Defining Information Flow Quantity

Gavin Lowe*

August 17, 2004

Abstract

We extend definitions of information flow so as to *quantify* the amount of information passed; in other words, we give a formal definition of the capacity of covert channels. Our definition uses the process algebra CSP, and is based upon counting the number of different behaviours of a high level user that can be distinguished by a low level user.

1 Introduction

Previous work has sought to capture the notion of *information flow* (sometimes called non-interference or independence) in a multi-level security system: that is the question of whether a high level user, High, can pass information to a low level user, Low, via a covert channel. A common approach has been to produce a definition of information flow in a process algebraic setting; see, for example [All91, Rya91, FG95, Ros95, For99, FGM00]. Most definitions are based around asking whether Low can distinguish between two different behaviours of High, so that High can use the system to pass at least one bit of information to Low.

However, in many circumstances, some flow of information will be inevitable and acceptable, providing it is not too high. For example, the Orange Book [US 85] includes requirements for the estimation of the capacity of covert channels, and recommendations for acceptable values.

In this paper we extend the previous work on defining information flow, and produce a definition, using the process algebra CSP, of the *quantity* of information flow. We will define the *information flow cardinality* to be the number of behaviours of High that are distinguishable from Low's point of view. If there are N such distinguishable behaviours, then High can use the system to encode an arbitrary number in the range $0, \ldots, N-1$ to send it to Low; in other words, $\log_2 N$ bits of information are passed. (A feature of this definition, that the reader

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

should be aware of, is that an absence of information flow is represented by an information flow cardinality of 1, rather than 0.)

An important consideration in models of information flow is the treatment of nondeterminism. Nondeterminism occurs in models of systems for a number of reasons, for example: the "noise" caused by other users of the system; inherent nondeterminism in the system; and aspects of the system that are resolved at a lower level of abstraction than that at which the model is built.

Previous formal approaches to the quantity of information flow have modelled this nondeterminism probabilistically. The capacity of covert channels is then defined making use of Shannon's theory of information flow [SW63]. See, for example, [Mil87, WJ90, Gra91]. Clearly, probabilistic models are useful only when one can accurately assign probabilities to the nondeterminism of the system. This is rarely the case: one simply knows that the system will behave in one of several ways, without knowing anything about the relative probabilities. We believe it is better to consider all possible ways in which the nondeterminism can be resolved, and to consider the worst case, i.e. the maximum information flow: thus we obtain an upper-bound on the channel capacity; if, in fact, the nondeterminism is resolved differently, then we will have over-estimated the capacity, which is the safe way in which to err.

An additional form of nondeterminism is underspecification in designs. Formal analysis is normally applied to designs rather than to implementations: by the time one has produced an implementation, there is too much detail to make formal analysis practical. Further, it is well known that the earlier an error is discovered, the easier and cheaper it is to fix; therefore it makes most sense to perform an analysis as early in the development as possible. Nondeterminism in designs represents underspecification: the intention is that later stages of the development resolve the nondeterminism. However, a design can only be considered correct if *all* ways of resolving the nondeterminism lead to a correct implementation; thus, when considering information flow quantity, we again want to consider the maximum flow for all ways in which the nondeterminism can be resolved.

Information can be passed from High to Low based upon the time at which events become available; it is therefore necessary to include a notion of time in order to obtain realistic models. It is reasonable to assume that there is a limit on how accurately agents can tell the time: that is, they effectively have a time quantum. In order to model this, it turns out to be convenient to work in the discrete-time model from [Oua00], taking the time quantum to be one time unit. In principle, one could work in a continuous-time model, such as Timed CSP [Sch99] (this was the approach taken in an earlier version of this paper); however, this tends to lead to less tractable models than discrete-time models. One advantage of working in a timed model is that we can define the quantity of information flow within some finite time period, and hence the *rate* of information flow.

In the following section, we give a brief introduction to the syntax and se-

mantics of discrete-time CSP. In Section 3, we consider some example processes, and for each give the information flow cardinality, thus helping to describe our intuitions. In Section 4, we formalise the notion and give our main definition; we illustrate it with some examples. The definition is slightly non-intuitive, and proved surprisingly difficult to get right: a number of simpler definitions fail to give the right answer for example processes. The main difficulty is to produce suitable models of High and Low: we argue that our intuitions are best captured by a model for Low based upon testing; it turns out that it is necessary to give High more control over events than one might expect (we need to allow High to be able to ensure that he can perform his events before Low, if he so desires; to do this it is necessary to give High some control over when Low's events are performed). In Section 5 we show that those processes for which our definition gives an information flow cardinality of 1—i.e. no information flow—are precisely those processes that satisfy a more intuitive definition of lack of information flow, similar to that in [FG95, Foc96]. In Section 6 we consider the amount of information passed within a bounded amount of time, thus leading to a definition of the capacity of covert channels in bits per time unit. Finally, we sum up and give pointers to future work in Section 7.

2 A brief overview of discrete-time CSP

In this section we give a brief overview of discrete-time CSP. More details can be obtained from [Ros97, Sch99, Oua00].

2.1 Untimed syntax

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, $c.5$ is an event of channel c. $\{|c|\}$ represents the set of events of channel c. The distinguished event *tock* represents the passage of one unit of time. All processes in a system will participate in this event, and none can prevent it from happening, i.e. no process can stop time. There is no limit on the number of non-*tock* events that processes can perform between successive *tock* events. Σ represents the set of all standard (i.e. non-*tock*) events, and $\Sigma_{tock} \mathrel{\widehat{=}} \Sigma \cup \{tock\}$.

The process *STOP* can perform no standard events, but simply allows time to progress. The process $a \rightarrow P$ can perform the event a, and then act like P; the process allows time to progress, and such progress does not remove the availability of a. The process $?a : A \rightarrow P_a$ offers the set of events A; if a particular event $a \in A$ is performed, the process then acts like P_a; again, the process allows time to progress, and such progress does not remove the availability of the events of A. (The prefixing operator "$\rightarrow$" has higher precedence than all other

operators.) The process $WAIT\ t\ ;\ P$ does nothing for the first t time units, and then acts like P.

The process $P\ \Box\ Q$ represents an external choice between P and Q; the initial events of both processes are offered to the environment; when an event is performed, that resolves the choice. The process allows time to progress, but *tock* events do not resolve the choice. $P \sqcap Q$ represents an internal or nondeterministic choice between P and Q; the process can act like either P or Q, with the choice being made according to some criterion that we do not model.

The process $P \overset{t}{\triangleright} Q$ initially acts like P; however, if no standard events of P occur within the first t time units, then a timeout occurs and the process acts like Q. By contrast, the process $P\ \triangle_t\ Q$ initially acts like P, but control is transferred to Q after t time units, regardless of the progress made by P.

$P \setminus A$ acts like P, except all events from the set A are hidden, i.e. made internal. Such internal events do not require the cooperation of the environment, and so they will occur as soon as they become available, i.e., before any *tock* events.

The process $RUN(A)$ can perform any events from A, and never refuse any such events; it can be defined by

$$RUN(A)\ =\ ?a : A \rightarrow RUN(A).$$

The process $CHAOS(A)$ can perform any sequence of events from A, or refuse any events; hence it is the most nondeterministic, nondivergent process with alphabet A; it can be defined by

$$CHAOS(A)\ =\ (\bigsqcap_{a:A} a \rightarrow CHAOS(A) \sqcap STOP) \overset{1}{\triangleright} CHAOS(A),$$

(where $\bigsqcap_{a:A}$ is an indexed version of the internal choice operator).

$P \underset{A}{\parallel} Q$ represents the parallel composition of P and Q, synchronising on events from A; the processes also synchronise on the event *tock*. $P \mid\mid\mid Q$ represents an interleaving of the processes P and Q; i.e. parallel composition synchronising only on the event *tock*.

The syntax of discrete-time CSP is summarised in Figure 1.

2.2 Semantics

A *refusal* is either:

- a set X of events, representing that the process is in a stable state—i.e. no internal activity is possible—and that none of the events of X is available; or
- the *null refusal* $\bullet$, representing the absence of refusal information, possibly because the process did not enter a stable state.

$STOP$	deadlocked process
$a \rightarrow P$	prefixing
$?a : A \rightarrow P$	generalised prefixing
$WAIT\ t\ ;\ P$	delay
$P \Box Q$	external choice
$P \sqcap Q$	internal choice
$P \overset{t}{\rhd} Q$	timeout
$P \triangle_t Q$	timed-transfer
$P \setminus A$	hiding
$RUN(A)$	process where all events from A are available
$CHAOS(A)$	arbitrary behaviour on A
$P \parallel Q$	parallel composition, synchronising on A
$P \overset{A}{\vert\vert\vert} Q$	parallel composition without synchronisation

Figure 1: Syntax summary

A *refusal trace*[1] is an alternating sequence of refusals and events, starting and finishing with a refusal. For example $\langle \{b\}, a, \bullet, tock, \{a, b\} \rangle$ represents that the process can: reach a stable state from which a b is not available; perform an a; perform a *tock* (the null refusal $\bullet$ gives us no information); and then reach a stable state from which neither a nor b is available.

Intuitively, a refusal trace represents the result of an experiment, in which the experimenter attempts to perform events, and records which are performed and which are refused. Note in particular that the null refusal $\bullet$ represents the absence of any refusal information, whereas the empty refusal $\{\}$ represents that the process reaches a stable state. The refusal traces of a process are downwards closed (axiom R2, below), so if a process has a refusal trace with a non-null refusal X, it also has a refusal trace where X is replaced by $\bullet$, representing that the experimenter did not detect any refusal information.

We can define operators $\cup^*$, $\cap^*$, $\subseteq^*$ over refusals as extensions of the corresponding standard set-theoretic operations, treating $\bullet$ like the empty set, but maintaining the normal definitions over sets. Rather than giving full definitions, we list a few relevant properties, where X is a non-$\bullet$ refusal: $\bullet \subseteq^* \bullet$; $\bullet \subseteq^* X$; $\bullet \cup^* \bullet = \bullet$; $\bullet \cup^* X = X$; $\bullet \cap^* \bullet = \bullet \cap^* X = \bullet$.

If s is a refusal trace, we write $\mathrm{trace}(s)$ for s stripped of all refusal information:

$$\mathrm{trace}(\langle A_0, a_1, A_1, \ldots, a_m, A_m \rangle) \ \widehat{=}\ \langle a_1, \ldots, a_m \rangle.$$

[1]Refusal traces are normally known as refusal tests; however, we use the term "test" for a somewhat different view of processes, and so we adopt the term "refusal trace" to avoid confusion.

We define an *information ordering* $\preceq$ over refusal traces as follows: if $s = \langle A_0, a_1, A_1, \ldots, a_m, A_m \rangle$ and $t = \langle B_0, b_1, B_1, \ldots, b_n, B_n \rangle$ then

$$\begin{aligned} s \preceq t \;\Leftrightarrow\; & 0 \leqslant m \leqslant n \;\wedge \\ & (\forall\, i \in \{1, \ldots, m\} \bullet a_i = b_i) \;\wedge \\ & (\forall\, i \in \{0, \ldots, m\} \bullet A_i \subseteq^* B_i). \end{aligned}$$

t might contain more information, either by being an extension of s, of by containing larger refusal sets.

