Process Algebra and Non-interference

P Y A Ryan
SRI Cambridge
23 Miller's Yard, Mill Lane, Cambridge CB2 1RQ

S A Schneider
Royal Holloway, University of London
Egham, Surrey, UK, NW20 0EX

Abstract

The information security community has long debated the exact definition of the term 'security'. Even if we focus on the more specific notion of confidentiality the precise definition remains controversial. In their seminal paper [4], Goguen and Meseguer took an important step towards a formalisation of the notion of absence of information flow with the concept of non-interference. This too was found to have problems and limitations, particularly when applied to systems displaying non-determinism which led to a proliferation of refinements of this notion and there is still no consensus as to which of these is 'correct'.

We show that this central concept in information security is closely related to a central concept of computer science: that of the equivalence of systems. The notion of non-interference depends ultimately on our notion of process equivalence. However what constitutes the equivalence of two processes is itself a deep and controversial question in computer science with a number of distinct definitions proposed in the literature. We illustrate how several of the leading candidates for a definition of non-interference mirror notions of system equivalence. Casting these security concepts in a process algebraic framework clarifies the relationship between them and allows many results to be carried over regarding, for example, composition and the completeness of unwinding rules.

We also outline some generalisations of a CSP formulation of non-interference to handle partial and conditional information flows.

1 Introduction

It is a source of mild embarrassment that the information security community has not yet reached a consensus as to the precise meaning on the term ‘security’ or even to the simpler question of what is meant by confidentiality. It is clear that confidentiality boils down to the absence of certain undesirable information flows, but this begs the question of what constitutes an information flow or its absence.

The thesis of this paper is that the characterisation of the absence of information flow is strongly related to characterisations of systems equivalence. The latter is of course a central concept in computer science and here too we find that there is no single agreed definition of what constitutes equivalence. Instead we again find a proliferation of definitions and it seems that which is appropriate depends on the particular application. We believe that identifying the analogies between the concepts in the security and the process algebra communities sheds further light on the notion of confidentiality and in particular helps explain the proliferation of definitions of confidentiality. Indeed the security community can perhaps derive some solace from the observation that the problem of defining non-interference is equivalent in difficulty to one of the central problems of computer science and so it should not be a source of surprise that consensus has eluded us.

A number of attempts to unify the various formulations of non-interference have been published recently, notably McLean [11] and Foccardi and Gorrieri [2]. These are very elegant and illuminating. We will show in detail in the later sections of this paper that there is a close correspondence between formulations of non-interference and formulations of process equivalence. It appears that by and large the proponents of the various forms of non-interference were unaware of the analogies with notions in the process algebra community and were independently reinventing these concepts in the security context.

Process algebras, in particular CSP, provide a general framework for describing interacting systems. Many of the formulations of confidentiality that one finds in the literature are presented in rather specialised and often specially tailored models of computation, so recasting them in a process algebra allows them to be generalised and compared. Another advantage of using a process algebraic framework is that it allows us to apply a number of established results, such as the completeness of unwinding rules, and compositionality.

We start with a brief introduction to those aspects of CSP that we use in this paper, followed by an introduction to non-interference. The next three sections present the main contribution of the paper: a number of formulations of non-interference in a CSP style, from the points of view of failures, bisimulation, and testing respectively. In Section 7 we discuss some generalisations of non-interference designed to encompass a richer class of security policies.

2 Process algebra

Process algebras provide a particular approach to the study of concurrency and interaction. This paper bases its discussion within the framework of the process algebra CSP (Communicating Sequential Processes). A full account of this process algebra can be found in [15, 20]. It provides a language for describing interacting systems, together with a semantic theory for understanding them. This section provides a brief introduction to those aspects most relevant to this paper.

The language of CSP is constructed around *events*: instantaneous synchronisations which provide the communication primitive. Events may have some structure, the most common being a channel communication of the form $c.v$, where c is the channel name, and v is the value communicated. For the purposes of this paper we will divide Σ, the set of all events, into two classes: H (high) and L (low). The set H will be further divided into high inputs HI and high outputs HO.

Processes are used to describe possible patterns of interaction, in terms of the events that they may engage in at any particular stage of an execution. In CSP, a process has an *alphabet*, or interface: the set of events it is able to synchronise on. If P already describes a process, then the process $c!v \rightarrow P$ describes another process which is prepared to output v on channel c, and behave subsequently as P. If $P(x)$ is a process for each possible value of x, then the input $c?x \rightarrow P(x)$ may take in some value v along channel c and behave subsequently as $P(v)$. Channels will generally have some type, which is the set of possible messages that can appear on that channel. The process $STOP_A$ has alphabet A and can perform no events at all. If the alphabet is clear from the context then the subscripted alphabet may be elided. For example, the process

$$in?x \rightarrow out!x \rightarrow STOP$$

is able initially to accept any input along channel *in*, and then to provide that same value along channel *out*, after which it can engage in no further activity.

The choice $P \sqcap Q$ may behave non-deterministically either as P or as Q. For example, the process

$$(out!1 \rightarrow STOP) \sqcap (out!2 \rightarrow STOP)$$

can choose to output either a 1 or a 2.

Processes may be put in parallel: $P \parallel Q$ behaves as P running concurrently with Q, synchronising on events in their common alphabets, and performing other events independently. An interleaving of two processes, $P \mid\mid\mid Q$, simply executes P and Q concurrently without any communication occurring between them. Values are passed between parallel processes by means of synchronisations on the channels, linking an output channel of one to an input channel of another. For example,

$$(in?x \rightarrow mid!(2*x) \rightarrow STOP) \parallel (mid?y \rightarrow out!(2*y) \rightarrow STOP)$$

has its two component processes synchronising on the channel *mid*. This enables information to flow from the left-hand process (which outputs a value) to the right-hand process (which inputs) along channel *mid*. Any value appearing on *out* will be four times a value that has been input on *in*.

The abstraction mechanism $P \setminus A$ describes the process P with all occurrences of A occurring internally in the resulting process. This is different from $P \parallel STOP_A$, which behaves as the process P with all occurrences of A blocked. For example,

$$((in?x \rightarrow mid!(2*x) \rightarrow STOP) \parallel (mid?y \rightarrow out!(2*y) \rightarrow STOP)) \setminus mid.\mathbb{Z}$$

will make the *mid* channel internal.

The process RUN_A has alphabet A, and it is always ready to perform any event from the set A. The process $CHAOS_A$ also has alphabet A, and can perform or refuse to perform any such event at any stage during an execution—it is the most non-deterministic (divergence-free) process with alphabet A.

Processes may also be recursively defined, by giving equations which contain the name of the process being defined as a subterm of the process expression. For example, the process

$$COPY = in?x \rightarrow out!x \rightarrow COPY$$

defines a buffer process *COPY* as one which repeatedly alternates input and output. Indexed processes may also be recursively defined using families of equations.

The semantics of processes are given in terms of observations. A process is identified with the set of behaviours that may possibly be observed of it, where the kind of behaviour considered determines the nature of the model.

The *Traces Model* is concerned with the traces of a process: the (finite) sequences of events that it can perform during some execution. For example,

$$\begin{array}{l} traces(COPY) = \\ \quad \{\langle\rangle\} \\ \quad \cup \{\langle in.v\rangle \mid v \in V\} \\ \quad \cup \{\langle in.v, out.v\rangle \mid v \in V\} \\ \quad \cup \{\langle in.v, out.v, in.w\rangle \mid v \in V, w \in V\} \\ \quad \vdots \end{array}$$

where V is the type of the channels *in*, *out*.

The notation $tr \upharpoonright A$ (pronounced '*tr* project *A*) denotes the projection of the trace onto the set A: the maximal subsequence of *tr* all of whose events are in A.

If *tr* is a possible trace of a process P, then P/tr (pronounced 'P after *tr*') denotes the process that P becomes after executing the trace *tr*. For example, $COPY/\langle in.7\rangle = out!7 \rightarrow COPY$.

The traces model is sufficient for many applications but it is not able to fully distinguish different non-deterministic behaviours. To deal with non-determinism we must turn to the failures model of CSP. A *refusal* of a process is a set of events that the process might initially refuse to engage in. For example, the set $\{out.1, out.2\}$ is a refusal of the process *COPY*. The set of all refusals of a process P is denoted $refusals(P)$

In order to consider refusals during an execution, the notion of *failure* is introduced: a failure of a process is a trace together with a *refusal set* which describes a set of events that the process might refuse to engage in after performing the trace. For example, $(\langle in.3\rangle, \{in.5, in.6\})$ is a failure of *COPY*. If (tr, X) is a failure of P, then X is a refusal of P/tr. The most non-deterministic process $CHAOS_A$ can exhibit all possible traces, and all possible refusals:

$$failures(CHAOS_A) \quad = \quad \{(tr, X) \mid tr \in A^* \wedge X \subseteq A\}$$

where A^* denotes all sequences whose elements are all drawn from A.

