On information flow and refinement-closure

Gavin Lowe

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK
gavin.lowe@comlab.ox.ac.uk

Abstract. The question of information flow considers whether a high-level user of a multi-level security system can pass information to a low-level user. One family of information flow properties is *non-deducibility on compositions*: that for all possible high-level behaviours, the low-level user's view is the same. Unfortunately, this family suffers from the *refinement paradox*: that a process can be classified as secure, yet a refinement can be classified as insecure. In this paper we consider the property that classifies a process as secure if all of its refinements satisfy non-deducibility on compositions. This property correctly classifies all processes for which we have performed thought experiments. The property appears, at first sight, very difficult to test automatically, because of the quantifications over all high-level behaviours and all refinements. However, we prove that it is equivalent to an operational property, and hence derive a test that can be carried out using a model checker such as FDR. We also compare the property with existing properties. We show that it is stronger than Focardi and Gorrieri's strong bisimulation non-deducibility on compositions, but weaker than Roscoe's lazy independence property. Finally we show that the strength of the equivalence is independent of whether the low-level user's ability to distinguish processes is based upon stable failures or bisimulation.

1 Introduction

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

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

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

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

Several authors have attempted to formalise this notion of information flow using process algebras. Unfortunately, most previous attempts have produced definitions that appear to be either too weak (failing to identify certain sources of information flow) or too strong (identifying processes as insecure when there seems to be no way of using them to pass information). In this paper we consider a property that we believe overcomes these problems.

We will make use of CSP [Hoa85,Ros97] in this paper; a brief overview of the syntax and semantics is given in Appendix A. Throughout this paper we will assume that the system in question is modelled as a divergence-free CSP process. We partition the alphabet Σ into two sets, H and L, representing the interfaces of High and Low, ranged over by events h, h', etc., and l, l', etc. We let CSP_H be the set of all CSP processes with alphabet H, ranged over by Hi, Hi', etc.

One family of definitions of information flow is the pioneering work of Focardi and Gorrieri on *non-deducibility on compositions*. The idea is that however High acts, the system should appear the same to Low. This is equivalent to saying that for all CSP processes $Hi \in CSP_H$ modelling High's behaviour, $P \underset{H}{\parallel} Hi$ and $P \underset{H}{\parallel} Stop$ appear the same from Low's point of view. Focardi and Gorrieri have considered three ways of capturing Low's point of view, corresponding to different distinguishing powers of Low.

The simplest definition, *traces non-deducibility on compositions* [FG95], says that when H is hidden, the two systems should be traces-equivalent:

Definition 1 (TNDC). *Process P satisfies* traces non-deducibility on compositions*, written $TNDC(P)$, if*

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

For example, the process

$$Proc_1 \mathrel{\widehat{=}} h \rightarrow l \rightarrow Stop \,\Box\, l \rightarrow Stop$$

satisfies this definition, since $(Proc_1 \underset{H}{\parallel} Hi) \setminus H$ is equivalent to $l \rightarrow Stop$, for all high-level processes Hi. However, this definition is not suitable if Low can detect the refusal of events. For example, the process

$$Proc_2 \mathrel{\widehat{=}} h \rightarrow (l \rightarrow Stop \sqcap Stop) \,\Box\, l \rightarrow Stop$$

satisfies the property, because the processes $l \rightarrow Stop \sqcap Stop$ and $l \rightarrow Stop$ are traces equivalent; yet if Low detects that the event l is not available, he can deduce that High performed h, so the process should be considered insecure.

The property of bisimulation non-deducibility on compositions [FG95] seeks to overcome this problem by changing the equivalence to weak bisimulation (see Definition 8 in Appendix A).

Definition 2 (BNDC). *Process P satisfies* bisimulation non-deducibility on compositions*, written $BNDC(P)$, if*

$$\forall\, Hi \in CSP_H \bullet (P \underset{H}{\parallel} Hi) \setminus H \approx_B P \underset{H}{\parallel} Stop.$$

This definition gives the right result for both of the above examples. However, we shall see some problems with it below (processes $Proc_4$ and $Proc_5$).

In the rest of this paper we will concentrate on the stable failures model of CSP [Ros97]. Recall that in this model, a process is represented by its stable failures, i.e. pairs of the form (tr, X), indicating that the process can perform the trace tr to reach a stable state (where no internal events are possible) and where none of the events from X is available.

The fact that refusals are not recorded in unstable states means that in this model we cannot simply hide H as in the definitions of TNDC and BNDC: to do so would cause difficulties when Hi can perform an infinite sequence of consecutive events, such as in the process

$$Proc_3 \mathrel{\widehat{=}} h \rightarrow Proc_3 \,\Box\, l \rightarrow Proc_3.$$

High can pass information to Low only by performing an infinite sequence of h events. It is conventional to not consider such information flow, because it is reasonable to expect that High is not willing to do an infinite amount of work, or that the system is fair to Low. If we were to hide H, then this would suggest a flow of information: $Proc_3 \underset{H}{\parallel} Stop$ has the stable failure $(\langle\rangle, \{l'\})$ (for $l' \neq l$) whereas $(Proc_3 \underset{H}{\parallel} RUN(H)) \setminus H$ does not.

As argued by Roscoe in [Ros97, Chapter 12], it is best to capture Low's view of the system by considering the *lazy abstraction*:

$$\mathcal{L}_H(P) \mathrel{\widehat{=}} (P \underset{H}{\parallel} Chaos(H)) \setminus H.$$

Note that we have[1]

$$failures(\mathcal{L}_H(P)) = \{(tr \upharpoonright L, X) \mid (tr, X \cap L) \in failures(P)\}.$$

This leads to the following definition:

Definition 3 (FNDC). *Process P satisfies* failures non-deducibility on compositions*, written $FNDC(P)$, if:*[2]

$$\forall\, Hi \in CSP_H \bullet \mathcal{L}_H(P \underset{H}{\parallel} Hi) \equiv_F P \underset{H}{\parallel} Stop.$$

[1] $tr \upharpoonright L$ represents the restriction of trace tr to alphabet L.

