Information Flow in Systems with Schedulers (Part I: Definitions)[1]

Ron van der Meyden

Chenyi Zhang[2]

School of Computer Science and Engineering,
The University of New South Wales, Sydney, Australia

Abstract

This paper studies information flow security in a synchronous state machine model, in which agents share a global clock and can make observations at all times, but in which an agents' ability to perform actions is subject to a scheduler. A number of definitions of security for this setting are proposed, depending on whether the attacker is active or passive, whether the security should be robust to discovery of the schedule by the attacker, and on whether the definition is trace-based or bisimulation-based. In particular, the paper studies the dependence of these definitions of security on implementation details of the scheduler. Such independence is shown to hold for the trace-based definitions, but not for bisimulation-based definitions. Stronger versions of the bisimulation-based definitions are proposed that recover implementation-independence. A complete characterization of relationships between the definitions of security introduced in the paper is derived.

1. Introduction

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

[1]NOTICE: this is the authors' version of a work that was accepted for publication in Theoretical Computer Science, Vol. 467, Jan 2013, pp. 68-88. DOI information: 10.1016/j.tcs.2012.10.047. Changes resulting from the publishing process, such as peer review, editing, corrections, structural formatting, and other quality control mechanisms may not be reflected in this document. Changes may have been made to this work since it was submitted for publication.

Part I of this two part series of papers is a revised and extended version of [vdMZ08] with all the proofs included. In addition, we have shown that all the properties we have proposed in the conference paper are distinct with nontrivial examples. Part II consists of a revised version of previously unpublished material from the thesis [Zhang09].

[2]Present address: School of Information Technology and Electrical Engineering, The University of Queensland, Brisbane, Australia

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 [McCullough88, BY94] and in richer semantic models such as the process algebras CCS [FG95, FPR02, HR02] and CSP [Ryan91, Roscoe95]. Although a few early works considered synchronous models [WJ90, Millen90, BC92, Gray91, McLean94], work in the process algebraic setting has, until recently [FPR02, BFPR03, BG09], concentrated attention on asynchronous models of computation, assuming that agents do not have access to a system clock.

While an asynchronous modelling has been argued to be appropriate for distributed systems [FG95], it also ascribes relatively weak powers to the adversary. Many information flow attacks that are of interest in practice are based on timing channels, often resulting from contention for shared resources [Gligor93]. An example of channels arising from resource sharing is the original motivation for the notion of noninterference, operating systems separation kernels [Rushby81, Rushby82, Rushby00]. The function of this class of software systems is to separate security domains in order to prevent information flow resulting from the use of shared resources such as memory (including registers, caches, main memory and disks) input-output devices (e.g., network cards), and processing units. 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 in early work in the literature [GM82, Rushby92] 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. The later process algebraic work has developed definitions of security that drop this assumption, but even the weakest definitions considered (e.g., the weakest notion NNI in a spectrum of definitions treated in [FG95]) have the property that every possible sequence of observations of a Low security agent in a secure system should be consistent with the High security agent having performed no input actions. This property fails in systems subject to a scheduler. For example, if the scheduler alternates the High agent and the Low agent, starting with High, then when Low is first scheduled it knows that some High actions have already been processed (even if it cannot discern *which* actions). More generally, knowledge of the schedule may permit the Low agent to deduce the *number* of actions that the High agent has taken. 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. Both the relevance of timing attacks and the fact that scheduling may imply an ability to deduce the number of actions of another agent suggest that a synchronous model is more appropriate.

In this two-part series of papers, we conduct a systematic study of the impact of schedulers and synchrony on a variety of approaches to the definition of

information flow security from the literature. In the present paper, part I of the series, we ask how these approaches can accommodate schedulers, and study the relationships between the resulting definitions of security. A companion paper, Part II of the series, will consider which of these definitions are preserved under refinement of schedulers, and applies the results to characterize definitions that state that a system is secure with respect to *all* schedulers. We focus in the remainder of this introduction on the contents of part I.

We begin by developing a formal model of systems running with a global clock visible to all agents, subject to a scheduler that schedules one agent at a time. Agents have actions and make observations of the state. We define schedulers at two levels of abstraction. At the abstract level, a scheduler is a rule for deciding which of the agents may execute an action at the next step as a function of the history of actions executed in the past. At the concrete level, a scheduler may be represented by a labelled transition system which has each state representing a scheduled agent by means of the actions enabled at the states reachable after the history of actions. We discuss the relationships between these two modellings. To allow for freedom of implementation and to model a discrete approximation of randomised scheduling, we allow schedulers to be nondeterministic.

We next develop new variants of several definitions of security from the literature on information flow in asynchronous systems, that are tailored for our setting of scheduled systems. We focus on systems with three agents H (representing the high security, or secret level), L (representing the low security, or public level), and Sys (representing events of the system within which H and L operate that are not attributable to either agent; this includes actions of the scheduler). The definitions of security we consider provide different formal answers to the meaning of a security policy that says that information (more precisely, information about which actions have been executed) is permitted to flow from L to H, but not from H to L. Specifically, we develop variants of the following types of definitions from the literature:

1. *nondeducibility on inputs* [Sutherland86], which is appropriate for settings where L acts as an outside observer attempting to infer, from its observations, information about H activity,
2. the stronger notion of *nondeducibility on strategies* [WJ90], which takes into account that L may have placed a Trojan horse at H, and requires that no flow of information from H to L is possible even if this is the case, and
3. *restrictiveness* [McCullough87, McCullough88], a definition stronger than both the above, which is closely related to the *unwinding* [GM84] proof technique for noninterference, and one of Focardi and Gorrieri's bisimulation based definitions of security [FG95].

Nondeducibility on inputs and nondeducibility on strategies can be stated using a trace-based semantics, but restrictiveness is a bisimulation-like definition that refers to the branching structure of the system. In developing variants of these notions that are suited for our setting of scheduled synchronous systems, we find that there is more than one plausible candidate for each of these notions in

a setting with nondeterministic schedulers. The main issue distinguishing these candidates is the question of whether the system should remain secure were the schedule for a given run to become known to L. (Agent L will always know when its own actions were scheduled, but because there are two other agents H and Sys, it may still have uncertainty about when H was scheduled; learning the schedule resolves this uncertainty and may give L more information.) One of our types of definition requires that a secure system remain secure even in case of such a release of the schedule, the other does not.

One question of particular concern is the extent to which these definitions are sensitive to how the scheduler is implemented. A scheduler in the abstract modelling may be implemented in many different ways in the concrete modelling. For example, nondeterminism in the abstract scheduler may be resolved in the concrete automaton representation either 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 our (trace-based) variants of nondeducibility on inputs and nondeducibility on strategies are independent of the scheduler implementation, but this is not the case for our (bisimulation-based) variants of restrictiveness. This means that, whereas it is possible to state the definitions of our variants of nondeducibility on inputs and nondeducibility on strategies with respect to our abstract notion of scheduler, our variants of restrictiveness must be stated at the concrete level, with respect to a particular scheduler implementation. This motivates our introduction of two stronger variants of restrictiveness (based on an existential or universal quantification over scheduler implementations) that we can show to be invariant with respect to scheduler implementation. (Together with the issue of security were the schedule to be discovered, this gives four different notions of restrictiveness stated at the level of abstract schedulers.)

We give a complete characterization of the relationships between all our versions of these notions of security, showing implications and giving examples where implications fail. One of the results of this study is that whereas, for asynchronous systems, nondeducibility on inputs and nondeducibility on strategies turn out to be identical, this fails for our synchronous variants.

The structure of the paper is as follows. In Section 2, we sketch a simple example where information flow properties are sensitive to time and scheduling. In Section 3 we introduce the semantic framework within which we work, a type of labelled transition system with observations. Section 4 defines schedulers and their representation within this model. Section 5 deals with trace-based definitions of security for systems with schedulers. Section 6 considers bisimulation-based definitions, where independence of the scheduler implementation becomes a non-trivial issue. Section 7 discusses related work in the area of noninterference with time and/or scheduling, and Section 8 concludes and proposes future directions.

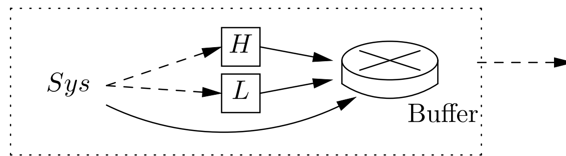

Figure 1: Motivating Example

2. A Motivating Example

Suppose we have the following scenario, as illustrated in Figure 1. Two processes H and L are controlled by an operating system Sys that is responsible for communicating messages to the network as well as scheduling the activity of the two processes. A process may request that a message be sent by writing it into a buffer. The buffer is shared between H, L and the system itself, and has a capacity of at most one message. (Intuitively, H performs actions from a high security level agent and L represents a low security level agent.)

The system is prompt to respond to requests from H and L: if there is a message from one of these processes in the buffer when the system regains control, it immediately dispatches the message and clears the buffer. However, the system makes greedy use of the buffer for its own purposes: whenever there is no message in the buffer, it places its own data in the buffer, which it may keep there for any period of time that it chooses. To counteract this, processes are allowed to preempt the system's use of the buffer. This is done in a two-step process: when the process attempts to write to the buffer and it contains Sys data, the buffer is first emptied and the process is sent a "retry" message from the system. If it then re-attempts its write to the buffer while it is clear, the write will be successful. On the other hand, when H or L tries to write a message to the buffer and it already contains a message from either process, its request is rejected and a "full" message returned.

In this example, the scheduler implemented in the system is important to information flow security analysis. Suppose that the security policy allows information to flow from L to H but not from H to L. Then, assuming the private state of the buffer is invisible to L, we have the following observations.

- Suppose the scheduler is a round-robin $HLSysHLSys\ldots$. Consider runs in which L attempts to write to the buffer each time that it is scheduled. Note that this guarantees that there will be a message in the buffer each time Sys is scheduled, so Sys will clear the buffer before passing control to H. Each time H fills the buffer, an attempt by L to write to the buffer in the next step will be rejected with a "full" message. On the other hand, if H does not write to the buffer, it will be empty, so L's attempt to write to the buffer will succeed. L will therefore be able to deduce, from the result of its write attempt, what action H performed in the preceding step. Intuitively, this means that the system is insecure: L is able to deduce information about H, so there is a flow of information from H to L, in contravention of the policy. Moreover, a Trojan horse at H could

use this as a covert channel to pass H secrets to L, by choosing to send or not send a message depending on the value of a bit of the secret.

- If the scheduler repeats the pattern $HHSysSysSysLLSysSysSys\ldots$, then, there is not such a flow of information, and, we claim, the system is secure. If H has filled the buffer during one of its steps, the system will transmit the message and clear the buffer in its first step. It will then greedily grab the buffer in its second step, and may or may not release it in its third step. Thus, in this case, L's attempt to write to the buffer will return either a "retry" or "success" message. On the other hand, if H did not fill the buffer, then Sys will grab the buffer in its first step, and may or may not have released it by its final step. In this case also, L's attempt to write to the buffer will return either a "retry" or "success" message. Thus, L will not be able to deduce anything about H activity from its observations[3]. (Furthermore, each agent is guaranteed to be able to transmit one message each time it is scheduled: if it tries to fill the buffer in its first step, but the buffer contains Sys data, it will receive a "retry" message, and the buffer will be cleared. A second attempt to fill the buffer in the next step will then be successful.)

- Interestingly, the greediness of the system means that if we use only *two* Sys steps between alternating scheduling of H and L (so that the schedule becomes $HHSysSysLLSysSysHH\ldots$) then if L tries to send a message and it is accepted then L can deduce that H did *not* send a message (else this message would have been transmitted in the first Sys step, and the buffer filled by Sys in the second Sys step, so that L would have received a "retry" message). This version is insecure.

Note that all the above schedulers are independent of the behaviours of the processes, so the reason for the insecurities is not that the scheduler directly passes information about H actions to L by altering the times at which L is scheduled.

3. A Discrete-Time System Model

We are concerned in this paper in systems that are *synchronous* and *timed* in the sense that there is a discrete global clock shared by all agents, and all agents are able to continue making observations at all times, including times when they are not scheduled to perform an action. As a formal model of such settings, we use an enrichment of the well-established *labelled transition system semantics* for process algebra, adding to it a notion of observation on states. However, we do not interpret labelled transition systems asynchronously, as is usually done. Our synchronous and timed interpretation is reflected in two

[3]Note that we assume that Sys choice to release or hold the buffer is nondeterministic, and there is no known probability with which this happens.

aspects of the semantics. One is our definition of the *view* that an agent has of a run of the system, in which it makes an observation of the system state at each moment of time in the run, even if that state was reached as the result of another agent's action. The other is a synchronous composition operator on our enriched transition systems. Our model could be further enriched, e.g., by allowing simultaneous actions, but we do not pursue this since we are specifically interested in systems subject to a scheduler.

The term *domain* is used in the literature to refer to security levels in an information flow security policy. Since, in the context of schedulers, we refer to domains but also need a label to refer to the system itself, which is not an explicit part of the security policy, we use the the more general term *agent*. A *signature* is a tuple (A, D, dom) consisting of a set of actions A, a set of agents D and a function $dom : A \to D$ associating an agent with each action.

Definition 3.1. *A* state-observed labelled transition system *(SOLTS) for a signature* (A, D, dom) *is a tuple of the form* $T = \langle S, S_0, \to, O, obs \rangle$ *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.*

Write $\mathbb{L}^\circ$ *for the set of all such systems.*

We make three different uses of SOLTS. One is the notion of machine, defined later in this section. The others are for schedulers and for machines running under the control of a scheduler, both defined in the next section.

For readability, we 'curry' the function *obs* (or its variants) by writing $obs_u(s)$ for $obs(u, s)$. 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 $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} s_n$ and $\alpha = a_1 a_2 \dots a_n$. A *run* r of a SOLTS is a sequence of the form $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} s_n$ with $s_0 \in S_0$. We write $\mathcal{R}(T)$ for the set of all runs of T. We write r_k for the prefix of r consisting of the first k transitions, i.e., $r_k = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \dots \xrightarrow{a_k} s_k$, provided r has at least k transitions. We denote the sequence of actions in a run r by $Act(r) = a_1 a_2 \dots a_n$, 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 $a \in A$.

An agent's observation at a state in a SOLTS gives it some information about which state the system is in, and may also give it some information about the past. In fact, the agent may be able to recall all its past actions and observations, but not actions of other agents. To give an optimal analysis

of security, we assume that agents have perfect recall of their history. This is captured in the following definition of the notion of *view*. For $u \in D$, we define $A_u = \{a \in A \mid dom(a) = u\}$.

