The successes and failures of behavioural models

A.W. Roscoe, G.M. Reed and R. Forster

1 Introduction

Concurrency is a complex and fascinating subject, and the ways in which a system composed of many processes running in parallel and interacting with each other might behave would seem to most people to defy description. One of Tony Hoare's great insights is that it is possible to get useful models of a concurrent system by recording the set of all the behaviours that an experimenter with an appropriate degree of interest might record from single observations of it, such as *traces* or *failures*.

The vital principles required of one of these *behavioural models* are that they should convey useful information about the system being modelled – possibly or possibly not a 'complete' description of it – and must be *compositional*. This means that whenever a construct in our chosen language is used to build a process from syntactic subcomponents – such as putting P and Q in parallel to get $P \parallel Q$ – it must be possible to calculate the set of behaviours our experimenter might observe of the compound process from those of its parts.

What we will do in this paper is to examine the successes and failings of this semantic approach, comparing where appropriate with ones based on transition systems. A particularly telling test-bed is the subject of specifying (the lack of) information flow in computer security research, so in the last section we concentrate on that.

2 Traces, failures and beyond

Here we examine the basic properties of behavioural models. Lack of space prevents us from presenting many technical details here. The best general source for finding out more is [31], though for more specialised topics it is necessary to consult the references directly quoted.

Before we start to look at specific behavioural models, it is useful to describe the most frequently-used alternative, the *labelled transition system* (LTS). In this, a process is assumed to have some set of states V and a ternary transition relation $p \xrightarrow{x} q$ where $p, q \in V$ and x is some action that might occur as state p is transformed to state q. The action x might be externally visible and controllable by whoever is interacting with the process, or it might be invisible and uncontrollable. (The idea of an action that is invisible but controllable is ridiculous; the concept of one that the environment can observe but not prevent – a *signal* event – does make sense but for simplicity this is a case that we will be ignoring.) In most treatments there is a single invisible action, labelled τ.

Thinking of processes in this way is enormously attractive because it seems to give us a very clear understanding of how the execution of a process proceeds: we can see, and reason about, the individual steps. It is just about universally accepted that this model contains too much detail about how processes act if what we are trying to do is decide on whether a pair if them are 'essentially' the same. For example, different finite patterns of τ actions leading overall to the same range of eventual states would often be regarded as creating equivalent processes. Thus, raw transition system semantics are usually characterised as 'operational', and theories are built over them to decide on whether a pair of processes are equivalent or not. The best known example of this type of theory is Milner's CCS[18], which has spawned both a wide range of theories for analysing its operational semantics and many related process algebras.

Transition systems do impose a very particular intuition about what processes are. We might observe, for example, that they give a highly sequential view of what a process is: to all intents and purposes they identify processes with a type of state machine, and imply that actions are always temporally ordered with intermediate states, even when in reality the actions may be happening quite independently at distant corners of a parallel network at the same time. Debate has flourished for many years between people who prefer this style in which processes that are really concurrent become identified with sequential ones (the so-called *interleaving* style of semantics) and a variety of semantic styles in which this identification is not made (usually called *true concurrency*). Secondly, they make it natural to define operators over processes by what they do operationally rather than how they look to an outside observer.

The most striking example of the latter feature is the $+$ operator of CCS (which creates a choice between two processes). Even though the observational theory of CCS equates the processes P and $\tau.P$, the operational theory allows the τ to resolve the choice, and therefore $P + Q$ and $\tau.P + Q$ are not always observationally equivalent. Thus observational equivalence is not a congruence: a feature of CCS which complicates it (as well as the other languages that have borrowed this operator) greatly.

The corresponding formal intuition underlying Hoare's development of the form of CSP found in [15] was rather different: mathematical models based on observable process behaviours. These were the *traces* model of [14], and the *failures* model of [3] (later replaced by the *failures/divergences* model of [4] – see below). In each case a process was identified with the set of things a particular observer might see. In the traces model these are finite sequences $t \in \Sigma^*$ (where Σ is the set of *visible* actions and A^* are the finite sequences with members drawn from A). A process is then a nonempty, prefix-closed subset of Σ^*. (These are the *healthiness* conditions which state which sets of behaviours are reasonable pictures of processes.) The traces model $\mathcal{T}$ is then the set of all these processes.