[2] $\mathcal{L}_H(P \underset{H}{\parallel} Stop) = P \underset{H}{\parallel} Stop$, so we do not need to take the lazy abstraction of the right hand side.

(This is equivalent to, but defined in a different way from, Focardi's definition from [Foc96].)

The BNDC and FNDC properties correctly classify many processes, particularly deterministic ones. However, they can give misleading results when applied to some processes exhibiting nondeterminism. For example, consider the following process (from [For99,FRR00]):

$$Proc_4 \mathrel{\widehat{=}} (h_1 \rightarrow l_1 \rightarrow Proc_4 \,\Box\, h_2 \rightarrow l_2 \rightarrow Proc_4) \triangleright (l_1 \rightarrow Proc_4 \sqcap l_2 \rightarrow Proc_4).$$

This process satisfies both BNDC and FNDC: regardless of High's behaviour, Low's view of the system is that it repeatedly offers either l_1 or l_2. However, this process should be considered as insecure: High can pass an unbounded amount of information to Low by performing his events before the timeout; even if High can perform his events before the timeout only some of the time, this process acts as an unreliable channel from High to Low, which could be turned into a reliable channel using suitable error-correcting codes. In both NDC conditions the nondeterministic behaviour after the timeout obscures any information passable by the part of the process before the timeout.

Consider now

$$Proc_5 \mathrel{\widehat{=}} h \rightarrow (l_1 \rightarrow Stop \sqcap l_2 \rightarrow Stop) \,\Box\, (l_1 \rightarrow Stop \sqcap l_2 \rightarrow Stop).$$

This process satisfies both BNDC and FNDC: regardless of High's behaviour, the system acts like $l_1 \rightarrow Stop \sqcap l_2 \rightarrow Stop$ from Low's point of view. However, suppose the first nondeterministic choice is always resolved to the left, and the second nondeterministic choice is always resolved to the right; then the process acts like $h \rightarrow l_1 \rightarrow Stop \,\Box\, l_2 \rightarrow Stop$, which is clearly insecure.

In both of the above examples, the reason the NDC conditions give the wrong result is the nondeterminism. The NDC conditions suffer from the *refinement paradox*, namely that there are processes P that satisfy them, yet there are refinements Q of P that do not. Nondeterminism arises in models of systems for two main reasons:

- Often analysis is carried out upon *designs* of systems, rather than concrete implementations. In designs, nondeterminism often represents underspecification, which is resolved at a subsequent stage of the development. We should consider a design secure only if all ways of resolving the nondeterminism lead to secure implementations.
- Sometimes nondeterminism represents low-level details of a system that one chooses to abstract away from, e.g. scheduling. In such cases, one should consider a system secure only if all ways in which that nondeterminism could be resolved causes the system to behave securely.

We therefore argue that any reasonable definition of information flow should be closed under refinement. This leads us to the property that we consider throughout the remainder of this paper, namely the refinement-closure of FNDC:

Definition 4 (RCFNDC). *Process P satisfies* refinement-closed, failures non-deducibility on compositions*, written $RCFNDC(P)$, if*

$$\forall Q \sqsupseteq P \bullet FNDC(Q).$$

For example, the processes $Proc_4$ and $Proc_5$ both fail to satisfy RCFNDC, because they respectively have refinements

$$Q_4 \mathrel{\widehat{=}} (h_1 \rightarrow l_1 \rightarrow Q_4 \,\Box\, h_2 \rightarrow l_2 \rightarrow Q_4) \triangleright l_2 \rightarrow Q_4,$$
$$Q_5 \mathrel{\widehat{=}} h \rightarrow l_1 \rightarrow Stop \,\Box\, l_2 \rightarrow Stop.$$

which clearly both fail FNDC.

It is never possible to *prove* that a definition of information flow is correct: the best one can do is fail to find problems with it. The RCFNDC condition gives us the expected result for every thought experiment we have tried so far.

I introduced a property similar to RCFNDC in [Low02], albeit in the context of a discrete-time model of CSP. In that paper, I considered the *quantity* of information passed from High to Low; I showed that zero information flow reduced to a property similar to RCFNDC. However, I did not consider the property further there.

One apparent problem with RCFNDC is that it looks infeasible to test: it appears that we have to consider *every* refinement and *every* high-level behaviour. Somewhat surprisingly, this is not the case. In the next section we show that it is equivalent to an operationally-defined property; then in Section 3 we show how to produce a CSP refinement test —checkable using FDR— based upon the operational characterisation.

In Section 4 we compare RCFNDC with other information flow properties from the literature. We show that it lies between Focardi and Gorrieri's Strong Bisimulation Non-deducibility on Compositions [FG95], and Roscoe's Lazy Independence [Ros95]. We also show that the decision to base our property upon FNDC, as opposed to BNDC, makes no difference: the refinement-closure of BNDC gives us the same property.

2 Operational noninterference

In this section we define a property based on the operational semantics of a process, and prove that it is equivalent to RCFNDC.