Definition 3.2. *Given a SOLTS T and an agent u, the function $view_u : \mathcal{R}(T) \to O((A_u \cup \{\frown\})O)^*$ is inductively defined by $view_u(s_0) = obs_u(s_0)$, and*

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

where $r \in \mathcal{R}(T)$, $a \in A$ and $s \in S$. We write $Views_u(T)$ for $\{view_u(r) \mid r \in \mathcal{R}(T)\}$.

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, with "$\frown$" where an action of another agent is performed. We note that implicit in this definition is an assumption of *synchrony*, in the sense that an agent can always determine from its view $view_u(r)$ of a run r what is the time (the length of r), simply by counting the number of elements of O or $A_u \cup \{\frown\}$.

Synchrony is also reflected in the way that we compose SOLTS. Given two SOLTS $T = \langle S, S_0, \to, O, obs\rangle$, $T' = \langle S', S'_0, \to', O', obs'\rangle$ with the same signature, define the parallel composition $T \parallel T'$ to be the SOLTS $T'' = \langle S \times S', S_0 \times S'_0, \to'', O \times O', obs''\rangle$ where $\to'' = \{((s_1, s'_1), a, (s_2, s'_2)) \mid s_1 \xrightarrow{a} s_2 \wedge s'_1 \xrightarrow{a} s'_2\}$ 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 common actions. We use this composition operator only as a technical device to describe the effect of schedulers (details are in the following section), rather than to view systems as being composed from smaller components.

Like most of the literature, we confine our attention to systems with two security domains H (High) and L (Low) and the security policy which permits information flow from L to H but 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 $D = \{H, L, Sys\}$, and assume there is a special action τ such that $A_{Sys} = \{\tau\}$.

The effect of Sys actions may be nondeterministic, but we assume that there is no 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 internal evolution of the state over time. On the other hand, we do not model outputs as actions; in our model outputs correspond to the observations made by the agents at the states of the system.

Definition 3.3. *A* machine *is an* input-enabled *SOLTS $M = \langle S, S_0, \to, O, obs\rangle$ for a signature (A, D, dom) with $D = \{H, L, Sys\}$ and $A_{Sys} = \{\tau\}$. We write $\mathbb{M}$ for the set of all machines.*

The restriction of input-enabledness is often applied in the security literature (eg. [GM82, Rushby92]). We do not adopt this assumption for machines running under the control of a scheduler, but it simplifies matters for us to adopt it for the bare machines that may be subject to scheduling. One reason is that it finesses the question of what would happen if an agent were scheduled in a state where it had no actions enabled. Another is that dropping this assumption raises the issue of whether, and how, the fact that an action is not enabled would provide information to an agent that it might exploit in an attack. Process algebraic approaches use a diversity of methods to capture this information, often representing it *implicitly* in the process semantics (e.g., through the failures of CSP [Roscoe95], or through a notion of bisimulation [FG95].) We prefer to take the philosophical position that any information that an agent has should be *explicitly* represented in its view. If an action cannot be "successfully" performed in some state, that fact can still be represented by a transition labelled by that action to a state where the agent makes an observation that informs it that the attempt was unsuccessful: this gives an input-enabled modelling of the situation.

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.

Diagrammatic Convention for Machines: we depict machines as graphs in which vertices correspond to states, and are labelled by the observation made by L at that state. Edges are labelled by actions and correspond to transitions. Not all transitions from a state are depicted: if the only transition with a given action is a self-loop, it may be elided. Since machines are input-enabled, where there is no edge labelled by an action a from a state s, this implies that there is a self-loop from s labelled by a. (This convention helps to reduce clutter in diagrams of machines.)

4. Schedulers

Machines can be given an asynchronous semantics, but here we are interested in the semantics in which machines execute under the control of a scheduler, which selects an agent at each moment of time. Our model of schedulers resolves only the nondetermism concerning the next agent to act: we leave this agent *free will* to choose which action to perform when it is scheduled.[4]

We allow that scheduling is nondeterministic: there are several reasons to consider this. Admitting non-deterministic schedulers gives a more gen-

[4]We note that the term "scheduler" is also used in the literature on formal models of nondeterminism to refer to a function that also resolves the nondeterminism concerning the scheduled agent's choice of action. The motivation for this is generally to reduce a system with nondeterminism to a completely deterministic or probabilistic system, so that better understood definitions and methods for the latter type of systems can be applied. This is a technical notion of scheduler that is orthogonal to our concerns in this paper.

eral theory that balances the nondeterminism in our machine model. A nondeterministic scheduler also leaves open a range of possible implementations, obtained by refining the nondeterminism; we study the impact of such refinement on security in Part II of the paper. Scheduling may also be done randomly for reasons of expected performance and fairness, as in, e.g. lottery scheduling [WW94]. Finally, nondeterministic or random scheduling may be done for reasons of security, in order to hide information from an attacker by creating uncertainty about which agent is acting. Several other works have dealt with nondeterministic or probabilistic scheduler models in the different context of security of programming languages [SS00, VS99, BC02]. (We do not attempt to deal with probabilistic systems and probabilistic notions of security in this paper, but nondeterminism can be viewed as an abstraction of probability, so our work points to some of the issues that would arise in such an extension.)

Definition 4.1. *A* scheduler *(for a machine M with actions A and domains D) is a function $\sigma : A^* \to \mathcal{P}(D)$. A* scheduled machine *is a pair (M, σ) consisting of a machine M and a scheduler σ for M.*

Intuitively, given a history of actions $\alpha \in A^*$, one of the agents in the set $\sigma(\alpha)$ will be scheduled next. (This definition leaves underspecified precisely how and when the nondeterminism in a scheduler is resolved; the notion of scheduler SOLTS defined later in this section is a more concrete modelling that allows specific mechanisms for making such decisions in a scheduler implementation to be represented.) We say that a run of a machine is *compatible* with a scheduler if the agent that acts at each step of the run is one of the agents enabled by the scheduler, given the history so far. Formally, 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 $a \in A$. An infinite action sequence is compatible with a scheduler σ if all its finite prefixes are compatible with σ. 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 also write $Views_u(M, \sigma)$ for $\{view_u(r) \mid r \in \mathcal{R}(M, \sigma)\}$.

We henceforth assume that schedulers do not terminate, so that if α is compatible with σ, then $\sigma(\alpha) \neq \emptyset$. A scheduler σ is *deterministic* if $\sigma(\alpha)$ is a singleton for all compatible $\alpha \in A^*$. Write Υ for the set of schedulers and Υ^d for the set of deterministic schedulers. A *schedule* is a finite or infinite sequence $\mathfrak{s} = u_0 u_1 u_2 u_3 \ldots$ where each $u_i \in D$. For $\alpha = a_0 a_1 a_2 \ldots$, we write $sch(\alpha)$ for the schedule $dom(a_0)\, dom(a_1)\, dom(a_2) \ldots$. If r is a run we also write $sch(r)$ for $sch(Act(r))$. We write $\mathfrak{s} \in \sigma$ if there is an action sequence α compatible with σ such that $\mathfrak{s} = sch(\alpha)$. For each infinite schedule $\mathfrak{s} = u_0 u_1 u_2 u_3 \ldots$, we define a deterministic scheduler $\sigma_{\mathfrak{s}}$, by $\sigma_{\mathfrak{s}}(\alpha) = \{u_{|\alpha|}\}$ for all $\alpha \in A^*$, such that $sch(\alpha)$ is a prefix of $\mathfrak{s}$ (where $|\alpha|$ is the length of a finite action sequence α), and $\sigma_{\mathfrak{s}}(\alpha) = \emptyset$ otherwise.

In order to prevent the scheduler being a channel for information flow, we define a notion that expresses that the decisions of the scheduler are in-

dependent 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 $\alpha = a_1 a_2 \ldots \in A^*$ 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 $\mu_u(A) = (A \setminus A_u) \cup \{\bot_u\}$. A scheduler is *oblivious* if it is u-oblivious for all $u \in D$.

Example 4.2. *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 there is a* `yield` *in the last* $|\alpha|$ *mod* k *actions in* α*,*
- $\sigma(\alpha) = \{L\}$ *otherwise.*

Here $|x|$ denotes the length of a sequence x, and div denotes integer division. Then σ is an H-oblivious scheduler, but not L-oblivious. (To see that σ is H-oblivious, note that $|\alpha| = |\mu_H(\alpha)|$ and the L action `yield` *is preserved in $\mu_H(\alpha)$. To see that σ is not L-oblivious, note that for any L action a other than* `yield`*, we have $\mu_L(a) = \mu_L(\texttt{yield}) = \bot_L$ but $\sigma(a) = \{L\}$ and $\sigma(\texttt{yield}) = \{H\}$.)*

It is worth noting that if we were to give H a yield action, with the effect of transferring the remainder of H's block to L, then there would be a covert channel and the system would be insecure, since upon seeing that it was scheduled within an H block, L would obtain information about H's actions, in contravention of the policy that information should not flow from H to L. The assumption of H-obliviousness prevents the existence of such covert channels. □

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 by the same agent, and all actions of that agent 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 agents 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.)

Diagrammatic Convention for Scheduler SOLTS: we depict scheduler SOLTS as graphs in which vertices correspond to states, and are labelled by the agent whose actions are enabled at that state. Edges are labelled by actions and correspond to transitions. All transitions from a state are depicted: we note that we use a different diagrammatic convention in the diagrams of machines, where self-loops are elided.

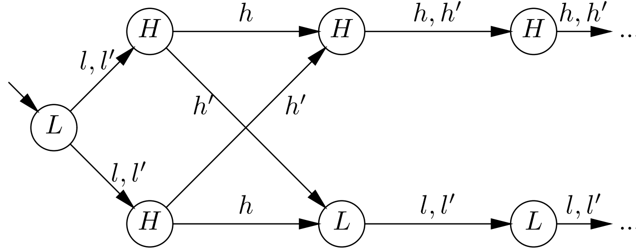

Figure 2: A non-oblivious implementation of an oblivious scheduler

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 $c \in A_u$. Intuitively, this says that the state t carries no information about which u action was used to reach it. A 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 agent that has its actions enabled at q. Now say $\mathcal{A}$ *represents* σ if for all $\alpha \in A^*$ compatible with σ, we have $\sigma(\alpha) = \sigma_{\mathcal{A}}(\alpha)$.[5]

Interestingly, a scheduler SOLTS representing a u-oblivious scheduler is not necessarily u-oblivious. Intuitively, this is because our definition of obliviousness of a scheduler SOLTS is based on just one (easily checked) way to ensure that the SOLTS represents an oblivious scheduler. In Figure 2 there is a scheduler SOLTS that is not H-oblivious, but it represents the H-oblivious scheduler that produces schedules $LH(H^\omega + L^\omega)$. (Here we follow the convention of using H^ω to denote the infinite sequence $HHH\ldots$, and similar for L^ω.) This scheduler SOLTS is not H-oblivious since at the second step, different actions from H lead necessarily to different states.

Since for every u-oblivious scheduler σ, there always exists a u-oblivious scheduler SOLTS that represents σ (by Proposition 4.4(2)), we restrict to the u-oblivious scheduler SOLTS representing σ when dealing with σ. The next construction defines a scheduler SOLTS for every scheduler.

Definition 4.3. *For every scheduler σ, define the (infinite state) characteristic scheduler SOLTS $\mathcal{A}^\sigma = \langle Q, Q_0, \rightarrow, \{\bot\}, obs\rangle$ by*

1. $Q = A^* \times D$,
2. $Q_0 = \{(\epsilon, v) \mid v \in \sigma(\epsilon)\}$,
3. $(\gamma, v) \xrightarrow{a} (\gamma', v')$ *iff* $dom(a) = v$ *and* $\gamma' = \gamma \cdot a$ *and* $v' \in \sigma(\gamma')$,
4. $obs(v, \gamma) = \bot$ *for all* $v \in D$ *and* $\gamma \in Q$.

It can be readily shown that $\mathcal{A}^\sigma$ represents σ, i.e., $\sigma(\alpha) = \sigma_{\mathcal{A}^\sigma}(\alpha)$ on α compatible with σ. If σ is u-oblivious, we define a similar construction on the state

[5]There exist schedulers σ' represented by $\mathcal{A}$ but different from $\sigma_{\mathcal{A}}$ on incompatible sequences.

space $(\mu_u(A))^* \times D$, with its transition relation defined by $(\gamma, v) \xrightarrow{a} (\gamma', v')$ iff $dom(a) = v$ and $\gamma' = \gamma \cdot \mu_u(a)$ and $v' \in \sigma(\gamma')$. We write $\mathcal{A}_u^\sigma$ for the resulting scheduler SOLTS. It will be then obvious that $\mathcal{A}_u^\sigma$ is u-oblivious. 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 4.4.

