Information Flow in Systems with Schedulers*

Ron van der Meyden, Chenyi Zhang

University of New South Wales and
National ICT Australia,
Sydney, Australia

Abstract. The focus of work on information flow security has primarily been on definitions of security in asynchronous systems models. This paper considers systems with schedulers, which require synchronous variants of these definitions. In particular, it studies the dependence of these variant definitions of security on implementation details of the scheduler. Such independence is shown to hold for synchronous variants of trace-based definitions, but not for a bisimulation-based definition. An approach to the latter problem is proposed that preserves the attractive computational properties of the bisimulation-based definition.

Information flow security is concerned with the ability of agents in a system to make deductions about the activity of others, or to cause information to flow to other agents. Since the seminal work of Goguen and Meseguer [GM82], which introduced the notion of *noninterference* in a deterministic state machine model, a significant body of literature has developed on this topic, dealing with how noninterference should be defined in nondeterministic state machines [McC88,BY94] and in richer semantic models such as the process algebras CCS [FG95] and CSP [Ros95]. Until recently, however, most work in this area confined itself to asynchronous models of computation, assuming that agents do not have access to a system clock, which is too strong in many applications.

Notably, this is the case in the original motivation for the notion of noninterference, operating systems separation kernels [Rus81,Rus82,Rus00]. The function of this class of software system is to separate security domains in order to prevent information flow and clashes of shared resource usage. In particular, in a uni-processor system this involves *scheduling* the activity of the agents in the system. In the presence of scheduling, many of the assumptions of the asynchronous systems models used in the literature break down. For example, a common assumption is that the system is "input enabled" in the sense that each action may be performed at any state. This is patently not the case in the presence of scheduling, which enables one agent's actions while simultaneously disabling the actions of others. Similarly, knowledge of the schedule may permit Low to deduce the number of actions that High has taken, whereas the asynchronous definitions of security would classify this as insecure. A correct treatment of systems such as separation kernels therefore requires different definitions of security.

* Work of the first author supported by an Australian Research Council Discovery Grant. An extended version of this paper including proofs of results is available at `http://www.cse.unsw.edu.au/~czhang/fst.pdf`. The first author thanks the Courant Institute, New York University and the Computer Science Department, Stanford University for their hospitality in hosting a sabbatical visit during which this work was conducted.

In this paper, we undertake a study of the impact of schedulers on some of the asynchronous definitions of security from the literature. We ask how these definitions should be modified, and study the relationships between the revised definitions. One question of particular concern is the extent to which these definitions are sensitive to how the scheduler is implemented. Intuitively, the function of a scheduler is only to make decisions as to when each agent in the system should be enabled. Schedulers may be nondeterministic, and the nondeterminism may be resolved early (e.g., by flipping a set of coins before their outcome is needed) or late (e.g., by flipping coins only at decision points). However, as the private state of the scheduler should be invisible to the agents in the system (although they may know the scheduler being used), such implementation details should not affect the security of the system. Independence of scheduler implementation also gives desirable flexibility to the implementer of the scheduler. We show that modified versions of trace-based definitions of security are independent of the scheduler implementation, but this is not the case for a definition motivated by McCullough's notion of restrictiveness [McC87,McC88] (which is also closely related to one of Focardi and Gorrieri's bisimulation based definitions of security [FG95]). Fortunately, in this case, we show that only the existence of a secure implementation makes the security definition sufficiently strong, which further yields feasible verification of this property.

The structure of the paper is as follows. In Section 1 we introduce the semantic framework within which we work, a type of labelled transition system with observations. Section 2 defines schedulers and their representation within this model. Section 3 deals with trace-based definitions of security for systems with schedulers. Section 4 considers bisimulation-based definitions, where independence of the scheduler implementation becomes a non-trivial issue. Section 5 concludes by discussing related work and future directions.

1 A Discrete-Time System Model

We take as the basic semantic model an enrichment of the well-established *labelled transition system semantics* for process algebra, adding to it a notion of observation on states. This provides a framework that seems better able to capture the fact that in synchronous systems, agents may observe state changes even when not participating in an action.

A *signature* is a tuple (A, D, dom) consisting of a set of actions A, a set of domains (or agents) D and a function $dom : A \to D$ associating a domain with each action. We define a *state-observed labelled transition system* (SOLTS) for such a signature to be a tuple of the form $\langle S, S_0, \to, O, obs \rangle$ where

- S is a set of *states* (with elements denoted by s, t, t_1, etc.),
- $S_0 \subseteq S$ represents the set of *initial* states,
- $\to \subseteq S \times A \times S$ is a *transition relation*,
- O is a set of observations,
- $obs : D \times S \to O$ is a function representing the *observation* made in each state by each agent. For readability, we 'curry' the function obs (or its variants) by writing $obs_u(s)$ for $obs(u, s)$.

Write $\mathbb{L}^\circ$ for the set of all such systems. We write $s \xrightarrow{a} t$ when $(s, a, t) \in \to$, and $s \xrightarrow{a}$ when there exists t such that $s \xrightarrow{a} t$. More generally, we write $s_0 \xrightarrow{\alpha} s_n$

when $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} s_n$ and $\alpha = a_1 a_2 \ldots a_n$. A *run* of a SOLTS is a sequence of the form $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} s_n$ with $s_0 \in S_0$. We write $\mathcal{R}(M)$ for the set of all runs of M. We denote the sequence of actions in a run r by $Act(r) = a_1 a_2 \ldots a_n$, and for each agent u write $Act_u(r)$ for the subsequence of $Act(r)$ consisting of actions a with $dom(a) = u$. A SOLTS is *deterministic* if for $s, t_1, t_2 \in S$ and $a \in A$, if $s \xrightarrow{a} t_1$ and $s \xrightarrow{a} t_2$ then $t_1 = t_2$. It is *input enabled* if $s \xrightarrow{a}$ for all $s \in S$ and $a \in A$.

