Quantifying Information Flow

Gavin Lowe*

February 3, 2001

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, for example [1, 12, 4, 10, 6, 5], 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. 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 [15] 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. Information flow is often measured in bits: we can easily calculate this as $\log_2 N$. (A disadvantage of this definition, that the reader should be aware of, is that an absence of information flow is represented by an information flow quantity of 1, rather than 0.)

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

We will define the quantity of information flow in both untimed and timed 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 consider some example processes, and for each give the information flow quantity, thus helping to describe our intuitions. In Section 3, 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. 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 4, we extend our definition to timed processes. In Section 5 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 6. A brief overview of CSP and Timed CSP can be found in Appendix A.

2 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.

Example 1 Consider

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

This has an information flow quantity of 2: if High chooses to perform an h then Low will be able to perform an l; if High chooses not to perform an h then Low will not be able to perform an l. Thus High can use P_1 to pass one bit of information to Low. □

Example 2 Let h and l be channels carrying messages from the set $\{1, \ldots, N\}$, and consider

$$P_2 \mathrel{\widehat{=}} h?x \to l!x \to STOP.$$

This has an information flow quantity of $N + 1$, corresponding to the N possible values that could be sent, plus the possibility of High performing no event so that Low receives no value. □

Example 3 Consider:

$$\begin{aligned} P_3 \mathrel{\widehat{=}}\ & h1 \rightarrow (l1 \rightarrow STOP \sqcap l2 \rightarrow STOP) \\ & \Box \\ & h2 \rightarrow (l1 \rightarrow STOP \sqcap l2 \rightarrow STOP). \end{aligned}$$

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. $\Box$

Example 4 Let

$$\begin{aligned} P_4 \mathrel{\widehat{=}}\ & h1 \rightarrow l1 \rightarrow STOP \\ & \Box\, h2 \rightarrow l2 \rightarrow STOP \\ & \Box\, h3 \rightarrow (l1 \rightarrow STOP \sqcap l2 \rightarrow STOP) \end{aligned} \quad .$$

This has an information flow quantity of 3, as with the process $h1 \rightarrow l1 \rightarrow STOP \Box h2 \rightarrow l2 \rightarrow STOP$. 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 here 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 $h3$ has indeed been performed. $\Box$

Example 5 Let

$$P_5 \mathrel{\widehat{=}} h1 \rightarrow l1 \rightarrow STOP \Box h2 \rightarrow (l1 \rightarrow STOP \Box l2 \rightarrow STOP).$$

Depending upon how High acts, from Low's point of view the system will appear to act like one of the processes:

$$STOP, \qquad l1 \rightarrow STOP, \qquad l1 \rightarrow STOP \,\square\, l2 \rightarrow STOP.$$

Our model will say that Low is not able to distinguish between all three of these processes (although he can distinguish between any pair of them) and so the information flow quantity is two. For example, if Low tries to perform $l2$, and it is refused, then he cannot tell whether he is looking at the first or second process. Alternatively, if Low manages to perform $l1$, he cannot tell whether he is looking at the second or third process.

One experiment that Low might—depending on the assumptions one makes—use to distinguish the three processes is to attempt the event $l2$, and if that is refused to then attempt the event $l1$. However, the normal assumption in the Failures Model of CSP is that one cannot tell in a finite amount of time whether an event, such as the $l2$ above, will become available sometime in the future, or equivalently that one cannot observe a refusal in a finite amount of time: this means that one cannot observe a refusal, and subsequently observe an event. Thus we will not allow the above experiment.

The above assumption is far from clear cut. If Low is able to detect that internal activity by the process has ceased (so the events available will not change), then it is reasonable to suppose that Low can detect a refusal and then perform further events. In this case, the information flow quantity for the above process will be 3, and the model we present in this paper will not be suitable. This latter philosophy is that of the Refusal Testing Model [7]; we briefly consider a model of information flow based on refusal testing in the conclusion. It is also the philosophy of the Timed Failures Model of CSP, and so the timed model we present in Section 4 would allow an experiment similar to the one above. □

Example 6 Consider

$$P_6 \mathrel{\widehat{=}} l1 \rightarrow h \rightarrow l3 \rightarrow STOP \,\square\, h \rightarrow l2 \rightarrow STOP.$$

The information flow quantity for this process depends upon the assumptions we make about High: we will assume that High can choose whether to perform his event before or after Low has performed $l1$, and so the quantity is 3, corresponding to the cases where High performs h before $l1$, after $l1$, or not at all. In effect, we allow High to observe Low's event. □

Example 7 Consider

$$P_7 \mathrel{\widehat{=}} h \rightarrow l \rightarrow l \rightarrow STOP \,\square\, l \rightarrow STOP.$$

If Low is able to perform two l events, then he can deduce that High performed h; however, if Low is able to perform only one l, 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 they cannot reliably do so. 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.

An alternative way of considering this process is that High might be able to ensure that his h is performed before Low's first l; for example, they might have agreed a strategy that High does his event at time 0, and Low attempts his event at time 1; alternatively, even if they both attempt their event at the same time, some property of the system, below the level of abstraction at which we are modelling it, might resolve the resulting nondeterminism in favour of High, i.e. give priority to High.

In effect, we allow High to have some limited control over events available to Low, preventing them from happening by preempting them with his own events. □

3 Quantifying information flow in an untimed model

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 divergence-free CSP process P.

We will model High's behaviour by a non-divergent process Q. A natural way of modelling High would be to take the alphabet of Q to be H, in which 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 2, especially those in Examples 6 and 7. 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 use a modelling trick: we include Low's events within the alphabet of the process Q. Low's view of the system will then be given by the process:

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

However, we do not want to allow all such processes Q; in a state of P where only events of Low are available, High should not be able to block any of those 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. We therefore restrict ourselves to processes Q satisfying the following condition for the given P:

$$\forall (s, X) \bullet (s, X \cup H) \in \mathcal{F}[\![P \underset{\Sigma}{\|} Q]\!] \Rightarrow (s, X \cap L) \in \mathcal{F}[\![P]\!].$$

This says that in every state of the system where no events from H are available, if any events from L are refused, it is because P is refusing those events.

This condition prevents examples like the following:

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

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: $P \underset{\Sigma}{\|} Q$ has failure $(\langle h1 \rangle, \{l2\} \cup H)$, but P does not have failure $(\langle h1 \rangle, \{l2\})$.

In fact, as explained in Example 3, we have to consider all ways in which nondeterminism in P might be resolved. We will do this 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:

$$\forall R \sqsupseteq P \bullet \forall (s, X) \bullet (s, X \cup H) \in \mathcal{F}[\![R \underset{\Sigma}{\|} Q]\!] \Rightarrow (s, X \cap L) \in \mathcal{F}[\![R]\!]. \qquad (1)$$

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

For simplicity, we restrict ourselves to processes P where $(P \underset{\Sigma}{\|} Q) \setminus H$ is non-divergent for all non-divergent Q; this will be true if P itself is non-divergent and cannot perform an unbounded number of consecutive high level events.

The following lemma shows that under the above conditions, considering compositions of the form $(P \underset{\Sigma}{\|} Q) \setminus H$ does not make more pairs of processes distinguishable from Low's point of view than by considering compositions of the form $(R \underset{H}{\|} Q') \setminus H$.

Lemma 1 If Q_0 and Q_1 satisfy equation (1), and

$$(P \underset{\Sigma}{\|} Q_0) \setminus H \not\equiv_F (P \underset{\Sigma}{\|} Q_1) \setminus H$$

then

$$\exists R \sqsupseteq P\,;\, Q_0', Q_1' \in CSP_H \bullet (R \underset{H}{\|} Q_0') \setminus H \not\equiv_F (R \underset{H}{\|} Q_1') \setminus H.$$

where CSP_H represents the set of CSP processes with alphabet 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 Section 3.2.

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 $\mathcal{Q}(k)$; we will therefore model High's behaviour as a function of the form:[1]

$$\mathcal{Q} : \mathbb{N} \nrightarrow CSP_H \qquad \text{where every } \mathcal{Q}(k) \text{ satisfies equation (1).}$$

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.

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, in particular that Low cannot offer further events after observing a refusal.

Traditional approaches to testing (e.g. [14]) make use of a test process $SUCCESS$, which represents successful completion of a test. We extend this to use test processes of the form "$SUCCESS(k)$", for $k \in \mathbb{N}$, to represent successful completion of the test, giving a result of k.

We also write

$$(\square_{a \in A}\, a \rightarrow T_a) \ \mathsf{else}\ SUCCESS(k)$$

to represent a test that attempts the set of event A (where $A \subseteq L$), and which then acts like T_a if a particular event a is performed, and which gives the result k if all the events are refused.

