Non-Interference for Deterministic Interactive Programs

David Clark[1] and Sebastian Hunt[2]

[1] King's College London
david.j.clark@kcl.ac.uk
[2] City University, London
seb@soi.city.ac.uk

Abstract. We consider the problem of defining an appropriate notion of non-interference (NI) for deterministic interactive programs. Previous work on the security of interactive programs by O'Neill, Clarkson and Chong (CSFW 2006) builds on earlier ideas due to Wittbold and Johnson (Symposium on Security and Privacy 1990), and argues for a notion of NI defined in terms of strategies modelling the behaviour of users. We show that, for deterministic interactive programs, it is not necessary to consider strategies and that a simple stream model of the users' behaviour is sufficient. The key technical result is that, for deterministic programs, stream-based NI implies the apparently more general strategy-based NI (in fact we consider a wider class of strategies than those of O'Neill et al). We give our results in terms of a simple notion of Input-Output Labelled Transition System, thus allowing application of the results to a large class of deterministic interactive programming languages.

1 Introduction

We consider the problem of defining an appropriate notion of non-interference (NI) [8] for deterministic interactive programs. By interactive programs we mean programs which perform channel-based IO, reading and writing primitive values on named channels over time, as the system executes, in contrast to the simple "batch-processing" style of computation assumed by much of the work in language-based security. Moving away from the simple batch-processing model introduces a number of complications and subtleties. Even so, in this paper we show that a relatively simple stream-based model of interaction may be adequate for the special (but common) case of deterministic programs.

Previous work on the security of interactive programs by O'Neill, Clarkson and Chong [14] builds on earlier ideas due to Wittbold and Johnson [16], and argues for a notion of NI defined in terms of strategies modelling the behaviour of users. We show that, for deterministic interactive programs, it is not necessary to consider strategies and that a simple stream model of the users' behaviour is sufficient. The key technical result is that, for deterministic programs, stream-based NI implies the apparently more general strategy-based NI (in fact we consider a wider class of strategies than those of O'Neill et al). We give our results in terms of a simple notion of Input-Output Labelled Transition System, thus allowing application of the results to a large class of deterministic interactive programming languages.

P. Degano, J. Guttman, and F. Martinelli (Eds.): FAST 2008, LNCS 5491, pp. 50–66, 2009.
© Springer-Verlag Berlin Heidelberg 2009

2 Overview

We start by considering some motivating examples. These examples show that interactive programs may enable quite subtle covert channels, in which attackers can exploit information gained from previous outputs in order to leak information via later outputs. Moreover the examples show that a simple stream-based model of user behaviour may not suffice to reveal the presence of such channels. We also argue that such channels are significant even outside the typical "military" scenario in which an insider collaborates to send secrets to an outsider, by showing how a trusted user may be duped into sending information on such channels without knowing it.

Following [16] and [14] we then show how such channels can be guarded against by requiring a more sophisticated notion of NI, one defined in terms of user *strategies* rather than input streams.

It is striking that the example covert channels mentioned above all involve the *combination* of interaction and internal nondeterminism. The main result of this paper is to show that this is not accidental: for purely deterministic systems, such covert channels do not arise. We formalise this by showing how input streams can be represented as a special class of strategy and then showing that defining NI over this restricted class of strategies is equivalent to the more general notion when the system (though not necessarily the environment) behaves deterministically. Rather than tie our results to a specific programming language, we define a simple notion of Input-Output Labelled Transition System (IOLTS) and sate our definitions and results for any IOLTS. To illustrate how programming languages can be modelled in such a setting, we give an IOLTS semantics for a simple deterministic interactive language.

We conclude with a discussion of the scope and limitations of the chosen definition of NI and, more generally, of the use of strategies to model a program's environment.

3 Information Flow in Interactive Programs

We start with two simple examples of interactive programs illustrating ways in which such programs may be insecure.

The first program is insecure because there is a direct flow from High input to Low output:

```
x := 0;
input y from H;
output (x XOR y) to L;
```

The second program is an example of indirect flow from High to Low. If we consider that Low and High are feeding a stream of inputs to the program, information about the High stream can be deduced from the way in which the program is consuming Low's inputs:

```
input x from H;
if (x = 0) then input y from L;
input z from L;
output z to L;
```

For example, suppose the Low input stream starts 01. Then the Low output will be 1 if High inputs 0, otherwise the Low output will be 0.

3.1 Two Approaches to Defining Security

Users interact with the programs via input and output on named channels each of which is associated with a security level in a Denning-style multi-level security classification system, whereby security levels form a lattice, $\langle L, \sqsubseteq \rangle$, $L = \{a, b, c, \ldots\}$ [6]. For simplicity's sake we identify a channel's name with its security level. We write $\downarrow a$ for the set of channels visible to users at level a, ie $\downarrow a = \{a' \in L | a' \sqsubseteq a\}$.

Consider two users with access to channels a and b respectively where $a \not\sqsubseteq b$, i.e. the security policy specifies that no information should flow from channel a to channel b. We might reasonably try to capture this requirement in two alternative ways:

All outputs on channel b are consistent with all possible inputs on channel a. (1)

Users of channel a cannot send messages to the users of channel b. (2)

These both seem reasonable, but are they equivalent? First, observe that 2 $\Rightarrow$ 1: if some outputs seen by Bob are inconsistent with some possible inputs from Alice, then Bob can deduce something about the values input by Alice so Alice clearly *can* send messages to Bob, hence (by contraposition) 2 implies 1. At first sight it seems as though 1 $\Rightarrow$ 2 should also hold. After all, if what Bob sees tells him nothing about what Alice has input, surely she cannot send him a message.

In fact, as Wittbold and Johnson show in [16], this reasoning is unsound: some systems which satisfy property 1 allow Alice to send Bob messages. Let $L \sqsubseteq H$ and suppose that the only values which may be sent on these channels are 0 and 1. Consider **Program 1a:**

```
while (true) do
    x := 0 | 1;
    input y from H;
    output x to H;
    output (x XOR y) to L;
```

Here | is a non-deterministic choice operator, so `0 | 1` evaluates to either 0 or 1, with the choice being made in a way which is unpredictable to any observer of the running program. Writing output of value v on channel a as $a!v$ and input as $a?v$, the possible traces for the first iteration of the loop are:

```
H?0 H!0 L!0
H?0 H!1 L!1
H?1 H!0 L!1
H?1 H!1 L!0
```

Observation of the first L-output thus reveals nothing to L-users about the value of the first H-input: L-users cannot observe the H-outputs and hence, whether L-users see 0 or 1, both 0 and 1 are possible values for the input on H. This clearly holds for longer traces as well: no matter how much of the stream of L-outputs is observed, nothing is learned about which values have been input on H. Program 1a thus satisfies property 1 and, indeed, would seem to be a secure program.

