Information Flow in Systems with Schedulers (Part II: Refinement)

Ron van der Meyden[a], Chenyi Zhang[a,b]

[a]*School of Computer Science and Engineering, University of New South Wales, Sydney, Australia*
[b]*School of Information Technology and Electrical Engineering, University of Queensland, Brisbane, Australia*

Abstract

Refinement is a relation on systems models: a concrete model is a refinement of a more abstract model if it has fewer behaviours. When properties of the abstract model are guaranteed to be preserved in the concrete model, refinement supports a top-down development process. This paper considers preservation of a range of information flow security properties in synchronous systems with schedulers, when these schedulers are refined. Notions of refinement are defined for both an abstract notion of scheduler as well as for their concrete representation as automata. The security properties that are preserved by refinement over schedulers are then characterized. The results are applied to characterize a number of scheduler independent security properties, which state that a system is secure with respect to all schedulers.

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 flow to other agents. This paper is part II of a two part series in which we conduct a systematic study of the impact of schedulers on information-flow security. In part I of the series, we proposed a number of variants of existing definitions of security from the literature that newly accommodate the setting of synchronous scheduled systems, and study the relationships between the new definitions.

In deploying a system, it may be desirable to adapt the scheduler under which it runs to accommodate the performance requirements of the particular

Email addresses: meyden@cse.unsw.edu.au (Ron van der Meyden), czhang@unsw.edu.au (Chenyi Zhang)

deployment context. The maximum degree of flexibility for such adaptations obtains when the system satisfies the desired properties for *all* schedulers. Failing this, an alternate approach would be to first show that the system has these properties for some abstract scheduler, and then rely on a result stating that the desired properties are preserved however that scheduler is made more concrete. Such relationships between the abstract and the concrete are formalized in the literature using notions of *refinement*, which are relations on systems, defined by stating that system A refines system B if every observable behaviour of A is also an observable behaviour of B (see e.g. [GM93]). Such refinement relations can be used in a top-down development process, where they generally ensure that properties of the more liberal system description B are preserved in the more specific implementation represented by A.

However, many information-flow properties are *not* preserved by refinement, i.e., when A refines B, the fact that B satisfies an information-flow property does not necessarily imply that A also satisfies that property. This is known as the *refinement paradox* [Jac88, McL94]. The present paper considers which of the new definitions of security from Part I 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 work with a setting in which systems contain a low security level agent L and a high security level agent H, and in which the security policy states, intuitively, that information about H should not flow to L. The definitions we developed in part I are variants of the following three types of definitions from the prior literature which capture ways that L might deduce information about H:

1. *nondeducibility on inputs* [Sut86], 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* [McC87, McC88], a definition stronger than both of 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].

In developing variants of these notions that are suited for our setting of scheduled synchronous systems, we found that there is more than one plausible candidate for each of these notions in a setting with nondeterministic schedulers.

Non-deterministic schedulers allow multiple concrete schedules, and the precise schedule in a given run, which orders H, L and system actions, might not be known to L. We generally work with H-oblivious schedulers, i.e., schedulers that are independent of H's behaviour, so that the scheduler itself does not represent a channel for flow of information from H to L. One of the dimensions causing the bifurcation of definitions is the question of whether security should be preserved even if the precise schedule were to become known to L.

Another factor affecting the formulation of the definitions is the representation for schedulers. We work with two types of representation. One is abstract: it views a scheduler as a function that determines which agent can be scheduled next after a given history. The other representation is concrete: it implements an abstract scheduler as an automaton. It turns out that our new variants of nondeducibility on inputs and nondeducibility on strategies are not sensitive to the choice of concrete scheduler implementation. Our variants of restrictiveness require a concrete scheduler implementation in their statement, but are sensitive to the choice of implementation. In order to obtain implementation-independent variant of the restrictiveness definitions that is meaningful at the level of abstract schedulers, we need to quantify over scheduler implementations. The quantification can be done universally or existentially, each leading to a different definition. We summarize the resulting definitions and their relationships in Section 2, but we refer the reader to part I of the paper for a detailed discussion of the notions defined within each class.

Our contribution in the present paper is to consider refinement relations on schedulers and their impact on the definitions of security from part I. In particular, at the abstract level, we take one scheduler to refine another when, for all histories, the set of enabled agents for the latter scheduler is a subset of that for the former. We ask whether security is preserved under this notion of scheduler refinement, i.e., whether, if a system is secure with respect to a scheduler, it is also secure with respect to all its refinements.

When we consider our variants of nondeducibility on inputs and nondeducibility on strategies, we find that the only versions of these notions that are preserved under refinement are the versions that require that the the system to remain secure even if the schedule were to become known to L. In case of our four implementation-independent variants of restrictiveness, we again find that only the two stronger variants (which quantify universally over scheduler implementations) are preserved under scheduler refinement. In order to establish this result, we introduce a notion of refinement on concrete scheduler implementations, and develop results on scheduler refinement at the concrete level.

As a benefit of this study of refinement, we also obtain answers to the question of when a system is secure with respect to *all* schedulers. As noted above, this is a quite desirable property, since it means that the system can be very flexibly configured: the scheduler can be selected arbitrarily according to the requirements of the specific environment within which the system is to operate, without loss of security. One can ask this question for each of the notions of security from part I. It turns out that many of the distinctions collapse in this case. We characterize the collapse, and moreover show that for each of the remaining distinct notions, the question of security with respect to *all* schedulers can be characterized as security with respect to a *particular* scheduler. This fact helps to simplify the problem of verifying that the system is secure for all schedulers by reducing it to a single case.

The paper is structured as follows. In Section 2 we briefly recall the system model and the security notions from part I. Section 3 introduces a notion of refinement on schedulers and studies the security properties that are preserved

by this relation. Section 4 considers a new refinement relation on scheduler implementations which preserves some bisimulation based properties. Section 5 deals with a generalization of scheduler implementation independence — the security of systems for all schedulers in certain classes. Section 6 addresses related work and in Section 7 we summarize the results of the paper and make some concluding remarks.

2. Preliminaries

This section present the formal model on which the new security notions are defined. For a detailed discussion of the security motivations and examples to distinguish the notions, we refer to part I of the paper. We consider a synchronous model in which 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. We define a signature as 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. For $u \in D$ we define $A_u = \{a \in A \mid dom(a) = u\}$ to be the set of actions associated to domain u. For a number of semantic purposes (machines and schedulers), we use the following general type of model that is essentially a classical labelled transition system enhanced by observation functions.

Definition 2.1. *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.

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 $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} s_n$ and $\alpha = a_1 a_2 \ldots a_n$. A *run* r of a SOLTS is a sequence of the form $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} s_n$ with $s_0 \in S_0$. We write $\mathcal{R}(T)$ 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., $r_k = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \xrightarrow{a_k} s_k$, provided r has at least k transitions. We denote the sequence of actions in a run r by $Act(r) = a_1 a_2 \ldots a_n$, and for each agent u write $Act_u(r)$ for the subsequence of $Act(r)$ consisting of actions a with

$dom(a) = u$. A SOLTS is *deterministic* if for $s, t_1, t_2 \in S$ and $a \in A$, if $s \xrightarrow{a} t_1$ and $s \xrightarrow{a} t_2$ then $t_1 = t_2$. It is *input-enabled* if $s \xrightarrow{a}$ for all $s \in S$ and $a \in A$.

An agent's observations give it knowledge about the present and past. The following definition of the *view* function aims to capture this intention; a view is a trace of an agent's past, alternating observations and actions.

Definition 2.2. *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 $A_u \cup \{\frown\}$.

Like most of the literature, we confine our attention to systems with two security domains High (H) and Low (L) and the security policy $L \leq H$, which 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 $D = \{H, L, Sys\}$. The systems we study in the paper are defined as follows:

Definition 2.3. *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 condition that $A_{Sys} = \{\tau\}$ still allows that 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 automatic evolution of the state over time. Given an action sequence $\alpha \in A^*$ and $A' \subseteq A$, write $\alpha | A'$ for the sequence obtained from α by removing all actions not in A'.

Diagrammatic Convention for Machines: When presenting examples 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.)

2.1. Schedulers

The scheduler defined in this paper only resolves the nondeterminism concerning the next agent to act: we leave this agent *free will* to choose which action to perform when it is scheduled.

Definition 2.4. *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.*

We write Υ for the set of all schedulers. 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. Later we introduce the notion of scheduler SOLTS which provides a more concrete way to model this nondeterminism. A *schedule* is a finite or infinite sequence $sch = u_0u_1u_2u_3\ldots$ where each $u_i \in D$. For $\alpha = a_0a_1a_2\ldots$, 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 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 $\alpha \in A^*$. We write Υ^d for the set of deterministic schedulers.

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.) Let $sched : Q \to D$ map each scheduler SOLTS state to the unique agent that has its actions enabled in the state. A scheduler SOLTS $\langle Q, Q_0, \rightarrow, \{\bot\}, obs\rangle$ *represents* a scheduler σ if for all $\alpha \in A^*$ compatible with σ, we have $\sigma(\alpha) = \{sched(q) \mid q_0 \xrightarrow{\alpha} q,\ q_0 \in Q_0\}$.

Scheduling may be represented as a parallel composition of a machine and a scheduler SOLTS. Given a machine $M = \langle S, S_0, \rightarrow, O, obs\rangle$, and a scheduler SOLTS $\mathcal{A} = \langle Q, Q_0, \rightarrow', \bot, obs'\rangle$ with the same signature, the parallel composition $M \parallel \mathcal{A}$ is the SOLTS $\langle S \times Q, S_0 \times Q'_0, \rightarrow'', O, obs\rangle$ where $\rightarrow''= \{((s_1, q_1), a, (s_2, q_2)) \mid s_1 \xrightarrow{a} s_2 \wedge q_1 \xrightarrow{a} q_2\}$. This corresponds to the lock-step execution of the two systems with synchronisation on actions.

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 independent of the actions of an agent. For the definition, we need an operation on actions that masks actions of agent u: define $\mu_u(a) = a$ when $a \in A \setminus A_u$ and $\mu_u(a) = \bot_u$ when $a \in A_u$. For a sequence $\alpha = a_1 a_2 \ldots \in A^*$ 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 $u \in D$. Similarly, 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 $u \in D$. A dual notion called *schedule-awareness* is defined as follows. An agent u is schedule-aware in a scheduled machine (M, σ), if for all runs $r, r' \in \mathcal{R}(M, \sigma)$, we have that $view_u(r) = view_u(r')$ implies $sch(r) = sch(r')$. Intuitively, this says that the agent always knows the schedule of the current run.

It can be seen that for an oblivious scheduler σ, we have that $\sigma(\alpha)$ depends only on $sch(\alpha)$. We may therefore represent an oblivious scheduler σ by the set of all its schedules, or more concisely by a set whose prefix closure is the set of schedules generated by σ. We often do this when presenting examples, where we represent such sets of schedules by regular expressions, using operators $+$ for union, concatenation for sequencing, $*$ for (Kleene) finite iteration, and ω for infinite iteration.

As we confine ourselves to the policy $L \leq H$, we need to ensure that the schedules obtained, which may be observable to L, do not convey information to L about H's activity. Therefore we focus on H-oblivious schedulers, in which schedules do not carry any information about H actions. Write $\Upsilon^{\mathtt{HO}}$ for the set of schedulers that are H-oblivious, and $\Upsilon^{\mathtt{O}}$ for the set of oblivious schedulers.

We will be interested in definitions of security that classify a machine M as secure or insecure when it is scheduled according to a scheduler σ. We define the security of scheduled machines (M, σ), and 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. In the case when a scheduler SOLTS is given, we define security of the composite

system $M \parallel \mathcal{A}$. For our trace-based security properties, it can be shown that (M, σ) is secure iff $M \parallel \mathcal{A}$ is secure for any scheduler SOLTS $\mathcal{A}$ that represents σ. That is, such definitions are *implementation independent*. However, this is not true for our bisimulation-based properties. Detailed explanations of this are presented in Part I of the paper.

We also seek properties with respect to a set of schedulers. Given $\Sigma \subseteq \Upsilon$, we write $X(\Sigma)$ for the set of machines M such that $(M, \sigma) \in X$ for all $\sigma \in \Sigma$. (When $\Sigma = \{\sigma\}$, we write simply $X(\sigma)$.) This gives a notion that is of independent interest: if $M \in X(\Sigma)$, we are guaranteed that M is secure according to property X whatever scheduler in Σ is selected. This gives flexibility in the choice of a scheduler for the machine, which is desirable for machines that need to be deployed into diverse settings.

Diagrammatic Convention for Scheduler SOLTS: we depict scheduler SOLTS as graphs in which vertices correspond 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.

2.2. Trace-based Security Definitions

For asynchronous systems, the notion of *nondeducibility on inputs* (NDI) [Sut86] states that a system is secure if L cannot deduce from its view any information about the sequence of H actions that have been performed. We formulate the following three versions of NDI for scheduled machines.

Definition 2.5. *Let M be a machine and σ a scheduler.*

- $(M, \sigma) \in \mathtt{tNDI}_1$ *if for all possible L views $\beta \in Views_L(M, \sigma)$ 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.*
- $(M, \sigma) \in \mathtt{tNDI}_2$ *if for all possible L views $\beta \in Views_L(M, \sigma)$, 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 $view_L(r) = \beta$, where $\mathtt{Pna}_H(M, \sigma, \beta)$ is the set of numbers n such that there exists $r \in \mathcal{R}(M, \sigma)$ with $view_L(r) = \beta$ and $|Act_H(r)| = n$.*
- $(M, \sigma) \in \mathtt{tNDI}_3$ *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 $Act_H(r') = \alpha$.*

Intuitively, the first definition $\mathtt{tNDI}_1$ 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 notion 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. The second definition $\mathtt{tNDI}_2$ is based on this intuition that the possible numbers of H actions should be all that L knows about the H actions. (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 last definition $\mathtt{tNDI}_3$ states that L should not be able to make any deductions about what actions H has performed, even if L were to discover the schedule.

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* (NDS). We can formulate a variant of this notion as follows. First we capture the effect of a particular H-level process:

Definition 2.6. *An H* strategy *in a scheduled machine (M, σ) is a function $\pi : Views_H(M, \sigma) \to A_H$. 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.

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

Definition 2.7. *Let M be a machine and σ a scheduler.*

- $(M, \sigma) \in \mathtt{tNDS}_1$ *if for every $r \in \mathcal{R}(M, \sigma)$ and H strategy π, there exists $r' \in \mathcal{R}(M, \sigma, \pi)$ such that $view_L(r) = view_L(r')$.*
- $(M, \sigma) \in \mathtt{tNDS}_2$ *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 $sch(r) = sch(r')$.*

Intuitively, $\mathtt{tNDS}_1$ 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. However, 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 $\texttt{tNDS}_2$.

2.3. Bisimulation-based Security Definitions

McCullough introduced a security properties called *Restrictiveness* (RES) for nondeterministic event systems with states [McC88]. The essence of this notion in state-based model is a nondeterministic generalization of the notion of *unwinding relation* [GM84], which is a binary relation $\approx$ on the set of machine states. In asynchronous systems, an unwinding relation is 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 $obs_L(s) = obs_L(s')$.

(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 $s \approx t$.

For scheduled systems, we defined in part I two variants of this idea.

Definition 2.8. *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* $t \approx t'$.

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 $S \times Q$.

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.

A problem that then arises is two implementations of a scheduler may vary in their branching structure, but unwinding is sensitive to this structure. It turns out that satisfaction of $\mathtt{tRES}_1$ and $\mathtt{tRES}_2$ depends on the choice of the scheduler implementations: there exists systems M and implementations $\mathcal{A}_1$ and $\mathcal{A}_2$ of a scheduler σ such that $M \parallel \mathcal{A}_1$ and $M \parallel \mathcal{A}_2$ do not both satisfy these properties. Detailed examples are given in part I. This observation leads to the following notions, in which we quantify over scheduler implementations in order to obtain implementation independent definitions of security.

Definition 2.9. 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*.*

3. Refinement of Schedulers

Refinement relations are used in the literature to relate abstract system descriptions to more concrete system descriptions in such a way that certain properties of interest are preserved. Such a relation is useful in ensuring that design decisions in a top-down development approach preserve system properties established earlier in the design process. Refinement has been studied for a variety of underlying system semantics and types of property. In this section we introduce a notion of refinement between schedulers and study whether the trace-based security properties in systems with scheduling are preserved by this definition.

Definition 3.1. *For schedulers* σ *and* σ'*, we write* $\sigma' \sqsubseteq \sigma$*, and say that* σ refines σ'*, if* $\sigma(\alpha) \subseteq \sigma'(\alpha)$ *for all* α *compatible with* σ*.*

Intuitively, σ refines σ' if σ has fewer choices than σ'. (The contravariance of this definition follows the convention in the literature, e.g., [HHS87], in the sense that if A refines B then A is said to be more *concrete* and thus more *deterministic* than B.) We have the following property for the refinement relation.