We can therefore define failures-based tests using a BNF-style grammar as follows:

$$T \in \mathit{Test} ::= SUCCESS(k) \mid (\square_{a \in A}\, a \rightarrow T_a) \ \mathsf{else}\ SUCCESS(k).$$

We consider only *finite* tests, i.e. where the test bottoms out to give a result after a bounded number of steps.

We will write the indexed external choice as a binary infix external choice, where convenient; for example, the test

$$T \ \widehat{=}\ l1 \rightarrow SUCCESS(1) \,\square\, l2 \rightarrow SUCCESS(2) \ \mathsf{else}\ SUCCESS(0)$$

represents Low offering $l1$ and $l2$, and giving the result 1 if $l1$ is accepted, the result 2 if $l2$ is accepted, and the result 0 if both are refused.

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

We associate a set of (trace, refusal, result) triples with each test, representing that the test can observe the given trace and refusal of the process it is testing, and then return the given result.

$$\begin{array}{l}
\mathcal{F}[\![SUCCESS(k)]\!] \mathrel{\widehat{=}} \{(\langle\rangle, \{\}, k)\}, \\
\mathcal{F}[\![(\Box_{a \in A} \, a \to T_a) \text{ else } SUCCESS(k)]\!] \mathrel{\widehat{=}} \\
\quad \{(\langle a \rangle ^\frown s, X, i) \mid a \in A \wedge (s, X, i) \in \mathcal{F}[\![T_a]\!]\} \cup \{(\langle\rangle, A, k)\}.
\end{array}$$

The test $SUCCESS(k)$ can immediately give the result k (following no trace or refusal information). The test $(\Box_{a \in A} \, a \to T_a)$ else $SUCCESS(k)$ can see an event $a \in A$ performed and then act like the test T_a; alternatively, it can see all of A refused and then give the result k.

We will write $results_F(S, T)$ for the set of results obtainable from failures-based test T with system S:

$$results_F(S, T) \mathrel{\widehat{=}} \{k \mid \exists (s, X) \in \mathcal{F}[\![S]\!] \bullet (s, X, k) \in \mathcal{F}[\![T]\!]\}.$$

The testing system can give a result k if S can display some failure (s, X) such that the test associates result k with this failure. The subscript "F" indicates that we are working in the Failures Model of CSP; we will drop the subscript where it is obvious from the context.

Note that if S is non-divergent—as we are assuming—then $results_F(S, T)$ will contain at least one value: for each state $(\Box_{a \in A} \, a \to T_a)$ else $SUCCESS(k)$ of T, the corresponding state of S will either be able to perform one of the events from A or refuse all of A; and we are considering finite tests so the testing system will eventually bottom out to give a result.

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:

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

For example, recall

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

and consider the strategy defined by:

$$\begin{array}{c}
\mathcal{Q}(0) \mathrel{\widehat{=}} RUN(L), \qquad \mathcal{Q}(1) \mathrel{\widehat{=}} h \to RUN(L), \\
T \mathrel{\widehat{=}} l \to SUCCESS(1) \text{ else } SUCCESS(0).
\end{array}$$

Then

$$results_F(P_1, \mathcal{Q}(0), T) = \{0\} \qquad \text{and} \qquad results_F(P_1, \mathcal{Q}(1), T) = \{1\},$$

corresponding, respectively, to the failure $(\langle\rangle, \{l\})$ of $(P_1 \underset{\Sigma}{\parallel} \mathcal{Q}(0)) \setminus H$, and the failure $(\langle l\rangle, \{\})$ of $(P_1 \underset{\Sigma}{\parallel} \mathcal{Q}(1)) \setminus H$.

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 can obtain the result k. We will write $ok_F(P, \mathcal{Q}, T)$ to capture this condition:

$$
\begin{aligned}
ok_F(P, \mathcal{Q}, T) \;\widehat{=}\; \forall\, k \in \operatorname{dom} \mathcal{Q} \bullet\ & results_F(P, \mathcal{Q}(k), T) = \{k\} \\
& \text{and } \mathcal{Q}(k) \text{ satisfies equation (1).}
\end{aligned}
$$

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

Note that we insist that the results caused by a particular $\mathcal{Q}(k)$ are precisely $\{k\}$: it is not enough simply for different $\mathcal{Q}(k)$ processes to lead to different sets of results. For example, the process and test

$$
\begin{aligned}
P \;&\widehat{=}\; h1 \to l1 \to STOP \\
&\quad \Box\ h2 \to l2 \to STOP \\
&\quad \Box\ h3 \to (l1 \to STOP \ \Box\ l2 \to STOP), \\
T \;&\widehat{=}\; l1 \to SUCCESS(1) \ \Box\ l2 \to SUCCESS(2) \ \mathsf{else}\ STOP
\end{aligned}
$$

give different sets of results for the four high level processes $\mathcal{Q}(0) \,\widehat{=}\, RUN(L)$, $\mathcal{Q}(1) \,\widehat{=}\, h1 \to RUN(L)$, $\mathcal{Q}(2) \,\widehat{=}\, h2 \to RUN(L)$ and $\mathcal{Q}(3) \,\widehat{=}\, h3 \to RUN(L)$; 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.

Given some process P and some strategy represented by $\mathcal{Q} \in \mathbb{N} \nrightarrow CSP$ and $T \in Test$ such that $ok_F(P, \mathcal{Q}, T)$, 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 2, 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, i.e., such that $ok_F(R, \mathcal{Q}, T)$. This prompts the following definition of information flow quantity:

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

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

3.1 Examples

Recall the process:

$$P_7 \;\widehat{=}\; h \to l \to l \to STOP \ \Box\ l \to STOP.$$

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

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

Note that $\mathcal{Q}(1)$ prevents an initial l from occurring by preempting it with h. (The final $SUCCESS(0)$ in T is unreachable, but is syntactically necessary.) With this strategy:

$$results_F(P_7, \mathcal{Q}(0), T) = \{0\} \quad \text{and} \quad results_F(P_7, \mathcal{Q}(1), T) = \{1\}.$$

Note also that the $\mathcal{Q}(k)$ processes satisfy equation (1): in particular, $P_7 \underset{\Sigma}{\|} \mathcal{Q}(1)$ cannot refuse H until after the h has occurred, after which $\mathcal{Q}(1)$ does not block events of L. Hence this process has an information flow quantity of 2.

As another example, consider

$$\begin{array}{l} h1 \to (l \to l1 \to STOP \,\Box\, l' \to l1 \to STOP) \\ \Box\, h2 \to (l \to l2 \to STOP \,\Box\, l' \to l1 \to STOP) \\ \Box\, h3 \to (l \to l1 \to STOP \,\Box\, l' \to l2 \to STOP) \\ \Box\, h4 \to (l \to l2 \to STOP \,\Box\, l' \to l2 \to STOP). \end{array}$$

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

$$\begin{array}{l} \mathcal{Q}(0) \mathrel{\widehat{=}} RUN(L), \quad \mathcal{Q}(1) \mathrel{\widehat{=}} h1 \to RUN(L), \quad \mathcal{Q}(2) \mathrel{\widehat{=}} h2 \to RUN(L), \\ T \mathrel{\widehat{=}} l \to (l1 \to SUCCESS(1) \,\Box\, l2 \to SUCCESS(2) \text{ else } SUCCESS(0)) \\ \qquad \text{else } SUCCESS(0). \end{array}$$

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 nondeterministicly searching either, for example with a test like:

$$\begin{array}{l} l \to (l1 \to SUCCESS(1) \text{ else } SUCCESS(2)) \\ \Box\, l' \to (l2 \to SUCCESS(2) \text{ else } SUCCESS(1)) \\ \text{else } SUCCESS(0), \end{array}$$

(which would distinguish the High behaviours $RUN(L)$, $h1 \to RUN(L)$ and $h4 \to RUN(L)$); 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

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

(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. The above 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.

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

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

High can pass an arbitrary value k by performing k h events. Fix N, and consider the following strategy:

$$\begin{array}{rcll} \mathcal{Q}(0) & \widehat{=} & RUN(L), & \\ \mathcal{Q}(k+1) & \widehat{=} & h \rightarrow l \rightarrow \mathcal{Q}(k), & \text{for } k = 0, \ldots, N-1, \\ T(k) & \widehat{=} & l \rightarrow T(k+1) \mathsf{\ else\ } SUCCESS(k), & \text{for } k = 0, \ldots, N-1, \\ T(N) & \widehat{=} & SUCCESS(N). & \end{array}$$

Then this strategy, using the test $T(0)$, will pass a value in the range $0, \ldots, N$. (Remember the requirement for finite tests; this is why we have to consider a fixed N.) But N is arbitrary so the information flow quantity of this process is infinite.