It is remarkably straightforward to give the traces semantics of CSP: we simply give rules which show how, for example, to compute the traces of $P \parallel Q$ from those of P and Q. For all non-recursive operators of the standard language it is possible to do this by lifting one or more relations on individual traces. The parallel operator has just one relation: a ternary one which relates a trace of P and one of Q to one of $P \parallel Q$, but sometimes there are more. For example, with the prefixing operator $a \to P$, the most natural presentation has a unary relation showing that we get the empty trace $\langle\rangle$ without P performing any trace, and a binary relation mapping each trace s of P to $\langle a\rangle\hat{\ }s$.

The existence of this type of representation automatically gives all these operators properties of distributivity over nondeterministic choice (the operator $P \sqcap Q$, which is described in all behavioural models by the union of the behaviours of P and Q), and of continuity with respect to the direct-inclusion order on behaviour.

This continuity is fortunate because it is necessary to compute the semantic value of recursive processes defined both by simple equations like $P = a \to P$ and by complex mutual systems of equations. For unlike the case with transition systems, where defining recursion by an unwinding rule is trivial, in a behavioural model we always have to have some mathematical structure on the model (and the class of definable operators over it) which guarantees the existence of a fixed point. In the case of the traces model the theory is that of least fixed points of continuous functions over the complete lattice $(\mathcal{T}, \subseteq)$.

If traces were enough to give a complete picture of how concurrent systems behave then all would be easy. Unfortunately the phenomenon of *nondeterminism* is fundamental in concurrency, meaning that in any reasonably complete theory it is certain that we will be able to create processes which can take internal decisions (one manifestation of which in LTS's is the selection of which of a number of τ transitions to pick), and traces alone can neither detect nondeterminism nor distinguish between processes that *must* offer a particular event and ones that may choose not to. Hoare's solution here was to add as little information as possible onto traces to allow the detection of deadlock: a *failure* (s, X) is the combination of a trace and a set of events that the process might refuse (indefinitely) if offered after s. There are really two ways in which a process might indefinitely refuse a set of events X that has been offered to it: it might get stuck in a state where neither internal progress nor a member of X is possible, or it might engage in an infinite succession of internal events (behaviour which is termed *divergence*). After attempts to model CSP without special treatment of divergence were found to be subtly flawed, it proved necessary to include explicit treatment of divergence in the model, thus representing each process by its sets of failures and divergences (traces on which it can diverge). The resulting *failures/divergences* model has proved extremely successful.

It has a more complex set of healthiness conditions than the traces model, the most interesting of which are those which say that we do not record details of processes after potential divergence and the following, which says that impossible events are always refused:

$$(s, X) \in F \wedge Y \cap \{a \mid (s\hat{\ }\langle a\rangle, \{\}) \in F\} = \{\} \Rightarrow (s, X \cup Y) \in F$$

All behavioural models naturally model process refinement by reverse inclusion: $P \sqsubseteq Q$ if the behaviours of P include all those of Q. This is because each of the behaviours is only a possibility whose presence in the model does not mean we can rely on it in a given execution, and therefore is perhaps better thought of as a potential violation of some specification than as an asset. Having less potential behaviours is regarded as reducing the potential nondeterminism of the process.

The fact that the traces model has a process that is maximal under refinement ($STOP$) is further evidence that it does not give a complete picture of processes. It is mainly as a consequence of the above healthiness condition that the failures/divergence model has much more satisfying structure: its maximal elements are precisely the *deterministic* processes, which are divergence-free and satisfy

$$\forall\, s \in \Sigma^*.\, \forall\, a \in \Sigma. \neg((s, \{a\}) \in F \wedge (s\hat{\ }\langle a\rangle, \{\}) \in F)$$

This simply says that a deterministic process is one that never has the option of whether to offer an event or not. Provided we ignore the complications sometimes introduced to handle process termination, there is exactly one deterministic process for each member of the traces model: the kind implementation that can be guaranteed to progress through any one of its traces if offered the right events in turn.

While this behavioural style of modelling and the LTS style were quickly brought together by many connecting theories such as testing [5, 13], full abstraction and giving CSP congruent operational semantics (e.g. [2, 22]), the differences in modelling style have led to some interesting contrasts in how the theories have developed. One of the most obvious is in the choice operators: CSP does not give the same role to τ and therefore has separate operators for nondeterministic (i.e., internal) and external choice, namely $P \sqcap Q$ and $P \,\Box\, Q$, which are given different semantics for their failures on the empty trace.

