Quantifying Information Flow

Gavin Lowe*

February 5, 2002

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 [1, 14, 4, 12, 7, 5]. 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 [17] 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 quantity* 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 should be

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

aware of, is that an absence of information flow is represented by an information flow quantity 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 [16]. See, for example, [9, 18, 8]. 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 a discrete-time model, taking the time quantum to be one time unit. In principle, one could work in a continuous-time model, such as Timed CSP [15] (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 semantics of discrete-time CSP. In Section 3, we consider some example processes, and for each give the information flow quantity, 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. In Section 5 we show that those processes for which our definition gives an information flow quantity 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 [4, 3]. 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 [13, 15, 10].

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. 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 $a \to P$ 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 $?a : A \to P_a$ offers the set of events A; if a particular event 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 process $WAIT\ t\ ; P$ does nothing for the first t time units, and then acts like P.

The process $P \;\square\; Q$ 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 criteria that we do not model.

The process $P \overset{t}{\rhd} Q$ 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 are *urgent* in the sense that 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. The process $CHAOS(A)$ is the most nondeterministic, nondivergent process with alphabet A; it can perform any sequence of events from A, and refuse any events.

$P \underset{A}{\parallel} Q$ 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*.

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 are available, or the *null refusal* $\bullet$, representing the absence of refusal information, possibly because the process did not enter a stable state. A *refusal trace*[1] is an alternating sequence of refusals and events, starting and finishing with a refusal, for example $\langle \{b\}, a, \bullet, tock, \{a, b\} \rangle$, representing that the process can alternately exhibit the given refusal information and perform the given events.

The *discrete-time refusal traces model* of CSP represents a process P by the set of its refusal traces, denoted $\mathcal{R}[\![P]\!]$. Compositional denotational semantic equations, giving the refusal traces of a compound process in terms of those of its components, can be found in [10].

3 Thought experiments

To help motivate the problem further, we consider some example processes, and in each case 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. We also assume that High and Low can both observe *tock* events.

[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.

Example 1 Consider the process

$$P_1 \mathrel{\widehat{=}} h \to l \to STOP \overset{1}{\rhd} STOP.$$

We assume that Low is not able to detect the precise time at which events occur. Otherwise, processes like the above would give an unbounded amount of information flow: High could pass an 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 becomes available, and so deduce the value High is trying to send; this seems counter-intuitive.

Instead, 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. For notational ease, we take the time quantum equal to one 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*.

Hence the above process has an information flow quantity of 2: if High chooses to perform an h within the first time unit then Low will be able to perform an l; 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 one bit of information to Low.

Example 2 Consider

$$P_2 \mathrel{\widehat{=}} h \to l \to STOP \overset{N}{\rhd} STOP.$$

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 restrict ourselves to information flow of the former kind, by allowing events to occur only within the first time unit, by including an explicit timeout at time 1.

Example 3 Consider:

$$P_3 \mathrel{\widehat{=}} \begin{pmatrix} h1 \to (l1 \to STOP \sqcap l2 \to STOP) \\ \Box\; h2 \to (l1 \to STOP \sqcap l2 \to STOP) \end{pmatrix} \overset{1}{\rhd} STOP.$$

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 is no (and the information flow quantity 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 quantity would be 3.

We indeed take the information flow quantity 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

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

This has an information flow quantity of 3, as does the process $(h1 \to l1 \to STOP \Box h2 \to l2 \to STOP) \overset{1}{\triangleright} STOP$. 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

$$P_5 \mathrel{\widehat{=}} (l \to STOP \Box h \to STOP) \overset{1}{\triangleright} STOP.$$

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.

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, we again consider the worst case scenario, and take the information flow quantity to be 2, for clearly there is the possibility of some flow of information. 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).

4 Defining information flow quantity

In this section we formalise information flow quantity, 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 quantity 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. [2, 11, 15]) make use of a test process $\mathit{SUCCESS}$, which can perform a distinguished event ω, representing the successful completion of a test. We extend this to use test processes of the form $\mathit{SUCCESS}(k)$, for $k \in \mathbb{N}$, that can perform events of the form $\omega.k$ (we take $\omega.k \notin \Sigma$), representing successful completion of the test with result k:

$$\mathit{SUCCESS}(k) \;\widehat{=}\; \omega.k \rightarrow \mathit{STOP}.$$

We represent Low's behaviour by a process T with alphabet $L \cup \{\mathit{tock}\} \cup \{|\omega|\}$; 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:

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

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:

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

(Note, though, that 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 [2, 15], 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 testing equivalence is slightly weaker than semantic equality: it ignores refusals that do not immediately precede *tock*s. Let $\overset{\bullet}{s}$ be obtained from s by replacing all refusals that do not immediately precede a *tock* with $\bullet$:

$$\begin{aligned}
\overbrace{\langle X, a\rangle^\frown s}^{\bullet} &= \langle \bullet, a\rangle^\frown \overset{\bullet}{s}, \qquad a \neq \mathit{tock}\\
\overbrace{\langle X, \mathit{tock}\rangle^\frown s}^{\bullet} &= \langle X, \mathit{tock}\rangle^\frown \overset{\bullet}{s},\\
\overbrace{\langle X\rangle}^{\bullet} &= \langle \bullet \rangle.
\end{aligned}$$

Note that $s \in \mathcal{R}[\![P]\!] \Rightarrow \overset{\bullet}{s} \in \mathcal{R}[\![P]\!]$. The following lemma shows that testing equivalence can distinguish processes only on the basis of refusal tests $\overset{\bullet}{s}$:

Lemma 1 $P \equiv_T Q \Leftrightarrow \{\overset{\bullet}{s} \mid s \in \mathcal{R}[\![P]\!]\} = \{\overset{\bullet}{s} \mid s \in \mathcal{R}[\![Q]\!]\}$.

The proof from right to left shows that if the right hand side holds, then for all T, $(P \underset{L}{\parallel} T) \setminus L$ and $(Q \underset{L}{\parallel} T) \setminus L$ can perform the same events, so $P \equiv_T Q$; the proof from left to right constructs a test T that will distinguish P and Q when the right hand side does not hold. The full proof is in Appendix A.

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 $P \;\widehat{=}\; (a \rightarrow \mathit{STOP} \sqcap \mathit{STOP}) \overset{1}{\rhd} \mathit{STOP}$ and $Q \;\widehat{=}\; (a \rightarrow \mathit{STOP} \;\Box\; c \rightarrow \mathit{STOP}) \setminus \{c\}$. These are not semantically equivalent, because $\langle\{b\}, a, \bullet\rangle \in \mathcal{R}[\![P]\!] - \mathcal{R}[\![Q]\!]$; i.e. P can refuse a b and then perform an a, but Q cannot, because Q performs a in an unstable state. 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; after such a *tock*, the a is withdrawn by both processes.

In untimed models, may-equivalence is the same as traces equivalence [2]; in real-time models, it is the same as finite failures equivalence [15]; it is interesting 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 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.

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:

$$P_5 \;\widehat{=}\; (l \to STOP \;\square\; h \to STOP) \overset{1}{\triangleright} STOP$$

from Example 5. High could pass one value by performing his event h; this is modelled by the process $Q \mathrel{\widehat{=}} h \to STOP$; the combination 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 $Q \mathrel{\widehat{=}} l \to STOP$, 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 $P = l \to STOP$, then the Q process $STOP$ should not be allowed, but $l \to STOP$ 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 $H \subseteq X$; this will be the case for all traces of the process $P \underset{\Sigma}{\parallel} Q$ in the context $(P \underset{\Sigma}{\parallel} Q) \setminus H$.

- We write $s[\langle X \cap L, tock\rangle / \langle X, tock\rangle]$ for the trace that is obtained from s by replacing every refusal set X that immediately precedes a *tock* with $X \cap L$.

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

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

This says that if no H events are blocked by the environment (i.e., 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:

$$\begin{aligned} P \;&\widehat{=}\; h1 \rightarrow (l1 \rightarrow STOP \,\square\, l2 \rightarrow STOP) \\ &\quad\;\square\; h2 \rightarrow (l1 \rightarrow STOP \,\square\, l2 \rightarrow STOP), \\ Q \;&\widehat{=}\; h1 \rightarrow l1 \rightarrow STOP. \end{aligned}$$

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:

$$\begin{aligned} &\forall\, R \sqsupseteq_T P \bullet \forall\, s \in \mathcal{R}[\![R \underset{\Sigma}{\parallel} Q]\!] \bullet \\ &\qquad s \text{ is } H\text{-urgent} \Rightarrow s[\langle X \cap L, tock\rangle/\langle X, tock\rangle] \in \mathcal{R}[\![R]\!]. \end{aligned} \tag{1}$$

The condition is satisfied by the process $RUN(L)$, 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 $(P \underset{\Sigma}{\parallel} Q) \setminus H$ does not make more pairs of processes distinguishable from Low's point of view than by considering compositions of the 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 strengthen 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 quantity