3.2 Formal analysis

In this section we prove 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 [3], Riccardo Focardi defines a security property *Failures Non Deducibility on Composition (FNDC)*, which can be written in CSP as:

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

which is equivalent to:

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

We propose the following strengthening of FNDC: a process P satisfies *Strong Failures Non Deducibility on Composition (SFNDC)* as follows:

$$P \textbf{ sat } SFNDC \mathrel{\widehat{=}} \forall\, R \sqsupseteq P \bullet R \in FDNC.$$

Our main result is that our definition gives information flow quantity of 1 to precisely those processes that satisfy SFNDC.

We prefer the condition SFNDC to FNDC because it overcomes the following objection of Forster and Roscoe [6]. Consider the processes

$$LEAK \sqcap CHAOS(L) \qquad \text{and} \qquad LEAK \mathbin{|||} CHAOS(L),$$

where $LEAK$ is any insecure process. These processes have clearly insecure behaviours, coming from $LEAK$; however they satisfy $FNDC$, because the insecure behaviour is masked by behaviours of $CHAOS(L)$. These processes do not satisfy SFNDC, because they are refined by $LEAK$.

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 2 If $P \sqsubseteq Q$ then $\forall\, T \bullet \mathit{results}_F(P, T) \supseteq \mathit{results}_F(Q, T)$.

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

$$\begin{array}{cl} & \mathit{results}(P, T) \\ = & \{k \mid \exists(s, X) \in \mathcal{F}[\![P]\!] \bullet (s, X, k) \in \mathcal{F}[\![T]\!]\} \\ \supseteq & \{k \mid \exists(s, X) \in \mathcal{F}[\![Q]\!] \bullet (s, X, k) \in \mathcal{F}[\![T]\!]\} \\ = & \mathit{results}(Q, T). \end{array}$$

□

We now show that our testing scheme precisely captures failures equivalence.

Lemma 3 Suppose P and Q are non-divergent. Then

$$P \equiv_F Q \;\Leftrightarrow\; (\forall\, T \bullet \mathit{results}_F(P, T) = \mathit{results}_F(Q, T)).$$

Proof: The left-to-right implication follows immediately from Lemma 2.

For the converse, we prove the contrapositive. Suppose $P \not\equiv_F Q$. Then without loss of generality, there is some failure (s, X) such that $(s, X) \in \mathcal{F}[\![P]\!]$ but $(s, X) \notin \mathcal{F}[\![Q]\!]$. Let $s = \langle a_1, a_2, \ldots, a_n\rangle$. We construct a test that will succeed with value 1 after this failure, but will succeed with value 0, otherwise:

$$\begin{array}{l} T \mathrel{\widehat=} a_1 \to (a_2 \to (\ldots (a_{n-1} \to (a_n \to (\Box_{x \in X}\, x \to \mathit{SUCCESS}(0) \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathsf{else}\ \mathit{SUCCESS}(1)) \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathsf{else}\ \mathit{SUCCESS}(0)) \\ \qquad\qquad\qquad\qquad\qquad\quad \mathsf{else}\ \mathit{SUCCESS}(0)) \\ \qquad\qquad\qquad\qquad\quad \ldots) \\ \qquad\qquad\qquad \mathsf{else}\ \mathit{SUCCESS}(0)) \\ \qquad\quad \mathsf{else}\ \mathit{SUCCESS}(0). \end{array}$$

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

In [2], de Nicola and Hennessy prove that must testing is equivalent to failures equivalence. Hence our testing scheme is equivalent to must testing.

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

Lemma 4 Let P be non-divergent, and let $(s, X) \in \mathcal{F}[\![P]\!]$. Then we can construct a process R that refines P, and that "forces" (s, X); that is, R cannot refuse the events in s, and cannot perform any of the events of X following s:

$$\begin{aligned}
\mathcal{T}[\![R]\!] &= \{s' \mid s' \in \mathcal{T}[\![P]\!] \wedge \nexists\, a \in X \bullet s^\frown\langle a\rangle \leqslant s'\}, \\
\mathcal{F}[\![R]\!] &= \{(s', X') \mid (s', X') \in \mathcal{F}[\![P]\!] \wedge s' \neq s \wedge \\
&\qquad\qquad (\nexists\, a \in X \bullet s^\frown\langle a\rangle \leqslant s') \wedge (\nexists\, a \in X' \bullet s'^\frown\langle a\rangle \leqslant s)\} \\
&\quad \cup \{(s, X') \mid X' \subseteq X \cup \overline{X}\},
\end{aligned}$$

where $\overline{X}$ is the set of events that P cannot perform following s:

$$\overline{X} \mathrel{\widehat{=}} \{a \mid s^\frown\langle a\rangle \notin \mathcal{T}[\![P]\!]\}.$$

Proof: We need to show that the traces and refusals constructed above satisfy the axioms of the Failures Model of CSP. The traces are clearly non-empty and prefix-closed. We prove the three failures-related axioms as follows:

1. Suppose $(s', X') \in \mathcal{F}[\![R]\!]$. We show $s' \in \mathcal{T}[\![R]\!]$. There are two cases:
   - Case $s' \neq s$. Then $(s', X') \in \mathcal{F}[\![P]\!]$ so $s' \in \mathcal{T}[\![P]\!]$. Also $\nexists\, a \in X \bullet s^\frown\langle a\rangle \leqslant s'$. Hence $s' \in \mathcal{T}[\![R]\!]$.
   - $s' = s$. Then $s' \in \mathcal{T}[\![P]\!]$, and so we immediately have $s' \in \mathcal{T}[\![R]\!]$.
2. Suppose $(s', X') \in \mathcal{F}[\![R]\!]$ and $X'' \subseteq X'$. We show $(s', X'') \in \mathcal{F}[\![R]\!]$. There are two cases:
   - Case $s' \neq s$. Then $(s', X') \in \mathcal{F}[\![P]\!]$ so $(s', X'') \in \mathcal{F}[\![P]\!]$. Also $\nexists\, a \in X \bullet s^\frown\langle a\rangle \leqslant s'$. And $\nexists\, a \in X' \bullet s'^\frown\langle a\rangle \leqslant s$ so $\nexists\, a \in X'' \bullet s'^\frown\langle a\rangle \leqslant s$. Hence $(s', X'') \in \mathcal{F}[\![R]\!]$.
   - Case $s' = s$. Then $X'' \subseteq X' \subseteq X \cup \overline{X}$, so $(s', X'') \in \mathcal{F}[\![R]\!]$.
3. Suppose $(s', X') \in \mathcal{F}[\![R]\!]$ and $\forall\, b \in X'' \bullet s'^\frown\langle b\rangle \notin \mathcal{T}[\![R]\!]$. We show $(s, X' \cup X'') \in \mathcal{F}[\![R]\!]$. We consider two cases:
   - Case $s \neq s'$. Hence $(s', X') \in \mathcal{F}[\![P]\!]$.
     Suppose for a contradiction that for some $b \in X''$, $s'^\frown\langle b\rangle \in \mathcal{T}[\![P]\!]$. But $s'^\frown\langle b\rangle \notin \mathcal{T}[\![R]\!]$, so from the definition of $\mathcal{T}[\![R]\!]$ we must have that for some $a \in X$, $s^\frown\langle a\rangle \leqslant s'^\frown\langle b\rangle$. But $s \neq s'$, so $s^\frown\langle a\rangle \leqslant s'$. But then the definition of $\mathcal{F}[\![R]\!]$ implies $s' \notin \mathcal{T}[\![R]\!]$, contradicting the above. Hence $\forall\, b \in X'' \bullet s'^\frown\langle b\rangle \notin \mathcal{T}[\![P]\!]$, and so $(s', X' \cup X'') \in \mathcal{F}[\![P]\!]$.
     We immediately have that $\nexists\, a \in X \bullet s^\frown\langle a\rangle \leqslant s'$ and $\nexists\, a \in X' \bullet s'^\frown\langle a\rangle \leqslant s$. Also, if $a \in X''$ then $s'^\frown\langle a\rangle \notin \mathcal{T}[\![R]\!]$ and so $s'^\frown\langle a\rangle \not\leqslant s$. Hence $(s, X' \cup X'') \in \mathcal{F}[\![R]\!]$.
   - Case $s = s'$. For all $b \in X'' - X$, if we had $s^\frown\langle b\rangle \in \mathcal{T}[\![P]\!]$, then we would have $s^\frown\langle b\rangle \in \mathcal{T}[\![R]\!]$, contradicting the above. Hence $X'' - X \subseteq \overline{X}$, so $X'' \subseteq X \cup \overline{X}$, so $(s, X'') \in \mathcal{F}[\![R]\!]$.