We use standard operational semantics notation: see Appendix A. We write $inits(P)$ for the initial visible events of process P:

$$inits(P) \mathrel{\widehat{=}} \{a \mid a \in \Sigma \wedge P(\xrightarrow{\tau})^* \xrightarrow{a} \}.$$

We say that two states are initially L-equivalent if they have the same initial events from L:

$$P_1 \sim_L P_2 \mathrel{\widehat{=}} inits(P_1) \cap L = inits(P_2) \cap L.$$

We define our operational property to say that every state P_1 reached after a trace tr containing at least one high-level event should be initially L-equivalent to every state P_2 reached after the L-restriction of that trace (written $tr \upharpoonright L$):

Definition 5 (ONI). *We say that process P satisfies* operational noninterference*, written $ONI(P)$, if*

$$\forall\, P_1, P_2, tr \bullet P \stackrel{tr}{\Longrightarrow} P_1 \wedge P \stackrel{tr\upharpoonright L}{\Longrightarrow} P_2 \wedge tr \upharpoonright H \neq \langle\rangle \Rightarrow P_1 \sim_L P_2.$$

Theorem 1. $RCFNDC(P) \Leftrightarrow ONI(P)$.

For convenience, we split the proof into the following lemmas.

Lemma 1. *If $\neg RCFNDC(P)$ then $\neg ONI(P)$.*

Proof. Suppose

$$P \sqsubseteq Q \wedge Hi \in CSP_H \wedge \mathcal{L}_H(Q \underset{H}{\parallel} Hi) \not\equiv_F Q \underset{H}{\parallel} Stop.$$

We show that $\neg ONI(P)$. We consider two possibilities.

Case 1: $(tr, X) \in failures(\mathcal{L}_H(Q \underset{H}{\parallel} Hi)) - failures(Q \underset{H}{\parallel} Stop)$. Pick a minimal length such tr, so $tr \in traces(Q \underset{H}{\parallel} Stop)$. Both processes can refuse all events from H, so without loss of generality assume $X \subseteq L$.

Let tr' be the corresponding trace of $Q \underset{H}{\parallel} Hi$, so $tr' \upharpoonright L = tr$ and $(tr', X) \in failures(Q \underset{H}{\parallel} Hi)$. Then $(tr', X) \in failures(Q)$ since $X \subseteq L$. Now, $P \sqsubseteq Q$, so $(tr', X) \in failures(P)$. So for some P_1:

$$P \stackrel{tr'}{\Longrightarrow} P_1 \wedge P_1 \textbf{ ref } X.$$

We must have $tr' \upharpoonright H \neq \langle\rangle$ (i.e. $tr \neq tr'$): otherwise we would have $(tr, X) \in failures(Q \underset{H}{\parallel} Stop)$, giving a contradiction.

Now, $tr \in traces(Q \underset{H}{\parallel} Stop)$, but $(tr, X) \notin failures(Q \underset{H}{\parallel} Stop)$ so for some $l \in X$ (necessarily $l \in L$) we have $tr^\frown\langle l\rangle \in traces(Q \underset{H}{\parallel} Stop)$. Hence $tr^\frown\langle l\rangle \in traces(Q)$. But $P \sqsubseteq Q$, so $tr^\frown\langle l\rangle \in traces(P)$. So for some P_2:

$$P \stackrel{tr}{\Longrightarrow} P_2 \wedge l \in inits(P_2).$$

Hence $P_1 \not\sim_L P_2$, and so $\neg ONI(P)$, as required.

Case 2: $(tr, X) \in failures(Q \underset{H}{\parallel} Stop) - failures(\mathcal{L}_H(Q \underset{H}{\parallel} Hi))$. Pick a minimal length such tr, so $tr \in traces(\mathcal{L}_H(Q \underset{H}{\parallel} Hi))$. Both processes can refuse all events from H, so without loss of generality assume $X \subseteq L$.

We have $(tr, X) \in \mathit{failures}(Q)$; and $P \sqsubseteq Q$, so $(tr, X) \in \mathit{failures}(P)$. Then for some P_2:

$$P \stackrel{tr}{\Longrightarrow} P_2 \wedge P_2 \textbf{ ref } X.$$

Now, $tr \in \mathit{traces}(\mathcal{L}_H(Q \underset{H}{\parallel} Hi))$, but $(tr, X) \notin \mathit{failures}(\mathcal{L}_H(Q \underset{H}{\parallel} Hi))$, so for some $l \in X$ (necessarily $l \in L$) we have $tr^\frown\langle l\rangle \in \mathit{traces}(\mathcal{L}_H(Q \underset{H}{\parallel} Hi))$. Let $tr'^\frown\langle l\rangle$ be the corresponding trace of $Q \underset{H}{\parallel} Hi$, so $tr' \upharpoonright L = tr$. Then $tr'^\frown\langle l\rangle$ is also a trace of Q, and hence of P. So for some P_1:

$$P \stackrel{tr'}{\Longrightarrow} P_1 \wedge l \in \mathit{inits}(P_1).$$

Hence $P_1 \not\sim_L P_2$.

Finally, we must have $tr' \upharpoonright H \neq \langle\rangle$ (i.e. $tr \neq tr'$): otherwise we would have $(tr', X) \in \mathit{failures}(Q)$, and $(tr, X) \in \mathit{failures}(\mathcal{L}_H(Q \underset{H}{\parallel} Hi))$, giving a contradiction. Hence $\neg \mathit{ONI}(P)$, as required.