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[2] $\mathcal{Q} : \mathbb{N} \nrightarrow \mathit{CSP}$ 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 $\{(P \parallel_{\Sigma} \mathcal{Q}(k)) \setminus H \mid k \in \operatorname{dom} \mathcal{Q}\}$. 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 $\mathit{results}(S, T)$ 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:

$$\mathit{results}(P, Q, T) \;\widehat{=}\; \mathit{results}((P \parallel_{\Sigma} Q) \setminus H, T).$$

For example, recall

$$P_1 \;\widehat{=}\; h \to l \to \mathit{STOP} \overset{1}{\rhd} \mathit{STOP},$$

and consider the strategy defined by:

$$\mathcal{Q}(0) \,\widehat{=}\, \mathit{RUN}(L), \qquad \mathcal{Q}(1) \,\widehat{=}\, h \to \mathit{RUN}(L),$$
$$T \,\widehat{=}\, l \to \mathit{SUCCESS}(1) \overset{1}{\rhd} \mathit{SUCCESS}(0).$$

Then $(P_1 \parallel_{\Sigma} \mathcal{Q}(0)) \setminus H = \mathit{STOP}$ so we have $\mathit{results}(P_1, \mathcal{Q}(0), T) = \{0\}$; and $(P_1 \parallel_{\Sigma} \mathcal{Q}(1)) \setminus H = l \to \mathit{STOP}$ so we have $\mathit{results}(P_1, \mathcal{Q}(1), T) = \{1\}$.

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 $\mathit{ok}(P, \mathcal{Q}, T)$ to capture this condition:

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

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

Given some process P and some strategy represented by $\mathcal{Q} \in \mathbb{N} \nrightarrow \mathit{CSP}$ and $T \in \mathit{Test}$ such that $\mathit{ok}(P, \mathcal{Q}, T)$, 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 quantity is then the maximum flow achievable over all such R, and over all corresponding $\mathcal{Q}$ and T (such that $\mathit{ok}(R, \mathcal{Q}, T)$):

$$\begin{aligned}\mathit{IFQ}(P) \;\widehat{=}\; \max\{\# \operatorname{dom} \mathcal{Q} \mid\, & \mathcal{Q} \in \mathbb{N} \nrightarrow \mathit{CSP} \wedge T \in \mathit{Test} \wedge R \in \mathit{CSP} \\ & \wedge P \sqsubseteq_T R \wedge \mathit{ok}(R, \mathcal{Q}, T)\}.\end{aligned}$$

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

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

For example, this definition gives an information flow quantity 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.

4.4 Examples

Recall the process

$$P_5 \ \widehat{=}\ (l \to STOP \mathbin{\Box} h \to STOP) \overset{1}{\triangleright} STOP.$$

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

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

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

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

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 after either (1) the h has occurred, after which $\mathcal{Q}(1)$ does not block events of L, or (2) the timeout has occurred, after which P_5 refuses all of L. Hence this process has an information flow quantity of 2.

As another example, consider

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

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

$$\mathcal{Q}(0) \mathrel{\widehat{=}} RUN(L), \qquad \mathcal{Q}(1) \mathrel{\widehat{=}} h1 \to RUN(L), \qquad \mathcal{Q}(2) \mathrel{\widehat{=}} h2 \to RUN(L),$$
$$T \mathrel{\widehat{=}} l \to (l1 \to SUCCESS(1) \mathbin{\Box} l2 \to SUCCESS(2)) \overset{1}{\triangleright} SUCCESS(0).$$

However, no higher information flow quantity 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:

$$\begin{pmatrix} l \to (l1 \to SUCCESS(1) \mathbin{\Box} l2 \to SUCCESS(2)) \\ \Box\ l' \to (l2 \to SUCCESS(2) \mathbin{\Box} l1 \to SUCCESS(1)) \end{pmatrix} \overset{1}{\triangleright} SUCCESS(0).$$

(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 quantity 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

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

$$T \mathrel{\widehat{=}} (l1 \rightarrow SUCCESS(1) \,\Box\, l2 \rightarrow SUCCESS(2)) \overset{1}{\rhd} SUCCESS(0)$$

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 quantity. Let $P \mathrel{\widehat{=}} h \rightarrow l \rightarrow WAIT\,1\,;P \overset{1}{\rhd} STOP$. High can pass an arbitrary value k by performing k h events:

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

Then this strategy, using the test $T(0)$, will pass an arbitrary value, so the information flow quantity 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 quantity of 1—i.e., no information flow.