Finally, it can be seen that $R \sqsupseteq P$ by checking inclusion of behaviours, noting that $(s, X \cup \overline{X}) \in \mathcal{F}[\![P]\!]$. $\square$

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:

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

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

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

then

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

Proof of Lemma 1: Consider $P \underset{H}{\parallel} STOP = (P \underset{H}{\parallel} STOP) \setminus H$. This cannot be equal 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

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

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

- Case there is some $(s, X) \in \mathcal{F}[\![(P \underset{\Sigma}{\parallel} Q_1) \setminus H]\!] - \mathcal{F}[\![P \underset{H}{\parallel} STOP]\!]$. Then there is some trace s' such that $s' \setminus H = s$ and $(s', X \cup H) \in \mathcal{F}[\![P \underset{\Sigma}{\parallel} Q_1]\!]$. But Q_1 satisfies equation (1), and so $(s', X \cap L) \in \mathcal{F}[\![P]\!]$. Let $R \mathrel{\widehat{=}} P$ and $Q_1' \mathrel{\widehat{=}} Tr(s' \upharpoonright H)$. Then $(s', X \cup H) \in \mathcal{F}[\![P \underset{H}{\parallel} Q_1']\!]$ and so $(s, X) \in \mathcal{F}[\![(P \underset{H}{\parallel} Q_1') \setminus H]\!]$. Hence $(P \underset{H}{\parallel} STOP) \setminus H \not\equiv_F (P \underset{H}{\parallel} Q_1') \setminus H$, as required.
- Case there is some $(s, X) \in \mathcal{F}[\![P \underset{H}{\parallel} STOP]\!] - \mathcal{F}[\![(P \underset{\Sigma}{\parallel} Q_1) \setminus H]\!]$, with s minimal, so $s \in \mathcal{T}[\![(P \underset{\Sigma}{\parallel} Q_1) \setminus H]\!]$.

  Then $(s, X \cap L) \in \mathcal{F}[\![P]\!]$. Let $R \sqsupseteq P$ be as in Lemma 4 such that $(s, X \cap L)$ is "forced". Then for all $a \in X \cap L$, we have $s^\frown\langle a\rangle \notin \mathcal{T}[\![R]\!]$, so

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

  Now, $s \in \mathcal{T}[\![(P \underset{\Sigma}{\parallel} Q_1) \setminus H]\!]$, so there is some s' such that $s' \setminus H = s$ and $s' \in \mathcal{T}[\![P \underset{\Sigma}{\parallel} Q_1]\!]$, and so $s' \in \mathcal{T}[\![P]\!]$. Thus $s' \in \mathcal{T}[\![R]\!]$, from the definition of R in Lemma 4 (there is no $a \in X \cap L$ such that $s^\frown\langle a\rangle \leqslant s'$). Hence $s' \in \mathcal{T}[\![R \underset{\Sigma}{\parallel} Q_1]\!]$ and so $s \in \mathcal{T}[\![(R \underset{\Sigma}{\parallel} Q_1) \setminus H]\!]$. But $(s, X) \notin \mathcal{F}[\![(P \underset{\Sigma}{\parallel} Q_1) \setminus H]\!]$ and $R \sqsupseteq P$ so $(s, X) \notin \mathcal{F}[\![(R \underset{\Sigma}{\parallel} Q_1) \setminus H]\!]$. Hence, there is some $a \in X \cap L$

such that $s^\frown\langle a\rangle \in \mathcal{T}[\![(R \mathbin{\underset{\Sigma}{\|}} Q_1) \setminus H]\!]$. Let s'' be the corresponding trace of $R \mathbin{\underset{\Sigma}{\|}} Q_1$, so $s'' \setminus H = s^\frown\langle a\rangle$, and $s'' \in \mathcal{T}[\![R]\!]$. Let $Q_1' \mathrel{\widehat{=}} Tr(s'' \upharpoonright H)$, so $s'' \in \mathcal{T}[\![R \mathbin{\underset{H}{\|}} Q_1']\!]$, and so $s^\frown\langle a\rangle \in \mathcal{T}[\![(R \mathbin{\underset{H}{\|}} Q_1') \setminus H]\!]$. But $s^\frown\langle a\rangle \notin \mathcal{T}[\![(R \mathbin{\underset{H}{\|}} STOP) \setminus H]\!]$, so $(R \mathbin{\underset{H}{\|}} Q_1') \setminus H \not\equiv_F (R \mathbin{\underset{H}{\|}} STOP) \setminus H$, as required.

- Case there is some trace $s^\frown\langle a\rangle \in \mathcal{T}[\![P \mathbin{\underset{H}{\|}} STOP]\!] - \mathcal{F}[\![(P \mathbin{\underset{\Sigma}{\|}} Q_1) \setminus H]\!]$ with s minimal, so $(s, \{a\}) \in \mathcal{F}[\![(P \mathbin{\underset{\Sigma}{\|}} Q_1) \setminus H]\!]$.

  Then $s^\frown\langle a\rangle \in \mathcal{T}[\![P]\!]$. Let $R \sqsupseteq P$ be as in Lemma 4 such that the failure $(s^\frown\langle a\rangle, \{\})$ is "forced". Then $(s, \{a\}) \notin \mathcal{F}[\![R]\!]$ so

  $$(s, \{a\}) \notin \mathcal{F}[\![(R \mathbin{\underset{H}{\|}} STOP) \setminus H]\!].$$

  Now, $\mathcal{T}[\![P]\!] = \mathcal{T}[\![R]\!]$ in this case, from the definition of R in Lemma 4, and so $s \in \mathcal{T}[\![(R \mathbin{\underset{\Sigma}{\|}} Q_1) \setminus H]\!]$ and $s^\frown\langle a\rangle \notin \mathcal{T}[\![(R \mathbin{\underset{\Sigma}{\|}} Q_1) \setminus H]\!]$. Hence $(s, \{a\}) \in \mathcal{F}[\![(R \mathbin{\underset{\Sigma}{\|}} Q_1) \setminus H]\!]$. Thus there is some s' such that $s' \setminus H = s$ and $(s', \{a\} \cup H) \in \mathcal{F}[\![R \mathbin{\underset{\Sigma}{\|}} Q_1]\!]$. Hence $(s', \{a\}) \in \mathcal{F}[\![R]\!]$, because Q_1 satisfies equation (1). Let $Q_1' \mathrel{\widehat{=}} Tr(s' \upharpoonright H)$. Then $(s', \{a\} \cup H) \in \mathcal{F}[\![R \mathbin{\underset{H}{\|}} Q_1']\!]$, so $(s, \{a\}) \in \mathcal{F}[\![(R \mathbin{\underset{H}{\|}} Q_1') \setminus H]\!]$. Hence $(R \mathbin{\underset{H}{\|}} STOP) \setminus H \not\equiv_F (R \mathbin{\underset{H}{\|}} Q_1') \setminus H$, as required.

$\square$

We now prove 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.

Lemma 5 If $(P \mathbin{\underset{H}{\|}} Q_0) \setminus H \not\equiv_F (P \mathbin{\underset{H}{\|}} Q_1) \setminus H$ then there exist $R \sqsupseteq P$, $Q_0', Q_1' \in CSP$ satisfying equation (1), and $T \in \mathit{Test}$ such that

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

Proof: Following Lemma 3, suppose

$$(s, X) \in \mathcal{F}[\![(P \mathbin{\underset{H}{\|}} Q_1) \setminus H]\!] - \mathcal{F}[\![(P \mathbin{\underset{H}{\|}} Q_0) \setminus H]\!],$$

and let T be as in Lemma 3. The results of that lemma show

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

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

Suppose (s, X) corresponds to the failure $(s', X \cup H)$ of $P \underset{H}{\parallel} Q_1$, and the failures (s', X_P) and $(s' \upharpoonright H, X_Q)$ of P and Q respectively, so that $X_P \cup X_Q = X \cup H$, $X_P \supseteq X \cap L$, $X_Q \subseteq H$. Let $R \sqsupseteq P$ be—as in Lemma 4—such that none of the events of s' is refused, and none of X_P is performed. Let $Q_1' \mathrel{\widehat{=}} f(s')$ where

$$\begin{array}{rcll} f(\langle l \rangle ^\frown s'') & \widehat{=} & l \rightarrow f(s'') \Box \Box_{l' \in L - \{l\}} l' \rightarrow RUN(L), & \text{for } l \in L \\ f(\langle h \rangle ^\frown s'') & \widehat{=} & h \rightarrow f(s''), & \text{for } h \in H \\ f(\langle\rangle) & \widehat{=} & RUN(L). & \end{array}$$

And 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. Further, R cannot refuse any of these events. After s', R will refuse X_P, and Q_1' will refuse H, so $R \underset{\Sigma}{\parallel} Q_1'$ will refuse $X \cup H$, so $(R \underset{\Sigma}{\parallel} Q_1') \setminus H$ will refuse X. Hence

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

Note that Q_1' satisfies equation (1): in those states that it refuses H, it offers all of L.

