A Note on Noninterference in the Presence of Colluding Adversaries
(Preliminary Report)

Kai Engelhardt

School of Computer Science and Engineering, The University of New South Wales, and NICTA*, Sydney, NSW 2052, Australia, kaie@cse.unsw.edu.au

Abstract. Whether adversaries can glean information from a distributed system in a formal sense hinges on the definition of such a system and what can be observed by those agents. In the presence of colluding adversaries, the standard definition of noninterference by Goguen and Meseguer and its many variants proposed in the literature fail in a very intuitive sense to capture a simple collusion attack. The crucial difference between what is modelled in those definitions and what we argue needs to be modelled is that teams can observe pomsets as Plotkin and Pratt stated. In this note we expose what goes wrong in the known approaches and explain how to fix the problem.

1 Introduction

As an introduction to the problem, consider the following scenario which will be our running example.1

Running Example. The secret agency $X$ consists of spies $A$ and $B$. Another secret agency $Z$ tasks its spies $D$ and $E$ to extract some information from $X$. It is well-known tradition in $X$’s country of origin to drink amnesia-inducing herbal tea. This tradition is particularly popular amongst secret agents. It is against the tradition to drink tea alone.

Spies $D$ and $E$ hatch a plan to capture $A$ or $B$ after work, but only if they did not drink tea on their way out.

* NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

1 We mark the end of each partial discussion of the running example with the symbol ♥. As one of the anonymous referees pointed out:

“The paper would benefit from an example from a realistic computer program rather than just a story about spies and amnesia-inducing beverages. For example, a program that branches on a high value and either forks two processes which write $x$ and $y$ to low locations, or sequentially writes $x$ and $y$ to two low locations.”

We intend to present such an example in the full version of the paper.
Fig. 1. The floor plan of agency X with the four spies involved. Spies D and E concurrently observe spies A and B.

From a leaked floor plan of X, which we reproduced in Fig. 1, spies D and E gather that, since the tea room has a fairly narrow door, A and B would have to leave the tea room one after the other. In contrast to the tea room, the remainder of the floor plan is fairly generous, allowing A and B to enter and leave next to each other.

To leave the building, A uses the left door and B uses the right one. Since the hallway is wide enough and they are equally fast, they may do so at the same time unless they visited the tea room on the way out, which forces them to exit one after the other. Spies D and E watch the doors from opposite sides.

The system models routinely used in discussions of noninterference are based on sequences, i.e., total orders of events [3, 4, 10, 9]. (See [13] for an overview.)

**Running Example.** These choices of models imply that spies D and E necessarily observe A and B leaving the building in the same order even if they did not have tea. In a richer system model based on partial orders of events, where individual observers observe one of the possible linearizations of the partial order, spies D and E could observe A and B leaving the building in contradicting orders. Pooling their observations, D and E could deduce that A and B did not visit the tea room, allowing them to pounce on A and B with their memories intact.

1.1 Noninterference

In 1982 Goguen and Meseguer defined

“one group of users, using a certain set of commands, is noninterfering with another group of users if what the first group does with those commands has no effect on what the second group of users can see.”

[3, p. 11]
1.2 Partial Orders

Partial orders surface in many a place in computer science, for instance, in formal semantics and in crucial optimizations of model checkers. The documentation of modern CPUs such as those in Intel’s x86 family describes the to be expected effects of executing instructions. For performance reasons, load and store operations can not necessarily be relied on being performed in their specified order. In multicore and multiprocessor configurations the effects of out-of-order execution of the pipelined instructions in individual cores is exacerbated by weak ordering constraints on loads and stores by different cores. The documentation merely states lower bounds on the amount of order, resulting in partial orders of load and store instructions even weaker than those induced by the concurrent instruction streams executed on the cores/processors.