In [4, 3, 5], 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:

$$P \,\mathbf{sat}\, \mathit{TNDC} \;\widehat{=}\; \forall\, Q \in \mathit{CSP}_H \bullet P \underset{H}{\parallel} \mathit{STOP} \equiv_T (P \underset{H}{\parallel} Q) \setminus H,$$

This condition is equivalent to:

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

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

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

Our main result is that our definition gives information flow quantity 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 [7]. Let LEAK be any insecure process, and consider the two processes $\mathit{LEAK} \sqcap \mathit{CHAOS}(L)$ and $\mathit{LEAK} \mathbin{|||} \mathit{CHAOS}(L)$. 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 $\mathit{CHAOS}(L)$. These processes do not satisfy STNDC, because they are refined by LEAK.

In [6], 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.

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 quantity of 2.

Lemma 3 If $(P \underset{H}{\parallel} Q_0) \setminus H \not\equiv_T (P \underset{H}{\parallel} Q_1) \setminus H$ then there exist $R \sqsupseteq_T P$, $Q_0', Q_1' \in \mathit{CSP}$ satisfying equation (1), and $T \in \mathit{Test}$ such that

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

The proof makes use of the test T from Lemma 1, which distinguishes processes according to whether or not they have a particular refusal trace s; the proof constructs a suitable refinement of P, and suitable high level processes, such that one of the resulting systems will always exhibit the trace s, and the other system will never exhibit this trace. The full proof is in the appendix.

The following theorem identifies the circumstances under which our definition identifies no information flow, i.e. gives an information flow quantity of 1:

Theorem 4 A process P has information flow quantity 1 iff for all refinements R, High cannot change the way the process appears to Low:

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

The above theorem is equivalent to

$$IFQ(P) = 1 \Leftrightarrow P \textbf{ sat } STNDC.$$

Proof: We prove the contrapositive. In one direction:

$$\begin{array}{ll}
 & \neg(\forall R \sqsupseteq_T P \,;\, Q_0, Q_1 \in CSP_H \bullet (R \underset{H}{\parallel} Q_0) \setminus H \equiv_T (R \underset{H}{\parallel} Q_1) \setminus H) \\
\Leftrightarrow & \exists R \sqsupseteq_T P \,;\, Q_0, Q_1 \in CSP_H \bullet (R \underset{H}{\parallel} Q_0) \setminus H \not\equiv_T (R \underset{H}{\parallel} Q_1) \setminus H \\
\Rightarrow & \langle \text{Lemma 3} \rangle \\
 & \exists R \sqsupseteq_T P \bullet \exists R' \sqsupseteq_T R \,;\, Q_0', Q_1' \in CSP \,;\, T \in Test \bullet \\
 & \quad results(R', Q_0', T) = \{0\} \wedge results(R', Q_1', T) = \{1\} \wedge Q_0',\ Q_1' \text{ satisfy (1)} \\
\Rightarrow & IFQ(P) \geqslant 2.
\end{array}$$

In the other direction:

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

□

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 section 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.

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

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

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

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 $P \mathrel{\widehat{=}} h \to l \to \mathit{STOP}$. Fix N, and consider the strategy

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

Then $\mathit{results}(P, \mathcal{Q}(k), T(0)) = \{k\}$, for $k = 0, \dots, N$, and all of the observations contain at most N *tock*s. 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 $\mathit{IFQ}_N(P) = N+1$, for all N, so the long term information flow rate is zero.

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:

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

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 $Q(k) \mathbin{|||} RUN(L)$ where:

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

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 quantity 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.

We consider now a few possible extensions to this work.

It would clearly be useful to have an automated procedure for calculating the quantity 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 quantity of information flow in a given system: proving lower bounds can be done simply be exhibiting a suitable strategy, 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 *tock*s—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 and Philippa Broadfoot for useful discussions about information flow. This work was partially supported by a grant from the US Office of Naval Research.

References

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

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

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

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

[5] 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.

[6] Riccardo Focardi and Sabina Rossi. A security property for processes in dynamic contexts. In *Proceedings of the Workshop on Issues in the Theory of Security (WITS '02)*, 2002.

[7] 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`.

[8] 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.

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

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

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

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

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

[14] 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.

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

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

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

[18] 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

A.1 Proofs about testing

We begin by proving some results about our testing scheme. First, we show that refining a process reduces the set of results obtained from a given test.

Lemma 5 If $P \sqsubseteq_T Q$ then $\forall\, T \bullet \mathit{results}(P, T) \supseteq \mathit{results}(Q, T)$.