1. *Every scheduler has a scheduler SOLTS that represents it.*
2. *A 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 for which there exists states $q_0, \ldots, q_n$ of $\mathcal{A}$ such that $(s_0, q_0) \xrightarrow{a_1} (s_1, q_1) \xrightarrow{a_2} (s_2, q_2) \ldots \xrightarrow{a_n} (s_n, q_n)$ is a run of $M \parallel \mathcal{A}$.*

Proof:

- Part (1) is trivial by Definition 4.3.
- For Part (2), the 'only if' direction is by the construction of $\mathcal{A}_u^\sigma$ similar to Definition 4.3, as shown above. For the 'if' direction, given a u-oblivious scheduler SOLTS $\mathcal{A} = \langle Q, Q_0, \rightarrow, obs \rangle$, we show that the scheduler $\sigma_{\mathcal{A}}$ is u-oblivious. Given any $\alpha, \alpha' \in A^*$ satisfying $\mu_u(\alpha) = \mu_u(\alpha')$, firstly we have $|\alpha| = |\alpha'|$, then by a straightforward induction on the length of α, it can be shown that the sets of states in $\mathcal{A}$ reachable by α and α' are exactly the same set, given $\mathcal{A}$ u-oblivious, i.e., $\{q \mid q_0 \xrightarrow{\alpha} q, q_0 \in Q_0\} = \{q \mid q_0 \xrightarrow{\alpha'} q, q_0 \in Q_0\}$. Therefore we have $\{sched(q) \mid q_0 \xrightarrow{\alpha} q, q_0 \in Q_0\} = \{sched(q) \mid q_0 \xrightarrow{\alpha'} q, q_0 \in Q_0\}$, then $\sigma_{\mathcal{A}}(\alpha) = \sigma_{\mathcal{A}}(\alpha')$ by definition.
- For Part (3), let $\mathcal{A}$ represent σ and $\alpha = a_1 a_2 \ldots a_n$. Given $r = s_0 \xrightarrow{a_1} s_1 \ldots s_{n-1} \xrightarrow{a_n} s_n \in \mathcal{R}(M, \sigma)$, by definition r is a run of M. Since α is compatible with σ, we have $dom(a_1) \in \sigma(\epsilon)$, and $dom(a_i) \in \sigma(a_1 \ldots a_{i-1})$ for all $i = 2 \ldots n$. If there does not exist $q_0 \xrightarrow{\alpha} q_n$, we will have r incompatible with $\sigma_{\mathcal{A}}$, which contradicts the fact that $\mathcal{A}$ represents σ. So there is a run of $\mathcal{A}$ of the form $q_0 \xrightarrow{a_1} q_1 \ldots \xrightarrow{a_n} q_n$. Combining this run with r we get a run $(s_0, q_0) \xrightarrow{a_1} (s_1, q_1) \xrightarrow{a_2} \ldots \xrightarrow{a_n} (s_n, a_n)$ of $M \parallel \mathcal{A}$. Conversely, for every run $r = (s_0, q_0) \xrightarrow{a_1} (s_1, q_1) \xrightarrow{a_2} (s_2, q_2) \ldots \xrightarrow{a_n} (s_n, q_n)$ of $M \parallel \mathcal{A}$, the run $r' = s_0 \xrightarrow{a_1} s_1 \ldots \xrightarrow{a_n} s_n$ is a run of M, and we need to show that $r' \in \mathcal{R}(M, \sigma)$, i.e., r' is also compatible with σ. Since $q_0 \xrightarrow{a_1} q_1 \ldots \xrightarrow{a_n} q_n$ is a run of $\mathcal{A}$, we have $dom(a_1) \in \sigma(\epsilon)$ by $q_0 \xrightarrow{a_1} q_1$. For every $i = 2 \ldots n$, since $\mathcal{A}$ represents σ and $q_0 \xrightarrow{a_1 a_2 \ldots a_{i-1}} q_{i-1}$, we have $dom(a_i) = sched(q_{i-1}) \in \sigma(a_1 a_2 \ldots a_{i-1})$. This shows that r' is compatible with σ. □

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. That is, we

are interested in defining the security of scheduled machines (M, σ). We assume agents have a *synchronous* view of the machine, make an observation at each moment of time, and are able to distinguish one moment of time from the next, even if they did not perform an action. 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 security policy 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 convey prohibited information to L about H's activity. This will sometimes require us to focus on H-oblivious schedulers, in which schedules do not carry any information about which specific actions H performed when scheduled. Write $\Upsilon^{\mathtt{HO}}$ for the set of schedulers that are H-oblivious, and $\Upsilon^{\mathtt{O}}$ for the set of oblivious schedulers.

Since our notion of scheduler σ leaves open a choice of implementation $\mathcal{A}$, a concrete implementation of a scheduled machine (M, σ) will have the form of a SOLTS $M \parallel \mathcal{A}$ where $\mathcal{A}$ is some scheduler SOLTS representing σ. Thus, on first principles, we should define security as a predicate on these composite SOLTS. 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 of implementations $M \parallel \mathcal{A}$ of scheduled machines (M, σ) is *implementation independent* if for all schedulers σ and all scheduler SOLTS $\mathcal{A}_1, \mathcal{A}_2$ representing σ, the SOLTS $M \parallel \mathcal{A}_1$ satisfies X iff $M \parallel \mathcal{A}_2$ satisfies X. If σ is H-oblivious, then we require both $\mathcal{A}_1$ and $\mathcal{A}_2$ to be H-oblivious. We seek properties for which this is the case. In this case, we may view X as a set of scheduled machines (M, σ).

5. 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.

5.1. 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 *epistemic frame* 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 $view_u(r) = \beta$.

Epistemic frames can be generated both from SOLTS and from scheduled machines. In particular, given a machine M, we define the epistemic frame $\mathcal{I}(M) = (\mathcal{R}(M), \{view_u\}_{u \in D})$ where the functions $view_u$ are defined as in Section 3. For a scheduled machine (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 $\mathcal{R}(M, \sigma)$.

Proposition 5.1. *If a scheduler SOLTS $\mathcal{A}$ represents the scheduler σ and M is a machine, then $\mathcal{I}(M \parallel \mathcal{A}) = \mathcal{I}(M, \sigma)$.*

Proof: Trivial by Proposition 4.4(3). □

All of the trace based security definitions we give can be stated as properties X of an epistemic frame $\mathcal{I}$. By Proposition 5.1, if scheduler SOLTS $\mathcal{A}_1$ and $\mathcal{A}_2$ both represent a scheduler σ, then $\mathcal{I}(M \parallel \mathcal{A}_1) = \mathcal{I}(M \parallel \mathcal{A}_2)$. This leads immediately to the implementation-independence of the definitions. For readability, sometimes we just write the pair (M, σ) for $\mathcal{I}(M, \sigma)$ as the system generated by machine M scheduled under σ.

The scheduling of an agent's own actions is visible in its view, but this may leave the agent uncertain as to the scheduling of the other agents. Say that agent 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 $sch(r) = sch(r')$. In particular, every agent u is schedule-aware in (M, σ) with deterministic σ that is v-oblivious for all $v \neq u$.[6] We show the following result for L, and the result for H is just symmetric. Note that since we do not distinguish the actions of agent Sys with respect to $D = \{H, L, Sys\}$, a scheduler σ is always Sys-oblivious.

Lemma 5.2. *L is schedule-aware in (M, σ) if σ is both deterministic and H-oblivious.*

Proof: Let $r, r' \in \mathcal{R}(M, \sigma)$ such that $view_L(r) = view_L(r')$. Let $sch(r) = u_0 u_1 u_2 \ldots u_n$, we prove $sch(r') = u'_0 u'_1 \ldots u'_n = sch(r)$ by induction. Base case: $\{u'_0\} = \sigma(\epsilon) = \{u_0\}$ since σ is deterministic. Suppose $u_i = u'_i$ for all $i \in \{0 \ldots k\}$, i.e., $sch(r_k) = sch(r'_k)$, we show the case of $k+1$. Since σ is deterministic, we have $\{u_{k+1}\} = \sigma(Act(r_k))$ and $\{u'_{k+1}\} = \sigma(Act(r'_k))$. By $view_L(r) = view_L(r')$, we have $Act_L(r_k) = Act_L(r'_k)$, i.e., the L parts of $Act(r_k)$ and $Act(r'_k)$ are the same, therefore after masking actions from H we have $\mu_H(Act(r_k)) = \mu_H(Act(r'_k))$. This implies $u_{k+1} = u'_{k+1}$ by σ being H-oblivious. □

5.2. Nondeducibility On Inputs

For asynchronous systems, the notion of *nondeducibility on inputs* [Sutherland86] states that a system is secure if L cannot deduce from its view any information

[6] Obviously this can be a scheduler $\sigma_{\mathfrak{s}}$ derived from a single infinite schedule $\mathfrak{s}$.

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 compatible with L's information. (We write X^ω for the set of all infinite sequences of elements of the set X.)

Definition 5.3. *$(M, \sigma) \in \mathtt{tNDI}_1$ 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. If L is able to rule out a prefix of α then L will be able to rule out α as an infinite sequence of H actions, therefore $\mathtt{tNDI}_1$ can be regarded as a basic security requirement — every finite H behaviour is (potentially) possible from every L-view. 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 (actually) 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 5.4. *$(M, \sigma) \in \mathtt{tNDI}_2$ 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 5.5. *$(M, \sigma) \in \mathtt{tNDI}_3$ if for all $r \in \mathcal{R}(M, \sigma)$, and $\alpha \in A_H^*$ with $|\alpha| = |Act_H(r)|$, there exists a run $r' \in \mathcal{R}(M, \sigma)$ with $sch(r) = sch(r')$ and $view_L(r') = view_L(r)$ and $Act_H(r') = \alpha$.*

The following result gives some relationships between these definitions.

Proposition 5.6.

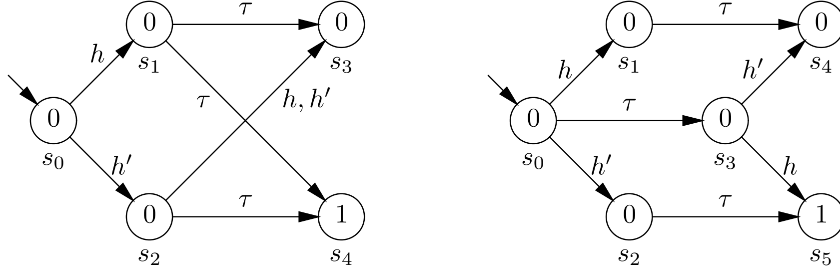

Figure 3: (a) (M, σ) in $\mathtt{tNDI}_1$ but not in $\mathtt{tNDI}_2$ (b) (M, σ) in $\mathtt{tNDI}_2$ but not in $\mathtt{tNDI}_3$

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, σ).

Proof: For Part (1) first we show $\mathtt{tNDI}_3 \subseteq \mathtt{tNDI}_2$. Suppose $(M, \sigma) \in \mathtt{tNDI}_3$. Let β be a possible L view and $\alpha \in A_H^*$ with $|\alpha| \in \mathtt{Pna}_H(M, \sigma, \beta)$. Let $|\alpha| \in \mathtt{Pna}_H(M, \sigma, \beta)$ be witnessed by the run r'. We need to show that there is a run $r \in \mathcal{R}(M, \sigma)$ such that $view_L(r) = \beta$ and $Act_H(r) = \alpha$. We obtain this directly from $(M, \sigma) \in \mathtt{tNDI}_3$, which implies that there exists a run r with $view_L(r) = view_L(r') = \beta$ and $Act_H(r) = \alpha$ and, moreover, $sch(r) = sch(r')$.

Next we show $\mathtt{tNDI}_2 \subseteq \mathtt{tNDI}_1$. Suppose $(M, \sigma) \notin \mathtt{tNDI}_1$, then there is a sequence $\alpha \in A_H^\omega$ and a possible L view β such that for all $r \in \mathcal{R}(M, \sigma)$, $view_L(r) = \beta$ implies $Act_H(r)$ is not a prefix of α. Let $r' \in \mathcal{R}(M, \sigma)$ be a run with $view_L(r') = \beta$ and $Act_H(r') = \alpha'$. Take α'' to be the prefix of α with $|\alpha''| = |\alpha'|$. Then we have $|\alpha''| \in \mathtt{Pna}_H(M, \sigma, \beta)$. However, there is no run r'' such that $view_L(r'') = \beta$ and $Act_H(r'') = \alpha''$, since α'' is a prefix of α. So $(M, \sigma) \notin \mathtt{tNDI}_2$.

To see that the containments $\mathtt{tNDI}_3 \subseteq \mathtt{tNDI}_2 \subseteq \mathtt{tNDI}_1$ are strict, consider the two examples in Figure 3. (Refer to the end of Section 3 for diagrammatic conventions relating to machines.)

1. To show $\mathtt{tNDI}_1 \not\subseteq \mathtt{tNDI}_2$, we give a scheduled machine (M, σ) that is in $\mathtt{tNDI}_1$ but not in $\mathtt{tNDI}_2$. Let σ be a scheduler that produces schedules of the form $H(H + Sys)(H + L)^\omega$. Let $A_H = \{h, h'\}$, $A_L = \{l\}$, $A_{Sys} = \{\tau\}$, and the states and transitions of M are depicted in Figure 3(a). We can observe that the L view $\beta = 0 \frown 0 \frown 0$ occurs on the run $r = s_0 \xrightarrow{h} s_1 \xrightarrow{\tau} s_3$, where we have $Act_H(r) = h$. From this we have $1 \in \mathtt{Pna}_H(M, \sigma, \beta)$. However, there is no run $r' \in \mathcal{R}(M, \sigma)$ with $view_L(r') = \beta$ and $Act_H(r') = h'$. So (M, σ) is not in $\mathtt{tNDI}_2$. However, (M, σ) is in $\mathtt{tNDI}_1$. To see this we check that each of the following two possible types of infinite H action sequence is compatible with all possible L views: 0, $0 \frown 0$, $0 \frown 0 \frown 0((\frown 0) + (l\ \ 0))^*$ and $0 \frown 0 \frown 1((\frown 1) + (l\ \ 1))^*$.
   - If the H input sequence is of the form $hh\ldots$ or $hh'\ldots$, schedules of the form $HSys(H + L)^\omega$ make it compatible with all the above L views.

- If the H input sequence is of the form $h'h\ldots$ or $h'h'\ldots$, schedules of the form $HSys(H+L)^\omega$ make it compatible with the L views 0, $0 \frown 0$, and $0 \frown 0 \frown 1((\frown 1)+(l\ \ 1))^*$, and schedules of the form $HH(H+L)^\omega$ make it compatible with the L views $0 \frown 0 \frown 0((\frown 0)+(l\ \ 0))^*$.

2. To show $\mathtt{tNDI}_2 \not\subseteq \mathtt{tNDI}_3$, we give a scheduled machine which is in $\mathtt{tNDI}_2$ but not in $\mathtt{tNDI}_3$. Let σ be a scheduler that produces schedules of the form $(HSys + SysH)(H + L)^\omega$. The machine M has $A_H = \{h, h'\}$, $A_L = \{l\}$, $A_{Sys} = \{\tau\}$, with its states and transitions shown in Figure 3(b). The L view $0 \frown 0 \frown 0$ is generated by the run $r = s_0 \xrightarrow{h} s_1 \xrightarrow{\tau} s_4$ with $Act_H(r) = h$ and $sch(r) = HSys$. However, if we consider the sequence h', the only run r' with $sch(r') = HSys$ and $Act_H(r') = h'$ is $r' = s_0 \xrightarrow{h'} s_2 \xrightarrow{\tau} s_5$. This run gives $view_L(r') = 0 \frown 0 \frown 1$. So (M, σ) is not in $\mathtt{tNDI}_3$. It is also easy to observe that (M, σ) is in $\mathtt{tNDI}_2$.

