Complexity and Unwinding for Intransitive Noninterference*

Sebastian Eggert\textsuperscript{1} Ron van der Meyden\textsuperscript{2}
Henning Schnoor\textsuperscript{1} Thomas Wilke\textsuperscript{1}

\textsuperscript{1}: Institut für Informatik, Kiel University
\textsuperscript{2}: School of Computer Science and Engineering, University of New South Wales

Email: sebastian.eggert@email.uni-kiel.de, meyden@cse.unsw.edu.au, henning.schnoor@email.uni-kiel.de, thomas.wilke@email.uni-kiel.de

May 11, 2014

\textbf{Abstract}

The paper considers several definitions of information flow security for intransitive policies from the point of view of the complexity of verifying whether a finite-state system is secure. The results are as follows. Checking (i) P-security (Goguen and Meseguer), (ii) IP-security (Haigh and Young), and (iii) TA-security (van der Meyden) are all in $\textbf{PTIME}$, while checking TO-security (van der Meyden) is undecidable, as is checking ITO-security (van der Meyden). The most important ingredients in the proofs of the $\textbf{PTIME}$ upper bounds are new characterizations of the respective security notions, which also lead to new unwinding proof techniques that are shown to be sound and complete for these notions of security, and enable the algorithms to return simple counter-examples demonstrating insecurity. Our results for IP-security improve a previous doubly exponential bound of Hadj-Alouane et al.

\textbf{Keywords:} noninterference, information flow, verification

\footnotesize{*This paper extends and significantly revises the paper \cite{1}. The main differences are that full proofs of results from \cite{1} are provided, new results concerning the notion of ITO-security are added, new results on unwindings for IP-security and TA-security are added, and these new unwindings are used as the basis for new algorithms that yield better complexity bounds than presented in \cite{1}.*}
1 Introduction

One of the fundamental methods in the construction of secure systems to high levels of assurance is to decompose the system into trusted and untrusted components, arranged in an architecture that constrains the possible causal effects and flows of information between these components. On the other hand, resource limitations and cost constraints may make it desirable for trusted and untrusted components to share resources. For example, it is cheaper for an intelligence analyst to handle high security and low security information on a single desktop machine than to use two physically separated machines. This leads to complex systems designs and implementations, in which the desired constraints on flows of information between trusted and untrusted components need to be enforced in spite of the fact that these components share resources. In order to provide high levels of assurance of implementations of this kind, it is desirable to have a formal theory of systems architecture and information flow, so that a design or implementation may be formally verified to conform to an information flow policy. Moreover, one would like, whenever possible, to automate the verification that a system satisfies such a formally defined policy. This motivates the problems we consider in this paper. We study the complexity of verification of a range of formally defined security policies that specify how a system is architecturally structured in terms of how information may flow between its components.

Attack model: The problems we consider in this paper address systems implementation attacks. We work in the paradigm of information flow security, where it is assumed that a (passive) adversary may attack the system by attempting to make subtle deductions from her possible observations of the system, exploiting covert channels that may exist in the system, in order to learn secrets that she is not authorized to possess. The automated analyses we consider aim to provide assurance that the system has been designed in such a way that such attacks are not possible, or to discover such attacks when they exist. The analysis can be applied both in circumstances where it is feared that a rogue systems developer may have deliberately constructed the system so as to contain such prohibited flows of information, as well as to ensure that such flows of information have not been inadvertently allowed to exist.

Policy model: Notions of noninterference—a first definition was given by Goguen and Meseguer [2]—are one approach to the formalisation of information flow and causal relationships. Noninterference was first proposed in the context of transitive information flow policies (with transitivity following from the partial order on security domains) but it was subsequently noted [3] that systems architectures often require intransitive policies. For example, a common architectural pattern is to restrict information flow from a high-level domain to a low-level domain so as to be possible only via a trusted downgrader (e.g., a declassification guard or encryption device). This pattern motivates an intransitive information flow policy, stating that information flow is permitted from the high-level domain to the downgrader and from the downgrader to the low-level domain, but not directly from the high-level domain to the low-level domain.
Goguen and Meseguer’s definition of noninterference, based on a “purge” function, does not yield the desired conclusions for intransitive policies. Haigh and Young proposed a variant (that we refer to as IP-security) for intransitive policies based on an “intransitive purge” function. Rushby [4] later refined their theory and developed connections to access control systems. Van der Meyden [5] has argued that the definitions of security for intransitive policies in these works suffer from some subtle flaws, and proposed some improved definitions, TA-security, TO-security and ITO-security, that first build an operational (full information protocol) model of the maximal permitted information flow in the system, and then compares the actual information flow to this maximal permitted information flow. The revised definitions can be shown to avoid the subtle flaws in the intransitive purge-based definition, and lead to a more satisfactory proof theory and connection to access control systems than in Rushby’s work (e.g., yielding both soundness and completeness results, whereas Rushby proved only soundness.)

**Verification:** The goal of high assurance systems development by formal verification motivates the investigation of techniques whereby a systems design or implementation can be formally shown to satisfy a formal definition of security. The technique of unwinding relations [6, 4] provides a proof method that has been applied to establish that a system satisfies noninterference properties, but it requires significant human ingenuity to define an unwinding relation that forms the basis for the proof, and typically also has involved manual driving (proof rule selection) of the theorem proving tool within which the proof is conducted.

A better alternative, more acceptable to engineers when it can be applied, is for the property to be verified by fully automatic techniques. There is a substantial body of work on automated verification techniques for transitive noninterference properties (which we discuss in Section [6]), but there has been significantly less work on automated verification techniques for intransitive noninterference properties.

**Contributions:** Our contribution in this paper is to provide a basis for automated verification of definitions of intransitive noninterference, by developing a characterization of the computational complexity of deciding whether a given finite-state system is secure with respect to an intransitive information flow policy according to this definition. In particular, we consider Goguen and Meseguer’s purge-based definition, IP-security, and van der Meyden’s definitions of TA-security, TO-security and ITO-security. We show that the last two of these definitions are undecidable, but the others are decidable in polynomial time and even in nondeterministic logarithmic space. We give algorithms for the decidable cases and analyse their complexity. Our results are based on new characterizations of IP-security and TA-security. Using these new characterizations, we develop new notions of unwinding for IP-security and TA-security, that give sound and complete proof techniques, and yield polynomial time decision procedures for these two notions. Our PTIME decision procedures exploit the new notions of unwinding. These new notions of unwinding are also of independent interest, in that they apply not just to the finite state case, where
we give the complexity bounds, but also to infinite state systems, where they can form the basis for proof theoretic verification methods.

The structure of the paper is as follows. In Section 2 we define the formal systems model that we work with, and recall the formal definitions of security for intransitive information flow policies that we study. New characterizations of IP-security and van der Meyden’s notion of TA-security are presented in Section 3. The new unwinding relations for IP-security and TA-security are developed in Section 4. Section 5 gives the complexity results for all the security notions that we consider. Our results are positioned within the literature in Section 6 where we discuss related work. Section 7 concludes with a discussion of open problems and future research directions. A reduction that deduces the undecidability of ITO-security from the undecidability of TO-security is presented in an appendix.

2 Basic Definitions and Notation

In this section, we introduce intransitive information flow policies and describe their motivation. We present a deterministic asynchronous systems model in which such policies may be interpreted, and then recall a number of different semantic interpretations of such policies in this system model that have been proposed in the literature.

2.1 Noninterference Policies

Noninterference policies are reflexive relations $\rightarrow \subseteq D \times D$, where $D$ is a set of “domains”. The intuitive reading of $u \rightarrow v$ is that “actions of domain $u$ are permitted to interfere with domain $v$”, or “information is permitted to flow from domain $u$ to domain $v$”. For any set $U \subseteq D$ the image of $U$, denoted $U^\rightarrow$, is defined by $U^\rightarrow = \{ v \in D \mid \exists u \in U : u \rightarrow v \}$. For a singleton set $\{u\}$ we also write $u^\rightarrow$ instead of $\{u\}^\rightarrow$.

The reason for the assumption of reflexivity is that, intuitively, a domain should be allowed to interfere with or have information about itself, since this cannot usually be prevented. In early work on noninterference [2], the relation $\rightarrow$ is also assumed to be transitive. This follows from the interpretation of domains as corresponding to security levels associated to classes of information and access rights, which have generally been taken to be partially ordered [7]. (In the classical multi-level security models, this partial order is derived from a linear order on security levels and the set containment order on sets of labels.)

One of the motivations for the consideration of policies $\rightarrow$ that are not transitive is that classical multilevel security policies are too restrictive for practical purposes, allowing flow of information from lower security levels to higher security levels, but prohibiting flow in the opposite direction. Such flows may be less frequent but are nevertheless required, e.g., for distribution of battle plans, in response to freedom of information requests, or for transmission of encrypted content across an insecure network. One of the ways this has been handled is to allow the general policy to be violated by a special downgrader component.
A typical downgrader policy is depicted in Figure 1. Here the usual (transitive) multi-level policy for domains Public, Secret and Top-Secret is extended by the addition of two domains DownS and DownP, that are responsible for downgrading of information from Top-Secret to Secret, and from Secret to Public, respectively. These domains are trusted to enforce whatever policy constraints apply to the downgrading of information. Note that it would not be appropriate to apply an assumption of transitivity on this setting, since then, e.g., the edges involving DownS would imply that Top-Secret $\rightarrow$ Secret, i.e., a direct flow of information from Top-Secret to Secret is permitted.

Subsequent work on intransitive noninterference has taken a somewhat extended interpretation of the term “domain,” treating this more as akin to “component” in a systems architecture. Figure 2 shows a systems architecture, discussed in [4] and [8], for a system in which messages are sent from a high security (Red) domain through a low security (Black) domain, with the global security policy stating that all content, except the message header, must be encrypted, and uncontrolled flow of information from Red to Black is prohibited. The architecture proposes to achieve this goal by having the Bypass component check a (more detailed) policy on the allowed header structure and content, and by having the Crypto component enforce a local policy stating that all output must be encrypted. These flows are recomposed into the encrypted message (with header) at the Black component. Crypto and Bypass are assumed to be trusted components of low enough complexity that they can be verified to enforce their local policies. Red (which may contain Trojans) and Black (which is at a low security level) are not assumed to be trusted. The argument for security of the system is intended to follow from the structure of the information flows.
in the architecture, plus the assumption that the trusted components correctly implement their local policies.

MILS security, as expounded in [8], proposes to base development of certifiably secure systems on design level arguments of this type, together with implementations in which mechanisms such as separation kernels or periods processing are used to enforce the systems architecture. We refer to [8] for a more detailed discussion of MILS security and the proposed structure of the argument for security of the system in Figure 2.

We note that intransitive information flow policies are intended to express just the architectural structure of information flow, rather than encompass all the details of security policy. One key point is that implementations may involve resource sharing, which may mean that it is not immediately apparent that the design level architecture is enforced in the implementation.

For example, Figure 3 illustrates a design level policy for a system with multiple independent security levels that could be implemented, as shown in Figure 4, by a trusted multiplexer component that handles information from multiple security levels. One of the issues in the verification of such systems is
to determine whether such a resource sharing implementation correctly enforces the design level architecture. The definitions in the following sections provide a number of distinct semantic interpretations of information flow policies that have been proposed to formalize what it means to implement the notion of correct enforcement.

2.2 State-Observed Machine Model

Several different types of semantic models have been used in the literature on noninterference. (See [9] for a comparison and a discussion of their relationships.) We work here with the state-observed machine model used by Rushby [4], but similar results would be obtained for other models.

This model consists of deterministic machines of the form \( \langle S, s_0, A, \text{step}, \text{obs}, \text{dom} \rangle \), where \( S \) is a set of states, \( s_0 \in S \) is the initial state, \( A \) is a set of actions, \( \text{dom}: A \to D \) associates each action with an element of the set \( D \) of security domains, \( \text{step}: S \times A \to S \) is a deterministic transition function, and \( \text{obs}: S \times D \to O \) maps states to an observation in some set \( O \), for each security domain. We may also refer to security domains more succinctly as “agents”. We write \( s \cdot \alpha \) for the state reached by performing the sequence of actions \( \alpha \in A^* \) from state \( s \), defined inductively by \( s \cdot \epsilon = s \), and \( s \cdot \alpha a = \text{step}(s \cdot \alpha, a) \) for \( \alpha \in A^* \) and \( a \in A \). Here, \( \epsilon \) denotes the empty sequence. For any string \( \alpha \), we say a symbol \( a \) occurs in \( \alpha \) if \( \alpha = \beta a \beta' \) for some strings \( \beta, \beta' \). We define \( \text{alph}(\alpha) \) as the set of all symbols occurring in \( \alpha \).

2.3 The Purge Function

Noninterference is given a formal semantics in the transitive case [2] using a definition based on a “purge” function. Given a set \( E \subseteq D \) of domains and a sequence \( \alpha \in A^* \), we write \( \alpha \rightharpoonup E \) for the subsequence of all actions \( a \) in \( \alpha \) with \( \text{dom}(a) \in E \). Given a policy \( \rightharpoonup \), we define the function \( \text{purge}: A^* \times D \to A^* \) by

\[
\text{purge}(\alpha, u) = \alpha \rightharpoonup \{ v \in D \mid v \rightharpoonup u \}.
\]
Figure 5: A system that is TO-secure but not P-secure

For clarity, we may use subscripting of agent arguments of functions, writing, e.g., \( \text{purge}(\alpha, u) \) as \( \text{purge}_u(\alpha) \). The system \( M \) is said to be secure with respect to the transitive policy \( \rightarrow \), when, for all \( \alpha \in A^* \) and domains \( u \in D \), we have \( \text{obs}_u(s_0 \cdot \alpha) = \text{obs}_u(s_0 \cdot \text{purge}_u(\alpha)) \). That is, each agent’s observations are as if only interfering actions had been performed. An equivalent formulation (which we state more generally for policies that are not necessarily transitive, in anticipation of later discussion) is the following:

**Definition 1 (P-security)** A system \( M \) is P-secure with respect to a policy \( \rightarrow \) if for all sequences \( \alpha, \alpha' \in A^* \) such that \( \text{purge}_u(\alpha) = \text{purge}_u(\alpha') \), we have \( \text{obs}_u(s_0 \cdot \alpha) = \text{obs}_u(s_0 \cdot \alpha') \).

This can be understood as saying that agent \( u \)'s observation depends only on the sequence of interfering actions that have been performed.

### 2.4 The Intransitive Purge Function

While P-security is a reasonable definition of security for transitive information flow policies, it works less well for intransitive policies. Figure 5 illustrates a system that is, intuitively, secure for the downgrader policy \( H \rightarrow D \rightarrow L \), but which does not satisfy P-security. Here \( h, d, l \) are actions of domains \( H, D, L \), respectively, and the observations in each domain are depicted below the states. Intuitively, the observations convey a single bit of information: “has \( H \) ever performed the action \( h \)?”. Domains \( H \) and \( D \) learn that \( H \) has performed \( h \) as soon as this action is performed (by their observations turning to value 1), but \( L \) does not learn this until after \( D \) subsequently performs the downgrading action \( d \). Since the policy permits \( D \) to transmit information about \( H \), the system is secure. However, this system does not satisfy P-security, since we have \( \text{purge}_L(hdl) = dl = \text{purge}_L(dl) \) but \( \text{obs}_L(s_0 \cdot hdl) = 1 \neq 0 = \text{obs}_L(s_0 \cdot \text{purge}_L(dl)) \). Intuitively, P-security says that \( L \) observations depend only on what \( D \) and \( L \) actions have been performed, so cannot contain information about \( H \), even though the policy, intuitively, permits \( D \) to transmit such information.
To address this deficiency, Haigh and Young [3] generalized the definition of the purge function to intransitive policies. Intuitively, the intransitive purge of a sequence of actions with respect to a domain $u$ is the largest subsequence of actions that could form part of a causal chain of effects (permitted by the policy) ending with an effect on domain $u$. More formally (we follow the presentation from [4]), the definition makes use of a function $\text{sources} : A^* \times D \rightarrow \mathcal{P}(D)$ defined inductively by $\text{sources}(\epsilon, u) = \{u\}$ and, for $a \in A$ and $\alpha \in A^*$, if there exists $v \in \text{sources}(\alpha, u)$ with $\text{dom}(a) \rightarrow v$, then

$$\text{sources}(aa, u) = \text{sources}(\alpha, u) \cup \{\text{dom}(a)\},$$

and else

$$\text{sources}(aa, u) = \text{sources}(\alpha, u).$$

Intuitively, $\text{sources}(\alpha, u)$ is the set of domains $v$ such that there exists a sequence of permitted interferences from $v$ to $u$ within $\alpha$. The intransitive purge function $\text{ipurge} : A^* \times D \rightarrow A^*$ is then defined inductively by $\text{ipurge}(\epsilon, u) = \epsilon$ and, for $a \in A$ and $\alpha \in A^*$, if $\text{dom}(a) \in \text{sources}(aa, u)$, then

$$\text{ipurge}(aa, u) = a \text{ipurge}(\alpha, u),$$

and else

$$\text{ipurge}(aa, u) = \text{ipurge}(\alpha, u).$$

The intransitive purge function is then used in place of the purge function in Haigh and Young’s definition:

**Definition 2 (IP-security)** A system $M$ is IP-secure with respect to a (possibly intransitive) policy $\rightarrow$ if for all sequences $\alpha \in A^*$ and $u \in D$, we have $\text{obs}_u(s_0 \cdot \alpha) = \text{obs}_u(s_0 \cdot \text{ipurge}_u(\alpha)).$

Since the function $\text{ipurge}_u$ on $A^*$ is idempotent, this definition, like the definition for the transitive case, can be formulated as: $M$ is IP-secure with respect to a policy $\rightarrow$ if for all $u \in D$ and all sequences $\alpha, \alpha' \in A^*$ with $\text{ipurge}_u(\alpha) = \text{ipurge}_u(\alpha')$, we have $\text{obs}_u(s_0 \cdot \alpha) = \text{obs}_u(s_0 \cdot \alpha')$. It can be seen that $\text{ipurge}_u(\alpha) = \text{purge}_u(\alpha)$ when $\rightarrow$ is transitive, so IP-security is in fact a generalisation of the definition of security for transitive policies.