A *divergence* of a process is a trace after which it may perform an infinite sequence of internal actions. In this paper, we are assuming that the systems modelled do not diverge.

A process is *deterministic* whenever, given an arbitrary trace $tr \frown \langle a\rangle$, it could not have refused the last event instead of performing it: $(tr, \{a\})$ should not be a failure. Thus given any trace, there is only one possible response for each potential next event. Hence a process is *non-deterministic* if it has some trace *tr* such that $tr \frown \langle a \rangle$ is a possible trace and $(tr, \{a\})$ is a possible refusal. In other words, having performed the trace *tr*, if an environment offers *a* then it might non-deterministically either be accepted or refused. In the failures model a process *Q* is a refinement of another process *P* when the set of *Q*'s failures are a subset of *P*'s failures. This means that *Q* is more deterministic. Thus $CHAOS_A$ is refined by any process with alphabet *A*, and deterministic processes cannot be further refined: no process is a strict refinement of a deterministic process.

Processes are considered equivalent in a semantic model if they have the same set of behaviours in that model. Thus if *P* and *Q* have the same traces, then they are equivalent in the traces model, written $P =_{traces} Q$. This means that if only their traces are examined, then they cannot be distinguished. Similarly, if *P* and *Q* have the same failures, then this is written $P =_{failures} Q$. The subscript to the equality symbol can be dropped if it is clear from the context.

Process semantics may also be described in terms of operational semantics, which describe transitions between states (process descriptions). Thus $P \xrightarrow{\mu} P'$ describes the performance by *P* of a single μ event, reaching the state P'. μ ranges over visible events (drawn from Σ) and the special internal event τ. If *s* is a sequence of events, then $P \overset{s}{\Longrightarrow} P''$ describes a sequence of steps between successive processes, each step labelled by the corresponding event in the sequence *s*. Operational semantics allows alternative approaches to comparing processes.

Strong bisimulation [12] identifies processes if the states in their execution graphs match. This will be the case if there is a symmetric *bisimulation relation* *R* such that if PRQ and $P \xrightarrow{\mu} P'$ then $\exists\, Q' \bullet (P'RQ' \wedge Q \xrightarrow{\mu} Q')$, where μ is any transition label, including τs. This is the strongest useful equivalence between processes, and implies equivalence in any of the CSP semantic models. As well as requiring traces to match, bisimulation also requires the points at which choices are made within processes to match.

This can be relaxed by not requiring the two processes to match hidden (τ) events. Here we allow a visible event *a* in *P* to be matched by a sequence $\hat{a}$, for some $\hat{a} \in \tau^*.a.\tau^*$, the set of all sequences of τ actions interleaved with a single *a*. For internal events τ, we write $P \overset{\hat{\tau}}{\Longrightarrow} P'$ if there is some *n* (possibly 0) such that $P \overset{\tau^n}{\Longrightarrow} P'$. This allows us to define weak-bisimulation.

P and *Q* are weakly bisimilar with respect to the symmetric relation $\approx$ if and only if whenever $P \approx Q$ then $P \xrightarrow{\mu} P' \Rightarrow \exists\, Q' \bullet Q \overset{\hat{\mu}}{\Longrightarrow} Q' \wedge P' \approx Q'$.

In [3] Gardiner proposes a further weakening, introducing an equivalence relation on the power set of states, known as power-bisimulation. This allows sets of states of process *P* to be related to sets of states of process *Q*. A relation *R* between sets of states is a *power-bisimulation* if for all $\mu \in \Sigma \cup \{\tau\}$ and all sets of states $\mathcal{P}$ and $\mathcal{Q}$ we have that

$$\mathcal{P} R \mathcal{Q} \quad \Rightarrow \quad \{P' \mid P \overset{\hat{\mu}}{\Longrightarrow} P' \wedge P \in \mathcal{P}\} R \{Q' \mid Q \overset{\hat{\mu}}{\Longrightarrow} Q' \wedge Q \in \mathcal{Q}\}$$

This has the effect of abstracting out the effect of the point at which non-deterministic choices are made. This allows the construction of a form of bisimulation that is precisely equivalent to failures equivalence and will be discussed more fully in Section 5.

Another approach to comparing processes is in terms of *testing*. A test *T* is a particular kind of process, with some *SUCCESS* states. We consider the execution of *T* in conjunction with a process *P* and if $P \parallel T$ can reach a success state then '*P* may *T*'. If *P* may *T* whenever *Q* may *T* and vice versa for all possible tests *T*, then *P* and *Q* are equivalent under may testing. For further information on testing see [5].

3 Non-interference

Confidentiality policies are concerned with restricting classes of information flows. In the early days of the subject such policies were typically constructed by analogy with the paper world and so involved assertions such as: information should not flow from an object of higher classification to one of lower classification. Thus information flows are treated in a binary fashion: it is either allowed to flow or not, and the objects of the policy were typically fairly gross: data bases, files, agents etc. Such analogies now look rather dated in the light of the capabilities provided by current information processing systems and we will address the question of constructing more elaborate policies, with finer granularity of objects and more subtle controls of information flows, in Section 7. For the moment we will stick to this traditional viewpoint as, even in this comparatively simple context, there are a number of subtleties to be considered.

A central problem is to characterise the absence of information flow between objects, which in effect means across interfaces or along channels. The Bell and La Padula model takes the notions of *read* and *write* as primitives and so makes no attempt to formalise them. A number of critiques of Bell and La Padula are based on this observation. An early attempt to formalise the absence of information flow was the concept of non-interference proposed in the seminal paper by Goguen and Meseguer, [4].

Intuitively the idea is as follows: to establish that information does not flow from object A to object B it is sufficient to establish that A's behaviour has no effect on what B can observe. Put differently: B's view of the system is independent of A's behaviour. This latter suggests an appropriate way of capturing this mathematically: for any pair of behaviours of the system that differ only in A's behaviour, B's observations of the system cannot distinguish these two systems. This makes it clear that the notion of indistinguishability of behaviours is central. For systems that are deterministic it is fairly straightforward to make such equivalence precise and indeed it is deterministic systems that Goguen and Meseguer originally considered.

We will cast the Goguen/Meseguer formulation into a more CSP like notation for ease of comparison with later formulations.

Assume that High interacts with the system S via the interface H and Low via the interface L and further that these two interfaces partition the full interface of S. Then High is said to be non-interfering with Low via S if:

$$\forall\, tr : I^*, c : I \bullet Output_L S(tr, c) = Output_L S(tr \upharpoonright L, c) \tag{1}$$

where I is the set of inputs to S, and I^* the set of sequences of I's. $Output_L S(tr, c)$ denotes the output to Low from the system when it is in the state resulting from the sequence of inputs tr and when it receives a further input c. Finally, the projection to L ($tr \upharpoonright L$) is the trace tr with all occurrences of H events removed (given that H and L partition the alphabet of S).

In other words, whatever inputs High has performed the output that Low sees is the same as he would see if High had done nothing. Note that for a system in which the Low output is uniquely determined this equality is straightforward to define.

Even in this seemingly straightforward definition a number of subtleties lurk, for example: it depends on drawing a distinction between 'inputs' and 'outputs' without making the semantics of this distinction clear. This is a bit like the failure of the Bell La Padula model to give a precise semantics to the terms 'read' and 'write'.

More significant is the point that for systems displaying non-determinism characterising the equality of Equation 1 turns out to be rather delicate. Thus the seemingly innocent phrase '...B's observations of the system cannot distinguish...' actually conceals some subtle problems. The question of deciding when two processes should be regarded as equivalent is a difficult one and one to which a number of answers have been proposed. It is far from clear which of these is to be regarded as 'correct'. Indeed it seems reasonable to suppose that there is in fact no 'correct' notion; which is appropriate depends on the context and application in question.

The diversity of notions of system equivalence shows up most clearly in the process algebra community where we find, for example, traces or failures equivalence, various flavours of bi-simulation as well as various forms of testing equivalence.

4 Failures Equivalences

In an attempt to resolve the problems associated with drawing input/output distinctions as well as address the issue of non-deterministic systems, one of the authors proposed a recasting of the Goguen/Meseguer formulation into CSP [18]. Again the notation is tweaked slightly from the original 1990 presentation to make it more compatible with the rest of this paper.

$$\forall\, tr, tr' : traces(S) \bullet (tr \approx tr' \Rightarrow refusals(S/tr) \cap L = refusals(S/tr') \cap L) \quad (2)$$

where $tr \approx tr' \Leftrightarrow tr \upharpoonright L = tr' \upharpoonright L$. Different refusals sets can result after a given trace corresponding to different non-deterministic choices. The possible non-determinism of the system S is thus allowed for in this definition. Abusing notation, we take the distributed intersection of the refusal sets with L to get the Low level view.