Let $Q_0' \mathrel{\widehat{=}} Q_0 \mathbin{|||} RUN(L)$. Note that Q_0' satisfies equation (1). Then

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

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 2. □

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

Theorem 6 A process P has information flow quantity 1 iff for all refinements R, High cannot change the way the process appears to Low:

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

The above theorem is equivalent to

$$IFQ_F(P) = 1 \Leftrightarrow P \ \mathbf{sat} \ SFNDC.$$

Proof: We prove the contrapositive. In one direction:

$$
\begin{array}{ll}
& \neg(\forall\, R \sqsupseteq P\ ;\ Q_0, Q_1 \in CSP_H \bullet (R \underset{H}{\parallel} Q_0) \setminus H \equiv_F (R \underset{H}{\parallel} Q_1) \setminus H \\
\Leftrightarrow & \exists\, R \sqsupseteq P\ ;\ Q_0, Q_1 \in CSP_H \bullet (R \underset{H}{\parallel} Q_0) \setminus H \not\equiv_F (R \underset{H}{\parallel} Q_1) \setminus H \\
\Rightarrow & \langle \text{Lemma 5} \rangle \\
& \exists\, R \sqsupseteq P \bullet \exists\, R' \sqsupseteq R\ ;\ Q_0', Q_1' \in CSP\ ;\ T \in \mathit{Test} \bullet \\
& \qquad \mathit{results}(R', Q_0', T) = \{0\} \wedge \mathit{results}(R', Q_1', T) = \{1\} \wedge Q_0',\ Q_1' \text{ satisfy (1)} \\
\Rightarrow & \mathit{IFQ}(P) \geqslant 2.
\end{array}
$$

In the other direction:

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

□

4 Quantifying information flow in a timed model

We now extend the definition of the quantity of information flow to timed processes. We assume that the behaviour of the system is modelled by a Timed CSP process P. Much of our development is very similar to as in the untimed setting; we concentrate on describing the differences.

We will model High's behaviour by a Timed CSP process Q, whose alphabet may include events from L. The system from Low's point of view will then look like $(P \underset{\Sigma}{\parallel} Q) \setminus H$. We will insist that Q satisfies the following condition, which is the timed equivalent of equation (1):

$$
\begin{array}{l}
\forall\, R \sqsupseteq P \bullet \forall (s, X) \bullet \\
\qquad (s, X \cup [0, \mathrm{end}(s, X)) \times H) \in \mathcal{TF}[\![R \underset{\Sigma}{\parallel} Q]\!] \Rightarrow (s, X \upharpoonright L) \in \mathcal{TF}[\![R]\!].
\end{array} \tag{2}
$$

We will model High's strategy as a function $\mathcal{Q} : \mathbb{N} \nrightarrow TCSP$ such that $\mathcal{Q}(k)$ satisfies equation (2) for all $k \in \mathrm{dom}\, \mathcal{Q}$.

As in the untimed case, Low will interact with the system via a test, offering events, and maybe seeing them observed. However, if none of the events are accepted within some time period, we will allow Low to timeout and change the events it offers. We will write

$$(\Box_{a \in A} a \to T_a) \overset{t}{\triangleright} T'$$

to represent a test that attempts the set of events A (where $A \subseteq L$), and which then acts like T_a if a particular event a is performed; if all the events of A are refused for t time units, then the test times-out and acts like T'.

We will assume that Low does not have access to a clock. However, tests of the above form can be used to simulate a discreet-time clock, by timing out at regular intervals and counting the number of such timeouts.

However, we need to ensure that Low cannot simulate an arbitrarily accurate clock, which would be possible if we allowed Low to timeout after arbitrarily short amounts of time. For example, consider the process

$$h \to l \to STOP \overset{1}{\triangleright} STOP.$$

If Low could simulate an arbitrarily accurate clock, then a value in an arbitrary range $0, \ldots, N$ could by passed as follows: split the time interval $[0, 1)$ into N sub-intervals of length $1/N$; High could then pass a value $k \in \{0, \ldots, N-1\}$ by performing h during the kth such interval; Low could recover the value from the time at which l becomes available. As N is arbitrary, this would give an unbounded quantity of information flow, which seems unreasonable.

We will therefore assume a finite speed for Low: we assume a *minimum delay* $\delta > 0$, such that Low cannot timeout after times of less than δ. We will restrict tests of the above form to those that have $t \geqslant \delta$.

We will write $Test(\delta)$ for the set of tests with minimum delay δ, defined as follows:

$$T \in Test(\delta) \ ::= \ SUCCESS(k) \mid (\Box_{a \in A} a \to T_a) \overset{t}{\triangleright} T', \qquad \text{where } t \geqslant \delta.$$

We will write $WAIT\ t\ ;\ T$ as a shorthand for $(\Box_{a \in \{\}} a \to T_a) \overset{t}{\triangleright} T$, for $t \geqslant \delta$.

We associate a set of (timed trace, timed refusal, result) triples with each test, representing that the test can observe the given timed trace and refusal of the process it is testing, and then return the given result.

$$\begin{aligned}
\mathcal{TF}[\![SUCCESS(k)]\!] \ &\widehat{=}\ \{(\langle\rangle, \{\}, k)\}, \\
\mathcal{TF}[\![(\Box_{a \in A} a \to T_a) \overset{t}{\triangleright} T']\!] \ &\widehat{=}\ \{(\langle(t', a)\rangle^\frown s + t', [0, t') \times A \cup X + t', k) \mid \\
&\qquad a \in A \wedge (s, X, k) \in \mathcal{TF}[\![T_a]\!] \wedge t' \in [0, t)\} \\
&\quad \cup \\
&\quad \{(s + t, [0, t) \times A \cup X + t, k) \mid \\
&\qquad (s, X, k) \in \mathcal{TF}[\![T']\!]\}.
\end{aligned}$$

The test $SUCCESS(k)$ can immediately give the result k, having observed no failures information. The test $(\Box_{a \in A} a \rightarrow T_a) \overset{t}{\triangleright} T'$ can see an event $a \in A$ performed at some time $t' \in [0, t)$, and then act like the test T_a, provided all of A is refused up to that time; alternatively, it can see all of A refused throughout $[0, t)$, and then act like the test T'.

We will write $results_{TF}(S, T)$ for the set of results obtainable from timed-failures-based test T with system S; that is, the set of results k such that there is some timed failure (s, X) that the system can perform, and such that the test associates result k with this failure:

$$results_{TF}(S, T) \mathrel{\widehat{=}} \{k \mid \exists (s, X) \in \mathcal{TF}[\![S]\!] \bullet (s, X, k) \in \mathcal{TF}[\![T]\!]\}.$$

As in the untimed model, we will overload the *results* function as follows:

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

However, we should only consider strategies of High and Low that are compatible:

$$\begin{aligned} ok_{TF}(P, \mathcal{Q}, T) \mathrel{\widehat{=}} \forall\, k \in \operatorname{dom} \mathcal{Q} \bullet\ & results_{TF}(P, \mathcal{Q}(k), T) = \{k\} \\ & \text{and } \mathcal{Q}(k) \text{ satisfies equation (2).} \end{aligned}$$

The definition of information flow quantity is then analogous to as in the untimed model:

$$\begin{aligned} IFQ_{TF}(P) \mathrel{\widehat{=}} \max\{\#\operatorname{dom} \mathcal{Q} \mid\ & \mathcal{Q} \in \mathbb{N} \nrightarrow TCSP \wedge T \in Test \wedge R \in TCSP \\ & \wedge P \sqsubseteq R \wedge ok_{TF}(R, \mathcal{Q}, T)\}. \end{aligned}$$

4.1 Example

Consider again the process:

$$P \;=\; h \rightarrow l \rightarrow STOP \overset{1}{\triangleright} STOP.$$

Let $N \mathrel{\widehat{=}} \lceil 1/\delta \rceil$. Suppose for the moment that $1/\delta$ is not an integer. We can construct a strategy for High as follows: in order to pass a value $k \in \{0, \ldots, N-1\}$, perform h at time $k\delta$; and in order to pass the value N, do not perform h:

$$\begin{aligned} \mathcal{Q}(k) &= WAIT\ k\delta\,; h \rightarrow l \rightarrow STOP, \qquad \text{for } k \in \{0, \ldots, N-1\} \\ \mathcal{Q}(N) &= STOP. \end{aligned}$$

Then

$$\begin{aligned} (P \underset{\Sigma}{\parallel} \mathcal{Q}(k)) \setminus H &= WAIT\ k\delta\,; l \rightarrow STOP, \qquad \text{for } k \in \{0, \ldots, N-1\} \\ (P \underset{\Sigma}{\parallel} \mathcal{Q}(N)) \setminus H &= STOP. \end{aligned}$$

We construct a corresponding strategy for Low, attempting to perform l, and giving result k if l is accepted during time $[k\delta, (k+1)\delta)$ for $k = 0, \ldots, N-1$, and result N otherwise. This is the test $T(0)$ where:

$$\begin{aligned} T(k) &= l \rightarrow \mathit{SUCCESS}(k) \overset{\delta}{\triangleright} T(k+1), \qquad \text{for } k \in \{0, \ldots, N-1\} \\ T(N) &= \mathit{SUCCESS}(N). \end{aligned}$$

Then

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

So with this strategy we achieve an information flow of $N + 1 = \lceil 1/\delta \rceil + 1$.