Lemma 3.2. *For schedulers* σ *and* σ'*,* $\sigma' \sqsubseteq \sigma$ *iff for all* $\alpha \in A^*$*, if* α *is compatible with* σ *then* α *is compatible with* σ'*.*

Proof: Straightforward induction on the length of the action sequence α. □

Now we seek an answer to the question whether, given a machine M and a scheduler σ such that (M, σ) satisfies a property X, a refinement σ' of σ satisfies

$(M, \sigma') \in X$. We find that for some properties, such as $\mathtt{tNDI}_3$ and $\mathtt{tNDS}_2$, which require that no information is permitted to flow from H to L even when L knows the exact schedule of a particular run, security is preserved by H-oblivious refinements. That is, if $(M, \sigma) \in \mathtt{tNDI}_3$ (or $\mathtt{tNDS}_2$), then $(M, \sigma') \in \mathtt{tNDI}_3$ (or $\mathtt{tNDS}_2$) provided $\sigma \sqsubseteq \sigma'$ and $\sigma' \in \Upsilon^{\mathtt{HO}}$. However, for the other trace-based security properties ($\mathtt{tNDI}_1$, $\mathtt{tNDI}_2$ and $\mathtt{tNDS}_1$), this result does not hold. We start with the following lemma which says that a scheduler σ being H-oblivious is a necessary condition for a machine to satisfy $\mathtt{tNDI}_3(\sigma)$ or $\mathtt{tNDS}_2(\sigma)$.

Lemma 3.3. *For $X \in \{\mathtt{tNDI}_3, \mathtt{tNDS}_2\}$, if $(M, \sigma) \in X$ then σ is H-oblivious.*

Proof: Suppose $(M, \sigma) \in \mathtt{tNDI}_3$. We need to show that for all $\alpha, \alpha' \in A^*$, if $\mu_H(\alpha) = \mu_H(\alpha')$, then $\sigma(\alpha) = \sigma(\alpha')$. If α is compatible with σ, then $\sigma(\alpha)$ is nonempty. Let $u \in \sigma(\alpha)$, we show $u \in \sigma(\alpha')$. Let $a \in A$ satisfy $dom(a) = u$, then $\alpha \cdot a$ is compatible with σ, and it is now sufficient to show $\alpha' \cdot a$ is also compatible with σ. By $\alpha \cdot a$ compatible with σ, there exists a run $r \in \mathcal{R}(M, \sigma)$ with $Act(r) = \alpha \cdot a$. Since $\mu_H(\alpha) = \mu_H(\alpha')$, we have $|(\alpha | A_H)| = |(\alpha' | A_H)|$, and thus $|(\alpha \cdot a | A_H)| = |(\alpha' \cdot a | A_H)|$. Then by $(M, \sigma) \in \mathtt{tNDI}_3$, there exists a run $r' \in \mathcal{R}(M, \sigma)$ such that $view_L(r) = view_L(r')$, $sch(r) = sch(r')$ and $Act_H(r') = (\alpha' \cdot a)|A_H$. Because L's actions are contained in the L-view and there exists only one Sys action, it is obvious that $Act(r') = \alpha' \cdot a$, thus $u = dom(a) \in \sigma(\alpha')$. This proves $\sigma(\alpha) \subseteq \sigma(\alpha')$. The converse containment follows by symmetry.

If $(M, \sigma) \in \mathtt{tNDS}_2$ then by Lemma 5.2(3), which is also Proposition 5.11(3) in part I of the series, $(M, \sigma) \in \mathtt{tNDI}_3$, and the result immediately follows. □

The next lemma shows that for an H-oblivious scheduler, two runs in a machine are necessarily both compatible or both incompatible with the scheduler if they have the same L-view and they are scheduled in the same way.

Lemma 3.4. *If $r \in \mathcal{R}(M, \sigma)$ and σ is H-oblivious, then for all $r' \in \mathcal{R}(M)$, $sch(r') = sch(r)$ and $view_L(r') = view_L(r)$ implies $r' \in \mathcal{R}(M, \sigma)$.*

Proof: Given $r' \in \mathcal{R}(M)$ and σ H-oblivious, we can show that r' is compatible with σ by induction on the prefixes of r'. □

We can now show that $\mathtt{tNDI}_3$ and $\mathtt{tNDS}_2$ are preserved by refinement by an H-oblivious scheduler. (Note that the requirement that the refining scheduler be H-oblivious cannot be relaxed, by Lemma 3.3.)

Proposition 3.5. *If $(M, \sigma) \in X$ then $(M, \sigma') \in X$ for all H-oblivious schedulers σ' with $\sigma \sqsubseteq \sigma'$, where $X \in \{\mathtt{tNDI}_3, \mathtt{tNDS}_2\}$.*

Proof:

- Let $M \in \mathtt{tNDI}_3(\sigma)$ and $\sigma \sqsubseteq \sigma'$, we show $M \in \mathtt{tNDI}_3(\sigma')$. Let $r \in \mathcal{R}(M, \sigma')$ and $\alpha \in A_H^*$ such that $|\alpha| = |Act_H(r)|$. By $\sigma \sqsubseteq \sigma'$, r is also a run in $\mathcal{R}(M, \sigma)$. By $M \in \mathtt{tNDI}_3(\sigma)$, there exists a run $r' \in \mathcal{R}(M, \sigma)$ such that $Act_H(r') = \alpha$, $sch(r') = sch(r)$ and $view_L(r') = view_L(r)$. Since σ' is H-oblivious by assumption, together with $sch(r') = sch(r)$, $view_L(r') = view_L(r)$, by Lemma 3.4, we have $r' \in \mathcal{R}(M, \sigma')$. This gives $M \in \mathtt{tNDI}_3(\sigma')$.

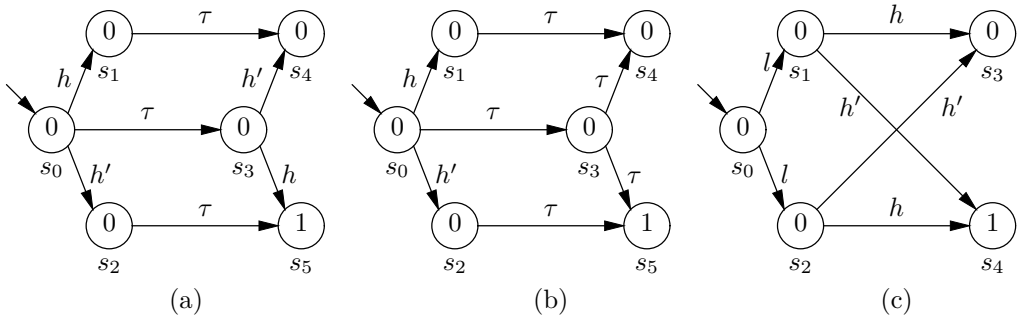

Figure 1: (a) a machine showing that $\texttt{tNDI}_2$ is not preserved by refinement, (b) a machine showing that $\texttt{tNDS}_1$ is not preserved by refinement (c) a machine showing $\texttt{tNDI}_3(\Upsilon^{\texttt{HO}}) \not\subseteq \texttt{tNDS}_1(\Upsilon^{\texttt{HO}})$

- Let $M \in \texttt{tNDS}_2(\sigma)$ and $\sigma \sqsubseteq \sigma'$, we show $M \in \texttt{tNDS}_2(\sigma')$. Let $r \in \mathcal{R}(M, \sigma')$ and let π be an H strategy. Then $r \in \mathcal{R}(M, \sigma)$. By $M \in \texttt{tNDS}_2(\sigma)$, there exists $r' \in \mathcal{R}(M, \sigma, \pi)$ such that $view_L(r') = view_L(r)$ and $sch(r) = sch(r')$. By assumption σ' is H-oblivious, thus r' is a run of $\mathcal{R}(M, \sigma')$ by Lemma 3.4, and r' is compatible with π. This gives all that we need to show $M \in \texttt{tNDS}_2(\sigma')$. □

A statement similar to that of Proposition 3.5 does not hold for $\texttt{tNDI}_1$, $\texttt{tNDI}_2$ and $\texttt{tNDS}_1$. For $\texttt{tNDI}_1$ and $\texttt{tNDI}_2$, Figure 1(a) represents a machine M satisfying $(M, \sigma) \in \texttt{tNDI}_2$ (hence $(M, \sigma) \in \texttt{tNDI}_1$) for the oblivious scheduler σ with set of schedules $(HSys + SysH)(H + L)^\omega$. Let σ' be the oblivious scheduler producing schedules of the form $HSys(H + L)^\omega$, then $\sigma \sqsubseteq \sigma'$ and σ' is H-oblivious. However, $(M, \sigma') \notin \texttt{tNDI}_1$, since the view $0 \frown 0 \frown 1$ is incompatible with any infinite H action sequences starting with h. Also $(M, \sigma') \notin \texttt{tNDI}_2$. Figure 1(b) represents a machine M satisfying $(M, \sigma) \in \texttt{tNDS}_1$ with oblivious scheduler σ producing schedules of the form $(H + Sys)Sys(H + L)^\omega$. Define σ' such that it produces a schedule of the form $HSysH^\omega$. Then $\sigma \sqsubseteq \sigma'$ and σ' is H-oblivious. (Moreover σ' is deterministic.) It is obvious that $(M, \sigma') \notin \texttt{tNDS}_1$.

Example 3.6. *As an example of the fact that* $\texttt{tNDI}_3$ *is preserved by refinement, consider the machine* M *depicted in Figure 1(c), and the oblivious scheduler* σ *that produces schedules in* $(L+H)^\omega$*. We have that* $(M, \sigma) \in \texttt{tNDI}_3$*. Because* Sys *is never scheduled,* L *always knows the schedule of the current run by observing when it is scheduled, but it has no way to determine from its view exactly which actions are being performed by* H*. Let another oblivious scheduler* σ' *produce schedules* $(LH)^\omega$*. Then we have* $\sigma \sqsubseteq \sigma'$*. It can also be shown that* $(M, \sigma') \in \texttt{tNDI}_3$*. Intuitively, since* σ' *generates a subset of the schedules of* σ*, and* M *is secure for every schedule in* σ*, the machine* M *is also secure for every schedule in* σ'*. (In this example the schedulers are* L*-oblivious. Proposition 3.5 also applies to the case where the scheduled domain may depend on what* L *actions have been performed.)*

Note that we refine just the scheduler, but not the system. We remark that the same example shows that $\mathtt{tNDI}_3$ *is not preserved by refinement of the system (by reducing nondeterminism in the system). For example, if the system were refined by removing the transition from* s_0 *to* s_2*, then* L *would be able to deduce from the view* $0\,\ell\,0 \frown 1$ *that* H *performed* h' *in the second step.*

Given the refinement relation on schedulers, one may regard a scheduler σ as a collection of its deterministic fragments, i.e., the set of deterministic schedulers that refine σ. Define σ^d as the set $\{\sigma' \in \Upsilon^d \mid \sigma \sqsubseteq \sigma'\}$. The following result states, intuitively, that the H-oblivious deterministic fragments of σ cover all the behaviour of σ.

Lemma 3.7.

1. $\mathcal{R}(M, \sigma') \subseteq \mathcal{R}(M, \sigma)$ *for all machines* M *and* $\sigma' \in \sigma^d$.
2. *If* σ *is* H*-oblivious, then for all* $r \in \mathcal{R}(M, \sigma)$*, there exists an* H*-oblivious scheduler* $\sigma' \in \sigma^d$ *such that* $r \in \mathcal{R}(M, \sigma')$.

Proof: For (1) it is trivial. For (2), given a run $r = s_0 \xrightarrow{a_1} s_1 \ldots \xrightarrow{a_n} s_n$ in $\mathcal{R}(M, \sigma)$, one can construct a deterministic H-oblivious scheduler σ' satisfying $\sigma'(a_1 \ldots a_i) = \{dom(a_{i+1})\}$ for all $i = 1 \ldots n$, in such a way that additionally, $\sigma'(\alpha) = \sigma'(\alpha')$ for all $\alpha, \alpha' \in A^*$ satisfying $\mu_H(\alpha) = \mu_H(\alpha')$. (This is done by assigning $\sigma'(\alpha) = \{dom(a_{i+1})\}$ whenever $\mu_H(\alpha) = \mu_H(a_1 \ldots a_i)$.) Then $a_1 \ldots a_n$ is also compatible with σ', so $r \in \mathcal{R}(M, \sigma')$. Note that such a scheduler may not be unique. □

Lemma 3.7(1) states that a refinement of a scheduler yields a smaller set of runs than the scheduler it refines. Lemma 3.7(2) states that the collection of H-oblivious deterministic fragments are sufficient to represent the behaviours of an H-oblivious scheduler.

We have shown that some properties, such as $\mathtt{tNDI}_3$ and $\mathtt{tNDS}_2$, are preserved by all H-oblivious refinements of a scheduler. Now we show the converse: if a machine is secure with respect to all H-oblivious schedulers in σ^d, then it is secure with respect to σ. Furthermore, this is the case not just for $\mathtt{tNDI}_3$ and $\mathtt{tNDS}_2$, but for all our trace-based properties.

Proposition 3.8. *For* $X \in \{\mathtt{tNDI}_1, \mathtt{tNDI}_2, \mathtt{tNDI}_3, \mathtt{tNDS}_1, \mathtt{tNDS}_2\}$ *and* σ *an* H*-oblivious scheduler, we have the following. If* $(M, \sigma') \in X$ *for all* H*-oblivious* $\sigma' \in \sigma^d$*, then* $(M, \sigma) \in X$.

Proof: For $\mathtt{tNDI}_1$, suppose a machine M does not satisfy $\mathtt{tNDI}_1(\sigma)$. Then M does not satisfy $\mathtt{tNDI}_3(\sigma)$, i.e., there exists a run $r \in \mathcal{R}(M, \sigma)$ and $\alpha \in A_H^*$ with $|\alpha| = |Act_H(r)|$ such that for all runs r', $sch(r) = sch(r')$ and $view_L(r) = view_L(r')$ implies that $Act_H(r') \neq \alpha$. By Lemma 3.7(2), there exists an H-oblivious scheduler $\sigma' \in \sigma^d$ such that $r \in \mathcal{R}(M, \sigma')$. We show that $M \notin \mathtt{tNDI}_1(\sigma')$. Since σ' is both deterministic and H-oblivious, L is schedule-aware in (M, σ'), so it is equivalent to show $M \notin \mathtt{tNDI}_3(\sigma')$. For this it suffices to show that for all $r'' \in \mathcal{R}(M, \sigma')$, $view_L(r'') = view_L(r)$ implies $Act_H(r'') \neq \alpha$.

(It is obvious that $sch(r'') = sch(r)$ and $|\alpha| = |Act_H(r)|$.) By Lemma 3.7(1) we have $r'' \in \mathcal{R}(M, \sigma)$. Then from the fact that for all runs r', $sch(r) = sch(r')$ and $view_L(r) = view_L(r')$ implies that $Act_H(r') \neq \alpha$, we have $Act_H(r'') \neq \alpha$. Therefore $M \notin \mathtt{tNDI}_3(\sigma')$, and thus $M \notin \mathtt{tNDI}_1(\sigma')$. The cases of $\mathtt{tNDI}_2(\sigma)$ and $\mathtt{tNDI}_3(\sigma)$ are similar.

For $\mathtt{tNDS}_1$, suppose a machine M does not satisfy $\mathtt{tNDS}_1(\sigma)$. Then M does not satisfy $\mathtt{tNDS}_2(\sigma)$. Thus there exists an H strategy π and run $r \in \mathcal{R}(M, \sigma)$ such that for all $r' \in \mathcal{R}(M, \sigma, \pi)$ with $sch(r') = sch(r)$, we have $view_L(r) \neq view_L(r')$. Similarly to the argument above, by Lemma 3.7(2) there exists an H-oblivious $\sigma' \in \sigma^d$ such that $r \in \mathcal{R}(M, \sigma')$. We show that $M \notin \mathtt{tNDS}_1(\sigma')$, which is equivalent to showing $M \notin \mathtt{tNDS}_2(\sigma')$, by the fact that L is schedule-aware in (M, σ'). Let r'' be a run of $\mathcal{R}(M, \sigma', \pi)$. We show that if $sch(r'') = sch(r)$ then $view_L(r'') \neq view_L(r)$. Suppose $view_L(r'') = view_L(r)$. Then from the fact that σ is H-oblivious, by induction on the length of r'' we have that r'' is also a run of $\mathcal{R}(M, \sigma, \pi)$. Thus from the fact that for all runs $r' \in \mathcal{R}(M, \sigma, \pi)$, $sch(r) = sch(r')$ implies $view_L(r) \neq view_L(r')$, we have $view_L(r'') \neq view_L(r)$, which is contradiction. Therefore $M \notin \mathtt{tNDS}_2(\sigma')$ and thus $M \notin \mathtt{tNDS}_1(\sigma')$. The case of $\mathtt{tNDS}_2(\sigma)$ can be proved in a similar way. □

Note that the contrapositive of this result means that to demonstrate that a system is insecure, it suffices to find a single deterministic schedule with respect to which it is insecure. We will use this fact in Section 5 to prove that some of the distinctions between our definitions of security collapse when we consider security with respect to all schedulers.

4. Refinement of Scheduler SOLTS

In Section 3, we considered a refinement relation between (abstract) schedulers, and studied the preservation of trace-based security definitions under this notion of refinement. This section, we take up the question of the preservation of the bisimulation-based definitions of the previous section under refinement. In particular, we show that $\mathtt{tRES}_1^\forall$ and $\mathtt{tRES}_2^\forall$ are preserved under scheduler refinement, but $\mathtt{tRES}_1^\exists$ and $\mathtt{tRES}_2^\exists$ are not. In order to do so, we first develop a notion of refinement on the concrete scheduler SOLTS representations of schedulers.