Now consider the variant **Program 1b:**

```
while (true) do
    x := 0 | 1;
    output x to H;
    input y from H;
    output (x XOR y) to L;
```

In this example, the value of x is output *before* the H-input is demanded. The possible traces for the first iteration of this variant are:

```
H!0 H?0 L!0
H!0 H?1 L!1
H!1 H?0 L!1
H!1 H?1 L!0
```

It clearly remains the case that, in ignorance of the value of x, any observation of an output on channel L is consistent with both possible inputs on H, and thus property 1 holds also for Program 1b. Crucially, though, with Program 1b, H-users can exploit their knowledge of x to *control* what is output on L. This allows H-users to send messages to L-users, thus violating property 2. For example, if an H-user wants to send a particular message to L, say $x_1 \ldots x_n$, behaving as follows will suffice:

```
for  (i = 1 to n) do
 input  k from H;
 output (k XOR xi) to H;
```

When composed with Program 1b, this behaviour results in the message $x_1 \ldots x_n$ being delivered on L without error.

Program 1b first appears in this form in O'Neill, Clarkson and Chong's paper [14]. This was an adaptation of a synchronous nondeterministic state machine used by Wittbold and Johnson [16] to illustrate the same phenomenon. (It is interesting to note that, in state machine form, the example actually appears much earlier in a paper by Shannon [15]. In this paper Shannon showed how, in certain cases, making "side information" available at the transmitting point may increase the capacity of a communication channel.)

Using Program 1b, an H-user is able deliberately to communicate secrets to L-users. But, even when a user does not *intend* to leak a secret, such covert channels can still pose a security risk, since one user's "cooperation" with another may be unwitting. Suppose we have two users, Alice and Bob, at incomparable security levels A and B, respectively. The following example is originally due to David Sands [11].

Alice is interacting with a web site. Alice is assured by the site that her credit card details are never sent to Bob, and this assurance is backed up by a proof of property 1. The web site requests Alice to input her credit card and then offers her a "special offer" code, inviting her to input this code at a later time to obtain a discount or free gift. Unbeknownst to Alice, this code is actually her own credit card number in encrypted form. If Alice does enter the code when requested, the system simply decrypts it and sends it to Bob. In simplified form (a boolean credit card number!), this may be coded as **Program 2**:

```
input  x from A;
k := 0 | 1;
output (k XOR x) on A;
.
.
.
input y from A;
output (k XOR y) on B;
```

The possible traces for this system are:

```
A?0 A!0 A?0 B!0   (*)
A?0 A!0 A?1 B!1
A?0 A!1 A?0 B!1
A?0 A!1 A?1 B!0   (*)
A?1 A!0 A?0 B!1   (*)
A?1 A!0 A?1 B!0
A?1 A!1 A?0 B!0
A?1 A!1 A?1 B!1   (*)
```

Now, since Bob cannot see channel A, both outputs 0 and 1 are consistent with all four possible input sequences by Alice, hence property 1 is satisfied. Clearly, though, if Alice behaves as expected - the traces marked (*) - her credit card number is leaked to Bob.

These examples illustrate that a simple security property based on consistency of one user's observed outputs with another user's possible inputs may not be adequate to provide desirable security guarantees. In particular, it seems that the problem with property 1 is that it fails to take account of the *interactive* nature of such systems, whereby a user's inputs may depend on previously seen outputs. Wittbold and Johnson [16] proposed instead a property stated in terms of consistency of observed outputs with user's *behaviours*, modelling behaviours as the *strategies* by which users provide inputs based on their observations of the system so far.

This use of a strategy-based security property is very elegant and is successful in accepting Program 1a while rejecting Program 1b and Program 2. On the other hand, it is also technically less straightforward than a security property based simply on the input and output streams of a program. It is striking that the examples above involve the *combination* of interactivity and non-determinism. In this paper we consider the (very common) sub-class of *deterministic* interactive systems and show that for this sub-class, stream-based and strategy-based security properties are actually equivalent. The intuition is that, for a deterministic program, a sufficiently high security user can, in principle, choose inputs and predict all outputs statically. Thus there should be no need to model dynamic behaviours of users in order to verify the security property.

3.2 Input-Output Labelled Transition Systems (IOLTS)

As illustrated by the examples above, we are interested in security properties of programs written in languages with input and output primitives. However, our treatment is not specific to a given language. Instead we express security properties at the level of input/output traces.

Definition 1. *An Input-Output Labelled Transition System (IOLTS) is an* input-neutral *labelled transition system with a set of labels given by*

$$A ::= \tau \mid a?v \mid a!v$$

where $a \in L$ *and* $v \in \mathbb{V}$ *(where* $\mathbb{V}$ *is some unspecified non-empty set of possible values). By* input-neutral *we mean that branching on inputs is never restricted for a state in which input is possible, i.e. for a state* s *of the LTS: if* $\exists v.s \xrightarrow{a?v}$ *then* $\forall v.s \xrightarrow{a?v}$.

Let s range over the states of IOLTSs. Let Tr denote the set of all possible IOLTS traces: $\mathrm{Tr} = A^*$. Let t, u range over Tr. For $t = \ell_1 \cdots \ell_n \in \mathrm{Tr}$ we write $s \overset{t}{\twoheadrightarrow} s'$ to mean that there exist states $s_1, \ldots, s_n$ such that $s \xrightarrow{\ell_1} s_1 \cdots \xrightarrow{\ell_n} s_n = s'$. We write $s \overset{t}{\twoheadrightarrow}$ to mean that there exists s' such that $s \overset{t}{\twoheadrightarrow} s'$.

Definition 2. *An IOLTS is* deterministic *iff:*

1. *If* $s \xrightarrow{\ell_1} s_1$ *and* $s \xrightarrow{\ell_2} s_2$ *and* $\ell_1 \neq \ell_2$ *then* $\ell_1 = a?v_1$ *and* $\ell_2 = a?v_2$*, for some channel* a *and values* v_1, v_2.
2. *If* $s \xrightarrow{\ell} s_1$ *and* $s \xrightarrow{\ell} s_2$ *then* $s_1 = s_2$.