Given two SOLTS $L = \langle S, S_0, \rightarrow, O, obs \rangle$, $L' = \langle S', S_0', \rightarrow', O', obs' \rangle$ with the same signature, define the parallel composition $L \parallel L'$ to be the SOLTS $\langle S \times S', S_0 \times S_0', \rightarrow'', O \times O', obs'' \rangle$ where $\rightarrow'' = \{((s_1, s_1'), a, (s_2, s_2')) \mid s_1 \xrightarrow{a} s_2 \wedge s_1' \xrightarrow{a} s_2'\}$ and $obs''_u((s, s')) = (obs_u(s), obs'_u(s'))$ for all $u \in D$. This corresponds to the lock-step execution of the two systems with synchronisation on actions.

Like most of the literature, we confine our attention to systems with two security domains High (H) and Low (L) and the security policy $L \leq H$, which prohibits information flow from H to L. However, in order to deal with scheduling and passage of time, it is convenient to include a third agent Sys that may act when both H and L are waiting. The agent Sys can be understood as corresponding to the scheduler activity as well as system internal actions. For the remainder of this paper, we let $D = \{H, L, Sys\}$. For $u \in D$ we define $A_u = \{a \in A \mid dom(a) = u\}$, and assume that $A_{Sys} = \{\tau\}$. (The effect of Sys actions may be nondeterministic, but we assume that there is not a need to distinguish specific Sys events. Whereas A_H and A_L can be thought of as representing inputs provided by the agents, Sys provides no inputs, but only represents the automatic evolution of the state over time.) Write $\mathbb{M}$ for the set of all input-enabled SOLTS with $D = \{H, L, Sys\}$ and $A_{Sys} = \{\tau\}$, and refer to these systems as *machines*. In addition, we assume that actions take unit time and that time continues to flow, so that agents cannot halt the system by failing to act when scheduled. If failure to act is a possibility in an application, it can be accommodated by including a "null" action for each agent, after which the scheduler may schedule another agent.

2 Schedulers

Machines can be given an asynchronous semantics, but in this paper we will study a semantics in which machines execute under the control of a scheduler, which selects an agent at each moment of time. Our definition of scheduler differs from others in the literature (such as that of [SS00]) in that a scheduler only partially resolves the system's nondeterminism. We give an agent *free will* to choose which action to perform when it is scheduled.

Formally, a *nondeterministic scheduler* is a function $\sigma : A^* \rightarrow \mathcal{P}(D)$. Intuitively, given a history of actions $\alpha \in A^*$, one of the agents in the set $\sigma(\alpha)$ will be scheduled next. *Compatibility* of a finite sequence of actions with a scheduler σ is defined by the following induction: the empty sequence ϵ is compatible with σ, and αa is compatible with σ iff α is compatible with σ and $dom(a) \in \sigma(\alpha)$, where $\alpha \in A^*$ and $a \in A$. A *run* r of a machine is defined to be compatible with a scheduler σ if $Act(r)$ is compatible with σ. Given a machine M, we write $\mathcal{R}(M, \sigma)$ for the set of all runs of M compatible with σ.

We henceforth assume some well-formedness conditions on schedulers. First, we assume that schedulers do not terminate, so that if α is compatible with σ, then $\sigma(\alpha) \neq \emptyset$. Second, we assume that if α is not compatible with σ, then

$\sigma(\alpha) = \emptyset$. (This simplifies the discussion of scheduler implementations below.) A scheduler σ is *deterministic* if $\sigma(\alpha)$ is a singleton for all compatible $\alpha \in A^*$. A *schedule* is a finite or infinite sequence $\varsigma = u_0u_1u_2u_3\ldots$ where each $u_i \in D$. For $\alpha = a_0a_1a_2\ldots$, we write $\varsigma(\alpha)$ for the schedule $dom(a_0)\, dom(a_1)\, dom(a_2)\ldots$. If r is a run we also write $\varsigma(r)$ for $\varsigma(Act(r))$.

In order to prevent the scheduler being a channel for information flow, we define a notion that captures that the decisions of the scheduler are independent of the actions of an agent. For the definition, we need an operation on actions that masks actions of agent u: define $\mu_u(a) = a$ when $a \in A \setminus A_u$ and $\mu_u(a) = \bot_u$ when $a \in A_u$. For a sequence $\alpha = a_1a_2\ldots \in A^*$ define $\mu_u(\alpha) = \mu_u(a_1)\,\mu_u(a_2)\ldots$. Define a scheduler σ to be *u-oblivious* if $\mu_u(\alpha) = \mu_u(\alpha')$ implies $\sigma(\alpha) = \sigma(\alpha')$ for all $\alpha, \alpha' \in A^*$. Intuitively, this says that scheduling decisions do not depend on the actions performed by agent u. We may therefore view a u-oblivious scheduler σ as a function from $(\mu_u(A))^*$ to $\mathcal{P}(D)$, where $\mu_u(A) = (A \setminus A_u) \cup \{\bot_u\}$.