CSP does not draw a distinction between inputs and outputs, both are regarded simply as 'events'. There is a danger in wrongly categorising events as input or output which is avoided, though arguably at the cost of a characterisation of confidentiality that errs towards being too strong. A simple example serves to illustrate this.

Consider a system with high and low inputs but only high outputs. Naively we would regard such a system as secure and yet it could fail the CSP characterisation. Whether or not this system really is secure depends critically on the semantics of the term 'input'. If 'input' is taken to mean an event wholly under the control of the environment that cannot be refused or delayed by the system then we probably would accept this system as being secure. However if there is any possibility of the system influencing the occurrence (even if not the value) of the 'input' event in any way we immediately have a channel from high to low. Any event that is wrongly characterised as an input in this sense could slip through and lead to a system incorrectly being deemed secure.

The CSP approach is thus a much safer criterion but could lead to some secure systems being rejected. If you are to draw such input/output distinctions and use them in the definition of security then they must be precisely defined.

In fact it turns out that, although CSP doesn't draw such distinctions, we can nonetheless use the framework, given in Equation 2, to distinguish at least the High (abstracted) input/output events.

This ability to distinguish inputs/outputs in CSP arises from our use of the more symmetric formulation of [18] instead of the more traditional form in which an arbitrary trace is compared to its purge. A consequence of this is that in this formulation it is not guaranteed that the purge of an arbitrary trace is itself a valid trace of the system. At first glance this appears to be a flaw as it seems to allow Low to deduce in some cases the occurrence of certain events at the high level. It actually turns out to be an advantage: the failure of the purge of a trace to be a trace is due to the occurrence of high signal events, i.e. events that cannot be refused or delayed by High. An example of such an event might be an alarm signal to alert High of some low-level activity. This clearly does not constitute a flow from High to Low and yet Low can deduce that such an alarm event has occurred.

For example, the process

$$S = l \rightarrow h \rightarrow l_2 \rightarrow STOP$$

has traces $\{\langle\rangle, \langle l\rangle, \langle l, h\rangle, \langle l, h, l_2\rangle\}$. The $purge_H$ of $\langle l, h, l_2\rangle$ is $\langle l, l_2\rangle$ which is not a trace of S. Thus when Low sees l_2 he knows h has occurred, but if High has no control over the occurrence of h it does not provide a means for High to signal to Low.

If we do not want h to be modelled as a signal event we should use the process

$$S = l \rightarrow ((h \rightarrow l_2 \rightarrow STOP) \;\square\; (l_2 \rightarrow STOP)$$

which allows High to refuse h without deadlocking the system. Now when Low sees l_2 he cannot tell if h has occurred.

Using the purge formulation would be equivalent to assuming that all events are refusable.

The issue of 'input totality' that was a concern for many of the early formulations is another manifestation of this difficulty, i.e. the problem of what happens if an invalid input sequence is presented to the system. This is dealt with automatically in a CSP approach: invalid inputs are simply refused by the system.

The approach of [18] deliberately tried to stay as close as possible to the spirit of the original Goguen/Meseguer formulation. In particular it sticks to quantifying over traces and comparison of the next events. It is interesting to consider how one might alter it to cast it in a more conventional CSP style. Firstly rather than just comparing next events we can compare the subsequent behaviours, i.e. assert equivalence as processes and use the conventional CSP hiding operator rather than having to use (distributed) set intersection:

$$\forall\, tr, tr' : traces(S) \bullet tr \approx tr' \Rightarrow ((S/tr) \upharpoonright L =_{failures} (S/tr') \upharpoonright L) \tag{3}$$

where $S \upharpoonright L$ is the projection of the set of failures of S to the set L: each failure (tr, A) is projected to $(tr \upharpoonright L, A \cap L)$.

In fact we can think of the earlier formulation as a kind of partial unwinding of this. Thus we can perform an inductive proof that Equation 2 implies Equation 3. It is also immediate that Equation 3 implies Equation 2, and hence that the two equations give the same characterisation.

In order to avoid the need to explicitly quantify over traces we might try expressing a characterisation directly as a CSP equivalence:

$$S \setminus H =_{failures} (S \parallel STOP_H) \setminus H \tag{4}$$

At first glance one might think that this is equivalent to Equation 3. Putting *STOP* in parallel with S over the alphabet H has the effect of preventing all traces with H events. We thus appear to be asserting that the process with all the H events enabled is equivalent to that with H events prevented. In fact it differs in some subtle but significant senses. Firstly this is really the purge formulation and so, in particular, it implies that the purge of any trace is itself a trace. Furthermore, the standard CSP semantics of such an equation asserts equality over stable states, that is states that cannot perform any internal (i.e. high level) events. For any state in which internal events can occur the refusal set might alter as the result the occurrence of such these transitions. It is therefore rather hard to make sense of the equality of refusal sets over unstable states, hence the decision to deal just with stable states in the standard CSP semantics. The quantification of Equation 3 however does force equality for all traces, not just traces leading to stable states (i.e. traces that can not be immediately extended by τ events). Equation 3 is therefore strictly stronger that Equation 4.

For example, the process

$S = h \to STOP \;\square\; (l \to STOP \sqcap STOP)$

meets Equation 4. On the other hand it fails Equation 3, since its failures after $\langle\rangle$ are not the same as its failures after $\langle h\rangle$, despite the low level views of both these traces being the same.

It turns out that we can cast Equation 3 in a form resembling Equation 4 and equivalent to it by using a different approach to concealing the High events. In Equation 3 we have concealed them by the rather obvious device of simply hiding them. An alternative is not to hide them but to camouflage them:

$$S \,|||\, RUN_H \quad =_{failures} \quad (S \parallel STOP_H) \,|||\, RUN_H \tag{5}$$

Here we interleave the two systems with RUN_H on the high level events. Now whenever Low sees an H event he cannot tell if it was performed by S or by RUN_H. By avoiding the use of hiding we have side-stepped the restriction to stable states. This form of abstraction was introduced by Roscoe et al [16], in which Equation 5 is shown to be equivalent to Equation 3.

The use of hiding to abstract certain events in this way provides us with another way of modelling signal events. Roscoe et al refers to this form of abstraction as ‘eager’ to reflect the idea that events abstracted in this way are thought of as occurring at the earliest opportunity. Correspondingly the interleaving abstraction is referred to as ‘lazy’. These forms of abstraction have also been characterised in a testing framework [21] which has the added advantage of allowing input/output distinctions to be drawn over the low-level events as well as the high-level.

These abstractions and variants of them are discussed in detail in chapter 12 of [15].

5 Bisimulation

It is usual when presenting a notion of non-interference to also present a so-called unwinding result. The idea is to present some constraints on the transitions of the system that together are equivalent to, or at least imply, the original property. Proving that a system obeys such constraints is more tractable than showing that it satisfies the original property.

In accordance with this tradition [18] presents such unwinding rules for an equivalence $\approx$:

- Rule 1:

$$\forall\, Y_i, Y_j : States(S) \bullet (Y_i \approx Y_j \Rightarrow Refusals(Y_i) \setminus H = Refusals(Y_j) \setminus H)$$

- Rule 2:

$$\forall\, Y_i, Y_j : States(S) \bullet Y_i \approx Y_j \Rightarrow \forall\, e : Initials(Y_i), \exists\, tr \bullet \begin{array}{l} \langle e \rangle \restriction L = tr \restriction L \\ \wedge\ Y_i/\langle e \rangle \approx Y_j/tr \end{array}$$

We have introduced an equivalence relation over states of the system. The first rule asserts that for two equivalent states Low's view of the possibilities for the next step of the system is identical.

The second rule has the effect of ensuring that the equivalence is exactly that induced by purging High events. That is, if we let Y_i denote a state reached after trace tr_i, then rule 2 implies: $tr_i \restriction L = tr_j \restriction L \Rightarrow Y_i \approx Y_j$.

Proving that these imply the original non-interference is a fairly straightforward induction style proof. Showing that these are also necessary is a bit more delicate and this is proved in a rather cumbersome fashion in an appendix of [18].

A far more elegant and instructive proof of completeness can be obtained by noticing that these rules bear a remarkable resemblance to a statement of bi-simulation equivalence as used in process algebras.

In fact weak bi-simulation is subtly stronger than failures equivalence as it insists on the bi-similarity of individual states and draws distinctions between processes on the basis of where non-determinism is resolved. A simple example (pictured in Figure 1) illustrates this:

$$\begin{array}{rcl} P & = & (a \rightarrow b \rightarrow STOP) \sqcap (a \rightarrow c \rightarrow STOP) \\ Q & = & a \rightarrow (b \rightarrow STOP \sqcap c \rightarrow STOP) \end{array}$$

The non-deterministic choice is made earlier in P than in Q and as a result it is easy to show that no bisimulation relation can be found for them. However an environment that can only observe the a, b and c events cannot distinguish between them. They are failures equivalent and testing equivalent.

