Semantic Models for Information Flow

Gavin Lowe*

April 22, 2004

Abstract

In the past, several definitions of information flow have been presented, based upon process algebras. Unfortunately, these appear to all be either too weak—failing to identify certain subtle forms of information flow—or too strong—indicating information flow when there is none. In this paper, we produce a definition that aims to overcome these shortcomings. We base our definition upon an operational model of CSP that reasons about the ways in which nondeterministic choices can be resolved, and so is more discriminating than previous models. Our definition of information flow is then that the behaviour of one agent can have some influence upon another agent's view of the system. This definition gives the expected results on all thought experiments tried to date, and also satisfies certain desirable properties.

1 Introduction

The question of *information flow* is of central importance in theoretical studies of computer security. In its simplest form, it considers two users, *High* and *Low* interacting with the same computer system, and asks if there is any flow of information from *High* to *Low*; in other words, can *Low*'s view of the system alter, depending upon *High*'s behaviour? This is variously known as *noninterference* (can *High*'s behaviour interfere with *Low*'s view of the system?), *invariance* (does *Low*'s view of the system vary as a result of *High*'s behaviour?), *nondeducibility* (can *Low* deduce anything about *High*'s behaviour?) or *independence* (is *Low*'s view of the system independent of *High*'s behaviour?).

The normal motivating application for these questions is multi-level security, where information flows should not occur from a user *High* with a high level security clearance to a user *Low* at a lower security level. Roscoe [Ros97] identifies two possible scenarios:

*Author's address: Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford, OX1 3QD, UK; e-mail: gavin.lowe@comlab.ox.ac.uk.

- *Low* is a spy who is trying to find out about *High*'s behaviour without *High*'s knowledge;
- *High* is a "mole" who is trying to pass information to *Low*, possibly using some pre-arranged scheme for representing information.

It turns out that these are subtly different; in this paper, we shall be mostly considering the latter scenario.

Several authors have attempted to formalise this notion of information flow using process algebras, typically CSP. Unfortunately, most previous attempts have produced definitions that appear to be either too weak (failing to identify certain sources of information flow) or too strong (identifying processes as insecure when there seems to be no way of using them to pass information).

In this paper we attempt to improve the situation, by producing a new definition of information flow. We believe that one of the reasons for the shortcomings of previous definitions is that they are based upon semantic models that do not make enough discriminations between processes, and in particular do not give enough information about the ways in which nondeterministic choices are resolved. We therefore have to introduce a new semantic model, which makes more distinctions.

In the next section we give a brief overview of the syntax and traditional semantic models of CSP. We then review some previous work, in Section 3, thereby helping to illustrate the idea of information flow, and also to explain some of the shortcomings that we hope to avoid.

In Section 4 we give an informal description of the operational semantic model we will be using. We formalise this operational model in Section 5, and then extract denotational information from the model in Section 6. In Section 7 we formalise our notion of security and consider some examples. We prove some results about our definition in Section 8. We sum up in Section 9.

2 A brief overview of CSP

In this section we give a brief overview of CSP. More details can be obtained from [Hoa85, Ros97].

An event represents an atomic communication; this might either be between two processes or between a process and the environment. Channels carry sets of events; for example, $a.x$ is an event of channel a. We write Σ for the set of all visible events, and define

$$\Sigma^\tau \;\widehat{=}\; \Sigma \cup \{\tau\}$$

where τ represents an internal event. We write αP for the alphabet of P (i.e. the set of events P can perform).

2.1 Syntax and operational semantics

We will consider the subset of CSP defined by:

$$\begin{aligned} P \in \mathit{CSP} \ ::= \ & \mathit{STOP} \mid a \to P \mid P \sqcap P \mid P \,\Box\, P \mid \\ & P \setminus A \mid P\backslash_A \mid P \underset{A}{\parallel} P \mid \mu\, X \bullet P. \end{aligned}$$

In the paragraphs below we describe each of the CSP syntactic constructs, and give an operational semantics to the language. We write

$$P \xrightarrow{a} Q$$

to mean that the process P can perform the event a, and evolve into the process Q.

STOP The process *STOP* can perform no events. It therefore has no transition rules.

Prefixing $a \to P$ is the process that can perform the event a, and then act like P:

$$a \to P \ \xrightarrow{a} \ P.$$

Nondeterministic choice $P \sqcap Q$ represents an internal or nondeterministic choice; the process can act like either P or Q, with the choice being made according to some criterion that we do not model. In the operational semantics, the resolution of the nondeterministic choice is represented by a τ-transition:

$$P \sqcap Q \ \xrightarrow{\tau} \ P, \qquad\qquad P \sqcap Q \ \xrightarrow{\tau} \ Q.$$

External choice $P \,\Box\, Q$ represents an external or environmental choice. The process offers the environment a choice between P and Q. If a visible event of either process is performed, then that resolves the choice:

$$\frac{P \xrightarrow{a} P'}{P \,\Box\, Q \xrightarrow{a} P'}\ a \neq \tau \qquad\qquad \frac{Q \xrightarrow{a} Q'}{P \,\Box\, Q \xrightarrow{a} Q'}\ a \neq \tau$$

However, the choice is not resolved by internal transitions:

$$\frac{P \xrightarrow{\tau} P'}{P \,\Box\, Q \xrightarrow{\tau} P' \,\Box\, Q} \qquad\qquad \frac{Q \xrightarrow{\tau} Q'}{P \,\Box\, Q \xrightarrow{\tau} P \,\Box\, Q'}$$

Hiding $P \setminus A$ is the process that acts like P, except all events from the set A are hidden, i.e. made internal. The operational semantics replaces each event of P from A by the internal event τ:

$$\frac{P \xrightarrow{a} P'}{P \setminus A \xrightarrow{\tau} P' \setminus A}\ a \in A \qquad \frac{P \xrightarrow{a} P'}{P \setminus A \xrightarrow{a} P' \setminus A}\ a \notin A$$

Blocking $P\backslash_A$ is the process that acts like P, except all events from the set A are blocked, i.e. cannot occur:

$$\frac{P \xrightarrow{a} P'}{P\backslash_A \xrightarrow{a} P'\backslash_A}\ a \notin A$$

If P can perform only events from A, then $P\backslash_A$ deadlocks.

Parallel composition $P \underset{A}{\|} Q$ represents a parallel composition of P and Q, synchronising on events from A. If one component of the parallel composition can perform an event not in the synchronisation set, then so can the parallel composition:

$$\frac{P \xrightarrow{a} P'}{P \underset{A}{\|} Q \xrightarrow{a} P' \underset{A}{\|} Q}\ a \notin A \qquad \frac{Q \xrightarrow{a} Q'}{P \underset{A}{\|} Q \xrightarrow{a} P \underset{A}{\|} Q'}\ a \notin A$$

If both sides of the parallel composition can perform an event in the synchronisation set, then the parallel composition can perform that event:

$$\frac{\begin{array}{l} P \xrightarrow{a} P' \\ Q \xrightarrow{a} Q' \end{array}}{P \underset{A}{\|} Q \xrightarrow{a} P' \underset{A}{\|} Q'}\ a \in A$$

Recursion $\mu\, X \bullet P$ represents a recursive process, that acts like P with occurrences of X representing recursive executions. The operational semantics represents the unwinding of the recursion by a τ-transition:

$$\mu\, X \bullet P \xrightarrow{\tau} P[\mu\, X \bullet P/X].$$

Syntactic sugar Below we define a few pieces of syntactic sugar.

If c is a channel carrying values from the finite set X, then we define $c?x \rightarrow P(x)$ as syntactic sugar for $\mathop{\Box}_{x:X} c.x \rightarrow P(x)$ (where $\Box$ is a replicated version of the external choice operator); this represents a process that can input a value x on channel c, and then act like $P(x)$.

We define $P \mid\mid\mid Q$ as syntactic sugar for $P \underset{\{\}}{\parallel} Q$; this represents an interleaving, i.e. a parallel composition without synchronisation.

If $\alpha P \subseteq A$, $\alpha Q \subseteq B$, then we define $P \;{}_A\|_B\; Q$ as syntactic sugar for $P \underset{A \cap B}{\parallel} Q$; this represents a parallel composition where the alphabets of each component are given explicitly, and the processes synchronise on common events.

We define $P \rhd Q$ as syntactic sugar for $(P \,\Box\, a \rightarrow Q) \setminus a$ where a is a fresh event; this represents a timeout.

We define $CHAOS(A)$ by

$$CHAOS(A) \;\widehat{=}\; \left(\textstyle\bigsqcap_{a \in A} a \rightarrow CHAOS(A)\right) \sqcap STOP$$

(where $\bigsqcap$ is a replicated version of the nondeterministic choice operator); this is the most nondeterministic non-divergent process with alphabet A.

2.2 Multi-step transition rules and denotational semantics

Following Roscoe [Ros97, page 170], we define two multi-step transition relations. The first such relation includes internal τ actions. If $s = \langle a_0, a_1, \ldots, a_{n-1} \rangle \in (\Sigma^\tau)^*$ is a sequence of (possibly internal) actions, then we write $P \stackrel{s}{\longmapsto} Q$ if there exist $P_0 = P$, P_1, $\ldots$, $P_n = Q$ such that $P_k \xrightarrow{a_k} P_{k+1}$ for each $k \in \{0, \ldots, n-1\}$.

The next multi-step transition relation is in terms of visible events, only. We write $P \stackrel{tr}{\Longrightarrow} Q$ if there exists s such that $P \stackrel{s}{\longmapsto} Q$ and $s \setminus \{\tau\} = tr$. We adopt the convention of writing tr, tr', etc. for traces of visible events, and s, s', etc. for traces possibly including internal events.

We write $traces(P)$ for the set of traces of visible events that P can perform:

$$traces(P) \;\widehat{=}\; \{tr \mid P \stackrel{tr}{\Longrightarrow}\}.$$

A process is *stable* if it can perform no internal events:

$$stable\, P \;\widehat{=}\; \neg P \xrightarrow{\tau}.$$

A process *refuses* X if it is stable and can perform none of the events of X:

$$P \; ref \; X \;\Leftrightarrow\; stable\, P \wedge \forall\, x \in X \bullet \neg P \xrightarrow{x}.$$

A process has *stable failure* (tr, X) if it can perform trace tr to evolve into a state where it can refuse X; we write $failures(P)$ for the set of stable failures of P:

$$failures(P) \;\widehat{=}\; \{(tr, X) \mid \exists\, P' \bullet P \stackrel{tr}{\Longrightarrow} P' \wedge P' \; ref \; X\}.$$

A process *diverges* immediately if it can perform an infinite sequence of internal events:

$$div\, P \;\Leftrightarrow\; \exists\, P_0 = P, P_1, \ldots \bullet \forall\, n \in \mathbb{N} \bullet P_n \xrightarrow{\tau} P_{n+1}.$$

A trace *tr* is a *divergence* of P if P can perform some prefix of *tr* and then diverge:

$$divergences(P) \;\widehat{=}\; \{tr^\frown tr' \mid \exists\, P' \bullet P \stackrel{tr}{\Longrightarrow} P' \wedge div\, P'\}.$$

The failures of a process are the stable failures, along with all (tr, X) pairs for *tr* a divergence; in other words, the failures include all possible behaviours following a divergence, which is the traditional approach, as in [Hoa85].

$$failures^{\perp}(P) \;\widehat{=}\; failures(P) \cup \{(tr, X) \mid tr \in divergences(P)\}.$$

Finally, we define refinement relations for the stable failures and failures-divergences models:

$$\begin{aligned} P \sqsubseteq_F Q \;&\widehat{=}\; traces(P) \supseteq traces(Q) \wedge failures(P) \supseteq failures(Q), \\ P \sqsubseteq_{FD} Q \;&\widehat{=}\; failures^{\perp}(P) \supseteq failures^{\perp}(Q) \wedge \\ &\phantom{\widehat{=}}\;\; divergences(P) \supseteq divergences(Q). \end{aligned}$$

3 Related work

In this section we review some previous work, thereby helping to illustrate the idea of information flow, and also to explain some of the shortcomings that we hope to avoid.

In all our examples, we will take the user *Low* to have alphabet L, containing events with names like l; and we will take the user *High* to have alphabet H, containing events with names like h. We assume $H \cap L = \{\}$, and take $\Sigma = H \cup L$.

3.1 Noninterference

One of the earliest attempts to formalise information flow was by Goguen and Meseguer [GM82], and called *noninterference*. Their formalism made use of deterministic state machines, in a setting where all operations were always available.

Informally, a state machine P is noninterfering if for all traces *tr* of operations (from some set Σ), and for all low-level operations l (from some set $L \subseteq \Sigma$), the result obtained by *Low* from l after *tr* is the same as he would have obtained had no high-level operations been performed. In other words, the operations performed by *High* during *tr* have had no effect upon (have not interfered with) *Low*'s view of the system.

This notion can be captured formally as follows:

Definition 1 A state-machine satisfies *noninterference* if:

$$\forall\, tr : \Sigma^* \,;\, l : L \bullet result(P\ \mathsf{after}\ tr, l) = result(P\ \mathsf{after}\ (tr \restriction L), l).$$

Here P after tr is the state P reaches following trace tr; $result(P', l)$ is the result obtained from operation l in state P'; and $tr \restriction L$ is the restriction of tr to low-level operations.

A limitation of this formalism is that it applies only to deterministic systems.

3.2 Nondeducibility

In [Sut86], Sutherland gives a definition of information flow termed *nondeducibility*. The idea is that *Low* should not be able to deduce anything about *High*'s input behaviour. The formalism uses nondeterministic state machines, in a setting where outputs cannot be blocked.

Sutherland's definition is rather abstract. He defines an *information function* to be a function over system traces. Given information functions f_1 and f_2, he defines information to flow from f_1 to f_2 if there is some trace tr, and some value z that is achieved by f_2 on some system trace, such that

$$\forall\, tr' \bullet f_1(tr') = f_1(tr) \Rightarrow f_2(tr') \neq z.$$

In other words, if A_1 sees only the results of f_1, and A_2's behaviour is captured by f_2, then if A_1 sees $f_1(tr)$, he can deduce that A_2 has not performed z.

This is specialised to the question of information flow in multi-level security systems by taking $f_1(tr) \mathrel{\widehat=} tr \restriction L$ and $f_2(tr) \mathrel{\widehat=} tr \restriction HI$, where HI is the set of input events for *High*.

In [All91], Allen translates this condition into CSP as follows.

Definition 2 Define $traces_L(P)$ and $traces_{HI}(P)$ to be the sets of traces of low-level events and high-level inputs, respectively:

$$\begin{aligned} traces_L(P) &\mathrel{\widehat=} \{tr \restriction L \mid tr \in traces(P)\}, \\ traces_{HI}(P) &\mathrel{\widehat=} \{tr \restriction HI \mid tr \in traces(P)\}. \end{aligned}$$

A process satisfies nondeducibility if for every pair of such traces tr_L and tr_{HI}, there is a trace of the system compatible with both tr_L and tr_{HI}:

$$\begin{aligned} &\forall\, tr_L : traces_L(P) \,;\, tr_{HI} : traces_{HI}(P) \bullet \\ &\quad \exists\, tr : traces(P) \bullet tr \restriction L = tr_L \wedge tr \restriction HI = tr_{HI}. \end{aligned}$$

In other words, if *Low* observes tr_L, he cannot deduce that *High* has not performed input behaviour tr_{HI}.

This definition is quite strong, as demonstrated by the following example:

Example 1 Let

$$P \mathrel{\widehat=} l \rightarrow STOP \sqcap h \rightarrow STOP.$$

There would seem to be no way for *High* to use P to pass information to *Low*. However, P does not satisfy nondeducibility, because $\langle l \rangle \in traces_L(P)$, $\langle h \rangle \in traces_{HI}(P)$, but there is no trace compatible with both. $\square$

In [WJ90], Wittbold and Johnson exhibit a weakness of this definition, making use of the fact that it ignores high-level outputs. Sutherland worked in a synchronous model, but the same weakness is apparent in an asynchronous setting; we translate their example into CSP, for ease of exposition.