Lemma 2. *If $\neg \mathit{ONI}(P)$ then $\neg \mathit{RCFNDC}(P)$.*

Proof. By the assumptions of the lemma, there are P_1, P_2 and tr such that

$$P \stackrel{tr}{\Longrightarrow} P_1, \qquad P \stackrel{tr\upharpoonright L}{\Longrightarrow} P_2, \qquad tr \upharpoonright H \neq \langle\rangle, \qquad P_1 \not\sim_L P_2.$$

Let Hi be a process that performs $tr \upharpoonright H$. We do a case analysis on the cause of $P_1 \not\sim_L P_2$:

Case 1: $l \in \mathit{inits}(P_1) - \mathit{inits}(P_2)$ for some $l \in L$. Then $(tr \upharpoonright L, \{l\}) \in \mathit{failures}(P)$. Let

$$\begin{aligned} F_1 \mathrel{\widehat{=}} \mathit{failures}(P) &- \{(tr \upharpoonright L^\frown\langle l\rangle^\frown tr', X) \mid tr' \in \Sigma^*, X \subseteq \Sigma\} \\ &- \{(tr \upharpoonright L, X) \mid (tr \upharpoonright L, X \cup \{l\}) \notin \mathit{failures}(P)\}. \end{aligned}$$

In Appendix B we show that there is a process Q with $\mathit{failures}(Q) = F_1$.[3] This construction forces Q to behave like P, except any nondeterministic branch leading to the trace $tr \upharpoonright L^\frown\langle l\rangle$ is removed. Clearly $P \sqsubseteq Q$.

Now $tr^\frown\langle l\rangle \in \mathit{traces}(P)$, so by construction, $tr^\frown\langle l\rangle \in \mathit{traces}(Q)$ (since $tr \neq tr \upharpoonright L$), so $tr^\frown\langle l\rangle \in \mathit{traces}(Q \underset{H}{\parallel} Hi)$; hence

$$tr \upharpoonright L^\frown\langle l\rangle \in \mathit{traces}(\mathcal{L}_H(Q \underset{H}{\parallel} Hi)).$$

But $tr \upharpoonright L^\frown\langle l\rangle \notin \mathit{traces}(Q)$, so

$$tr \upharpoonright L^\frown\langle l\rangle \notin \mathit{traces}(Q \underset{H}{\parallel} \mathit{Stop}).$$

[3] We relegate this proof to the appendix, because it uses a technical result not relevant to the rest of this paper.

Hence $\mathcal{L}_H(Q \underset{H}{\parallel} Hi) \not\equiv_F Q \underset{H}{\parallel} Stop$, as required.

Case 2: $l \in inits(P_2) - inits(P_1)$ for some $l \in L$. Then $tr \upharpoonright L^\frown\langle l\rangle \in traces(P)$. Let

$$F_2 \mathrel{\widehat{=}} failures(P) - \{(tr \upharpoonright L, X) \mid l \in X \subseteq \Sigma\}.$$

In Appendix B we show that there is a process Q with $failures(Q) = F_2$. Clearly $P \sqsubseteq Q$.

Now, $(tr, \{l\}) \in failures(P)$, so by construction $(tr, \{l\}) \in failures(Q)$ (since $tr \neq tr \upharpoonright L$); and $(tr \upharpoonright H, \{\}) \in failures(Hi)$, so $(tr, \{l\}) \in failures(Q \underset{H}{\parallel} Hi)$; hence

$$(tr \upharpoonright L, \{l\}) \in failures(\mathcal{L}_H(Q \underset{H}{\parallel} Hi)).$$

But $(tr \upharpoonright L, \{l\}) \notin failures(Q)$, so

$$(tr \upharpoonright L, \{l\}) \notin failures(Q \underset{H}{\parallel} Stop).$$

Hence $\mathcal{L}_H(Q \underset{H}{\parallel} Hi) \not\equiv_F Q \underset{H}{\parallel} Stop$, as required.

3 Testing for *RCFNDC*

In this section we show how to test whether a process P satisfies $RCFNDC$ using the model checker FDR. Following Theorem 1, we need to test whether $ONI(P)$.

Our approach closely follows Roscoe's test for determinism from [Ros05], which is in turn based upon Lazić's test from [Laz99]. The idea is to run two copies of P within a testing harness, so that they evolve together to processes P_1 and P_2 such that for some trace tr

$$P \stackrel{tr}{\Longrightarrow} P_1 \wedge P \stackrel{tr\upharpoonright L}{\Longrightarrow} P_2.$$

Following the definition of ONI, in each such state, we need to check that if an event from H has occurred, then P_1 and P_2 can perform the same events from L.

Let $clunk$ be a new event, not in $H \cup L$. Define

$$\begin{aligned}
Clunk1 &= ?l : L \rightarrow clunk \rightarrow Clunk1 \;\Box\; ?h : H \rightarrow Clunk1,\\
Clunk2 &= ?l : L \rightarrow clunk \rightarrow Clunk2,\\
Repeat &= ?l : L \rightarrow l \rightarrow Repeat,\\
Harness(P) &= \left(\left((Clunk1 \underset{H\cup L}{\parallel} P) \underset{\{clunk\}}{\parallel} (Clunk2 \underset{H\cup L}{\parallel} P)\right) \underset{L}{\parallel} Repeat\right) \setminus \{clunk\}.
\end{aligned}$$

Within this harness, the two copies of P are forced to each perform the same L events; the left-hand P may also perform H events. The effect of the $Clunk1$ and $Clunk2$ processes, synchronising on $clunk$, is to ensure that each P performs

at most one L event more than the other; the effect of the *Repeat* process is to ensure that they perform the same event.