“In a multiple-processor system, the following ordering principles apply:

- Individual processors use the same ordering principles as in a single-processor system.
- Writes by a single processor are observed in the same order by all processors.
- Writes from an individual processor are NOT ordered with respect to the writes from other processors.
- Memory ordering obeys causality (memory ordering respects transitive visibility).
- Any two stores are seen in a consistent order by processors other than those performing the stores” [6, p. 8-10]

[...] The memory-ordering model allows concurrent stores by two processors to be seen in different orders by those two processors; specifically, each processor may perceive its own store occurring before that of the other.” [6, p. 8-15]

Pratt demonstrated the relevance of partial orders as fundamental models of concurrent behaviour in [8]. The uptake of his arguments has been less than enthusiastic so far. Back in 1986, Pratt listed three reasons for this lack of popularity:

(i) formal language constructs for words (i.e., totally ordered multisets of letters) match their programming language counterparts and are well-understood,
(ii) partial orders can be represented as the sets of their linearizations, (iii) only physicists need to worry about non-rigid, relativistic models of the world, whereas engineers happily idealize physical reality to linear orders of instantaneous events.

In [7] Plotkin and Pratt explored under which circumstances point (iii) above is justified. For us, their most relevant result is that, if there are multiple observers and if events have duration then those observers may collaboratively observe partial order as distinct from any linear order. They also considered variability of events, by which they refer to settings in which for an event \( e \) to occur either \( e_1 \) or \( e_2 \) needs to occur.

On the topic of models traditionally used to study noninterference, Goguen and Meseguer state that
“Most of the models given in the literature […] do not support a sufficiently
general view of security (for example, they may fail to handle breaches of se-
curity by cooperating multiple users)”

[3, p. 12]

which is precisely our concern in this note. To illustrate our point, it suffices to
consider a fairly basic setting. Next we motivate the two crucial ingredients that
distinguish our setting from those to which the more standard settings apply.

**Where do we encounter multiple observers?** Multiple observers are a given
in common settings for investigating noninterference. Assuming a singleton ad-
versary is implicit in some models but hardly ever justifiable. In a more concrete
setting, multiple threads running on some cores of a multicore CPU can observe
aspects of how other threads running on their and other cores interact with the
shared memory architecture.

*Running Example.* Spies D and E observe concurrently.

**When do events have duration?** Even if, at the abstract level of a noninter-
ference proof, events do not have duration, they typically gain duration in later
refinements of the system under consideration during so-called *action refinement*
steps. Such refinements commonly take place between high level instructions in
some programming language and the sequences of machine instructions as exe-
cuted by the stages of a pipeline in a CPU core.

*Running Example.* The event “spy A leaves through the left door” could end up
being refined into the sequence “A opens the left door”; “A passes through the
door”; “A closes the door”.

### 1.3 Formal Modeling in PVS

All of the theory and the example presented in this note have been encoded in
PVS [11]. At the time of writing, some proofs are unfinished (in PVS only) due
to some idiosyncrasies, presumably of the author. A PVS dump of the develop-
ment is available for download at [http://www.cse.unsw.edu.au/~kaie/pvs/

### 2 From Trace-Based Noninterference to Partial
Order-Based Noninterference

This section presents a simple generalization of a common choice for modeling
systems for noninterference purposes to accommodate the running example. In
the terminology of [12], the model is *action-observed.*

The presentation and content of the more common parts of this section are
inspired by Rushby [9] and van der Meyden [12]. Let \( D \) be a set of *security*
domains, e.g., \{H, L\} for high and low. A security policy is a reflexive interference relation \( \rhd \subseteq \mathbb{D}^2 \) which is often defined by describing just the missing edges, e.g., \( H \not\rhd L \) for \( \{H \rhd H, L \rhd H, L \rhd L\} \), the policy prohibiting information flow from \( H \) to \( L \).