Example 2 Let *High* have input channel h_{in} carrying values $\{q, 0, 1\}$; and let *High* and *Low* have output channels h_{out} and l_{out}, respectively, each carrying values $\{0, 1\}$. The example process is parameterised by two keys $k_1, k_2 : \{0, 1\}$, both of which are initialised nondeterministically.

$$\begin{array}{rcl} P(k_1, k_2) & = & h_{in}.q \rightarrow h_{out}.k_1 \rightarrow l_{out}.k_2 \rightarrow \sqcap_{k_2':\{0,1\}} P(k_1, k_2') \\ & & \Box \\ & & h_{in}?x : \{0,1\} \rightarrow l_{out}.(x \oplus k_1) \rightarrow \sqcap_{k_1',k_2':\{0,1\}} P(k_1', k_2'). \end{array}$$

If *High* inputs q then he receives k_1, *Low* receives k_2, and k_2 is updated nondeterministically; if *High* inputs $x \in \{0, 1\}$ then *Low* receives $x \oplus k_1$, where $\oplus$ represents exclusive-or, and both k_1 and k_2 are updated nondeterministically.

It can be shown that this process satisfies nondeducibility: for every sequence of high-level inputs, *Low* can receive an arbitrary sequence of outputs. But there is a simple strategy that allows *High* to pass information: to pass the bit b, *High* inputs q to learn the value k_1 (*Low* receives k_2, which he ignores), then inputs $k_1 \oplus b$, causing *Low* to receive $(k_1 \oplus b) \oplus k_1 = b$; this strategy can be iterated. $\Box$

Wittbold and Johnson adapt nondeducibility to define *nondeducibility on strategies*: *High*'s behaviour, or strategy, is modelled by a function from observed outputs to subsequent inputs; *Low*'s view of the system should then be compatible with all high-level strategies.

3.3 Invariance

Various authors have produced definitions of information flow based on the notion of *invariance*: the behaviour of the system as seen by *Low* should not vary as a result of behaviour of *High*.

One such definition is due to Ryan [Rya91]. Ryan defines a process P to be invariant in L if whenever tr and tr' are two traces of P that differ only in events performed by *High*, then *Low*'s view of P should be the same after tr as after tr'. In a process algebraic setting, it is normally assumed that the only ways in which *Low* can interact with the system are through performing events or having events refused. The events available to *Low* following trace tr can be written as[1]:

$$\mathsf{inits}_L(P \mathbin{\mathsf{after}} tr) \mathrel{\widehat{=}} \{a \in L \mid tr ^\frown \langle a \rangle \in traces(P)\}.$$

Similarly, the refusals from L following tr can be written as:

$$\mathsf{refs}_L(P \mathbin{\mathsf{after}} tr) \mathrel{\widehat{=}} \{X \in \mathbb{P}\, L \mid (tr, X) \in failures(P)\}.$$

[1]For convenience, we have adapted Ryan's notation, slightly.

Hence Ryan's definition of security can be formalised as follows:

Definition 3 A process is said to be *failures-invariant* in L if:

$$\begin{array}{l} INV_L(P) \mathrel{\widehat{=}} \forall\, tr, tr' \in traces(P) \bullet tr \upharpoonright L = tr' \upharpoonright L \Rightarrow \\ \qquad \mathsf{inits}_L(P \mathbin{\mathsf{after}} tr) = \mathsf{inits}_L(P \mathbin{\mathsf{after}} tr') \\ \qquad \wedge\ \mathsf{refs}_L(P \mathbin{\mathsf{after}} tr) = \mathsf{refs}_L(P \mathbin{\mathsf{after}} tr'). \end{array}$$

Similar CSP-based definitions have been proposed by Allen [All91], Graham-Cumming [GC92] and Jacob [Jac90b].

Example 3 Consider the process

$$P_1 \mathrel{\widehat{=}} h \to l \to STOP.$$

This is correctly identified as insecure. Consider, for example, the traces $tr \mathrel{\widehat{=}} \langle\rangle$ and $tr' \mathrel{\widehat{=}} \langle h\rangle$; then $tr \upharpoonright L = \langle\rangle = tr' \upharpoonright L$, but

$$\mathsf{inits}_L(P_1 \mathbin{\mathsf{after}} tr) = \{\} \neq \mathsf{inits}_L(P_1 \mathbin{\mathsf{after}} tr') = \{l\}.$$

The event l is available to *Low* if and only if *High* performs h, so this represents a flow of information from *High* to *Low*. □

Example 4 The process

$$P_2 \mathrel{\widehat{=}} h \to l \to STOP \,\square\, l \to STOP$$

is secure, because the behaviour of the process from *Low*'s point of view does not vary as a result of *High*'s behaviour; for example, taking $tr \mathrel{\widehat{=}} \langle\rangle$, $tr' \mathrel{\widehat{=}} \langle h\rangle$:

$$\begin{array}{rcccl} \mathsf{inits}_L(P_2 \mathbin{\mathsf{after}} tr) & = & \{l\} & = & \mathsf{inits}_L(P_2 \mathbin{\mathsf{after}} tr'), \\ \mathsf{refs}_L(P_2 \mathbin{\mathsf{after}} tr) & = & \mathbb{P}(L - \{l\}) & = & \mathsf{refs}_L(P_2 \mathbin{\mathsf{after}} tr'). \end{array}$$

The event l is available to *Low* regardless of whether or not *High* performs an h. □

However, this definition mis-classifies some processes.

Example 5 Consider the process

$$P \mathrel{\widehat{=}} l \to STOP \rhd h \to STOP.$$

This is not failures-invariant, for consider the traces $tr \mathrel{\widehat{=}} \langle\rangle$, $tr' \mathrel{\widehat{=}} \langle h\rangle$; then

$$\mathsf{inits}_L(P \mathbin{\mathsf{after}} tr) = \{l\} \neq \mathsf{inits}_L(P \mathbin{\mathsf{after}} tr') = \{\}.$$

However, there appears to be no way in which *High* can use this process to pass information to *Low*: it is the occurrence of the timeout that changes *Low*'s view of the system, rather than any action of *High*. □

Example 6 Consider:

$$\begin{array}{rl} Q_1 \mathrel{\widehat{=}} & h \to (l \to STOP \sqcap STOP) \\ & \square \\ & (l \to STOP \sqcap STOP). \end{array}$$

This satisfies failures-invariance, because *Low* will, nondeterministically, either be able to perform an l, or will have the l refused, whether or not *High* performs an h. So, for example, again taking $tr \mathrel{\widehat{=}} \langle\rangle$, $tr' \mathrel{\widehat{=}} \langle h\rangle$:

$$\begin{array}{rcccl} \mathsf{inits}_L(Q_1 \mathbin{\mathsf{after}} tr) & = & \{l\} & = & \mathsf{inits}_L(Q_1 \mathbin{\mathsf{after}} tr'), \\ \mathsf{refs}_L(Q_1 \mathbin{\mathsf{after}} tr) & = & \mathbb{P}(L) & = & \mathsf{refs}_L(Q_1 \mathbin{\mathsf{after}} tr'). \end{array}$$

However, this assumes that the two different nondeterministic choices are implemented in the same way, which is by no means necessary. The normal intuition in

CSP is that a nondeterministic choice represents under-specification: this can be resolved by the implementer deciding how to implement the under-specification, or it can be resolved by some mechanism at run time. For example, if the first nondeterministic choice is implemented to always select its first argument, and the second nondeterministic choice is implemented to always select its second argument, then Q_1 will act like

$$h \rightarrow l \rightarrow STOP \;\Box\; STOP = h \rightarrow l \rightarrow STOP,$$

which is the process P_1 from Example 3, and which does not satisfy failures-invariance. In other words we have a process Q_1 that is considered secure, but one of its implementations, P_1, is insecure.

Similarly, if the two nondeterministic choices act probabilistically, but with different associated probabilities, then there will be a (probabilistic) flow of information to *Low*.

For these reasons, we would consider this process to be insecure, contrary to the definition of failures-invariance. □

Example 7 Let $LEAK$ be the following—clearly insecure—process

$$LEAK \;\widehat{=}\; h?x \rightarrow l.x \rightarrow LEAK,$$

which receives input from *High* and passes it to *Low*. Consider also the process $LEAK \sqcap CHAOS(\Sigma)$. This process is equivalent to $CHAOS(\Sigma)$ in the failures-divergences model. $CHAOS(\Sigma)$ satisfies the above definition of security, because

$$\mathsf{inits}_L(CHAOS(\Sigma) \;\mathsf{after}\; tr) \;=\; L$$
$$\mathsf{refs}_L(CHAOS(\Sigma) \;\mathsf{after}\; tr) \;=\; \mathbb{P}(L),$$

for every trace tr. However, $LEAK \sqcap CHAOS(\Sigma)$ should be considered insecure, because it can evolve into a state, $LEAK$, that is insecure. □

3.4 Nondeducibility on compositions

In [FG95, Foc96, FGM00], Focardi, Gorrieri and Martinelli define several security properties, jointly termed *Non Deducibility on Composition (NDC)*. We can translate these properties into CSP as follows, writing CSP_H for the set of high-level CSP processes:

$$\forall\, Q \in CSP_H \bullet P \underset{H}{\parallel} STOP \equiv (P \underset{H}{\parallel} Q) \setminus H,$$

where the nature of the equivalence depends upon the security property in question: bisimulation, failures and timed-bisimulation are all considered. In other words, however *High* acts, as captured by the process Q, Low's view of the system, as captured by $(P \underset{H}{\parallel} Q) \setminus H$, does not change.

These definitions all suffer from the problems identified in Examples 6 and 7: apparently secure processes can be refined by insecure ones.

3.5 Independence through determinism

In an attempt to overcome the problems described above, Roscoe introduced a definition of security based upon determinism [Ros95, Ros97]. Essentially, Roscoe considers a process to be secure if *Low*'s view of the process is deterministic: this implies, in particular, that *High* cannot change *Low*'s view.

Recall that in CSP a process is nondeterministic if there is some trace tr and some event a such that following trace tr, the event a can be either performed or refused:

$$\exists\, tr \in \Sigma^* \,;\, a \in \Sigma \bullet tr ^\frown \langle a \rangle \in traces(P) \wedge (tr, \{a\}) \in failures(P),$$

and is deterministic otherwise.

For a divergence-free process P, Roscoe defines the *lazy abstraction* of P to L (and away from H), written $\mathcal{L}_H(P)$, to be the divergence-free process with failure set:

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

High level events are hidden in the traces; the refusal sets are closed up to include extra high level events.

Roscoe's definition of security is then as follows:

Definition 4 A finitely nondeterministic, divergence-free process P is *lazily independent* from H (written $\mathcal{L}IND_H(P)$) if and only if $\mathcal{L}_H(P)$ is deterministic.

Example 8 Consider again the process

$$P_1 \mathrel{\widehat{=}} h \rightarrow l \rightarrow STOP$$

from Example 3. This is correctly identified as insecure, because $\mathcal{L}_H(P_1)$ can perform the trace $\langle l \rangle$ (corresponding to the trace $\langle h, l \rangle$ of P_1), but also has the failure $(\langle\rangle, \{l\})$ (corresponding to the failure $(\langle\rangle, \{l\})$ of P_1). $\square$

Example 9 Recall the process

$$\begin{aligned} Q_1 \mathrel{\widehat{=}}\ & h \rightarrow (l \rightarrow STOP \sqcap STOP) \\ & \Box \\ & (l \rightarrow STOP \sqcap STOP) \end{aligned}$$

from Example 6. This is identified as insecure—as we believe it should be—because its lazy abstraction is nondeterministic; there are several ways of seeing this, but the way most closely related to the discussion of the previous subsection is to note that $\mathcal{L}_H(Q_1)$ can perform the trace $\langle l \rangle$ (corresponding to the trace $\langle h, l \rangle$ of Q_1), but also has the failure $(\langle\rangle, \{l\})$ (corresponding to the failure $(\langle\rangle, \{l\})$ of Q_1). $\square$

However, this definition of security identifies some processes as being insecure, when really they should be considered as secure.

Example 10 The process

$$P_3 \mathrel{\widehat{=}} l \rightarrow STOP \sqcap l' \rightarrow STOP$$

is flagged as insecure, because of the nondeterminism, even though *High* can't perform any events, and so has no way of influencing the nondeterminism so as to pass information to *Low*. □

Example 11 The process

$$P_4 \mathrel{\widehat{=}} h \rightarrow STOP \mid\mid\mid (l \rightarrow STOP \sqcap l' \rightarrow STOP)$$

is similarly flagged as being insecure for the same reason as with the process in the previous example. However, if this system really were built as the interleaving above then the *High* and *Low* parts of the process would be independent, and so would have no way of communicating. □

Example 12 Finally, consider the process

$$\begin{aligned} Q_2 \mathrel{\widehat{=}} \ & (h \rightarrow l \rightarrow STOP \,\square\, l \rightarrow STOP) \\ & \sqcap \\ & h \rightarrow STOP. \end{aligned}$$

This is considered insecure, again because of the nondeterminism. However, the nondeterministic choice is resolved before *High*'s action, so *High* can have no influence upon how the choice is resolved, and hence *High* cannot influence what events are available to *Low*, so there is no way for *High* to pass information to *Low*. □

Note, though, that the two processes Q_1 and Q_2 of Examples 9 and 12 are considered equivalent in all standard models of CSP; but, we consider Q_1 to be insecure, and Q_2 to be secure. This shows that standard models of CSP are not sufficient for reasoning about information flow. The difference between these two processes is that they display different branching behaviour, which has an effect upon their security; standard models of CSP abstract away from such differences. It is clear, therefore, that any model that hopes to accurately identify information flow must be more discriminatory than the standard models.

3.6 Local noninterference

In [For99], Forster considers several definitions of information flow, termed *local noninterference*. One of his definitions is the following:

Definition 5 Let P be a process whose alphabet is partitioned by H and L. P satisfies *strong failures-divergences local noninterference*, written $\mathsf{SLNI}^{FD}(P)$, if[2]

$$\begin{aligned} & \forall\, Q \in States(P) \bullet \forall\, h \in H \\ & \quad \text{if } Q \xrightarrow{h} Q' \\ & \quad \text{then } failures(Q\backslash_H) = failures(Q'\backslash_H) \\ & \qquad \wedge\ divergences(Q\backslash_H) = divergences(Q'\backslash_H). \end{aligned}$$

[2] $States(P)$ represents all the states of P; i.e. all processes Q such that $P \stackrel{s}{\longmapsto} Q$ for some s.

The definition essentially says that the behaviour of the system, from *Low*'s point of view, cannot be changed by actions of *High*.

The quantification over all states of P is to deal with processes like $LEAK \sqcap CHAOS(\Sigma)$ from Example 7, where a potential insecurity can be masked by behaviour elsewhere in the system.

Example 13 Consider the process $P_1 \mathrel{\widehat{=}} h \to l \to STOP$ from Example 3. This does not satisfy this condition because $P_1 \xrightarrow{h} l \to STOP$, and $failures(P_1 \backslash_H) \neq failures((l \to STOP)\backslash_H)$. $\Box$

Example 14 Consider also the process

$$Q_2 \mathrel{\widehat{=}} (h \to l \to STOP \,\Box\, l \to STOP) \sqcap h \to STOP$$

from Example 12. This has two high level transitions:

- $h \to l \to STOP \,\Box\, l \to STOP \xrightarrow{h} l \to STOP$ and

  $$\begin{aligned}&failures((h \to l \to STOP \,\Box\, l \to STOP)\backslash_H) = \\ &\quad failures((l \to STOP)\backslash_H),\end{aligned}$$

  and likewise for divergences;

- $h \to STOP \xrightarrow{h} STOP$ and

  $$failures((h \to STOP)\backslash_H) \;=\; failures(STOP\backslash_H),$$

  and likewise for divergences.

Hence the condition is satisfied. $\Box$

Example 15 Consider, however, the process

$$Q_1 \mathrel{\widehat{=}} h \to (l \to STOP \sqcap STOP) \,\Box\, (l \to STOP \sqcap STOP).$$

from Example 6. This satisfies the above condition, because

$$Q_1 \xrightarrow{h} l \to STOP \sqcap STOP$$

and $failures(Q_1\backslash_H) = failures((l \to STOP \sqcap STOP)\backslash_H)$, and likewise for divergences. However, this is in contravention of our intuition, outlined above, that this process should be considered insecure. $\Box$

4 Introduction to the operational model

The overall aim of this paper is to correctly capture the notion of information flow. We aim to produce a definition of information flow that does not suffer from the weaknesses of existing definitions, described in the previous section.

As the previous remarks make clear, existing models do not make enough discriminations between processes. We therefore have to introduce a new semantic model, which makes more distinctions.