Proof: Suppose $P \sqsubseteq_T Q$, and let T be a test. Then

$$\begin{aligned} \mathit{results}(P, T) &= \{k \mid \exists\, n \in \mathbb{N} \bullet \langle \bullet, \mathit{tock}\rangle^n \frown \langle \bullet, \omega.k, \bullet\rangle \in \mathcal{R}[\![(P \underset{L}{\parallel} T) \setminus L]\!]\} \\ &\supseteq \{k \mid \exists\, n \in \mathbb{N} \bullet \langle \bullet, \mathit{tock}\rangle^n \frown \langle \bullet, \omega.k, \bullet\rangle \in \mathcal{R}[\![(Q \underset{L}{\parallel} T) \setminus L]\!]\} \\ &= \mathit{results}(Q, T). \end{aligned}$$

□

We now prove Lemma 1:

$$P \equiv_T Q \;\Leftrightarrow\; \{\overset{\bullet}{s} \mid s \in \mathcal{R}[\![P]\!]\} = \{\overset{\bullet}{s} \mid s \in \mathcal{R}[\![Q]\!]\}.$$

Proof of Lemma 1: For the right-to-left implication, we begin by noting:

1. If $\overset{\bullet}{s}_P = \overset{\bullet}{s}_Q$ then $\overset{\bullet}{\overbrace{(s_P \underset{L}{\parallel} t)}} = \overset{\bullet}{\overbrace{(s_Q \underset{L}{\parallel} t)}}$;

2. If $\overset{\bullet}{s}_P = \overset{\bullet}{s}_Q$ and s_P is L-urgent then s_Q is L-urgent;

3. If $\overset{\bullet}{s}_P = \overset{\bullet}{s}_Q$ then $\overbrace{s_P \setminus L}^{\bullet} = \overbrace{s_Q \setminus L}^{\bullet}$.

Suppose $\{\overset{\bullet}{s} \mid s \in \mathcal{R}[\![P]\!]\} = \{\overset{\bullet}{s} \mid s \in \mathcal{R}[\![Q]\!]\}$ and pick $T \in \mathit{Test}$. Then

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

and vice versa. Hence

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

and hence $\mathit{results}(P, T) = \mathit{results}(Q, T)$ for all T, so $P \equiv_T Q$.

For the converse, we prove the contrapositive. Suppose $P \not\equiv_T Q$. Then without loss of generality, there is some s such that $s \in \mathcal{R}[\![P]\!]$ but $\overset{\bullet}{s} \notin \mathcal{R}[\![Q]\!]$. Let

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

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

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

We construct a test that will succeed with value 1 after the trace $\overset{\bullet}{s}$, but will succeed with value 0, otherwise:

$$
\begin{array}{rl}
T \mathrel{\widehat{=}} & a_{11} \xrightarrow{*} a_{12} \xrightarrow{*} \ldots \xrightarrow{*} a_{1m_1} \xrightarrow{*} (?x : A_1 \rightarrow \mathit{SUCCESS}(0) \\
& \overset{1}{\rhd} \\
& a_{21} \xrightarrow{*} a_{22} \xrightarrow{*} \ldots \xrightarrow{*} a_{2m_2} \xrightarrow{*} (?x : A_2 \rightarrow \mathit{SUCCESS}(0) \\
& \overset{1}{\rhd} \\
& \ldots \\
& \overset{1}{\rhd} \\
& a_{n1} \xrightarrow{*} a_{n2} \xrightarrow{*} \ldots \xrightarrow{*} a_{nm_n} \xrightarrow{*} \mathit{SUCCESS}(1)) \ldots)
\end{array}
$$

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

A.2 Forcing refusal traces

We now prove a technical lemma that 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 6 Let $s \in \mathcal{R}[\![P]\!]$. Then we can construct 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.

Proof: Let $\bar{s}$ be obtained from s by replacing each refusal set X such that $s'^\frown\langle X\rangle \leqslant s$ by $\bar{X} \mathrel{\widehat{=}} X \cup^* \{a \mid s'^\frown\langle\bullet, a, \bullet\rangle \notin \mathcal{R}[\![P]\!]\}$. Let P after s' represent the behaviour of P following trace s':