3.3 IOLTS Example: A Simple Interactive Imperative Language

The simple interactive imperative language used for the examples above is essentially the same language defined in [14]. To demonstrate one possible instantiation of an IOLTS at the language level, we present a semantics for this language and observe that it does indeed define an IOLTS (see Figure 1). Note that the IOLTS for this particular language will be deterministic iff the expression evaluation relation is single valued (which will not be the case if the $\mid$ operator is admitted).

4 Strategies and Non-Interference

We assume that a user at level a can only observe input/output events on channels $b \sqsubseteq a$ and that no user can see τ actions (modelling internal state transitions), making this a timing insensitive model. Different traces may thus appear the same to a given user. We write $t =_a t'$ to mean that two traces look the same to a user at level a. More generally, for a subset of security levels $A \subseteq L$, we write $t =_A t'$ to mean that $t{\restriction}A = t'{\restriction}A$, where $t{\restriction}A$ is t with all τ events removed and with all IO events $b?v$ and $b!v$ removed except those for which $b \in A$. Each such $=_A$ is clearly an equivalence relation on traces. Note that, since users at level a can see events at level a and below, $=_a$ is shorthand for $=_{\downarrow a}$ rather than $=_{\{a\}}$ (wherever we actually intend $=_{\{a\}}$ we will write this explicitly).

4.1 Strategies

Each user provides inputs on the channel corresponding to his or her security level and is aware of the history of usage (both inputs and outputs) on all channels at or below

$$[\text{Skip}]\ \langle \mathsf{skip}, \sigma\rangle \xrightarrow{\tau} \langle \mathsf{skip}, \sigma\rangle$$

$$[\text{Seq1}]\ \langle \mathsf{skip}; c_2, \sigma\rangle \xrightarrow{\tau} \langle c_2, \sigma\rangle$$

$$[\text{Seq2}]\ \frac{\langle c_1; c_2, \sigma\rangle \xrightarrow{l} \langle c_1'; c_2, \sigma'\rangle}{\langle c_1; c_2, \sigma\rangle \xrightarrow{l} \langle c_1'; c_2, \sigma'\rangle}\ l \in A$$

$$[\text{Assign}]\ \frac{\sigma \vdash e \to v}{\langle x := e, \sigma\rangle \xrightarrow{\tau} \langle \mathsf{skip}, \sigma[x := v]\rangle}$$

$$[\text{If1}]\ \frac{\sigma \vdash e \to v \neq 0}{\langle \mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2, \sigma\rangle \xrightarrow{\tau} \langle c_1, \sigma\rangle}$$

$$[\text{If2}]\ \frac{\sigma \vdash e \to 0}{\langle \mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2, \sigma\rangle \xrightarrow{\tau} \langle c_2, \sigma\rangle}$$

$$[\text{While}]\ \langle \mathsf{while}\ e\ \mathsf{do}\ c, \sigma\rangle \xrightarrow{\tau} \langle \mathsf{if}\ e\ \mathsf{then}\ (c; \mathsf{while}\ e\ \mathsf{do}\ c)\ \mathsf{else}\ \mathsf{skip}, \sigma\rangle$$

$$[\text{In}]\ \langle \mathsf{input}\ x\ \mathsf{from}\ a, \sigma\rangle \xrightarrow{a?v} \langle \mathsf{skip}, \sigma[x := v]\rangle$$

$$[\text{Out}]\ \frac{\sigma \vdash e \to v}{\langle \mathsf{output}\ e\ \mathsf{to}\ a, \sigma\rangle \xrightarrow{a!v} \langle \mathsf{skip}, \sigma\rangle}$$

Fig. 1. IOLTS semantics for a simple language

that level. The behaviour of a user in choosing inputs on a channel may be influenced by this knowledge of the history (as when High uses Program 1 as a covert channel) and is modeled as a channel strategy: a function from what the user knows to the user's choice of the next input on the channel. We allow strategies to be nondeterministic, thus we define them to be functions from traces to non-empty sets of values:

Definition 3. *An* a-strategy *is a function* $\omega_a : \mathrm{Tr} \to (\wp(\mathbb{V}) - \emptyset)$ *such that* $t_1 =_a t_2 \Rightarrow \omega_a(t_1) = \omega_a(t_2)$.

In the special case that ω_a is deterministic, we will write $\omega_a(t) = v$ as shorthand for $\omega_a(t) = \{v\}$. We use ω for arbitrary (ie possibly nondeterministic) strategies and δ for deterministic strategies.

A strategy modeling the behaviour of the program's whole environment is a collection of individual channel strategies indexed by the security lattice. We say two strategies are equivalent with respect to a given security level if the channel strategies at and below that level are identical.

Definition 4. *A* strategy ω *is an* L*-indexed family such that each* ω_a *is an* a*-strategy.*

Let Strat denote the set of all strategies. We write $\omega =_a \omega'$ to mean $\omega_b = \omega'_b$ for all $b \sqsubseteq a$.

The interaction between a program and its environment is modelled by playing a strategy against a state of an IOLTS to produce a trace of input and output events. Let s be a state of an IOLTS. Playing strategy ω against state s may produce trace t, written $\omega \models s \overset{t}{\twoheadrightarrow}$, if t is a *possible* trace for s and, for every input event $a?v$ in t, v is a value which may be chosen by ω_a when applied to the sequence of events leading up to the input event. Formally:

Definition 5. *$\omega \models s \overset{t}{\twoheadrightarrow}$ iff $s \overset{t}{\twoheadrightarrow}$ and $v \in \omega_a(t')$ for all $t'.a?v \leq t$, where $\leq$ is the prefix ordering on traces.*