The *discrete-time refusal traces model* of CSP represents a process P by the set S of its refusal traces, denoted $\mathcal{R}[\![P]\!]$. The set S will satisfy the following axioms or healthiness conditions:

R1. The behaviours of a process are non-empty; it is always possible to observe a null refusal:

$$\langle \bullet \rangle \in S.$$

R2. The behaviours of a process are downwards-closed with respect to the information ordering:

$$s \in S \wedge t \preceq s \;\Rightarrow\; t \in S.$$

R3. Processes cannot block time, so a refusal trace can always be extended with a *tock*:

$$s \in S \;\Rightarrow\; s^\frown\langle \mathit{tock}, \bullet \rangle \in S.$$

R4. All internal activity will finish before a *tock* event; hence the process will stabilise, and so an empty refusal can be observed (recall that refusals, even empty ones, can be observed only in stable states):

$$s^\frown\langle \bullet, \mathit{tock} \rangle^\frown t \in S \;\Rightarrow\; s^\frown\langle \{\}, \mathit{tock} \rangle^\frown t \in S.$$

R5. If a process refuses a set of events X, it cannot then perform any event a from X:

$$s^\frown\langle X, a \rangle^\frown t \in S \Rightarrow a \notin X.$$

R6. If a process can refuse a non-$\bullet$ set of events X[2], then it can also refuse every event a that cannot be performed in that state:

$$X \neq \bullet \wedge s^\frown\langle X \rangle \in S \wedge s^\frown\langle X, a, \bullet \rangle \notin S \;\Rightarrow\; s^\frown\langle X \cup \{a\} \rangle \in S.$$

[2]The restriction to non-$\bullet$ refusals is to ensure that the process can stabilise in this state.

A couple of examples will help to illustrate the semantics:

- $\mathcal{R}[\![STOP]\!]$ contains all refusal traces of the form

$$\langle A_0, tock, \ldots, tock, A_n \rangle,$$

  where $A_i \subseteq^* \Sigma$ for each i.

- $\mathcal{R}[\![a \rightarrow STOP]\!]$ contains all refusal traces of the form

$$\langle A_0, tock, \ldots, tock, A_m \rangle$$

  or

$$\langle A_0, tock, \ldots, tock, A_m, a, B_0, tock, \ldots, tock, B_n \rangle,$$

  where $A_i \subseteq^* \Sigma - \{a\}$ and $B_i \subseteq^* \Sigma$ for each i.

Compositional denotational semantic equations, giving the refusal traces of a compound process in terms of those of its components, are given in [Oua00]; we give a selection (those that we make use of later in this paper) in Appendix B.

3 Thought experiments

To help to motivate the problem further, we consider some example processes; in each case we quantify the information passed, thus explaining our intuitions.

Throughout this paper we will let H be the set of events for High, which will have names like h, and we will let L be the set of events for Low, which will have names like l. We assume that Σ is partitioned by H and L.

High and Low form the environment of the process. Hence an event will occur only when both the process and High or Low are willing to perform it. Our models of High and Low will ensure that they can observe events from H and L, respectively, and to a limited extent (explained below) observe such events being refused by the process; they will also both observe *tock* events, and so will be synchronised on time.

Example 1 Consider the process[3]

$$P_1 \mathrel{\widehat{=}} h \rightarrow l \rightarrow STOP \stackrel{1}{\triangleright} STOP.$$

We assume that Low is not able to detect the precise time at which events occur, for that would seem physically infeasible. Not making this assumption would have unfortunate consequences for the model, because many processes like the above would give an unbounded amount of information flow: High could pass an

[3]The effect of the "$\stackrel{1}{\triangleright}\ STOP$" is to remove the availability of the h event after the first time unit.

arbitrary real number t in the range $[0, 1)$ by performing his event at time t; Low could observe the time at which the l event became available, and so deduce the value High was trying to send.

Instead, we make the physically more reasonable assumption that there is a limit to how accurately Low can record the time. In particular, we assume a finite speed for Low: we assume a time quantum, such that Low cannot vary his behaviour more rapidly than once per time quantum; more precisely, if he offers an event, he will not be able to withdraw that offer within a time quantum. For notational ease, we take the time quantum equal to one time unit (if necessary, by rescaling the time unit). Thus Low can only tell the time by observing *tock* events performed, and cannot tell the difference between two times within the same time unit. Further, he can only tell that an event is not available by seeing that event refused up to a *tock*. We formalise these assumptions in Section 4.

Hence the above process has an information flow cardinality of 2: if High chooses to perform an h within the first time unit then Low will be able to perform an l; alternatively, if High chooses not to perform an h within the first time unit, then Low will see the event refused up to the first *tock*. Thus High can use P_1 to pass at least one bit of information to Low. Clearly, High cannot interact with P_1 in any ways other than those identified above, so this value is also an upper bound.

Example 2 Consider

$$P_2 \mathrel{\widehat{=}} h \rightarrow l \rightarrow STOP \overset{N}{\triangleright} STOP\,.$$

This process has an information flow capacity of $N+1$. High can pass an arbitrary value k in the range $0, \ldots, N-1$ by performing his event after k time units; Low will then be able to perform his event after observing k *tock*s. High can pass an additional value by not performing his event; Low will observe that he is unable to perform his event throughout the first N time units.

Note the difference between this example and the previous: in the previous example, Low could only tell whether or not High had performed an event; in this example, there is additional information from timing considerations. In the rest of this section we consider only examples illustrating information flow of the former kind, so as to concentrate on the non-timing aspects; we allow events to occur only within the first time unit, by including an explicit timeout at time 1.

Example 3 Consider:

$$P_3 \mathrel{\widehat{=}} \begin{pmatrix} h1 \rightarrow (l1 \rightarrow STOP \sqcap l2 \rightarrow STOP) \\ \Box\ h2 \rightarrow (l1 \rightarrow STOP \sqcap l2 \rightarrow STOP) \end{pmatrix} \overset{1}{\triangleright} STOP\,.$$

This process certainly has an information flow of at least 2, for Low can tell whether or not High has performed some event. The question is whether Low can distinguish the two behaviours of the system following $h1$ and $h2$ events. If

the two nondeterministic choices were implemented identically, then clearly the answer would be no (and the information flow cardinality would be 2). However, 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 were implemented to always select its first argument, and the second nondeterministic choice were implemented to always select its second argument, then Low would indeed be able to distinguish the two behaviours, and so the information flow cardinality would be 3.

We indeed take the information flow cardinality of this process to be 3, being the higher of the above possible values; in other words, we consider the worst case scenario.

Example 4 Let

$$P_4 \mathrel{\widehat{=}} \begin{pmatrix} h1 \rightarrow l1 \rightarrow STOP \\ \Box\ h2 \rightarrow l2 \rightarrow STOP \\ \Box\ h3 \rightarrow (l1 \rightarrow STOP \sqcap l2 \rightarrow STOP) \end{pmatrix} \stackrel{1}{\triangleright} STOP\,.$$

This has an information flow cardinality of 3, as does the process

$$(h1 \rightarrow l1 \rightarrow STOP \Box\ h2 \rightarrow l2 \rightarrow STOP) \stackrel{1}{\triangleright} STOP\,.$$

The $h3$ branch does not add any information flow: if Low can perform an $l1$ then he can deduce that High performed either $h1$ or $h3$, but he cannot tell which; similarly, if Low can perform an $l2$ then he can deduce that High performed either $h2$ or $h3$, but he cannot tell which.

We are assuming that Low cannot make copies of the system so as to perform repeated experiments: if he could, he might be able to make a copy of the system following an $h3$, and then in repeated experiments observe both $l1$ and $l2$ events (corresponding to different resolutions of the nondeterministic choice) and hence deduce that $h3$ had indeed been performed.

Example 5 Let

$$P_5 \mathrel{\widehat{=}} (l \rightarrow STOP \Box\ h \rightarrow STOP) \stackrel{1}{\triangleright} STOP\,.$$

If High performs his event (before the first *tock*) then Low will see his l event refused; however, if High does not try to perform his event, then Low will be able to perform his event. However, if Low is able to perform an l, then he cannot tell whether or not High was attempting to perform h: High's h might have been preempted by Low's l. High and Low might be able to use this process to pass one bit of information, but we cannot be sure that they can do so reliably. (In [McC88], McCullough terms this a positive half-bit channel.)

The normal CSP intuition is that if both High and Low attempt their events at the same time, then it will be nondeterministic which succeeds. However, it might be that this nondeterminism is resolved such that High always succeeds: it might be that the process scheduler gives priority to High; or maybe High can always get his event in before Low's (but both attempt their event within the first time unit). We again consider the worst case scenario, and take the information flow cardinality to be 2.

4 Defining information flow cardinality

In this section we formalise information flow cardinality, by considering the way in which High and Low interact with the system in order for High to pass information to Low. We assume that the behaviour of the system is modelled by a CSP process P. High will act in different ways depending upon the value he is trying to send; Low will interact with the system to try to deduce the value being sent.

In Section 4.1 we consider the way in which Low will interact with the system; this induces an equivalence over systems—corresponding to indistinguishability by Low—for which we derive an alternative characterisation. In Section 4.2 we consider High's behaviour; it turns out that we have to give High some limited control over events not in H. We bring the threads together in Section 4.3 to define the information flow cardinality of a system, and give some examples in Section 4.4.

4.1 Low's strategy and testing equivalence

Low will interact with the system by repeatedly offering a set of events and either seeing one of them accepted, or seeing them all refused; Low will associate results with particular sequences of event acceptances and refusals. We will call Low's strategy for interacting with a process a *test*. The precise form of the tests will reflect the assumptions we make about Low, discussed informally above.

Traditional approaches to testing (e.g. [dNH84, Phi86, Sch99]) make use of a test process $SUCCESS$, which can perform a distinguished event ω representing the successful completion of a test. We extend this to use test processes of the form $SUCCESS(k)$, for $k \in \mathbb{N}$, that can perform events of the form $\omega.k$ representing successful completion of the test with result k:

$$SUCCESS(k) \mathrel{\widehat{=}} \omega.k \rightarrow STOP\,.$$

(We assume $\omega.k \notin \Sigma$.)

We represent Low's behaviour by a process T with alphabet $L \cup \{tock\} \cup \{|\omega|\}$; we define $Test$ to be the set of all such processes. The process will interact with the system S, and then give results via ω events; we will therefore consider

compositions of the form $(S \underset{L}{\parallel} T) \setminus L$, and consider the results that such a composition can produce:

$$results(S, T) \mathrel{\widehat{=}} \{k \mid \exists\, n \in \mathbb{N} \bullet \langle \bullet, tock\rangle^n \frown \langle \bullet, \omega.k, \bullet\rangle \in \mathcal{R}[\![(S \underset{L}{\parallel} T) \setminus L]\!]\}.$$

Note that the results is a *set*, because of the possibility of nondeterminism.

On testing equivalence

This definition of *results* invokes a natural equivalence over systems, which we call *testing equivalence*, namely that two systems are equivalent if all tests give the same set of possible results; it also invokes a natural refinement pre-order:

$$S \equiv_T S' \mathrel{\widehat{=}} \forall\, T \bullet results(S, T) = results(S', T),$$
$$S \sqsubseteq_T S' \mathrel{\widehat{=}} \forall\, T \bullet results(S, T) \supseteq results(S', T).$$

(In general we are interested in whether Low can distinguish more than two systems.) In this section we consider the testing equivalence further.

Note, firstly, that the equivalence is the same as may-equivalence [dNH84, Sch99], where tests simply succeed or fail—rather than giving a numeric result—and two processes are may-equivalent if whenever a test can succeed with one process, it can also succeed with the other. The fact that our tests have arbitrarily many possible results does not allow us to distinguish more pairs of processes than with tests that simply succeed or fail.

It turns out that testing equivalence is slightly weaker than semantic equality: testing can distinguish processes on the basis of events performed, and on the basis of refusals that immediately precede a *tock*, but cannot distinguish processes on the basis of refusals that do not immediately precede a *tock*.

Let $\dot{s}$ be obtained from s by replacing all refusals that do not immediately precede a *tock* with $\bullet$:

$$\dot{\overbrace{\langle X, a\rangle \frown s}} = \langle \bullet, a\rangle \frown \dot{s}, \qquad a \neq tock$$
$$\dot{\overbrace{\langle X, tock\rangle \frown s}} = \langle X, tock\rangle \frown \dot{s},$$
$$\dot{\overbrace{\langle X\rangle}} = \langle \bullet\rangle.$$

Note that

$$s \in \mathcal{R}[\![P]\!] \Rightarrow \dot{s} \in \mathcal{R}[\![P]\!],$$

by Axiom R2.