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

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

Define $R \mathrel{\widehat{=}} R_P(\bar{s})$ where:

$$
\begin{array}{rcl}
R_Q(\langle X\rangle) & \mathrel{\widehat{=}} & ?x : \Sigma - X \rightarrow Q \text{ after } \langle X, x\rangle \overset{1}{\rhd} Q \text{ after } \langle X, \mathit{tock}\rangle \\
R_Q(\langle X, a\rangle^\frown s) & \mathrel{\widehat{=}} & ?x : \Sigma - X \rightarrow \\
& & \qquad \text{if } x = a \text{ then } R_{Q\text{after}\langle X, a\rangle}(s) \text{ else } Q \text{ after } \langle X, x\rangle \\
& & \overset{1}{\rhd} Q \text{ after } \langle X, \mathit{tock}\rangle \\
R_Q(\langle X, \mathit{tock}\rangle^\frown s) & \mathrel{\widehat{=}} & ?x : \Sigma - X \rightarrow Q \text{ after } \langle X, x\rangle \overset{1}{\rhd} R_{Q\text{after}\langle X, \mathit{tock}\rangle}(s).
\end{array}
$$

Then $P \sqsubseteq_T R$ (it is straightforward to prove by induction that $Q \sqsubseteq_T R_Q(s')$ provided s' is maximal, $s' \in \mathcal{R}[\![Q]\!]$). Further, by construction, R cannot refuse any events of s, and cannot perform any of the events of refusal sets in s. $\square$

Note that the above lemma cannot be strengthened to talk about refinement in the standard refusal-testing model. Let $P = (a \to STOP \,\Box\, c \to b \to STOP) \setminus \{c\}$, and let $s = \langle \bullet, a, \bullet \rangle$. Suppose R satisfies the conditions of the lemma. Then with axiom R3, $\langle \{\} \rangle \in \mathcal{R}[\![R]\!]$, so $\langle \{\}, a, \bullet \rangle \in \mathcal{R}[\![R]\!]$ or $\langle \{a\} \rangle \in R$, giving a contradiction in each case. The above construction would give $R = (a \to STOP \,\Box\, b \to STOP) \overset{1}{\rhd} b \to STOP)$, which is not a R-refinement, but is a T-refinement.

Note further that if $s' \in \mathcal{R}[\![P]\!]$ and $s = \overset{\bullet}{s}$ then $\overset{\bullet}{s'} \in \mathcal{R}[\![R]\!]$ unless one of the following holds:

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

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

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

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

The following lemma will be useful later.

Lemma 7 Suppose $(s' \setminus H) = \overset{\bullet}{s}{}^\frown\langle\bullet\rangle$, $s'[\langle X \cap L, tock\rangle/\langle X, tock\rangle] \in \mathcal{R}[\![P]\!]$, $\overset{\bullet}{s}[X \cap L/X]^\frown s'' \in \mathcal{R}[\![P]\!]$, for some s'', and let $R \sqsupseteq_T P$ be defined as in Lemma 6 to "force" $\overset{\bullet}{s}[X \cap L/X]^\frown s''$. Then

$$\overbrace{s'[\langle X \cap L, tock\rangle/\langle X, tock\rangle]}^{\bullet} = \overset{\bullet}{s'}[\langle X \cap L, tock\rangle/\langle X, tock\rangle] \in \mathcal{R}[\![R]\!].$$

Proof: Suppose not; then by the above observation, we have one of the following:

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

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

Then $a \in H$ (because $s' \setminus H = \overset{\bullet}{s}{}^\frown\langle\bullet\rangle$ and $a \neq tock$). But $a \in A \subseteq L$, giving a contradiction.

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

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

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

$\Box$

A.3 Lemma 2

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:

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

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

  Then $\dot{s}[X \cap L/X]^\frown\langle\bullet, a, \bullet\rangle \in \mathcal{R}[\![P]\!]$. Let $R \sqsupseteq_T P$ be as in Lemma 6 such that $\dot{s}[X \cap L/X]^\frown\langle\bullet, a, \bullet\rangle$ is "forced". Then $\dot{s}[X \cap L/X]^\frown\langle\{a\}, tock, \bullet\rangle \notin \mathcal{R}[\![R]\!]$ so

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

  Now, $\dot{s}^\frown\langle\bullet\rangle \in \mathcal{R}[\![(P \underset{\Sigma}{\parallel} Q_1) \setminus H]\!]$, so there is some s' such that $s' \setminus H = \dot{s}^\frown\langle\bullet\rangle$, s' is H-urgent, and $s' \in \mathcal{R}[\![P \underset{\Sigma}{\parallel} Q_1]\!]$. Hence $s'[\langle X \cap$

$L, tock\rangle/\langle X, tock\rangle] \in \mathcal{R}\llbracket P\rrbracket$ by equation (1). Then from Lemma 7 we have $\overset{\bullet}{s'}[\langle X \cap L, tock\rangle/\langle X, tock\rangle] \in \mathcal{R}\llbracket R\rrbracket$.