It seems to make a great difference that in behavioural models a process is modelled directly by its semantic value rather than having to argue what equivalence relation to put over operational semantics. One of the most attractive consequences is that it leads to a completely natural and unproblematic theory of refinement (as discussed above), something which is often not true of other types of model.

The behavioural style of model has led to a greater emphasis on the application of "continuous" mathematics such as topology, order analysis and metric spaces (even though the structures they were applied to were usually relatively discrete when looked at carefully).

The first topological application was a recursion induction theorem for the failures divergences model based on contraction mappings over a complete metric space defined on processes [26] (developed further in, for example, [28]); such a theorem was needed since the Scott recursion induction theorem for complete partial orders requires that the specification be true of the bottom element (which in the failures divergences model is rarely desirable – the bottom element is the most nondeterministic and divergent process **div**). Recently in [17], new recursion induction theorems have been established for CSP which directly relate the partial orders to developable spaces (a class more general than metric spaces) in a natural and unexpected manner.

A second topological application for the behavioural semantics of CSP is found in the models for timed CSP [23, 24, 25]. In developing these models, it quickly became clear that complete partial orders were not appropriate. For example, there is no obvious "most nondeterministic" timed process, particularly if one wishes to postulate that a process can do only a finite number of events in a finite amount of time. Furthermore, there proved to be no least upper bound for increasing sequences arising from the natural candidates for partial orders. Fortunately, the concept of time itself provided a natural metric on the set of timed behaviours; the distance between two processes is simply expressed using the time for which they behave indistinguishably. By imposing a constant representing the minimum time necessary for the unwinding of any recursion, it was possible both to satisfy the finitary postulate above and to ensure that all recursions represented contraction mappings over the resulting complete metric space of behaviours. Hence, unique fixed points are guaranteed by the Banach fixed point theorem, and recursion induction is again accomplished under the metric approach.

Finally, the behavioural style of modelling CSP has led to models which are *neither* complete partial orders nor topological spaces. Roscoe established in [29] that unbounded nondeterminism in CSP would require such models. The appropriate models were presented in [29, 1, 20], and were based on the concept of "locally complete" partial orders. These models required the development of a new fixed point theory for domains, and have inspired considerable further theoretical work, e.g. [19].

A great advantage of the behavioural approach has been the possibility to reason about features of a CSP process using several of the above models simultaneously. Although involving metric spaces, a variety of partial orders, and new mathematical structures, the natural mappings between behaviours have provided useful theories for formal analysis and proof.

Many behavioural models have been created, encompassing time and infinite nondeterminism as discussed above, probability [16], processes with assignable state [27], different flavours and interpretations of of behaviour (two examples closely related to what we have seen are [21, 35], and there are many others) and combinations of these things. Of course, transition system models have been created for all these things too. Some of the literature could be said to be at the boundary between the two styles, since it relates to congruences (i.e., language preserving equivalences) over transition systems rather than having the primary aim of giving the semantics in the chosen model. The major technical difference here is that in this case there is not the same need to provide a (fixed point) semantics for recursion, which is therefore often not done. Applications of such congruences are then, of course, limited to recursion-free contexts.

The greatest semantic difference between transition systems and behavioural models appears in their ability (respectively inability) to see when a nondeterministic choice gets made. The processes $C_1 = a \rightarrow (P \sqcap Q)$ and $C_2 = (a \rightarrow P) \sqcap (a \rightarrow Q)$ will be equivalent in any behavioural model, assuming it does not see internal activity directly (or the time this takes), since on any *single* execution from the *initial* state of the processes the two processes have exactly the same possibilities. This is, of course, the distributivity of prefixing over nondeterministic choice – and, as described above, distributivity is often a natural consequence of the behavioural style.

There are strong arguments for not recording details of when these choices are made if it is unnecessary, and even stronger ones for using something like LTS's when it is necessary. It might be necessary for one of two reasons, both of which are rare. One of these, the possibility that the specification might need this level of detail, will be the subject of the next section. The other is if our language contains some operator which can do something like take a snapshot of a process in mid-flight and then compare several instances of this. Standard process algebras have no such operators, but they have appeared, for example, in many exercises. Imagine, for example, an operator $Checkpoint(P)$ which behaves like P except for two special events *snapshot* and *backtrack*, where communicating the latter has the effect of restarting P from the state it was in the last time *snapshot* occurred. Consider $Checkpoint(C_1)$ and $Checkpoint(C_2)$ in the context that *snapshot* was communicated after the initial a and subsequently enough has happened to allow the user to tell whether the nondeterminism has chosen P as opposed to Q. If *backtrack* then occurs in C_1, it might behave like either P or Q, since we do not know if the *snapshot* occurred after the choice or not. The same experiment on C_2, on the other hand, would be certain to lead to behaviour like P again, since we know the choice has already been made.