Example 1. Consider a scheduler σ in which H and L are allocated alternate blocks of time of length k, except that L may relinquish some of its share to H by performing a `yield` operation. More precisely:

- $\sigma(\alpha) = \{H\}$, if $|\alpha|$ *div* k is odd or $|\alpha|$ *div* k is even and the rightmost action of domain L in α is `yield`,
- $\sigma(\alpha) = \{L\}$ otherwise.

Then σ is an H-oblivious scheduler. Here $|x|$ denotes the length of a sequence x and *div* denotes integer division. □

Schedulers may be represented as SOLTS. A *scheduler SOLTS* is a SOLTS of the form $\langle Q, Q_0, \rightarrow, \{\bot\}, obs\rangle$ that satisfies

1. there is a transition from each state,
2. all transitions from a state are from the same domain, and all actions from that domain are enabled, i.e., if $s \xrightarrow{a}$ and $s \xrightarrow{b}$ then $dom(a) = dom(b)$, and $s \xrightarrow{c}$ for all $c \in A_{dom(a)}$, and
3. $obs_u(s) = \bot$ for all states s and domains u.

(Note that (3) means that agents do not obtain information about the scheduled agents from their observations on any state of a scheduler SOLTS.)

We say that a scheduler SOLTS is *u-oblivious* for $u \in D$ if for all states s, t and actions a, if $s \xrightarrow{a} t$ and $dom(a) = u$ then $s \xrightarrow{c} t$ for all actions $c \in A_u$. A scheduler or scheduler SOLTS is *oblivious* if it is u-oblivious for all $u \in D$.

Given a scheduler SOLTS $\mathcal{A} = \langle Q, Q_0, \rightarrow, \{\bot\}, obs\rangle$, define a scheduler $\sigma_{\mathcal{A}}$ by $\sigma_{\mathcal{A}}(\alpha) = \{sched(q) \mid q_0 \xrightarrow{\alpha} q,\ q_0 \in Q_0\}$, where $sched(q)$ is the unique domain that has its actions enabled at q. Now say $\mathcal{A}$ *represents* σ if $\sigma = \sigma_{\mathcal{A}}$. Interestingly, a scheduler SOLTS representing a u-oblivious scheduler is not necessarily u-oblivious. Therefore we decide to restrict to the u-oblivious scheduler SOLTSs representing σ when a u-oblivious scheduler σ is refered to, and we justify the existence of such u-oblivous implementations as follows.

Definition 1. *For all u-oblivious σ, define the (infinite state) characteristic scheduler SOLTS $\mathcal{A}^\sigma = \langle W, W_0, \rightarrow, \{\bot\}, obs\rangle$ by*

1. $W = (\mu_u(A))^* \times D$,

2. $W_0 = \{(\epsilon, u) \mid u \in \sigma(\epsilon)\}$,
3. $(\gamma, u) \xrightarrow{a} (\gamma', v)$ *iff* $dom(a) = u$ *and* $\gamma' = \gamma \cdot \mu_u(a)$ *and* $v \in \sigma(\gamma')$,
4. $obs(u, w) = \bot$ *for all* $u \in D$ *and* $w \in W$.

It can be readily shown that $\mathcal{A}^\sigma$ represents σ. Since $\mu_u(a) = \bot_u$ for all $a \in A_u$, it is also clear that $\mathcal{A}^\sigma$ is u-oblivious. If we drop the condition that σ be u-oblivious we can give a similar definition by removing all occurrences of the operator μ_u. This yields the following result, in which we also note that the set of runs of a SOLTS compatible with a scheduler can be understood as being obtained via parallel composition with a scheduler SOLTS representing the scheduler:

Proposition 1.

1. *Every (well-formed) scheduler has a scheduler SOLTS that represents it.*
2. *A (well-formed) scheduler is u-oblivious iff it is represented by some u-oblivious SOLTS.*
3. *If $\mathcal{A}$ is a scheduler SOLTS that represents σ and M is a machine, then $\mathcal{R}(M, \sigma)$ is the set of runs $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \ldots \xrightarrow{a_n} s_n$ of M such that there exists states $q_0, \ldots, q_n$ of $\mathcal{A}$ with $(s_0, q_0) \xrightarrow{a_1} (s_1, q_1) \xrightarrow{a_2} (s_2, q_2) \ldots \xrightarrow{a_n} (s_n, q_n)$ being a run of $M \parallel \mathcal{A}$.*

We will be interested in definitions of security that classify a machine M as secure or insecure when it is scheduled according to a scheduler σ. We assume agents have a *synchronous* view of the machine, making an observation at each moment of time and being aware of clock ticks. We permit that the agents are aware of the scheduler being used, but may not have complete information concerning the schedule in a particular run.

As already mentioned, we confine ourselves to the policy $L \leq H$, which permits information to flow from L to H but not vice-versa. To prevent the scheduler providing a channel for the prohibited information flow, we need to ensure that the schedules obtained, which may be observable to L, do not provide information to L about H's activity. To this end, we focus on H-oblivious schedulers, in which schedules do not carry any information about H actions.