A critical feature of the model will be that it will deal with nondeterministic choices in such a way that we can argue about the ways in which the choices are resolved, so that we can deal correctly with processes such as Q_1 from Example 6.

In this section we give an informal introduction to the operational model we will use to capture information flow; this will help to further explain our intuitions about information flow.

The main difficulty we face is that posed by nondeterminism. We want to be able to investigate whether the nondeterministic choices of a process can be resolved in a way that would allow a flow of information. Consider, again, the process

$$P_4 \mathrel{\widehat{=}} h \rightarrow STOP \mid\mid\mid (l \rightarrow STOP \sqcap l' \rightarrow STOP).$$

We would not expect the nondeterministic choice on the right to be influenced by whether or not the h event has been performed. However, traditional operational models do not provide enough information to capture this. For example, the standard transition diagram for the above process is as in Figure 1. The

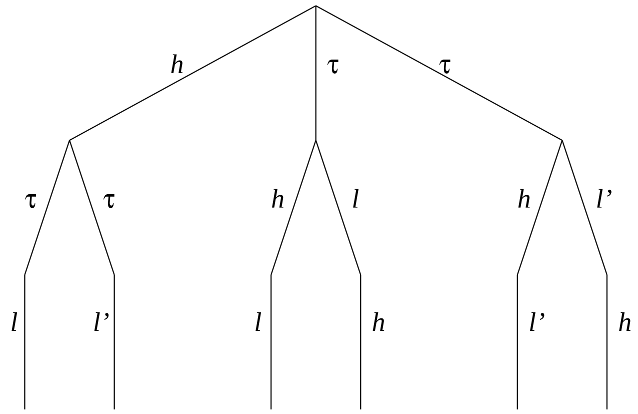

Figure 1: Traditional transition diagram for P_4

nondeterministic choice can be resolved in two different places in this diagram, either before or after the h is performed. However, it is the same nondeterministic choice in each case, and should be resolved in the same way. The transition diagram does not include enough information to record this fact.

Our operational model will therefore use a labelled transition diagram, but where the transitions contain extra information, which we call markers, recording when different transitions correspond to the same nondeterministic choice being resolved.

For example, the process P_4 will be represented by the transition system in Figure 2, where the λ_i are the markers mentioned above[3]. Note that in both

[3]The choice of the subscripts for the λs is for consistency with the formal definition of the

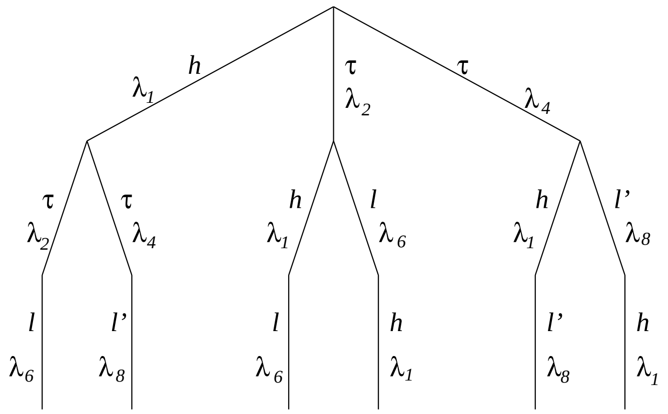

Figure 2: Transition diagram with markers for P_4

places where the nondeterministic choice is resolved, the transition representing the choice being resolved to the left (in favour of l) is given the marker λ_2, while the transition representing the choice being resolved to the right (in favour of l') is given the marker λ_4.

Our intuition is that identically labelled nondeterministic choices should be resolved in the same way; however, we have no way of determining *a priori* how these choices will be resolved. We need to define a mechanism for resolving the nondeterministic choice, which we will term a *demon*. We will suppose that the demon makes his decisions as to how to resolve the nondeterministic choices based upon the markers on the available transitions; this will mean that he does indeed resolve different instances of the same nondeterministic choice in the same way.

The normal intuition in CSP is that a nondeterministic choice may represent either *under-specification*, which the implementor of the process can resolve, or *unpredictable behaviour*, which is resolved at run time. Our notion of demon captures both of these forms.

We will represent a demon by a total order over the markers, corresponding to the demon's order of preference for those markers. For example, if in Figure 2 the demon prefers λ_2 to λ_4, then the nondeterministic choice will be resolved in favour of the event l, regardless of where in the diagram the nondeterministic choice is resolved, i.e. regardless of *High*'s actions.

Note that all the transitions in the transition diagram receive additional markers, not just those representing nondeterministic choices: this is because such transitions can become nondeterministic, for example by the application of the hiding operator.

operational model we will present in the next section; the reader is advised not to study them too closely for the moment.

Consider, again, the two processes:

$$Q_1 \mathrel{\widehat{=}} h \rightarrow (l \rightarrow STOP \sqcap STOP) \,\Box\, (l \rightarrow STOP \sqcap STOP),$$
$$Q_2 \mathrel{\widehat{=}} (h \rightarrow l \rightarrow STOP \,\Box\, l \rightarrow STOP) \sqcap h \rightarrow STOP,$$

from Examples 6 and 12, that standard models fail to distinguish. The transition diagrams for these processes are in Figures 3 and 4. For the moment, we will just present the transition diagrams, so as to illustrate our intuition; we will show how they are obtained in the next section.

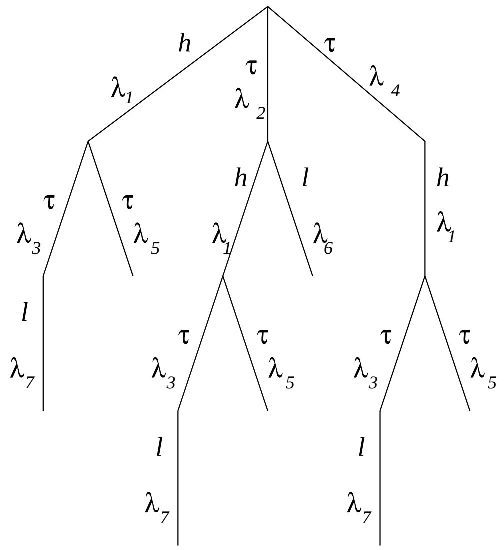

Figure 3: Transition diagram for Q_1

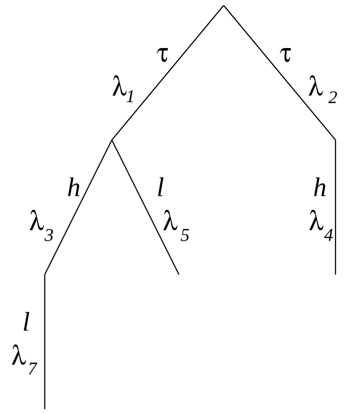

Figure 4: Transition diagram for Q_2

Consider what happens when Q_1 is in the presence of a demon that prefers λ_3 to λ_5, and prefers λ_4 to λ_2. If *High* performs h, then the demon will select the nondeterministic transition labelled with λ_3, and so *Low* will be able to perform l. Alternatively, if *High* does not performs h then *Low* will not be able to perform l. Hence, *Low*'s view of the process will depend upon *High*'s behaviour, and so we have captured the insecurity of this process.

Compare this, however, with Q_2; here, the nondeterministic choice is resolved before *High* does any action, and so is independent of *High*'s action. If the demon prefers λ_1 to λ_2, then *Low* will be able to perform l, whether or not *High* performs h; and if the demon prefers λ_2 to λ_1 then *Low* will be unable to perform l, whether or not *High* performs h; the change in *Low*'s view of Q_2 is caused by the demon, rather than *High*; hence this process is secure.

5 Operational semantics

In this section we will formally define the operational semantic model that we introduced in the previous section.

We assume some set *Marker* of markers, ranged over by λ, λ', etc. We write *MarkerSeq* for the set of countably infinite sequences of distinct markers[4]:

$$\mathit{MarkerSeq} \;\widehat{=}\; \{\Lambda \mid \Lambda \in \mathit{Marker}^{\omega} \wedge \forall\, i, j \in \mathbb{N}_1 \bullet i \neq j \Rightarrow \Lambda(i) \neq \Lambda(j)\}.$$

We let Λ, Λ', etc. range over *MarkerSeq*.

It turns out to be convenient to label transitions that correspond to synchronisations between parallel components with pairs of markers, one marker corresponding to each parallel component. For example, a synchronisation of the process $P \underset{A}{\parallel} Q$ might be labelled by the pair of markers (λ_P, λ_Q) where λ_P corresponds to P and λ_Q corresponds to Q. In such compositions, we will also use notation such as $(\lambda_P, \_)$ to represent an event performed by only P, where the λ_P corresponds to P, and the "_" indicates that Q performs no part in the event.

We define *CpdMarker* to be the space of all such compound markers:

$$\mu \in \mathit{CpdMarker} \;::=\; \lambda \mid (\mu, \mu) \mid (\mu, \_) \mid (\_, \mu).$$

We will define operational semantics for a process with respect to some initial sequence of markers; the markers from this sequence are used to label the transitions. We will write

$$(P, \Lambda) \xrightarrow{a,\mu} (P', \Lambda')$$

[4] Marker^{ω} represents the set of countably infinite sequences with elements taken from *Marker*; $\Lambda(i)$ represents the ith element of sequence Λ, counting from 1.

to mean that the process P, with marker sequence Λ, can perform the event a, marked with μ, and evolve into process P', with remaining markers Λ'. The (possibly compound) marker μ will be built up from markers in Λ; the remaining markers Λ' will be a subsequence of Λ.

Throughout this paper, we define relations and functions similar to those from [Ros97], described in Section 2, and use similar notation; however, we take care to distinguish them, by the number of arguments.

We call a process–marker sequence pair (P, Λ) a *configuration*:

$$Config \mathrel{\widehat{=}} CSP \times MarkerSeq.$$

Formally, the relation:

$$\longrightarrow\ :\ \mathbb{P}(Config \times (\Sigma^\tau \times CpdMarker) \times Config)$$

is defined by the rules in the following subsections.

The parts of the rules concerning the events performed and the ways in which processes evolve will be identical to the traditional operational semantics, given in Section 2.1. What is new is the treatment of markers. In several places in the rules, we have to decide how the markers are allocated to different transitions in the transition diagram. Often these decisions are made fairly arbitrarily, and certainly could have been made differently. We present specific allocations of markers merely for the sake of definiteness.

We will adopt the normal operational semantics conventions, for example writing $(P, \Lambda) \xrightarrow{a,\mu}$ as a shorthand for $\exists\, P', \Lambda' \bullet (P, \Lambda) \xrightarrow{a,\mu} (P', \Lambda')$.

5.1 Transition rules

STOP The process *STOP* can perform no transitions.

Prefixing The process $a \to P$ can perform an a and evolve into process P; in the context of a configuration of the form $(a \to P,\ \lambda : \Lambda)$[5], the transition receives the marker λ, and the markers Λ are used to label subsequent transitions. Formally:

$$(a \to P,\ \lambda : \Lambda) \xrightarrow{a,\lambda} (P,\ \Lambda).$$

Nondeterministic choice In a nondeterministic choice $P \sqcap Q$, the first two markers in the associated marker sequence are used to label the nondeterministic choice itself, while the rest of the markers are split, alternately, between P and Q:

$$(P \sqcap Q,\ \lambda : \lambda' : \Lambda) \xrightarrow{\tau,\lambda} (P,\ \mathsf{odds}\,\Lambda),$$
$$(P \sqcap Q,\ \lambda : \lambda' : \Lambda) \xrightarrow{\tau,\lambda'} (Q,\ \mathsf{evens}\,\Lambda),$$

[5] We use the functional programming cons operator ":", so $\lambda : \Lambda$ is the sequence whose first element is λ, and which continues with the elements of Λ.

where the operators odds and evens select the elements of a sequence with odd or even indices, respectively. Formally:

$$\begin{aligned}\mathsf{odds}(\lambda : \lambda' : \Lambda) &\mathrel{\widehat{=}} \lambda : \mathsf{odds}\,\Lambda,\\ \mathsf{evens}(\lambda : \lambda' : \Lambda) &\mathrel{\widehat{=}} \lambda' : \mathsf{evens}\,\Lambda.\end{aligned}$$

External choice Similarly, in an external choice of the form $P \mathbin{\Box} Q$, the markers are split, alternately, between P and Q, the odd indexed markers labelling P and the even indexed markers labelling Q. If one of the processes, so labelled, can perform a visible event, then this resolves the choice.

$$\frac{(P, \mathsf{odds}\,\Lambda) \xrightarrow{a,\mu} (P', \Lambda')}{(P \mathbin{\Box} Q, \Lambda) \xrightarrow{a,\mu} (P', \Lambda')}\ a \neq \tau$$

$$\frac{(Q, \mathsf{evens}\,\Lambda) \xrightarrow{a,\mu} (Q', \Lambda')}{(P \mathbin{\Box} Q, \Lambda) \xrightarrow{a,\mu} (Q', \Lambda')}\ a \neq \tau$$

However, the choice is not resolved by internal τ actions; if P can do an internal transition to P', then $P \mathbin{\Box} Q$ can do an internal transition to $P' \mathbin{\Box} Q$, where subsequent transitions are labelled using those markers left over for P, and those allocated to Q:

$$\frac{(P, \mathsf{odds}\,\Lambda) \xrightarrow{\tau,\mu} (P', \Lambda')}{(P \mathbin{\Box} Q, \Lambda) \xrightarrow{\tau,\mu} (P' \mathbin{\Box} Q, \mathsf{interleave}(\Lambda', \mathsf{evens}\,\Lambda))}$$

$$\frac{(Q, \mathsf{evens}\,\Lambda) \xrightarrow{\tau,\mu} (Q', \Lambda')}{(P \mathbin{\Box} Q, \Lambda) \xrightarrow{\tau,\mu} (P \mathbin{\Box} Q', \mathsf{interleave}(\mathsf{odds}\,\Lambda, \Lambda'))}$$

where the $\mathsf{interleave}$ operator essentially undoes the splitting of a sequence of markers:

$$\mathsf{interleave}(\lambda : \Lambda, \lambda' : \Lambda') \mathrel{\widehat{=}} \lambda : \lambda' : \mathsf{interleave}(\Lambda, \Lambda').$$

Hiding The operational semantics for $P \setminus A$ is derived immediately from the semantics of P, by renaming events from A to the internal event τ:

$$\frac{(P, \Lambda) \xrightarrow{a,\mu} (P', \Lambda')}{(P \setminus A, \Lambda) \xrightarrow{\tau,\mu} (P' \setminus A, \Lambda')}\ a \in A$$

$$\frac{(P, \Lambda) \xrightarrow{a,\mu} (P', \Lambda')}{(P \setminus A, \Lambda) \xrightarrow{a,\mu} (P' \setminus A, \Lambda')}\ a \notin A$$

Restriction The operational semantics for $P\backslash_A$ is similarly derived from the semantics of P, by allowing only events not from A:

$$\frac{(P, \Lambda) \xrightarrow{a,\mu} (P', \Lambda')}{(P\backslash_A, \Lambda) \xrightarrow{a,\mu} (P'\backslash_A, \Lambda')}\ a \notin A$$

Parallel composition In a parallel composition, the markers are split between the two sides, as for the other binary operators. If one component of the parallel composition can perform an event not in the synchronisation set, then so can the parallel composition; the markers labelling subsequent transitions are those left over from the subprocess performing the event, combined with those of the other subprocess:

$$\frac{(P, \mathsf{odds}\,\Lambda) \xrightarrow{a,\mu} (P', \Lambda')}{(P \underset{A}{\parallel} Q, \Lambda) \xrightarrow{a,(\mu,-)} (P' \underset{A}{\parallel} Q, \mathsf{interleave}(\Lambda', \mathsf{evens}\,\Lambda))}\ a \notin A$$

$$\frac{(Q, \mathsf{evens}\,\Lambda) \xrightarrow{a,\mu} (Q', \Lambda')}{(P \underset{A}{\parallel} Q, \Lambda) \xrightarrow{a,(-,\mu)} (P \underset{A}{\parallel} Q', \mathsf{interleave}(\mathsf{odds}\,\Lambda, \Lambda'))}\ a \notin A$$