Let $Q_1' \mathrel{\widehat{=}} \mathit{Tr}(\mathrm{trace}(s') \upharpoonright H)$; then $\overset{\bullet}{s'} \in \mathcal{R}\llbracket R \mathbin{\underset{H}{\parallel}} Q_1'\rrbracket$, and hence $\overset{\bullet}{s}{}^\frown\langle\bullet\rangle \in \mathcal{R}\llbracket(R \mathbin{\underset{H}{\parallel}} Q_1') \setminus H\rrbracket$. Now, $\overset{\bullet}{s}{}^\frown\langle\bullet, a, \bullet\rangle \notin \mathcal{R}\llbracket(P \mathbin{\underset{\Sigma}{\parallel}} Q_1) \setminus H\rrbracket$, so $\mathrm{init}\, s'{}^\frown\langle\bullet, a, \bullet\rangle \notin \mathcal{R}\llbracket P \mathbin{\underset{\Sigma}{\parallel}} Q_1\rrbracket$ (init s' represents trace s' with the last element removed) so $\mathrm{init}\, s'{}^\frown\langle\bullet, a, \bullet\rangle \notin \mathcal{R}\llbracket R \mathbin{\underset{\Sigma}{\parallel}} Q_1\rrbracket$ because $R \sqsupseteq_T P$. Hence $\mathrm{init}\, s'{}^\frown\langle\{a\}, tock, \bullet\rangle \in \mathcal{R}\llbracket R \mathbin{\underset{\Sigma}{\parallel}} Q_1\rrbracket$ so $\mathrm{init}\, s'[\langle X \cap L, tock\rangle/\langle X, tock\rangle]^\frown\langle\{a\}, tock, \bullet\rangle \in \mathcal{R}\llbracket R\rrbracket$ by equation (1). Hence $\overset{\bullet}{s}{}^\frown\langle\{a\}, tock, \bullet\rangle \in \mathcal{R}\llbracket(R \mathbin{\underset{H}{\parallel}} Q_1') \setminus H\rrbracket$. Hence $(R \mathbin{\underset{H}{\parallel}} Q_1')\setminus H \not\equiv_T (R \mathbin{\underset{H}{\parallel}} \mathit{STOP}) \setminus H$, as required.

- Case there is some $s^\frown\langle A, tock, \bullet\rangle \in \mathcal{R}\llbracket P \mathbin{\underset{H}{\parallel}} \mathit{STOP}\rrbracket$ such that $A \cap L \neq \{\}$ and $\overset{\bullet}{s}{}^\frown\langle A, tock, \bullet\rangle \notin \mathcal{R}\llbracket(P \mathbin{\underset{\Sigma}{\parallel}} Q_1) \setminus H\rrbracket$. Take s minimal, so $\overset{\bullet}{s}{}^\frown\langle\bullet\rangle \in \mathcal{R}\llbracket(P \mathbin{\underset{\Sigma}{\parallel}} Q_1) \setminus H\rrbracket$.

  Then $\overset{\bullet}{s}[X \cap L, X]^\frown\langle A \cap L, tock, \bullet\rangle \in \mathcal{R}\llbracket P\rrbracket$. Let $R \sqsupseteq_T P$ be as in Lemma 6 such that $\overset{\bullet}{s}[X \cap L/X]^\frown\langle A \cap L, tock, \bullet\rangle$ is "forced". Then for all $a \in A \cap L$, we have $\overset{\bullet}{s}[X \cap L/X]^\frown\langle\bullet, a, \bullet\rangle \notin \mathcal{R}\llbracket R\rrbracket$, so

  $$\forall\, a \in A \cap L \bullet \overset{\bullet}{s}{}^\frown\langle\bullet, a, \bullet\rangle \notin \mathcal{R}\llbracket(R \mathbin{\underset{H}{\parallel}} \mathit{STOP}) \setminus H\rrbracket.$$