### 2.5 The $\text{ta}$ Function

It has been noted by van der Meyden [5] that IP-security classifies some systems as secure where there is, intuitively, an insecure flow of information that relates to a domain learning ordering information about the actions of other domains that it should not have.

Figure [6] depicts part of a system $M$ and a policy $\rightarrow$ such that $M$ is IP-secure, but for which the conclusion that the system is secure is questionable. We
Figure 6: A system that is IP-secure but not TA-secure
sketch the argument for this here, and refer the reader to [5] for a more rigorous presentation. Intuitively, the system is comprised of two High security level domains $H_1, H_2$, each with a downgrader ($D_1, D_2$, respectively) to the Low security domains $L$. The actions $h_1, h_2, d_1, d_2$ are associated to the domains $H_1, H_2, D_1, D_2$, respectively, and state transitions are depicted only when there is a change of state. The observations of $L$ are depicted at two of the states; at all other states we assume that $L$ makes observation 0. All other agents may be assumed to make observation 0 at all states. Intuitively, at the state where $L$ observes 1, it is possible for $L$ to deduce that there has been an occurrence of $h_1$ followed by an occurrence of $h_2$; the state where $L$ observes 2, it is possible for $L$ to deduce that these actions have occurred in the opposite order.

We show that this system is IP-secure: Suppose we have $\text{ipurge}_L(\alpha) = \text{ipurge}_L(\beta)$, and one of $\text{obs}_L(s_0 \cdot \alpha)$ or $\text{obs}_L(s_0 \cdot \beta)$ is 1 or 2, say the former is equal to 1. Then this sequence must contain an occurrence of $h_1$ before an occurrence of $h_2$, and each is followed by $d_1$ and $d_2$, respectively. This observation shows, in fact, that $L$ knows the order of the first two $H_1$ and $H_2$ actions in the sequence $\alpha$. Because $\text{ipurge}_L$ preserves $h_1$ when it is followed by $d_1$, and similarly for $h_2$ and $d_2$, and also preserves the order of actions that it retains, the same statement must hold for $\beta$, and it then follows that also $\text{obs}_L(s_0 \cdot \beta) = 1 = \text{obs}_L(s_0 \cdot \alpha)$. If neither observation is in 1, 2, then both are equal to 0, and again we have the required equality of observations.

On the other hand, the conclusion that the system is secure is somewhat peculiar. Each of the downgraders is individually permitted by the policy to know only about activity in its associated High level domain, and its own activity. Thus, individually, neither $D_1$ nor $D_2$ can know the order of the first two $H_1$ and $H_2$ actions. Moreover, since the system is asynchronous, even if we were to combine all the information that the downgraders are permitted to know, we would still not be able to deduce the order on the $H_1, H_2$ actions. We therefore have the peculiar conclusion that the system is classified by IP-security to be secure, but it allows $L$ to learn information that would not be permitted to be known to the two domains $D_1, D_2$, which are supposed to filter all flow of information from $H_1, H_2$, even if these domains were to combine their information.

To address this peculiarity, van der Meyden has proposed some other interpretations of intransitive policies. Both proceed by first defining a concrete operational model of the maximal amount of information that an agent is permitted to have after some sequence of actions has been performed. Security of the system is then defined by requiring that an agent’s observation may not contain more than this maximal amount of information.

In the first operational model, when an agent performs an action, it transmits what it is permitted to know to other agents, subject to constraints in the policy. The following definition expresses this in a weaker way than the ipurge function.

Given sets $X$ and $A$, let the set $\mathcal{T}(X, A)$ be the smallest set containing $X$ and such that if $x, y \in \mathcal{T}$ and $z \in A$ then $(x, y, z) \in \mathcal{T}$. Intuitively, the elements of $\mathcal{T}(X, A)$ are binary trees with leaves labelled from $X$ and interior nodes labelled from $A$.  

11
Given a policy \( \rightarrow \), define, for each agent \( u \in D \), the function \( \mathit{ta}_u : A^* \to \mathcal{T}\{\epsilon\}, A\) inductively by \( \mathit{ta}_u(\epsilon) = \epsilon \), and, for \( \alpha \in A^* \) and \( a \in A \),

\[
\mathit{ta}_u(\alpha a) = \begin{cases} 
 \mathit{ta}_u(\alpha) & \text{if } \text{dom}(a) \not\rightarrow u, \\
 (\mathit{ta}_u(\alpha), \mathit{ta}_{\text{dom}(a)}(\alpha), a) & \text{otherwise.}
\end{cases}
\]

Intuitively, \( \mathit{ta}_u(\alpha) \) captures the maximal information that agent \( u \) may, consistently with the policy \( \rightarrow \), have about the past actions of other agents. Initially, an agent has no information about what actions have been performed. The recursive clause describes how the maximal information \( \mathit{ta}_u(\alpha) \) permitted to flow to \( u \) after the performance of \( \alpha \) changes when the next action \( a \) is performed. If \( a \) may not interfere with \( u \), then there is no change, otherwise, \( u \)'s maximal permitted information is increased by adding the maximal information permitted to \( \text{dom}(a) \) at the time \( a \) is performed (represented by \( \mathit{ta}_{\text{dom}(a)}(\alpha) \)), as well the fact that \( a \) has been performed. Thus, this definition captures the intuition that an agent may only transmit information that it is permitted to have, and then only to agents with which it is permitted to interfere.

**Definition 3 (TA-security)** A system \( M \) is TA-secure with respect to a policy \( \rightarrow \) if for all agents \( u \) and all \( \alpha, \alpha' \in A^* \) such that \( \mathit{ta}_u(\alpha) = \mathit{ta}_u(\alpha') \), we have \( \mathit{obs}_u(s_0 \cdot \alpha) = \mathit{obs}_u(s_0 \cdot \alpha') \).

Intuitively, this says that each agent’s observations provide the agent with no more than the maximal amount of information that may have been transmitted to it, as expressed by the functions \( \mathit{ta} \).

### 2.6 The to Function

In the definition of TA-security, the operational model of information flow given by the function \( \mathit{ta} \) permits a domain to transmit information that it may have, even if it has never observed anything from which it could deduce that information. Arguably, this is too liberal.
Figure 7 shows a system for the downgrader policy $H \rightarrow D \rightarrow L$, similar to that in Figure 5. It can be argued that the system is TA-secure; we leave the details to the reader. Again, when $L$ observes 1, it can deduce that $H$ has performed the action $h$, and indeed, this observation can only occur after $D$ has performed the action $d$, thereby downgrading the information about $H$. On the other hand, note that in this system, $D$’s observation is always 0, so $D$ cannot know, on the basis of its observations, whether $H$ has performed $h$. $D$ is therefore transmitting to $L$ information that it does not itself have.

Van der Meyden [5] therefore also considers a variant operational model in which a domain transmits only what it has actually observed. This yields the alternate notion of TO-security.

The sequence of all observations and actions of a domain is denoted as its view. Formally, the notion of view is defined as follows. The definition uses an absorptive concatenation function $\circ$, defined over a set $X$ by $s \circ x = s$ if $x$ is equal to the final element of $s$ (if any), and $s \circ x = s \cdot x$ (ordinary concatenation) otherwise, for every $s \in X^*$ and $x \in X$. Define the view of domain $u$ with respect to a sequence $\alpha \in A^*$ using the function $\text{view}_u: A^* \rightarrow (A \cup O)^*$ (where $O$ is the set of observations in the system) defined by

$$
\text{view}_u(\epsilon) = \text{obs}_u(s_0), \quad \text{and}
\text{view}_u(\alpha a) = (\text{view}_u(\alpha) \cdot b) \circ \text{obs}_u(s_0 \cdot \alpha),
$$

where $b = a$ if $\text{dom}(a) = u$ and $b = \epsilon$ otherwise. That is, $\text{view}_u(\alpha)$ is the sequence of all observations and actions of domain $u$ in the run generated by $\alpha$, compressed by the elimination of stuttering observations. Intuitively, $\text{view}_u(\alpha)$ is the complete record of information available to agent $u$ in the run generated by the sequence of actions $\alpha$. The reason we apply the absorptive concatenation is to capture that the system is asynchronous, with agents not having access to a global clock. The effect of this operation is to reduce any stuttering of an observation in the run to a single copy. Thus, two sequences that only differ from each other in repetitions of a single observation are not distinguishable by the agent.

Given a policy $\rightarrow$, for each domain $u \in D$, define the function $\text{to}_u: A^* \rightarrow T((A \cup O)^*, A)$ by $\text{to}_u(\epsilon) = \text{obs}_u(s_0)$ and

$$
\text{to}_u(\alpha a) = \begin{cases} 
\text{to}_u(\alpha) & \text{if } \text{dom}(a) \not\rightarrow u, \\
(\text{to}_u(\alpha), \text{view}_{\text{dom}(a)}(\alpha), a) & \text{otherwise.}
\end{cases}
$$

Intuitively, this definition takes the model of the maximal information that an action $a$ may transmit after the sequence $\alpha$ to be the fact that $a$ has occurred, together with the information that $\text{dom}(a)$ actually has, as represented by its view $\text{view}_{\text{dom}(a)}(\alpha)$. By contrast, TA-security uses in place of this the maximal information that $\text{dom}(a)$ may have. We may now base the definition of security on the function $\text{to}$ rather than $\text{ta}$.

**Definition 4 (TO-security)** The system $M$ is TO-secure with respect to $\rightarrow$.
if for all domains \( u \in D \) and all \( \alpha, \alpha' \in A^* \) with \( \text{to}_u(\alpha) = \text{to}_u(\alpha') \), we have \( \text{obs}_u(s_0 \cdot \alpha) = \text{obs}_u(s_0 \cdot \alpha') \).

It is possible to give a flatter representation of the information in \( \text{to}_u(\alpha) \) that clarifies the relationship of this definition to P-security. Define the possibly transmitted view of domain \( u \) for a sequence of actions \( \alpha \) to be the largest prefix \( \text{tview}_u(\alpha) \) of \( \text{view}_u(\alpha) \) that ends in an action \( a \) with \( \text{dom}(a) = u \). Then we have the following result, which intuitively says that \( u \)'s observations depend only on (1) the parts of the views of other agents which are permitted to pass information to \( u \) that they have actually acted to transmit, and (2) \( u \)'s knowledge of the ordering of its own actions and the actions of these other agents.

**Proposition 1** (Characterization of TO-security [5]) The system \( M \) is TO-secure with respect to a policy \( \rightarrow \) iff for all sequences \( \alpha, \alpha' \in A^* \), and domains \( u \in D \), if \( \text{purge}_u(\alpha) = \text{purge}_u(\alpha') \) and \( \text{tview}_u(\alpha) = \text{tview}_u(\alpha') \) for all domains \( v \neq u \) such that \( v \rightarrow u \), then \( \text{obs}_u(s_0 \cdot \alpha) = \text{obs}_u(s_0 \cdot \alpha') \).

### 2.7 The ito Function

In order to compare with a definition of Roscoe and Goldsmith [10], van der Meyden has also introduced a variant of TO-security called ITO-security, in which information is transmitted slightly faster. We also consider this notion here since our complexity results bear on algorithmic claims of Roscoe and Goldsmith.

Given a policy \( \rightarrow \), for each domain \( u \in D \), define the function \( \text{ito}_u : A^* \rightarrow \mathcal{T}(O(A \cup O)^*, A) \) by \( \text{ito}_u(\epsilon) = \text{obs}_u(s_0) \) and

\[
\text{ito}_u(\alpha a) = \begin{cases} 
\text{ito}_u(\alpha) & \text{if } \text{dom}(a) \neq u, \\
(\text{ito}_u(\alpha), \text{view}_{\text{dom}(a)}(\alpha), a) & \text{if } \text{dom}(a) = u, \\
(\text{ito}_u(\alpha), \text{view}_{\text{dom}(a)}(\alpha a), a) & \text{otherwise}.
\end{cases}
\]

This definition is just like that of \( \text{to} \), with the difference that the information that may be transmitted to \( u \) by an action \( a \) such that \( \text{dom}(a) \rightarrow u \) but \( \text{dom}(a) \neq u \), includes the observation \( \text{obs}_{\text{dom}(a)}(s_0 \cdot \alpha a) \) obtained in domain \( \text{dom}(a) \) immediately after the occurrence of action \( a \). Intuitively, the definition of security based on this notion will allow that the action \( a \) transmits not just the information observable to \( \text{dom}(a) \) at the time that it is invoked, but also the new information that it computes and makes observable in \( \text{dom}(a) \). This information is not included in the value \( \text{ito}_{\text{dom}(a)}(\alpha a) \) itself, since the definition of security will state that the new observation may depend only on this value. The nomenclature in this case is intended to be suggestive of immediate transmission of information about observations.

The following definition follows the pattern of the others, but based is on the functions \( \text{ito} \).

**Definition 5** The system \( M \) is ITO-secure with respect to \( \rightarrow \) if for all domains \( u \in D \) and all \( \alpha, \alpha' \in A^* \) with \( \text{ito}_u(\alpha) = \text{ito}_u(\alpha') \), we have \( \text{obs}_u(s_0 \cdot \alpha) = \text{obs}_u(s_0 \cdot \alpha') \).
Figure 8 gives an example of a system, for the downgrader policy $H \hookrightarrow D \hookrightarrow L$, that is ITO-secure, but not TO-secure. Intuitively, in the action sequence $hd$, the downgrader learns that $h$ has been performed (by making observation 1) from the observation that it makes after performing the action $d$. ITO-security permits that the information in this observation is transmitted to $L$ by the action $d$, whereas TO-security does not.

The definitions introduced above are shown in [5] to be related as follows: P-security implies TO-security implies ITO-security implies TA-security implies IP-security. The converse of each of these implications does not hold: Figures 5-8 provide counter-examples. In the special case of transitive policies $\hookrightarrow$, all these notions are equivalent.

3 Characterization of IP-security and TA-security

In this section, we develop new characterizations of IP-security and TA-security, that enable the new unwinding and decision procedures for these notions of security.

3.1 Characterization of IP-security

We present a new characterization of IP-security. This characterization is the main tool for our later algorithm that verifies IP-security in polynomial time.

Intuitively, the ipurge function that defines IP-security removes actions that should be irrelevant for the domain $u$ from its “visible trace.” This leads us to the definition of the relation $\rightarrow_{u}^{irr}$: for $u \in D$ and $\alpha, \alpha' \in A^*$, we define $\alpha \rightarrow_{u}^{irr} \alpha'$ if $\text{ipurge}_u(\alpha) = \text{ipurge}_u(\alpha')$ and there exist $\beta, \beta' \in A^*$, and $a \in A$ such that $\alpha = \beta a \beta'$ and $\alpha' = \beta'$. That is, $\alpha \rightarrow_{u}^{irr} \alpha'$ if $\alpha'$ is obtained from $\alpha$ by removing a single action that is “irrelevant” in the sense that (according to the information flow allowed by the policy) $u$ should not be able to observe whether the removed action has occurred at all. The symmetric closure of $\rightarrow_{u}^{irr}$ is denoted with $\leftrightarrow_{u}^{irr}$.
Note that if \( \text{ipurge}_u(\alpha) = \text{ipurge}_u(\alpha') \), then there exists a sequence \( \alpha = \alpha_0 \leftrightarrow \alpha_1 \leftrightarrow \ldots \leftrightarrow \alpha_n = \alpha' \). If \( \text{obs}_u(s_0 \cdot \alpha) \neq \text{obs}_u(s_0 \cdot \alpha') \), then we must have \( \text{obs}_u(s_0 \cdot \alpha_k) \neq \text{obs}_u(s_0 \cdot \alpha_{k+1}) \) for some \( k \). Thus, directly from the definition of IP-security, we obtain that a system is IP-insecure if there exists a domain \( u \in D \), a reachable state \( q \in S \), \( a \in A \) and \( \alpha \in A^* \) such that \( \text{ipurge}_u(aa) = \text{ipurge}_u(\alpha) \) and \( \text{obs}_u(q \cdot aa) \neq \text{obs}_u(q \cdot \alpha) \). We now state a lemma that shows that we can put some restrictions on \( \alpha \). The lemma shows that if a system is not IP-secure, then an \( \alpha \) and \( a \) as above exist such that additionally, \( \alpha \) does not contain any action \( c \) whose domain \( \text{dom}(c) \) can be influenced by \( \text{dom}(a) \). This allows us to reduce the search space for a witness of insecurity significantly when designing our algorithms.

In the following, we will always assume that every state \( s \in S \) is reachable, i.e., there is a sequence \( \alpha \in A^* \) such that \( s_0 \cdot \alpha = s \).

**Lemma 1** A system \( M \) is IP-insecure iff there exist \( u \in D \), \( q \in S \), \( a \in A \) and \( \alpha \in A^* \) such that

(i) \( \text{ipurge}_u(aa) = \text{ipurge}_u(\alpha) \),

(ii) \( \text{obs}_u(q \cdot aa) \neq \text{obs}_u(q \cdot \alpha) \), and

(iii) \( \text{dom}(a)^{-1} \cap \{ \text{dom}(c) | c \in \text{alph}(\alpha) \} = \emptyset \).

**Proof:** Let a system \( M \) be IP-insecure. Then there exist \( u \in D \), \( q \in S \), \( a \in A \) and \( \alpha \in A^* \) such that \( \text{ipurge}_u(aa) = \text{ipurge}_u(\alpha) \) and \( \text{obs}_u(q \cdot aa) \neq \text{obs}_u(q \cdot \alpha) \).

We fix this domain \( u \) and choose \( q, a, \alpha \) such that \( |\alpha| \) is minimal for all choices of \( q, a, \alpha \) satisfying \( \text{ipurge}_u(aa) = \text{ipurge}_u(\alpha) \) and \( \text{obs}_u(q \cdot aa) \neq \text{obs}_u(q \cdot \alpha) \).

Assume that there exists \( b \in A \) such that \( \alpha = \beta b' \) for some \( \beta, b' \in A^* \) with \( \text{dom}(a) \rightarrow \text{dom}(b) \). We show that such an action \( b \) can’t exist by considering the following three cases:

1. \( \text{obs}_u(q \cdot b\beta') \neq \text{obs}_u(q \cdot \beta') \): We set \( q' = q \cdot \beta, a' = b \) and \( \alpha' = \beta' \).