If both sides of the parallel composition can perform an event in the synchronisation set, then the parallel composition can perform that event; this event is labelled with a marker composed from the markers for the sub-processes:

$$\frac{\begin{array}{l}(P, \mathsf{odds}\,\Lambda) \xrightarrow{a,\mu_P} (P', \Lambda_P) \\ (Q, \mathsf{evens}\,\Lambda) \xrightarrow{a,\mu_Q} (Q', \Lambda_Q)\end{array}}{(P \underset{A}{\parallel} Q, \Lambda) \xrightarrow{a,(\mu_P,\mu_Q)} (P' \underset{A}{\parallel} Q', \mathsf{interleave}(\Lambda_P, \Lambda_Q))}\ a \in A$$

Recursion Finally, recursions are unwound via a τ event:

$$(\mu\, X \bullet P,\ \lambda : \Lambda) \xrightarrow{\tau,\lambda} (P[\mu\, X \bullet P/X], \Lambda).$$

Remark Note that when these rules are applied to the examples in the previous section, they really do produce the labelled transition diagrams we claimed, when the operational semantics is taken with respect to the initial sequence of markers $\Lambda \mathrel{\widehat{=}} \langle \lambda_1, \lambda_2, \ldots \rangle$. For example, the semantics of the process

$$Q_1 \mathrel{\widehat{=}} h \to (l \to STOP \sqcap STOP) \,\square\, (l \to STOP \sqcap STOP),$$

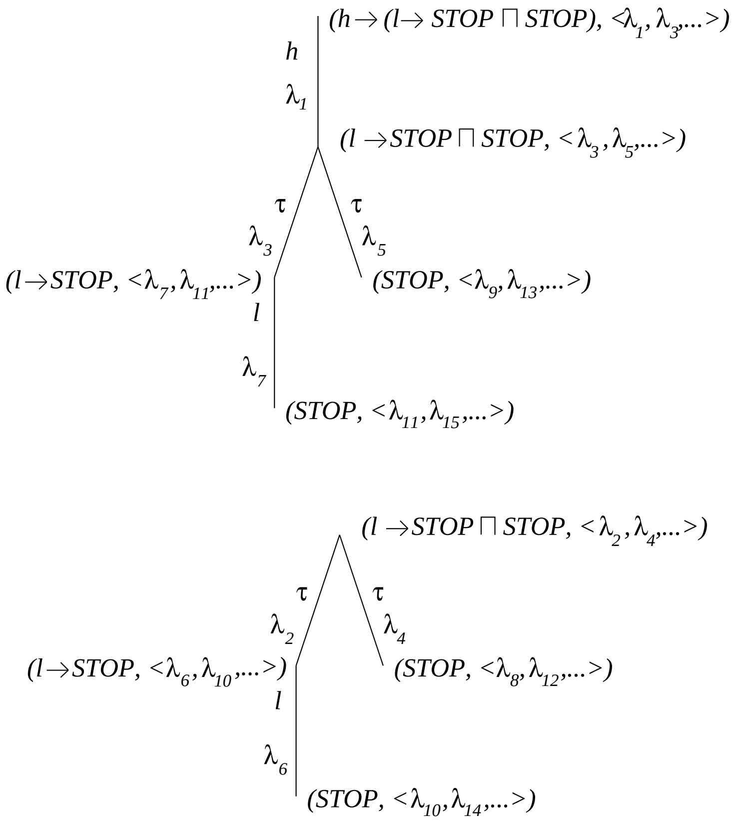

Figure 5: Transition diagrams for the processes $h \rightarrow (l \rightarrow STOP \sqcap STOP)$ and $l \rightarrow STOP \sqcap STOP$.

is indeed as in Figure 3; we give the semantics for the two sub-processes

$$h \rightarrow (l \rightarrow STOP \sqcap STOP) \quad \text{and} \quad l \rightarrow STOP \sqcap STOP$$

with respect to the marker sequences $\mathsf{odds}\,\Lambda$ and $\mathsf{evens}\,\Lambda$, respectively, in Figure 5. The reader is encouraged to understand these transition diagrams, and how the transition diagram for Q_1 is obtained from them.

5.2 Some lemmas

We now state a few lemmas concerning the operational semantics. Some proofs are omitted and can be found in [Low00].

The following lemma relates the operational semantics, above, to the standard operational semantics from Section 2.1.

Lemma 1 If $(P, \Lambda) \xrightarrow{a,\mu} (P', \Lambda')$ then $P \xrightarrow{a} P'$. Conversely, if $P \xrightarrow{a} P'$, then for every Λ, there exist μ, Λ' such that $(P, \Lambda) \xrightarrow{a,\mu} (P', \Lambda')$.

The following lemma shows that the operational semantics is essentially independent of the choice of initial marker sequence. If we evaluate the operational semantics with respect to two different initial marker sequences, then we will obtain two similar transition systems that differ only by having had markers uniformly renamed.

Lemma 2 Let $\phi : \mathit{Marker} \to \mathit{Marker}$ be an injective function. We lift ϕ to sequences of markers by pointwise application:

$$\phi(\langle\lambda_1, \lambda_2, \ldots, \lambda_n\rangle) \mathrel{\widehat{=}} \langle\phi(\lambda_1), \phi(\lambda_2), \ldots, \phi(\lambda_n)\rangle,$$

and to compound markers by component-wise application:

$$\phi(\mu, \mu') \mathrel{\widehat{=}} (\phi(\mu), \phi(\mu')), \qquad \phi(\mu, \_) \mathrel{\widehat{=}} (\phi(\mu), \_), \qquad \phi(\_, \mu) \mathrel{\widehat{=}} (\_, \phi(\mu)).$$

Then

$$(P, \Lambda) \xrightarrow{a,\mu} (P', \Lambda') \ \ \textit{implies} \ \ (P, \phi(\Lambda)) \xrightarrow{a,\phi(\mu)} (P', \phi(\Lambda')).$$

The markers labelling a transition are indeed taken from the initial marker sequence; they do not appear in the resulting marker sequence; the markers in the resulting marker sequence are a subset of those in the initial marker sequence. We define the atomic markers comprising a (possibly compound) marker by:

$$\begin{aligned} \mathit{parts}(\lambda) &\mathrel{\widehat{=}} \{\lambda\}, \\ \mathit{parts}(\mu, \mu') &\mathrel{\widehat{=}} \mathit{parts}(\mu) \cup \mathit{parts}(\mu'), \\ \mathit{parts}(\mu, \_), \mathit{parts}(\_, \mu) &\mathrel{\widehat{=}} \mathit{parts}(\mu). \end{aligned}$$

Then:

Lemma 3 If $(P, \Lambda) \xrightarrow{a,\mu} (P', \Lambda')$ then[6]:

- $\forall \lambda \in \mathit{parts}(\mu) \bullet \lambda \ \mathsf{in} \ \Lambda$;
- $\forall \lambda \in \mathit{parts}(\mu) \bullet \neg \lambda \ \mathsf{in} \ \Lambda'$;
- $\forall \lambda \ \mathsf{in} \ \Lambda' \bullet \lambda \ \mathsf{in} \ \Lambda$.

Each marker will be used to label at most one transition from any state.

Lemma 4 If $(P, \Lambda) \xrightarrow{a,\mu} (P', \Lambda')$ and $(P, \Lambda) \xrightarrow{a',\mu} (P'', \Lambda'')$ then $a = a' \wedge P' = P'' \wedge \Lambda' = \Lambda''$.

The transition graph is finitely branching.

Lemma 5 Let $P \in \mathit{CSP}$, $\Lambda \in \mathit{MarkerSeq}$; then the set of transitions from (P, Λ) is finite.

6 Extracting denotational information

In this section we extract denotational information from the operational semantics in the previous section. Much of the development follows that of [Ros97],

[6]The notation $\lambda \ \mathsf{in} \ \Lambda$ means that λ is an element of the sequence Λ.

decribed in Section 2. We begin by formalising the notion of demon, and define the transitions of a configuration in the presence of a particular demon; in Section 6.2 we lift this to multi-step transitions; in Section 6.3, we define a denotational semantic model, similar to the traditional denotational models of CSP; in Section 6.4 we define some equivalence and refinement relations.

6.1 Demons

We model the demon who resolves the nondeterministic choices by a total order $\leqslant$ over compound markers. If $\mu \leqslant \mu'$ then we say that the demon prefers the marker μ' to μ; the intuition is that the demon will select a transition labelled with μ' in preference to one labelled with μ.

The following definition captures this, and also defines a healthiness property that we will require of demons:

Definition 6 (Demons) A demon is a total order $\leqslant$ over compound markers such that for all μ_1, μ_2, μ:

$$\mu_1 \leqslant \mu_2 \;\Rightarrow\; (\mu_1, \mu) \leqslant (\mu_2, \mu) \wedge (\mu, \mu_1) \leqslant (\mu, \mu_2), \tag{1}$$

$$\begin{aligned} \mu_1 \leqslant \mu_2 \;\Rightarrow\; & (\mu_1, \mu) \leqslant (\mu_2, \_) \wedge (\mu_1, \_) \leqslant (\mu_2, \mu) \wedge \\ & (\mu, \mu_1) \leqslant (\_, \mu_2) \wedge (\_, \mu_1) \leqslant (\mu, \mu_2). \end{aligned} \tag{2}$$

Equation (1) says that preferences on one side of a parallel composition are reflected in overall preferences; it is equivalent to:

$$\mu_1 \leqslant \mu_1' \wedge \mu_2 \leqslant \mu_2' \;\Rightarrow\; (\mu_1, \mu_2) \leqslant (\mu_1', \mu_2'). \tag{3}$$

Equation (2) similarly shows how preferences on one side of a parallel composition are reflected in overall preferences where the other side may or may not perform an action. These healthiness conditions will be useful when we come to prove compositionality results, in Section 8.5.1.

Where a particular demon $\leqslant$ is obvious from the context, we will write $\geqslant$ for its converse, $<$ for its strict version, and $>$ for its strict converse.

We introduce a new transition relation, describing the behaviour of a process in the presence of a particular demon. We write $(P, \Lambda) \xhookrightarrow[\leqslant]{a} (P', \Lambda')$ if the configuration (P, Λ) in the presence of demon $\leqslant$ can perform the event a and evolve into (P', Λ'). For this to be the case:

- the initial configuration itself should be able to perform the corresponding transformation, i.e. $(P, \Lambda) \xrightarrow{a,\mu} (P', \Lambda')$, for some μ;
- the demon should not prefer any other possible transition labelled with the same event a, i.e. $\nexists\, \mu' \bullet (P, \Lambda) \xrightarrow{a,\mu'} \wedge\, \mu' > \mu$;
- the demon should not prefer any internal transition, i.e.

$$\nexists\, \mu' \bullet (P, \Lambda) \xrightarrow{\tau,\mu'} \wedge\, \mu' > \mu\,.$$

The $\overset{a}{\underset{\leqslant}{\hookrightarrow}}$ transition is formally defined by the following rule:

$$\frac{\begin{array}{l}(P,\Lambda) \xrightarrow{a,\mu} (P',\Lambda') \\ \nexists\, \mu' \bullet (P,\Lambda) \xrightarrow{a,\mu'} \wedge\ \mu' > \mu \\ \nexists\, \mu' \bullet (P,\Lambda) \xrightarrow{\tau,\mu'} \wedge\ \mu' > \mu\end{array}}{(P,\Lambda) \overset{a}{\underset{\leqslant}{\hookrightarrow}} (P',\Lambda')}$$

Example 16 Consider

$$P \ \widehat{=}\ a \to STOP \sqcap b \to STOP.$$

This has the following operational semantics with respect to the initial sequence of markers $\langle \lambda_1, \lambda_2, \ldots \rangle$:

$$\begin{array}{lll} (P, \langle \lambda_1, \lambda_2, \ldots \rangle) & \xrightarrow{\tau,\lambda_1} & (a \to STOP, \langle \lambda_3, \lambda_5, \ldots \rangle) \\ & \xrightarrow{a,\lambda_3} & (STOP, \langle \lambda_5, \lambda_7, \ldots \rangle), \\ (P, \langle \lambda_1, \lambda_2, \ldots \rangle) & \xrightarrow{\tau,\lambda_2} & (b \to STOP, \langle \lambda_4, \lambda_6, \ldots \rangle) \\ & \xrightarrow{b,\lambda_4} & (STOP, \langle \lambda_6, \lambda_8, \ldots \rangle). \end{array}$$

If we consider a demon $\leqslant$ such that $\lambda_1 > \lambda_2$ then:

$$(P, \langle \lambda_1, \lambda_2, \ldots \rangle) \overset{\tau}{\underset{\leqslant}{\hookrightarrow}} (a \to STOP, \langle \lambda_3, \lambda_5, \ldots \rangle) \overset{a}{\underset{\leqslant}{\hookrightarrow}} (STOP, \langle \lambda_5, \lambda_7, \ldots \rangle)$$

(and no other transitions). Informally, the demon selects the left hand side of the nondeterministic choice. Similarly, if $\lambda_2 > \lambda_1$ then the demon selects the right hand side of the nondeterministic choice. $\square$

Example 17 Consider

$$P \ \widehat{=}\ a \to STOP \,\square\, b \to STOP.$$

This has the following operational semantics with respect to the initial sequence of markers $\langle \lambda_1, \lambda_2, \ldots \rangle$:

$$\begin{array}{l} (P, \langle \lambda_1, \lambda_2, \ldots \rangle) \xrightarrow{a,\lambda_1} (STOP, \langle \lambda_3, \lambda_5, \ldots \rangle), \\ (P, \langle \lambda_1, \lambda_2, \ldots \rangle) \xrightarrow{b,\lambda_2} (STOP, \langle \lambda_4, \lambda_6, \ldots \rangle). \end{array}$$

If we consider *any* demon $\leqslant$ then *both* transitions are available:

$$\begin{array}{l} (P, \langle \lambda_1, \lambda_2, \ldots \rangle) \overset{a}{\underset{\leqslant}{\hookrightarrow}} (STOP, \langle \lambda_3, \lambda_5, \ldots \rangle), \\ (P, \langle \lambda_1, \lambda_2, \ldots \rangle) \overset{b}{\underset{\leqslant}{\hookrightarrow}} (STOP, \langle \lambda_4, \lambda_6, \ldots \rangle). \end{array}$$

The choice is resolved by the environment, not the demon. $\square$

Example 18 Consider

$$P \ \widehat{=}\ a \to STOP \rhd b \to STOP.$$

This has the following operational semantics with respect to the initial sequence of markers $\langle \lambda_1, \lambda_2, \ldots \rangle$:

$$\begin{array}{lll} (P, \langle \lambda_1, \lambda_2, \ldots \rangle) & \xrightarrow{a,\lambda_1} & (STOP, \langle \lambda_3, \lambda_5, \ldots \rangle), \\ (P, \langle \lambda_1, \lambda_2, \ldots \rangle) & \xrightarrow{\tau,\lambda_2} & (b \to STOP, \langle \lambda_4, \lambda_6, \ldots \rangle) \\ & \xrightarrow{b,\lambda_4} & (STOP, \langle \lambda_6, \lambda_8, \ldots \rangle). \end{array}$$

If we consider a demon $\leqslant$ such that $\lambda_2 > \lambda_1$ then:

$$\begin{array}{lll} (P, \langle \lambda_1, \lambda_2, \ldots \rangle) & \underset{\leqslant}{\overset{\tau}{\hookrightarrow}} & (b \to STOP, \langle \lambda_4, \lambda_6, \ldots \rangle) \\ & \underset{\leqslant}{\overset{b}{\hookrightarrow}} & (STOP, \langle \lambda_6, \lambda_8, \ldots \rangle). \end{array}$$

Informally, this demon forces the timeout to occur, without allowing the environment the chance to perform the a.

Alternatively, if we consider a demon such that $\lambda_1 > \lambda_2$, we have two possible transitions from the initial state:

$$(P, \langle \lambda_1, \lambda_2, \ldots \rangle) \underset{\leqslant}{\overset{a}{\hookrightarrow}} (STOP, \langle \lambda_3, \lambda_5, \ldots \rangle),$$
$$(P, \langle \lambda_1, \lambda_2, \ldots \rangle) \underset{\leqslant}{\overset{\tau}{\hookrightarrow}} (b \to STOP, \langle \lambda_4, \lambda_6, \ldots \rangle) \underset{\leqslant}{\overset{b}{\hookrightarrow}} (STOP, \langle \lambda_6, \lambda_8, \ldots \rangle).$$

The demon allows the environment to perform the a; however, the timeout may also occur, in the case where the environment does not offer an a.