If forced to give semantics to an operator like *Checkpoint* over a behavioural model, all one can do is to give the most pessimistic (i.e., least refined) answer, which in the above case would be $Checkpoint(C_1)$. However one has to recognise that this is a true approximate answer in a sense that does not occur with the usual operators, for which exact operational congruence holds.

3 Modelling information flow

As we have begun to see in the first part of this paper, there are many subtle questions involved in deciding what is the right model for concurrent systems. Either fortunately or unfortunately, depending on your point of view, most practical specifications which arise in real life are fairly blind to the distinctions between these theories. With hardly an exception, in eight years of applications of the CSP refinement checker FDR [6] in a wide variety of practical examples, it has always been possible to formulate the desired correctness properties in the standard CSP behavioural models it supports. The great majority of these applications, in turn, reduced to simply checking that the proposed implementation *Impl* refined a process *Spec* representing the specification. Such *behavioural specifications* could equally well be formulated in any of the standard theories of transition systems. Thus, for most purposes, the question of which theory to use reduces to secondary (though important) considerations such as ease of expression and efficiency of implementation.

By far the most important exception to this rule that we know is in the area of computer security, when we attempt to analyse the potential for information flow across a process from one user to another. Imagine the alphabet of process P is partitioned into two sets H and L (respectively the interfaces of high and low level users with the same names), and that we are happy for what appears in the H events to be influenced by the decisions made by the user L, but do not wish L to be able to detect anything about what H has done from its interactions with P. This is an important topic as it relates to areas such as multi-level file systems and shared use of communication media and other resources by users with different security classifications. It has been the subject of an enormous amount of literature, much of it formulated in various process algebras. A small selection is [11, 34, 12, 33].

The concept of "security" is not one that we would wish to depend on the precise model used to give semantics to a process: it is, after all, unrealistic to assume that a spy is going to restrict his observations to those recorded in a specific model of concurrency. However, since it is clearly open to the spy to watch every detail of a process that a model records and to infer what he can about H's behaviour from these, it is difficult in general to satisfy this aim.

Most formulations of the virtually indistinguishable notions of *independence* and *noninterference* (both meaning lack of information flow) take the form of asserting that the behaviour visible to L does not depend on what H does. There is, of course, a lot of variation in how the details are put into this outline. It is easy to make subtle mistakes in doing this, most often by believing in one's model – and the way it handles nondeterminism – a little too much. One of the most common phenomena is the so-called *refinement paradox*, under which we can have $P \sqsubseteq Q$, with P satisfying an independence property and Q not. If $\sqsubseteq$ really does represent refinement as it usually understood, this is ridiculous as it is permissible for P always to behave like Q, namely insecurely.

It follows that whenever there is nondeterminism visible to L subsequent to a decision H has made, we cannot consider a process secure unless a lot more is known about how the nondeterminism is resolved than we usually choose to know. We will see some more examples later that will re-emphasise this.

If the only potential source of nondeterminism visible to L is high-level behaviour then the situation is easy: if an appropriately formulated low-level view of the process (i.e., with the high-level behaviour abstracted away) is nondeterministic then there is a mechanism for H to pass information to L through P. On the other hand, if it is deterministic then whatever H does has no effect on what L sees, so no information can flow (at least through observations supported by the model in use). In other words, if P is *locally deterministic* in L (implied by full determinism of P) then it is independent if and only if $\mathcal{L}_H(P)$ (the result of applying an operator that turns all action or inaction by H into internal nondeterministic choices) is itself deterministic.

The failures/divergences model has considerable advantages when it comes to specifying all of the important terms in this statement. We need its extensional formulation that a process is deterministic if it is divergence free and can, after no trace s both accept and refuse any event a (or an event a in L for local determinism in L). The most natural definition of "determinism" in LTS models is that of a deterministic state machine, which is too restrictive since the essence of the security conditions are that turning all high-level choices into nondeterminism within the LTS somehow manages to preserve extensional determinism (i.e., every choice apparently introduced leads to equivalent processes).