It is convenient to define

$$\mathcal{T}[\![P]\!] \;\widehat{=}\; \{\dot{s} \mid s \in \mathcal{R}[\![P]\!]\}.$$

The following lemma shows that testing equivalence can distinguish processes only on the basis of refusal tests $\dot{s}$:

Lemma 1 $P \equiv_T Q \Leftrightarrow \mathcal{T}[\![P]\!] = \mathcal{T}[\![Q]\!]$.

Proof: We begin by noting the following facts (the operators $\underset{L}{\|}$ and $\setminus$, and what it means for a trace to be L-urgent, are defined in Appendix B).

1. If $\dot{s_P} = \dot{s_Q}$ then $\overbrace{(s_P \underset{L}{\|} t)}^{\cdot} = \overbrace{(s_Q \underset{L}{\|} t)}^{\cdot}$;
2. If $\dot{s_P} = \dot{s_Q}$ and s_P is L-urgent then s_Q is L-urgent;
3. If $\dot{s_P} = \dot{s_Q}$ then $\overbrace{s_P \setminus L}^{\cdot} = \overbrace{s_Q \setminus L}^{\cdot}$.

For the right-to-left implication, suppose

$$\mathcal{T}[\![P]\!] = \{\dot{s} \mid s \in \mathcal{R}[\![P]\!]\} = \{\dot{s} \mid s \in \mathcal{R}[\![Q]\!]\} = \mathcal{T}[\![Q]\!].$$

Pick $T \in \mathit{Test}$, and let s_P be a trace. Then

$$\begin{array}{l}
\quad s_P \in \mathcal{R}[\![(P \underset{L}{\|} T) \setminus L]\!] \\
\Rightarrow \langle \text{semantic definitions} \rangle \\
\quad \exists\, s'_P \in \mathcal{R}[\![P]\!], t \in \mathcal{R}[\![T]\!] \bullet s'_P \underset{L}{\|} t \text{ is } L\text{-urgent} \wedge s_P = (s'_P \underset{L}{\|} t) \setminus L \\
\Rightarrow \langle \text{assumption} \rangle \\
\quad \exists\, s'_P \in \mathcal{R}[\![P]\!], s'_Q \in \mathcal{R}[\![Q]\!], t \in \mathcal{R}[\![T]\!] \bullet \\
\qquad s'_P \underset{L}{\|} t \text{ is } L\text{-urgent} \wedge s_P = (s'_P \underset{L}{\|} t) \setminus L \wedge \dot{s'_P} = \dot{s'_Q} \\
\Rightarrow \langle \text{above observations} \rangle \\
\quad \exists\, s'_P \in \mathcal{R}[\![P]\!], s'_Q \in \mathcal{R}[\![Q]\!], t \in \mathcal{R}[\![T]\!] \bullet \\
\qquad s'_P \underset{L}{\|} t \text{ is } L\text{-urgent} \wedge s_P = (s'_P \underset{L}{\|} t) \setminus L \wedge \dot{s'_P} = \dot{s'_Q} \wedge \overbrace{s'_P \underset{L}{\|} t}^{\cdot} = \overbrace{s'_Q \underset{L}{\|} t}^{\cdot} \wedge \\
\qquad s'_Q \underset{L}{\|} t \text{ is } L\text{-urgent} \wedge \overbrace{(s'_P \underset{L}{\|} t) \setminus L}^{\cdot} = \overbrace{(s'_Q \underset{L}{\|} t) \setminus L}^{\cdot} \\
\Rightarrow \left\langle \text{taking } s_Q = (s'_Q \underset{L}{\|} t) \setminus L \right\rangle \\
\quad \exists\, s_Q \in \mathcal{R}[\![(Q \underset{L}{\|} T) \setminus L]\!] \bullet \dot{s_P} = \dot{s_Q},
\end{array}$$

and vice versa. Hence

$$\{\dot{s_P} \mid s_P \in \mathcal{R}[\![(P \underset{L}{\|} T) \setminus L]\!]\} \;=\; \{\dot{s_Q} \mid s_Q \in \mathcal{R}[\![(Q \underset{L}{\|} T) \setminus L]\!]\},$$

and so

$$\{\text{trace}(s) \mid s \in \mathcal{R}[\![(P \underset{L}{\parallel} T) \setminus L]\!]\} \;=\; \{\text{trace}(s) \mid s \in \mathcal{R}[\![(Q \underset{L}{\parallel} T) \setminus L]\!]\}.$$

Hence

$$\mathit{results}(P, T) \;=\; \mathit{results}(Q, T).$$

The above holds for all T, so $P \equiv_T Q$.

For the converse, we prove the contrapositive. Suppose

$$\mathcal{T}[\![P]\!] \;\neq\; \mathcal{T}[\![Q]\!].$$

Then without loss of generality, there is some s such that $\dot{s} \in \mathcal{R}[\![P]\!]$ but $\dot{s} \notin \mathcal{R}[\![Q]\!]$. Let

$$\begin{aligned}\dot{s} \;=\; \langle &\bullet, a_{11}, \bullet, a_{12}, \ldots, \bullet, a_{1m_1}, A_1, \mathit{tock},\\ &\bullet, a_{21}, \bullet, a_{22}, \ldots, \bullet, a_{2m_2}, A_2, \mathit{tock},\\ &\ldots\\ &\bullet, a_{n1}, \bullet, a_{n2}, \ldots, \bullet, a_{nm_n}, \bullet\rangle.\end{aligned}$$

For convenience, we define a syntactic operator that attempts an event for one time unit, giving result 0 if it is not accepted:

$$a \xrightarrow{*} T \;\mathrel{\widehat{=}}\; a \rightarrow T \overset{1}{\triangleright} \mathit{SUCCESS}(0).$$

And define

$$B_i \;\mathrel{\widehat{=}}\; \text{if } A_i = \bullet \text{ then } \{\} \text{ else } A_i \qquad i = 1, \ldots, n-1.$$

We construct a test that succeeds with value 1 when the process it is testing performs the trace $\dot{s}$, but succeeds with value 0, otherwise:

$$\begin{aligned}T \;\mathrel{\widehat{=}}\; & a_{11} \xrightarrow{*} a_{12} \xrightarrow{*} \ldots \xrightarrow{*} a_{1m_1} \xrightarrow{*} (?x : B_1 \rightarrow \mathit{SUCCESS}(0)\\ & \overset{1}{\triangleright}\\ & a_{21} \xrightarrow{*} a_{22} \xrightarrow{*} \ldots \xrightarrow{*} a_{2m_2} \xrightarrow{*} (?x : B_2 \rightarrow \mathit{SUCCESS}(0)\\ & \overset{1}{\triangleright}\\ & \ldots\\ & \overset{1}{\triangleright}\\ & a_{n1} \xrightarrow{*} a_{n2} \xrightarrow{*} \ldots \xrightarrow{*} a_{nm_n} \xrightarrow{*} \mathit{SUCCESS}(1))\ldots).\end{aligned}$$

Then $\mathit{results}(Q, T) = \{0\}$ but $\mathit{results}(P, T) \supseteq \{1\}$. □

The lemma shows that testing equivalence fits with our intuition that Low cannot tell the time with total accuracy, and so can detect that an event is not available only by seeing it refused when a *tock* is performed.

To illustrate the difference between semantic equality and testing equivalence, consider the processes

$$P \mathrel{\widehat{=}} (a \rightarrow STOP \sqcap STOP) \stackrel{1}{\triangleright} STOP,$$
$$Q \mathrel{\widehat{=}} (a \rightarrow STOP \,\Box\, c \rightarrow STOP) \setminus \{c\}.$$

Note that in Q, the hidden c will occur before the first *tock* unless it is preempted by the a, because hidden events do not need the cooperation of the environment and so are urgent; hence the a will occur before the first *tock*, if at all. These processes are not semantically equivalent, because

$$\langle \{b\}, a, \bullet \rangle \in \mathcal{R}[\![P]\!] - \mathcal{R}[\![Q]\!].$$

P can refuse a b and then perform an a; but Q cannot, because Q performs a in an unstable state (recall that refusals are recorded only in stable states). However, the processes are testing equivalent: a test can only detect a b being refused if it is refused throughout a time unit, up to a *tock* being performed; hence no test can detect whether b is refused before the a is performed.

In untimed models, may-equivalence is the same as traces equivalence [dNH84]; in real-time models, it is the same as finite failures equivalence [Sch99]; a consequence of Lemma 1 is that in the discrete-time model the equivalence lies between these two.

4.2 High's strategy

It would seem natural to model High's behaviour, when he is trying to pass a particular value, by a CSP process Q with alphabet $H \cup \{tock\}$; we write CSP_H for the set of such processes. In that case Low's view of the system would be given by $(P \underset{H}{\parallel} Q) \setminus H$. However, it turns out that this won't allow us to model the assumptions about High from Section 3, especially that in Example 5. In particular, we need to assume that High is able to arrange for his events to occur either before or after events of Low as he desires.

Instead, we take the process Q to represent not only High's behaviour, but also the behaviour of the scheduler that resolves which event should occur when more than one is available. We therefore include Low's events within the alphabet of the process Q. Low's view of the system will then be given by the process

$$(P \underset{\Sigma}{\parallel} Q) \setminus H.$$

For example, recall the process:

$$P_5 \mathrel{\widehat{=}} (l \rightarrow STOP \,\Box\, h \rightarrow STOP) \stackrel{1}{\triangleright} STOP$$

from Example 5. High could pass one value by performing his event h; this is modelled by the process $Q \mathrel{\widehat{=}} h \rightarrow STOP$; the composition $(P \underset{\Sigma}{\parallel} Q) \setminus H$ of the processes ensures that Low's event l does not preempt the h. Alternatively, High can pass a different value by not performing his event; this is modelled by the process $Q \mathrel{\widehat{=}} l \rightarrow STOP$, which allows Low to perform his event.

However, we do not want to allow all such processes Q; in a state of P where only *tock*s and events of Low are available, High should not be able to block any Low events: High can only prevent events of Low from occurring by preempting them with one of his own events. For example, if $P = l \rightarrow STOP$, then the Q process $STOP$ should not be allowed, but $l \rightarrow STOP$ or $RUN(L)$ would be allowed. In order to capture this requirement, we need two pieces of notation:

- We say that a trace s is *H-urgent* if for every refusal set X that immediately precedes a *tock*, we have $H \subseteq X$; note that if the environment attempts to perform all H events, then every corresponding trace of the process in question will be H-urgent; in particular, this will be true of all traces of the process $P \underset{\Sigma}{\parallel} Q$ in the context $(P \underset{\Sigma}{\parallel} Q) \setminus H$.
- We write $s[X \cap L/X]$ for the trace that is obtained from s by replacing every refusal set X with $X \cap L$.

We therefore restrict ourselves to processes Q satisfying the following condition for the given P:

$$\forall\, s \in \mathcal{T}[\![P \underset{\Sigma}{\parallel} Q]\!] \bullet s \text{ is } H\text{-urgent} \Rightarrow s[X \cap L/X] \in \mathcal{T}[\![P]\!].$$

This implies that if the environment attempts to perform all H events (so s is H-urgent) then if any events from L are refused, it is because P is refusing those events.

This condition prevents examples like the following:

$$\begin{aligned} P &\mathrel{\widehat{=}} h1 \rightarrow (l1 \rightarrow STOP \,\Box\, l2 \rightarrow STOP) \\ &\qquad \Box\, h2 \rightarrow (l1 \rightarrow STOP \,\Box\, l2 \rightarrow STOP), \\ Q &\mathrel{\widehat{=}} h1 \rightarrow l1 \rightarrow STOP. \end{aligned}$$

This shouldn't be allowed: High shouldn't be able to determine which of $l1$ and $l2$ becomes available for Low after $h1$. This is forbidden by the above condition: $P \underset{\Sigma}{\parallel} Q$ has H-urgent refusal trace $\langle \bullet, h1, \{l2\} \cup H, tock, \bullet \rangle$, but P does not have trace $\langle \bullet, h1, \{l2\}, tock, \bullet \rangle$.