Note that the latter pair of transitions represents a form of nondeterminism that remains despite the fact that we have explicitly modelled the demon. It is necessary to allow both transitions in the model when considering information flow: if the event a is a high level event, then we would expect *High* to be able to influence whether or not the timeout occurs by either not offering or offering this event (respectively); if b is a low level event, this will allow information flow from *High* to *Low*. □

The following two lemmas compare the relations $\longrightarrow$ and $\hookrightarrow$:

Lemma 6 Let $P, P' \in CSP$, $\Lambda, \Lambda' \in MarkerSeq$ and $a \in \Sigma^\tau$; then

$$(\exists \leqslant \in Demon \bullet (P, \Lambda) \underset{\leqslant}{\overset{a}{\hookrightarrow}} (P', \Lambda'))$$
$$\Leftrightarrow$$
$$(\exists \mu \in CpdMarker \bullet (P, \Lambda) \xrightarrow{a,\mu} (P', \Lambda')).$$

Note that the above result holds for both visible and internal events.

Lemma 7 Let $P \in CSP$, $\Lambda \in MarkerSeq$; then

$$(\exists \mu \bullet (P, \Lambda) \xrightarrow{\tau,\mu}) \Rightarrow (\forall \leqslant \in Demon \bullet (P, \Lambda) \underset{\leqslant}{\overset{\tau}{\hookrightarrow}}).$$

The following lemma relates the $\hookrightarrow$ relation to the standard operational semantics:

Lemma 8

- If $(P, \Lambda) \underset{\leqslant}{\overset{a}{\hookrightarrow}} (P', \Lambda')$ then $P \xrightarrow{a} P'$;

- If $P \xrightarrow{a} P'$, then $\forall \Lambda \bullet \exists \leqslant, \Lambda' \bullet (P, \Lambda) \underset{\leqslant}{\overset{a}{\hookrightarrow}} (P', \Lambda')$;
- If $P \xrightarrow{\tau}$, then $\forall \Lambda, \leqslant \bullet (P, \Lambda) \underset{\leqslant}{\overset{\tau}{\hookrightarrow}}$.

6.2 Multi-step transitions

We now define two multi-step transition relations. The first such relation includes internal τ actions. If $s = \langle a_0, a_1, \ldots, a_{n-1} \rangle \in (\Sigma^\tau)^*$ is a sequence of (possibly internal) actions, then we write

$$(P, \Lambda) \underset{\leqslant}{\overset{s}{\longmapsto}} (Q, \Lambda')$$

if there exist $P_0 = P$, P_1, ..., $P_n = Q$ and $\Lambda_0 = \Lambda$, Λ_1, ..., $\Lambda_n = \Lambda'$ such that

$$(P_k, \Lambda_k) \underset{\leqslant}{\overset{a_k}{\hookrightarrow}} (P_{k+1}, \Lambda_{k+1}) \qquad \text{for each } k \in \{0, \ldots, n-1\}.$$

The next multi-step transition relation is in terms of visible events, only. We write

$$(P, \Lambda) \underset{\leqslant}{\overset{tr}{\Longrightarrow}} (Q, \Lambda')$$

if there exists s such that $(P, \Lambda) \underset{\leqslant}{\overset{s}{\longmapsto}} (Q, \Lambda')$ and $s \setminus \{\tau\} = tr$.

The following lemma relates this relation to the corresponding one from Section 2.2:

Lemma 9 If $(P, \Lambda) \underset{\leqslant}{\overset{tr}{\Longrightarrow}} (P', \Lambda')$ then $P \overset{tr}{\Longrightarrow} P'$. Conversely, if $P \overset{tr}{\Longrightarrow} P'$, then for every Λ, there exist $\leqslant, \Lambda'$ such that $(P, \Lambda) \underset{\leqslant}{\overset{tr}{\Longrightarrow}} (P', \Lambda')$.

6.3 Denotational information

The semantics so far has been operational in nature. We now extract some denotational information from the operational semantics.

We say that a process P is *stable* in the presence of demon $\leqslant$ and marker sequence Λ, written $stable_{\Lambda,\leqslant}\, P$, if it cannot perform any internal transitions:

$$stable_{\Lambda,\leqslant}\, P \;\widehat{=}\; \neg (P, \Lambda) \underset{\leqslant}{\overset{\tau}{\hookrightarrow}} .$$

We say that a process *refuses* X (where X is a set of visible events), written $P\; ref_{\Lambda,\leqslant}\; X$, if the process is stable and cannot perform any events from X:

$$P\; ref_{\Lambda,\leqslant}\; X \;\Leftrightarrow\; stable_{\Lambda,\leqslant}\, P \wedge \forall\, x \in X \bullet \neg (P, \Lambda) \underset{\leqslant}{\overset{x}{\hookrightarrow}} .$$

We say that a process *diverges* immediately, written $div_{\Lambda,\leqslant} P$, if it can perform an infinite sequence of internal events:

$$\begin{aligned} div_{\Lambda,\leqslant} P \;\Leftrightarrow\; & \exists\, P_0 = P, P_1, \ldots ; \Lambda_0 = \Lambda, \Lambda_1, \ldots \bullet \\ & \forall\, n \in \mathbb{N} \bullet (P_n, \Lambda_n) \underset{\leqslant}{\overset{\tau}{\hookrightarrow}} (P_{n+1}, \Lambda_{n+1}). \end{aligned}$$

The following two lemmas relate these operators to the corresponding operators from Section [Ros97] and Section 2.2:

Lemma 10

$$\begin{aligned} stable\,P \;&\Rightarrow\; \forall\,\Lambda, \leqslant \bullet\, stable_{\Lambda,\leqslant} P, \\ P\ ref\ X \;&\Rightarrow\; \forall\,\Lambda, \leqslant \bullet\, P\ ref_{\Lambda,\leqslant}\ X, \\ div\,P \;&\Rightarrow\; \forall\,\Lambda \bullet \exists \leqslant \bullet\, div_{\Lambda,\leqslant} P. \end{aligned}$$

Proof: The first two results follow directly from the definitions and Lemma 8.

For the third case, suppose $div\,P$, and fix Λ. From the definition of divergence, there exist $P_0 = P, P_1, \ldots$ such that $P_n \xrightarrow{\tau} P_{n+1}$ for each $n \in \mathbb{N}$. We construct a demon to select the corresponding transitions in our model. From Lemma 8, for each $n \in \mathbb{N}$:

$$\forall\,\Lambda \bullet \exists \leqslant, \Lambda' \bullet (P_n, \Lambda) \underset{\leqslant}{\overset{\tau}{\hookrightarrow}} (P_{n+1}, \Lambda').$$

Define $\Lambda_0 \mathrel{\widehat{=}} \Lambda$, and inductively define $\leqslant_n$ and Λ_{n+1} such that:

$$(P_n, \Lambda_n) \underset{\leqslant_n}{\overset{\tau}{\hookrightarrow}} (P_{n+1}, \Lambda_{n+1}).$$

These τ-transitions are potentially all in terms of different demons, so we now unify them. Let the corresponding transitions with markers be

$$(P_n, \Lambda_n) \xrightarrow{\tau, \mu_n} (P_{n+1}, \Lambda_{n+1}).$$

Note that by Lemma 3, the markers in μ_n will not appear on any other transition from a configuration (P_m, Λ_m) for $m > n$. Let demon $\leqslant$ be such that the μ_n (for $n \in \mathbb{N}$) are at the top of the order, in order; formally:

$$\mu_0 > \mu_1 > \ldots \text{ and if } \mu \notin \{\mu_k \mid k \in \mathbb{N}\} \text{ then } \forall\, n \in \mathbb{N} \bullet \mu < \mu_n\,.$$

Then by construction this demon selects all the required transitions:

$$\forall\, n \in \mathbb{N} \bullet (P_n, \Lambda_n) \underset{\leqslant}{\overset{\tau}{\hookrightarrow}} (P_{n+1}, \Lambda_{n+1}).$$

So $div_{\Lambda,\leqslant} P$, as required. $\square$

Lemma 11 For marker sequence Λ and demon $\leqslant$:

$$\begin{aligned} stable_{\Lambda,\leqslant} P \;&\Rightarrow\; stable\,P, \\ P\ ref_{\Lambda,\leqslant}\ X \;&\Rightarrow\; P\ ref\ X, \\ div_{\Lambda,\leqslant} P \;&\Rightarrow\; div\,P. \end{aligned}$$

Proof: The first and third equations follow directly from Lemma 8. For the second equation:

$$
\begin{array}{ll}
 & P\ ref_{\Lambda,\leqslant}\ X \\
\Rightarrow & \langle \text{definition} \rangle \\
 & stable_{\Lambda,\leqslant}\ P \wedge \forall\, x \in X \bullet \neg (P, \Lambda) \xhookrightarrow[\leqslant]{x} \\
\Rightarrow & \left\langle \text{first equation; defining rule for } \hookrightarrow \right\rangle \\
 & stable\, P \wedge \forall\, x \in X \bullet \forall\, \mu \bullet \neg (P, \Lambda) \xrightarrow{x,\mu} \\
\Rightarrow & \langle \text{Lemma 1} \rangle \\
 & stable\, P \wedge \forall\, x \in X \bullet \neg P \xrightarrow{x} \\
\Rightarrow & \langle \text{definition} \rangle \\
 & P\ ref\ X.
\end{array}
$$

□

We use the above operators to define analogues of the standard denotational functions over CSP:

$$
\begin{array}{rcl}
traces_{\Lambda,\leqslant}(P) & \widehat{=} & \{tr \mid (P, \Lambda) \overset{tr}{\underset{\leqslant}{\Longrightarrow}}\}, \\
failures_{\Lambda,\leqslant}(P) & \widehat{=} & \{(tr, X) \mid \exists\, P', \Lambda' \bullet (P, \Lambda) \overset{tr}{\underset{\leqslant}{\Longrightarrow}} (P', \Lambda') \\
 & & \qquad\qquad\qquad\qquad \wedge\ P'\ ref_{\Lambda',\leqslant}\ X\}, \\
divergences_{\Lambda,\leqslant}(P) & \widehat{=} & \{tr^\frown tr' \mid \exists\, P', \Lambda' \bullet (P, \Lambda) \overset{tr}{\underset{\leqslant}{\Longrightarrow}} (P', \Lambda') \wedge div_{\Lambda',\leqslant}\ P'\}, \\
failures^{\perp}_{\Lambda,\leqslant}(P) & \widehat{=} & failures_{\Lambda,\leqslant}(P) \cup \{(tr, X) \mid tr \in divergences_{\Lambda,\leqslant}(P)\}.
\end{array}
$$

The following theorem relates this semantics to the standard denotational semantic models:

Theorem 12

$$
\begin{array}{rcl}
traces(P) & = & \bigcup\{traces_{\Lambda,\leqslant}(P) \mid \Lambda \in MarkerSeq, \leqslant \in Demon\}, \\
failures(P) & = & \bigcup\{failures_{\Lambda,\leqslant}(P) \mid \Lambda \in MarkerSeq, \leqslant \in Demon\}, \\
divergences(P) & = & \bigcup\{divergences_{\Lambda,\leqslant}(P) \mid \Lambda \in MarkerSeq, \leqslant \in Demon\}, \\
failures^{\perp}(P) & = & \bigcup\{failures^{\perp}_{\Lambda,\leqslant}(P) \mid \Lambda \in MarkerSeq, \leqslant \in Demon\}.
\end{array}
$$

Proof: We prove the results for stable failures and divergences; the result for traces is similar to that for stable failures; the result for unstable failures follows from the results for stable failures and divergences.

The proof for failures is as follows:

$$
\begin{array}{ll}
 & \bigcup\{failures_{\Lambda,\leqslant}(P) \mid \Lambda \in MarkerSeq, \leqslant \in Demon\} \\
= & \{(tr, X) \mid \exists\, \Lambda, \leqslant \bullet\ \exists\, P', \Lambda' \bullet (P, \Lambda) \overset{tr}{\underset{\leqslant}{\Longrightarrow}} (P', \Lambda') \wedge P'\ ref_{\Lambda',\leqslant}\ X\}
\end{array}
$$

$$
\begin{aligned}
&= \left\langle \begin{array}{l}\text{Lemmas 9 and 11 in the forward direction;}\\ \text{Lemmas 9 and 10 in the reverse direction}\end{array}\right\rangle \\
&\quad \{(tr, X) \mid \exists\, P' \bullet P \stackrel{tr}{\Longrightarrow} P' \wedge P' \ ref\ X\} \\
&= \mathit{failures}(P).
\end{aligned}
$$

For divergences, we proceed via a number of intermediate results, which can be used to unify two demons involved in a divergence.

1. First, we show that if $(P, \Lambda) \underset{\leqslant_1}{\stackrel{a}{\hookrightarrow}} (P', \Lambda') \underset{\leqslant_2}{\stackrel{s}{\longmapsto}} (P'', \Lambda'') \wedge div_{\Lambda'', \leqslant_2} P''$, then there exists $\leqslant$ such that $(P, \Lambda) \underset{\leqslant}{\stackrel{\langle a \rangle ^\frown s}{\longmapsto}} (P'', \Lambda'') \wedge div_{\Lambda'', \leqslant} P''$. Suppose that the first transition in the hypothesis corresponds to marker μ:

   $$(P, \Lambda) \xrightarrow{a, \mu} (P', \Lambda').$$

   Note that from Lemma 3, the markers in μ are not used to label any subsequent transitions from P'. Let demon $\leqslant$ be defined by taking $\leqslant_2$ and moving μ to the top of the order; formally:

   $$\mu' \leqslant \mu'' \Leftrightarrow \mu'' = \mu \vee \mu' \leqslant_2 \mu'' \wedge \mu' \neq \mu \neq \mu''\,.$$

   Then by construction $(P, \Lambda) \underset{\leqslant}{\stackrel{a}{\hookrightarrow}} (P', \Lambda') \underset{\leqslant}{\stackrel{s}{\longmapsto}} (P'', \Lambda'') \wedge div_{\Lambda'', \leqslant} P''$.

2. We can then show that if $(P, \Lambda) \underset{\leqslant_1}{\stackrel{s}{\longmapsto}} (P', \Lambda') \wedge div_{\Lambda', \leqslant_2} P'$ then there exists $\leqslant$ such that $(P, \Lambda) \underset{\leqslant}{\stackrel{s}{\longmapsto}} (P', \Lambda') \wedge div_{\Lambda', \leqslant} P'$. The proof of this result is a straightforward induction on the length of s, using the previous result in the inductive step.

3. We then immediately obtain that if $(P, \Lambda) \underset{\leqslant_1}{\stackrel{tr}{\Longrightarrow}} (P', \Lambda') \wedge div_{\Lambda', \leqslant_2} P'$ then there exists $\leqslant$ such that $(P, \Lambda) \underset{\leqslant}{\stackrel{tr}{\Longrightarrow}} (P', \Lambda') \wedge div_{\Lambda', \leqslant} P'$.

4. Finally, we can prove the main result:

   $$
   \begin{aligned}
   &\bigcup \{ divergences_{\Lambda, \leqslant}(P) \mid \Lambda \in MarkerSeq, \leqslant \in Demon\} \\
   &= \{tr^\frown tr' \mid \exists\, \Lambda, \leqslant \bullet \exists\, P', \Lambda' \bullet (P, \Lambda) \underset{\leqslant}{\stackrel{tr}{\Longrightarrow}} (P', \Lambda') \wedge div_{\Lambda', \leqslant} P'\} \\
   &= \left\langle \begin{array}{l}\text{Lemmas 9, 11 for the forward direction;}\\ \text{Lemmas 9, 10 and the above result for the reverse direction}\end{array}\right\rangle \\
   &\quad \{tr^\frown tr' \mid \exists\, P' \bullet P \stackrel{tr}{\Longrightarrow} P' \wedge div\ P'\} \\
   &= divergences(P).
   \end{aligned}
   $$

□

Note that the quantifications over Λ in the set comprehensions of the above theorem are unnecessary, because of Lemma 2; for example

$$traces(P) = \bigcup\{traces_{\Lambda,\leqslant}(P) \mid \leqslant \in Demon\} \qquad \text{for all } \Lambda \in MarkerSeq.$$

Of course, we don't necessarily have $traces(P) = traces_{\Lambda,\leqslant}(P)$ (and hence likewise for failures and divergences): consider the process Q_1 in Figure 3 with a demon $\leqslant$ such that $\lambda_5 > \lambda_3$; the trace $\langle h, l\rangle$ is in $traces(Q_1)$ but not in $traces_{\Lambda,\leqslant}(Q_1)$.