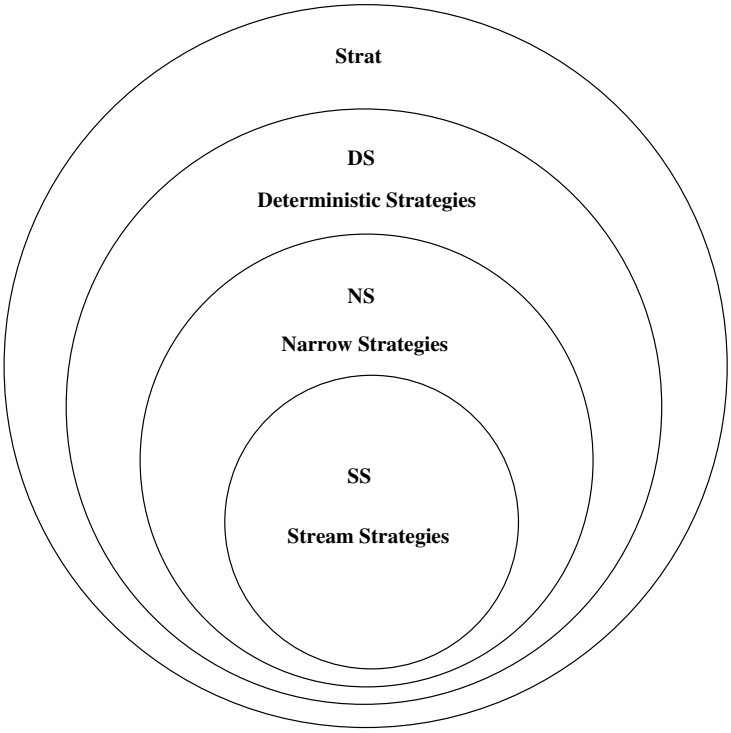

Fig. 2. An inclusion hierarchy of strategies

We define three interesting sub-classes of strategy:

DS. The deterministic strategies.

NS. The "narrow" strategies. This is the class of strategies considered in [14] (the term "narrow" is ours; in [14] they are simply called strategies). These are deterministic strategies such that the user's choice is influenced only by events on that particular channel, not by events on channels at lower security levels. The formal definition is as follows:

Definition 6 (Narrow Strategy). *A strategy ω is* narrow *iff it is deterministic and, for all a, if $t =_{\{a\}} t'$ then $\omega_a(t) = \omega_a(t')$.*

SS. The "stream" strategies. A stream strategy is just a family of streams (one for each channel) presented as a strategy. Concretely, each time a stream strategy is asked for an input on a channel it simply returns the next item in the stream for that channel. Each channel strategy in such a strategy returns a value which depends only on the number of inputs which have been requested on that channel so far, since this number is precisely the position in the stream which has been reached. For a channel a we say that traces t, t' are *a-stream-pointer equivalent*, written $t \bowtie_a t'$, iff t and t' contain the same number of a-input events. A stream strategy is thus a family of channel strategies each of which respects stream-pointer equivalence:

Definition 7 (Stream Strategy). *A strategy ω is a* stream strategy *iff it is deterministic and, for all a, if $t \bowtie_a t'$ then $\omega_a(t) = \omega_a(t')$*

Figure 2 illustrates how these sub-classes form an inclusion hierarchy. It is straightforward to verify that the inclusions shown do indeed hold and are, in fact, strict: $\mathrm{SS} \subset \mathrm{NS} \subset \mathrm{DS} \subset \mathrm{Strat}$.

4.2 Non-Interference

Our definition of non-interference [8] is framed in terms of strategies and traces. It is a generalisation of Definition 1 from [14]. The definition says that a state s of an IOLTS is non-interfering for a given set of strategies if, for each user, any two strategies drawn from the set which look the same, also produce sets of traces which look the same, when played against s.

Definition 8. *Let W be a set of strategies. A state s of an IOLTS is* non-interfering for *W (or W-*NI *for short) iff*

$$\forall\, \omega_1, \omega_2 \in W \,.\, (\omega_1 =_a \omega_2 \;\wedge\; \omega_1 \models s \overset{t_1}{\twoheadrightarrow}) \;\Rightarrow\; (\exists\, t_2 \,.\, t_2 =_a t_1 \;\wedge\; \omega_2 \models s \overset{t_2}{\twoheadrightarrow})$$

We say that s is simply non-interfering *(or NI for short) if it is non-interfering for the set of all strategies.*

We now explore the relationship between the NI properties corresponding to the sub-classes of strategy shown in Figure 2. We start with the obvious fact that inclusion of sub-classes of strategy implies reverse-inclusion of the corresponding NI properties:

Lemma 1. *Let $W_1, W_2 \subseteq \mathrm{Strat}$. If $W_1 \subseteq W_2$ then W_2-NI $\Rightarrow$ W_1-NI.*

We thus immediately have a sequence of inclusions of NI properties which mirrors the inclusions shown in Figure 2:

Proposition 1. *NI $\Rightarrow$ DS-NI $\Rightarrow$ NS-NI $\Rightarrow$ SS-NI.*

In Section 4.3, we establish that DS-NI and NI are actually equivalent. We conjecture that NS-NI is also equivalent to NI but verifying this is left for future work.

Note that SS-NI is essentially "property 1" from Section 3.1. By considering the sets of possible traces for the various programs in Section 3.1 it can be established, for example, that Program 1a is NI, whereas Program 1b is SS-NI but not NI. It is clear, therefore, that SS-NI is, in general, a strictly weaker property than NI. Nonetheless, we are able to show (Section 4.4) that for *deterministic* IOLTS, SS-NI and NI are equivalent.

4.3 Non-Interference for Deterministic Strategies

The following theorem says that, to establish non-interference, it is only necessary to consider deterministic strategies.

Theorem 1. *DS-NI* $\Longleftrightarrow$ *NI.*

By Proposition 1, to prove the theorem it is sufficient to show that if s is DS-NI then s is NI. We prove the contrapositive.

Let S be a state of an IOLTS and suppose that S does *not* have the NI property. Thus there must be two (possibly nondeterministic) strategies W, W', level B and trace T such that $\mathrm{W} =_{\mathrm{B}} \mathrm{W}'$ and:

1. $\mathrm{W} \models \mathrm{S} \overset{\mathrm{T}}{\twoheadrightarrow}$
2. For all t', if $\mathrm{W}' \models \mathrm{S} \overset{t'}{\twoheadrightarrow}$ then $t' \neq_{\mathrm{B}} \mathrm{T}$.

The key proof idea is to construct two B-equivalent deterministic strategies which, when played against S, result in the same NI-violating behaviours as W, W'.