   Then the condition \( \text{ipurge}_u(b\beta') = \text{ipurge}_u(\beta') \) is satisfied and we could choose \( q', a', \alpha' \) instead of \( q, a, \alpha \). This contradicts the minimal length of \( \alpha \), since \( |\alpha'| < |\alpha| \).

2. \( \text{obs}_u(q \cdot a\beta b') \neq \text{obs}_u(q \cdot a\beta') \): We set \( q' = q \cdot a\beta, a' = b \) and \( \alpha' = \beta' \).

   With the same argument as in the previous case, we get a contradiction.

3. \( \text{obs}_u(q \cdot b\beta') = \text{obs}_u(q \cdot \beta') \) and \( \text{obs}_u(q \cdot a\beta b') = \text{obs}_u(q \cdot a\beta') \): This gives

\[
\text{obs}_u(q \cdot \beta') = \text{obs}_u(q \cdot b\beta') \neq \text{obs}_u(q \cdot a\beta b') = \text{obs}_u(q \cdot a\beta') .
\]

   We set \( q' = q, a' = b \) and \( \alpha' = \beta' \). Again, the condition \( \text{ipurge}_u(b\beta') = \text{ipurge}_u(a\beta') \) is satisfied. We get a contradiction since \( |\alpha'| < |\alpha| \).

\( \square \)

In fact, with nearly the same proof, one can show that, if the length of \( \alpha \) is minimal for all choices of \( q, a, \alpha \), then \( \alpha = \text{ipurge}_u(\alpha) \).
3.2 Characterization of TA-security

We present a new characterization of TA-security that also makes precise its relationship to IP-security. As seen earlier, the latter is concerned with the question which actions an agent $u$ may observe at all, hence $\text{ipurge}_u(\alpha)$ is obtained from $\alpha$ by removing from $\alpha$ actions that should be “unobservable” for $u$, provided that information only flows as specified by the security policy. The definition of TA-security in [5] was motivated by the observation that the security-relevant information that should be unobservable to some agents is not just which actions appear at all, but also information about the order in which certain actions are performed. This type of information-flow is not prohibited by the definition of IP-security.

In this section we show that what separates the definition of IP-security from that of TA-security is the question how much information is known about execution orders of actions. TA-security can essentially be seen as IP-security plus the requirement that an agent should only have access to “timing information” (i.e., information about the order of the occurrence of actions) insofar as permitted by the security policy.

To formalize this, we require a few technical definitions. The following definition captures the situation in which an agent $u$ should not have information about the order in which certain actions are performed, although it may know whether these actions have been performed, and how often.

**Definition 6 (swappable)** Let $\alpha, \alpha' \in A^*$ and $a, b \in A$ and $u \in D$. We write $\alpha ab \alpha' \leftrightarrow \text{swap}_u ab \alpha'$ iff $\text{dom}(a) \cap \text{dom}(b) \cap \{u, \text{dom}(c) | c \in \text{alph}(ab\alpha')\} = \emptyset$. In this case, we call the actions $a$ and $b$ swappable in $\alpha ab \alpha'$.

For any relation $\rightarrow$, we define $\rightarrow^*$ as the reflexive closure of $\rightarrow$ and $\rightarrow^*$ as the reflexive, transitive closure of $\rightarrow$.

We will see later that Definition 6 captures exactly the issue mentioned above: If $\alpha ab \alpha' \leftrightarrow \text{swap}_u ab \alpha'$, then the action sequences $\alpha ab \alpha'$ and $ab \alpha'$ should be indistinguishable for agent $u$, even though it is allowed to know whether actions $a$ and $b$ have been performed. The reason why, intuitively, u should not have access to this “timing information” is that only agents $w \in \text{dom}(a) \cap \text{dom}(b)$ can directly observe whether $a$ or $b$ is performed first. If no agent that can observe this information directly performs any action in $\alpha$, then, after performing $\alpha$, the agent $u$ should not have this information either (unless of course, $u$ is in the intersection.)

We call strings $\alpha, \alpha' \in A^*$ order indistinguishable for $u$, and write $\alpha \equiv^*_u \alpha'$, if $\alpha \leftrightarrow^*_u \alpha'$.

We note that the definition of swappable could be relaxed to also allow swaps of actions $a$ and $b$ when the agents that observe both $a$ and $b$ only perform actions in $\alpha$ that cannot be observed by $u$ via the policy—however we will later only apply these definitions to action sequences to which $\text{ipurge}$ has already been applied. We therefore use the above definition to simplify notation. The following lemma shows that our definition correctly captures the above intuition.
It states that information about the order of “swappable” operations are indeed hidden from an agent by the definition of TA-security.

Lemma 2 Let \( u \in D \), \( \alpha, \alpha' \in A^* \) with \( \alpha \leftrightarrow_u \alpha' \), then \( \mathbf{ta}_u(\alpha) = \mathbf{ta}_u(\alpha') \).

Proof: Let be \( \beta, \beta' \in A^* \) and \( a, b \in A \) such that \( \alpha = \beta \beta' \) and \( \alpha' = \beta' b \beta \). We proceed with an induction on the length of \( \beta' \). First we assume that \( \beta ab \leftrightarrow_u \beta ba b' \) and show for all \( u' \not\in \text{dom}(a)^- \cap \text{dom}(b)^- \) that \( \mathbf{ta}_u(\beta ab) = \mathbf{ta}_u(\beta ba b') \) holds. Note that by definition of \( \leftrightarrow_u \) we have \( u, \text{dom}(a), \text{dom}(b) \not\in \text{dom}(a)^- \cap \text{dom}(b)^- \). Without loss of generality, assume that \( u' \in \text{dom}(a)^- \cup \text{dom}(b)^- \), since otherwise \( \mathbf{ta}_u(\beta ab) = \mathbf{ta}_u(\beta) = \mathbf{ta}_u(\beta ba) \). Also without loss of generality assume that \( u' \in \text{dom}(a)^- \). Therefore \( \text{dom}(b) \not\leftrightarrow \text{dom}(a) \) and \( \text{dom}(b) \not\leftrightarrow u' \). This gives

\[
\mathbf{ta}_u(\beta ab) = (\mathbf{ta}_u(\beta), \mathbf{ta}_u(\beta'), a) = (\mathbf{ta}_u(\beta b), \mathbf{ta}_u(\beta b), a) = \mathbf{ta}_u(\beta ba) .
\]

Assume that \( \alpha c \leftrightarrow_u \alpha' c \) for some \( c \in A \). And we assume that inductively, for any agent \( u' \in D \):

\[
\text{If } \alpha \leftrightarrow_u \alpha' \text{, then } \mathbf{ta}_{u'}(\alpha) = \mathbf{ta}_{u'}(\alpha') .
\]

The definition of \( \leftrightarrow_u \) gives

\[
0 = \text{dom}(a)^- \cap \text{dom}(b)^- \cap \{ u, \text{dom}(d) | d \in \text{alph}(ab'c) \}
\]

\[
= \text{dom}(a)^- \cap \text{dom}(b)^- \cap \{ u, \text{dom}(d) | d \in \text{alph}(ab'c) \}
\]

\[
\cup \{ \text{dom}(c), \text{dom}(d) | d \in \text{alph}(ab'c) \}
\]

\[
= \text{dom}(a)^- \cap \text{dom}(b)^- \cap \{ u, \text{dom}(d) | d \in \text{alph}(ab'c) \}
\]

\[
\cup \{ \text{dom}(c), \text{dom}(d) | d \in \text{alph}(ab'c) \}
\].

Therefore \( \beta ab \leftrightarrow_u \beta ba b' \), and from the prerequisites we also know that \( \beta ab \leftrightarrow_u \beta ba b' \). Applying the induction hypothesis gives \( \mathbf{ta}_u(\alpha) = \mathbf{ta}_u(\alpha') \) and \( \mathbf{ta}_{\text{dom}(c)}(\alpha) = \mathbf{ta}_{\text{dom}(c)}(\alpha') \). If \( \text{dom}(c) \not\leftrightarrow u \) the we get directly,

\[
\mathbf{ta}_u(\alpha c) = \mathbf{ta}_u(\alpha) = \mathbf{ta}_u(\alpha') = \mathbf{ta}_u(\alpha' c) .
\]

In the case of \( \text{dom}(c) \leftrightarrow u \) we get:

\[
\mathbf{ta}_u(\alpha c) = (\mathbf{ta}_u(\alpha), \mathbf{ta}_{\text{dom}(c)}(\alpha), c) = (\mathbf{ta}_u(\alpha'), \mathbf{ta}_{\text{dom}(c)}(\alpha'), c) .
\]

\( \square \)

The following corollary combines the above result and the fact that TA-security implies IP-security:

Corollary 1 Let be \( u \in D \) and \( \alpha, \alpha' \in A^* \) with \( \mathbf{ipurge}_u(\alpha) \equiv_u \mathbf{ipurge}_u(\alpha') \), then \( \mathbf{ta}_u(\alpha) = \mathbf{ta}_u(\alpha') \).
Proof: From the definition of TA-security it follows (see also the proof of Theorem 1 in the full version of [5]) that ipurge\(_u\)(\(\alpha\)) \(\equiv\) ipurge\(_u\)(\(\alpha'\)) implies \(\text{ta}_u(\alpha) = \text{ta}_u(\alpha')\). The corollary now follows from this result and repeated application of Lemma 2.

We now state the result mentioned earlier: TA-security is, in a very precise sense, IP-security plus the requirement that agents should not be able to distinguish between action sequences that are order indistinguishable. The following theorem shows that the information that an agent is not permitted to have in IP-security, in addition to information already forbidden to it by IP-security, is exactly the information about the orders of actions that are “swappable.”

Theorem 1 Let be \(u \in D\) and \(\alpha, \alpha' \in A^*\). Then \(\text{ta}_u(\alpha) = \text{ta}_u(\alpha')\) if and only if ipurge\(_u\)(\(\alpha\)) \(\equiv\) ipurge\(_u\)(\(\alpha'\)).

Proof: If ipurge\(_u\)(\(\alpha\)) \(\equiv\) ipurge\(_u\)(\(\alpha'\)), then \(\text{ta}_u(\alpha) = \text{ta}_u(\alpha')\) follows from Corollary 1.

For the other direction, assume that \(\text{ta}_u(\alpha) = \text{ta}_u(\alpha')\). Since ipurge\(_u\)(\(\alpha\)) \(\equiv\) ipurge\(_u\)(\(\alpha'\)) implies \(\text{ta}_u(\alpha) = \text{ta}_u(\alpha')\) (Corollary 1), we can without loss of generality assume that \(\alpha = \text{ipurge}_u(\alpha)\) and \(\alpha' = \text{ipurge}_u(\alpha')\). We also assume \(\alpha \neq \alpha'\). Note that the number of occurrences of an action \(a\) in \(\alpha\) is the same as in \(\alpha'\). Let be \(\alpha'' \in A^*\) such that \(\alpha'' \leftrightarrow_{\text{swap}} \alpha'\) and \(\alpha''\) has a common prefix with \(\alpha\) of maximal length among all \(\alpha''\) with this property. We can write \(\alpha = \beta a \beta'\) and \(\alpha'' = \beta b \beta''\) with \(\beta, \beta' , \beta'' \in A^*, a \in A\) and \(a b' \neq \beta''\). We also assume that the position of \(a\) in \(\beta''\) is the left-most position among all possible choices of \(\alpha''\). The \(\beta''\) is of the form \(\gamma b a'\gamma'\) with \(\gamma b a' \neq_{\text{swap}} \gamma a\beta\gamma'\) and \(\gamma, \gamma' \in A^*, b \in A\). Therefore there is some agent \(u' \in \text{sources}_u(\gamma)\) with \(\text{dom}(a) \rightarrow u'\) and \(\text{dom}(b) \rightarrow u'\). Therefore the tree \(\text{ta}_{u'}(\beta b a)\) is a subtree of \(\text{ta}_u(\alpha')\). Because of the corresponding number of occurrences of \(a\) in \(\alpha\), the corresponding subtree would be \(\text{ta}_{u'}(\beta a)\). But the number of occurrences of \(b\) in these two trees does not match. This contradicts the assumption that \(\alpha' \neq \alpha''\).

The characterization obtained by the above theorem is now stated in the following corollary:

Corollary 2 A system \(M\) is TA-secure if and only if it is IP-secure and for every state \(q\), every agent \(u\), and every \(a, b \in A, \alpha \in A^*\), if \(a \text{ and } b\) are swappable in \(aba\), then \(\text{obs}_u(q \cdot aba) = \text{obs}_u(q \cdot baa)\).

Proof: We first show that if a system is not TA-secure, then it violates the condition. Theorem 1 says that for all action sequences \(\alpha\) and \(\alpha'\) we have \(\text{ta}_u(\alpha) = \text{ta}_u(\alpha')\) if and only if there exist actions \(\alpha_0, \ldots, \alpha_n\) such that
For the converse, we assume that $M$ is IP-secure and not TA-secure. Then there exist traces $\alpha$ and $\alpha'$ and a state $q$ such that $\text{obs}_u(q \cdot \alpha) \neq \text{obs}_u(q \cdot \alpha')$, and $\text{ta}_u(\alpha) = \text{ta}_u(\alpha')$. Let $\alpha_0, \alpha_1, \ldots$ be the sequence of action sequences as above. Since $M$ is IP-secure, it follows that if $\alpha_i \leftrightarrow \text{irr} \alpha_{i+1}$, then $\text{obs}_u(q \cdot \alpha_i) = \text{obs}_u(q \cdot \alpha_{i+1})$. Therefore, the above implies that there are traces $\alpha_i$ and $\alpha_{i+1}$ such that $\alpha_i \leftrightarrow \text{swap} \alpha_{i+1}$ and $\text{obs}_u(q \cdot \alpha_i) \neq \text{obs}_u(q \cdot \alpha_{i+1})$ as claimed.

Now assume that the system $M$ is TA-secure. In [5], it was shown that TA-security implies IP-security. It remains to prove that $M$ satisfies the condition of the corollary. If the condition is not satisfied, then there is a state $q$, agent $u$, and actions $a, b \in A$ and $\alpha \in A^*$ such that $a$ and $b$ are swap-pable in $aba$, and $\text{obs}_u(q \cdot aba) \neq \text{obs}_u(q \cdot baa)$. Since Theorem 1 implies that $\text{ta}_u(aba) = \text{ta}_u(baa)$, it follows that the system is not TA-secure, a contradiction. □

4 Unwindings

The characterizations discussed above provide the basis for complexity results for each of the notions of security. In the case of P-security, IP-security and TA-security, our complexity results are obtained by means of an appropriate notion of unwinding. For P-security, this is the classical notion of unwinding for transitive policies, but for IP-security and TA-security, we develop, in this section, novel notions of unwinding that we show to be both sound and complete. These notions are of independent interest in that they provide proof methods that can be applied to establish security (e.g., by use of theorem provers) even when the decision procedures are inapplicable (e.g., because the system is infinite-state, or beyond the practical scope of our decision procedures.) The complexity results are given in the following section.

For the remainder of this section, we assume that all states of a system $M$ are reachable, noting that all our notions of security hold in $M$ iff they hold in the reachable part of $M$.

We first recall the notions of unwinding defined in [4]. Given a system $M$, an unwinding with respect to a policy $\rightarrow$ is a collection of equivalence relations $\sim_u$ on the set of states of $M$, for $u \in D$, satisfying the following conditions, for all $u \in D$, $a \in A$ and $s, t \in S$:

$\text{OC}_P$: If $s \sim_u t$ then $\text{obs}_u(s) = \text{obs}_u(t)$.

$\text{SC}_P$: If $s \sim_u t$ then $s \cdot a \sim_u t \cdot a$. 

20
LR\textsuperscript{IP}: If $\text{dom}(a) \notightarrow u$ then $s \sim_u s \cdot a$.

Rushby shows that, for transitive policies $\rightarrow$ (where all our notions of security coincide \[4, 5\]), a system $M$ is P-secure with respect to $\rightarrow$ iff there exists an unwinding on $M$. In fact, this result applies even for intransitive policies, with exactly the same proof.

Rushby also defines a weak unwinding with respect to a policy $\rightarrow$ to be a similar collection of relations that satisfy OC\textsuperscript{P}, LR\textsuperscript{P} and the following weaker version of SC\textsuperscript{P}:

WSC: If $s \sim_u t$ and $a \in A$ and $s \sim_{\text{dom}(a)} t$ then $s \cdot a \sim_u t \cdot a$.

It is shown in [4] that if there exists a weak unwinding on a system $M$ with respect to $\rightarrow$, then $M$ is IP-secure with respect to $\rightarrow$. Subsequently, it was shown in [5] that the conclusion can be strengthened to TA-security of $M$, and also that if $M$ is TA-secure then there exists a weak unwinding on the unfolding of $M$, a system that behaves exactly like $M$ except that it records in the system state the sequence of actions performed. However, it is not the case that there is always a weak unwinding on the original state space of a TA-secure system.

### 4.1 An Unwinding for IP-security

We now define a new notion of unwinding that we show to be sound and complete for IP-security.

We define an unwinding relation $\sim_v^u$ on the states of the system $M$ for every pair of domains $u, v \in D$. The domain $u$ takes the part of the observer. The domain $v$ corresponds to a domain whose actions are not supposed to directly affect the observations of $u$. We say that the collection of equivalence relations $\sim_v^u$ for $u, v \in D$ is an IP-unwinding on $M$ with respect to $\rightarrow$, if for all domains $u, v$, states $s, t$ of $M$, and actions $a \in A$, the following hold:

OC\textsuperscript{IP}: If $s \sim_v^u t$ then $\text{obs}_u(s) = \text{obs}_u(t)$.

SC\textsuperscript{IP}: If $s \sim_v^u t$ and $v \notightarrow \text{dom}(a)$ then $s \cdot a \sim_v^u t \cdot a$.

LR\textsuperscript{IP}: If $v \notightarrow u$ and $\text{dom}(a) = v$ then $s \sim_v^u s \cdot a$.