We will need the following property of the failures of a process. If a divergence-free process is unable to perform events from some set X, then it can refuse X:

Lemma 13 If P is divergence-free, and

$$tr \in traces_{\Lambda,\leqslant}(P) \wedge \forall\, x \in X \bullet tr^\frown\langle x\rangle \notin traces_{\Lambda,\leqslant}(P)$$

then

$$(tr, X) \in failures_{\Lambda,\leqslant}(P).$$

6.4 Refinement and equivalence

We can define notions of refinement between processes. We say that P is refined by Q if for every way of resolving the nondeterminism of Q, there is some way of resolving the nondeterminism of P such that the two processes have the same behaviour; that is, for every demon for Q it is possible to find a demon for P such that the two processes have identical behaviours. We define two notions of refinement, corresponding to stable failures and to failures-divergences[7].

Definition 7

$$\begin{aligned}
P \sqsubseteq^{IF}_{F} Q \;\widehat{=}\; \forall\, \Lambda, \leqslant_Q \bullet \exists \leqslant_P \bullet\; & traces_{\Lambda,\leqslant_P}(P) = traces_{\Lambda,\leqslant_Q}(Q) \wedge \\
& failures_{\Lambda,\leqslant_P}(P) = failures_{\Lambda,\leqslant_Q}(Q), \\
P \sqsubseteq^{IF}_{FD} Q \;\widehat{=}\; \forall\, \Lambda, \leqslant_Q \bullet \exists \leqslant_P \bullet\; & failures^{\perp}_{\Lambda,\leqslant_P}(P) = failures^{\perp}_{\Lambda,\leqslant_Q}(Q) \wedge \\
& divergences_{\Lambda,\leqslant_P}(P) = divergences_{\Lambda,\leqslant_Q}(Q).
\end{aligned}$$

We prove that these refinement relations satisfy some standard conditions.

Lemma 14 Nondeterministic choices are refined by either of their components:

$$P \sqcap Q \sqsubseteq P \qquad \text{and} \qquad P \sqcap Q \sqsubseteq Q,$$

where the refinement relation is either of the relations defined above.

Proof: We sketch the proof of the first statement. Pick a marker sequence $\Lambda = \langle\lambda_1, \lambda_2, \ldots\rangle$, and a demon $\leqslant$ for P. Note that

$$(P \sqcap Q, \Lambda) \xrightarrow{\tau,\lambda_1} (P, \Lambda') \qquad \text{where} \qquad \Lambda' = \langle\lambda_3, \lambda_5, \ldots\rangle.$$

[7] Out refinement relations have the superscripts "*IF*", for "information flow", to distinguish them from the refinement relations in the standard models.

Define demon $\leqslant'$ to select this transition, i.e. $\lambda_1 \geqslant' \lambda_2$, and then to act in the same way over the resulting configuration (P, Λ') as $\leqslant$ acts over (P, Λ); that is:

$$\forall \mu, \mu' \bullet \mu \leqslant \mu' \Leftrightarrow \phi(\mu) \leqslant' \phi(\mu')$$

where the function ϕ maps a marker obtained from Λ to the corresponding marker obtained from Λ':

$$\phi(\lambda_k) = \lambda_{2k+1},$$

extended pointwise to compound markers (as in Lemma 2). Then the graph of the $\underset{\leqslant'}{\hookrightarrow}$ relation for $(P \sqcap Q, \Lambda)$ is identical to that of the $\underset{\leqslant}{\hookrightarrow}$ relation for (P, Λ), except for an additional initial τ-transition. Hence

$$traces_{\Lambda,\leqslant'}(P \sqcap Q) = traces_{\Lambda,\leqslant}(P),$$

because the same traces of visible events are possible in both graphs, and the denotational semantics ignores the initial τ-transition of $P \sqcap Q$. Similarly, the stable failures, unstable failures and divergences are equal. Hence $P \sqcap Q \sqsubseteq P$.
$\square$

Lemma 15 $P \sqsubseteq Q$ if and only if $P \sqsubseteq P \sqcap Q$.

The proof is similar to that of the previous lemma.

We can use the above definitions of refinement to define equivalences between process:

$$\begin{aligned} P \equiv_F^{IF} Q &\mathrel{\widehat{=}} P \sqsubseteq_F^{IF} Q \wedge Q \sqsubseteq_F^{IF} P, \\ P \equiv_{FD}^{IF} Q &\mathrel{\widehat{=}} P \sqsubseteq_{FD}^{IF} Q \wedge Q \sqsubseteq_{FD}^{IF} P. \end{aligned}$$

The following corollary follows immediately from the previous two lemmas, and gives an alternative characterisation of refinement:

Corollary 16

$$P \sqsubseteq Q \Leftrightarrow P \equiv P \sqcap Q.$$

The following theorem relates refinement in this model to refinement in the standard models:

Theorem 17 If P is refined by Q in our model, then the same is true in the standard model:

$$\begin{aligned} &P \sqsubseteq_F^{IF} Q \Rightarrow P \sqsubseteq_F Q, \\ &P \sqsubseteq_{FD}^{IF} Q \Rightarrow P \sqsubseteq_{FD} Q. \end{aligned}$$

Proof: Firstly, suppose $P \sqsubseteq_F^{IF} Q$. Then

$$\begin{array}{cl} & \mathit{failures}(Q) \\ = & \langle \text{Lemma 12} \rangle \\ & \bigcup \{\mathit{failures}_{\Lambda,\leqslant_Q}(Q) \mid \Lambda \in \mathit{MarkerSeq}, \leqslant_Q \in \mathit{Demon}\} \\ \subseteq & \langle \text{from the assumption, and the definition of refinement} \rangle \\ & \bigcup \{\mathit{failures}_{\Lambda,\leqslant_P}(P) \mid \Lambda \in \mathit{MarkerSeq}, \leqslant_P \in \mathit{Demon}\} \\ = & \langle \text{Lemma 12} \rangle \\ & \mathit{failures}(P). \end{array}$$

Similarly, $\mathit{traces}(Q) = \mathit{traces}(P)$. Hence $P \sqsubseteq_F Q$.

Similarly, if $P \sqsubseteq_{FD}^{IF} Q$ we can show that $\mathit{failures}^{\perp}(Q) \subseteq \mathit{failures}^{\perp}(P)$ and $\mathit{divergences}(Q) \subseteq \mathit{divergences}(P)$, and hence $P \sqsubseteq_{FD} Q$. □

As an immediate corollary, our notions of equivalence are more discriminating than the standard notions of equivalence.

Corollary 18

$$P \equiv_F^{IF} Q \;\Rightarrow\; P \equiv_F Q,$$
$$P \equiv_{FD}^{IF} Q \;\Rightarrow\; P \equiv_{FD} Q.$$

Our equivalences are *strictly* more discriminating than the standard ones, as shown by the processes Q_1 and Q_2 from Examples 6 and 12: these are equivalent in the standard model, but distinct in our model.

7 Nondeterministic local noninterference

We are now ready to return to the question of information flow. We use the semantic model of the previous sections to produce a formal definition of information flow. We restrict ourself to divergence-free processes throughout the rest of the paper.

Our definition of information flow is a straightforward adaptation of Forster's Local Noninterference property:

Definition 8 We say that divergence-free process P satisfies *nondeterministic local noninterference* (NLNI) if

$$\forall\, Q \in \mathit{States}(P) \bullet \forall\, \Lambda, \Lambda', \leqslant \bullet \forall\, h \in H \bullet$$
$$\text{if } (Q, \Lambda) \xhookrightarrow[\leqslant]{h} (Q', \Lambda') \text{ then } \mathit{failures}_{\Lambda,\leqslant}(Q\backslash_H) = \mathit{failures}_{\Lambda',\leqslant}(Q'\backslash_H),$$

where $\mathit{States}(P)$ is the set of states that P can reach (for some demon).

If a process can evolve via a high-level action, then this should not change the low-level user's view of the system. Note the quantification over all demons: this means that if there is any way of resolving the nondeterminism such that information is passed, then the process should be considered insecure.

7.1 Examples

We now look at how our definition relates to some of the examples from the introduction.

Example 19 Consider again the process:

$$Q_1 \mathrel{\widehat{=}} h \rightarrow (l \rightarrow STOP \sqcap STOP) \,\Box\, (l \rightarrow STOP \sqcap STOP)$$

from Example 6, which the invariance property mis-classifies as secure. The operational semantics for this process, with respect to the marker sequence $\langle \lambda_1, \lambda_2, \ldots \rangle$, is in Figure 3. Consider a demon $\leqslant$ such that $\lambda_3 > \lambda_5$ and $\lambda_4 > \lambda_2$, i.e. a demon that resolves the first nondeterministic choice to the left, and the second to the right. Following the definition of NLNI, define:

$$Q = Q_1, \qquad Q' = l \rightarrow STOP \sqcap STOP, \qquad \Lambda' = \langle \lambda_3, \lambda_5, \ldots \rangle.$$

Then

$$\begin{array}{rcl} (Q, \Lambda) & \xhookrightarrow[\leqslant]{h} & (Q', \Lambda'), \\ (\langle l \rangle, \{\}) & \in & \mathit{failures}_{\Lambda', \leqslant}(Q' \backslash_H), \\ (\langle l \rangle, \{\}) & \notin & \mathit{failures}_{\Lambda, \leqslant}(Q \backslash_H). \end{array}$$

Hence this process does not satisfy NLNI. $\Box$

Example 20 Consider next

$$Q_2 \mathrel{\widehat{=}} (h \rightarrow l \rightarrow STOP \,\Box\, l \rightarrow STOP) \sqcap h \rightarrow STOP,$$

from Example 12, whose transition diagram, with initial marker sequence $\Lambda = \langle \lambda_1, \lambda_2, \ldots \rangle$, is in Figure 4. To prove this satisfies nondeterministic local noninterference, we consider the two h-transitions separately.

- Firstly, consider a demon $\leqslant$ that selects the left hand side of the nondeterministic choice, and let

  $$\begin{array}{rclcrcl} Q & = & h \rightarrow l \rightarrow STOP \,\Box\, l \rightarrow STOP, & \qquad & Q' & = & l \rightarrow STOP, \\ \Lambda' & = & \langle \lambda_3, \lambda_5, \ldots \rangle, & & \Lambda'' & = & \langle \lambda_7, \lambda_{11}, \ldots \rangle. \end{array}$$

  Then

  $$(Q, \Lambda') \xhookrightarrow[\leqslant]{h} (Q', \Lambda'') \;\text{ and }\; \mathit{failures}_{\Lambda', \leqslant}(Q \backslash_H) = \mathit{failures}_{\Lambda'', \leqslant}(Q' \backslash_H).$$

- Alternatively, consider a demon that selects the right hand side of the nondeterministic choice, and let

  $$\begin{array}{rclcrcl} Q & = & h \rightarrow STOP, & \qquad & Q' & = & STOP, \\ \Lambda' & = & \langle \lambda_2, \lambda_4, \ldots \rangle, & & \Lambda'' & = & \langle \lambda_4, \lambda_6, \ldots \rangle. \end{array}$$

  Then

  $$(Q, \Lambda') \xhookrightarrow[\leqslant]{h} (Q', \Lambda'') \;\text{ and }\; \mathit{failures}_{\Lambda', \leqslant}(Q \backslash_H) = \mathit{failures}_{\Lambda'', \leqslant}(Q' \backslash_H).$$

$\Box$

Finally, we consider a process that Ryan's definition identifies as insecure, but which we consider to be secure:

Example 21 Recall the process:

$l \to STOP \rhd h \to STOP$

from Example 5. This satisfies nondeterministic local noninterference, because the only h-transition is from $h \to STOP$ to $STOP$, and these processes have the same failures when H is blocked (regardless of the demon). $\Box$

8 Some properties of nondeterministic local noninterference

In this section, we prove a few results about nondeterministic local noninterference.

8.1 Comparison with lazy independence

In this subsection we show that our definition is strictly weaker than Roscoe's lazy independence property from Section 3.5. We will need the following result, comparing the failures of P with those of $P\backslash_A$.

Lemma 19 For all P, tr, A, X, Λ and $\leqslant$,

$tr \upharpoonright A = \langle\rangle \wedge (tr, X - A) \in \mathit{failures}_{\Lambda,\leqslant}(P)$
$\Leftrightarrow$
$(tr, X) \in \mathit{failures}_{\Lambda,\leqslant}(P\backslash_A)$.

The following lemma relates restriction in our model to lazy abstraction in the standard model:

Lemma 20 If $(tr, X) \in \mathit{failures}_{\Lambda,\leqslant}(P\backslash_H)$ then $(tr, X) \in \mathit{failures}(\mathcal{L}_H(P))$.

Proof: If $(tr, X) \in \mathit{failures}_{\Lambda,\leqslant}(P\backslash_H)$ then $tr \setminus H = tr$ and $(tr, X \cap L) \in \mathit{failures}_{\Lambda,\leqslant}(P)$, from the previous lemma. Hence $(tr, X \cap L) \in \mathit{failures}(P)$ from Theorem 12, and so $(tr, X) \in \mathit{failures}(\mathcal{L}_H(P))$ from the definition of lazy abstraction. $\Box$

We can now prove the main theorem of this section.

Theorem 21 If $\mathcal{L}IND_H(P)$ (i.e. $\mathcal{L}_H(P)$ is deterministic), then P satisfies NLNI.

Proof: We prove the contrapositive. Suppose P does not satisfy NLNI. Then for some states Q and Q' of P, marker sequences Λ and Λ', and $h \in H$, we have

$$(Q, \Lambda) \xhookrightarrow[\leqslant]{h} (Q', \Lambda') \qquad \text{and} \qquad \mathit{failures}_{\Lambda,\leqslant}(Q\backslash_H) \neq \mathit{failures}_{\Lambda',\leqslant}(Q'\backslash_H).$$

We show that $\mathcal{L}_H(Q)$ is nondeterministic, which implies that $\mathcal{L}_H(P)$ is nondeterministic.

Suppose, firstly, that for some (tr, X) that $(tr, X) \in \mathit{failures}_{\Lambda,\leqslant}(Q\backslash_H)$, but $(tr, X) \notin \mathit{failures}_{\Lambda',\leqslant}(Q'\backslash_H)$, and consider a shortest such tr. There are two cases to consider:

- Case $tr \in \mathit{traces}_{\Lambda',\leqslant}(Q'\backslash_H)$. Now,

  $$(tr, X) \notin \mathit{failures}_{\Lambda',\leqslant}(Q'\backslash_H),$$

  so from Lemma 13, there is some $x \in X$ such that

  $$tr^\frown\langle x\rangle \in \mathit{traces}_{\Lambda',\leqslant}(Q'\backslash_H).$$

  Hence

  $$tr^\frown\langle x\rangle \in \mathit{traces}(\mathcal{L}_H(Q')),$$

  from Lemma 20, and so

  $$tr^\frown\langle x\rangle \in \mathit{traces}(\mathcal{L}_H(Q)),$$

  using the definition of lazy abstraction. Also

  $$(tr, \{x\}) \in \mathit{failures}_{\Lambda',\leqslant}(Q\backslash_H),$$

  (because our model satisfies the property that if (tr, X) is a failure of a process, then so is (tr, Y) for every $Y \subseteq X$, as in the standard failures model) and so

  $$(tr, \{x\}) \in \mathit{failures}(\mathcal{L}_H(Q)),$$

  again from Lemma 20. Hence $\mathcal{L}_H(Q)$ is nondeterministic.

- Case $tr \notin \mathit{traces}_{\Lambda',\leqslant}(Q'\backslash_H)$. Let $tr = tr'^\frown\langle a\rangle$. By the assumed minimality of tr,

  $$tr' \in \mathit{traces}_{\Lambda',\leqslant}(Q'\backslash_H),$$

  so from Lemma 13,

  $$(tr', \{a\}) \in \mathit{failures}_{\Lambda',\leqslant}(Q'\backslash_H).$$

  Hence

  $$(tr', \{a\}) \in \mathit{failures}(\mathcal{L}_H(Q'))$$

  from Lemma 20, and so

  $$(tr', \{a\}) \in \mathit{failures}(\mathcal{L}_H(Q)).$$

  But

  $$tr = tr'^\frown\langle a\rangle \in \mathit{traces}_{\Lambda,\leqslant}(Q\backslash_H)$$

  so

  $$tr'^\frown\langle a\rangle \in \mathit{traces}(\mathcal{L}_H(Q))$$

  again from Lemma 20. Hence $\mathcal{L}_H(Q)$ is again nondeterministic.