To see if $ONI(P)$ holds, we want to check that, once an H event has been performed, if one copy of P performs an event $l \in L$, then the other can also perform l. This can be achieved by testing whether $Spec1 \sqsubseteq Harness(P)$, where

$$\begin{aligned} Spec1 &= \$l : L \to Spec2(l) \sqcap \$h : H \to Spec3 \sqcap Stop, \\ Spec2(l) &= l \to Spec1 \sqcap \$h : H \to Spec4(l) \sqcap Stop, \\ Spec3 &= \$l : L \to Spec4(l) \sqcap \$h : H \to Spec3 \sqcap Stop, \\ Spec4(l) &= \$h : H \to Spec4(l) \triangleright l \to Spec3. \end{aligned}$$

In the states $Spec2(l)$ and $Spec4(l)$, the next L event will be l. The states $Spec3$ and $Spec4(l)$ are reached when there has been at least one H event. Note that in state $Spec2(l)$, the l might not be available: the process may perform an H event or deadlock. However, in the state $Spec4(l)$, the l must be available (although H events may also be available).

If P has N states, then $Harness(P)$ has $O(N^2)$ states, since it runs two copies of P. However, in most cases, for each state of the first copy of P, there will be a fairly small number of states that the second copy of P can be in at the same time; if this number is bounded by some constant k, then the total number of states is $O(k.N)$, so checking is linear in the size of P.

4 Comparisons

In this section we compare the strength of RCFNDC with other noninterference properties from the literature. The results are summarised in Fig. 1. We do not have enough space to describe all the properties of Fig. 1; we give references for those we do not define. We also show that the refinement-closures of most of the properties in Fig. 1 coincide, and consider the relationship with separability.

4.1 Comparison with strong BNDC

In [FG95], Focardi and Gorrieri introduced a property called *strong bisimulation non-deducibility on compositions*. Forster independently introduced it in [For99], and called it *strong local noninterference*.

Definition 6 (SBNDC). *P satisfies* strong bisimulation non-deducibility on compositions*, written $SBNDC(P)$, if, whenever $P \stackrel{tr}{\Longrightarrow} Q \stackrel{h}{\longrightarrow} R$ with $h \in H$, we have $Q \underset{H}{\parallel} Stop \approx_B R \underset{H}{\parallel} Stop$.*

Theorem 2. *For all processes P, $RCFNDC(P)$ implies $SBNDC(P)$.*

Proof. Suppose $RCFNDC(P)$. Following the definition of $SBNDC$, suppose $P \stackrel{tr}{\Longrightarrow} Q \stackrel{h}{\longrightarrow} R$. Let

$$\mathcal{B} \mathrel{\widehat{=}} \{(Q', R') \mid \exists\, tr' \bullet Q \underset{H}{\parallel} Stop \stackrel{tr'}{\Longrightarrow} Q' \wedge R \underset{H}{\parallel} Stop \stackrel{tr'}{\Longrightarrow} R'\}.$$

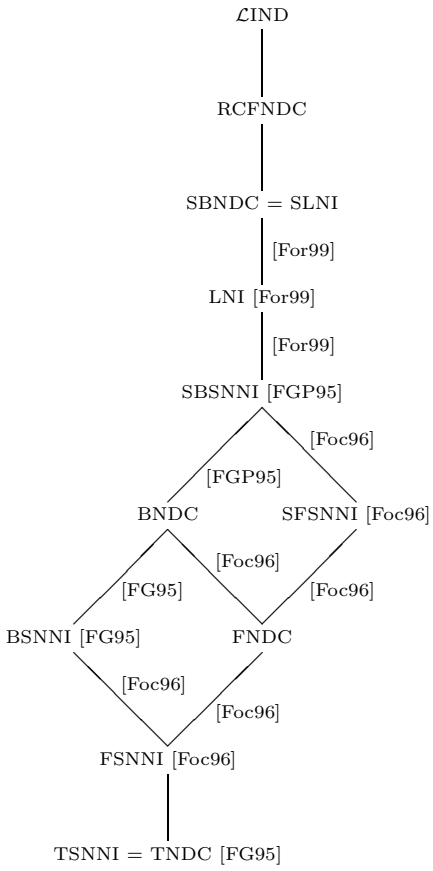

Fig. 1. The relative strengths of the properties.

We will show that $\mathcal{B}$ is a bisimulation. Since $(Q \underset{H}{\parallel} Stop, R \underset{H}{\parallel} Stop) \in \mathcal{B}$, this will show that $Q \underset{H}{\parallel} Stop \approx_B R \underset{H}{\parallel} Stop$, and we will be done.

So, pick $(Q', R') \in \mathcal{B}$, and let tr' be the corresponding trace (as in the definition of $\mathcal{B}$), so $P \stackrel{tr^\frown tr'}{\Longrightarrow} Q'$, $P \stackrel{tr^\frown\langle h\rangle^\frown tr'}{\Longrightarrow} R'$, and $tr' \upharpoonright H = \langle\rangle$. We claim that there is a process $\hat{P}$ such that $P \stackrel{(tr^\frown tr')\upharpoonright L}{\Longrightarrow} \hat{P} \wedge \hat{P} \sim_L Q'$. If $tr\upharpoonright H = \langle\rangle$ then we can simply take $\hat{P} = Q'$. Otherwise, $tr^\frown tr' \in traces(P)$; and P satisfies RCFNDC and hence TNDC, and so $(tr^\frown tr') \upharpoonright L \in traces(P)$. Let $\hat{P}$ be any process such that $P \stackrel{(tr^\frown tr')\upharpoonright L}{\Longrightarrow} \hat{P}$; then the ONI characterisation of RCFNDC gives us that $\hat{P} \sim_L Q'$. Now, $P \stackrel{tr^\frown\langle h\rangle^\frown tr'}{\Longrightarrow} R'$, and $(tr^\frown\langle h\rangle^\frown tr') \upharpoonright L = (tr^\frown tr') \upharpoonright L$, so we have that $\hat{P} \sim_L R'$, again by the ONI characterisation of RCFNDC. Hence, by transitivity of $\sim_L$, we have $Q' \sim_L R'$.