For Part (2) we show that if L is schedule-aware in (M, σ), then $(M, \sigma) \in \mathtt{tNDI}_1$ implies $(M, \sigma) \in \mathtt{tNDI}_3$. Suppose L is schedule-aware in (M, σ) in $\mathtt{tNDI}_1$. Let $r \in \mathcal{R}(M, \sigma)$ and let $\alpha \in A_H^*$ satisfy $|\alpha| = |Act_H(r)|$. Take α' to be any sequence in A_H^ω with prefix α. Since $(M, \sigma) \in \mathtt{tNDI}_1$ there exists a run r' such that $view_L(r') = view_L(r)$ and $Act_H(r')$ is a prefix of α'. Since L is schedule-aware in (M, σ), we have $sch(r) = sch(r')$, from which it follows that $|Act_H(r')| = |Act_H(r)|$, so in fact $Act_H(r') = \alpha$. This gives everything that we require for $(M, \sigma) \in \mathtt{tNDI}_3$. □

Which of the three definitions introduced in this section is appropriate may depend on the situation, and on application specific concerns such as what harm could be done with the information that the definition of security allows L to learn. In general, it is better to show that a system has a stronger security property than a weaker one, suggesting the use of $\mathtt{tNDI}_3$. However, for systems that do not satisfy this property, e.g., because the scheduler is itself intended to play a role in security of the system by obscuring the schedule, and in which information about the schedule is itself carefully protected, $\mathtt{tNDI}_2$ may be adequate. For systems that do not satisfy $\mathtt{tNDI}_2$, even $\mathtt{tNDI}_1$ might suffice, since it still provides assurance that H's inputs are not known to L.

5.3. Nondeducibility on Strategies

Nondeducibility on inputs 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 on inputs 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, vdMZ10]. 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. The following example resembles Wittbold and Johnson's.

Example 5.7. *Let* $A_H = \{h, h'\}$ *and* $A_L = \{\ell\}$. *Define a scheduled machine* (M, σ) *with* $M = \langle\{s_0, s_1, s_2, s_3, s_4\}, \{s_0\}, \rightarrow, O, obs\rangle$. *The transition relation of the machine* M *is depicted in Figure 4(a). The observation function is defined as*

- $obs_H(s_0) = obs_H(s_1) = obs_H(s_3) = obs_H(s_4) = 0$ *and* $obs_H(s_2) = 1$,
- $obs_L(s_0) = obs_L(s_1) = obs_L(s_2) = obs_L(s_3) = 0$ *and* $obs_L(s_4) = 1$.

Let σ *be the deterministic scheduler corresponding to the schedule* $LHLHLH\ldots$. *This scheduled machine is secure with respect to the notions* $\mathtt{tNDI}_1$, $\mathtt{tNDI}_2$ *and* $\mathtt{tNDI}_3$, *which are equivalent in this case because the scheduler is deterministic and* H*-oblivious (thus* L *is schedule-aware). The views of* L *can be described in the pattern* $0\ell 0 \frown (0(\ell 0 \frown 0)^* + 1(\ell 1 \frown 1)^*)$, *but it is sufficient to examine only the* L *views of length* 2. *Each of these is compatible with all choices of* H *action in the second step. For example,* $0\ell 0 \frown 0$ *is compatible with the* H *action* h, *if the run passes through* s_0, s_1 *and* s_3, *and it is compatible with the* H *action* h', *if the run passes through* s_0, s_2 *and* s_3.

However, such a scheduled machine should not be deemed as secure against an attacker at H *that is attempting to deliberately pass information to* L, *as* H *can act in such a way as to control* L *observations. Suppose that* H *wishes to transmit the bit* 0 *to* L. *It can do so by the following behaviour:*

- *if* H*'s view is* $0 \frown 0$, *it performs* h,
- *if* H*'s view is* $0 \frown 1$, *it performs* h'.

In this way, H *ensures that* L*'s third observation is* 0, *so that* H *has transmitted the bit* 0 *to* L. *By means of a similar pattern of behavior, if* H *wishes to transmit the value* 1, *it can ensure that* L*'s third observation is* 1. □

In response to the issues illustrated in this example, Wittbold and Johnson propose a definition of security that takes into account that L, rather considering all H behaviours possible, may know *a priori* that H is engaging in some specific pattern of behaviour. This rule is formally captured using the notion of strategy. In our framework, strategies can be formulated as follows.

Definition 5.8. *An* H strategy *in a scheduled machine* (M, σ) *is a function* $\pi : Views_H(M, \sigma) \rightarrow A_H$. *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.

Intuitively, an H strategy is a rule describing how the agent H chooses its next action as a function of its view, and a run is consistent with a strategy if at each stage in the construction of the run, the next H action executed is chosen according to this rule. There are some subtle differences between these

definitions and the definitions of strategies that have been used in the literature for the synchronous, simultaneous-action context [WJ90], or in the asynchronous system context [FG95, vdMZ10]. One is that in both these settings, it is always the case or, respectively, always possible, that the action selected by the strategy will be executed. On the other hand, in our setting, because of scheduling, it is possible that the action selected by the strategy on a given view will not be executed in *any* run giving that view. Moreover, in our framework, because of scheduler non-determinism and incompleteness of H observations, even when H has been scheduled, it need not always be the case that H knows that it has been scheduled. We therefore need to define π on any view where it is *possible* that H has been scheduled. Intuitively, given a view β, the action $\pi(\beta)$ is the action that H *would* perform if it were in fact scheduled after a run in which it makes view β.[7]

Using the notion of strategy, we may now formulate a definition of security in scheduled machines that is similar to Wittbold and Johnson's notion of nondeducibility on strategies in their simultaneous action setting.

Definition 5.9. $(M, \sigma) \in \texttt{tNDS}_1$ *if for every* $r \in \mathcal{R}(M, \sigma)$ *and* H *strategy* π, *there exists* $r' \in \mathcal{R}(M, \sigma, \pi)$ *such that* $view_L(r) = view_L(r')$.

Intuitively, this definition says that for all strategies π that H might choose to run, there is no change to the set of possible L views, which is always the same as the set of possible L views when H does not constrain its behaviour in any way. Thus, there is no way that a Trojan horse at H could pass information to L by constraining H behaviour to a particular strategy.

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

Definition 5.10. $(M, \sigma) \in \texttt{tNDS}_2$ *if for every* $r \in \mathcal{R}(M, \sigma)$ *and* H *strategy* π, *there exists* $r' \in \mathcal{R}(M, \sigma, \pi)$ *such that* $view_L(r) = view_L(r')$ *and* $sch(r) = sch(r')$.

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

Proposition 5.11.

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

Proof:

[7] One could refine our definitions by restricting the domain of π to views that could actually occur when executing π in a given scheduled machine, but this would have no effect on our results, so we prefer to work with the simpler over-defined functions.

- For Part (1), $\texttt{tNDS}_2 \subseteq \texttt{tNDS}_1$ is trivial by definition. To show that the containment is strict, Figure 4(b) is an example in $\texttt{tNDS}_1$ but not in $\texttt{tNDI}_2$, so it is also not in $\texttt{tNDS}_2$ by the result $\texttt{tNDS}_2 \subseteq \texttt{tNDI}_3 \subseteq \texttt{tNDI}_2$ (Proposition 5.11(3) and Proposition 5.6(1)).

- For Part (2), we show $(M,\sigma) \in \texttt{tNDS}_1$ implies $(M,\sigma) \in \texttt{tNDS}_2$ if L is schedule-aware. Suppose $(M,\sigma) \in \texttt{tNDS}_1$ let $r \in \mathcal{R}(M,\sigma)$ with $view_L(r) = \beta$. From $(M,\sigma) \in \texttt{tNDS}_1$, for any H strategy π, there is a run $r' \in \mathcal{R}(M,\sigma)$ compatible with π and $view_L(r') = \beta$, i.e., $r' \in \mathcal{R}(M,\sigma,\pi)$. Since L is schedule-aware, we also have $sch(r) = sch(r')$. Then $(M,\sigma) \in \texttt{tNDS}_2$ by definition.

- For Part (3), first we show $\texttt{tNDS}_2 \subseteq \texttt{tNDI}_3$. by proving the contrapositive. Suppose $M \notin \texttt{tNDI}_3(\sigma)$, then there exist $r \in \mathcal{R}(M,\sigma)$ and $\alpha = a_1 \dots a_n \in A_H^*$ with $|\alpha| = |Act_H(r)|$, such that for all r' with $sch(r') = sch(r)$ and $Act_H(r') = \alpha$ we have $view_L(r') \neq view_L(r)$. We are going to show that $M \notin \texttt{tNDS}_2(\sigma)$. In order to do this, take an arbitrary infinite sequence $\alpha' \in A_H^\omega$. We construct an H strategy π that enforces H's actions in every run to strictly follow the pattern of $\alpha \cdot \alpha'$, as follows. Define π to be the strategy such that, on an H view β with exactly k actions of H, $\pi(\beta)$ is the $(k+1)$-st element of $\alpha \cdot \alpha'$. (Since we do not assume H is schedule-aware, H may not always know if it has been scheduled, but once it is, it will see this action appear in its view at the next step, at which point it can switch to offering the next action.) Then for all $r'' \in \mathcal{R}(M,\sigma,\pi_\alpha)$, if $sch(r'') = sch(r)$, we will have $Act_H(r'') = \alpha$ (r and r'' are of the same length and scheduled in the same way). But from the above assumption, $view_L(r'') \neq view_L(r)$. So $M \notin \texttt{tNDS}_2(\sigma)$. The inclusion $\texttt{tNDS}_1 \subseteq \texttt{tNDI}_1$ can be proved in a similar way. □

The containment $\texttt{tNDS}_2(\sigma) \subseteq \texttt{tNDI}_3(\sigma)$ is strict even on deterministic schedulers. The machine in Example 5.7 (as shown in Figure 4(a)) is not in $\texttt{tNDS}_2(\sigma)$ because H can decide L's observation in the next state if it knows the current state is s_1 or s_2. However, one can easily verify that every possible L view is compatible with every possible H input sequence, so it is in $\texttt{tNDI}_3(\sigma)$. Note this is also an example to show $\texttt{tNDS}_1(\sigma) \subseteq \texttt{tNDI}_1(\sigma)$ is strict.

The statement $\texttt{tNDS}_1 \subseteq \texttt{tNDI}_2$ does not hold. Consider the machine M in Figure 4(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 $SysSys(H+L)^\omega$ produce all possible L views independent of any action by H. However, the scheduled machine (M,σ) is not in $\texttt{tNDI}_3$ 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 $\texttt{tNDI}_2$ because there does not exist a run r with $view_L(r) = 0 \frown 0 \frown 0$ and $Act_H(r) = h'$, although $|h'| \in \texttt{Pna}_H(M,\sigma,0 \frown 0 \frown 0)$.

Together with the witness of Figure 4(a) which is a machine in $\texttt{tNDI}_3(\sigma)$ but not in $\texttt{tNDS}_1(\sigma)$, we have the following result.

Proposition 5.12. $\texttt{tNDS}_1 \not\subseteq \texttt{tNDI}_2$ *and* $\texttt{tNDI}_3 \not\subseteq \texttt{tNDS}_1$.

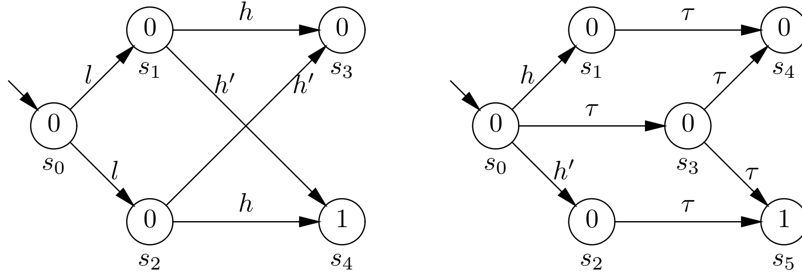

Figure 4: (a) (M, σ) in $\texttt{tNDI}_3$ but not in $\texttt{tNDS}_1$ (b) (M, σ) in $\texttt{tNDS}_1$ but not in $\texttt{tNDI}_2$

5.4. Revisiting the Motivating Example

To illustrate the above definitions, we formulate the motivating example from Section 2.

Define the signature (A, D, dom) by $D = \{H, L, Sys\}$, $A = \{\tau, m_H, m_L, \epsilon_H, \epsilon_L\}$, and $dom(\tau) = Sys$, $dom(x_u) = u$ for $x \in \{m, \epsilon\}$. Here m_u denotes that u is trying to push a message into the buffer, and ϵ_u denotes u is skipping. Based on the signature, we present the system as a machine $M = \langle S, \{s_0\}, \rightarrow, O, obs \rangle$, where

- $S = \mathit{Buffer} \times \mathit{Out}^{\{H,L\}}$, where $\mathit{Buffer} = \{\langle\rangle, \langle m_L \rangle, \langle m_H \rangle, \langle \tau \rangle\}$, and $\mathit{Out} = \{\mathit{ack}, \mathit{fail}, \mathit{retry}, \bot\}$,
- $s_0 = (\langle\rangle, f_\bot)$, where $f_\bot = \{H \mapsto \bot, L \mapsto \bot\}$,
- $O = \mathit{Buffer} \times \mathit{Out}$,
- $obs(L, (b, f)) = (\langle\rangle, f(L))$ and $obs(H, (b, f)) = (b, f(H))$, and
- $\rightarrow \subseteq S \times A \times S$ is the transition relation.

The first component b of the state (b, f) is a buffer state, where $\langle\rangle$ is the empty buffer; the second component f is used to express output values returned by agent operations. For agent u, the value $f(u)$ represents the message received by that agent: *ack* means that the agent's attempt to write to the buffer was successful, *fail* means that it failed because the buffer contains an agent message, *retry* means that the buffer contained Sys data but has now been cleared, and the agent should retry the write, and $\bot$ is a null message. Agent L observes only an output message; the state of the buffer is masked and always looks to it like $\langle\rangle$. Agent H can observe the buffer state as well as its output message. The transition relation contains the following transition types:

- Idling: $(b, f) \xrightarrow{\epsilon_u} (b, \{H \mapsto \bot, L \mapsto \bot\})$ for $u \in \{H, L\}$ and $b \in \mathit{Buffer}$, (agent u performs a skip: the state of the buffer is unchanged and no output is returned)

- Push-in: $(\langle\rangle, f) \xrightarrow{m_u} (\langle m_u\rangle, \{u \mapsto ack, \overline{u} \mapsto \bot\})$ for $u \in \{H, L\}$,
(the buffer is empty, and agent u successfully writes a message to the buffer)

- Buffer-full: $(\langle m_v\rangle, f) \xrightarrow{m_u} (\langle m_v\rangle, \{u \mapsto fail, \overline{u} \mapsto \bot\})$ for $u, v \in \{H, L\}$,
(the buffer contains an agent message, and agent u makes an unsuccessful attempt to write a message to the buffer)

- Request-ready: $(\langle\tau\rangle, f) \xrightarrow{m_u} (\langle\rangle, \{u \mapsto retry, \overline{u} \mapsto \bot\})$
(the buffer contains Sys data, and agent u makes an unsuccessful attempt to write a message to the buffer: the buffer is cleared as a result and the agent receives the "retry" message)

- Pop-out: $(\langle m_u\rangle, f) \xrightarrow{\tau} (\langle\rangle, \{H \mapsto \bot, L \mapsto \bot\})$ for $u \in \{H, L\}$,
(the buffer contains an agent message: Sys clears the buffer and transmits the data)

- System-possess: $(\langle\rangle, f) \xrightarrow{\tau} (\langle\tau\rangle, \{H \mapsto \bot, L \mapsto \bot\})$,
(the system occupies the buffer when neither H nor L is using it)

- System-release: $(\langle\tau\rangle, f) \xrightarrow{\tau} (\langle\rangle, \{H \mapsto \bot, L \mapsto \bot\})$,
(the buffer contains Sys data, and Sys chooses to empty it)

- System-hold: $(\langle\tau\rangle, f) \xrightarrow{\tau} (\langle\tau\rangle, \{H \mapsto \bot, L \mapsto \bot\})$,
(the buffer contains Sys data, and Sys chooses to keep this data in the buffer).