Gardiner, [3], introduces the notion of a power bisimulation specifically to address this point and constructs a bisimulation-style equivalence that is exactly as discriminating as failures equivalence. [3] gives an elegant formulation and proof in terms of predicate transformers.

In order to construct a suitable power bisimulation for pairs of processes arising in the definition of non-interference it appears to be necessary (or at least convenient) to distinguish two distinct sources of non- determinism as observed by Low.

Firstly we assume that the system S with the H events hidden and τ (and L) events considered visible (with the τ events all distinguishable) is deterministic.

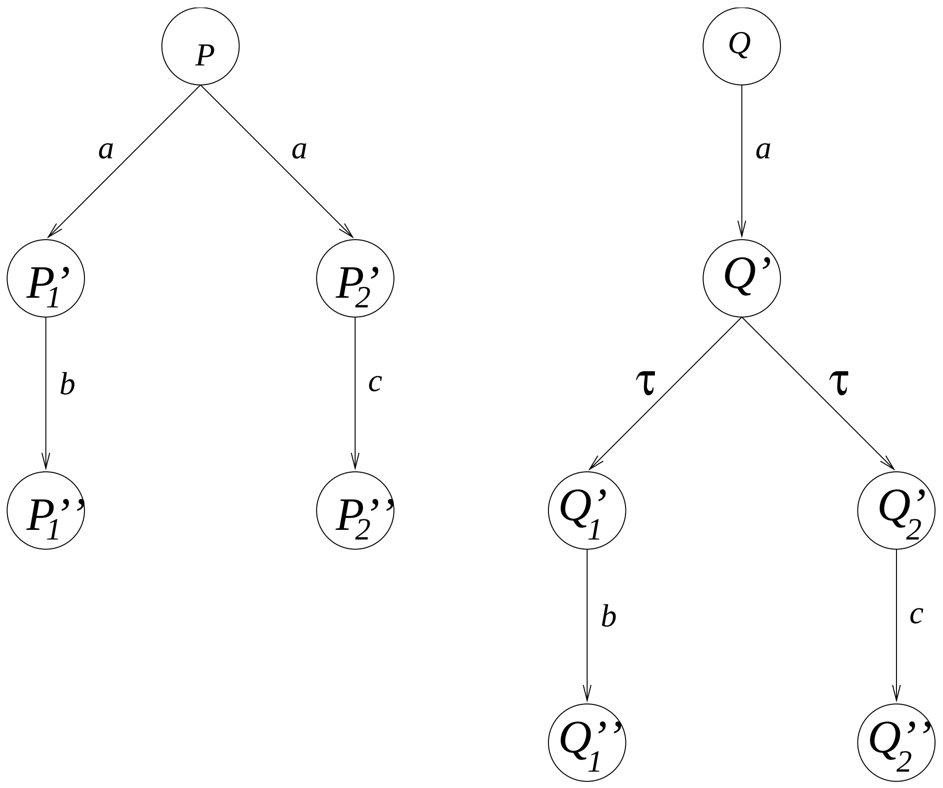

Figure 1: Processes P and Q

Thus, after any two runs of $S \setminus H$ with the same $L \cup \{\tau\}$ projection the system will end up in a pair of states for which the acceptance sets visible to Low will be identical.

In effect we are assuming that there is no non-determinism due to ambiguous visible (low) events and that all the non-determinism that low sees is due purely to the τs when we subsequently hide them.

In the earlier version if this paper [19] we allowed the possibility that $S \setminus H$ could itself manifest non-determinism and introduced the notion of loose-bisimulation to handle this. This degree of generality seems unnecessary and assuming that $S \setminus H$ is deterministic makes for a cleaner presentation. It also makes clearer the link between this approach and that of Roscoe, Woodcock and Wulf, [16]. In any case it is always possible to transform a system with non-determinism into a (failures) equivalent one in which all the non-determinism is absorbed into the τs.

In the following we will associate with every state s an acceptance set $\mathit{Initials}(s)$ that represents the set of events that the system will be prepared to participate in if any of them is offered by the environment. It is thus the complement of the maximal refusal set. When dealing with notions of bisimulation it is more convenient to think in terms of acceptance sets rather than refusal sets. Two states that are bisimilar will have the same initials as any event that can be performed in one state must also be possible in the other state. Note also that it is more usual in CSP to define such Initials only over the stable states. Here however it is convenient to extend the notation to all states.

For a system that is deterministic the initials set is sufficient to define the possible behaviours on the next step from a given state. When the system displays non-determinism we will be dealing with sets of states S and so we will have a corresponding set of acceptance sets that we denote $\mathit{AcceptSet}(S)$. Each element of $\mathit{AcceptSet}(S)$ is itself a set and will be the complement of the corresponding refusal set associated with that state.

We now define a power bisimulation relation $\approx_S$ induced by purging H's and τ's:

$$\approx_S \quad = \quad \{(\{S' \mid S \overset{tr}{\Longrightarrow} S'\}, \{S' \mid S \overset{tr'}{\Longrightarrow} S'\}) \mid tr, tr' : traces(S) \wedge tr \upharpoonright L = tr' \upharpoonright L\}$$

and for a set of states S and a define $\langle[S]\rangle_a$ by:

$$\langle[S]\rangle_a \quad = \quad \{S' : States(S) \mid \exists\, \tilde{a} \in H_\tau^*.a.H_\tau^* \bullet S \overset{\tilde{a}}{\Longrightarrow} S'\}$$

This can be interpreted as follows: given that the system is in a state from the set S and that the event a has occurred, then the resulting state reached must be in the set $\langle[S]\rangle_a$.

We can then easily prove the following:

Lemma 5.1 If $S \setminus H$ is deterministic then it satisfies a power-bisimulation with respect to $\approx_S$: in other words, given any two states S_1 and S_2 of S,

$$S_1 \approx_S S_2 \Rightarrow \langle[S_1]\rangle_a \approx_S \langle[S_2]\rangle_a$$

$\square$

We define the relation $\sim_S$ on states of S induced by purging the H events (but leaving L and τ's).

$$\sim_S \quad = \quad \{(S/tr, S/tr') \mid tr, tr' : traces(S) \wedge tr \upharpoonright L_\tau = tr' \upharpoonright L_\tau\}$$

where $L_\tau = L \cup \{\tau\}$

We can then further prove the following:

Lemma 5.2 If $S_1 \approx_S S_2$ then for all $s_1 : S_1$ there exists $s_2 : S_2$ such that $s_1 \sim_S s_2$, and vice versa.

$\square$

Lemma 5.3

$$s_1 \sim_S s_2 \Rightarrow \mathit{AcceptSet}_L(s_1) = \mathit{AcceptSet}_L(s_2)$$

$\square$

where $\mathit{AcceptSet}_L(s) = \mathit{AcceptSet}(s) \upharpoonright L$. This follows immediately from the fact that $S \setminus H$ is deterministic.

These two lemmas prove the following:

Lemma 5.4

$$S_1 \approx_S S_2 \Rightarrow \mathit{AcceptSet}_L(S_1) = \mathit{AcceptSet}_L(S_2)$$

$\square$

We have thus shown that a process that satisfies the assumption of being deterministic in its L and τ alphabet satisfies the power-bisimulation property with respect to the equivalence $\approx$. We have further shown that such a process satisfies the non-interference property formulated in terms of acceptance sets. This is subtly different and slightly stronger that the failures formulation in that it requires the acceptance sets to match exactly. For failures equivalence the subset closure of the refusal sets is automatically taken, which can mask certain distinctions. For most purposes the failures semantics is appropriate, as the only way that the environment can probe the system is by offering events and seeing whether or not one of them is accepted. However, if there is some more direct way to observe the acceptance sets, for example if the system simply displays lights corresponding to which events are available, then the acceptance sets semantics is appropriate. The acceptance sets formulation is safer: the set of systems that it will deem insecure is a superset of those using the failures formulation.

It is worth examining this result more carefully. We see that there are two sources of potential non-determinism in Low's view: High's activity and the occurrence of τ events. We are in effect asserting that for a non-interfering system all of the non-determinism that Low sees is attributable to the internal τ activity. Furthermore we see that a corollary of the determinism of $S \setminus H$ is that High is non-interfering with the τ's, i.e. High cannot influence the occurrence of τ events.

We will see the significance of this observation when we discuss the interaction of flavours of non-determinism in Section 6.

We can think of the τs as encoding the internal sources of non- determinism, for example due to a scheduler or even due to the output of a stream cipher.