The failures/divergences model also provides an ideal vehicle for describing the form of abstraction required: the *lazy abstraction* of the events H in a divergence-free, finitely nondeterministic process P is the process $\mathcal{L}_H(P)$ which behaves like $P \setminus H$ (the process where events from H are turned into τ actions) except that we cannot rely on the internalised H actions occurring (as the high-level agent may opt to do nothing). Its failures are

$$\{(s \setminus H, X) \mid (s, X \cap L) \in \mathit{failures}(P)\}$$

as opposed to those of $P \setminus H$:

$$\{(s \setminus H, X) \mid (s, X \cup H) \in \mathit{failures}(P)\}$$

Remarkably, the structure of the model – specifically the maximality under refinement of deterministic processes – provides the key to an efficient decision procedure for whether a process P is deterministic or not. We simply pick (using LTS determinisation) an arbitrary deterministic refinement P' of P and test to see if $P' \sqsubseteq P$. This is true if and only if P was deterministic (and hence failures/divergences equivalent to P').

The definition that *lazy independence* holds if $\mathcal{L}_H(P)$ is deterministic is still safe even when P is not locally deterministic in L, but may well give false negative results. It regards all nondeterminism of this process as potentially conveying information about H, even though it may not. This is the formulation of security proposed in [32] with further development in [30, 31, 36], for example.

There is a high degree of agreement that this is the right definition in the class $\mathcal{LD}$ of processes that are locally deterministic in L – because many other formulations, whether in terms of behavioural models [30] or transition systems [8, 10] all coincide with it in this case. The situation when we step outside this area becomes much more controversial, however.

Consider the process $LEAK \sqcap Chaos_L$, which can opt either to behave like the arbitrary and potentially leaky process $LEAK$ or to behave like $Chaos_L$, which never communicates with H at all but behaves like the most nondeterministic divergence-free process in the alphabet L. (A convenient example process to imagine for $LEAK$ is one like $LEAK = high?x \to low!x \to LEAK$.) Do you think this process is secure, and would it be any more secure if $Chaos_L$ were replaced by $Chaos_{L\cup H}$? In both cases one can argue that L can infer nothing definite about what H has chosen to do since the process can opt to behave like *Chaos* and show anything to L off its own bat. On the other hand we may choose (under the conventional understanding of nondeterminism in CSP) to implement this process as *LEAK*, and even if we do not it *may* behave like *LEAK*. In the first case ($Chaos_L$) it is certain that anything H actually gets to communicate is transmitted straight to L, and in either case it might be reasonable for L to infer that, if what he sees is making sense, the nondeterministic choice was probably resolved in the direction of *LEAK*. We believe it is very dangerous to regard either of these processes as secure: this is all the more confusing in the second case, since this process is actually failures/divergences *equivalent* to

$$Chaos_{H\cup L} = (?x : H \cup L \rightarrow Chaos_{H\cup L}) \sqcap STOP$$

a process with no mechanism for transmitting information from high to low.

This certainly illustrates that the failures/divergences model is not adequate to tell secure processes from insecure ones outside the class $\mathcal{LD}$. Both examples illustrate the problems that can be caused by confusing the sort of nondeterminism created by H with intrinsic nondeterminism, and by identifying nondeterministic choices made immediately (as the crucial choice is made in both these examples) with ones made subsequently. One of the major questions it asks is whether the possibility that a nondeterministic choice might lead the process into a disastrously insecure state, like the specific *LEAK* quoted above, should or should not automatically condemn the entire process as insecure. Natural caution suggests to us that it should.

The obvious solution is to look for definitions of independence over LTS's, because these give finer equivalences than the failures/divergences model, and in particular allow us to see exactly when a nondeterministic choice gets made. This is what Focardi and Gorrieri did (expressed in a modified form of CCS) [7]. They give a wide range of security conditions, and show that in what is essentially the case $\mathcal{LD}$ that their main conditions (including the ones quoted below) agree with lazy independence. We do not have space here to quote many of their definitions or results, but it is fair to say that they regard their primary definition of independence as the following condition BNDC (standing for *bisimulation nondeducibility on compositions* and expressed in their modified CCS):

$$\forall \Pi \in \mathcal{E}_H . P \setminus H \cong_B (P \mid \Pi)\backslash_H$$

This says that the view of the low-level agent is independent of what the high-level one chooses to behave like ($\mathcal{E}_H$ being the set of all possible high level agents), equivalence being judged using weak bisimulation equivalence [18] (which for divergence-free processes is much stronger than failures/divergences equivalence).[1] Since this is squarely in accordance with the outline definition of independence given earlier,