Now, as implemented, the compound machine will have the form $M \parallel \mathcal{A}$ where $\mathcal{A}$ is some scheduler SOLTS representing σ, so on first principles, we should define security as a predicate on these composite systems. This raises a concern: is the security of the scheduled machine sensitive to the choice of implementation $\mathcal{A}$? Intuitively, this should not be the case: the role of the scheduler is only to enable and disable agent activity, and its internal state is made invisible to the agents H and L, so what matters is the set of possible schedules, not the implementation details of how these schedules are generated. We say that a security property X is *implementation independent* if for all H-oblivious schedulers σ and all H-oblivious scheduler SOLTS $\mathcal{A}_1, \mathcal{A}_2$ representing σ, the machine $M \parallel \mathcal{A}_1$ satisfies X iff $M \parallel \mathcal{A}_2$ satisfies X. We seek properties for which this is the case. In this case, we may view X as a set of pairs (M, σ), where M is a machine and σ is a scheduler. Sometimes for readability we also write $X(\sigma)$ for the set of machines M such that $(M, \sigma) \in X$.

3 Trace-based Security Definitions

In this section, we present a number of definitions of security that adapt some notions of security from the literature on asynchronous systems. The common

feature of these definitions is that they can be defined using a trace-based semantics of machines.

3.1 A Trace Set Semantics

All of the trace-based definitions we give can be stated with respect to a weaker notion of semantics than SOLTS, in a way that easily leads to the property of scheduler independence. To clarify this, we define the notion of *system*, which is a tuple $\mathcal{I} = (\mathcal{R}, \{view_u\}_{u \in D})$ consisting of a set $\mathcal{R}$ and functions $view_u : \mathcal{R} \to V$ where V is some set. Intuitively, $\mathcal{R}$ represents the set of possible states of the world, and $view_u(r)$ for $r \in \mathcal{R}$ represents the information u has about r. We say β is a possible view for $u \in D$ in $\mathcal{I}$ if there exists $r \in \mathcal{R}$ with $view_u(r) = \beta$.

Systems can be generated both from SOLTS and from pairs consisting of a SOLTS and a scheduler. In particular, given a SOLTS M, we define the system $\mathcal{I}(M) = (\mathcal{R}(M), \{view_u\}_{u \in D})$ where the functions $view_u : \mathcal{R}(M) \to O(A_{u^+}O)^*$ are inductively defined by $view_u(s_0) = obs_u(s_0)$, and

$$view_u(\alpha \cdot a \cdot s) = \begin{cases} view_u(\alpha) \cdot a \cdot obs_u(s) & \text{if} \quad a \in A_u \\ view_u(\alpha) \cdot \frown \cdot obs_u(s) & \text{otherwise} \end{cases}$$

where $\alpha \in \mathcal{R}(M)$ and $A_{u^+} = A_u \cup \{\frown\}$. Intuitively, this says that an agent's view of a run is the log of all its observations as well as its own actions in the run, and "$\frown$" is where an action not from u is performed.

The scheduling of an agent's own actions are visible in its view, but this may leave the agent uncertain as to the scheduling of the other agents. Say that u is *schedule aware* in (M, σ) if for all runs $r, r' \in \mathcal{R}(M, \sigma)$ with $view_u(r) = view_u(r')$ we have $\varsigma(r) = \varsigma(r')$. In particular, every u is schedule aware in (M, σ) with deterministic σ. For a pair (M, σ), the definition is given similarly by $\mathcal{I}(M, \sigma) = (\mathcal{R}(M, \sigma), \{view_u\}_{u \in D})$, with the identical definition of the view functions except that the domain is now $\mathcal{R}(M, \sigma)$.

All of the trace based security definitions we give can be stated as properties X of a system $\mathcal{I}$. The definitions satisfy the following condition:

> Let M be a machine, let σ be a (H-oblivious) scheduler and let $\mathcal{A}$ be a scheduler SOLTS representing σ. Then $\mathcal{I}(M \parallel \mathcal{A})$ satisfies X iff $\mathcal{I}(M, \sigma)$ satisfies X.

This leads immediately to the implementation independence of the definitions with respect to the schedulers. As the above condition is straightforward to check for each of the definitions, we give the statements directly in terms of (M, σ) and leave the verification of the above property to the reader.

3.2 Nondeducibility On Inputs

For asynchronous systems, the notion of *nondeducibility on inputs* [Sut86] states that a system is secure if L cannot deduce from its view any information about the sequence of H actions that have been performed. We would like to formulate a similar definition for systems that are subject to a scheduler. There are a number of subtleties that lead us to state several different definitions.

One difference in the synchronous case is that, using its knowledge of the scheduler, L can make deductions about the number of H actions that may have been performed. It may also be able to deduce when these actions occurred. The following definition abstracts from these concerns by focussing on the possible infinite sequences of H actions that are consistent with L's information.

Definition 2. $(M,\sigma) \in \mathtt{tNDI}_1$ *if for all possible* L *views* β *in* $\mathcal{I}(M,\sigma)$ *and* H *sequences* $\alpha \in A_H^\omega$*, there is a run* $r \in \mathcal{R}(M,\sigma)$ *such that* $view_L(r) = \beta$ *and* $Act_H(r)$ *is a prefix of* α*.*

Intuitively, this definition says that L is never able to rule out α as the sequence of actions that will be performed by H over time. This definition does not take into account the fact that L may be able to determine from its view some constraints on the number of H actions that have been performed in the run. Plainly, the number of H actions cannot be more than the number of observations in the view. However, knowledge of the scheduler may enable L to further restrict this set of possibilities, or even to determine the exact number of H actions. Given a possible view β of $\mathcal{I}(M,\sigma)$, define the set of *possible numbers of* H *actions* $\mathtt{Pna}_H(M,\sigma,\beta)$ to be the set of numbers n such that there exists $r \in \mathcal{R}(M,\sigma)$ with $view_L(r) = \beta$ and $|Act_H(r)| = n$. The intuition for the next definition is that the possible numbers of H actions should be all that L knows about the H actions.