In fact, as explained earlier, we have to consider all ways in which nondeterminism in P might be resolved, by considering all refinements of P—that is, all the ways in which P might act—and so we require that all refinements R of P satisfy the above equation:

$$\forall\, R \sqsupseteq_T P \bullet \forall\, s \in \mathcal{T}[\![R \underset{\Sigma}{\parallel} Q]\!] \bullet s \text{ is } H\text{-urgent} \Rightarrow s[X \cap L/X] \in \mathcal{T}[\![R]\!]. \qquad (1)$$

The condition is satisfied by the process $RUN(L)$ (in the place of Q), and so we will make frequent use of this process when defining strategies for High.

The following lemma shows that, under the above condition, considering compositions of the form $(P \underset{\Sigma}{\parallel} Q) \setminus H$ is reasonable in the sense that it does not allow High to distinguish too many processes. If Low can distinguish two behaviours of High when we consider compositions of this form, then he can also distinguish two behaviours of High when we consider compositions of the more natural form $(R \underset{H}{\parallel} Q') \setminus H$ for $Q' \in CSP_H$ and $R \sqsupseteq_T P$.

Lemma 2 If Q_0 and Q_1 satisfy equation (1), and $(P \underset{\Sigma}{\parallel} Q_0) \setminus H \not\equiv_T (P \underset{\Sigma}{\parallel} Q_1) \setminus H$ then there exist $R \sqsupseteq_T P$ and $Q_0', Q_1' \in CSP_H$ such that

$$(R \underset{H}{\parallel} Q_0') \setminus H \not\equiv_T (R \underset{H}{\parallel} Q_1') \setminus H.$$

(We conjecture that the lemma can be strengthened so that the result talks about P rather than a refinement R of P.) This lemma is proved in Appendix A.

4.3 Defining information flow cardinality

We presume that High and Low have devised some strategy for passing values in the range $\{0, \ldots, N-1\}$. What form must that strategy take? We suppose that for each value k that High wants to pass, he will act like some process $\mathcal{Q}(k)$, as above; we will therefore model High's behaviour as a function of the form[4] $\mathcal{Q} : \mathbb{N} \nrightarrow CSP$ where every $\mathcal{Q}(k)$ satisfies equation (1).

Given some particular High strategy $\mathcal{Q}$, Low's possible views of the system will be given by $\{(P \underset{\Sigma}{\parallel} \mathcal{Q}(k)) \setminus H \mid k \in \operatorname{dom} \mathcal{Q}\}$. However, there might be no strategy for Low that distinguishes all of these. We therefore ask whether a particular test T for Low distinguishes these processes: we consider the results obtained by the testing process T when interacting with the above processes, and ask whether the results obtained are those that High is trying to send.

Recall that we write $results(S, T)$ for the results obtained by test T when interacting with system S. When we are talking about a system composed of the given process and a high level process, we will overload the *results* function to take those two processes as separate arguments:

$$results(P, Q, T) \mathrel{\widehat{=}} results((P \underset{\Sigma}{\parallel} Q) \setminus H, T).$$

For example, recall

$$P_1 \mathrel{\widehat{=}} h \rightarrow l \rightarrow STOP \overset{1}{\triangleright} STOP,$$

[4]The notation $A \nrightarrow B$ represents *partial* functions from A to B.

and consider the strategy defined by:

$$\mathcal{Q}(0) \mathrel{\widehat{=}} RUN(L), \qquad \mathcal{Q}(1) \mathrel{\widehat{=}} h \rightarrow RUN(L),$$
$$T \mathrel{\widehat{=}} l \rightarrow SUCCESS(1) \stackrel{1}{\triangleright} SUCCESS(0).$$

Then $(P_1 \underset{\Sigma}{\parallel} \mathcal{Q}(0)) \setminus H = STOP$ so we have

$$results(P_1, \mathcal{Q}(0), T) = \{0\}.$$

And $(P_1 \underset{\Sigma}{\parallel} \mathcal{Q}(1)) \setminus H = l \rightarrow STOP$ so we have

$$results(P_1, \mathcal{Q}(1), T) = \{1\}.$$

We should only consider strategies of High and Low that are compatible, in the sense that if High wants to send a value k, then Low obtains the result k. We will write $ok(P, \mathcal{Q}, T)$ to capture this condition:

$$ok(P, \mathcal{Q}, T) \mathrel{\widehat{=}} \forall\, k \in \operatorname{dom} \mathcal{Q} \bullet results(P, \mathcal{Q}(k), T) = \{k\} \text{ and } \mathcal{Q}(k) \text{ satisfies (1).}$$

For example, the process P_1 satisfies this condition with the above strategy.

Given some process P and some strategy represented by $\mathcal{Q} \in \mathbb{N} \nrightarrow CSP$ and $T \in Test$ such that $ok(P, \mathcal{Q}, T)$, the associated information flow is the number of different values that can be sent, i.e., $\#\operatorname{dom} \mathcal{Q}$. However, as discussed in Section 3, particularly Example 3, we want to consider not just P, but also all refinements R of P; the information flow cardinality is then the maximum flow achievable over all such R, and over all corresponding $\mathcal{Q}$ and T (such that $ok(R, \mathcal{Q}, T)$):

$$IFC(P) \mathrel{\widehat{=}} \max\{\#\operatorname{dom} \mathcal{Q} \mid \mathcal{Q} \in \mathbb{N} \nrightarrow CSP \wedge T \in Test \wedge R \in CSP \wedge P \sqsubseteq_T R \wedge ok(R, \mathcal{Q}, T)\}.$$

If the set in the above definition is unbounded, then we will define the information flow cardinality to be infinite.

For example, this definition gives an information flow cardinality of 2 for the process P_1, as illustrated by the above $\mathcal{Q}$ and T. Clearly, High cannot pass more information than this, because there are only two ways in which High can interact with the system: either he performs his event before the first *tock*, or he never performs it.

4.4 Examples

Recall the process

$$P_5 \mathrel{\widehat{=}} (l \rightarrow STOP \,\Box\, h \rightarrow STOP) \stackrel{1}{\triangleright} STOP.$$

The appropriate strategy for High and Low to adopt is the following:

$$\mathcal{Q}(0) \mathrel{\widehat{=}} RUN(L), \qquad \mathcal{Q}(1) \mathrel{\widehat{=}} h \rightarrow RUN(L),$$
$$T \mathrel{\widehat{=}} l \rightarrow SUCCESS(0) \overset{1}{\triangleright} SUCCESS(1).$$

Note that $\mathcal{Q}(1)$ prevents an initial l from occurring by preempting it with h. With this strategy we have

$$results(P_5, \mathcal{Q}(0), T) = \{0\} \quad \text{and} \quad results(P_5, \mathcal{Q}(1), T) = \{1\}.$$

Note also that the $\mathcal{Q}(k)$ processes satisfy equation (1): in particular, $P_5 \underset{\Sigma}{\parallel} \mathcal{Q}(1)$ cannot refuse H until the h has occurred, after which $\mathcal{Q}(1)$ does not block events of L. Hence this process has an information flow cardinality of at least 2. Clearly High cannot pass more information than this, because there are only two ways in which he can interact with the system: either he performs his event before the first *tock*, or he never performs it.

As another example, consider

$$\begin{pmatrix} h1 \rightarrow (l \rightarrow l1 \rightarrow STOP \,\Box\, l' \rightarrow l1 \rightarrow STOP) \\ \Box\; h2 \rightarrow (l \rightarrow l2 \rightarrow STOP \,\Box\, l' \rightarrow l1 \rightarrow STOP) \\ \Box\; h3 \rightarrow (l \rightarrow l1 \rightarrow STOP \,\Box\, l' \rightarrow l2 \rightarrow STOP) \\ \Box\; h4 \rightarrow (l \rightarrow l2 \rightarrow STOP \,\Box\, l' \rightarrow l2 \rightarrow STOP) \end{pmatrix} \overset{1}{\triangleright} STOP\,.$$

An information flow cardinality of 3 can be obtained using the strategy

$$\begin{aligned} \mathcal{Q}(0) &\mathrel{\widehat{=}} RUN(L), \\ \mathcal{Q}(1) &\mathrel{\widehat{=}} h1 \rightarrow RUN(L), \\ \mathcal{Q}(2) &\mathrel{\widehat{=}} h2 \rightarrow RUN(L), \\ T &\mathrel{\widehat{=}} l \rightarrow (l1 \rightarrow SUCCESS(1) \,\Box\, l2 \rightarrow SUCCESS(2)) \overset{1}{\triangleright} SUCCESS(0). \end{aligned}$$

However, no higher information flow cardinality can be obtained, for example by exploiting the l' branches: Low has to commit himself to either exploring the l branches, or exploring the l' branches, or maybe nondeterministically searching either, for example with a test like:

$$\begin{pmatrix} l \rightarrow (l1 \rightarrow SUCCESS(1) \,\Box\, l2 \rightarrow SUCCESS(2)) \\ \Box\; l' \rightarrow (l2 \rightarrow SUCCESS(2) \,\Box\, l1 \rightarrow SUCCESS(1)) \end{pmatrix} \overset{1}{\triangleright} SUCCESS(0)$$

(which would distinguish the High behaviours $RUN(L)$, $h1 \rightarrow RUN(L)$ and $h4 \rightarrow RUN(L)$); but in no case can Low explore all of the resulting process tree.

This example also shows why we have adopted a testing-style definition, as opposed to a definition such as

$$\#\{\mathcal{R}[\![(P \underset{\Sigma}{\parallel} Q) \setminus H]\!] \mid Q \in CSP\}$$

(maybe with some additional restrictions on Q), that is, the number of different ways that the process might look from Low's point of view. Such a definition would have given an information flow cardinality of 5 for the above process, as $RUN(L)$, $h1 \rightarrow RUN(L)$, $h2 \rightarrow RUN(L)$, $h3 \rightarrow RUN(L)$ and $h4 \rightarrow RUN(L)$ would all cause the system to look different from Low's point of view.

We now consider an example that shows why we insist—via the *ok* predicate—that the results caused by a particular $\mathcal{Q}(k)$ are precisely $\{k\}$: it is not enough for different $\mathcal{Q}(k)$ processes to simply lead to different sets of results. Consider

$$P \mathrel{\widehat{=}} \begin{pmatrix} h1 \rightarrow l1 \rightarrow STOP \\ \Box\ h2 \rightarrow l2 \rightarrow STOP \\ \Box\ h3 \rightarrow (l1 \rightarrow STOP \,\Box\, l2 \rightarrow STOP) \end{pmatrix} \stackrel{1}{\triangleright} STOP,$$

$$T \mathrel{\widehat{=}} (l1 \rightarrow SUCCESS(1) \,\Box\, l2 \rightarrow SUCCESS(2)) \stackrel{1}{\triangleright} SUCCESS(0).$$

These give different sets of results for the four high level processes $\mathcal{Q}(0) \mathrel{\widehat{=}} RUN(L)$, $\mathcal{Q}(1) \mathrel{\widehat{=}} h1 \rightarrow RUN(L)$, $\mathcal{Q}(2) \mathrel{\widehat{=}} h2 \rightarrow RUN(L)$ and $\mathcal{Q}(3) \mathrel{\widehat{=}} h3 \rightarrow RUN(L)$; however the results from $\mathcal{Q}(3)$—namely 1 and 2—are both results that could be obtained from other high level behaviours—$\mathcal{Q}(1)$ and $\mathcal{Q}(2)$—and so this does not contribute any additional information flow.