Following the definition of bisimulation, suppose $Q' \xrightarrow{l} Q''$ with $l \in L$. Then by definition of $\sim_L$, $l \in inits(R') \cap L$, so there is some R'' such that $R' \stackrel{\langle l \rangle}{\Longrightarrow} R''$. But then

$$Q \underset{H}{\parallel} Stop \stackrel{tr'^\frown\langle l \rangle}{\Longrightarrow} Q'' \quad \text{and} \quad R \underset{H}{\parallel} Stop \stackrel{tr'^\frown\langle l \rangle}{\Longrightarrow} R''$$

so $(Q'', R'') \in \mathcal{B}$. Likewise, if $Q' \xrightarrow{\tau} Q''$, then taking $R'' = R'$, we have $R'(\xrightarrow{\tau})^* R''$, and $(Q'', R'') \in \mathcal{B}$. The vice versa conditions are identical.

RCFNDC is strictly stronger than SBNDC, as shown by $Proc_5$ from the Introduction:

$$Proc_5 \mathrel{\widehat{=}} h \rightarrow (l1 \rightarrow Stop \sqcap l2 \rightarrow Stop) \,\Box\, (l1 \rightarrow Stop \sqcap l2 \rightarrow Stop).$$

As argued earlier, this does not satisfy RCFNDC. However, it is easy to see that it satisfies SBNDC.

4.2 Comparison with lazy independence

Roscoe introduced the notion of *lazy independence* in [Ros95]; the form we give here is from [Ros97, Section 12.4].

Definition 7 (Lazy independence). *A process P is lazily independent, written $\mathcal{L}IND(P)$, if $\mathcal{L}_H(P)$ is deterministic.*

The following lemma relates lazy independence to an operational property, similar in style to ONI.

Lemma 3. *If $\mathcal{L}IND(P)$ then*

$$\forall\, P_1, P_2, tr_1, tr_2 \bullet P \stackrel{tr_1}{\Longrightarrow} P_1 \wedge P \stackrel{tr_2}{\Longrightarrow} P_2 \wedge tr_1 \upharpoonright L = tr_2 \upharpoonright L \Rightarrow P_1 \sim_L P_2.$$

Proof. We prove the contra-positive. Suppose

$$P \stackrel{tr_1}{\Longrightarrow} P_1 \wedge P \stackrel{tr_2}{\Longrightarrow} P_2 \wedge tr_1 \upharpoonright L = tr_2 \upharpoonright L \wedge P_1 \not\sim_L P_2.$$

Then, without loss of generality there is some $l \in L$ such that $l \in inits(P_1)$ and $l \notin inits(P_2)$. But then $tr_1 \upharpoonright L^\frown\langle l \rangle \in traces(\mathcal{L}_H(P))$ and $(tr_1 \upharpoonright L, \{l\}) \in failures(\mathcal{L}_H(P))$. So $\mathcal{L}_H(P)$ is nondeterministic.

Theorem 3. *$\mathcal{L}IND(P)$ implies $RCFNDC(P)$.*

Proof. The theorem follows immediately from the previous lemma, noting that the property there implies ONI.

The property $\mathcal{L}IND$ is strictly stronger than RCFNDC, since the former is failed by processes that exhibit nondeterminism to Low, even when that nondeterminism cannot be affected by High's behaviour, such as $l1 \rightarrow Stop \sqcap l2 \rightarrow Stop$.

Theorem 3 and the operational property identified in Lemma 3 help to shed light on the difference between the properties: lazy independence fails processes whenever Low's view is nondeterministic; RCFNDC fails processes whenever Low's view is nondeterministic with High performing an event on one of the traces leading to that nondeterminism.

4.3 Refinement-closure of other properties

We now show that the refinement-closures of most of the properties from Fig. 1 agree with RCFNDC. We can define the refinement-closure of property ϕ as follows:

$$RC(\phi)(P) \mathrel{\widehat{=}} \forall Q \sqsupseteq P \bullet \phi(Q).$$

So, in particular $RCFNDC = RC(FNDC)$.

Theorem 4. *If ϕ is any property such that $RCFNDC \Rightarrow \phi \Rightarrow FNDC$, then $RC(\phi) \equiv RCFNDC$.*

Proof. Note that RC is monotonic: $\phi \Rightarrow \psi$ implies $RC(\phi) \Rightarrow RC(\psi)$. Hence we have

$$RCFNDC \equiv RC(RCFNDC) \Rightarrow RC(\phi) \Rightarrow RC(FNDC) \equiv RCFNDC.$$

In particular, this means that the refinement-closure of BNDC coincides with RCFNDC. This is important as it means that it doesn't matter whether Low is able to distinguish processes on the basis of bisimulation or failures: the same processes are secure in either case.

The refinement-closure of BNDC looks like a slightly odd property, since it combines failures and bisimulation semantics, which do not normally sit well together. However, the two semantics are used for different purposes in the definition: the failures semantics is used to describe the refinement of a system's design or abstract model to an implementation; and the bisimulation semantics is used to describe the interaction of Low with that final implementation; it is not unreasonable that different semantics should be appropriate for these two purposes. One could consider a number of related properties, with both different notions of refinement and different notions of Low's distinguishing power: it would be interesting to ask whether these vary in strength.