Definition 3. $(M,\sigma) \in \mathtt{tNDI}_2$ *if for all possible* L *views* β *in* $\mathcal{I}(M,\sigma)$ *and sequences of* H *actions* $\alpha \in A_H^*$ *with* $|\alpha| \in \mathtt{Pna}_H(M,\sigma,\beta)$*, there exists* r *in* $\mathcal{R}(M,\sigma)$ *such that* $Act_H(r) = \alpha$ *and* $view_L(r) = \beta$*.*

The above definition says that we may change the sequence of H actions in a run to a sequence of the same length without changing the L view. However, the fact that there is nondeterminism in the scheduler leaves open the possibility that the new sequence of H actions may need to be scheduled in a different way in order to preserve the L view. The following definition says that the change may be made without changing how the H actions are scheduled.

Definition 4. $(M,\sigma) \in \mathtt{tNDI}_3$ *if for all* $r \in \mathcal{R}(M,\sigma)$*, and* $\alpha \in A_H^*$ *with* $|\alpha| = |Act_H(r)|$*, there exists a run* r' *with* $\varsigma(r) = \varsigma(r')$ *and* $view_L(r') = view_L(r)$ *and* $Act(r') = \alpha$*.*

The following result gives some relationships between these definitions.

Proposition 2.

1. $\mathtt{tNDI}_3 \subseteq \mathtt{tNDI}_2 \subseteq \mathtt{tNDI}_1$.
2. $(M,\sigma) \in \mathtt{tNDI}_1$ *iff* $(M,\sigma) \in \mathtt{tNDI}_2$ *iff* $(M,\sigma) \in \mathtt{tNDI}_3$*, given* L *schedule aware in* M*.*

3.3 Nondeducibility on Strategies

Nondeducibility represents an attack model in which it is assumed that L is the attacker and H is a trusted agent that may engage in any of its possible behaviours. A stronger attack model is to consider situations where H may be a Trojan horse or insider that is attempting to pass information to L. By engaging in specific behaviour, known to L, it may be possible for the insider to pass information to L. Wittbold and Johnson [WJ90] showed by example that nondeducibility is too weak for this type of attack, and proposed an alternative definition called *nondeducibility on strategies*. In asynchronous systems, nondeducibility on strategies turns out to be equivalent to nondeducibility on inputs [FG95,vdMZ06]. However, Wittbold and Johnson's example concerns synchronous systems with simultaneous actions. It is therefore of concern to check how this notion behaves on scheduled synchronous systems.

Define an H strategy to be a function $\pi : O(A_{H^+}O)^* \to A_H$. Intuitively, given a view β, the action $\pi(\beta)$ is the action that H would perform if it is scheduled after making view β. A run $r = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} s_n$ is consistent with π if $dom(a_i) = H$ implies $a_i = \pi(view_H(r_{i-1}))$ for all i. Write $\mathcal{R}(M, \sigma, \pi)$ for the set of runs in $\mathcal{R}(M, \sigma)$ that are consistent with π. Similarly, write $\mathcal{I}(M, \sigma, \pi)$ for $(\mathcal{R}(M, \sigma, \pi), \{view_u\}_{u \in D})$.

Definition 5. $(M, \sigma) \in \mathtt{tNDS}_1$ *if for all* $r \in \mathcal{R}(M, \sigma)$ *and* H *strategy* π*, there exists* $r' \in \mathcal{R}(M, \sigma, \pi)$ *such that* $view_L(r) = view_L(r')$.

As above, it is also of interest to consider the security of a system when L may learn the schedule producing a particular run. This leads to the following stronger definition.

Definition 6. $(M, \sigma) \in \mathtt{tNDS}_2$ *if for all* $r \in \mathcal{R}(M, \sigma)$ *and* H *strategy* π*, there exists* $r' \in \mathcal{R}(M, \sigma, \pi)$ *such that* $view_L(r) = view_L(r')$ *and* $\varsigma(r) = \varsigma(r')$.

The following result gives some relationships between these notions and those of the previous section.

Proposition 3.

1. $\mathtt{tNDS}_2 \subseteq \mathtt{tNDS}_1$.
2. *If* L *is schedule aware in* (M, σ)*, then* $(M, \sigma) \in \mathtt{tNDS}_1$ *iff* $(M, \sigma) \in \mathtt{tNDS}_2$.
3. $\mathtt{tNDS}_1 \subseteq \mathtt{tNDI}_1$ *and* $\mathtt{tNDS}_2 \subseteq \mathtt{tNDI}_3$.

The containment $\mathtt{tNDS}_2(\sigma) \subseteq \mathtt{tNDI}_3(\sigma)$ is strict even on deterministic schedulers. Consider the example in Fig. 1(a) with $A_H = \{h, h'\}$, $A_L = \{l\}$, and the deterministic scheduler σ with schedule $(L \cdot H)^\omega$. It is obvious that the system is in $\mathtt{tNDI}_3(\sigma)$ but not in $\mathtt{tNDS}_2(\sigma)$ if H can distinguish s_1 and s_2.