The case where $(tr, X) \in \mathit{failures}_{\Lambda',\leqslant}(Q', \backslash_H)$ but $(tr, X) \notin \mathit{failures}_{\Lambda,\leqslant}(Q\backslash_H)$ is similar. $\square$

That our definition is strictly weaker than Roscoe's is shown by the process P_3 from Example 10.

8.2 Comparisons with failures-divergences local noninterference

We now show that our definition of security is stronger than Forster's failures-divergences local noninterference property.

Theorem 22 If P satisfies NLNI, then it satisfies SLNI.

Proof: We prove the contrapositive. Suppose P does not satisfy SLNI. Then there exists some $Q, Q' \in \mathit{States}(P),\ h \in H$, such that

$$Q \xrightarrow{h} Q' \quad \text{and} \quad \mathit{failures}(Q\backslash_H) \neq \mathit{failures}(Q'\backslash_H).$$

Pick Λ; then from Lemma 1:

$$(Q, \Lambda) \xrightarrow{h,\mu} (Q', \Lambda'),$$

for some Λ', μ. Also, from Theorem 12, there is some demon $\leqslant$ such that:

$$\mathit{failures}_{\Lambda,\leqslant}(Q\backslash_H) \neq \mathit{failures}_{\Lambda',\leqslant}(Q'\backslash_H).$$

Define $\leqslant'$ so as to select the h, μ transition to Q', then to act like $\leqslant$. So

$$(Q, \Lambda) \underset{\leqslant'}{\overset{h}{\hookrightarrow}} (Q', \Lambda')$$

and

$$\begin{aligned} \mathit{failures}_{\Lambda,\leqslant'}(Q\backslash_H) &= \mathit{failures}_{\Lambda,\leqslant}(Q\backslash_H) \\ &\neq \mathit{failures}_{\Lambda',\leqslant}(Q'\backslash_H) \\ &= \mathit{failures}_{\Lambda',\leqslant'}(Q'\backslash_H). \end{aligned}$$

So P does not satisfy NLNI. $\square$

Our definition is strictly stronger than Forster's, as shown by process $Q1$ from Examples 15 and 19.

8.3 High and low processes

We now state two results which say that if either *High* or *Low* is unable to perform any events, then the process is secure.

Theorem 23

1. If $\alpha P \subseteq L$ then $\mathsf{NLNI}(P)$;
2. If $\alpha P \subseteq H$ then $\mathsf{NLNI}(P)$.

The first result is particularly worthy of note because it does not hold under Roscoe's definition of security, as shown by the process P_3 from Example 10.

8.4 Closure under nondeterministic choice

We now show that our property is closed under nondeterministic choices.

Theorem 24 If $\mathsf{NLNI}(P_1)$ and $\mathsf{NLNI}(P_2)$ then $\mathsf{NLNI}(P_1 \sqcap P_2)$.

Proof: Suppose $Q \in \mathit{States}(P_1 \sqcap P_2)$ and $(Q, \Lambda) \underset{\leqslant}{\overset{h}{\hookrightarrow}} (Q', \Lambda')$. Then $Q \neq P_1 \sqcap P_2$, so assume without loss of generality that $Q \in \mathit{States}(P_1)$. Then

$$\mathit{failures}_{\Lambda,\leqslant}(Q \backslash_H) = \mathit{failures}_{\Lambda',\leqslant}(Q' \backslash_H),$$

from $\mathsf{NLNI}(P_1)$, as required. $\square$

8.5 Compositionality

We now consider the subject of compositionality. In Section 8.5.1 we prove some results showing how behaviours of a parallel composition reflect the behaviours of the subcomponents. In Section 8.5.2 we do the opposite: we show how behaviours of the components reflect behaviours of a parallel composition. In Section 8.5.3 we compare the failures of a parallel composition with those of its components, and in Section 8.5.4 we show that our nondeterministic local noninterference property is closed under parallel composition with high level processes.

8.5.1 Compositional lemmas

In this section we state compositional results, i.e. results that show how behaviours of a parallel composition reflect behaviours of the subcomponents. We begin with the $\hookrightarrow$ relation.

Lemma 25

1. If $a \in A \cap B$, $(P_1, \Lambda_1) \underset{\leqslant}{\overset{a}{\hookrightarrow}} (P_1', \Lambda_1')$, and $(P_2, \Lambda_2) \underset{\leqslant}{\overset{a}{\hookrightarrow}} (P_2', \Lambda_2')$, then

$$(P_1 \ {}_A\|_B \ P_2, \mathsf{interleave}(\Lambda_1, \Lambda_2)) \underset{\leqslant}{\overset{a}{\hookrightarrow}} (P_1' \ {}_A\|_B \ P_2', \mathsf{interleave}(\Lambda_1', \Lambda_2')).$$

2. If $a \in A - B$, P_2 is non-divergent, and $(P_1, \Lambda_1) \underset{\leqslant}{\overset{a}{\hookrightarrow}}$, then

$$(P_1 \ {}_A\|_B \ P_2, \mathsf{interleave}(\Lambda_1, \Lambda_2)) \underset{\leqslant}{\overset{\tau}{\hookrightarrow}}{}^{*} \underset{\leqslant}{\overset{a}{\hookrightarrow}},$$

and similarly for transitions of P_2.

3. If $(P_1, \Lambda_1) \underset{\leqslant}{\overset{\tau}{\hookrightarrow}} (P_1', \Lambda_1')$, and P_2 is non-divergent, then

$$\begin{aligned}(P_1 \;{}_A\|_B\; P_2, \mathsf{interleave}(\Lambda_1, \Lambda_2)) &\underset{\leqslant}{\overset{\tau}{\hookrightarrow}}{}^{*} (P_1 \;{}_A\|_B\; P_2', \mathsf{interleave}(\Lambda_1, \Lambda_2')) \\ &\underset{\leqslant}{\overset{\tau}{\hookrightarrow}} (P_1' \;{}_A\|_B\; P_2', \mathsf{interleave}(\Lambda_1', \Lambda_2'))\end{aligned}$$

for some P_2' and Λ_2', and similarly for transitions of P_2.

The proof uses the conditions on demons from Definition 6.

However, it turns out that the above results cannot be extended to traces. One might expect that if $tr \in (A \cup B)^*$, $(P_1, \Lambda_1) \underset{\leqslant}{\overset{tr \upharpoonright A}{\Longrightarrow}} (P_1', \Lambda_1')$ and $(P_2, \Lambda_2) \underset{\leqslant}{\overset{tr \upharpoonright B}{\Longrightarrow}} (P_2', \Lambda_2')$ then $(P_1 \;{}_A\|_B\; P_2, \mathsf{interleave}(\Lambda_1, \Lambda_2)) \underset{\leqslant}{\overset{tr}{\Longrightarrow}} (P_1' \;{}_A\|_B\; P_2', \mathsf{interleave}(\Lambda_1', \Lambda_2'))$. The following example shows that this is not the case.

Example 22 Let

$P_1 \;\widehat{=}\; a \rightarrow STOP,$

$P_2 \;\widehat{=}\; b \rightarrow STOP \rhd STOP,$

and consider a demon that prefers the b event to the timeout, and prefers the timeout to the a. Then $P_1 \;{}_{\{a\}}\|_{\{b\}}\; P_2$ cannot perform the trace $\langle a, b\rangle$ (although it can perform $\langle b, a\rangle$), despite the fact that the two components can perform their parts of that trace, because the timeout would occur before the a, thus preventing the b from occurring.

This demon has a natural interpretation in a timed setting: the fact that the demon prefers b to the timeout means that the b is available for some time before the timeout occurs; the fact that the demon prefers the timeout to the a means that the a only becomes available *after* the timeout occurs; putting these together, we see that the trace $\langle a, b\rangle$ cannot occur. □

One can, however, obtain some results about traces in the case that one side of the parallel composition performs no events, as shown by the following lemma.

Lemma 26 If $tr \upharpoonright B = \langle\rangle$, $(P_1, \Lambda_1) \underset{\leqslant}{\overset{tr}{\Longrightarrow}} (P_1', \Lambda_1')$, and P_2 is non-divergent, then

$$(P_1 \;{}_A\|_B\; P_2, \mathsf{interleave}(\Lambda_1, \Lambda_2)) \underset{\leqslant}{\overset{tr}{\Longrightarrow}} (P_1' \;{}_A\|_B\; P_2', \mathsf{interleave}(\Lambda_1', \Lambda_2'))$$

for some P_2', and Λ_2'.

8.5.2 Decompositional lemmas

In this section, we state some results showing how behaviours of subcomponents reflect the behaviours of parallel compositions. We begin by considering the $\hookrightarrow$ relation.

Lemma 27

1. If $a \in A \cap B$, and
$$(P_1 \ {}_A\|_B \ P_2, \mathsf{interleave}(\Lambda_1, \Lambda_2)) \underset{\leqslant}{\overset{a}{\hookrightarrow}} (P_1' \ {}_A\|_B \ P_2', \mathsf{interleave}(\Lambda_1', \Lambda_2')),$$
then
$$(P_1, \Lambda_1) \underset{\leqslant}{\overset{a}{\hookrightarrow}} (P_1', \Lambda_1') \quad \text{and} \quad (P_2, \Lambda_2) \underset{\leqslant}{\overset{a}{\hookrightarrow}} (P_2', \Lambda_2').$$
2. If $a \in A - B$, and
$$(P_1 \ {}_A\|_B \ P_2, \mathsf{interleave}(\Lambda_1, \Lambda_2)) \underset{\leqslant}{\overset{a}{\hookrightarrow}} (P_1' \ {}_A\|_B \ P_2, \mathsf{interleave}(\Lambda_1', \Lambda_2)),$$
then
$$(P_1, \Lambda_1) \underset{\leqslant}{\overset{a}{\hookrightarrow}} (P_1', \Lambda_1'),$$
and similarly for transitions from $B - A$.
3. If
$$(P_1 \ {}_A\|_B \ P_2, \mathsf{interleave}(\Lambda_1, \Lambda_2)) \underset{\leqslant}{\overset{\tau}{\hookrightarrow}} (P_1' \ {}_A\|_B \ P_2', \mathsf{interleave}(\Lambda_1', \Lambda_2')),$$
then
$$(P_1, \Lambda_1) \underset{\leqslant}{\overset{\tau}{\hookrightarrow}} (P_1', \Lambda_1') \quad \text{and} \quad (P_2, \Lambda_2) = (P_2', \Lambda_2'),$$
or
$$(P_2, \Lambda_2) \underset{\leqslant}{\overset{\tau}{\hookrightarrow}} (P_2', \Lambda_2') \quad \text{and} \quad (P_1, \Lambda_1) = (P_1', \Lambda_1').$$

We now extend the above result to traces. The following lemma shows how a trace of a parallel composition is reflected in traces of the components.

Lemma 28 If
$$(P_1 \ {}_A\|_B \ P_2, \mathsf{interleave}(\Lambda_1, \Lambda_2)) \underset{\leqslant}{\overset{tr}{\Longrightarrow}} (P_1' \ {}_A\|_B \ P_2', \mathsf{interleave}(\Lambda_1', \Lambda_2')),$$
then:
$$(P_1, \Lambda_1) \underset{\leqslant}{\overset{tr \restriction A}{\Longrightarrow}} (P_1', \Lambda_1') \quad \text{and} \quad (P_2, \Lambda_2) \underset{\leqslant}{\overset{tr \restriction B}{\Longrightarrow}} (P_2', \Lambda_2').$$

8.5.3 Failures

We now extend the above results to failures. We first show how the failures of a parallel composition are reflected in the failures of the components.

Lemma 29 If $(tr, X) \in \mathit{failures}_{\mathsf{interleave}(\Lambda_1,\Lambda_2),\leqslant}(P_1 \ {}_A\|_B \ P_2)$, then
$$(tr \restriction A, X \cap (A - B)) \in \mathit{failures}_{\Lambda_1,\leqslant}(P_1),$$
$$(tr \restriction B, X \cap (B - A)) \in \mathit{failures}_{\Lambda_2,\leqslant}(P_2).$$

Proof: We just prove the first result. From the conditions of the lemma, for some P_1', P_2', Λ_1', Λ_2':

$$\begin{array}{l}(P_1 \;{}_A\|_B\; P_2, \mathsf{interleave}(\Lambda_1, \Lambda_2)) \underset{\leqslant}{\overset{tr}{\Longrightarrow}} (P_1' \;{}_A\|_B\; P_2', \mathsf{interleave}(\Lambda_1', \Lambda_2')) \\ \wedge\; stable_{\mathsf{interleave}(\Lambda_1', \Lambda_2'), \leqslant}(P_1' \;{}_A\|_B\; P_2') \\ \wedge\; \forall\, x \in X \bullet \neg(P_1' \;{}_A\|_B\; P_2', \mathsf{interleave}(\Lambda_1', \Lambda_2')) \underset{\leqslant}{\overset{x}{\hookrightarrow}} .\end{array}$$

From Lemma 28:

$$(P_1, \Lambda_1) \underset{\leqslant}{\overset{tr \restriction A}{\Longrightarrow}} (P_1', \Lambda_1').$$

From the definition of stability:

$$stable_{\Lambda_1', \leqslant}(P_1') \qquad \text{and} \qquad stable_{\Lambda_2', \leqslant}(P_2').$$

From Lemma 25:

$$\forall\, x \in X \cap (A - B) \bullet \neg(P_1', \Lambda_1') \underset{\leqslant}{\overset{x}{\hookrightarrow}} .$$

Hence

$$(tr \restriction A, X \cap (A - B)) \in failures_{\Lambda_1, \leqslant}(P_1).$$

□

We now show how the failures of a parallel composition reflects the failures of the subcomponents. As shown by Example 22, we do not have as strong a result as one might expect. However, we have the following result, dealing with the case when one side performs no events.

Lemma 30 If $tr \restriction B = \langle\rangle$, $X \cap B = \{\}$, Q_2 is non-divergent, and

$$(tr, X) \in failures_{\Lambda_1, \leqslant}(Q_1),$$

then

$$(tr, X) \in failures_{\mathsf{interleave}(\Lambda_1, \Lambda_2), \leqslant}(Q_1 \;{}_A\|_B\; Q_2).$$

Proof: From the assumptions of the lemma and the definition of failures, we have

$$(Q_1, \Lambda_1) \underset{\leqslant}{\overset{tr}{\Longrightarrow}} (Q_1', \Lambda_1') \;\wedge\; stable_{\Lambda_1', \leqslant}(Q_1') \;\wedge\; \forall\, x \in X \bullet \neg(Q_1', \Lambda_1') \underset{\leqslant}{\overset{x}{\hookrightarrow}},$$

for some Q_1' and Λ_1'. Then from Lemma 26

$$(Q_1 \;{}_A\|_B\; Q_2, \mathsf{interleave}(\Lambda_1, \Lambda_2)) \underset{\leqslant}{\overset{tr}{\Longrightarrow}} (Q_1' \;{}_A\|_B\; Q_2', \mathsf{interleave}(\Lambda_1', \Lambda_2')).$$

for some Q_2' and Λ_2'. Now, Q_2 is non-divergent, and Q_1' is stable, so

$$(Q_1' \;{}_A\|_B\; Q_2', \mathsf{interleave}(\Lambda_1', \Lambda_2')) \underset{\leqslant}{\overset{\tau}{\hookrightarrow}}{}^{*} (Q_1' \;{}_A\|_B\; Q_2'', \mathsf{interleave}(\Lambda_1', \Lambda_2''))$$

for some Q_2'' and Λ_2'', with Q_2'' stable. Further,

$$\forall\, x \in X \bullet \neg(Q_1' \;{}_A\|_B\; Q_2'', \mathsf{interleave}(\Lambda_1', \Lambda_2'')) \underset{\leqslant}{\overset{x}{\hookrightarrow}}$$

from the above, and Lemma 27. Hence

$$(tr, X) \in \mathit{failures}_{\mathsf{interleave}(\Lambda_1,\Lambda_2),\leqslant}(Q_1 \;{}_A\|_B\; Q_2),$$

as required. □

8.5.4 A closure property

We now prove that our definition satisfies a useful closure property, namely that it is preserved by composition with high level processes: if P_1 is secure, and $\alpha(P_2) \subseteq H$ then $P_1 \underset{H}{\|} P_2$ is secure; P_2 can be thought of as modelling a high level user; this result says that this high level user cannot turn a secure process into an insecure one.