It is not difficult to see that High and Low cannot pass more information than this, because Low cannot sample more quickly, and so this figure is indeed the information flow quantity.

If $1/\delta$ is an integer, then the same strategy works, and achieves an information flow quantity of $N + 1 = 1/\delta + 1$. It might be thought that we can achieve one higher, because h can now be performed at time $N\delta = 1$. However, that strategy:

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

doesn't work. The reason is that when an event is withdrawn by a process—for example, when h is withdrawn at time 1 by P—it is nondeterministic whether or not the event is available at that instant; this is often referred to as a *point nondeterminism*. This means that

$$(P \underset{\Sigma}{\|} Q(N)) \setminus H = \mathit{WAIT}\, 1\, ; l \rightarrow \mathit{STOP} \sqcap \mathit{STOP}.$$

It is nondeterministic whether the h will be performed at time 1, or whether it will be preempted by the timeout. Hence

$$\mathit{results}(P, Q(N), T(0)) = \{N, N+1\},$$

and so the *ok* predicate is not satisfied.

One might consider the refinement R of P such that the point nondeterminism at time 1 is resolved so that the h is always available at that time. However, this R is not a Timed CSP process: the Timed Failures Model [9, 14] defines refusals to be composed of refusal tokens of the form $[t_0, t_1) \times A$ (where $t_0, t_1 \in [0, \infty)$, $A \in \mathbb{P}\,\Sigma$); however R would have a maximal refusal including $(1, 2) \times \{h\}$ but not $(1, h)$. This property is an artifact of the model, chosen because it makes sense in most circumstances; however, it is not what we want here. We believe the way to overcome this would be to allow refusal tokens of the form $(t_0, t_1) \times A$. We intend to investigate the consequences of this change to the model.

However, we do not consider this a major shortcoming: in any real system, it will not be possible to estimate the minimum delay δ and the timeout 1 with total accuracy, so any values we give them will be approximations; the chances of $1/\delta$ really being an integer seem remote.

4.2 Analysis

The following lemma is analogous to Lemma 2 in the untimed model:

Lemma 7 If $P \sqsubseteq_{TF} Q$ then $\forall\, T \bullet \mathit{results}_{TF}(P, T) \supseteq \mathit{results}_{TF}(Q, T)$.

Proof: Suppose $P \sqsubseteq_{TF} Q$, and let T be a test. Then

$$\begin{array}{ll} & \mathit{results}(P, T) \\ = & \{k \mid \exists (s, X) \in \mathcal{TF}[\![P]\!] \bullet (s, X, k) \in \mathcal{TF}[\![T]\!]\} \\ \supseteq & \{k \mid \exists (s, X) \in \mathcal{TF}[\![Q]\!] \bullet (s, X, k) \in \mathcal{TF}[\![T]\!]\} \\ = & \mathit{results}(Q, T). \end{array}$$

□

The following corollary is immediate from the previous result.

Corollary 8 If $P \equiv_{TF} Q$ then $\forall\, T \bullet \mathit{results}_{TF}(P, T) = \mathit{results}_{TF}(Q, T)$.

One might suspect, as in the untimed case, that the converse of the above result holds: that if P and Q are any two distinct processes, then given a suitably small minimum delay, we can find a test T that distinguishes them. This turns out to be not quite true. Consider the two processes

$$\begin{aligned} P_1 &\mathrel{\widehat{=}} \textstyle\bigsqcap_{t \geqslant 0} \mathit{WAIT}\; t \,;\, a \to \mathit{STOP}, \\ P_2 &\mathrel{\widehat{=}} \textstyle\bigsqcap_{t > 0} \mathit{WAIT}\; t \,;\, a \to \mathit{STOP}. \end{aligned}$$

P_1 and P_2 are distinct: only the former can perform a at time 0. However no test distinguishes them. The obvious form of test to consider is:

$$T \mathrel{\widehat{=}} a \to \mathit{SUCCESS}(1) \overset{t}{\rhd} \mathit{SUCCESS}(0),$$

for some t. However, this will only distinguish P_1 and P_2 if we take $t = 0$, which is not allowed, because of our bounded-speed restriction on tests.

The two processes above are unusual in that the involve an infinite nondeterministic choice. If we restrict ourselves to sufficiently well behaved processes, then tests can be found to distinguish any pair of processes. More precisely, we restrict ourselves to processes satisfying the following condition:

$$\forall\, s, a, t \bullet \exists\, \varepsilon > 0 \bullet \nexists\, t' \in (t, t+\varepsilon) \bullet (s \frown \langle (t', a) \rangle, [t, t') \times \{a\}) \in \mathcal{TF}[\![P]\!]. \quad (3)$$

The condition says that following trace s, there is a lower bound $t + \varepsilon$ on those times strictly after t at which a can become available. This condition is satisfied by all processes that do not use an infinite nondeterministic choice or an infinite prefix choice.

Lemma 9 If P and Q satisfy equation (3), then

$$(\forall\, \delta > 0 \bullet \forall\, T \in \mathit{Test}(\delta) \bullet \mathit{results}(P, T) = \mathit{results}(Q, T)) \;\Rightarrow\; P \equiv_{TF} Q.$$

Proof: We prove the contrapositive. Suppose $P \not\equiv_F Q$. Then, without loss of generality, there is some failure $(s, X) \in \mathcal{TF}[\![P]\!] - \mathcal{TF}[\![Q]\!]$.

We construct a test T that will succeed with result 1 following the timed failure (s, X), but will not give the result 1 for any timed failure of Q. T will offer each event of s and each refusal token of X in turn, and will give result 1 only if all the former are accepted and all the latter are refused. However, we have to be careful in our construction that T does not accept any other traces of Q, that differ from s only in the precise times at which events occur.

Let $s = \langle (t_1, a_1), \ldots, (t_n, a_n) \rangle$, and for convenience let $t_0 \mathrel{\widehat{=}} 0$ and $t_{n+1} \mathrel{\widehat{=}} \mathrm{end}(s, X)$. X contains a finite number of refusal tokens. By splitting and combining refusal tokens, we can assume that they have disjoint and non-empty time domains, that no event occurs within one of those time domains, and that the time domains cover the period $[0, \mathrm{end}(s, X))$. Let the interval $[t_i, t_{i+1})$ be split into subintervals $[t_{i,j}, t_{i,j+1})$ for $j = 0, \ldots, k_i - 1$, where $t_{i,0} = t_i$, and $t_{i,k_i} = t_{i+1}$ for $i = 0, \ldots, n$, and write X as follows:

$$X \;=\; \bigcup_{i=0,\ldots,n} \;\bigcup_{j=0,\ldots,k_i-1} [t_{i,j}, t_{i,j+1}) \times A_{i,j},$$

From (3), for each $k \in \{1, \ldots, n\}$, there exists $\varepsilon_k > 0$ such that

$$\begin{array}{l} \nexists\, t' \in (t_k, t_k + \varepsilon_k) \bullet \\ \qquad (\langle (t_1, a_1), \ldots, (t_{k-1}, a_{k-1}), (t', a_k) \rangle,\ [t_k, t') \times \{a_k\}) \in \mathcal{TF}[\![Q]\!]. \end{array} \quad (4)$$

We let $T = T_{0,0}$, where:

$$\begin{array}{rcll} T_{i,j} & \widehat{=} & \Box_{a \in A_{i,j}}\, a \to \mathit{SUCCESS}(0) \overset{t_{i,j+1} - t_{i,j}}{\rhd} T_{i,j+1}, & \text{for } i = 0, \ldots, n, \\ & & & \quad j = 0, \ldots, k_i - 1, \\ T_{i,k_i} & \widehat{=} & a_{i+1} \to T_{i+1,0} \overset{\varepsilon_{i+1}}{\rhd} \mathit{SUCCESS}(0), & \text{for } i = 0, \ldots, n-1, \\ T_{n,k_n} & \widehat{=} & \mathit{SUCCESS}(1). & \end{array}$$

Pick δ smaller than all the ε_k and all the $t_{i,j+1} - t_{i,j}$; then $T \in \mathit{Test}(\delta)$.

This test can give result 1 with process P, corresponding to failure (s, X). Result 1 is not possible with process Q: from (4), and a simple induction, Q will only be able to perform each event a_i at time t_i, if at all; hence the only possible failure giving result 1 would be (s, X), but we have assumed this is not a failure of Q. $\square$

In [13], Schneider proves that timed may testing is equivalent to finite timed failures equivalence; hence for processes satisfying equation (3), our testing scheme is equivalent to may testing.