To return to our simple example of Figure 1, we see that the power-bisimulation relation:

$$\{\ \{P\} \approx \{Q\}, \{P'_1, P'_2\} \approx \{Q', Q'_1, Q'_2\}, \{P''_1\} \approx \{Q''_1\}, \{P''_2\} \approx \{Q''_2\}\}$$

establishes the power-bisimilarity of P and Q above.

In [3] Gardiner provides constructions for power-bisimulations giving precisely the same discrimination as the equivalences of the traces, failures and acceptance sets semantics. The power-bisimulation relation constructed above for processes that are deterministic in their L/τ alphabets corresponds to that of Gardiner for the acceptance model and so we can immediately deduce the completeness of unwinding rules formulated as a power-bisimulation. The proof of this correspondence and further discussion of the traces and failures formulation will be given in a forthcoming paper.

The relevance of the notions of bisimulation to defining confidentiality was proposed by Foccardi and Gorrieri in, for example, [2], apparently without reference to analogies with unwinding. The use of the notion of power-bisimulation in security is, to our knowledge, new.

5.1 Composability

McCullough [9] was the first to consider composability issues: whether non-interference properties in the presence of non-determinism were preserved under various forms of composition. He proposed a generalisation of the original Goguen and Meseguer formulation to try to account for non-determinism but found that it failed to be compositional with respect to a certain composition, somewhat akin to CSP parallel. This prompted him to propose a new property (restrictiveness) precisely to obtain composability.

[18] contains proofs of the composability of the formulation of non-interference considered above with respect to various CSP operators.

Note however that composability depends on the property and the operators in question. The property of determinacy, for example, defined by Milner in [12] turns out not to be composable with respect to all the CCS operators, which prompts him to introduce the closely associated concept of confluence, which is.

Again there appear to be analogies here with some of the quests for composability in the security community and indeed confluence appears to bear some resemblance to the notion of forward correctability. The details of the correspondence are complex however and will not be addressed here.

6 Testing

A testing approach to non-interference will consider information to pass from the high level to the low level of a system if tests purely on the low level can enable some deductions to be made about what is occurring at the high level. There are different ways in which this may be formulated, depending on the power of the tests and what constitutes success of a test. It turns out that three formulations of flavours of non-interference can be given testing characterisations.

6.1 Non-deducibility

Sutherland's theory of non-deducibility, [23] characterises the lack of information flow in terms of an inability to deduce anything about high level behaviour from a low level view. A system M is said to exhibit this property if every low level view tr of an system execution is compatible with every high level view of inputs tr' (in the sense that there is some execution tr'' which presents both views). This captures the idea that no high-level behaviour can be ruled out by any low-level observation. It is described in process-algebraic terms as follows:

$$\forall\, tr \in traces(M \upharpoonright L), tr' \in traces(M \upharpoonright HI), \exists\, tr'' \in traces(M) \bullet \\ (tr'' \upharpoonright L = tr \wedge tr'' \upharpoonright HI = tr') \tag{6}$$

This is slightly different to the original definition, which was given in a synchronous framework (so all inputs and outputs occur together on every step), but it is the equivalent characterisation in an asynchronous setting.

It may also be characterised in a testing framework. A low level user may be considered as testing the system, attempting to elicit information concerning the high level inputs. Thus for non-deducibility to be present, such a low level user should not be able to rule out any possibility for $tr \upharpoonright HI$—the low level behaviour should be compatible with any high level user inputs.

Suppose that the test T has an alphabet L, and some distinguished *SUCCESS* states. A process may pass a test if there is some execution of the process together with the test which may reach a success state. A process P with its own success states may pass a test T through a system M if there is some execution of $P \parallel M \parallel T$ in which both P and T may reach a success state.

Then a system M exhibits non-deducibility if no test T through M may differentiate any two high level users $U1$ and $U2$ (drawn from the space of all possible high level users) which have alphabet HI, whose traces are only those of M restricted to HI, and which have some success states.

The restriction to alphabet HI corresponds to the requirement that any low level information is required to be compatible only with any sequence of high level inputs. This means essentially that no information at the low level is available about the high level input to the system—it is all masked by the system S. In other words, for any test T and any two high level users $U1$ and $U2$, $U1 \,|[\, HI \,]|\, S$ may pass T if and only if $U2 \,|[\, H1 \,]|\, S$ may pass T.

In the state machine formulations of the original definitions, all processes must always be able to accept any input values and provide outputs. This characterises an important subclass of CSP processes. If the space of high level users is restricted to such processes, then the testing formulation is equivalent to Sutherland's. However, CSP allows the consideration of more general classes of process if appropriate.

For example, the system

$$S \quad = \quad in_h?x \rightarrow (out_l!0 \rightarrow S \sqcap out_l!1 \rightarrow S) \tag{7}$$

apparently passes no information from high to low. However, the high level process $STOP$ can be distinguished from $in_h!1 \rightarrow STOP$ because a low level test will receive an output when the second is tested, but not when the first is tested. It depends whether $STOP$ is considered a valid description of a high level user or not as to whether S is considered to permit information flow. The characterisation is thus dependent on the space of processes over which the high level users U can range—the kind of activity which must be indistinguishable.

It is observed in [24] that Sutherland's definition has some unwanted consequences. A system meeting this property still permits a high level user to communicate information to a low level user, essentially because high level outputs (which are ignored in the definition) may have a bearing on later high level inputs. It is instructive to recast their example in CSP terms. The system $M(k_1, k_2)$ is parametrised by two one-bit keys k_1 and k_2. It takes a high level input h_i, and offers a high level output h_o and a low level output l. On a high level input x of 0 or 1, it outputs that value xor'ed with the key k_1. Both keys are then randomly reset. On any other high level input (the original example uses a special value q) the second key k_2 is output to the low level and randomly reset, and the first key k_1 is output to the high level.

$$
\begin{array}{rcl}
M(k_1, k_2) & = & h_i?x \rightarrow \ \textbf{if } x \in \{0, 1\} \\
 & & \qquad\qquad \textbf{then } \ h_o!0 \rightarrow l!(x \oplus k_1) \rightarrow \\
 & & \qquad\qquad\qquad \sqcap_{k_1, k_2 \in \{0,1\}} M(k_1, k_2) \\
 & & \qquad\qquad \textbf{else } h_o!k_1 \rightarrow l!k_2 \rightarrow \sqcap_{k_2 \in \{0,1\}} M(k_1, k_2)
\end{array}
$$

It is easily checked that this system allows any high level inputs with any low level view, and hence M exhibits non-deducibility.

However, a high level user can use this system to communicate to the low level, by employing a particular strategy. If the high level wishes to transmit a particular bit b, then he first learns the value of k_1 by inputting q, and then inputs $k_1 \oplus b$, which will result in the transmission of b to the low level. Based on this, a transmitter wishing to communicate a sequence $\langle b \rangle ^\frown u'$ of bits behaves as follows:

$$
T(\langle b \rangle ^\frown u') \ = \ h_i!q \rightarrow h_o?k \rightarrow h_i!(k \oplus b) \rightarrow h_o?k' \rightarrow T(u')
$$

The CSP formulation in terms of HI draws attention to the weak point of Sutherland's definition. The only high level activity it is concerned with is high level input. However, high level users also interact with the system by receiving output, and in this case the output received can influence later input. Thus the definition does not capture all the ways in which the high level user can interact with the system.

The example of [24] can be replaced with a rather more intuitive and immediate example that we believe shows up the essence of the problem.

Consider a system in which high level data is encrypted and the encrypted form transmitted over a low channel. Thus far we have the classic scenario that arises in many security architectures and yet has proved so difficult to capture in a non-interference style framework. Assuming that the encryption scheme is itself 'secure' (in a cryptanalytic sense) then we would tend to accept that such a system is secure.

Suppose now that the high user can in fact observe or predict the key stream before he submits his plain-text for encryption. We can easily imagine this occurring in a number of ways and in fact Wittbold et al's example achieves essentially this. Then High can modify the plain-text in such a way as to communicate it to

Low, in particular he can simply xor the predicted bits into the plain-text before submitting it to the encryption resulting in raw plain-text being broadcast over the low channel.

This is not too surprising so far but let us examine it more closely as it is a pointer to the root of the problem. The example involves two kinds of non-determinism: external (under the control of the users or processes, in particular High) and probabilistic, arising, in this case, from the cipher stream. In the above example we have allowed these to interact: we have allowed High to resolve his non-determinism basing his decisions on observations of how the probabilistic non-determinism is being resolved. If we were to force High to resolve all his non-determinism at the outset, before any of the cipher stream has been generated, or in the absence of any knowledge of this source of non-determinism, then he could not exploit this channel.