With this definition it is possible to give a full characterisation of IP-security.

**Theorem 2** A system $M$ is IP-secure iff there exists an IP-unwinding on $M$ with respect to $\rightarrow$.

**Proof:** We first prove the implication from left to right. Let $M$ be an IP-secure system. For $u, v \in D$, we define the following relation:

$s \sim_v^u t$ iff $\forall \alpha \in \{a \in A | v \notightarrow \text{dom}(a)\}^* : \text{obs}_u(s \cdot \alpha) = \text{obs}_u(t \cdot \alpha)$.

We show that this collection of relations is an IP-unwinding on $M$ with respect to $\rightarrow$. 

21
Obviously, $\sim_u^v$ is an equivalence relation. For any $s, t \in S$, OC$^{IP}\!$ is trivially satisfied by taking the empty string for $\alpha$. It is left to show that $\sim_u^v$ satisfies the conditions SC$^{IP}$ and LR$^{IP}\!$.

For SC$^{IP}$, let $a \in A$ with $v \not\sim dom(a)$. By definition of $s \sim_u^v t$ we have, for all $\alpha \in \{c \in A| v \not\sim dom(c)\}^+$, that $\text{obs}_u(s \cdot \alpha) = \text{obs}_u(t \cdot \alpha)$. This implies that, for all $\alpha \in \{c \in A| v \not\sim dom(c)\}^+$, we have $\text{obs}_u(s \cdot a) = \text{obs}_u(t \cdot a)$. Again by definition, this gives $s \cdot a \sim_u^v t \cdot a$.

For LR$^{IP}$ we assume that $v \not\sim u$. Let $s \in S$ and $a \in A$ with $\text{dom}(a) = v$. Since we assume that all states are reachable, there exists a sequence of actions $\beta$ such that $s = s_0 \cdot \beta$. Then by applying IP-security of $M$:

$$\forall \alpha \in \{c \in A| v \not\sim \text{dom}(c)\}^+ : \text{obs}_u(s \cdot \alpha) = \text{obs}_u(s_0 \cdot \beta \alpha) = \text{obs}_u(s_0 \cdot \text{ipurge}_u(\beta \alpha)) = \text{obs}_u(s_0 \cdot \text{ipurge}_u(\beta a \alpha)) = \text{obs}_u(s \cdot a a \alpha) = \text{obs}_u(s \cdot a \alpha) .$$

This shows that $s \sim_u^v s \cdot a$.

For the other direction of this proof we assume that $M$ is IP-insecure, and show that there does not exist an IP-unwinding. By Lemma 1 there exists a reachable state $q \in S$, $u \in D$, $a \in A$ and $\alpha \in A^*$ such that $\text{ipurge}_u(\alpha) = \text{ipurge}_u(\alpha) + \text{obs}_u(q \cdot a) \neq \text{obs}_u(q \cdot a)$ and $\text{dom}(a) \cap \text{dom}(c) \cap \text{alp}(\alpha) = \emptyset$. Set $v = \text{dom}(a)$ and let $\sim_u^v$ be an equivalence relation on $S$ that satisfies SC$^{IP}$ and LR$^{IP}$, and may exist. We derive a contradiction to OC$^{IP}$, showing that no IP-unwinding can exist. By LR$^{IP}$ we have $q \sim_u^v q \cdot a$, since $v \not\sim u$. By applying SC$^{IP}$ multiple times, we get $q \cdot a \sim_u^v q \cdot a a$. Since $\text{obs}_u(q \cdot a) \neq \text{obs}_u(q \cdot a a)$, OC$^{IP}$ is not satisfied.

Note that the equivalence relations satisfying SC$^{IP}$ and LR$^{IP}$ form a sublattice of $\mathcal{P}(S^2)$. This gives the following corollary, that will form the basis for the decision procedure for IP-security described in Section 5.2.

**Corollary 3** Let $M$ be a finite system and, for every $u, v \in D$, let $\sim_u^v$ be the smallest equivalence relation that satisfies SC$^{IP}$ and LR$^{IP}$ with respect to $\rightarrow$. Then $M$ is IP-secure with respect to $\rightarrow$ if and only if $\sim_u^v$ satisfies OC$^{IP}$ for every $u, v \in D$.

### 4.2 An unwinding for TA-security

Next, we characterize TA-security by means of a new notion of unwinding.

Define a TA-unwinding of system $M$ with respect to policy $\rightarrow$ to be a collection of equivalence relations $\sim_u^{v,w}$ on the set of states of $M$, for every $u \in D$ and every $v, w \in D$ with $v \neq w$, satisfying the following conditions, for all states $s, t \in S$ and actions $a, b \in A$:

OC$^{TA}$: If $s \sim_u^{v,w} t$ then $\text{obs}_u(s) = \text{obs}_u(t)$. 

SC\textsuperscript{TA}: If \( s \sim_{u,w}^{v} t \) and if \( a \in A \) with \( v \not\rightarrow \text{dom}(a) \) or \( w \not\rightarrow \text{dom}(a) \) then \( s \cdot a \sim_{u,w}^{v} t \cdot a \).

LR\textsuperscript{TA}: If \( \text{dom}(a) = v \) and \( \text{dom}(b) = w \) and \( v \not\rightarrow w \) and \( w \not\rightarrow v \), and either \( v \not\rightarrow u \) or \( w \not\rightarrow u \), then \( s \cdot ab \sim_{u,w}^{v} s \cdot ba \).

(If there is just one domain, then the empty collection is taken to be a TA-unwinding.) The following result characterises TA-security using this notion of unwinding.

**Theorem 3** A system \( M \) is TA-secure with respect to \( \rightarrow \) iff \( M \) is IP-secure with respect to \( \rightarrow \) and there exists a TA-unwinding of \( M \) with respect to \( \rightarrow \).

**Proof:** We first show the direction from left to right. Let system \( M \) be TA-secure with respect to \( \rightarrow \). It follows that \( M \) is IP-secure with respect to \( \rightarrow \), so it suffices to show that there exists a TA-unwinding with respect to \( \rightarrow \). For this, we define the following relation on the states of \( M \), for \( u,v,w \in D \) with \( v \not\equiv w \):

\[
\sim_{u,w}^{v} t \iff \forall \alpha \in \{ a \in A | v \not\rightarrow \text{dom}(a) or w \not\rightarrow \text{dom}(a) \}^*: \text{obs}_{u}(s \cdot \alpha) = \text{obs}_{u}(t \cdot \alpha).
\]

That the property OC\textsuperscript{TA} is satisfied is immediate from the choice of \( \alpha = \varepsilon \).

To show SC\textsuperscript{TA}, let \( s,t \in S \) with \( s \sim_{u,w}^{v} t \) and \( a \in A \) with \( v \not\rightarrow \text{dom}(a) \) or \( w \not\rightarrow \text{dom}(a) \). By definition of \( s \sim_{u,w}^{v} t \) we have for all \( \alpha \in \{ c \in A | v \not\rightarrow \text{dom}(c) or w \not\rightarrow \text{dom}(c) \}^* \), that \( \text{obs}_{u}(s \cdot \alpha) = \text{obs}_{u}(t \cdot \alpha) \). This implies that for all \( \alpha \in \{ c \in A | v \not\rightarrow \text{dom}(c) or w \not\rightarrow \text{dom}(c) \}^* \), we have \( \text{obs}_{u}(s \cdot \alpha) = \text{obs}_{u}(t \cdot \alpha) \).

Again by definition, this gives \( s \cdot a \sim_{u,w}^{v} t \cdot a \).

For LR\textsuperscript{TA} let \( a,b \in A \) with \( \text{dom}(a) = v \) and \( \text{dom}(b) = w \) and \( v \not\rightarrow w \) and \( w \not\rightarrow v \) and \( v \not\rightarrow u \) or \( w \not\rightarrow u \). Let \( s \in S \). Since we assume that all states are reachable, there exists \( \beta \in A^* \) with \( s = s_0 \cdot \beta \). For any \( \alpha \in \{ c \in A | v \not\rightarrow \text{dom}(c) or w \not\rightarrow \text{dom}(c) \}^* \) we have

\[
v \not\rightarrow c \not\rightarrow w \not\rightarrow \{ u, \text{dom}(c) | c \in \alpha \text{ph}(ab\alpha) \} = \emptyset .
\]

This shows that \( a \) and \( b \) are swappable in \( \beta ab\alpha \). By Lemma 2 we have \( \text{ta}_{u}(\beta ab\alpha) = \text{ta}_{u}(\beta ba\alpha) \). Then by applying TA-security of \( M \) we get for any such an \( \alpha \):

\[
\text{obs}_{u}(s \cdot ab\alpha) = \text{obs}_{u}(s_0 \cdot \beta ab\alpha) = \text{obs}_{u}(s_0 \cdot \beta ba\alpha) = \text{obs}_{u}(s \cdot ba\alpha)
\]

This shows that \( s \cdot ab \sim_{u,w}^{v} s \cdot ba \).

For the other direction of this proof we assume that \( M \) is ta-insecure, but IP-secure, with respect to \( \rightarrow \), and show that there does not exist a TA-unwinding on \( M \) with respect to \( \rightarrow \). From Corollary 2 it follows that there exists \( q \in S \), \( u \in D \), \( a,b \in A \), \( \alpha \in A^* \) such that \( a \) and \( b \) are swappable in \( ab\alpha \) and \( \text{obs}_{u}(q \cdot ab\alpha) \neq \text{obs}_{u}(q \cdot ba\alpha) \). It follows that \( v \not\equiv w \). We set \( v = \text{dom}(a) \) and \( w = \text{dom}(b) \). We suppose that \( \sim_{u,w}^{v} \) is an equivalence relation on \( S \) that satisfies SC\textsuperscript{TA} and LR\textsuperscript{TA}, and show that it cannot also satisfy OC\textsuperscript{TA}.

Since \( a \) and \( b \) are swappable in \( ab\alpha \) it follows directly that \( v \not\rightarrow w \), \( w \not\rightarrow v \) and \( v \not\rightarrow u \) or \( w \not\rightarrow u \). Therefore by LR\textsuperscript{TA}, we have \( q \cdot ab \sim_{u,w}^{v} q \cdot ba \). Since \( v \not\rightarrow \text{dom}(c) \)
or $w \not\rightarrow \text{dom}(c)$ for all $c \in \text{alph}(a)$, it follows by $\text{SC}^{\text{TA}}$ that $q \cdot ab\alpha \sim_{u}^{w} q \cdot ba\alpha$. Since $\text{obs}_u(q \cdot ab\alpha) \neq \text{obs}_u(q \cdot ba\alpha)$, we obtain that $\text{OC}^{\text{TA}}$ is not satisfied. \hfill $\Box$

Note that by Theorem 2, the reference to IP-security may be replaced by the existence of an IP-unwinding, giving a characterization of TA-security that is stated entirely in terms of the existence of unwinding relations.

5 Complexity

In this section, we consider the complexity of algorithmic verification of the notions we have discussed, in the case of finite state systems.

We show that three of these notions (P-security, IP-security, and TA-security) are decidable, and in fact can be decided in polynomial time, and we prove that TO-security and ITO-security are undecidable.

5.1 Complexity of P-security

The algorithm presented in Figure 9 checks P-security. The main idea of the algorithm is to use the unwinding characterization of P-security, and compute the minimal equivalence relations satisfying conditions $\text{SC}^{P}$ and $\text{LR}^{P}$ and check that these satisfy $\text{OC}^{P}$. The equivalence relations are represented as partitions of the set $S$, using the disjoint-set data structure which provides functions $\text{MAKE}\text{-SET}$, $\text{UNION}$ and $\text{FIND}$, with low amortized cost per operation [11].

The set $P$ maintains the pairs of states that result in a union step. To compute a witness in the case of insecurity, the store data structure keeps track of the justification for each union step. Every entry of $\text{store}$ consists of three pairs of the form $(s, t), (s', t'), (a, b)$. Such an entry is stored if a union is applied on $s$ and $t$ in a step, where $s$ is reached from $s'$ by an action (or the empty trace) $a$ and $t$ is reached from $t'$ by $b$. Since a union is only applied if $\text{FIND}(s) \neq \text{FIND}(t)$ and after the union $\text{FIND}(s) = \text{FIND}(t)$ holds, only at most one stored entry has $(s, t)$ as a first pair. If the union is performed on states with different observations, the system is insecure and the compute-witness procedure computes a witness for insecurity, i.e., two runs that have the same $\text{purge}_u$ value, but end in states with different observations.

To reference different values of the $\text{FIND}$ function, we parameterize it with the number of unions, done during each iteration of the outer foreach-loop, i.e., for a fixed value of domain $u$. The function $\text{FIND}_i$, for $i \geq 0$, denotes the values of $\text{FIND}$ after the $i$-th application of union during one iteration of the outer foreach-loop. We call $i$ a union-number (of this iteration).

Similarly, $P_i$ denotes the set of pairs in $P$ after the $i$-th application of union. Note that $P$ maintains this value until the line just before the next application of union. The set $P_{\leq i} = \bigcup_{j \leq i} P_j$ is the set of pairs inserted into $P$ up to the $i$-th union-number, including the pairs that are removed from $P$. Since $P_{\leq i}$ is a set of pairs of states, we will consider it as a binary relation on $S$. 

24
Algorithm 1: Decide P-security

```plaintext
1 foreach u ∈ D do /* create a new partition */
2     foreach s ∈ S do
3         Make-Set(s);
4     let P be an empty list;
5     let store be empty;
6     /* apply LR conditions */
7     foreach s ∈ S do
8         foreach a ∈ A with dom(a) ̸↣ u do
9             if Find(s) ̸= Find(s·a) then
10                add ((s·a, s), (s, s), (a, ϵ)) to store;
11                insert (s·a, s) into the list P;
12                UNION(s·a, s);
13                if obsu(s·a) ̸= obsu(s) then
14                    return compute-witness(s·a, s, ϵ, ϵ);
15     /* apply SC conditions */
16     while P ̸= ∅ do
17         take a pair (s, t) out of P;
18         foreach a ∈ A do
19             if FIND(s·a) ̸= FIND(t·a) then
20                 add ((s·a, t·a), (s, t), (a, a)) to store;
21                 insert (s·a, t·a) into the list P;
22                 UNION(s·a, t·a);
23                 if obsu(s·a) ̸= obsu(t·a) then
24                     return compute-witness(s·a, t·a, ϵ, ϵ);
25     return "secure";
```

Figure 9: An algorithm for P-security.

Procedure compute-witness(s, t, α, β)

```plaintext
1 if s = t then
2     find a shortest path γ from s₀ to s;
3     return (γα, γβ);
4 else
5     choose stored entry ((s, t), (s', t'), (a, b)) ;
6     compute-witness(s', t', aα, bβ);
```
Define for all $s,t \in S$

$$s \sim_{\text{obs}} t \text{ iff } \text{obs}_u(s) = \text{obs}_u(t).$$

To refer to stages in the construction of the equivalence relation, define for all union-numbers $i$ and for all $s,t \in S$

$$s \sim_{\text{FIND}}^i t \text{ iff } \text{FIND}_i(s) = \text{FIND}_i(t)$$

$$s \sim_{\text{SC}}^i t \text{ iff } s \sim_{\text{FIND}}^i t \text{ and } s \cdot a \sim_{\text{FIND}}^i t \cdot a \text{ for all } a \in A$$

Note that both relations are equivalence relations on $S$. Note also that these relations are monotone in $i$, i.e., if $s \sim_{\text{FIND}}^i t$ then $s \sim_{\text{FIND}}^{i+1} t$ and if $s \sim_{\text{SC}}^i t$ then $s \sim_{\text{SC}}^{i+1} t$.

To show correctness of the algorithm, we will show that is sufficient to guarantee $\text{SC}^n$ and $\text{OC}^n$ for the pairs of states collected in $P$.

**Lemma 3** Fix an iteration of the outer foreach-loop, and let $i$ be a union-number of this iteration. Then $\sim_{\text{FIND}}^i$ is the smallest equivalence relation on $S$ that includes $P_{\leq i}$.

**Proof:** By induction on $i$. For $i = 0$ we have $P_i$ empty, so the smallest equivalence relation is the identity relation, which is also the relation corresponding to the initial value of $\text{FIND}_i$. At each union, the pairs inserted into $P$ are exactly those that are used by the union. Therefore, $\sim_{\text{FIND}}^i$ is the reflexive, symmetric, transitive closure of $P_{\leq i}$. Hence, $\sim_{\text{FIND}}^i$ is the smallest equivalence relation on $S$ that includes $P_{\leq i}$. \hfill $\square$

First, we show that the witness produced by the algorithm is correct.

**Lemma 4** If the algorithm terminates with a witness $(\alpha, \beta)$, the analyzed system is $P$-insecure and we have $\text{purge}_u(\alpha) = \text{purge}_u(\beta)$ and $\text{obs}_u(s_0 \cdot \alpha) \neq \text{obs}_u(s_0 \cdot \beta)$, where $u$ is the agent chosen by the outer foreach-loop in the iteration where the compute-witness procedure is called.

**Proof:** The procedure compute-witness is called with two states as parameter having different observations for an agent $u$. From these states, the compute-witness procedure constructs two paths back to $s_0$. From the stored values, it follows that these paths only differ in actions $a$ with $\text{dom}(a) \neq u$. Therefore, we have $\text{purge}_u(\alpha) = \text{purge}_u(\beta)$. Hence the constructed paths $\alpha$ and $\beta$ are a witness for the insecurity of the system. \hfill $\square$

The correctness of the algorithm follows from this Lemma, showing that the converse holds, too.

**Lemma 5** If the algorithm terminates with “secure”, the analyzed system is $P$-secure.

26
Proof: Consider one iteration of the outer foreach-loop, where some agent \( u \in D \) is chosen. Let \( m \) be the last union-number after this iteration of the outer foreach-loop. We will show that the unwinding conditions LR\(^P\), SC\(^P\) and OC\(^P\) hold for the LR\(^P\). It is clear that after the foreach-loop starting in line 6 the LR\(^P\) condition holds for \( \sim^{\text{Find}}_m \), since for every \( s \in S \) and every \( a \in A \) with \( \text{dom}(a) \not\rightarrow u \), we have \( s \sim^{\text{Find}}_m s \cdot a \).