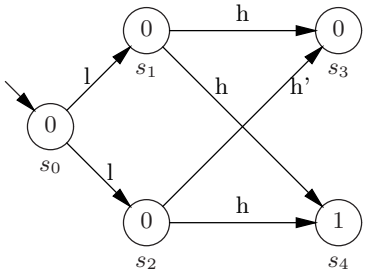

Fig. 1. (a) (M, σ) in $\mathtt{tNDI}_3$ but not in $\mathtt{tNDS}_1$ (b) (M, σ) in $\mathtt{tNDS}_1$ but not in $\mathtt{tNDI}_2$

The statement $\mathtt{tNDS}_1 \subseteq \mathtt{tNDI}_2$ does not hold in general. Consider the system M in Fig. 1(b), controlled by a nondeterministic scheduler σ producing schedules $(H + Sys)Sys(H + L)^\omega$. Any L observation is compatible with any H strategy, because the schedules $SysSys(H+L)^\omega$ produces all possible L views independent of any action by H. However, the system is not in $\mathtt{tNDI}_3$ because if L learns that the schedule is among $HSys(H + L)^\omega$, it can determine from the view $0 \frown 0 \frown 0$ that the first H action was h. Moreover, it is not in $\mathtt{tNDI}_2$ because there does not exist a run r with $view_L(r) = 0 \frown 0 \frown 0$ and $Act_H(r) = h'$, although $|h'| \in \mathtt{Pna}_H(M, \sigma, 0 \frown 0 \frown 0)$. Together with the witness of Fig. 1(a) which is a machine in $\mathtt{tNDI}_2(\sigma)$ but not in $\mathtt{tNDS}_1(\sigma)$, we have the following result.

Proposition 4. $\mathtt{tNDS}_1 \not\subseteq \mathtt{tNDI}_2$ *and* $\mathtt{tNDI}_2 \not\subseteq \mathtt{tNDS}_1$.

4 Bisimulation-based Definitions

Restrictiveness (RES) is a security property introduced by McCullough [McC88]. In state-observed systems, it can be characterised by the existence of an *unwinding relation*, which is a binary relation $\approx$ on the set of reachable system states [vdMZ07]. In asynchronous systems, the unwinding relation is essentially a bisimulation relation treating L's inputs as external actions, and H actions do not cause changes distinguishable by L. The conditions are as follows:

(OC) if $s \approx s'$ then $obs_L(s) = obs_L(s')$
(SC) if $s \approx s'$ and $s \xrightarrow{a} t$ for $a \in A_L$, then there exists a state t' such that $s' \xrightarrow{a} t'$ and $t \approx t'$; and if $s \approx s'$ and $s' \xrightarrow{a} t'$ for $a \in A_L$, then there exists a state t such that $s \xrightarrow{a} t$ and $t \approx t'$
(LR_a) for all reachable states s, t and actions $a \in A_H$, if $s \xrightarrow{a} t$ then $s \approx t$.

We would like to formulate a similar notion in scheduled systems. Since we allow that L is aware that an H action has been scheduled, the condition LR_a is too strong. We reformulate the definition so as to permit L to distinguish that H (or Sys) has performed an action, but masks *which* action has been performed. We have two different variants of LR_a, corresponding to the assumptions that L may or may not know the schedule.

Definition 7. *Given a SOLTS* $M = \langle S, S_0, \rightarrow, O, obs\rangle \in \mathbb{L}^\circ$,

1. A *insensitive synchronous unwinding relation* is a relation $\approx\ \subseteq S \times S$ satisfying OC, SC and LR, where LR is defined as: *for all reachable states* s, s' *with* $s \approx s'$ *and actions* $a, b \in A_H \cup A_{Sys}$, *if* $s \xrightarrow{a} t$ *and* $s' \xrightarrow{b}$ *then there exists* $s' \xrightarrow{b} t'$ *such that* $t \approx t'$; *if* $s' \xrightarrow{b} t'$ *and* $s \xrightarrow{a}$ *then there exists* $s \xrightarrow{a} t$ *such that* $t \approx t'$.
2. A *sensitive synchronous unwinding relation* is a relation $\approx\ \subseteq S \times S$ satisfying OC, SC, LR_H and LR_{Sys}, where $\mathrm{LR}_H(\mathrm{LR}_{Sys})$ is defined as: *for all reachable states* s, s' *with* $s \approx s'$ *and actions* $a, b \in A_H(A_{Sys})$, *if* $s \xrightarrow{a} t$ *then there exists* $s' \xrightarrow{b} t'$ *such that* $t \approx t'$; *if* $s' \xrightarrow{b} t'$ *then there exists* $s \xrightarrow{a} t$ *such that* $t \approx t'$.

One major difference between RES and NDI/NDS is that the definition of RES requires an explicit representation of *states and transitions*, which is more discriminative than the notion of sets of *runs* required for NDI/NDS. In order to formulate a version of RES for a scheduled system (M, σ) we need to apply the notion of unwinding relation to the SOLTS $(M \parallel \mathcal{A})$ where $\mathcal{A}$ represents σ. An apparent problem is that whereas unwinding is sensitive to bisimilarity, different scheduler SOLTS representing the scheduler σ need not be bisimilar.