Here $\overline{u}$ denotes L when $u = H$, and denotes H when $u = L$. The following cases illustrate how this machine behaves with respect to the security properties we have proposed so far.

1. For the (round-robin) scheduler σ corresponding to the schedule $(HLSys)^\omega$, the scheduled machine (M, σ) is not in $\texttt{tNDI}_1$, hence not secure for any of the security properties we have discussed. To see this, suppose L's first action is to try to push in a message, then
   - if H's first action is ϵ_H, then we have a (deterministic) trace $s_0 \xrightarrow{\epsilon_H} (\langle\rangle, f_\bot) \xrightarrow{m_L} (\langle m_L\rangle, \{H \mapsto \bot, L \mapsto ack\})$
   - if H's first action is m_H, then we have a trace $s_0 \xrightarrow{m_H} (\langle m_H\rangle, f_\bot) \xrightarrow{m_L} (\langle m_H\rangle, \{H \mapsto \bot, L \mapsto fail\})$.

   Thus, the L view "$\bot \frown \bot m_L ack$" is incompatible with every infinite H action sequence starting with m_H; similarly the L view "$\bot \frown \bot m_L fail$" is incompatible with every infinite H action sequence starting with ϵ_H.
2. For the scheduler σ corresponding to the schedule $(HHSysSysSysLLSysSysSys)^\omega$ the scheduled machine (M, σ) is in $\texttt{tNDS}_2$ (hence also $\texttt{tNDS}_1$). To see this, note that the fact that there are three Sys steps between the two agents H and L means that it is an invariant that at each of the first moments of a block of two L or H steps, the buffer could have value either $\langle\rangle$ or $\langle\tau\rangle$, independent of the previous H actions or L view. This holds initially, and

(a) from buffer $\langle\rangle$, the transitions System-possess, System-hold, System-release result in buffer $\langle\rangle$ and the transitions System-possess, System-hold, System-hold result in buffer $\langle\tau\rangle$, and

(b) from buffer $\langle m_u\rangle$, the transitions Pop-out, System-possess, System-release result in buffer $\langle\rangle$ and the transitions Pop-out, System-hold, System-hold result in buffer $\langle\tau\rangle$.

In each case, the resulting fragment of the L view corresponding to these three steps is $\frown \bot \frown \bot \frown \bot$. The fragment of the L view for any two H steps is always $\frown \bot \frown \bot$, and the fragment of the L view for any two L steps depends only on the state of the bufffer, which is independent at that time of H actions, as just noted. Thus, whatever strategy H follows, there is no impact on the set of possible L views.

3. The scheduler σ corresponding to the schedule $(HHSysSysLLSysSys)^\omega$ makes the system insecure. Considering the case when H skips its first round and pushes-in a message in its second round, it enforces a run $s_0 \xrightarrow{\epsilon_H} (\langle\rangle, f_\bot) \xrightarrow{m_H} (\langle m_H\rangle, \{H \mapsto ack, L \mapsto \bot\}) \xrightarrow{\tau} (\langle\rangle, f_\bot) \xrightarrow{\tau} (\langle\tau\rangle, f_\bot)$. Thus, when L is next scheduled and tries to push-in a message, it will observe "*retry*" instead of "*ack*". That is, the infinite H action sequence $\epsilon_H m_H \ldots$ is incompatible with the L view "$\bot \frown \bot \frown \bot \frown \bot \frown \bot m_L ack$", and the scheduled machine is not in $\texttt{tNDI}_1$.

4. For an example with a non-deterministic scheduler (not discussed in Section 2), if σ generates the set of schedules $(LSys(H + Sys)Sys)^\omega$, then (M, σ) is in $\texttt{tNDS}_1$ but not in $\texttt{tNDI}_2$. To see that it is not in $\texttt{tNDI}_2$, note that the L view $\bot m_L ack \frown \bot \frown \bot \frown \bot m_L ack$ is compatible with H having performed the single action m_H, but not compatible with it having performed ϵ_H. However, this scheduled machine is in $\texttt{tNDS}_1$ since H can be skipped for arbitrarily long periods of time and Sys alone can produce all possible L views without H's participation.[8]

6. Bisimulation-based Definitions

Restrictiveness (RES) is a security property introduced by McCullough [McCullough88]. In state-observed systems, it can be characterised [vdMZ07] by the existence of an *unwinding relation* [GM84], which is a binary relation on the set of system states. In asynchronous systems, the unwinding relation is essentially a bisimulation relation treating L's inputs as external actions, with H actions not causing changes distinguishable by L. The conditions are as follows:

(OC) If $s \approx s'$ then $obs_L(s) = obs_L(s')$.

[8] We are not concerned in this paper with denial of service attacks, but it is interesting to note that where there is a single Sys step between L and H, agent L can launch a "denial-of-service" attack by keeping the buffer empty, so that after the transition System-possess, the buffer is occupied by a system message "τ" and an attempt by H to write to the buffer will fail. (We thank one of the reviewers for this observation.)

(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 states s, t and actions $a \in A_H$, if $s \xrightarrow{a} t$ then $s \approx t$.

Intuitively, an unwinding relation requires that H actions do not cause changes distinguishable by L. 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 mask *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 6.1. *Given a SOLTS* $M = \langle S, S_0, \rightarrow, O, obs\rangle$*,*

1. An *insensitive synchronous unwinding relation* is a relation $\approx \subseteq S \times S$ including (s_0, s_0) for all $s_0 \in S_0$, satisfying OC, SC and LR, where LR is defined as: *for all 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 $\approx \subseteq S \times S$ including (s_0, s_0) for all $s_0 \in S_0$, satisfying OC, SC, LR_H and LR_{Sys}, where $\mathrm{LR}_H(\mathrm{LR}_{Sys})$ is defined as: *for all 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* $t \approx t'$*.*

We will generally apply these notions to SOLTS of the form $M \parallel \mathcal{A}$ where M is a machine and $\mathcal{A}$ is a scheduler SOLTS. Say that a state s is reachable if there exists an initial state s_0 and a sequence of actions $\alpha \in A^*$ such that $s_0 \xrightarrow{\alpha} s$. The existence of an (in)sensitive unwinding on a SOLTS may depend on the behaviour of the SOLTS on unreachable states. On the other hand, it seems unreasonable that the security of the system should be affected by the behaviour of the system on unreachable states. Thus, we henceforth assume that every state is reachable. Note that given the set of states S of the machine M and Q of the scheduler SOLTS $\mathcal{A}$, it is possible that not all states in $S \times Q$ are reachable in the combined SOLTS $M \parallel \mathcal{A}$ even if all states in S and Q are reachable in M and $\mathcal{A}$, respectively. In this case we restrict the combined SOLTS $M \parallel \mathcal{A}$ to the reachable subset of $S \times Q$.

The following lemma states a connection between unwinding and scheduling.

Lemma 6.2. *Let* M *be a machine and* $\mathcal{A}$ *be a scheduler SOLTS, then*

1. *if there is a sensitive synchronous unwinding relation* $\approx$ *on* $M \parallel \mathcal{A}$ *such that* $(s, q) \approx (t, q')$*, then* $sched(q) = sched(q')$*.*
2. *if there is an insensitive synchronous unwinding relation* $\approx$ *on* $M \parallel \mathcal{A}$ *such that* $(s, q) \approx (t, q')$*, then either* $sched(q) = sched(q') = L$*, or* $sched(q) \neq L$ *and* $sched(q') \neq L$*.*

Proof: Trivial by definition. $\square$

As is usual for bisimulation-like relations, we can obtain a largest (in)sensitive synchronous unwinding relations on a combined SOLTS $M \parallel \mathcal{A}$, by taking the union of all (in)sensitive synchronous unwinding relations, if they do exist.

Lemma 6.3. *Let $\approx$ be the largest (in)sensitive synchronous unwinding relation on $M \parallel \mathcal{A}$, then $\approx$ is an equivalence relation.*

Proof: (sketch) For symmetricity, if $\sim$ is an (in)sensitive synchronous unwinding relation, then one can show that $\sim^{-1}$ is also an (in)sensitive synchronous unwinding relation, thus both are included in the largest relation $\approx$. For transitivity, if $\sim_1$ and $\sim_2$ are (in)sensitive synchronous unwinding relations, then so is $\sim_1 \cdot \sim_2$, therefore $(\sim_1 \cdot \sim_2) \subseteq \approx$. For reflexivity, we show that $(s, q) \approx (s, q)$, for all reachable states (s, q). By reachability, there exists a run from an initial state (s_0, q_0) to (s, q). Then by $(s_0, q_0) \approx (s_0, q_0)$, and by induction on the length of the run, one can find a state (s', q') reachable by the same sequence of actions with $(s, q) \approx (s', q')$. Then $(s, q) \approx (s, q)$ follows the fact that $\approx$ is symmetric and transitive. $\square$

Another property is that the distinction between sensitive and insensitive unwinding no longer exists for scheduler SOLTS representing deterministic H-oblivious schedulers.

Lemma 6.4. *Let $\mathcal{A}$ be a scheduler SOLTS that represents a deterministic H-oblivious scheduler σ and a machine M, then there exists an insensitive synchronous unwinding relation on $M \parallel \mathcal{A}$ iff there exists a sensitive synchronous unwinding relation on $M \parallel \mathcal{A}$.*

Proof: For the "only if" part, suppose there is an insensitive unwinding relation $\sim$ on $M \parallel \mathcal{A}$. Define a relation $\approx$, by $(s_1, q_1) \approx (s_2, q_2)$ if $(s_1, q_1) \sim (s_2, q_2)$ and $sched(q_1) = sched(q_2)$ and there exist sequences of actions $\alpha, \alpha' \in A^*$ satisfying $\mu_H(\alpha) = \mu_H(\alpha')$ and an initial state (s_0, q_0), such that $(s_0, q_0) \xrightarrow{\alpha} (s_1, q_1)$ and $(s_0, q_0) \xrightarrow{\alpha'} (s_2, q_2)$.

We show that $\approx$ is a sensitive unwinding relation. The property OC for $\approx$ is immediate from the fact that $\sim$ satisfies OC. Suppose we have states $(s_1, q_1) \approx (s_2, q_2)$, witnessed by $(s_0, q_0) \xrightarrow{\alpha} (s_1, q_1)$ and $(s_0, q_0) \xrightarrow{\alpha'} (s_2, q_2)$ with $\mu_H(\alpha) = \mu_H(\alpha')$ and $sched(q_1) = sched(q_2)$.

- For one case of LR, suppose $sched(q_1) = sched(q_2) = H$, and let $a_1, a_2 \in A_H$ and $(s_1, q_1) \xrightarrow{a_1} (s_3, q_3)$. Since $(s_1, q_1) \sim (s_2, q_2)$, there exists $(s_2, q_2) \xrightarrow{a_2} (s_4, q_4)$ such that $(s_3, q_3) \sim (s_4, q_4)$. We also have $\mu_H(\alpha \cdot a_1) = \mu_H(\alpha' \cdot a_2)$ by $\mu_H(\alpha) = \mu_H(\alpha')$ and $\mu_H(a_1) = \perp_H = \mu_H(a_2)$. Thus, by H-obliviousness, $\sigma(\alpha \cdot a_1) = \sigma(\alpha' \cdot a_2)$, and by determinism it follows that $sched(q_3) = sched(q_4)$. Thus, we have all that we need for $(s_3, q_3) \approx (s_4, q_4)$.
- The other case of $sched(q_1) = sched(q_2) = Sys$ for LR, and the case of $sched(q_1) = sched(q_2) = L$ for SC, can be proved following the same pattern of reasoning as the above case.

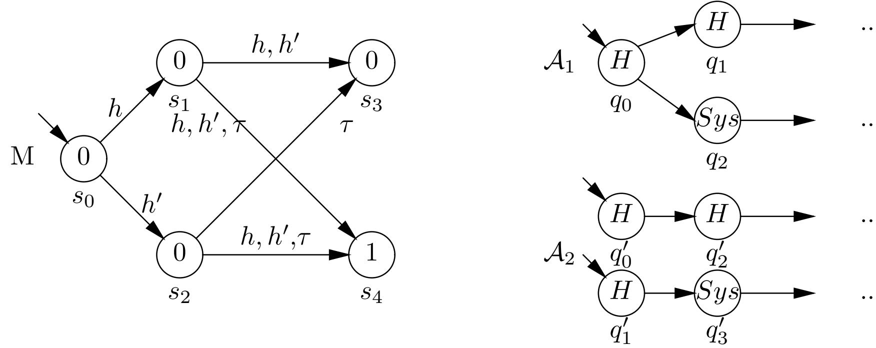

Figure 5: Insensitive unwinding is implementation dependent

- For all initial states (s_0, q_0), we have $(s_0, q_0) \sim (s_0, q_0)$, then $(s_0, q_0) \approx (s_0, q_0)$ by taking the two sequences both as ϵ.

The "if" part is by the fact that every sensitive unwinding is also an insensitive unwinding relation. □

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 machine (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, like bisimilarity, is sensitive to the branching structure of $\mathcal{A}$, different scheduler SOLTS representing the scheduler σ may have a different branching structure.