We will need the following lemma.

Lemma 31 If $B \subseteq H$ and Q_2 is non-divergent then
$\mathit{failures}_{\mathsf{interleave}(\Lambda_1,\Lambda_2),\leqslant}((Q_1 \;{}_A\|_B\; Q_2)\backslash_H) = \mathit{failures}_{\Lambda_1,\leqslant}(Q_1\backslash_H)$.

Proof: We calculate as follows:

$$\begin{array}{ll}
& (tr, X) \in \mathit{failures}_{\mathsf{interleave}(\Lambda_1,\Lambda_2),\leqslant}((Q_1 \;{}_A\|_B\; Q_2)\backslash_H) \\
\Leftrightarrow & \langle \text{Lemma 19} \rangle \\
& (tr, X - H) \in \mathit{failures}_{\mathsf{interleave}(\Lambda_1,\Lambda_2),\leqslant}(Q_1 \;{}_A\|_B\; Q_2) \wedge tr \upharpoonright H = \langle\rangle \\
\Leftrightarrow & \left\langle \begin{array}{l} \text{Lemma 29 in the forwards direction;} \\ \text{Lemma 30 in the backwards direction} \end{array} \right\rangle \\
& (tr, X - H) \in \mathit{failures}_{\Lambda_1,\leqslant}(Q_1) \wedge tr \upharpoonright H = \langle\rangle \\
\Leftrightarrow & \langle \text{Lemma 19} \rangle \\
& (tr, X) \in \mathit{failures}_{\Lambda_1,\leqslant}(Q_1\backslash_H).
\end{array}$$

□

We now prove the result alluded to above.

Theorem 32 If P_1 satisfies NLNI, $B \subseteq H$ and P_2 is non-divergent, then $P_1 \;{}_A\|_B\; P_2$ satisfies NLNI.

Proof: Suppose $Q_1 \;{}_A\|_B\; Q_2 \in \mathit{States}(P_1 \;{}_A\|_B\; P_2)$ and

$$(Q_1 \;{}_A\|_B\; Q_2, \Lambda) \underset{\leqslant}{\overset{h}{\hookrightarrow}} (Q_1' \;{}_A\|_B\; Q_2', \Lambda'),$$

with $h \in H$ (it is easy to show that all states and transitions of $P_1 \;{}_A\|_B\; P_2$ must be of this form). We consider two cases:

- Case $h \in B - A$. Then necessarily $Q_1 = Q_1'$ and $\mathsf{odds}\,\Lambda = \mathsf{odds}\,\Lambda'$. Hence

$$\begin{array}{ll} & \mathit{failures}_{\Lambda,\leqslant}((Q_1 \;{}_A\|_B\; Q_2)\backslash_H) \\ = & \langle \text{Lemma 31} \rangle \\ & \mathit{failures}_{\mathsf{odds}\,\Lambda,\leqslant}(Q_1\backslash_H) \\ = & \langle \text{from the above observation} \rangle \\ & \mathit{failures}_{\mathsf{odds}\,\Lambda',\leqslant}(Q_1'\backslash_H) \\ = & \langle \text{Lemma 31} \rangle \\ & \mathit{failures}_{\Lambda',\leqslant}((Q_1' \;{}_A\|_B\; Q_2')\backslash_H), \end{array}$$

  as required.

- Case $h \in A$ (whether or not $h \in B$). Then

$$(Q_1, \mathsf{odds}\,\Lambda) \underset{\leqslant}{\overset{h}{\hookrightarrow}} (Q_1', \mathsf{odds}\,\Lambda')$$

  from Lemma 27. Hence

$$\begin{array}{ll} & \mathit{failures}_{\Lambda,\leqslant}((Q_1 \;{}_A\|_B\; Q_2)\backslash_H) \\ = & \langle \text{Lemma 31} \rangle \\ & \mathit{failures}_{\mathsf{odds}\,\Lambda,\leqslant}(Q_1\backslash_H) \\ = & \langle P_1 \text{ satisfies } \mathsf{NLNI} \rangle \\ & \mathit{failures}_{\mathsf{odds}\,\Lambda',\leqslant}(Q_1'\backslash_H) \\ = & \langle \text{Lemma 31} \rangle \\ & \mathit{failures}_{\Lambda',\leqslant}((Q_1' \;{}_A\|_B\; Q_2')\backslash_H), \end{array}$$

  as required.

□

The following corollary shows that our definition of information flow satisfies *separability* [Jac90a].

Corollary 33 If $\alpha P_1 \subseteq L$, $\alpha P_2 \subseteq H$, and P_2 is non-divergent, then $P_1 \;|||\; P_2$ satisfies NLNI.

Surprisingly, perhaps, our property of nondeterministic local noninterference isn't closed under various other forms of composition. For example, one might expect that whenever P_1 and P_2 both satisfy NLNI, then so does $P_1 \;|||\; P_2$. The following example shows that this is false:

Example 23 Let

$$P_1 \;\widehat{=}\; l1 \to STOP \;\square\; h \to l1 \to STOP,$$
$$P_2 \;\widehat{=}\; l2 \to STOP \rhd l3 \to STOP.$$

Then P_1 and P_2 both satisfy NLNI.

However, consider $P_1 \;|||\; P_2$ in the presence of a demon that prefers the first $l1$ event to the timeout, and prefers the timeout to the second $l1$ event. (In a timed setting, this demon would correspond to a situation where the first $l1$ event becomes available before the timeout, but the second $l1$ event doesn't.) In the initial state, the trace $\langle l1, l2 \rangle$ is possible; however, if the process performs an h-transition to $l1 \to STOP \mid\mid\mid P_2$, then the trace $\langle l1, l2 \rangle$ becomes impossible, because the timeout would occur before the $l1$, which would remove the possibility of the $l2$.

Following the definition of NLNI, $\langle l1, l2 \rangle$ is a trace of $(P_1 \mid\mid\mid P_2)\backslash_H$, but not of $(l1 \to STOP \mid\mid\mid P_2)\backslash_H$, and so $P_1 \mid\mid\mid P_2$ does not satisfy NLNI. $\square$

9 Discussion

In this paper we have presented a new definition of information flow, nondeterministic local noninterference. The definition is based upon a model of CSP that is more discriminating than the standard models, and talks about the ways in which nondeterministic choices are resolved. The model allows us to make more distinctions between processes, which we have argued is necessary: we have presented examples of processes that standard models fail to distinguish, but where one displays information flow, and the other doesn't.

The definition of information flow has much to recommend it: it produces the expected answer for all thought experiments we have tried so far; it satisfies pleasant closure properties (Theorem 32 and Corollary 33); it gives no flow of information if either *High* or *Low* can perform no events (Theorem 23).

On the down-side, the model is moderately complicated, but I suspect that this is inevitable, because we have to make quite fine distinctions between processes.

Surprisingly, the model suffers from the *refinement paradox*, where a secure process can be refined by an insecure one, as shown by the following example.

Example 24 Consider

$$\begin{aligned} P \ &\widehat{=}\ l0 \to l1 \to STOP \\ &\quad\ \triangleright \\ &\quad\ l0 \to (h \to l2 \to STOP \triangleright l2 \to STOP), \\ Q \ &\widehat{=}\ l0 \to (l1 \to STOP \,\square\, h \to l2 \to STOP) \\ &\quad\ \triangleright \\ &\quad\ l0 \to l2 \to STOP. \end{aligned}$$

Note that

- P is secure, because there is no way for *High* to affect *Low*'s view of the system;
- Q is insecure, specifically in the state $l1 \to STOP \,\square\, h \to l2 \to STOP$;
- P is refined by Q: following Definition 7, we need to show that for every demon $\leqslant_Q$ for Q, there is a demon $\leqslant_P$ for P that gives the same behaviour:

– if $\leqslant_Q$ forces the timeout in Q then define $\leqslant_P$ to force both timeouts in P;
– if $\leqslant_Q$ does not force the timeout in Q then define $\leqslant_P$ to force neither timeout in P (note that the timeouts can still occur with such demons).

$\square$

One thing that is slightly odd about the above example is that Q's insecure state, $l1 \rightarrow STOP \,\square\, h \rightarrow l2 \rightarrow STOP$, does not correspond to any state of P. This suggests that we should consider a stronger refinement relation—possibly based around a simulation relation—such that these two processes are not related. It might be the case that such a stronger refinement relation avoids the refinement paradox.

The standard models of CSP have complete sets of algebraic laws (see, for example, [Ros97, Chapter 11]). It is clear, though, that our model does not satisfy all of these laws; for example, our model distinguishes the processes Q_1 and Q_2 of Examples 6 and 12, which are equivalent in the standard models. It would be informative to discover precisely which rules do and do not hold in our model.

One of the most appealing properties of Roscoe's definition of security—in terms of the determinism of the lazy abstraction of a process—is that it is efficiently checkable using the model checker FDR [For97, Ros97]: the lazy abstraction of P is equivalent, in the stable failures model, to $(P \underset{H}{\parallel} CHAOS(H)) \setminus H$, and so FDR can be used to test whether this latter process is deterministic. I intend to investigate algorithms for checking nondeterministic local noninterference.

The model in this paper considered only untimed processes. It would be interesting to extend this model to consider time, so as to reason about information flow caused by timing information, through so-called *timing covert channels*. Timed models of information flow have been presented in [FGM00, Low02].

Several researchers have attempted to extend process algebras with probabilities; for example [JS90, Han91, JHSY94, MMSS95]. However, many of these approaches do not correctly model the interplay between probabilities and nondeterminism. For example, consider the process

$$(a \rightarrow STOP \oplus_{1/2} b \rightarrow STOP) \parallel (a \rightarrow STOP \sqcap b \rightarrow STOP),$$

where $\oplus_p$ is a probabilistic choice operator that selects its left hand side with probability p, and selects its right hand side with probability $1 - p$. One would expect the nondeterministic choice to be resolved in a way that is independent of the way in which the probabilistic choice is resolved. However, this is not the case with most existing models: such models effectively allow the demon to see how probabilistic choices elsewhere in the system have been resolved. I believe that extending the operational model of the current paper with probabilistic transitions, and modelling the demons as currently, would give the correct behaviour:

the demon's behaviour would depend upon actions elsewhere in the system only if there has been a suitable information flow.

Acknowledgements

I would like to thank Bill Roscoe, Peter Ryan, Steve Schneider, Mei Lin Hui and Richard Forster for interesting discussions about information flow. I would also like to thank the anonymous referee for useful comments.

This work was mostly carried out while I was employed at the University of Leicester. It was supported by grants from DERA Malvern and the US Office of Naval Research.

References

[All91] P. G. Allen. A comparison of non-interference and non-deducibility using CSP. In *Proceedings of the 4th IEEE Computer Security Foundations Workshop*, 1991.

[FG95] Riccardo Focardi and Roberto Gorrieri. A classification of security properties. *Journal of Computer Security*, 1995.

[FGM00] Riccardo Focardi, Roberto Gorrieri, and Fabio Martinelli. Information flow analysis in a discrete-time process algebra. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, pages 170–184, 2000.

[Foc96] Riccardo Focardi. Comparing two information flow security properties. In *Proceedings of 9th IEEE Computer Security Foundations Workshop*, pages 116–122, 1996.

[For97] Formal Systems (Europe) Ltd. *Failures-Divergence Refinement—FDR 2 User Manual*, 1997. Available via URL `http://www.formal.demon.co.uk/FDR2.html`.

[For99] Richard Forster. *Non-Interference Properties for Nondeterministic Processes.* D.Phil, Oxford University, 1999. Available from `http://www.comlab.ox.ac.uk/oucl/research/areas/concurrency/papers/thesis.ps.gz`.

[GC92] J. C. Graham-Cumming. *The Formal Development of Secure Systems.* D.Phil, Oxford University, 1992.

[GM82] J. A. Goguen and J. Meseguer. Security policies and security models. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 11–20, 1982.

[Han91] Hans A. Hansson. *Time and Probability in Formal Design of Distributed Systems*. PhD thesis, Swedish Institute of Computer Science, 1991. Published as SICS Dissertation Series, number 05.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[Jac90a] J. L. Jacob. Seperability and the detection of hidden channels. *Information Processing Letters*, 34:27–29, 1990.

[Jac90b] J. L. Jacob. Specifying security properties. In C. A. R. Hoare, editor, *Developments in Concurrency and Communication*. Addison-Wesley, 1990.

[JHSY94] Bengt Jonsson, Chris Ho-Stuart, and Wang Yi. Testing and refinement for nondeterministic and probabilistic processes. In H. Langmaack, W.-P. de Roever, and J. Vytopil, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems, LNCS 863*, pages 418–430, 1994.

[JS90] Chi-Chang Jou and Scott A. Smolka. Equivalences, congruences and complete axiomatizations for probabilistic processes. In *Concur '90, LNCS 458*, 1990.

[Low00] Gavin Lowe. Semantic models for information flow. Technical Report 2000/25, Department of Mathematics and Computer Science, University of Leicester, 2000.

[Low02] Gavin Lowe. Quantifying information flow. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, pages 18–31, 2002.

[MMSS95] Carroll Morgan, Annabelle McIver, Karen Seidel, and Jeff Sanders. Refinement-oriented probability for CSP. *Formal Aspects of Computing*, 8(6):617–647, 1995.

[Ros95] A. W. Roscoe. CSP and determinism in security modelling. In *Proceedings of 1995 IEEE Symposium on Security and Privacy*, 1995.

[Ros97] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1997.

[Rya91] P. Y. A. Ryan. A CSP formulation of non-interference. *Cipher*, pages 19–27, 1991. Also in Proceedings of the 3rd IEEE Computer Security Foundations Workshop, 1990.

[Sut86] D. Sutherland. A model of information. In *Proceedings of the 9th National Computer Security Conference*, pages 175–183, 1986.

[WJ90] J. Todd Wittbold and Dale M. Johnson. Information flow in nondeterministic systems. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 144–161, 1990.

A Index of notation

Notation	Description	Section
$\xrightarrow{a,\mu}$	$(P,\Lambda) \xrightarrow{a,\mu} (Q,\Lambda')$ means P with marker sequence Λ can perform event a, labelled with marker μ, and evolve into Q, with remaining markers Λ'.	5
$\underset{\leqslant}{\overset{a}{\hookrightarrow}}$	$(P,\Lambda) \underset{\leqslant}{\overset{a}{\hookrightarrow}} (Q,\Lambda')$ means P with marker sequence Λ, and in the presence of demon $\leqslant$, can perform event a, and evolve into Q, with remaining markers Λ'.	6.1
$\underset{\leqslant}{\overset{s}{\longmapsto}}$	$(P,\Lambda) \underset{\leqslant}{\overset{s}{\longmapsto}} (Q,\Lambda')$ means P with marker sequence Λ, and in the presence of demon $\leqslant$, can perform the trace s, possibly including internal events, and evolve into Q, with remaining markers Λ'.	6.2
$\underset{\leqslant}{\overset{tr}{\Longrightarrow}}$	$(P,\Lambda) \underset{\leqslant}{\overset{tr}{\Longrightarrow}} (Q,\Lambda')$ means P with marker sequence Λ, and in the presence of demon $\leqslant$, can perform the visible trace tr, and evolve into Q, with remaining markers Λ'.	6.2
$stable_{\Lambda,\leqslant}$	$stable_{\Lambda,\leqslant}\, P$ means that P with marker sequence Λ, and in the presence of demon $\leqslant$, cannot perform any internal transitions.	6.3
$ref_{\Lambda,\leqslant}$	$P\ ref_{\Lambda,\leqslant}\ X$ means that P with marker sequence Λ, and in the presence of demon $\leqslant$, will refuse the set of events X.	6.3
$div_{\Lambda,\leqslant}$	$div_{\Lambda,\leqslant}\, P$ means that P with marker sequence Λ, and in the presence of demon $\leqslant$, can diverge, i.e. perform an infinite sequence of internal events.	6.3