A system \( S = (\Sigma, s_0, A, O, \tau, o, \delta) \) consists of a set \( \Sigma \) of states, an initial state \( s_0 \in \Sigma \), a set \( A \) of actions, a set \( O \) of observations, a transition function \( \tau : A \rightarrow \Sigma \rightarrow \Sigma \), an observation function \( o : A \rightarrow \Sigma \rightarrow O \), and a domain function \( \delta : A \rightarrow \mathbb{D} \). We often use subscripts for a more compact representation of the state transition functions and observations induced by actions, e.g., \( \tau_a \) for \( \tau(a) : \Sigma \rightarrow \Sigma \).

**Running Example.** One possible representation of our running example scenario as a system is the following, which we call \( S_0 \).

- States of the system reflect where spies \( A \) and \( B \) are, and the contents of their heads, which we abstract to being empty, containing a secret, or remembering whether they were the first to leave through their door.

\[
\Sigma = \{\text{in, out}\}^2 \times \{0, 1, \text{AthenB, BthenA}\}.
\]

- Initially, both spies are inside \( X \) and know some secrets:

\[
s_0 = (\text{in, in}, 1).
\]

- Actions are what spies \( A \) and \( B \) might do, that is, either work, drink tea, or leave. Only leaving is an individual activity—the other two are executed jointly:

\[
A = \{\text{work, tea, leave}_A, \text{leave}_B\}.
\]

- Observers such as spies \( D \) and \( E \) observe the locations of \( A \) and \( B \):

\[
O = \{\text{in, out}\}^2.
\]

- To describe the transition function we let \( \ell \) and \( \ell' \) range over \( \{\text{in, out}\} \), and we let \( s \) range over \( \{0, 1, \text{AthenB, BthenA}\} \). Working places the spies inside \( X \) (if necessary) and installs the secret (“1”) in their heads.

\[
\tau_{\text{work}}(\ell, \ell', s) = (\text{in, in}, 1)
\]

Drinking tea only has an effect if both spies are inside. Then it erases any information in their heads.

\[
\tau_{\text{tea}}(\ell, \ell', s) = \begin{cases} (\ell, \ell', 0), & \text{if } \ell = \ell' = \text{in} \\ (\ell, \ell', s), & \text{otherwise} \end{cases}
\]
Leaving takes a spy outside $X$ and, if he is the first to leave and his brain is empty, then he remembers that he left first. Otherwise, if the other spy is outside already, their brains remain unchanged.

\[
\tau_{\text{leave}}(\ell, \ell', s) = \begin{cases} 
\text{out}, \ell, A\text{then}B, & \text{if } \ell = \ell' = \text{in} \text{ and } s = 0 \\
\text{out}, \ell, & \text{otherwise}
\end{cases}
\]

\[
\tau_{\text{leave}}(\ell, \ell', s) = \begin{cases} 
(\ell, \text{out}, B\text{then}A), & \text{if } \ell = \ell' = \text{in} \text{ and } s = 0 \\
(\ell, \text{out}, s), & \text{otherwise}
\end{cases}
\]

- The observation function is given by $o_a(s) = (\Pi_1(\tau_a(s)), \Pi_2(\tau_a(s)))$, where $\Pi_i$ denotes the $i$'th projection. In other words, the locations of $A$ and $B$ after $a$ has been performed are observable.
- All actions are in the $L$ domain except for those representing $A$’s and $B$’s drinking of tea.

\[
\delta(a_x) = \begin{cases} 
H & \text{if } a \text{ is tea} \\
L & \text{otherwise}
\end{cases}
\]

To compare the traditional trace-based view to our new partial-order-based view, we present both. The purpose of this presentation is to demonstrate that a system considered secure in the trace-based view could be insecure in the partial-order-based view.

Since it suffices to make our point and reduces clutter, let us restrict our attention to the perhaps best-studied interference relation, namely $H \not\Rightarrow L$. This simplification allows us to omit the qualification “for $\Rightarrow$” from our definitions and to ignore many subtleties including those related to intransitivity of $\Rightarrow$. (For recent progress on that topic see [12].)