We define a refinement relation on scheduler SOLTS as the reverse of simulation on states of schedulers SOLTS. Given two scheduler SOLTS $\mathcal{A}_1 = \langle Q_1, Q_0^1, \rightarrow, O, obs\rangle$ and $\mathcal{A}_2 = \langle Q_2, Q_0^2, \rightarrow, O, obs\rangle$, a *simulation* of $\mathcal{A}_1$ by $\mathcal{A}_2$ is a relation $\leq \subseteq Q_1 \times Q_2$, such that if $q_1 \leq q_2$ then

1. $sched(q_1) = sched(q_2)$,
2. for all $a \in A$, if $q_1 \xrightarrow{a} q_1'$ then there exists q_2' such that $q_2 \xrightarrow{a} q_2'$ and $q_1' \leq q_2'$.

Intuitively, $q_1 \leq q_2$ means that every possible behavior from the state q_1 is also possible from the state q_2. Define a *bisimulation* between $\mathcal{A}_1$ and $\mathcal{A}_2$ to be a relation $\equiv \subseteq Q_1 \times Q_2$ such that if $q_1 \equiv q_2$ then

1. $sched(q_1) = sched(q_2)$,

2. for all $a \in A$, if $q_1 \xrightarrow{a} q_1'$ then there exists q_2' such that $q_2 \xrightarrow{a} q_2'$ and $q_1' \equiv q_2'$.
3. for all $a \in A$, if $q_2 \xrightarrow{a} q_2'$ then there exists q_1' such that $q_1 \xrightarrow{a} q_1'$ and $q_1' \equiv q_2'$.

Plainly, the identity relation on the states of $\mathcal{A}$ is a simulation of $\mathcal{A}$ by $\mathcal{A}$, in fact, a bisimulation. Given two scheduler SOLTS $\mathcal{A}_1$, $\mathcal{A}_2$, with states q_1, q_2, respectively, write $(\mathcal{A}_1, q_1) \leq (\mathcal{A}_2, q_2)$ if there exists a simulation of $\mathcal{A}_1$ by $\mathcal{A}_2$ such that $q_1 \leq q_2$. Similarly, write $(\mathcal{A}_1, q_1) \equiv (\mathcal{A}_2, q_2)$ if there exists a bisimulation between $\mathcal{A}_1$ and $\mathcal{A}_2$ such that $q_1 \equiv q_2$. We have the following observations.

Lemma 4.1. *For all q_1, q_2, q_3 as states of scheduler SOLTS $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$,*

1. *$(\mathcal{A}_1, q_1) \leq (\mathcal{A}_2, q_2)$ and $(\mathcal{A}_2, q_2) \leq (\mathcal{A}_3, q_3)$ implies $(\mathcal{A}_1, q_1) \leq (\mathcal{A}_3, q_3)$. (transitivity)*
2. *$(\mathcal{A}_1, q_1) \equiv (\mathcal{A}_2, q_2)$ implies $(\mathcal{A}_1, q_1) \leq (\mathcal{A}_2, q_2)$.*

Proof: Trivial by definition. □

Simulation can be used to define refinement between states of scheduler SOLTS. If state s simulates state t, then state s has at least as much behaviour as state t. We lift it to a relation between scheduler SOLTS, by defining a relation $\sqsubseteq_S$ on all scheduler SOLTS on the signature (A, D, dom).

Definition 4.2. *For H-oblivious scheduler SOLTS $\mathcal{A}_1$, $\mathcal{A}_2$, define $\mathcal{A}_1 \sqsubseteq_S \mathcal{A}_2$, or $\mathcal{A}_2$ refines $\mathcal{A}_1$, if for all initial states q_2 of $\mathcal{A}_2$, there exists an initial state q_1 of $\mathcal{A}$ such that $(\mathcal{A}_2, q_2) \leq (\mathcal{A}_1, q_1)$.*

Our main concern in this section is the preservation of the unwinding-based security notions under this notion of refinement. Before we come to the main results, we develop a few lemmas. The following result states that for two states in a machine running under a scheduler to be related by a sensitive synchronous unwinding, the corresponding scheduler states must be bisimilar. (Since it also talks about future behaviour, this is a stronger conclusion than that of the similar Lemma 6.2(1) in part I, which says only that the same agent is scheduled at states related by the unwinding.)

Lemma 4.3. *Let M be a machine and $\mathcal{A}$ be an H-oblivious scheduler SOLTS. For all states s_1, s_2 of M and q_1, q_2 of $\mathcal{A}$, if there exists a sensitive synchronous unwinding relation $\sim$ on $M \parallel \mathcal{A}$ such that $(s_1, q_1) \sim (s_2, q_2)$, then $(\mathcal{A}, q_1) \equiv (\mathcal{A}, q_2)$.*

Proof: Define a relation $\approx$ on the states of $\mathcal{A}$ by $q_1 \approx q_2$ if there exists a sensitive synchronous unwinding relation $\sim$, and states s_1, s_2 of M such that $(s_1, q_1) \sim (s_2, q_2)$. We show that $\approx$ is a bisimulation on the state space of $\mathcal{A}$. Let $q_1 \approx q_2$. Then by definition, there exists states s_1, s_2 of M such that $(s_1, q_1) \sim (s_2, q_2)$ by a sensitive synchronous unwinding relation $\sim$.

(1) $sched(q_1) = sched(q_2)$ is by $(s_1, q_1) \sim (s_2, q_2)$ and the definition of sensitive unwinding.

(2) We consider the cases $a \in A_L$, $a \in A_H$ and $a \in A_{Sys}$ separately.

For all $a \in A_L$, and $q_1 \xrightarrow{a} q_1'$, we have $(s_1, q_1) \xrightarrow{a} (s_1', q_1')$ for some s_1'. Since $\sim$ is a sensitive unwinding relation, by SC, there exists a state (s_2', q_2') such that $(s_2, q_2) \xrightarrow{a} (s_2', q_2')$ and $(s_1', q_1') \sim (s_2', q_2')$. So $q_1' \approx q_2'$ by definition, and $q_2 \xrightarrow{a} q_2'$.

For all actions $a \in A_H$, and $q_1 \xrightarrow{a} q_1'$, we have $(s_1, q_1) \xrightarrow{a} (s_1', q_1')$ for some s_1'. Since $\sim$ is a sensitive unwinding relation, by an instance of LR_H with pair of actions a, a, there exists a state (s_2', q_2') such that $(s_2, q_2) \xrightarrow{a} (s_2', q_2')$ and $(s_1', q_1') \sim (s_2', q_2')$. So $q_1' \approx q_2'$ by definition, and $q_2 \xrightarrow{a} q_2'$.

The case of $a \in A_{Sys}$ is similar to that of $a \in A_H$.

(3) The argument for part (3) of the definition of bisimulation is symmetric to that for (2). □

Using the above lemma, we can now show that the existence of a sensitive unwinding is preserved under refinement of the scheduler implementation.

Proposition 4.4. *For all machines M and H-oblivious scheduler SOLTS $\mathcal{A}_1$ and $\mathcal{A}_2$, if $\mathcal{A}_2 \sqsubseteq_S \mathcal{A}_1$ and there exists a sensitive unwinding relation on $M \parallel \mathcal{A}_2$, then there exists a sensitive unwinding relation on $M \parallel \mathcal{A}_1$.*

Proof: Suppose $\mathcal{A}_2 \sqsubseteq_S \mathcal{A}_1$. Let S be the state space of M, and Q_1, Q_2 be the state spaces of $\mathcal{A}_1$ and $\mathcal{A}_2$. Suppose $\sim \subseteq (S \times Q_2)^2$ is a sensitive unwinding relation on $M \parallel \mathcal{A}_2$, define a relation $\approx$ on $S \times Q_1$, by $(s_1, q_1) \approx (s_2, q_2)$ if $(\mathcal{A}_1, q_1) \equiv (\mathcal{A}_1, q_2)$ and there exists $r_1, r_2 \in Q_2$ with $(\mathcal{A}_1, q_1) \leq (\mathcal{A}_2, r_1)$, $(\mathcal{A}_1, q_2) \leq (\mathcal{A}_2, r_2)$, and $(s_1, r_1) \sim (s_2, r_2)$. We show that $\approx$ is a sensitive unwinding relation on $M \parallel \mathcal{A}_1$.

Let $(s_1, q_1) \approx (s_2, q_2)$, we show the following conditions hold.

- OC is trivial.
- To show SC, if $(s_1, q_1) \xrightarrow{a} (s_1', q_1')$ with $a \in A_L$, then $s_1 \xrightarrow{a} s_1'$ and $q_1 \xrightarrow{a} q_1'$. By $(\mathcal{A}_1, q_1) \leq (\mathcal{A}_2, r_1)$ there exists a state r_1' such that $r_1 \xrightarrow{a} r_1'$ and $(\mathcal{A}_1, q_1') \leq (\mathcal{A}_2, r_1')$. Therefore $(s_1, r_1) \xrightarrow{a} (t_1, r_1')$. Since $\sim$ is a sensitive unwinding relation there is a state (t_2, r_2') such that $(s_2, r_2) \xrightarrow{a} (t_2, r_2')$ and $(t_1, r_1') \sim (t_2, r_2')$. Then we have $s_2 \xrightarrow{a} t_2$. Since $(\mathcal{A}_1, q_1) \equiv (\mathcal{A}_1, q_2)$, there exists a state $q_2' \in Q_2$ with $q_2 \xrightarrow{a} q_2'$ and $(\mathcal{A}_1, q_1') \equiv (\mathcal{A}_1, q_2')$. Then we have $(s_2, q_2) \xrightarrow{a} (t_2, q_2')$. Note by Lemma 4.3 $(\mathcal{A}_2, r_1') \equiv (\mathcal{A}_2, r_2')$. Since $(\mathcal{A}_1, q_2') \equiv (\mathcal{A}_1, q_1')$ and $(\mathcal{A}_1, q_1') \leq (\mathcal{A}_2, r_1')$ we obtain $(\mathcal{A}_1, q_2') \leq (\mathcal{A}_2, r_2')$ by Lemma 4.1. Thus we have all that we need to show $(t_1, q_1') \approx (t_2, q_2')$.

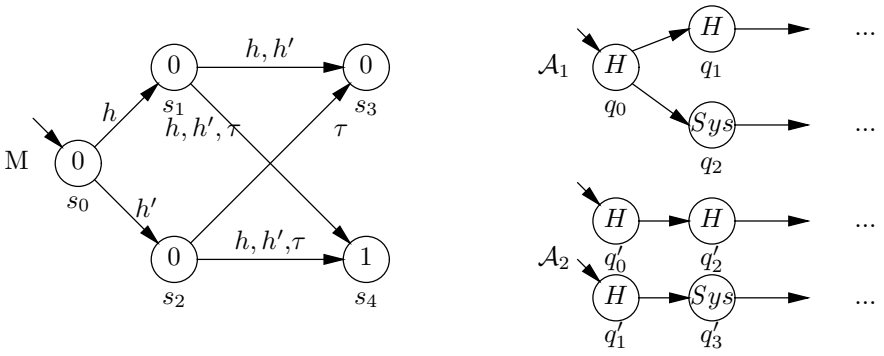

Figure 2: Insensitive unwinding is not preserved by scheduler SOLTS refinement

- For LR_H, let $a, b \in A_H$, if $(s_1, q_1) \xrightarrow{a} (t_1, q'_1)$, we show there exists a state (t_2, q'_2) such that $(s_2, q_2) \xrightarrow{b} (t_2, q'_2)$ and $(t_1, q'_1) \approx (t_2, q'_2)$. By assumption we have $s_1 \xrightarrow{a} t_1$ and $q_1 \xrightarrow{a} q'_1$. By $(\mathcal{A}_1, q_1) \leq (\mathcal{A}_2, r_1)$ there exists a state r'_1 such that $r_1 \xrightarrow{a} r'_1$ and $(\mathcal{A}_1, q'_1) \leq (\mathcal{A}_2, r'_1)$. Therefore $(s_1, r_1) \xrightarrow{a} (t_1, r'_1)$. Since $\sim$ is a sensitive unwinding relation there exists $(s_2, r_2) \xrightarrow{b} (t_2, r'_2)$ such that $(t_1, r'_1) \sim (t_2, r'_2)$, by LR_H. Therefore $s_2 \xrightarrow{b} t_2$. Since $(\mathcal{A}_1, q_1) \equiv (\mathcal{A}_1, q_2)$ by Lemma 4.3, there exists a state $q'_2 \in Q_2$ with $q_2 \xrightarrow{a} q'_2$ and $(\mathcal{A}_1, q'_1) \equiv (\mathcal{A}_2, q'_2)$. As $\mathcal{A}_1$ is H-oblivious we also have $q_2 \xrightarrow{b} q'_2$. Therefore $(s_2, q_2) \xrightarrow{b} (t_2, q'_2)$. Since $(\mathcal{A}_2, r'_1) \equiv (\mathcal{A}_2, r'_2)$ by Lemma 4.3, we have $(\mathcal{A}_1, q'_2) \leq (\mathcal{A}_2, r'_2)$ by Lemma 4.1. Thus we have all that we need to show that $(t_1, q'_1) \approx (t_2, q'_2)$.

- If $(s_1, q_1) \xrightarrow{\tau} (t_1, q'_1)$, then there exists a state (t_2, q'_2) such that $(s_2, q_2) \xrightarrow{\tau} (t_2, q'_2)$ and $(t_1, q'_1) \approx (t_2, q'_2)$. This can be proved in a similar way to the above cases.

- To show $(s_0, q_0^1) \approx (s_0, q_0^1)$ for every $s_0 \in S_0$ and $q_0^1 \in Q_1$, note that since $\mathcal{A}_2 \sqsubseteq_S \mathcal{A}_1$, there exists $q_0^2 \in Q_2$ such that $(\mathcal{A}_1, q_0^1) \leq (\mathcal{A}_2, q_0^2)$. Moreover we have $(\mathcal{A}_1, q_0^1) \equiv (\mathcal{A}_1, q_0^1)$, and $(s_0, q_0^2) \sim (s_0, q_0^2)$ by the fact that $\sim$ is a sensitive unwinding relation. □

Note that this result does not hold for insensitive unwinding relations. This can be seen from an example presented in Part I, shown here as Figure 2. We showed in part I that $\mathcal{A}_1$ and $\mathcal{A}_2$ represent the same scheduler, and there is an insensitive unwinding relation on $M \parallel \mathcal{A}_1$, but there are no insensitive unwinding relations on $M \parallel \mathcal{A}_2$. Since we also have that $\mathcal{A}_1 \sqsubseteq_S \mathcal{A}_2$, the same example shows that insensitive unwinding is not preserved by refinement of scheduler SOLTS.

The following example provides an illustration of Proposition 4.4.

Example 4.5. *Consider a system in which a transmission buffer is shared between two users, H and L, and the system Sys, under the control of a scheduler.*

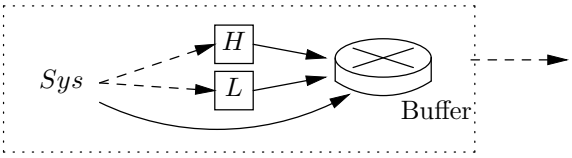

Figure 3: A transmission buffer example

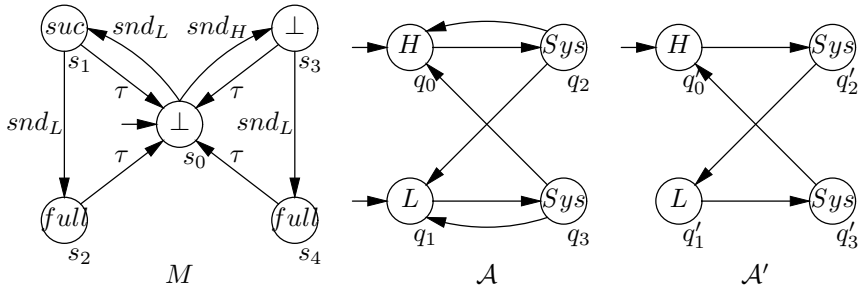

Figure 4: Machine for the buffer and two scheduler SOLTS

The high level structure of the system is shown in Figure 3. It works in the following way. A user H or L, when allocated a time slot, may request a message to be sent by writing it to the buffer. A user may also choose to skip. If the buffer contains a message when the system Sys is scheduled, it will dispatch the message immediately. The system does not change the buffer if it is empty.[2] *We assume the buffer has a capacity of at most one message. As usual, the security policy says that H is not allowed to interfere with L.*

A machine M representing the buffer is shown on the left of Figure 4. State s_0 represents that the buffer is empty, in states s_1 and s_1 it contains an L message and in states s_3 and s_4 it contains an H message. A user can attempt to write to the buffer by performing the action snd_L (or snd_H). If L writes to the buffer when it is empty, then L subsequently observes "suc", indicating that the write was successful. If the buffer is not empty when L tries to write, the write will fail and a "$full$" message is returned to L. (Our graphical convention for machines is that only L's observations are shown in the graph, but in this example we may assume, for the sake of simplicity, that H makes the same observation as L at every state, which implies that H can also determine, either from its observation or its knowledge of its action, whether the buffer contains a message.) Observation $\perp$ is used to denote empty observation. Recall that we elide self-loops, so, e.g., there is an implicit edge from s_1 to s_1 labelled snd_H.