  Now, $\overset{\bullet}{s}{}^\frown\langle\bullet\rangle \in \mathcal{R}\llbracket(P \mathbin{\underset{\Sigma}{\parallel}} Q_1) \setminus H\rrbracket$, so there is some s' such that $s' \setminus H = \overset{\bullet}{s}{}^\frown\langle\bullet\rangle$, s' is H-urgent, and $s' \in \mathcal{R}\llbracket P \mathbin{\underset{\Sigma}{\parallel}} Q_1\rrbracket$. Hence $s'[\langle X \cap L, tock\rangle/\langle X, tock\rangle] \in \mathcal{R}\llbracket P\rrbracket$ from equation (1). Hence from Lemma 7 $\overset{\bullet}{s'}[\langle X \cap L, tock\rangle/\langle X, tock\rangle] \in \mathcal{R}\llbracket R\rrbracket$. Then $\overset{\bullet}{s'} \in \mathcal{R}\llbracket R \mathbin{\underset{\Sigma}{\parallel}} Q_1\rrbracket$, so $\overset{\bullet}{s}{}^\frown\langle\bullet\rangle \in \mathcal{R}\llbracket(R \mathbin{\underset{\Sigma}{\parallel}} Q_1) \setminus H\rrbracket$. But $\overset{\bullet}{s}{}^\frown\langle A, tock, \bullet\rangle \notin \mathcal{R}\llbracket(R \mathbin{\underset{\Sigma}{\parallel}} Q_1) \setminus H\rrbracket$ (because $R \sqsupseteq_T P$). Hence, there is some $a \in A \cap L$ such that $\overset{\bullet}{s}{}^\frown\langle\bullet, a, \bullet\rangle \in \mathcal{R}\llbracket(R \mathbin{\underset{\Sigma}{\parallel}} Q_1) \setminus H\rrbracket$. Then there is some s'' such that s'' is H-urgent, and $s'' \in \mathcal{R}\llbracket R \mathbin{\underset{\Sigma}{\parallel}} Q_1\rrbracket$. Hence $s''[\langle X \cap L, tock\rangle/\langle X, tock\rangle] \in \mathcal{R}\llbracket R\rrbracket$ by equation (1). Let $Q_1' \mathrel{\widehat{=}} \mathit{Tr}(\mathrm{trace}(s'') \upharpoonright H)$, so $s'' \in \mathcal{R}\llbracket R \mathbin{\underset{H}{\parallel}} Q_1'\rrbracket$, and so $\overset{\bullet}{s}{}^\frown\langle\bullet, a, \bullet\rangle \in \mathcal{R}\llbracket(R \mathbin{\underset{H}{\parallel}} Q_1') \setminus H\rrbracket$. But $\overset{\bullet}{s}{}^\frown\langle\bullet, a, \bullet\rangle \notin \mathcal{R}\llbracket(R \mathbin{\underset{H}{\parallel}} \mathit{STOP}) \setminus H\rrbracket$, so $(R \mathbin{\underset{H}{\parallel}} Q_1') \setminus H \not\equiv_T (R \mathbin{\underset{H}{\parallel}} \mathit{STOP}) \setminus H$, as required.

$\square$

A.4 Lemma 3

We now prove Lemma 3, 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 quantity of 2:

> If $(P \underset{H}{\|} Q_0) \setminus H \not\equiv_T (P \underset{H}{\|} Q_1) \setminus H$ then there exist $R \sqsupseteq_T P$, $Q_0', Q_1' \in \mathit{CSP}$ satisfying equation (1), and $T \in \mathit{Test}$ such that
>
> $$\mathit{results}(R, Q_0', T) = \{0\} \quad \text{and} \quad \mathit{results}(R, Q_1', T) = \{1\}.$$

Proof of Lemma 3: Following Lemma 1, suppose

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

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

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

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

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

Let $R \sqsupseteq_T P$ be—as in Lemma 6—such that s' is "forced". Let $Q_1' \mathrel{\widehat{=}} Q(\dot{s}')$ where

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

Consider $(R \underset{\Sigma}{\|} Q_1') \setminus H$ 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 $X \cap L$, Q_1' prevents all events from H so $R \underset{\Sigma}{\|} Q_1'$ refuses $X \cup H$, so $(R \underset{\Sigma}{\|} Q_1') \setminus H$ refuses X, so the test selects the *tock*.

Further, R cannot refuse any of these events. Hence

$$results(R, Q'_1, T) \ = \ \{1\}.$$

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 $Q'_0 \mathrel{\widehat{=}} Q_0 \ ||| \ RUN(L)$. 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

$$results(R, Q'_0, T) = results((R \underset{\Sigma}{\parallel} Q'_0) \setminus H, T) \subseteq results(P \underset{H}{\parallel} Q_0) \setminus H, T) = \{0\}$$

by Lemma 5. But, by construction, T always gives at least one result, so

$$results(R, Q'_0, T) = \{0\}.$$

$\square$