### 2.1 Noninterference in a Trace-Based Execution Model

We use $\epsilon$ for empty sequences and $\cdot$ to denote concatenation. Let $\text{purge}_L : A^* \rightarrow A^*$ be the function that purges all $H$-labelled actions

\[
\text{purge}_L(\epsilon) = \epsilon
\]

\[
\text{purge}_L(\alpha \cdot a) = \begin{cases} 
\text{purge}_L(\alpha), & \text{if } \delta(a) = H \\
\text{purge}_L(\alpha) \cdot a, & \text{otherwise}
\end{cases}
\]

and consider $\tau$ lifted to sequences of actions.

\[
\tau_\epsilon(s) = s \\
\tau_{\alpha \cdot a}(s) = \tau_a(\tau_\alpha(s))
\]

**Definition 1.** System $S = (\Sigma, s_0, A, O, \tau, o, \delta)$ is secure if

\[
o_a(\tau_\alpha(s_0)) = o_a(\tau_{\text{purge}_L(\alpha)}(s_0))
\]

for all $a \in A$ with $\delta(a) = L$, and all $\alpha \in A^*$.
Running Example. System $S_0$ is secure.

Only a small change is needed to allow the spies $D$ and $E$ to observe the order in which $A$ and $B$ leave the building $X$. Let us define a minor variation $S_1$ of $S_0$. This new system is almost the same as $S_0$ but uses a slightly bigger set of observations,

$$O' = O \cup \{\text{AthenB}, \text{BthenA}\},$$

to encode the order in which $A$ and $B$ leave once they are both outside. We define the observation function $o'$ to be the same as $o$ except for two cases:

$$
\begin{align*}
o'_{\text{leave}_A}(\text{in}, \text{out}, s) &= \text{BthenA} \\
o'_{\text{leave}_B}(\text{out}, \text{in}, s) &= \text{AthenB}
\end{align*}
$$

Note that $D$ and $E$ observe the order when the second spy leaves and they do so regardless of whether $A$ and $B$ remember the order. We check that also $S_1$ is secure.

This is not too surprising because the observations still do not capture much information about the contents of the spies’ heads.

2.2 Definitions of Noninterference Based on Partial Order Execution Models

In order to formalize partial orders and related structures, we follow Plotkin and Pratt’s presentation by defining a labelled partial order (or lpo) over a set of actions $A$ to be a structure $(V, \leq, \sigma, A)$ where $\leq$ partially orders $2^V$ and $\sigma : V \rightarrow A$ assigns an action to each node in $V$. We write $<$ for the strict version $\leq \setminus = \bot$ of the order.$^2$ Two nodes concur in $p$ if they are unrelated by $\leq$. By $\text{frontier}(p)$ we refer to the set of $\leq$-maximal elements of $p$. A linearization of an lpo $p = (V, \leq, \sigma, A)$ is an lpo $(V, \leq', \sigma, A)$ such that $\leq' \supseteq \leq$ is a total order, that is, there are no concurrent nodes. Let $\text{linearizations}(p)$ denote the set of all linearizations of $p$. We often identify a linearization with the corresponding trace in $A^*$. We say that two nodes $v, w \in V$ are neighbours if they are different and if there is no node $z \in V$ satisfying either of $v < z < w$ and $w < z < v$.

Two actions may concur in a partially ordered execution of a system if their state changes commute. Given a state $s$ we say that two actions $a, b \in A$ are non-conflicting in $s$ and write $a \parallel_s b$ if the state changes they induce when started in state $s$ commute:

$$a \parallel_s b = \tau_{a,b}(s) = \tau_{b,a}(s).$$

$^2$ Recall that $\leq$ partially orders $V$ if $\leq$ is reflexive, transitive, and antisymmetric.