Consider the scheduler SOLTS $\mathcal{A}$ in the middle of Figure 4, which represents an oblivious scheduler that generates schedules of the form $(HSys{+}LSys)^\omega$. We

[2]This is a simplified variant of the motivating example from Part I of the series, where the system may also store information in the buffer.

show that $M \parallel \mathcal{A}$ is secure, by showing that there exists a sensitive synchronous unwinding relation in the combined system $M \parallel \mathcal{A}$. For example, one may define a relation $\sim$ as the set

$$\{((s_0,q_0),(s_0,q_0)),\ ((s_0,q_1),(s_0,q_1)),\ ((s_0,q_2),(s_3,q_2)),\\ ((s_3,q_2),(s_0,q_2)),\ ((s_3,q_2),(s_3,q_2)),\ ((s_1,q_3),(s_1,q_3))\}.$$

This can be shown to be a sensitive unwinding relation on the scheduled system $M \parallel \mathcal{A}$. We leave this as an exercise for the interested reader.

Now consider the scheduler SOLTS $\mathcal{A}'$, on the right of Figure 4 which represents a scheduler that contains schedules of the form $(HSysLSys)^\omega$. Now we have $\mathcal{A} \sqsubseteq_S \mathcal{A}'$, i.e., the SOLTS $\mathcal{A}'$ refines the SOLTS $\mathcal{A}$, in that $\mathcal{A}'$ is simulated by $\mathcal{A}$. Taking effectively the same relation as above, i.e., defining $(s_i, q'_j) \sim' (s_k, q'_l)$ iff $(s_i, q_j) \sim (s_k, q_l)$, it can also be verified that $\sim'$ is a sensitive synchronous unwinding relation on $M \parallel \mathcal{A}'$.

Essentially, the machine M is insecure, but the only way for H to pass information to L is to write to the buffer and let L detect whether the buffer is full. A carefully designed scheduler, such as the one represented by $\mathcal{A}$, may rule out such possible flows. Moreover, removing choices from $\mathcal{A}$, as implemented in $\mathcal{A}'$, does not turn a secure system into an insecure one. □

Next, we connect the refinement relation $\sqsubseteq$ on schedulers and the refinement relation $\sqsubseteq_S$ on scheduler SOLTS by the following lemmas. Given a scheduler σ, its *characteristic* scheduler $\mathcal{A}^\sigma$ (as per Definition 4.3 in Part I of the paper) is $\langle Q, Q_0, \rightarrow, \{\bot\}, obs\rangle$ where

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

If σ is H-oblivious, in a similar way, $\mathcal{A}^\sigma_H$ is a deterministic H-oblivious scheduler SOLTS that characterizes σ. We show that both $\mathcal{A}^\sigma$ and $\mathcal{A}^\sigma_H$ are *maximal* implementations, in the sense that if σ is refined by σ', then $\mathcal{A}^\sigma$ is refined by every implementation of σ'.

Lemma 4.6. *Let σ and σ' be two (H-oblivious) schedulers satisfying $\sigma' \sqsubseteq \sigma$. Then $\mathcal{A}^{\sigma'} \sqsubseteq_S \mathcal{A}$ ($\mathcal{A}^{\sigma'}_H \sqsubseteq_S \mathcal{A}$) for every ($H$-oblivious) scheduler SOLTS $\mathcal{A}$ representing σ.*

Proof: Let $\mathcal{A}^{\sigma'} = \langle Q', Q'_0, \rightarrow, \{\bot\}, obs\rangle$, and let $\mathcal{A} = \langle Q, Q_0, \rightarrow, \{\bot\}, obs\rangle$ such that $\mathcal{A}$ represents σ, i.e., for all sequences $\alpha \in A^*$ compatible with σ, we have $\sigma_{\mathcal{A}}(\alpha) = \sigma(\alpha)$. Now we define a relation $\leq \subseteq Q \times Q'$ by for all $q_1 \in Q$ and $q_2 \in Q'$, $q_1 \leq q_2$ if there exists $\alpha \in A^*$ and $q_0 \in Q_0$ such that $q_0 \xrightarrow{\alpha} q_1$ and $q_2 = (\alpha, sched(q_1))$. We show that $\leq$ is a simulation.

1. If $q_1 \leq q_2$ then for some α, we have $q_2 = (\alpha, sched(q_1))$, so $sched(q_2) = sched(q_1)$, as required.

2. Suppose $q_1 \leq (\alpha, u)$ and $q_1 \xrightarrow{a} q_1'$. Then there exists $q_0 \in Q_0$ such that $q_0 \xrightarrow{\alpha} q_1$ and $u = sched(q_1) = dom(a)$. Thus $u \in \sigma_{\mathcal{A}}(\alpha) = \sigma(\alpha)$, hence $u \in \sigma'(\alpha)$ by $\sigma' \sqsubseteq \sigma$. Moreover, with $v = sched(q_1')$, we have $v \in \sigma_{\mathcal{A}}(\alpha \cdot a) = \sigma(\alpha \cdot a)$, hence $v \in \sigma'(\alpha \cdot a)$. By construction of $\mathcal{A}^{\sigma'}$, $(\alpha, u) \xrightarrow{a} (\alpha \cdot a, v)$ and since $q_0 \xrightarrow{\alpha \cdot a} q'$, we have $q_1' \leq (\alpha \cdot a, v)$.

It is obvious that for all $q_0 \in Q$, we have $q_0 \leq (\epsilon, sched(q_0)) \in Q_0'$. Therefore $\mathcal{A} \leq \mathcal{A}^{\sigma'}$, and $\mathcal{A}^{\sigma'} \sqsubseteq_S \mathcal{A}$.

Suppose σ' is H-oblivious. An argument similar to that above shows that for every H-oblivious scheduler SOLTS $\mathcal{A}$ representing σ we have $\mathcal{A} \leq \mathcal{A}_H^{\sigma'}$, and thus $\mathcal{A}_H^{\sigma'} \sqsubseteq_S \mathcal{A}$. □

In particular, it follows from this result that if there exists a refinement relation between two schedulers, we can establish a refinement relation between their corresponding characteristic scheduler SOLTS. Moreover, we obtain the equivalence of $\mathtt{tRES}_2^{\forall}$ with respect to a scheduler σ and the existence of sensitive unwinding relation on the SOLTS $M \parallel \mathcal{A}^{\sigma}$.

Corollary 4.7.

1. *$\mathcal{A}^{\sigma_1} \sqsubseteq_S \mathcal{A}^{\sigma_2}$ and $\mathcal{A}_H^{\sigma_1} \sqsubseteq_S \mathcal{A}_H^{\sigma_2}$ for all schedulers σ_1, σ_2 satisfying $\sigma_1 \sqsubseteq \sigma_2$.*
2. *$(M, \sigma) \in \mathtt{tRES}_2^{\forall}$ iff there exists a sensitive unwinding relation on $M \parallel \mathcal{A}_H^{\sigma}$.*

Proof: Part (1) is immediate from Lemma 4.6. For part (2), suppose that there exists a sensitive unwinding on $M \parallel \mathcal{A}_H^{\sigma}$. Let $\mathcal{A}$ be any H-oblivious scheduler SOLTS representing σ. By Lemma 4.6, $\mathcal{A}_H^{\sigma} \sqsubseteq_S \mathcal{A}$. Thus, by Proposition 4.4, there exists a sensitive unwinding on $M \parallel \mathcal{A}$. This shows that $(M, \sigma) \in \mathtt{tRES}_2^{\forall}$. The converse is immediate from the definition of $\mathtt{tRES}_2^{\forall}$ and the fact that $\mathcal{A}_H^{\sigma}$ is an H-oblivious scheduler SOLTS representing σ. □

The above connections imply the following result that $\mathtt{tRES}_2^{\forall}$ is preserved by refinement of H-oblivious schedulers.

Proposition 4.8. *Let σ_1, σ_2 be two H-oblivious schedulers satisfying $\sigma_1 \sqsubseteq \sigma_2$. Then for all machines M, if $M \in \mathtt{tRES}_2^{\forall}(\sigma_1)$ then $M \in \mathtt{tRES}_2^{\forall}(\sigma_2)$.*

Proof: Suppose $\sigma_1 \sqsubseteq \sigma_2$. Let M be a machine in $\mathtt{tRES}_2^{\forall}(\sigma_1)$. Since $M \in \mathtt{tRES}_2^{\forall}(\sigma_1)$ and $\mathcal{A}_H^{\sigma_1}$ is an H-oblivious scheduler SOLTS representing σ_1, there exists a sensitive unwinding relation on $M \parallel \mathcal{A}_H^{\sigma_1}$. By Lemma 4.6, $\mathcal{A}_H^{\sigma_1} \sqsubseteq_S \mathcal{A}'$ for every H-oblivious scheduler SOLTS $\mathcal{A}'$ representing σ_2. Thus, by Proposition 4.4, there also exists a sensitive unwinding relation on $M \parallel \mathcal{A}'$ for all $\mathcal{A}'$ representing σ_2. Therefore $M \in \mathtt{tRES}_2^{\forall}(\sigma_2)$. □

A similar result can be proved for $\mathtt{tRES}_1^{\forall}$ using a simpler proof technique. (Actually, this simpler technique could also be applied to give proof of Proposition 4.8, but the sequence of results on scheduler SOLTS refinement yields more information, and we have other applications for them below.)

Proposition 4.9. *Let σ_1, σ_2 be two H-oblivious schedulers satisfying $\sigma_1 \sqsubseteq \sigma_2$. Then for all machines M, if $M \in \mathtt{tRES}_1^{\forall}(\sigma_1)$ then $M \in \mathtt{tRES}_1^{\forall}(\sigma_2)$.*

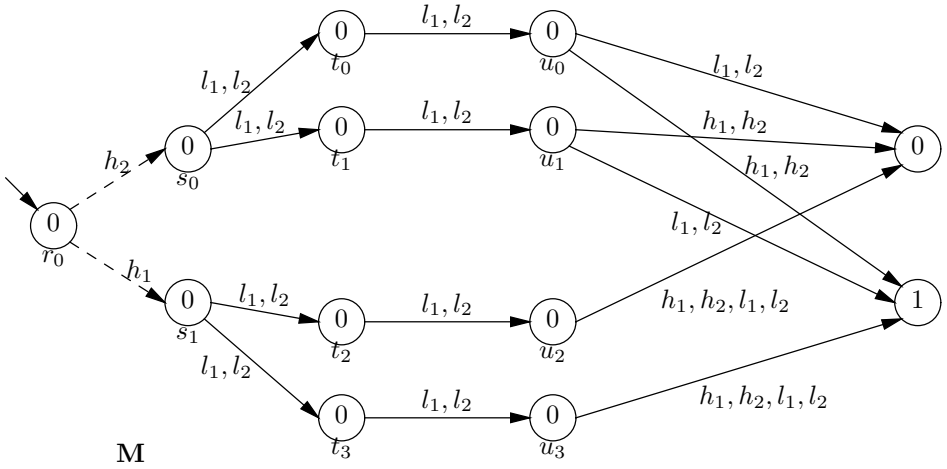

Figure 5: A counterexample showing $\texttt{tRES}_2^{\exists}$ is not preserved by refinement

Proof: Suppose $(M, \sigma_1) \in \texttt{tRES}_1^{\forall}$, we show that $(M, \sigma_2) \in \texttt{tRES}_1^{\forall}$. Let $\mathcal{A}$ be a scheduler SOLTS that implements σ_2, with state space Q. Define a scheduler SOLTS $\mathcal{A}'$ that is constructed by taking a disjoint union of $\mathcal{A}_H^{\sigma_1}$ with $\mathcal{A}$. First we show that $\mathcal{A}'$ represents σ_1. Given $\alpha \in A^*$, we have $\sigma_{\mathcal{A}'}(\alpha) = \sigma_1(\alpha) \cup \sigma_2(\alpha)$, by $\mathcal{A}_H^{\sigma_1}$ representing σ_1 and $\mathcal{A}$ representing σ_2. Since σ_1 is refined by σ_2, we have $\sigma_2(\alpha) \subseteq \sigma_1(\alpha)$, and thus $\sigma_{\mathcal{A}'}(\alpha) = \sigma_1(\alpha)$.

Since $M \in \texttt{tRES}_1^{\forall}(\sigma_1)$, there exists an insensitive unwinding relation $\sim$ on $M \parallel \mathcal{A}'$. Let M have state space S, and we define $\sim' = \sim \cap (S \times Q)$ to be the restriction of $\sim$ to the component $M \parallel \mathcal{A}$ of $M \parallel \mathcal{A}'$. (Note that $M \parallel \mathcal{A}'$ consists of two disconnected components $M \parallel \mathcal{A}_H^{\sigma_1}$ and $M \parallel \mathcal{A}$. It follows from the fact that $\sim$ is an insensitive unwinding relation on $M \parallel \mathcal{A}'$ that $\sim'$ is an insensitive unwinding relation on $M \parallel \mathcal{A}$. For the constraint $s_0 \sim' s_0$ on initial states s_0 this is straightforward, as is the constraints OC for $\sim'$. For SC, note that if $s \sim' t$ and $s \xrightarrow{a} s'$ where s, t, s are states of $M \parallel \mathcal{A}$, then we have $s \sim t$ and $s \xrightarrow{a} s'$ in $M \parallel \mathcal{A}'$. By SC for $M \parallel \mathcal{A}'$ we have that there exists a state t' of $M \parallel \mathcal{A}'$ with $t \xrightarrow{a} t'$ and $t \sim t'$. But since $\mathcal{A}'$ is constructed as a disjoint union, the fact that $t \xrightarrow{a} t'$ and t is a state of $\mathcal{A}$ implies that also t' is a state of $\mathcal{A}$, and it follows that $t \sim' t'$, giving the requirement of SC for $\mathcal{A}$. The other constraints for insensitive unwinding follow by a similar argument. □

However, neither $\texttt{tRES}_1^{\exists}$ nor $\texttt{tRES}_2^{\exists}$ is preserved by refinement of H-oblivious schedulers. We take the machine M which is depicted in Figure 5. Let the H-oblivious scheduler σ encode schedules in the form $HLL(H + L)L^{\omega}$. Then we have $M \in \texttt{tRES}_2^{\exists}(\sigma)$, i.e., there exists an H-oblivious scheduler SOLTS representing σ, such that an (in)sensitive unwinding relation can be found on the composed SOLTS. To see this, consider the scheduler SOLTS $\mathcal{A}_1$ shown in Figure 7. A synchronous sensitive unwinding relation $\sim$ on $M \parallel \mathcal{A}_1$ can be constructed as follows.

- For the lower schedule $HLLL\ldots$, we let $(s_0, q_1) \sim (s_0, q_1)$, $(t_1, q_1') \sim (t_2, q_1')$, $(s_1, q_1'') \sim (s_3, q_1'')$ and $(s_2, q_1'') \sim (s_4, q_1'')$. Since the fourth transition is scheduled to L, s_5 and s_7 both lead to "0" as L's observations. Similarly, s_6 and s_8 both lead to "1" as L's observations.

- For the upper schedule $HLLH\ldots$, we have $(s_0, q_1) \sim (s_0, q_1)$, $(t_1, q_1') \sim (t_2, q_1')$, $(s_1, q_1'') \sim (s_4, q_1'')$ and $(s_2, q_1'') \sim (s_3, q_1'')$. Now since the fourth transition is scheduled to H, s_5 and s_7 both lead to L's observation "0", and s_6 and s_8 both lead to L's observation "1".

Therefore $(M, \sigma) \in \mathtt{tRES}_2^{\exists}$, and thus $(M, \sigma) \in \mathtt{tRES}_1^{\exists}$, by Proposition 6.6(2) in part I. Next, we define an H-oblivious scheduler σ' satisfying $\sigma \sqsubseteq \sigma'$ and $M \notin \mathtt{tRES}_2^{\exists}(\sigma')$.

- $\sigma'(\epsilon) = \{H\}$,
- $\sigma'(h_1) = \sigma'(h_2) = \{L\}$,
- $\sigma'(\alpha) = \{L\}$ for all compatible α with $|\alpha| = 2$,
- $\sigma'(x\, l_1\, l_1) = \sigma'(x\, l_2\, l_2) = \{H\}$ for all $x \in A_H$,
- $\sigma'(x\, l_1\, l_2) = \sigma'(x\, l_2\, l_1) = \{L\}$ for all $x \in A_H$,
- $\sigma'(\alpha) = \{L\}$ for all compatible α with $|\alpha| > 3$.

We show that for all scheduler SOLTS $\mathcal{A}$ that represent σ', there does not exist a synchronous sensitive unwinding relation on $M \parallel \mathcal{A}$. Intuitively, the reason is as follows. Note that in the second and third step, L performs two actions, and under the scheduler σ', these actions determine whether L or H is scheduled in the fourth step. Each of the four horizontal streams under this scheduler results in a single final observation. The choice of L or H in the fourth step can change the final observation in the upper two streams, but not in the lower two streams. However, an unwinding would need to relate, already after the second step, each one of the upper two streams to one of the lower two streams in such a way as to lead to the same final observation. Since there is, after the second step, not enough information to know which lower stream will produce the same result as an upper stream, this cannot be done.