After each union step, it is checked that the observations are equal for the two merged sets. Assuming inductively that the observations are all equal on each of the two merged set of states, it follows that the observations are equal on the merged set. Therefore, for every \( i \leq m \) we have \( \sim_i^{\text{Find}} \subseteq \sim_{\text{obs}} \). Moreover, the relation \( \sim_m^{\text{Find}} \) satisfies OC\(^P\).

After this iteration of the outer foreach-loop, \( P_m \) is empty and therefore, we have for every \( s,t \in S \) that if \( (s,t) \in P_{\leq m} \), then \( s \sim^{\text{Find}}_m t \) and \( s \cdot a \sim^{\text{Find}}_m t \cdot a \) for all actions \( a \in A \). This is guaranteed by the foreach-loop in line 16 for every pair taken out of \( P \) in the line above. Therefore \( P_{\leq m} \subseteq \sim^S \). Since \( \sim^S \) is an equivalence relation and since \( \sim^{\text{Find}}_m \) is the smallest equivalence relation that contains \( P_{\leq m} \), it follows that \( \sim_m^{\text{Find}} \subseteq \sim^S_m \). Therefore, \( \sim_m^{\text{Find}} \) satisfies SC\(^P\). □

The following Lemma shows the correctness of the compute-witness procedure and gives a bound for its running time.

Lemma 6 The procedure compute-witness computes a witness in \( O(|S| \cdot |A|) \).

Proof: First, we will show that the graph induced by the stored values is a directed rooted tree. For every stored entry \( e \) of the form \( e=((s,t),(s',t'),(a,b)) \) consider the projections \( \pi_0(e) = (s,t) \), \( \pi_1(e) = (s',t') \) and \( \pi_2(e) = (a,b) \). For every union-number \( i \), the graph of the stored values is \( G_i = (V_i,E_i) \) with \( V_i = \{ \pi_0(e), \pi_1(e) \mid e \text{ is a stored entry up to the } i\text{-the union number} \} \) and \( E_i = \{ (\pi_0(e),\pi_1(e)) \mid e \text{ is a stored entry up to the } i\text{-the union number} \} \). We will show by an induction on \( i \), that the connected components of \( G_i \) are directed rooted trees where all edges are oriented towards some root and all roots are of the form \( (s,s) \) and that \( P_{\leq i} \subseteq V_i \subseteq P_{\leq i} \cup \{(s,s) \mid s \in S \} \) holds. In the iterations of the loop for the LR-conditions, only edges of the form \(((s,t),(s',t'))\) with \( s \neq t \) are inserted. Therefore the resulting graph consists of directed rooted trees. In the iterations of the loop for the SC-conditions, if an edge of the form \( e = ((s,t),(s',t')) \) is inserted into \( G_i \), then \( \text{Find}_i(s) \neq \text{Find}_i(t) \) and therefore \( (s,t) \not\in P_{\leq i} \cup \{(s,s) \mid s \in S \} \) and by induction hypothesis, \( (s,t) \not\in V_i \). Since \( (s',t') \in P_{\leq i} \), the edge \( e \) connects a new vertex with a vertex from \( V_i \). Therefore, the resulting graph \( G_{i+1} \) is again a directed rooted tree.

It follows that if the procedure compute-witness is called with some states \( (s,t) \in P_{\leq i} \) then it finds a path to the unique root \((s',t')\) of the connected component of \((s,t)\) within \( O(|S|) \). A shortest path from \( s' \) to \( s_0 \) can be found in \( O(|S| \cdot |A|) \).

The running time of the whole algorithm can be analyzed as follows. Since the union is only applied on states in different sets, for each \( u \in D \), the total
number of unions is bounded by $|S|$. Note, that the insertion into $P$ is only combined with a union step, therefore the sets of pairs inserted into $P$ during the whole run of the algorithm is bounded by $|S|$, too. Also, the number of stored triple of pairs is the same as the number of unions, since for every pair $(s,t)$ the union is only applied at most once. The stored values provide a function from the first to the second and third pair. The union-find operations can be implemented so as to have an amortized cost of $\alpha(|S|)$, where $\alpha$ is the very slow growing (effectively constant for practical purposes) “inverse” of Ackermann’s function [11]. Thus, the running time of this algorithm is bounded by $O(|D| \cdot |A| \cdot |S| \cdot \alpha(|S|))$.

5.2 Complexity of IP-security

An approach similar to that for P-security works in the case of IP-security, based on the unwinding characterization of IP-security. The algorithm is given in Figure 10.

The argument for correctness is similar to that for the algorithm for P-security: the algorithm computes the minimal equivalence relations satisfying $\text{SC}_{\text{IP}}$ and $\text{LR}_{\text{IP}}$, and checks that these satisfy $\text{OC}_{\text{IP}}$. The argument for the SC conditions is similar to that of Lemma 4 and 5. The running time of the algorithm for checking IP-security is $O(|D|^2 \cdot |A| \cdot |S| \cdot \alpha(|S|))$.

5.3 Complexity of TA-security

The argument for correctness of the algorithm of Figure 11 is similar to that for the algorithm for P-security: the algorithm computes the minimal equivalence relations satisfying $\text{SC}_{\text{TA}}$ and $\text{LR}_{\text{TA}}$, and checks that these satisfy $\text{OC}_{\text{TA}}$. The argument for the SC conditions is similar to that of Lemma 4 and 5. The running time is $O(|D|^3 \cdot |A| \cdot |S| \cdot \alpha(|S|))$.

5.4 Space Complexity and Symbolic Implementation for P-security, IP-security and TA-security

The algorithms presented above for P-security, IP-security and TA-security all require space linear in the size of the number of states of the system. Due the “state space explosion” problem, i.e., the fact that the number of states of a system grows exponentially with the number of state variables, the number of states may be very large in practical examples. For each of these notions of security, it is possible to trade off space for time, at the cost of introducing nondeterminism. Instead of computing an explicit representation of the minimal unwinding relation $\sim$ of the appropriate type, we search through a graph in which the vertices are pairs of states $(s,t)$ such that $s \sim t$, and in which edges correspond to one of the unwinding rules of type LR or SC. The search begins at a vertex $(s,s)$, where $s$ is reachable state, and terminates and declares “insecure” if a pair $(s,t)$ is reached such that $\text{obs}_s(s) \neq \text{obs}_s(t)$. The search can be conducted using nondeterminism and terminated at depth $|S|^2$ with a
Algorithm 2: Decide IP-security

1 foreach $u \in D$ do
2   /* create a new partition */
3   foreach $v \in D$ with $v \not\leftrightarrow u$ do
4     foreach $s \in S$ do
5       MAKE-SET($s$);
6       let $P$ be an empty list;
7       let $store$ be empty;
8     /* apply LR conditions */
9     foreach $s \in S$ do
10    if $\text{Find}(s) \neq \text{Find}(s \cdot a)$ then
11       add $((s \cdot a, s), (s, s), (a, \epsilon))$ to $store$;
12       UNION($s \cdot a, s$);
13       if $\text{obs}_u(s \cdot a) \neq \text{obs}_u(s)$ then
14          return compute-witness($s \cdot a, s, \epsilon, \epsilon$);
15    /* apply SC conditions */
16    while $P \neq \emptyset$ do
17      take a pair $(s, t)$ out of $P$;
18      foreach $a \in A$ with $\text{dom}(a) = v$ do
19        if $\text{Find}(s \cdot a) \neq \text{Find}(t \cdot a)$ then
20          add $((s \cdot a, t \cdot a), (s, t), (a, a))$ to $store$;
21          insert $(s \cdot a, t \cdot a)$ into the list $P$;
22          UNION($s \cdot a, t \cdot a$);
23        if $\text{obs}_u(s \cdot a) \neq \text{obs}_u(t \cdot a)$ then
24          return compute-witness($s \cdot a, t \cdot a, \epsilon, \epsilon$);
25      endforeach
26    endwhile
27    return "secure"

Figure 10: An algorithm for IP-security.
Algorithm 3: Decide TA-security

1 Run the Algorithm for IP-security: if it returns "secure", then
2 foreach \( u \in D \) do
3     foreach \( v \in D \) do
4         foreach \( w \in D \) with \( w \not\rightarrow v \) and \( v \not\rightarrow w \) and \( w \not\rightarrow u \) do
5             /* create a new partition */
6                 foreach \( s \in S \) do
7                     Make-Set(\( s \)) ;
8                     let \( P \) be an empty list ;
9                     let \( store \) be empty ;
10                /* apply LR conditions */
11                    foreach \( s \in S \) do
12                        foreach \( a \in A \) with \( \text{dom}(a) = v \) do
13                            foreach \( b \in A \) with \( \text{dom}(b) = w \) do
14                                if \( \text{Find}(s \cdot ab) \neq \text{Find}(s \cdot ba) \) then
15                                    add \((s \cdot ab, s \cdot ba, (s, s), (ab, ba))\) to \( store \);
16                                    insert \((s \cdot ab, s \cdot ba)\) into the list \( P \) ;
17                                    if \( \text{obs}_u(s \cdot ab) \neq \text{obs}_u(s \cdot ba) \) then
18                                        return compute-witness\((s \cdot ab, s \cdot ba, \epsilon, \epsilon)\) ;
19                    /* apply SC conditions */
20                        while \( P \neq \emptyset \) do
21                            take a pair \((s, t)\) out of \( P \) ;
22                            foreach \( a \in A \) with \( v \not\rightarrow \text{dom}(a) \) or \( w \not\rightarrow \text{dom}(a) \) do
23                                if \( \text{Find}(s \cdot a) \neq \text{Find}(t \cdot a) \) then
24                                    add \((s \cdot a, t \cdot a, (s, t), (a, a))\) to \( store \);
25                                    insert \((s \cdot a, t \cdot a)\) into the list \( P \) ;
26                                    UNION\((s \cdot a, t \cdot a)\) ;
27                                    if \( \text{obs}_u(s \cdot a) \neq \text{obs}_u(t \cdot a) \) then
28                                        return compute-witness\((s \cdot a, t \cdot a, \epsilon, \epsilon)\) ;
29                        return "secure"
30
Figure 11: An algorithm for TA-security.
declaration of “secure” in the case that no such pair is found. This gives a co-NLOGSPACE = NLOGSPACE algorithm. Since graph search trivially reduces to these problems, verification of all three security notions is complete for nondeterministic logarithmic space.

Both the algorithms given above and the nondeterministic logarithmic space approach rely on explicit representation of states. We note that, in practice, an effective approach to deciding the three properties may be to instead use symbolic representations of states and relations to represent the fixpoint computation for the minimal relation on reachable states satisfying the SC and LR type rules, following techniques well known from the model checking area [12], and then to intersect with a symbolic representation of the complement of the OC type rule and check for non-emptiness. The performance of this approach is unpredictable in general, so comparison with the algorithms above is a matter for experimental research.

5.5 TO-security

We now prove that TO-security is undecidable. The proof also shows that the source of the undecidability does not lie in using complex policies, in fact the problem remains undecidable for a very simple, small policy.

**Theorem 4** It is undecidable whether $M$ is TO-secure with respect to $\rightarrow$, even for a fixed policy containing 4 domains.

**Proof:** We prove the undecidability of TO-security by a reduction from the Post Correspondence Problem [13]. An instance of this problem consists of a pair of sequences $U = U_1, \ldots, U_n$ and $W = W_1, \ldots, W_n$ of words over an alphabet $\Sigma$ with at least two letters. The problem PCP is the set of such pairs $(U, W)$ such that there exists a sequence of indices $i_1, \ldots, i_k$ with $1 \leq i_j \leq n$ for each $j = 1, \ldots, k$, such that $U_{i_1} \ldots U_{i_k} = W_{i_1} \ldots W_{i_k}$. We encode an instance of this problem as a machine $M(U, W)$ for the (intransitive) policy for agents $A, B, C, D$ given by $A \rightarrow C$, $A \rightarrow D$, $B \rightarrow C$ and $C \rightarrow D$, such that $(U, W) \in PCP$ iff $M(U, W)$ is not TO-secure with respect to $\rightarrow$.

Intuitively, in the machine $M(U, W)$, agent $A$ guesses a word over $\Sigma$, and agent $B$ chooses whether this word is to be compared to a sequence of $U_i$ or $W_i$, and guesses a sequence of indices used to make the comparison. Agent $C$ observes the indices guessed by $B$, and guesses when the word being constructed is complete. Agent $D$ observes nothing until $C$ declares the end of the construction, and then observes whether the word guessed by $A$ does in fact correspond to the sequence of indices guessed by $B$. The definition of TO-secure will be guaranteed to hold with respect to agents $A, B$ and $C$, so the determination as to whether $M(U, W)$ is TO-secure depends on how the observations of agent $D$ relate to the actions and observations of $A$ and $C$. More precisely, $M(U, W)$ has

1. states of the form $(p, V, i, x)$, where
(a) $p \in \{U,U',W\}$ indicates whether the sequence of letters guessed by $A$ is to be compared with a sequence of $U_i$ (when $p \in \{U,U'\}$) or as a sequence of $W_i$ (when $p = W$).

(b) $V$ is either a word over $\Sigma$ which is a prefix (possibly the empty word $\epsilon$) of one of the $U_i$ or $W_i$, or $\top$. Intuitively, this indicates a part of the word guessed by $A$ that will be compared to an index guessed by $B$.

The case of $\top$ represents that an inconsistency has been detected.

(c) $i \in \{0,\ldots,n\}$ is either 0 (no activity so far) or the last index guessed by $B$.

(d) $x \in \{0,1\}$ is used to represent the state of the computation, with 0 meaning ongoing and 1 meaning complete.

2. initial state $(U,\epsilon,0,0)$,

3. actions

(a) of $A$: an action $a$ for each $a \in \Sigma$, corresponding to guessing the letter $a$

(b) of $B$: an action $w$ (corresponding to the selection of $W$) plus an action $g_i$ for each $i \in \{1,\ldots,n\}$ (corresponding to a guess of the index $i$)

(c) of $C$: an action $\text{end}$

(d) of $D$: none

The transition function is defined as follows. For all actions $b$ and states $s = (p,V,i,x)$, if $x = 1$ then we will have $\text{step}(s,b) = s$, i.e., once the computation has terminated, no action changes the state. We therefore confine the definitions below to the case $x = 0$. We make use of two functions $G: \{U,U',W\} \rightarrow \{U,W\}$ defined by $G(U) = G(U') = U$ and $G(W) = W$, and $F: \{U,U',W\} \rightarrow \{U',W\}$ defined by $F(U) = F(U') = U'$ and $F(W) = W$.

In a state $(p,V,i,x)$, the value $G(p)$ captures the choice of $U$ or $W$ with which to compare the word being generated by $A$. Intuitively, both $p = U$ and $p = U'$ represent that the word being processed is to be compared with the $U$. This the default, as indicated in the initial state. The reason for including $U'$ is that agent $B$ is given an opportunity to switch the system to comparing with $W$ only in the first step of a run. If it does not act, then the choice remains with $U$ for the remainder of the run.

In the case of action $w$, we define $\text{step}((p,V,i,0),w) = (W,V',i,0)$ if $p = U$ and $\text{step}((p,V,i,0),w) = (p,V,i,0)$ otherwise. This says that $w$ switches the choice of comparison to $W$. That the choice can be made only if $w$ is the initial action of a run is captured by defining all other actions $b \neq w$ so that if $\text{step}((p,V,i,0),b) = (p',V',i',x')$ then $p' = F(p)$.

For the actions $a$ of $A$, we define $\text{step}((p,V,i,x),a) = (F(p),V',i,x)$, where $V' = V \cdot a$ if $V \cdot a$ is a prefix of $G(p)_j$ for some $j$, and $V' = \top$ otherwise. Intuitively, $V$ is used to collect a fragment of the sequence being generated by $A$ for comparison with the $G(p)_j$. We accumulate the fragment while it is a prefix of such a string, and as soon as this is no longer the case we flag the

\footnote{Throughout, we use $\bot$ to represent undetermined information and $\top$ to represent inconsistency.}
inconsistency.

For the actions $g_j$ of $B$, we define $\text{step}\big((p, V, i, 0), g_j\big) = (F(p), V', j, 0)$, where

1. if $G(p)_j = V$ then $V' = \epsilon$, and
2. if $G(p)_j \neq V$ then $V' = \top$.

Intuitively, this captures that the effect of the action $g_j$ is to compare $G(p)_j$ with the current fragment of the string being generated by $A$. If they are equal, we reset $V$ to $\epsilon$ in order to check the next fragment. Otherwise, we flag the inconsistency.

For the action end of $C$, we define $\text{step}\big((p, V, i, 0), \text{end}\big) = (F(p), V', i, 1)$, where

1. if $V = \epsilon$ then $V' = \epsilon$, and
2. if $V \neq \epsilon$ then $V' = \top$.

Intuitively, this action checks that the end is declared at a time when there is no fragment currently being processed, and flags an inconsistency otherwise.

The observations are now defined as follows. The observations of $A$ and $B$ are trivial: $\text{obs}_A(s) = \text{obs}_B(s) = \bot$ for all states $s$. For $C$, we define

$$\text{obs}_C((p, V, i, x)) = \begin{cases} 
i & \text{if } V = \epsilon \\
\top & \text{if } V = \top \\
\bot & \text{otherwise.} \end{cases}$$

Note that this means that for the initial state $s_0$ we have $\text{obs}_C(s_0) = 0$. Intuitively, since $i$ records the last (successful) guess of index for a fragment of the word being generated by $A$, we have that $C$ becomes aware of a guess whenever it is correct, and can see from its observation $\bot$ that a further fragment is in the process of being constructed.

For $D$ we define

$$\text{obs}_D((p, V, i, x)) = \begin{cases} 
\bot & \text{when } x = 0, \\
G(p) & \text{when } x = 1, \text{ and } V = \epsilon \text{ and } i \neq 0 \\
\top & \text{when } x = 1 \text{ and either } V \neq \epsilon \text{ or } i = 0. \end{cases}$$

Intuitively, this means that $D$ observes $\bot$ until $C$ declares the end of the string, and learns whether $U$ or $V$ fragments were being checked when a decomposition has been successfully guessed. Otherwise, it learns that the guesses do not match. This completes the definition of the system $M(U, W)$.