Figure 5 is an example which shows that there exists a machine M and a scheduler σ such that for different implementations $\mathcal{A}$ of σ, the composed SOLTS $M \parallel \mathcal{A}$ may have different results with respect to the existence of an *insensitive* synchronous unwinding relation. The two scheduler SOLTS $\mathcal{A}_1$ and $\mathcal{A}_2$ 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 the scheduled agent. The states of the machine M 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 $M \parallel \mathcal{A}_2$. The reason is that $\mathcal{A}_1$ resolves the nondeterministic choice between H and Sys at the second level into states q_1 and q_2. However, in the implementation $\mathcal{A}_2$, H and Sys are split apart at the beginning into states q'_0 and q'_1, which leaves the second level fewer choices to 'respond' to transitions in the machine M when it is combined to M to build up an unwinding relation.

An example showing that *sensitive* synchronous unwinding is also implementation dependent is depicted in Figure 6. Here the two scheduler SOLTS $\mathcal{A}_1$, $\mathcal{A}_2$ give the schedules $HLL(HH+LL)(LH)^\omega$, and one may observe there is a sensi-

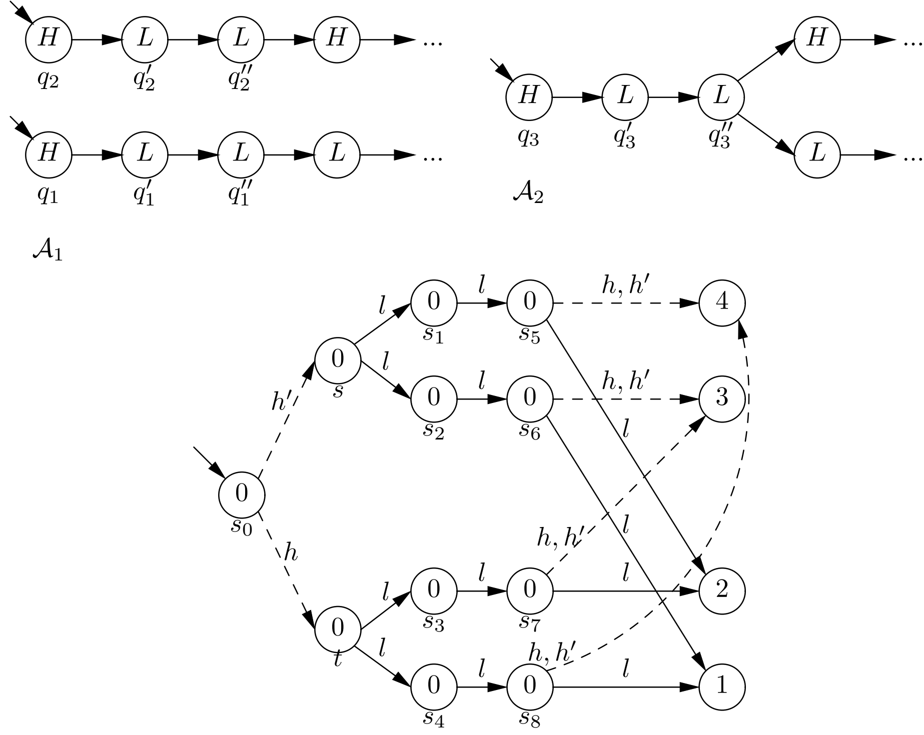

Figure 6: Sensitive unwinding is implementation dependent

tive synchronous unwinding relation on $M \parallel \mathcal{A}_1$: for the upper schedule, we have $(s_0, q_2) \approx (s_0, q_2)$, $(s, q_2') \approx (t, q_2')$, $(s_1, q_2'') \approx (s_4, q_2'')$ and $(s_2, q_2'') \approx (s_3, q_2'')$; for the lower schedule, we have $(s_0, q_1) \approx (s_0, q_1)$, $(s, q_1') \approx (t, q_1')$, $(s_1, q_1'') \approx (s_3, q_1'')$ and $(s_2, q_1'') \approx (s_4, q_1'')$. However, there is no sensitive synchronous unwinding relation on $M \parallel \mathcal{A}_2$, where $(s, q_3') \not\approx (t, q_3')$. To see this, note that the trees of height 2 originating in the states (s_1, q_3''), (s_2, q_3''), (s_3, q_3''), (s_4, q_3'') are all non-bisimilar, since they have sets of leaves labelled $\{2, 4\}$, $\{1, 3\}$, $\{2, 3\}$ and $\{1, 4\}$, respectively. Intuitively, the reasoning that explains why $M \parallel \mathcal{A}_2$ has no sensitive unwinding relation is different from the example for the insensitive case we have presented above: $\mathcal{A}_2$ is more flexible than $\mathcal{A}_1$, but here it is the extra choices that impede the construction of an unwinding relation in $M \parallel \mathcal{A}_2$.

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. We define both variants:

Definition 6.5. 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 scheduler SOLTS $\mathcal{A}$ 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 scheduler SOLTS* $\mathcal{A}$ *representing* σ.

Whether a universal or existential quantification 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 relation 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.

The properties $\mathtt{tRES}_i^Q$ (for $Q = \forall, \exists$) are the strongest of the security properties we have discussed so far, with the following relations holding.

Proposition 6.6.

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* $Q = \forall, \exists$.
3. $(M,\sigma) \in \mathtt{tRES}_2^Q$ *iff* $(M,\sigma) \in \mathtt{tRES}_1^Q$ *for deterministic* σ *and* $Q = \forall, \exists$.

Proof:

- For part 1, $\mathtt{tRES}_i^{\forall} \subseteq \mathtt{tRES}_i^{\exists}$ is trivial by definition.

- To show $\mathtt{tRES}_2^{\exists} \subseteq \mathtt{tNDS}_2$, we proceed as follows. Suppose the machine M is of the form $\langle S, S_0, \rightarrow, obs\rangle$ and there is an H-oblivious scheduler SOLTS $\mathcal{A} = \langle Q, Q_0, \rightarrow, obs\rangle$ representing σ, such that there is a sensitive unwinding relation $\approx$ on $M \parallel \mathcal{A}$. We show $M \in \mathtt{tNDS}_2(\sigma)$ by showing for all $r \in \mathcal{R}(M \parallel \mathcal{A})$ and H-strategy π, there exists $r' \in \mathcal{R}(M \parallel \mathcal{A})$ compatible with π, such that $view_L(r) = view_L(r')$ and $sch(r) = sch(r')$. Let $r = (s_0,q_0) \xrightarrow{a_1} (s_1,q_1) \xrightarrow{a_2} \ldots \xrightarrow{a_n} (s_n,q_n)$. We prove by induction that there is a run r' of $M \parallel \mathcal{A}$ of the form of $(s_0,q_0) \xrightarrow{a'_1} (s'_1,q'_1) \ldots \xrightarrow{a'_n} (s'_n,q'_n)$ which is compatible with π, and moreover, satisfies $(s_i,q_i) \approx (s'_i,q'_i)$, and $view_L(r_i) = view_L(r'_i)$ for all $i \in [0 \ldots n]$. Then $sch(r_i) = sch(r'_i)$, by Lemma 6.2(1), which gives $sched(q_i) = sched(q'_i)$ for all i. That is, every prefix of r' is compatible with π while preserving the schedule, so that the result immediately follows.

  The base case is by Definition 6.5, which gives that the empty run $r_0 = (s_0,q_0)$ trivially satisfies the conditions that $(s_0,q_0) \approx (s_0,q_0)$ and $view_L(r_0) = view_L(r_0)$. Suppose $(s_i,q_i) \approx (s'_i,q'_i)$, then from Lemma 6.2 we have $sched(q_i) = sched(q'_i)$. We show the requirements for the next transition.

  - If $sched(q_i) = L$, by $(s_i,q_i) \approx (s'_i,q'_i)$ and SC, there is a transition $(s'_i,q'_i) \xrightarrow{a_{i+1}} (s'_{i+1},q'_{i+1})$ such that $(s_{i+1},q_{i+1}) \approx (s'_{i+1},q'_{i+1})$.

Then, we have $obs_L((s_{i+1}, q_{i+1})) = obs_L((s'_{i+1}, q'_{i+1}))$, so together with the fact $view_L(r_i) = view_L(r'_i)$ from the induction hypothesis, we have $view_L(r_{i+1}) = view_L(r_i \cdot a_{i+1} \cdot (s_{i+1}, q_{i+1})) = view_L(r'_i \cdot a_{i+1} \cdot (s'_{i+1}, q'_{i+1}))$.

- If $sched(q_i) = H$, by $(s_i, q_i) \approx (s'_i, q'_i)$ and LR_H, there is a transition $(s'_i, q'_i) \xrightarrow{a'} (s'_{i+1}, q'_{i+1})$ and $(s_{i+1}, q_{i+1}) \approx (s'_{i+1}, q'_{i+1})$, where $a' = \pi(r'_i) \in A_H$. Then from the induction hypothesis, $view_L(r_{i+1}) = view_L(r_i) \frown obs_L((s_{i+1}, q_{i+1})) = view_L(r'_i \cdot a' \cdot (s'_{i+1}, q'_{i+1}))$.
- The case $sched(q_i) = Sys$ can be treated similarly, using LR_{Sys}.

- The proof for $\mathtt{tRES}_1^{\exists} \subseteq \mathtt{tNDS}_1$ is similar to that above, except that the two cases LR_H and LR_{Sys} are replaced by a single case LR for actions in both A_H and A_{Sys}.

- For $\mathtt{tRES}_2^Q \subseteq \mathtt{tRES}_1^Q$ for $Q \in \{\forall, \exists\}$, it suffices to show that for all machines M and scheduler SOLTS $\mathcal{A}$, every sensitive synchronous unwinding relation is also an insensitive synchronous unwinding relation. Let $\approx$ be a sensitive unwinding relation, then $\approx$ satisfies OC and SC, and LR_H and LR_{Sys}. We need to show $\approx$ satisfies LR as well. From Lemma 6.2(1), for any $(s_1, q_1) \approx (s_2, q_2)$, we have $sched(q_1) = sched(q_2)$. If q_1, q_2 both schedule L, then LR is vacuously satisfied; if they both schedule H or both schedule Sys, let $s_1 \xrightarrow{a}$ and $s_2 \xrightarrow{b}$. Then we have $dom(a) = dom(b)$. Now LR is satisfied by LR_H if $dom(a) = dom(b) = H$, and by LR_{Sys} if $dom(a) = dom(b) = Sys$.

- For part 3, let $(M, \sigma) \in \mathtt{tRES}_1^Q$, we show $(M, \sigma) \in \mathtt{tRES}_2^Q$. If $Q = \exists$ (i.e., $(M, \sigma) \in \mathtt{tRES}_1^{\exists}$), and suppose $\mathcal{A}$ represents σ and there exists an insensitive synchronous unwinding relation on $M \parallel \mathcal{A}$, then by Lemma 6.4 there exists a sensitive synchronous unwinding relation on $M \parallel \mathcal{A}$. Therefore $(M, \sigma) \in \mathtt{tRES}_2^{\exists}$. If $Q = \forall$ (i.e., $(M, \sigma) \in \mathtt{tRES}_1^{\forall}$), we apply the above reasoning on all H-oblivious implementations of σ, to get $(M, \sigma) \in \mathtt{tRES}_2^{\forall}$. That $(M, \sigma) \in \mathtt{tRES}_2^Q$ implies $(M, \sigma) \in \mathtt{tRES}_1^Q$ is by $\mathtt{tRES}_2^Q \subseteq \mathtt{tRES}_1^Q$ for $Q \in \{\forall, \exists\}$. □

The containments of Proposition 6.6 are all proper, which we justify as follows.

Proposition 6.7. $\mathtt{tRES}_i^{\exists} \not\subseteq \mathtt{tRES}_i^{\forall}$

This follows from the failure of implementation independence shown above.

Proposition 6.8. $\mathtt{tNDS}_i \not\subseteq \mathtt{tRES}_i^{\exists}$

Consider the machine M in Figure 7 where $A_H = \{h, h'\}$, $A_L = \{l\}$, and let the deterministic scheduler σ generate the schedule $(HL)^\omega$. We show first that $(M, \sigma) \in \mathtt{tNDS}_2$ but $(M, \sigma) \notin \mathtt{tRES}_2^{\exists}$. For the latter, we need to show that

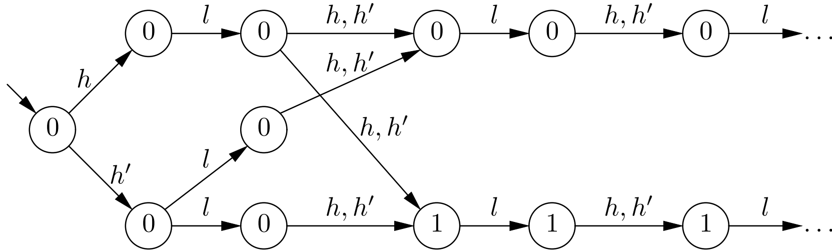

Figure 7: (M, σ) in $\texttt{tNDS}_1$ and $\texttt{tNDS}_2$, but not in $\texttt{tRES}_1^{\exists}$ or $\texttt{tRES}_2^{\exists}$

there does not exist a sensitive synchronous unwinding on $M \parallel \mathcal{A}$ for any H-oblivious scheduler SOLTS $\mathcal{A}$ representing σ. We argue that there does not exist a sensitive synchronous unwinding on the *particular* SOLTS $M \parallel \mathcal{A}_H^\sigma$. It turns out that this is sufficient: since σ is deterministic, it follows using Corollary 4.6 and Corollary 4.10 from Part II of the paper that there is no sensitive unwinding on $M \parallel \mathcal{A}$, for *every* H-oblivious $\mathcal{A}$ representing σ.

Since the structure of $\mathcal{A}_H^\sigma$ is linear, the initial part of $M \parallel \mathcal{A}_H^\sigma$ is isomorphic to that depicted in Figure 7, where (instead of the usual convention in which we omit self-loops) we interpret the lack of a transition labelled by an action a from a state as meaning there is no transition labelled by that action a. We show that there are no sensitive unwinding relations on $M \parallel \mathcal{A}_H^\sigma$ by contradiction. Suppose that "$\approx$" were a sensitive synchronous unwinding. Since the initial state (at level 0) must be $\approx$-related to itself, by LR_H the two states at level 1 must be $\approx$-related, and they both schedule L. By SC, the top state at level 2 must be $\approx$-related to one of the other two at level 2. We let it be $\approx$-related to the middle state at level 2 (the case of the bottom state is symmetric). Both states schedule H. Then by LR_H, the two states at level 3 must be $\approx$-related, because if the top state of level 2 makes a transition to the bottom state of level 3, the only state reachable by the middle state of level 2 is the top state of level 3. But, by OC, two states at level 3 cannot be $\approx$-related, since they have different L observations, which gives a contradiction.

