }s \mathbin{[\![A]\!]} \langle c\rangle\hat{\ }t & = & \{\langle c\rangle\hat{\ }w \mid w \in \{\langle b\rangle\hat{\ }s \mathbin{[\![A]\!]} t\} \\
 & & \cup\{\langle b\rangle\hat{\ }w \mid w \in s \mathbin{[\![A]\!]} \langle c\rangle\hat{\ }t\} \\
 & & \text{if } b, c \notin A
\end{array}$$

The interface parallel operator $[\![A]\!]$ is a generalisation of the parallel operators found in [6]. Its two process arguments are run in parallel and are allowed to communicate freely, except that all communications in the interface set must be agreed. It can express both the 'alphabetised' parallel and the pure interleaving operators as follows: if the alphabets of P and Q are A and B respectively, then (provided P and Q cannot communicate outside their alphabets)

$$P \llbracket A \mid B \rrbracket Q = P \llbracket A \cap B \rrbracket Q$$

and interleaving is achieved by synchronising on nothing:

$$P \mid\mid\mid Q = P \llbracket \emptyset \rrbracket Q$$

Over this model (as for both the others), we say that one process P *refines* another Q if $P \subseteq Q$. In the case of the Traces model this order is the opposite of the one used to compute fixed points. This is because the Traces model is largely only able to specify *safety* properties: ones which say the process will never communicate incorrectly, so that the more active a process is the less safety properties it will satisfy.

In the body of this paper, *tracesP* means the representation of P in the traces model. It is worth noting that this set is always equal to the set of finite traces predicted by the two other models *provided* the process P is divergence-free. The obfuscation required past divergence by the other two means that it does not usually hold for divergent processes (such as $P_4 \setminus H$, where P_4 is as defined in the text).

10.2 Failures/Divergences Model

The most standard model for (untimed) CSP is the *failures/divergences* model, where each process is represented by two sets of behaviours:

- *failures* are pairs (s, X) where s is a finite trace of the process and X is a set of events it can refuse after s;
- *divergences* are finite traces on which the process can perform an infinite sequence of consecutive internal actions.

For various technical reasons it is more difficult to model accurately the behaviour of a process after the possibility of divergence than on traces where no such possibility has arisen. Therefore, after each minimal (under the prefix order) divergence trace, the model identifies a process with $\perp$, the most nondeterministic process. The clearest way to understand accurately what the model represents is that

- the failures on non-divergent traces are precisely those (s, X) where the process can, after s, come into a *stable* state (one with no internal progress possible) that has no transition in X;
- the minimal divergences are the actual least traces after which the process can perform an infinite sequence of τ's (internal actions);
- all failures on divergent traces, and non-minimal divergences, are present because of our decision not to model this type of behaviour: the fact that they are there carries no information (positive or negative) about what the process can actually do. It is a deliberate obfuscation, introduced to get the theory to work better.

The *failures/divergences model* $\mathcal{N}_\Sigma$ over a given alphabet Σ of communications is the set of all pairs (F, D), $F \subseteq \Sigma^* \times \mathcal{P}(\Sigma)$, $D \subseteq \Sigma^*$ satisfying

$$\begin{array}{ll}
(F1) & (F \neq \emptyset) \wedge ((s\hat{\ }t, \emptyset) \in F \Rightarrow \\
 & \qquad (s, \emptyset) \in F) \\
(F2) & (t, X) \in F \wedge Y \subseteq X \Rightarrow (t, Y) \in F \\
(F3) & ((t, X) \in F \wedge (t\hat{\ }\langle a \rangle, \emptyset)) \notin F \Rightarrow \\
 & \qquad (t, X \cup \{a\}) \in F \\
(D1) & s \in D \Rightarrow s\hat{\ }t \in D \\
(D2) & s \in D \Rightarrow (s\hat{\ }t, X) \in F.
\end{array}$$

These are straightforward healthiness conditions setting out what a reasonable processmust look like. For example $(F3)$ says that if a process can refuse the set of events X then it can additionally refuse any events which it can never perform after the current trace t.