$^3$ In theory, we ought to abstract from the elements of $V$ and consider partially ordered multisets of actions as isomorphism classes of lpos, where the isomorphisms allow us to exchange the partially ordered node set $(V, \leq)$ for something similar. In other words, we do not care about the names of nodes in lpos but about their labels. For this note, it suffices to work with the less abstract notion of lpos.
The definition of $\parallel$ is slightly technical because we opt for a relatively weak version of this restriction in order to allow actions to concur in certain states even if they are in conflict elsewhere.

**Definition 2.** Let $S = (\Sigma, s_0, A, O, \tau, o, \delta)$ be a system and $p = (V, \leq, \sigma, A)$ be an lpo. We call $p$ a partially ordered execution (or poex for short) of $S$ if only non-conflicting pairs of actions concur in $p$. We write $\text{Poex}(S)$ for the set of poexes of $S$. More formally, $p \in \text{Poex}(S)$ if, for all concurrent $v, w \in V$, for all linearizations $p' = (V, \leq', \sigma, A)$ of $p$ in which $v$ and $w$ are neighbours,

$$\sigma(v) \parallel \tau_{\sigma}(v) \sigma(w),$$

where $\alpha$ is the $\leq'$-ordered sequence of elements of $V$ that are strictly smaller than both $v$ and $w$.

This particular modeling choice simplifies the calculation of the resulting state since any linearization of such a poex will generate the same final state.

**Running Example.** Two examples of poexes are $p$ and $q$ given by

$$p = \begin{array}{c}
\text{work} \leftarrow \text{leave}_A \\
\text{leave}_B
\end{array}$$

$$q = \begin{array}{c}
\text{work} \rightarrow \text{tea} \\
\text{leave}_A \\
\text{leave}_B
\end{array}$$

where an arrow from $a$ to $b$ expresses that $a \leq b$. We tend to omit arrows implied by reflexivity and transitivity, e.g., the one from tea to leave$_B$ in $q$ above. We have that $\text{frontier}(p)$ consists of the nodes labelled with leave$_A$ and leave$_B$. It has two linearizations, $\text{work} \rightarrow \text{leave}_A \rightarrow \text{leave}_B$ and $\text{work} \rightarrow \text{leave}_B \rightarrow \text{leave}_A$. The lpo $q$ is a total order and hence has a singleton frontier and a singleton linearizations($q$) = $\{q\}$.

The lpo $p'$ depicted below is a linearization of $p$ and thus also a poex of $S$. The lpo $q'$ is not in $\text{Poex}(S)$ since it does not order $\text{leave}_A$ and $\text{leave}_B$ even though these actions occur in a context that does not allow them to commute.

$$p' = \begin{array}{c}
\text{work} \leftarrow \text{leave}_A \\
\downarrow \\
\text{leave}_B
\end{array}$$

$$q' = \begin{array}{c}
\text{work} \rightarrow \text{tea} \\
\downarrow \\
\text{leave}_A \\
\text{leave}_B
\end{array}$$

They do not commute because

$$\tau_{\text{work tea leave}_A \text{leave}_B}(s_0) = (\text{out, out, AthenB})$$

$$\neq (\text{out, out, BthenA}) = \tau_{\text{work tea leave}_B \text{leave}_A}(s_0).$$

By the definition of poex, all linearizations of a poex are also poexes and induce the same state change.

**Lemma 3.** Let $S = (\Sigma, s_0, A, O, \tau, o, \delta)$ be a system, $p \in \text{Poex}(S)$, and $\alpha, \beta \in \text{linearizations}(p)$. Then $\tau_{\alpha}(s_0) = \tau_{\beta}(s_0)$.
Before we can give a poex-based definition of security, the purge and observation functions need to be adapted to partial orders. Purging on lpos amounts to restricting everything to the set of nodes mapped to L:

$$\text{purge}_L((V, \leq, \sigma, \Sigma)) = (V', \leq \cap (V')^2, \sigma', \Sigma),$$

where $V' = \{ v \in V \mid \delta_v = L \}$ and $\sigma'$ is the restriction of $\sigma$ to $V'$.