For $(M, \sigma) \in \texttt{tNDS}_2$, we may observe there is no H strategy to pass information to L regardless of H and L's observational power. However H chooses its first two actions, L is always able to obtain both its possible views $0 \frown 0\, l\, 0 \frown 0$ and $0 \frown 0\, l\, 0 \frown 1$. This completes the proof that the containment $\texttt{tRES}_2^{\exists} \subseteq \texttt{tNDS}_2$ is strict.

Since σ is deterministic and H-oblivious, (M, σ) is L schedule-aware by Lemma 5.2. Then $(M, \sigma) \in \texttt{tNDS}_1$ iff $(M, \sigma) \in \texttt{tNDS}_2$ by Proposition 5.11(2), and $(M, \sigma) \in \texttt{tRES}_1^{\exists}$ iff $(M, \sigma) \in \texttt{tRES}_2^{\exists}$ by Proposition 6.6(3). Therefore, this is also an example showing that $\texttt{tRES}_1^{\exists} \subseteq \texttt{tNDS}_1$ is strict, i.e., we have $(M, \sigma) \in \texttt{tNDS}_1$, but there exist no insensitive unwinding relations on $M \parallel \mathcal{A}$ for *every* H-oblivious $\mathcal{A}$ representing σ.

Proposition 6.9. $\texttt{tRES}_1^{\exists} \not\subseteq \texttt{tRES}_2^{\exists}$

Consider the scheduled machine (M, σ) in Figure 5, where σ produces schedules

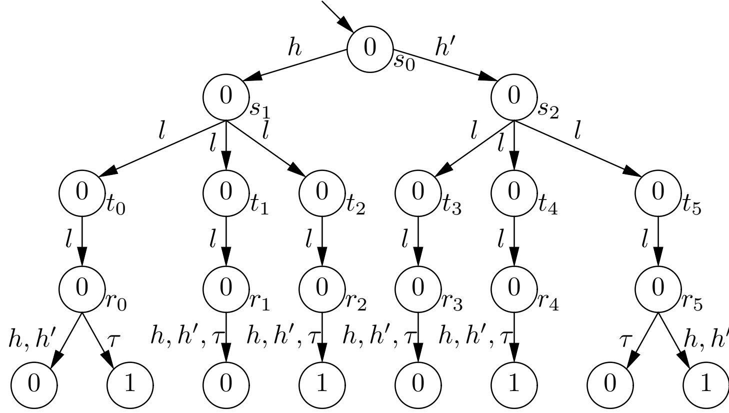

Figure 8: A machine showing $\texttt{tRES}_2^\forall$ is strictly stronger than $\texttt{tRES}_1^\forall$

in the form of $H(H + Sys)(H + L)^\omega$. We showed above that $(M, \sigma) \in \texttt{tRES}_1^\exists$. We show there exist no sensitive unwinding on $M \parallel \mathcal{A}$ for all scheduler SOLTS $\mathcal{A}$ representing σ. Let $\mathcal{A}$ represent σ. Suppose there were a sensitive unwinding relation $\sim$ on $M \parallel \mathcal{A}$. For (s_0, q_0) an initial state of $M \parallel \mathcal{A}$, we would have $(s_0, q_0) \sim (s_0, q_0)$ and $sched(q_0) = H$. Since $h, h' \in A_H$, and $(s_0, q_0) \xrightarrow{h} (s_1, q_1)$, there would exist $(s_0, q_0) \xrightarrow{h'} (s_2, q_2)$ and $(s_1, q_1) \sim (s_2, q_2)$. Since $\sim$ is sensitive, we have $sched(q_1) = sched(q_2)$. We have the following two cases to show that this is impossible.

- Suppose $sched(q_1) = sched(q_2) = H$, and consider the pair $h, h' \in A_H$ and a transition $(s_1, q_1) \xrightarrow{h} (s_3, q_1')$. By LR_H, there exists $(s_2, q_2) \xrightarrow{h'} (s_4, q_2')$, with $(s_3, q_1') \sim (s_4, q_2')$. But this contradicts OC since $obs_L(s_3) \neq obs_L(s_4)$.
- Suppose $sched(q_1) = sched(q_2) = Sys$ and $(s_2, q_2) \xrightarrow{\tau} (s_3, q_2')$, similar to the above, for all $(s_1, q_1) \xrightarrow{h} (s_4, q_1')$, it cannot be $(s_4, q_1') \sim (s_3, q_2')$.

As $\mathcal{A}$ is arbitrarily chosen, this shows $(M, \sigma) \notin \texttt{tRES}_2^\exists$.

Proposition 6.10. $\texttt{tRES}_1^\forall \not\subseteq \texttt{tRES}_2^\forall$

We take the machine M as depicted in Figure 8, and combine it with a scheduler σ that produces schedules in the form $HLL(H + Sys)L^\omega$. We show that $(M, \sigma) \in \texttt{tRES}_1^\forall$. A number of scheduler SOLTS that implement σ are depicted in Figure 9, which enlist all possible ways (modulo bisimilarity) on resolving the nondeterministic part (i.e., the choice on $H + Sys$) in σ. One may find that in M, t_1 and t_3 are isomorphic, t_2 and t_4 are isomorphic. The subtrees rooted at t_0 and t_5 need to be treated specially.

- In $M \parallel \mathcal{A}_1$, we have an insensitive unwinding relation $\sim$ satisfying $(s_0, q_0) \sim (s_0, q_0)$ and $(s_0, q_1) \sim (s_0, q_1)$. For the left part of $\mathcal{A}_1$, starting from

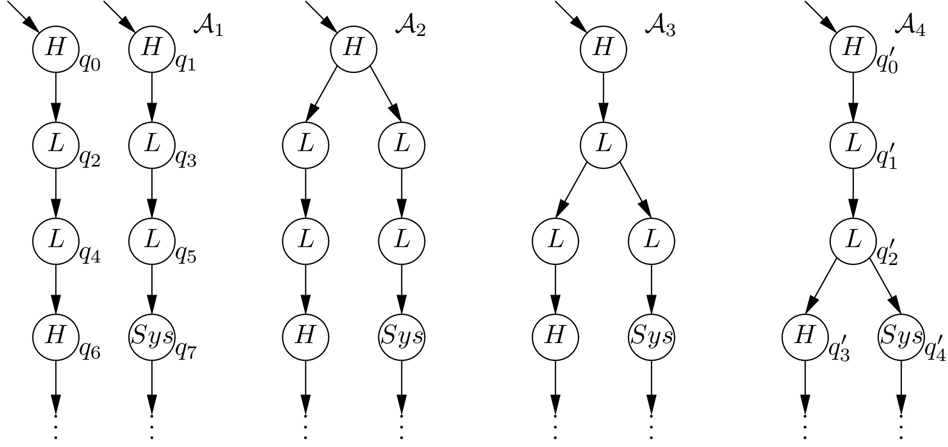

Figure 9: Implementations of a scheduler that produces schedules $HLL(H^\omega + Sys^\omega)$

(s_0, q_0) H takes transitions from h and h', we have $(s_1, q_2) \sim (s_2, q_2)$. If $(s_1, q_2) \xrightarrow{l} (t_0, q_4)$, we have $(s_2, q_2) \xrightarrow{l} (t_3, q_4)$, and $(t_0, q_4) \sim (t_3, q_4)$, which further derives $(r_0, q_6) \sim (r_3, q_6)$ with both states give L observation 0 in the next step, as the schedule $HLLH\ldots$ is known from the beginning. For the right part of $\mathcal{A}_1$, starting from (s_0, q_1) and H takes transitions from h and h', we have $(s_1, q_3) \sim (s_2, q_3)$. Then if $(s_1, q_3) \xrightarrow{l} (t_0, q_5)$, we have $(s_2, q_3) \xrightarrow{l} (t_4, q_5)$ and $(t_0, q_5) \sim (t_4, q_5)$, which derives $(r_0, q_7) \sim (r_4, q_7)$ with both states give L observation 1 in the next step as the schedule $HLLSys$ is known from the beginning. The other part of the relation $\sim$ can be derived similarly.

- In $M \parallel \mathcal{A}_4$, we have an insensitive unwinding relation $\sim$ satisfying $(s_0, q'_0) \sim (s_0, q'_0)$, $(s_1, q'_1) \sim (s_2, q'_1)$. The essential part of the relation is $(t_0, q'_2) \sim (t_5, q'_2)$, since if $(t_0, q'_2) \xrightarrow{l} (r_0, q'_3)$, we have $(t_5, q'_2) \xrightarrow{l} (r_5, q'_4)$; and if $(t_0, q'_2) \xrightarrow{l} (r_0, q'_4)$, we have $(t_5, q'_2) \xrightarrow{l} (r_5, q'_3)$. Note in this case we need to have a state scheduled to H to be related to a state scheduled to Sys, as the nondeterminism on $H + Sys$ is settled only in the last step.

Similarly one can also show that there exist insensitive unwinding relations on $M \parallel \mathcal{A}_2$ and $M \parallel \mathcal{A}_3$, which we leave as an exercise to the interested reader. In this way we have shown $(M, \sigma) \in \mathtt{tRES}_1^\forall$. There are no sensitive unwinding relations on $M \parallel \mathcal{A}_4$. Suppose there were a sensitive unwinding relation $\approx$ satisfying $(s_0, q'_0) \approx (s_0, q'_0)$, then by taking H actions h, h', we would have $(s_1, q'_1) \approx (s_2, q'_1)$. If $(s_1, q'_1) \xrightarrow{l} (t_0, q'_2)$, there are no states reachable by l from (s_2, q'_1) that can be $\approx$-related to (t_0, q'_2).

We further show that the existence of insensitive unwinding relations in a machine with some implementation of a scheduler, i.e., $(M, \sigma) \in \mathtt{tRES}_1^\exists$, is still

insufficient for (M, σ) to be secure for all trace-based properties except for $\texttt{tNDI}_1$. This is stated as follows.

Proposition 6.11. $\texttt{tRES}_1^{\exists} \not\subseteq \texttt{tNDI}_2$.

We take the machine M and sheduler σ as depicted in Figure 5, where two implementations of σ are given as scheduler SOLTS $\mathcal{A}_1$ and $\mathcal{A}_2$. First, we know (M, σ) is in $\texttt{tRES}_1^{\exists}$, as there exists an insensitive unwinding relation on $M \parallel \mathcal{A}_1$ (though no insensitive unwinding relation exists on $M \parallel \mathcal{A}_2$). We show that (M, σ) is not in $\texttt{tNDI}_2$. Consider the L view $\beta = 0 \frown 0 \frown 0$, the set of possible numbers of H actions compatible with β is $\texttt{Pna}_H(M, \sigma, \beta) = \{1, 2\}$. We choose the action sequence α that contains a single action h. For every run r that satisfies $\mathit{view}_L(r) = \beta$,

- if the machine is scheduled by $HH\ldots$, then $|\mathit{Act}(r)| = 2$, then $\mathit{Act}(r)$ cannot be just α which has length 1;
- if the machine is scheduled by $HSys\ldots$, then $\mathit{Act}(r)$ can only be of a single action h' which is not α.

Note that we also have $\texttt{tRES}_1^{\exists} \not\subseteq \texttt{tNDI}_3$ and $\texttt{tRES}_1^{\exists} \not\subseteq \texttt{tNDS}_2$, by the existing results $\texttt{tNDS}_2 \subseteq \texttt{tNDI}_3 \subseteq \texttt{tNDI}_2$ (Proposition 5.6(1) and Proposition 5.11(3)).

In the literature, unwinding relations have been proposed as a proof technique for information flow security properties [GM84, Rushby92, Mantel00, BFPR03, Oheim04]. Existence of an unwinding relation has been shown to be equivalent to noninterference in deterministic systems [Rushby92] (for what are called transitive policies in that paper;- these include the policy $L \leq H$ we consider in this paper). However, in general, the existence of an unwinding is a strictly stronger notion than trace-based information flow properties, as shown in the above example. Interestingly, this phenomenon can also be seen in our motivating example.

Non-existence of unwinding proof for the motivating example. Taking the motivating example as shown in Figure 1 and defined in Section 5.4, given a scheduler σ producing the infinite schedule $(HHSysSysSysLLSysSysSys)^{\omega}$, we show that the scheduled machine (M, σ) is not secure with respect to $\texttt{tRES}_2^{\exists}$ (or $\texttt{tRES}_1^{\exists}$), although it satisfies $\texttt{tNDS}_2$ (as was argued in Section 5.4). As σ is deterministic, by Corollary 4.10 from Part II of the paper, it suffices to show that there exist no sensitive unwinding relations on the machine controlled by $\mathcal{A}$ for some H-oblivious scheduler SOLTS $\mathcal{A}$ that represents σ. (Also note $(M, \sigma) \in \texttt{tRES}_2^{\exists}$ iff $(M, \sigma) \in \texttt{tRES}_1^{\exists}$ for deterministic σ, by Proposition 6.6(3).) We consider the specific case of the following ten-state scheduler SOLTS that represents σ. Define $\mathcal{A} = \langle \{q_0, q_1, \ldots q_9\}, \{q_0\}, \rightarrow, \{\bot\}, \mathit{obs}\rangle$ where

- $\mathit{sched}(q_0) = \mathit{sched}(q_1) = H$, $\mathit{sched}(q_5) = \mathit{sched}(q_6) = L$, and $\mathit{sched}(q) = Sys$ for all $q \in \{q_2, q_3, q_4, q_7, q_8, q_9\}$,
- $q_i \xrightarrow{a} q_{(i+1)\ mod\ 10}$ for all $a \in A_{\mathit{sched}(q_i)}$.

Suppose that there exists a sensitive unwinding relation $\sim$ on $M \parallel \mathcal{A}$. We produce a contradiction. The first two rounds of H leave 4 possible states of the combined SOLTS $M \parallel \mathcal{A}$: $((\langle m_H \rangle, f_\perp), q_2)$ (from m_H, ϵ_H) , $((\langle m_H \rangle, \{H \mapsto ack, L \mapsto \perp\}), q_2)$ (from ϵ_H, m_H), $((\langle m_H \rangle, \{H \mapsto fail, L \mapsto \perp\}), q_2)$ (from $m_H m_H$), and $((\langle\rangle, f_\perp), q_2)$ (from $\epsilon_H \epsilon_H$). Since all H-transitions are deterministic, and the initial states are $\sim$-related, all these states must be $\sim$-related, by LR_H. In each of the next three rounds, where $i \in \{3, 4, 5\}$, we have two possible states: $((\langle\rangle, f_\perp), q_i)$ and $((\langle\tau\rangle, f_\perp), q_i)$. Using LR_{Sys}, and the conclusion at level 2, we deduce that $((\langle\rangle, f_\perp), q_5)$ has to be $\sim$-related to $((\langle\tau\rangle, f_\perp), q_5)$, and then using SC on the L action m_L that $((\langle m_L \rangle, \{L \mapsto ack, H \mapsto \perp\}), q_6)$ and $((\langle\rangle, \{L \mapsto retry, H \mapsto \perp\}), q_5)$ are $\sim$-related. But these states have different L observations, contradicting OC. Therefore it is impossible to have an unwinding relation on $M \parallel \mathcal{A}$.