Finally, we consider an example that gives an infinite information flow cardinality. Let

$$P \mathrel{\widehat{=}} h \rightarrow l \rightarrow WAIT\,1\,;\, P \stackrel{1}{\triangleright} STOP\,.$$

High can pass a value k by performing k h events:

$$\begin{aligned} \mathcal{Q}(0) &\mathrel{\widehat{=}} RUN(L), \\ \mathcal{Q}(k+1) &\mathrel{\widehat{=}} h \rightarrow l \rightarrow WAIT\,1\,;\, \mathcal{Q}(k), \\ T(k) &\mathrel{\widehat{=}} l \rightarrow WAIT\,1\,;\, T(k+1) \stackrel{1}{\triangleright} SUCCESS(k). \end{aligned}$$

Then this strategy, using the test $T(0)$, will pass an arbitrary value, so the information flow cardinality of this process is infinite.

5 No information flow

In this section we give a number of results about our model and definition. Our main goal is to understand under what circumstances our definition gives a process an information flow cardinality of 1—i.e., no information flow.

In [FG95, Foc96, FGM00], Focardi, Gorrieri and Martinelli define several security properties, jointly termed *Non Deducibility on Composition (NDC)*, which can be written in CSP as:

$$P \textbf{ sat } NDC \mathrel{\widehat{=}} \forall\, Q \in CSP_H \bullet P \underset{H}{\parallel} STOP \equiv (P \underset{H}{\parallel} Q) \setminus H,$$

where the nature of the equivalence depends upon the security property in question. We argued earlier that testing equivalence is appropriate in our setting, so we define *Testing Non Deducibility on Composition (TNDC)* by:

$$P \textbf{ sat } TNDC \;\widehat{=}\; \forall\, Q \in CSP_H \bullet P \underset{H}{\parallel} STOP \equiv_T (P \underset{H}{\parallel} Q) \setminus H.$$

This condition is equivalent to:

$$\forall\, Q_0, Q_1 \in CSP_H \bullet (P \underset{H}{\parallel} Q_0) \setminus H \equiv_T (P \underset{H}{\parallel} Q_1) \setminus H.$$

We propose the following strengthening of TNDC: a process P satisfies *Strong Testing Non Deducibility on Composition (STNDC)* as follows:

$$P \textbf{ sat } STNDC \;\widehat{=}\; \forall\, R \sqsupseteq_T P \bullet R \textbf{ sat } TDNC.$$

Our main result, in Section 5.2, is that our definition gives information flow cardinality of 1 to precisely those processes that satisfy STNDC.

We prefer the condition STNDC to TNDC (and the other nondeducibility on composition properties) because it overcomes the following objection of Forster and Roscoe [For99]. Let $LEAK$ be any insecure process, and consider the two processes $LEAK \sqcap CHAOS(L)$ and $LEAK \;|||\; CHAOS(L)$. These processes have clearly insecure behaviours, coming from $LEAK$; however they satisfy $TNDC$ (and the other nondeducibility on composition properties), because the insecure behaviour is masked by behaviours of $CHAOS(L)$. These processes do not satisfy STNDC, because they are refined by $LEAK$.[5]

5.1 Forcing refusal traces

We will need the following technical lemma, which shows that given a process P with a particular refusal trace s, we can construct a refinement R that will always follow s in a suitable environment; in other words, we can remove all the nondeterminism that might cause the process to deviate from s.

Lemma 3 Let $s \in \mathcal{R}[\![P]\!]$. Then there is a process R that testing-refines P, and that "forces" s; that is, R cannot refuse the events in s, and cannot perform any of the events of refusal sets in s:

$$\forall\, s'^\frown\langle A, a\rangle \leqslant s \bullet s'^\frown\langle A \cup^* \{a\}\rangle \notin \mathcal{R}[\![R]\!],$$
$$\forall\, s'^\frown\langle A\rangle \leqslant s \,;\, a \in A \bullet s'^\frown\langle \bullet, a, \bullet\rangle \notin \mathcal{R}[\![R]\!].$$

[5] In [FR02], Focardi and Rossi introduce the notion of *Persistent Non Deducibility on Compositions*, namely that all reachable states of a process satisfy Non Deducibility on Compositions. This property appears very similar to Strong Non Deducibility on Compositions; we intend to investigate the relationship.

Proof: Let $\bar{s}$ be obtained from s by replacing each refusal set X such that $s'^\frown\langle X\rangle \leqslant s$ by $\bar{X} \mathrel{\widehat{=}} X \cup^* \{a \mid s'^\frown\langle \bullet, a, \bullet\rangle \notin \mathcal{R}[\![P]\!]\}$. That is, every refusal set is closed so as to include all events that cannot be performed by P.

Let P **after** s_0 represent the behaviour of P following trace s_0, defined semantically by:

$$\mathcal{R}[\![P \textsf{ after } s_0]\!] \mathrel{\widehat{=}} \{u \mid s_0{}^\frown u \in \mathcal{R}[\![P]\!]\}.$$

Note that P **after** s_0 is a process provided $s_0{}^\frown\langle\bullet\rangle \in \mathcal{R}[\![P]\!]$.

We define a family of processes $R(s', s_0)$ where s' and s_0 are such that $\bar{s} = s_0{}^\frown s'$. The idea is that $R(s', s_0)$ will refine P **after** s_0 and will "force" s'.

$$\begin{aligned}
R(\langle X\rangle, s_0) \mathrel{\widehat{=}}\ & ?b : \Sigma - X \to (P \textsf{ after } (s_0{}^\frown\langle\bullet, b\rangle)) \\
& \overset{1}{\triangleright} \\
& P \textsf{ after } (s_0{}^\frown\langle X, tock\rangle), \\
R(\langle X, a\rangle{}^\frown s', s_0) \mathrel{\widehat{=}}\ & ?b : \Sigma - X \to \text{if } b = a \text{ then } R(s', s_0{}^\frown\langle X, a\rangle) \\
& \qquad\qquad\qquad\qquad \text{else } (P \textsf{ after } (s_0{}^\frown\langle\bullet, b\rangle)) \\
& \overset{1}{\triangleright} \\
& (P \textsf{ after } (s_0{}^\frown\langle X, tock\rangle)), \\
R(\langle X, tock\rangle{}^\frown s', s_0) \mathrel{\widehat{=}}\ & ?b : \Sigma - X \to (P \textsf{ after } (s_0{}^\frown\langle\bullet, b\rangle)) \\
& \overset{1}{\triangleright} \\
& R(s', s_0{}^\frown\langle X, tock\rangle).
\end{aligned}$$

We define $R \mathrel{\widehat{=}} R(\bar{s}, \langle\rangle)$.

First, note that the invariant $\bar{s} = s_0{}^\frown s'$ is maintained by the recursive definition.

We now show that $P \sqsubseteq_T R$. We show that

$$\forall\, t \in \mathcal{R}[\![R(s', s_0)]\!] \bullet s_0{}^\frown\dot{t} \in \mathcal{R}[\![P]\!]$$

by induction on the length of t, by performing a case analysis on s'.

- Case $s' = \langle X\rangle$. There are three possibilities for t:
  - Case $t = \langle Y\rangle$ for $Y \subseteq^* X$. Then $\dot{t} = \langle\bullet\rangle$, so $s_0{}^\frown\dot{t} \in \mathcal{R}[\![P]\!]$ by Axiom R2.
  - Case $t = \langle Y, b\rangle{}^\frown t'$ for $Y \subseteq^* X,\ b \in \Sigma - X,\ t' \in \mathcal{R}[\![P \textsf{ after } (s_0{}^\frown\langle\bullet, b\rangle)]\!]$. Then by definition of $\bar{s}$, $\langle\bullet, b, \bullet\rangle \in \mathcal{R}[\![P \textsf{ after } s_0]\!]$ and so $P \textsf{after} (s_0{}^\frown\langle\bullet, b\rangle)$ is well defined. Hence $s_0{}^\frown\langle\bullet, b\rangle{}^\frown t' \in \mathcal{R}[\![P]\!]$, so $s_0{}^\frown\dot{t} \in \mathcal{R}[\![P]\!]$ by Axiom R2.
  - Case $t = \langle Y, tock\rangle{}^\frown t'$ for $Y \subseteq^* X,\ t' \in \mathcal{R}[\![P \textsf{ after } (s_0{}^\frown\langle X, tock\rangle)]\!]$. By definition of $\bar{s}$, and by Axioms R4 and R6, $s_0{}^\frown\langle X, tock, \bullet\rangle \in \mathcal{R}[\![P]\!]$, so $P \textsf{ after } (s_0{}^\frown\langle X, tock\rangle)$ is well defined. Hence $s_0{}^\frown\langle X, tock\rangle{}^\frown t' \in \mathcal{R}[\![P]\!]$ and so $s_0{}^\frown\dot{t} \in \mathcal{R}[\![P]\!]$ by Axiom R2.

- Case $s' = \langle X, a\rangle^\frown s''$. There are four possibilities for t, three of which are identical to corresponding cases for $s' = \langle X\rangle$, and so are omitted. The interesting case is when $t = \langle Y, a\rangle^\frown t'$ for $Y \subseteq^* X,\ t' \in \mathcal{R}[\![R(s'', s_0{}^\frown\langle X, a\rangle)]\!]$. Then by the inductive hypothesis, $s_0{}^\frown\langle X, a\rangle^\frown \dot{t'} \in \mathcal{R}[\![P]\!]$ so $s_0{}^\frown\dot{t} \in \mathcal{R}[\![P]\!]$.
- Case $s' = \langle X, tock\rangle^\frown s''$. There are three possibilities for t, most of which are again identical to corresponding cases for $s' = \langle X\rangle$. The interesting case is when $t = \langle Y, tock\rangle^\frown t'$ for $Y \subseteq^* X,\ t' \in \mathcal{R}[\![R(s'', s_0{}^\frown\langle X, tock\rangle)]\!]$. Then by the inductive hypothesis, $s_0{}^\frown\langle X, tock\rangle^\frown \dot{t'} \in \mathcal{R}[\![P]\!]$ so $s_0{}^\frown\dot{t} \in \mathcal{R}[\![P]\!]$.

Hence $\forall\, t \in \mathcal{R}[\![R]\!] \bullet \dot{t} \in \mathcal{R}[\![P]\!]$ and so $P \sqsubseteq_T R$.

Finally, by construction, R cannot refuse any events of s, and cannot perform any of the events of refusal sets in s. □

Note that the above lemma cannot be strengthened to talk about refinement in the standard refusal-testing model. Let

$$P = (a \rightarrow STOP \,\Box\, c \rightarrow b \rightarrow STOP) \setminus \{c\},$$

and let $s = \langle \bullet, a, \bullet\rangle$. Suppose R satisfies the conditions of the lemma. Then $\langle\{\}\rangle \in \mathcal{R}[\![R]\!]$, so $\langle\{\}, a, \bullet\rangle \in \mathcal{R}[\![R]\!]$ or $\langle\{a\}\rangle \in \mathcal{R}[\![R]\!]$ by Axiom R6, giving a contradiction in each case. The above construction would give

$$R = (a \rightarrow STOP \,\Box\, b \rightarrow STOP) \stackrel{1}{\triangleright} b \rightarrow STOP,$$

which is not an R-refinement of P, but is a T-refinement.

5.2 The main result

We now show that if two high level processes can make the system look different from Low's point of view, then we can find some testing strategy to give an information flow cardinality of 2.

Lemma 4 If $Q_0, Q_1 \in CSP_H$ are such that

$$(P \underset{H}{\parallel} Q_0) \setminus H \not\equiv_T (P \underset{H}{\parallel} Q_1) \setminus H,$$

then there exist $R \sqsupseteq_T P,\ Q_0', Q_1' \in CSP$ satisfying equation (1), and $T \in Test$ such that

$$results(R, Q_0', T) = \{0\} \quad \text{and} \quad results(R, Q_1', T) = \{1\}.$$

Proof: Suppose

$$s \in \mathcal{T}[\![(P \underset{H}{\parallel} Q_1) \setminus H]\!] \quad \text{and} \quad s \notin \mathcal{T}[\![(P \underset{H}{\parallel} Q_0) \setminus H]\!],$$

and let T be as in Lemma 1. The results of that lemma show

$$\mathit{results}((P \underset{H}{\parallel} Q_0) \setminus H, T) = \{0\} \quad \text{and} \quad \mathit{results}((P \underset{H}{\parallel} Q_1) \setminus H, T) \supseteq \{1\}.$$

However, we cannot be sure that 0 is not a member of $\mathit{results}((P \underset{H}{\parallel} Q_1) \setminus H, T)$, because of the possibility of nondeterminism.