More precisely, suppose there were a scheduler SOLTS $\mathcal{A} = \langle Q, Q_0, \rightarrow, \{\bot\}, obs\rangle$ representing σ' and an unwinding relation $\sim$ on $M \parallel \mathcal{A}$. For $q_0 \in Q_0$, we would have $(s_0, q_0) \sim (s_0, q_0)$. Then step by step we show that the existence of $\sim$ leads to contradiction.

1. Since $\sigma'(\epsilon) = \{H\}$, we must have $sched(q_0) = H$. Let q_1 be a state of $\mathcal{A}$ such that $(s_0, q_0) \xrightarrow{h_1} (t_1, q_1)$. Choosing h_1, h_2 for an application of LR_H, there exists a state (t_2, q_1') for some $q_1' \in Q$, such that $(s_0, q_0) \xrightarrow{h_2} (t_2, q_1')$ and $(t_1, q_1) \sim (t_2, q_1')$. By definition of σ', we have $sched(q_1) = sched(q_1') = L$.

2. Since $(t_1, q_1) \sim (t_2, q_1')$, we choose $l_1 \in A_L$ for an application of SC. Let q_2 be a state of $\mathcal{A}$ such that $(t_1, q_1) \xrightarrow{l_1} (s_1, q_2)$. Then by SC, there exists some $s \in \{s_3, s_4\}$ and $q_2' \in Q$, such that $(t_2, q_1') \xrightarrow{l_1} (s, q_2')$ and $(s_1, q_2) \sim (s, q_2')$. By definition of σ', we have $sched(q_2) = sched(q_2') = L$. For the destination state s, we show it is impossible to have $(s_1, q_2) \sim (s, q_2')$ no matter whether we let s be s_3 or s_4.
   (a) Suppose $s = s_3$. We show it cannot be the case that $(s_1, q_2) \sim (s_3, q_2')$. Because if so, let us choose $l_1 \in A_L$ as the action for the next transition. Since M is deterministic from s_1 and s_3, it follows using SC that we would have $(s_5, q_3) \sim (s_7, q_3')$ for some $q_3, q_3' \in Q$. Since $\sigma'(h_1\, l_1\, l_1) = \sigma'(h_2\, l_1\, l_1) = \{H\}$, we must have that $sched(q_3) = sched(q_3') = H$. Now an application of action h_1 from (s_5, q_3) leads to the state (v_0, q_4), and by LR_H we must be able to match this by a transition on h_1 from (s_7, q_3'). Any such transition must be to a state of the form (v_1, q_4') for some $q_4' \in Q$, so that we have $(v_0, q_4) \sim (v_1, q_4')$ by LR_H. But since L makes observation 0 at (v_0, q_4) and observation 1 at (v_1, q_4'), this gives a contradiction with OC.
   (b) Suppose $s = s_4$. An argument similar to that above shows that we cannot have $(s_1, q_2) \sim (s_4, q_2')$ either. For, if so, then we choose $l_2 \in A_L$ as the action for the next transition. Since $\sigma'(h_1\, l_1\, l_2) = \sigma'(h_2\, l_1\, l_2) = \{L\}$, the fourth step will be an action of L. The determinism of M, together with two applications of SC, leads to the conclusion that we must have a relation of the form $(v_0, q_4) \sim (v_1, q_4')$, which again contradicts OC.

Thus, there does not exist a sensitive unwinding relation on $M \parallel \mathcal{A}$, i.e., $(M, \sigma') \notin \mathtt{tRES}_2^{\exists}$. As both σ and σ' never schedule Sys, there also does not exist an insensitive unwinding relation on $M \parallel \mathcal{A}$, and we have $(M, \sigma') \notin \mathtt{tRES}_1^{\exists}$. Nevertheless, one may easily observe that (M, σ') is in $\mathtt{tNDS}_2$, and this is another example showing that the inclusion $\mathtt{tRES}_2^{\exists} \subset \mathtt{tNDS}_2$ is proper.

Recall that for scheduled machines in which L is schedule-aware, the tNDI properties collapse (see Lemma 5.2(1), which is Proposition 5.6(2) in part I of the series) as do the tNDS properties (see Lemma 5.2(2), which is proposition 5.11(3) in part I of the series). We now consider whether there is a similar collapse for the tRES properties. Since the tRES properties concern scheduler implementations, the existence of unwinding relations in a scheduled machine (M, σ) does not seem to be related with L's ability to know the current schedule. We strengthen the requirement by showing the relationship between tRES properties when a scheduler is deterministic. (Recall that L is schedule-aware when the scheduler is deterministic and H-oblivious.)

Proposition 4.10. *If H-oblivious scheduler SOLTS $\mathcal{A}$ represents deterministic scheduler σ, then $\mathcal{A} \sqsubseteq_S \mathcal{A}_H^{\sigma}$.*

Proof: Recall that the characteristic scheduler SOLTS $\mathcal{A}_H^{\sigma}$ has the state space $(\mu_H(A))^* \times D$, initial states $\{(\epsilon, v) \mid v \in \sigma(\epsilon)\}$, and transition relation defined

by $(\gamma, v) \xrightarrow{a} (\gamma', w)$ iff $dom(a) = v$, $\gamma' = \gamma \cdot \mu_H(a)$ and $w \in \sigma(\gamma')$. Let the scheduler SOLTS $\mathcal{A} = \langle Q, Q_0, \rightarrow, \{\bot\}, obs\rangle$ represent σ, i.e., for all states q and $\gamma \in A^*$, we have $\sigma(\gamma) = \{sched(q) \mid q_0 \xrightarrow{\gamma} q, q_0 \in Q_0\}$. Let σ be deterministic, i.e., for all compatible $\alpha \in A^*$, we have that $\sigma(\alpha)$ is a singleton set. We show that $\mathcal{A}^\sigma_H$ is simulated by $\mathcal{A}$.

Define the relation $\leq \subseteq (\mu_H(A)^* \times D) \times Q$, by $(\gamma, u) \leq q$ if there exists $q_0 \in Q_0$ such that $q_0 \xrightarrow{\gamma} q$ and $u = sched(q)$. We show that $\leq$ is a simulation. Suppose that $(\gamma, u) \leq q$. We check the two conditions for $\leq$ to be a simulation. By definition, we have that $q_0 \xrightarrow{\gamma} q$ and $u = sched(q)$ for some $q_0 \in Q_0$.

(1) Since $sched((\gamma, u)) = u$, we have that $sched((\gamma, u)) = u = sched(q)$, as required.

(2) Suppose $a \in A$ and $(\gamma, u) \xrightarrow{a} (\gamma', v)$. Then $dom(a) = sched((\gamma, u)) = u = sched(q)$, so a is enabled at q. Also, by construction of $\mathcal{A}^\sigma$, we have $\gamma' = \gamma \cdot \mu_H(a)$ and $v \in \sigma(\gamma')$. Let $q \xrightarrow{a} q'$. Then $q_0 \xrightarrow{\gamma} q \xrightarrow{a} q'$, and we obtain that $q_0 \xrightarrow{\gamma'} q'$. Since $\mathcal{A}$ represents σ, it follows that $sched(q') \in \sigma(\gamma')$. But, since σ is deterministic, and we also have $v \in \sigma(\gamma')$, this implies that $sched(q') = v$. Thus we have $q_0 \xrightarrow{\gamma'} q'$ and $v = sched(q')$, so that $(\gamma', v) \leq q'$. We now have that $q \xrightarrow{a} q'$ and $(\gamma', v) \leq q'$, as required by part (2) of the definition of simulation.

It remains to show that for all initial states (ϵ, v) of $\mathcal{A}^\sigma$ there exists an initial state q_0 of $\mathcal{A}$ such that $(\epsilon, v) \leq q_0$. By determinism of σ, there is a unique agent v such that (ϵ, v) is an initial state of $\mathcal{A}^\sigma$. Take q_0 to be any initial state of $\mathcal{A}$. Since $\mathcal{A}$ represents σ, we also have $sched(q_0) = v$. Moreover, $q_0 \xrightarrow{\epsilon} q_0$, so we have $(\epsilon, v) \leq q_0$. We now have all that we need for $\mathcal{A} \sqsubseteq_S \mathcal{A}^\sigma_H$. □

A useful corollary of this result is that the distinction between $\mathtt{tRES}^\forall_i$ and $\mathtt{tRES}^\exists_i$ collapses with respect to deterministic schedulers, and *any* scheduler SOLTS representing the scheduler can be used for the verification of these properties. (We already used this fact in some of the examples above.)

Corollary 4.11. *If H-oblivious scheduler SOLTS $\mathcal{A}$ represents deterministic scheduler σ, then the following are equivalent:*

1. $(M, \sigma) \in \mathtt{tRES}^\forall_1$,
2. $(M, \sigma) \in \mathtt{tRES}^\forall_2$,
3. $(M, \sigma) \in \mathtt{tRES}^\exists_1$,
4. $(M, \sigma) \in \mathtt{tRES}^\exists_2$,
5. *there exists an insensitive unwinding relation on* $M \parallel \mathcal{A}$,
6. *there exists a sensitive unwinding relation on* $M \parallel \mathcal{A}$.

Proof: Since σ is deterministic, there exists a sensitive unwinding relation on $M \parallel \mathcal{A}'$ iff there exists an insensitive unwinding relation on $M \parallel \mathcal{A}'$, by Lemma 6.4 in part I. It therefore suffices to show the equivalence of (2),(4) and (6).

From (2) to (6) is immediate from the definition: if $(M, \sigma) \in \mathtt{tRES}_2^\forall$ and $\mathcal{A}$ represents σ, then there exists a sensitive unwinding relation on $M \parallel \mathcal{A}$. It is also immediate from the definition that (6) implies (4). To show (4) implies (2), suppose there exists a sensitive unwinding relation on $M \parallel \mathcal{A}'$. We have $\mathcal{A}' \sqsubseteq_S \mathcal{A}_H^\sigma$ by Proposition 4.10. It follows using Proposition 4.4 that there is a sensitive unwinding relation on $M \parallel \mathcal{A}_H^\sigma$. Now, $(M, \sigma) \in \mathtt{tRES}_2^\forall$ follows by Corollary 4.7(2). □

Proof that $\mathtt{tRES}_1^\forall \subseteq \mathtt{tRES}_2^\exists$:

We use the results in this section to finish the only missing case for the the relationships between the properties proposed in Part I.

To show this, we treat σ as a collection of deterministic schedulers refining σ, i.e. the set σ^d.

Given an action sequence α, write σ^α for the set $\{\sigma' \in \sigma^d \mid \sigma'(\alpha) \neq \emptyset\}$. That is, we take the subset of deterministic H-oblivious fragments of σ that are compatible with α. (Note by the well-formedness assumption, if α is incompatible with σ' then $\sigma'(\alpha) = \emptyset$.)

Lemma 4.12. *For all $\alpha \in A^*$, we have $\sigma(\alpha) = \bigcup_{\sigma' \in \sigma^\alpha} \sigma'(\alpha)$.*

Proof: We assume α is compatible with σ, because otherwise both sides have empty sets. For all $u \in \sigma(\alpha)$, we show $u \in \bigcup_{\sigma' \in \sigma^\alpha} \sigma'(\alpha)$. Construct an infinite sequence $\gamma \in A^\omega$ that has its every prefix compatible with σ, and also has $\alpha \cdot a$ as a prefix with $dom(a) = u$. Then it is obvious that σ_γ is a deterministic H-oblivious scheduler that is compatible with σ and $\sigma_\gamma(\alpha) = \{u\}$. Since every prefix of γ is compatible with σ, we can show that $\sigma \sqsubseteq \sigma_\gamma$. Therefore $\sigma_\gamma \in \sigma^\alpha$. Therefore $u \in \bigcup_{\sigma' \in \sigma^\alpha} \sigma'(\alpha)$.

For all $u \in \bigcup_{\sigma' \in \sigma^\alpha} \sigma'(\alpha)$, there exists $\sigma' \in \sigma^\alpha$ such that $\sigma'(\alpha) = u$. Since $\sigma \sqsubseteq \sigma'$, we have $u \in \sigma(\alpha)$ by definition. □

Lemma 4.13. *Given σ, for each $\sigma' \in \sigma^d$, let $\mathcal{A}_{\sigma'}$ be a scheduler SOLTS that represent σ'. Then if $\mathcal{A}(\sigma)$ is the scheduler SOLTS constructed as the disjoint union of the $\mathcal{A}_{\sigma'}$ with $\sigma' \in \sigma^d$, then $\mathcal{A}(\sigma)$ represents σ.*

Proof: We only need to show that for all $\alpha \in A^*$, we have $\sigma_{\mathcal{A}(\sigma)}(\alpha) = \sigma(\alpha)$. In fact $\sigma_{\mathcal{A}(\sigma)}(\alpha) = \bigcup_{\sigma' \in \sigma^d} \mathcal{A}_{\sigma'}(\alpha) = \bigcup_{\sigma' \in \sigma^d} \sigma'(\alpha)$. By Lemma 4.12, we have $\bigcup_{\sigma' \in \sigma^d} \sigma'(\alpha) = \sigma(\alpha)$, and the result immediately follows. □

Proposition 4.14. *For all machines M and schedulers σ, if $(M, \sigma) \in \mathtt{tRES}_1^\forall$ then $(M, \sigma) \in \mathtt{tRES}_2^\exists$.*

Proof: Suppose $(M, \sigma) \in \mathtt{tRES}_1^\forall$. For every $\sigma' \in \sigma^d$, since $\sigma \sqsubseteq \sigma'$, we have $(M, \sigma') \in \mathtt{tRES}_1^\forall$ by Proposition 4.9. Since σ' is deterministic and H-oblivious, by Corollary 4.11, we have $(M, \sigma') \in \mathtt{tRES}_2^\exists$, i.e., there exists a scheduler SOLTS $\mathcal{A}_{\sigma'}$ that represents σ', such that there exists a sensitive unwinding relation on $M \parallel \mathcal{A}_{\sigma'}$. Taking the disjoint union of the scheduler SOLTS $\mathcal{A}_{\sigma'}$ for all $\sigma' \in \sigma^d$, we get a scheduler SOLTS $\mathcal{A}(\sigma)$ which, by Lemma 4.13, represents σ. Therefore we have a sensitive unwinding relation on $M \parallel \mathcal{A}(\sigma)$. This proves $(M, \sigma) \in \mathtt{tRES}_2^\exists$. □

5. Scheduler-Independent Properties

All the previous properties are about security with respect to a particular scheduler. As already noted above, it is also of interest to consider security with respect to sets of schedulers, since this gives flexibility to configure the system. In this section, we consider security with respect to the set of *all* H-oblivious schedulers, corresponding to flexibility to choose an arbitrary scheduler while retaining security.

It turns out that some of the distinct trace-based notions of security we have identified collapse when we require security with respect to all H-oblivious schedulers. However, we do not obtain a similar collapse for the bisimulation-based notions. We also show that for the trace-based properties, it is possible to give a very simple characterization of security with respect to all H-oblivious schedulers in terms of security with respect to one *particular* scheduler. We also find such characterizations for two of the bisimulation-based notions ($\mathtt{tRES}_1^\forall$ and $\mathtt{tRES}_2^\forall$). Moreover, in one case ($\mathtt{tRES}_2^\forall$) we are able to reduce security with respect to all schedulers to security with respect to a particular scheduler SOLTS.

Intuitively, given a system controlled by a nondeterministic scheduler, if there is evidence that the system is insecure, then that evidence can also be produced by a finite deterministic fragment of that scheduler. As a consequence of Proposition 3.8, we obtain the following results.

Proposition 5.1. *For all* $X \in \{\mathtt{tNDI}_1, \mathtt{tNDI}_2, \mathtt{tNDI}_3, \mathtt{tNDS}_1, \mathtt{tNDS}_2\}$*, we have* $X(\Upsilon^{\mathtt{HO}}) = X(\Upsilon^{\mathtt{HO}} \cap \Upsilon^d)$.

Proof: As $\Upsilon^{\mathtt{HO}} \cap \Upsilon^d \subseteq \Upsilon^{\mathtt{HO}}$, we only need to show that for all machines M, if $M \in X(\Upsilon^{\mathtt{HO}} \cap \Upsilon^d)$ then $M \in X(\Upsilon^{\mathtt{HO}})$. For $\mathtt{tNDI}_1$, if $M \notin \mathtt{tNDI}_1(\Upsilon^{\mathtt{HO}})$, we show $M \notin \mathtt{tNDI}_1(\Upsilon^{\mathtt{HO}} \cap \Upsilon^d)$. By assumption there exists an H-oblivious scheduler σ such that $M \notin \mathtt{tNDI}_1(\sigma)$. From Proposition 3.8, there exists $\sigma' \in \sigma^d \subseteq \Upsilon^{\mathtt{HO}} \cap \Upsilon^d$ such that $M \notin \mathtt{tNDI}_1(\sigma')$. Hence, $M \notin \mathtt{tNDI}_1(\Upsilon^{\mathtt{HO}} \cap \Upsilon^d)$. The cases of $\mathtt{tNDI}_2, \mathtt{tNDI}_3, \mathtt{tNDS}_1, \mathtt{tNDS}_2$, can be proved similarly. □

Using this result, we obtain that there is a collapse of the trace-based definitions of security when we require security for all H-oblivious schedulers. We first recall the relationship between trace-based security properties by the following lemma. (The results have been proved as Proposition 5.6 and Proposition 5.11 in part I of the series.)