We see that this discussion touches on the points made earlier regarding the loose-bisimulation. The loose- bisimulation property implied that High should not be able to interfere with the τ events. If we can think of the τ events as representing the non-determinism associated with the cipher stream then we see that this is asserting that High cannot interfere with the cipher stream. The situation is a little delicate as the bisimulation does not prevent flow from the τ's to High which potentially could allow the scenario described above. In fact High xor'ing the cipher stream to the data stream is tantamount to interfering with the stream but it is not immediately clear that this has been formally captured. This will be a topic for further investigation. An obvious solution is to also require that the τ's do not interfere with High but this seems a bit heavy handed. It would however be perfectly reasonable where the τs represent the output of a stream cipher for example.

Constructing semantic models for a process algebra to accurately capture the distinction between the various forms of non-determinism has proved difficult. A number of probabilistic process algebras have been proposed, for example Morgan [10] and Lowe [7]. These assume that non-determinism is resolved at the outset and so do not allow for the possibility of delaying choices and making decisions on the basis of observations of how non-deterministic choices are resolved 'at run time'. We thus see why it has proved so difficult to model this scenario in a non-interference style.

Currently we do not appear to have theories rich enough to fully capture the distinction between the various flavours of non-determinism. Indeed it is not clear that it would be particularly effective to try to develop and apply such extended theories as the greater complexity would probably render them unusable. For the crypto example at least it would seem more effective to use arguments outside the model to establish that the crypto is wholly independent of the rest of the system. In fact a well designed crypto device appropriately incorporated in a secure architecture will have precisely these characteristics: the cipher stream will be unobservable and unpredictable by High (or indeed any other user or process). High may of course be able to deduce the cipher stream after the event if he can observe the cipher-text stream but, crucially, this is too late to exploit. Furthermore an ideal encryption algorithm will have the property that observing an arbitrary length of the stream should not enable further bits to be predicted, i.e. the stream is effectively random. We of course have to take care that any implementations maintain these assumptions. Conventional refinement techniques will not preserve the non-determinism so from a refinement point of view this non-determinism has to be handled differently.

Such a situation is not entirely satisfactory but appears quite workable, at least for a large class of systems. Of course it might be that we encounter systems which are secure but for which the flavours of non-determinism cannot be so conveniently isolated. This could probably be regarded as a symptom of a poorly designed system.

6.2 Non-deducibility on strategies

To avoid the problem with non-deducibility outlined in the previous section, the notion of non-deducibility on strategies was introduced [24]. In this formulation, any low view needs to be compatible with any high transmitter strategy, where a strategy describes what a high level user will input depending on previous inputs and outputs. Any low level trace view should thus be compatible with any high level view that is possible for the system. In testing terms, a strategy will be a high-level user U, and a system S will provide non-deducibility on strategies if for any test T and any two high level users $U1$ and $U2$ meeting the same restrictions as previously, $U1 \,|[\, H \,]|\, S$ may pass T if and only if $U2 \,|[\, H \,]|\, S$ may pass T.

This is equivalent to a simple variation on Equation 6:

$$\forall\, tr \in traces(S \upharpoonright L), tr' \in traces(S \upharpoonright H), \exists\, tr'' \in traces(S) \bullet \\ (tr'' \upharpoonright L = tr \wedge tr'' \upharpoonright H = tr') \qquad (8)$$

(Note that the example $M(k_1, k_2)$ above does not have this property.)

This definition incorporates all possible high level interactions between the system and the high level users, because they can interact on the entire high level interface *H* and not simply on *HI* as in Equation 6. This means essentially that no information at the low level is available about any high level activity of the system at all.

The nature of the possible high level users (or strategies) affects whether or not a system meets this definition. For example, when high level users are restricted to the subclass of non-deterministic state machines (which alternate on inputs and outputs for ever), then the system *S* of Equation 7 meets the definition. However, if *U* can range over all possible CSP processes, then *S* does not meet the definition: a low level user could distinguish between *STOP* and $in_h!0 \rightarrow STOP$. The high level users permitted will depend on the nature of the situation being modelled: if high level users must always provide input when requested, then considering the users to range over such a subclass would be appropriate.

It is immediate that in the testing formulation Equation 8 implies Equation 6: if a test distinguishes the inputs of two high level users, then the same test will distinguish their high level activity.

6.3 Noninference

Noninference (see e.g. [13]) is a weaker property than the two previously considered. Rather than require any low level view to be compatible with any high level activity, it simply requires that the low level view of any execution must be compatible with no activity at all at the high level. Hence it must always be possible to elicit the low level activity even if there is no high level activity at all. This property follows immediately from both of those given above: if a low level trace is compatible with any high level trace (either on just high-level inputs, or on all high level events), then it must be compatible with the high-level empty trace.

This is expressed as follows:

$$\forall\, tr \in traces(S) \bullet (tr \upharpoonright L \in traces(S)) \qquad (9)$$

The equivalent formulation in testing terminology gives some insight into the nature of non-inference. The testing characterisations of non-deducibility and non-deducibility on strategies permitted the high level user (as well as the test) some control over when a successful state is reached. If this capability is removed, then success is dependent entirely on the state of the testing process *T* (i.e. ω is not in the alphabet of any of the *U* processes). Non-interference for *S* can still be viewed as the requirement that $U1 \,|[\, H \,]|\, S$ may pass *T* if and only if $U2 \,|[\, H \,]|\, S$ may pass *T*, but the notion of success is independent of the state that the $U1$ and $U2$ processes might have reached.

It turns out that this formulation is equivalent to non-inference. Assume first that S meets Equation 9. Given $U1$ and $U2$, assume without loss of generality that $U1 \parallel S$ may pass T. Then there is some trace tr of $U1 \parallel S \parallel T$ that takes T to a success state. Now $tr \in traces(S)$, and so $tr \upharpoonright L \in tracesS$. Since $\langle\rangle \in traces(U2)$, it follows that $tr \upharpoonright L$ is a trace of $U2 \parallel S \parallel T$. Thus T can reach the same success state when testing $U2 \parallel S$, and hence $U2 \parallel S$ may pass T. Thus no two high level users can be distinguished by any low level may test T.

Assume now that S meets the testing characterisation. Given a trace tr of S, there is a test T that succeeds only after performing $tr \upharpoonright L$, and a high level user U that can perform $tr \upharpoonright H$. Thus $U \parallel S$ may pass T. Hence $STOP_H \parallel S$ may pass T, so there is some trace tr' of $STOP_H \parallel S$ which takes T to a success state. Hence $tr' \upharpoonright L = tr \upharpoonright L$, and $tr' \upharpoonright H = \langle\rangle$, since $STOP_H$ blocks all high level events. Hence $tr' = tr \upharpoonright L$, and so $tr \upharpoonright L$ is a trace of S.

In fact, it also turns out that S in conjunction with any high level user must give rise to the same set of low level traces: that $(U \parallel S) \setminus H)$ should give the same (low level) traces whatever high level process H is used.

6.4 A deterministic approach to Non-interference

An alternative definition of non-interference in the process algebra context was proposed by Roscoe, Woodcock and Wulf [16]. In their approach, a system S does not allow information flow from high to low if the system

$$Abstract_H(CHAOS_H \parallel S) \tag{10}$$

is deterministic. $Abstract_H$ denotes an appropriate abstraction of the H events to obtain a Low view of the system. Typically it will involve a mix of lazy and eager abstraction depending on whether the high events are considered delayable on not. This system presents events only at the low level, and its determinism means that there is only one possibility for each event at any stage (offer, or refusal), however the non-determinism in the system is resolved. Since any high level user will be a refinement of $CHAOS_H$, and since all CSP operators preserve the refinement relation, this means that any particular high level user U_H in parallel with the system will have to be a refinement of $CHAOS_H$ is parallel with the system. Since the result (when H is hidden) is already deterministic, this means that it cannot be further refined so it must remain unchanged. This is true for any high level user U_H, so all possible high level users must give the same result at the low level. Hence, no low level test can distinguish between them. It follows that this formulation implies non-deducibility on strategies. In a sense, $CHAOS_H$ embodies all possible strategies, and the requirement that the resulting system is deterministic states that all such strategies must always give the same result.

Advantages of this approach are that it side-steps much of the debate about what is the appropriate notion of process equivalence, since virtually all the process algebras agree on what processes are deterministic. It is compositional, it is preserved by conventional refinement and it is automatically checkable (using for example using the built-in determinism checking facility of FDR).

It is clear that where it can be applied this is an extremely effective characterisation of absence of information flow. The drawback is that there appears to be a large class of systems for which this property is too strong: for which some degree of low-level non-determinism is unavoidable. Perhaps such systems are best handled by isolating such 'essential' non-determinism and showing that the property is preserved by refinement of the rest of the system.

An example will be instructive:

$$S \quad = \quad in_h?x \rightarrow STOP \mid\mid\mid ((out_l!0 \rightarrow STOP) \sqcap (out_l!1 \rightarrow STOP))$$

As it stands, this system does not allow information to flow from the high to the low level. It meets all of the characterisations of non-interference covered in this paper, except for the deterministic one.