Fig. 2 is an example which shows that there exists a machine M and a scheduler σ such that for different implementations of σ, the composed system may have different results with respect to the existence of an insensitive synchronous unwinding relation. The two scheduler SOLTS above give the schedules $H(H + Sys)(H + L)^\omega$, (we omit the states with heights greater than 2 since the first two steps suffice for our purpose) with each state labelled by the name of

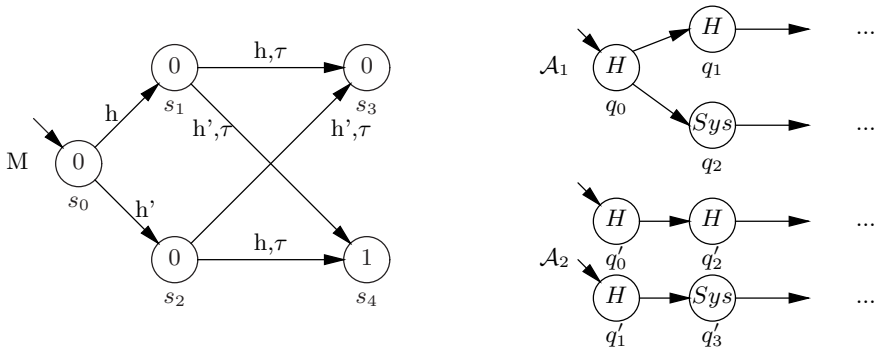

Fig. 2. A system composed with two scheduler SOLTS representing a same scheduler

the scheduled domain. The states of the machine are labelled by L observations. One may easily observe that there is an insensitive synchronous unwinding relation on $M \parallel \mathcal{A}_1$, in particular we have $(s_0, q_0) \approx (s_0, q_0)$, $(s_1, q_1) \approx (s_2, q_2)$ and $(s_1, q_2) \approx (s_2, q_1)$; however, there is no insensitive synchronous unwinding relation on $M \parallel \mathcal{A}_2$. An example for sensitive unwinding relation can be created in a similar way.

The dependence on scheduler implementation suggests that in order to obtain an implementation-independent, bisimulation-based definition of security, we should quantify over scheduler implementations. We could do this either by a *universal* or by an *existential* quantification over scheduler implementations. Which is preferred depends on one's attitude to bisimulation based definitions of security. One attitude is that the property most of interest is given by a trace-based definition, but bisimulation-based definitions provide a useful *proof technique*. In this case, an existential quantification is appropriate. On the other hand, if one adheres to an understanding in which existence of an unwinding is what makes a system secure, then it is necessary to quantify universally over scheduler implementations in order to obtain an implementation-independent notion of security. We define both variants:

Definition 8. *1.* $(M, \sigma) \in \mathtt{tRES}_1^{\forall}(\mathtt{tRES}_1^{\exists})$ *if there exists an insensitive synchronous unwinding relation on the SOLTS* $M \parallel \mathcal{A}$ *for all (some)* H*-oblivious* $\mathcal{A}$ *representing* σ*.*

2. $(M, \sigma)) \in \mathtt{tRES}_2^{\forall}(\mathtt{tRES}_2^{\exists})$ *if there exists a sensitive synchronous unwinding relation on the SOLTS* $M \parallel \mathcal{A}$ *for all (some)* H*-oblivious* $\mathcal{A}$ *representing* σ*.*

As is the case on the asynchronous systems [vdMZ07], the $\mathtt{tRES}_i^Q$ (for $Q = \forall, \exists$) are the strongest of the security properties we have discussed so far, with the following relations holding.

Proposition 5. *1.* $\mathtt{tRES}_i^{\forall} \subseteq \mathtt{tRES}_i^{\exists} \subseteq \mathtt{tNDS}_i$ *for* $i = 1, 2$*.*

2. $\mathtt{tRES}_2^Q \subseteq \mathtt{tRES}_1^Q$ *for* $Q = \forall, \exists$*.*

That the containments of $\mathtt{tRES}_i^{\forall} \subseteq \mathtt{tRES}_i^{\exists}$ are proper follows from the failure of implementation independence shown above. To show the containments of $\mathtt{tRES}_i^{\exists} \subseteq \mathtt{tNDS}_i$ are proper, we present the system in Fig. 3 where $A_H = \{h, h'\}$, $A_L = \{l\}$, and the deterministic scheduler σ contains the schedule $(HL)^\omega$. If

"≈" were a (in)sensitive synchronous unwinding, we would have that the states at level 2 are related by "≈". However, these states are not bisimilar, as they do not satisfy the $\mathrm{LR}_H(\mathrm{LR})$ property if we relate any two of them. We may also observe there is no H strategy to pass information to L regardless of H and L's observational power.

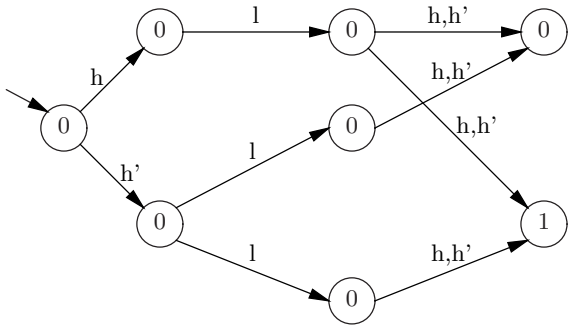

Fig. 3. (M, σ) in $\mathtt{tNDS}_1$ and $\mathtt{tNDS}_2$, but not in $\mathtt{tRES}_1^{\exists}$ or $\mathtt{tRES}_2^{\exists}$

5 Conclusion and Related Work

Our focus has been on the interaction of schedulers with notions of information flow security. There appears to be relatively little literature on this topic.