**Running Example.** Since purging preserves the order induced by purged actions $\text{purge}_L(q) = p'$. Note that there is no poex $p''$ satisfying $\text{purge}_L(p'') = p$ in which both $A$ and $B$ drink tea right before leaving.

In general, purging may turn a poex of $S$ into an lpo that is not a poex of $S$. To simplify the presentation, we restrict our attention to systems where this is not an issue. Neither $S_0$ nor $S_1$ have this issue.

Adapting the observation function to lpos is slightly more involved since we do not have a variant of Lemma 3 for observations instead of states. Poexes do not have singleton frontiers in general. Since we do not want to change the model more than necessary, we continue assuming that individual observers have a sequential view of the world, just as in the trace-based model. Observers, when concurrently observing a poex, may witness different serialization of the same poex. By pooling the observers’ information gleaned form a single poex, we encounter sets of observations.

**Definition 4.** Given a system $S = (\Sigma, \sigma_0, A, O, \tau, o, \delta)$ the possible observations of a poex $p = (V, \leq, \sigma, \Sigma)$ are collected in the set

$$o_p(s_0) = \{ o_{\alpha}(\tau_{\alpha}(s_0)) \mid \alpha \cdot a \in \text{linearizations}(p) \}.$$

System $S$ is p-secure if, for all $p = (V, \leq, \sigma, \Sigma) \in \text{Poex}(S)$ satisfying $\delta_{\sigma(v)} = L$ for all $v \in \text{frontier}(p)$,

$$o_p(s_0) = o_{\text{purge}_L(p)}(s_0).$$

**Running Example.** Systems $S_0$ and $S_1$ are also p-secure.

This means that the notion of p-security fails to detect the problem despite our efforts to adapt the model such that it is detected! What went wrong? We were implicitly assuming that what works for traces must work for poexes, when it does not. We simply compared the observations of arbitrary executions (i.e., with $H$ and $L$ actions) to those obtained by purging the $H$ actions, and assumed that we would achieve full coverage as if we had started with executions consisting of low actions and padded them with arbitrary $H$ actions. In the partial order world, this is no longer the case.

This motivates yet another definition of security, this time comparing the (sets of) observations made on an execution consisting of $L$ actions only with the observations made on any extension of that execution with arbitrary $H$ actions. Before we can give that definition we need to formalize the augmentation of a given poex with $H$ actions.

---

4 Pooling information here is a sloppy reference to what is otherwise known as acquiring *distributed knowledge* in the terminology of [5].
Definition 5. Let $p, q \in \text{Poex}(S)$ such that $p = (V, \leq, \sigma, A)$ satisfies $\delta_{\sigma(v)} = \mathbb{L}$ for all $v \in V$. We call $q$ a H-augment of $p$ if there exist a set of fresh nodes $V'$, a binary relation $\leq' \subseteq (V \cup V')^2$, and an assignment $\sigma' : V' \rightarrow \Delta$ of actions to the fresh nodes such that

- $q = (V \cup V', \leq \cup \leq', \sigma \cup \sigma', A)$,
- $\delta_{\sigma'(v)} = \mathbb{H}$ for all $v \in V'$,
- $\text{frontier}(q) \subseteq V$, and
- $\leq'$ is minimal on the nodes of $p$, i.e., removing any part from $\leq' \cap V^2$ would render poex-ness unattainable. More formally, we require that

$$(V \cup V', \leq \cup \leq'', \sigma \cup \sigma', A) \notin \text{Poex}(S)$$

for all $\leq'' \subseteq (V \cup V')^2$ satisfying $(\leq'' \cap V^2) \subsetneq (\leq \cap V^2)$.

Running Example. Returning to our example poexes we note that $q$ is a H-augment of $p'$, but more importantly, $q$ is also a H-augment of $p$.

Finally, we are prepared for our last definition of security.