Lemma 5.2. 1. *If L is schedule-aware in (M, σ), then $(M, \sigma) \in \mathtt{tNDI}_1$ iff $(M, \sigma) \in \mathtt{tNDI}_2$ iff $(M, \sigma) \in \mathtt{tNDI}_3$.*
2. *If L is schedule-aware in (M, σ), then $(M, \sigma) \in \mathtt{tNDS}_1$ iff $(M, \sigma) \in \mathtt{tNDS}_2$.*
3. $\mathtt{tNDS}_1 \subseteq \mathtt{tNDI}_1$ *and* $\mathtt{tNDS}_2 \subseteq \mathtt{tNDI}_3$.

Corollary 5.3. $\mathtt{tNDS}_1(\Upsilon^{\mathtt{HO}}) = \mathtt{tNDS}_2(\Upsilon^{\mathtt{HO}}) \subseteq \mathtt{tNDI}_1(\Upsilon^{\mathtt{HO}}) = \mathtt{tNDI}_2(\Upsilon^{\mathtt{HO}}) = \mathtt{tNDI}_3(\Upsilon^{\mathtt{HO}})$

Proof: By Proposition 5.1 we have $X(\Upsilon^{\mathtt{HO}}) = X(\Upsilon^{\mathtt{HO}} \cap \Upsilon^d)$ for all trace-based properties X. Since L is schedule-aware for every scheduler σ in $\Upsilon^{\mathtt{HO}} \cap \Upsilon^d$,

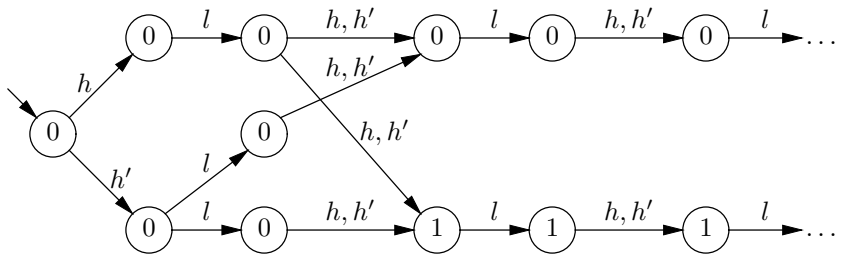

Figure 6: A machine showing containment $\mathtt{tRES}_1^{\exists}(\Upsilon^{\mathtt{HO}}) \subseteq \mathtt{tNDS}_1(\Upsilon^{\mathtt{HO}})$ is strict.

we have $\mathtt{tNDI}_1(\sigma) = \mathtt{tNDI}_2(\sigma) = \mathtt{tNDI}_3(\sigma)$, by Lemma 5.2(1) and $\mathtt{tNDS}_1(\sigma) = \mathtt{tNDS}_2(\sigma)$, by Lemma 5.2(2). By Lemma 5.2(3), we have $\mathtt{tNDS}_1(\sigma) \subseteq \mathtt{tNDI}_1(\sigma)$. The result immediately follows. □

To show that $\mathtt{tNDS}_i(\Upsilon^{\mathtt{HO}})$ for $i \in \{1, 2\}$ is strictly stronger than $\mathtt{tNDI}_j(\Upsilon^{\mathtt{HO}})$ for $j \in \{1, 2, 3\}$ we refer to Figure 1(c) which is a machine in $\mathtt{tNDI}_3(\sigma)$ for all deterministic H-oblivious σ, but it is obviously not in $\mathtt{tNDS}_1(\Upsilon^{\mathtt{HO}})$.

We now show, using the results on refinement relations over schedulers, that we are able to characterize security over the set of schedulers $\Upsilon^{\mathtt{HO}}$ by security with respect to a single scheduler. Define a chaos scheduler σ_c by $\sigma_c(\alpha) = D = \{H, L, Sys\}$ for all $\alpha \in A^*$.

Theorem 5.4. *$X(\Upsilon^{\mathtt{HO}}) = X(\sigma_c)$ for $X \in \{\mathtt{tNDI}_1, \mathtt{tNDI}_2, \mathtt{tNDI}_3, \mathtt{tNDS}_1, \mathtt{tNDS}_2\}$.*

Proof: By Corollary 5.3, it suffices to consider only $X = \mathtt{tNDI}_3$ and $X = \mathtt{tNDS}_2$. Since $\sigma_c \in \Upsilon^{\mathtt{HO}}$, we only need to show $X(\sigma_c) \subseteq X(\Upsilon^{\mathtt{HO}})$. Let X be $\mathtt{tNDI}_3$ and let M be a machine with $M \in \mathtt{tNDI}_3(\sigma_c)$. By Proposition 5.1, it suffices to show to show that for all $\sigma \in \Upsilon^{\mathtt{HO}} \cap \Upsilon^d$ we have $M \in \mathtt{tNDI}_3(\sigma)$. This can be derived by Proposition 3.5 using $\sigma_c \sqsubseteq \sigma$ and σ is H-oblivious. For $X = \mathtt{tNDS}_2$ it can be proved in a similar way. □

This result helps us to reduce verification of a trace-based property with respect to $\Upsilon^{\mathtt{HO}}$ to the same security property with respect to the particular scheduler σ_c. (Recall that the trace-based properties are scheduler-implementation independent, so we may further work with the smallest implementation of σ_c for verification purposes.)

Considering the relation to the bisimulation-based properties, it is immediate from Proposition 6.6 in part I [vdMZ12] and Proposition 4.14 that we have the chain of containments $\mathtt{tRES}_2^{\forall}(\Upsilon^{\mathtt{HO}}) \subseteq \mathtt{tRES}_1^{\forall}(\Upsilon^{\mathtt{HO}}) \subseteq \mathtt{tRES}_2^{\exists}(\Upsilon^{\mathtt{HO}}) \subseteq \mathtt{tRES}_2^{\exists}(\Upsilon^{\mathtt{HO}}) \subseteq \mathtt{tNDS}_i(\Upsilon^{\mathtt{HO}})$, for $i \in \{1, 2\}$.

The containment $\mathtt{tRES}_1^{\exists}(\Upsilon^{\mathtt{HO}}) \subseteq \mathtt{tNDS}_1(\Upsilon^{\mathtt{HO}})$ can be seen to be strict using the above characterization of $\mathtt{tNDS}_1(\Upsilon^{\mathtt{HO}})$ as $\mathtt{tNDS}_1(\sigma_c)$. Consider the machine M of Figure 6. We argued in part I of our series (in the proof of Proposition 6.8) that this is in $\mathtt{tNDS}_1(\sigma_{rr})$ for the scheduler σ_{rr} that alternates H and L; similar reasoning shows that it is in $\mathtt{tNDS}_1(\sigma_c)$: even for this more flexible scheduler, all that H can control with its strategy is whether the upper or the lower branch is taken, but both yield the same possible L views. Hence $M \in \mathtt{tNDS}_1(\Upsilon^{\mathtt{HO}})$. We already have shown in part I that $M \notin \mathtt{tRES}_1^{\exists}(\sigma_{rr})$, hence $M \notin \mathtt{tRES}_1^{\exists}(\Upsilon^{\mathtt{HO}})$.

Now it is straightforward that we have that $\mathtt{tRES}_2^{\exists}(\Upsilon^{\mathtt{HO}}) \subseteq \mathtt{tNDS}_2(\Upsilon^{\mathtt{HO}})$ is also strict, since $\mathtt{tRES}_2^{\exists}(\Upsilon^{\mathtt{HO}}) \subseteq \mathtt{tRES}_1^{\exists}(\Upsilon^{\mathtt{HO}})$, $\mathtt{tRES}_1^{\exists}(\Upsilon^{\mathtt{HO}}) \subset \mathtt{tNDS}_1(\Upsilon^{\mathtt{HO}})$, and $\mathtt{tNDS}_1(\Upsilon^{\mathtt{HO}}) = \mathtt{tNDS}_2(\Upsilon^{\mathtt{HO}})$ by Corollary 5.3. For the relationships between the rest of the properties, by Proposition 4.14, we have $\mathtt{tRES}_1^{\forall}(\Upsilon^{\mathtt{HO}}) \subseteq \mathtt{tRES}_2^{\exists}(\Upsilon^{\mathtt{HO}})$, and the following result (Proposition 5.5) shows that this inclusion is strict. Furthermore, it is immediate that the containments $\mathtt{tRES}_1^{\forall}(\Upsilon^{\mathtt{HO}}) \subseteq \mathtt{tRES}_1^{\exists}(\Upsilon^{\mathtt{HO}})$ and $\mathtt{tRES}_2^{\forall}(\Upsilon^{\mathtt{HO}}) \subseteq \mathtt{tRES}_2^{\exists}(\Upsilon^{\mathtt{HO}})$ are both strict, by $\mathtt{tRES}_1^{\forall}(\Upsilon^{\mathtt{HO}}) \subset \mathtt{tRES}_2^{\exists}(\Upsilon^{\mathtt{HO}}) \subseteq \mathtt{tRES}_1^{\exists}(\Upsilon^{\mathtt{HO}})$, and by $\mathtt{tRES}_2^{\forall}(\Upsilon^{\mathtt{HO}}) \subseteq \mathtt{tRES}_1^{\forall}(\Upsilon^{\mathtt{HO}}) \subset \mathtt{tRES}_2^{\exists}(\Upsilon^{\mathtt{HO}})$, respectively. We refer to Figure 11 as an overview of the aforementioned relationships, where the solid arrows denote strict inclusion and dashed arrows denote inclusion with their strictness left open in this paper.

Proposition 5.5. $\mathtt{tRES}_2^{\exists}(\Upsilon^{\mathtt{HO}}) \not\subseteq \mathtt{tRES}_1^{\forall}(\Upsilon^{\mathtt{HO}})$.

Proof: To show this, we assume the signature $(\{h, h', l, \tau\}, \{H, L, Sys\}, dom)$ with $dom(h) = dom(h') = H$, $dom(l) = L$, and $dom(\tau) = Sys$. Consider the machine M and the scheduler SOLTS $\mathcal{A}_1$ and $\mathcal{A}_2$ depicted in Figure 7. We used this example in part I to show that sensitive unwinding relation is implementation dependent: we showed there that there exists a sensitive unwinding relation on $M \parallel \mathcal{A}_1$, but there do not exist sensitive unwinding relations on $M \parallel \mathcal{A}_2$. Thus, we have $M \notin \mathtt{tRES}_2^{\forall}(\Upsilon^{\mathtt{HO}})$ since there does not exist a synchronous sensitive unwinding relation on $M \parallel \mathcal{A}_2$. Moreover, as Sys is never scheduled in the scheduler represented by $\mathcal{A}_2$, there does not exist a synchronous insensitive unwinding relation on $M \parallel \mathcal{A}_2$, either. Therefore $M \notin \mathtt{tRES}_1^{\forall}(\Upsilon^{\mathtt{HO}})$.

However, we have $M \in \mathtt{tRES}_2^{\exists}(\Upsilon^{\mathtt{HO}})$. The argument for this is as follows. Since L and Sys have a single action each, every H-oblivious scheduler σ can be implemented by an H-oblivious scheduler SOLTS $\mathcal{A}$, in which all transitions are deterministic, and the only source of nondeterminism is that there exist multiple initial states. More precisely, consider a scheduler SOLTS $\mathcal{A}_1$ consisting of an infinite set of states q_0, q_1, ..., with $q_i \xrightarrow{a} q_{i+1}$ for all actions a enabled on q_i. Every scheduler σ can be implemented by a scheduler SOLTS $\mathcal{A}$ that is a disjoint union of such linear scheduler SOLTS. We show that there exists a sensitive unwinding on $M \parallel \mathcal{A}_1$ for such linear $\mathcal{A}_1$, and by taking unions of these sensitive unwindings, it follows that there exists a sensitive unwinding on $\mathcal{A}$, hence $M \in \mathtt{tRES}_2^{\exists}(\Upsilon^{\mathtt{HO}})$.

We briefly describe how to establish a sensitive unwinding relation $\sim$ on $M \parallel \mathcal{A}_1$. Note that we do not leave state s_0 until H is scheduled. Let q_i be the first state in $\mathcal{A}$ with $sched(q_i) = H$, and let $(s_0, q_j) \sim (s_0, q_j)$ for all $0 \leq j \leq i$. In the next step we move to s or t, and remain there until L is scheduled. Let q_k be the first state in $\mathcal{A}$ such that $k > i$ and $sched(q_k) = L$, and then we let $(s, q_j) \sim (t, q_j)$ for all $i + 1 \leq j \leq k$. To construct the unwinding on the states reached by this next L action, we need to look ahead to the next time H is scheduled. Let α be the longest sequence of states in the form $q_k q_{k+1} \dots q_m$ satisfying $sch(\alpha) \in L(D \setminus \{L\})^* L Sys^*$. We have the following two cases.

- If $sched(q_{m+1}) = L$, we let $(s_1, q_{k+1}) \sim (s_3, q_{k+1})$ and $(s_2, q_{k+1}) \sim (s_4, q_{k+1})$.

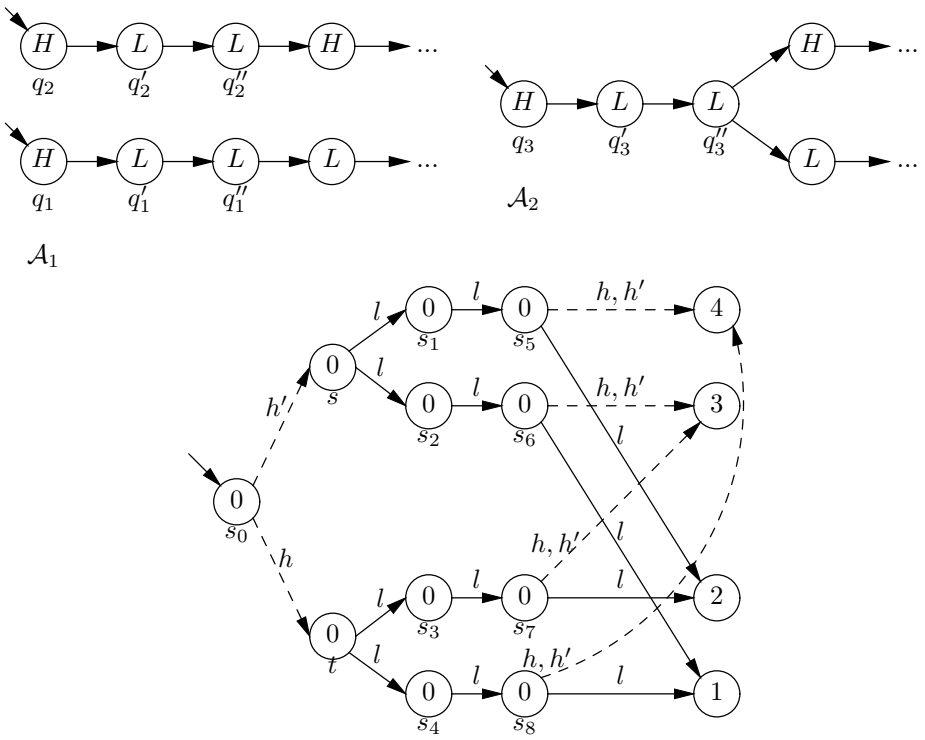

Figure 7: An example showing $\mathtt{tRES}_2^{\exists}(\Upsilon^{\mathtt{HO}}) \not\subseteq \mathtt{tRES}_2^{\forall}(\Upsilon^{\mathtt{HO}})$

- If $sched(q_{m+1}) = H$, we let $(s_1, q_{k+1}) \sim (s_4, q_{k+1})$ and $(s_2, q_{k+1}) \sim (s_3, q_{k+1})$.

The rest of the definition for $\sim$ is straightforward. This shows that $M \in \mathtt{tRES}_2^{\exists}(\Upsilon^{\mathtt{HO}})$, and $M \notin \mathtt{tRES}_1^{\forall}(\Upsilon^{\mathtt{HO}})$. □

However, it is unclear whether the inclusions $\mathtt{tRES}_2^{\forall}(\Upsilon^{\mathtt{HO}}) \subseteq \mathtt{tRES}_1^{\forall}(\Upsilon^{\mathtt{HO}})$ and $\mathtt{tRES}_2^{\exists}(\Upsilon^{\mathtt{HO}}) \subseteq \mathtt{tRES}_1^{\exists}(\Upsilon^{\mathtt{HO}})$ are strict. We leave these questions open for future research.