The process $P = (F, D)$ *refines* $P' = (F', D')$, written $P' \sqsubseteq P$, if and only if

$$F \subseteq F' \qquad \text{and} \qquad D \subseteq D'\ .$$

which means that every behaviour of P is one of P': we cannot tell, if looking at P, that we are not looking at P'.

The semantics of CSP over this model may be found in many places, including [3, 6]. Shortly we will give the semantics of the operators that have been most important in this paper. Some important structural properties are summarised below.

1. The least process under the refinement order, $\perp$, equates to any process that can diverge immediately (i.e., without performing any visible actions first). The refinement order, under the restrictions given, is complete provided Σ is finite (ways of dealing with the case when Σ is infinite are discussed in [13]), and its maximal elements are the *deterministic* processes: divergence free and, after each trace s, only able to refuse those events that it cannot communicate after s.

2. Every divergence-free process is the nondeterministic composition (componentwise union) of its deterministic refinements.

3. Each standard CSP operator can be defined as an operator over $\mathcal{N}_\Sigma$; each is monotonic with respect to the refinement order; using that order and least fixed points for the semantics of recursion gives a denotational semantics that is congruent to a natural operational semantics.

The failures and divergences of a CSP process P are respectively denotes $\mathit{Failures}(P)$ and $\mathit{Divs}(P)$.

The semantics of hiding and parallel now become:

$$\begin{array}{rcl} \mathit{Divs}(P \setminus A) & = & \{(s \setminus A)\hat{~}t \mid s \in \mathit{Divs}(P)\} \\ & & \cup\{s \mid \{t \mid (t, \emptyset) \in \mathit{Failures}(P) \\ & & \quad \wedge t \setminus A \leq s\} \text{ is infinite}\} \\ \mathit{Failures}(P \setminus A) & = & \{(s, X) \mid s \in \mathit{Divs}(P \setminus A) \wedge X \subseteq \Sigma\} \\ & & \cup\{(s \setminus A, X) \mid \\ & & \quad (s, X \cup A) \in \mathit{Failures}(P)\} \end{array}$$

$$\begin{array}{rcl} \mathit{Divs}(P \mathbin{[\![A]\!]} Q) & = & \{v\hat{~}w \mid \exists\, s, t.(s, \emptyset) \in \mathit{Failures}(P) \\ & & \quad \wedge(t, \emptyset) \in \mathit{Failures}(Q)\wedge \\ & & \quad (s \in \mathit{Divs}(P) \vee t \in \mathit{Divs}(Q))\wedge \\ & & \quad v \in s \mathbin{[\![A]\!]} t\} \\ \mathit{Failures}(P \mathbin{[\![A]\!]} Q) & = & \{(s, X) \mid s \in \mathit{Divs}(P \mathbin{[\![A]\!]} Q) \wedge X \subseteq \Sigma\} \\ & & \cup\{(v, (X \cap Y) \cup (X \cap A) \cup (Y \cap A)) \\ & & \mid \exists\, s, t.(s, X) \in \mathit{Failures}(P)\wedge \\ & & (t, Y) \in \mathit{Failures}(Q) \wedge v \in s \mathbin{[\![A]\!]} t\} \end{array}$$

Both these definitions contain specific 'closure' components to preserve the catastrophic divergence conditions *D1, D2*. The 'ordinary' cases of how to construct a refusal set are both interesting: in the case of hiding, a refusal of P only relates to a refusal of $P \setminus A$ when P can refuse the whole of A, for if an element of A is accepted we assume P moves on immediately and does not have time to refuse anything[9]. In the case of $P \mathbin{[\![A]\!]} Q$, an event outside A can only be refused if both P and Q refuse it, for either is allowed to communicate it independently. An event in A, on the other hand, requires the co-operation of both sides and so either can propagate a refusal. The most interesting thing about the above definitions is how divergence is introduced by hiding: a process that communicates an infinite sequence of hidden events can diverge, but there is no direct representation of infinite traces here so we have to *infer* the existence of an infinite trace from the finite ones.