4.4 Separability

A process P is said to be *separable* if it is equivalent to $Hi \mid\mid\mid Lo$ where Hi and Lo have alphabets H and L, respectively. Separability does not imply RCFNDC. For example,

$$Proc_6 \mathrel{\widehat{=}} (l1 \rightarrow Stop \sqcap l2 \rightarrow Stop) \mid\mid\mid h \rightarrow Stop$$

does not satisfy RCFNDC, since it is refined by $l1 \rightarrow h \rightarrow Stop \mathbin{\Box} h \rightarrow l2 \rightarrow Stop$, which clearly doesn't satisfy FNDC.

Whether $Proc_6$ should be considered secure depends upon how one views a CSP model of a process. If the process really is built as an interleaving, as above, then it is reasonable to suppose that there is no flow of information. However, the normal philosophy is that a CSP model captures the allowable behaviours of the process, rather than necessarily describing how it is constructed; in such a case,

if the process could be constructed in an insecure way, it should be considered insecure.

It is well known that separability is a rather strong property, since it is symmetric in H and L. Therefore there are many processes that are not separable, yet should be considered secure: a simple example is $l \rightarrow h \rightarrow Stop$, which is not separable, yet satisfies all the properties of Fig. 1.

5 Related work

In [Man01], Mantel considers the same problem as us. However, rather than considering an information flow property that is closed under refinement, he considers how to restrict the refinement relation such that information flow properties are preserved. In particular, he produces a restriction of trace refinement that preserves the so-called perfect security property [ZL97].

Bossi et al. [BFPR03] and Alur et al. [ACZ06] take similar approaches. They each give sufficient conditions for particular refinements (using different notions of refinement from that in the current paper) to preserve various information flow properties based on NDC and secrecy of properties of runs, respectively.

Acknowledgements

I would like to thank Bill Roscoe, Michael Goldsmith, Tomasz Mazur, Toby Murray and the anonymous referees for useful comments on this work.

References

[ACZ06] Rajeev Alur, Pavol Cerny, and Steve Zdancewic. Preserving secrecy under refinement. In *33rd International Colloquium on Automata, Languages, and Programming*, 2006.

[BFPR03] Annalisa Bossi, Riccardo Focardi, Carla Piazza, and Sabina Rossi. Refinement operators and information flow security. In *Proccedings of the IEEE International Conference on Software Engineering and Formal Methods*, pages 44–53, 2003.

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

[FGP95] R. Forcardi, R. Gorrieri, and V. Panini. The security checker: A semantics-based tool for the verification of security properties. In *Proceedings of the 8th IEEE Computer Security Foundations Workshop*, pages 60–69, 1995.

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

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

[FRR00] R. Forster, G. M. Reed, and A. W. Roscoe. *Millennial Perspectives in Computer Science*, chapter The successes and failures of behavioural models. Palgrave, 2000.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.
[Laz99] Ranko Lazić. *A Semantic Study of Data Independence with Applications to Model Checking*. D.Phil., Oxford University, 1999.
[Low02] Gavin Lowe. Quantifying information flow. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, pages 18–31, 2002.
[Man01] Heiko Mantel. Preserving information flow properties under refinement. In *Proccedings of the IEEE Symposium on Security and Privacy*, pages 78–91, 2001.
[Ros95] A. W. Roscoe. CSP and determinism in security modelling. In *Proceedings of 1995 IEEE Symposium on Security and Privacy*, 1995.
[Ros97] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1997.
[Ros05] A. W. Roscoe. On the expressive power of CSP refinement. *Formal Aspects of Computing*, 17(2):93–112, 2005.
[ZL97] Aris Zakinthinos and E. S. Lee. A general theory of security properties. In *Proccedings of the IEEE Symposium on Security and Privacy*, pages 94–102, 1997.

A A brief overview of CSP

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

Syntax An event represents an atomic communication; this might either be between two processes or between a process and the environment. We write Σ for the set of all visible events. The event τ represents an internal event; we define $\Sigma^\tau \mathrel{\widehat{=}} \Sigma \cup \{\tau\}$.

The process $Stop$ can perform no events. The process $a \rightarrow P$ can perform the event a, and then act like P. The process $?a : A \rightarrow P_a$ offers the set of events A; if a particular event a is performed, the process then acts like P_a. (The prefixing operator "$\rightarrow$" binds tighter than all other operators.)

The process $P \,\Box\, Q$ represents an external choice between P and Q; the initial events of both processes are offered to the environment; when an event is performed, that resolves the choice. $P \sqcap Q$ 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. $\$a : A \rightarrow P_a$ is the process that nondeterministically chooses an event a from A, performs it, and then acts like P_a. The process $P \triangleright Q$ acts like a timeout: it initially acts like P, but if no event of P is performed then a timeout occurs and the process acts like Q.

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. $P \,|||\, Q$ represents an interleaving of the processes P and Q; i.e. parallel

composition without any synchronisation. $P \setminus A$ acts like P, except all events from the set A are hidden, i.e. made internal.

Operational semantics We use standard notation for operational semantics. We write $P \xrightarrow{a} Q$ to represent that P can perform the event $a \in \Sigma^\tau$ and become Q. We write $P \stackrel{tr}{\Longrightarrow} Q$ to represent that P can perform the trace $tr \in \Sigma^*$ of visible events and become Q. We write P **ref** X to indicate that P can refuse X: it is a stable state, i.e. where no internal transitions are possible, and it cannot perform any event from X.