We leave it to the reader to check that the conditions for TO-security are satisfied in $M(U, W)$ for the agents $u = A, B, C$. For $u = D$, we claim that the definition is violated iff there exists a sequence of indices $i_1, \ldots, i_k$ such that $U_{i_1} \ldots U_{i_k} = W_{i_1} \ldots W_{i_k}$. For, suppose that such a sequence exists. Define

$$\alpha = U_{i_1}g_{i_1} \ldots g_{i_k-1}U_{i_k}g_{i_k}\text{end} \quad (1)$$

and

$$\alpha' = wW_{i_1}g_{i_1} \ldots g_{i_k-1}W_{i_k}g_{i_k}\text{end}. \quad (2)$$
Then we have $\text{purge}_D(\alpha) = U_{t_1} \ldots U_{t_k}\text{end} = W_{t_1} \ldots W_{t_k}\text{end} = \text{purge}_D(\alpha')$.

Moreover, $v \mapsto D$ and $v \neq D$ if $v = A$ or $v = C$. If $U_{t_1} \ldots U_{t_k}$ is the sequence of letters $a_1 a_2 \ldots a_m$, then $\text{view}_A(\alpha) = \perp a_1 \perp a_2 \ldots \perp a_m = \text{view}_A(\alpha')$. Also, we have $\text{view}_C(\alpha) = 0 \perp i_1 \perp i_2 \ldots i_k \perp \text{end} = \text{view}_C(\alpha)$. However, $s_0 \cdot \alpha = (U', \epsilon, i_k, 1)$ and $s_0 \cdot \alpha' = (W, \epsilon, i_k, 1)$, so $\text{obs}_D(s_0 \cdot \alpha) = U \neq W = \text{obs}_D(s_0 \cdot \alpha')$. Thus the condition for TO-security is violated.

Conversely, suppose that the condition for TO-security is violated. As noted above, the violation can only occur for agent $D$. Let the witness be the sequences of actions $\alpha, \alpha'$. Then we have $\text{purge}_D(\alpha) = \text{purge}_D(\alpha')$ and $\text{view}_A(\alpha) = \text{view}_A(\alpha')$ and $\text{view}_C(\alpha) = \text{view}_C(\alpha')$ and $\text{obs}_D(s_0 \cdot \alpha) \neq \text{obs}_D(s_0 \cdot \alpha')$.

The action $\text{end}$ must occur in at least one of these sequences, else we have $\text{obs}_D(s_0 \cdot \alpha) = \text{obs}_D(s_0 \cdot \alpha') = \perp$, since only the action $\text{end}$ can set $x = 1$. Since $\text{view}_C(\alpha) = \text{view}_C(\alpha')$, and the action $\text{end}$ of $C$ is recorded in its possibly transmitted observations, it follows that $\text{end}$ in fact occurs in both sequences. Since no action changes the state after the occurrence of $\text{end}$, we may assume without loss of generality that $\text{end}$ is the final action in both sequences. (Deleting any subsequent actions preserves the observations. Any actions of $A$ or $C$ must occur in both, so their deletion maintains equality of $\text{purge}_D$ and $\text{view}_A$ and and $\text{view}_C$. All other actions can be deleted without change to $\text{purge}_D, \text{view}_A$ or $\text{view}_C$.) Similarly, we may also assume that any occurrence of $w$ must be as the initial action in the sequence, since otherwise this action changes nothing.

Since both sequences contain $\text{end}$ we do not have $\text{obs}_D(s_0 \cdot \alpha) = \perp$. Let $s_0 \cdot \alpha = (p, V, i, 1)$ and suppose that $\text{obs}_D(s_0 \cdot \alpha) = \top$. Then we have $V \neq \epsilon$ or $i = 0$. There are several possibilities. We show that each implies $\text{obs}_D(s_0 \cdot \alpha') = \top$, yielding a contradiction with $\text{obs}_D(s_0 \cdot \alpha) \neq \text{obs}_D(s_0 \cdot \alpha')$. The different cases are:

1. If $i = 0$ then $\alpha$ contains no action $g_j$, so $\text{view}_C(\alpha)$ is $0\text{end}$ or $0\perp\text{end}$. Hence $\text{view}_C(\alpha')$ also has this form, from which it follows that $\alpha'$ also contains no action $g_j$, so $\text{obs}_D(s_0 \cdot \alpha') = \top$.

2. If $V = \top$ then at some stage of the computation either the current fragment ceased to be a prefix of any $G(p)_j$, or $B$ guessed an index not matching the current fragment. In either case, $\top$ must occur in $\text{view}_C(\alpha)$, hence in $\text{view}_C(\alpha')$ also. This implies, by the definitions of $\text{obs}_C$ and $\text{obs}_D$ and the fact that once $V$ becomes $\top$ it remains $\top$, that $\text{obs}_D(s_0 \cdot \alpha') = \top$.

3. Alternatively, if $V \in \Sigma^* \setminus \epsilon$, then $\text{view}_C(\alpha)$ ends with $\perp\text{end}$, hence also $\text{view}_C(\alpha')$ ends with $\perp\text{end}$. This means that in $\alpha'$ a fragment was under construction when $\text{end}$ occurred, hence if $s_0 \cdot \alpha' = (p', V', i', 1)$ then we have $V' \neq \epsilon$, and hence $\text{obs}_D(s_0 \cdot \alpha') = \top$

It therefore follows that $\text{obs}_D(s_0 \cdot \alpha)$ and $\text{obs}_D(s_0 \cdot \alpha')$ take (since they differ) the values $U$ and $W$. The sequences of the $g_j$ in $\alpha$ must be the same, since no errors were detected in either computation, so for each occurrence $g_j$ we have that $j$ appears in $\text{view}_C$. The last action before $\text{end}$ in these sequences must
be an action $g_j$, else we end in a state $(p, V, i, 1)$ with $V \neq \epsilon$ or $i = 0$. It follows that $\alpha$ and $\alpha'$ have the forms in equations (1) and (2), and that $(U, W) \in PCP$.

□

We note that the undecidability result for TO-security implies that there are no simple unwinding conditions that are complete for this notion of security. In particular, any first-order set of conditions on a collection of binary relations on states can be checked in PTIME, hence cannot be both sound and complete.

5.6 ITO-security

A proof for the undecidability of ITO-security could be given that is similar to that for TO-security. However, the result can also be obtained by noting that the similarity of the two definitions allows for a reduction from TO-security to ITO-security. The details of the reduction are given in the appendix. The following result is then immediate from Theorem 4.

**Theorem 5** It is undecidable to determine whether $M$ is ITO-secure with respect to $\rightarrow$, even for a fixed policy with 4 domains.

6 Related Work

The notion of noninterference was first proposed by Goguen and Meseguer [2]. Early work in this area was motivated by multi-level secure systems, and dealt with partially ordered (hence transitive) information flow policies. The simplest of these is the two-domain policy with domains $L$ and $H$ and $L \rightarrow H$, but not $H \rightarrow L$. Much of the literature is confined to this simple policy. Even with this restriction, there exists a large set of proposed definitions of noninterference [14, 15, 16, 17, 18]. These definitions generally agree when applied to deterministic systems, and the differences relate to how the definitions should behave on nondeterministic systems. In addition to state-observed systems model used in the present paper, a variety of systems models have been considered, including action-observed systems, trace semantics, and process algebraic semantics (both CCS and CSP flavours). A number of works have sought to classify the definitions when formulated in a very general process algebraic setting [17], as well as to establish formal relations between definitions cast in different semantic models [9].

The main point of overlap of our work with this literature is to consider how our results concerning P-security, when applied to transitive policies, relate to other algorithmic verification approaches in the literature for such policies. Our approach here is similar to other work in the area. In particular, the idea of running two copies of the system in parallel, in order to compare two different runs, has been used before [19]. Other approaches have been developed for automated verification of noninterference based on process algebraic bisimulation techniques [17, 20]. Mantel [21] has characterised many of the existing definitions of noninterference as compositions of a set of Basic Security Properties.
The complexity of verifying these basic properties has been studied [22]. A few works have considered richer systems models than finite state systems, e.g. pushdown systems [23]. We note that our results in this paper, and in much of the literature, is concerned with asynchronous systems in which agents are unaware of the passage of time. Some of the literature deals with synchronous systems, where a similar spectrum of definitions of noninterference exists for nondeterministic systems. Some recent work has investigated verification of synchronous notions of noninterference [24, 25].

Some work on development of tools based on decidable cases of such definitions of noninterference has been performed. Focardi et al. describe a tool based on process algebraic techniques [26]. Whalen et al. [27] present an approach to model checking noninterference that is in use at Rockwell Collins for verification of MILS systems. Their approach is a mix of model checking and static analysis, in which a model checker is used to search through an enriched version of the model in which labels computed by static analysis are associated to systems components. They formally prove it to be sound with respect to a definition of noninterference from work by Greve, Wilding and van Fleet [28]. While they discuss examples requiring intransitive policies, they leave details of this for future work.

Since we have confined ourselves in this paper to deterministic systems, but focus on richer intransitive policies, much of the work discussed above, which is confined to transitive policies and nondeterministic systems, is orthogonal to our concerns. Algorithmic verification of intransitive noninterference has had less attention in the literature. After the work of Rushby [4], IP-security has generally been taken to be the definition studied.

Pinsky [29] presents a PTIME procedure for deciding IP-security that, in effect, generates a relation that is claimed to satisfy Rushby’s unwinding conditions for transitive noninterference just when the system is secure. However, in fact the relation may fail to satisfy the Output Consistency condition, so this claim is incorrect. (Pinsky’s argument supporting the claim that the relation satisfies Output Consistency, in the corollary to Theorem 2, states that $SA(basis,\pi(z),\alpha)$ is a subset of $view(state,action(z,\alpha))$. This is correct for transitive policies, but could be false for intransitive policies.) That such an approach cannot work for IP-security also follows from results in [5], where it is shown that Rushby’s unwinding conditions are sound also for TA-security, which is a stronger notion than IP-security. Moreover, an example in [5] shows that a system may be TA-secure, but no Rushby unwinding exists on the system (although one will exist on the infinite state unfolded system, when the system satisfies TA-security). Thus, an approach based on finding a Rushby unwinding, on the system as given, will also fail to be complete for the notion of TA-security.

Hadj-Alouane et al. [30] also present a decision procedure for IP-security, but it has complexity $O(2^{1/2}\cdot 2^{1/2})$, which is less efficient than our procedure by two exponentials.

Roscoe and Goldsmith [10] have presented a critique of IP-security (arguing
that it is too liberal in the information flows it permits), and have proposed two alternate definitions cast in the process algebra CSP, based on what they call lazy and mixed abstraction operators. It has been shown by van der Meyden \[31\] that the definition based on lazy abstraction corresponds to P-security, and the version based on mixed abstraction corresponds to ITO-security. Roscoe and Goldsmith give an informal discussion, without precise complexity results or proof, of algorithms for deciding “the generalised noninterference condition”. Based on van der Meyden’s characterization of the definition based on the mixed abstraction as as corresponding to ITO-security, we obtain that the definition based on mixed abstraction is undecidable. It therefore seems that their comments should be interpreted as concerned (like most of the preceding content in their paper) just with the lazy abstraction based definition, and hence comparable to our PTIME result for P-security.

Unwinding was introduced in \[6\] and given a crisp presentation in \[4\], for both transitive and intransitive security policies. In particular, Rushy shows that a notion of unwinding for transitive policies is both sound (if an unwinding exists, then the system is secure) and complete (if the system is secure, then there exists an unwinding). He also defines a notion of weak unwinding tailored to intransitive security policies, and proves its soundness for IP-security, but not completeness. The reason for this gap was identified in \[5\]: weak unwinding is also sound for the strictly stronger notion of TA-security, so cannot be complete for IP-security.

In a context of systems with local policies, sound and complete unwindings are given in \[32\]. The unwindings for IP-security in this work are a special cases of the unwindings given there.

Van der Meyden also shows that weak unwinding is complete for TA-security in the sense that if a system \(M\) is TA-secure, then there exists a weak unwinding on the system \(uf(M)\) obtained by unfolding \(M\). The system \(uf(M)\) is essentially the same as \(M\), but with a new component added to states that records the complete history of the system. The system \(uf(M)\) is bisimilar to \(M\) in an appropriate sense of bisimilarity. It is generally held that two bisimilar systems are equivalent with respect to all properties of interest. However, somewhat unusually, the existence of a weak unwinding is not preserved by bisimilarity. This means that, while unfolding and then searching for a weak unwinding is a complete proof technique for TA-security, it falls short of providing a set of unwinding conditions on the system as given. This is significant, in that the existence of a first-order expressible set of such conditions would also yield a decision procedure for TA-security of finite state systems. (The procedure would be of at most nondeterministic polynomial-time complexity, involving guessing an unwinding and verifying its properties.) By contrast, the process of unfolding turns a finite state system into an infinite-state system, so a proof technique that relies upon it does not trivially yield a decision procedure. Indeed, the decidability of TA-security was an open problem until the present work.

In practice, definitions of the kind we have studied are very liberal in the information flows that they permit: when a (source) domain acts, everything that it knows (in some sense of knowledge) may be transmitted to any domain.
with which the source is permitted to interfere. In practice, one generally wants to limit the information that flows from one domain to another to be just a subset of the information available to the source. A framework for such policies that generalizes the definitions studied in the present paper has been developed by van der Meyden and Chong [33, 34].

Approaches to stating policies expressing such limitations have also been developed in the context of language-based approaches to security, where they are generally supported by means of sound but incomplete static analyses [35, 36, 37, 38, 39]. In the existing work, the policy is generally taken to be $L \rightarrow H$ with exceptions allowed to $H \not\rightarrow L$, or more generally, a partial order with exceptions. The system is given by a single, typically deterministic program, and the focus is on relating initial values of input variables to final values of output variables, rather than on what can be deduced from ongoing observations in the state machine approach we have considered here.

7 Conclusion

In this paper, we have determined the computational complexity of verifying whether a finite-state system satisfies an intransitive noninterference security property. The polynomial-time upper bounds build on new characterizations and unw windings for two of the notions of noninterference dealt with. They also allow counterexamples (which can be used to improve the system in question) to be found when the system is insecure.

We have considered only deterministic systems: there have been several proposals to define intransitive noninterference in nondeterministic systems [40, 41, 10, 42, 43], the issue of complexity of these definitions remains open.

It would also be desirable to investigate algorithms and complexity for information flow policies of the richer types studied in the literature on programming languages approaches to declassification, in order to obtain sound and complete approaches for such specifications. Since intransitive noninterference policies provide a format for specifying architectural structure of a system, it would be interesting to combine the strengths of the programming languages perspective and the state machine model approach we have followed in this paper.

Acknowledgments: Work supported by Australian Research Council Discovery grant DP1097203. The second author thanks the Courant Institute, New York University, and the Computer Science Department, Stanford University for hosting sabbatical visits during which this research was conducted in part.

References

[1] S. Eggert, R. van der Meyden, H. Schnoor, and T. Wilke, “The complexity of intransitive noninterference,” in IEEE Symposium on Security and Privacy. IEEE Computer Society, 2011, pp. 196–211.
[2] J. Goguen and J. Meseguer, “Security policies and security models,” in Proc. IEEE Symp. on Security and Privacy, Oakland, 1982, pp. 11–20.

[3] J. Haigh and W. Young, “Extending the noninterference version of MLS for SAT,” IEEE Trans. on Software Engineering, vol. SE-13, no. 2, pp. 141–150, Feb 1987.

[4] J. Rushby, “Noninterference, transitivity, and channel-control security policies,” SRI International, Tech. Rep. CSL-92-02, Dec 1992. [Online]. Available: http://www.csl.sri.com/papers/csl-92-2/

[5] R. van der Meyden, “What, indeed, is intransitive noninterference?” in European Symposium On Research In Computer Security (ESORICS), ser. Lecture Notes in Computer Science, J. Biskup and J. Lopez, Eds., vol. 4734. Springer, 2007, pp. 235–250.

[6] J. Goguen and J. Meseguer, “Unwinding and inference control,” in IEEE Symp. on Security and Privacy, 1984.

[7] D. E. Denning, “A lattice model of secure information flow,” Communications of the ACM, vol. 19, no. 5, pp. 263–243, 1976.

[8] C. Boettcher, R. DeLong, J. Rushby, and W. Sifre, “The MILS component integration approach to secure information sharing,” in Proc. 27th IEEE/AIAA Digital Avionics Systems Conference, Oct. 2008, pp. 1.C.2–1–1.C.2–14.

[9] R. van der Meyden and C. Zhang, “A comparison of semantic models for noninterference,” Theoretical Computer Science, vol. 411, no. 47, pp. 4123–4147, Oct. 2010.

[10] A. W. Roscoe and M. H. Goldsmith, “What is intransitive noninterference?” in IEEE Computer Security Foundations Workshop, 1999, pp. 228–238.

[11] R. E. Tarjan, “Efficiency of a good but not linear set union algorithm,” J. ACM, vol. 22, no. 2, pp. 215–225, 1975.

[12] J. Burch, E. Clarke, K. McMillan, D. Dill, and L. Hwang, “Symbolic model checking: $10^{20}$ states and beyond,” Information and Computation, vol. 98, no. 2, pp. 142–170, 1992.

[13] E. Post, “A variant of a recursively unsolvable problem,” Bulletin of the American Mathematical Society, vol. 52, 1946.

[14] D. Sutherland, “A model of information,” in Proc. 9th National Computer Security Conf., 1986, pp. 175–183.

[15] J. T. Wittbold and D. M. Johnson, “Information flow in nondeterministic systems,” in IEEE Symposium on Security and Privacy, 1990, pp. 144–161.
[16] D. McCullough, “Noninterference and the composability of security properties,” in *Proc. IEEE Symp. on Security and Privacy*, 1988, pp. 177–186.