The accuracy of the failures/divergences model is confined to just when this inference is valid. Provided we make sure that the operational branching structure of processes is only finitely nondeterministic, namely that a given process can only have finitely many different results from a single action (whether visible or invisible), then everything is alright. However a number of operators one might desire, including the hiding operator $P \setminus A$ when A is infinite, do not preserve this property and so require a more sophisticated model.

10.3 Infinite Traces Model

This model [12] is the same as the Failures/Divergences model except that processes are now given a third component, the set of all infinite traces from Σ^ω they can perform. This set is always simply the closure $\overline{T} = \{u \in \Sigma^\omega \mid u = s\hat{~}u' \wedge s \in \Sigma^* \Rightarrow s \in T\}$ of the set T of finite traces, in the case that the process satisfies the finite nondeterminism constraint set out above. The Infinite Traces model $\mathcal{U}_\Sigma$ over Σ is just the set of triples $\langle F, D, I\rangle$ where $\langle F, D\rangle$ satisfies the five axioms of $\mathcal{N}_\Sigma$ and three further properties are satisfied, the details of which may be found in [12].

All the CSP operators (a set now extended to include the infinitely nondeterministic ones) have definitions of $\mathcal{U}_\Sigma$ that are straightforward extensions of those over $\mathcal{N}_\Sigma$. The only really interesting differences are in recursion, where it becomes much more difficult to justify both the use of, and existence of, least fixed points, and in the divergence clause of hiding. Here we no longer have to make inferences and so can define:

$$\begin{array}{rcl} \mathit{Divs}(P \setminus A) & = & \{(u \setminus A)\hat{~}t \mid u \in \mathit{Infs}(P)\wedge \\ & & u \setminus A \text{ is finite}\} \\ & & \cup\{(s \setminus A)\hat{~}t \;\mid\; s \in \mathit{Divs}(P)\} \end{array}$$

A good example of the sort of process we can model in $\mathcal{U}_\Sigma$ but not in $\mathcal{N}_\Sigma$ is FINITE_A as defined in Section 6. This is indistinguishable from CHAOS_A over $\mathcal{N}_\Sigma$ but, as we have seen, has a very different interpretation and uses.

11 Appendix: Decision procedures for finite-state systems

The properties we have introduced are simple to state but, unusually, none is a property that can be checked by refinement. This is because they are not *behavioural* specifications (assertions that each individual behaviour satisfies some condition), but are rather statements about the entire set of a process' traces. Thus they are not amenable to verification on the oroginal version of FDR, which can only check behavioural conditions.

Nevertheless it is possible to check them via algorithms that are closely related to those which FDR uses. Here we only describe the algorithm for checking determinism, which is both faster and checks the

[9] It is precisely this that makes the basic hiding security conditions 'eager'.

most satisfactory conditions, The approach to other conditions is described in the full version of the paper.

All our algorithms, like those of FDR, are based on representations of finite-state processes as members of labelled transition systems.

A *standard* LTS (SLTS) (V, E, v_0), with nodes V, labelled edges E and initial node v_0 is one where τ actions are permitted in addition to the visible actions from Σ, nodes may have multiple edges out of them with the same label, and nodes have no marking which is significant to the interpretation of the behaviour of the LTS. The usual operational semantics of CSP gives each term a meaning as an SLTS.

A node of an SLTS is said to be *stable* if it has no τ actions possible.