Results similar to Theorem 5.4 hold for the universal bisimulation-based properties, as shown in the following theorem. (The main reason is that due to Proposition 4.8 and Proposition 4.9, both $\mathtt{tRES}_1^{\forall}$ and $\mathtt{tRES}_2^{\forall}$ are preserved by refinement on schedulers. However, Proposition 4.8 and Proposition 4.9 do not hold for $\mathtt{tRES}_1^{\exists}$ or $\mathtt{tRES}_2^{\exists}$, as we have discussed in Section 4. It therefore is not clear whether a result similar to the following could be obtained for $\mathtt{tRES}_1^{\exists}$ or $\mathtt{tRES}_2^{\exists}$.)

Theorem 5.6. $\mathtt{tRES}_i^{\forall}(\Upsilon^{\mathtt{HO}}) = \mathtt{tRES}_i^{\forall}(\sigma_c)$ *for* $i = 1, 2$.

Proof: Since $\sigma_c \in \Upsilon^{\mathtt{HO}}$, it is immediate that $\mathtt{tRES}_i^{\forall}(\Upsilon^{\mathtt{HO}}) \subseteq \mathtt{tRES}_i^{\forall}(\sigma_c)$. To show the converse containment, suppose $M \in \mathtt{tRES}_i^{\forall}(\sigma_c)$. For any $\sigma \in \Upsilon^{\mathtt{HO}}$,

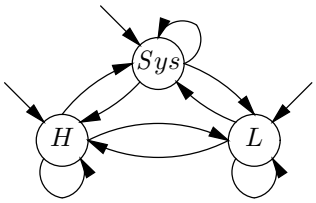

Figure 8: A scheduler SOLTS $\mathcal{A}_c$ that represents the chaos scheduler σ_c

we have $\sigma_c \sqsubseteq \sigma$, so by Proposition 4.8 and Proposition 4.9, we obtain that $M \in \mathtt{tRES}_i^{\forall}(\sigma)$. Thus $M \in \mathtt{tRES}_i^{\forall}(\Upsilon^{\mathtt{HO}})$. □

As above, this gives a characterization of $\mathtt{tRES}_i^{\forall}(\Upsilon^{\mathtt{HO}})$ that potentially simplifies verification. However, to check $\mathtt{tRES}_i^{\forall}(\sigma_c)$ it is still necessary to consider all scheduler SOLTS representing σ_c. We now give a sharper characterization that also eliminates this quantification. We show that the existence of a sensitive unwinding relation on machine M composed with a particular scheduler SOLTS is sufficient for the existence of a sensitive unwinding relation on $M \parallel \mathcal{A}$ for all H-oblivious scheduler implementations $\mathcal{A}$. Define $\mathcal{A}_c$ as the H-oblivious implementation of σ_c which is depicted in Figure 8. (All states are initial.) We may note that this is the most abstract scheduler SOLTS.

Lemma 5.7. *For all scheduler SOLTS $\mathcal{A}$, we have $\mathcal{A}_c \sqsubseteq_S \mathcal{A}$.*

Proof: It is easily checked that the relation $\leq$ on states q of $\mathcal{A}$ and q' of $\mathcal{A}_c$ defined by $q \leq q'$ if $sched(q) = sched(q')$ is a simulation. □

The following result says that in the case of the *sensitive* unwinding relation, security over all H-oblivious scheduler SOLTS (i.e., $\mathtt{tRES}_2^{\forall}(\Upsilon^{\mathtt{HO}})$) can be reduced to consideration of the single scheduler SOLTS $\mathcal{A}_c$.

Theorem 5.8. *For every machine M, we have $M \in \mathtt{tRES}_2^{\forall}(\Upsilon^{\mathtt{HO}})$ iff there exists a sensitive unwinding relation on $M \parallel \mathcal{A}_c$.*

Proof: Since $\mathcal{A}_c$ implements σ_c and $\sigma_c \in \Upsilon^{\mathtt{HO}}$, we only need to show that if there exists a sensitive unwinding relation on $M \parallel \mathcal{A}_c$, then $M \in \mathtt{tRES}_2^{\forall}(\Upsilon^{\mathtt{HO}})$. Let $\mathcal{A}$ be an H-oblivious scheduler SOLTS representing $\sigma \in \Upsilon^{\mathtt{HO}}$, then by Lemma 5.7, $\mathcal{A}_c \sqsubseteq_S \mathcal{A}$. Then since there is a sensitive unwinding relation on $M \parallel \mathcal{A}_c$, by Proposition 4.4, there also exists a sensitive unwinding relation on $M \parallel \mathcal{A}$. Therefore $M \in \mathtt{tRES}_2^{\forall}(\Upsilon^{\mathtt{HO}})$. □

As synchronous insensitive unwinding relations are not preserved by refinement on scheduler SOLTS, the argument in Theorem 5.8 does not help to show that $M \in \mathtt{tRES}_1^{\forall}(\Upsilon^{\mathtt{HO}})$ is equivalent to the existence of an insensitive unwinding on $M \parallel \mathcal{A}_c$. Consider the machine M and scheduler SOLTS $\mathcal{A}_2$ depicted in Figure 2. There exists an insensitive unwinding relation on $M \parallel \mathcal{A}_c$, but $M \notin \mathtt{tRES}_1^{\forall}(\Upsilon^{\mathtt{HO}})$ as $M \notin \mathtt{tRES}_1^{\forall}(\sigma_{\mathcal{A}_2})$.

We introduce the following example which is a multi-level file store shared between a high level agent H and a low level agent L. This example is adapted

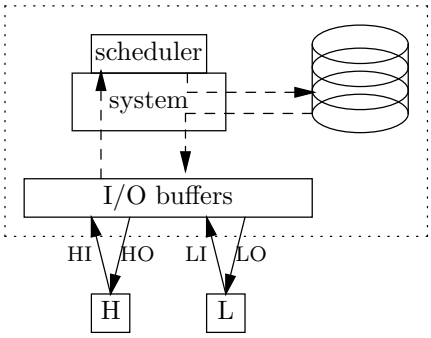

Figure 9: A file store device shared by H and L

and simplified from a case study in [Zha09, Chapter 6.2], which is motivated from a block device that uses a file server to support networks with multiple security levels [Gra08].

Example 5.9. *Here we assume there are only two agents H and L. The structure of the system is depicted in Fig. 9. The device contains a single processor with a hard disk for data-storage. We omit the processor in the figure because it does not have an effect on our analysis. We further assume there is only a single file f_H belonging to agent H and a single file f_L belonging to agent L, both of a fixed size. The communication between the system and agents is done via input buffers and output buffers. Either H or L can read and write its corresponding input buffer and can read but not write the contents of its output buffer. The system operates at multiple security levels and can both read and write these buffers. Apart from the above components, a scheduling algorithm is implemented that allocates time slices for reading and executing instructions. When an agent is scheduled, it may request to read or write a file by writing to its input buffer, and read from its output buffer written by the system. The system handles a request only when it is scheduled.*

In the model we assume a signature $(A, \{H, L, Sys\}, dom)$, where the set of actions $A = \bigcup_{u \in D \setminus \{Sys\}} I_u \cup A_{Sys}$, with I_u the instruction set of user u, and $A_{Sys} = \{\tau\}$. The passage of time is observable to the users, as they share a global clock with the system, and the finest time unit distinguishable by a user is a single slice (tick). An action of each agent, including τ, takes one tick.

We assume each file can be regarded as a string of binary numbers with length n. Given a user u, the set of instructions I_u includes $r_u(f, i)$: read file $f \in \{f_H, f_L\}$ at index $i \in \{1, 2, \ldots, n\}$, $w_u(f, x, i)$: write a bit x to file f at index i, and ϵ_u: a special input that denotes skipping (this could be viewed as an abstraction of any action that is local to an agent and not pertinent to the filestore). When the system is writing to a user's output buffer, it uses 'ok' to indicate access granted, 'deny' for access declined, '$\perp$' for the default empty *observation, or a binary number as the bit value being returned. A buffer may contain at most one message. We formally define the device $M = \langle S, S_0, \rightarrow, O, obs \rangle$ as follows.*

- $S = F_H \times F_L \times HI \times HO \times LI \times LO$ *is a set of states, and we sometimes write* $s(c)$ *for the component* $c \in \{F_H, F_L, HI, HO, LI, LO\}$ *in* s. *Here* $F_H = \{0,1\}^n$ *represents the state of file* f_H, $F_L = \{0,1\}^n$ *represents the state of file* f_L, $HI = \{\bot\} \cup \{w_H(f,i,x), r_H(f,i) \mid f \in \{f_L, f_H\}, i \in \{1..n\}, x \in \{0,1\}\}$ *represents the* H *input buffer,* $HO = \{0,1,\bot,ok,deny\}$ *represents the* H *output buffer, and similarly* $HI = \{\bot\} \cup \{w_L(f,i,x), r_L(f,i) \mid f \in \{f_L, f_H\}, i \in \{1..n\}, x \in \{0,1\}\}$ *and* $LO = \{0,1,\bot,ok,deny\}$ *represent the* L *input and output buffer, respectively. We write* $(f_H, f_L, h_i, h_o, l_i, l_o)$ *for a generic state.*
- $S_0 = \{(0^n, 0^n, \bot, \bot, \bot, \bot)\}$ *the set of initial states,*
- $O = (HI \cup LI) \times (HO \cup LO)$,
- *obs is defined by* $obs_H(s) = (s(HI), s(HO))$ *and* $obs_L(s) = (s(LI), s(LO))$,
- $\rightarrow \subseteq S \times A \times S$ *is a transition relation.*

Given a file f, *we write* $f[i \mapsto v]$ *to represent the result of updating the* i*-th bit of* f *to value* v. *Given a state* s, *we write* $s[c \mapsto v]$ *to represent the state in which the value of component* c *is updated to* v. *The transition relation is defined by the following rules.*

Actions of the agents H, L *are stored in the corresponding input buffer, except for the skip action, which has no effect on state:*

$s \xrightarrow{a_L} s[LI \mapsto a_L]$ *for* $a_L \in I_L$ *(act-L)*
$s \xrightarrow{a_H} s[HI \mapsto a_H]$ *for* $a_H \in I_H$ *(act-H)*
$s \xrightarrow{\epsilon_u} s$ *for* $u \in \{H, L\}$ *(skip)*

In order to satisfy the policy that information is only allowed to flow from L *to* H, *but not from* H *to* L, Sys *enforces a number of access rules. The following rules handle requests by* L.

$(f_H, f_L, h_i, h_o, r_L(f_L, i), l_o) \xrightarrow{\tau} (f_H, f_L, h_i, h_o, \bot, f_L(i))$ *(r-grt-LL)*
$(f_H, f_L, h_i, h_o, r_L(f_H, i), l_o) \xrightarrow{\tau} (f_H, f_L, h_i, h_o, \bot, deny)$ *(r-deny-LH)*
$(f_H, f_L, h_i, h_o, w_L(f_L, i, x), l_o) \xrightarrow{\tau} (f_H, f_L[i \mapsto x], h_i, h_o, \bot, ok)$ *(w-grt-LL)*
$(f_H, f_L, h_i, h_o, w_L(f_H, i, x), l_o) \xrightarrow{\tau} (f_H[i \mapsto x], f_L, h_i, h_o, \bot, ok)$ *(w-grt-LH)*

That is, if there is a pending read request from L *to access* f_L *when* Sys *is scheduled, the system will return with* $f_L(i)$, *and if it is requesting to read* f_H, *the system will decline the access. Requests by* L *to write either file are granted. After performing the action, the input buffer is cleared.*

The following rules handle requests by H. *Note that in each case, the starting state of the rule has* $l_i = \bot$, *so that there is no pending request by* L. *That is, the system prioritizes requests by* L *over those by* H.

$(f_H, f_L, r_H(f, i), h_o, \bot, l_o) \xrightarrow{\tau} (f_H, f_L, \bot, f(i), \bot, l_o)$ *for* $f \in \{f_H, f_L\}$ *(r-grt-H)*
$(f_H, f_L, w_H(f_H, i), h_o, \bot, l_o) \xrightarrow{\tau} (f_H[i \mapsto x], f_L, \bot, ok, \bot, l_o)$ *(w-grt-HH)*

$$(f_H, f_L, w_H(f_L, i), h_o, \bot, l_o) \xrightarrow{\tau} (f_H, f_L, \bot, deny, \bot, l_o) \qquad \textit{(w-deny-HL)}$$

These rules state that all read requests from H are granted, but only a requests to write to f_H from agent H is allowed. If there is no request from either H or L, there is no change to the state when Sys acts:

$$(f_H, f_L, \bot, h_o, \bot, l_o) \xrightarrow{\tau} (f_H, f_L, \bot, h_o, \bot, l_o) \qquad \textit{(sys-skip)}$$

This completes the presentation of the transition relation.

A real system sometimes implements complicated scheduling algorithms, which brings difficulty in verification. However, by Theorem 5.8, it suffices to find a sensitive unwinding relation in $M \parallel \mathcal{A}_c$ where $\mathcal{A}_c$ is a simple three state scheduler SOLTS representing the chaos scheduler σ_c, to ensure that M satisfies $\mathtt{tRES}_2^{\forall}(\sigma)$, the strongest property introduced in the paper, for all H-oblivious schedulers σ. (Note that it follows from this that we also have $M \in X(\sigma)$ for all trace-based properties X.)

Let the state space Q of $\mathcal{A}_c$ be $\{q_H, q_L, q_{Sys}\}$. On the combined state space $S \times Q$, define a relation $\sim$ by $(s_1, q_1) \sim (s_2, q_2)$ if $s_1(f_L) = s_2(f_L)$, $s_1(LI) = s_2(LI)$, $s_1(LO) = s_2(LO)$, and $q_1 = q_2$. We show $\sim$ is a synchronous sensitive unwinding relation.

- *For OC, we have $(s_1, q) \sim (s_2, q)$ implies $obs_L((s_1, q)) = obs_L((s_2, q))$ by $s_1(LO) = s_2(LO)$ and $s_1(LI) = s_2(LI)$,*
- *For SC, suppose $(s_1, q) \sim (s_2, q)$, and we have a transition $a \in I_L$, $(s_1, q) \xrightarrow{a} (s_1[LI \mapsto a], q')$. Then we have $sched(q) = L$. Therefore there exists $(s_2, q) \xrightarrow{a} (s_2[LI \mapsto a], q')$, such that $s_1[LI \mapsto a](LI) = s_2[LI \mapsto a](LI) = a$. Since the LO and f_L components are never changed by (act-L), we have $(s_1[LI \mapsto a], q') \sim (s_2[LI \mapsto a], q')$. In case of a (skip) transition $(s_1, q) \xrightarrow{\epsilon_L} (s_1, q')$, we can take the matching transition to be $(s_2, q) \xrightarrow{\epsilon_L} (s_2, q')$, and obviously $(s_1, q') \sim (s_2, q')$.*
- *For LR_{Sys}, suppose $(s_1, q) \sim (s_2, q)$ and $(s_1, q) \xrightarrow{\tau} (s_1', q')$. Then $sched(q) = Sys$. We have the following cases.*
  - *If there is no pending request from agent L in s_1, i.e., $s_1(LI) = \bot$, then also $s_2(LI) = \bot$. Thus, both from (s_1, q) and from (s_2, q) the system will execute a pending H request, if any, using one of the rules (r-grt-H), w-grt-HH), w-deny-HL), or (rule sys-skip). The same successor scheduler state q' is also available for the transition from (s_2, q). Note the transition rule for the transition $(s_2, q) \xrightarrow{\tau} (s_2', q')$ need not be the same as that for $(s_1, q) \xrightarrow{\tau} (s_1', q')$. However, a simple inspection of the rules (r-grt-H), (w-grt-HH), (w-deny-HL) and (rule sys-skip) shows that they do not change the components f_L, LI, LO, so in all cases we derive $(s_1', q') \sim (s_2', q')$ from $(s_1, q) \sim (s_2, q)$.*

– *If* $s_1(LI) \neq \bot$*, then the system prioritizes the* L *request, and* $s_1' = s_1[LI \mapsto \bot, LO \mapsto o]$ *for some* L *output* o*, by (*r-grt-LL*), (*r-deny-LH*), (*w-grt-LL*) and (*w-grt-LH*). Since* $(s_1, q) \sim (s_2, q)$*, we have* $s_2(LI) = s_1(LI)$ *or* $s_2(f_L) = s_1(f_L)$*. In particular, the request to be processed is the same in* (s_1, q) *and* (s_2, q)*, and the same transition rule will apply. Suppose* $s_1(LI)$ *is a read request, then either we have the access both granted and the same content returned from* $s_1(f_L)$ *and* $s_2(f_L)$ *(by (*r-grt-LL*)), or the access is declined from both* s_1 *and* s_2 *(by (*r-deny-LH*)). Suppose* $s_1(LI)$ *is a write request, then it is always granted with ok returned to* LO *(by (*w-grt-LL*) and (*w-grt-LH*)), and either the same part of* $s_1(f_L)$ *and* $s_2(f_L)$ *is modified, or the same part of* $s_1(f_H)$ *and* $s_2(f_H)$*. Therefore, in all above cases, there exists* $(s_2, q) \xrightarrow{\tau} (s_2', q')$*, with* $s_1(LI) = s_2(LI)$*,* $s_1(LO) = s_2(LO)$*, and* $s_1(f_L) = s_2(f_L)$*.*

- *The case of* LR_H *can be proved by easily checking that* H *actions never change the components* f_L*,* LI *and* LO *(rules (*act-H*) and (*skip*)).*

□

6. Related Work

Refinement has previously been considered in the literature on information flow security. The main difference with our work is that we consider the impact of schedulers on security. We consider systems consisting of a machine and scheduler, and focus on refinement of the scheduler in particular, rather than on refinement of the system as an undifferentiated whole.