One related set of papers is Rushby's work on separability [Rus81,Rus00], which aims to define a security property for operating systems security kernels. A separation kernel provides each user the abstraction of a local abstract machine, which is unaffected by the behaviour of other users. Rushby's definitions assume that agent's views are defined in an asynchronous fashion, but his machine model is based on a notion of "colours" that amounts to scheduling of the processes. The information flow policy for separability is stronger than that we have considered (since it prohibits information flow in both directions). It would be interesting to revisit his work in the light of our results in this paper.

Another work in which schedulers are explicitly considered is that of Sabelfeld et al. [SS00], who present an elegant formulation of language based security properties with dynamic thread handling by schedulers, with a probabilistic bisimulation-based definition of security. Their schedulers make decisions either deterministically or probabilistically, but they consider only a single input and output, and treat timing as unobservable to Low (but observable to the scheduler). Their definition of security is independent of *all* schedulers, compared with our focus on independence on implementation for a known scheduler. Russo et al, in a sequence of papers (a recent one is [RS06]) have studied a similar setting for non-probabilistic bisimulation-based definitions of security.

In addition to the focus on schedulers, our work differs from much of the literature in the use of a synchronous rather than asynchronous notion of process view. A number of recent works have also considered this direction. Focardi et al. extended the asynchronous definitions of security of [FG95] from the CCS-like setting of the process algebra SPA into a timed version in the framework called tSPA, based on a discrete timed weak bisimulation [FGM00]. They consider

bisimulation based definitions of security that are based on NDS-like intuitions. In [HA06], Huang et al. restated a set of failure-divergence based and low determinism based definitions [Ros95] in a version of timed CSP. The underlying model of timed CCS and timed CSP seems to be essentially more general than ours, in the sense that every SOLTS can be cast into a timed LTS with every action followed by a single *tick*. However, the SOLTS model is technically more suitable to express the effect of a system being controlled by a scheduler via a lock-step synchronization with a scheduler SOLTS, assuming each step taking unit time.

We intend to conduct a more detailed technical comparison between these works and our definitions in future work. We also intend to study the complexity of verification the notions of security we have introduced above.

References

[BY94] William R. Bevier and William D. Young. A state-based approach to noninterference. In *Computer Security Foundations Workshop*, pages 11–21, 1994.

[FG95] Riccardo Focardi and Roberto Gorrieri. A classification of security properties for process algebras. In *Journal of Computer Security*, 1, pages 5–33. 1995.

[FGM00] Riccardo Focardi, Roberto Gorrieri, and F. Martinelli. Information flow analysis in a discrete-time process algebra. In *Computer Security Foundations Workshop*, pages 170–184. IEEE Computer Society Press, 2000.

[GM82] J.A. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symp. on Security and Privacy*, pages 11–20, April 1982.

[HA06] J. Huang and A.W.Roscoe. Extending noninterference properties to the timed world. In *ACM Sym. on Applied Computing*, pages 376–383, 2006.

[McC87] Daryl McCullough. Specifications for multi-level security and a hook-up property. In *IEEE Symp. on Security and Privacy*, pages 161–166, 1987.

[McC88] Daryl McCullough. Noninterference and the composability of security properties. In *IEEE Symp. on Security and Privacy*, pages 177–186, 1988.

[Ros95] A.W. Roscoe. CSP and determinism in security modelling. In *IEEE Symp. on Security and Privacy*, pages 114–221, 1995.

[RS06] Alejandro Russo and Andrei Sabelfeld. Securing interaction between threads and the scheduler. In *CSFW*, pages 177–189. IEEE Computer Society, 2006.

[Rus81] John Rushby. The design and verification of secure systems. In *8th ACM Symp. on Operating System Principles*, pages 12–21, Dec 1981.

[Rus82] John Rushby. Proof of separability a verification technique for a class of security kernels. In *5th International Symp. on Programming, Turin, Italy*, pages 352–367, April 1982.

[Rus00] John Rushby. Partitioning in avionics – architectures: Requirements, mechanisms, and assurance. Technical report, SRI international, March 2000.

[SS00] Andrei Sabelfeld and David Sands. Probabilistic noninterference for multithreaded programs. In *IEEE Computer Security Foundations Workshop*, pages 200–214, 2000.

[Sut86] D. Sutherland. A model of information. In *Proc. 9th National Computer Security Conference*, pages 175–183, 1986.

[vdMZ06] Ron van der Meyden and Chenyi Zhang. A comparison of semantic models for noninterference, 2006. 4th International Workshop on Formal Aspect in Security and Trust (FAST'06).

[vdMZ07] Ron van der Meyden and Chenyi Zhang. Algorithmic verification on noninterference properties. In *ENTCS*, volume 168, pages 61–75. Elsevier, 2007.

[WJ90] J. T. Wittbold and D. M. Johnson. Information flow in nondeterministic systems. In *IEEE Symp. on Security and Privacy*, pages 144–161, 1990.