Suppose s corresponds to the trace s' of $P \underset{H}{\parallel} Q_1$, and the traces s_P and s_Q of P and Q respectively, so that $\mathrm{trace}(s_P) = \mathrm{trace}(s')$, and the refusals of s_P include all the events of L from the corresponding refusal of s', by equation (1).

Let $R \sqsupseteq_T P$ be—as in Lemma 3—such that $\dot{s_P}$ is "forced". We construct a high level process corresponding to s'. Let $Q_1' \mathrel{\widehat{=}} Q(s')$ where

$$\begin{aligned}
Q(\langle \bullet, l \rangle^\frown s'') &\mathrel{\widehat{=}} l \rightarrow Q(s'') \,\Box\, ?\, l' : L - \{l\} \rightarrow \mathit{RUN}(L), \qquad \text{for } l \in L, \\
Q(\langle \bullet, h \rangle^\frown s'') &\mathrel{\widehat{=}} h \rightarrow Q(s''), \qquad \text{for } h \in H, \\
Q(\langle A, \mathit{tock} \rangle^\frown s'') &\mathrel{\widehat{=}} \mathit{RUN}(L) \overset{1}{\triangleright} Q(s''), \\
Q(\langle \bullet \rangle) &\mathrel{\widehat{=}} \mathit{RUN}(L).
\end{aligned}$$

Consider $(R \underset{\Sigma}{\parallel} Q_1') \setminus H$ with the test T. By construction, the combination of Q_1' and T force R to perform the trace s':

- In states where an event from H is due next, Q_1' forces it to happen;
- In states where an event from L is due next, Q_1' prevents events from H from happening, and T selects the appropriate event from L (L is hidden, so this event must happen–silently–rather than a *tock*);
- In states where a *tock* following refusal X is due next, R prevents all events in $X \cap L$, and Q_1' prevents all events from H. Hence $R \underset{\Sigma}{\parallel} Q_1'$ refuses $X \cup H$, so $(R \underset{\Sigma}{\parallel} Q_1') \setminus H$ refuses X, so the test selects the *tock*.

Further, R cannot refuse any of these events. Hence

$$\mathit{results}(R, Q_1', T) \;=\; \{1\}.$$

Note that Q_1' satisfies equation (1): after H-urgent traces, Q_1' must be refusing H, and in such states it offers all of L.

Finally, let $Q_0' \mathrel{\widehat{=}} Q_0 \mid\mid\mid \mathit{RUN}(L)$. Note that Q_0' satisfies equation (1). Then

$$(R \underset{\Sigma}{\parallel} Q_0') \setminus H = (R \underset{H}{\parallel} Q_0) \setminus H \sqsupseteq_T (P \underset{H}{\parallel} Q_0) \setminus H,$$

and so

$$\begin{aligned} \mathit{results}(R, Q_0', T) &= \mathit{results}((R \underset{\Sigma}{\parallel} Q_0') \setminus H, T) \\ &\subseteq \mathit{results}(P \underset{H}{\parallel} Q_0) \setminus H, T) \\ &= \{0\}. \end{aligned}$$

But, by construction, T always gives at least one result, so

$$\mathit{results}(R, Q_0', T) = \{0\}.$$

□

We can now prove our main theorem:

Theorem 5 A process P has information flow cardinality 1 iff for all refinements R, High cannot change the way the process appears to Low:

$$\mathit{IFC}(P) = 1 \;\Leftrightarrow\; \forall\, R \sqsupseteq_T P\,;\, Q_0, Q_1 \in \mathit{CSP}_H \bullet (R \underset{H}{\parallel} Q_0) \setminus H \equiv_T (R \underset{H}{\parallel} Q_1) \setminus H.$$

This is equivalent to

$$\mathit{IFC}(P) = 1 \;\Leftrightarrow\; P \textbf{ sat } \mathit{STNDC}.$$

Proof: We prove the contrapositive. In one direction:

$$\begin{array}{ll} & \neg(\forall\, R \sqsupseteq_T P\,;\, Q_0, Q_1 \in \mathit{CSP}_H \bullet (R \underset{H}{\parallel} Q_0) \setminus H \equiv_T (R \underset{H}{\parallel} Q_1) \setminus H) \\ \Leftrightarrow & \exists\, R \sqsupseteq_T P\,;\, Q_0, Q_1 \in \mathit{CSP}_H \bullet (R \underset{H}{\parallel} Q_0) \setminus H \not\equiv_T (R \underset{H}{\parallel} Q_1) \setminus H \\ \Rightarrow & \langle \text{Lemma 4} \rangle \\ & \exists\, R \sqsupseteq_T P \bullet \exists\, R' \sqsupseteq_T R\,;\, Q_0', Q_1' \in \mathit{CSP}\,;\, T \in \mathit{Test} \bullet \\ & \quad \mathit{results}(R', Q_0', T) = \{0\} \wedge \mathit{results}(R', Q_1', T) = \{1\} \wedge Q_0',\ Q_1' \text{ satisfy (1)} \\ \Rightarrow & \mathit{IFC}(P) \geqslant 2. \end{array}$$

In the other direction:

$$\begin{array}{ll} & \mathit{IFC}(P) \geqslant 2 \\ \Rightarrow & \exists\, R \sqsupseteq_T P\,;\, Q_0, Q_1 \in \mathit{CSP}\,;\, T \in \mathit{Test} \bullet \\ & \quad \mathit{results}(R, Q_0, T) = \{0\} \wedge \mathit{results}(R, Q_1, T) = \{1\} \wedge Q_0,\ Q_1 \text{ satisfy (1)} \\ \Rightarrow & \langle \text{definition of testing equivalence} \rangle \\ & \exists\, R \sqsupseteq_T P\,;\, Q_0, Q_1 \in \mathit{CSP} \bullet \\ & \quad (R \underset{\Sigma}{\parallel} Q_0) \setminus H \not\equiv_T (R \underset{\Sigma}{\parallel} Q_1) \setminus H \wedge Q_0,\ Q_1 \text{ satisfy (1)} \\ \Rightarrow & \langle \text{Lemma 2} \rangle \\ & \exists\, R \sqsupseteq_T P\,; \exists\, R' \sqsupseteq_T R\,;\, Q_0', Q_1' \in \mathit{CSP}_H \bullet (R' \underset{H}{\parallel} Q_0') \setminus H \not\equiv_T (R' \underset{H}{\parallel} Q_1') \setminus H \\ \Rightarrow & \neg(\forall\, R \sqsupseteq_T P\,;\, Q_0, Q_1 \in \mathit{CSP}_H \bullet (R \underset{H}{\parallel} Q_0) \setminus H \equiv_T (R \underset{H}{\parallel} Q_1) \setminus H). \end{array}$$

□

6 Bounded time information flow

Many processes can be used to pass an unbounded amount of information given sufficient time; however, they might still have a finite *rate* of information flow. In this section we extend the work of the previous sections to consider the amount of information passed in some finite interval, and hence define the rate of information flow.

We can define the results obtainable from a system S with test T before time $t+1$ as follows:

$$results_t(S, T) \mathrel{\widehat=} \{k \mid \exists\, n \leqslant t \bullet \langle \bullet, tock\rangle^n \frown \langle \bullet, \omega.k, \bullet\rangle \in \mathcal{R}[\![(S \underset{L}{\parallel} T) \setminus L]\!]\}.$$

Note that we only consider observations containing at most t tocks, i.e. those observations finishing before time $t+1$. The following definitions are obvious adaptations of previous ones.

$$\begin{aligned}
results_t(P, Q, T) &\mathrel{\widehat=} results_t((P \underset{\Sigma}{\parallel} Q) \setminus H, T),\\
ok_t(P, \mathcal{Q}, T) &\mathrel{\widehat=} \forall\, k \in \operatorname{dom} \mathcal{Q} \bullet results_t(P, \mathcal{Q}(k), T) = \{k\}\\
&\qquad\qquad \text{and } \mathcal{Q}(k) \text{ satisfies (1)},\\
IFC_t(P) &\mathrel{\widehat=} \max\{\#\operatorname{dom} \mathcal{Q} \mid \mathcal{Q} \in \mathbb{N} \nrightarrow CSP \wedge T \in Test \wedge\\
&\qquad\qquad R \in CSP \wedge R \sqsupseteq_T P \wedge ok_t(R, \mathcal{Q}, T)\}.
\end{aligned}$$

We can define the *long term information flow rate* (in bits per time unit) as follows:

$$LTIFR(P) \mathrel{\widehat=} \lim_{t\to\infty} \frac{\log_2(IFC_t(P))}{t} \qquad \text{if the limit exists.}$$

Of course, the above definition doesn't give the whole story. A process that leaked a gigabyte of information at time 0, but nothing subsequently, would have a long term information flow rate of zero, but would probably not be acceptable as a secure system.

6.1 Examples

As a first example, consider $P \mathrel{\widehat=} h \to l \to STOP$. Fix N, and consider the following strategy, where High passes the value k by performing an h after k time units:

$$\begin{aligned}
\mathcal{Q}(k) &= WAIT\ k\,;\, h \to l \to STOP, \qquad \text{for } k = 0, \ldots, N-1,\\
\mathcal{Q}(N) &= STOP,\\
T(k) &= l \to SUCCESS(k) \overset{1}{\triangleright} T(k+1), \qquad \text{for } k = 0, \ldots, N-1,\\
T(N) &= SUCCESS(N).
\end{aligned}$$

Then

$$results_N(P, \mathcal{Q}(k), T(0)) = \{k\} \qquad \text{for } k = 0, \ldots, N.$$

Clearly, Low cannot distinguish more behaviours than this within N time units, because he will only ever be able to perform zero or one l events. Hence $IFC_N(P) = N + 1$, for all N, so the long term information flow rate is

$$\lim_{N \to \infty} \frac{\log_2(N+1)}{N} = 0.$$

As another example, consider a processor, shared by High and Low, scheduled using a round robin scheduler with time quantum q. Scheduling consists of two phases: an execution phase, where one agent has control of the processor; and a contention phase, where it is decided who should have the processor next.

Let $PROC_L$ and $PROC_H$ denote the behaviour of the processor when scheduled for Low and High, respectively. During the execution phase, the processor acts like either $PROC_H$ or $PROC_L$ for a period of q, before being interrupted and moving to the contention phase:

$$\begin{aligned} EXEC_H &\mathrel{\widehat{=}} PROC_H \triangle_q CONTENTION_L, \\ EXEC_L &\mathrel{\widehat{=}} PROC_L \triangle_q CONTENTION_H. \end{aligned}$$

During the contention phase, High and Low can request the processor via events req_h and req_l, respectively. The scheduler will repeatedly offer one req event, but if it is not accepted within time w, the offer is withdrawn and the other req event offered. After an execution phase, the processor is first offered to the other agent:

$$\begin{aligned} CONTENTION_H &\mathrel{\widehat{=}} req_h \rightarrow EXEC_H \overset{w}{\triangleright} CONTENTION_L, \\ CONTENTION_L &\mathrel{\widehat{=}} req_l \rightarrow EXEC_L \overset{w}{\triangleright} CONTENTION_H. \end{aligned}$$

Initially, High has priority:

$$SCHEDULER \mathrel{\widehat{=}} CONTENTION_H.$$

It turns out that High and Low will be able to pass one bit per $q + w$ time units. Fix t, and let $n \mathrel{\widehat{=}} \lfloor \frac{t}{q+w} \rfloor$ (this will be the number of bits passable in time t). The strategy will involve High using the processor to pass a bit 1, and not using the processor to pass a bit 0. We pass the bits one at a time, least significant bit first. High's strategy to pass the value k can be modelled by the process $Q(k) \mid\mid\mid RUN(L)$ where:

$$\begin{aligned} Q(k) \mathrel{\widehat{=}} &\text{ if } k \bmod 2 = 1 \text{ then } req_h \rightarrow WAIT(q+w)\,;\, Q(k \mathbin{\mathsf{div}} 2) \\ &\text{else } WAIT(q+w)\,;\, Q(k \mathbin{\mathsf{div}} 2). \end{aligned}$$

Low's strategy is to attempt req_l events at times of the form $w + (q + w)i$, and is modelled by the test $WAIT\ w\ ;\ T(0,0)$, below. The test accumulates the result in the parameter r; the parameter i records the number of bits received so far.

$$\begin{aligned} T(i,r) &\mathrel{\widehat{=}} (req_l \rightarrow WAIT(q+w)\ ;\ T(i+1,r)) \overset{w}{\triangleright} (WAIT\ q\ ;\ T(i+1, 2^i + r)), \\ &\qquad \text{for } i = 0, \ldots, n-1, \\ T(n,r) &\mathrel{\widehat{=}} SUCCESS(r). \end{aligned}$$

Note that High will succeed in performing req_h at times that are multiples of $q+w$. Low will perform req_l at times of the form $w + (q + w)i$ if and only if High did not perform req_h at the corresponding time $(q + w)i$.