We have considered two definitions of refinement, at the level of abstract schedulers σ and at the level of their concrete implementations by scheduler SOLTS. Both definitions are more or less standard, in that they treat refinement as reduction of nondeterminism. Our notion of refinement of abstract schedulers states that a refinement of σ reduces the set of possibilities for the next agent scheduled after a given sequence α of actions. This is related to trace refinement, as defined, e.g., in CSP [Ros97], since if $\sigma \sqsubseteq \sigma'$ then any trace of (M, σ') will be a trace of (M, σ). Note, however, that it is not equivalent to trace refinement of the combined system (M, σ), in that scheduler refinement cannot constrain *which* actions the scheduled agent is able to perform, whereas trace refinement does have this power. Our notion of refinement on scheduler SOLTS is a variant of *simulation* [Par81] which is well known to be a sound proof technique for trace refinement.

Our notion of scheduler in SOLTS differs from that used in process algebra, in which schedulers are used to resolve nondeterministic behaviours, e.g., by selecting a predefined label on actions. In this way, a process, if combined with scheduler, becomes a new process that refines the original process. Bisimulation relations are not preserved by this type of refinement. An interesting work by

Konstantinos et al. [CNP09] introduces a stronger type of bisimulation relation which requires bisimilarity to hold for all schedulers. This relation can be used to prove secrecy of security protocol even if an attacker controls the scheduler. That is, after a scheduler is introduced to refine the original system, the new refined system is secure provided the original system is secure by equivalence-based definitions of security.

Besides refinement of nondeterminism, there are notions of *action refinement* in the literature [vGG01], in which a refinement of a system can replace an abstract action by behaviours expressed in terms of new concrete actions. A recent work by Bossi et al. [BPR07] formalizes action refinement in the security process algebra SPA. They treat both trace-based and bisimulation based information flow properties [FG95] defined in SPA and conclude that the unwinding characterized persistent properties [FR06] are closed under action refinement. We have not studied this sense of refinement in this paper. *Data refinement* [dRE98], which operates at the level of state representation e.g. the implementation of an abstract set data type into a list based implementation, is also out of the scope of the present work, although simulation is also a valuable proof technique in this area.

It has frequently been noted in the literature that information flow security is not necessarily preserved by refinement of a system: this is known as the *refinement paradox* [Jac88, McL94]. Whether this is a problem depends on the role that nondeterminism plays in the system model, which can impact whether one expects that reduction of the nondeterminism during system design will be necessary. In case that nondeterminism arises from underspecification, one expects to move to more determinate specifications, so the refinement paradox is a genuine concern. However, if the nondeterministic behaviour is inherent in the system and is not under the control of any agent (e.g., represents deliberate randomization that is used to hide information), then reduction of nondeterminism is undesirable, and the refinement paradox is less problematic.

We note that our definitions address refinement concerns in a number of ways: first, we have allowed nondeterminism in the scheduler, but have stated definitions such as $\texttt{tNDI}_3$ and $\texttt{tNDS}_2$ that require system security for all schedules, which can be viewed as deterministic refinements of a given nondeterministic scheduler – such definitions are preserved under refinement of the nondeterminism in the scheduler. Similarly, the strategic definitions we have stated ($\texttt{tNDS}_1$ and $\texttt{tNDS}_2$) implicitly quantify over all deterministic H behaviours, so are also preserved under refinement of H behaviour. However, all our definitions retain the inherent nondeterminism in the system itself, so may not be preserved under refinement of this source of nondeterminism: this is a question that we have not considered in this paper.

As preservation of security under refinement has generally been considered to be desirable, a number of works have had the objective of developing refinement operators that preserve information flow properties. The details depend on the underlying systems and security property definitions. If the security definition is trace-based and refinement is the subset relation on traces, one approach is to construct a deterministic subsystem. Jacob [Jac88, Jac89] presents

a way to derive secure implementations from an insecure system S, by parallel composition with the most general deterministic process P satisfying the security property, i.e., $S \parallel P$ will be secure. Roscoe's determinism-based security notions [Ros95] are preserved with respect to CSP refinement, which yields a verification methodology using the FDR model checker [GGH+99]. Graham-Cumming and Sanders [GCS91] specify security as trace equivalence with respect to a given user, and refinement as a downwards simulation, using the Z-specification language.

Heisel et al. [HPS01, SHP02] define a confidentially preserving notion of refinement for a probabilistic version of CSP and also discuss the compositionality of that refinement operation. Mantel [Man01] presents two step-wise refinement operators on trace-based properties of event systems (ES). These two operations work by adding or deleting transitions to preserve the unwinding relations on state event systems (SES). A number of examples are given to show that this technique could be applied for all trace properties expressible by the basic security properties (BSP) [Man00]. Bossi et al. [BFPR03b] study refinement for persistent security properties [BFPR03a] in a CCS based system, where they define refinement as the reverse of the simulation relation (similar to our treatment on scheduler SOLTS). Similar to Mantel's approach, the refinement operators introduced in that paper involve deleting more transitions than the traditional refinement operator, in order to maintain observational equivalence to the low security level agent L. Alur et al. [AČZ06] present a very general framework for representing deducibility-based information flow properties. Their secrecy preserving refinement operator requires a strong condition that the implementation simulates its specification. These works all deal with automaton-like representations of systems. Program-like representations have also been considered: [Mor06] defines a refinement relation for a simple sequential calculus that preserves *ignorance* (as expressed in a logic of knowledge [FHMV95]).

We have not studied compositionality or contextuality of our security properties, but it would be interesting to pursue this. Compositional security or related notions of security on automaton and trace theoretic models have been discussed in the prior literature [McC87, McC90].

The complexity of verifying several information flow security properties closely related to those of the present paper in finite state synchronous systems has been studied in [CvdMZ10]. A slightly different notion of scheduled system is adopted in that work, but but based on the results we would conjecture that the unwinding-based properties of the present paper are tractable (in PTIME), but the `tNDI` and `tNDS` variants are intractable, being complete in PSPACE and EXPSPACE, respectively. (It remains to be shown that these complexities hold for the exact model and the multiple variants of the security notions of the present paper.) This resembles the results in the literature, as for asynchronous sytems, verification of trace-based information flow properties (such as NDI [vdMZ07] and NDC [FG95]) are PSPACE-complete, but the verification of unwinding based properties or persistent bisimulation based properties usually can be performed in PTIME [FPR02, vdMZ07].

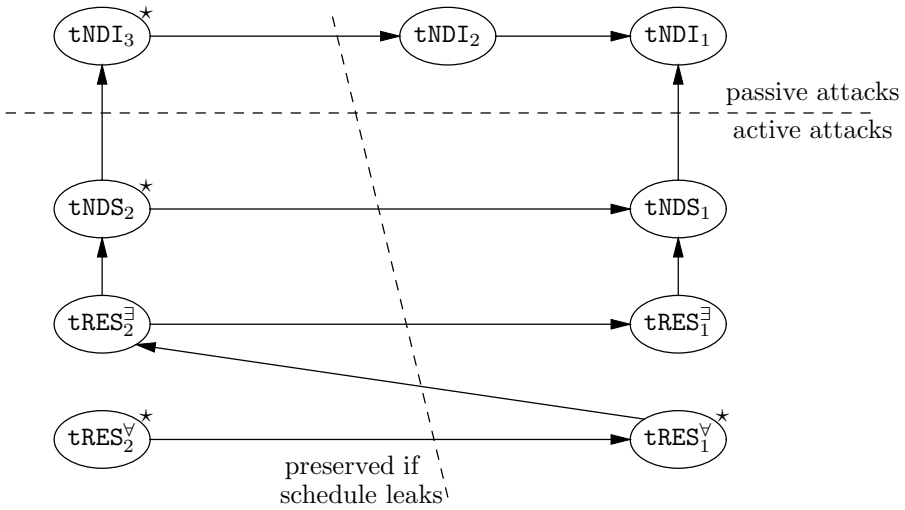

Figure 10: The relationship between synchronous scheduler-based security properties, where the properties marked by $\star$ are preserved under refinement of schedulers.

7. Conclusion

In this paper we have considered information flow security properties on a synchronous state machine model with a discrete-time semantics and a definition of scheduler. The security properties extend nondeducibility [Sut86], nondeducibility on strategies [WJ90] and restrictiveness [McC88] in a synchronous setting with scheduling. We have studied the preservation of these security properties under refinement of schedulers. The relationship between the properties are shown in Figure 10, where we mark by $\star$ the notions that are preserved by refinement of schedulers.

To summarize, we find that for trace-based properties, only the stronger ones, i.e., $\mathtt{tNDI}_3$ and $\mathtt{tNDS}_2$ are preserved by H-oblivious scheduler refinement, while the other trace-based properties are not preserved by scheduler refinement. For bisimulation-based properties, we have shown that $\mathtt{tRES}_1^\forall$ and $\mathtt{tRES}_2^\forall$ are preserved by scheduler refinement, while $\mathtt{tRES}_1^\exists$ and $\mathtt{tRES}_2^\exists$ are not. We also defined a refinement notion for scheduler SOLTS. This refinement relation preserves sensitive unwinding relations, but does not preserve insensitive unwinding relations.

Finally, we have reduced the verification of a number of security properties that quantify over all H-oblivious schedulers (including all trace-based properties with respect to $\Upsilon^{\mathtt{HO}}$, $\mathtt{tRES}_1^\forall(\Upsilon^{\mathtt{HO}})$ and $\mathtt{tRES}_2^\forall(\Upsilon^{\mathtt{HO}})$) to security properties that refer just to a single scheduler. In particular for $\mathtt{tRES}_2^\forall(\Upsilon^{\mathtt{HO}})$, we reduce its verification to finding sensitive unwinding relations when combining with a particular scheduler SOLTS. This allows us to characterize the relationships between security properties with respect to the set of schedulers $\Upsilon^{\mathtt{HO}}$. The results are depicted in Figure 11, where the solid arrows denote strict inclusion and

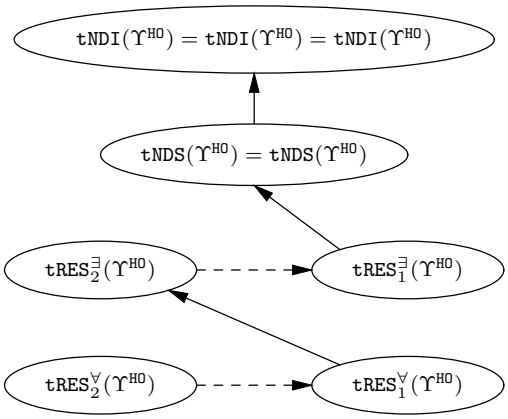

Figure 11: The relationship between synchronous scheduler-based security properties on $\Upsilon^{\mathtt{HO}}$

dashed arrows denote inclusion with their strictness left open.

The positive results of this paper have the following application. Where we have shown that a notion of security is preserved under scheduler refinement, we obtain a desirable level of flexibility in the choice of scheduler when configuring a system: any refinement of a scheduler for which a machine is known to be secure can be used in place of that scheduler, while maintaining security. Moreover, Theorem 5.4 and Theorem 5.8 show that to prove security with respect to the class of all H-oblivious schedulers, for all trace-based security definitions and for the bisimulation-based security definition $\mathtt{tRES}_2^{\forall}$, it suffices to verifying security with respect to the special scheduler σ_c (which has a finite state representation).

However, we have left open the question of how, practically, to show that a particular scheduled machine satisfies a security property. Automated verification of these properties would be desirable, if possible.

Acknowledgments: Work supported by Australian Research Council Discovery grant DP1097203. Thanks to Kai Engelhardt for helpful comments on a version of this paper.

References

[AČZ06] R. Alur, P. Černý, and S. Zdancewic. Preserving secrecy under refinement. In *Proc. International Colloquium on Automata, Languages and Programming (ICALP'06)*, 2006.

[BFPR03a] A. Bossi, R. Focardi, C. Piazza, and S. Rossi. Bisimulation and unwinding for verifying possibilistic security properties. In *Proc. International Conference on Verication, Model Checking, and Abstract Interpretation*, 2003.

[BFPR03b] A. Bossi, R. Focardi, C. Piazza, and S. Rossi. Refinement operators and information flow security. In *Proc. the First International Con-*

ference on Software Engineering and Formal Methods, SEFM'03, 2003.

[BPR07] A. Bossi, C. Piazza, and S. Rossi. Action refinement in process algebra and security issues. In *Proc. International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR)*, pages 201–217, 2007.

[CNP09] K. Chatzikokolakis, Gethin Norman, and David Parker. Bisimulation for demonic schedulers. In *Proc. Foundations of Software Science and Computational Structures (FOSSACS)*, pages 318–332, 2009.

[CvdMZ10] F. Cassez, R. van der Meyden, and C. Zhang. The complexity of synchronous notions of information flow security. In *Proc. Foundations of Software Science and Computational Structures (FOSSACS)*, pages 282–296, 2010.

[dRE98] Willem P. de Roever and Kai Engelhardt. *Data Refinement: Model-oriented Proof Theories and their Comparison*, volume 46 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1998.

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

[FHMV95] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning About Knowledge*. MIT press, 1995.

[FPR02] R. Focardi, C. Piazza, and S. Rossi. Proofs methods for bisimulation based information flow security. In *Verification, Model Checking, and Abstract Interpretation (VMCAI)*, pages 16–31, 2002.

[FR06] R. Focardi and S. Rossi. Information flow security in dynamic contexts. *Journal of Computer Security*, pages 65–110, 2006.

[GCS91] J. Graham-Cumming and J.W. Sanders. On the refinement of non-interference. In *Proc. IEEE Computer Security Foundations Workshop*, pages 35–42, 1991.

[GGH+99] P. Gardiner, M. Goldsmith, J. Hulance, D. Jackson, A. W. Roscoe, B. Scattergood, and P. Armstrong. *Failure Divergence Refinement - FDR2 User Manual version 2.64*. Formal Systems (Europe) Ltd, Aug 1999.

[GM84] J.A. Goguen and J. Meseguer. Unwinding and inference control. In *Proc. IEEE Symposium on Security and Privacy*, 1984.

[GM93] Paul H. B. Gardiner and Carroll C. Morgan. A single complete rule for data refinement. *Formal Aspects of Computing*, 5(4):367–382, 1993.

[Gra08] P. Graunke. Verified safety and information flow of a block device. In *Proceedings 3rd International Workshop on Systems Software Verification (SSV'08)*, Feb 2008.

[HHS87] C. A. R. Hoare, Jifeng He, and Jeff W. Sanders. Prespecification in data refinement. *Inf. Process. Lett.*, 25(2):71–76, 1987.

[HPS01] M. Heisel, A. Pfitzmann, and T. Santen. Confidentiality-preserving refinement. In *Proc. the 14th IEEE workshop on Computer Security Foundations*, pages 295–305, 2001.

[Jac88] Jeremy Jacob. Security specifications. In *Proc. IEEE Symposium on Security and Privacy*, pages 14–23, 1988.

[Jac89] J. Jacob. On the derivation of secure components. In *Proc. IEEE Symposium on Security and Privacy*, pages 242–247, 1989.

[Man00] Heiko Mantel. Possiblistic definitions of security – an assembly kit. In *Proc. IEEE Computer Security Foundation Workshop*, pages 185–199, July 2000.

[Man01] H. Mantel. Preserving information flow properties under refinement. In *Proc. the IEEE Symposium on Secrecy and Privacy*, pages 78–91, 2001.

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

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

[McC90] Daryl McCullough. A hookup theorem for multi-level security. *IEEE Transactions on Software Engineering*, 16(6):563–568, 1990.

[McL94] John McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *Proc. IEEE Symposium on Security and Privacy*, pages 79–93, May 1994.

[Mor06] C. C. Morgan. The shadow knows: Refinement of ignorance in sequential programs. In *Proc. Mathematics of Program Construction (MPC)*, pages 359–378, 2006.

[Par81] D. M. R. Park. Concurrency and automata on infinite sequences. In *Proc. Theoretical Computer Science*, volume 104 of *LNCS*, pages 167–183. Springer, 1981.

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

[Ros97] A. W. Roscoe. *The Theory and Practice of CSP*. Prentice-Hall (Pearson), 1997.

[SHP02] T. Santen, M. Heisel, and A. Pfitzmann. Confidentiality-preserving refinement is compositional - sometimes. In *Proc. the 7th European Symposium on Research in Computer Security (ESORICS'02)*, pages 194–211, London, UK, 2002. Springer-Verlag.

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

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

[vdMZ12] R. van der Meyden and C. Zhang. Information flow in systems with schedulers (part i: Definitions). *Theoretical Computer Science*, 2012.

[vGG01] Rob J. van Glabbeek and Ursula Goltz. Refinement of actions and equivalence notions for concurrent systems. *Acta Inf.*, 37(4/5):229–327, 2001.

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

[Zha09] C. Zhang. *Information Flow Security — models, verification and schedulers*. PhD thesis, UNSW, 2009.