We remind the reader of the definition of a weak bisimulation:

Definition 8 (Weak bisimulation). *A* weak bisimulation *is a binary relation $\mathcal{B}$ such that for all $(P, Q) \in \mathcal{B}$:*

- *If $P \xrightarrow{x} P'$ with $x \in \Sigma$, then $\exists\, Q' \bullet Q \stackrel{\langle x \rangle}{\Longrightarrow} Q' \wedge (P', Q') \in \mathcal{B}$;*
- *If $P \xrightarrow{\tau} P'$, then $\exists\, Q' \bullet Q(\xrightarrow{\tau})^* Q' \wedge (P', Q') \in \mathcal{B}$;*
- *If $Q \xrightarrow{x} Q'$ with $x \in \Sigma$, then $\exists\, P' \bullet P \stackrel{\langle x \rangle}{\Longrightarrow} P' \wedge (P', Q') \in \mathcal{B}$;*
- *If $Q \xrightarrow{\tau} Q'$, then $\exists\, P' \bullet P(\xrightarrow{\tau})^* P' \wedge (P', Q') \in \mathcal{B}$.*

We say that P and Q are bisimilar*, written $P \approx_B Q$, if there exist a weak bisimulation that contains (P, Q).*

Denotational semantics A trace of a process is a sequence of visible events that a process can perform. A stable failure of a process is a pair $(tr, X) \in \Sigma^* \times \mathbb{P}\,\Sigma$, such that the process can perform the trace tr and then refuse X, i.e., for some P', $P \stackrel{tr}{\Longrightarrow} P' \wedge P'$ **ref** X. The stable failures model represents a process by its traces T and stable failures F. For divergence-free processes, they are related by $T = \mathit{traces}(F) = \{tr \mid (tr, X) \in F\}$. T and F satisfy the following axioms:

F1. T is non-empty and prefix-closed.
F2. $(s, X) \in F \wedge Y \subseteq X \Rightarrow (s, Y) \in F$.
F3. $(s, X) \in F \wedge s^\frown\langle a \rangle \notin \mathit{traces}(P) \Rightarrow (s, X \cup \{a\}) \in F$.

The traces and stable failures of a process can either be extracted from the operational semantics, or calculated using rules for each operator. We need the following two rules in this paper: if the alphabet of Q is a subset of A, then

$$\mathit{failures}(P \underset{A}{\parallel} Q) = \{(tr, X \cup Y) \mid (tr, X) \in \mathit{failures}(P) \wedge (tr \upharpoonright A, Y) \in \mathit{failures}(Q)\};$$

and

$$\mathit{failures}(P \setminus A) = \{(tr \upharpoonright (\Sigma - A), X) \mid (tr, X \cup A) \in \mathit{failures}(P)\}.$$

We say that P is refined by Q, written $P \sqsubseteq Q$ if $\mathit{traces}(P) \supseteq \mathit{traces}(Q) \wedge \mathit{failures}(P) \supseteq \mathit{failures}(Q)$.

B Proofs from Section 2

In this appendix we show that processes exist that have the failures F_1 and F_2, as defined in Lemma 2. We will need the following result from [Ros97, Section 9.3]:

Theorem 5. *For any choice of $F \in \mathbb{P}\,\Sigma$ such that the axioms F1, F2, F3 (see Appendix A) are satisfied, there is a CSP process Q such that $failures(Q) = F$.*

Hence it will be enough to show that F_1 and F_2 satisfy the axioms.

Lemma 4. *F_1 and F_2, as defined in Lemma 2, satisfy the axioms of the stable failures model.*

Proof. Recall that $failures(P)$ satisfies the axioms.

Consider first F_1. Note that

$$traces(F_1) = traces(P) - \{(tr \restriction L^\frown\langle l\rangle^\frown tr' \mid tr' \in \Sigma^*\}.$$

We prove each axiom in turn.

- F1 is satisfied since $traces(P)$ satisfies it, and the set of traces removed is postfix-closed.
- F2 is satisfied since $failures(P)$ satisfies it, and the refusals removed are superset closed.
- For F3, suppose $(s, X) \in F_1$ and $s^\frown\langle a\rangle \notin traces(F_1)$.
  If $s = tr \restriction L$ and $a = l$, then by construction of F_1 we have $(s, X \cup \{l\}) \in failures(P)$; and so $(s, X \cup \{l\}) \in F_1$, again by construction.
  In all other cases, we have $s^\frown\langle a\rangle \notin traces(P)$, so $(s, X \cup \{a\}) \in failures(P)$, since $failures(P)$ satisfies F3. If $s \neq tr \restriction L$ then $(s, X \cup \{a\}) \in F_1$ by construction. If $s = tr \restriction L$ then from $(s, X) \in F_1$ and the construction of F_1 we have $(s, X \cup \{l\}) \in failures(P)$; so $(s, X \cup \{a, l\}) \in failures(P)$, again since $failures(P)$ satisfies F3; hence $(s, X \cup \{a\}) \in F_1$ by construction.

Now consider F_2. Recall that $tr \restriction L^\frown\langle l\rangle \in traces(P)$. Note that $traces(F_2) = traces(P)$.

- F1 is satisfied since $traces(P)$ satisfies it.
- F2 is satisfied since $failures(P)$ satisfies it, and the refusals removed are superset closed.
- For F3, suppose $(s, X) \in F_2$ and $s^\frown\langle a\rangle \notin traces(F_2) = traces(P)$. Then $(s, X) \in failures(P)$ and so $(s, X \cup \{a\}) \in failures(P)$ since $failures(P)$ satisfies F3.
  If $s \neq tr \restriction L$ then $(s, X \cup \{a\}) \in F_2$ by construction.
  If $s = tr \restriction L$ then $a \neq l$ since $tr \restriction L^\frown\langle l\rangle \in traces(P)$. Also, since $(s, X) \in F_2$ we have $l \notin X$ by construction of F_2. Hence $l \notin X \cup \{a\}$ so $(s, X \cup \{a\}) \in F_2$, by construction.