First, we derive a deterministic strategy $\theta(\mathrm{W})$ from W, as follows. Let $\chi : (\wp(\mathbb{V}) - \emptyset) \to \mathbb{V}$ be some function such that $\chi(X) \in X$. Then:

$$\theta(\mathrm{W})_a(u) = \begin{cases} v & \text{if } \exists u' =_a u.\ u'a?v \leq \mathrm{T} \\ \chi(\mathrm{W}_a(u)) & \textit{otherwise} \end{cases}$$

It is necessary to show that $\theta(\mathrm{W})$ is well-defined. In particular, we must show:

a) if $u' =_a u$ and $u'' =_a u$ and $u'a?v \leq \mathrm{T}$ and $u''a?v' \leq \mathrm{T}$, then $v = v'$;
b) if $u =_a u'$ then $\theta(\mathrm{W})_a(u) = \theta(\mathrm{W})_a(u')$.

First we need the following technical lemma:

Lemma 2. *Let* $t_1 a?v \leq t_2 \ell$ *be such that* $t_1 \bowtie_a t_2$. *Then* $t_2 = t_1$ *and* $\ell = a?v$.

Proof. Let n_i be the number of a-input events in t_i. Since $t_1 \bowtie_a t_2$, we have $n_1 = n_2$. Now suppose towards a contradiction that $t_1 a?v \neq t_2 \ell$, hence $t_1 a?v \leq t_2$. But then we would have $n_1 + 1 \leq n_2$, which contradicts $n_1 = n_2$. □

Now we can established well-definedness of $\theta(\mathrm{W})$.

Proposition 2. $\theta(\mathrm{W})$ *is a well defined deterministic strategy.*

Proof

a) By assumption of $u' =_a u$ and $u'' =_a u$ we have $u' =_a u''$. Clearly $u' =_a u''$ implies that u', u'' have the same number of a-input events. Furthermore, by assumption that $u'a?v$ and $u''a?v'$ are both prefixes of T, one must be a prefix of the other. Thus, by Lemma 2, $a?v = a?v'$, hence $v = v'$.
b) Suppose $u =_a u'$. If the first case in the definition of $\theta(\mathrm{W})_a$ applies to u then, by essentially the same argument as in a), it must also apply to u' and give the same result. If the second case applies, then, since W_a is an a-strategy, $\mathrm{W}_a(u) = \mathrm{W}_a(u')$, hence $\chi(\mathrm{W}_a(u)) = \chi(\mathrm{W}_a(u'))$. □

Next we derive a deterministic strategy $\lambda(\mathrm{W}')$ from W'. In this case we must ensure that $\lambda(\mathrm{W}')_a = \theta(\mathrm{W})_a$ for all $a \sqsubseteq \mathrm{B}$, since we want $\theta(\mathrm{W}) =_{\mathrm{B}} \lambda(\mathrm{W}')$. We define:

$$\lambda(\mathrm{W}')_a(u) = \begin{cases} \theta(\mathrm{W})_a(u) & \text{if } a \sqsubseteq \mathrm{B} \\ \chi(\mathrm{W}'_a(u)) & otherwise \end{cases}$$

The proof that $\lambda(\mathrm{W}')$ is a well-defined deterministic strategy is essentially as for $\theta(\mathrm{W})$ and is omitted. It is immediate from the definition of $\lambda(\mathrm{W}')$ that $\theta(\mathrm{W}) =_{\mathrm{B}} \lambda(\mathrm{W}')$.

It remains to show that this pair of strategies constitute a counterexample to DS-NI and, for this, it suffices to show that:

1. $\theta(\mathrm{W})$ produces T when played against S.
2. The set of traces produced by $\lambda(\mathrm{W}')$ is a subset of those produced by W'.

For the first of these, it is given that $\mathrm{S} \overset{\mathrm{T}}{\twoheadrightarrow}$, so we need only show that $\theta(\mathrm{W})_a(u) = v$ whenever $ua?v \leq \mathrm{T}$, and this is clear from the definition of $\theta(\mathrm{W})_a$, since $u =_a u$. For the second, say that strategy ω' *refines* strategy ω iff $\omega'_a(u) \subseteq \omega_a(u)$, for all a, u. It is then immediate from Definition 5 that the refining strategy produces a subset of the traces of the original when played against the same state. Formally:

Lemma 3. *If ω' refines ω and $\omega' \models s \overset{u}{\twoheadrightarrow}$ then $\omega \models s \overset{u}{\twoheadrightarrow}$.*

It is straightforward to verify that $\theta(\mathrm{W})$ refines W and hence that $\lambda(\mathrm{W}')$ refines W'. This completes the proof that DS-NI $\Rightarrow$ NI.

4.4 Non-Interference for Deterministic IOLTS

Here we establish our main result. That, *for deterministic IOLTS*, to establish NI it is only necessary to consider stream strategies. Thus, for deterministic systems, when reasoning about information flow it can suffice to work with a simple stream-based semantic model of the environment and a corresponding stream-based definition of NI, rather than strategies.

Theorem 2. *A state s of a deterministic IOLTS is NI iff it is SS-NI.*

Corollary 1. *For deterministic IOLTS: NI, DS-NI, NS-NI and SS-NI are all equivalent.*

Given Proposition 1 and Theorem 1, to prove Theorem 2 it suffices to show that, for any state s of a deterministic IOLTS, if s is SS-NI then s is DS-NI. Again, we prove the contrapositive.

Let S be a state of a deterministic IOLTS and suppose that S does *not* have the DS-NI property. Thus there must be two deterministic strategies D, D', level B and trace T such that $\mathrm{D} =_{\mathrm{B}} \mathrm{D}'$ and:

1. $\mathrm{D} \models \mathrm{S} \overset{\mathrm{T}}{\twoheadrightarrow}$
2. For all t', if $\mathrm{D}' \models \mathrm{S} \overset{t'}{\twoheadrightarrow}$ then $t' \neq_{\mathrm{B}} \mathrm{T}$.

The proof mimics the one above for DS-NI $\Rightarrow$ NI, but this time we derive stream strategies from deterministic strategies. We derive the stream strategy $\phi(\mathrm{D})$ from D as follows:

$$\phi(\mathrm{D})_a(u) = \begin{cases} v & \text{if } \exists u' \bowtie_a u.\ \mathrm{D} \models \mathrm{S} \overset{u'a?v}{\twoheadrightarrow} \\ \mathrm{K} & otherwise \end{cases}$$

where K is some (arbitrary) constant in $\mathbb{V}$.