Information will not flow from the high to the low level if the non-determinism in the system is essential, and cannot be removed or subverted in some way, (perhaps by a poor implementation of random choice). However, different runs with the same high level user (say $in_h!1 \rightarrow STOP$) can result in a number of different possible low level behaviours. Thus S system does not meet definition 10, since it is non-deterministic at the low level.

However, if the system S can be refined when it is implemented, then it is not clear that the resulting system will retain the non-interference properties required. For example, the process

$$S' \quad = \quad out_l!0 \rightarrow in_h?x \rightarrow STOP \;\Box\; in_h?x \rightarrow out_l!1 \rightarrow STOP$$

is a refinement of S above. However, S' does not provide non-inference, let along non-deducibility or non-deducibility on strategies. For example, the trace $\langle in_h.1, out_l.1\rangle$ is possible for S', but $\langle out_l.1\rangle$ is not. The value on the low level output provides information about whether the high level input has occurred.

If we are dealing with such a situation, where the non-determinism in the system is not guaranteed to all be retained, or where there is some doubt over its eventual form when the system is implemented, then a definition of non-interference which allows some measure of refinement will be useful. In this case definition 10 is useful, since it is preserved by refinement: any refinement of S will have to result in a refinement of $\mathit{Abstract}_H(\mathit{CHAOS}_H \parallel S)$, which must therefore be deterministic.

Another characterisation of information flow which allows non-interference to be preserved by refinement is proposed in [8], where Lowe introduces a notion of capturing information flow. His definition is weaker than the deterministic characterisation, though it is also preserved by (a restricted form of) refinement. Lowe states that there is absence of information flow if no 'consistent' way of resolving the internal choices in the system can allow information to flow from high to low in Ryan's sense. In the example S above, Lowe's approach will not allow the resolution of the internal choice to depend on the occurrence of the high level input, since there is no causal relationship between the occurrence of that input and the occurrence of the outputs. Hence such a refinement will not be permitted. This requires an extension to the CSP semantics, essentially to model true concurrency.

On the other hand, if the random choices in S can be guaranteed to be retained, we might say that the non-determinism in the system is essential. In this case refinement of S is not a concern and the CSP formulation of non-deducibility on strategies is sufficient to establish a lack of information flow.

It may be that we can establish a link between the determinism approach and the power-bisimulation approach presented in Section 5. If we think of all the non-determinism observed by Low as resulting from different τ activities then we could assert that to be non-interfering an abstraction of the system that obscures the High events but retains the τ's should be deterministic. This would have many of the advantages of the deterministic approach whilst allowing some non-determinism to manifest itself in Low's view, allowing for example the encrypted channel examples.

The difficulty is that τ's are not really part of the standard CSP framework and that further non-determinism could arise from ambiguous τ transitions: two distinct transitions from a given state that are both labelled by a τ. Both of these problems can addressed by not labelling them as τ events but simply regarding them as another set of events hidden from Low and disjoint from High.

7 Generalisations of Non-interference

7.1 Motivation

The idea of non-interference is clearly a central one in information security. It is however often argued that in practice it is too strong, that no real policy ever calls for total absence of information flow over any channel and that in any case it is not achievable. For example even the so-called one-way regulators in fact allow for a very low bandwidth feedback from high to low to regulate the upward flow and prevent buffer overflows etc.

Even for the comparatively straightforward MLS style policies simple non-interference runs into problems, for example the encryption problem discussed earlier and the need to incorporate downgrades.

As information processing systems steadily grow more sophisticated and distributed the old paper world analogies look increasingly dated. The demand for security in the commercial sector as well as trends in the military sector are prompting the need for more flexible policies, with a finer granularity of objects and more subtle controls of flows. New paradigms like object orientation, hypertext, virtual machines, mobile code and agents etc allow a far greater granularity of objects to be considered as the elements of a security policy. More sophisticated styles of policy often call for history or location based access decisions and these again cannot be reduced to predicates on the classification labels of individual events or interfaces.

All of these suggests the need to investigate ways in which we might generalise non-interference to allow for partial, conditional flows etc.

7.2 Formalisation

We use as our starting point the formulation given in Section 3. The most liberal generalisation of this appears to be:

$$\begin{aligned}&\forall\, tr, tr' : traces(S) \bullet \\ &(tr \approx tr' \Rightarrow Abs_H((S \parallel Constrain)/tr) \equiv Abs_H((S \parallel Constrain)/tr'))\end{aligned} \tag{11}$$

where Abs_H denotes an appropriate form of abstraction to model the low-level view. We have a number of abstraction operators available to us, for example the various flavours of lazy and eager hiding. Another abstraction that has been used in formalising notions of anonymity but hitherto not confidentiality is ‘projection’, renaming a set of events to a single event. We will see later how this operator is useful in addressing the encryption problem. These abstractions can be used selectively across the alphabet of the system, i.e. we might want to abstract certain events that we think of a signal events eagerly whilst other we abstract lazily.

$\approx$ denotes an equivalence relation between behaviours. Hitherto this has typically been defined in terms of some kind of purge function on traces but there seems to be no reason to restrict ourselves to such equivalences.

$\equiv$ denotes an appropriate process equivalence. Which is appropriate depends on the system and policy in question but failures or testing equivalence would seem most likely. In particular using a testing equivalence framework makes it possible to tailor the class of tests to the system and policy in question.

'*Constrain*' defines a set of high-level behaviours for which we wish to restrict the flow of information. The influence of behaviours of S that fall outside this envelope on the low-level will be unconstrained by Equation 11. We are regarding them as innocuous and we do not care how they interfere with Low. Presumably a policy might be formulated as the complement of this: stating what behaviours must not interfere, in which case Constrain might not itself be a process, i.e. might not satisfy the closure axioms for a process. How useful a degree of freedom this represents is unclear without trying it out against a sample of policies but we have included it for completeness.

A policy might be encoded as the conjunction of a set of equations of the form of Equation 11.

We can see how various forms of partial or constrained non-interference could be captured in this way. Such a formulation allows Low to determine High's behaviour up to the $\approx$ equivalence. That is, Low can determine which equivalence class High's behaviour belongs to but not where it lies within that class. We can thus define partial information flows and indeed we can arrange to have 'non-transitive partial flows': for which information flows from A to B and from B to C yet none flows from A to C.

Consider a data space that can be covered by the coordinates (x,y). Our $A \rightarrow B$ projection loses the y information so projects to (x,y'), for some arbitrary y'. The $B \rightarrow C$ projection loses the x information so we get $(x,y) \rightarrow (x',y)$. Putting these two projections together one after the other then gives a projection $A \rightarrow C$: $(x,y) \rightarrow (x',y')$, i.e. loss of all information about the original point in the A space.

This example is a bit simplistic as one could easily capture it in a simple labelling formulation. It does serve to illustrate the idea and one can easily construct something more elaborate. A more realistic instance would be a policy that allows for automatic downgrading of certain statistical information from a data base.

Another example in which a more general notion of equivalence seems appropriate is where we consider a high level process editing a file. In general many edit sequences could result in the same final text (we are assuming here that the editor is such that all the details of changes are forgotten and only the final resulting text retained, i.e. there is no mark-up facility). We thus want to regard all edit sequences resulting in the same final text as equivalent. Here again the idea of confluence crops up: that different sequences of actions give rise to equivalent states, in particular that certain pairs of actions may be commutative. Thus we might define an equivalence on traces with respect to certain permutation groups on actions.

Another possibility is to use a formulation with High processes rather than High traces, requiring that for any two high level processes U_1 and U_2:

$$U_1 \sim U_2 \quad \Rightarrow \quad Abs_H(S \parallel U_1 \parallel Constrain) = Abs_H(S \parallel U_2 \parallel Constrain)$$

where $\sim$ denotes a suitable equivalence over High processes.

Turning now to the problem of modelling high-level data being encrypted and send out over a low-level channel. Suppose that we take $\approx$ to be defined by:

$$tr \approx tr' \Leftrightarrow \#(tr \upharpoonright H) = \#(tr' \upharpoonright H)$$

and for our abstraction on the encrypted channel we rename the 0's and 1's of the enciphered stream to * say. Then we see that L can figure out the length but not the content of the H string. This would appear to accurately model the encrypted channel scenario. We have of course to carefully justify the projection on the cipher channel, presumably using cryptanalytic arguments.

There remains an issue of how to capture the effect of breaches of the crypto system in such a framework, particularly the retrospective effect (previously concealed information is revealed). This remains an open issue, though it does appear to bear a curious resemblance to the downgrading problem, discussed in the next section.