This strategy achieves an information flow cardinality of 2^n where $n = \lfloor \frac{t}{q+w} \rfloor$, and so the long term information flow rate is $\frac{1}{q+w}$.

7 Conclusions

In this paper we have presented a formal definition of the quantity of information passed from a high level user to a low level user in a system, based upon the number of behaviours of High that can be accurately distinguished from Low's point of view.

The correct definition proved remarkably difficult to find: many alternative definitions failed to give the correct values for example processes. The aspect that proved hardest to get right was that High might be able to prevent events of Low occurring by preempting them with his own events.

The model appears to be accurate: it gives the expected results for all the examples considered. We have proved that it agrees with a more intuitive definition on those processes that exhibit no flow of information.

Our choice of CSP as the modelling language allowed us to capture a number of subtleties, such as those surrounding nondeterminism, that might be missed in other formalisms. Of course, the price of this is added complexity in the model; however, we believe this is a price worth paying, in order to end up with a model powerful enough to accurately deal with such issues.

We consider now a few possible extensions to this work.

It would clearly be useful to have an automated procedure for calculating the cardinality of information flow in a system. We believe there is some scope for considering all possible High processes that can successfully interact with the system (i.e. only those High processes that attempt events the system might actually offer), and all corresponding Low tests (i.e. only those tests that attempt events the system might actually offer), and identifying which subsets satisfy the *ok* predicate. Making this efficient might prove challenging.

It would also be useful to develop more systematic ways of calculating the cardinality of information flow in a given system: proving lower bounds can

be done simply be exhibiting a suitable strategy, which is normally reasonably straightforward; but proofs of upper bounds tend to be rather ad hoc.

As explained in the introduction, we have treated nondeterminism by considering the worst case, rather than attempting to assign probabilities to the different possibilities. It would be interesting to compare the two approaches. However, to do so formally would need a semantic model that correctly treats the interplay between probabilities and nondeterminism; building such a semantic model has proved remarkably difficult (most existing models allow nondeterminism in one component of a system to be resolved in a way that depends upon probabilistic behaviour in another component, even when there has been no flow of information from the latter component to the former).

We have assumed that Low's time quantum—the degree of accuracy with which Low can tell the time—was equal to one time unit. This assumption might not be valid, so it would be interesting to relax it. All delays in the system are assumed to be an integer number of time units, and so we cannot necessarily achieve equality by a rescaling of the time unit, particularly when the time quantum is greater than one time unit.

In this paper we effectively ignored refusals that do not immediately precede *tock*s. We believe that the semantic model that ignores such refusals—and is thus more abstract than the refusal traces model—is worthy of further study in its own right, because it captures an intuitive notion of process equivalence, as shown by Lemma 1.

Acknowledgements

I would like to thank Bill Roscoe, Steve Schneider, Philippa Broadfoot and Joël Ouaknine for useful discussions about information flow. I would also like to thank the anonymous referees for useful comments. This work was partially supported by a grant from 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.

[dNH84] R. de Nicola and M. C. B. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[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.

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

[FR02] Riccardo Focardi and Sabina Rossi. Information flow security in dynamic contexts. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, pages 307–319, 2002.

[Gra91] James W. Gray, III. Towards a mathematical foundation for information flow security. In *Proceedings of the 1991 IEEE Computer Society Symposium on Research in Computer Security and Privacy*, 1991.

[McC88] D. McCullough. Covert channels and degrees of insecurity. In *Proceedings of the First IEEE Computer Security Foundations Workshop*, pages 1–33, 1988.

[Mil87] Jonathan Millen. Covert channel capacity. In *Proceedings of the 1987 IEEE Computer Society Symposium on Computer Security and Privacy*, 1987.

[Oua00] Joel Ouaknine. *Discrete Analysis of Continuous Behaviour in Real-Time Concurrent Systems*. D.Phil thesis, Oxford University, 2000.

[Phi86] Iain Phillips. Refusal testing. In *Proceedings of 13th International Colloquium on Automata, Languages and Programming, LNCS 226*, pages 304–313. Spring Verlag, 1986.

[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.

[Sch99] Steve Schneider. *Concurrent and Real-time Systems: The CSP Approach*. Wiley, 1999.

[SW63] C. E. Shannon and W. Weaver. *The Mathematical Theory of Communication*. University of Illinois Press, 1963.

[US 85] US Department of Defense. *DoD Trusted Computer System Evaluation Criteria*, 1985. DOD 5200.28-STD.

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

A Proofs of lemmas

In this appendix, we prove Lemma 2. We will need a couple of additional results.

Observation 6 Suppose $R \sqsupseteq_T P$ is such that the trace $s = \dot{s}$ is forced, as in Lemma 3, and let $s' \in \mathcal{T}[\![P]\!]$; then $s' \in \mathcal{T}[\![R]\!]$ unless one of the following holds:

1. s' includes an event that is refused in s:

$$\exists\, u, u', a, A \bullet u^\frown\langle A, tock\rangle \leqslant s \wedge u'^\frown\langle \bullet, a\rangle \leqslant s' \wedge \\ \mathrm{trace}(u) = \mathrm{trace}(u') \wedge a \in A.$$

2. s' includes a refusal including events that could be performed in s:[6]

$$\exists\, u, u', a, A, B \bullet u^\frown\langle B, a\rangle \leqslant s \wedge u'^\frown\langle A, tock\rangle \leqslant s' \wedge \\ \mathrm{trace}(u) = \mathrm{trace}(u') \wedge \\ A \cap \mathrm{inits}(P \mathsf{\ after\ } (u^\frown\langle \bullet\rangle)) \not\subseteq^* B.$$

Lemma 7 Suppose $s = \dot{s}$, $(s' \setminus H) = s^\frown\langle\bullet\rangle$, $s'[X \cap L/X] \in \mathcal{T}[\![P]\!]$, and $s[X \cap L/X]^\frown s'' \in \mathcal{T}[\![P]\!]$, for some s''. Let $R \sqsupseteq_T P$ be defined as in Lemma 3 to "force" $s[X \cap L/X]^\frown s''$. Then $s'[X \cap L/X] \in \mathcal{T}[\![R]\!]$.

Proof:

Assume the conditions of the Lemma, and suppose the result does not hold. Then by Observation 6, we have one of the following:

1. There is some u, u', a, A with $\mathrm{trace}(u) = \mathrm{trace}(u')$, $a \in A$ and

$$u^\frown\langle A, tock\rangle \leqslant s[X \cap L/X]^\frown s'' \wedge u'^\frown\langle \bullet, a\rangle \leqslant s'[X \cap L/X].$$

[6] $\mathrm{inits}(Q)$ represents the set of events that can be performed by Q in its initial state.

Then $a \in A \subseteq L$, so

$$\begin{aligned}\mathrm{trace}(u') \setminus H^\frown\langle a\rangle &= \mathrm{trace}((u'^\frown\langle \bullet, a\rangle) \setminus H) \leqslant \mathrm{trace}(s'[X \cap L/X] \setminus H)\\ &= \mathrm{trace}(s' \setminus H) = \mathrm{trace}(s) \leqslant \mathrm{trace}(s^\frown s'').\end{aligned}$$

And

$$\begin{aligned}\mathrm{trace}(u) \setminus H^\frown\langle tock\rangle &= \mathrm{trace}(u^\frown\langle A, tock\rangle) \leqslant \mathrm{trace}(s[X \cap L/X]^\frown s'')\\ &= \mathrm{trace}(s^\frown s'').\end{aligned}$$

Which gives a contradiction, because $\mathrm{trace}(u) = \mathrm{trace}(u')$ but $a \neq tock$.

2. There is some u, u', a, A, B with $\mathrm{trace}(u) = \mathrm{trace}(u')$ and

$$\begin{array}{l}u^\frown\langle B, a\rangle \leqslant s[X \cap L/X]^\frown s'' \wedge u'^\frown\langle A, tock\rangle \leqslant s'[X \cap L/X] \wedge\\ A \cap \mathrm{inits}(P \mathsf{\ after\ } (u^\frown\langle \bullet\rangle)) \not\subseteq^* B.\end{array}$$

But $s' \setminus H = s^\frown\langle \bullet\rangle$ so we have $a = tock$, and $A = A \cap L = B \cap L = B$, giving a contradiction.

□

If s is a trace, we write $Tr(s)$ for the "trace process" for s, i.e., the process that will perform just the events of s:

$$\begin{aligned}Tr(\langle\rangle) &\mathrel{\widehat{=}} STOP,\\ Tr(\langle a\rangle^\frown s') &\mathrel{\widehat{=}} a \rightarrow Tr(s'),\\ Tr(\langle tock\rangle^\frown s') &\mathrel{\widehat{=}} WAIT\ 1\ ;\ Tr(s').\end{aligned}$$

We now prove Lemma 2: if Q_0 and Q_1 satisfy equation (1), and

$$(P \underset{\Sigma}{\parallel} Q_0) \setminus H \not\equiv_T (P \underset{\Sigma}{\parallel} Q_1) \setminus H$$

then

$$\exists\, R \sqsupseteq_T P\ ;\ Q_0', Q_1' \in CSP_H \bullet (R \underset{H}{\parallel} Q_0') \setminus H \not\equiv_T (R \underset{H}{\parallel} Q_1') \setminus H.$$

Proof of Lemma 2: Consider $P \underset{H}{\parallel} STOP \equiv_T (P \underset{H}{\parallel} STOP) \setminus H$. This cannot be equivalent to both $(P \underset{\Sigma}{\parallel} Q_0) \setminus H$ and $(P \underset{\Sigma}{\parallel} Q_1) \setminus H$, so without loss of generality assume

$$P \underset{H}{\parallel} STOP \not\equiv_T (P \underset{\Sigma}{\parallel} Q_1) \setminus H.$$

We will take $Q_0' \mathrel{\widehat{=}} STOP$. We consider cases as follows:

Case 1 There is some $s \in \mathcal{T}[\![(P \underset{\Sigma}{\parallel} Q_1) \setminus H]\!]$ such that $s \notin \mathcal{T}[\![P \underset{H}{\parallel} STOP]\!]$. Then there is some trace s' such that $s' \setminus H = s$, s' is H-urgent, and $s' \in$

$\mathcal{T}[\![P \underset{\Sigma}{\|} Q_1]\!]$. Hence $s'[X \cap L/X] \in \mathcal{T}[\![P]\!]$ by equation (1). Let $R \mathrel{\widehat{=}} P$ and $Q_1' \mathrel{\widehat{=}} Tr(\text{trace}(s') \upharpoonright H)$. Then $s' \in \mathcal{T}[\![P \underset{H}{\|} Q_1']\!]$ and so $s \in \mathcal{T}[\![(P \underset{H}{\|} Q_1') \setminus H]\!]$. Hence $(P \underset{H}{\|} STOP) \setminus H \not\equiv_T (P \underset{H}{\|} Q_1') \setminus H$, as required.