We must show that $\phi(\mathrm{D})$ is a well-defined stream strategy. In particular, we must show:

a) If $u' \bowtie_a u$ and $u'' \bowtie_a u$ and $\mathrm{D} \models \mathrm{S} \overset{u'a?v}{\twoheadrightarrow}$ and $\mathrm{D} \models \mathrm{S} \overset{u''a?v'}{\twoheadrightarrow}$, then $v = v'$.
b) If $u \bowtie_a u'$ then $\phi(\mathrm{D})_a(u) = \phi(\mathrm{D})_a(u')$.

Part b) follows immediately from the definition of $\phi(\mathrm{D})$ once we have shown a).

To show a) we make use of a lemma which states an expected consequence of determinism: if we play a deterministic strategy against any state of a deterministic IOLTS, there will be no branching in the set of traces produced.

Lemma 4. *Let s be a state of a deterministic IOLTS and let δ be a deterministic strategy. If $\delta \models s \overset{t_1}{\twoheadrightarrow}$ and $\delta \models s \overset{t_2}{\twoheadrightarrow}$ then either $t_1 \leq t_2$ or $t_2 \leq t_1$.*

Proof. Suppose, without loss of generality, that $\mathrm{length}(t_1) \leq \mathrm{length}(t_2)$. Proceed by induction on $\mathrm{length}(t_1)$ to show that $t_1 \leq t_2$.

If $\mathrm{length}(t_1) = 0$ then $t_1 = \epsilon \leq t_2$.

If $\mathrm{length}(t_1) > 0$ then t_1 has the form $t_1'\ell_1$ and $s \overset{t_1'}{\twoheadrightarrow} s_1' \overset{\ell_1}{\rightarrow} s_1''$. Then $\mathrm{length}(t_1') < \mathrm{length}(t_2)$ and by IH $t_1' \leq t_2$, hence $t_1' < t_2$. Thus, for some ℓ_2, $t_1'\ell_2 \leq t_2$ and $s \overset{t_1'}{\twoheadrightarrow} s_2' \overset{\ell_2}{\rightarrow} s_2''$. Part 2 of the definition of deterministic IOLTS (Definition 1) entails (by a simple induction on the length of t_1') that $s_2' = s_1'$. It thus remains to show that $\ell_1 = \ell_2$. Suppose towards a contradiction that $\ell_1 \neq \ell_2$. By part 1 of Definition 1 we must have $\ell_1 = a?v_1$ and $\ell_2 = a?v_2$. Then, since $\delta \models s \overset{t_1'a?v_1}{\twoheadrightarrow}$ and $\delta \models s \overset{t_1'a?v_2}{\twoheadrightarrow}$, we have $v_1 \in \delta_a(t_1')$ and $v_2 \in \delta_a(t_1')$. But then, since δ is deterministic, $v_1 = v_2$, a contradiction. □

Well-definedness of $\phi(\mathrm{D})$ then follows:

Proposition 3. *$\phi(\mathrm{D})$ is a well-defined stream strategy.*

Proof. It remains to show that condition a) holds. That is, if $u' \bowtie_a u$ and $u'' \bowtie_a u$ and $\mathrm{D} \models \mathrm{S} \overset{u'a?v}{\twoheadrightarrow}$ and $\mathrm{D} \models \mathrm{S} \overset{u''a?v'}{\twoheadrightarrow}$ then $v = v'$. Now S is a state of deterministic IOLTS and D is deterministic, so, by Lemma 4, either $u'a?v \leq u''a?v'$ or $u''a?v' \leq u'a?v$. From $u' \bowtie_a u$ and $u'' \bowtie_a u$ we also have $u' \bowtie_a u''$. Hence by Lemma 2, $v = v'$. □

Next we derive a stream strategy $\psi(\mathrm{D}')$ from D'. For $a \sqsubseteq \mathrm{B}$ we define $\psi(\mathrm{D}')_a = \phi(\mathrm{D})_a$ and for $a \not\sqsubseteq \mathrm{B}$ we derive $\psi(\mathrm{D}')$ from D' exactly as we derived $\phi(\mathrm{D})$ from D:

$$\psi(\mathrm{D}')_a = \begin{cases} \phi(\mathrm{D})_a & \text{if } a \sqsubseteq \mathrm{B} \\ \phi(\mathrm{D}')_a & \text{if } a \not\sqsubseteq \mathrm{B} \end{cases}$$

The proof that $\psi(\mathrm{D}')$ is a well-defined stream strategy is essentially as for $\phi(\mathrm{D})$ and is omitted. It is immediate from the definition of $\psi(\mathrm{D}')$ that $\phi(\mathrm{D}) =_{\mathrm{B}} \psi(\mathrm{D}')$.

For the final step in the proof that SS-NI $\Rightarrow$ DS-NI we introduce the notion of a-prefix:

Definition 9. *Trace u is an a-*prefix *of u', written $u \preceq_a u'$, iff $u =_a u''$ for some $u'' \leq u'$.*

We state without proof some obvious properties of $\preceq_a$:

- If $a_1 \sqsubseteq a_2$ then $\preceq_{a_2} \subseteq \preceq_{a_1}$.
- If $u \leq u' \preceq_a u''$ then $u \preceq_a u''$.
- If $u =_a u'$ then $u \preceq_a u'$.

We will use these freely in the remainder of the proof.

The proof is now essentially completed by the following lemma, which says that, when played against S, $\phi(\mathrm{D})$ and D produce exactly the same sets of traces (including, in particular, T), whereas every trace produced by $\psi(\mathrm{D}')$ is either also produced by D' or is not a B-prefix of T.

Lemma 5

1. *$\phi(\mathrm{D}) \models \mathrm{S} \overset{u}{\twoheadrightarrow}$ iff $\mathrm{D} \models \mathrm{S} \overset{u}{\twoheadrightarrow}$.*
2. *If $\psi(\mathrm{D}') \models \mathrm{S} \overset{u}{\twoheadrightarrow}$ and $u \preceq_{\mathrm{B}} \mathrm{T}$ then $\mathrm{D}' \models \mathrm{S} \overset{u}{\twoheadrightarrow}$.*

Proof. The lemma holds vacuously if u is not a trace of S, so we need only show that it holds for all u such that $\mathrm{S} \overset{u}{\twoheadrightarrow}$. Let $\#_{\mathrm{I}}(u)$ denote the number of input events in u. We proceed by induction on $\#_{\mathrm{I}}(u)$. Take the two parts in turn:

1. For $\#_{\mathrm{I}}(u) = 0$ we have both $\phi(\mathrm{D}) \models \mathrm{S} \overset{u}{\twoheadrightarrow}$ and $\mathrm{D} \models \mathrm{S} \overset{u}{\twoheadrightarrow}$ by assumption that $\mathrm{S} \overset{u}{\twoheadrightarrow}$. In the inductive case, $\#_{\mathrm{I}}(u) = n+1$, hence $u'a?v \leq u$ for some u' with $\#_{\mathrm{I}}(u') = n$. By IH $\phi(\mathrm{D}) \models \mathrm{S} \overset{u'}{\twoheadrightarrow}$ iff $\mathrm{D} \models \mathrm{S} \overset{u'}{\twoheadrightarrow}$. Note that, since $a?v$ is the last input event in u, we have:
   (i) $\mathrm{D} \models \mathrm{S} \overset{u}{\twoheadrightarrow}$ iff $(\mathrm{D} \models \mathrm{S} \overset{u'}{\twoheadrightarrow}) \wedge (\mathrm{D}_a(u') = v)$
   (ii) $\phi(\mathrm{D}) \models \mathrm{S} \overset{u}{\twoheadrightarrow}$ iff $(\phi(\mathrm{D}) \models \mathrm{S} \overset{u'}{\twoheadrightarrow}) \wedge (\phi(\mathrm{D})_a(u') = v)$
   Thus it suffices to show that, if $\mathrm{D} \models \mathrm{S} \overset{u'}{\twoheadrightarrow}$ then $\mathrm{D}_a(u') = \phi(\mathrm{D})_a(u')$. Let $\mathrm{D}_a(u') = w$. Then, since $\mathrm{S} \overset{u'a?v}{\twoheadrightarrow}$ and the IOLTS is input-neutral, we have $\mathrm{S} \overset{u'a?w}{\twoheadrightarrow}$. Hence, since $u' \bowtie_a u'$, by definition of $\phi(\mathrm{D})_a$ we have $\phi(\mathrm{D})_a(u') = w$.
2. Assume $\psi(\mathrm{D}') \models \mathrm{S} \overset{u}{\twoheadrightarrow}$ and $u \preceq_{\mathrm{B}} \mathrm{T}$. The base case is as for part 1. In the inductive case, again we have $\#_{\mathrm{I}}(u) = n+1$, hence $u'a?v \leq u$ for some u' with $\#_{\mathrm{I}}(u') = n$. By assumption of $u \preceq_{\mathrm{B}} \mathrm{T}$ we have $u' \preceq_{\mathrm{B}} \mathrm{T}$, hence by IH $\mathrm{D}' \models \mathrm{S} \overset{u'}{\twoheadrightarrow}$. Thus it suffices to show $\mathrm{D}'_a(u') = v$. We proceed by cases according to whether $a \sqsubseteq \mathrm{B}$.

   If $a \not\sqsubseteq \mathrm{B}$, let $w = \mathrm{D}'_a(u')$. Then, since $\mathrm{S} \overset{u'a?v}{\twoheadrightarrow}$ and the IOLTS is input-neutral, we have $\mathrm{S} \overset{u'a?w}{\twoheadrightarrow}$. Thus, by definition of $\psi(\mathrm{D}')$, $\psi(\mathrm{D}')_a(u') = w$. But, by assumption of $\psi(\mathrm{D}') \models \mathrm{S} \overset{u}{\twoheadrightarrow}$, we have $\psi(\mathrm{D}')_a(u') = v$, hence $w = v$.

   If $a \sqsubseteq \mathrm{B}$ then, from $u \preceq_{\mathrm{B}} \mathrm{T}$, we have $u'a?v \preceq_a \mathrm{T}$. Thus $u''a?v \leq \mathrm{T}$ and $u'' =_a u'$, for some u''. Thus $\mathrm{D}_a(u'') = \mathrm{D}_a(u') = v$. But, since $\mathrm{D} =_{\mathrm{B}} \mathrm{D}'$ and $a \sqsubseteq \mathrm{B}$, we have $\mathrm{D}'_a = \mathrm{D}_a$, hence $\mathrm{D}'_a(u') = v$. $\square$

Proposition 4. $\phi(\mathrm{D}), \psi(\mathrm{D}')$ *are a counterexample to SS-NI.*

Proof. By part 1 of Lemma 5, $\phi(\mathrm{D}) \models \mathrm{S} \overset{\mathrm{T}}{\twoheadrightarrow}$. Now suppose $\psi(\mathrm{D}') \models \mathrm{S} \overset{t'}{\twoheadrightarrow}$ and $t' =_{\mathrm{B}} \mathrm{T}$. But then $t' \preceq_{\mathrm{B}} \mathrm{T}$ and hence, by part 2 of Lemma 5, $\mathrm{D}' \models \mathrm{S} \overset{t'}{\twoheadrightarrow}$, contradicting the original assumption that D, D' are a counterexample to NI. □

This concludes the proof of Theorem 2.

5 Conclusions

We have defined a notion of Input-Output Labelled Transition System (IOLTS) suitable for modelling interactive programming languages. Following previous work by Wittbold and Johnson [16] and O'Neill, Clarkson and Chong [14] we have defined a notion of non-interference (NI) for IOLTS, modelling the users' input behaviours as strategies. Our main result has been to show that, for deterministic IOLTS, a simpler definition of NI, based on a stream model of user input, is equivalent.

5.1 Non-Interference and Nondeterminism

The definition of NI we use in this paper is (essentially) the one used for deterministic programs in [14]. However, although the definition can also be applied to nondeterministic programs (as our use of it illustrates) it is interesting to note that the authors of [14] actually modify the definition when they add non-determinism to the language. (Unfortunately, in modifying it, they render it unable to distinguish between the insecure Program 1b and its secure variant Program 1a). The modification is motivated by the desire to avoid so-called *refinement attacks*, in which refining a secure program (removing some nondeterminism) renders it insecure. We chose not to follow this route since it identifies two uses of nondeterminism which we prefer to differentiate: the use of nondeterminism to allow under-specification, and the use of nondeterminism as a programming construct, essentially as a source of deliberate "noise" intended to disrupt information flows. It is this latter use which is relevant in the covert channel examples described above.