[1] $P\backslash_H$ is a restriction operator: it means "P with events from H prevented" as distinct from $P \setminus H$, where they are turned into τ's and thus hidden.

all seems well. Unfortunately, however, it is possible to construct something very like $LEAK \sqcap Chaos_L$ which puts it very much in question: for any process P, the process

$$P \sqcap \bigsqcap\{(P \mid \Pi)\backslash_H \mid \Pi \in \mathcal{E}_H\}$$

(expressed, for convenience, in a mixture of CSP and CCS[2]) satisfies BNDC even though it can choose to behave like the arbitrary process P and certainly does whenever H does anything.

Yet again, we can guarantee here that if H has been allowed to do anything at all by the process then it has behaved insecurely. The root causes of this problem are again the difficulties caused by the different sources, and decision points of, nondeterminism. It was certainly much easier to create this counter-example because BNDC looks at multiple views of the initial state of the process rather than demanding that something appropriate holds of each state that P can reach.

It is interesting that Focardi and Gorrieri introduce a condition stronger than BNDC which does have this latter form. This is the condition SBSNNI (*strong bisimulation strong nondeterministic noninterference*), whose main motivation is that it implies, and is easier to decide than, BNDC:

$$\forall P'.(\exists s.P \stackrel{s}{\Longrightarrow} P') \Rightarrow P' \setminus H \cong_B P'\backslash_H$$

In other words, all reachable states P' of P behave equivalently with H actions hidden and, respectively, prevented. The extra strength of this condition, and in particular the "for all states" form, makes it more difficult to find arguably insecure processes satisfying it. However because it does not distinguish between different H events there are still examples in which L can distinguish which – if any – H action has occurred at some point. For example the process shown in Figure 1 – drawn as an LTS for clarity – satisfies it. Here each bit that H communicates is passed to L amongst some randomly created noise. One might reasonably expect L to be able to tell the difference between what he sees when H constantly presses the 0 key as opposed to the 1 key. The same example would, of course, work with a much larger type than bits, in which case an intelligent L might have a reasonable expectation on seeing a coherent message appearing that it was much more likely to result from H rather than noise.

Forster [9], having discovered these problems, goes on to introduce two conditions which take as their basic thesis that each individual state of an implementation has to be examined for security. The simpler of these is *strong local noninterference*, which P satisfies when all its reachable states P' have

$$h \in H \wedge P' \stackrel{h}{\longrightarrow} P'' \Rightarrow P'\backslash_H \cong_B P''\backslash_H$$

[2] The finer equivalences of CCS make the concept of a universal most nondeterministic process on an alphabet like $Chaos_L$ problematic, but the construction above produces essentially the same result on a case-by-case basis. Depending on the value of P, the number of $\cong_B$ equivalence classes of $(P \mid \Pi)\backslash_H$ as Π varies might or might not be finite – if it is finite then an equivalent process could be created without the infinitary nondeterministic choice used above.

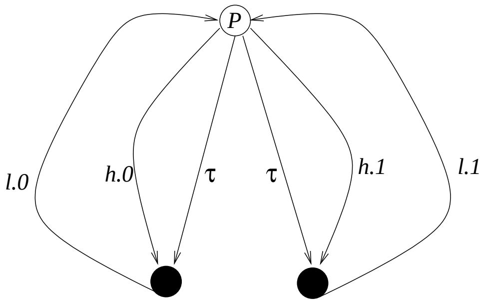

Figure 1: Process that satisfies SBSNNI

In other words, each individual H action has no effect on what L can do to P thereafter. The difference with the weaker condition *local noninterference* is that when a given P' has several transitions with the same h label, these are aggregated on the basis that H has no control over which of this set of transitions is followed:

$$h \in H \wedge P' \xrightarrow{h} P'' \Rightarrow P' \backslash_H \cong_B \bigsqcap \{P'' \backslash_H \mid P \xrightarrow{h} P''\}$$

As shown in [10], where they are examined in detail, both these conditions agree on $\mathcal{LD}$ with lazy independence, imply SBSNNI and hence BNDC, and find all the potential security leaks in examples we have found.