By analogy with the untimed case, one might expect that if there are high level processes Q_0 and Q_1 such that $(P \underset{H}{\parallel} Q_0) \setminus H \not\equiv_{TF} (P \underset{H}{\parallel} Q_1) \setminus H$,, and both systems satisfy equation (3), then P would have information flow quantity of at least 2, given a suitable minimum delay. However, this is not the case, because of the possibility of point nondeterminism, mentioned in Section 4.1. Consider

$$P \mathrel{\widehat{=}} h \to l \to STOP \overset{0}{\triangleright} STOP,$$
$$Q_0 \mathrel{\widehat{=}} STOP, \qquad Q_1 \mathrel{\widehat{=}} h \to STOP.$$

Then $(P \underset{H}{\parallel} Q_0) \setminus H \not\equiv_{TF} (P \underset{H}{\parallel} Q_1) \setminus H$, because the latter process can perform l, but the former can't. However, if we consider the corresponding strategy

$$\mathcal{Q}(0) \mathrel{\widehat{=}} STOP, \qquad \mathcal{Q}(1) \mathrel{\widehat{=}} h \to l \to STOP,$$
$$T \mathrel{\widehat{=}} l \to SUCCESS(1) \ \mathsf{else}\ SUCCESS(0)$$

then this does not work: it is nondeterministic whether or not the h will be performed at time 0 or not, and so $\mathit{results}(P, \mathcal{Q}(1), T) = \{0, 1\}$. As above, we would like to consider the refinement of P where the h is always accepted at time 0; however this is not a Timed CSP process. The problem can again be considered unlikely: in practice the timeout is unlikely to occur at time 0, but instead at some small time $\varepsilon > 0$; in this case, the above strategy does indeed achieve an information flow of 2. We suspect that adapting the Timed Failures Model, as described above, would fix this problem.

In [5], Focardi, Gorrieri and Martinelli define a security property *Timed Bisimulation Non Deducibility on Composition (TBNDC)* (within the context of a discreet-time process algebra) which can be written using our notation as:

$$P \ \mathbf{sat}\ TBNDC \Leftrightarrow \forall\, Q \in TCSP_H \bullet P \underset{H}{\parallel} STOP \equiv_B (P \underset{H}{\parallel} Q) \setminus H,$$

where $\equiv_B$ is bisimulation equivalence. This prompts definitions for security properties *Timed Failures Non Deducibility on Composition (TFNDC)* and *Strong Timed Failures Non Deducibility on Composition (STFNDC)* as follows:

$$P \ \mathbf{sat}\ TFNDC \Leftrightarrow \forall\, Q \in TCSP_H \bullet P \underset{H}{\parallel} STOP \equiv_F (P \underset{H}{\parallel} Q) \setminus H,$$
$$P \ \mathbf{sat}\ STFNDC \Leftrightarrow \forall\, R \sqsupseteq P \bullet R \ \mathbf{sat}\ TFDNC.$$

We conjecture that the above changes to the Timed Failures Model would mean that no information flow in our model would correspond to Strong Timed Failures Non Deducibility on Composition.

5 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 up to time t as follows:

$$results_t(S, T) \mathrel{\widehat{=}} \{k \mid \exists (s, X) \in \mathcal{TF}[\![S]\!] \bullet (s, X, k) \in \mathcal{TF}[\![T]\!] \wedge \mathrm{end}(s, X) \leqslant t\}.$$

Note that we only consider observations up to time t.

The following definitions are obvious adaptations of previous ones.

$$\begin{aligned} results_t(P, Q, T) &\mathrel{\widehat{=}} results_t((P \mathbin{\underset{\Sigma}{\|}} Q) \setminus H, T), \\ ok_t(P, \mathcal{Q}, T) &\mathrel{\widehat{=}} \forall\, k \in \mathrm{dom}\,\mathcal{Q} \bullet results_t(P, \mathcal{Q}(k), T) = \{k\} \\ &\qquad\qquad\qquad\qquad \text{and } \mathcal{Q}(k) \text{ satisfies (2)}, \\ IFQ_t(P) &\mathrel{\widehat{=}} \max\{\#\,\mathrm{dom}\,\mathcal{Q} \mid \mathcal{Q} \in \mathbb{N} \nrightarrow TCSP \wedge T \in Test \wedge R \in TCSP \\ &\qquad\qquad\qquad\qquad \wedge R \sqsupseteq P \wedge ok_t(R, \mathcal{Q}, T)\}. \end{aligned}$$

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

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

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.

5.1 Examples

As a first example, consider

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

Then, by reasoning as in Section 4.1, we can show that

$$IFQ_t(P) = \lfloor t/\delta \rfloor + 1, \qquad \text{for all } t.$$

So the long term information flow rate is zero.

Now consider

$$P \;\widehat{=}\; h \rightarrow l \xrightarrow{1} P,$$

and suppose, for ease of calculation, that $\delta = 1$. It is fairly clear that this process can be used to pass one bit per time unit, by High either performing an h or not, and Low either observing an l or not.

We present a strategy that can be used to pass a number in the range $0, \ldots, 2^{\lfloor t \rfloor} - 1$, by passing each of the $\lfloor t \rfloor$ bits, one at a time, starting with the least significant bit. High's behaviour is given by

$$Q(k) \;\widehat{=}\; \text{if } k \bmod 2 = 1 \text{ then } h \rightarrow l \xrightarrow{1} Q(k \operatorname{div} 2) \text{ else } \mathit{WAIT}\,1 \;;\; Q(k \operatorname{div} 2).$$

Performing an h is used to signal a 1 bit. Low's test is defined by $T(0,0)$, below. The test accumulates the result in the parameter r; the parameter i records the number of digits received so far.

$$\begin{aligned} T(i,r) \;&\widehat{=}\; l \rightarrow (\mathit{WAIT}\,1 \;;\; T(i+1, 2^i + r)) \overset{1}{\triangleright} T(i+1, r), \\ &\qquad \text{for } i = 0, \ldots, \lfloor t \rfloor - 1, \\ T(\lfloor t \rfloor, r) \;&\widehat{=}\; \mathit{SUCCESS}(r). \end{aligned}$$

This achieves an information flow quantity of $2^{\lfloor t \rfloor}$. Note that each result corresponds to a failure ending at time $\lfloor t \rfloor$. This is the maximum achievable within time t. Hence $\mathit{IFQ}_t(P) = 2^{\lfloor t \rfloor}$, and the long term information flow rate is 1.

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:

$$\begin{aligned} \mathit{EXEC}_H \;&\widehat{=}\; \mathit{PROC}_H \mathbin{\triangle_q} \mathit{CONTENTION}_L, \\ \mathit{EXEC}_L \;&\widehat{=}\; \mathit{PROC}_L \mathbin{\triangle_q} \mathit{CONTENTION}_H. \end{aligned}$$

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.

$$\begin{aligned} \mathit{CONTENTION}_H \;&\widehat{=}\; \mathit{req}_h \rightarrow \mathit{EXEC}_H \overset{w}{\triangleright} \mathit{CONTENTION}_L, \\ \mathit{CONTENTION}_L \;&\widehat{=}\; \mathit{req}_l \rightarrow \mathit{EXEC}_L \overset{w}{\triangleright} \mathit{CONTENTION}_H. \end{aligned}$$

Initially, High has priority:

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

We assume that δ is less than w and q.

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 again pass the bits in order, least significant bit first. High's strategy to pass the value k can be modelled by the process $Q(k) \mathbin{|||} RUN(L)$[2] where:

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

Low's strategy is to attempt req_l events at times of the form $w+(q+w)i$, and is modelled by the test $WAIT\ w\,;\, T(0,0)$ where

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

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 $\frac{1}{q+w}$.

6 Conclusions

In this paper we have presented a formal definition for 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.

[2] Strictly speaking, $RUN(L)$ is not a Timed CSP process, because it does not satisfy the finite speed axiom of the Timed Failures Model; we take $RUN(L)$ to be a process that will perform as many events from L as its environment requires.

The untimed model gives the correct values for all the examples considered. It agrees with a more intuitive definition on those processes that exhibit no flow of information. The timed model runs into problems with point nondeterminism if High attempts events at the precise point at which they are withdrawn, but this is an artifact of the Timed Failures Model, which we believe can be overcome by adapting the model.

In this section we discuss a slight alternative to the untimed model, and then suggest some directions for future work.

6.1 On refusal testing

The untimed model of information flow in Section 3 was built around the Failures Model of CSP. The intuition in this model is that Low is unable to perform any events after observing a refusal. We now consider briefly how we could relax this assumption.

We extend our language of tests to allow Low to continue to offer events after observing a refusal:

$$T \in \mathit{Test} ::= \mathit{SUCCESS}(k) \mid (\Box_{a \in A}\, a \to T_a) \; \mathsf{else} \; T'.$$

The test $(\Box_{a \in A}\, a \to T_a)$ else T' offers the events from A; if one is performed, it then acts like the corresponding T_a; if all are refused, then it acts like T'.