But there is a possible weakness in the security delivered by our version of NI for nondeterministic programs. Consider the following example:

```
input x from H
if (1 | x) then
    output 0 to L
else
    while (true) do skip
```

(Recall that | here is nondeterministic choice.) This program is NI by our definition. Whether it should be regarded as secure depends on our assumptions about the observability of non-termination in the presence of nondeterminism. If we wish to make the definition of NI sensitive to the possibility of non-termination in this example, we might use a more sophisticated definition based, for example, on a form of bisimulation rather

than trace-equivalence. This approach would suggest transposing the problem into a process algebraic setting, as explored in [7] (see Section 5.2 for further discussion on this point). Alternatively, we might consider *weakening* the definition of NI to make it, more generally, termination insensitive. In the latter case it would be interesting to try to adapt the work of [5] to establish computational bounds on the rate at which information could be leaked.

5.2 Future Work

Our longer term goal is to be able to reason about the security properties of programs in interaction with their environments in a compositional way. Ideally we do not want to treat these two actors differently. One stumbling block we face in achieving this is that a very common environment for a program is another program or even a set of programs. What is the relationship between programs and strategies?

A strategy is defined in Wittbold and Johnson [16] as a map from the history of inputs and outputs on a given channel to the next input on that channel. There is no computational content in that definition and, in general, a strategy could be non-computable (and clearly not representable by a program). On the other hand, not every program has a semantics which can be characterised as providing an appropriate input to another program whenever it is required. In fact the setting in which Wittbold and Johnson introduce strategies is a purely synchronous one in which inputs are always supplied to the program. So, in particular, a program written in the interactive core imperative language defined in this paper will not in general define a strategy for another program written in the same language, or even define a strategy at all. Consider for example a program which only updates its internal state and never engages with input or output at all.

If we assume the programs are interacting in an asynchronous fashion, a program which expects input on a given channel may never get it from the other programs in its environment. Even supposing a program is structured correctly so that it acts as a strategy for another program (and presumably vice versa) termination problems may mean that it never produces an expected output. For example:

```
P1:
input x from H;
input z from L;
output (z XOR x) on H;

P2:
output y on H;
while(x < 0)  x--;
output x on L;
input w from H;
```

Program P2 will provide input for Program P1 on L only some of the time. It could be described as a partial strategy. Any reasoning about environments formed from programs would have to take partiality into account.

There are two directions in which we could take this work.

We could continue to model environments as strategies and ask what kinds of systems could be strategies for each other and ask what kinds of constraints on the systems would that require. We have discussed some of the issues above but an interesting enquiry along these lines is the possibility of modelling the interaction using game semantics [1,2,9].

On the other hand, why constrain the model of the environment to be a strategy? Our use of IOLTS suggests some more general formal process model might be a suitable setting for extending our results, building on the foundational work of [7]. A potential issue to be addressed in this case would be that *sequentiality* seems to be an essential characteristic of the deterministic programs on which we have focused. It may be that security properties such as NDC and BNDC as defined in [7] are so strong as to effectively rule out many sequential systems of interest.

Orthogonal to these two lines of enquiry is the question of probabilistic models of the the behaviour of the environment. With respect to strategies, for example, Jürjens has shown [13] that Gray's security property Probabilistic Noninterference (PNI) [12] is a generalisation of Wittbold and Johnson's nondeducibility on strategies [16] while Aldini, Bravetti and Gorrieri have analysed probabilistic noninterference using a probabilistic process algebra [3,4].

Acknowledgments

This work was supported by EPSRC research grant EP/C545605/1 & EP/C009746/1 (Quantitative Information Flow). Thanks to Aslan Askarov, Catuscia Palamidessi, Andrei Sabelfeld and David Sands for their encouragement and comments on early versions of this work. We also thank the anonymous reviewers for their very helpful criticisms and suggestions for improvement.

References

1. Abramksy, S., McCusker, G.: Game semantics. In: Berger, U., Schwichtenberg, H. (eds.) Logic and Computation: Proc. 1997 Marktoberdorf Summer School. NATO Science Series. Springer, Heidelberg (1998)
2. Abramsky, S., Jagadeesan, R., Malacaria, P.: Full abstraction for pcf. Information and Computation, 409–470 (December 2000)
3. Aldini, A.: Probabilistic information flow in a process algebra. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, pp. 152–168. Springer, Heidelberg (2001)
4. Aldini, A., Bravetti, M., Gorrieri, R.: A process-algebraic approach for the analysis of probabilistic noninterference. J. Comput. Secur. 12(2), 191–245 (2004)
5. Askarov, A., Hunt, S., Sabelfeld, A., Sands, D.: Termination-insensitive noninterference leaks more than just a bit. In: Jajodia, S., Lopez, J. (eds.) ESORICS 2008. LNCS, vol. 5283. Springer, Heidelberg (2008)
6. Denning, D.E.: A lattice model of secure information flow. Comm. of the ACM 19(5), 236–243 (1976)
7. Focardi, R., Gorrieri, R.: A classification of security properties for process algebras. J. Computer Security 3(1), 5–33 (1995)
8. Goguen, J.A., Meseguer, J.: Security policies and security models. In: Symposium on Security and Privacy, April 1982, pp. 11–20 (1982)

9. Harmer, R., Mccusker, G.: A fully abstract game semantics for finite nondeterminism. In: Proceedings of the Fourteenth Annual Symposium on Logic in Computer Science, LICS 1999, pp. 422–430. IEEE Computer Society Press, Los Alamitos (1999)
10. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. In: Principles Of Programming Languages (January 2008)
11. Hunt, S., Sands, D.: Just forget it – the semantics and enforcement of information erasure. In: Drossopoulou, S. (ed.) ESOP 2008. LNCS, vol. 4960, pp. 239–253. Springer, Heidelberg (2008)
12. Gray III, J.W.: Toward a mathematical foundation for information flow security. In: Proc. of the 1991 Symposium on Security and Privacy, pp. 21–35. IEEE, Los Alamitos (1991)
13. Jürjens, J.: Secure information flow for concurrent processes. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 395–409. Springer, Heidelberg (2000)
14. O'Neill, K.R., Clarkson, M.R., Chong, S.: Information-flow security for interactive programs. In: CSFW, pp. 190–201. IEEE Computer Society, Los Alamitos (2006)
15. Shannon, C.E.: Channels with side information at the transmitter. IBM journal of Research and Development 2(4), 289–293 (1958)
16. Wittbold, J.T., Johnson, D.M.: Information flow in nondeterministic systems. In: IEEE Symposium on Security and Privacy, pp. 144–161 (1990)