Case 2 There is some $s^\frown\langle \bullet, a, \bullet\rangle \in \mathcal{T}[\![P \underset{H}{\|} STOP]\!]$ such that $s^\frown\langle \bullet, a, \bullet\rangle \notin \mathcal{T}[\![(P \underset{\Sigma}{\|} Q_1) \setminus H]\!]$, and $a \in L$. Take s minimal, so $s^\frown\langle \bullet\rangle \in \mathcal{T}[\![(P \underset{\Sigma}{\|} Q_1) \setminus H]\!]$.

Then $s[X \cap L/X]^\frown\langle \bullet, a, \bullet\rangle \in \mathcal{T}[\![P]\!]$. Let $R \sqsupseteq_T P$ be as in Lemma 3 such that $s[X \cap L/X]^\frown\langle \bullet, a, \bullet\rangle$ is "forced". Then $s[X \cap L/X]^\frown\langle \{a\}, tock, \bullet\rangle \notin \mathcal{T}[\![R]\!]$ so

$$s^\frown\langle \{a\}, tock, \bullet\rangle \notin \mathcal{T}[\![(R \underset{H}{\|} STOP) \setminus H]\!].$$

Now, $s^\frown\langle \bullet\rangle \in \mathcal{T}[\![(P \underset{\Sigma}{\|} Q_1) \setminus H]\!]$, so there is some s' such that $s' \setminus H = s^\frown\langle \bullet\rangle$, s' is H-urgent, and $s' \in \mathcal{T}[\![P \underset{\Sigma}{\|} Q_1]\!]$. Hence $s'[X \cap L/X] \in \mathcal{T}[\![P]\!]$ by equation (1). Then from Lemma 7 we have $s'[X \cap L/X] \in \mathcal{T}[\![R]\!]$.

Let $Q_1' \mathrel{\widehat{=}} Tr(\text{trace}(s') \upharpoonright H)$; then $s' \in \mathcal{T}[\![R \underset{H}{\|} Q_1']\!]$, and hence $s^\frown\langle \bullet\rangle \in \mathcal{T}[\![(R \underset{H}{\|} Q_1') \setminus H]\!]$. Now, $s^\frown\langle \bullet, a, \bullet\rangle \notin \mathcal{T}[\![(P \underset{\Sigma}{\|} Q_1) \setminus H]\!]$, so[7] front $s'^\frown\langle \bullet, a, \bullet\rangle \notin \mathcal{T}[\![P \underset{\Sigma}{\|} Q_1]\!]$ so front $s'^\frown\langle \bullet, a, \bullet\rangle \notin \mathcal{T}[\![R \underset{\Sigma}{\|} Q_1]\!]$ because $R \sqsupseteq_T P$. Hence front $s'^\frown\langle \{a\}, tock, \bullet\rangle \in \mathcal{T}[\![R \underset{\Sigma}{\|} Q_1]\!]$ by Axioms R3, R4 and R6. So front $s'[X \cap L/X]^\frown\langle \{a\}, tock, \bullet\rangle \in \mathcal{T}[\![R]\!]$ by equation (1). Hence

$$s^\frown\langle \{a\}, tock, \bullet\rangle \in \mathcal{T}[\![(R \underset{H}{\|} Q_1') \setminus H]\!].$$

So $(R \underset{H}{\|} Q_1') \setminus H \not\equiv_T (R \underset{H}{\|} STOP) \setminus H$, as required.

Case 3 There is some $s^\frown\langle A, tock, \bullet\rangle \in \mathcal{T}[\![P \underset{H}{\|} STOP]\!]$ such that $A \cap L \neq \{\}$ and $s^\frown\langle A, tock, \bullet\rangle \notin \mathcal{T}[\![(P \underset{\Sigma}{\|} Q_1) \setminus H]\!]$. Take s minimal, so $s^\frown\langle \bullet\rangle \in \mathcal{T}[\![(P \underset{\Sigma}{\|} Q_1) \setminus H]\!]$.

Then $s[X \cap L/X]^\frown\langle A \cap L, tock, \bullet\rangle \in \mathcal{T}[\![P]\!]$. Let $R \sqsupseteq_T P$ be as in Lemma 3 such that $s[X \cap L/X]^\frown\langle A \cap L, tock, \bullet\rangle$ is "forced". Then for all $a \in A \cap L$, we have $s[X \cap L/X]^\frown\langle \bullet, a, \bullet\rangle \notin \mathcal{T}[\![R]\!]$, so

$$\forall\, a \in A \cap L \bullet s^\frown\langle \bullet, a, \bullet\rangle \notin \mathcal{T}[\![(R \underset{H}{\|} STOP) \setminus H]\!].$$

Now, $s^\frown\langle \bullet\rangle \in \mathcal{T}[\![(P \underset{\Sigma}{\|} Q_1) \setminus H]\!]$, so there is some s' such that $s' \setminus H = s^\frown\langle \bullet\rangle$, s' is H-urgent, and $s' \in \mathcal{T}[\![P \underset{\Sigma}{\|} Q_1]\!]$. Hence $s'[X \cap L/X] \in \mathcal{T}[\![P]\!]$ from equation (1). Hence from Lemma 7, $s'[X \cap L/X] \in \mathcal{T}[\![R]\!]$. Then $s' \in \mathcal{T}[\![R \underset{\Sigma}{\|} Q_1]\!]$,

[7] front s' represents trace s' with the last element removed.

so $s^\frown\langle\bullet\rangle \in \mathcal{T}[\![(R \underset{\Sigma}{\parallel} Q_1) \setminus H]\!]$. But $s^\frown\langle A, tock, \bullet\rangle \notin \mathcal{T}[\![(R \underset{\Sigma}{\parallel} Q_1) \setminus H]\!]$ (because $R \sqsupseteq_T P$). Hence, there is some $a \in A \cap L$ such that $s^\frown\langle\bullet, a, \bullet\rangle \in \mathcal{T}[\![(R \underset{\Sigma}{\parallel} Q_1) \setminus H]\!]$. Then there is some s'' such that $s'' \setminus H = s^\frown\langle\bullet, a, \bullet\rangle$, s'' is H-urgent, and $s'' \in \mathcal{T}[\![R \underset{\Sigma}{\parallel} Q_1]\!]$. Hence $s''[X \cap L/X] \in \mathcal{T}[\![R]\!]$ by equation (1). Let $Q_1' \mathrel{\widehat{=}} Tr(\text{trace}(s'') \upharpoonright H)$, so $s'' \in \mathcal{T}[\![R \underset{H}{\parallel} Q_1']\!]$, and so $s^\frown\langle\bullet, a, \bullet\rangle \in \mathcal{T}[\![(R \underset{H}{\parallel} Q_1') \setminus H]\!]$. But $s^\frown\langle\bullet, a, \bullet\rangle \notin \mathcal{T}[\![(R \underset{H}{\parallel} STOP) \setminus H]\!]$, so $(R \underset{H}{\parallel} Q_1') \setminus H \not\equiv_T (R \underset{H}{\parallel} STOP) \setminus H$, as required. □

B Denotational semantic definitions

We give here the denotational semantic definitions for parallel composition and hiding, which we use elsewhere in the paper.

We begin by considering parallel compositions of the form $P \underset{A}{\parallel} Q$. We assume that A is the intersection of the alphabets of P and Q, as is the case for all parallel compositions we use. For convenience, we define $A_{tock} \mathrel{\widehat{=}} A \cup \{tock\}$. We define an operator that, given refusals X and Y of P and Q, respectively, gives the corresponding refusal for the parallel composition:

$$X \underset{A}{\parallel} Y \;=\; (X \cap^* A) \cup^* (Y \cap^* A) \cup^* (X \cap^* Y).$$

We now define an operator that, given compatible refusal traces s and t of P and Q, respectively, such that $\text{trace}(s) \upharpoonright A_{tock} = \text{trace}(t) \upharpoonright A_{tock}$, gives the corresponding refusal traces for the parallel composition; note that there may be several traces for the parallel composition, corresponding to different ways of interleaving non-synchronisation events.

$$\langle X\rangle \underset{A}{\parallel} \langle Y\rangle = \{\langle X \underset{A}{\parallel} Y\rangle\},$$

$$(\langle X, a\rangle^\frown s) \underset{A}{\parallel} \langle Y\rangle = \{\langle X \underset{A}{\parallel} Y, a)\rangle^\frown u \mid u \in s \underset{A}{\parallel} \langle Y\rangle\}, \quad \text{for } a \notin A_{tock},$$

$$\langle X\rangle \underset{A}{\parallel} (\langle Y, b\rangle^\frown t) = \{\langle X \underset{A}{\parallel} Y, b)\rangle^\frown u \mid u \in \langle X\rangle \underset{A}{\parallel} t\}, \quad \text{for } b \notin A_{tock},$$

$$(\langle X, a\rangle^\frown s) \underset{A}{\parallel} (\langle Y, b\rangle^\frown t) =$$

$$\begin{array}{ll}
\{\langle X \underset{A}{\parallel} Y, a)\rangle^\frown u \mid u \in s \underset{A}{\parallel} t\}, & \text{if } a = b \in A_{tock} \\
\{\langle X \underset{A}{\parallel} Y, a)\rangle^\frown u \mid u \in s \underset{A}{\parallel} (\langle Y, b\rangle^\frown t)\}, & \text{if } a \in A_{tock}, b \notin A_{tock} \\
\{\langle X \underset{A}{\parallel} Y, b)\rangle^\frown u \mid u \in (\langle X, a\rangle^\frown s) \underset{A}{\parallel} t\}, & \text{if } a \notin A_{tock}, b \in A_{tock} \\
\{\langle X \underset{A}{\parallel} Y, a)\rangle^\frown u \mid u \in s \underset{A}{\parallel} (\langle Y, b\rangle^\frown t)\} \cup & \\
\{\langle X \underset{A}{\parallel} Y, b)\rangle^\frown u \mid u \in (\langle X, a\rangle^\frown s) \underset{A}{\parallel} t\}, & \text{if } a \notin A_{tock}, b \notin A_{tock}.
\end{array}$$

The semantics for parallel composition is then defined by combining refusal traces for the two components:

$$\mathcal{R}[\![P \underset{A}{\parallel} Q]\!] \;=\; \{u \mid \exists\, s \in \mathcal{R}[\![P]\!]\,;\, t \in \mathcal{R}[\![Q]\!] \bullet \\ \qquad\qquad \mathrm{trace}(s) \upharpoonright A_{tock} = \mathrm{trace}(t) \upharpoonright A_{tock} \wedge u \in s \underset{A}{\parallel} t\}.$$

We now consider hiding. We first define a hiding operator over refusals. Recall that non-$\bullet$ refusals are recorded only in stable states, where no internal activity is possible. Hence, if we are hiding A, then we can have non-$\bullet$ refusals only when all of A is refused:

$$X \setminus A \;=\; \text{if } A \subseteq X \text{ then } X \text{ else } \bullet$$

We can define a corresponding operator over refusal traces:

$$\begin{aligned} \langle X \rangle \setminus A \;&=\; \langle X \setminus A \rangle \\ (\langle X, a \rangle^\frown s) \setminus A \;&=\; \text{if } a \in A \text{ then } s \setminus A \text{ else } \langle X \setminus A, a \rangle^\frown s \setminus A. \end{aligned}$$

We say that a trace s is A-*urgent* if for every refusal set X that immediately precedes a *tock*, we have $A \subseteq X$. The *maximal progress assumption*—that hidden events occur as soon as they become available—means that we should consider only A-urgent traces of P in the setting $P \setminus A$. We can then define the semantics of hiding as follows:

$$\mathcal{R}[\![P \setminus A]\!] \;=\; \{t \mid \exists\, s \in \mathcal{R}[\![P]\!] \bullet s \text{ is } A\text{-urgent} \wedge t \preceq s \setminus A\}$$