7. 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.

Rushby's separability. One related set of papers is Rushby's work on separability [Rushby81, Rushby00], which aims to define a security property for operating systems security kernels. A separation kernel provides each agent the abstraction of a local abstract machine, which is unaffected by the behaviour of other agents. 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.

Language based approaches. Language based information flow analysis, pioneered by Denning & Denning [DD77], is the subject of a large body of literature which has been surveyed in [SM03] (up to year 2003). By contrast to our focus on reactive systems and timing, most of this work deals with a single input and output, and is concerned with *storage channels* instead of *timing channels* (for the classification of *covert channels* we refer to [Lampson71]). Agat [Agat00] is the first to propose a bisimulation based definition of information flow that takes into account time spent on program execution in terms of the number of executable instructions. He proposes to close timing channels by padding programs with dummy computations, the transformation being automated using a type based mechanism.

Sheduling of concurrent threads has been considered in the context of language-based security. Volpano et al. [VS99] studied probabilistic information flow of multi-process programs in the presence of a uniform scheduler. 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. Their scheduler is also defined in a programming language syntax, so that the interaction between threads and the scheduler is made explicit. Their *noninterferent* schedulers also rely on a bisimulation-based relation, which is comparable to our definition of *oblivious* scheduler SOLTS. More recently, Mantel and Sudbrock defined a probabilistic trace-based scheduler-independent security property in a language-based setting [Mantel10]. They also design a sound type system to enforce that property.

Boudol and Castellani [BC02] applied a type system to rule out illegal flows of information in both sequential and concurrent settings with scheduling. In addition to typing constraints from the previous works (e.g., of [DD77]), they require that `while` loops guarded by High variables are not followed by Low assignments, so that termination of that loop will not be detectable by observing the result of the subsequent Low assignment. Another similar but independent work, using a uniform probabilistic scheduler, is [Smith01]. An extended and mechanized version is in [BN04]. A follow up work [ABC07] discusses noninterference for a class of synchronous programs called reactive programs, which incorporate features such as broadcasting, suspension and preemption. They have two different types of parallel composition: a global asynchronous composition, and a local synchronous composition with a deterministic cooperative scheduling discipline. They also identify a new type of illegal flow called *suspension leak*. Their security properties are all bisimulation-based and time-insensitive.

Compared with language based approaches on information flow with scheduling, our work proceeds on a different level, in which we define both system and scheduler in a language of automata with discrete-timed transitions. Therefore, our definitions do not naturally come with type systems. Another limit of our work is that it does not support dynamic process creation or dynamic policies, due to our static approach to system modeling. We focus on both trace-based and bisimulation-based definitions, in a time-sensitive way. Our model does not deal with probability, which is a modelling aspect that is orthogonal to nondeterminism (a good discussion of their relationship and combination is [Segala95]). Further research could be done by extending our automata model to a synchronous version of probabilistic automata [Segala95] with observations.

Process algebra. A number of recent works from the process algebra community also consider synchronous notions of process. 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 [FGM03]. They consider bisimulation based definitions of security that are based on NDS-like intuitions. [HR06] develop a set of failure-divergence based and low determinism based definitions [Roscoe95]

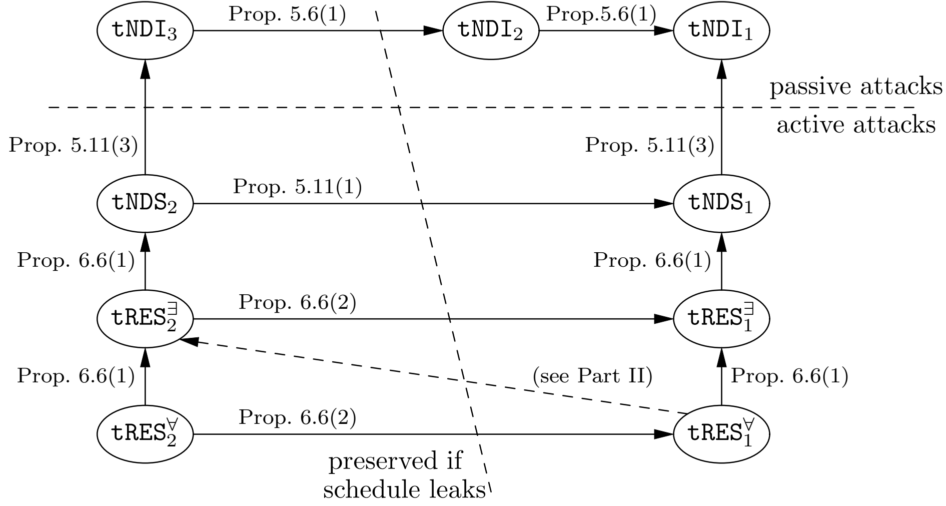

Figure 10: The relationship between synchronous scheduler-based security properties

in a version of timed CSP. The underlying model of timed CCS and timed CSP seems to be 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.

8. Conclusion and Future Work

In this paper we have introduced a class of security properties on a synchronous state machine model with a discrete-time semantics and a definition of scheduler. The security properties extend nondeducibility [Sutherland86], nondeducibility on strategies [WJ90] and restrictiveness [McCullough88] in a synchronous setting with scheduling. Our security definitions work with an abstract notion of scheduler as well as a concrete notion of scheduler implementation. We have identified which of the definitions of security are independent of the implementation of the scheduler. Except for one case, we have given a complete characterization of the relationships between the scheduler-implementation independent definitions of security we have introduced: the results are summarized in Figure 10, in which all implications as shown by the solid arrows are proper. The missing case corresponds to the dashed arrow, indicating that $\mathtt{tRES}_1^\forall$ implies $\mathtt{tRES}_2^\exists$. We establish this result in Part II of this series, since the proof uses results on refinement that are developed there.

Which of our notions of security are appropriate for a particular application depends on the circumstances. One of the factors is whether a passive

or an active attacker is of concern: the notions $\mathtt{tNDI}_i$ deal with a passive attacker model, and the notions $\mathtt{tNDS}_i$ deal with an active attacker model. A further distinction within each of these cases is whether the system should still be secure if the schedule were to become known to the low level agent L. The bisimulation-based definitions based on $\mathtt{tRES}_i$ are even stronger than these definitions, but, for independence of scheduler implementation, need to be defined using a quantification over scheduler implementations.

In principle, it is desirable to prove as strong a security property as possible for a given system. Thus, even if a weaker property such as $\mathtt{tNDI}_1$ suffices for the requirements of an application, it may be more convenient (and more informative) to prove that the system satisfies a stronger property such as $\mathtt{tRES}_2^{\forall}$. However, as our examples show, a stronger (and more easily proved) property may not always hold, preventing the application of such an approach to proving that a system is secure. This requires that an adequate, hopefully automatable, proof methodology exists for each of the definitions of interest. A first step in this direction is given in [CMZ10], which studies the complexity of several synchronous notions of security similar to those we have introduced here, though in a slightly different semantic model – it remains to relate the two frameworks to determine the impact of these results on the present definitions.

References

[Agat00] J. Agat. Transforming out timing leaks. In *Proc. ACM Symp. on Principles of Programming Languages*, pages 40–53, 2000.

[ABC07] A. Almeida Matos, G. Boudol and I. Castellani. Typing noninterference for reactive programs. *Journal of Logic and Algebraic Programming*, 72(2):124–156, 2007.

[BN04] G. Barthe and L. P. Nieto. Formally verifying information flow type systems for concurrent and thread systems. In *Proc. ACM Workshop on Formal Methods in Security Engineering*, pages 13-22, 2004.

[BG09] N. Busi and R. Gorrieri. Structural non-interference in elementary and trace nets. In *Mathematical Structures in Computer Science*, 19(6):1065–1090, 2009

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

[BFPR03] A. Bossi, R. Focardi, C. Piazza, and S. Rossi. Bisimulation and unwinding for verifying possibilistic security properties. In *Proc. Int. Conf. on Verification Model Checking, and Abstract Interpretation*, pages 223–237, 2003.

[BC02] G. Boudol and I. Castellani. Noninterference for concurrent programs and thread systems. *Theoretical Computer Science*, 281(1):109–130, 2002.

[BC92] P. Bieber and F. Cuppens. A logical view of secure dependencies. *Journal of Computer Security*, 1(1):99–129, 1992.

[CMZ10] F. Cassez, R. van der Meyden and C. Zhang. The complexity of synchronous otions of information flow security, *FoSSaCS 2010, 13th Int. Conf. on Foundations of Software Science and Computation Structures*, pages 282-296, 2010.

[DD77] D. E. Denning and P. J. Denning Certification of programs for secure information flow. *Communications of the ACM*, 20(7):504–513, 1977.

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

[FGM03] R. Focardi, R. Gorrieri and F. Martinelli. Real-time information flow analysis. *IEEE Journal on Selected Areas in Communications*, 21(1):20–35, 2003.

[FPR02] R. Focardi, C. Piazza, and S. Rossi. Proof methods for bisimulation based information flow security *Proc. of International Workshop on Verification Model Checking and Abstract Interpretation*, pages 16–31, 2002.

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

[GM84] J. A. Goguen and J. Meseguer. Unwinding and inference control. In *Proc. IEEE Symp. on Security and Privacy*, pages 75–87, 1984.

[Gligor93] V. D. Gligor. A guide to understanding covert channel analysis of trusted systems. National Computer Security Center, NCSC- TG-030, 1993.

[Gray91] J. W. Gray III. Toward a mathematical foundation for information flow security. In *Proc. IEEE Symp. on Security and Privacy*, pages 21–34, 1991

[HR02] M. Hennessy and J. Riely. Information flow vs. resource access in the asynchronous pi-calculus. In *ACM Transactions on Programming Languages and Systems*, 24(5):566–591, 2002.

[HR06] J. Huang and A. W. Roscoe. Extending noninterference properties to the timed world. In *Proc. ACM Symp. on Applied Computing*, pages 376–383, 2006.

[Lampson71] B. Lampson. Protection. In *Proc. 5th Princeton Symp. Information Sciences and Systems*, pages 437–443, 1971.

[Mantel00] H. Mantel. Unwinding security properties. In *Proc. European Symp. on Research in Computer Security*, pages 238–254, 2000.

[Mantel10] H. Mantel and H. Sudbrock. Flexible scheduler-independent security. In *Proc. European Symp. on Research in Computer Security*, pages 116–133, 2010.

[McCullough87] D. McCullough. Specifications for multi-level security and a hook-up property. In *Proc. IEEE Symp. on Security and Privacy*, pages 161–166, 1987.

[McCullough88] D. McCullough. Noninterference and the composability of security properties. In *Proc. IEEE Symp. on Security and Privacy*, pages 177–186, 1988.

[McLean94] J. McLean. A general theory of composition for trace sets closed under selective interleaving functions In *Proc. IEEE Symp. on Security and Privacy*, pages 79–93, 1994.

[Millen90] J. K. Millen. Hookup security for synchronous machines. In *Proc. IEEE Computer Security Foundation Workshop*, pages 84–90, 1990

[Roscoe95] A. W. Roscoe. CSP and determinism in security modelling. In *Proc. IEEE Symp. on Security and Privacy*, pages 114–121, 1995.

[Rushby81] J. Rushby. The design and verification of secure systems. *Proc. 8th ACM Symp. on Operating System Principles*, pages 12–21, 1981.

[Rushby82] J. Rushby. Proof of separability a verification technique for a class of security kernels. *Proc. 5th International Symp. on Programming*, pages 352–367, 1982.

[Rushby92] J. Rushby. Noninterference, transitivity, and channel-control security policies. Technical report, SRI international, Dec 1992.

[Rushby00] J. Rushby. Partitioning in avionics – architectures: requirements, mechanisms, and assurance. Technical report DOT/FAA/AR-99/58, SRI international, Mar 2000.

[RS06] A. Russo and A. Sabelfeld. Securing interaction between threads and the scheduler. *Proc. IEEE Computer Security Foundation Workshop*, pages 177–189, 2006.

[Ryan91] P. Y. A. Ryan. A CSP formulation of noninterference. *Cipher*, pages 19–27, 1991.

[SM03] A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):1–15, 2003.

[SS00] A. Sabelfeld and D. Sands. Probabilistic noninterference for multi-threaded programs. *Proc. IEEE Computer Security Foundation Workshop*, pages 200–214, 2000.

[Segala95] R. Segala. Modeling and verification of randomized distributed real-time systems. *PhD Thesis*, MIT, 1995.

[Smith01] G. Smith. A new type system for secure information flow. *Proc. IEEE Computer Security Foundation Workshop*, pages 115–125, 2001.

[Sutherland86] D. Sutherland. A model of information. *Proc. 9th National Computer Security Conference*, pages 175–183, 1986.

[vdMZ07] R. van der Meyden and C. Zhang. Algorithmic verification of noninterference properties. In *ENTCS*, volume 168, pages 61–75, 2007.

[vdMZ08] R. van der Meyden and C. Zhang. Information flow in systems with schedulers. In *Proc. Computer Security Foundation Symp.*, pages 301–312, 2008.

[vdMZ10] R. van der Meyden and C. Zhang. A comparison of semantic models for noninterference. In *Theoretical Computer Science*, 411(7):4123–4147, 2010.

[VS99] D. Volpano and G. Smith. Probabilistic noninterference in a concurrent language. In *Journal of Computer Securiy*, 7(2-3):231–253, 1999.

[Oheim04] D. von Oheimb. Information flow control revisited: Noninfluence = Noninterference + Nonleakage. In *Proc. European Symp. on Research in Computer Security*, pages 225–243, 2004.

[WW94] C. A. Waldspurger and W. E. Weihl. Lottery scheduling: flexible proportional-share resource management. In *Proc. 1st USENIX Symp. on Operating Systems Design and Implementation*, pages 1–11, 1994.

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

[Zhang09] C. Zhang. Information Flow Security — models, verification and schedulers. *PhD Thesis*, The University of New South Wales, 2009.