It is interesting to note that in order to get what seem to be satisfactory conditions over transition systems we have been forced into direct examination of the transition structure, as opposed to the pleasing abstraction of BNDC and lazy independence. This has another effect: the new conditions rely on a literal interpretation of what an LTS is, for example that each time we reach a state with nondeterminism in it (τ branching or multiple actions with label h) then each choice is one that might actually be chosen. There must be no mechanisms that resolve this nondeterminism hidden in the abstraction process. Since, as discussed earlier, the LTS model usually is best regarded an abstraction of some more concrete operational model, this issue must be given attention in any application.

Because of the reliance that these conditions place on the range of possible behaviours in an LTS definitely being possible on every run, they are inconsistent with any notion of refinement. In order for these conditions to make sense, our idea of what nondeterminism is has to become a lot narrower than usual.

The more one thinks about it, the more one realises that one's view of whether some processes outside $\mathcal{LD}$ are secure or not depends crucially on one's exact assumptions about how nondeterministic choice is resolved and how much the processes themselves can see this mechanism. Consider, for example, the pair of processes, in which all the data values are bits and $\oplus$ is exclusive or

$$\begin{aligned} XOR1 &= (high?x \to low!x \to STOP) \sqcap (high?x \to low!x \oplus 1 \to STOP) \\ XOR2 &= high?x \to (low!x \to STOP) \sqcap (low!x \oplus 1 \to STOP) \end{aligned}$$

Even though standard CSP equivalences would make these two processes equivalent, there is a potential channel from high to low which is present in $XOR1$ but not $XOR2$: if H could determine somehow which way the nondeterministic choice in $XOR1$ had been resolved, he could decide what to output on *high* in order to achieve the desired output to L, an option not available on $XOR2$ where the nondeterminism is not resolved until after he makes his choice. (Both these processes have the additional insecurity that L will in any case know that H has done *something* when communication appears on *low*. The same issue can be reproduced in otherwise secure processes, but this would lose the simplicity of $XOR1$ and $XOR2$.) The local noninterference properties (and to some extent SBSNNI) will catch this type of potential leak and some which are even more subtle and rely on L being able to do things similar to those seen in the *Checkpoint* operator in the last section. Whether or not these conditions are thereby over-pessimistic or correctly vigilant will depend on the answers to questions we usually prefer not to ask about the operation and observation of transition systems.

All the conditions we have seen giving definitions of security outside $\mathcal{LD}$ have essentially the same intent: whatever H does, the range of nondeterministic options on what L sees should not change. This does, of course, assume that whatever demon resolves this nondeterminism is benign (does not base its choices on what H has done). They are *possibilistic*, as opposed to *probabilistic* definitions because they do not take account of any changing probability distributions amongst the nondeterminism. There are strong arguments for wanting to incorporate this type of reasoning into noninterference analysis, but as yet we are not aware of any model of concurrency which simultaneously deals with both probabilistic and non-probabilistic nondeterminism sufficiently flexibly to achieve this.

Our conclusion, then, is that the behavioural models provide the most natural way of identifying the case in which specifying noninterference is relatively uncontroversial and of specifying it in that case. They are, however, far too weak (thanks to the same distributive properties over nondeterminism which caused problems with *Checkpoint*) to address the issue outside this range, but if one does choose to step outside it then extreme care is required. We think it is also true that the subject of noninterference provides the best practical arena for exploring the differences between ways of modelling concurrent systems, since the nuances which make little difference elsewhere seem to loom large when discussing information flow.

References

[1] G. Barrett. "The fixed-point theory of unbounded nondeterminism", *Formal Aspects of Computing*, **3**, 110–128, 1991.

[2] S. D. Brookes. "A Model for Communicating Sequential Processes", *DPhil, Oxford University* (1983). (Published as a Carnegie-Mellon University technical report.)

[3] S. D. Brookes, C. A. R. Hoare and A. W. Roscoe. "A Theory of Communicating Sequential Processes", *Journal of the ACM* **31:3** (1984), 560–599.

[4] S. D. Brookes, A. W. Roscoe. "An Improved Failures Model for CSP", *Proceedings of the Pittsburgh Seminar on Concurrency, Springer LNCS 197* (1985).

[5] R. de Nicole and M. Hennessy. "Testing Equivalences for Processes", *Theoretical Computer Science* **34:1** (1987), 83–134.

[6] Formal Systems (Europe) Limited. "Failures-Divergences Refinement: User Manual and Tutorial", *Available by FTP from Oxford University.*

[7] R. Focardi, R. Gorrieri. "An Information Flow Security Property for CCS", *Proceedings of Second North American Process Algebra Workshop (NAPAW'93)* (August 1993).