Alternative approaches to the encryption problem suggest themselves. One is to think in terms of ensembles of possible experiments, or tests, that Low might attempt. We can them compare the ensemble of all possible outcomes of such experiments for two different High-level input streams of length n. We assume that for each run of the experiment a fresh, random cipher stream is generated. For a truly secure channel these will be the same in both cases, i.e. Low will get the set of all possible bit streams of length n in both cases.

Another issue is the fact that Low can test for equality of cipher streams even if the streams themselves are meaningless to him. The situation has analogies with the notion of data-independence with equality testing. A process is said to exhibit such data-independence in a given data-type if the only operation the process can perform on variables in the data type is equality testing. Thus the absolute values of variables in the data-type are irrelevant. From Lows point of view a system with a secure encryption over a channel visible to him can be thought of as being having data-independence with equality testing in the data type of that channel.

A recent paper by Lazić and Nowak [6] gives a semantic definition of the concept of data-independence with equality testing that could be applied to the encryption problem. In essence we are seeking a notion of process equivalence up to isomorphism, or more precisely in this case up to renaming of events of the appropriate type. This will be discussed more fully in a forthcoming paper.

Notice also that the notion of confidentiality introduced in this example appears to be analogous to some of the notions of anonymity introduced in, for example, [22]. Here of course we are thinking of anonymity over a message space rather than an agent space.

7.3 Non-transitive non-interference

Another class of application that is amenable to this approach is that of so-called non-transitive non-interference and channel control policies addressed by Rushby in [17] . A couple of examples are presented for motivation: downgrading and a crypto device. The essence of the problem appears to be that though we want to allow flow from H to L it must be regulated, or at the very least audited, by an intermediary. Thus if a high-level data item is to be downgraded and issued to Low this can only happen if accompanied by appropriate actions by a Downgrader process. Similarly high-level data should only be passed to a low-level channel via the crypto device.

Rushby captures this by introducing a more elaborate purge function that, rather than acting in a purely pointwise fashion on the traces, takes account of the affect of downgrade events on the security labels of high events. Clearly we can capture such a policy in our framework with a suitable choice of equivalence, in particular that induced by Rushby's 'ipurge' will do. However we could envisage capturing different forms of intransitive policies to those considered by Rushby by exploiting the full generality of the equivalence $\approx$ of Equation 11.

A later paper by Pinsky [14] presents an algorithm to construct, where it exists, a minimal equivalence and associated unwinding rule for a downgrading policy.

The details are quite complex but it does appear that the algorithm presented by Pinsky can be thought of as an algorithm to construct the appropriate bisimulation relation. Indeed it seems probable that many of the unwinding results presented in the security literature can be interpreted in this way. A number of algorithms for establishing bisimulation relations between putatively equivalent processes are known in the process algebra community and it is probable that they could be usefully applied in the security context.

Note that for a policy encoded in Equation 11 the equivalence relation used can be used to induce a (power) bisimulation relation so establishing an unwinding result. Similarly the composability of such a property follows directly from the composability of the process equivalence.

8 Conclusions/Discussion

The central thesis of this paper is that the problem of formalising the notion of confidentiality boils down to that of formalising the equivalence of processes. The latter is a central and difficult question at the heart of computer science to which there is no unique answer. Which notion of equivalence is appropriate depends on the context and application. Consequently we should not be surprised that the information security community has failed to come up with a consensus on which constitutes confidentiality. Indeed, in this paper we have shown a close correspondence between various proposals for definitions of confidentiality in the security literature and forms of process equivalence in the process algebra literature. Where the system's security can be characterised as the determinism of the low-level view we are in better shape as the definition of determinism is fairly uncontroversial. It is not currently clear how large a class of real systems can be handled in this way.

Viewing security from a process algebraic framework brings with it a number of ready-made results and insights, particularly regarding composition and unwinding. It also helps to pin-point and isolate the source of many of the problems that have been encountered in the security literature, for example regarding the encryption problem, the lack of compositionality of certain formulations etc.

The usefulness of the concept of testing equivalence has also emerged for this viewpoint. This concept seems to have been curiously neglected in the context of defining non-interference though it has recently been used in the analysis of security protocols, see for example Abadi and Gordon, [1]. The philosophy is strikingly similar: in [1] tests can be understood as encapsulating all possible attacks, and equivalence under testing establishes that no such attack can succeed in distinguishing a real system from an ideal one. In our context, tests might be understood as encapsulating all possible ways in which information may flow from high to low, either as a result of High attempting to communicate information, or from the point of view of Low attempting to elicit it, or indeed involving some collusion. Equivalence under testing establishes that no such strategy can succeed.

It seems very natural to think of two systems as being equivalent if no test can distinguish them. Of course the problem then becomes one of what class of tests is appropriate. In particular the generalised notion of testing introduced in [21] may prove useful in drawing a distinction between delayable and undelayable events at the low-level.

Furthermore we have shown the significance of the notion of power-bisimulation. It establishes a correspondence between the testing, bisimulation and denotational styles of defining process equivalence.

We have proposed a generalised form of non-interference and shown that it can encompass a number of systems and policies of interest: with encrypted channels, partial, statistical and conditional flows, downgraders etc.

We have only addressed the so-called ‘possibilistic’ notions of non-interference, i.e. whether High can influence the possibility of Low performing certain observations. In other words we are abstracting away from issues of probability and time, in particular. Clearly these are important and need to be addressed. However it is also clear that even in the comparatively simple context of possibilistic models there are many subtleties lurking. We hope that this paper has served to identify and shed some more light on these subtleties.

8.1 Acknowledgements

Much of the research reported here was carried out during a collaborative visit at Royal Holloway, and the first author is grateful to the Royal Holloway Department of Computer Science for their hospitality. We would also like to thank Bill Roscoe, Michael Goldsmith, Sylvan Pinsky, Paul Gardiner, Martin Abadi and Nick Moffat as well as the reviewers for useful discussions and comments.

References

[1] M. Abadi and A. Gordon. A calculus for cryptographic protocols: the spi calculus. *Information and Computation*, 1999.

[2] R. Focardi, A. Ghelli, and R. Gorrieri. Using noninterference for the analysis of security protocols. In *DIMACS workshop on Design and Formal Verification of Security protocols*, 1997.

[3] P. Gardiner. Power simulation and its relation to traces and failures refinement. In *DERA/RHUL Workshop on Secure Architectures and Information Flow*, volume 32 of *ENTCS*, 2000. `URL: http://www.elsevier.nl/locate/entcs/volume32.html`.

[4] J. A. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy*, 1982.

[5] M. Hennessy. *Algebraic Theory of Processes*. MIT press, 1988.

[6] Ranko Lazić and David Nowak. A unifying approach to data-independence. In *Proceedings of the 11th International Conference on Concurrency Theory (CONCUR 2000)*, Lecture Notes in Computer Science. Springer-Verlag, 2000.

[7] G. Lowe. *Probabilities and Priorities in Timed CSP*. D. Phil thesis, Oxford University, 1993.

[8] G. Lowe. Defining information flow. Technical Report 1999/3, Leicester University, 1999.

[9] D. McCullough. Specificaitons for multi-level security and a hook-up property. In *IEEE Symposium on Security and Privacy*, 1987.

[10] A. McIver, C. Morgan, K. Seidel, and J. Sanders. Refinement-oriented probability for CSP. *Formal Aspects of Computing 8(9)*, 1996.

[11] J McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *IEEE Symposium on Research in Security and Privacy*, 1994.

[12] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[13] C. O'Halloran. A calculus of information flow. In *European Symposium on Research in Computer Security*, 1990.

[14] S. Pinsky. Absorbing covers and intransitive non-interference. In *IEEE Symposium on Research in Security and Privacy*, 1995.

[15] A. W. Roscoe. *The theory and practice of concurrency*. Prentice-Hall, 1997.

[16] A. W. Roscoe, J. Woodcock, and L. Wulf. Non-interference through determinism. In *ESORICS*, 1994.

[17] J. Rushby. Noninterference, transitiivity and channel-control security policies. Technical report, SRI, 1992.

[18] P. Y. A. Ryan. A CSP formulation of non-interference and unwinding. *Cipher*, 1991.

[19] P. Y. A. Ryan and S. A. Schneider. Process algebra and non-interference. In *Computer Security Foundations Workshop 12*, 1999.

[20] S. A. Schneider. *Concurrent and Real time systems: the CSP approach*. John Wiley, 1999.

[21] S. A. Schneider. Testing and abstraction. Technical Report TR-99-02, Royal Holloway, University of London, 1999.

[22] S. A. Schneider and A. Sidiropoulos. CSP and anonymity. In *ESORICS*, 1996.

[23] D. Sutherland. A model of information. In *9th National Computer Security Conference*, 1986.

[24] J. T. Wittbold and D. M. Johnson. Information flow in nondeterministic systems. In *Proceedings of the 1990 Symposium on Research on Security and privacy*, 1990.