In [7], Mukkaram introduced the Refusal Testing Model of CSP, influenced by similar work by Phillips [8]. This models a process by a set of refusal traces, where a refusal trace is an alternating sequence of events and refusals, where a refusal is either a set of events—representing that those events were refused—or a special value "•"—representing lack of refusal information.

We can associate a set of (refusal trace, result) pairs with tests of the above form:

$$\begin{aligned}
\mathcal{R}[\![\mathit{SUCCESS}(k)]\!] &\mathrel{\widehat=} \{(\langle \bullet \rangle, k)\}, \\
\mathcal{R}[\![(\Box_{a \in A}\, a \to T_a) \; \mathsf{else} \; T']\!] &\mathrel{\widehat=} \{(\langle \bullet, a \rangle ^\frown s, k) \mid a \in A \wedge (s, k) \in \mathcal{R}[\![T_a]\!]\} \\
&\phantom{\mathrel{\widehat=}} \cup \\
&\phantom{\mathrel{\widehat=}} \{(\langle R \sqcup A \rangle ^\frown s', k) \mid (\langle R \rangle ^\frown s', k) \in \mathcal{R}[\![T']\!]\},
\end{aligned}$$

where the $\sqcup$ operator forms the upper bound of two pieces of refusal information; if X and Y range over sets of events, then

$$\bullet \sqcup \bullet = \bullet, \qquad \bullet \sqcup X = X \sqcup \bullet = X, \qquad X \sqcup Y = X \cup Y.$$

We will write $\mathit{results}_R(S, T)$ for the set of results obtainable from refusal-testing-based test T with system S:

$$\mathit{results}_R(S, T) \mathrel{\widehat=} \{k \mid \exists\, s \in \mathcal{R}[\![S]\!] \bullet (s, k) \in \mathcal{R}[\![T]\!]\}.$$

We can then use this information to define the information flow quantity, as in the previous models.

For example, recall the following process from Example 5:

$$P_5 \mathrel{\widehat{=}} h1 \to l1 \to STOP \,\Box\, h2 \to (l1 \to STOP \,\Box\, l2 \to STOP).$$

This process had an information flow quantity of 2 in the Failures Model. However, consider the following strategy in the Refusal Testing Model:

$$\mathcal{Q}(0) = RUN(L), \qquad \mathcal{Q}(1) \mathrel{\widehat{=}} h1 \to RUN(L), \qquad \mathcal{Q}(2) \mathrel{\widehat{=}} h2 \to RUN(L),$$
$$T \mathrel{\widehat{=}} l2 \to SUCCESS(2) \text{ else } l1 \to SUCCESS(1) \text{ else } SUCCESS(0).$$

This strategy gives an information flow quantity of 3. Note in particular that $results_R(P, \mathcal{Q}(1), T) = \{1\}$, corresponding to the refusal trace $\langle \{l2\}, l1, \bullet \rangle$.

It is interesting that the choice between the Failures and Refusal Testing Models seems to make more of a difference to our results than it normally does in related areas.

6.2 Future work

As discussed above, we intend to investigate the consequences of adapting the Timed Failures Model to allow refusal tokens of the form $(t, t') \times A$; we need to ensure that the model, with this change, is still closed under the operators of the language. We will then attempt to prove the above conjecture, that lack of information flow in our model corresponds to Strong Timed Failures Non Deducibility on Composition.

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 interesting to ask what algebraic laws are satisfied by our definition. For example, we conjecture that the following laws hold:

$$\begin{aligned} IFQ(P1 \sqcap P2) &= \max(IFQ(P1), IFQ(P2)), \\ IFQ(P1 \mathbin{|||} P2) &= IFQ(P1) + IFQ(P2). \end{aligned}$$

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. It was started while I was employed by the University of Leicester.

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] 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`.

[7] Abida Mukkaram. *A Refusal Testing Model for CSP*. D.Phil, Oxford University, 1993.

[8] Iain Phillips. Refusal testing. *Theoretical Computer Science*, 1987.

[9] G. M. Reed and A. W. Roscoe. The timed failures-stability model for CSP. *Theoretical Computer Science*, pages 85–127, 1999.

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

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

[12] 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.

[13] Steve Schneider. An operational semantics for Timed CSP. *Information and Computing*, 116(2):193–213, 1995.

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

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

A A brief overview of CSP

A.1 Untimed syntax

In this section we give a brief overview of CSP. More details can be obtained from [11, 14].

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, $l.5$ is an event of channel l. Σ represents the set of all events.

In this paper we use processes defined using the following syntax:

$STOP$	process that can perform no events.
$a \rightarrow P$	process that can perform the event a, and then act like P.
$P \Box Q$	external choice; the process can act like either P or Q; the choice is made by the environment.
$\Box_{i \in I} P_i$	external choice between the processes P_i, where i ranges over I.
$P \sqcap Q$	internal or nondeterministic choice; the process can nondeterministically choose to act like either P or Q.
$\bigsqcap_{i \in I} P_i$	nondeterministic choice between the processes P_i, where i ranges over I.
$P \setminus A$	process that acts like P, except all events from the set A are hidden, i.e. made internal.
$RUN(A)$	process that can perform any sequence of events from A, and never refuse any events from A.
$CHAOS(A)$	the most nondeterministic, nondivergent process with alphabet A; the process can perform any sequence of events from A, and refuse any events.
$P \underset{A}{\parallel} Q$	parallel composition of P and Q, synchronising on events from A.
$P \mid\mid\mid Q$	interleaving of the processes P and Q; i.e. parallel composition with no synchronisation.

A.2 Untimed semantics

A *trace* is a sequence of events performed by a process. A *failure* is a pair (s, X), where s is a trace and X is a set of events known as a *refusal*. This represents that the process can perform the trace s, and then reach a stable state (i.e. from where no internal activity is possible) from which it can perform none of the events in X.

The Failures Model of CSP represents a process by the sets of its traces and failures. We write $\mathcal{T}[\![P]\!]$ for the traces of P, and $\mathcal{F}[\![P]\!]$ for the failures of P. We say that process P is failures equivalent to process Q if they have the same traces and failures; we say that P is refined by Q if the traces and failures of P are supersets of those of Q:

$$P \equiv_F Q \Leftrightarrow \mathcal{T}[\![P]\!] = \mathcal{T}[\![P]\!] \wedge \mathcal{F}[\![P]\!] = \mathcal{F}[\![Q]\!],$$
$$P \sqsubseteq_F Q \Leftrightarrow \mathcal{T}[\![P]\!] \supseteq \mathcal{T}[\![P]\!] \wedge \mathcal{F}[\![P]\!] \supseteq \mathcal{F}[\![Q]\!].$$

For non-divergent processes, the traces are uniquely defined by the failures:

$$\mathcal{T}[\![P]\!] = \{s \mid \exists X \bullet (s, X) \in \mathcal{F}[\![P]\!]\},$$

and so it is enough to consider just the failures.

A.3 Timed syntax

The syntax of Timed CSP is that of untimed CSP, extended as follows:

$a \xrightarrow{t} P$	process that can perform the event a at any time, and then act like P after a delay of t.
$a \rightarrow P$	shorthand for $a \xrightarrow{0} P$.
$WAIT\ t\,; P$	process that acts like P after a delay of t.
$P \overset{t}{\triangleright} Q$	process that acts like P, except if no event occurs within t time units, the process times-out and acts like Q.
$P \triangle_t Q$	process that acts like P for t time units, after which P is interrupted and the process then acts like Q.

A.4 Timed semantics

A *timed event* is a pair (t, a), representing the occurrence of event a at time t. A *timed trace* is a sequence of timed events in non-decreasing order of times. A *refusal token* is a set of timed events of the form $[t, t') \times A$. A *timed refusal* is a set of timed events formed as the union of a finite number of refusal tokens. A *timed failure* is a pair (s, X), where s is a timed trace and X is a timed refusal, representing the performance of the events in s, and the refusal to perform the events in X.

The Timed Failures Model represents a process P by the set of timed failures it can exhibit, written as $\mathcal{TF}[\![P]\!]$. Equivalence and refinement are defined analogously to as in the untimed model:

$$P \equiv_{TF} Q \Leftrightarrow \mathcal{TF}[\![P]\!] = \mathcal{TF}[\![Q]\!],$$
$$P \sqsubseteq_{TF} Q \Leftrightarrow \mathcal{TF}[\![P]\!] \supseteq \mathcal{TF}[\![Q]\!].$$

We use the following notation on failures in this paper: $X \upharpoonright A$ represents refusal X restricted to events from A; $\mathrm{end}(s, X)$ represents the supremum of times of events in s and X; $s + t$ and $X + t$ represent s and X temporally shifted, by adding t onto all times.