Definition 6 (poex-security). System $S$ is poex-secure if, for all $p = (V, \leq, \sigma, A) \in \text{Poex}(S)$ that satisfy $\delta_{\sigma(v)} = \mathbb{L}$ for all $v \in V$, and for all H-augments $q$ of $p$:

$$o_q(s_0) = o_p(s_0)$$

It is immediate from the three definitions of security that


Running Example. System $S_0$ is poex-secure, however, that $q$ is a H-augment of $p$ makes all the difference for system $S_1$.

$$o_p(s_0) = \{\text{AthenB, BthenA}\} \neq \{\text{AthenB}\} = o_q(s_0)$$

Consequently, $S_1$ is not poex-secure.

We conclude that

Theorem 8. Security does not necessarily imply poex-security.

Arguably, security is the notion typically studied and proved in the literature whereas poex-security is needed as soon as partial rather than total orders are the appropriate models of execution, e.g., when reasoning at the machine instruction level of x86-based systems or when there are multiple observers and events may have duration.
2.3 A Hyperproperty-Based View

Maintaining the focus on traces, Clarkson and Schneider extended the universe of discourse set out by Alpern and Schneider in their seminal paper [1] on the topological view of system properties from a separation of safety and liveness to hyperproperties [2]. To do so, they lifted the topology used to characterize safety properties as limit-closed and liveness properties as dense from sets of traces to sets of sets of traces. A richer area of study arises, covering many interesting notions that have been studied, a.o., knowledge and distributed knowledge [5], and, in particular, many notions of security, including noninterference. Given that partial orders are isomorphic to certain sets of traces, we need to check whether our partial-order-based view is subsumed by a hyperproperty-based view of noninterference. We conjecture that it is not. In order to prove this conjecture, we need to construct a poex-secure system $S$ and a poex-insecure system $S'$ that both generate the same set of infinite traces using the same observation function. In this setting it probably suffices to consider systems as sets of poexes with a suitable notion of extending finite executions to infinite ones by stuttering. For $S'$ we could use $\text{POEX}(S_1)$ and for $S$ the set $\bigcup \{ \text{linearizations}(p) \mid p \in S' \}$. We emphasize that this is still a conjecture in need of a more thorough investigation.

2.4 Simultaneity

If individual observers can detect sets of actions that happen simultaneously, then a single observer can potentially detect partial order by observing that two events happened simultaneously. Neither multiple observers nor events with duration are necessary to do so; observers, however, need to be more powerful. In this note we succeeded to make partial order detectable without enhancing observers, by subtly weakening the model of execution.

3 Conclusion and Future Work

Multiple observers and events having duration are hardly avoidable in real systems. Plotkin and Pratt proved that this combination suffices for these observers to be able to distinguish partial from total order. The literature on noninterference since Goguen and Meseguer uses total orders as execution models. We have shown by way of a simple example that total order models can suggest a system as being secure whereas a more refined partial-order-based model of the same system exposes it as insecure.

The main practical point of this note is to caution against fallacies arising from trace-based noninterference proofs at abstract levels. Such a fallacy is likely when the abstract system is refined to lower level systems that are exposed to multiple observers and that represent individual abstract level events with sequences of lower level events or events having duration.

The main theoretical point of this note is that noninterference once again proves to be rather mercurial in nature and that it continues to provide an interesting area of research. We have contributed a definition of poex-security that
allows security-preserving refinement even in the presence of multiple observers and events with duration.

Future work will focus on exploring further properties of the new poex-security, how to verify it efficiently, and whether it applies meaningfully to nondeterministic systems. The latter would lead to investigations of the relation of poex-security to other formulations of security such as nondeducibility of inputs, nondeducibility of strategies, and unwinding-like properties. All of this would take place in an asynchronous setting since it is so far unclear whether our result has any relevance at all to synchronous systems.

Acknowledgments Ron van der Meyden inspired this paper by giving talks on his line of research into (mostly intransitive) noninterference and for discussions about an earlier draft of this paper. Yoram Moses is thanked for providing helpful comments on a later draft.

References