[17] R. Focardi and R. Gorrieri, “Classification of security properties (Part I: information flow),” in *Foundations of Security Analysis and Design, FOSAD 2000, Bertinoro, Italy, September 2000*, ser. LNCS, R. Focardi and R. Gorrieri, Eds. Springer, 2001, vol. 2171, pp. 331–396.

[18] P. Ryan, “Mathematical models of computer security,” in *Foundations of Security Analysis and Design, FOSAD 2000, Bertinoro, Italy, September 2000*, ser. LNCS, R. Focardi and R. Gorrieri, Eds. Springer, 2001, vol. 2171, pp. 1–62.

[19] G. Barthe, P. R. D’Argenio, and T. Rezk, “Secure information flow by self-composition,” in *CSFW*. IEEE Computer Society, 2004, pp. 100–114.

[20] R. Focardi and R. Gorrieri, “Automatic compositional verification of some security properties,” in *Proceedings of Second International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS’96)*, ser. Springer LNCS, vol. 1055, 1996, pp. 167–186.

[21] H. Mantel, “A uniform framework for the formal specification and verification of information flow security,” Ph.D. dissertation, Universität des Saarlandes, 2003.

[22] D. D’Souza, K. R. Raghavendra, and B. Sprick, “An automata based approach for verifying information flow properties,” *Electr. Notes Theor. Comput. Sci.*, vol. 135, no. 1, pp. 39–58, 2005.

[23] D. D’Souza, R. Holla, J. Kulkarni, R. K. Ramesh, and B. Sprick, “On the decidability of model-checking information flow properties,” in *ICISS*, ser. Lecture Notes in Computer Science, R. Sekar and A. K. Pujari, Eds., vol. 5352. Springer, 2008, pp. 26–40.

[24] B. Köpf and D. A. Basin, “Timing-sensitive information flow analysis for synchronous systems,” in *Proc. European Symp. on Research in Computer Security*, ser. Springer LNCS, vol. 4189, 2006, pp. 243–262.

[25] F. Cassez, R. van der Meyden, and C. Zhang, “The complexity of synchronous notions of information flow security,” in *FoSSaCS 2010, 13th International Conference on Foundations of Software Science and Computation Structures*, ser. Springer LNCS, vol. 6014, 2010, pp. 282–296.

[26] R. G. Riccardo Focardi and V. Panini, “The security checker: a semantics-based tool for the verification of security properties,” in *Proceedings of Eighth IEEE Computer Security Foundations Workshop (CSFW’95)*, 1995, pp. 60–69.
[27] M. W. Whalen, D. A. Greve, and L. G. Wagner, “Model checking information flow,” in Design and Verification of Microprocessor Systems for High-Assurance Applications, D. Hardin, Ed. Springer-Verlag, Berlin Germany, March 2010.

[28] D. Greve, M. Wilding, and M. Vanfleet, “A separation kernel formal security policy,” in Proc. Fourth International Workshop on the ACL2 Theorem Prover and Its Applications, 2003.

[29] S. Pinsky, “Absorbing covers and intransitive non-interference.” in Proc. IEEE Symp. on Security and Privacy, 1995, pp. 102–113.

[30] N. Hadj-Alouane, S. Lafrance, F. Lin, J. Mullins, and M. Yeddes, “On the verification of intransitive noninterference in multilevel security,” IEEE Trans. on Systems, Man and Cybernetics, Part B, vol. 35, no. 5, pp. 948–958, Oct. 2005.

[31] R. van der Meyden, “A comparison of semantic models for intransitive noninterference,” Dec 2007, unpublished manuscript, available at http://www.cse.unsw.edu.au/~meyden.

[32] S. Eggert, H. Schnoor, and T. Wilke, “Noninterference with local policies,” in Proceedings of the 38th International Symposium on Mathematical Foundations of Computer Science (MFCS 2013), June 2013, to appear.

[33] S. Chong and R. van der Meyden, “Deriving epistemic conclusions from agent architecture,” in Proc. Conf. on Theoretical Aspects of Knowledge and Rationality, 2009, pp. 61–70.

[34] ——, “Using architecture to reason about information security,” in Layered Assurance Workshop, 2012, to appear.

[35] H. Mantel and D. Sands, “Controlled declassification based on intransitive noninterference,” in Proc. Asian Symp. on Programming Languages and Systems, ser. LNCS, vol. 3302. Springer-Verlag, Nov. 2004, pp. 129–145.

[36] A. Sabelfeld and D. Sands, “Dimensions and principles of declassification,” in Proceedings of the 18th IEEE Computer Security Foundations Workshop. IEEE Computer Society Press, 2005, pp. 255–269.

[37] S. Chong and A. C. Myers, “Security policies for downgrading,” in 11th ACM Conf. on Computer and Communications Security (CCS), Oct 2004.

[38] A. Banerjee, D. A. Naumann, and S. Rosenberg, “Expressive declassification policies and modular static enforcement,” in IEEE Symposium on Security and Privacy. IEEE Computer Society, 2008, pp. 339–353.

[39] A. C. Myers, A. Sabelfeld, and S. Zdancewic, “Enforcing robust declassification,” in CSFW. IEEE Computer Society, 2004, pp. 172–186.
[40] W. Bevier and W. Young, “A state-based approach to noninterference,” in Proc. IEEE Computer Security Foundations Workshop, 1994, pp. 11–21.

[41] H. Mantel, “Information flow control and applications - bridging a gap,” in FME, ser. Lecture Notes in Computer Science, J. N. Oliveira and P. Zave, Eds., vol. 2021. Springer, 2001, pp. 153–172.

[42] D. von Oheimb, “Information flow control revisited: Noninfluence = Non-interference + Nonleakage,” in Computer Security – ESORICS 2004, ser. LNCS, vol. 3193, 2004, pp. 225–243.

[43] K. Engelhardt, R. van der Meyden, and C. Zhang, “Intransitive noninterference in nondeterministic systems,” in Proc. ACM Conf. on Computer and Communications Security, 2012, pp. 869–880.

[44] 17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA. IEEE Computer Society, 2004.
Appendix: Reduction from TO-security to ITO-security

We show that TO-security can be reduced to ITO-security.

First, we need some new definitions: For all $u \in D$ and all $\alpha \in A$, $\text{lpre}_u(\alpha)$ is the largest prefix of $\alpha$ that ends in an action $a$ with $\text{dom}(a) = u$. (If there is no such action, then we take $\text{lpre}_u(\alpha)$ to be $\epsilon$.) Similar to the definition of $\text{tview}$, we define

$$\text{ftview}_u(\alpha) = \text{view}_u(\text{lpre}_u(\alpha)).$$

Now we can give a characterization of ITO-security, similar to the characterization of TO-security in Proposition 1.

**Proposition 2** $M$ is ITO-secure with respect to a policy $\rightarrow$ iff for all sequences $\alpha, \beta \in A^*$, and domains $u \in D$, if $\text{purge}_u(\alpha) = \text{purge}_u(\beta)$ and $\text{ftview}_u(\alpha) = \text{ftview}_u(\beta)$ for all domains $v \neq u$ such that $v \rightarrow u$, then $\text{obs}_u(s_0 \cdot \alpha) = \text{obs}_u(s_0 \cdot \beta)$.

**Proof:** We first argue from left to right. We show by induction on the combined length of $\alpha, \beta \in A^*$ that, for $u \in D$, if we have $\text{purge}_u(\alpha) = \text{purge}_u(\beta)$ and $\text{ftview}_u(\alpha) = \text{ftview}_u(\beta)$ for all domains $v \neq u$ with $v \rightarrow u$, then $\text{ito}_u(\alpha) = \text{ito}_u(\beta)$. Thus, if $M$ is ITO-secure and the antecedent of the right hand side holds, then so does the consequent $\text{obs}_u(s_0 \cdot \alpha) = \text{obs}_u(s_0 \cdot \beta)$, by ITO-security. The proof of the induction hypothesis is trivial in the base case of $\alpha = \beta = \epsilon$. Suppose that $\alpha = \alpha' a$ where $a \in A$, and $\text{purge}_u(\alpha') = \text{purge}_u(\beta)$ and $\text{ftview}_u(\alpha') = \text{ftview}_u(\beta)$ for all domains $v \neq u$ with $v \rightarrow u$. We consider several cases.

**Case 1:** $\text{dom}(a) \not\rightarrow u$. Then $\text{purge}_u(\alpha') = \text{purge}_u(\alpha' a) = \text{purge}_u(\beta)$, and $\text{ftview}_u(\alpha') = \text{ftview}_u(\alpha' a) = \text{ftview}_u(\beta)$. By the induction hypothesis, $\text{ito}_u(\alpha') = \text{ito}_u(\beta)$. Since $\text{ito}_u(\alpha' a) = \text{ito}_u(a \alpha)$ in this case, we obtain $\text{ito}_u(\alpha' a) = \text{ito}_u(\beta)$.

**Case 2:** $\text{dom}(a) \rightarrow u$. In this case, $\text{purge}_u(\alpha'a) = \text{purge}_u(\alpha' a) = \text{purge}_u(\beta)$, so $\beta = b^* \beta$ for some $b \in A$. In case $\text{dom}(b) \not\rightarrow u$, we may apply Case 1 with the roles of $\alpha$ and $\beta$ swapped. We therefore assume without loss of generality that $\text{dom}(b) \rightarrow u$. Then $\text{purge}_u(\beta) = \text{purge}_u(\beta'b)$, and it follows that $a = b$ and $\text{purge}_u(\alpha') = \text{purge}_u(\beta')$.

Consider now $v \neq u$ with $v \rightarrow u$. In the case $v \not\in \text{dom}(a)$, we have $\text{ftview}_u(\alpha') = \text{ftview}_u(\alpha' a) = \text{ftview}_u(\beta') = \text{ftview}_u(\beta')$. In the case that $v = \text{dom}(a)$, we have $\text{ftview}_u(\alpha'a) = \text{view}_u(\alpha'a)$ and $\text{ftview}_u(\beta'b) = \text{view}_u(\beta'b)$, so we obtain that $\text{view}_u(\alpha'a) = \text{view}_u(\beta'b)$, so also $\text{view}_u(\alpha') = \text{view}_u(\beta')$. The latter implies that $\text{ftview}_u(\alpha') = \text{ftview}_u(\beta')$ in this case also.

If therefore follows by the induction hypothesis that $\text{ito}_u(\alpha') = \text{ito}_u(\beta')$. Thus, in case $\text{dom}(a) \neq u$, using what was shown above, we have

$$\text{ito}_u(\alpha'a) = (\text{ito}_u(\alpha'), \text{view}_{\text{dom}(a)}(\alpha'a), a) = (\text{ito}_u(\beta'), \text{view}_{\text{dom}(a)}(\beta'b), b) = \text{ito}_u(\beta'b).$$

43
Similarly, in the case $\text{dom}(a) = u$, we have

$$\text{ito}_u(\alpha'a) = (\text{ito}_u(\alpha'), \text{view}_{\text{dom}(a)}(\alpha'), a)$$

$$= (\text{ito}_u(\beta'), \text{view}_{\text{dom}(a)}(\beta'), b)$$

$$= \text{ito}_u(\beta'b).$$

Thus, in either case we have $\text{ito}_u(\alpha) = \text{ito}_u(\beta)$, as claimed. This completes the proof of the direction from left to right.

For the converse, define for each domain $u$, a function $P_u$ and functions $F_v$, all on the range of the function $\text{ito}_u$, by $P_u(\epsilon) = \epsilon$ and $F_v(\epsilon) = \text{obs}_v(s_0)$, and $P_u((x,y,a)) = P_u(x)a$ and

$$F_v((x,y,a)) = \begin{cases} y & \text{if dom}(a) = v, \\ F_v(x) & \text{if dom}(a) \neq v. \end{cases}$$

We claim that for all $u,v \in D$ with $u \neq v$ and $v \mapsto u$, and all $\alpha \in A^*$, we have $P_u(\text{ito}_u(\alpha)) = \text{purge}_u(\alpha)$ and $F_v(\text{ito}_u(\alpha)) = \text{ftview}_v(\alpha)$. Now, if $M$ is not ITO-secure then there exist $\alpha, \beta \in A^*$ and domain $u$ with $\text{ito}_u(\alpha) = \text{ito}_u(\beta)$ and $\text{obs}_u(s_0 \cdot \alpha) \neq \text{obs}_u(s_0 \cdot \beta)$. It follows from the claim that $\text{purge}_u(\alpha) = \text{purge}_u(\beta)$ and $\text{ftview}_v(\alpha) = \text{ftview}_v(\beta)$ for domains $v \neq u$ with $v \mapsto u$, so the right hand side of the proposition is also false.

The proof of the claim that $P_u(\text{ito}_u(\alpha)) = \text{purge}_u(\alpha)$ is a straightforward induction, left to the reader. For $F_v(\text{ito}_u(\alpha)) = \text{ftview}_v(\alpha)$ we argue inductively, as follows. The base case of $\alpha = \epsilon$ is trivial. For $\alpha = \alpha'a$, we consider the following cases:

**Case 1**: $\text{dom}(a) \nrightarrow u$. In this case we have $v \neq \text{dom}(a)$. Thus,

$$F_v(\text{ito}_u(\alpha'a)) = F_v(\text{ito}_u(\alpha'))$$

$$= \text{ftview}_v(\alpha')$$

by induction,

$$= \text{ftview}_v(\alpha'a)$$

**Case 2**: $\text{dom}(a) \rightarrow u$ and $\text{dom}(a) \neq u$ and $v = \text{dom}(a)$. In this case

$$F_v(\text{ito}_u(\alpha'a)) = F_v((\text{ito}_u(\alpha'), \text{view}_{\text{dom}(a)}(\alpha'a), a))$$

$$= \text{view}_v(\alpha'a)$$

$$= \text{ftview}_v(\alpha'a)$$

**Case 3**: $\text{dom}(a) \rightarrow u$ and $\text{dom}(a) \neq u$ and $v \neq \text{dom}(a)$. In this case

$$F_v(\text{ito}_u(\alpha'a)) = F_v((\text{ito}_u(\alpha'), \text{view}_{\text{dom}(a)}(\alpha'a), a))$$

$$= F_v(\text{ito}_u(\alpha'))$$

$$= \text{ftview}_v(\alpha')$$

by induction

$$= \text{ftview}_v(\alpha'a)$$

44
Case 4: \( \text{dom}(a) = u \). In this case,

\[
F_v(\text{ito}_u(\alpha'a)) = F_v((\text{ito}_u(\alpha'), \text{view}_u(\alpha', a))
\]

\[
= F_v(\text{ito}_u(\alpha')) \quad \text{since } v \neq \text{dom}(a)
\]

\[
= \text{ftview}_v(\alpha') \quad \text{by induction}
\]

\[
= \text{ftview}_u(\alpha'a) \quad \text{since } v \neq \text{dom}(a)
\]

This completes the proof of the claim. □

We now show to reduce TO-security to ITO-security. Given a system \( M \), we construct a system \( M' \) such that \( M \) is TO-secure iff \( M' \) is ITO-secure. The intuition for the construction is that ITO-security permits a faster flow of information than TO-security. In particular, when action \( a \), with \( \text{dom}(a) \not\rightarrow u \) and \( \text{dom}(a) \neq u \), is performed after sequence \( \alpha \), ITO-security permits the transmission of the information in the view \( \text{view}_{\text{dom}(a)}(\alpha a) \), whereas TO-security permits transmission only of the information in the shorter view \( \text{view}_{\text{dom}(a)}(\alpha) \).

The reduction handles this by replacing the final observation \( \text{obs}_u(s_0 \cdot a_0) \) made by \( \text{dom}(a) \) by the uninformative observation \( \bot \). This makes the faster flow of ITO-security, in the system \( M' \) equivalent to the slower flow of TO-security in \( M \).

More precisely, given a system \( M = \langle S, s_0, A, \text{step}, \text{obs}, \text{dom} \rangle \), we define a new system \( M' = \langle S', s'_0, A', \text{step}', \text{obs}', \text{dom}' \rangle \) as follows. We define the set of final actions \( A^f = \{ a^f | a \in A \} \). For every such an action we have \( \text{dom}(a^f) = \text{dom}(a) \). The idea underlying final actions is that, if an agent has performed one of its final actions, all further actions of this agent are ignored by the system. Therefore the system has to keep track of which agent has performed one of its final actions. If an agent has performed one of its final actions, its observation is set to \( \bot \) and will never change. More formally:

\[
S' = S \times \mathcal{P}(D)
\]

\[
s'_0 = (s_0, \emptyset)
\]

\[
A' = A \cup A^f
\]

For any \( s \in S \) and \( U \subseteq D \):

\[
\text{obs}'_u(s, U) = \begin{cases} 
\text{obs}_u(s) & \text{if } u \not\in U \\
\bot & \text{otherwise}
\end{cases}
\]

For any \( s \in S \), \( a \in A \) and \( U \subseteq D \):

\[
(s, U) \cdot a = \begin{cases} 
(s \cdot a, U) & \text{if } \text{dom}(a) \not\in U \text{ and } a \in A \\
(s \cdot b, U \cup \{\text{dom}(a)\}) & \text{if } \text{dom}(a) \not\in U \text{ and } a \in A^f \text{ with } a = b^f \\
(s, U) & \text{if } \text{dom}(a) \in U
\end{cases}
\]

For the functions \( \text{view}, \text{tview}, \text{etc.} \), we use the same notation in both systems. The intended system will be clear from the context.
Lemma 7  A system $M$ is TO-secure with respect to a policy $\rightarrow$ iff the system $M'$ is ITO-secure with respect to the same policy $\rightarrow$.

Proof: We first show the implication from left to right. We begin by defining a function that transfers between runs in the two systems, and prove several of its properties. For an action $a \in A'$, we define $\pi = a$ if $a \in A$ and $\pi = b$ if $a = b^f \in A^f$. We define the function convertback: $A^* \rightarrow A^*$ by

$$convertback(\alpha) = \begin{cases} convertback(\alpha) & \text{if there is a final action of } \dom(\alpha) \text{ in } \alpha \\ convertback(\alpha) & \text{if there is no final action of } \dom(\alpha) \text{ in } \alpha. \end{cases}$$

This function convertback removes for each agent its actions performed after the agent’s first final action. This first final action is replaced by the corresponding action from $A$.