[8] R. Focardi. "Comparing Two Information Flow Security Properties", *Proceedings of 9th IEEE Computer Security Foundations Workshop (CSFW'96)* (June 1996), 116–122.

[9] R. Forster. "Non-interference Properties for Nondeterministic Systems", *Dissertation for transfer to DPhil status, Oxford University Computing Laboratory*, 1997.

[10] R. Forster. "Non-interference Properties for Nondeterministic Systems", *DPhil, Oxford University* (1999).

[11] J. A. Goguen, J. Meseguer. "Security Policies and Security Models", *IEEE Symposium on Security and Privacy* (1982), 11–20.

[12] J. Graham-Cumming. "The Formal Development of Secure Systems", *DPhil, Oxford University* (1992).

[13] M. Hennessy. "Algebraic Theory of Processes", MIT Press (1988).

[14] C. A. R. Hoare. "A Model for Communicating Sequential Processes", *On the Construction of programs* (McKeag and MacNaughten, editors), Cambridge University Press (1980).

[15] C. A. R. Hoare. "Communicating Sequential Processes", Prentice Hall (1985).

[16] G. Lowe "Probabilistic and prioritized models of timed CSP", Theoretical Computer Science **138:2** pp315–352 (1995).

[17] K. Martin and G. M. Reed. "Measurements and Models in Domain Theory", *to appear.*

[18] R. Milner. "Communication and Concurrency", Prentice Hall (1989).

[19] M. W. Mislove. "Denotational models for unbounded nondeterminism", *Proceedings of MFPS XI*, Electronic Notes in Theoretical Computer Science, Vol 1 (1995).

[20] M.W. Mislove, A.W. Roscoe and S.A. Schneider. "Fixed points without completeness", *Theoretical Computer Science* **138**, 2, 273–314, 1995.

[21] A. Mukarram. "A refusal testing model for CSP", Oxford University D.Phil. thesis (1993).

[22] E. R. Olderog and C. A. R. Hoare. "Specification-Oriented Semantics for Communicating Processes", *Acta Informatica* **23** (1986) 9–66.

[23] G. M. Reed. "A Hierarchy of Models for Real-Time Distributed Computing", *Proceedings of the Fifth Workshop on the Mathematical Foundations of Programming Language Semantics, LNCS 442* (April 1989), 80–128.

[24] G. M. Reed and A. W. Roscoe. "Metric spaces as models for real-time concurrency" *Proceedings of MFPS 1987*, Springer LNCS 298, 1987.

[25] G. M. Reed and A. W. Roscoe. "The timed failures-stability model for CSP", *Theoretical Computer Science* **211** pp85-127 (1999).

[26] A. W. Roscoe. "A Mathematical Theory of Communicating Processes", *DPhil, Oxford University* (1982).

[27] A. W. Roscoe. "Denotational semantics for OCCAM", *Proceedings of the Pittsburgh Seminar on Concurrency, Springer LNCS 197* (1985).

[28] A. W. Roscoe. "Topology, computer science and the mathematics of convergence", *Topology and category theory in computer science*, OUP (1991).

[29] A. W. Roscoe. "Unbounded Nondeterminism in CSP", *Two papers on CSP, Technical Monograph PRG-67, Oxford University Computing Laboratory* (July 1988). *Journal of Logic and Computation* **2:5** (1992), 557–577.

[30] A. W. Roscoe. "CSP and Determinism in Security Modelling", *1995 IEEE Symposium on Security and Privacy* (1995), 114–127.

[31] A. W. Roscoe. "The Theory and Practice of Concurrency", Prentice Hall (1998).

[32] A. W. Roscoe, J. C. P. Woodcock, L. Wulf. "Non-interference through Determinism" *ESORICS 94, Springer LNCS 875* (1994), 33–53.

[33] P. Y. A. Ryan, S. A. Schneider. "Process Algebra and Non-interference", *12th IEEE Computer Security Foundation Workshop* (28–30 June 1999), 214–227.

[34] D. Sutherland. "A Model of Information", *Proceedings of the 9th National Computer Security Conference* (September 1986), 175–183.

[35] A. Valmari and M. Tienari. "An Improved Failures Equivalence for Finite-State Systems with a Reduction Algorithm", *Protocol Specification, Testing and Verification XI, North-Holland* (1991).

[36] L. Wulf. "Interaction and Security in Distributed Computing", *DPhil, Oxford University* (1997).