11.1 Determining determinism

Our algorithm for checking whether a process P represented as (V, E, v_0) is deterministic comes in two parts. The first is to find a deterministic process Q that refines P. This may be done by resolving all the nondeterminism that is encountered during a search through (V, E, v_0) starting at v_0: if a non-stable node is found we arbitrarily select one of its τ-successors; while for a stable node we select a single representative successor for each action it can perform. This search, either finds a divergence in P (which answers the determinism question in the negative) or delivers a deterministic LTS (one where all actions are visible and there is no ambiguity of successors under any action at a node) representing a deterministic Q such that $P \sqsubseteq Q$. The second phase of the algorithm is simply to check whether $Q \sqsubseteq P$. For clearly P is deterministic if and only if $P =_{FD} Q$, as the deterministic processes are all maximal (and therefore incomparable) in the failures/divergences model.

Because of the construction of Q, it is an easy specification to check against. (For example, the normalisation process is greatly simplified for processes that are known to be deterministic.) The final check might either be a full failures/divergences check or, if we can exclude the possibility of divergence some other way, the faster failures-only check. It will, for example, be sufficient to use the failures check on $P \parallel\!\mid RUN_H$ provided P is divergence-free.

Theorems 1 and 3 make the above a very powerful tool for deciding whether all our security conditions hold, not just the ones based explicitly on determinism. At the time of writing (November 1994), an implementation of the above algorithm for extracting a deterministic implementation has been produced and tried with promising results (using the standard refinement FDR routines for the refinement check phase).

References

[1] P.G. Allen, *A comparison of non-interference and non-deducibility using CSP*, In Proceedings of the 1991 IEEE Computer Security Workshop, pp43-54. IEEE Computer Society Press 1991.

[2] S.D. Brookes, *A model for communicating sequential processes*, Oxford University D.Phil thesis, 1983.

[3] S.D. Brookes and A.W. Roscoe, *An improved failures model for communicating processes,* in Proceedings of the Pittsburgh seminar on concurrency, Springer LNCS 197 (1985), 281-305.

[4] J.A. Goguen and J. Meseguer. *Security policies and security models*, in Proceedings of the 1982 IEEE Symposium on Security and Privacy, pp 11-20. IEEE Computer Society Press, 1982.

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

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

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

[8] D.M. Johnson and F.J. Thayer. *Security properties consistent with the testing semantics for communicating processes.* in Proceedings of the 1989 IEEE Symposium on Secrity and Privacy, pp9-21, IEEE Computer Society Press, 1989.

[9] D. McCullough, *Noninterference and the composability of security properties*, in Proceedings of 1988 IEEE Symposium on Security and Privacy, pp1770186. IEEE Computer Society Press, 1988.

[10] G.M. Reed and A.W. Roscoe, *Analysing TM_{FS}: a Study of Nondeterminism in Real-Time Concurrency*, in Concurrency: Theory Language and Architecture, (Yonezawa and Ito, eds) Springer LNCS 491.

[11] A.W. Roscoe, *Model-checking CSP*, in A Classical Mind: Essays in Honour of C.A.R. Hoare, A.W. Roscoe (ed.) Prentice-Hall 1994.

[12] A.W. Roscoe, *Unbounded Nondeterminism in CSP*, in 'Two Papers on CSP', PRG Monograph PRG-67. Also Journal of Logic and Computation **3**, 2 pp131-172 (1993).

[13] A.W. Roscoe, *An alternative order for the failures model* , in 'Two papers on CSP', technical monograph PRG-67, Oxford University Computing Laboratory, July 1988. Also appeared in Journal of Logic and Computation **2**, 5 pp557-57 7.

[14] A.W. Roscoe and H. MacCarthy, *Verifying a replicated database: A case study in model-checking CSP*, Submitted for publication.

[15] A.W. Roscoe, J.C.P. Woodcock and L. Wulf, *Non-interference through determinism*, Proc. ESORICS 94, Springer LNCS 875, pp 33-53.

[16] A.W. Roscoe and L. Wulf, *Composing and decomposing processes under security properties*, submitted for publication.

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