We observe, that we have for any $\alpha \in A'^*$: If $s_0' \cdot \alpha = (s, U)$ for some $s \in S$ and $U \subseteq D$, then $s = s_0 \cdot convertback(\alpha)$. This implies that for any $u \in D$ and any $\alpha \in A'^*$ that does not contain a final action of $u$, we have

$$\text{obs}_u(s_0 \cdot convertback(\alpha)) = \text{obs}_u'(s_0' \cdot \alpha).$$

This also implies that for any $u \in D$ and any $\alpha \in A'^*$ that does not contain any final action of $u$, we have

$$\text{view}_u(convertback(\alpha)) = \text{view}_u(\alpha).$$

(Here the left hand side is computed in $M$ and the right hand side in $M'$.) The argument for this is an induction on $\alpha$. The claim is trivial in case $\alpha = \epsilon$.

Suppose $\alpha = \alpha'a$ where $a \in A$, and $\alpha'a$ does not contain any final actions of $u$.

We consider several cases:

- **Case 1**: $\dom(\alpha) = u$. Then $a$ is not final and $\alpha'$ contains no final action of $u$, so $\text{convertback}(\alpha'a) = \text{convertback}(\alpha')a$, and

$$\text{view}_u(\alpha'a) = \text{view}_u(\alpha') a \text{obs}_u'(s_0' \cdot \alpha'a)$$

$$= \text{view}_u(\text{convertback}(\alpha')) a \text{obs}_u(s_0 \cdot \text{convertback}(\alpha'a))$$

$$= \text{view}_u(\text{convertback}(\alpha')) a \text{obs}_u(s_0 \cdot \text{convertback}(\alpha'a))$$

$$= \text{view}_u(\text{convertback}(\alpha'a)).$$

- **Case 2**: $\dom(\alpha) \neq u$ and there is no final action of $\dom(\alpha)$ in $\alpha'$. Then
convertback(α′a) = convertback(α′)π, so

\[
\operatorname{view}_u(α′a) = \operatorname{view}_u(α′) \circ \operatorname{obs}_u'(s_0 \cdot α′a)
\]

= \operatorname{view}_u(\operatorname{convertback}(α′)) \circ \operatorname{obs}_u'(s_0 \cdot α′a)

by induction

= \operatorname{view}_u(\operatorname{convertback}(α′)) \circ \operatorname{obs}_u(s_0 \cdot \operatorname{convertback}(α′a))

= \operatorname{view}_u(\operatorname{convertback}(α′)) \circ \operatorname{obs}_u(s_0 \cdot \operatorname{convertback}(α′))

= \operatorname{view}_u(\operatorname{convertback}(α′a))

= \operatorname{view}_u(\operatorname{convertback}(α′a)) .

• Case 3: dom(α) ≠ u and there is a final action of dom(α) in α′. Then convertback(α′a) = convertback(α′), so

\[
\operatorname{view}_u(α′a) = \operatorname{view}_u(α′) \circ \operatorname{obs}_u'(s_0 \cdot α′a)
\]

= \operatorname{view}_u(\operatorname{convertback}(α′)) \circ \operatorname{obs}_u'(s_0 \cdot α′a)

by induction

= \operatorname{view}_u(\operatorname{convertback}(α′)) \circ \operatorname{obs}_u(s_0 \cdot \operatorname{convertback}(α′a))

= \operatorname{view}_u(\operatorname{convertback}(α′)) \circ \operatorname{obs}_u(s_0 \cdot \operatorname{convertback}(α′))

= \operatorname{view}_u(\operatorname{convertback}(α′))

= \operatorname{view}_u(\operatorname{convertback}(α′a)) .

This completes the proof that \(\operatorname{view}_u(\operatorname{convertback}(α)) = \operatorname{view}_u(α)\). Plainly, this implies that for any α, β ∈ A* that do not contain any final action of u, we have

\[
\operatorname{view}_u(\operatorname{convertback}(α)) = \operatorname{view}_u(\operatorname{convertback}(β)) \text{ iff } \operatorname{view}_u(α) = \operatorname{view}_u(β) .
\]

We now claim that for any u ∈ D and any α, β ∈ A* that do not contain a final action of domain u, that if \(\operatorname{purge}_u(α) = \operatorname{purge}_u(β)\) and \(\operatorname{ftview}_u(α) = \operatorname{ftview}_u(β)\) for all v ≠ u, v → u, then \(\operatorname{purge}_u(\operatorname{convertback}(α)) = \operatorname{purge}_u(\operatorname{convertback}(β))\) and \(\operatorname{ftview}_u(\operatorname{convertback}(α)) = \operatorname{ftview}_u(\operatorname{convertback}(β))\) for all v ≠ u, v → u.

We prove the claim by induction over the combined length of α and β. It is clear that it holds for α = β = ε. Consider α = α′a and β, neither containing a final action of domain u, such that \(\operatorname{purge}_u(α′a) = \operatorname{purge}_u(β)\) and for all v ≠ u, v → u: \(\operatorname{ftview}_v(α′a) = \operatorname{ftview}_v(β)\) and that the implication holds for strings of shorter combined length. We consider several cases:

• Case 1: dom(α) ↗ u. This gives

\[
\operatorname{purge}_u(α′) = \operatorname{purge}_u(α′a) = \operatorname{purge}_u(β)
\]

and

\[
\operatorname{ftview}_v(α′) = \operatorname{ftview}_v(α′a) = \operatorname{ftview}_v(β)
\]

for all v ≠ u, v → u. By the induction hypothesis,

\[
\operatorname{purge}_u(\operatorname{convertback}(α′)) = \operatorname{purge}_u(\operatorname{convertback}(β))
\]
and
\[
\text{tview}_v(\text{convertback}(\alpha')) = \text{tview}_v(\text{convertback}(\beta))
\]
for all \( v \neq u, \ v \mapsto u \). If there is a final action of \( \text{dom}(a) \) in \( \alpha' \), then \( \text{convertback}(\alpha'a) = \text{convertback}(\alpha') \) and the claim for the pair \( \alpha'a \) and \( \beta \) is immediate. We assume in the following that there is no final action of \( \text{dom}(a) \) in \( \alpha' \). Then, since \( \text{dom}(a) = \text{dom}(a) \mapsto u \), we have
\[
\text{purge}_u(\text{convertback}(\alpha'a)) = \text{purge}_u(\text{convertback}(\alpha')) = \text{purge}_u(\text{convertback}(\beta))
\]
and for all \( v \neq u, \ v \mapsto u \):
\[
\text{tview}_v(\text{convertback}(\alpha'a)) = \text{tview}_v(\text{convertback}(\alpha')\pi)
= \text{tview}_v(\text{convertback}(\alpha')) = \text{tview}_v(\text{convertback}(\beta)).
\]

• Case 2: \( \text{dom}(a) \mapsto u \). In this case we have \( \text{purge}_u(\alpha')a = \text{purge}_u(\alpha'a) = \text{purge}_u(\beta) \), so \( \beta = \beta'b \) for some action \( b \). If \( \text{dom}(b) \mapsto u \), then we may swap the roles of \( \alpha \) and \( \beta \) and apply the previous case. Hence, without loss of generality, \( \text{dom}(b) \mapsto u \), and we have \( \text{purge}_u(\beta) = \text{purge}_u(\beta'b) \). It follows that \( a = b \) and \( \text{purge}_u(\alpha') = \text{purge}_u(\beta') \). For all \( v \neq u, \ v \mapsto u \) with \( v \neq \text{dom}(a) \) we have
\[
\text{ftview}_v(\alpha') = \text{ftview}_v(\alpha'a)
= \text{ftview}_v(\beta'a)
= \text{ftview}_v(\beta').
\]
If \( v = \text{dom}(a) \) then we have
\[
\text{view}_v(\alpha'a) = \text{ftview}_v(\alpha'a)
= \text{ftview}_v(\beta'a)
= \text{view}_v(\beta'a).
\]
This implies \( \text{view}_v(\alpha') = \text{view}_v(\beta') \) and therefore, \( \text{ftview}_v(\alpha') = \text{ftview}_v(\beta') \).
Thus, by the induction hypothesis,
\[
\text{purge}_u(\text{convertback}(\alpha')) = \text{purge}_u(\text{convertback}(\beta'))
\]
and \( \text{tview}_v(\text{convertback}(\alpha')) = \text{tview}_v(\text{convertback}(\beta')) \) for all \( v \neq u, \ v \mapsto u \).
If there is a final action of \( \text{dom}(a) \) in \( \alpha' \), then, by \( \text{purge}_u(\alpha') = \text{purge}_u(\beta') \), there is also a final action of \( \text{dom}(a) \) in \( \beta' \), and we have \( \text{convertback}(\alpha'a) = \text{convertback}(\alpha') \) and \( \text{convertback}(\beta'a) = \text{convertback}(\beta') \). The desired
conclusion is then direct from the above inductive conclusion. Alternately, if there is no final action of \( \text{dom}(a) \) in \( \alpha' \) then there is no final action of \( \text{dom}(a) \) in \( \beta' \). In this case, since \( \text{dom}(\pi) = \text{dom}(a) \rightarrow u \), we have

\[
\text{purge}_u(\text{convertback}(\alpha'a)) = \text{purge}_u(\text{convertback}(\alpha')\pi)
= \text{purge}_u(\text{convertback}(\beta')\pi)
= \text{purge}_u(\text{convertback}(\beta'a))
\]

Also, for all \( v \neq u, v \rightarrow u \), in case \( \text{dom}(a) \neq v \) we have

\[
\text{tview}_v(\text{convertback}(\alpha'a)) = \text{tview}_v(\text{convertback}(\alpha')\pi)
= \text{tview}_v(\text{convertback}(\alpha'))
= \text{tview}_v(\text{convertback}(\beta')\pi)
= \text{tview}_v(\text{convertback}(\beta'a))
\]

In the case \( \text{dom}(a) = v \) we argue as follows. If \( a \) is not a final action, we have

\[
\text{view}_{\text{dom}(a)}(\alpha'a) = \text{ftview}_{\text{dom}(a)}(\alpha'a)
= \text{ftview}_{\text{dom}(a)}(\beta'a)
= \text{view}_{\text{dom}(a)}(\beta'a)
\]

Since there is no final action of \( \text{dom}(a) \) in \( \alpha' \) or \( \beta' \), and \( a \) is not final, there is no final action of \( \text{dom}(a) \) in \( \alpha'a \) or \( \beta'a \), and therefore

\[
\text{view}_{\text{dom}(a)}(\text{convertback}(\alpha'a)) = \text{view}_{\text{dom}(a)}(\text{convertback}(\beta'a))
\]

using the equivalence proved above. From this equation it follows that

\[
\text{tview}_{\text{dom}(a)}(\text{convertback}(\alpha'a)) = \text{tview}_{\text{dom}(a)}(\text{convertback}(\beta'a))
\]

In the case that \( a \) is a final action, we have

\[
\text{view}_{\text{dom}(a)}(\alpha'a) = \text{ftview}_{\text{dom}(a)}(\alpha'a)
= \text{ftview}_{\text{dom}(a)}(\beta'a)
= \text{view}_{\text{dom}(a)}(\beta'a)
\]

Therefore \( \text{view}_{\text{dom}(a)}(\alpha') = \text{view}_{\text{dom}(a)}(\beta') \). Since there is no final action of \( \text{dom}(a) \) in \( \alpha' \) or \( \beta' \), it follows using the equivalence proved above that

\[
\text{view}_{\text{dom}(a)}(\text{convertback}(\alpha')) = \text{view}_{\text{dom}(a)}(\text{convertback}(\beta'))
\]
Thus,
\[
\text{tview}_{\text{dom}(a)}(\text{convertback}(\alpha' a)) = \text{tview}_{\text{dom}(a)}(\text{convertback}(\alpha') \beta)
= \text{view}_{\text{dom}(a)}(\text{convertback}(\alpha')) \beta
= \text{view}_{\text{dom}(a)}(\text{convertback}(\beta')) \beta
= \text{tview}_{\text{dom}(a)}(\text{convertback}(\beta' a)) .
\]

This completes the proof of the claim.

We are now positioned to prove that if \( M \) is TO-secure then \( M' \) is ITO-secure. We show the contrapositive. Suppose \( M' \) is not ITO-secure. By Proposition \[\text{Proposition 2}\] there exist \( \alpha, \beta \in A^* \) and domain \( u \) such that \( \text{purge}_u(\alpha) = \text{purge}_u(\beta) \) and \( \text{ftview}_u(\alpha) = \text{ftview}_u(\beta) \) for all domains \( v \neq u \) such that \( v \rightarrow u \), and \( \text{obs}_u(s'_0, \alpha) \neq \text{obs}_u(s'_0, \beta) \). It follows from \( \text{purge}_u(\alpha) = \text{purge}_u(\beta) \) that \( \alpha \) contains a final action of domain \( u \) iff \( \beta \) contains a final action of domain \( u \).

But if both contain such a final action, then \( \text{obs}_u(s'_0, \alpha) = \bot = \text{obs}_u(s'_0, \beta) \), contrary to assumption. Thus neither \( \alpha \) nor \( \beta \) contain a final action of \( u \).

By what was shown above, \( \text{purge}_u(\text{convertback}(\alpha)) = \text{purge}_u(\text{convertback}(\beta)) \) and \( \text{ftview}_u(\text{convertback}(\alpha)) = \text{ftview}_u(\text{convertback}(\beta)) \) for all domains \( v \neq u \) such that \( v \rightarrow u \). Also, \( \text{obs}_u(s_0 \cdot \text{convertback}(\alpha)) \neq \text{obs}_u(s_0 \cdot \text{convertback}(\beta)) \).

By the characterization of TO-security of Proposition \[\text{Proposition 1}\] this implies that \( M \) is not TO-secure with respect to \( \rightarrow \).

For the other direction of the proof we define a function \( \text{convert}: D \times A^* \rightarrow A^* \), which, for each domain \( v \neq u \) with \( v \rightarrow u \), replaces the rightmost action \( a \) with \( \text{dom}(a) = v \) by the action \( a^f \).

We observe that, for all \( u \in D \), if \( \gamma, \gamma' \) are prefixes of \( \alpha \) and \( \text{convert}_u(\alpha) \), respectively, and have the same length, then for all \( U \subseteq D \), if \( s'_0 \cdot \gamma' = (s, U) \) then \( s = s_0 \cdot \gamma \). Therefore, since \( \text{convert}_u(\alpha) \) contains no final action of domain \( u \), we have \( \text{obs}_u(s_0 \cdot \text{convert}_u(\alpha)) = \text{obs}_u(s_0 \cdot \alpha) \). Moreover, if \( \gamma, \gamma' \) are prefixes of \( \alpha \) and \( \text{convert}_u(\alpha) \), respectively, have the same length, and \( \gamma \) does not contain the rightmost action of domain \( v \) in \( \alpha \) (if any), then \( \text{view}_v(\gamma) = \text{view}_v(\gamma') \).

We show that, for all \( u \in D \) and all \( \alpha, \alpha' \in A^* \), if \( \text{purge}_u(\alpha) = \text{purge}_u(\alpha') \) and \( \text{tview}_u(\alpha) = \text{tview}_u(\alpha') \) for all \( v \neq u \), \( v \rightarrow u \), then \( \text{purge}_u(\text{convert}_u(\alpha)) = \text{purge}_u(\text{convert}_u(\alpha')) \) and \( \text{ftview}_u(\text{convert}_u(\alpha)) = \text{ftview}_u(\text{convert}_u(\alpha')) \) for all \( v \neq u \), \( v \rightarrow u \). By an argument similar to that for the opposite direction, it then follows that if \( M' \) is ITO-secure then \( M \) is TO-secure.

We observe that, for any agent \( u \), the functions \( \text{purge}_u \) and \( \text{convert}_u \) commute. This shows that for all \( \alpha, \alpha' \in A^* \), if \( \text{purge}_u(\alpha) = \text{purge}_u(\alpha') \) then, \( \text{purge}_u(\text{convert}_u(\alpha)) = \text{purge}_u(\text{convert}_u(\alpha')) \).

To complete the argument, we assume \( \text{purge}_u(\alpha) = \text{purge}_u(\alpha') \) and for a domain \( v \neq u \) with \( v \rightarrow u \), we have \( \text{tview}_u(\alpha) = \text{tview}_u(\alpha') \), and show that \( \text{ftview}_u(\text{convert}_u(\alpha)) = \text{ftview}_u(\text{convert}_u(\alpha')) \). From \( \text{purge}_u(\alpha) = \text{purge}_u(\alpha') \) it follows that the sequences of actions of domain \( v \) in \( \alpha \) and \( \alpha' \) are the same. In particular, if neither sequence contains an action of domain \( v \), then the claim is trivial. Suppose that \( \alpha \) is the last action of domain \( v \) in both \( \alpha \) and \( \alpha' \). Then we may write \( \alpha = \alpha_1 a \alpha_2 \) and \( \alpha' = \alpha'_1 a \alpha'_2 \), where \( \alpha_2, \alpha'_2 \) contain no actions of
domain $v$. Since $\text{tview}_v(\alpha) = \text{tview}_v(\alpha')$, we have $\text{view}_v(\alpha_1) = \text{view}_v(\alpha'_1)$. Also $\text{convert}_u(\alpha) = \gamma_1 a^f \gamma_2$ and $\text{convert}_u(\alpha') = \gamma'_1 a^f \gamma'_2$, where $\gamma_1, \gamma'_1$ are of the same length as $\alpha_1, \alpha'_1$, respectively, and $\gamma_2, \gamma'_2$ contain no actions of domain $v$. Thus, using the observation above,

$$\text{ftview}_v(\text{convert}_u(\alpha)) = \text{ftview}_v(\gamma_1 a^f \gamma_2)$$

$$= \text{view}_v(\gamma_1) a^f \bot$$

$$= \text{view}_v(\alpha_1) a^f \bot$$

$$= \text{view}_v(\alpha'_1) a^f \bot$$

$$= \text{view}_v(\gamma'_1) a^f \bot$$

$$= \text{ftview}_v(\text{convert}_u(\alpha')) .$$